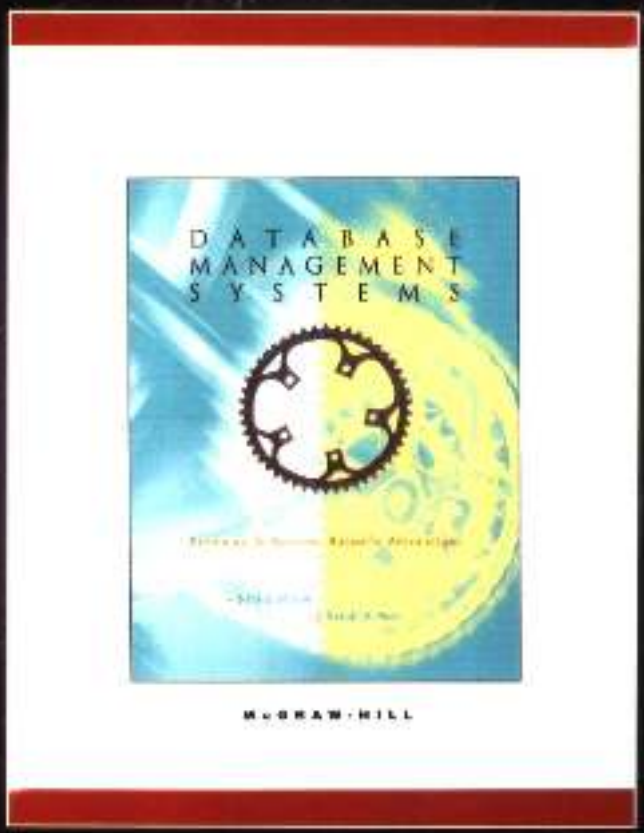
计 算 机 科 学 丛 书

原书第3版

数据库管理系统

(美) Gerald V. Post 太平洋大学 著 冯建华 刘旭辉 周维续 等译



Database Management Systems
Designing and Building Business Applications
Third Edition

机械工业出版社
China Machine Press

数据库管理系统（原书第3版）

对于管理和信息技术的专业人员来说，数据库是最重要和最实用的工具之一。数据库为收集、组织和共享数据提供了基础。数据库管理方法提供了很多传统编程技术无法比拟的优势，主要包括更短的开发时间、更容易修改、更好的数据完整性与安全性以及更强的数据共享和集成。而DBMS是最复杂的实用技术工具之一，本书细致讲解了如何在商业应用中使用DBMS。

本书涵盖了构建数据库前的两个关键主题：数据库设计（规范化）和SQL（查询）。所有主要的数据库系统都涉及这两个主题。规范化说明了如何细致设计数据库以获得DBMS能力。SQL是一种标准查询语言，事实上用于应用程序开发的每一步。

本书特色

1. 侧重于现代业务应用程序开发。
   - 根据业务模型来阐述数据库设计。
   - 通过很多示例和练习强调动手实践应用。
   - 侧重于新型图形用户界面应用程序。
   - 包含数据库编程和应用程序开发的内容。
   - 关于编程和开发细节内容的附录。
2. 热点主题。
   - 介绍并使用统一建模语言（UML）来建模和绘制系统图表。
   - 关于数据库环境下安全主题的深入讨论。
   - 因特网和内联网的数据库开发。
   - 强调SQL 92，同时介绍SQL 99和SQL 200x的XML特性。
   - 数据库中完整的应用程序和对象。
3. 实用的业务练习和案例。
   - 很多数据库设计问题。
   - 涵盖应用程序开发所有方面的练习。
   - 适用于期末实践项目的案例。
4. 完整的示例数据库应用程序。
   - 功能完善的业务数据库。
   - 示例数据和数据产生例程。
   - 一般数据库操作的示例程序代码。

作者简介

Gerald V. Post 1983年于艾奥瓦州立大学获得博士学位，现为太平洋大学管理信息系统教授，讲授管理信息系统、数据库管理、系统开发、网站开发等课程。其网站http://jerrypost.com/DBBook/index.html提供本书相关资源下载。



随书光盘中包括：PowerPoint幻灯片形式的讲课记录；为特定数据库技术编制的工作手册；实践项目案例。

ISBN 7-111-19296-6
9 787111 192961

HZ BOOKS
华章图书

上架指导：计算机\数据库

华章网站 http://www.hzbook.com
网上购书：www.china-pub.com
投稿热线：(010) 88379604
购书热线：(010) 68995259, 88995264
读者信箱：hzjsj@hzbook.com
ISBN 7-111-19296-6
定价：48.00 元（附光盘）

计 算 机 科 学 丛 书

原书第3版

数据库管理系统

(美) Gerald V. Post 太平洋大学 著 冯建华 刘旭辉 周维续 等译



Database Management Systems

Designing and Building Business Applications

Third Edition

机械工业出版社
China Machine Press

本书涵盖了构建数据库前的两个关键主题：数据库设计（规范化）和SQL（查询）。这两个主题贯穿所有主要的数据库系统。

本书分四个部分，首先，对数据库设计和数据规范化进行介绍；其次，讨论如何将商业问题转化为SQL查询以及包括子查询和外连接的查询；然后讲解表单、报表及应用，数据库完整性和事务以及数据仓库和数据挖掘；最后介绍数据库管理中的各种主题，如安全性、分布式数据库和因特网等。

本书适合作为高等院校相关专业数据库课程的教材，还适合非计算机专业的管理人员建立和应用数据库时参考。

Gerald V. Post: Database Management Systems: Designing and Building Business Applications, Third Edition.(ISBN 0-07-111180-8).

Copyright © 2005 by The McGraw-Hill Companies, Inc.

Original English edition published by The McGraw-Hill Companies, Inc.All rights reserved. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher.

Simplified Chinese translation edition jointly published by McGraw-Hill Education(Asia) Co. and China Machine Press.

本书中文简体字翻译版由机械工业出版社和美国麦格劳－希尔教育（亚洲）出版公司合作出版。未经出版者预先书面许可，不得以任何方式复制或抄袭本书的任何部分。

本书封面贴有McGraw-Hill公司防伪标签，无标签者不得销售

版权所有，侵权必究

本书法律顾问　北京市展达律师事务所

本书版权登记号：图字：01-2005-5262

图书在版编目（CIP）数据

数据库管理系统（原书第3版）/（美）波斯特（Post, G. V.）著；冯建华等译. －北京：机械工业出版社，2006.8

（计算机科学丛书）

书名原文：Database Management Systems: Designing and Building Business Applications, Third Edition

ISBN 7-111-19296-6

Ⅰ.数… Ⅱ.①波… ②冯… Ⅲ.数据库管理系统 Ⅳ.TP311.13

中国版本图书馆CIP数据核字（2006）第059580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：王　玉

北京瑞德印刷有限公司印刷 · 新华书店北京发行所发行

2006年8月第1版第1次印刷

184mm×260mm · 23.25印张

定价：48.00元（附光盘）

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzjsj@hzbook.com
联系电话：(010) 68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			

译 者 序

数据库是最重要的和最有用的专业管理与信息技术工具之一。数据库提供了收集、组织和共享企业数据的基础。例如，市场人员使用数据库来分析销售数据，人力资源管理者使用它来评价员工的表现，业务管理者用它来跟踪并提高产品的质量，会计人员用它来整合各个工厂的数据，而财务分析家使用它来分析整个公司的效益。

数据库管理方法与传统的程序设计技术相比，提供了更加重要的优势。基本的优势包括开发时间短、容易更新、较好的数据集成和数据安全，以及更高程度的数据共享和数据集成。但是，数据库管理系统（DBMS）是可用的最复杂的技术工具之一，只有仔细设计数据库才能获得这些优势。大型的商用DBMS提供了成千上万的选项。要花费几个月的时间才能全部掌握DBMS特有的功能。

目前市场上有关数据库方面的书籍主要集中在高校教材方面，主要讲述数据库系统的原理和设计。虽然也有一些关于如何使用数据库系统方面的书籍，但这些书更像是数据库管理员（DBA）的专业培训教材。而有关如何利用DBMS建立商业应用的书籍很少。

虽然数据库常常是由信息技术方面的专家来创建和维护的，但是其他领域的管理专家也越来越多地亲自设计和建立自己的数据库应用。因此，我们将本书翻译介绍给非计算机专业的人员，尤其是管理人员，为他们建立自己的商业数据库和应用提供全方位的指导与训练。从专业角度讲，本书适合大学三年级的学生学习如何设计商业数据库并建立应用，任何非计算机专业的学生都能够理解本书的内容，尤其是经济管理专业和MBA的学生更适合以本书作为数据库课程的教材。

本书的前言和第1章由冯建华负责翻译，第一部分（第2、3章）由刘旭辉负责翻译，第二部分（第4、5章）由林峰负责翻译，第三部分（第6、7、8章）由周维续负责翻译，第四部分（第9、10章）和词汇表由贺宇凯和李国良负责翻译，冯建华和周维续对全书进行了审校。

由于水平有限，在翻译过程中难免存在着这样或那样的错误，欢迎广大读者批评、指正。

译 者

2006年3月于清华园

前　言

两个网站的故事

Orinoco音乐公司引以为豪的是他们的网站。它上面有很酷的图片，很火爆的音乐片段。刚开始，新闻的宣传为公司带来了大量的潜在客户。订单以网络订购的形式蜂拥而至。但几周之后，出现了一些问题。职员从网页向公司现有的邮件订购系统复制订单时经常出现错误。由于许多商品缺货，客户不断地取消订单，并且抱怨如果他们知道那些商品缺货的话，决不会去订购。几个月之后，新闻开始批评这个网站，说上面的图片很旧，并且大家不能获得最新乐队的音乐片段。由于经常更新网站需要很大的开销，Orinoco音乐公司正在考虑修改这个网站，把它改成一个基本的公司信息网站。

客户们开始光顾Salt Peanuts音乐公司的新网站。在网站运行的一个月间，根据客户数量和每笔订单的金额计算，订购几乎翻了一倍。用户可以立即查看某件商品是否仍有库存。只需几次点击，他们便可以获取任何艺术家的背景信息，并且可以试听歌曲的小片段。已注册的客户付一些费用就可以将歌曲下载到自己的计算机中，随意播放。客户也可以通过即时消息和销售代表沟通。销售代表可以快速获取所有客户的数据。但每人都特别喜欢的特色是系统能够跟踪每个人的消费并提示与消费者相似的组。提示方法部分基于专家的意见，但主要由对消费进行分组决定。客户可以查看组中相似客户所购买的商品。每个人都很喜欢这个系统。公司经理喜欢它是因为它能促进销售，并且提供消费趋势的详细数据。客户们喜欢它是因为他们能够立刻获得想要的信息。唱片作者喜欢它是因为它提供了访问他们作品的途径并能提高销量。

上述两个网站的区别在于Salt Peanuts网站建立在数据库管理系统之上，数据库管理系统集成了公司的数据。它能够使公司创建一个更完整、可交互的站点。

本书内容简介

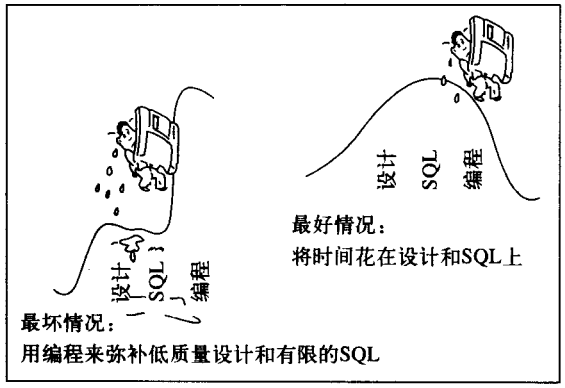
对于管理和信息技术的专业人员来说，数据库是最重要和最实用的工具之一。数据库为在机构内集成、组织和共享数据提供了基础。事实上，管理的每一个领域都使用数据库管理系统（DBMS）。例如，销售人员利用数据库来分析销售数据，人力资源经理用它来评估员工，执行经理用它来跟踪并提高质量，会计用它来合计公司内的数据，而财务分析员用它来分析一个公司的业绩。

数据库管理方法具备很多传统编程技术无法比拟的优势。主要的优势包括更短的开发时间、更容易修改、更好的数据完整性与安全性及更强的数据共享和集成。然而，DBMS是最复杂的实用技术工具之一。需要细致设计数据库以实现这些优势。一个大规模商用数据库提供了数千种选项并且价值数十万美元。掌握某一特定DBMS的全部特性可能需要花费数月时间。

尽管数据库一般由信息技术专业人员创建和维护，越来越多的其他学科的管理专业人员也开始设计和创建他们自己的数据库应用程序。本书适用于大学三年级水平的初级商业数据库课程。任何专业的学生都能够理解本书的内容，但如果他们已经学过一门初级编程课程的话，理解起来将会更容易一些。

本书目标和基本原理

本书的目标明确：学习过本书之后，学生应该能够评估业务情况并构建一个数据库应用程序。



本书的核心包括每个学生在构建数据库前必须学习的两个关键主题：数据库设计（规范化）和SQL（查询）。所有主要的数据库系统都涉及这两个主题。规范化说明了如何细致设计数据库以获得DBMS能力。SQL是一种标准查询语言，事实上用于应用程序开发的每一步。这两个主题必须仔细彻底地对待，特别是由于它们对于学生来说都是困难的主题。

尽管某些学生可以通过一次普通讨论或讲座学会如何构建应用程序，但大部分人则需要在知识渊博的教师的指导下通过示例和亲手实践来弄懂。本书通过清楚的讲解、众多示例、练习和示例数据库为学习过程提供支持。附带的工作手册提供了使用特定数据库系统构建应用程序的更多细节。

利用传统的关系数据库构建应用程序需要三种特别的技能：数据库设计、SQL知识和编程。这三种技能中的每一项都很复杂，但每一项对于构建成功的应用程序都是至关重要的。

数据库设计是构建应用程序的基础。一个设计良好的数据库能够简化应用程序的构建、维护和扩充过程。关系数据库设计的一个重点是它的灵活性。一个设计合理的数据库能够扩充以适应业务情形的变化。另一方面，如果设计很差，构建应用程序实际上将会更难且花费更多时间。一般来说，抛弃一个设计很差的数据库并重新开始，要好于尝试修补或扩展它。

SQL是一种强大的查询语言。它最强的能力之一在于它能够应用于许多不同的产品中。一旦你学会了SQL的基础，你将能够从几乎所有的数据库系统中获取数据。很多SQL查询语句是相对简单的，所以学会SQL基础是很快的。然而，SQL也可以用于解决复杂的问题。

编程技巧组成了构建可靠商业应用程序所需知识的第三个层次。某些应用程序和数据库系统需要详细的编程技巧。然而，在很多情况下编程用得还很不够。编程可用于集成不同的组件或添加新的功能，以使应用程序更容易使用。

大部分应用程序都会经历在数据库设计、SQL语言和编程三者间的协调。设计越差、对SQL的依赖越小，在构建应用程序时需要的编程将会越多。由于编写代码似乎更容易出错并且更难以修改，应用程序开发人员应该依赖于合理的数据库设计和SQL语言的能力。

学习评估

学习评估对于学生、教师和老板来说都是重要的。学生需要确定哪些方面是他们的强项，

而哪些方面则需要额外的努力。学生需要明白，如果能够成功学会本书的内容，他们将能够掌握很多技巧，这些技巧将会使他们获得工作，并能通过快速构建和维护业务应用程序，从而在职业上有所作为。

这门课程的学习评估很明确：学习本课程之后，学生应该能够评估业务情况并开发一个数据库应用程序。这个应用程序的复杂度和所使用的工具将取决于特定的班级和学生的背景。一般来说，布置一个为期一学期的实践项目是评估全部技能的好方法。期末的实践项目可以通过以下几方面来评定：(1) 正确符合业务需求，(2) 有效的数据库结构，(3) 可用性。

对个人技能进行独立评估也是很有用的，尤其是以组为单位创建期末实践项目的时候。在这种情况下，评估包括以下个人测试：(1) 数据库设计和规范化，(2) SQL和根据业务问题创建查询，(3) 精选的主题包括数据库编程、安全性、数据挖掘和分布式系统。

结构

本书的结构遵从应用程序开发的基本步骤：设计、查询、应用、管理和高级主题。一些教师可能会选择在讲授数据库设计前讲授查询，因此，前面几章在写作时保留了这种灵活性。

简介解释了数据库的重要性，并将数据库应用程序与学生可能在其他课上学过的主题进行联系。

数据库设计部分有两章：第2章是设计技术概述（系统技术，图表和控制），第3章详述数据规范化。这部分内容的目标是：在学期初阐述设计以便学生能够开始他们的实践项目。

查询用两章来讲解。第4章介绍查询并集中于将业务问题转化为SQL查询的基础。第5章讨论了更复杂的查询，包括子查询和外连接。

第3部分描述了数据库应用程序开发，第6章作为该部分的开始讲解表单、报表和应用的开发。第7章研究在多用户环境中产生的常见问题，阐述了用于处理数据完整性和事务的常用技术。第8章解释为何与事务处理相比分析处理需要不同的数据库配置。它涵盖用于在非静态上下文中进行分析和数据挖掘的主要工具。

第1章 简介
第1部分 系统设计
第2章 数据库设计
附录 数据库设计系统
第3章 数据规范化
附录 规范化的形式化定义
第2部分 查询
第4章 数据查询
附录 SQL语法
第5章 高级查询和子查询
附录 程序设计简介
第3部分 应用
第6章 表单、报表和应用
第7章 数据库集成和事务
第8章 数据仓库和数据挖掘
第4部分 数据库管理
第9章 数据库管理
第10章 分布式数据库和因特网

第4部分研究了数据库管理中的各种主题。第9章研究管理问题，重点强调计划、实现、性能和安全性。它阐述了管理员需要负责的主要任务和控制。第10章探讨了为数据库提供分布式访问的日益增加的重要性。它阐述各种网络配置的效果，还讨论如何将数据库连接到网站以通过浏览器访问。

此外，全书中有四章包含附录，讨论了技术性更强的编程概念。第2章的附录描述了可供教师和学生使用的联机数据库设计系统。它为数据库设计提供了即时反馈，使学生更容易理解问题和探究不同的设计。第3章的附录介绍了规范化的形式化定义。它是为那些想查看更多形式化的集合理论定义的教师和学生而提供的。第4章的附录是SQL基本语句的简捷列表。第5章的附录提供了程序设计简介。它可作为总结或简单备忘记录。

教学特点

本书的教学目标很明确，并且在每一章中都进行强调，那就是学习本书之后，学生应该能够使用DBMS构建商业应用程序。本书使用示例来应用书中所阐述的概念。应当鼓励学生通过解答每章的练习题和完成期末实践项目来应用知识。

每章包含若干节帮助读者对本书的理解，并帮助读者将其应用于设计和创建业务应用程序上。这些部分包括：

- *本章学习内容*：关于本章内容重点和用途的简短讨论。
- *小结*：对本章主题的简要回顾。
- *开发漫谈*：每章都以一段情景对话作为开头和结尾，对话中讲述了本章所学内容在实际中的应用。
- *关键词*：本章中介绍的术语列表。本书的最后提供了一个完整的词汇表。
- *补充读物*：关于主题的更详细研究的参考文献。
- *参考网站*：一些提供主题相关详细信息的网站。其中一些是新闻组，开发人员分享问题和技巧。
- *复习题*：作为学习指南用于备考。
- *练习*：涉及本章中出现的概念的问题。大部分需要使用DBMS。
- *项目*：在正文后面的附录中提供了若干较长的项目。它们适合作为期末实践项目。
- *工作手册*：工作手册对于第3版来说是新内容，它提供了使用特定工具构建数据库的细节工作。工作手册的每一章都举例说明了与书中所讨论内容配套的任务。工作手册还提供练习，用于为不同公司构建6个数据库。
- *示例数据库*：提供了两个示例数据库用于举例说明书中讲述的概念。Sally的宠物商店例举了一个处于早期设计阶段的数据库，而Rolling Thunder自行车提供了一个更加完整的应用程序。它由现实的数据构成。在本书各章提供了有关这两个数据库的练习。

本书特色

1. 侧重于现代业务应用程序开发。
   - 根据业务模型来阐述数据库设计。
   - 通过很多示例和练习强调动手实践应用。
   - 侧重于新型图形用户界面应用程序。

- 讲述数据库编程和应用程序开发的章节。
- 关于编程和开发细节内容的附录。

2. 热点主题。
   - 介绍并使用统一建模语言（UML）来建模和绘制系统图表。这种新标准将很快被所有设计者使用。
   - 关于数据库环境下安全主题的深入讨论。
   - 因特网和内联网的数据库开发。
   - 强调SQL 92，同时介绍SQL 99和SQL 200x的XML特性。
   - 数据库中集成的应用程序和对象。
3. 实用的业务练习和案例。
   - 很多数据库设计问题。
   - 涵盖应用程序开发所有方面的练习。
   - 适用于期末实践项目的案例。
4. 一个完整的示例数据库应用程序（Rolling Thunder自行车）。
   - 功能完善的业务数据库。
   - 示例数据和数据生成器例程。
   - 常见数据库操作的示例程序代码。
5. 用于比较和附加任务的第二个数据库（Sally的宠物商店）。
6. PowerPoint幻灯片形式的讲课记录。
7. 为特定数据库技术编制的工作手册，其中举例说明了亲手构建一个实际应用程序所需的步骤。最初，印刷好的工作手册是和微软的Access和Oracle配套的。读者与出版者和教师协商以获取对于其他系统和工具的支持。
8. 附加信息和章节。为了缩减本书的篇幅，早期的一些章节——特别是对物理数据存储（B树）的分析已经以电子文件的形式转存到随书光盘上了。

期末实践项目

各章后面的附录中介绍了很多项目。这些项目适合作为期末的实践项目。学生应当能够在一学期内构建一个完整的应用程序。最终的成绩应当侧重于这个期末实践项目。教师至少应当对两个中间阶段进行评定：（1）在学完第3章后的短期内收集一系列规范化表，（2）在学完第6章后的短期内收集设计预览，预览至少要包括两个主要表单和两个报告。工作手册中的6个额外案例也可以用做期末实践项目。

一些教师可能会选择以小组作业的方式来布置实践项目。然而，应当尽量避免这种方式，要求学生独立完成工作。实践项目是一种关键的学习工具。如果组内的一些成员逃避实践项目的工作，他们将失去一次重要的学习机会。

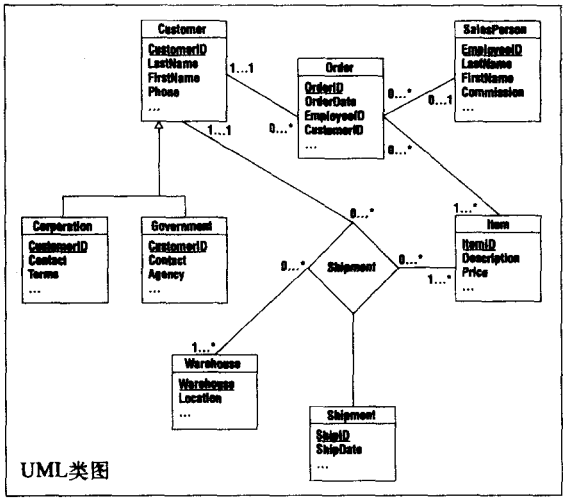
数据库设计和统一建模语言

许多年来，实体关系图在数据库设计建模方法中占有统治地位。然而，由于有很多种不同的绘图方法，这种建模方法给教师（和学生）带来了麻烦。与以往版本一样，本书使用统一建模语言（UML）方法代替传统的实体关系（ER）图作为数据库设计的建模方法，从而解决这些问题。

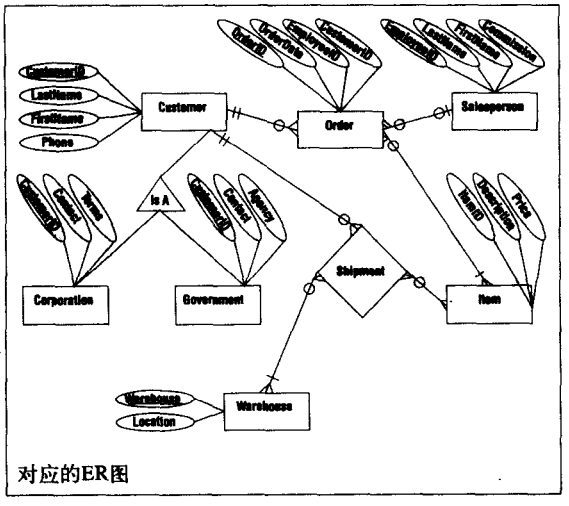
从表面上看，这种改变几乎就是将ER图的符号和术语用UML类图中的并行概念进行替代。

UML类图与ER图非常相似，但它在很多方面具有优势。首先，它们是标准化的，这样学生（和教师）只需学习一种符号集合。第二，它们因为没有传统ER图中无用的和含义模糊的符号而更容易阅读，从这种意义上说它们更“干净”。第三，它们提供对于面向对象设计的介绍，因此学生能够更好地为将来的开发做准备。第四，随着UML作为一种标准设计方法被迅速采纳，学生能够更好地为开展未来的工作做准备。很多系统设计团体已经采用UML作为设计系统的一种标准方法。UML由系统设计的主要创造者（例如Booch，Rumbaugh和Jacobsen）提供支持，同时重要的软件公司（包括IBM，Microsoft和Oracle）也对它提供支持。Mircosoft Access和SQL Server都使用一种类似于UML的绘图工具。此外，如果学生需要使用老式的ER方法，他们在迁移其UML方法的知识时将不会感到困难。

ER图和类图的基本相似点在于：（1）实体（类）用矩形绘制；（2）二元关系（联系）以连接线绘制；（3）*n*元联系（关联）用菱形绘制。这样，总体结构是相似的。UML和ER图的区别主要体现在细节上。在UML中，关联的多样性以简单的数字符号显示，而不是以一个含义模糊的图标显示。下面两图中列举了一个示例。



UML类图



对应的ER图

UML也能表示*n*元关联并允许将关联定义为类。这里能够表示所有关联的命名，包括导航名称以便于辅助读图。在一些情形下还定义了表示关联结束的图标，例如布局（ER很少进行处理）和子类型（ER中只提供了很少支持）。

UML方法的更多细节在第2章和第3章中讲解。本书中只使用很小部分的UML图、符号和术语。你可以在http://www.rational.com/uml/网站上找到完整的规范。

教学支持

- 具有多项选择题、简答题和小规模项目的测试库可以和Irwin/McGraw-Hill的电子测试库软件一起使用。
- 讲课记录和帮助以Microsoft PowerPoint幻灯片格式提供。幻灯片包含所有图片和附加记录。幻灯片以讲课形式组织，并且可以重新排列以适合个人偏好。
- 光盘上提供了许多数据库和练习。教师可以添加新数据，修改练习，或者使用它们扩充本书中的讨论。
- Sally的宠物商店数据库应用程序以后端数据库格式提供，数据库的格式包括Microsoft Access，SQL Server和Oracle。前端以Microsoft Access、Oracle、Visual Basic 6和ASP网页形式提供。在本书中大量使用了此应用程序来阐明主题。与Rolling Thunder示例相比，宠物商店示例处于设计的早期阶段，这是由于以下两个主要原因。（1）学生能够比较这两个应用程序并且洞察开发步骤；（2）可以给学生布置作业来为宠物商店应用程序添加额外的功能。
- Rolling Thunder示例数据库适用于Oracle，SQL Server和Microsoft Access，但应用程序窗体只能运行在Microsoft Access下。它是一个独立的应用程序，示例说明了很多概念并使学生能够研究一个完整数据库应用程序的众多方面，包括应用程序的代码。
- 通过互联网站点http://JerryPost.com可以与作者进行直接沟通。
- 通过互联网站点http://www.mhhe.com可以与出版者进行直接沟通。

第3版的变化

第3版中最重要的变化就是增加了工作手册。将数据库细节材料转移到工作手册中能够缩减正文的篇幅并且在工作手册中提供更详细的帮助。尽管本书仍然是以应用程序为导向的，本书还是在主体内容中采取更一般的方法来介绍较难的主题。本书将重点为学生提供设计、查询和构建数据库所需的基本技能和知识。工作手册提供详细的数据库细节帮助，并将任务转化为解决构建应用程序中的特定问题。工作手册可用于实验室环境，或者作为帮助学生设计和构建其实践项目的单独说明。

第3版中还包含关于数据仓库和数据挖掘的扩充讨论。这些概念在一个新章（第8章）中详述。第7章关于完整性和事务的介绍包含数据库触发器、事务和键生成问题重要性的扩展。构建一个应用程序的概念已经被整理并简化到第6章。详细的步骤在工作手册中进行介绍。

此外，需要注意的是大部分练习都已经替换过。旧的练习可在光盘中和作者的网站上找到。此版本在附录中包含案例，称为数据库项目。所有旧的案例都已经转移到了光盘和网站上。除提供新项目之外，本书的目的是创建稍小型的案例以便学生能够在一个学期中完成。

设计反馈工具

本书的另一个重要的新元素是拥有联机数据库设计系统。这个联机专家系统在教授数据

库设计和数据规范化方面是一个很大的进步。学生使用支持Java的网页浏览器来绘制设计图并将它们保存在中央服务器上。鼠标单击之后，服务器评定设计图并提供对设计的即时反馈。反馈以提示和问题的形式指导学生重新考虑表的设计。教师能够在几分钟之内设定班级和作业，并且可以定制问题、解决方案和成绩评定标准。教师甚至能够创建新的问题，但是系统已经包含了很多练习，包括本书三个版本中的所有问题。教师需要签约才能在课堂中使用此系统。确认网站为http://time-post.com/dbdesign。[⊖]

致谢

创建一种教授数据库管理的新方法需要很多人的努力和支持。数据库可能是一门较难但有乐趣的课程。这需要教师为改进教授本书的方法做出很大贡献。第1版和第2版的评审者花费了不少时间和精力改进此书，他们的贡献值得称赞。

我要特别感谢John Gerdes为测试库所做的工作，同时也要感谢下面这些杰出的专业人员，他们贡献了自己的时间和精力来审阅本书，才有了此版本的改进。

Bhagyavati博士（哥伦布州立大学）
Andy Borchers（凯特灵大学）
Sudip Bhattacharjee（康涅狄格大学）
Subhasish Dasgupta（乔治华盛顿大学）
Sherif Elfayoumy博士（北佛罗里达大学）
Sudha Ram博士（亚利桑那大学）
John Gerdes, Jr.（加利福尼亚大学）
Karlene Sanborn博士（富兰滋大学）
Allen Gray（Loyola玛丽蒙特大学）
Ashraf Shirani（圣琼斯州立大学）
Jim Kattke（奥古斯堡学院）
Leon Zhao（亚利桑那大学）
T. M. Rajkumar（迈阿密大学）

我还要深深感谢以下这些教师，他们仔细思考并大力支持将本书第3版组织成核心课本加两本工作手册的形式：

Barbara Beccue（伊利诺伊州立大学）
Steve Rau（玛魁特大学）
Jim Chen（圣克劳得州立大学）
Werner Schenk（罗切斯特大学）
Mike Collins（海波因特大学）
Richard Segall（阿肯色州立大学）
Donald Dawley（迈阿密大学，牛津）
Tarun Sen（维吉尼亚工艺学院）
Carlos Ferran（罗切斯特理工学院）
Conrad Shayo（加州州立大学，圣贝纳迪诺）
Philip Friedlander（圣彼得斯堡学院，Clearwater）
Monica Garfield（南佛罗里达大学，坦帕）
Kazem Taghva（拉斯维加斯内华达大学）
Marilyn Griffin（维吉尼亚工艺学院）
S. Varden（裴斯大学）
Jeff Guan（路易斯维尔大学）
James Yao（Montclair州立大学）
John Molluzzo（裴斯大学）
Brian Zelli（布法罗纽约州立大学）
Anne Nelson（海波因特大学）
Larisa Preiser（加州州立职业工艺大学）

与McGraw-Hill/Irwin的工作人员合作是一件很愉快的事情。Paul Ducham的指导及其积极尝试新思路对于编写本书是至关重要的。Charlie Fisher和Greta Kleinert的支持，设计和生产人员对细节问题的无比关注铸就了更好的书籍。全体工作人员使我的工作变得更容易和更快乐。

⊖ 请与麦格劳－希尔教育出版公司教师服务中心联系。具体联系方式请见本书最后所附的“教学服务沟通表”。——编辑注

目　录

出版者的话
专家指导委员会
译者序
前言
第1章　简介……1
1.1　开发漫谈……1
1.2　简介……1
1.3　数据库和应用开发……2
1.4　数据库管理系统的组成……3
1.4.1　数据库引擎……3
1.4.2　数据字典……4
1.4.3　查询处理器……5
1.4.4　报表编写器……6
1.4.5　表单生成器……6
1.4.6　应用生成器……8
1.4.7　通信与集成……8
1.4.8　安全性与其他工具……9
1.5　使用数据库管理系统的优势……9
1.5.1　集中精力于数据……10
1.5.2　数据独立性……10
1.5.3　数据独立性与客户/服务器系统……11
1.6　重要的商用数据库……11
1.7　数据库管理系统简史……12
1.7.1　层次数据库……12
1.7.2　网状数据库……12
1.7.3　关系数据库……13
1.7.4　面向对象数据库……14
1.8　应用开发……17
1.9　Sally的宠物商店……18
1.10　Rolling Thunder自行车……18
1.11　可行性研究……19
1.11.1　成本……19
1.11.2　效益……20
小结……20
关键词……21
复习题……21
练习……22
参考网站……23
补充读物……24

第一部分　系统设计

第2章　数据库设计……26
2.1　开发漫谈……26
2.2　简介……26
2.3　开始设计之前……28
2.4　设计数据库……28
2.4.1　确定用户需求……28
2.4.2　业务对象……29
2.4.3　表和关系……29
2.4.4　定义……30
2.4.5　主码……30
2.5　类图……31
2.5.1　类和实体……31
2.5.2　关联和关系……32
2.5.3　类图细节……32
2.6　Sally的宠物商店类图……38
2.7　数据类型（域）……40
2.7.1　文本……40
2.7.2　数值……41
2.7.3　日期和时间……42
2.7.4　二进制对象……43
2.7.5　计算值……43
2.7.6　自定义类型（域/对象）……43
2.8　事件……44
2.9　大型项目……45
2.10　Rolling Thunder自行车……46
2.11　应用设计……51
小结……51
关键词……52
复习题……52

练习 ……53
参考网站 ……57
补充读物 ……57
附录：数据库设计系统 ……57
第3章 数据规范化 ……65
3.1 开发漫谈 ……65
3.2 简介 ……65
3.3 表、类和码 ……66
3.3.1 复合码 ……66
3.3.2 代理码 ……67
3.3.3 标记 ……67
3.4 音像店的示例数据库 ……70
3.4.1 初始对象 ……71
3.4.2 初始表单评估 ……72
3.4.3 重复部分的问题 ……73
3.5 第一范式 ……75
3.5.1 重复组 ……75
3.5.2 嵌套重复组 ……76
3.6 第二范式 ……76
3.6.1 第一范式的问题 ……77
3.6.2 第二范式的定义 ……78
3.6.3 依赖 ……79
3.7 第三范式 ……80
3.7.1 第二范式的问题 ……80
3.7.2 第三范式的定义 ……81
3.7.3 检查你的工作 ……83
3.8 超越第三范式 ……84
3.8.1 Boyce-Codd范式 ……84
3.8.2 第四范式 ……85
3.8.3 域－码范式 ……85
3.9 数据规则和完整性 ……87
3.10 业务规则的影响 ……88
3.11 将类图转化为规范化的表 ……90
3.11.1 一对多关系 ……91
3.11.2 多对多关系 ……92
3.11.3 多重关联 ……93
3.11.4 概括或子类型 ……94
3.11.5 组合 ……94
3.11.6 自反关联 ……95
3.11.7 小结 ……95
3.11.8 Sally的宠物商店示例 ……96
3.12 视图集成 ……98
3.12.1 Sally的宠物商店示例 ……99
3.12.2 Rolling Thunder示例中的集成问题 ……100
3.13 数据字典 ……105
3.13.1 DBMS表定义 ……106
3.13.2 数据量与使用率 ……109
小结 ……111
关键词 ……112
复习题 ……112
练习 ……112
参考网站 ……119
补充读物 ……119
附录：规范化的形式化定义 ……120

第二部分 查 询

第4章 数据查询 ……126
4.1 开发漫谈 ……126
4.2 简介 ……126
4.3 查询语言的三个任务 ……127
4.4 检索数据的四个问题 ……127
4.4.1 你想得到什么结果 ……128
4.4.2 已经知道什么 ……128
4.4.3 涉及哪些表 ……128
4.4.4 如何连接表 ……128
4.5 Sally的宠物商店 ……128
4.6 版本差异 ……129
4.7 基本查询 ……129
4.7.1 单表 ……130
4.7.2 SQL介绍 ……130
4.7.3 输出排序 ……131
4.7.4 关键字Distinct ……132
4.7.5 条件 ……132
4.7.6 布尔代数 ……133
4.7.7 德摩根定律 ……134
4.7.8 有用的WHERE子句 ……137
4.8 计算 ……137
4.8.1 基本运算 ……137
4.8.2 聚集运算 ……138
4.8.3 函数 ……139

4.9 部分和与GROUP BY语句 ……140
4.9.1 求和条件（HAVING） ……141
4.9.2 WHERE子句与HAVING子句 ……142
4.9.3 最好和最差 ……142
4.10 多表 ……143
4.10.1 连接表 ……143
4.10.2 标识不同表中的列 ……145
4.10.3 连接多张表 ……145
4.10.4 表连接提示 ……146
4.10.5 表的别名 ……148
4.10.6 创建视图 ……148
小结 ……150
关键词 ……150
复习题 ……151
练习 ……151
参考网站 ……154
补充读物 ……154
附录：SQL语法 ……154
第5章 高级查询和子查询 ……157
5.1 开发漫谈 ……157
5.2 简介 ……157
5.3 Sally的宠物商店 ……158
5.4 子查询 ……158
5.4.1 计算或简单查找 ……158
5.4.2 子查询和数据集合 ……159
5.4.3 带有ANY和ALL的子查询 ……160
5.5 差：NOT IN ……161
5.6 外连接 ……162
5.7 关联子查询存在危险 ……164
5.8 SQL SELECT的更多特征和技巧 ……166
5.8.1 UNION、INTERSECT和EXCEPT …166
5.8.2 多JOIN列 ……167
5.8.3 自反连接 ……167
5.8.4 CASE函数 ……168
5.8.5 不等连接 ……169
5.8.6 带有“每一个”的查询需要EXISTS子句 ……169
5.8.7 SQL SELECT小结 ……170
5.9 SQL数据定义命令 ……171
5.10 SQL数据操纵命令 ……173
5.10.1 INSERT和DELETE ……173
5.10.2 UPDATE ……174
5.11 质量：查询检查 ……175
小结 ……176
关键词 ……177
复习题 ……177
练习 ……178
参考网站 ……181
补充读物 ……181
附录：编程简介 ……181

第三部分 应 用

第6章 表单、报表和应用 ……190
6.1 开发漫谈 ……190
6.2 简介 ……190
6.3 报表和表单的有效设计 ……191
6.3.1 人性化设计 ……191
6.3.2 Windows控件 ……193
6.3.3 用户界面——网络要点 ……195
6.3.4 用户界面——访问问题 ……195
6.4 表单布局 ……195
6.4.1 表格表单 ……196
6.4.2 单行或单列表单 ……196
6.4.3 子表单 ……198
6.4.4 导航表单 ……199
6.5 建立表单 ……200
6.5.1 可更新的查询 ……200
6.5.2 连接表单 ……201
6.5.3 属性和控件 ……201
6.5.4 表单上的控件 ……202
6.5.5 多表单 ……205
6.5.6 国际属性 ……206
6.6 直接操作图形对象 ……207
6.6.1 Sally的宠物商店示例 ……207
6.6.2 因特网 ……208
6.6.3 图形方式的复杂性和局限性 ……208
6.7 报表 ……209
6.7.1 报表设计 ……209
6.7.2 术语 ……210
6.7.3 基本报表类型 ……211

6.7.4 图表 ……214
6.8 应用软件的功能……215
6.8.1 菜单和工具栏 ……216
6.8.2 定制帮助 ……217
小结……221
关键词……221
复习题……222
练习……222
参考网站……223
补充读物……224
第7章 数据库完整性和事务 ……225
7.1 开发漫谈……225
7.2 简介……225
7.3 过程语言……226
7.3.1 代码应该放在哪里 ……226
7.3.2 用户定义的函数 ……227
7.3.3 查找数据 ……228
7.4 数据触发器……228
7.4.1 语句与行触发器 ……229
7.4.2 利用触发器取消数据更新 ……230
7.4.3 级联触发器 ……230
7.4.4 INSTEAD OF触发器 ……232
7.5 事务……232
7.5.1 事务的例子 ……232
7.5.2 事务的开始和结束 ……232
7.5.3 保存点 ……233
7.6 多用户与并发访问……234
7.6.1 悲观锁：串行化 ……235
7.6.2 多用户数据库：并发访问与死锁 …236
7.6.3 乐观锁 ……237
7.7 事务的ACID特征 ……238
7.8 码生成……240
7.9 数据库游标……241
7.9.1 游标基础 ……241
7.9.2 可滚动的游标 ……241
7.9.3 利用游标修改或删除数据 ……243
7.9.4 带参数的游标 ……244
7.10 Sally的宠物商店的存货清单 ……244
小结……247
关键词……248
复习题……248
练习……249
参考网站……251
补充读物……251
第8章 数据仓库和数据挖掘 ……252
8.1 开发漫谈……252
8.2 简介……252
8.3 索引……253
8.3.1 二分查找 ……253
8.3.2 指针和索引 ……254
8.3.3 位图索引和统计方法 ……254
8.3.4 索引的问题 ……255
8.4 数据仓库和联机分析处理……256
8.4.1 数据仓库的目标 ……256
8.4.2 数据仓库的问题 ……257
8.5 OLAP的概念 ……258
8.6 OLAP数据库设计 ……259
8.6.1 OLAP数据分析 ……261
8.6.2 SQL中的OLAP ……263
8.6.3 SQL分析函数 ……266
8.6.4 SQL的OLAP窗口 ……266
8.7 数据挖掘……269
8.7.1 分类 ……269
8.7.2 关联规则/购物篮分析……270
8.7.3 聚类分析 ……272
8.7.4 地理分析 ……272
小结……273
关键词……274
复习题……274
练习……275
参考网站……276
补充读物……276

第四部分 数据库管理

第9章 数据库管理与安全 ……278
9.1 开发漫谈……278
9.2 简介……278
9.3 数据管理员……279
9.4 数据库管理员……280
9.5 数据库结构……281

9.6 元数据……282
9.7 开发阶段的数据库任务……282
9.7.1 数据库规划……283
9.7.2 数据库设计……283
9.7.3 数据库实现……283
9.7.4 数据库运行和维护……284
9.8 备份和恢复……284
9.9 安全和隐私……286
9.9.1 数据隐私……286
9.9.2 威胁……287
9.9.3 物理安全……287
9.9.4 管理控制……288
9.9.5 逻辑安全……288
9.9.6 职责分割……292
9.9.7 软件升级……293
9.10 加密……294
9.11 Sally的宠物商店……295
小结……296
关键词……297
复习题……297
练习……298
参考网站……300
补充读物……300
第10章 分布式数据库和因特网……301
10.1 开发漫谈……301
10.2 简介……301
10.3 Sally的宠物商店……302
10.4 分布式数据库……302
10.4.1 目标和规则……303
10.4.2 优点和应用……304
10.4.3 创建分布式数据库系统……305
10.4.4 分布式查询处理……306
10.4.5 数据复制……307
10.4.6 并发、锁和事务……308
10.4.7 独立的事务管理器……309
10.4.8 分布式设计问题……310
10.5 客户/服务器数据库……310
10.5.1 客户/服务器与文件服务器……311
10.5.2 三层客户/服务器模型……312
10.5.3 后端：服务器数据库……313
10.5.4 前端：Windows客户端……314
10.5.5 在客户端维护数据库的独立性……315
10.6 电子商务数据库……316
10.7 作为客户/服务器系统的Web……317
10.7.1 受限的HTML客户端……317
10.7.2 Web服务器数据库基础……319
10.8 应用中的数据传输问题……321
10.9 XML：将数据传输到不同的系统……323
10.10 Java语言和JDBC……326
小结……326
关键词……328
复习题……328
练习……328
参考网站……330
补充读物……330
词汇表……331

第 1 章

简　介

本章学习内容

- 什么是数据库?
- 什么是数据库管理系统?
- 数据库管理系统的主要组件是什么?
- 使用数据库管理系统的优势是什么?
- 数据库管理系统的主要类型有哪些?

1.1 开发漫谈

Miranda：我叔叔刚刚打电话给我，说他的公司经营得很艰难。公司需要找个人为销售部写一个应用。公司需要一个便携式电脑系统，每个销售人员可以用它来输入订单。系统需要随时跟踪订单状态并且生成通知和周报。我叔叔说我应该去做这份工作，因为我在电脑方面懂得很多。他的公司愿意支付6000美元，并且我可以兼职。

Ariel：哇！听起来似乎是个不错的工作。有什么困难吗?

Miranda：唔，我会使用基本的计算机工具，也会一点点编程，但是我不确定我能不能创建一个完整的应用。这可能会花费很长时间。

Ariel：为什么不使用数据库管理系统呢？这要比从零开始写程序容易很多。

Miranda：真是这样吗？数据库系统能做些什么？怎么做?

1.2 简介

你想建立计算机管理的商业应用吗？你想创建能在多个地点运行的商业应用吗？你想在互联网上管理业务吗？你想让用户在网上提交订单吗？如果想创建一个现代化的商业应用，你就需要数据库管理系统。

现代化的数据库系统是建立商业应用最强有力的工具之一。它具有很多功能，相比传统的编程方法具有巨大的优势。但它十分复杂。为了实现这些优势，数据必须进行精心地组织。要获取数据并建立应用，你必须学会使用一种强大的查询语言。一旦你了解了数据库设计、查询和建立应用的概念，你就能够创建复杂的应用，而所花费的时间只是传统编程技术所需时间的一小部分。

数据库是以标准格式存储的数据的集合，一般设计成多用户共享方式。数据库管理系统（DBMS）是一种特殊的软件，它定义数据库、存储数据、支持某种查询语言、生成报表并且创建数据项显示界面。

创建应用过程中最复杂的一些问题来源于存储和获取数据。这些问题包括节约空间，快速获取数据，多用户同时共享数据以及提供数据备份和恢复。起初，对创建的每一个应用，程序员都不得不处理这些问题。如今，DBMS已经为这些问题提供了一个最好的解决办法，将数据库作为应用的基础，意味着你可以获得所有强大的功能和安全性而不需要做太多额外工作。

1.3 数据库和应用开发

在过去的几年间，数据库系统已经成为几乎所有应用开发项目的基础。从大型的企业关系系统，到电子商务网站、独立的商业应用。数据库系统高效地存储和获取数据，保障安全性，并使得建立应用更加容易。现在，当建立或修改一个应用的时候，你会首先创建数据库。要理解DBMS的能力和如何利用它们来创建应用，最好研究一下开发应用的过程。

企业组织一般遵循图1-1中描述的基本步骤来开发科技应用。大型项目的每个阶段可能需要几个人，而小一些的项目可能只需要一两个开发人员就能完成所有任务。企业组织可以依据每一步骤重新安排任务，但是整个项目要成功，必须完成所有的任务。在可行性步骤中要定义项目和提供成本估算。在分析阶段，系统分析员从用户那里收集数据定义、表单和报表，用于设计数据库和所有的新表单、报表以及用户交互。在开发阶段，要创建表单、报表和应用功能，例如帮助文档。执行阶段大体上包括传输数据、安装、训练和评价。

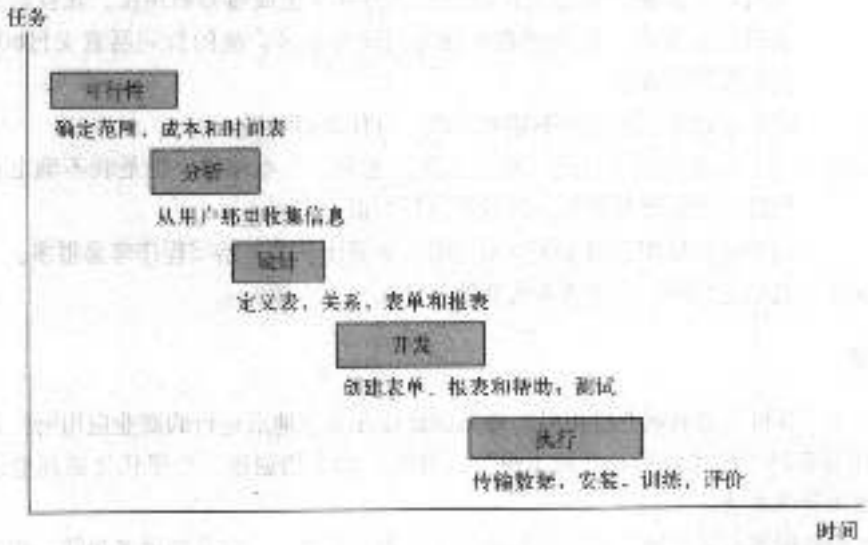


图1-1 系统开发。尤其对于大型项目，将应用开发分解为独立的步骤是非常有用的。这样能够跟踪开发团队的进度并且突出尚未完成的步骤。对于某些项目而言，一些任务可能重叠或迭代，但不应跳过任何步骤

对于数据库驱动的应用，设计阶段是至关重要的。数据库系统及其相关的开发工具极其强

大，而数据库必须精心设计才能有效地利用。图1-2显示了业务规则和流程被转化为数据库表和关系的定义。表单用于向数据库传输数据，而报表用于通过查询获取并显示用户所需的数据。这些表单、报表连同诸如菜单和帮助这样的功能组成了应用。用户通常看到的只是应用程序界面而非底层的数据库或表。

在创建数据库应用的过程中，设计数据库表和关系是一个关键步骤。定义表的过程和规则将在第2章和第3章进行详细介绍。使用数据库需要获取和操纵数据。这些工作由查询系统完成，将在第4章和第5章进行描述。有了这些基础，使用工具创建表单、报表并将其集成到应用当中就相对简单了，这些将在第6章进行讨论。

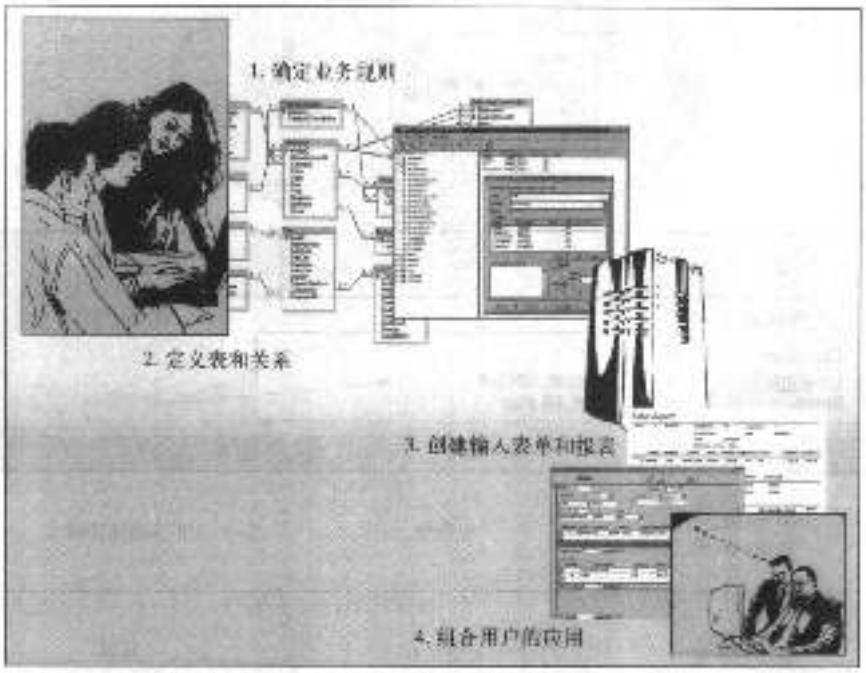


图1-2 数据库设计的步骤。业务规则和数据用于定义数据库表。表单用于输入新数据，数据库系统获取数据来回答查询并生成报表。用户只能以表单和报表的形式查看应用

1.4 数据库管理系统的组成

通过DBMS提供的一般组件可以了解DBMS的价值。当你需要评价不同的产品，确定你的公司应该使用哪种DBMS时，本节的基本组件列表将会十分有用。每个DBMS都有自己的长处和短处。你可以依据各自组件运行情况的好坏来评价各种产品。DBMS的评价取决于数据库引擎，数据字典，查询处理器，报表编写器，表单生成器，应用生成器，通信与集成及安全性。

1.4.1 数据库引擎

数据库引擎是DBMS的核心，负责存储，获取和更新数据。这个组件对性能（速度）和处理大型问题的能力（可度量性）影响最大。其他组件依赖于引擎来存储应用数据和定义应用

如何操作系统内部的数据。图1-3说明了数据库引擎和数据表的基本关系。

对于某些系统，数据库引擎是一个独立的组件，可以购买并作为独立的软件模块使用。例如，Microsoft的"jet engine"构成了Access的基础。类似地，Oracle和Microsoft SQL Server的数据库引擎也可以单独购买。

数据库引擎还负责强制执行关于数据的业务规则。例如，大部分业务不允许在数据库中使用负数价格。一旦设计者创建了那条规则，数据库引擎便会警告用户并阻止其输入负值。

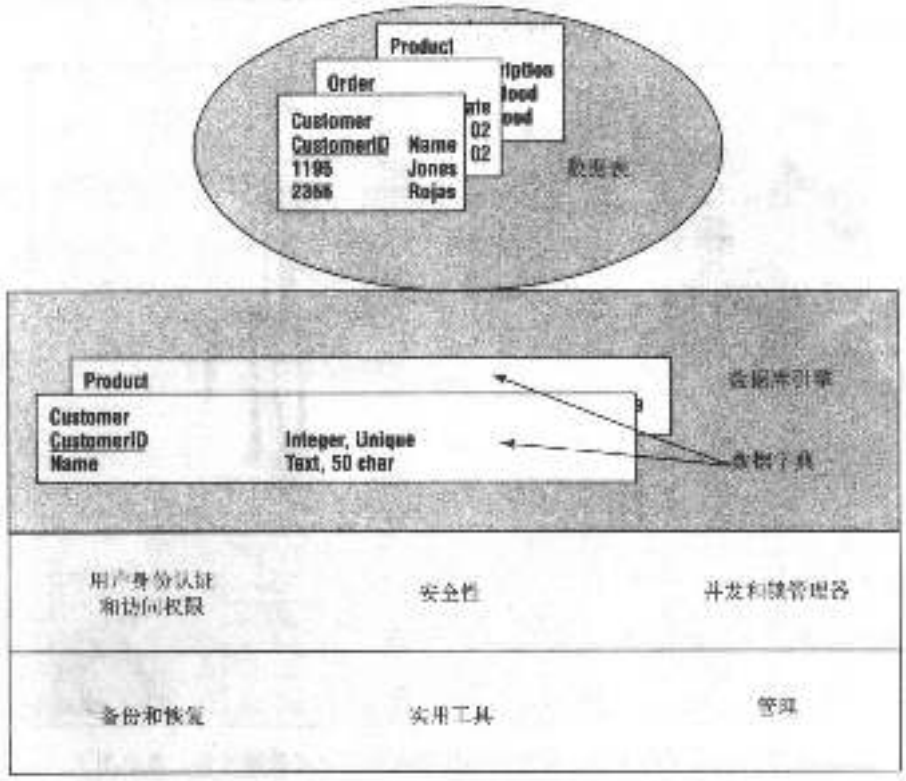


图1-3 数据库引擎。引擎负责定义、存储和获取数据。引擎的安全子系统确认用户身份认证并控制对数据的访问权限

如图1-4所示，数据库引擎将数据存储在精心设计的表中。表的名称能反映表中存储的数据。列代表能描述对象的简单属性，例如员工的姓名、电话号码和地址。每一行代表表中的一个对象。

数据库性能是一个重要的问题。应用的速度取决于硬件、DBMS软件、数据库设计和存储数据的方式。第9章会讨论提高数据库应用性能的一些常用方法，例如索引等。

1.4.2 数据字典

数据字典保存所有数据表的定义。它描述存储数据的类型，允许DBMS跟踪数据，并帮助开发者和用户找到他们需要的数据。大部分现代化的数据库系统将数据字典作为一个系统表的集合进行存储。例如，Microsoft Access将所有表的清单保存在一个称为MsysObjects的隐藏

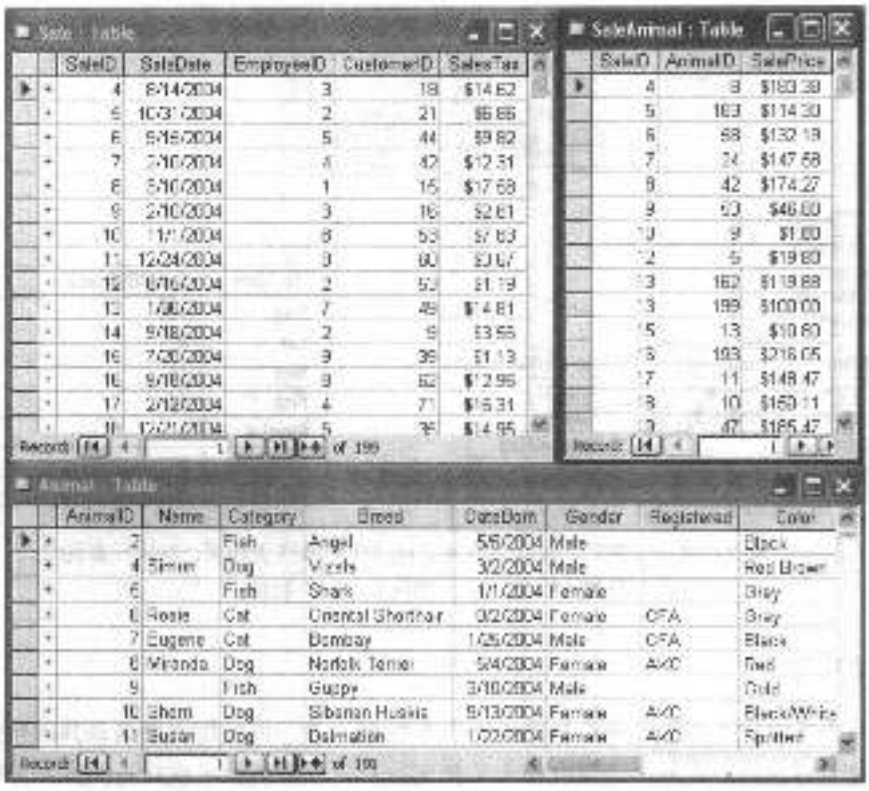


图1-4 Access中的数据库表。表中存储了关于一个业务实体的数据。例如，Animal表中的每一行存储关于某种动物的数据

系统表中。Oracle包含许多系统表，它们由数据字典提供信息。例如，sys.dba_tables存储了关于数据库中表的数据。

这些表由系统使用，很少需要直接使用它们。DBMS会提供其他工具帮助你分析数据库的结构。例如，Oracle、SQL Server、Access和DB2提供了一个管理工具用于列出所有的表，还提供了一个直观的可视化工具来帮助你输入列名、选择数据类型和其他属性。

1.4.3 查询处理器

查询处理器是DBMS的基本组件。它使开发者和用户能存储和获取数据。在某些情况下，查询处理器是你与数据库的惟一连接，即，所有的数据库操作都可以通过查询语言来执行。第4章和第5章将介绍查询语言的特征和功能——尤其是标准SQL。

查询来源于业务问题。由于自然语言诸如英语太含糊，不能够用于查询，因而查询语言是必不可少的。为了将通信问题最小化，并确保DBMS明白你的查询，你应当使用一种比英语更精确的查询语言。如图1-5所示，DBMS依据数据字典创建一个查询。当查询运行时，DBMS的查询处理器与数据库引擎共同工作来查找合理的数据。然后，查询结果格式化并显示在界面上。

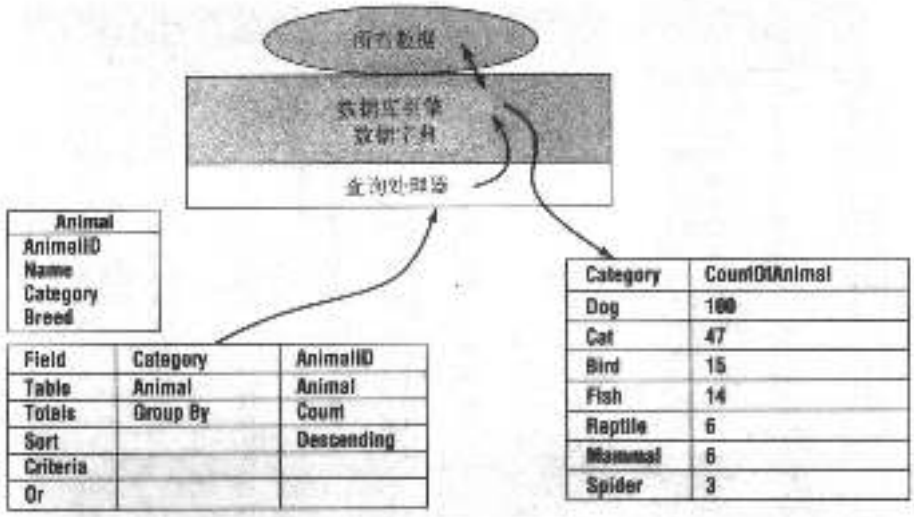


图1-5 数据库查询处理器。数据字典决定应该使用哪些表和列。当执行查询时，查询处理器与数据库引擎通信以便获取所需的数据

1.4.4 报表编写器

大部分商业用户希望以某种类型的报表查看数据汇总。很多报表遵循公共的格式。现代化的报表编写器允许你在界面上建立报表，指定项目应该如何显示或计算。大部分操作可通过往界面上拖放数据来完成。专业级的编写器能让你在短时间内完成复杂的报表，而不必编写任何代码。第6章将介绍若干种通用的商业报表，并描述如何使用数据库报表编写器来创建它们。

报表编写器可以集成到DBMS中，也可以作为一个独立的应用，开发者用它生成代码，创建所需的报表。如图1-6所示，开发者创建了一个基本的报表设计。此设计大体上基于一个查询。在报表生成过程中，报表编写器将查询传递给查询处理器，后者与数据库引擎通信，获取所需的数据行。然后，报表编写器依据报表模板格式化数据，并为报表添加页码、页眉和页脚，完成整个报表。

图1-7显示了Oracle（一个DBMS厂商）提供的报表编写器，连同该DBMS的表单和报表工具。报表编写器生成的报表可分配给其他用户使用。你可以设立报表上的所有部分并从数据库中获取要显示的数据。报表编写器具有执行计算和格式化列的功能。你还可以控制报表的颜色，在报表上放置图片（例如商标），绘制线段和其他图形，使报表更加美观或者引起用户对特定部分的注意。

1.4.5 表单生成器

表单生成器或输入界面帮助开发者创建输入表单。如第6章所述，表单生成器的目的在于创建用户公用的表单以方便用户输入数据。表单可以包含图表和图片。表单生成器使开发者能在屏幕上拖放项目的方式构造表单。图1-8说明要在表单上显示数据，查询处理器起了很大的作用。

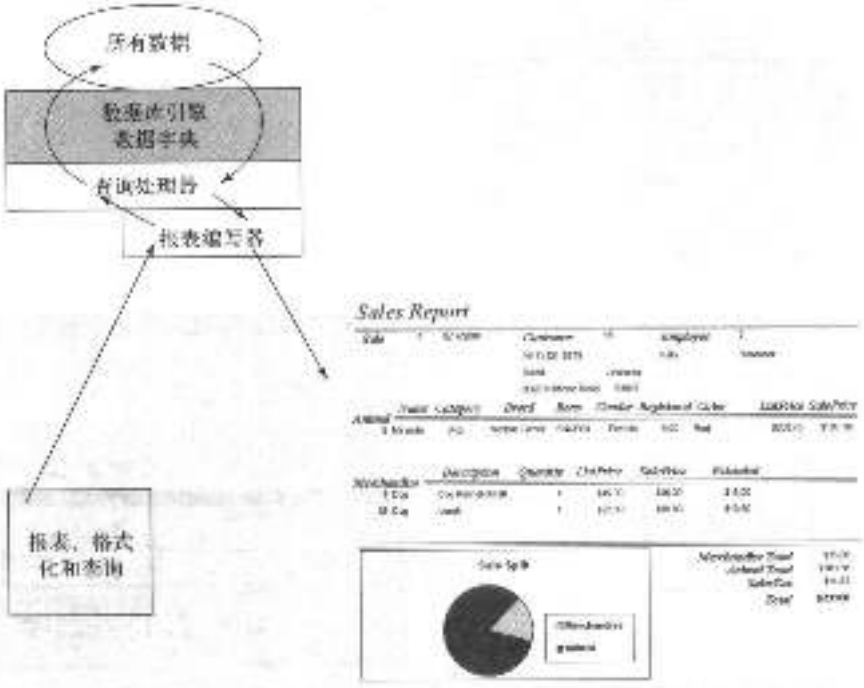


图1-6 数据库报表编写器。设计模板设定了报表的内容和布局。报表编写器通过查询处理器获取所需数据，然后格式化并打印报表

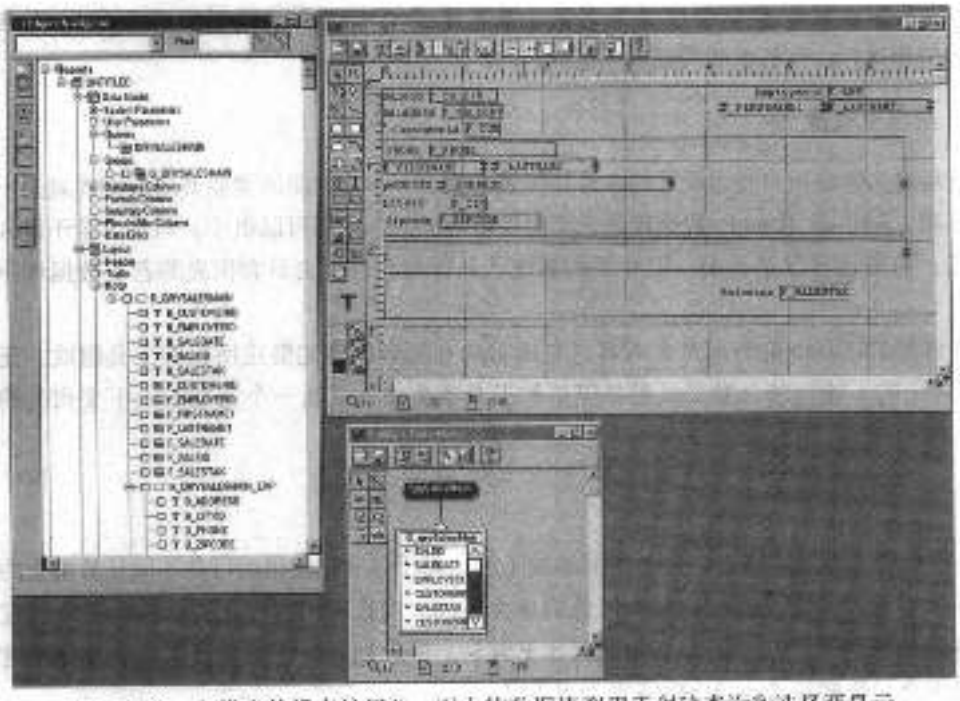


图1-7 Oracle报表的报表编写器。图中的数据模型用于创建查询和选择要显示的数据。然后报表编写器创建基本的报表布局，你可以修改设计和添加功能来改进设计或突出显示某些部分



图1-8 数据库表单。表单用于收集数据。它按照符合用户的操作来设计，方便数据输入和信息查找。查询处理器用于获取相关数据并填充组合框中可供选择的数据

很多数据库系统还提供对使用传统的、第三代语言（3GL）访问数据库的支持。使用此种方法编写程序和访问数据的问题将在第7章中介绍。

1.4.6 应用生成器

应用是为特定用户需求设计的表单和报表的集合。本书附带的宠物商店数据库就是一个应用的基础，Rolling Thunder数据库是一个更完整的应用。应用可以很小，只包含若干输入表单和报表，应用也可以是很大、很复杂的系统，从包含几百个表单和报表的若干数据库中集成数据。

高质量的DBMS包含应用生成器，它由帮助开发者创建完整应用包的工具组成。正如第6章将讨论的，流行的开发工具包括菜单和工具条生成器以及一个完整的上下文相关的帮助系统。

1.4.7 通信与集成

有些数据库系统提供专门的通信与集成工具，用于存储和使用运行在不同计算机上的多个数据库中的数据，即使这些计算机所处的地点不同。现代化的操作系统和包括互联网在内的独立网络使连接运行在不同地点的数据库更容易。尽管如此，一些数据库系统在使用这些工具和提供全局数据共享连接上做得更加出色。

图1-9显示了现代化的DBMS如何利用附加组件跨越通信网络与其他计算机共享数据。3GL连接器提供了用传统语言（Visual Basic、COBOL、C++等）连接数据库引擎的方式。

开发者既可以拥有这些语言的强大功能和灵活性，还能够使用查询处理器获取和存储数据库中的数据。

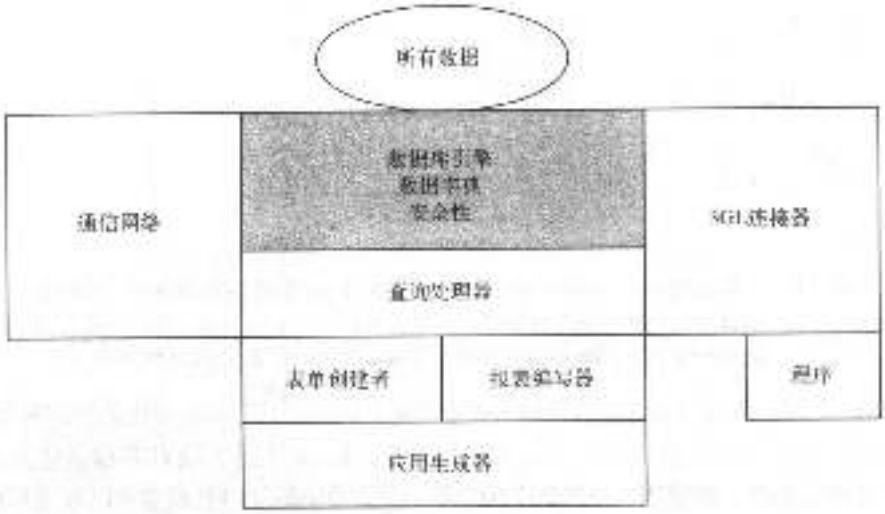


图1-9 数据库组件。通信网络连接不同计算机上的数据库。3GL连接器使开发者能够使用传统编程语言访问数据库。应用生成器用于建立包含菜单和帮助文件的完整应用

1.4.8 安全性与其他工具

数据库的一个主要目标是多用户共享数据，因此，DBMS必须负责建立和维护安全访问控制。第10章将讨论如何为单个用户或用户组授予特权，以及如何将他们对数据库的操作限制在特定的范围内。

对于运行在个人电脑上的数据库而言，安全性是一个复杂的问题，因为大部分个人电脑的操作系统只有很少的控制。因此，DBMS必须对安全性的更多方面负责。特别地，DBMS必须确认用户，然后允许或限制其对数据库的不同部分进行访问。

DBMS提供了各种管理工具，将在第10章进行讨论。通常的功能包括备份和恢复、用户管理，数据存储评价和性能监测工具。

1.5 使用数据库管理系统的优势

很多商业应用需要相同的功能（有效的数据存取、多用户共享数据、安全性等）。相比在每个应用程序内部重新创建这些功能，购买一套包含这些基本功能的数据库管理系统更有意义。这样开发者就可以集中精力解决业务问题。DBMS的主要优势如图1-10所示。

首先，DBMS能有效存储数据。正如第2章和第3章将要描述的，如果你依据某些基本规则建立数据库，数据的存储将浪费最少的空间。另外，数据能够迅速获取，以便回答任何查询。尽管这两个目标似乎很明显，如果你每次都要从零开始写程序处理，将会十分复杂。

DBMS还能以最小的代价维护数据的一致性，当你定义数据时，大多数系统允许你创建基本的业务规则。例如，价格应始终大于零。这些规则强制作用于每一个表单、用户或访问数

据的程序。使用传统的方式编程，你必须强迫每个人遵守相同的规则。此外，这些规则放置在成千上万个不同程序文件中——这使得在业务改变时，查找和修改变得十分困难。



图1-10　DBMS的优势。DBMS为数据存取中的基本问题提供解决办法。通过使用DBMS处理数据存储问题，程序员可以集中精力开发应用——这样能够节约开发新系统所需的时间和经费，简化对已有应用的维护

DBMS，尤其是查询语言，使得数据集成更容易。例如，一个应用收集关于顾客销售数据，另一个应用收集关于顾客退货数据。如果程序员创建不同的程序和文件存储这些数据，合并数据将会十分困难的。相反地，如果使用DBMS，数据库中的任何数据都可以方便地获取、合并和使用查询系统进行比较。

1.5.1　集中精力于数据

使用原始的程序文件方法时，开发者将精力集中于过程和程序上。开发者在启动一个项目时首先会问这样的问题：程序应该如何组织？需要进行哪些计算？而数据库方法将精力集中于数据上。开发者现在启动项目时会问：需要收集哪些数据？这种改变不仅仅是技术性的。它改变了整个开发过程。

花一点时间考虑一下开发过程，哪个部分最常改变：程序（表单和报表）还是数据？对，公司随时都在收集新数据，而数据的结构是不常改变的。当它确实要改变时，通常是由于你要添加新的元素——比如手机号码。另一方面，用户需要经常对表单和报表进行修改。

如图1-11所示，数据库方法将精力集中在数据上。DBMS负责定义、存储和获取数据。所有的数据请求都必须经过数据库引擎。因此，DBMS负责数据的有效存取、并发性、数据安全性等等。精心定义数据结构之后，附加的工具诸如报表编写器、表单生成器和查询语言会使商业应用的开发更方便和快捷。

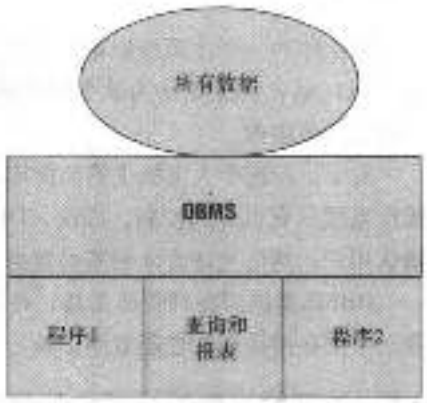


图1-11　DBMS将精力集中放在数据上。首先，定义数据。然后所有的查询、报表和程序都要通过DBMS访问数据。通常DBMS处理诸如并发性和安全性这类公共问题

1.5.2　数据独立性

将精力集中于数据的另外一个重要特征是将数据定义从程序中分离出来——即数据独立性。数据独立性使你能改变数据定义而不必修改程序。类似地，数据可以转移到新硬件或一

台完全不同的计算机上。只要DBMS知道如何访问数据，就不必改变表单、报表或使用那些数据的程序。可以修改独立的程序而不必改变数据定义。

这种理想主义的描述有一些例外情况。显然，如果删除了整个数据库结构，那么部分应用程序将不会正常工作。同样，如果对数据定义做了根本的改变——比如将电话号码的存储类型从数值类型改为文本类型——你可能必须修改报表和表单。然而，一个合理设计的数据库很少需要这种根本的改变。

考虑向Employee表中添加手机号码。图1-12显示了员工表的部分数据定义。无论存在多少表单、报表或程序，步骤是相同的。只需简单地在表定义中插入手机号码字段CellPhone。已有的查询、表单、报表和程序会和以前一样正常运行。当然，它们会忽略这个新增加的手机号码字段。如果你想在报表上看到这个新字段的值，你得添加它。对于新型的报表编写器来说，这种修改可能简单到只需将手机号码字段CellPhone拖放到表单或报表的适当位置。

字段名称	数据类型	描 述
EmployeeID	Number	Autonumber . . .
TaxpayerID	Text	Federal ID
LastName	Text	
FirstName	Text	
. . .		
Phone	Text	
. . .		
CellPhone	Text	Cellular . . .

图1-12 向Employee表中添加手机号码字段。向表中添加新元素不会影响已有的查询、报表、表单或程序

将注意力集中于数据和精心的设计能够避免在传统程序文件方法中遇到的问题。将公共数据库的方法合并到一个应用中使得专家能够创建功能强大的数据库管理系统，也使得应用程序员不必集中精力建立解决商业问题的应用。

1.5.3 数据独立性与客户/服务器系统

过去10年间开发的不断强大的个人计算机开辟了很多设计和建立商业应用的新方法。其中最重要的是客户/服务器模型。数据库管理系统在创建客户/服务器系统中起到了重要作用。在一个客户/服务器的例子中，数据存储在一台中央计算机的DBMS中。分散的个人计算机运行一个前端应用，从服务器获取并显示数据。

数据独立性的能力在于客户端应用本质上独立于数据库。开发者不必修改数据库就能创建新的应用。同样，他们也可以扩充数据库甚至将其转移到多个服务器上，而不必修改应用程序。用户仍旧使用他们所熟悉的个人计算机应用。开发者保持对数据的控制。为了保护数据DBMS可以监测并强制安全性和完整性条件，而仍然赋予授权用户访问权限。第10章将更详细地讨论分布式数据库系统的使用，包括在万维网上构建客户/服务器系统。

1.6 重要的商用数据库

重要的数据库系统包括DB2、Oracle、CA-Ingres、SQL Server和Informix。IBM、Oracle和CA-Ingres提供可用于不同平台的工具，从大型、中型到个人计算机。Informix数据库系统还支持各种硬件平台，但它主要运行在UNIX操作系统的计算机上。很多数据库包可供个人计算机使用，但大部分生产厂商将其限制在开发小型应用上。Microsoft Access是一个小型系统数据库的普通例子，而Microsoft还提供了一个SQL Server的低价版本，称为MSDE。MSDE

提供与SQL Server相同的功能，但它仅限五个用户。Oracle提供了单机使用的Personal Oracle。

选择数据库系统可能会是一个难题。很多大型企业统一使用某一厂商的产品，通过谈判降低许可证的价格，并使之能够用于企业内的所有项目中。然而，如果你需要为某一特定项目选取DBMS，你需要仔细调研确定厂商。

重要的数据库系统可用于大型项目，它们提供对数千种精细功能的众多选项和控制。然而，这些选项对于要理解数据库主要概念的初学者来说是很复杂的。对于初学者，最好从简单一些的数据库系统开始学习，例如Microsoft Access。在结构上，Access与大型数据库相似：可以很容易地随时改变数据库设计，它有帮助建立表单和报表的向导，而且SQL查询遵守绝大部分SQL-92标准，相对便宜并且使用广泛。然而，Microsoft推荐说只应在相对小型的、独立的应用中使用Access。要获得更优的性能、安全性和数据保护，你总归要使用一个更强大的数据库系统。

1.7　数据库管理系统简史

开发者们很快意识到，很多商业应用需要一个用于共享数据的通用功能集合，因而，他们开始开发数据库管理系统。开发者们逐渐提炼目标并提高编程技术。很多早期的数据库方法依然有用，部分原因是因为很难抛弃目前能正常运行的应用。理解这些老的方法之间的基本区别是有意义的。下面的讨论简化了相关概念并且忽略了细节。其目的在于强调各种数据库系统之间的区别——而不是教你如何设计或使用它们。

最早的数据库管理系统基于存储数据的层次方法，它们是COBOL文件结构的扩展。为了提供灵活的访问，这些系统扩展为网状数据库。然而，E. F. Codd发明的关系数据库方法最终成为处于统治地位的数据存取方法。最近，出现了面向对象方法。从某种角度来说，它是关系模型的扩展，从其他角度来看，它又是不同的。然而，由于它刚刚出现，以此概念为基础的系统还在发展之中。

1.7.1　层次数据库

*层次数据库*方法认为业务数据一般表现出一种分层的关系。例如，在一个没有电脑的小办公室里，数据可能会存放在档案柜中。档案柜可以按顾客来组织。每一个顾客部分包含存放个人订单的文件夹，这些订单列出该顾客购买的每一件商品。要存取数据，数据库系统必须从顶端开始——在此例中是某个顾客。如图1-13所示，当数据库存储顾客数据时，它同时存储了其他相关的层次信息。

层次数据库方法是相对快速的——如果你只想从顶端开始访问数据。与数据存储相关的最严重的问题是，搜索位于层次结构底部或中部的数据项十分困难。例如，要查找订购了特定商品的所有顾客，数据库将必须检查每个顾客，每个订单以及订单上的每件商品。

1.7.2　网状数据库

*网状数据库*与物理网络（例如，局域网）无关。相反，网络模型的命名来源于数据元素之间的关系网。网络模型的主要目的在于解决层次模型中从不同角度搜索数据的问题。

图1-14说明在一个网络模型中顾客、订单和商品的数据组成。首先，应注意现在各个项是物理分离的。其次，要注意它们以箭头相连接。最后，注意入口点，它们用箭头指示。入口点是预先定义的可以被搜索的项，箭头的用途是表示一旦你进入了数据库，DBMS就能够沿着箭头进行搜索并显示匹配的数据。只要图中有一个箭头，数据库就能建立一个有效的连接。

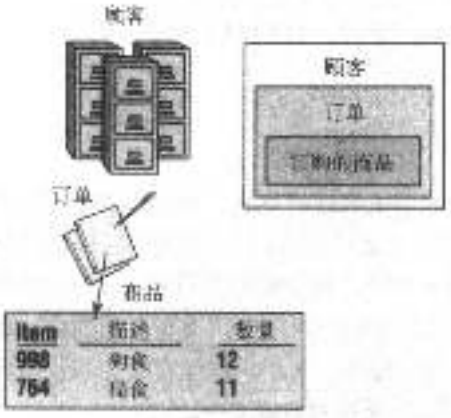


图1-13 层次数据库。DBMS从顶端（顾客）开始获取数据。获取了一个顾客后，接着获取所有嵌套的数据（订单，然后是订购的商品）

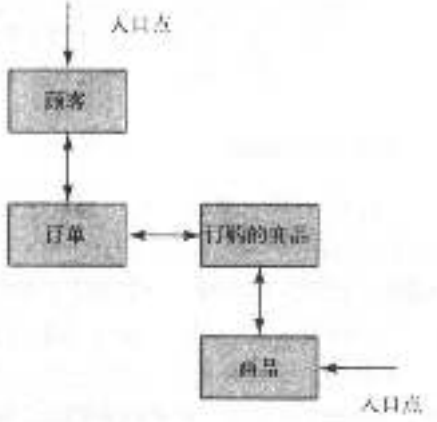


图1-14 网状数据库。所有数据集必须以索引相连接，如箭头所示。同样地，所有入口点（一个查询的起点）必须在解答问题前定义和创建

尽管这种方法似乎解决了搜索问题，但代价很高，所有的箭头必须以索引或内嵌指针的方式进行物理实现。本质上，索引复制了关联数据集合中的每个关键数据项，并将其与指向数据其余部分存储位置的指针相关联。网状数据库方法的问题在于索引必须在用户进行查询前建立。因此，开发者必须预测用户可能提出的每一个关于数据的问题。更糟的是，建立和维护索引可能需要大量的处理器时间和存储空间。

1.7.3 关系数据库

E. F. Codd于20世纪70年代创建了关系数据库方法。几年之内，在三个因素的共同作用下，关系数据库成为数据存储的主要方法。首先，理论家定义了基本概念并阐明了优点。第二，建立数据库管理系统软件的程序员创建了有效的组件。第三，硬件性能的提高满足了系统不断增长的要求。

图1-15说明例子中的四个基本表在关系数据库中是如何表示的。关键在于表（Codd称其为"关系"）是数据的集合。每张表将属性存储在列中，列描述了特定的实体。这些数据表不在物理上相互连接。连接通过每张表中存储的相匹配的数据来表示。例如，Order表包含CustomerID列。如果你找到一笔CustomerID为15的订单，数据库就自动找到匹配的CustomerID并获取相关的顾客数据。

关系数据库方法的作用在于设计者不必了解用户会对数据提出哪些问题。如果数据是精心定义的（参见第2章和第3章），数据库能有效地回答几乎所有的问题（参见第4章和第5章）。这种灵活性和有效性是关系模型处于统治地位的主要原因。本书的绝大部分集中使用关系数据库建立应用。

```
Customer(CustomerID, Name, . . .
Order(OrderID, CustomerID, OrderDate, . . .
ItemsOrdered(OrderID, ItemID, Quantity, . . .
Items(ItemID, Description, Price, . . .
```

图1-15 关系数据库。数据存储在分类的数据集合中。所有表在物理上并未相连，相反，数据在列之间连接。例如，当获取一个订单时，数据库可以依据CustomerID匹配并获取对应的顾客数据

1.7.4 面向对象数据库

面向对象（OO）数据库是一种新型的、发展中的组织数据的方法。面向对象方法起源于一种创建程序的新方法。其目的在于定义能在很多程序中重用的对象——这样可以节约时间并减少错误。如图1-16所示，一个对象主要有三个部分：名称、特性或属性的集合和方法或函数的集合。属性用来描述对象，就像关系数据库中用属性描述实体一样。“方法”是面向对象方法的真正创新。方法是定义每个对象可执行操作的简短程序。例如，添加新顾客的代码可以存储在Customer对象中。这里的创新在于这些方法可以存储在对象定义中。

图1-16同样暗示了面向对象方法的能力，注意，基本对象（Order、Customer、OrderItem和Item）与关系数据库方法中的一致。然而，有了面向对象方法，新对象可以根据已有对象来定义。例如，公司可以创建不同类别的顾客，如商业账户和政府账户，这些新对象可以包含原始Customer对象所有的属性和方法，并且还可以添加只适用于新类型顾客的变量。

面向对象方法从根本上改变了程序员创建应用的方法，作为一个应用开发者，你将使用由面向对象方法创建的DBMS软件。这样，你就能使用已经定义好的对象、属性和方法。

处理真正的面向对象数据有两种基本方法：（1）扩展关系模型以便能够处理典型的面向对象的功能要求，或者（2）创建新型的面向对象的DBMS。现在，大部分成功的商业数据库系统都遵循第一种方法，向关系模型添加对象功能。

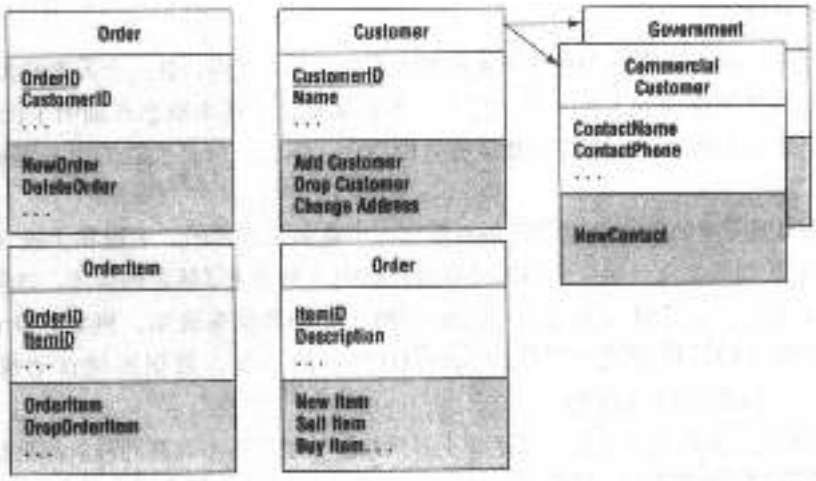


图1-16 面向对象的数据库。对象具有属性（就像关系实体具有属性一样），它拥有描述该对象的数据。对象具有方法，即该对象能够执行的功能。对象可以从其他对象继承

美国国家标准化组织（ANSI）为向关系模型中添加面向对象功能的方法提供了最好的示范。面向对象功能是SQL99版本的一个主要部分。SQL200x标准的提议中也阐明了部分面向对象问题。1997年，SQL3开发小组与对象数据库管理组织（ODMG）合并。有三个功能建议增加面向对象的能力：（1）抽象数据类型，（2）子表和（3）持久存储模块。数据库管理系统厂商已经实现了其中的部分功能，以及一些更先进的功能。

1. 对象属性

第一个问题包括定义和存储属性。特别地，面向对象程序员需要能够创造新的复合属性，这些复合属性由其他数据类型构成。SQL支持抽象数据类型，使开发者能够继承已有类型来创建新的数据类型。这种技术支持属性的继承。存储在某列的数据类型可以是几种已有抽象类型的复合。考虑图1-17，其中显示了某地理信息系统（GIS）的部分数据库。GIS定义了根据纬度、经度和海拔表示方位的一种抽象数据类型。同样地，一条线段（例如国界）就是这些方位点的集合。通过将数据存储在表中，应用可以依据用户需求搜索和获取信息。数据库还为共享和更新数据提供了方便。在GIS示例中，数据库处理选择条件（Region=Europe）。数据库还能匹配并获取储存在其他表中的人口统计学数据。这种方法的优点在于，DBMS负责处理数据的存储和获取，使开发者不必将精力集中于应用细节。

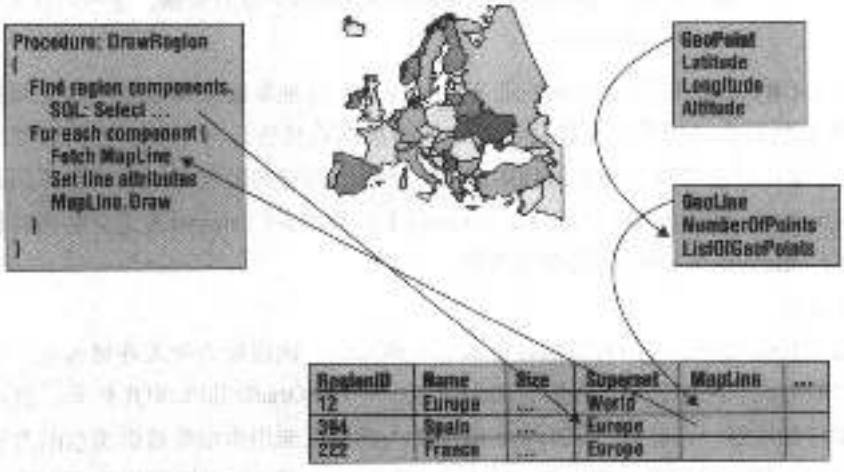


图1-17 抽象数据类型或对象。一个地理信息系统需要储存和共享复杂的数据类型。例如，区域由地理线段定义。每条线段是点的集合，点由纬度、经度和海拔定义。使用数据库能简化查找和共享数据

抽象数据类型使开发者能够创建和存储应用所需的任何数据。抽象数据类型还能为应用开发提供更强的控制能力。首先，在DBMS中存储数据能将所有开发者访问数据的方法简化、标准化。第二，数据类型内部的元素可以被封装。通过将元素定义为私有的，应用开发者（和用户）只能通过预定义的过程访问内部元素。例如，通过将元素定义为私有的，可以避免开发者直接修改任何方位的经度和纬度坐标。

SQL提供了另一种处理继承的方法，通过定义子表来实现。子表继承了基本表的所有列，提供类似于抽象数据类型的继承，但是，所有数据存储在不同的列中，这种技术与图1-16中所示方法类似，将子类存储在不同的表中，不同的是，面向对象子表不必包含主码。如图1-18所

示。继承由UNDER语句指定。从定义最高层级的表（例如Customer）开始，当你创建一个新表（例如CommercialCustomer）时，可以通过添加UNDER语句指定它是一个子表。如果你使用统一建模语言（UML）的三角形指针符号标记继承，在SQL中创建表将会很容易。如果图中有一个“指针”指向另一张表，只需定义表的属性并添加一个UNDER语句即可。

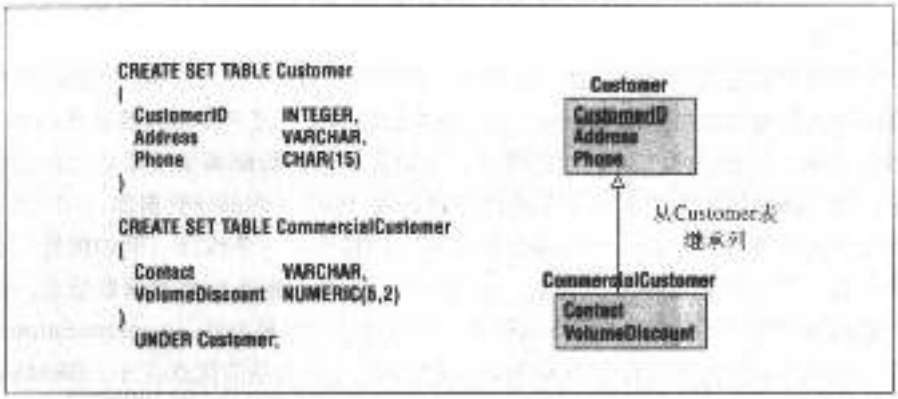


图1-18　SQL子表。子表继承所选基本表的列。对CommercialCustomer表的查询也会从CustomerID、Address和Phone列获取数据，这些列是从Customer表继承的

不必担心CREATE TABLE命令的细节。相反，明白抽象数据类型与子表之间的区别是很重要的。抽象数据类型用于设置存储于某一列的数据的类型，有了复杂数据类型，很多数据（纬度、经度等）将会存储于单独的列中。对于子表，高层的数据项仍保留在不同的列中。例如，CommercialCustomer子表可以从Customer基本表继承。Customer表定义的所有属性作为不同的列对于CommercialCustomer都是有效的。

2. 对象方法

每种抽象数据类型还可以有方法或函数。在SQL中，例程称为持久存储模块。它们可以写成SQL语句的形式。SQL语言也在扩充编程命令——很像Oracle的PL/SQL扩充。例程有多种用途。它能支持触发器，这已经在SQL中实现，持久例程也能用作抽象数据类型的方法。设计者可以定义适用于特定数据类型的函数。例如，GIS方位数据类型可以使用减运算符，表示计算两点之间的距离。

为了利用数据库的能力，每种抽象数据类型应当定义两个特殊功能：（1）检测两元素的等价性，（2）比较元素，以便排序。这些功能使DBMS能够搜索和排列数据。这些功能可能不适用于某些数据类型（例如，声音片断），但只要可能就应当定义它们。

3. 面向对象语言和共享持久对象

面向对象程序员习惯在内存中创建他们自己的对象，因此，他们需要一种方法来存储和共享这些对象，并发纯OODBMS模型正是为了解决这种问题。尽管目标看起来可能与改进的关系方法很相似，但产生的数据库系统却是独特的。

大部分面向对象开发是从编程语言发展而来的。有些语言为了利用面向对象的特性进行了专门设计，一般的例子如C++，Smalltalk和Java。这些语言中的数据变量被定义为对象。每个

类定义了属性和方法。现在，开发者使用这些语言建立应用，必须创建自己的存储机制，或者将内部数据转到关系数据库中。

复杂的对象很难在关系数据库中存储。大部分语言提供了从文件存取数据的便捷手段，但对数据库的支持很少。例如，C++库具有直接将对象串行化为磁盘文件的函数，这种手段有两个基本问题：(1) 搜索文件或从不同对象中匹配数据十分困难，(2) 开发者应负责创建所有共享、并发以及安全操作。由于数据在本质上依赖于程序并且不再独立，这种方法引起了很多问题。

本质上，面向对象程序员需要创建*持久对象*，即对象可以在任何时候保存和获取。理想地，数据库会使定义标准化，控制数据共享，并提供搜索与合并数据的例程。最根本的困难在于没有标准理论用于解释如何完成这些任务。然而，如图1-19所示，已有很多OODBMS，并且据说用户已经用这些工具创建了很多成功的应用。

GemStone Systems, Inc.
Hewlett Packard, Inc. (OpenODB)
IBEX Corporation, SA.
Illustra (Informix, Inc.)
Matisse Software, Inc.
O2 Technology, Inc.
Objectivity, Inc.
Object Design, Inc.
ONTOS, Inc.
POET Software Corporation.
UniSQL
Unisys Corporation (OSMOS)
Versant Object Technology

图1-19 OODBMS厂商及其产品。每个工具都有不同的功能和作用。可以与厂商联系以获取详细信息或在网上搜索用户评论

OODBMS的关键之处在于，对于程序员来说它看起来只是简单的存储扩展。无论对象存储在RAM还是通过DBMS共享，对象及其关联的链接处理是一样的。很显然，这些系统使面向对象程序员开发起来更方便。问题是，要使用这个系统，你得是一个面向对象程序员。换句话说，如果你一开始就关注面向对象编程，那么真正的OODBMS可能很有用。如果你最初使用传统的关系数据库，那么使用一个添加了面向对象功能的关系DBMS可能会更好。

理论上，1997年ANSI与ODMG签定的协定使SQL和OODBMS模型更接近于一个联合的标准。实际上，在商界可能需要几年并要进行相当多的实践。目前，如果你对存储和共享数据很关注，那么要依据你初始的兴趣进行选择：面向对象编程还是关系数据库。

1.8 应用开发

如果你仔细研究了图1-14、图1-15和图1-16，你会注意到他们实际上都有相同的数据集。这种相似不是偶然。应当继续学习第2章和第3章介绍的数据库设计方法，而不必关心实现数据库的方法。换句话说，任何数据库项目都开始于确认所需数据和分析数据以尽可能高效地进行存储。

建立应用的第二步是确定用户需要的表单和报表。查询是这些表单和报表的基础，所以正如第4章和第5章将要介绍的，必须创建生成表单和报表所需的任何查询或视图。然后使用报表编写器和表单生成器创建每个报表和表单，如第6章所述。下一步是将表单和报表合并到应用中，应用处理用户所需的所有操作：总的目标是创建符合用户工作要求的应用来帮助他们完成工作。

第7章讲述如何处理多用户环境下的普遍问题：保护数据完整性和支持事务。第8章介绍另一种存储数据的设计方法：数据仓库。对于大型数据库，事务处理系统使用索引和其他功能来优化存储任务。现在，管理者想获取和分析数据。数据仓库提供了特殊设计和工具以支持联机分析处理（OLAP）。

当应用设计完成和开始使用时，需要执行一些数据库管理任务。设置安全参数和数据的访问控制是一项更重要的任务。第9章讨论各种管理和安全问题。

随着企业的扩大，计算机系统和应用变得更加复杂。现代企业里的一个重要特点是用户需要通过整个企业内部的不同计算机访问和使用数据。某些场合需要扩大你的应用范围，以便更多的人从不同地点使用它。第10章讨论的分布式数据库是一种创建设有地域限制的应用的强大方法。互联网迅速成为一种强大的工具，用于建立和实现世界上任何人都能使用的数据库应用。同样的技术也可用于建立仅允许内部职员访问的应用。那些使用互联网技术但仅限内部人员访问的系统称为内联网（intranets）。

1.9 Sally的宠物商店

一位喜爱小动物的年轻女士Sally开了一家新型宠物商店。Sally希望能为宠物们找到细心照顾它们的主人。她的主要目标之一是严格监督饲养人，以确保他们细心照顾所有动物。小动物得到适当的照顾才会变成温顺的宠物。第二个目标是开拓与顾客们的长期关系。她想帮助顾客在各种情况下都能选择最优品种的动物，并且确保顾客具有照顾好动物所需的所有支持和信息。

Sally认识到要实现这两个目标需要收集和管理大量的数据。在参加了MBA课程中的信息系统课程之后，她认识到需要数据库来帮助自己收集数据和管理宠物商店的经营。

当前Sally只有一个商店，但她渴望能扩展到其他城市。她想将雇佣的员工培训为“动物的朋友”，而不是销售人员。这些员工能帮助顾客选择合适的动物，能回答有关宠物健康、营养和习性的问题。他们甚至会说服某些顾客不要购买动物。

由于员工会将他们的大部分时间花在顾客和动物上，他们需要技术来帮助他们完成任务。这个新系统必须容易使用，因为用于计算机培训的时间很少。

在与Sally进行了几次简短讨论之后，很明显她的系统需要花费时间进行建立和测试。幸运的是，Sally表示她并不需要立刻完成整个系统。她首先需要一个基本的系统处理商店的经营：销售、订购、顾客跟踪和动物的基本信息。然而，她强调这个系统必须足够灵活，能够添加新的功能和应用。

Sally的宠物商店的详细信息将会在其他章节中进行分析。目前，你最好光顾本地的宠物商店或者与朋友交流，对他们遇到的问题有一个初步的了解，并弄清楚数据库将如何帮助他们。

1.10 Rolling Thunder自行车

Rolling Thunder自行车公司定制自行车。它的数据库应用相比宠物商店应用更完整，并且它提供了例子来说明如何将数据库系统的各部分组合到一起。这个应用还包含很多详细的表单，说明如何创建用户界面。此外，大部分表单包含处理一般业务操作的代码。学习此代码可以帮助你建立自己的应用。Rolling Thunder应用具有一个综合帮助系统，描述公司和每个表单。这个数据库包含数百个顾客和自行车的实际数据。

Rolling Thunder自行车公司的一项重要任务是接收新自行车的订单。已有的一些功能帮助非专业人员挑选一辆好的自行车。制造自行车时，员工将设计记录在Assembly表单上。自行车送到后，顾客付款。顾客的付款记录在金融表单中。当自行车装好零部件后，库存数量自动递减。零部件从供应商那里订购，货到付款。

Rolling Thunder自行车公司执行的操作与其他任何业务都很相似。通过学习这个应用和技术，你将能为任何企业创建稳固的应用。

1.11 可行性研究

信息系统的想法可以有很多种来源：用户、上级管理者、信息系统分析学家、竞争者或其他行业的公司。获得最初若干人支持的想法可能会提议为新项目。如果项目规模很小很容易创建，只需要几天时间。而较大规模的项目则需要更仔细地研究。如果项目将关系到企业内部的重要部门，需要昂贵的硬件，或者需要充裕的开发时间，那么就要采用一种更为正式的可行性研究。

系统分析课程会对可行性研究进行详细介绍。然而，由于它们具有的特性，对使用数据库方法引发的典型成本和效益进行研究是有用的。

可行性研究的目标在于决定计划中的项目是否值得进行。研究基本分为两类：成本和潜在效益。如图1-20所示，成本一般划分为两类：预先或一次性成本和当项目使用时的运行成本。效益一般分为三类：降低运行成本、提高价值或超越竞争对手的战略优势。

成本	效益
预先/一次性	节约成本
软件	软件维护
硬件	减少错误
通信	减少数据维护
数据转化	减少用户培训
研究和设计培训	提高价值
运行成本	更好的数据访问
人力	更优的决策
软件升级	更好的通信
经费	更及时的报表
支持	对变化更快的反应
软件和硬件维护	新产品和服务
	战略上的优势
	胜过竞争对手

图1-20 引入数据库管理系统的一般成本和效益。注意，效益可能很难测量，尤其对于战术和战略的决策。但列出潜在的效益仍然是重要的。即使不能准确赋值，管理者仍需要查看完整的列表

1.11.1 成本

几乎所有项目都会承担相似的预先成本。企业通常必须购买额外的硬件、软件以及通信设备（例如，组建局域网）。开发系统的成本在图中列出，包括所有附加研究的成本。其他一次性成本包括将数据转化至新系统和初期对用户进行的培训的费用。数据库管理系统是昂贵的软件。例如，对于大型项目，诸如Oracle这样的软件成本很容易达到几百万美元。每年还必须

购买软件的维护和升级服务。

厂商可以帮助你估算硬件和软件的成本。只要你知道最终系统的大概规模（例如，用户数），厂商可以提供比较精确的成本估算。数据转换的成本可由数据量大小来估算。最大的难题通常在于估算开发新系统的成本。如果企业具有类似项目的经验，那么依据项目规模使用历史数据可以估算时间和成本。否则，可以通过要投入的人员和时间来估算成本。

一旦项目完工并且系统安装完毕，还有一些地方将会产生成本。例如，新系统可能需要额外的人力和经费。软件和硬件必须修改和替换——承担维护成本。处理员工周转和系统修改可能需要额外的训练和支持。同样，只要你了解项目的规模，这部分成本很容易估计。

不幸的是，信息系统（IS）设计者对于成本的估算并不很成功。例如，1995年1月《PC Week》报导说31%的新信息系统项目在完成前就取消了。此外，53%的结题项目超过了预算的189%。最大的困难在于估算设计和开发新软件所需的时间。每个开发者在软件开发能力方面都是不同的。在大型项目中，人员始终在变，因此，精确预测设计和开发新系统所需的时间一般是不可能的。尽管这样，管理者仍然需要对成本进行估算。

1.11.2 效益

很多情况下效益更难估算。有些效益是有形的，可以用精确的百分比度量。例如，事务处理系统比决策支持系统（DSS）容易评估，因为前者的效益一般来自它们降低运行成本的能力。一个系统可能会提高员工处理产品的数量，这样便允许公司进行扩展而不增加人力成本。数据库方法可以使员工更容易创建和修改报表，从而减少信息系统的人力成本。最后，新的信息系统会减少数据中的错误，获得更优的决策。

很多效益是无形的，无法赋予确定的金钱价值。例如，管理者能更好地访问数据可以带来效益。改进通信，做出更优的决策，并且管理者在变化的环境中能做出更快的反应。类似地，新系统可能会使公司生产出新产品和提供新服务，或向已有客户销售附加产品。类似地，公司实现的系统可能会带来竞争优势。例如，自动订货系统通常会促使顾客进行更多的订购。这样公司对于其竞争者而言就具有了优势。

当建立信息系统用于使操作级任务自动化，并获得切实的效益时，评估系统的经济效益是相对简单的。改进数据访问的影响在降低成本和增长收入方面是容易观察和估量的。然而，当建立信息系统改进战术和战略决策时，确定和估算效益就十分困难。例如，对于市场经理来说，在周一而不是周三就能获得上周销售数据的价值是多少呢？

在数据库项目中，效益可以来自改进操作——这将节省成本。还有其他额外的效益，因为现在为用户创建报表更加便捷，修改系统所需的时间也更少。用户还可以获得更好的数据访问，因为这可以通过用户创建自己的查询来完成——不必等待程序员编写新程序。

数据库项目可以带来很多效益，但企业获得这些效益的前提是项目正确、按时完成，并且不超出预算。要完成这个任务，必须仔细设计系统。此外，你的团队必须与用户沟通，合理分配工作，跟踪开发进度。需要遵守设计方法学。

小结

商业应用最重要的特色之一是允许多用户同时共享数据。没有DBMS，共享数据会引发很

多问题。例如，如果数据定义分别存储在每个程序中，改变数据文件会变得非常困难。一个程序及其数据文件的改变会造成其他程序崩溃。每个应用将需要特别的代码来提供数据安全性、并发性和完整性。通过将精力集中于数据，数据库方法将数据与程序分离。这种独立性使得扩展数据库而不使程序崩溃成为可能。

DBMS拥有很多组件，DBMS必备的功能包括通过数据库引擎存取数据，通过数据字典帮助DBMS及其用户定位数据。其他一般功能包括查询语言，用于从DBMS获取数据来回答业务问题，应用开发工具包括报表编写器、表单生成器和用于创建菜单及帮助文件的应用生成器。高级数据库系统提供用于控制数据访问安全性，与其他软件包协作及与其他数据库系统通信的工具。

数据库系统的发展经历了几个阶段。早期的层次数据库适用于特殊目的，但能提供有限制的数据访问。网状数据库允许用户建立复杂查询，但必须预先为链接建立索引。关系数据库是目前建立商业应用的主导方法。数据一旦精心定义，就能高效存取来回答业务问题。面向对象方法是创建软件的新方法。面向对象系统允许你创建你自己的新抽象数据类型。它们还支持子表，很容易扩充一个对象的类而不必从零开始重新定义。

无论使用哪种类型的数据库，应用开发遵循相似的步骤。第一，确认用户需求，根据需要收集数据，定义数据库的结构。然后，开发将要使用的表单和报表，建立所需要的查询。最后，将不同元素合并到一起，组成符合用户需求的完善应用。如果需要，将数据库在企业内部或通过互联网、内联网分散部署。将数据库与强大的分析和表现工具相结合，可以提供额外的功能。这些分析和表现工具诸如电子表格、统计包和文字处理器等。

开发漫谈

Miranda要开始建的数据库项目，首先必须了解要用到的工具的功能。在一个数据库项目的起始点，应该收集那些要用到的工具的信息，获取最新的参考手册，安装最新的软件补丁，创建工作目录和项目空间，对于一类项目，应当登录，访问DBMS，确保你能够创建表，学习帮助系统里讲述的基本内容。

关键词

抽象数据类型
数据独立性
数据库管理系统（DBMS）
层次数据库
面向对象（OO）的数据库
持久存储模块
子表
应用生成器
数据库
可行性研究
内联网
联机分析处理（OLAP）
关系数据库
数据字典
数据库引擎
表单生成器
网状数据库
持久对象
报表编写器

复习题

1. DBMS方法对于应用开发有什么好处？

2. DBMS的基本组成有哪些部分?
3. 为什么关系数据库方法优于早先的方法?
4. 面向对象方法与关系方法有什么不同?
5. 为什么说抽象数据类型和子表是关系方法和对象方法的混合物?
6. 使用数据库系统进行应用开发的主要步骤是什么?
7. 可行性研究的目的是什么?

练习

Employee(EmployeeID, LastName, FirstName, Address, DateHired)

332	Ant	Adam	354 Elm	5/5/1964
442	Bono	Sonny	765 Pine	8/8/1972
553	Cass	Mama	886 Oak	2/2/1985
673	Donovan	Michael	421 Willow	3/3/1971
773	Moon	Keith	554 Cherry	4/4/1972
847	Morrison	Jim	676 Sandalwood	5/5/1968

Client(ClientID, LastName, FirstName, Balance, EmployeeID)

1101	Jones	Joe	113.42	442
2203	Smith	Mary	993.55	673
2256	Brown	Laura	225.44	332
4456	Dieter	Jackie	664.90	442
5543	Wodkoski	John	984.00	847
6673	Sanchez	Paula	194.87	773
7353	Chen	Charles	487.34	332
7775	Hagen	Fritz	595.55	673
8890	Hauer	Marianne	627.39	773
9662	Nguyen	Suzie	433.88	553
9983	Martin	Mark	983.31	847

报表

Ant, Adam	5/5/1964	
	Brown, Laura	225.24
	Chen, Charles	487.34
		712.58
Bono, Sonny	8/8/1972	
	Dieter, Jackie	664.90
	Jones, Joe	114.32
		779.22

1. 使用图中的两个表创建一个新的数据库。随意添加更多数据。确保将具有下划线的列设为主码。接着，创建包含两个表中每一列的查询。依据此查询创建一个与图中相似的报表。提示：使用向导创建报表。
2. 阅读你使用的DBMS的文档，写一个简要的提纲解释如何：
   a. 创建一个表。
   b. 创建一个简单查询。
   c. 创建一个报表。
3. 采访一个朋友或亲戚，询问他或她的工作，拟定两张用于其工作的数据库应用的表单。

4. 找最近的一篇比较两个以上DBMS软件包的参考文献。列出各自的主要特长。描述各自的基本功能（查询、报表编写器等）。
5. 描述大学俱乐部或学生组织如何使用数据库改进其服务水平。
6. 利用互联网资源，选择一个DBMS厂商，确定所需组件，估算一个中等规模项目购买完整系统所需的花费。该数据库将包含至少100个主要表和大约800兆字节的数据存储。系统将至少被20个用户同时使用——他们大部分使用与中心服务器相连的个人计算机。
7. 一个公司正在考虑建立一个新系统用于跟踪评价雇员。在与用户面谈后，你了解到系统潜在的效益包括减少确定升迁所需的时间；减少一个全职岗位；避免每年四次且每次打印3 000页报表；更明智的决策，这将减少每年250 000美元的EEO诉讼。系统成本包括最初的开发成本（35 000美元）；新硬件（12 000美元）；新软件（10 000美元）；和每年的维护费用大约4 000美元。还会有大约每年6 000美元的培训成本。为这个项目准备一个可行性研究。在不同利率（2%，5%，8%和10%）和不同的项目期望寿命（1年，3年，5年和10年）的前提下进行分析。还有没有其他应当考虑的效益和成本？
8. 一个公司要你创建一个小型应用，用来帮助管理跟踪销售订单。订单来自电话或网站上的电子邮件，秘书将把数据输入到你的应用中。现在，公司通过电话接收很多订单，这需要两个秘书同时上班以避免电话占线。如果50%的订单可以转移到联机系统（已经开发完成）上去，那么公司可以减少一个话务秘书（最低工资标准）。该公司已经拥有一台计算机，但是需要购买DBMS。公司愿意为这个应用支付多少费用？假设公司的产品很简单并且数据转移很方便，你开发这个系统要花费多长时间？

Sally的宠物商店

9. 安装宠物商店数据库，如果已经安装，则在局域网内找到它。打印（或书写）数据库中所使用表的列表。使用帮助命令找出你使用的Microsoft Access的版本号。
10. 光顾附近的一个宠物商店，做一个列表包含待售的10种商品和5种动物。将这些数据输入到宠物商店数据库的适当表中。
11. 创建一个邮寄地址报表，列出所有居住在加利福尼亚（CA）的顾客。首先建立一个包含Customer表和City表的查询。然后在查询的基础上使用报表向导创建一个报表。
12. 概述宠物商店运营的基本任务。确认将会用到的一些基本数据项。

Rolling Thunder自行车

13. 安装Rolling Thunder自行车数据库，如果已经安装，则在局域网内找到它。使用BicycleOrder，创建用于登录新自行车的表单。
14. 利用Rolling Thunder帮助系统简要描述公司及其主要流程。确认公司内的主要业务实体。
15. 当自行车制造完成后，如何将数据输入到数据库？在此阶段你预计会产生什么问题？
16. 查阅关系/类图，解释自行车部件是什么，以及它是怎么连接到自行车上的。从数据中举例说明。

参考网站

网站	描述
http://www.microsoft.com/office/access	Microsoft Access

http:// www.microsoft.com/sql	Microsoft SQL Server
http://www.oracle.com	Oracle
http://otn.oracle.com	Oracle技术网站，包括软件下载
http://www.cai.com/products/ingres.htm	Ingres
http://www.sybase.com	Sybase
http://www,software.ibm.com/data/db2	IBM DB2
http://www.mysql.com	免费但有限制的DBMS
http://www.postgresql.org	一个更好的免费DBMS
http://www.acm.org	计算机协会
http://groups.google.com/groups?group=comp.databases	数据库问题的新闻组
http://dbforums.com	
http://dbaclick.com	
http://www.devx.com/dbzone/Door/7022	

补充读物

Anderson, V. *Access 2002: The Complete Reference.* Berkeley: Osborne McGraw-Hill, 2002. [One of many reference books on Access. Spend some time at your local (or electronic) bookstore and choose your favorite.]

Loney, K., and G. Koch. *Oracle 9i: The Complete Reference.* Berkeley: Osborne McGraw-Hill, 2002. [One in series of reference books on Oracle.]

Shapiro, J. *SQL Server 2000: The Complete Reference.* New York: McGraw-Hill, 2001. [One of many reference books for SQL Server.]

Willis, T. *Beginning SQL Server 2000 for Visual Basic Developers.* Indianapolis: Wrox Press, 2000. [One of many introductions to SQL Server 2000.]

Zikopoulos, P. C., and R. B. Melnyk. *DB2: The Complete Reference,* New York: McGraw-Hill, 2001. [One of few reference books on DB2 and written by IBM employees.]

第一部分

系统设计

要创建有效的应用，必须首先了解业务并决定如何帮助用户。在数据库的背景下，最重要的问题是确定必须要存储的数据。这个过程需要两个基本步骤。在第2章你要设计一个用于分析业务实体及其关系的逻辑（或概念）数据模型。这个逻辑数据模型显示在类图里，并且指明公司的各种业务规则。

设计的第二步是创建一个执行模型，表示数据如何在数据库管理系统中存储。这一步一般包括创建一系列设计合理的表，这些表组成关系数据库。第3章描述如何创建这些表，并解释为什么精心定义它很重要。

第 2 章

数据库设计

本章学习内容

- 在设计系统时，为什么模型是重要的？
- 什么是对象？
- 什么是类图（或实体关系图）？
- 什么是不同的数据类型？
- 什么是事件？在数据库设计中如何描述？

2.1 开发漫谈

Miranda：唔，Ariel，你说得对。数据库似乎是完成这份工作的正确手段。

Ariel：那你决定接手你叔叔公司的这份工作啦？

Miranda：是的，这份工作报酬不错，而且公司似乎愿意让我边学边干。但是，在我完成这个项目前公司只能付给我一小笔钱。

Ariel：不错。那你什么时候开始呢？

Miranda：这正是另一个问题。我不是很确定应该从哪里开始。

Ariel：这可是个问题。你知道这个应用程序要做些什么吗？

Miranda：唔，我与经理和一些员工交流过，但是还有很多地方我不清楚。这个项目比我想象得要大。我感觉要了解所有的细节很困难。我不知道的报表和术语太多了。而且其中一个销售人员开始谈论起数据的所有规则——例如对于团体客户来说，顾客编号是五位数字，而对于政府账户来说是四位数字加两个字母。

Ariel：或许你需要一个系统来记录他们告诉你的所有东西。

2.2 简介

任何信息系统的目标都是为用户增加价值。要达到这个目标需要回答两个重要的问题：用户是谁？信息系统怎样才能帮助他们？这两个问题回答起来可能都十分困难，需要不断调研。

在对一个项目花费大量金钱之前，大多数企业会进行可行性研究，为这两个问题提供最初的解答。企业对以下三个关键领域的利益评估特别感兴趣：（1）降低成本，（2）提高销售额或收入，（3）竞争优势或长期利益。

按时并在预算之内完成项目是一个难题。只有几个用户和一两个开发者的小型项目一般是

很简单的。然而，你仍然必须精心设计数据库，使它们足够灵活以便处理将来的需求。同样，你需要做记录，以便将来的开发者能够很容易地理解这个系统，它的目标和你所做的决定。大型项目带来额外的复杂性。由于有众多用户和若干开发者，你需要将项目划分为较小的问题，在用户和设计者之间交流意见，并且跟踪团队的进度。

正如系统分析和设计课程里所述，为设计系统和管理项目创造了很多正规的方法。详细信息可以查阅任何一本有关系统开发的教材。最近，为加快开发进度进行了很多尝试，著名的有快速应用开发（RAD）方法。Steve McConnell的著作《Rapid Development：Taming Wild Software Schedules》对设计的重要性、如何减少开发时间以及何时不能减少开发时间的精彩分析。这些方法皆可用于数据库应用的开发上。

所有这些方法中重要的一步是构建系统模型。模型是现实世界中系统的简单抽象。在很多情况下，模型由一张提供系统直观图形的图片组成。就像建筑工人需要蓝图建造建筑物一样，信息系统开发者需要设计来帮助他们创建有用的系统。如图2-1所示，概念模型建立在系统的用户视图基础上。执行模型建立在概念模型基础上，描述数据如何存储。DBMS使用执行模型来存储数据。

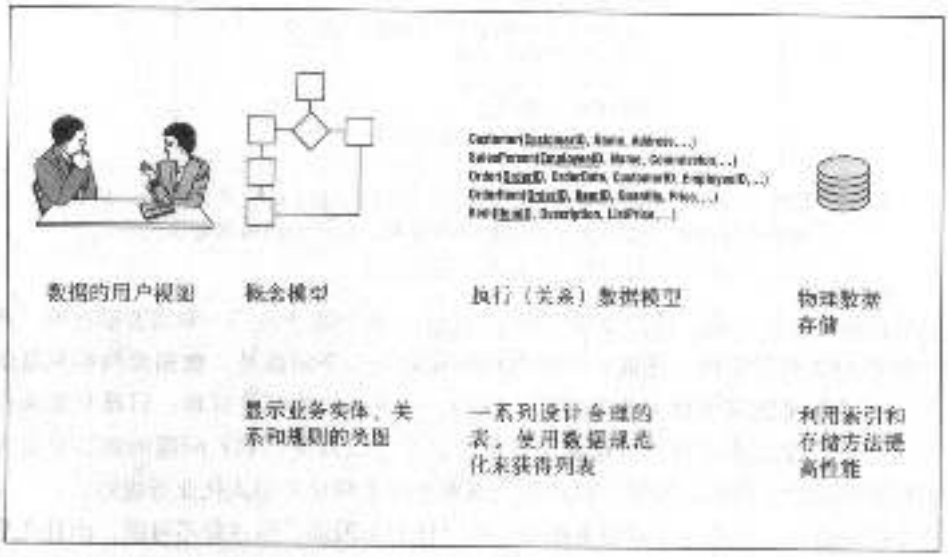


图2-1 设计模型，概念模型记录和描述系统的用户视图。执行模型描述数据存储的方法。最终的物理数据库可能使用像索引这样的存储技术来提高性能

设计系统一般使用三种通用模型：过程模型、类或对象模型和事件模型。过程模型由协作图或数据流图（DFD）表示。它们一般在系统分析课程中详细介绍，用于重新设计企业内的信息流。类图或以前的实体关系图用于显示系统中的主要实体或对象。事件模型如序列图或状态图是比较新颖的，用于描述各种事件的时间选择和消息在各个对象间如何传递。每种模型都用于从不同角度显示当前设计的系统。优秀的设计者应当能够创建并使用所有三种模型。不过，类图是设计和建立数据库应用所使用的最重要的工具。

数据库应用可以是较大型项目的一部分，这种项目需要正规的项目管理技术来控制成本和监视进度。数据库项目也可以是较小型的、独立的项目，由一个接近用户的小团队开发，快

速建立新系统。在这两种情况下，项目管理控制和系统设计方法将会不同。然而，某些基本的数据库设计技术是一样的。本书集中于数据库设计，而把系统和项目管理问题留给系统开发课程。

2.3 开始设计之前

如今的DBMS工具是华丽和吸引人的。在使用它建立用户希望看到的表单和报表时，通常是很诱人的。然而，在你能建立表单和报表之前，必须正确设计数据库。如果在数据库设计中出错，那么创建表单和报表将会很困难，而且，以后进行修改将会花费相当多的时间。

在试图建立任何东西之前，要通过与用户交谈来准确决定需要什么数据。有时，用户明确知道他们想要什么。但是大部分情况下，用户对于他们想要什么只有粗略的想法，对于计算机能做什么只有模糊的认识。在项目开发中，与用户交流是关键的一步。最重要的方面在于确认（1）要收集的确切数据，（2）各种数据如何关联和，（3）数据需要在数据库中存储多长时间。图2-2描述了设计流程的初始步骤。

1. 明确系统的目标
2. 与用户交谈来确定基本的表单和报表
3. 确认要存储的数据项
4. 设计类（表）和关系
5. 确认所有业务约束
6. 检验设计与业务规则是否相符

图2-2　数据库设计中的初始步骤。数据库设计代表了企业的业务规则。你必须与用户仔细面谈来确保正确理解业务规则。这个流程通常是迭代的，当你设计类时你需要向用户获取更详细的信息

一旦你确定了数据元素，就需要恰当地组织它们。正如本章及下一章将要描述的，将数据放入表中的规则是很简单的。然而，开发者经常面对的一个问题是，数据库结构总是依赖于业务规则。真正的难题在于确定业务规则。例如，考虑客户的标准订单：订单只能来自一位客户，还是允许两个或多位客户一起提交订单？这个（以及类似的）问题的解答在很大程度上影响数据库的设计。因此，数据库设计的全部要点在于确认并形式化业务规则。

要建立商业应用，就必须了解业务细节。这个任务很困难，但并非不可能，而且几乎总是很有趣。尽管每种业务不同，但在商业界有很多普遍问题。这些问题中的一部分将贯穿本书。在本书中使用的开发方法可以应用并扩展到很多普遍的业务问题上。

2.4 设计数据库

信息系统是复杂的、不断变化的，并且创建和维护都很昂贵。但是设计合理的系统能给企业带来巨大的利益。建立有用的系统需要理解用户并与之交流。这需要组织和控制一个开发团队。系统设计是用于促进这种交流和团队工作的模型。设计是基本业务操作的简化或描绘。设计模型还记录业务中的所有基本功能、假定和约束。

2.4.1 确定用户需求

设计系统的挑战之一在于确定需求。在创建一个实用系统前，必须彻底了解业务需要。一

个关键步骤是访问用户和观察公司的运转。尽管这一步听起来容易，但它可能是十分困难的——尤其当各个用户的意见不一致时。甚至在最好的情况下，交流也会十分困难。良好的交流技巧和经验对成为一个好的设计者是非常重要的。

设计数据库应用最重要的任务之一就是要确定存储什么数据。只要你仔细收集并组织数据，DBMS会使创建和修改报表变得很容易。当与用户交谈时，你会收集用户文档，例如报表和表单。这些文档提供了有关公司的基本数据和操作信息。在初始设计阶段，你需要获取三条基本信息：（1）需要收集的数据，（2）数据类型（域），（3）数据的数量。

2.4.2 业务对象

数据库设计关注需要存储的数据。随后，可以根据用户需要创建查询搜索数据，创建输入表单添加新数据，创建报表获取和显示数据。目前，最重要的步骤在于正确组织数据，以便数据库系统能够有效处理它。

所有的业务都涉及实体或对象，例如客户、产品、雇员和销售。从系统的角度来看，实体是现实世界中你希望跟踪的信息。实体由其*属性*或特性描述。例如，客户实体具有姓名、地址和电话号码。在建模术语中，列出其属性的实体称为*类*。在编程环境中，类还可以包含它能执行的*方法*或函数，它们可以与类列在一起。例如，客户类可能会有一个方法用于添加一个新客户。数据库设计很少需要描述方法，所以它们一般不被列出。

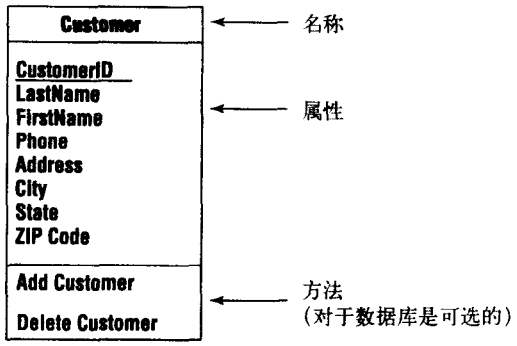


图2-3 类。类具有名称、属性和方法。属性用于描述类和要收集的数据。方法是类能执行的操作，它们在数据库设计中很少使用

数据库设计者需要某种方法记录和向用户和其他设计者显示类的列表。已经开发出了许多图形技术，但较新颖的手段是使用类图。*类图*将每个类显示为一个框，其中包含该类的属性。通过用线段连接类与类，类图还能显示这些类是如何相互连接的。图2-3说明了如何描述一个单独的类。

2.4.3 表和关系

数据库设计的主要目的是确定用户需要什么样的数据。这些数据需要谨慎存储，以便数据库能高效存取数据。关系数据库通过在表中存储数据来达到这种功效。这些表代表业务类，类中的属性成为表中的列，而每一行代表其中一个对象的数据。因此，当你与用户交谈并开始确定要收集的数据时，需要将它们组织成类。尽管这一过程并不困难，但仍需仔细对待。数据表需要满足某些需求。

类是相互关联的，它最后变成表。例如，销售类包含一个属性用于确定参与某次消费的客户，因此客户类是与销售类相连的。在类图中，这种关系表示为连接这两个类的一条线段。这些关系具有一个数值包含附加的业务信息。在客户/销售例子中，大部分公司有一条规则，那就是一次销售活动只涉及一位客户，而一位客户可以参与多个销售活动。因此，客户表和

销售表间具有一对多的关系。这些关系对设计数据库非常关键。

2.4.4 定义

要学习如何创建有用和高效的数据库，需要理解一些基本定义。主要的定义如图2-4所示。E. F. Codd在定义关系数据库时给出了这些术语的形式数学定义，这些形式定义在第3章的附录中描述。然而，对于设计和建立商业应用来说，这里提出的定义更容易理解。

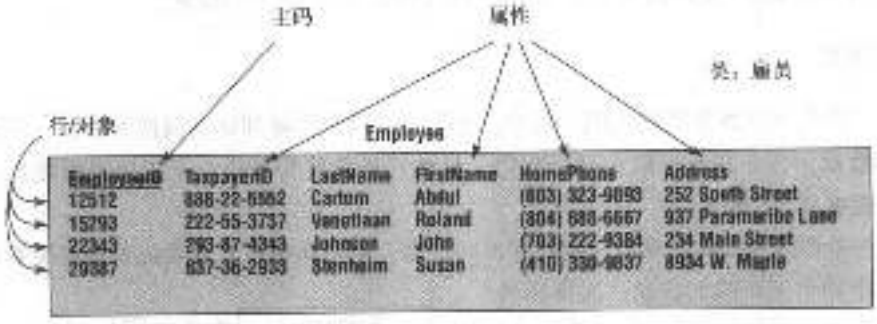


图2-4 基本数据库定义。Codd提出了很多形式术语和数学定义，但这些更容易理解。数据表中的一行代表一个单独的对象，在当前情况下是某个特定雇员。每列（属性）包含关于该雇员的一条数据

关系数据库是谨慎定义的表的集合。表是用于描述实体的列（属性或特性）的集合。单独的对象作为行（元组）存储在表中。例如，EmployeeID 12512代表一个雇员的实例，作为雇员表中的一行来存储。属性（特性）是实体的特征或描述符。关系数据库的两个重要方面是（1）所有数据必须存储在表中，（2）所有表必须谨慎定义以提供灵活性并将问题减到最少。数据规范化是适当定义表的过程，利用它来提供灵活性，最小化冗余，并保证数据完整性。数据库设计和数据规范化的目标是创建一系列适当的表。每个表描述企业内部一种单独的对象类型。

2.4.5 主码

每个表必须有一个主码。主码是确定某一特定行的列或列的集合。例如，在客户表中，可能会使用客户姓名来查找某一条目。但那列并不是很好的主码。如果有八位客户都叫John Smith怎么办？大部分情况下，会创建单独的主码以保证它们是惟一的。例如，通常创建一位客户标识符来确保正确区分所有客户。主码和数据的其余部分之间是一对一的关系。换句话说，每一个主码代表的条目指向确定的某个客户行。为了强调主码，组成主码的列的名称会加下划线。

在某些情况下主码的组成会有几种选择。在客户例子中，你可以选择姓名或电话号码或创建一个惟一的CustomerID。你的选择应使组成惟一标识符所需列的集合最小。

某些美国企业可能会对使用个人社会保障号码（SSN）作为主码感兴趣。即使你需要收集SSN，也最好使用一个单独的数字作为主码。一个原因是主码必须永远惟一，而使用SSN你会冒一定风险，因为某人可能会提供一份伪造的文档。另外，主码在数据库中的很多地方使用

和显示。如果你使用SSN，太多的雇员将会访问客户的私有信息。由于SSN用于很多金融的、政府的和健康记录，你应当通过限制雇员访问这些号码来保护客户的隐私。

主码最重要的问题是它只能指向数据库中的一行或一个对象。例如，假设你在为人力资源管理部门建立数据库。经理告诉你公司用姓名确定雇员。你问是否有两个雇员具有相同的姓名，经理查看雇员列表并告诉你已有的30名雇员里没有重名的。经理还提出如果你将雇员姓名中间的首字母加入进去，确认雇员将永远不会存在问题。这样看来，姓名似乎可用作主码。但请等一下！实际上你需要询问将来主码的值可能是多少。如果你使用雇员姓名作为主码建立数据库，你实际上明确假设雇员从不会重名。毫无疑问，这种假设在将来会引起问题。

本质上，类图和数据表代表相同的概念。本章更详细地讨论类图，而第3章会详细介绍将其转化为数据表。目前，我们关注业务类和它们的关系，这些关系由业务规则定义。

2.5 类图

DBMS方法关注数据。在很多企业中数据保持相对稳定。例如，公司现在收集的有关客户的基本数据与它们20年前或30年前收集的相同。例如姓名、地址和电话号码这样的基本信息总是会需要的。尽管现在你可能会选择收集额外的数据（例如，手机号码和互联网地址），但你仍然要使用相同的基本数据。而另一方面，公司接受和处理销售订单的方式随时在改变，因此，表单和报表需要不断修改。数据库方法通过关注正确定义的数据来对付这种不同。这样，DBMS便能很容易改变报表和表单。任何设计的第一步都是确定你希望观察和跟踪的事物或实体。

2.5.1 类和实体

在阐述模型之前最好先定义一些术语。基本的定义在图2-5中给出。注意，这些定义是非正式的。每个条目根据Codd的关系模型都有一个更加形式化的定义，在统一建模语言（UML）中还有更准确的语义学的定义。然而，不需要这些数学基础知识也能开发数据库。

术 语	定 义	宠物商店例子
实体	现实世界中你希望描述或跟踪的事物	Customer, Merchandise, Sales
类	具有属性（特性）和行为（方法）的实体描述	Customer, Merchandise, Sale
对象	具有特定数据的类的对象	Joe Jones, Premium Cat Food, Sale #32
属性	类或实体的特征或描述符	LastName, Description, SaleDate
方法	类执行的函数	AddCustomer, UpdateInventory, ComputeTotal
关联	两个或多个类之间的关系	每次销售活动只涉及一位客户

图2-5 基本定义。这些术语描述了创建类图所需的主要概念。第一步是确定业务实体及其属性。在数据库背景下，方法不如属性重要，但你应当确定重要的函数或计算方法

要设计数据库，必须理解类、属性和关联之间的区别。你的解决方案取决于业务如何对应实体以及需要收集什么数据。例如，考虑一个雇员。很明显雇员是一个单独的实体，因为你总要记录雇员的详细信息（雇佣日期、姓名、地址等）。但雇员的配偶呢？是将其作为雇员实体的一个属性，还是一个单独的实体？如果企业只关心配偶的姓名，那么可以将其作为雇员

实体的一个属性储存。而相反，如果企业希望记录关于配偶的额外信息（例如，生日和职业），那么创建一个单独的拥有自身属性的配偶实体更好一些。你设计数据库的第一步应当是确定实体和定义它们的属性，第二步是指出这些实体之间的关系。

2.5.2　关联和关系

设计数据库的重要步骤是确定实体间的关联或关系。这些关系的详细信息代表了业务规则。正确确定这些业务关系非常关键。实体通常与其他实体相关。类似地，一个实体中的属性可以与其他属性相关。关联或关系代表了业务规则。例如，很明显，一位客户可以提交多个订单。但相反方向的关系就不是很明显。在某一订单中可以包含多少位客户？很多业务会回答说每个订单只能来自一位客户。而另一方面，某些企业可能允许一个订单包含多位客户。这就是所谓的多对多的关系。

关联可以命名；UML指所谓的关联角色。二元关联的每一端都可以标记。通常使用箭头来指明应当如何理解标记，图2-6说明如何指明一位客户可以提交多个订单。

统一建模语言使用数字和星号来指明关联的重复度。如图2-6所示，星号（*）代表多个。因此，每个供应商可以接受多个订单。一些较老的实体关系设计方法使用多个箭头或用字母M和N来表示关系中的"多个"一方。

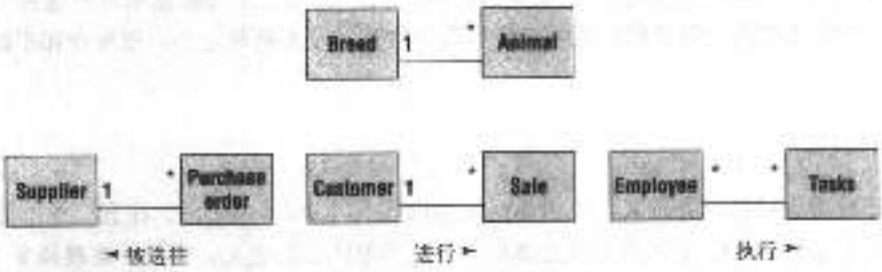


图2-6　关联。实体间有三类关系（一对一，一对多和多对多）。它们可以用多种方式描述，但它们代表业务或企业规则。在模糊的定义中几乎任何关系都可以划分为多对多的关系，应当避免这种定义，它们使数据库设计更复杂

正确确定关系对于设计数据库应用是十分重要的。记住，关系由业务规则决定，因此与用户交谈，仔细确定这些关系是很重要的。当你从用户那里收集表单和报表时，仔细检查它们并确定最初的实体——例如客户、订单和产品。然后与用户交谈，确定业务如何处理这些实体之间的关系。

要确保了解建立两个类之间的关系或关联的重要性。例如，客户/销售关联的业务规则表示：只有当某位客户的数据已经在客户表中存在时，该客户才能加入到销售活动中。

2.5.3　类图细节

类图是企业中的类和关联的视觉模型。这些类图有很多选项，但基本功能必须包含框中的类名（实体）和连接它们的线段（关系）。一般地，你希望包含关于类和关联的信息越多越好。例如，你最终会希望在框里面包含类的属性。

关联也有一些选项。数据库设计最重要的问题之一是关系的重复度，包括两个方面：(1)

可以关联的对象的最大数目，(2) 如果有，必须包含的对象的最小数目，如图2-6所示。重复度由最小值、省略号（...）和最大值表示。星号（*）代表未知的数量，在图2-6的例子中，只有一位客户（1...1）能涉及零到多个（0...*）销售活动中。

很多时候，关系需要相关联的两个实体都存在。例如，如果你的销售表单中列出了一位客户，但在文件中不存在该客户的数据时将会怎样？订单和客户实体之间有一个参照关系。业务规则要求客户数据必须已经存在，该客户才能购物。这个关系可以通过指定关系的最小值（如果是可选的则为0，如果是必须的则为1）来表示。



图2-7 类图或实体关系图。每位客户可以提交零个或多个订单。每次销售活动必须仅仅来自一位客户。零（0）代表可选项，因此，一位客户可以未提交任何订单

一定要在两个方向上理解关系。例如，在图2-7中，客户/销售关联的第二个部分声明，一位客户可以提交零个或多个销售订单。换句话说，客户可以不提交订单。

往下看类图，注意销售和产品之间的多对多关系（在两个类的右边都有*），一个销售活动必须包含至少一件产品（空订单在业务中无效），但公司也可能有某产品还未卖出去过一件。

1. 关联细节：多重关联

在数据库设计中，类之间的多对多关系会引发问题。它们在初始类图中是可接受的，如图2-8所示，但它们最终必须被划分成一对多的关系。这个过程及其原因在第3章详细介绍。

如图2-8所示，在关联的情形下，实体并非总是明显的。考虑一个基本的制造业环境，其中雇员将组件装配成最终产品。乍一看，图中有三个实体：雇员、组件和产品。这个设计指明数据库应当明示生产每件产品的雇员和构成每件产品的组件。注意，存在两个多对多关系。

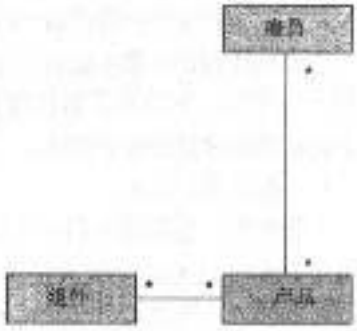


图2-8 多对多关系引发数据库中的问题。在这个例子中，多个雇员将多个组件装配到多个产品上，但我们并不知道哪个雇员实际装配了哪些组件

要理解多对多关系所引发的问题，考虑当公司想知道各个组件由哪位雇员装配成产品时会怎样？要处理这种情况，图2-9中的三个主要实体（雇员、产品和组件）实际上通过Assembly关联互相联系。当多于两个类相互关联时，这种关系称为多重关联，用菱形表示。这种关联（实际上任何关联）可以用其自身的类数据来描述。在这个例子中，装配列表中的一个条目会包含EmployeeID、ComponentID和ProductID。总之，多个雇员可以装配多个产品，多个组件可以装配到多个产品上，每个独立事件由Assembly关联类记录。Assembly关联能解决多对多的问题，这是因为Assembly类的每行保存了某一雇员、某一组件和某一产品的数据。在实际生活中可能还会有Date/Time列，用于记录每个事件发生的时间。

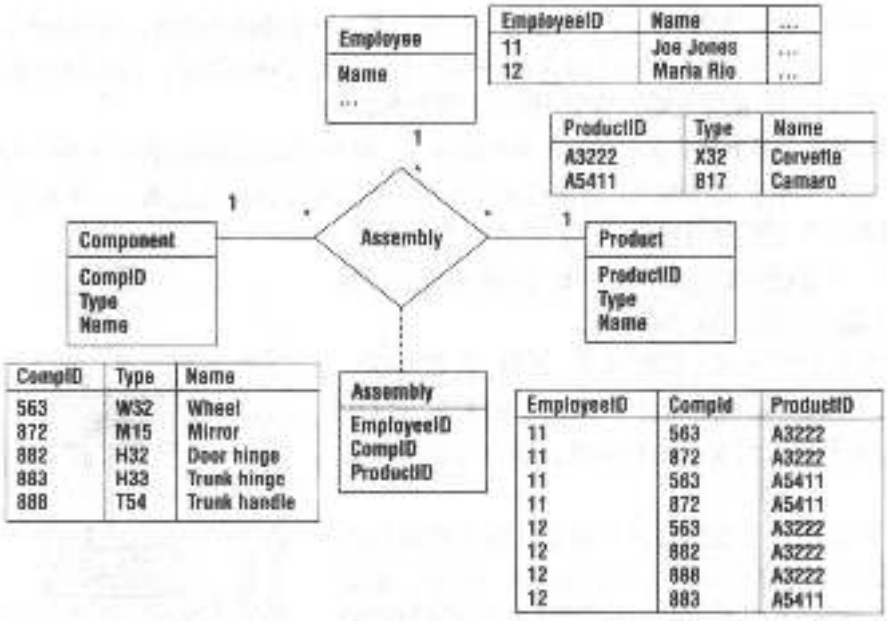


图2-9 多对多关联转化为具有多重关联的一对多关系集合。此多重关联包含一个新类。在此例中，Assembly类的每行存储一个雇员、一个组件和一个产品的数据。注意，Assembly类（框）由虚线与Assembly关联（菱形）相连接

根据UML标准，重复度在多重关联背景下没有意义。类上放置的重复度数字代表当关联中其他n-1个值确定时，关联中可能的对象数目。例如，如果ComponentID和EmployeeID确定，将会有多少产品？换句话说，一个雇员能将相同组件装配到多个产品中吗？在大多数情况下，答案为“是的”，因此重复度一般会是一个表示“多个”的星号。

最终，所有的多对多关系必须通过添加新实体转化为一对多关系的集合。像Assembly实体一样，这个新实体一般代表一种活动，并且包括一个Date/Time时间戳。

作为设计者你会将类图用作不同的用途。有时你需要查看细节；其他时候你只关心整体。对于大型项目，有时用于创建概要图，显示主要类之间的基本关系。在这种图中可以使用多对多关系来隐藏某些细节实体。

2. 关联细节：聚集

关联中的一些特殊类型经常出现，UML定义了专门方法来处理它们。其中一类就是聚集或集合。例如，Sale包括购买的Items的集合。如图2-10所示，聚集由置于关联线上靠近类的小菱形表示，该类是集合体。在此例中，菱形靠近Sale类，具有“多”关系的关联可以有序或无序。在此例中，Items存储的顺序无关紧要。如果顺序有关系，则只需在关联下面放置{ordered}标记即可。要确保在描述词的两侧加上大括号。

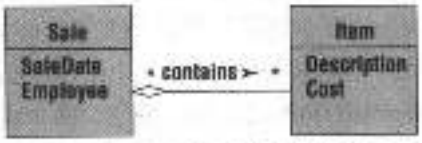


图2-10 关联聚集。Sale包含购买的很多商品。在关联上放置一个小菱形来表示这种特殊的关系

3. 关联细节：组合

简单的聚集表示在业务环境中并不常用。然而，组合是一种更强的聚集关联，它出现得更

多。在组合中，单独的项成为新对象。考虑自行车，它由组件的集合（车轮、曲柄和车闸等）构成。统一建模语言提供两种方式显示组合。图2-11中，单独的类是分开的，并以填充的菱形标记。另一种可选的显示方法如图2-12所示，它通过将组件类绘制在主要的Bicycle类中来表示组合。在内嵌的图中更容易看出关系，但如果需要20个不同的对象来定义自行车，那么表示起来将会非常混乱。

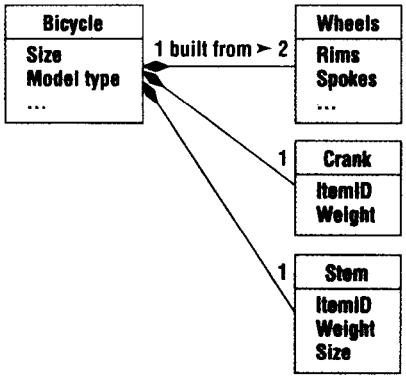


图2-11 关联组合。一辆自行车由很多单独的零部件组成。这些零部件不再单独存在，它们变成了自行车

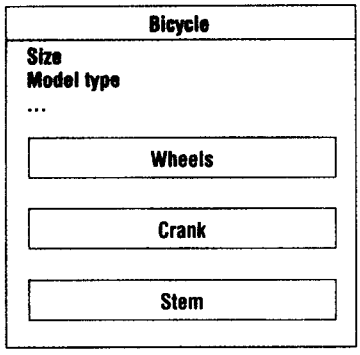


图2-12 关联组合。通过将组件项嵌套在主类中更容易查看组合

聚集和组合之间的区别很小。UML标准规定组合只能表示一对多关系。任何多对多关联必须使用简单的聚集指示符。组合关系一般比聚集关系容易确认，它们在制造业环境中更通用。只要记住，仅当单独的项成为新类时组合才存在。完成自行车制造后，就不再涉及单独的组件。

4. 关联细节：概括

业务环境中出现的另一通用关联是概括。在这种情况下产生一个类层次。最一般的描述在顶端给出，然后更特殊的类从它派生。图2-13显示了Sally的宠物商店中的例子。每种动物都有某些一般的属性（例如，DataBorn、Name，Gender和ListPrice），包含在一般的Animal类中。但特别的动物类型需要一些不同的信息。例如，对于哺乳动物（或许是猫），买主希望了解它的窝的大小以及动物是否有爪子。另一方面，鱼没有爪子，用户希望了解不同的信息，例如它们是淡水鱼还是咸水鱼以及鳞的情况。对每种动物都可以收集同类动物相关数据。每一代有多个阶段在宠物商店例子中，Memmal种类可以进一步划分为Cat、Dog和其他种类。

使用小的、未填充的三角形来表示概括关系。可以将所有子类连接至一个三角形，如图2-13所示，也可以分别绘制每条线段。对于此例的情形，汇集的表示方法是最好的选择，因为，此关联代表了一个不相交（互斥）的集合。一种动物只可能属于某一子类。

概括的一个重要属性是低层类继承高层类的属性和方法。类通常起始于相当一般的描述。更细节的类从这些基类派生。每个低层类从高层类继承属性和函数。继承意味着派生类的对象除了具有它们自身类中定义的属性外，还具有高层类的所有属性。相似地，在相关类中定义的函数对于新类也是有效的。

考虑图2-14所示的银行账户系统的例子。设计者会从客户账户的基本描述开始。银行总需

要其账户的基本信息，例如AccountID、CustomerID、DateOpened和CurrentBalance。相似地，会有公共函数包括开通账户和关闭账户。所有这些基本属性和操作会在基类Accounts中定义。

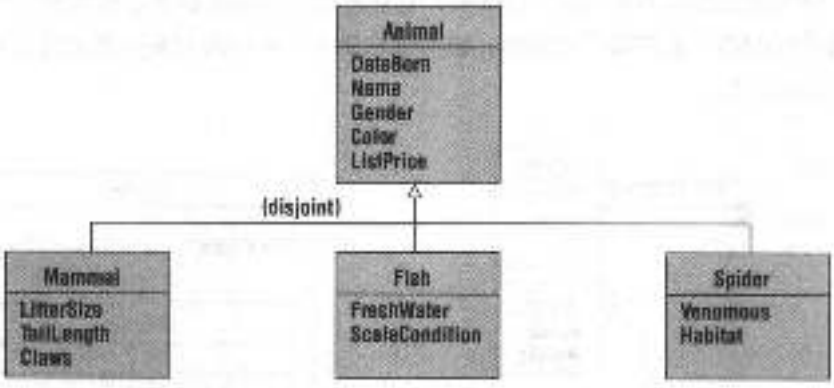


图2-13　关联概括。一般的Animal类保存适用于所有动物的数据。派生的子类包含特定于某一物种的数据

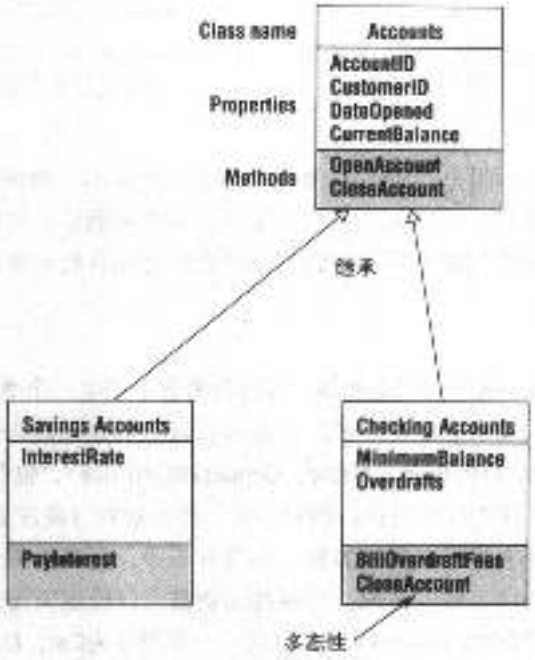


图2-14　类继承。对象类起源于一个基类（例如Account）。其他类从基类继承而来。它们继承基类的属性和方法，并添加新的属性。在银行中，所有账户都需要跟踪基本的客户数据，只有支票账户需要跟踪透支费用

新账户可以由这些账户派生，设计者只需添加新功能——这样便能节省时间和减少错误。例如，Checking Accounts具有MinimumBalance以避免费用，银行必须跟踪每个月Overdraft的次数。Checking Accounts类派生自Accounts基类，开发者添加新的属性和函数，这个新类自动从Accounts类继承所有属性和函数，因此不必重新定义它们。相似地，银行对储蓄存款账户感兴趣，因此，创建Savings Accounts类，它记录当前的InterestRate并包括一个函数用于计算和

存储每月应得的利息。

其他类可以从Savings Accounts和Checking Accounts类派生。例如，银行可能为年长者和学生设立专门的支票账户。这些新账户可能提供较低的费用、不同的最小余额需求或不同的利率。要适应这些修改，设计图只需通过在这些初始定义下面添加新类进行扩展。这些图显示了*类层次*，类层次表示类如何相互派生，并突出哪些属性和函数被继承。UML使用不封口的菱形箭头表示高层的类是更一般的类。例如，Savings Accounts和Checking Accounts类派生自一般的Accounts类，因此，关联线段从前者指向后者。

图2-14中的每个类也能执行独立的函数。在类中定义属性和方法称为*封装*。它的优点在于将所有相关定义放在一个地方。由于属性和函数可以在应用的其他部分受到保护，因此封装还提供了一些安全和控制功能。

注意，Accounts类有一个用于关闭账户的函数，从这里可以看出封装的另一个有趣的属性。仔细观察，你会发现Checking Accounts类也有一个函数用于关闭账户（CloseAccount）。当派生类定义了与父类相同的函数，称为*多态性*。当系统激活此函数时，会自动确定对象的类并执行匹配的函数。设计者也能指定派生的函数（Checking Accounts类中的CloseAccount方法）可以调用基类中的相关函数。在银行业务的例子中，Checking Account类的CloseAccount函数会取消未兑现支票、计算当前费用，并更新主要的余额。然后它会调用Accounts类的CloseAccount函数，这个函数会自动将数据存档并从当前记录中删除对象。

多态性对于应用建立者来说是有用的工具。它意味着可以调用一个函数而不管数据的类型。在银行业务例子中，可以简单地调用CloseAccount函数。为响应调用，不同的账户会执行不同的操作，但应用并不关心。应用的复杂性转移到了设计阶段（在那里定义所有类）。应用建立者不必关心细节。

注意，在复杂情况下，一个子类可以从多个父类继承属性和方法。在图2-15中，小汽车是自动化的，并且设计为上路使用，因此它从两个类（还从一般的Vehicle类）中继承属性。自行车的情况稍微复杂一些，因为它可以从On-Road类或从Off-Road类继承属性，这取决于自行车的类型。如果你要记录混合型自行车的数据，Bicycle类必须从On-Road和Off-Road这两个类继承数据。

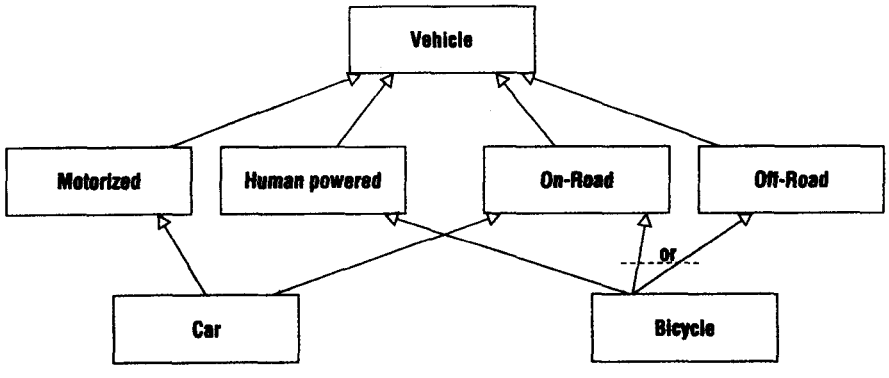


图2-15 多重父类。类可以从若干父类继承属性。关键在于绘制结构以便用户能够理解并确保绘制的结构符合业务规则

5. 关联细节：自反关联

自反关系是业务中产生的需要特殊处理的另一种情况。自反关联是类到其自身的关系。最通用的业务情形如图2-16所示。一些雇员是管理者，他们管理其他雇员。这样就会有从雇员（管理者）回到雇员（工人）的关联。注意，UML如何允许你在关系（管理者和工人）的两端进行标记。另外，“◀ manages”标签指明应该如何识别关联。标签和文字阐明了关联的目的。某些关联可能不需要标注，但自反关联总是应当仔细说明。

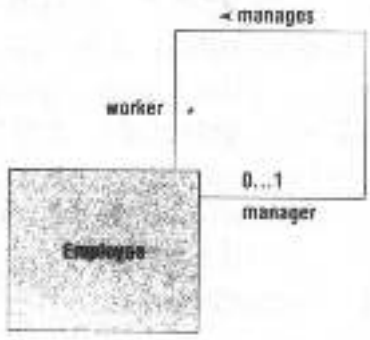


图2-16　自反关系。管理者是管理其他员工的雇员。注意标记是如何表达这种关系的

6. 关联细节：小结

最后这几节描述的不仅仅是给你的类图做一点变化，它们描述了通用业务状况。你需要确认这些状况，因为它们会影响你设计和构建数据库应用的方法。要创建类图，首先确定主要的类，包括它们的属性，然后注意这些类之间的关联，特别要注意正确的重复度。一定要警惕多对多关系。当你看到它们时，寻找多重关联或者能提供一对多关系的新类。寻找组合或聚集的情况，是否一个类由很多小对象组成？如果是，那么考虑主类中嵌入的子类。注意概括的例子，是否有相同目的的类？如果有，试着定义一个一般的类并使用继承派生这些细节的类。注意不相交（或互斥）的类。确保在图中指明它们。最后，寻找自反关联，在自反关联中，一个类的对象与相同类中的其他对象关联。在Employee类中查找这种关系，但它可能出现在其他类中。例如，公司可以生产某种产品，而这种产品由其他产品构成。

此时，你不必绘制一个完美的类图，只需正确确认所有的业务规则。第3章讲解如何分析类和关联，以及如何创建更好的类集。

2.6　Sally的宠物商店类图

宠物商店的主人Sally希望分阶段创建应用程序。第一个阶段将追踪商店的基本事务数据。因此，你需要确认宠物商店运营中的主要实体。

设计宠物商店数据库应用的第一步是与其主人（Sally）交谈，调查其他商店，确定所需的基本组件。在与Sally交谈后发现，这个宠物商店具有一些与其他零售商店不同的功能。最重要的区别在于宠物商店必须追踪两种不同的销售类型：对动物的处理与对产品的处理不同。例如，宠物商店对于动物要追踪更细节的信息。另外，产品能够以多个单位销售（例如，六罐狗食），但动物必须单独追踪。图2-17显示了Sally的宠物商店的初始类图。宠物商店建立在这些基本实体之上。类图突出显示了动物和货物的两条不同路径。

由于Sally要为宠物提供好的去处，她想收集每位客户的详细信息。Sally还比大部分店主更关注供应商。她甚至在考虑雇人去调查各个动物饲养者。调查人员会提供多方面的报告，例如清洁程度、动物的数目、驯养员的数目、每餐的食物类型和兽医照顾的质量。

当与Sally交流的时候，优秀的设计者会记录下将要包含在数据库中的数据。这个列表由实体组成，我们需要收集这些实体的数据。例如，对于宠物商店数据库，很明显需要收集客

户、供应商、动物和产品的数据。同时，需要记录每一笔购买和销售信息。开始时，要确定这些实体的不同属性或特性。例如，客户具有姓名、地址和电话号码。对于每个动物，要了解动物的类型（猫、狗等等）、品种、出生日期等等。

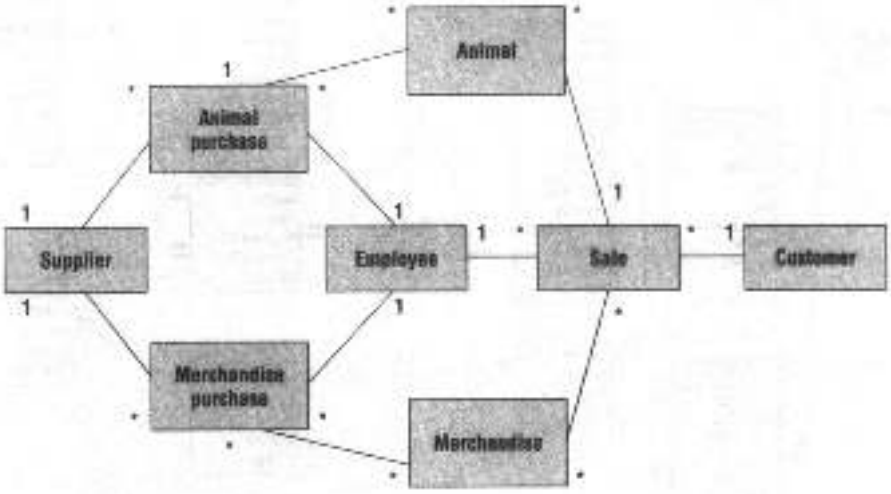


图2-17 宠物商店的初始类图，动物的购买和销售与货物不同，因此这个商店需要对这两类实体记录不同的数据

详细类图会包括每个实体的属性。注意图2-17所示的初始类图包含几个多对多关系。所有这些都需要添加中间类。考虑MerchandiseOder类，一次可以订购多件商品，因此，你将创建一个新实体（OrderItem），它包含每个MerchandiseOrder订购的所有商品。AnimalOrder和Sale实体的作用类似。

图2-18给出了宠物商店更详细的类图，它包含这些新的中间类。它还包含City、Breed和Category这几个新类。几乎在每个业务数据库中邮政编码和城市都会引发问题。城市和邮政编码有关，但并非一对一的关系。一种简单的解决办法是为每位客户和供应商存储城市、州和邮政编码，然而，对于本地客户来说，为每次销售存储城市和州的名称是非常烦琐的。一个解决办法是在单独的类中存储城市和邮政编码数据，通用的值可以在初始时输入。雇员可以从已有列表中选择想要的城市而不必重新输入数据。

Breed和Category类用于确保数据的一致性。文本数据的输入问题之一是人们输入的数据可能不一致。例如，有些职员可能将达尔马提亚狗简写为Dal，其他人可能使用Dalma，而少数人可能输入全称。要解决这个问题，我们希望在单独的类中一次性存储所有的种类名称。这样雇员只需从这些类存储的列表中选取种类即可。这样，每次输入的数据便都一致。

宠物商店的概括和详细的类图都可以用来与用户交流。通过其中的实体和关系，类图显示了公司的业务规则。例如，分别对待动物和商品对于店主是很重要的。同样，每次销售中只能有一位客户也是重要的业务规则。这些规则应由Sally确认。如果一个家庭购买了一只动物，她是否希望了解这个家庭中的每个成员？如果是，你需要添加一个Family类列出每位客户的家庭成员。重点在于你可以使用类图显示新系统，检验假设，获得新想法。

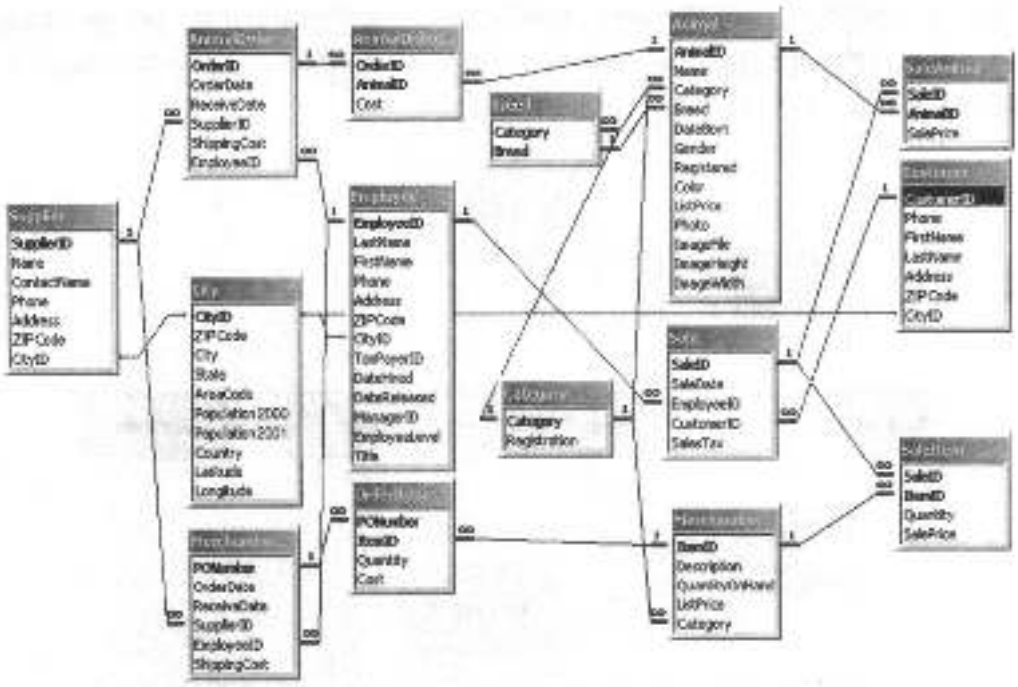


图2-18　宠物商店的详细类图。注意为解决多对多问题而添加的新表：OrderItem、AnimalOrderItem、SaleItem和SaleAnimal，添加City表来减少数据条目，添加Breed和Category表来保证数据一致性，用户从这些表中选择类型和品种，而不是输入可能每次都不同的文字或缩写。Microsoft Access使用无穷大符号（∞）代替星号（*）来标记关系中表示多数的一边

2.7　数据类型（域）

当你列出每个类中的属性时，应当考虑它们是什么数据类型。每个属性具有一个特定的数据类型或数据域。例如，EmployeeID是什么？它是数值吗？从什么值开始？怎样递增？它包含字母或其他数字字符吗？你必须确定每个属性或列的域，图2-19给出了一些通用域。最通用的是文本，它存储字符。

注意，任何域都可以保存缺失数据。用户并不总知道某些项的值，因为它可能未输入。缺失的数据由空值定义。

2.7.1　文本

文本列通常限制在不超过255个字符。有些数据库管理系统要求区分定长和变长文本。定长字符串总是占用你分配的空间规模，在处理像身份证号码或双字母州缩写这样的短字符串方面能最有效地提升速度。变长字符串存储时，它们只占用每行数据实际所需的空间。

备注或注释列也用于存储变长文本数据。不同于变长文本的是数据库中备注可以分配更多的空间，确切的限制取决于所使用的DBMS和计算机，但备注类型的数据库列一般大至32K或64K。备注列一般用于长注释甚至短报表。然而，有些系统限制在备注列上执行的操作，例如不允许排序备注数据。

	Access	SQL Server	Oracle
Text			
fixed		char,	CHAR
variable	Text	varchar	VARCHAR2
Unicode		nchar, nvarchar	NVARCHAR2
memo	Memo	text	LONG
Number			
Byte (8 bits)	Byte	tinyint	INTEGER
Integer (16 bits)	Integer	smallint	INTEGER
Long (32 bits)	Long	int	INTEGER
(64 bits)	NA	bigint	NUMBER(127,0)
Fixed precision	NA	decimal(p,s)	NUMBER(p,s)
Float	Float	real	NUMBER, FLOAT
Double	Double	float	NUMBER
Currency	Currency	money	NUMBER(38,4)
Yes/No	Yes/No	bit	INTEGER
Date/Time	Date/Time	datetime smalldatetime	DATE
Interval	NA	interval year . . .	INTERVAL YEAR . . .
Image	OLE Object	image	LONG RAW, BLOB
AutoNumber	AutoNumber	Identity rowguidcol	SEQUENCES ROWID

图2-19 数据类型（域）。通用数据类型及其在三个数据库系统中的变化。在SQL Server和Oracle中以“N”开头的文本类型保存Unicode字符集，对于非拉丁语言特别有用

2.7.2 数值

数值数据也很通用，计算机认可若干不同种类的数值数据。对于数值数据列，需要做的最重要决定是在整数和浮点数之间进行选择。整数不能存储分数（小数点右边的值）。整数通常用于计数和存储诸如1、2、100和5 000这样的值。浮点数能够包含分数值和存储像3.141 59和2.718这样的数字。

整数和浮点数引发的第一个问题是，为什么要关心它们？为什么不将所有的数字作为浮点数存储？答案在于计算机存储两种数值类型的方式。特别地，大部分计算机用2（或4）字节存储每个整数值；但用4（或8）字节存储每个浮点数值。尽管2字节的差别似乎是微不足道的，但当数据有几百万行时会产生巨大的差异。此外，整数计算本质上要快于浮点数计算。简单的如两个数字相加，整数要比浮点数快10到100倍。尽管计算机变得越来越快，存储成本也一直在降低，但当处理大规模数据库时，性能仍然是一个重要的问题。如果可以将一个数值存储为整数，那么尽量这样存储，你会获得很高的性能。

大部分系统还支持长整数和双精度浮点数。这两种情况都需要双倍于单精度数据的存储空间。设计者的主要问题是确定用户所需数值和精度的大小。例如，如果预期有100 000客户，那么，不能使用整数来追踪客户（码值）。注意，16位整数只有65 536个值。要计算或度量较大的值，需要使用长整数，它的范围在+/−2 000 000 000之间。相似地，浮点数能支持6位有效数字。尽管量值（幂）可以很大，但只能存储六位或七位数字。如果用户需要更高的精度，

那么使用双精度值，它能存储14位有效数字。图2-20列出了通用数据类型的最大值。

数据类型	数据规模		
	Access	**SQL Server**	**Oracle**
Text (characters) fixed variable memo	 255 64K	 8K, 4K 8K, 4K 2M, 1M	 2K 4K 2G
Numeric Byte (8 bits) Integer (16 bits) Long (32 bits) (64 bits) Fixed precision Float Double Currency Yes/No	 255 +/− 32 767 +/− 2 B NA NA +/− 1 E 38 +/− 1 E 308 +/− 900.000 0 trillion 0/1	 255 +/− 32 767 +/− 2 B 18 digits +/− 1 E 38 +/− 1 E 38 +/− 1 E 308 +/− 900.000 0 trillion 0/1	 38 digits 38 digits 38 digits p: 38 digits p: −84 to 127; s: 1 to 38 38 digits 38 digits 38 digits
Date/Time	1/1/100 − 12/31/9999 (1 sec)	1/1/1753 − 12/31/9999 (3 ms) 1/1/1900 − 6/6/2079 (1 min)	1/1/−4712, 1/31/9999 (sec)
Image	OLE Object	2 GB	2 GB, 4 GB
AutoNumber	Long (2 B)	2 B or 18 digits with bigint	Column: 38 digit maximum

图2-20　数据规模。确保你选取的数据类型能存储你将遇到的最大值。选取过大的尺寸会浪费空间和降低计算速度，但如果不确定数据大小的话，还是选取较大的尺寸

很多商业数据库遇到了一个不同的问题。货币值通常需要很多位数，而且用户不能容忍四舍五入的错误。即便使用长整数，仍会限制低于2 000 000 000（如果你需要双小数点值则为20 000 000）。双精度浮点数允许存储上十亿的数字，即便是双小数点值。然而，浮点数的存储通常有四舍五入的错误，这会影响会计的工作，因为他们的计算需要精确到美分。要弥补这些问题，数据库系统提供了一个货币数据类型，它作为整数值（带有一个附加的小数点）存储和计算。它们计算速度很快，可以存储大至万亿的数值，并且将四舍五入的错误最小化。有些系统提供一个通用的定点数据类型。例如，可以指定你需要4位十进制数字精度，然后，数据库在存储数据和执行计算时将使用4位十进制数字。

2.7.3　日期和时间

所有数据库都需要一个特别的数据类型来存储日期和时间。大部分系统将这两者合为一个域，有些则提供两种单独的定义。很多初学者试图将日期存成字符串或数值。请避免这种尝试。日期类型具有重要的特性。日期（和时间）实际上存储为单独的数字。日期一般存储为整数，是距某个起始日期的天数的计数。这个起始日期可能随系统不同而不同，但它只在内部使用。以计数方式存储日期的价值在于系统能够自动执行日期计算。你能够很方便地计算两个日期之间的天数，也可以要求系统计算今天往后30天的日期。即使那一天处在不同的月份或年份之中，系统也可以自动计算出正确的日期。尽管大部分系统需要8字节来存储日期/时

到列，这样做不需要关心任何年份转换问题。

使用内部日期和时间表示方法的第二个重要原因在于，数据库系统能在内部格式和任何通用格式之间进行转化。例如，在欧洲国家，日期一般以日/月/年的格式显示，而不是美国通用的月/日/年格式。使用通用的内部表示方法，用户可选择他们喜欢的输入或查看日期的方法，DBMS自动将其转化为内部格式，因此内部日期总是一致的。

数据库还需要存储时间间隔的能力。通用的范例是用一列存储年、月、日、分钟甚至秒。例如，你可能要存储雇员完成一个任务所需的时间长度。如果没有特定的时间间隔数据类型，你会将它存储为数值。然而，你必须记录这个数值的含义——它可能是小时，分钟或秒。使用一个特定的时间间隔类型，混淆的几率会降低。

2.7.4 二进制对象

将对象或二进制大对象（BLOB）作为一个单独的类是一个相对新的领域。它使你能够存储任何计算机创造的对象类型。一个实例是使用BLOB存储其他软件包中的图片和文件。例如，数据库中的每行可以存储不同的电子数据表、图片或图表。工程数据库可以存储不同组件的图片和说明书。其优势是所有数据存储在一起，使用户更容易查找他们需要的信息并简化备份。相似地，数据库可以存储一个电子数据表的不同版本，以显示它如何随时间而变化或记录不同用户对它的修改。

2.7.5 计算值

有些业务属性可以计算。例如，一次销售的总额可以由单个商品的价格之和加上销售税来计算，而雇员的年龄可以由今天的日期与DateOfBirth之差计算。在设计阶段，你应当指明哪些数据属性可以计算，UML的符号表示方法是在属性名称前加斜杠（/），并在注释中描述计算方法。例如，一个人年龄的计算如图2-21所示。注释用一个带折角的框显示。用虚线与适当的属性相连。

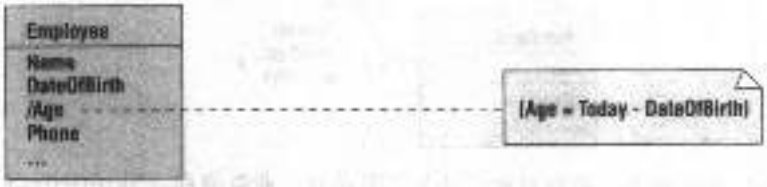


图2-21 派生值。不必存储Age属性，因为它可以从出生日期计算得到。因此，需要在类图中标注，计算的属性名称由斜杠开始

2.7.6 自定义类型（域/对象）

一些较大的数据库系统支持相对较新的对象-关系属性，你可以定义自己的域为现有类型的结合。这个域本质上成为一个新的对象类型。最容易明白的例子之一是经纬度代码。你可以用纬度和经度的方式定义一个地理位置，如果有可能，加入海拔高度。在简单的关系DBMS中，这个数据以不同的列存储。当你想使用这个数据时，必须向你的代码查找并传递所有的值。有了自定义数据类型，你可以创建一个叫做位置的新数据类型，它包括需要的组成部分。

这样，你的列定义只有一个单独的数据类型（位置），但它实际上存储着两个或三个数据。DBMS把这些元素作为一个单独的条目对待。注意，当你创建一个新域时，你还必须创建用于比较域值大小的函数，这样你才能排序和查找新的数据类型。

2.8 事件

事件是现代数据库系统中需要了解的另一个重要的组成部分。数据库环境会产生三类基本的事件：

1) 触发某些函数的业务事件，例如销售会触发库存减少。

2) 数据改变导致发出某些警告，例如库存低于预先设定的值会触发新的购买订单。

3) 用户界面触发某些操作，例如用户点击图标向供应商发出购买订单。

事件是依赖于时间的操作。UML提供了许多图用于描述事件。协作图可能是记录业务流程和事件最有效的方法。复杂的用户界面事件可以用序列图或状态图来显示。后两种图超出了本书的范围。你可以参考面向对象设计课程来获取如何描述它们的详细信息。

业务事件可能相互关联。因此一个事件可以触发另一个事件，等等。对于复杂的事件链，或许应当绘制一个流程图来显示期望的事件序列。图2-22是一张小型协作图。它显示了三个类如何通过交换信息和调用不同类的函数来相互作用。注意，由于顺序很重要，因此三个主要触发行动被顺序编号。首先，调用Order类传送一个订购，这触发一条消息给Inventory类，以减少适当的库存数量。当库存数量改变时，自动触发器调用一个例程分析当前的库存等级。如果达到适当的标准，会产生一个购货订单并记录购买的产品。

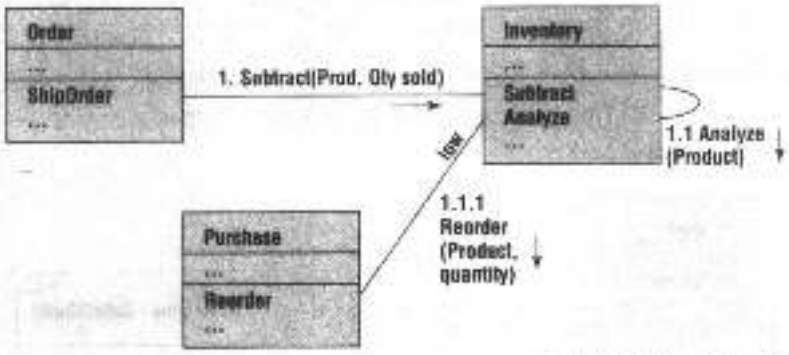


图2-22　合并模型，数据对象设计为易于存储，业务流程（ShipOrder和AnalyzeInventory）是引发数据对象改变的事件。例如，运送一个订单引起库存的改变，它接着触发对于当前库存级别的分析，它能触发新的库存订货

这个例子描述了一个事件链，它相对容易理解和测试。更复杂的链可以自身循环并包含更复杂的选项。UML序列图可用于更详细地显示单个消息是如何按适当顺序处理的。UML状态图突出类/对象的状态随时间的变化情况。事件在构建系统中是很重要的，它能够包含复杂的交互。现在，你应该能够绘制简单的协作图指示主要的消息事件。

简单的情况下，可以保存一个重要事件列表，将重要事件定义为触发器，它描述事件的起源和采取的相应操作。例如，基于库存数据的业务事件可以描述成图2-23所示。像Oracle和

SQL Server这样的大型数据库系统直接支持触发器。你可以定义事件并附加上当条件满足时将要执行的代码。

在设计阶段，这些触发器可以描述成任何基本形式（例如伪代码），随后转成数据库触发器或程序代码。统一建模语言还提供了一种对象约束语言（OCL），可以用来编写触发器和其他代码段。如果使用一种能将OCL代码转入所使用的数据库工具，将会十分通用和有效。

```
ON (QuantityOnHand < 100)
THEN Notify purchasing Manager
```

图2-23 示例触发器。列出条件和操作

2.9 大型项目

如果为自己或单独的用户建立一个小型数据库，可能不会花时间为整个系统绘图。然而，你确实应当提供一些文档，这样，当其他设计者修改你的工作的时候，知道你做了些什么。另一方面，如果你正在开发一个大型项目，包括很多开发者和用户，那么每人都必须遵守一个公共的设计方法。何谓大型项目，何谓小型项目？这里没有固定规则，但当项目包括若干开发人员和许多用户时，你会遇到图2-24中列出的问题。

大型项目的方法论起始于绘图，就像本章描述的类图和协作图。然后，每个公司或团队添加细节。例如，选择标准来指定命名规范，需要的文档类型和检查流程。

大型项目的挑战在于将项目分割成可由单个开发人员处理的小模块。而且这些小模块在最后必须合成一体。项目经理还需要根据时间和费用计划项目。开发项目时，管理者能够根据进度评估团队成员。

大型项目的设计更困难
- 与多用户交流
- IT工作者之间的交流
- 需要为团队将项目分块
- 查找数据/组件
- 人员周转——再培训

需要监督设计过程
- 安排进度
- 评估

建立以后可修改的系统
- 文档
- 交流/基本假设和模型

图2-24 开发大型项目的问题。大型项目更需要进行交流、坚持标准以及监督项目

很多类型的工具能帮你设计数据库系统，而且都特别适用于大型项目。要辅助设计和监控进度，管理者可以使用项目计划工具（例如Microsoft Project），帮助建立Gantt和PERT图，将项目分成小块，并突出各部分之间的关系。计算机辅助软件工程（CASE）工具（例如Rational Rose）可以帮助团队绘图，贯彻标准和存储所有项目文档。此外，群件工具（例如Lotus Notes/Domino）帮助团队成员以文档、设计和程序方式共享他们的工作。这些工具标记变化，记录作出修改的人和他们的意见，并跟踪版本。

就像图2-25中总结的那样，CASE工具可以为开发者提供很多有用的功能。除了辅助绘图以外，CASE工具最重要的功能之一是维护项目的数据仓库。开发者定义的每个元素都存储在数据仓库中，数据在此可以与其他开发者共享。换句话说，数据仓库是一个特殊的数据库，它保存所有与项目设计相关的信息。有些CASE工具依据你提供给CASE项目的信息，能够生成数据库和应用。逆向工程工具能从现有的应用中读取文件，生成对应的设计元素。很多公

司提供CASE工具，包括Rational Software、IBM、Oracle和Sterling Software。通过增进开发者之间的交流和生成代码，CASE工具能够加速设计和开发过程。通过提供系统的完备文档，它们还具有缩短维护时间的潜力。

优秀的CASE工具已经有若干年的历史，但仍有很多公司不使用它们，一些公司虽然使用，但是没能体会到潜在的优势。CASE工具的两个缺陷是其复杂性和成本。如果对于一个已知项目，这个工具能减少开发者的人数，就可以降低成本。但其复杂性问题是很大的障碍。学会有效使用一种CASE工具可能要花费一个开发者几个月的时间。幸运的是，有些CASE厂商提供折扣给大学，帮助培训学生使用他们的工具。如果你有机会接触CASE工具，应尽可能多地在任务中使用它。



图2-25　CASE工具的功能。CASE工具辅助建立和维护图，还支持团队工作和文档控制。有些可以从设计生成代码。其他可以检查应用并通过逆向工程创建相应的代码

2.10　Rolling Thunder自行车

Rolling Thunder自行车案例阐明了一些业务环境中出现的通用关联。因为这个应用是为课程设计的，所以很多业务有意简化了。最高层的视图如图2-26所示。图中列出了看似松散的几个公司行为，分为六个包：Sale、Bicycles、Assembly、Employees、Purchasing和Location。包并不相等：有些包的信息细节远多于其他包。特别是，Location和Employee包目前只含有一到两个类，被当作单独的包对待，因为它们都与多个包中的若干类交互。由于它们要处理独立、自包含的问题，所以分离它们才有意义。

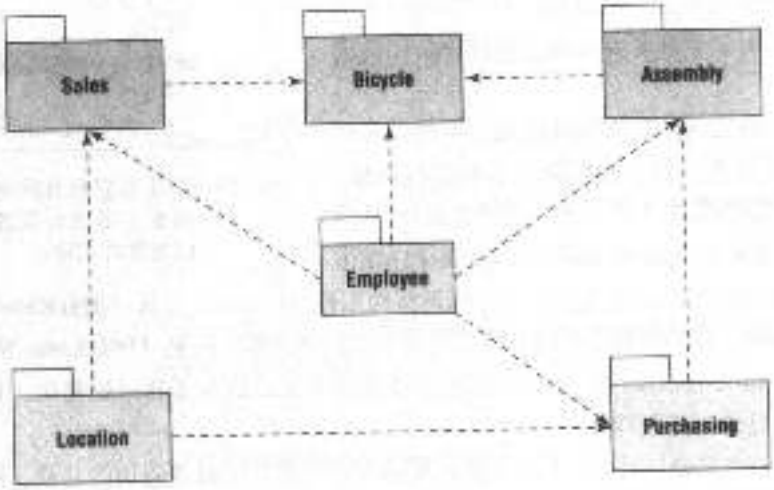


图2-26　Rolling Thunder自行车——顶层视图。这些包松散地基于公司的活动。每个包的目标是描述一个独立的对象集合，它与其他包相互作用

每个包包括类和关联的集合。Sales包在图2-27中更详细地描述。为了最小化复杂性，与

其他包之间的关联在此图中没有显示，例如，Customer和RetailStore类与Location::City类具有关联。这些关系将会在Location包中显示。因此，Sales包是很直接的。Customers提交订单订购Bicycles，他们可以使用RetailStore来帮助提交订单，但也不必这样做。这样，关联的RetailStore端有（0...1）复杂度。

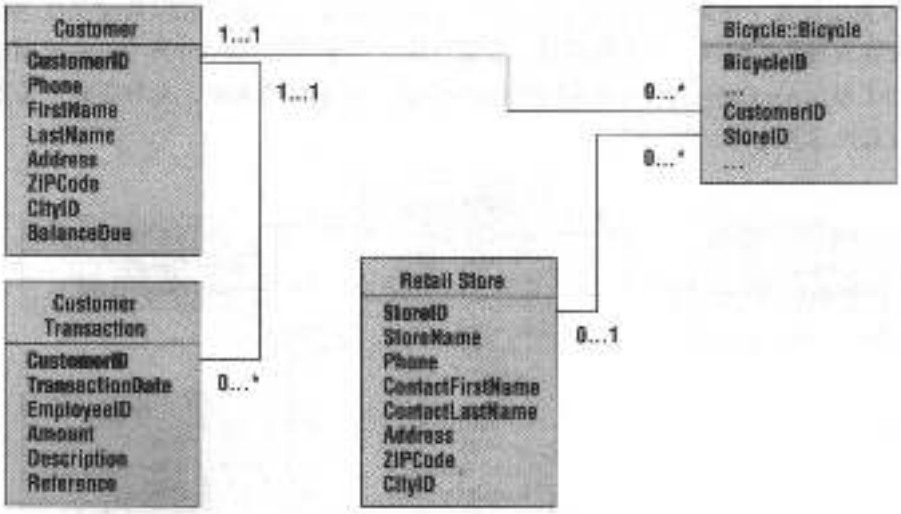


图2-27 Rolling Thunder自行车——Sales包，与其他包的一些关联没有显示在这里（参考其他包）

Bicycle包具有使这个公司独特的许多细节。为了节省空间，图2-28中仅显示了Bicycle类的一部分属性。注意，自行车由管子集合与零部件集合组成。用户可以选择用于制造自行车的材料种类（铝、钢、碳化纤维等等）。他们还可以选取自行车的零部件（车轮、曲柄、踏板等等）。这两类都与Bicycle类具有组合关联。Bicycle类是这家公司最重要的类之一，它与

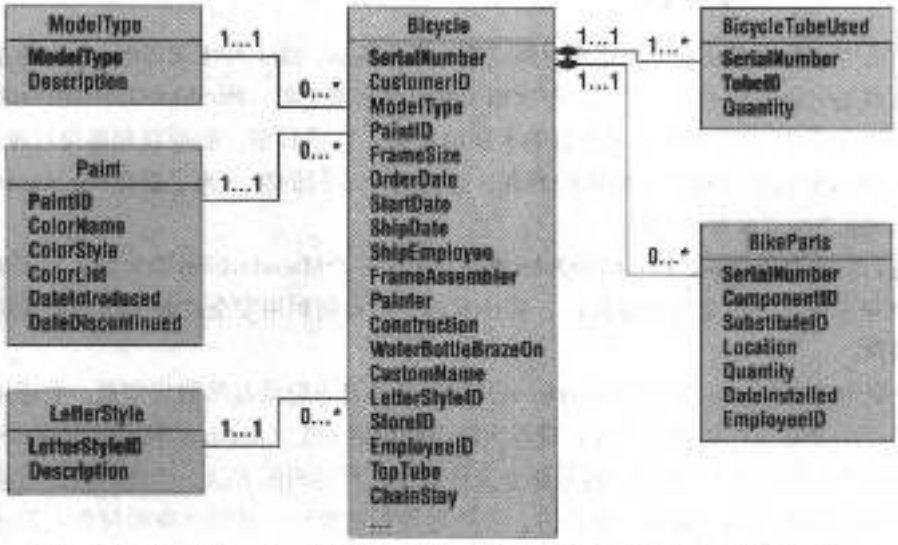


图2-28 Rolling Thunder自行车——Bicycle包，注意从BikeTubes类和BikeParts类到Bicycle类的组合关联，为了节省空间，只显示了Bicycle的部分属性

BicycleTubeUsed和BikeParts类一起完全定义了每辆自行车。Bicycle类还包含制造了这辆自行车的雇员信息。这种决定是简化设计的选择。另一种选择是将ShipEmployee、FrameAssembler和其他雇员属性转移到Assembly包中的一个新类中。

如图2-29所示，Assembly包包含有关组成自行车的各种零部件和管子的详细信息。实际上，Assembly类还包含一些重要的事件。当自行车装配完成后，数据加入以指明完成这项任务的雇员以及完成任务的时间。这个数据现在存储在Bicycle包中的Bicycle类中。需要创建协作图或序列图来显示Assembly包中各种事件的详细信息。对于目前来说，类和关联更为重要，因此这里没有显示其他图。

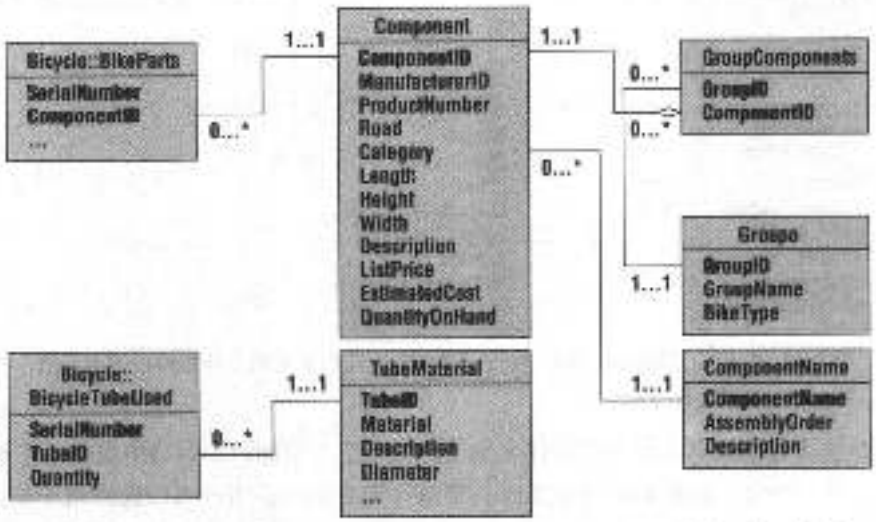


图2-29　Rolling Thunder自行车——Assembly包。装配过程中会产生若干事件，但没有显示在这张图上。当装配自行车时，附加数据输入到Bicycle包的Bicycle表中

所有的零部件需要从其他制造商（供应商）那里购买。图2-30所示的Purchase包是这个行动相当传统的一种表现形式。注意，每次购买需要使用两个类：PurchaseOrder和PurchaseItem类。PurchaseOrder类是主类，包含订购本身的数据，包括日期、制造商和提交订单的雇员。PurchaseItem类包含订购物品的详细列表。包含这个专门的类，为了避免PurchaseOrder和Component类之间的多对多关联。

注意，根据业务规则，PurchaseOrder类必须包含一个ManufacturerID字段。不了解制造商的身份就提交购买订单是非常危险的。第10章将介绍如何利用安全控制为这方面业务提供更安全的保障。

一个附加类（ManfacturerTransactions）用作交易日志记录每笔购货信息。它还用于记录付给制造商的货款。在购买这一方，表现出微小的数据重复（AmountDue在PurchaseOrder和Transaction类中都存在）。然而，这是建立会计系统相对通用的方法。传统的会计方法依赖于将所有相关的交易数据存储在一个地方。无论如何都需要记录对制造商的付款。因此数据重复是相对次要的。

图2-31中的Location包用于集中城市和地址相关的数据。很多类具有地址属性。在较老的

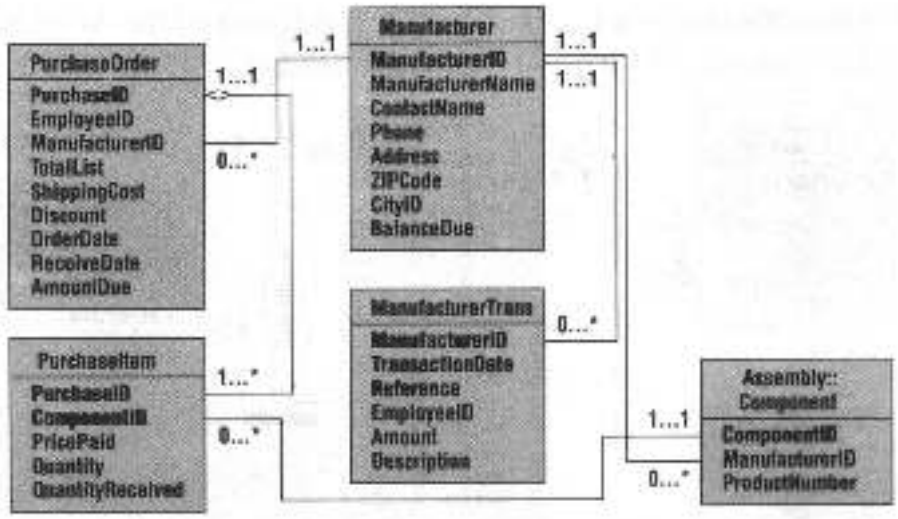


图2-30 Rolling Thunder 自行车——Purchasing包。注意使用Transaction类在同一位置为制造商存储所有相关的金融数据

系统中，很容易将城市、州和邮编数据复制并存储到每个涉及地点的类中。然而现在，获取关于城市的有用信息并将其存储在一个集中的表中是相对简单的。这种方法在速度和数据完整性方面简化了数据条目。职员可简单地从列表中选取一个地点。数据的输入总能保持一致。例如，你不必担心城市的缩写。如果电话区号或邮政编码发生改变，你只需在一张表中修改它们。你还能存储对于管理者有用的额外信息。例如，人口和地理位置信息可用于分析销售数据并指导销售计划。

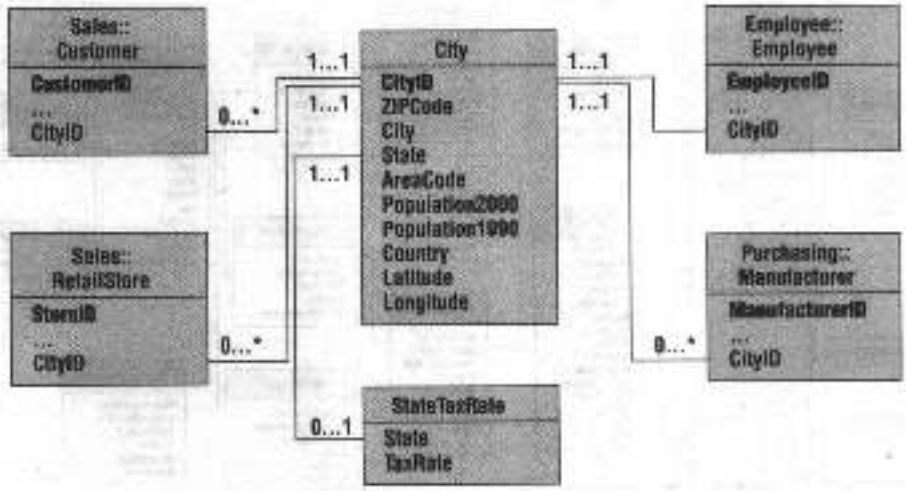


图2-31 Rolling Thunder 自行车——Location包。通过将与城市相关的数据集中，你能够提高职员的办公效率和数据的质量，还可以存储对经理有用的关于地点的额外信息

Employee包要单独对待，因为它与其他很多包相互作用。图2-32所示的Employee类属性非常简单易懂。注意表示管理关系的自反关联。当前Employee包中只有一个类。实际上这是

你存放人力资源数据和关联的地方，例如，你可能希望随时追踪雇员评估、任务和晋升情况。附加类一般会与利益有关，例如休假时间、私人时间和保险选项。

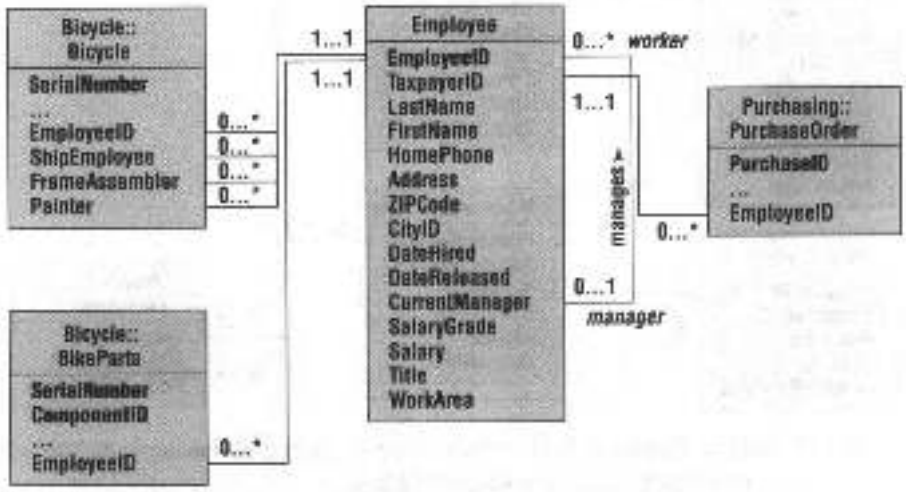


图2-32　Rolling Thunder自行车——Employee包。注意使用自反关联来表示经理

图2-33显示了Rolling Thunder自行车企业的详细、综合的类图。有些关联不包含在图中——部分原因是为了节省空间。更重要的原因是图中所有关联都是Microsoft Access绘制的。

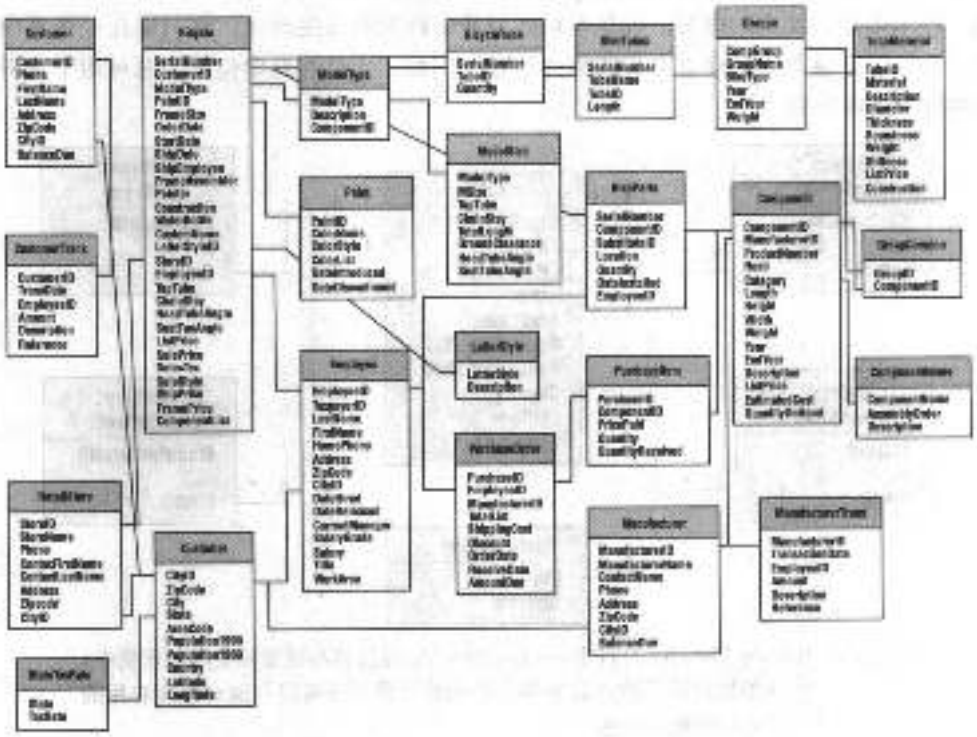


图2-33　Rolling Thunder详细的类图。Microsoft Access与UML稍有不同。例如，Access用无穷大符号（∞）代替星号（*）来表示多的关系。类图是了解企业的不错的参考工具，但对于很多公司，这种详细程度的类图会很大，以至于不能显示

例如，当你定义一个从Employee到Bicycle的关联时，Access将只允许你向Bicycle类中加入一个Employee类中存在的EmployeeID字段。这种执行方式对于处理这项任务的人来说是有意义的。事实上，金融关联应当这样严格地定义。另一方面，公司可能会雇佣临时工人来喷漆和设计装配。在这种情况下，管理者可能不希望记录这些额外的人员，因此从Employee到Bicycle表中Painter的关联是不严格的。

2.11 应用设计

类和属性的概念似乎很简单，但是很快就变得非常复杂。练习和实践能使这个过程容易一些。目前，先学会把精力集中在项目中最重要的对象上。一般来说，从问题的某一部分开始是最容易的，先定义基本元素，添加细节，然后扩展到其他部分。当你设计项目时，请记住，每个类都会变成数据库中的一张表，类中的每个属性是一列，每一行表示某一特定对象。

你还应当根据用户可见的表单或界面来考虑应用设计。考虑图2-34中的简单表单。在纸上，表单中需要输入的每项数据会简单地显示为空白。最终，你可以建立一个相同的带空白的表单作为数据库表单。在此例中，你可能考虑到只有一张表与这个表单相关联；然而，你需要思考潜在的问题。在表单中的空白处，人们可以输入任何想要输入的数据。例如，用户真正了解所有的种属类别吗？或者他们会不会经常空着它、填写缩写或拼错单词？所有这些情况都会给你的数据库带来问题。因此，为他们提供一个选择框会更好一些，用户只需简单地从列表中选取合适的项。但那意味着你需要另一张表来存储可能的种属类别。还意味着你需要在Breed表和Animal表之间建立关系。这个关系会影响应用的实用方式。例如，某人必须先将所有预先定义的名称输入到Breed表中，然后Animal表才能使用它。

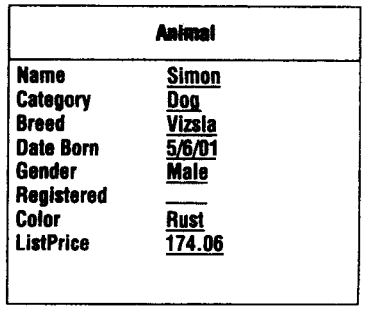


图2-34 基本的Animal表单。开始时这张表单似乎只需要一张表（Animal）。但是，为了将数据输入错误减至最小，实际上需要一张表来存储Category和Breed的数据，在表单上这些数据可以通过一个选择框进行输入

在开发中的这一点上，你应当与用户交流并收集他们想要的任何表单和报表。你应当能够绘制一个初始类图，展示主要的业务对象和相互关联——包括关联的重复度。你还应当把哪些属性当作主码，用哪些主码来建立表作为设计的重点。同时，还需要指定每个属性的数据域。

小结

管理项目以建立有效的应用和控制成本是一项重要的任务。项目管理的主要步骤是可行性研究、系统分析、系统设计和实现。尽管这些步骤可以被压缩，但是不能被跳过。

主要的目标是设计一个能给用户提供所需利益的应用。建立系统模型来描述系统。这些模型可用于与用户交流，与其他开发者交流，并帮助我们记住系统的细节。因为在开发数据库应用中，定义数据是至关重要的一步，因此，类图是很流行的模型。

通过确认系统中的主要实体来创建类图。实体由类来定义，类由名称标识并由每个实体的

属性定义。类还拥有可执行的函数。

类之间的关联是业务设计的重要元素，因为它们确定了业务规则。关联在类图中显示为连接的线段。你应当在适当的地方用名称和关系的重复度标识关联。你应当小心地确认特殊的关联，例如聚集、组合、概括和自反关联。

设计者还需要确定应用需要的主要事件或触发器。有三种类型的事件：业务事件、数据改变事件和用户事件。事件可以用包含条件和操作的触发器来描述。复杂的事件串可以用顺序图或状态图来表示。

设计通常要经过若干阶段的修正，每个阶段都更加详细和精确。一种有效的途径是从一张大的蓝图开始，确保你的设计符合系统所需的主要部分。包的定义可以为元素分组以隐藏细节。随后细节被添加到系统类图中的每个包的支持图里。

模型和设计对于大型项目尤其有用。模型为设计者、程序员和用户提供了沟通的机制。CASE工具有助于创建、修改和共享设计模型。除类图以外，CASE的存储器将维护建立最终应用所需的所有定义、描述和注释。

开发漫谈

与任何开发者一样，Miranda需要一种方法来记录系统的目标和细节。可行性研究论证了目标并提供了关于成本和效益的粗略估计。类图确定了主要实体及其如何关联。类图和数据字典中的记录一起标明业务规则。对于你自己的类项目，应当学习这个例子。然后建立可行性研究和一张初始类图。

关键词

聚集（运算）
关联
关联角色
封装
实体
类
类图
类层次
协作图
组合
数据规范化
数据类型
派生类
快速应用开发
概括
继承
方法
重复度
多重关联
空值
多态性
主码
属性
二进制大对象（BLOB）
自反关联
关系数据库
关系
表
统一建模语言

复习题

1. 业务规则在类图中是如何表示的？
2. 类图（或实体关系图）的作用是什么？
3. 什么是自反关联，它在类图中如何表示？
4. 什么是主码？
5. 什么是重复度，它在类图中如何表示？

6. 商业应用中用到的主要数据类型有哪些?
7. 继承在实体关系图中是如何表示的?
8. 事件和触发器是如何与对象或实体相关联的?
9. 大型项目中的复杂问题有哪些?
10. 计算机辅助软件工程工具在大型项目中如何应用?

练习

1. 一家小型游艇租赁公司需要一个数据库来记录关于租赁和游艇的基本信息。最后，公司希望能够确定给游艇造成损坏的客户，但目前，经理只希望记录成本。经理已经以表单的形式描述了数据。为此例创建一个类图。

Rental ID	Canoe Rental	Rent Date			
Customer Last Name, First Name Email Phone Address City, State PostalCode Country		Credit Card Number Expiration Date Name on Card Deposit Amount			
Number	Description: Length, Material	Returned Date	Fee	Damage Charge	Total
					Total

2. 一家小型的宠物饲养公司需要一个数据库来记录其雇员所做的工作。目前，公司使用与这里所示的表单相似的纸质表格。对于每个时间空档，雇员记录关于客户、宠物和完成任务的信息。所有任务都有一个基本费用，但雇员依据任务的难度和宠物的不同可以对数额进行修改。公司希望雇员记录有关宠物和特定任务的注释。为此例创建一个类图。

Employee Last Name First Name DateHired Specialty	Daily Grooming Record Date
Start Time Owner Name, Phone, Address Pet Name, Category, Gender Task Time Fee Comments	

3. 一位牙医经营一家小公司，包括他自己、三个卫生学者和一位接待员，他需要一个数据库记录预约计划。他拥有一个商业记账系统来处理付款和保险，但预约目前是记录在与这里所示类似的纸质表单上的。目前表单只记录患者姓名和要执行的主要程序（清洗、X光、补牙等等）。他希望添加更多的信息，列出所有计划好的程序，估算每项程序的费用，并评估保险责任范围。基本费用应该与程序存储在一起，但保险金额是高度可变的，它将由接待员根据记账系统的估算进行输入。系统还需要记录患者的电话号码和电子邮件地址，并记录他或她希望以何种方式被通知。最后，它能够向患者发送电子邮件消息作为提醒。为此例创建一个类图。

Date Time	Dentist	Hygienist: Mary	Hygienist: Susan	Hygienist: David
10:00	Patient Name Procedure Fee	Patient Name Procedure		
10:30			Patient Name Procedure	
11:00				

4. 一家老字号制鞋公司因为市场竞争激烈，决定放弃生产低成本大市场的鞋子。它希望客户根据自己的要求定制鞋子。客户描述或测量他们的脚，然后公司可以为该客户建立特定的模型。客户脚的信息将被数字化并存储在数据库中，以便当用户订购鞋子的时候可以获取使用。公司将会生产多种标准样式的鞋子。客户可以选择一种样式、选取一种颜色和尺寸，并选择一种材料。订单被送往公司的生产厂，为每个人定制鞋子。对大多数鞋，客户还可以指定轻微的调整——例如因为某种原因要求稍微宽松一些的鞋子。基本信息依据示例表单输入，但公司需要在数据库中存储所有的信息。为此例创建一个规范化的表格。

OrderID　　Order Date Estimated Ship Date　　Actual Ship Date　　Shipping Cost
Customer Last Name, First Name, Phone, Email Address, City, State, PostalCode Country LeftFootID, RightFoodID, Comments
Shipping Address City, State, PostalCode Country
StyleID　Description　Color　Material　Size　Adjustments　Price

5. 实践练习：与本地一家商店的经理交流，为商店的系统创建一张类图。

6. 指出下列实体之间的典型关系。写下任何影响你决定的假设和注释。确保包含最小值和最大值。

   a. 书店，书

   b. 货车，司机

   c. 计算机，IP地址

   d. 患者，药瓶

   e. 房屋，厨房

   f. 游艇，水池

   g. 游艇，码头

   h. 银行，联邦储备区

   i. 公司，主管

   j. 电视节目，时间空档

   k. 公司，NAICS6位数字

   l. 检查账户号码，客户

   m. 汽车，车辆识别号码（VIN）

   n. 捕鱼许可证，渔场

   o. 驾驶证，司机

   p. 飞机票，乘客

   q. 处方，患者

   r. 经纬点，道路

7. 对于下面列表（左侧）中的每一个实体，指出右边的每一项是否应当作为左侧实体的属性还是一个独立的实体。

   a. 鞋　　尺寸、销售日期、颜色、价格、销售员、样式、生产厂

   b. 汽车　　模型、拥有者、生产厂、销售人员、引擎、司机

   c. 工厂　　机器、城市、雇员、经理、所有者、尺寸

   d.监狱　　警卫、囚犯、典狱官、地点、州/联邦、容量

   e.宴会　　地点、场合、嘉宾、菜单、开始时间、备办宴会者、乐队

8. 你所在大学的广播站遇到一个问题。要广播歌曲——尤其是以流媒体方式向互联网广播，广播站需要记录哪些歌播放过的详细日志。广播站还希望了解多少听众接收了这些歌。对于互联网流媒体来说，这个数字可由流媒体软件显示。尽管广播站想要关于每首歌的详细信息，但DJ一般更容易通过作者和CD来点歌。广播站想要一个易用的数据库，以便DJ迅速挑选歌曲。每次播放时，输入听众的数量，以及任何评注——尤其因为某种特殊原因这首歌没能播完。根据你对音乐的了解和示例日志页，为此例创建一个类图。

Date Time	Song	Employee Artist CD	Studio	Comments	#Listeners

9. 一家当地公司正在创建一种新式车内收费系统。客户要购买一个发射器，它是安装在车内的一个小型发射机。他们将向公司注册一个信用卡号码。当客户驾车通过收费点时，发射器发出的ID信号被收费站接收。时间和费用会被记录，客户在月底将收到一个账单，当月的总金额将自动从信用卡中划走。稍微复杂的部分在于每个站点依据星期几和一天中的不同时间收取不同的费用。例如，高峰时段的收费一般比其他时段高。下面的报表提供了费用结构和在一个单独站点需要采集的数据的部分图示。当然，会有很多站点，客户的数据需要累计从而提供一个总金额。为减少偷窃，发射器由特定汽车标识，但当客户售车时可以进行转移。为此例创建一个类图。

Location Description Traffic Log Date			Fee Structure			
Time	EmitterID	Fee	DayOfWeek	Start Time	End Time	Fee
			MTWTF	6:00	8:00	$3
			MTWT	15:00	18:00	$3
			F	15:00	18:00	$5
			SS	00:00	24:00	$2

Sally的宠物商店

10. 对零售和宠物商店做一些初步的调研。确定你期望从Sally的宠物商店事务处理系统中能获取的主要效益。估算设计和建立数据库应用所需的时间和成本。
11. 通过添加记录所有动物谱系所需的细节来扩充Sally的宠物商店的类图。
12. 通过添加记录动物健康状况和兽医记录所需的细节来扩充Sally的宠物商店的类图。
13. 通过添加宠物饲养计划来扩充Sally的宠物商店的类图。

Rolling Thunder自行车

14. 使用面向对象方法重新设计Rolling Thunder的类图。确定所有的类、属性和方法。
15. Rolling Thunder自行车公司在考虑开设连锁店。说明数据库应如何修改以适应这种变化。向类图中添加新计划的组件。
16. 如果Rolling Thunder自行车公司希望建立一个网站来通过网上销售自行车，需要收集什么额外的数据？扩充类图来处理这些额外的数据。

参考网站

网站	描述
http://www.rational.com/uml/	UML文档和示例的主要站点
http://www.iconixsw.com	UML文档和评论
http://otn.oracle.com/docs/products/oracle9i/doc_library/release2/server.920/a96540/sql_elements2a.htm#54201	Oracle数据类型描述
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odbc/htm/odappdpr_2.asp	SQL Server数据类型描述
http://time-post.com/dbdesign	数据库设计系统

补充读物

Codd, E. F. "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM* 13 no. 6 (1970), pp. 377–87. [The paper that initially described the relational model.]

Constantine, L. "Under Pressure." *Software Development*, October 1995, pp. 111–12. [The importance of design.]

———. "Re: Architecture." *Software Development*, January 1996, pp. 87–88. [Update on a design competition.]

McConnell, S. *Rapid Development: Taming Wild Software Schedules.* Redmond: Microsoft Press, 1996. [An excellent introduction to building systems, with lots of details and examples.]

Penker, M., and H. Eriksson. *Business Modeling with UML: Business Patterns at Work.* New York: John Wiley & Sons, 2000. [Detailed application of UML to business applications.]

Silverston, L. *The Data Model Resource Book, Vols. 1 and 2,* New York: John Wiley & Sons, 2001. [A collection of sample models for a variety of businesses.]

附录：数据库设计系统

很多学生发现数据库设计很难学。基本概念似乎很简单：定义一张表代表一个基本实体，表的列描述实体的属性，存储必需的数据。例如，Customer表将有CustomerID、LastName、FirstName等列。但通常很难决定一张表具有哪些列。主码列用于建立表之间的关系，确定主码列通常也是很困难的。设计的复杂性源于表反映了底层的业务规则，因此学生还必须理解业务流程和限制，以便能创建一个满足业务需求的设计。

除了仔细阅读第2章和第3章之外，学习数据库设计最重要的手段之一在于做尽可能多的

练习。而学生也需要反馈他们在设计中存在的问题以改进设计。一个面向教师和学生的联机专家系统可以提供即时的反馈信息。这个联机系统的网址是：http://time-post.com/dbdesign。本附录使用这个DB设计系统来突出可视化数据库设计的优势。然而，即使你不使用这个DB设计系统，本附录仍然提供一种有效进行数据库设计的方法。

1 示例问题：客户订单

通过一个示例最容易理解数据库设计和DB设计系统。客户订单在商业数据库中很常见，考虑图2-1A所示的订单示例。表单大致提供了关于业务规则和惯例的信息。例如，订单上只有填写一位客户姓名的空间，因此看上去一个订单只能有一位客户参与。相反，订单中的多行则表明允许一次订购多项货物。这些一对多的关系在数据库设计中是很重要的要素。

典型的订购表单

Order Form

Order # **Date**

Customer
FirstName LastName
Address
City, State, ZIP

Item	Description	List Price	Quantity	COH	Value
				Order Total	

图2-1A 典型的订购表单。每张订单只能由一位客户提交，但是可以包含多个订购项，如重复部分所示

2 开始：确定列

创建数据库设计的初始步骤之一要确定要收集数据的所有属性或项。在此例中，需要存储客户的姓、名、地址等等。还需要存储一个订单号、订购日期、项描述等等。基本上要确认表单上的每一项并赋予一个惟一的名称。注意，有些项可以很容易地计算出来，因而不必存储。例如，值可由价格乘以数量得到，订单总额是各值项的总和。

如图2-2A所示，在你打开一个问题之后，DB设计系统为你提供了表单中项的列表。这个列表显示在右边的列中。这些列是数据库设计的基础。你的工作是创建表，选取属于每个表的列。你可以右击列名，选择Rename选项来重命名列。你要确保名字是惟一的，如果列名相同的话，系统不会报错，但你在从列表中选取列时会出现问题。要更好地查看可用的列，你可以右击列表并选择Sort选项对列表进行排序。

3 创建表并添加列

主要的目标是创建表并指出每个表具有哪些列。很明显这个问题将需要一张表来存储客户数据，因此，右击主绘制窗体并选择选项添加一张表。然后，在表的顶部灰颜色的边上右击

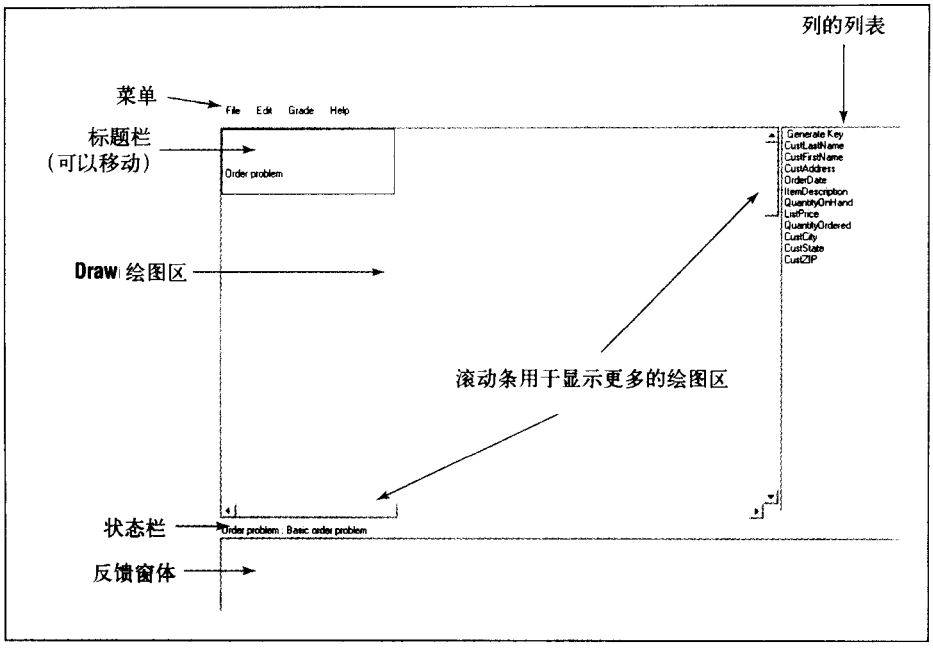


图2-2A DB设计界面。当你登录后，使用File菜单中的选项来打开Order Problem。Help菜单中有一个选项来查看问题。右手边的窗体中包含一个可用列的列表，它们将被放入表中。选择Grade菜单选项可以在反馈窗体中生成评注

表，选择重命名选项。输入“Customer”提供一个新名字。

每张表都必须有一个主码——一个或多个列能惟一标识表中的每一行。查看订单表单及列表中的列，你会发现没有一个列可用作主码。你可能会考虑使用电话号码，但当客户改变电话号码时会出现问题。因此，最好的办法是生成一个名为CustomerID的新列。这个数字的值可由市场部给出或由数据库系统自动生成。要创建一个新码，将Generate Key项从列表中拖放到Customer表中。然后，右击列名，选择重命名选项，输入CustomerID作为新的名称。注意，现在CustomerID将会显示在Customer表中并作为列表中的一个新列。此外，注意图2-3A中，CustomerID标有#号，表明它是Customer表主码的一部分。通过右击列你可以将列设置为主码或取消设置为主码。当你双击一张表的列名时，表会自动缩放以显示它所有的列。

现在表和主码都已建立，你可以向表中添加其他的列。但添加哪些列呢？Customer表应当包含那些能标明某一特定客户属性的列。因此，查找每个能由新的主码CustomerID惟一标识的列，将其拖放到Customer表中。

很明显这个数据库设计需要一张Order表。添加一张新表，命名为“Orders”，并生成一个主码OrderID。同样，你需要选取属于Order表的列。查看订单，你会添加OrderDate列。注意，订单也包含客户的信息。但要求职员为每张订单输入客户的姓名和地址是一种浪费。因此，你只需要将CustomerID添加到Order表中。记住，CustomerID不是Order表的主码，因为每张订单只能有一位客户。如果将它设为主码，那么会表明一个订单可以有多位客户参与。你的表应当与图2-4A相似。在你离开以前一定要保存你的工作。如果时间过长，互联网连接将会超时，你也将丢失你所做的修改。

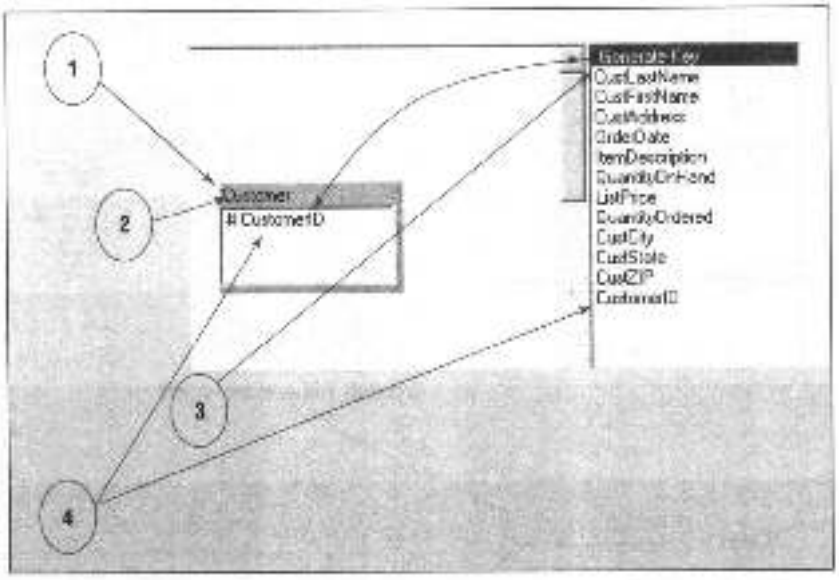


图2-3A 添加一个表和码。(1) 右击并选择Add表。(2) 右击临时名称，选择Rename，输入新名称。(3) 将Generate Key拖放到表上。(4) 右击列名并重命名

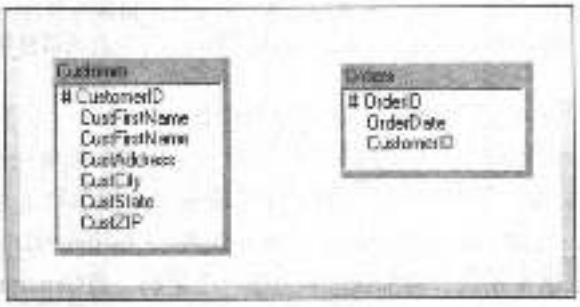


图2-4A 两张表，每张表代表了一个单独的实体，所有列都是为该实体收集的数据。Orders表包含CustomerID，它提供了一种获取Customer表中对应数据的方法

4. 关系：连接表

数据库系统最终需要知道，Order表中的CustomerID列和Customer表中的CustomerID列构成的关系。需要在图中绘制一条线段，连接两个表以显示这种关系。创建连接最简单的办法是将CustomerID列从Customer表中拖放到Order表中的CustomerID列。第二步是指出关系每一边的最小和最大值。图2-5A表明了关系的选项在设计视图中是如何显示的。在此例中，一个订单只能有一位客户，因此，客户的最小值和最大值都是1。从关系的另一方面看，每位客户可以提交零到多个订单。有些人可能争论说，如果一位客户没有提交任何订单，那么他或她只是一个潜在的客户，但这个区别对于数据库并不重要。

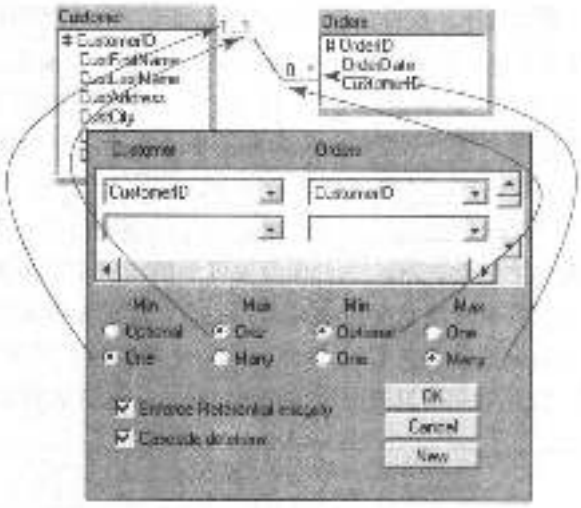


图2-5A 关系，将CustomerID列从Customer表拖放到Order表中的CustomerID列上，然后为关系的每一边设置最小值和最大值。一个订单只能涉及一位客户，而一位客户可以提交零到多个订单

5 评判：检测并解决问题

你将会重复这些步骤来创建数据库设计：添加表、设置主码、添加数据列、连接表。DB设计系统使这些流程变得相对容易，你可以将表拖拽到周围以更好地显示它，你可以保存你的工作，等一会回来再打开它继续解决问题。但是，你仍然不知道你的设计是好是坏。

考虑向订单示例问题添加另一张表。为Items添加一张表并生成一个新主码叫做ItemID，添加ItemDescription、ListPrice和QuantityOnHand列。现在你遇到的问题是将这张新表与Order表连接起来。但是，到目前为止，它们没有任何相关的列。因此，作为一种实验，尝试向Items表中添加OrderID列。现在连接OrderID列，建立从Items表到Orders表的连接，如图2-6A所示。

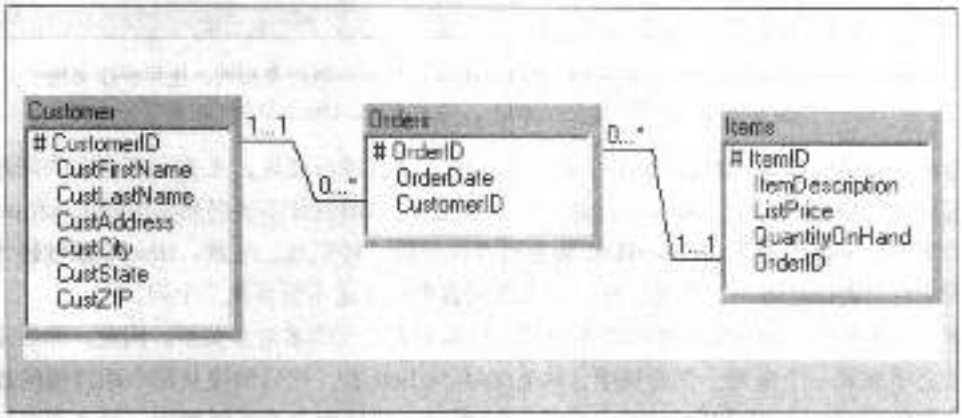


图2-6A 创建错误，要描述一个可能的问题，向Item表中添加OrderID列，然后将它与Orders表进行连接

你随时可以提交当前的设计，查看是否存在问题。事实上，在你创建表的时候最好进行多次检查，以便能尽早发现问题。使用菜单中的Grade选项，这个选项会在窗体底部生成一个评注列表。Grade to HTML选项生成相同的列表，以表格方式显示在单独的窗体中。

如图2-7A所示，当你评判这个问题时，你得到一个合理的高分（88.3）。然而，也有几条重要的评注。当你双击一条评注，系统高亮显示这个错误。在此例中，大部分评注指出Items表存在问题。特别地，OrderID列显示有一个问题。这个问题通过关系半高亮显示。一个订单是否能有多个项？如果你重新检查图2-1A，你会发现是的，一个订单具有多行可以表示多个项。你可以尝试修改这个关系，将它变为多对多的关系，但关系数据库并不直接支持两个表之间的多对多关系。如果你更仔细查看Items表，你会发现问题。目前只有ItemID作为主码，因为OrderID不是主码，因此每一项只能出现在一张订单上，这意味着每一项只能售出一次。

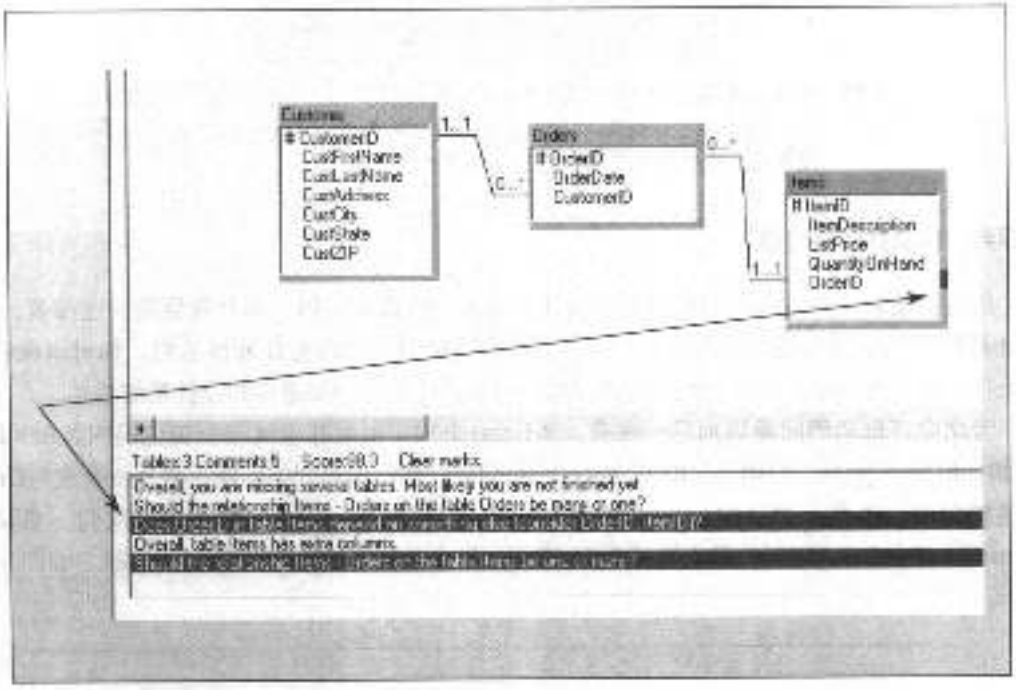


图2-7A　为练习评分，双击一条评注以突出显示引发问题的表和列。使用评注来指出引发问题的表和列。在此例中，考虑为什么OrderID列不应属于Item表

你可以尝试将OrderID与ItemID一起作为主码，试着进行修正，来看一下存在的问题。图2-8A显示这些修改的结果。首先，注意，分数降低了！DB设计系统仍然指出Orders和Items之间的关系存在问题，显示将OrderID作为主码存在问题。特别地，注意，ItemID被创建为生成码，因此它总能保证惟一。如果这样，那么在此表中你永远不需要第二个列。

解决方案是要认识到关系数据库不直接支持两个表之间的多对多关系。因此，你必须在这两个表之间插入一个新表。在此例中，称它为OrderItem表。然后确保从两个相连接的表中添加两个主码列（OrderID和ItemID）。如图2-9A所示，将这两个关系添加为一对多的连接，注意，这里Items到OrderItems的关系指出有些项可能还未订购过。

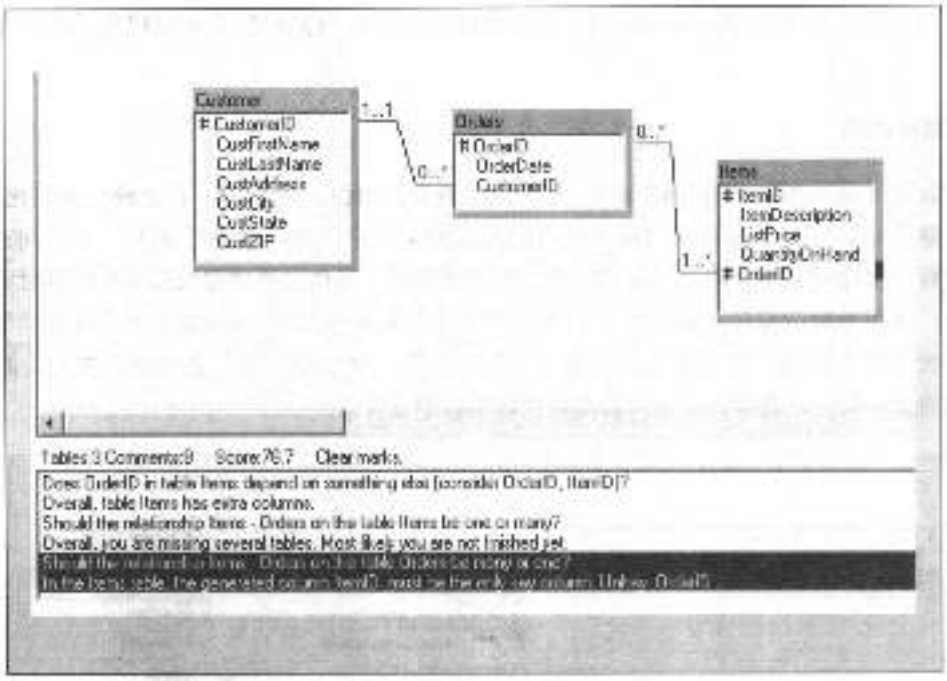


图2-8A 试着改正问题。注意，分数降低了，因此“改正”实际上使情况变得更糟。OrderID和关系中存在的问题并没解决

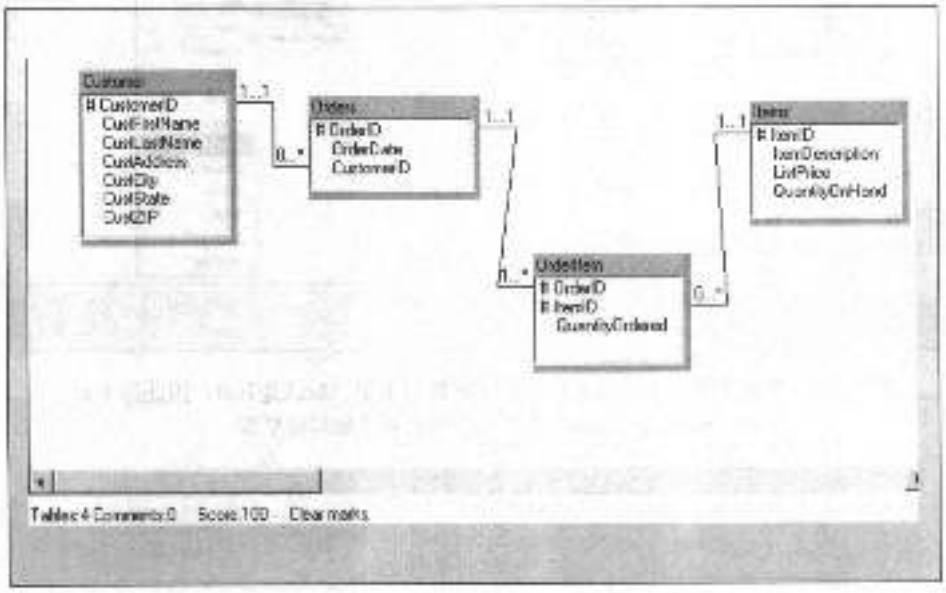


图2-9A 一种解决方法。添加中间表OrderItem并包含两张表的码（OrderID和ItemID），利用一对多关系与两张表连接

从分数可以看出，这个四表的解决方案对于这个典型的订单问题来说是最好的数据库设计方案。Customer表存储关于每位客户的数据，Items表包含描述每个销售项的行，Orders表提供订单号、日期和一个到提交订单客户的连接，OrderItem表表示订单中重复的部分，列出

每个订单中购买的多个项。你应该确认初始表单中的所有数据项都至少出现在最后的某一个表中。

6 指定数据类型

在数据库设计完成之前需要执行一个额外的步骤。最终，这个设计将会转化为数据库表。当你创建表时，需要了解存储到每一列中的数据的类型。例如，名称是文本，主码列通常是32位整数。确保所有的日期和时间都赋予Date数据类型。当你需要浮点数而不是整数时需要小心核对：使用单精度或是双精度取决于最大数值将会是多少。图2-10A显示设置数据类型的方法：在表中右击列名，移动光标选择当前的数据类型，然后移动鼠标选择新的类型。默认值是常用的文本类型。这样，很多列例如客户姓名将不必修改。

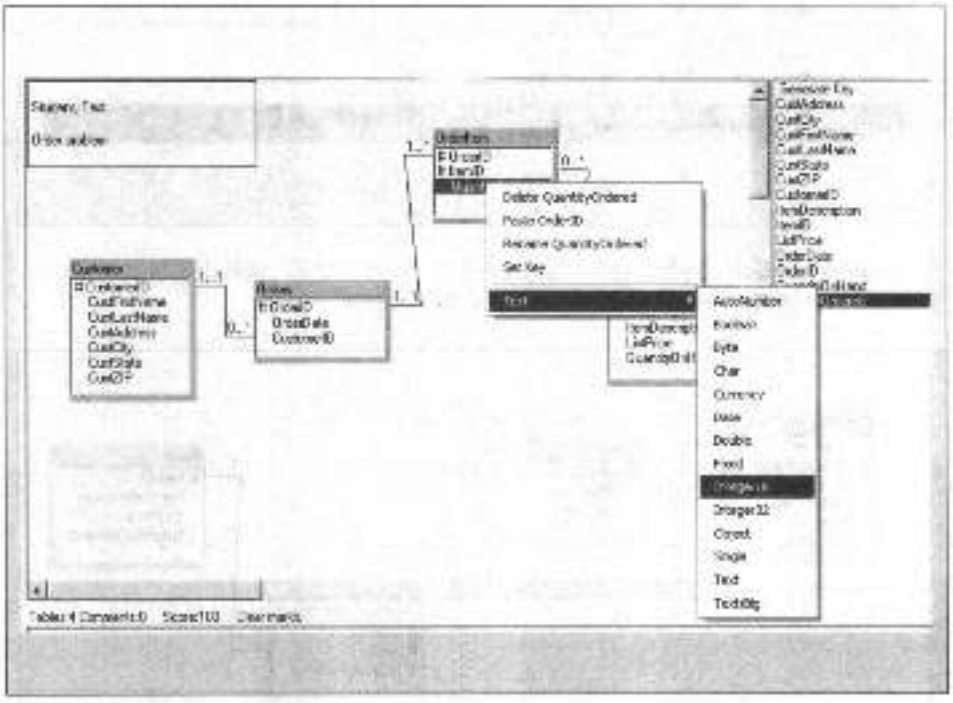


图2-10A　数据类型。右击列名然后设置其数据类型。默认是Text，因此你不必修改一般的列，例如客户姓名。码列通常是32位整型

第 3 章

数据规范化

本章学习内容

- 为什么重复组需要放到单独的表中?
- 规范化的三条主要规则是什么?
- 当表中存在隐式依赖时会产生什么问题?
- 为了保证数据完整性，DBMS要强制实施什么类型的规则?
- DBMS如何保持多个表中的数据一致性?
- 如何将一个类图转化为规范的表?
- 如何估计数据库的大小?

3.1 开发漫谈

Miranda： 真有意思，我对公司的流程和规则有了很多了解。我想我已经把每一件事情都完全记录在类图里了，并且在数据字典中作了一些记录。

Ariel： 真棒！我们今天晚上应该去听音乐会庆祝一下。

Miranda： 我可以放松一个晚上了。或许让我的脑细胞休息一下能让我搞清楚下一步该做什么。

Ariel： 你的意思是什么？你觉得这个项目要花费多长时间?

Miranda： 正是这个问题。我这一段的所有时间都花进去了，但事实上我还一点都没有开始做这个项目。

Ariel： 唔，数据难道不是构建一个数据库应用程序最重要的方面吗？我听说数据库系统挺难对付的。你必须第一次就正确地定义数据；否则，你将不得不重新开始。

Miranda： 或许你是对的。我计划用这个晚上来放松一下，然后我要研究一下这些规则，看看怎样将类图转化为一组数据库表。

3.2 简介

数据库是一个强有力的工具，相比传统的编程和层次文件具有很多优点。然而，只有正确地进行数据库设计，才能获得这些优点。回忆一下，数据库是表的集合。本章的目标是教会你如何为数据库设计表。

数据规范化的本质在于将数据划分到多个表中，依据表中的数据，这些表相互连接。机械地说，这个过程不是很难。大致需要学习四条规则。另一方面，表是为处理的特殊业务或应用而设计的。因而，必须首先了解业务，表必须符合业务规则。因此，设计一个数据库的挑战在于必须首先理解业务如何运转，它的规则是什么。这些规则中的一部分在第2章有所涉及，集中体现在关系上。业务关系（一对一和一对多）组成了数据规范化的基础。这些关系对于如何建立数据库起至关重要的作用。这些规则因公司的不同而变化，有时甚至取决于企业中与你进行交流的那个人。因此，当你创建数据库时，需要建立一张公司如何运转的图景。与多个人交流来理解数据之间的关系。数据规范化的目标在于确定业务规则，以便能设计出合理的数据库表。

要仔细设计数据库表，需要（1）节约空间，（2）最小化重复，（3）保护数据以确保其一致性，（4）通过传送少量数据提供快速事务。定义数据库表的方法之一是使用第2章介绍的绘图方法建立一个类图。另外一种方法是收集基本的文书，从你可能会用到的每张表单和每个报表开始。然后将各个数据集合拆开，分解到各个表中。大部分人发现将这两种方法结合是最有效的。但本章将从分别描述这两种方法开始。

3.3 表、类和码

第2章关注确定业务类和业务关联。现在，需要更仔细地定义这些类，以便能将它们转换为数据库表。当然，当你修改表时，也会更新类图。类之间的关系对于最终表的形式起决定作用。这些关系也以表的主码方式呈现。主码由能惟一确定每一行的列的集合组成。由于码必须始终保证惟一，因此通常创建一个新列来存储生成的主码。但是，很多情况下，你会使用多个列来组成主码。这些情形十分重要，需要进行详细阐述。

3.3.1 复合码

在很多情况下，设计数据库的时候，会使用多个列作为表的一部分主码。这称为*复合码*。当某个表包含与另一个表存在一对多或多对多的关系时，需要使用复合码。

复合码的例子如图3-1中的OrderItems表所示。这两个表在业务中很常见，它们组成了主从或父子关系。Order表很简单。它只有一列作为主码，就是你所创建的OrderID。这张表包含关于某一订单的基本信息，包括日期和客户。OrderItems表使用两列作为码：OrderID和Item。OrderItems表的目的是说明客户选择购买哪些商品。这样设计主码的关键之处在于每张订单可以包含多件不同的商品。在此例中，OrderID为8367的有三项。由于每张订单可以有很多不同的商品，因此，Item必须作为主码的一部分。从左向右阅读表的描述，可以看到每个OrderID

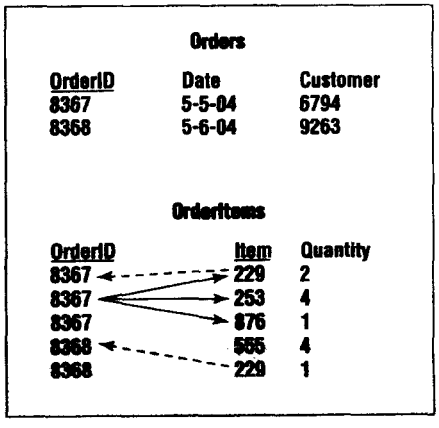


图3-1　复合码。OrderItems使用复合码（OrderID+Item），因为这里有一个多对多的关系。每个订单可以包含多个商品（由实箭头表示）。每个商品可以出现在多张不同的订单上（虚箭头）

可以有很多Items。“多”说明Item必须成为主码。OrderItems表的另一个方向又如何呢？是否真正需要OrderID作为主码？答案是肯定的，因为公司可以将相同的商品售给不同的人（或在不同的订单中售给相同的客户）。例如，Item为229出现在OrderID为8367和8368中。因为每一项可以出现在不同的订单中，因此，OrderID必须成为主码的一部分。作为对照，重新考虑图3-1中的Orders表。每个OrderID只能拥有一个Customer，因此，不必将Customer作为主码。

为了确保你理解主码和关系如何相互作用，再看一遍OrderItems表。看看ItemID列，问问你自己，对于每个OrderID，能有一个还是多个ItemID？如果回答是多个，那么ItemID必须设为主码（加下划线）。现在，看看OrderID列然后问自己，一个ItemID能否出现在一张或多张订单上？回答仍然是肯定的，因此，OrderID也必须设为主码。

看看Order表中的CustomerID列，然后发问：对于每张订单，能有一个或者多个客户吗？通常的业务规则回答说每张订单只能有一个客户，因此，CustomerID不是主码的一部分。另一方面，由于CustomerID是Customer表的主码，它就认为是Order表的外码。把它想象为一位外国要人访问一个不同的国家（表）。Order表中需要CustomerID列，因为后者作为一个指向Customer表中其他用户数据的连接。

要想适当地规范化数据并将其尽可能有效地进行存储，必须合理地确定主码。对于主码的选择取决于业务关系，企业内的术语以及公司内一对多和多对多的关系。

3.3.2 代理码

要确保现实世界中的数据总是生成惟一的码是很困难的。因此，通常需要数据库系统生成它自己的码值。这些代理码只在数据库内部使用，而且通常是隐含的，因此，用户甚至不知道它们的存在。例如，数据库系统会为每个客户赋予一个惟一的码，而职员将会通过常规的数据如姓名和地址查找客户。当业务码具有某种不确定性时，代理码就显得尤为有用。考虑你可能会遇到的这个问题：一个公司每两年改变一次它的产品号格式。如果你依赖于业务码，那么你必须相信它们总是一致的，并且从来不会重复。

当数据库变得很大时，代理码的使用很需要技巧。在有很多并发用户的情形下，创建惟一的数字变得更加困难。此外，在大型数据库中代理码会引发很多性能问题。例如，生成代理码的常用方法是找到现有的最大码值，然后增加它。如果两个用户同时要产生一个新码，将会发生什么呢？一个优秀的DBMS能自动处理这些问题。

Mircosoft Access使用自动标识这种数据类型来为主码列生成惟一的数字。类似地，SQL Server使用Identity数据类型。Oracle具有一个SEQUENCES命令来生成惟一的数字，但它与Microsoft的运行方式不同。作为一个程序员，当一个新行插入时，你生成并使用新值这个过程并不是自动的。这两种方法都有优缺点。Microsoft方法的最大困难在于当新的一行插入时有时很难获得新值。Oracle方法的缺点在于你必须保证所有用户和开发者在整个应用中都使用恰当的数字生成命令。此外，两种方法在转移数据——尤其是转移到其他数据库系统中时都可能引发问题。

3.3.3 标记

一张详细的类图能描述每张表，包含每个类中的所有属性并标记主码列。使用类图的优点

在于突出类之间的关联。此外，可视化的表示能使一些人更好地理解系统。图3-2显示了一张简单示例类图，但省略了属性。

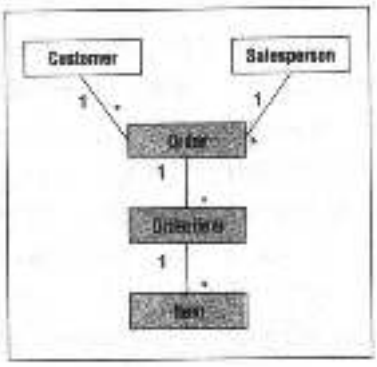


图3-2 一个基本订购系统的小规模类图。数字指示关系。例如，每位客户可以提交多个订单，但一张订单只能来自一位客户

类图的缺点在于它们可能会变得非常大。当有30个类时，将所有信息放在一页上变得很困难。而且，很多关联线段相互交叉，使得类图很难阅读。CASE工具有助于解决部分上述问题，使你只查看类图的一小部分。

然而，你也可以使用简短的标记，如图3-3所示。标记由一个表的列表组成。每一列由表名列出，主码加了下划线，一般首先列出。这种标记很容易手写或通过电脑输入，在一个小空间内它可以显示很多表。然而，显示表之间的关系非常困难。你可以在表之间画箭头，但这样会变得很凌乱。

```
Customer(CustomerID, Name, Address, City, Phone)
Salesperson(EmployeeID, Name, Commission, DateHired)
Order(OrderID, OrderDate, CustomerID, EmployeeID)
OrderItem(OrderID, ItemID, Quantity)
Item(ItemID, Description, ListPrice)
```

图3-3 表标记。从一个表的简单列表中更容易看出列的详细信息。这个列表还在把表输入到数据库中时起作用

设计者通常既创建类图也创建表的列表。列表确定所有的列和码。类图显示表之间的关系。类图还可以包含附加的详细信息，例如存在的限制和最小值要求。

考虑图3-4所示的小型客户记账系统。基本数据包括客户和助手。你可能会有一个关系显示分配给每个客户的助手。你还想记录助手为每位客户所做的工作量。Client和Partener表使用专门创建的列作为主码。这样你不必担心主码不惟一或可能存在重复姓名的问题。记住，客户和助手都不必知道他们的标识号。第8章将介绍如何在查找客户数据的同时隐藏主码值。

```
Client(ClientID, Name, Address, BusinessType)
Partner(PartnerID, Name, Speciality, Office, Phone)
PartnerAssignment(PartnerID, ClientID, DateAcquired)
Billing(ClientID, PartnerID, Date/Time, Item,
Description, Hours, AmountBilled)
```

图3-4 客户记账示例。注意，业务规则规定每位客户可以分配多个助手，因而PartnerID和ClientID都成为PartnerAssignment表的主码

注意，在PartnerAssignment表中，PartnerID和ClientID设为主码。仅仅通过将表写为这种形式，你就能提出关于这个公司如何运行的假设。首先，每位助手为多个客户执行任务——这很平常。此外，每位客户可以分配多个助手。这第二个假设在一些公司当中可能行不通。较小的公司可能只将一位主要助手分配给每个客户。如果一个不同的助手为某位客户完成任务，

那么这个客户仍能在记账表中列出。PartnerAssignment表中主码的选择取决于公司业务运行的方式。

图3-5描述了当公司规定每位客户只能分配一个主要助手时将会发生什么。在这种情况下，PartnerID不再作为PartnerAssignment表的主码。还要注意，Client和PartnerAssignment表具有相同的主码（ClientID）。如果这些主码是正确的，那么列应当合并到一张表中（Client）。没有理由存在具有相同主码的两张表。这样，第二个公司的数据表将会与第一个公司的不同——只因为它们的业务流程不同。

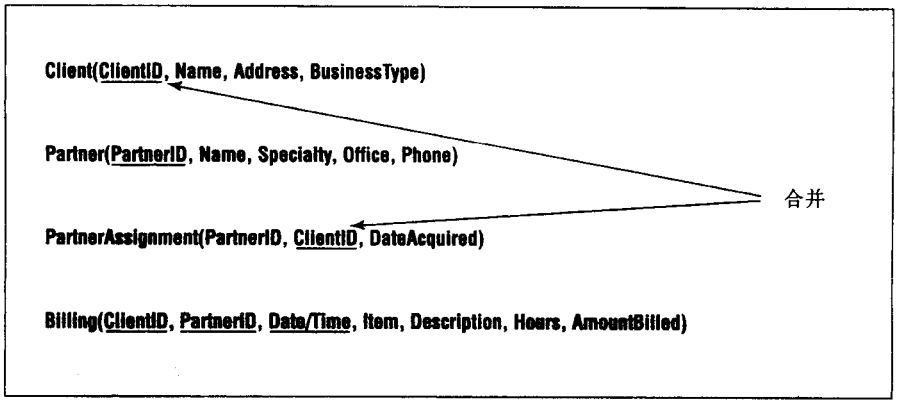


图3-5 简化的客户记账系统。如果每位客户只能分配一个主要助手，那么PartnerID就不应设为PartnerAssignment表中的主码。但是，现在Client表和PartnerAssignment表都有相同的主码（ClientID）。因此，两个表中的列应当合并到同一个表中（Client）

图3-6所示的Billing表具有三个主码列：ClientID，PartnerID和Date/Time。这样的主码表明对于每位客户，很多助手可以为他完成任务。反过来，每个助手能为很多不同的客户服务。同样，每位客户可以让每个助手在不同时间多次完成任务。如果你不将Date/Time设为主码，那样每位客户可以给很多助手付账，但每个助手只能有一次。尽管有了这种限制客户会非常高兴，但这不是使用此数据库的公司的实际观点。如果不将Date/Time设为主码，就会出现第1行和第3行不再惟一的问题。为了测试主码，键入示例数据并覆盖其他列。观察表中的前两列，你就会发现第1行和第3行的前两项是重复的。要解决这个问题，你不得不问，一个助手如何多次为一位客户服务？回答是任务必须在不同的时间完成。这样，Date/Time列就添加成为主码的一部分。

Billing

ClientID	PartnerID	Date/Time	Item	Description	Hours	AmountBilled
115	963	8-4-04 10:03	967	Stress analysis	2	$500
295	967	8-5-04 11:15	754	New Design	3	$750
115	963	8-8-04 09:30	967	Stress analysis	2.5	$650

图3-6 Billing表的示例数据。注意，Partner963可以为Client115多次完成相同的任务。因为Date/Time是主码的一部分

现在你已经可以看到业务规则如何影响数据库设计。主码的选取很大程度上取决于业务关系，而在每个企业都可能不同。要小心地双重检查你设置的所有主码。如果在设置主码上出错，将很难使数据库的其他部分正确。在简单实体（客户、雇员等等）的情形下，你通常需要创建惟一的主码。对于更复杂的实体，你需要观察一对多和多对多的关系。你可以这样检查复合码：看着第一个加下划线的列（ClientID），然后发问，对于所有其他加下划线的列（在此例中，PartnerID），存在很多这样的助手吗？如果存在，此列应当设为主码。如果只有一个条目，那么此列不应设为主码。确保也这样检查相反的关系（从PartnerID到ClientID）。

3.4　音像店的示例数据库

介绍数据规范化的最好方法是研究一个示例问题。记住，你获得的结果（创建的表）很大程度上取决于特定的例子和所做的假设。下面的例子用到了大部分学生都很熟悉的场景：音像店内的主要业务。

音像店最重要的功能是检查录像带以供租赁。一个检查界面示例如图3-7所示。

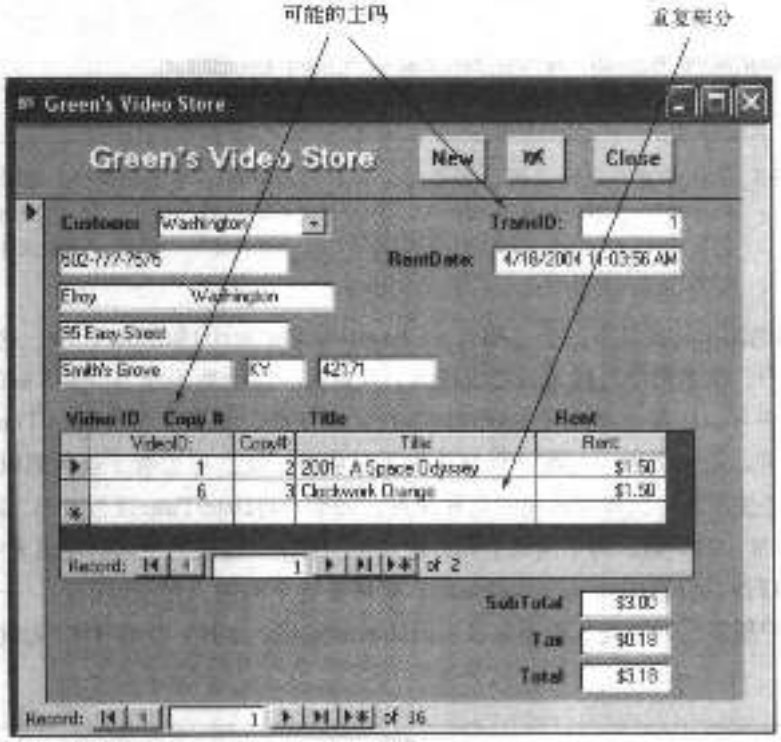


图3-7　录像带租赁界面。首先查找可能的主码，记住重复部分（一对多关系）将最终需要连接码

示例表单的主要组成部分是客户和出租的录像带。在数据库中建立表单后，会自动显示总金额。它还应当自动生成惟一的RentalID。注意，表单还有按钮和控件用于帮助用户更容易输入数据。表单和控件将在第5章讨论。目前，在你与经理交谈后，你应该获得表单要提供的功能，可以计算的值（如subtotals）应当标记，需要的话，应当提供适当的计算公式。在大多数

情况下，不希望将计算得到的数值存储到数据表中。

3.4.1 初始对象

从确定所需的主要对象开始。显而易见的是客户和录像带。在实际生活中还会有雇员，出租录像机，还可能销售其他东西，但现在你可以忽略那些东西。然而，经理还需要记录谁租了特定的录像带。例如，一盘录像带没有按时归还，经理需要知道该去找谁。这样，你还需要两个对象。第一个是事务对象，它记录日期和客户。第二个是那个客户当时租的录像带的列表。

检查图3-8所示的初始对象。你需要为Customer（和Video）设置主码。显然，Name不行，但你可能会考虑使用Phone号码。这种方法可能行，但它以后可能造成一些困难。例如，如果一个客户拥有了一个新的电话号码，就必须在与他相关的每个表中修改相应的号码。作为一个主码，它可能出现在多个不同的表中。当客户（Adams）搬家时，会引发更大的问题，电话公司在几个月后会将Adams不再用的电话号码重新分配给另一个人（Brown）。如果Brown在你的店里开账户，你的数据库很可能错误地将Brown当成Adam。最安全的方法是让数据库为每一个客户创建一个新号码。

初始对象	主码	示例属性	评注
Customer	Assign CustomerID	Name Address Phone	
Video	Assign MovieID	Title RentalPrice Rating Description	
RentalTransaction	Assign TransactionID	CustomerID Date	Event/Relationship
VideosRented	TransactionID + MovieID	VideoCopy#	Event/Repeating list

图3-8 音像店的初始对象。注意，事务包括两部分，RentalTransaction和VideosRented，因为一次可以租很多录像带

Video对象（可能包括视频游戏）也需要一个主码。实际当中你或许可以使用出版商提供的产品标识。目前，单独分配一个号码是最简单的。常用的属性包括Title、Rating、Description和RentalPrice。如果需要，将来可以添加更多的属性。

每个事务必须记录。一个事务是一个事件，它指出某位客户租某一盘录像带和租赁发生的时间。这个对象涉及到基本的租赁表单，并且分配了一个惟一的主码值。记住这种方法。几乎所有你遇到的问题都能以一张表解决，这张表存储基础表单或报表中的数据。

很多情况下重复部分的出现是一个很重要的问题，它可能在存储数据时引发问题（参见“重复部分的问题”）。这样，重复部分从主事务中分开，并存储在它自己的表中。这里的主码包括Transaction表的TransactionID列和VideoID列。注意，主码是复合码，因为存在多对多的

关系。一位客户一次可以租很多录像带，一盘录像带（在不同的时间）可以租给多位客户。

VideosRented表中的VideoCopy#表示租出的是电影的哪份拷贝（一部电影有多份拷贝）。当电影送还时，拷贝号告知经理是谁送还了录像带。注意，你可以选择把VideoCopy#设为主码。如果VideoCopy#不是主码（就像现在这样）相应的业务假设就是一位客户可以租许多电影，但只能租该电影的一份拷贝。以这种方式设计数据库，规定一位客户永远不能同时租同一部电影的两份拷贝。这是否合理的假设依赖于具体业务。如果你这样建立数据库，一个客户永远不能一次租某一电影的多份拷贝。如果一位客户想要租两个拷贝，你将不得不另写一个事务。

3.4.2　初始表单评估

如果没有经过实践，想要确定图3-9所示的四张表是很困难的。大部分人能识别Customer和Video表。一部分人会发现需要RentalTransaction表。然而，对VideosRented表的用途就不那么清楚了。幸运的是，有一种方法通过将整个表单分解得到每张表。这种方法就是数据规范化的方法。它是遵循业务假设的一种机械过程。

图3-9显示了初始表单评估的第一步。当你刚开始学习规范化时，应当很小心地进行这一步。熟练以后，可以选择跳过这一步。这个过程是追历表单或报表，记录所有你要存储的东西。目标是写成一种结构化的格式。

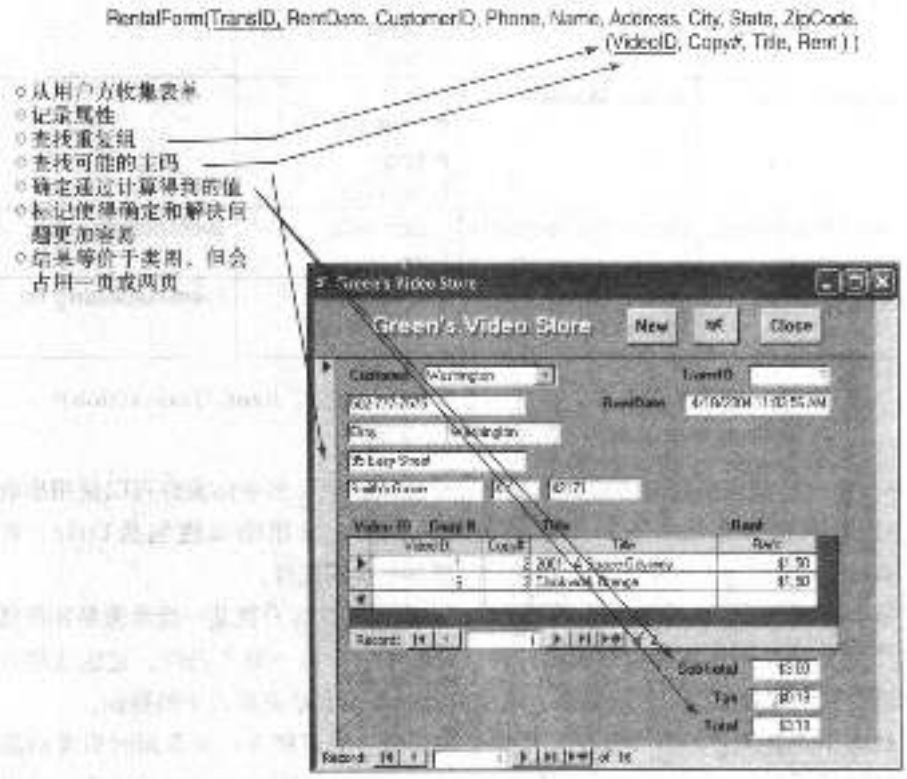


图3-9　初始表单评估。一旦收集完基本的用户表单，就可以将它们转化为更简洁的标记。这种标记通过强调可能的问题使规范化的步骤更简单

为表单起一个名称，然后列出所有的项作为列名。你可以从表单的左上角开始，逐个为每个数据元素写下列名。试着将项排列在一起，形成自然的组，例如所有的客户数据。RentalForm以TransID开始，看起来可以作为一个不错的主码。接下来列出RentDate和CustomerID，紧接着是基本的客户数据。下一步稍微有些复杂，因为你需要表示出包含录像带数据的重复部分。即，具有很多行数据或可能存在若干相似项。重复数据代表一对多的关系，需要小心处理。表示重复部分的一个简单方法是将它放在另一组括号中。有些人将它列在新行中。

注意，这里并不包含计算得到的属性（小计、税和总计），因为它们可以在需要的时候重新计算得到。然而，有些情况下，你可能希望存储计算得到的数据。将这些列与这个表单放在一起时一定要小心——不要将它放在重复数据部分中。另外，还应当标记这些项，或者在数据字典中做个标记，以便记住它们是通过计算得到的值。

在你进行第一步时，一定要记下你想要在数据库中存储的每一项。另外，确信标识了每一个重复的部分。这里你要多加小心。有时候重复部分是很明显的：它们可能在一个单独的区域，用不同颜色突出显示，或者包含示例数据使你能看见重复。而另一些时候，重复部分就不那么明显了。例如，在较大的表单中，重复部分可能出现在单独的数页里。其他时候，一些条目看上去似乎并不重复。例如电话号码：在一个商业环境中，有些客户只有一个电话号码，但另一些可能有好几个：办事处、办公室、手机、寻呼机等等。如果你需要存储多个电话号码，那么它们就成为重复的条目，需要进行标记。例如，电话号码可以存储为(PhoneType, Number)。在这种时候你还应当试着标出可能的主码，为了指出重复的部分，同时也突出那些你知道将会包含惟一数据的列。

3.4.3 重复部分的问题

你必须小心指明重复部分或一对多关系的原因在于它们在数据库中可能引发问题。图3-10说明了当你试着按照现在列出的格式存储表单中的数据时会产生的问题。这种初始格式存在的第一个问题是包含重复数据。例如，客户每租一盘录像带，职员都必须重新输入地址、电话等等，这是因为重复数据与基本数据放在同一张表中。因此，每盘出租的录像带都需要所有基本数据的一份拷贝。计算机可能速度很快并且有很大的内存，但反复列出客户是没有意义的。

当某人想成为音像店的会员时也会产生问题。由于他们还没有租任何录像带，因而，不能将他们的个人信息存入数据库。相反地，当你删除了所有的旧数据，例如所有去年的租赁记录时会发生什么？删除了租赁记录也就删除了用户数据。突然间，你会发现你删除了一半用户的基本数据。从技术的角度讲，这些问题就是所谓的*插入异常*和*删除异常*；即，当数据存储格式不合理时，添加或删除数据时会遇到困难。这些问题之所以会发生，是因为你想将所有的数据存储到一张表中。

重复部分的另一问题如图3-11所示，它与COBOL程序员曾经遇到过的问题相似。在使用以前的层次文档系统时，数据库设计者必须为每个重复部分分配一定量的空间。在音像店例子中，程序员将会为每张表单分配出租固定数量录像带的空间。估算所需的最大空间是一个困难。问题在于如果一两个客户要租很多录像带（比如说10盘或更多），程序将永远为每次租赁分配10盘录像带的存储空间。而这个空间在大多数事务中不会用到而浪费掉。另一方面，



图3-10 重复数据的问题。将重复数据与主表单存储在一起，造成重复部分的每条记录都要复制基本数据。在此例中，每此录像带租出时都必须输入客户数据——甚至是那些同时租出的

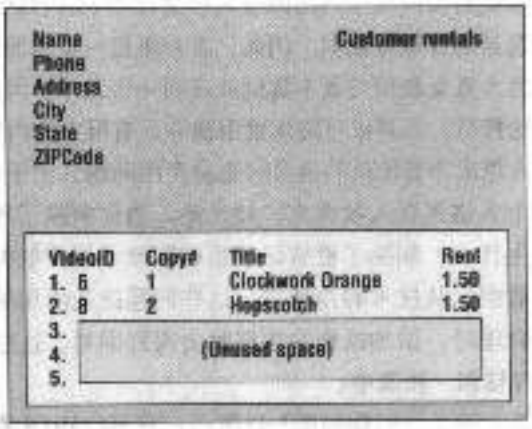


图3-11 重复数据的分配空间问题。将重复数据以层次格式存储一般强迫设计者为每条重复记录分配固定数量的空间。没有确定将会需要多少空间的好办法，因此数据库会有很多未使用的空间

不能留出足够大的空间将会引发问题，使最好的客户失望，再次考虑电话号码的问题。如果不把它们当成重复数据，那么需要多少列来存放不同的电话号码，又怎么能确定它够用？通过将重复数据移到单独的表，每个条目占一行，不必猜测需要多少行。当需要的时候，数据库会直接分配新的一行。

3.5 第一范式

重复部分问题的解决方法是将它们放到一个单独的表中。当一个表没有重复组，就称为满足第一范式，即，表中的每一个单元（由一行和一列确定）只能有一个值。这个值应当是原子的，意思是它不能分解为更小的部分。

3.5.1 重复组

在前面的一些例子中可以看到，有些重复组是显而易见的，而另外一些则很微妙，需要决定是不是将它们划分进单独的表更加困难。第一范式的规则很清楚：如果一个组的项是重复的，那么应该将其划分到一个新的表中。问题是，在一种情形下重复的项可能不适用于另一种情形。考虑电话号码的例子。在很多情况下，你可以简单地在客户表中为电话号码设置一到两列（当作不重复）。在另一个情况下，当有大量的客户并且电话号码集有很多种可能时，最好的解决办法是将电话码号划分到一个新表中。

回到音像店的例子，如图3-12所示，注意括号中标出的重复部分。要分割这个表单，首先分离出所有不在重复组中的东西。这些列以后可能会有其他改变，但它们不包含重复组。其次，将重复的录像带租赁部分的所有列放到一个新表中。但是，要小心。当你移出一个重复部分时，必须将原始表的主码也带上。RentalForm表具有TransID作为主码。这个主码，连同VideoID主码必须成为新表RentalLine的一部分。你需要将TransID作为主码，这样两个表中的数据以后才能合并到一起。注意，新表（RentalLine）总是会有复合码——用来标记租赁和录像带之间的多对多关系。

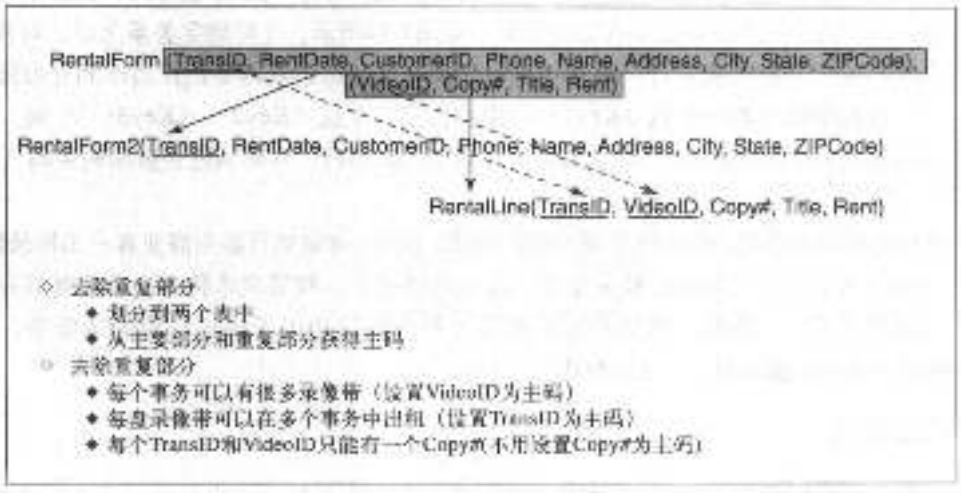


图3-12 第一范式。所有重复组必须划分到新表中，确保新表包含原始表主码的一份拷贝。存储重复组的表必须具有一个连接码，以便查询时数据可以合并

将重复组分开解决了很多基本问题。首先，减少了重复：不必再为租出的每盘录像带输入用户数据。另外，不必担心分配存储空间：每盘租出的录像带将存放在一个新行中。通过将所有客户的租赁数据存放在一张表中，避免了为每个客户分配空间的问题。

很多表单会有多个不同的重复组。如图3-13所示，如果它们的重复相互独立，那么划分是很直接的；每个重复组变成一张新表。只需小心地将原始表的主码包含进每张新表中，以便这些表将来可以相互连接到一起。使用基本的标记，只要括号不重叠，这些组就是相互独立的。例如，对于一个更复杂的音像店，可能拥有租出录像带的重复组，还可能有一个单独的部分列出某个客户的家庭成员。家庭成员的列表与租出的录像带的列表应当分别存储。

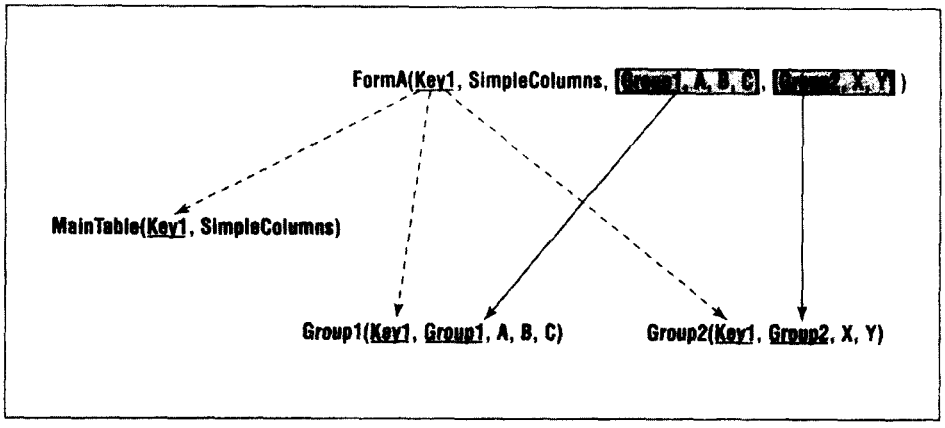


图3-13　独立的组。在此例中，两个组相互独立地重复。它们各自划分到新表中。记住在每张新表中包含主码（Key1）。

3.5.2　嵌套重复组

当一个表中出现许多不同的重复组，尤其一个重复组嵌套在另一个之中时，会产生更加复杂的情形。最困难之处在于确认嵌套组的性质。如图3-14所示，在你确定关系之后，对表进行划分是很直接的。只需一次进行一步，首先移出最外面的组。永远要记住当你划分每张表时带上主码。因此当你从第一个组（Key1…）中移出第二个组（Key2…（Key3…））时，新表TableA必须包含Key1和Key2。当你从TableA中移出Table3时，你必须连带前面的主码（Key1和Key2），并添加第三个主码（Key3）。

一家更复杂的音像店将遇到嵌套重复组的情况。例如，这家店可能向商业客户出租录像带，而客户内部的多个部门可能同时租录像带。在这种情况下，租赁表单将有一个重复部分表示部门（或家庭成员）。然后，嵌套的重复部分为每个部门列出了该部门租的录像带。例如Department（DepartmentID，…（VideoID，…））。

3.6　第二范式

满足第一范式的要求很简单：只需确定重复组，将它们放到自己的表中，并通过初始主码与主表相连接。下一步稍微有一点复杂，因为你需要检查表中主码值和其他列（非主码）之

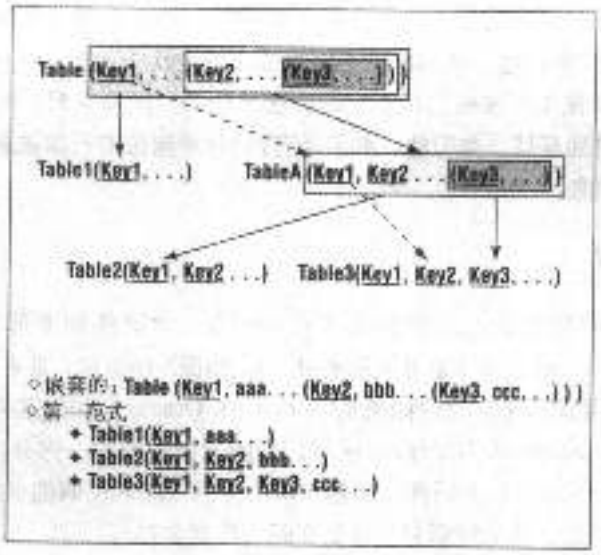


图3-14 嵌套重复组。组是嵌套的，当它们在另一个组之中（Key3在Key2中，Key2在Key1中），按步骤划分它们：从group1中移出所有group2，然后从group2中移出所有group3。注意，每张表将包含原始码（Key1）。由于有三层，最终的表（Table3）必须包含三个码列

间的关系。正确地指定主码是至关重要的。此时，应当双重检查所有主码以确保它们惟一并能正确地表示多对多的关系。

3.6.1 第一范式的问题

在图3-12中，从表的名称你可以猜测第一范式在有效存储数据方面可能仍存在问题。考虑图3-15所示的情形，它描述了当前的VideoRental表。每当有人租6号录像带时，数据库都会存

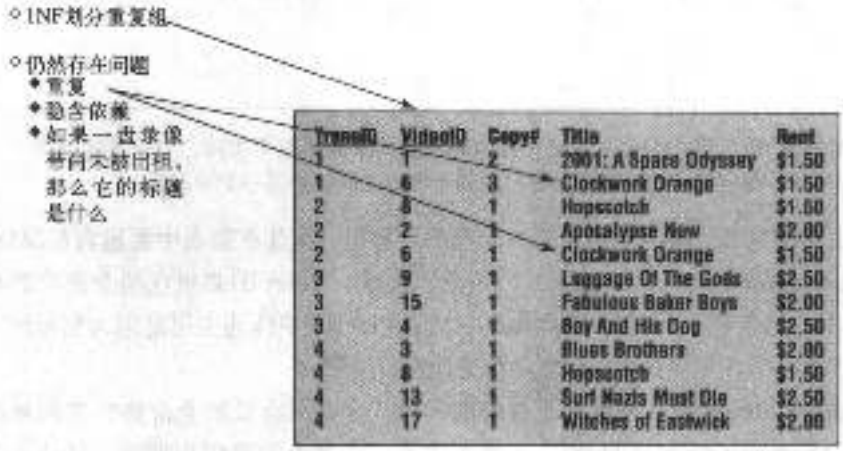


图3-15 第一范式的问题。不存在重复组，但VideoRental表仍然包含重复数据。每当一盘录像带被租出，我们必须重新输入它的标题。而且，如果一盘录像带尚未被出租，那么它的标题是什么？问题出在标题只依赖于VideoID，而不依赖于TransID

储标题Clockwork Orange。问题是，电影标题只依赖于一部分主码（VideoID）。如果你知道VideoID，就会知道相应的标题。电影标题不随事务而发生改变。除了浪费空间以外，还有另一个问题：如果一盘录像带尚未被出租，它的标题是什么？因为电影只随事务输入到数据库中，标题的数据不会单独存储。类似地，如果所有租8号录像带的行都被删除了，你将失去所有与那部电影相关的信息。

3.6.2 第二范式的定义

前面那个例子的问题在于一旦你知道了VideoID，你就能知道那部电影的标题。在VideoID和Title存在一对一的关系（也可能是多对一）。如图3-16所示，重点在于它与事务无关。如果某人在六月份租6号录像带，它的标题是Clockwork Orange。如果某人在十二月租该录像带，它的标题仍然是Clockwork Orange。这样，标题只由主码的一部分决定（VideoID而非TransID）。如果一张表的每个非主码列都依赖于整个主码（而非主码的一部分），则称此表满足*第二范式*（2NF）。注意，这个问题只针对复合码（具有多列）。

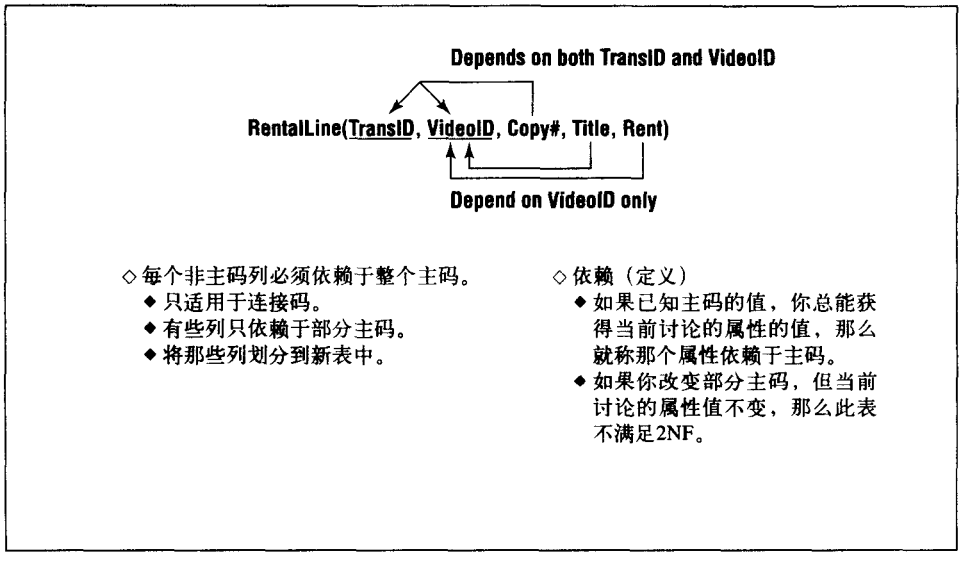


图3-16　第二范式的定义。每个非主码列必须依赖于整个主码。它只是连接码的问题。解决方法是将只依赖于部分主码的那部分划分出来

解决办法是划分表。将只依赖于部分主码的列移出。记住在新表中要包含那部分主码列。新表（VideosRented和Videos）如图3-17所示。注意，VideoID必须在两个表中都有。它在VideosRented表中表明每个人租了那些电影。它在Video表中作为主码是因为它是惟一的标识符。在两张表中包含此列使我们以后能将数据连接到一起。

在创建新的Video表时，遇到一个有趣的问题：在哪里存放租金价格？有两种选择：放在VideosRented表中或放在Video表中。答案取决于业务中的操作和规则。从技术上说，存在哪张表中都可以。然而，从业务的角度看有很大的区别。考虑将租金放到Videos表中的情况。这种模型表明只要你知道VideoID，你就能知道租金。换句话说，租金对于每部电影是

固定的。例如，新发布的电影可能会有更高的租金。现在来考虑租金存储在VideosRented表的情况。这很明显地说明租金依赖于VideoID和特定的事务。换句话说，对于某位客户，Clockwork Orange的租金可能是2.00美金，然而另一位客户可能只需支付1.50美金。价格不同是因为对租过很多电影的人给他打折，或者音像店每天的价格不同。大部分商业数据库的设计者都很快遇到哪里存放价格的问题。一种解决方案是将价格存储到两张表中。即，存储在Video（产品）表中的价格是标价。存储在VideosRented表中的价格是经过打折后实际应支付的租金。关键在于最终决定不只依赖于机械的规则，也有业务流程的因素。对业务流程的假设决定了你得到的表。现在，你可以保持简单一点的假设，为每盘录像带赋予固定的价格。

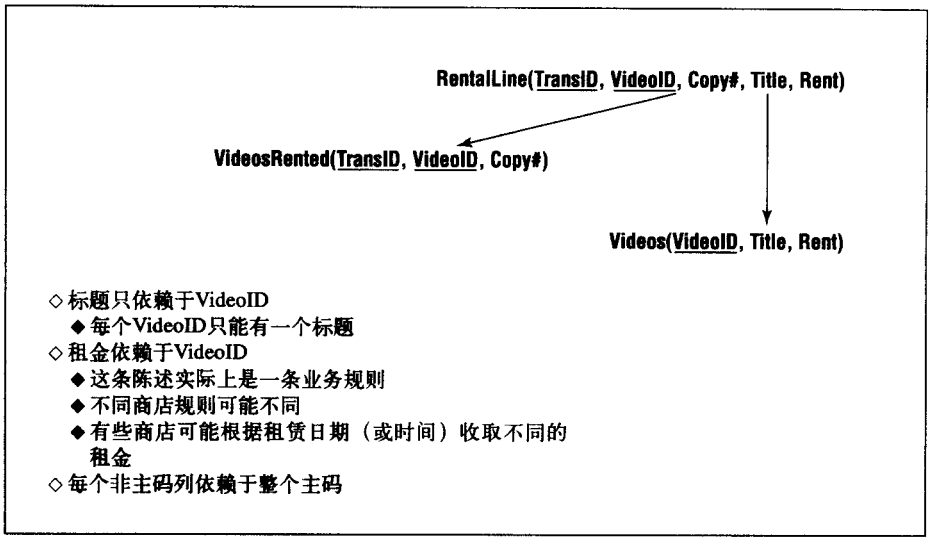


图3-17 创建第二范式。划分原始表使得只依赖于部分主码的项移到一张单独的表中。注意，两张表都必须包含VideoID主码

图3-18给出了新表的示例数据。注意，2NF解决了每次租赁重复记录电影标题的问题。电影的基本数据只在Videos表中存储一次。它在VideosRented表中通过VideoID列引用。浏览VideosRented表时，通过在Videos表中查找对应的ID，可以很容易获得相应的电影标题。第4章将介绍数据库查询系统如何自动处理这种连接。

3.6.3 依赖

在讨论2NF（和3NF）的时候使用术语依赖。即，属性Y依赖于属性X当且仅当每个X值能惟一确定一个Y值。在音像店的例子中，如果知道VideoID（6），就只有一个对应的电影标题（Clockwork Orange）。类似地，如果有一个CustomerID（3），也只有一个LastName（Washington）。

2NF从指出如果TransID改变电影标题仍保持不变开始引出讨论。由此，电影标题不依赖于TransID。这种依赖（或不依赖）对于大部分学生来说是很困难的。一旦你了解数据间的关系，规范化是自然而然的。问题在于在现实生活中确定那些关系。依赖是关于业务假设

和流程的问题。当你写出最终的规范化表（3NF或更高）时，你已经能明确地陈述那些业务关系。

VideosRented(TransID, VideoID, Copy#)

TransID	VideoID	Copy#
1	1	2
1	6	3
2	2	1
2	6	1
2	8	1
3	4	1
3	9	1
3	16	1
4	3	1
4	8	1
4	13	1
4	17	1

Videos(VideoID, Title, Rent)

VideoID	Title	Rent
1	2001: A Space Odyssey	$1.50
2	Apocalypse Now	$2.00
3	Blues Brothers	$2.00
4	Boy And His Dog	$2.50
5	Brother From Another Planet	$2.00
6	Clockwork Orange	$1.50
7	Gods Must Be Crazy	$2.00
8	Hopscotch	$1.50

（未改变）

RentalForm2(TransID, RentDate, CustomerID, Phone, Name, Address, City, State, ZIPCode)

图3-18　第二范式数据。电影标题现在只存储一次。其他表（VideosRented）可以只通过主码（VideoID）引用一部电影。这个主码提供到Videos表的连接。注意，RentalForm2表自动满足2NF，因为它不包含连接码

实际上，通常可以要求客户阐明属性之间的关系，但是，应该避免使用术语，诸如一对一和依赖，而是应当像这样提问：每一项能不能对应有多个条目？或者，不同的客户，收费会不同吗？然而，作为一个数据库设计者，你会发现你很少有时间向客户询问所有你想问的问题。你自己尽量确定一般的关系，将那些困难的问题留给客户。很多商业问题具有相似的规则和假设。经验可以节约你的时间，因为你不必要求用户说清楚每一条规则。

3.7　第三范式

设计第三范式（3NF）用到的逻辑、分析和元素与获取2NF相似。特别地，你仍然要关注依赖的问题。大部分有经验的设计者将推导2NF和3NF合并为一步。在技术上，满足3NF的表必须满足2NF。

3.7.1　第二范式的问题

这里，你需要察看RentalForm2表，它在前面的分析中一直被忽略。如图3-19所示。特别要注意，TransID是主码。通过示例数据可以看出问题。每当一个客户租一盘录像带，数据库都再次记录他或她的姓名、地址和电话号码，这些不必要的重复既浪费空间，也可能浪费职员录入数据的时间。考虑当一个客户迁居会发生什么。对该客户的每笔事务，你都不得不查找并更新地址。

如果客户尚未租任何电影，就没有地方存储客户的数据。类似地，如果从数据库中删除以前的事务，可能会丢失客户数据。再一次，你不得不处理隐含依赖。

RentalForm2(TransID, RentDate, CustomerID, Phone, Name, Address, City, State, ZIPCode)



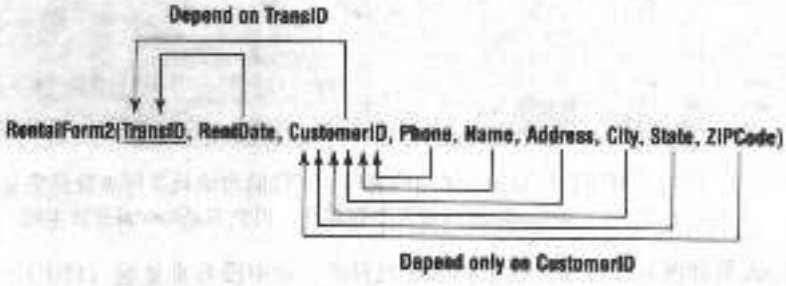
- 尽管满足2NF，仍然有问题：
  - 重复
  - 隐含依赖
  - 如果一位客户的未租借录像带，我们在哪里存储客户的个人数据呢
- 解决方法：划分表

图3-19 第二范式的问题。客户数据中的隐含依赖导致客户每次从音像店租录像带时都会复制客户的地址。类似地，如果删除了日事务数据行，公司可能会丢失一些客户的所有数据

3.7.2 第三范式的定义

问题在前一节中已经阐述清楚。客户的姓名、地址、电话号码等等依赖于CustomerID，而CustomerID并不构成表的主码的一部分。换句话说，有些非主码列不依赖于主码。那么他们为什么在这张表中？问题同样也是答案。如果列不依赖于主码，那么他们应当放在不同的表中。

一张表要满足3NF，他必须首先满足2NF，并且每个非主码列必须只依赖于主码。在图3-20所示的音像店例子中，问题出在基本客户数据列依赖于CustomerID，而它并不是主码的一部分。



- 每一非主码列必须依赖且仅依赖于主码
- 如果某一列所依赖的列并非主码的一部分，将这些列划分到新表中去
  - 例：客户的姓名不随每个事务而变化

图3-20 第三范式定义。这张表不满足3NF，因为有些列依赖于CustomerID，而后者不是主码的一部分

乍一看，似乎有两种可行方案：(1) 将CustomerID作为部分主码，(2) 划分表。如果表已经满足2NF，那么只有 (2) 是可行的。第一种方案存在的问题是，将CustomerID作为主码

的一部分等价于说每个事务可以包括多个客户。这个假设似乎并不成立。然而，即便假设成立，你的表也将不再满足2NF，因为客户数据只依赖于主码的一部分（CustomerID而非TransID）。这样，正确的解决方案是将表分为两部分：依赖于整个主码的列和依赖于其他（CustomerID）的列。

对于音像店示例，解决办法是将依赖于CustomerID的列移出，记住，一定要在两个表中都包含CustomerID列，以便它们以后可以重新连接起来。结果表如图3-21所示。注意，CustomerID列不是Rentals表的主码，表可以通过列进行连接，即使这些列不是主码的一部分。图3-21还描述了如何划分表以解决隐含依赖的问题。

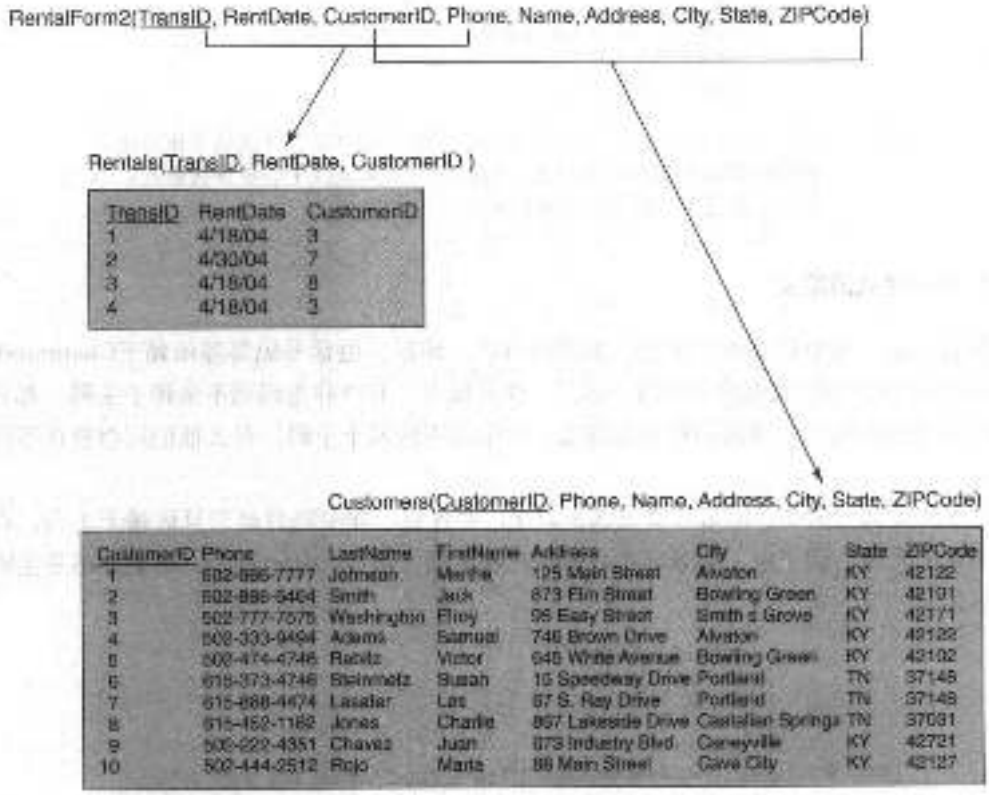
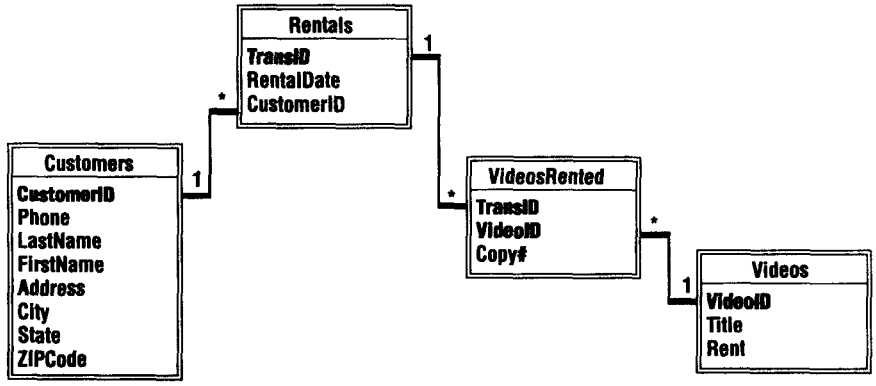


图3-21　第三范式，将客户数据放入单独的表中，消除隐含依赖并解决数据重复的问题。注意，CustomerID在两张表中都存在，但它不是Rental表的主码

最后得到的表集如图3-22所示。这个列表满足3NF：表中没有重复组（1NF），每个非主码列都依赖于所有主码（2NF），并且只依赖于主码（3NF）。这些表显示为标记表单和类图格式。类图使用Microsoft Access数据库管理系统创建，通过公共列显示这些表如何相互连接。

敏锐的读者或许会对地址数据产生疑问。即，City、State和ZipCode具有某种依赖关系。或许Customer表并不真的满足3NF？理论上讲，确实是这样，ZIP代码用于确定位置。此处的陷阱在于5位数字的ZIP代码标准是否够用，因此，这种关系非常弱。一个ZIP代码确定一个邮局。每个城市会有很多ZIP代码，并且一个ZIP代码可以用于多个城市。目前，一个ZIP代码总可以确定一个州。但是，你能确定这种关系永远满足吗——甚至在世界范围内？因此，在同一

个表中包含所有这三项通常是可以接受的。另一方面，正如在第2章宠物商店的讨论中所指出的那样，单独创建一个City表确实有一些好处。最重要的好处是，通过从预定义的列表中选取城市，可以减少输入数据的时间和错误。



Rentals(TransID, RentDate, CustomerID)

Customers(CustomerID, Phone, Name, Address, City, State, ZIPCode)

VideosRented(TransID, VideoID, Copy#)

Videos(VideoID, Title, Rent)

图3-22 第三范式表。表中没有重复组，而且每个非主码列依赖且仅依赖于整个主码

3.7.3 检查你的工作

在此关键点上，你必须双重检查你的工作。在大型项目中，应当让若干团队成员参与校验，以确保定义数据表所用的假设符合业务流程。

数据规范化的本质是收集所有表单和报表，然后检查每张表单，确定将要存储的数据。以标准符号书写列能使规范化过程更直接，将出错的可能性最小化。特别地，要注意查找主码并突出一对一和一对多的关系。检查工作时，需要查看每张表，确保它正确描述了假设和公司的流程。

检查表基本上是重复规范化中的步骤。首先，确保将所有的重复组移出。当进行这一步时，你应当双重检查主码。从表中第一个主码列开始，询问自己该列与其他每列是否存在一对一或者一对多的关系。如果存在一对多（或多对多）的关系，需要把该列的标题加上下划线。如果是一对一（或多对一）的关系，那么此列就不应当加下划线。第二步是检查每个非主码列，询问自己它是不是依赖于并且仅依赖于整个主码。第三，确认表可以重新连接起来。试着在表与表之间连线。不与其他任何表相连的表很可能是错误的。第四，问问自己是否每张表表示了一个单独的对象。试着给它起个名称。如果你找不到一个合适的单独的名字来命名一张表，那么这张表很可能代表了多个对象，需要进行划分。最后，给每张表输入示例数据，保证没有重复的行。当你开始输入数据的时候，一些潜在的问题可能就变得明显起来。最好在设计阶段输入测试数据，而不要等到最后实现才输入。

3.8 超越第三范式

在关系数据库设计理论中，E. F. Codd首先提出了这三种范式的规则。在观察现实世界的情况中，他和其他作者认识到在某些情形下可能产生另外的问题。特别地，Codd最初的3NF定义可能范围太窄了。因此，他和Boyce定义了一个新的版本，称为Boyce-Codd范式(BCNF)。

其他作者最终确定了可能产生的新问题，并进而创建了“范式”。幸运的是，这些情况在实际中并不常见。如果你在设计数据库时很仔细，尤其是创建主码时，你就不会遇到太多问题。但是，问题会偶尔出现，因此一个优秀的数据库设计者会检查下面各节中描述的问题。有许多设计人员的大型项目，团队中的每一个成员应当检查最终的表。

3.8.1 Boyce-Codd范式

你已经看到当表中存在隐含依赖时问题会如何产生。表中列之间的次要关系能引起重复和数据丢失的问题。考虑图3-23所示的例子，它包含雇员的数据。从业务规则可知，这张表满足3NF。主码是正确的，并且根据规则（c），非主码列（Manager）依赖于整个主码。即，对于每个雇员的每个专业，可以有一位不同的经理。问题的产生是由于规则（d）：每位经理只能有一个专业。经理可以确定专业，但经理不是表的主码，表中有一个隐含依赖（Manager→Specialty）。当你删除旧数据行，将某个经理的引用全删除了会怎样呢？你就会失去表示那位经理专业的数据。BCNF通过规定任何依赖必须显式地以主码表示，避免了这个问题。

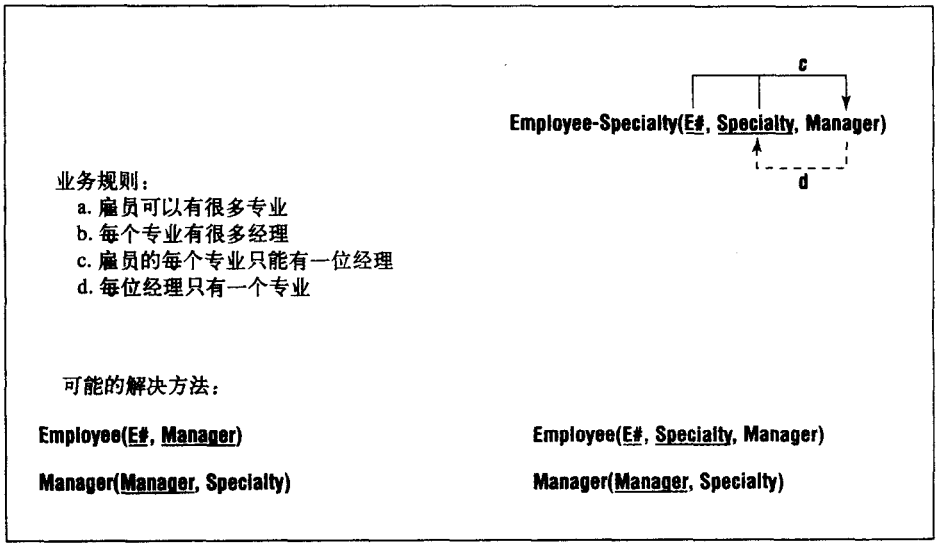


图3-23　Boyce-Codd范式。经理和专业之间有一个隐含依赖（d）。如果我们从原始表中删除数据行，那可能会丢失有关经理的数据。解决方法是添加一张表来使隐含显式化。从灵活角度考虑，保留原始的表可能更好一些——以防假设发生变化

解决方法是添加一张表，使这种依赖显示出来。因为每个专业可以有很多经理，因此最好的解决方案是添加一张表Manager（Manager，Specialty）。注意，技术上你现在可以将

Specialty列（连同主码Manager）从原表中移出。因为一位经理只能有一个专业，只要你知道经理，你就能通过连接获得专业。然而，如果你在原始表中保留三列，数据库会更加灵活，尽管它会产生一些重复数据。原因是，这个问题的产生是基于一些不寻常的假设。如果这些假设发生改变，就要修改表。在此例中，公司的经理只能拥有一个专业并不很现实。最好是保留原始表，并添加新的Manager表。如果假设发生改变，只需将Specialty作为Manager表的一个主码即可。重点在于你已经显式地表示了隐含关系，因此，不必再担心删除数据时会丢失重要的关系。

3.8.2 第四范式

当存在两个二元关系，建模人员试图将它们表示为一个合并的关系时，会引起第四范式（4NF）的问题。一个例子应当能说明这种情况。

图3-24中，雇员可以有很多专业，每种专业都可以完成很多任务。由于所有三列都是主码，这张表一定满足3NF。根据业务规则，你可以看到主码是合理的。然而，这里实际上有两个二元关系：Employee→Specialty和Employee→Tool，而不是一个三元关系。

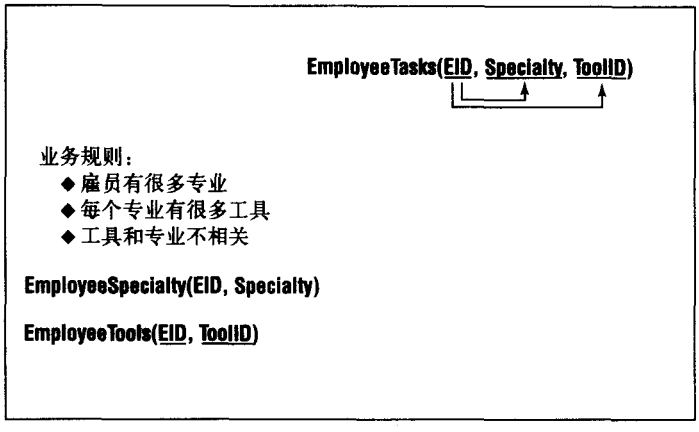


图3-24 第四范式。原始表满足3NF，因为不存在非主码列。主码是合理的，但存在隐含（多值）依赖，因为Specialty和ToolID不相关。解决方法是创建两张表——每张表显示依赖的一方

由于第三条业务规则指明，Specialty和ToolID并不直接相互依赖，因此，需要拆分原始表成两个表以去除隐含依赖。使用原始的设计，你会遇到的问题是，如果为每个雇员列出其每个专业所用的工具，那么就会存在客观的重复。将专业和工具分开列出会更有效。

第四范式的问题在实际当中相对少见。但是，它们仍会发生并且造成问题，因此，你要能够发现它们。主要的技巧是寻找隐含依赖，确保他们变得清楚。

3.8.3 域－码范式

1981年，Fagin提出了一种不同的规范化表的方法，命名为域－码范式（DKNF）。它描述设计数据库的最终目标。Fagin证明了，如果一张表满足DKNF，那么它一定也满足4NF，3NF以及其他所有范式。关键是没有一种特定的方法能使一张表转换到满足DKNF。事实上，有些

表可能永远无法转换为满足DKNF的表。

尽管有这些困难，DKNF对于应用开发者仍是很重要的，因为它是设计应用的目标。将它想象成当你没有精确方向时驾车去购物中心。只要你知道如何开始（1NF、2NF和3NF都是明确定义的），并且当你到达时（DKNF）能够辨认出购物中心，你仍然是可以到达的。

DKNF的目标是用每张表表示一个主题，并将业务规则以域约束和主码关系的形式表示。即，所有业务规则明确地表示为表的规则。域约束是很简单的，它们表示对一列中数据的限制。例如，价格不能为负数。

其他所有业务规则必须以主码的形式表现为关系。特别地，不能存在任何隐含关系。考虑图3-25所示的例子，它显示了一张学生和老师的表。它可能满足DKNF，只有在你检查业务规则之后才会清楚。一个典型大学的规则是：一个学生可以有多个老师，但每个专业只能有一个老师。此外，老师只能是自己所在学科的老师。有了这两条规则，上面两张表显然不满足DKNF。首先，主码SID可能不惟一。其次，必须有关于Major和Discipline的明确规则。图3-26表示了满足DKNF的三个新表。注意，现在所有的业务规则都明确地以主码和外码关系的形式表示。

```
Student(SID, Name, Major, Advisor)
Advisor(FID, Name, Office, Discipline)

业务规则:
一个学生可以有多位老师，但每个专业只能有一位老师。
老师只能是自己所在学科的老师。
```

图3-25　DKNF示例。此表不满足DKNF。因为一个学生可以有多个专业，SID不会是惟一的主码。另外，Advisor与Major相关，这是一种隐含关系

要定义一组满足DKNF的表，必须从满足3NF规则开始。然后确保有关于业务规则的完整列表。接下来确保业务规则都以域约束和主码关系的形式表示。检查主码以确保它们是惟一的，并且你找到了所有的多对多关系。确保没有任何隐含规则或依赖。设置外码关系以加强现有规则并与其他表中的数据相匹配。

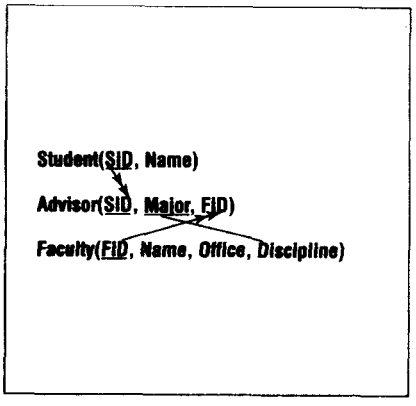


图3-26　DKNF解决方法。划分表使主码惟一。早先Major与Discipline之间的隐含关系通过外码约束变得明显

要了解域－码范式，返回到第2章的开始。设计数据库的目标是要建立组织模型，DKNF通过说明最好的数据库设计就是能用数据库规则明确描述所有的商业规则，从而澄清这一目标。

理论上，不会有任何超越DKNF的范式。这个原理很不错，因为目前还没有一种特定的方法能使一组表满足DKNF，所以它并不总是很有用。有些作者指出了其他潜在的问题，推导出另外版本的范式，诸如第五范式。由于这些定义大部分在实际中并不非常有用，在这里就不再进行描述。你可以参考C. J. Date的

教材了解更多有关理论概念的详细内容和一些示例。

3.9 数据规则和完整性

当你与用户和经理进行交流，设计报表和数据表时，你还要考虑需要强制执行哪些业务规则。数据库设计者的目标之一是保证数据的准确性。很多情况下业务规则是很显然的。例如，一般希望保证价格要大于零。类似地，可能有一条约束保证工资不应当超过诸如100000美元，或者雇用日期不能早于公司成立的日期。这些数据完整性约束在大部分数据库中是很容易指定的。典型地，你可以在表的定义中添加一条简单的约束连带着一条消息。将这些约束与表存储在一起的好处是，DBMS为该表上的每个操作强制检查该条件，无论数据源或数据输入的方法是什么。不需要编程，而且约束存储在一个位置。如果你需要修改条件，它很容易访问(对于授权用户)。

第二类约束是从某一组预定义的选项中选取数据。例如，性别可以列为male、female或未知。提供一个列表辅助职员输入数据，并且强制他们只输入提供的选项。例如，你不必担心有人会输入f、F或fem。数据的一致性更强。

第三类数据完整性更复杂一些，但在关系数据库中非常重要。数据表依据属性很好地组织，确保有效地存储数据。然而，你要能将表中的数据重新连接起来，以获得用户需要的报表和表单。考虑图3-27所示的音像店示例，职员向Rentals表输入一个客户号。如果职员输入的客户号在Customer表中不存在，将会发生什么？在那盘录像带归还前，你没有办法找到那位客户。因此，你需要一个约束来保证当一个客户号输入到Rentals表中时，那个号码必须已经在Customer表中存在。Rentals表中的CustomerID是该表的一个外码，你所需的约束就是所谓的参照完整性。参照完整性指的是，只有当一个外码值在原表中存在时才能被输入。

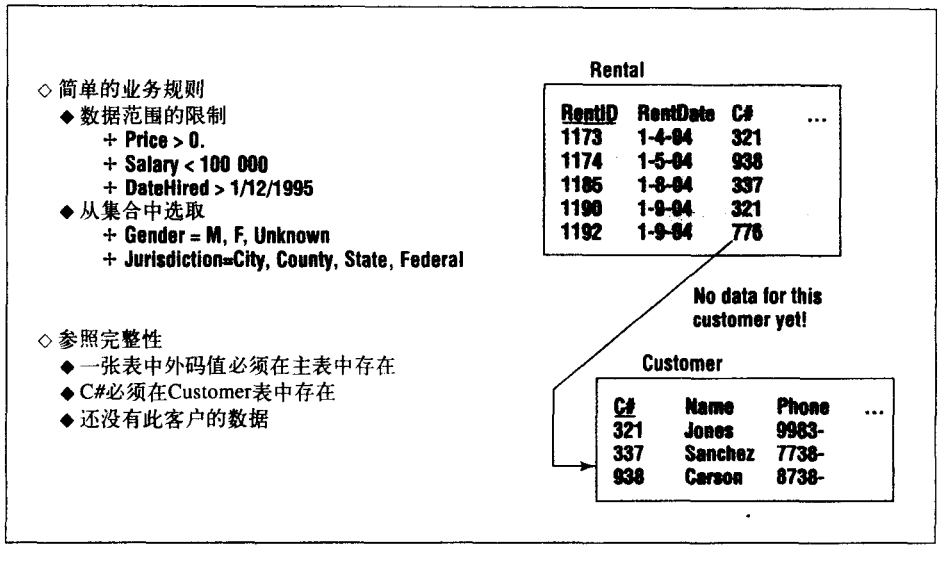


图3-27 数据完整性。数据完整性可以通过简单的规则来维护。关系数据库依赖参照完整性约束来保证在用户数据存在之后用户编号才能输入到Rental表中

本质上，一旦定义了表之间的关系，你可以要求DBMS强制参照完整性。定义参照完整性的方法与具体的DBMS相关。在Access中，使用一种类图建立和显示这些关系，其中连接线段表示参照完整性约束。大多数关系数据库还支持级联删除，使用了相同的概念。如果用户删除了Customer表中的一行，那么你还需要删除Rentals表中相关的数据。然后你需要删除VideosRented表中的相应行。如果你建立了关系并指定级联删除，当用户从Customer表中删除一条记录时，数据库会自动删除相关的行。通过保证表之间的连接总是引用合法的行，这些操作维护了数据库的一致性。

Oracle和SQL Server通过在创建VideosRented表时声明外码来提供对参照完整性的支持。图3-28显示了可用于创建带有三列的Order表的命令。这家公司希望保证所有的订单都发往有效的客户，因此Order表中的客户号（CID）必须在Customer表中存在。外码约束强制了这个关系。这个约束还指明，此关系应当处理级联删除。Oracel和SQL Server使用标准SQL语言创建表。

```
CREATE TABLE Order
( OID    NUMBER(5) NOT NULL,
  Odate  DATE,
  CID    NUMBER(5),
         CONSTRAINT pk_Customer
           PRIMARY KEY (OID),
         CONSTRAINT fk_Customer
           FOREIGN KEY (CID)
           REFERENCES Customer (CID)
           ON DELETE CASCADE
)
```

图3-28　SQL参照完整性定义。在Order表中，将一列声明为外码会使DBMS检查表中的每个值，在参照表（例如，Customer）中找到匹配的值

当你开始向DBMS输入数据，你很快会发现参照完整性所起的作用。考虑两张表：Order（OrderID，Odate，CustomerID）和Customer（CustomerID，Name，Address，等等）。你有一个参照完整性约束，将Order表中的CustomerID列和Customer表中的CustomerID列连接到一起。现在你向两张表输入示例数据，但从Order表开始。DBMS不会接受任何数据，因为相应的CustomerID必须首先在Customer表中存在。即，参照完整性规则强迫你按特定顺序输入数据。很明显，这些规则会给用户带来问题，因此你不能指望用户直接向表中输入数据。第6、7和8章将介绍如何使表单和应用程序自动保证用户以适当的顺序输入数据。

3.10　业务规则的影响

理解不同的业务规则如何影响数据库设计和规范化过程是十分重要的。作为数据库设计者，必须确认基本规则并建立数据库满足它们。但需要小心的是，业务规则可能改变。如果你觉得当前的业务规则限制性太强，你就应当为数据库设计一个更灵活的结构。

考略图3-29所示的例子。当地的公园和娱乐部门经营一个足球俱乐部，在每次比赛之后收集基本的统计信息。你要为这个问题设计数据表。

Location Date Played					Referee Name Phone Number, Address				
Team 1 Name Sponsor		Score			Team 2 Name Sponsor		Score		
Player Name	Phone	Age	Points	Penalties	Player Name	Phone	Age	Points	Penalties

图3-29 足球俱乐部的数据库设计。设计和规范化的表依赖于业务规则

要描述不同规则的影响，考虑两种主要的规则及其结果表，如图3-30所示。第一条规则规定，每场比赛只能有一名裁判。因此，RefID可以放在Match表中。注意，它不是主码的一部分。第二条规则规定，一个运动员只能为一支球队效力；因此，适当的TeamID可以放在Player表中。

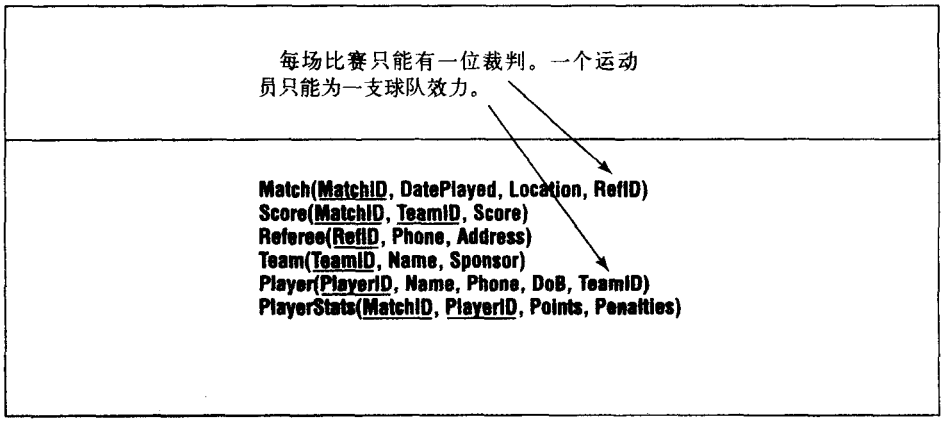


图3-30 限制规则。每场比赛只能有一位裁判，裁判表的主码添加到Match表中。类似地，TeamID列放置在Player表中

现在考虑如果这两条规则不严格的情况，如图3-31所示。俱乐部经理认为某一天每场比赛可以有多名裁判。另外，替补球员产生了问题。一名替补球员在一个赛季中可能为多个球队效力——但每场比赛只属于一个队。要处理这些新规则，主码值必须改变。你可能会进行图3-31所示的简单修改；即，将RefID作为Match表的部分主码，并使TeamID成为Player表的部分主码。现在每场Match可以有多名Referee，而且每名Player可以为多个球队效力。这种方法的问题在于，Match表和Player表不再满足3NF。例如，DatePlayed不依赖RefID。同样，Player表中的Name不依赖于TeamID。例如，Paul Ruiz在不同球队效力时他的名字并不变。

解决办法如图3-32所示。添加一个新表来处理裁判和比赛之间多对多关系。类似地，球员的TeamID转移到PlayerStats表中，但它不是主码的一部分。在这种方案中，每场比赛有多名球

员，每名球员可以参加多场比赛。然而，对于每场比赛，每名球员只能为一支球队效力。这种新的数据库设计与原始的设计不同。更重要的是，它的限制性更少。作为一个设计者，你在设计数据库的时候必须考虑到未来，以便它能处理部门未来的需要。

每场比赛可以有多位裁判。一个队员可以在多个球队踢球（替补），但每场比赛只属于一个队。

Match(MatchID, DatePlayed, Location, RefID)
Score(MatchID, TeamID, Score)
Referee(RefID, Phone, Address)
Team(TeamID, Name, Sponsor)
Player(PlayerID, Name, Phone, DoB, TeamID)
PlayerStats(MatchID, PlayerID, Points, Penalties)

图3-31　放松规则来允许多对多的关系。你可能试图将RefID列和TeamID列作为部分主码，但结果表不满足3NF。Location不依赖于RefID，Player Name不依赖于TeamID

Match(MatchID, DatePlayed, Location)
RefereeMatch(MatchID, RefID)
Score(MatchID, TeamID, Score)
Referee(RefID, Phone, Address)
Team(TeamID, Name, Sponsor)
Player(PlayerID, Name, Phone, DoB)
PlayerStats(MatchID, PlayerID, TeamID, Points, Penalties)

图3-32　放松规则与规范化表。RefereeMatch表允许每场比赛有多个裁判。将TeamID移到PlayerStats表中表明某个人可以为多个球队踢球——但每场比赛只属于一个球队

这些数据库设计哪个是正确的？答案取决于部门的需要。在实际当中，选择那个更灵活的设计是比较明智的，它能为一场比赛分配多名裁判，并允许运动员在赛季中为多个球队效力。但是，在实际当中，你需要对这个数据库设计进行一些小改动。如果尚未进行任何比赛，你怎么能知道每支球队有哪些球员效力呢。按照目前的设计，数据库无法回答这个问题。解决办法是向Player表中添加一个BaseTeamID列。在赛季初，每支球队会提交一个名册，列出球队的初始成员。球员只能出现在一支球队的初始名册上。如果某人替补或转会，数据会重新记录在PlayerStats表中。

3.11　将类图转化为规范化的表

每个规范化的表代表了一个业务实体或类。因此一张类图可以转化为一系列规范化的表。同样，一系列规范化的表可以绘制成一张类图。技术上，一张类图中的实体不必非要满足3NF（或更高）。有些设计者使用类图作为业务的概念模型，这省略了一些规范化的细节。在这种情况下，需要将类转化成一系列规范化的表。正如第2章所述，类图中一些公共的功能，因此你要学会如何进行基本的转化。

图3-33描述了一个典型的订购单类图，它包含四类基本关系：（1）供应商和购买者订单之间的一对多关系；（2）购买订单和商品之间的多对多关系；（3）包含不同属性的子类型关系；（4）Employee实体中的递归关系，表示有些雇员是其他雇员的经理。

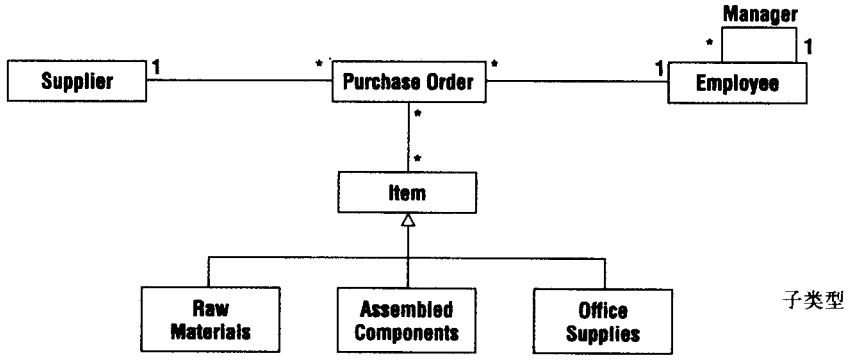


图3-33 将类图转化为规范化的表。注意四种关系类型：（1）一对多，（2）多对多，（3）子类型和（4）递归

3.11.1 一对多关系

将类图转化成规范化表的最重要原则是，通过在每个相关表中设置一个公共列来处理关系。此列通常是某张表的主码列。在一对多关系里可以很容易地看到这个过程。

订购单例子里有两个一对多关系：（1）很多不同的订购单可以发往每个供应商，但每个订购单上只能出现一个供应商。（2）每个订购单只能由一名职员创建，而一名职员可以创建很多订购单。要创建规范化的表，首先为每个实体（Supplier、Employee和Purchase Order）创建一个主码。如图3-34所示，规范化的表可以通过在PurchaseOrder表中放置Supplier表的主码（SID）和Employee表的主码（EID）来进行连接。仔细注意，所有的类图关联都以主码之间的关系来表示。

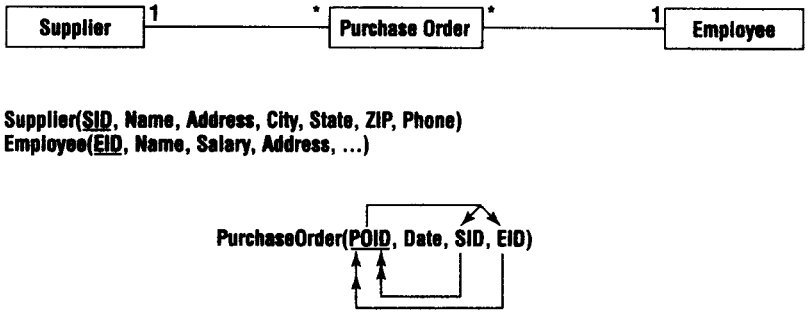


图3-34 转换一对多关系。将一方实体的主码添加到多方实体的表中。在示例中，SID和EID添加到PurchaseOrder表中。注意，它们不是Purchase-Order表的主码

还要注意，SID和EID不是PurchaseOrder表的主码列。你可以检查哪些列应当设为主码。从POID列开始。对于每个PurchaseOrder（POID），有多少供应商？业务规则规定每个订购单

只能有一位供应商；因而SID不应设为主码，不必为SID加下划线。现在从SID开始从相反方向检查，对于每位供应商，有多少订购单？业务规则规定多个订购单可以发往一个已知的供应商，因此PID列需要设为主码。相同的过程指出EID不应为主码；它属于PurchaseOrder表，因为每位Employee可以下多个订单。图3-35使用示例数据说明表之间是如何通过主码列连接的。

Supplier

SID	Name	Address	City	State	ZIP	Phone
5676	Jones	123 Elm	Ames	IA	50010	515-777-8988
6731	Markle	938 Oak	Boston	MA	02109	617-222-9999
7831	Paniche	873 Hickory	Jackson	MS	39205	601-333-9932
8872	Swensen	773 Poplar	Wichita	KS	67209	316-999-3312

PurchaseOrder

POID	Date	SID	EID
22234	9-9-2004	5676	221
22235	9-10-2004	5676	554
22236	9-10-2004	7831	221
22237	9-11-2004	8872	335

Employee

EID	Name	Salary	Address
221	Smith	67,000	223 W. 2300
335	Sanchez	82,000	37 W. 7200
554	Johnson	35,000	440 E. 5200

图3-35　一对多关系的示例数据。Supplier表和PurchaseOrder表通过SID列连接。类似地，Employee表通过EID列中的数据连接。SID和EID列都是PurchaseOrder表的外码，但它们不是那张表的主码

3.11.2　多对多关系

概观类图通常包含多对多关系，但是，在关系数据库中，多对多的关系必须划分成两个对多的关系以满足BCNF。图3-36描述了处理PurchaseOrder表和Item表的过程。

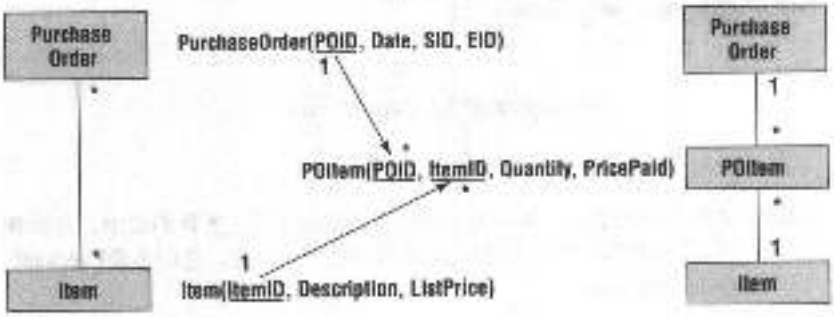


图3-36　转换多对多关系。多对多关系使用一个新的中间表来连接两张表。新的POItem表包含PurchaseOrder表和Item表的主码

两个初始实体中的每个都变成一张表（PurchaseOrder和Item）。下一步是创建一张新表（POItem），它包含另两张表的主码（POID和ItemID）。此表表示多对多的关系。每个订购单（POID）能包含多项，因此ItemID必须设为主码。类似地，每项也可以在多个订购单（POID）中订购，因此POID也必须设为主码。

你必须有一张包含POID和ItemID作为复合码的表。能否建立这种关系而不创建第3张表？大多数情况答案是否定的。考虑一下，如果将ItemID列放入PurchaseOrder表，并将其设为部分主码，会发生什么？结果实体将是不满足3NF的表，因为Date、SID和EID都不依赖于ItemID。如果将主码POID放入Item表，会产生类似的问题，因此，中间表是必需的。图3-37使用示例数据说明三个表是如何通过主码连接起来的。

PurchaseOrder

POID	Date	SID	EID
22234	9-9-2004	5675	221
22235	9-10-2004	5675	554
22236	9-10-2004	7831	221
22237	9-11-2004	8872	335

POItem

POID	ItemID	Quantity	PricePaid
22234	444098	3	2.00
22234	444185	1	25.00
22235	444185	4	24.00
22236	555828	10	150.00
22236	555982	1	5,800.00

Item

ItemID	Description	ListPrice
444098	Staples	2.00
444185	Paper	28.00
555828	Wire	158.00
555982	Sheet steel	5,928.00
888371	Brake assembly	152.00

图3-37 多对多关系的示例数据。注意，中间表POItem连接其他两张表。检查这三张表满足3NF，其中每一非主码列依赖且只依赖于整个主码

3.11.3 多重关联

正如第2章所述，多重关联以菱形标记，这个菱形关联也会成为一个类。某种意义上讲，一个多重关联可以简单地看作是若干二元关联的集合。如图3-38所示，新的关联类拥有其他每个类的主码。只要二元关联是一对多的，Assembly类中的每一列都会成为主码的一部分。如果一个二元关联是一对一的，那么相应的列就不应设为主码。

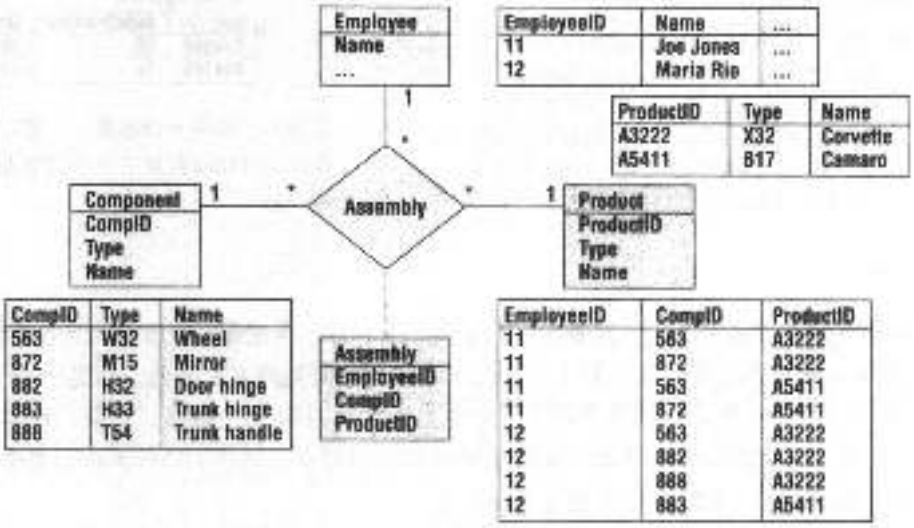


图3-38 多重关联。Assembly关联也是一个类。它可以看作是一个二元（一对多）关联的集合。新的Assembly类包含每个主类的主码

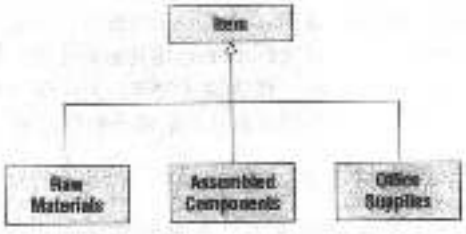
3.11.4　概括或子类型

有些业务实体作为子类型创建。图3-39通过Item实体描述这种关系。一个项是购买物品的一般描述。每个项都有一个描述和一个标价。但是，公司处理三种类型的项：原料、装配元件和办公用品。每种子类型都有一些你希望记录的附加属性。例如，公司记录原料的重量、装配元件的尺寸以及办公用品的数量折扣。

在将此设计转化为关系数据库时，有两种基本方法。（1）如果子类型都很相似，可以忽略子类，将所有的子类压缩到主类中，主类将会包含每个子类的所有属性。在这种情况下，每条数据项将会有若干空值。（2）大多数情况下，更好的办法是为每个子类创建单独的表。每张表都将包含主类Item的主码。

如图3-40所示，每一项都在Item表中有一条记录。在三个子类型表之一还有另外一条记录依赖于特殊项。例如项444098在Item表中描述。其余数据存储在OfficeSupplies表中。

如果子类关系不互斥，那么每个主项可以在多个子类表中拥有相应的行。



Item(ItemID, Description, ListPrice)

RawMaterials(ItemID, Weight, StrengthRating)

AssembledComponents(ItemID, Width, Height, Depth)

OfficeSupplies(ItemID, BulkQuantity, Discount)

图3-39　转化子类型。每件购买商品具有基本属性，记录在Item表中。每件商品属于三种类型之一，这三类具有不同的属性。要把这些关系转化为3NF，为每种子类型创建新表。在新表和普通表中使用相同的主码。为每种子类型添加特殊的属性

Item

ItemID	Description	ListPrice
444098	Staples	2.00
444185	Paper	28.00
555828	Wire	158.00
555982	Sheet steel	5928.00
888371	Brake assembly	152.00

RawMaterials

ItemID	Weight	StrengthRating
555828	57	2000
555982	2578	8321

AssembledComponents

ItemID	Width	Height	Depth
888371	1	3	1.5

OfficeSupplies

ItemID	BulkQuantity	Discount
444098	20	10%
444185	10	15%

图3-40　子类型关系的示例数据。注意，现在每件商品在Item表及三个子类型表之一都有一行

3.11.5　组合

从某种角度讲，组合是一个多重关联和子类型的合并。考虑图3-41所示的自行车例子，一辆自行车由各种元件组装而成。第一个要做的决定是如何处理多个元件。这是一个子类型的问题。在此例中，业务几乎记录了每个元件各自的数据（ID号、规格、重量、成本、标价，等等）。因此，一个好的解决办法是将所有子类型压缩到一个通用的Component类中。然而，单独处理车轮也可以，因为它们是更复杂组件。

你可以通过在主Bicycle表中为每个元件（WheelID、CrankID、StemID，等等）创建属性来解决主要的组合问题。这些列是Bicycle表的外码（但非主码）。当制造一辆自行车时，安装元件的ID值将存储在Bicycle表的适当列中。通过查看实际的Rolling Thunder数据库，可以找

到更多信息。

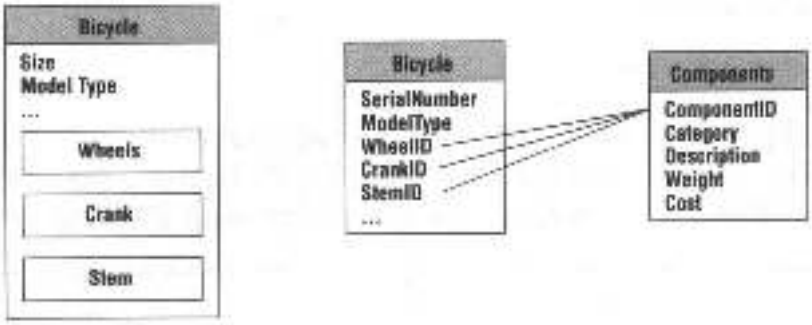


图3-41 规范化一个组合关联。首先决定如何处理子类。在此例中，把它们合并到一个Component表中。第二，通过在主Bicycle表中存储适当的Component主码值来处理组合

3.11.6 自反关联

有时候，一个实体可能与自身连接。一个常见的例子如图3-42所示，其中职员里包含经理。由于经理本身也是职员，所以这个实体连接到自身。当建立相应的表时会很容易看出这种关系。简单地向Employee表添加一个Manager列即可。这一列的数据包含一个EID。例如，第一个职员（Smith，EID 221）向经理335（Sanchez）汇报工作。Manager列是主码的一部分吗？不，因为业务规则规定一个雇员只能有一个经理。



Employee(EID, Name, Salary, Address, Manager)

Employee

EID	Name	Salary	Address	Manager
221	Smith	67,000	223 W. 2300	335
335	Sanchez	82,000	37 W. 7200	
554	Johnson	35,000	440 E. 5200	335

图3-42 转化递归关系。一个雇员只能有一位经理，因此向Employee表添加Manager列，表中包含指向经理的EID列。在示例中，Smith向Manager335（Sanchez）汇报

一个雇员能有多个经理的情况该如何处理呢？将Manager列设为主码？不，因为那样的话Employee表就不满足BCNF了（一个职员的地址不依赖于他的经理）。解决方案是创建一个新表列出EmployeeID和ManagerID，两列都是主码的一部分。这个新表可能有额外的数据来描述职员和经理之间的关系，例如一个项目或任务。

3.11.7 小结

创建一个详细的类图其实和创建一系列规范化的表是一样的。事实上，类图和规范化的表都是业务模型，无论是用线段还是通过主码来表示关联，都必须符合业务规则。如果从类图

开始，那么一定要验证每个类都满足BCNF。而且，一定要检查特殊的情况，诸如多重关联、概括、组合以及自反关联。

3.11.8 Sally的宠物商店示例

要设计Sally的宠物商店数据库，和店主交谈并且调研其他商店的经营方式。在收集各种表单信息的过程中，你开始了解业务规则。为了加快开发进度并降低成本，你和Sally都同意先从一个简单的模型开始，以后再添加功能。图3-43所示的销售表单包含销售活动所需的主要数据。

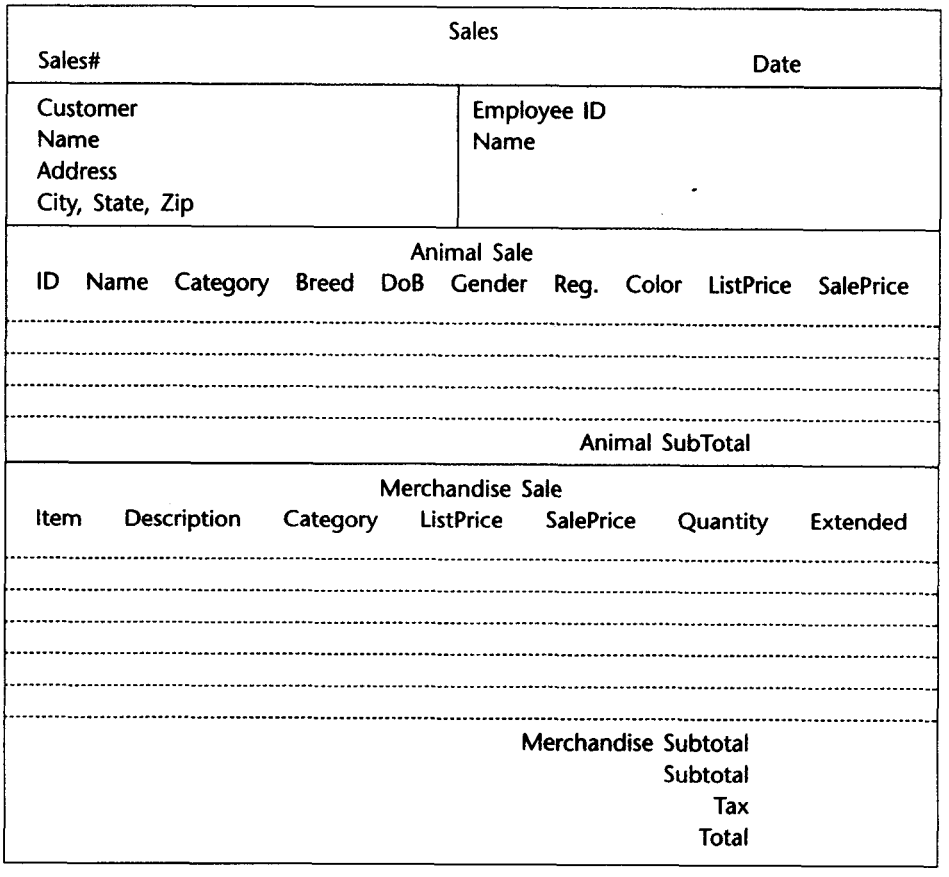


图3-43 宠物商店示例的销售表单。将销售的动物和商品分开，表示业务规则上要分别对待它们

Sally希望你为动物和产品创建单独的订购单。她反复强调收集动物详细信息的重要性，从收集饲养者的信息到最终收集动物谱系的数据。对于像猫和狗这样已注册的动物，这些数据很容易获得。然而，要获取鱼的记录就很难。Sally还希望获取所购宠物的医疗记录。一般的数据包括它们注射的疫苗、相关疾病和它们接受过药物或治疗。目前，她依赖于饲养者保存这些信息。然而，一旦创建了销售和基本的金融应用之后，她希望将这些功能添加到数据库。

目前最重要的工作是收集交易数据。图3-44显示从饲养者那里购买动物时必须记录的最少金融数据。注意收集关于动物、供应商/饲养者、客户和职员数据的重要性。由于预期的变化，

将数据库设计得灵活是很重要的。设计需要很容易为所有主要实体添加新属性。还应当很容易添加新表（例如健康记录）而不必对初始结构做大量修改。

Purchase Order for Animals

Order# Date Ordered
Date Received

Supplier Name Contact Phone Address City, State **ZIPcode**	Employee ID Name Home Phone Date Hired

Animal Descriptions

Name	Category	Breed	Gender	Registration	Price

Subtotal
Shipping Cost
Total

图3-44 宠物商店示例的购买动物订单。更多的信息会在未来收集——尤其是每种动物的健康和谱系数据

从供应商购买商品的流程与购买宠物相似。但是，有一些细微的区别。特别地，你需要为每件商品收集不同的数据。

一个示例表单如图3-45所示。再次记住，Sally希望从一个小型数据库开始。以后将会收集附加的数据。例如，当订单的货物到达，有些商品丢失，会发生什么？当前的表单只能记录完整货运的到达。类似地，每家供应商可能使用独特的Item编号集合。例如，对于同一罐猫食，一家供应商可能以ItemID 3325订购，另一家供应商可能会以ItemIDA9973订购。最终，Sally希望记录其主要供应商的编号。这样，当货物带着编号抵达时，将产品与她的订单相匹配就变得更加容易。

设计宠物商店数据库的下一个步是根据每张表单创建将用于存储该表单数据的所有规范化的表。图3-46显示从Sales表单得到的表。在详细检查结果之前，应当进行数据规范化。看你是不是得到了相同的结果。还应当获取另两张表单的规范化表。要双重检查你的工作。首先确保主码正确，然后检查每个非主码列是否依赖且仅依赖于整个主码。

在SaleAnimal和AnimalOrderItem表中（图3-46）存在一个有趣的假设。SaleAnimal表使用SaleID和AnimalID作为主码。这意味着每次销售可以包含若干动物。这还意味着每只动物可以售出若干次。后一种假设是可能的吗？一只动物能否售出多次？如果不能，那么SaleID就不应设为主码的一部分，可以把它直接插入到Animal表中。同样，能否多次购买同一只动物？如果不能，那么OrderID可以放入Animal表中，但并不设为主码的一部分。SaleAnimal和AnimalOrderItem表的其他数据也可以放到Animal表中。尽管这种方法显得更实际，但缺乏灵活性。这样设计数据库意味着宠物商店永远不能两次售出同一只动物。那么该如何处理一只

退回的动物呢？

Purchase Order for Merchandise

Order#　　Data Ordered
Date Received

Supplier Name Contact Phone Address City, State **ZIPcode**	Employee ID Name Home Phone

Items Ordered

ItemID	Description	Category	Price	Quantity	Ext.	QOH
				Subtotal		
				Shipping Cost		
				Total		

图3-45　宠物商店示例的购买商品订单。注意这两种订单的相似性与不同。记住，更多的数据会在未来收集

```
Sale(SaleID, Date, CustomerID, EmployeeID)
SaleAnimal(SaleID, AnimalID, SalePrice)
SaleMerchandise(SaleID, ItemID, SalePrice, Quantity)
Customer(CustomerID, Name, Address, City, State, Zip)
Employee(EmployeeID, Name)
Animal(AnimalID, Name, Category, Breed, DateOfBirth, Gender,
Registration, Color, ListPrice)
Merchandise(ItemID, Description, Category, ListPrice)
```

图3-46　宠物商店基本销售表单的规范化表。应当首先进行规范化，然后检查你的结果是否与这些表一致

3.12　视图集成

到目前为止，已经使用单独的报表和表单对数据库设计和规范化进行了讨论，这是设计数据库的基本手段。但是，大部分项目包含很多报表和表单。有些项目会有一个设计团队，其中每个人都从不同的用户和部门收集表单和报表。每位设计者为各自的表单创建规范化的表，你最后得到与同一主题相关的很多表。此时，需要将所有这些表集成到一个完整的、一致的数据表定义集合中。

当你完成这一阶段，就可以将表的定义输入DBMS中。尽管你可能最后得到很多相互关联的表，这一步一般还是要比初始推导获得满足3NF的表要容易。此时，你收集所有的表，确保命名的一致性，从相似表中合并数据。合并表的基本步骤如下：

- 收集多种视图（文档、表单，等等）

- 为每篇文档创建规范化的表
- 将视图合并为一个完整的模型

3.12.1 Sally的宠物商店示例

图3-47描述宠物商店示例的视图集成过程。首先列出由三个输入表单生成的表。集成从查找包含相似数据的表开始。一个好的起点是查看主码。如果两张表有相同的主码，那么一般应该合并。然而，要小心。有时主码是错误的，有时主码的名称可能有细微的不同。

```
Sale(SaleID, Date, CustomerID, EmployeeID)
SaleAnimal(SaleID, AnimalID, SalePrice)
SaleItem(SaleID, ItemID, SalePrice, Quantity)
Customer(CustomerID, Name, Address, City, State, Zip)
Employee(EmployeeID, Name, Phone, DateHired)
Animal(AnimalID, Name, Category, Breed, DateOfBirth,
Gender, Registration, Color, ListPrice, Cost)
Merchandise(ItemID, Description, Category, ListPrice,
QuantityOnHand)

AnimalOrder(OrderID, OrderDate, ReceiveDate, SupplierID,
EmpID, ShipCost)
AnimalOrderItem(OrderID, AnimalID, Cost)
Supplier(SupplierID, Name, Contact, Phone, Address, City,
State, Zip)
Employee(EmployeeID, Name, Phone, DateHired)
Animal(AnimalID, Name, Category, Breed, Gender,
Registration, Cost)

MerchandiseOrder(PONumber, OrderDate, ReceiveDate, SID,
EmpID, ShipCost)
MerchandiseOrderItem(PONumber, ItemID, Quantity, Cost)
Supplier(SupplierID, Name, Contact, Phone, Address, City,
State, Zip)
Employee(EmployeeID, Name, Phone)
Merchandise(ItemID, Description, Category, QuantityOnHand)
```

图3-47 宠物商店的视图集成。相似表中的数据列可以合并到一张表中。例如，我们只需要一张Employee表。查找具有相同主码的表。目标是得到一个规范化表的集合，它能储存所有表单和报表的数据

注意，Employee表在例子中出现三次。通过仔细检查每个列表中的数据，可以构造一张包含所有列的新表。因此，Phone列和DateHired列移到一张表中。另外两张表被删除。对Supplier表、Animal表和Merchandise表可以进行相似处理。此过程的目标是创建一系列完整的规范化的表。将存储所有表单和报表中的数据。一定要双重检查，以确保最终的所有表满足3NF或BCNF。另外，确保这些表可以通过相关列的数据进行连接。

最终确定的表也可以显示在一张详细的类图上。宠物商店的类图如图3-48所示。类图的一个好处是能够显示类（表）之间是如何通过关系相连的。双重检查你的规范化处理以确保基本的表单可以重新创建。例如，销售表单可以从Customer、Employee和Sale表开始建立。动物的销售需要SaleAnimal和Animal表。产品的销售需要SaleItem和Merchandise表。所有这些表都能通过它们属性的关系进行连接。

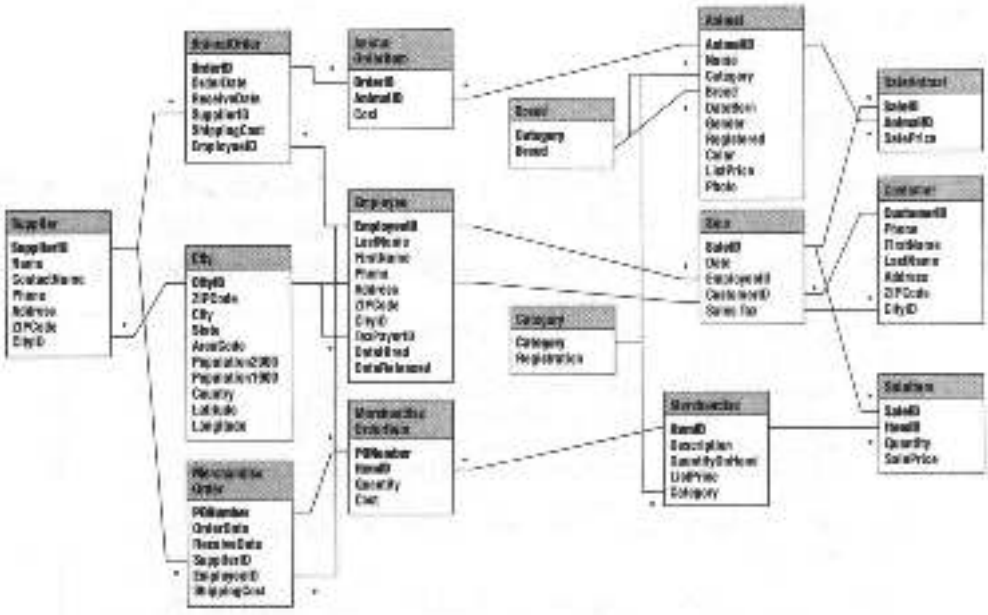


图3-48　宠物商店类图。表变成类图中的实体。关系验证表之间通过数据互相连接。添加了一些新职员。另外，为了简化数据条目，城市定义在一张单独的表中。同样，新Breed和Category表确保了数据的一致性

大部分关系是一对多的关系，但要注意方向。Access用一个无穷大（∞）符号来标记多的一边。当然，你首先应当从业务规则来确定恰当的关系。例如，每位客户可以有多条消费记录，但是每条消费记录只能列出一位客户。

图3-48所示的最终类图包含三张新表：City，Breed和Category。添加这些确认表用于简化数据输入和保持数据一致性。如果没有这些表，雇员将不得不重复输入城市名称、品种和类别的文本数据。让用户录入这些数据存在两个问题：（1）浪费时间。（2）用户可能每次输入不同的数据。通过在这些表中存放标准数据，雇员可以从列表中选择适当的数据。因为这些标准数据总会拷贝到新表中去，因此，每次输入的数据都是相同的。

让DBMS强制特定的关系造成了一个有趣的结果。这些关系需要数据按指定顺序输入。外码关系指定某位客户的数据必须首先在Customer表中存在，然后才能放进Sale表。从业务观点上看这条规则是有意义的，你必须首先遇见客户，然后才能卖给他们东西。然而，这条规则可能会给输入销售数据的职员带来问题。你需要某种机制，在他们要输入Sales数据之前，帮助他们输入新的Customer数据。

3.12.2　Rolling Thunder示例中的集成问题

掌握数据库设计和理解规范化的惟一方法是进行更多的练习。为了对数据规范化的概念进行实践和介绍合并表集的方法，考虑一个包含数据库的小型制造厂的新问题：Rolling Thunder自行车。这家公司为客户定制的自行车。在公司内部制造框架并喷漆。零部件从生产厂购买，按照顾客的意愿组装成自行车。零部件（曲柄、踏板、变速器等）一般进行分组以便客户可以选定一整套零部件，而不必单独指定每一项。其他有关自行车和公司运营的详细信息可从

要从各种角度理解规范化和表集成的过程，考虑四张输入表单：Bicycle Assembly、Manufacturer Transactions、Purchase Orders和Components。

制造者使用图3-49所示的Bicycle Assembly表单决定框架的基本设计，喷漆的样式和需要安装的零部件。当完成框架制造和零部件组装之后，工人们要对操作进行核对，雇员标识和日期/时间存储进数据库中。当零部件组装完毕后，库存数量自动减少。当自行车发送出去，触发器执行代码记录用户的应付款以便打印并邮寄账单。

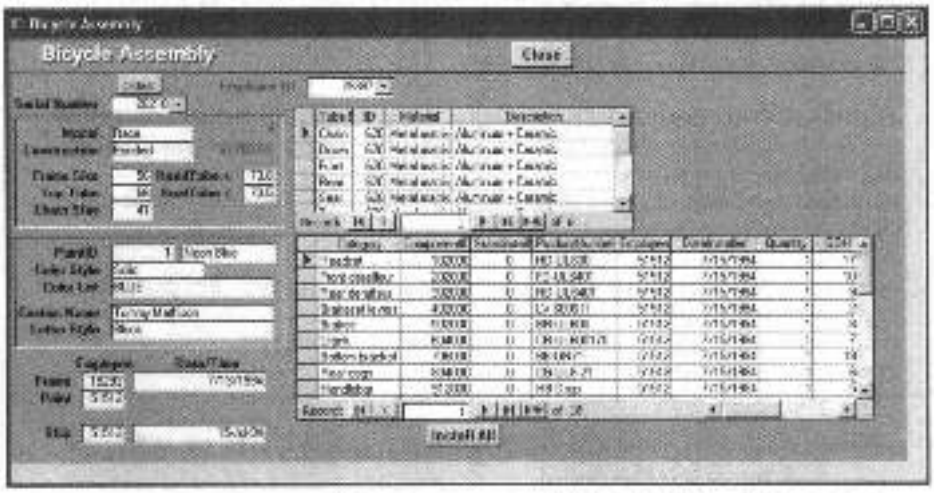


图3-49 Bicycle Assembly表单，主EmployeeID控制不直接存储，但是当雇员点击Frame框时，该值在Bicycle表的FrameAssembler列中输入

从表单中收集数据列，得到图3-50所示的结果。注意，出现了两个重复组（管子和零部件），但它们相互独立地重复，不嵌套。

```
BicycleAssembly(
SerialNumber, Model, Construction, FrameSize, TopTube,
ChainStay, HeadTube, SeatTube, PaintID, PaintColor,
ColorStyle, ColorList, CustomName, LetterStyle, EmpFrame,
EmpPaint, BuildDate, ShipDate,
  (Tube, TubeType, TubeMaterial, TubeDescription),
  (CompCategory, ComponentID, SubstID, ProdNumber,
  EmpInstall, DateInstall, Quantity, QOH) )

Bicycle(SerialNumber, Model, Construction, FrameSize,
TopTube, ChainStay, HeadTube, SeatTube, PaintID,
ColorStyle, CustomName, LetterStyle, EmpFrame, EmpPaint,
BuildDate, ShipDate)
Paint(PaintID, ColorList)
BikeTubes(SerialNumber, TubeID, Quantity)
TubeMaterial(TubeID, Type, Material, Description)
BikeParts(SerialNumber, ComponentID, SubstID, Quantity,
DateInstalled, EmpInstalled)
Component(ComponentID, ProdNumber, Category, QOH)
```

图3-50 BicycleAssembly表单的标记。存在两个重复组，但它们相互独立。从此表单得到4NF表，你应当试着自己推导这些表

零部件和其他用料从生产厂购买。当用料过少时就会下订单并记录在Purchase Order表单上。如图3-51所示，Purchase Order包含关于生产厂的标准数据，以及订购的零部件（或其他用料）列表。

```
PurchaseOrder(PurchaseID, PODate, EmployeeID, FirstName,
LastName, ManufacturerID, MfgName, Address, Phone, CityID,
CurrentBalance, ShipReceiveDate, {ComponentID, Category,
ManufacturerID, ProductNumber, Description, PricePaid,
Quantity, ReceiveQuantity, ExtendedValue, QOH,
ExtendedReceived}, ShippingCost, Discount)

PurchaseOrder{PurchaseID, POData, EmployeeID, ManufacturerID,
ShipReceiveDate, ShippingCost, Discount} Employee(EmployeeID,
FirstName, LastName)
Manufacturer{ManufacturerID, Name, Address, Phone, Address,
CityID, CurrentBalance)
City{CityID, Name, ZipCode)
PurchaseItem{PurchaseID, ComponentID, Quantity, PricePaid,
ReceivedQuantity}
Component(ComponentID, Category, ManufacturerID,
ProductNumber, Description, QOH)
```

图3-51　从Purchase Order表单得到的表。注意，计算得到的列（扩展为价格*数量）没有存储在表中

经过推导的标记和满足4NF的表如图3-52所示。你应当自己独立练习规范化的过程。注意，计算得到的列不需要存储。但是，要小心存储运输成本和折扣，因为它们可能是每次单独商议确定的。

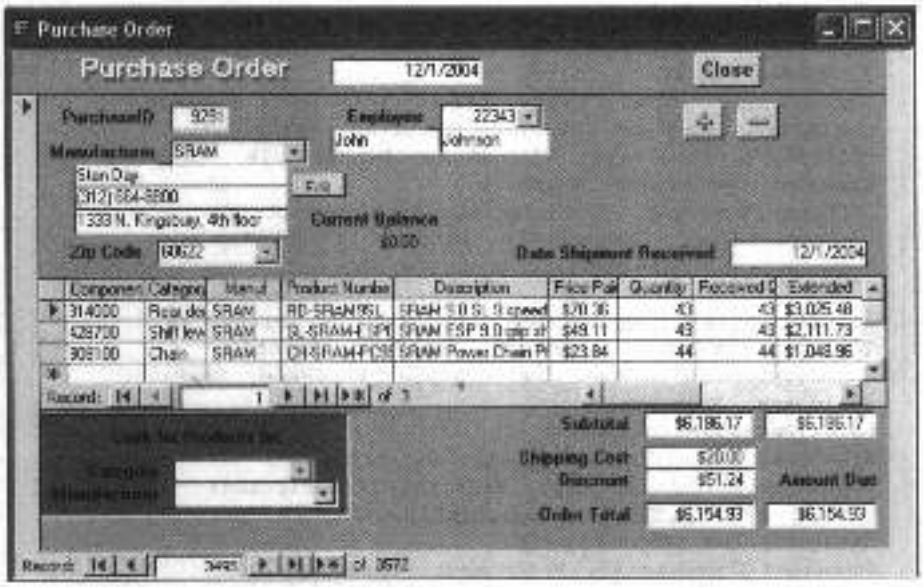


图3-52　Purchase Order表单。只有订购的商品是一个重复组。Look for Products部分为用户提供便利，它并不存储数据。Date Shipment Received框初始时是空白的，当产品到达码头时才被填充

给生产厂的付款记录在图3-53所示的基本交易表单中。注意，初始余额和不足额由表单所支持的代码计算得到，用来显示添加新交易带来的影响。购买的数据条目由Purchase Order表单自动生成，因此此表单一般用于付款或校正。

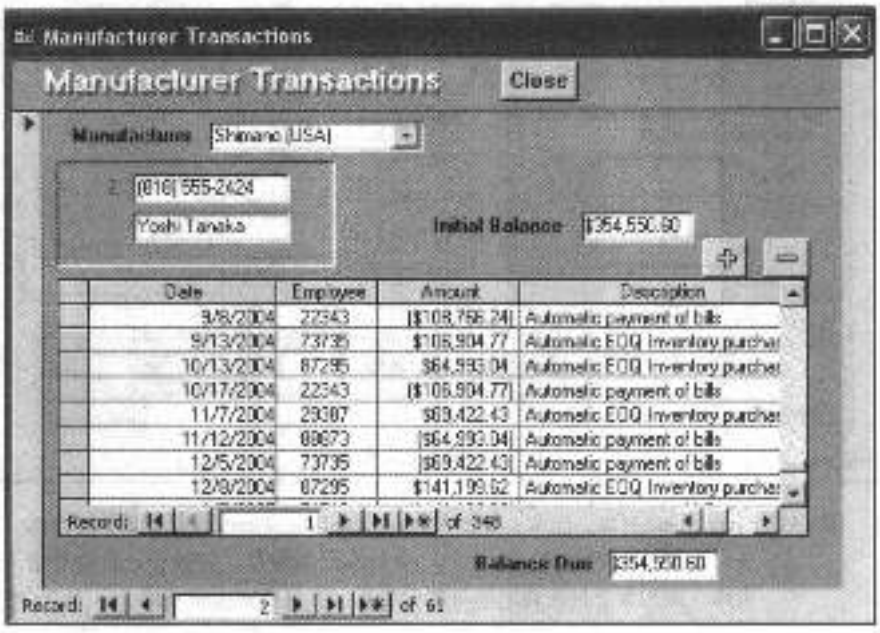


图3-53 Manufacturer Transaction表单。不足额存储在数据库中，但只存储一次。Initial Balance和Balance Due框由表单计算得到，用于显示用户添加事务的影响

生产厂交易的4NF表如图3-54所示。你仍应当自己进行推导。实践和经验是学会规范化的最好方法。不要被误导：读者通常会从书中阅读"答案"并认为规范化很容易。当你自己解决问题时规范化会变得复杂得多，从一开始，调研和确定业务规则是很有挑战性的。

```
ManufacturerTransactions(ManufacturerID, Name, Phone,
Contact, BalanceDue, (TransDate, Employee, Amount,
Description))

Manufacturer(ManufacturerID, Name, Phone, Contact,
BalanceDue)
ManufacturerTransaction(ManufacturerID, TransactionDate,
EmployeeID, Amount, Description)
```

图3-54 从Manufacturer Transaction表单得到的表。这个规范化过程是很直接的。注意，TransactionDate列还存储了时间，因此有可能在同一天当中与一个生产厂进行多次交易

图3-55所示的Component表单用于向列表中添加新零部件和修改零部件的规格描述。它也可以用于修改生产厂的数据。注意两个标识号码的用处：一个由Rolling Thunder分配，另一个由生产厂分配。分配我们自己的号码可以确保数据格式的一致性，保证标识符是惟一的。生产厂的产品号用于下订单，因为生产厂不会使用我们的内部数据。

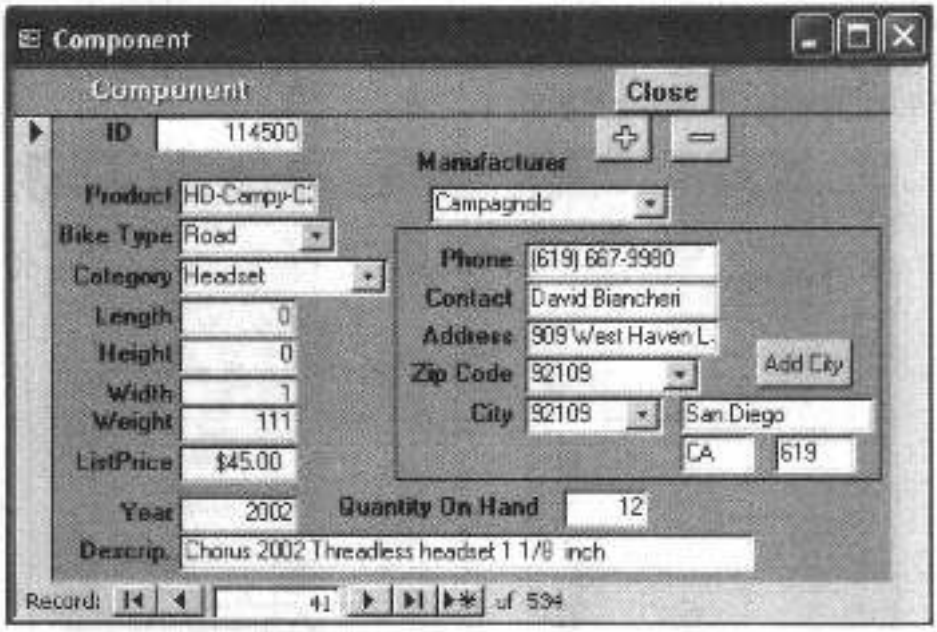


图3-55　Component表单。注意，零部件有一个Rolling Thunder最初赋予的内部编号。产品通常也有一个生产厂分配的产品编号。依赖于这个编号很困难，因为多个供应商之间可能会重复，并且格式也很不同

从Component表单得到的4NF表如图3-56所示。其中绝大部分是很简单的。Rolling Thunder数据库中一个有趣的不同是对地址和城市的处理。很多关于顾客、雇员和供应商等业务表包含城市、州和ZIP编码的列。从技术上，由于这三项是有关联的，在这些基本数据中含有一个隐含约束。因此，数据库可以通过设置一个单独的City表来节约空间和数据输入时间。当然，一个覆盖整个美国的City表，尽管与整个世界相比小得多，但已经很庞大了。更困难的问题在于城市和ZIP编码之间没有一对一的关系。有些城市具有很多ZIP编码，而有些ZIP编码覆盖多个城市。Rolling Thunder通过维护一个基于惟一CityID的City表来解决这两个问题。如果空间非常宝贵，这张表可以缩减到只包含数据库中用到的城市。当有新城市的客户到来时，只需添加基本的城市数据。ZIP编码问题通过为每个城市存储一个基本ZIP编码来解决。与每个地址相关的特定ZIP编码存储在适当的表中（例如Manufacturer）。这个特定ZIP编码还可以是能更确切指出客户或生产厂的九位数字编码。尽管创建一个完整的九位数字编码表是有可能的，但表会非常巨大，并容易改变。依赖于九位数字进行邮寄的公司通常要购买包含验证过的数据库的确认软件来核对他们的地址和编码。

再次考虑图3-50、图3-51、图3-54和图3-56中的表。注意每张图中列出的相似表。特别要注意Manufacturer表。注意，重叠的表通常包含来自不同表单的数据。现实当中，尤其对一个设计团队来说，相似的列可能具有不同的名称，因此一定要小心。这一步骤的目标是合并相似的表。最好的方法是从寻找公共主码开始。具有相同主码列的表应该合并。例如，对Manufacturer表的各种变形进行重新创建，如图3-57所示。通过添加其他变形中的Contact和ZipCode列，可以扩充PO表的形式。

```
ComponentForm(ComponentID, Product, BikeType, Category,
Length, Height, Width, Weight, ListPrice, Description, QOH,
ManufacturerID, Name, Phone, Contact, Address, ZipCode,
CityID, City, State, AreaCode)

Component(ComponentID, ProductNumber, BikeType, Category,
Length, Height, Width, Weight, ListPrice, Description, QOH,
ManufacturerID)
Manufacturer(ManufacturerID, Name, Phone, Contact, Address,
ZipCode, CityID)
City(CityID, City, State, ZipCode, AreaCode)
```

图3-56 从Component表单得到的表。Manufacturer表中的ZipCode是该公司特有的（可能是一个九位编码）。City表中的ZipCode是一个基本（五位）编码，作为一个参照点，一个城市通常有很多个这样的编码

```
PO    Manufacturer(ManufacturerID, Name, Address, Phone,
      CityID, CurrentBalance)
Mfg   Manufacturer(ManufacturerID, Name, Phone, Contact,
      BalanceDue)
Comp  Manufacturer(ManufacturerID, Name, Phone, Contact,
      Address, ZipCode, CityID)
```

图3-57 Manufacturer表的多个版本。具有相同主码的表应当合并为一张表。将Contact和ZipCode移到第一张表中意味着可以删除其他两张表。不要把两个名称（CurrentBalance和BalanceDue）误解为相同的列

在合并重复表之后，你应该获得单一系列的包含表单中所有数据的表，如图3-58所示。在此时对你的结果进行双重检查。特别要确认主码惟一以及复合码表示多对多的关系。然后验证3NF规则：每个非主码列是否依赖并且惟一依赖于整个主码。还要寻找隐含依赖，你可能需要将它们显式化。确保通过列中的数据，这些表可以连接到一起。你应当能在所有的表之间连线。现在是绘制一个更加完整的类图的时候了。每一个规范化的表成为一个实体。关系显示这些表如何连接（完整的例子请查看Rolling数据库）。

最后，查看每张表，确定你是否还需要收集更多的数据。例如，Employee表肯定需要更多数据，诸如Address和DateHired。类似地，你会发现ManufacturerTransaction表会使用一个Reference列，当一次交易由Purchase Order表单自动产生时，该列会包含PurchaseID。这一列的作用是作为查账索引，从源头开始跟踪交易的账目。有些人可能会使用日期/时间，但精确到秒一级的交易可能会引发问题。

3.13 数据字典

在收集数据和创建规范化表的过程中，一定要维护一个数据字典记录数据域和你所做的各种假设。*数据字典或数据仓库由元数据组成*，元数据是描述存储在数据库中数据的数据。它一般列出所有的表、列、数据域和假设。将此数据存在记事本中也可以，但存储在计算机中更容易组织。有些设计者创建一个单独的数据库来记录底层的项目数据。特殊的计算机工具如计算机辅助软件工程（CASE）工具有助于软件设计的展开。它们的用处之一是创建、存储

和搜索一个综合的数据字典。

```
Bicycle(SerialNumber, Model, Construction, FrameSize,
TopTube, ChainStay, HeadTube, SeatTube, PaintID, ColorStyle,
CustomName, LetterStyle, EmpFrame, EmpPaint, BuildDate,
ShipDate)
Paint(PaintID, ColorList)
BikeTubes(SerialNumber, TubeID, Quantity)
TubeMaterial(TubeID, Type, Material, Description)
BikeParts(SerialNumber, ComponentID, SubstID, Quantity,
DateInstalled, EmpInstalled)
Component(ComponentID, ProductNumber, BikeType, Category,
Length, Height, Width, Weight, ListPrice, Description, QOH,
ManufacturerID)
PurchaseOrder(PurchaseID, PODate, EmployeeID, ManufacturerID,
ShipReceiveDate, ShippingCost, Discount)
PurchaseItem(PurchaseID, ComponentID, Quantity, PricePaid,
ReceivedQuantity)
Employee(EmployeeID, FirstName, LastName)
Manufacturer(ManufacturerID, Name, Contact, Address, Phone,
CityID, ZipCode, CurrentBalance)
ManufacturerTransaction(ManufacturerID, TransactionDate,
EmployeeID, Amount, Description, Reference)
City(CityID, City, State, ZipCode, AreaCode)
```

图3-58 完整的表。重复表已经合并，规范化（4NF）已经进行了验证。另外，绘制一张类图以确保表能够连接到一起。注意，附加的Reference列存储相应的PurchaseID作为查账索引。注意，有些表（例如，Employee）将来还需要附加数据

3.13.1 DBMS表定义

在定义好逻辑表并了解所有列的域之后，你可以将这些表输入到DBMS中。大部分系统具有图形用户界面，方便表的定义。但是，在有些情况下，你可能不得不使用第4章介绍的SQL数据定义命令。两种情况下，流程是相似的。定义表名，输入列名，为列选择数据类型，然后确定主码。有时主码通过创建单独的索引来定义。有些系统允许你为每列和每张表创建描述。这些描述可能包含对用户的指导，也可能是你的数据字典的扩展，来帮助设计者在将来进行修改。

此时，你应当决定哪些主码希望使用自动标识函数生成。类似地，确定需要计算的列并指出它们所需的计算公式。有些数据库允许将这些计算公式与数据库定义存储在一起；对于其他的数据库，需要将它们写到查询语句中。

你还可以为每列设置默认值来加速数据输入。在音像店的例子中，可以设置一个基本租金的默认值。默认值对于日期尤其有用。大部分系统允许为日期设定一个默认值，自动输入当前的日期。此时，你还应当设置验证规则以强制数据完整性。表定义一结束，你就可以设定关系。在Microsoft Access中，进入关系界面，添加所有表，然后画线来显示连接（很像类图）。一定要选择选复选框，表示“强制参照完整性”，“级联删除”和“级联更新”。在SQL Server和Oracle中，在定义表的时候可以把参照完整性指定为约束。

图3-59显示了Microsoft Access用于定义表的表单。主码的设定可以通过选择适当的行然后点击图标来完成。数据类型的详情诸如字符长度和数据子类型等，可以在表单底部的列表中设定。任何时候都可以修改表。如果表中已经有数据，当你选择一个范围较小的数据类型时可能会丢失某些信息。

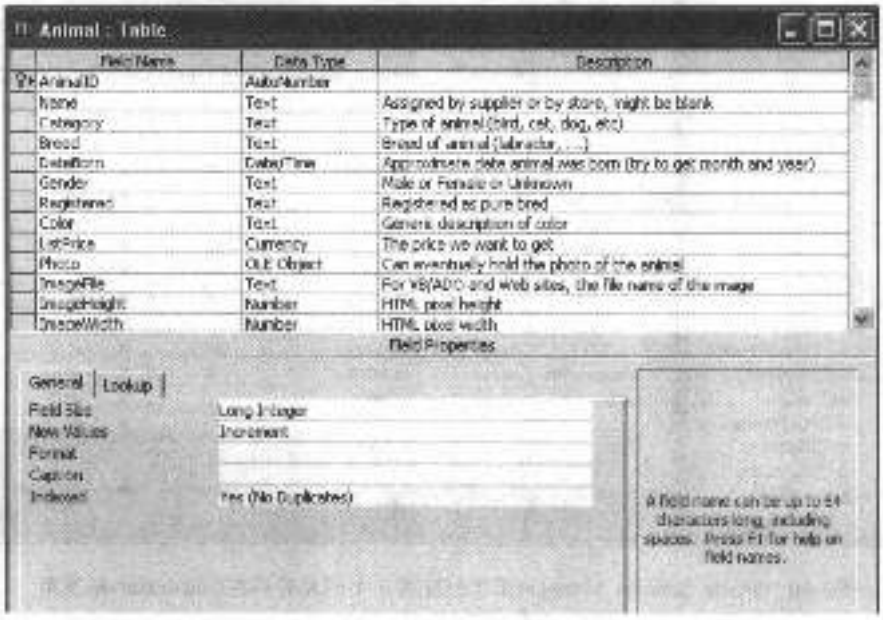


图3-59 Microsoft Access中的表定义。注意主码指示符。另外注意，文本大小和数字的子类型定义在表单底部的列表中

图3-60显示了Oracle中用于创建表的表单，即Schema Manager。使用Constraints标签可以设置主码和外码。记住，一旦在Oracle（和SQL Server）中创建了一张表，以后将很难改变。它允许添加新列，但可能不允许你修改已有列的数据类型或删除列。

如果使用版本号等于或高于8的Oracle，一旦完成建表，你可以执行额外的一个步骤来让Oracle分析表。例如，使用SQL Plus工具执行的命令类似于：Analyze table Animal compute statistic。这些命令告诉Oracle为每张表生成统计信息（注意Schema Manager中的Statistics标签）。Oracle使用这些统计信息能够显著提高查询的性能。

图3-61显示了SQL Server用于创建和编辑表的表单。基本的设计与Micorsoft Access所使用的类似。但关系和约束在Properties表单上指定。主设计表单由Enterprise Manager激活，后者列出了数据库和其中所有的表。

很容易看出各种数据库设计产品的相似之处，而系统之间重要的区别在于各系统内部使用的数据类型。虽然有些数据类型名称相似，但是特别要小心Oracle数据库。它的底层数据类型与其他的不同——尤其对于数字来说。对于Oracle和SQL Server，图形表单似乎容易使用；但是，有经验的开发者几乎总是依靠存储在文本文件中的SQL 语句来创建表。图3-62给出了Animal表的示例。你可以建立所有表的创建语句并将它们存储在一个文本文件中。数据库的SQL处理器阅读此文件并创建表。文本文件相比图形界面方法具有很多优点：（1）修改文本文件更容易。（2）在不同数据库或不同计算机中重新建表更容易。（3）修改表的定义通常需要创建新表，将

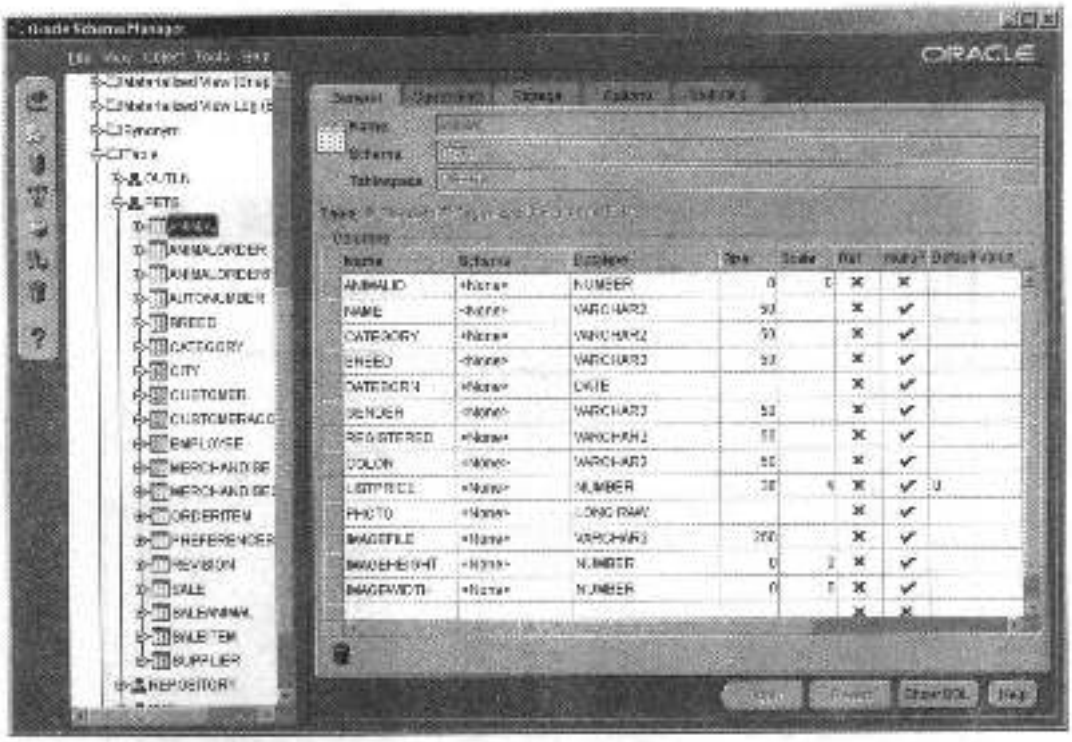


图3-60 Oracle Schema Manager用于创建表。主码和外码在Constraints标签里设定。对于主码列，确保检验值不能为空

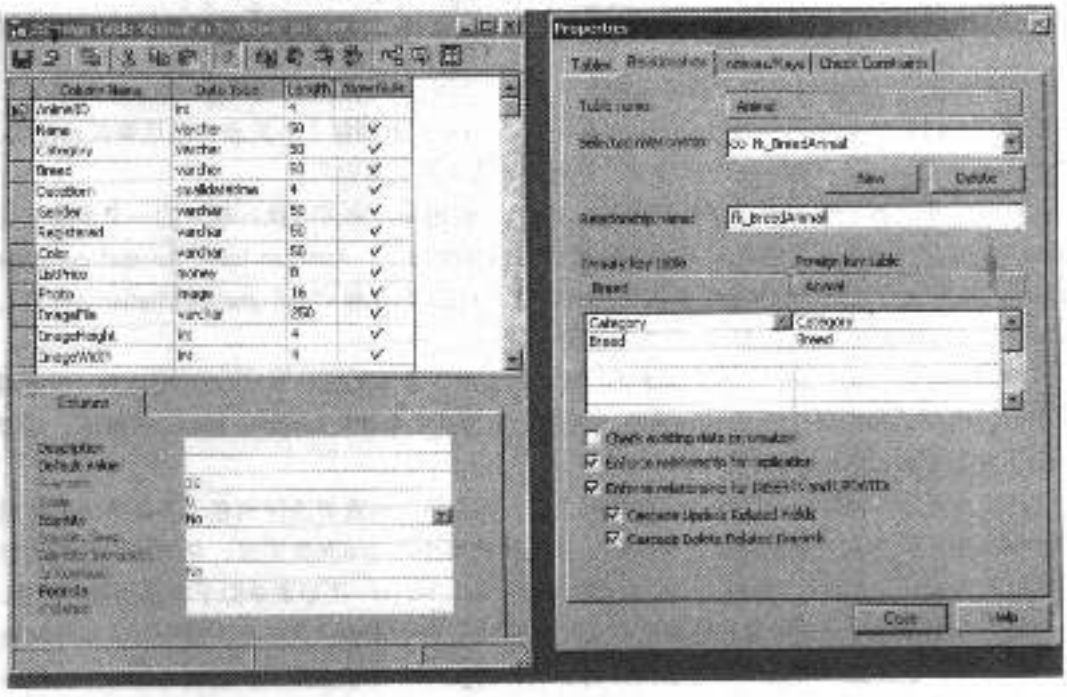


图3-61 Microsoft SQL Server中用于创建表的表单。注意它与Microsoft Access设计表单的相似性。但是关系和约束在Properties表单中指定

贝已有数据，删除旧表和重命名新表。使用文本文件，可以快速定义新表并执行语句创建新表。(4) 使用文本文件更容易设定主码和外码关系。大多数图形方法笨重而且难懂。而且，有些版本的Oracle对使用图形界面具有限制，而通过直接使用SQL语句则可以避免这些限制。(5) 由于外码约束，创建表的顺序是至关重要的。必须先创建一个表，然后才能在外码中引用它。例如，Category表和Breed表必须在创建Animal表之前创建。在文本文件中记录表的定义意味着你只需一次设定次序即可。如果你对SQL的建表语法不熟悉，可以查看已有的结构文件，或使用Schema Manager输入基本信息，然后点击Show SQL按钮,剪切并粘贴基本的SQL代码。

```
CREATE TABLE Animal
(
  AnimalID       INTEGER,
  Name           VARCHAR2(50),
  Category       VARCHAR2(50),
  Breed          VARCHAR2(50),
  DateBorn       DATE,
  Gender         VARCHAR2(50)
    CHECK (Gender='Male' Or Gender='Female'
      Or Gender='Unknown' Or Gender Is Null)
  Registered     VARCHAR2(50),
  Color          VARCHAR2(50),
  ListPrice NUMBER(38,4)
    DEFAULT 0,
  Photo          LONG RAW,
  ImageFile      VARCHAR2(250),
  ImageHeight    INTEGER,
  ImageWidth     INTEGER,
    CONSTRAINT pk_Animal PRIMARY KEY (AnimalID)
    CONSTRAINT fk_BreedAnimal FOREIGN KEY (Category, Breed)
    REFERENCES Breed(Category, Breed)
    ON DELETE CASCADE
  CONSTRAINT fk_CategoryAnimal FOREIGN KEY (Category)
    REFERENCES Category(Category)
    ON DELETE CASCADE
);
```

图3-62 创建Animal表的Oracle SQL语句。用于SQL Server的语句与此类似，只需修改数据类型

创建Oralce表时，需要知道另外一个关键的功能。一旦完成建表，需要让Oracle对表进行分析并收集关于该表的统计信息。首先，使用SQL建表。第二，向表中装载现有数据。第三，使用SQL Plus工具为每张表执行分析命令。例如：Analyze table Animal compite statistics。

3.13.2 数据量与使用率

设计数据库的下一步是估算数据库的大小。这个过程比较简单，但必须向用户询问很多问题。设计一个数据库时，估算数据库的大小和使用率是很重要的。估算的结果能使你估计硬件需求和系统的成本。第一步是估算表的大小。一般来说，需要调查三种情形：数据库现在多大？2~3年后数据库会多大？10年后数据库会多大？

从规范化表开始。这个过程包括估算表中每行的平均字节数和估算表中的行数。将这两个数相乘，得到表大小的估算值，然后将所有表的规模相加，得到数据库总规模的估算值。这个数字代表数据库的最小规模。很多数据库会比这个基本估算值大3到5倍。有些系统具有更复杂的规则和估算流程。例如，Oracle提供了一个工具来帮助估计数据库所需的存储空间。仍从每列的数据类型和近似的行数开始。这个工具使用关于Oracle流程的内部规则来辅助估算所需的总存储空间。

一个估算数据量的例子如图3-63所示。考虑Customer表，数据库系统为每列数据留出一定大小的存储空间，这个值取决于特定系统。因此请参考文档以得到精确的值。在简化的Customer表中，长整型的标识数字占4字节，估计Names平均占15字符。其他平均值显示在表中。更好的估算可从对示例数据的统计分析获得。无论如何，每行Customer数据的估算值是76字节。估计此业务拥有大约1 000个客户。因此，Customer表约为76K字节。

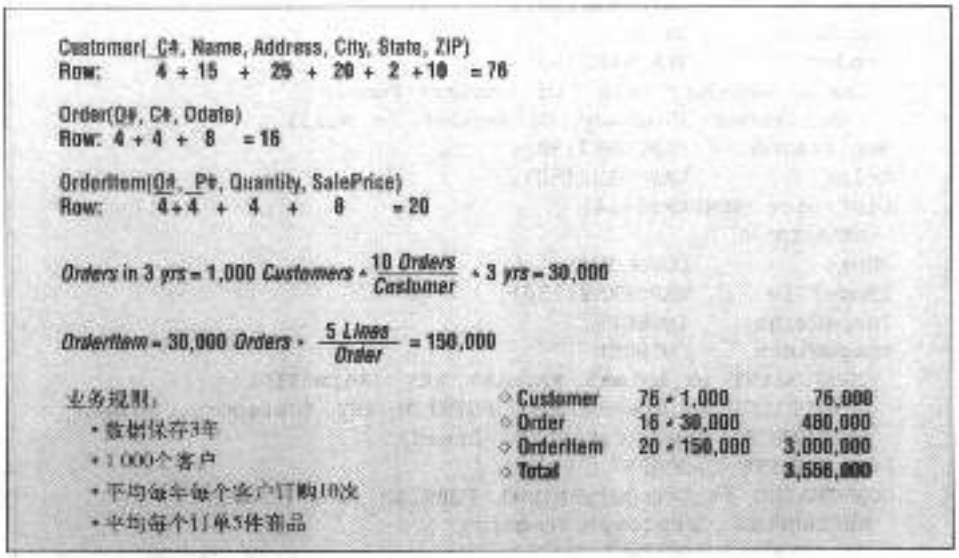


图3-63　估算数据量。首先估算每行的大小，然后估算表中的行数。如果存在连接码，通常会用平均值乘以主码表中的行数，就像计算OrderItem一样

对Order表大小的估算遵循类似的过程，得到的估算结果是每行16字节。经理可能知道某1年的订单数目。但是，可能更容易获得某1客户1年内的平均订单数。如果这个数是10，那么你可以认为1年会有10 000个订单。类似地，要获取OrderItem表中的行数，需要了解平均每张订单订购的产品数目。如果那个数是5，那么，可以认为1年后OrderItem表将会有150 000行。

下一步是估计数据存储的时间长度。一些公司计划将数据联机保存多年，而另一些则遵循严格的保留和删除策略。合法用途的数据必须保存一定年份，由它的种类决定。记住，诸如IRS等代理还要求还原软件（例如DBMS）能够重建数据。

除了基本的数据存储，数据库还要为索引、日志文件、表单、程序和备份数据保留空间。拥有特定数据库系统的经验将提供更具体的估计，但最终的总计可能是基本估计大小的3~5倍。

最终的估计结果对支持数据库所需要的硬件提供参考意见。尽管性能和价格在不断变化，仍然只有小型数据库能够在个人计算机上高效运行。较大的数据库可以放到局域网（LAN）

内的一台文件服务器上。LAN提供数据的多用户访问，但性能很大程度取决于数据库的大小，DBMS的特征以及网速。随着数据库规模的增长（几百或几千兆字节），有必要转移到专门的计算机来处理数据。超大规模数据库（万亿字节）需要多台计算机和特殊的磁盘驱动器来将容量和性能瓶颈最小化。数据估算不必非常精确，但它们能提供基本的信息给计划委员会以便申请用于开发、硬件、软件和人员的资金。

当与用户谈论每张表时，应当让他们指出一些基本的安全信息。你最后需要赋予每张表安全访问的权限。第10章将具体介绍，但目前应该搞清楚哪些人使用表，以及哪些人不应当授予某些权力。例如，订购商品的职员不应允许确认接收该商品，否则，一位不道德的职员会订购商品，确认接收，然后偷走它。可允许数据四种基本操作：读取、修改、删除或添加新数据。应当记录谁可以或不可以访问某张表。

小结

数据库设计依赖规范化或将数据划分到表的过程。基本上，每张表涉及一个单独的实体或概念。每张表必须拥有能惟一标识每行数据的主码。你从数据的集合开始建表，数据集合一般由用户表单或报表得到，通过寻找数据中的重复组并将它们放到一个单独的表中，就可以满足1NF的要求。接着，遍历每张中间表，确定主码，通过检查每个非主码列并判断它是否依赖于整个主码来满足2NF的要求。如果不依赖于整个主码，就将这列连同它所依赖的部分主码移出放入一个新表。要达到3NF，检查每个非主码列是否依赖于任何非主码的列。如果是这样，那么将此列和它所依赖的列移到一个新表中。BCNF和4NF涉及主码当中的类似问题，特别希望查找主码中的隐含依赖，如果找到了，则创建一个新表使这种依赖显式化。

从用户那里收集的每个表单、报表或描述，必须进行分析并推导得到满足4NF的表定义集合。对于大型项目，若干分析师可能分析不同的表单，产生若干系列规范化的表。这些表必须集成为一个标准的规范化表的集合。在这个过程中必须为每一列指定域或者数据类型。这些最终的表连同注释将输入到DBMS中，以创建数据库。

还应当根据每张表的行数对数据量进行估算。这些数字使你能够估算数据库的平均大小和最大规模，以便你能选择适当的硬件和软件。还应当收集关于安全条件的信息：谁拥有数据？谁可以进行读访问？谁可以进行写访问？所有这些条件都在创建表的时候输入到DBMS中。

在回顾你的工作时，可以输入示例数据来测试你的表。当确定设计完整和精确之后，就可以通过创建查询、表单和报表来建立应用了。

开发漫谈

Miranda明白了要将要转化为规范化表的集合。这些表是数据库应用的基础。数据库设计对于应用开发是至关重要的。一定要牢记基本的规范化规则：每个非主码列依赖且仅依赖于整个主码。由于设计依赖于业务规则，因此，一定要理解这些规则。仔细倾听用户的要求，拿不准的时候，选择灵活的设计方案。至于你的课堂项目，你现在应当能够创建一系列规范化的表。你还应当能够估算数据库的规模。

关键词

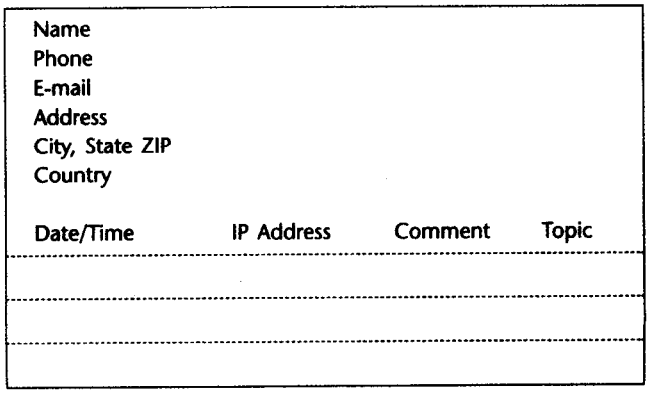
自动标识
Boyce-Codd范式（BCNF）
级联删除
复合码
数据字典/仓库
数据完整性
数据量
元数据
删除异常
依赖
域-码范式(DKNF)
第一范式（1NF）
外码
第四范式（4NF）
默认值
插入异常
主从关系
参照完整性
重复组
第二范式（2NF）
第三范式（3NF）
隐含依赖

复习题

1. 什么是主码，为什么需要它?
2. 什么是复合码?
3. 规范化的主要规则是什么?
4. 如果数据没有存储到规范化的表中，会遇到什么问题?
5. 解释短语：表中的一列依赖于另一列。
6. BCNF和4NF如何不同于3NF?
7. 能够存储到表中的主要数据类型是什么?
8. 给出一个作为数据完整性存储规则的例子。
9. 当集成视图时需要寻找什么元素?
10. 如何估算数据库的潜在规模?
11. 参照完整性为什么很重要?
12. 设置参照完整性规则会引起什么复杂情况?

练习

1. 本地一家零售商店正在建立一个网站，想请你创建一个数据库用来记录需求信息。公司想收集基本客户数据，当用户反馈对主题发表评论时进行记录。主题是预先定义好的项目列表（经理可以修改），用户可以通过选择框进行选择。为这个项目定义需要的规范化表。为此例创建类图和规范化的表。



2. 你被雇用来为公司开发一个小型数据库以提供公司产品的网上销售。为此例创建类图和规范化的表。

Date/Time	**Order Form**							
Customer			Credit Card			Internet		
Name Phone	Shipping Address City, State ZIP		Card # Expiration Date Bank			E-mail IP Address Referred From		
Items								
Item #	Name	Description	Quantity	List Price	Sale Price	Quantity Shipped	Back Order	Extended
						Item Total Shipping Tax Total Due		

3. 本地一家公司的人力资源部分需要帮助。经理想要为员工提供自助类型的福利，使雇员可以选择他们想要的福利组合。公司出固定数量的钱，如果所选项超过限制，雇员需要支付差价。人力资源部门还希望记录每个雇员工作的列表。当前，这些信息通过与下图相似的表单来收集。为此例创建类图和规范化的表。

Employee Last Name, First Name Office, Phone Date Hired					
Benefit	Date	Monthly Cost			
Job/Title	Location	Start Date	End Date	Salary	Supervisor

4. 你的朋友开了一家车库乐队。乐队已经写了若干首歌并且在多家夜总会演出。为改进演出

和乐队，她想要一个数据库来记录演奏了哪些歌和大家的反响如何。乐队成员也想记录每首歌是谁演奏的并指出是否存在问题（或优点）以便他们知道该做什么。数据库必须易用，她草拟了一张示例表单。为此例创建类图和规范化的表。

Gig			
Date	Location		
Start Time	End Time		
# People	Money		
Comments			
Song		Person	Instrument　Comment
Length	Actual		
Author			
Style	Key		
Comments			
Song		Person	Instrument　Comment
Length	Actual		
Author			
Style	Key		
Comments			

5. 一家草坪护理公司需要记录客户的任务及雇员。现在，店主将每天的记录记在一沓纸上。与这里所示的表单类似，它根据时间列出工作。雇员可以为每位客户完成多项任务，包括割草、修边和剪枝。为此例创建类图和规范化的表。

Lawn Care				
Date				
Time	Customer, Address			Total
	Task	Employee	Time	Charge

6. 本地一家乡村俱乐部拥有漂亮的新俱乐部会所，它出租给个人或公司用于小型聚会或招待会。俱乐部会所有四个不同大小的房间，可以单独出租，但经理一般不喜欢同时承接两个以上大型活动。客户必须提前预订，然后根据预计客人数量选择房间。客户可以选择点餐或吃自助餐。餐费取决于所点的项目。如示例表单，只要另付费用，客户可以选择停车服务和俱乐部提供的鲜花。酒吧服务与用餐选项类似。为此例创建类图和规范化的表。

Reservation Form	
Date Room/Hall	Employee Maximum Capacity
Cancel Date Total Due Deposit	Event Date, Time, Length Event Title ☐ Valet Parking (cost:) ☐ Flowers (cost:)
Lunch (sit down) # Guests # Servers Menu Item Description Cost Quantity Subtotal	
Dinner (sit down) # Guests # Servers Menu Item Description Cost Quantity Subtotal Buffet ... Bar ...	

7. 一家小型木工店专门制造大座钟。这家店从不同的供应商那里订购木材、发条和其他零部件。木板经过刨平粗木、上胶、成型和装配得到。顶部的装饰雕刻从一个单独的供应商那里购买，在那里用手工完成这些雕刻。有些钟表由客户订购，客户可以选择诸如高度和发条等。对于一般的订单，店主通常填写与图中所示相似的表单来进行记录。时钟供不应求，店主故意限量生产，因此价格一直很高。根据订单和销售表单，为此例创建类图和规范化的表。注意，Sale表单上的项目总计不等于总金额，因为总金额还要包括一些额外的费用，但不包括邮费。尽管在表单中没有显示，店主还希望记录每个时钟开始制造和完成制造的日期。

Sale				Employee
Order Date Estimated Delivery Date Actual Delivery Date				Customer Phone Address
Total Price Delivery Method	Payment Method Delivery Charge			City, State ZIP
Item/Feature	Color	Quantity	Price	Subtotal
				Item Total

Purchase Order

Supplier Order Date
Contact Phone Employee
Estimated Ship Date
Date Received

Item	Quantity	Cost	Quantity Received	Clock (custom orders)

Total Charges Date Due
Date Paid Amount Paid

8. 本地一家比萨店需要一个数据库记录客户订购和送货信息。基本的订购表单和比萨送到时司机需要填写的一张表单如下图所示。司机被鼓励记下用户关于送货的评论或订单，对下次送餐的司机可能有用。必须记录小费，因为他们要向IRS报告。为此例创建类图和规范化的表。

Pizza Order

Customer Order Date/Time
Phone Employee
Address

For each pizza:
Crust Type
Size
Base Price
Specialty Pizza

Custom Toppings

Item	Cost	Comments (e.g., half/half)

Topping Cost
Tax
Discount/Coupon
Total

Delivery Time Employee Directions Payment Method Driver Tip Comments (e.g., dog)	Credit Card #	Expiration

9. 一家制造动画片的公司想要你建立一个数据库记录制作动画片的过程。依据附带的表单，工作主要集中在记录单独的角色和场景上。首席美工绘制一种角色并将基本数据进行存储，以便其他美工可以获取并在他们的场景中应用。公司还要求员工记录制作各种场景所需的时间。为此例创建类图和规范化的表。

Character				
Name Description Relative Size			Artist in Charge	
Character Views: digitized and stored				
Front	Left	Right	Rear	Top

Scene		
Sequence/Position Lead Writer Synopsis		
Character View Role Action Description		

Work Status				
Date		Employee		
Time	Scene	Character	Changes/Work	Amount of Time

10. 一家制作带设计师签名的牛仔裤的公司遇到了困难。经理需要让生产和订购数量相匹配，并且确保产品按时发给客户。由于每种款式的牛仔裤会有很多种尺寸，而且客户有很特殊的订单，因此问题变得更加困难。订单可能会很大，通常由多家制造厂共同完成，因此完成一个订单需要多次送货。订单、送货单和生产表单显示了所需的主要项目。每家制造厂有多道工序。一条生产线包括大约5个雇员，完成牛仔裤的不同部分。为此例创建类图和规范化的表。

Order

Order ID　　Order Date
Customer PO　　Exp. Delivery Date
Customer Contact　　Phone

Style	Name	Size	Description	List Price	Sale Price	Value
						Order Total

Delivery

Order ID　　Delivery Date
Factory: Location　　Manager　　Phone
Shipping Costs

Style	Color	Size	Quantity

Factory Production

Address　　Maximum Capacity
City, State ZIP
Country　　Date

Shift	Line	Style	Gender	Quantity	Defective	Hours	# People

Sally的宠物商店

11. 定义需要的表扩展宠物商店数据库以处理动物的谱系记录。
12. 定义需要的表扩展宠物商店数据库以处理动物的健康和兽医的记录。
13. Sally想在数据库中增加薪水和月度雇员评价信息。定义需要的表。

14. Sally想添加宠物训练服务。定义记录预约计划所需的表，假设有两个员工负责这项工作。

Rolling Thunder自行车

15. 使用类图，指出表定义和表关系所描述的五种业务规则（与音像店示例中描述的RentPrice规则相似）。
16. 公司想为人力资源添加更多的数据，诸如扣税、所选福利以及雇员和公司对福利的支付情况。研究处理这种类型数据的通用方法，定义所需要的表。

参考网站

网站	描述
http://www.intelligententerprise.com/info_centers/database	数据库杂志
http://www.for.gov.bc.ca/isb/datadmin/	Canadian Ministry of Forests数据管理网站，有很多关于数据管理和设计的有用信息。从开发标准开始。
http://support.microsoft.com/support/kb/articles.q209/5/34.asp	规范化的介绍
http://www.phpbuilder.com/columns/barry20000731.php3	规范化示例

补充读物

Date, C. J. *An Introduction to Database Systems,* 8th ed. Reading: Addison-Wesley, 2003. [A classic higher-level textbook that covers many details of normalization and databases.]

Diederich, J., and J. Milton. "New Methods and Fast Algorithms for Database Normalization." *ACM Transactions on Database Systems* 13, no. 3 (September 1988), pp. 339–65. [One of many attempts to automate the normalization process.]

Fagin, R. "Multivalued Dependencies and a New Normal Form for Relational Databases." *ACM Transactions on Database Systems* 2, no. 3 (September 1977), pp. 262–78. [A classic paper in the development of normal forms.]

———. "A Normal Form for Relational Databases That Is Based on Domains and Keys." *ACM Transactions on Database Systems* 6, no. 3 (September 1981), pp. 387–415. [The paper that initially described domain-key normal form.]

Kent, W. "A Simple Guide to Five Normal Forms in Relational Database Theory." *Communications of the ACM* 26, no. 2 (February 1983), 120–25. [A nice presentation of normalization with examples.]

Rivero, L., J. Doorn, and V. Ferraggine. "Elicitation and Conversion of Hidden Objects and Restrictions in a Database Schema." *Proceedings of the 2002 ACM Symposium on Applied Computing,* 2002, 463–69. [Good discussion of referential integrity issues and problems with weak designs heavily dependent on surrogate ID columns.]

Wu, M. S. "The Practical Need for Fourth Normal Form." *Proceedings of the Twenty-third SIGCSE Technical Symposium on Computer Science Education,* 1992, 19–23. [A small study showing that fourth normal form violations are common in business applications.]

附录：规范化的形式化定义

关系数据库模型的优点之一是从集合论的数学基础上发展而来。尽管不需要了解形式化的定义，但有时能使你更好地理解整个过程。要更详细地了解范式及其复杂性，应当阅读C. J. Date的高级教程。记住，形式化定义使用专业术语。图3-1A列出了主要术语和它们的一般解释。尽管形式化术语更精确，但很少有人对术语有标准的理解，因此，在大部分交流中使用非形式化术语更好一些。

形式化	定义	非形式化
Relation	数据随时间不断变化的属性集合。通常记作R	Table
Attribute	现实世界中域的属性。属性的子集是多个列，通常记作X或Y	Column
Tuple	特定属性集返回的数据，通常记作t[X]	Row of data
Schema	表和约束/关系的集合	
Functional dependency	X → Y	Business rule dependency

图3-1A　术语。形式化术语更精确并且有数学上的定义，但是很难让开发者和用户理解

1 初始定义

关系是数据随时会改变的属性集合。每个**属性**具有相应的域并且涉及现实世界中的某种特性。形式化定义指属性的子集，是列的集合。属性X的特定子集返回的元组中的数据值记为t[X]。

规范化的本质是承认属性集具有某些现实世界的关系。其目标是精确地描述这些关系约束。这些语义上的约束称为**函数依赖**（FD）。

(1) 定义：*函数依赖和决定因素*

若X和Y是属性的子集，则函数依赖记为X→Y，当任意数据行具有相同的X属性值时，总会有相同的Y属性值。即，对于关系R的元组t1和t2，如果t1[X]=t2[X]，那么t1[Y]=t2[Y]。在一个FD中，X也称为**决定因素**，因为一给定X属性的值，根据依赖，就能确定Y的属性值。

主码对于关系数据库很重要，因为它们用于确定数据行。有时多个属性的集合可以用来构成不同的主码，因此它有时作为**候选主码**。

(2) 定义：*主码*

主码是属性的集合K，若U是关系中所有属性的集合，则

1) 存在一个函数依赖K→U

2) 如果K'是K的子集，那么不存在FDK'→U

即，主码的属性集K能在函数上确定关系中的其他所有属性，并且它是具有这种关系的最小集

合（不存在更小的能决定其他属性的集合K）。

2 范式定义

第一范式的定义与原子属性的定义紧密相关，因此需要同时定义。

(1) 定义：第一范式（1NF）

一个关系满足**第一范式**当且仅当它的所有属性都是原子的。

(2) 定义：原子属性

原子属性是单独赋值的，这意味着不能是复合的、多值的或嵌套的关系。

本质上，一个1NF关系是一张表，它的每一属性列都是简单的单元。不能使用技巧试图向一列中压入额外的数据、其他关系或者多个列。图3-2A提供了一张表的例子，它不满足第一范式，因为它有两个属性不是原子的。

第二范式依据主码和函数依赖来定义。

(3) 定义：第二范式（2NF）

一个关系满足**第二范式**，如果它满足第一范式，并且每个非主码属性完全函数依赖于主码。即，对于每个非主码属性Ai，有K→Ai。因此，不存在子集K'，使得任何属性满足K'→Ai。

这个定义与本章介绍的简单定义相符：每个非主码列依赖于整个主码，而不是主码的一部分。图3-3A显示了一个不满足第二范式的关系的例子。

Customer(CID, Name: First + Last, Phones, Address)

CID	Name: First + Last	Phones	Address
111	Joe Jones	111-2223 111-3393 112-4582	123 Main

图3-2A 非原子属性。这张表不满足第一范式，因为Name属性是两个属性元素的组合，Phones属性用来处理多个值

OrderProduct(OrderID, ProductID, Quantity, Description)

OrderID	ProductID	Quantity	Description
32	15	1	Blue hose
32	16	2	Pliers
33	15	1	Blue hose

图3-3A 非完全依赖。产品描述只依赖于ProductID，而非整个主码{OrderID，ProductID}，因此这个关系不满足第二范式

第三范式的形式化定义有点难理解，因为它依赖于一个新概念：传递依赖。

(4) 定义：传递依赖

已知函数依赖X→Y和Y→Z，则**传递依赖**X→Z也成立。

传递的概念在初等代数中很常见。它起源于集合论。要理解这个定义，需要记住，函数依赖表示业务语义关系。考虑OrderID、Customer ID和客户Name属性之间的关系。每张订单只能有一个客户的业务规则翻译为函数依赖是OrderID→CustomerID。只要知道OrderID的值，总能知道CustomerID的值。同样，CustomerID和其他属性诸如Name之间的主码关系意味着存在一个函数依赖CustomerID→Name。根据传递性，只要知道OrderID的值，就能获得CustomerID的值，进而得到客户Name的值。

(5) 定义：第三范式（3NF）

一个关系满足**第三范式**当且仅当它们满足第二范式，并且没有传递依赖于主码的非主码属性。即，已知第二范式：对于每个属性Ai有K→Ai，那么不存在属性子集X使得K→X→Ai。

用简单一点的术语，每个非主码属性依赖于完整的主码，而不依赖于某中间属性。图3-4A显示一个普通的不满足第三范式的关系示例，因为客户属性传递依赖于CustomerID。

Order(OrderID, OrderDate, CustomerID, Name, Phone)

OrderID	OrderDate	CustomerID	Name	Phone
32	5/5/2004	1	Jones	222-3333
33	5/5/2004	2	Hong	444-8888
34	5/6/2004	1	Jones	222-3333

图3-4A　传递依赖。客户Name和Phone属性传递依赖于CustomerID，因此这个关系不满足第三范式

正如第3章讨论的，Boyce-Codd范式更难理解。它和形式化定义表示相同的基本问题：移除隐含依赖。

(6) 定义：Boyce-Codd范式（BCNF）

一个关系满足Boyce-Codd范式当且仅当它满足第三范式，并且每个决定因素都是一个候选主码。即，对于每个属性都有K→Ai，并且不存在子集X（主码或非主码）使得X→Ai且X与K不同。

如图3-5A所示的例子，考虑雇员有很多专业，每个专业有很多雇员，而且一个雇员可以有很多经理，但每个经理只是一个专业的经理。函数依赖（ManagerID→Specialty）不是关系EmpspecMgr（EID，Specialty，ManagerID）的主码，因此，关系不满足BCNF。必须分解，创建新的关系ManagerSpecialty（ManagerID，Specialty）和EmployeeManager（EmployeeID，ManagerID），新关系的每个函数依赖都作为主码。

例：雇员可以有很多专业，每个专业可以有很多雇员。雇员可以有很多经理，但每个经理只是一个专业的经理：Mgr → Specialty

EmpSpecMgr(EID, Specialty, ManagerID)

EID	Specialty	ManagerID
32	Drill	1
33	Weld	2
34	Drill	1

FD ManagerID → Specialty is not currently a key.

图3-5A　Boyce-Codd范式。注意，存在从ManagerID到Specialty的函数依赖。因为这个FD不是关系中的候选主码，它是隐含的，因此这个关系不满足Boyce-Codd范式

第四范式的定义需要技巧，但幸运的是它在实际中并不常见。但是，如果它出现了仍会引起问题，因此，你应当理解它的定义。第四范式与多值依赖的定义密切相关。

(7) 定义：多值依赖（MVD）

当一个关系中至少存在三个属性（A、B和C，也可能是属性集），并且属性A决定其他两个（B和C），但另外两个属性相互独立时，就存在**多值依赖**（MVD）。即，A→B且A→C，但B和C在函数上并不相互依赖。

例如，雇员可以有很多专业并且被分配很多工具，但工具和专业并不直接相关。

(8) 定义：第四范式（4NF）

一个关系满足**第四范式**，当且仅当它满足Boyce-Codd范式并且不存在多值依赖。即，关系的所有属性函数依赖于A。如果A→→B，那么对于所有属性Ai，有A→Ai。

在雇员专业和工具的多值依赖例子中，关系EmpSpecTools（EID，Specialty，ToolID）不满足第四范式，因为存在两个函数依赖：EID→Specialty和EID→ToolID。解决这个问题得到两个更简单的关系：EmployeeSpecialty（EID，Specialty）和EmployeeTools（EID，ToolID）。

第二部分

查　询

在构造应用软件的过程中，为了准确地查出想要的数据，创建查询是一个重要的步骤。查询可以回答业务问题，并用于表单和报表。

第4章介绍如何使用两个基本的查询系统：SQL和QBE。SQL作为标准的优势在于许多数据库管理系统支持它。学会SQL就能够使用许多不同的系统。

第5章介绍SQL查询的部分强大功能。特别是，检查子查询来回答复杂的业务问题。此外，SQL是一个完备的数据库语言，可以用于定义新的数据库和表。SQL还是一个操作数据的强大工具。

第 4 章

数据查询

本章学习内容

- 如何从数据库得到商务查询的答案?
- 为了使用数据库系统，为什么需要学习一种特殊的查询语言?
- 为了创建一个查询，需要回答哪四个问题?
- 如何利用查询执行计算?
- 如何利用分组比较结果?
- 为什么在查询中要连接表? 如何连接表?

4.1 开发漫谈

Miranda：哇！工作太累了！我真希望以后规范化工作可以简单些。

Ariel：你现在至少有了好用的数据库。下一步呢? 你准备好开始构造应用程序了吗?

Miranda：还没有。我告诉我的叔叔我有一些样例数据。他已经问我具体问题，例如：哪种产品订货最频繁? 上个月哪个员工卖得最多? 我认为在构造应用程序之前，应该知道如何回答这些问题。

Ariel：难道不能通过观察数据得到答案吗?

Miranda：也许，但那样太费时。相反，我要使用一个查询系统帮我做大部分工作。只要知道如何把具体问题表达为正确的查询就可以了。

4.2 简介

为什么需要查询语言? 为什么不使用自然语言，比如英语? 自然语言处理器已经改善，有的公司试图把它和数据库结合。类似地，语音识别也在改进。其结果是，计算机有可能用自然语言回答特定的问题。然而，即使有优秀的自然语言处理器，最好还是使用专业的查询语言。主要的原因是交流。如果你问一个关于数据库的问题，一台计算机，或者另外一个人，都可以得到答案。问题是，计算机给出的答案是你要的吗? 换句话说，你必须知道计算机（或者其他人）按照你希望的方式理解问题。使用自然语言的问题就是具有歧义性。如果在理解上存在任何疑问，就要冒着风险，得到一个看上去合理，但实际上并不对应你的问题的答案。

一个查询系统比自然语言更加结构化，可以减少歧义。查询系统还可以变得更加标准化，开发者和用户可以学习一种语言并且在不同的系统中使用。SQL是一种标准的数据库查询语言。

国际标准化组织（International Organization of Standards，ISO）建立的标准，每隔几年更新一次。大部分数据库管理系统至少支持SQL-99标准，但是一小部分使用SQL-92版本。ISO工作组正在开发新标准，但是还需要几年才能完成。虽然大部分厂商支持这些标准，但是仍然存在不同的SQL语法，所以针对一个数据库系统编写的查询语句在别的系统可能就会出现问题。

大部分数据库系统也支持QBE方法（示例查询，QBE），帮助初学者创建SQL查询。这些面向可视化的工具通常允许用户从列表中选择项，并且处理语法细节，使得创建特定的查询更为容易。虽然QBE设计很容易，并且尽量减少了输入从而节省时间，但学习SQL命令仍然是必须的。很多时候，需要在程序代码中插入SQL语句，或者复制和编辑SQL语句。

既然和查询打交道，就需要全面地考虑数据库设计。第3章介绍如何规范地把数据分解为表，实现数据的高效存储。查询是问题的另一个方面：把表连接，从而获得特定问题的答案并产生报表。

4.3 查询语言的三个任务

创建数据库并构造应用程序需要做三件事情：（1）定义数据库，（2）改变数据，（3）检索数据。有的系统使用形式化术语描述上述工作。定义数据表和数据库的其他特征的命令组合为*数据定义语言*（DDL）。通用DDL命令包括ALTER、CREATE和DROP。用来修改数据的命令分类为*数据操纵语言*（DML）。常用的DML命令是DELETE、INSERT和UPDATE。有的系统把数据检索也作为DML组，但是SELECT命令非常复杂，需要单独讨论。本章的附录列出了多种SQL命令的语法。本章关注SELECT命令。DML和DDL命令将在第5章详细讨论。

SELECT命令用于检索数据，是最复杂的SQL命令，带有几个不同的选项。SELECT命令的主要功能是选择满足某些条件的特定数据列。

数据库管理系统是由查询系统驱动的。实际上，所有的工作都可以通过DDL、DML或者查询命令来完成。现代系统通过提供图形接口简化了部分数据库管理工作（例如创建表）。接口实际上构造了合适的CREATE TABLE命令。

4.4 检索数据的四个问题

从关系DBMS中检索数据需要回答四个基本问题，如图4-1所示。查询系统的区别是如何回答这些问题。记住这四个问题，但是顺序并不重要。初学者每次创建查询的时候需要写下这四个问题。对于简单的问题，不知不觉地就可以回答它们。对于复杂的问题，可以先部分地回答问题，并且在问题间切换，直到完全理解了查询。

- 你想得到什么结果？
- 已经知道什么（或者有哪些限制）？
- 涉及哪些表？
- 如何连接表？

图4-1 创建查询的四个问题。每次建立查询都要问这四个问题

注意在一些简单的情况下，可能不需要回答所有的四个问题。很多简单问题只涉及一个表，所以不需要考虑表的连接（问题3）。再举一个例子：如果想得到整个公司的销售总额，而非某一个特定员工的总销售额，所以可能没有任何限制（问题2）。

4.4.1 你想得到什么结果

回答这个问题需要从存储在数据库中的多个表中选择数据列。当然，需要知道所有的列名。一般而言，最难的部分是搞清楚所有表，并且找到确实需要查询的列。如果数据库有几百个个表和几千个列，问题将更为复杂。如果有类图列出了所有的表、列以及表之间的关系，建立查询会简单一些。

查询系统可以生成聚集运算，例如求和、求平均值。类似地，计算机可以对数字型数据做基本的算术运算（加、减、乘、除）。

4.4.2 已经知道什么

大部分情况下查询带有多个限制条件。例如，只对某一特定日期或者某一部门的销售感兴趣。查询条件需要转换成标准的布尔表达式（用AND和OR连接的表达式）。这一步最重要的是写下所有的限制条件，以帮助理解查询的目的。

4.4.3 涉及哪些表

对于较少的表，这个问题很容易。对于数百个表，需要确定究竟需要哪些表。好的数据字典带有同义词和注释，会简化选择查询所需相关表的过程。表的命名是否准确地反映了内容和目的也是至关重要的。

提示：表的选择可以从包含上述前两个问题（结果和条件）中列出的相关列的表开始。然后判断是否有其他表需要作为中介来连接这些表。

4.4.4 如何连接表

这个问题和数据规范化问题相关，并且是关系数据库的核心。具有相似列的表可以连接。例如，Order表含有Customer ID列。相应的数据存储在Customer表中，这个表中也有Customer ID列。大多数情况下，不同表中匹配的列具有相同的名字（例如：Customer ID），这就比较简单。然而，列名并不一定相同，所以有时需要格外小心。例如，Order表可能有一列叫做SalesPerson，它和Employee表中的Employee ID匹配。

4.5 Sally的宠物商店

最初的宠物商店数据库已经建好，一些基本的历史数据也已经从Sally的旧文件中导出。当把这些展示给Sally时，她激动异常。Sally立即开始提出业务上的问题，并且希望知道数据库如何回答。

本章的例子以宠物商店数据库为基础。表和关系如图4-2所示。阅读每一节之后，独立写出查询。完成本章末尾的练习题。答案摆在眼前的时候，查询总是看起来很简单。在学习写查询的过程中，必须静下心来，认真回答四个基本问题。

第3章描述了数据规范化后得到企业的商务模型。表的列表给出公司如何运作的图示。注意宠物商店对待商品和动物有所不同。例如，每个动物都会分开列在一个销售中，但是顾客可以购买多个商品（例如：几袋猫食）。这样做的原因是需要关于动物的额外信息，而这些信

息在普通商品中是没有的。

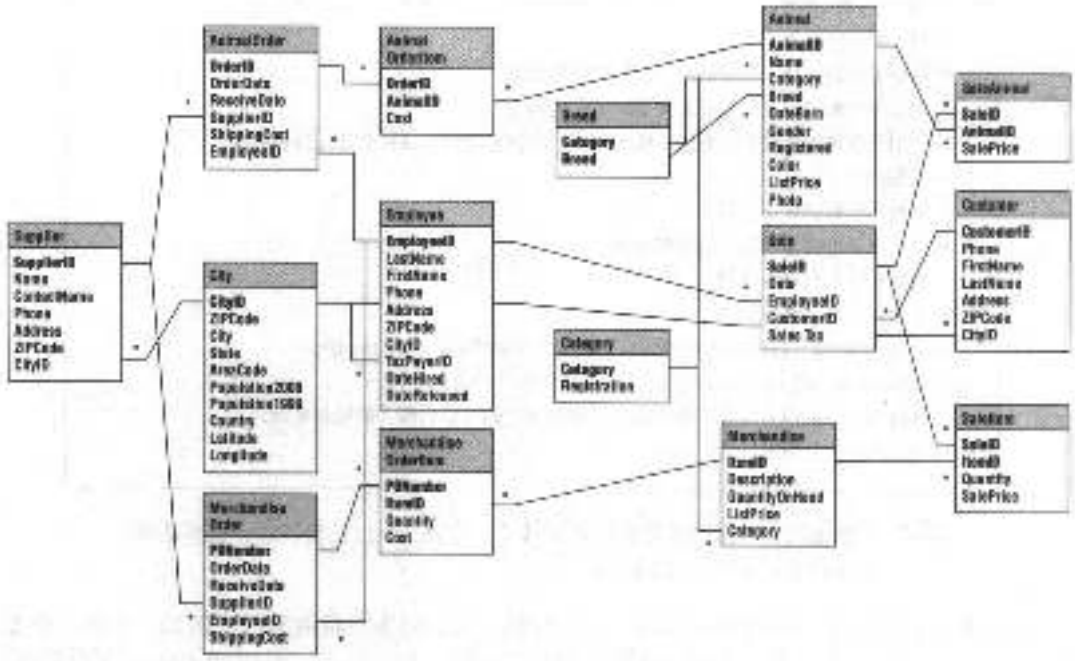


图4-2 宠物商店数据库表。注意动物和商品是类似的，但是分别处理它们

当开始接触已经存在的数据库时，第一步需要熟悉表和列。还要浏览部分主表，熟悉存储在每一个表中数据的类型和数据存储量。理解术语并发现潜在的假设，例如，在宠物商店的例子中，动物可能登记有领养人，但是只能有一个领养人。如果没有登记，则该动物的领养人列为NULL（或缺失）。对于一个特定的公司这第一步较容易，因为已经熟悉了该公司的运作和标识各种对象的术语。

4.6 版本差异

SQL标准是一个典型的软件发展折中的例证。新版本的标准提供有用的功能，但是生产者面临保证已经大量安装的程序和用户的兼容性，从而数据库产品有实质性的区别。这些区别甚至比看到的图形化界面还要显著。为了保持最新，本章（和下一章）中的描述将使用最新的SQL标准。当前大部分系统使用的版本支持多种SQL标准。例如，Oracle 9i支持最新的多表JOIN语法。

4.7 基本查询

查询要从简单做起。本章开始提出只涉及一个表的查询来说明创建查询的基本知识，然后讨论限制、计算和聚集细节，接下来是分组与部分和，最后讨论如何同时从几个数据表中选出数据。

图4-3列出了宠物商店的几个常见业务问题，大部分比较易于回答。事实上，如果表Animal的行比较少，可以人工搜索表得到答案。实际上，开始的时候，可以手动查询问题来

理解查询系统所做的工作。

- 列出所有黄颜色的动物。
- 列出黄颜色，且出生于2004年6月1日以后的狗。
- 列出所有和猫相关，且标价大于10美元的商品。
- 列出所有的雄性并且已登记，或者出生于2004年6月1日以前并且是白色的狗。
- 动物的平均售价是多少?
- 所有动物的总成本是多少?
- 列出前10位顾客和他们的消费总额。
- 动物列表中有多少只猫?
- 计算每一类动物的数量。
- 列出2004年4月1日至2004年5月31日购物的每位顾客的CustomerID。
- 列出2004年4月1日至2004年5月31日购物的顾客的名和电话。
- 列出2004年6月1日至2004年12月31日购买登记的白猫的顾客的姓和电话。
- 哪个雇员卖出的项目最多?

图4-3　宠物商店问题示例。这些问题只涉及一个表，所以比较简单。这些是经理或顾客经常问的典型问题

查询的本意是从表中找到想要的列，并且按照一定的标准限制输出行。例如，在第一个查询（黄颜色的动物）中，希望的查询结果包括AnimalID、Category、Breed和Color。限制返回结果只显示黄颜色的，而不是全部动物。

4.7.1　单表

第一个要考虑的查询是：列出所有黄颜色的动物。动物可以有很多种颜色。数据库的一列是用来存储动物颜色的。一个动物的颜色可以用“黄、白、棕”来描述。一般地说，主要的颜色放在前面，但是没有机制保证数据按照这种方式输入。

首先考虑使用QBE系统解决这个问题，如图4-4所示。QBE系统询问涉及哪些表。这种情况只涉及一个表：Animal。所有的颜色输出来自于表Animal。类似地，限制条件中的列Color也在表Animal中。在表中，可以选择需要输出的列。这个问题有一点含糊，所以选择AnimalID、Category、Breed和Color。

下一步是输入已知的查询条件。在这个例子中，要查询的是黄颜色的动物。在QBE屏幕上列Color输入条件“黄”。有一点需要注意：列Color通常包括多个词。对于有的动物，黄可能作为第二个或者第三个颜色出现在列表中。为了匹配动物，而不考虑yellow出现的位置，需要使用模式匹配函数LIKE。输入条件LIKE ‘%yellow%’，要求查询系统匹配列表中任何位置出现的yellow（在yellow前后可以出现任意数量的字符）。现在可以执行查询了。注意保证列Color中确实存在yellow。

4.7.2　SQL介绍

SQL是一种强大的查询语言。但是和QBE不同，通常需要输入整个语句。Microsoft Access等系统允许在QBE和SQL之间反复切换，这样可以减少输入。也许SQL最大的优势在于

它是一种大部分DBMS版本都支持的标准。因此，一旦学会了这个基本语言，就可以在所有当前主要的系统上创建查询。一些人把SQL读成“sequel”，争辩说它由一种叫做quel的早期DBMS发展而来。而且，“Sequel”比“ess-cue-el”更容易读。

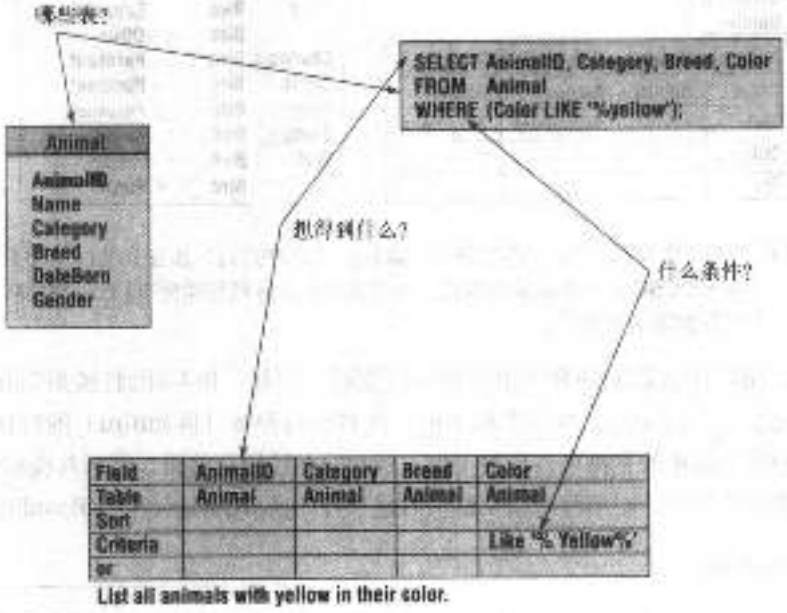


图4-4 QBE和SQL查询示例。因为只有一张表，只需回答三个问题：哪些表？想得到什么？什么条件？

在SQL中最常用的命令是SELECT语句，其作用是从表中找出数据。图4-5是一个简单命令的例子，包括四个基本部分：*SELECT*、*FROM*、*JOIN*和*WHERE*。这些部分与每个查询都需要的基本问题相匹配。在图4-4的例子中，注意QBE和SQL的相似性。

SELECT	columns	你想得到什么?
FROM	tables	涉及到哪些表?
JOIN	conditions	如何连接表?
WHERE	criteria	有哪些限制?

图4-5 基本的SQL语句SELECT命令匹配创建查询需要回答的四个基本问题。大写字母表示SQL关键字，也可以使用小写字母

4.7.3 输出排序

数据库系统把表看作数据集合，为了提高效率，DBMS以任意的方式，或以任意顺序存储表数据。然而大部分情况下，希望返回结果按照一定的顺序显示。SQL的*ORDER BY*子句是一种简单快速的方法，它可以按照指定的顺序显示输出结果。如图4-6所示，直接列出排序的列。默认为升序（从A到Z或对于数值型从小到大）。如果希望从大到小排序，则在列后加上DESC（降序）。在QBE中，可以在QBE格中选择排列顺序。

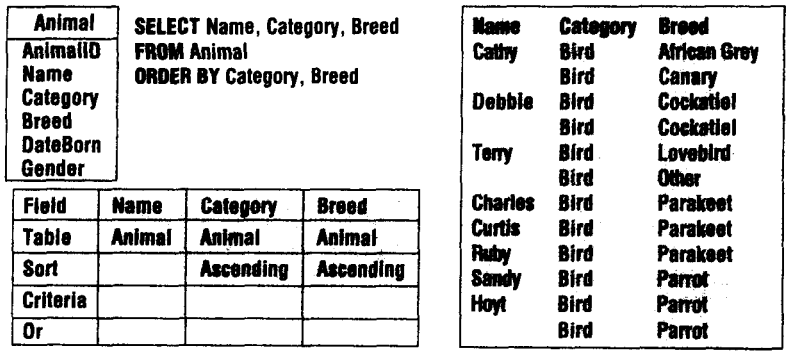


图4-6 ORDER BY子句对输出行排序。默认按照升序排列，加在列名后面的关键字DESC表示按照降序排列。如果像Category列那样含有相同的数据，则依据第二列排序

有的时候，排序所依据的列并不包含惟一的数据。例如，图4-6的行按照Category排序。这时，可能需要另一个排序列。在这个例子中，具有相同种类（例如Bird）的行按照列Breed排序。首先按照写在前面的列排序。在例子中，所有的鸟被排在前面，然后鸟按照Breed排序。在QBE中为了改变排序列，必须移动QBE格中的整个列，使得Category位于Breed的左侧。

4.7.4 关键字Distinct

在有些查询中，SELECT语句可以使用一个有用的选项。关键字DISTINCT告诉DBMS只显示独特的行。例如，图4-7的查询（SELECT Category FT Animal）会返回一长串动物类型（Bird、Cat、Dog等）。实际上，列出了表中所有动物的种类。很明显，有很多猫和狗。为了避免显示重复数据，使用SELECT DISTINCT短语。

注意DISTINCT关键字应用于整行。如果一行中含有任何不同，就会显示。例如，查询SELECT DISTINCT Category, Breed FROM Animal会返回不止图4-7所示的7行，因为每一类动物有很多种。也就是说，每一个类/种组合只显示一次，例如Dog/Retriever。Microsoft Access支持DISTINCT关键字，但是必须在SQL语句中使用。

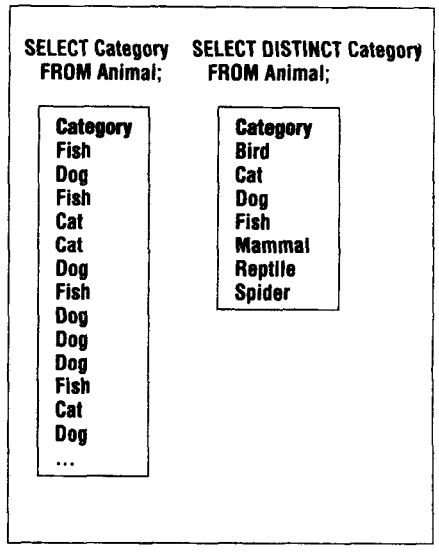


图4-7 关键字DISTINCT消除输出中重复的行。如果没有它，则显示数据库中所有动物的种类

4.7.5 条件

大部分情况下，识别输出列和表是简单的。如果有几百个表，可能会花一些时间来决定需要哪些表和哪些列，但这是一个长期的问题。另一方面，找出限制条件并且准确地表达出来更具挑战性。更重要的是，如果限制条件出现错误，仍然可以得到“答案”。问题是这个答案不是希望得到的，而且通常很难发现错误。

限制的主要概念是基于数学课上学习的布尔代数。实际上，就是不同的条件用AND和OR连接起来。有时也需要NOT语句，表示否定或者对后面语句的真值取反。例如，NOT (Category = 'Dog')表示除了狗以外的所有动物。

考虑图4-8的例子。第一步要注意三个条件定义查询问题：狗、黄色和出生日期。第二步是认识到这三个条件必须同时为真，所以使用AND连接。数据库系统会逐行检查，计算所有三个子句。如果任何一个子句为假，该行被跳过。



Field	AnimalID	Category	DateBorn	Color
Table	Animal	Animal	Animal	Animal
Sort				
Criteria	'Dog'		>'01-Jun-2004'	Like '%Yellow%'
Or				

列出所有黄颜色的狗，并且出生日期在2004年6月1日之后

图4-8 布尔代数。一个由AND连接起来的三个条件的例子。注意#把日期括起来，这是在Microsoft Access中识别日期的规则。如果输入一个文本日期（例如June 1，2004）就更有用了

注意SQL语句是直截了当的，只要写下条件即可。QBE有一些麻烦：写在同一行的每个条件用AND子句连接，而不同行的条件用OR子句连接。创建（或者阅读）QBE语句必须小心，尤其是有很多条件行的时候。

4.7.6 布尔代数

查询的一个最重要方面就是选择想要的行。大部分表包含大量的行，而只需要查看满足某一查询条件的几行。一些条件是直接的。例如，只查询狗。有时条件比较复杂，涉及几个条件。例如，一个顾客想要所有黄颜色的狗，出生在2004年6月1日之后，或者注册为黑色实验室。条件通过布尔代数计算，布尔代数是用于计算条件的标准规则集。你可能已经从基本代数课程中学过了，但是小心一点总是没错的。

DBMS使用布尔代数计算包含多个子句的条件。子句由运算符：AND、OR、NOT连接而成。每一个独立子句计算布尔值得到真或者假。然后使用这些运算符计算整个条件的真值。图4-9展示了主要的运算符（AND，OR）是如何工作的。DBMS检查每一个数据行，并且计算布尔条件值。只有条件为真的时候才显示相应的行。

由AND连接两个子句构成的条件语句，如果它们（a和b）都取真，则结果为真。由OR连接两个子句构成的条

a	b	a AND b	a OR b
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

图4-9 真值表展示AND和OR的区别。由AND连接起来的两个子句必须同时为真。由OR连接起来的子句只要有一个为真即可

件语句，只要至少有一个条件为真，则结果为真。考虑图4-10的例子。第一个条件要求两个子句都为真，但第一个为假，所以结果为假。第二个例子只要求两个子句中的一个为真，所以结果为真。考虑宠物商店的例子。如果一个顾客查看黄色狗的列表，他（或者她）想得到的是种类为Dog并且AND颜色是黄色的动物列表。

如图4-11，添加另外的子句使条件语句变得更加复杂。复杂性源于条件由多个AND连接符和多个OR连接符组成。在这种情况下，结果的真值取决于子句的计算顺序。括号可以决定运算顺序。首先计算最内层的括号。图4-11中上面的例子AND运算符在OR运算符之前计算，结果为真。图中下面的例子OR连接符先计算，结果为假。

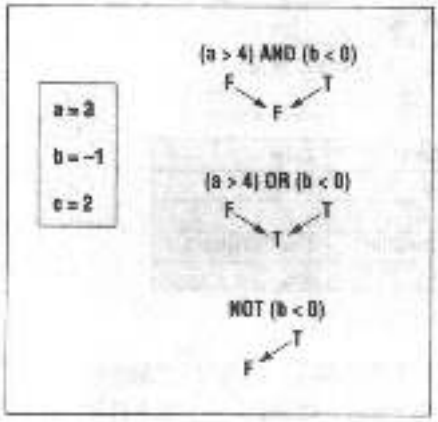


图4-10　布尔代数示例。分别计算每一个子句，然后计算连接值，NOT运算符对真值取反

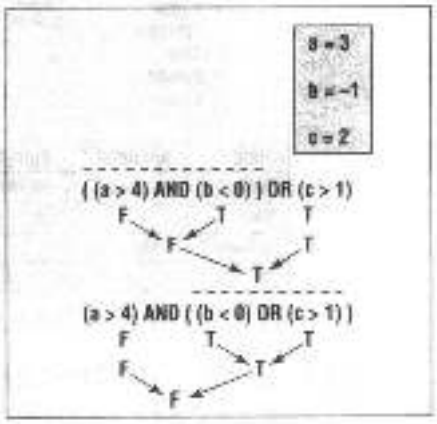


图4-11　由AND和OR混合构成的布尔代数。运算结果取决于哪个运算符先计算。必须使用括号确定运算顺序，首先计算最内层的子句

如果不用括号，运算按照运算符从左到右顺序执行。这个结果可能和希望的不符。但是，DBMS仍然会给出运算结果。为了安全起见，复杂的条件应该一次在一个子句中输入完毕。每次检查输出结果确保符合要求。为了找到图4-11的匹配条件，首先输入（a>4）子句，显示所有结果。然后增加（b<0）子句，显示结果。最后，增加括号和（c>1）子句。

不管如何小心地设计布尔代数，错误总是无法避免的。原因在于像英语这样的自然语言本身是有歧异性的。例如，考虑顾客提出的问题列出“所有黄色的或者白色的，并且6月1日之后出生的狗”。这句话有两种解释：

1）（狗AND黄色）OR（白色AND6月1日之后出生）。

2）（狗）AND（黄色OR白色）AND（6月1日之后出生）。

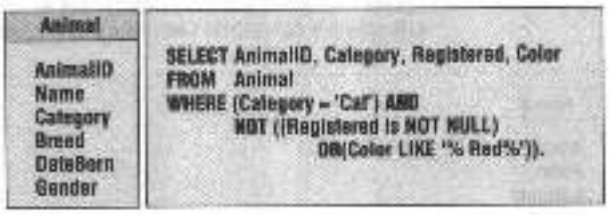
这两个请求显然是不同的。第一种解释返回所有黄色的狗，即使年龄大一些。第二种解释只返回年轻的狗，并且它们必须是黄色或者白色。大部分人说话时不使用括号，尽管停顿有助于理解说话的含义。一个好的设计者（或者售货员）会要求顾客澄清一下。

4.7.7　德摩根定律

设计查询是一个逻辑练习。逻辑学家德摩根（Augustus DeMorgan）发明了一种化简复杂查询的技术。考虑图4-12所示的宠物商店例子。一个顾客光临，并且说：“我要一只猫，但是

不想要已经登记的或者红色的”。即使使用SQL，查询条件也有些复杂：(Category = “cat”) AND NOT ((Registered is NOT NULL) OR (Color LIKE ‘%red%’))。取反运算符（NOT）增加了理解问题的难度。如果用QBE创建查询就更为复杂了。

顾客：“我想要一只猫，但是不想要已经登记的或者红色的。”



Field	AnimalID	Category	DateBorn	Color
Table	Animal	Animal	Animal	Animal
Sort				
Criteria		'Cat'	Is Null	Not Like '%Red%'
Or				

图4-12 带有取反的例子。顾客知道他（或者她）不想要什么，SQL可以使用NOT，但是应该使用德摩根定律对Registered和Color语句取反

问题的解决有赖于德摩根定律（DeMorgan's Law），这个定律阐述了当两个子句用AND或者OR连接的时候，如何对条件取反。德摩根定律指出：对带有AND或者OR连接符的条件取反，等价于对每一个子句取反，并且改变连接符。AND变成OR，反之亦然。图4-13展示了宠物商店顾客例子中如何使用取反条件。每一个条件都取反（NOT NULL变成NULL，red变成NOT red）。连接符从OR变成AND。图4-13展示了使用两种方法计算的最终真值相同。

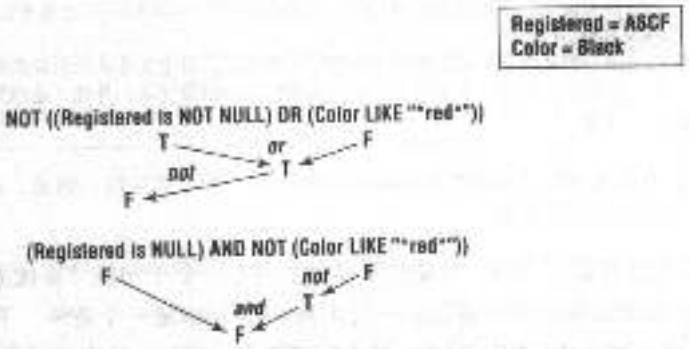


图4-13 德摩根定律。复合语句的否定通过对每一项取反，同时交换连接符（AND变成OR）实现。使用真值表计算示例

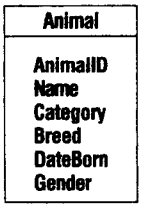
条件的新表达形式的优点在于更容易理解，在QBE中更容易表达。在QBE中分别输入Registration和Color两个子句。把它们放在同一行，用AND连接。在自然语言中，新的形式表达如下：一只没有登记且不是红色的猫。

实际上，德摩根定律用于简化复杂语句。但是，用测试数据计算真值表仍然是必要的。

如果在一个查询中混合使用AND和OR子句，条件将变得更为复杂。考虑图4-14中的问题，

列出所有的狗，雄性的并且登记过，或者出生于2004年6月1日之前并且是白颜色的。

```
SELECT AnimalID, Category, Gender, Registered, DateBorn, Color
FROM  Animal
WHERE ((Category = "Dog") AND
                ( ( (Gender = "Male") AND (Registered is Not Null) ) OR
                  ( (DateBorn < #6/1/2004#) AND (Color Like "*White*") ) ) );
```



Field	AnimalID	Category	Gender	Registered	DateBorn	Color
Table	Animal	Animal	Animal	Animal	Animal	Animal
Sort						
Criteria		'Dog'	'Male'	Is Not Null	< '01-Jun-2004'	Like '%White%'
Or		'Dog'				

列出所有的狗，雄性的并且登记过，或者出生于2004年6月1日之前并且是白颜色的

图4-14　布尔条件——混合使用AND和OR。注意在SQL中使用括号保证计算顺序符合预想。并且，QBE要求两行中都出现重复条件“Dog”

首先，注意英语中将两个子句组合存在歧义。图4-15给出两种可能。第二种解释和第一种有所不同，消除歧义的方法或者是使用括号，或者使用更多的文字。

列出所有的狗，雄性的并且登记过，或者出生于2004年6月1日之前并且是白颜色的
1：（雄性的并且登记过）或者（出生于2004年6月1日之前并且是白颜色的）
2：（雄性的）并且（登记过或者出生于2004年6月1日之前）并且（是白颜色的）

图4-15　自然语言的歧义使得这句话可以用任何一种方式理解。但是，通常理解为第一种含义

用SQL表达查询是直截了当的，只要使用括号表明每一个子句的计算优先级。最内层的子句首先计算。一个校正查询的技巧是在某一行上用T和F标识每一个条件。下一步，用括号和连接符（AND、OR）把这些标志连起来。然后阅读原来的语句，检查结果是否相同。

如果使用QBE，列出同一行用AND连接的子句，每个子句可以放在一对括号里而含义不变。OR的每个子句分开放在不同行。为了解释查询，需要分别观察每一个条件行。如果一行中的所有条件为真，这一行匹配成功。数据行只要满足一个独立的条件行（不需要满足所有行）。

构造复杂查询的另一个提示：一次只测试条件的一部分，对于QBE更是如此。在这个例子中，先写下并测试条件：雄性并且登记过。然后增加另外的条件并且在每一步检查结果。虽然这样做比一步跳到最后查询要慢一些，但是，这样做有助于保证得到正确答案。检查查询里SQLWHERE子句中括号的正确性总是明智的。

4.7.8 有用的WHERE子句

大部分数据库系统提供比较运算符，如图4-16所示。标准的数据可以用等号和不等号比较。文本比较通常使用LIKE运算符做模式匹配。Oracle和SQL Server支持的SQL标准用百分号（%）匹配多个字符，用下划线(_)匹配单个字符。而Microsoft Access使用星号（*）和问号（？）。单字匹配对于搜索文本字符串，例如产品号，特别有用。例如，产品号可能具有DDDCCC9999的形式，前三个字母代表部门，中间三个代表产品类，最后四位是无重复序号。为了找到所有关于Dog类的产品，可以使用WHERE条件如下：ProductID LIKE "___dog____"。大部分系统支持文本提供比较是否对大小写敏感的控制选项。许多人倾向于忽略此项，因为忽略大小写输入更方便。

比　较	例　子
运算符	<, =, >, <>, BETWEEN, LIKE, IN
数字	AccountBalance > 200
文本	
简单的	Name > 'Jones'
单一匹配模式	License LIKE 'A__82_'
任意匹配模式	Name LIKE 'J%'
日期	SaleDate BETWEEN '15-Aug-2004' AND '31-Aug-2004'
缺值	City IS NULL
否定	Name IS NOT NULL
集合	Category IN ('Cat', 'Dog', 'Hamster')

图4-16 WHERE子句中常用的比较。BETWEEN子句可以用于任何数据，对于日期尤为有用

BETWEEN子句不是必需的，但可以节省很多输入，使一些条件更清晰。子句（SaleDate BETWEEN '15-Aug-2004' AND '31-Aug-2004'）等价于（SaleDate >= '15-Aug-2004' AND SaleDate <= '31-Aug-2004'）。这种日期标识法可以在大部分数据库系统上使用。一些系统也允许使用更短的形式，但是必须指定转换形式。这些转换函数不是标准的。例如，Access可以接受大部分日期格式，只要用井号（#）代替括号。Oracle通常要求使用TO_DATE转换函数，例如SaleDate>=TO_DATE('8/15/04','mm/dd/yy')。尤其是开始使用一个新的DBMS时，一定注意小心检查所有日期转换。

另一个有用的条件是空值（NULL）测试。两个常用的形式是IS NULL和IS NOT NULL。注意语句（City = NULL）在大部分系统中是不对的，因为NULL不是一个真实的值。必须使用（City IS NULL）。

4.8 计算

大部分情况使用电子表格或者独立的程序进行复杂计算。但是查询语句也可以完成两类计算：聚集运算和简单的逐行运算。有时候，这两类计算结合在一起。首先考虑逐行计算。

4.8.1 基本运算

SQL和QBE都可以用于对每一行数据进行基本计算。这个技术可以用作自动基本任务，减

少数据存储。考虑一个普通的订购表或销售表。如图4-17所示，基本表包括一系列购物的基本信息：OrderItem(OrderID, ItemID, Price, Quantity)，大部分情况下，需要把Price和Quantity相乘得到每一笔订单的总额。因为这个计算是通用的（没有任何特殊条件），没有必要存储计算结果——在需要的时候，都可以计算得到，只要构造一个查询，并且加入一列，新列使用初等代数，并且给出名字：Price * Quantity AS Extended。切记每一行的计算在查询时执行。

OrderItem(OrderID, ItemID, Price, Quantity)

```
Select OrderID, ItemID, Price, Quantity,
Price*Quantity As Extended
From OrderItem;
```

OrderID	ItemID	Price	Quantity	Extended
151	9764	19.50	2	39.00
151	7653	8.35	3	25.05
151	8673	6.89	2	13.78

图4-17　计算。查询中可以进行数字数据基本运算（+，−，*，/）。新显示的列应该有一个有意义的名字

一些系统提供附加的数学函数。例如：绝对值，对数，三角函数等基本数学函数。虽然这些函数提供了额外的功能，但是要记住，它们只能对表或查询中的某一行存储的数据进行操作。

4.8.2　聚集运算

在商务处理中，数据库经常需要计算总数和部分和。因此，查询系统提供了对数据的聚集运算函数。图4-18列出了常见的函数。它们都对若干行进行运算，返回一个值。最常用的函数是Sum和Avg，这一点和电子表格的情况相似。

```
SELECT Avg(SalePrice) AS AvgOfSalePrice
FROM SaleAnimal;
```

SaleAnimal
SaleID
AnimalID
SalePrice

Sum
Avg
Min
Max
Count
StDev or StdDev
Var

Field	SalePrice
Table	SaleAnimal
Total	Avg
Sort	
Criteria	
Or	

图4-18　聚集函数。QBE和SQL的查询例子：所有动物的平均售价是多少？注意：使用Microsoft Access时必须点击工具栏上的求和按钮（Σ）显示QBE格的Total行

在SQL中，函数简单地加入SELECT语句即可。在QBE中，函数一般作为单独的行Total列出。在Microsoft Access中，先点击工具栏上的求和按钮（Σ），把Total行加入QBE格中。在SQL和QBE中，应为新列起一个有意义的名字。

Count函数在很多情况下是有用的，但是一定要理解Sum和Count的区别。Sum计算一列数的总和。Count计算行数。可以为Count函数传递一个参数，但是很少有不同，一般使用Count(*)就可以了。Count函数的困难之处在于理解何时使用它。必须首先理解语言的问题。例如，Sally有多少个雇员？这个问题用Count函数：SELECT Count(*) From Employee。商品9764一共卖出了多少套？这个问题用Sum函数：SELECT Sum(Quantity) FROM OrderItem。区别在于在Employee表中每行只有一个雇员，但是顾客可以一次购买多个同一物品。还要记住Sum只能用在数字型的数据列（例如：Quantity）。

在很多情况下，把逐行计算和聚集函数结合在一起。图4-19中的例子要计算某一订单的总价格。为了计算总价格，数据库必须先计算每一行的Quantity*Cost，然后对这一列求和。这个例子还说明，经常使用特定的条件（WHERE）限制用于计算总数的行。在这个例子中，只需要一个订单的总数。

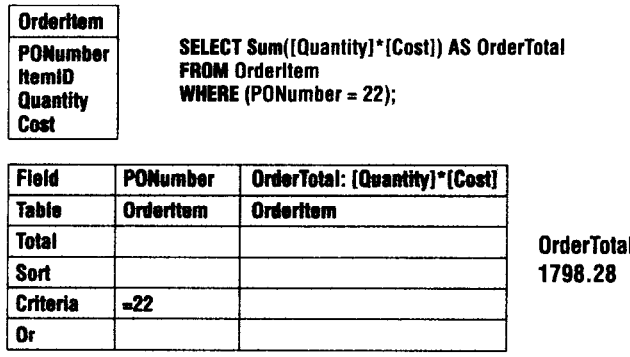


图4-19 计算。在聚集函数（Sum）中，进行逐行计算（Quantity*Cost），但是只有最后的总数显示在结果中

聚集运算有一个重要的限制需要记住。不能同时既显示细节行（逐行）又显示总数。在订单例子中，或者显示细节的计算（如图4-17），或者显示总价格（如图4-19）。大多数情况下，运行两个查询是简单的。然而，如果想同时看到细节和总数，必须像第6章所描述的那样创建一个报表。

可以同时计算多个聚集函数。例如，可以同时显示Sum、Average和Count：SELECT Sum (Quantity), Avg (Quantity), Count (Quantity) From OrderItem。事实上，如果需要这三个值，可以一次计算。考虑一下，如果表有一百万行数据会发生什么。如果分成三个独立的查询，DBMS就会访问数据三次。通过把计算合并在一个查询中，可以把查询时间减少到三分之一。对于大型表或复杂系统，这些查询中的小改动就可以使有的系统成功，而有的系统花费数天时间去运行。

有时使用Count函数，可能需要包括DISTINCT运算符。例如：SELECT COUNT (DISTINCT Category) FROM Animal，计算不同种类的数目，不考虑重复的。虽然这个命令是SQL标准的一部分，一些系统（特别是Access）不支持Count语句中的DISTINCT子句。为了用Access获得同样的效果，首先用DISTINCT关键字构造如图4-7所示的查询。保存查询结果，在此结果上构造新的查询计算所存查询数量。

4.8.3 函数

SELECT命令也支持数据的运算函数。这些运算包括数字形式的比如三角函数，字符串函数比如连接两个字符串，日期函数和格式函数来控制数据显示。不幸的是，这些函数没有标准化，所以每个DBMS版本有不同的函数名字和不同的功能。尽管这样，不管现在使用的是什么DBMS，都应该学会如何做某些典型工作。图4-20列出了一些可能需要的常见函数。即使只学习一种DBMS，也应该把这张表放在手边，以备需要把查询从一个系统移植到另一个系统。

字符串运算相当有用。连接运算功能更为强大，它可以把多列数据合并为一个显示域。对于把姓和名连接起来这种情况尤其有用。其他常见的字符串函数把字符转换为小写或大写字母。长度函数计算字符串列包含的字符数。子串函数返回字符串的所选部分。例如，选项显示长标题的前20个字符。

Task	Access	SQL Server	Oracle
Strings			
Concatenation	FName & " " & LName	FName +' '+ LName	FName || ' ' || LName
Length	Len(LName)	Length(LName)	LENGTH(LName)
Uppercase	UCase(LName)	Upper(LName)	UPPER(LName)
Lowercase	LCase(LName)	Lower(LName)	LOWER(LName)
Partial string	MID(LName,2,3)	Substring(LName,2,3)	SUBSTR(LName,2,3)
Dates			
Today	Date( ), Time( ), Now( )	GetDate()	SYSDATE
Month	Month(myDate)	DateName(month, myDate)	TRUNC(myDate, 'mm')
Day	Day(myDate)	DatePart(day, myDate)	TRUNC (myDate, 'dd')
Year	Year(myDate)	DatePart(year, myDate)	TRUNC (myDate, 'yyyy')
Date arithmetic	DateAdd	DateAdd	ADD_MONTHS
	DateDiff	DateDiff	MONTHS_BETWEEN
			LAST_DAY
Formatting	Format(item, format)	Str(item, length, decimal)	TO_CHAR(item, format)
		Cast, Convert	TO_DATE(item, format)
Numbers			
Math functions	Cos, Sin, Tan, Sqrt	Cos, Sin, Tan, Sqrt	COS, SIN, TAN, SQRT
Exponentiation	2 ^ 3	Power(2,3)	POWER(2,3)
Aggregation	Min, Max, Sum, Count,	Min, Max, Sum, Count,	MIN, MAX, SUM, COUNT,
Statistics	Avg StDev, Var	Avg, StDev, Var,	AVG, STDDEV, VARIANCE,
		LinReqSlope, Correlation	REGR, CORR

图4-20 SQL函数的区别。这张表显示了几种数据库的常见区别。基本运算在查询中很常见，但是进行这些运算的语法依赖于特定的DBMS

日期函数在商务应用程序中很常见。日期列可以相减，得到两个日期之间的天数。还有其他功能，获得当前日期和时间，或者选出日期列中的月、日、年部分。日期函数可以对日期进行月、周、年相加（减）。一个需要注意的问题是在查询语句中输入日期。大部分系统对于世界上不同地区使用不同标准的日期输入和显示方式比较敏感。例如，5/1/2004在美国是5月的第1天；在欧洲则是1月的第5天。为了确保DBMS准确表达用户所要的日期，必须使用转换函数确定日期格式。对于其他类型的数据有附加的格式函数，例如设置固定的十进制小数点或者显示货币符号。

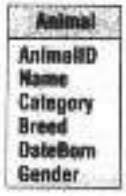
DBMS可能有几十个数值函数，但是常用的屈指可数。大部分系统包括常见的三角函数（比如正弦、余弦）、指数函数等。大部分还支持一些有限的统计计算，例如平均值、标准差、偶尔还有相关、回归函数。一定要查看DBMS文档，了解其可用性和附加函数的细节。然而，记住，任何时候都可以自己编写函数，并且可以在查询中使用，和使用内置函数一样方便。

4.9 部分和与GROUP BY语句

为了查看一部分动物的总数，可以使用带有WHERE子句的Sum函数。例如，可能会问动物列表中有多少只猫？查询是直接的：SELECT Count (AnimalID) FROM Animal Where (Category = "Cat")。运行后得到正确的答案。然后返回继续编辑查询语句，得到狗或者其他任何种类的动物。然而，频繁的修改查询语句令人生厌。而且，如果不知道所有的动物种类，该怎么办呢？

考虑更普遍的查询：输出每一种动物的数量。如图4-21所示，GROUP BY子句可以很好地

解决这类查询问题，QBE和SQL都有这种技术。SQL语法简单明了：只要加上GROUP BY子句。GROUP BY语句只能和一个聚集函数（Sum、Avg、Count等）共同使用。有了GROUP BY语句，DBMS查看所有的数据，找到分组中的独特项，然后对分组中的每一项进行聚集运算。




```
SELECT    Category, Count(AnimalID) AS CountOfAnimalID
FROM      Animal
GROUP BY  Category
ORDER BY  Count(AnimalID) DESC;
```

Field	Category	AnimalID
Table	Animal	Animal
Total	Group By	Count
Sort		Descending
Criteria		
Or		

Category	CountOfAnimalID
Dog	100
Cat	47
Bird	15
Fish	14
Reptile	6
Mammal	6
Spider	3

图4-21 GROUP BY计算部分和，计算每一种动物的数量。这种方法比为每一种动物创建一个WHERE子句高效，为了把商务问题转换到SQL，注意by或for each等词组，通常指定使用GROUP BY子句

默认情况下，输出按照分组项目排序。但是，对于商务问题，根据计算排序（ORDER BY）很常见。宠物商店的例子按照Count排序，具有最多数量的动物排在前面。

在GROUP BY子句中增加多个列必须小心。对于整个GROUP BY子句的每一个不同的项计算部分和。如果增加多个列（例如：Category和Breed），会得到比预想更加细致的分类。

Microsoft增加了和ORDER BY语句连用的一个有用的特征。有时一个查询会返回数千行结果，虽然行已经排序，但是可能只需要查看前几行。例如，列出10个最佳售货员，或者顾客的前10%。对结果排序后，可以用TOP语句轻松地限制结果输出。例如：SELECT TOP 10 SalesPerson, SUM(Sales) FROM Sales GROUP BY SalesPerson ORDER BY SUM(Sales) DESC。这个查询计算每一个销售员的销售总额，并且按照降序显示列表。然而，只有前10行结果会显示出来。当然，可以用任何值代替10。还可以输入一个百分数（例如：TOP 5 PERCENT），这样5%之后的行不再显示。如果一个管理者想查看“最好的”并且跳过剩余行，这些查询命令十分有用。注意Oracle不支持TOP条件。

4.9.1 求和条件（HAVING）

GROUP BY子句功能强大，提供有用的决策信息。如果涉及到多个组，可能需要限制输出列表，尤其是当一些分组相对较小。宠物商店有爬行动物和蜘蛛类动物，但是它们通常比较特殊。在分析销售时，管理者可能更关注卖得最好的种类。

一种减少显示数据量的方法是增加HAVING子句。HAVING子句作为条件，控制GROUP BY的输出。在图4-22的例子中，管理者想要跳过少于10个动物的类别。注意SQL语句只增加一行。同样的条件也可以加到QBE查询的标准格子中。HAVING子句功能强大，和WHERE有相似之处。保证加在计算结果的条件是由GROUP BY子句得到的。在Oracle中，HAVING子句

是TOP语句的一个可能的替代。可以对部分和排序，然后只显示其中满足限制条件的那部分。

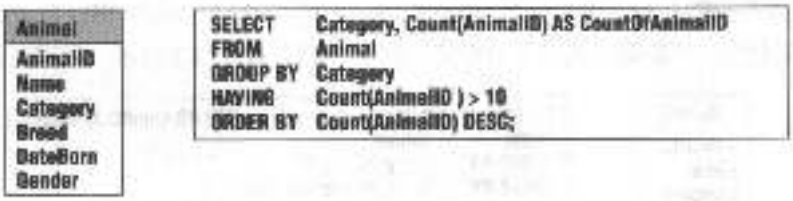


图4-22 用HAVING子句限制输出。带有Count函数的GROUP BY子句可以计算每一种动物的数量。HAVING子句限制输出，只有那些多余10个动物的类别才输出

4.9.2 WHERE子句与HAVING子句

开始学习QBE和SQL时，WHERE和HAVING看起来很相似，不知道选哪个更合适。理解它们的不同非常关键。如果输入有误，DBMS也会返回答案，但答非所问。

关键在于WHERE语句应用于原始表的每一个单独行。HAVING语句只应用于带有GROUP BY的查询语句输出的部分和。为了增加难度，甚至可以把WHERE和HAVING子句结合在一个查询语句中使用，因为只想查看部分数据行，并且限制输出部分和。

考虑图4-23的问题，出生于2004年6月1日之后的每一种动物的数目，但是只列出它们中超过10个的。这个查询的结构和图4-22的例子相似。SQL的不同之处在于增加了WHERE子句(DateBorn > #6/1/2004#)。这个子句对原始表的每一行起作用，决定是否包括在计算中。比较图4-23所示狗的数量是30，而图4-22中有100，只有30只狗出生于2004年6月1日之后。由于HAVING子句的限制，只有超过10个动物的种类才能显示。

查询处理首先逐行检查是否满足WHERE条件。如果满足，查看Category，对应的种类计数增加。表中所有行处理完毕，检查总数是否满足HAVING条件。只有满足的行才显示。

QBE查询略显复杂，两个条件都列在标准格中。然而，仔细观察Total行，可以看到DateBorn列有个Where项。这一项就是用来区分HAVING和WHERE条件的。为了保险起见，一定要检查SQL语句，确保查询符合用户要求。

4.9.3 最好和最差

考虑商务查询：哪种商品卖得最好？如何构造SQL语句来回答这个问题？一开始，必须确定“最好”是按照数量还是收入（价格乘以数量）来衡量。现在先按数量。常见查询写法类似：SELECT Max(Quantity) FROM SaleItem，这个查询可以运行。它返回数量最多的一次销售，但这个查询不能返回总销售量。进一步的可能是：SELECT ItemID, Max(Sum(Quantity)) FROM

SaleItem GROUP BY ItemID，但这个查询是不正确的，因为数据库在计算总数前不能计算最大值。所以最佳的答案是：SELECT ItemID, Sum(Quantity) FROM SaleItem GROUP BY ItemID ORDER BY Sum(Quantity) DESC。这个查询计算每一项的总购买量，并且按照降序显示结果——卖得最多的位于列表最上面。




```
SELECT      Category, Count(AnimalID) AS CountOfAnimalID
FROM        Animal
WHERE       DateBorn > #6/1/2004#
GROUP BY    Category
HAVING      Count(AnimalID) > 10
ORDER BY    Count(AnimalID) DESC;
```

Field	Category	AnimalID	DateBorn
Table	Animal	Animal	Animal
Total	Group By	CountBy	Where
Sort		Descending	
Criteria		>10	>'01-Jun-2004'
Or			

Category	CountOfAnimalID
Dog	30
Cat	18

图4-23 WHERE与HAVING。按照种类计算在2001年6月1日之后出生的动物的数目，但是只需要列出动物数目大于10的动物种类。WHERE子句首先确定是否每一行参与动物数目的计算。GROUP BY语句为每一种类生成动物总数。而HAVING子句则限制输出结果，只有那些大于10个动物的种类才输出

这种做法的一个缺点是返回整个销售表。一般而言，大部分业内人士不仅看最上面和最下面的项，所以这不是一个严重的问题——除非列表太长。如果那样，可以使用TOP或HAVING命令减少列表的长度。

4.10 多表

截至目前，所有的例子都在一个数据表上操作，这样做是为了集中讨论特定的话题。然而实际上，常需要从多个表中得到组合数据。事实上，DBMS的优势在于组合多个表的数据。

第3章展示如何把业务表单和报表分成有关系的表。虽然规范化使数据存储效率更高并且避免了常见的问题，但最终回答业务问题还需要把各个表的数据连接起来。例如，Sale表只包括CustomerID标识特定顾客。大部分人更喜欢看顾客的名字和其他属性。这部分数据存储在Customer表——也带有CustomerID。需要把Sale表的CustomerID和Customer表的相应数据联系起来。

4.10.1 连接表

在现代查询语言中，连接多个表的数据是直截了当的。只要简单地指定涉及哪些表和这些表如何连接即可。QBE在此方面尤为简单。

为了理解方便，首先考虑图4-24所示的查询问题：列出在2004年4月1日至2004年5月31日购物的顾客的CustomerID，因为有的顾客可能在几天内购物，DISTINCT子句用于消除重复元组。

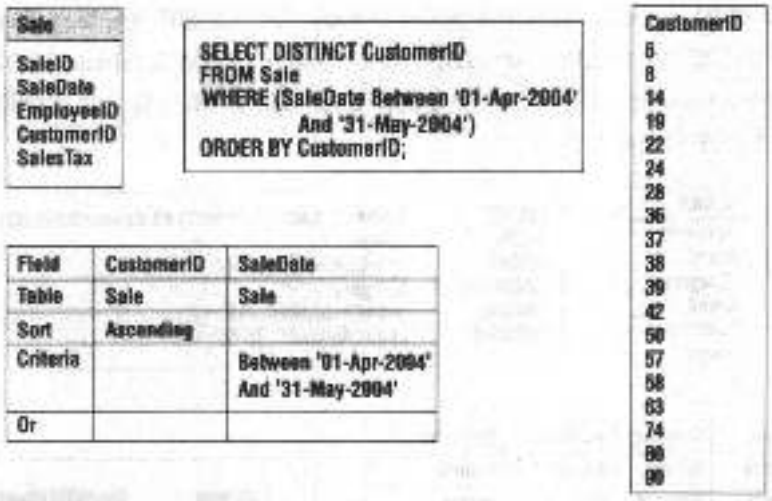


图4-24 列出每一个在2004年4月1日至2004年5月31日购物的顾客的CustomerID。大部分人更喜欢看顾客的名字和地址，这些属性存储在Customer表中

大部分管理者可能更希望看到顾客姓名，而不是CustomerID。但是，姓名存储在Customer表中，因为把顾客的所有属性都复制到和顾客相关的所有表中严重浪费空间。如果要打印这些表，必须把销售报表的CustomerID和Customer表的相应行匹配获得顾客姓名。当然，手工匹配非常耗时。查询系统可以轻松做到。

如图4-25所示，QBE查询系统在某些方面比SQL语法要简单。但概念是一致的。首先，找到涉及的两个表（Sale和Customer）。在QBE中，从列表中选择表名，显示在表单的最上方。在SQL中，在FROM行输入表名。然后，告诉DBMS每一个表的哪些列匹配。在本例中，Sale表的CustomerID和Customer表的CustomerID匹配。大部分情况列名相同，但也可以不同。

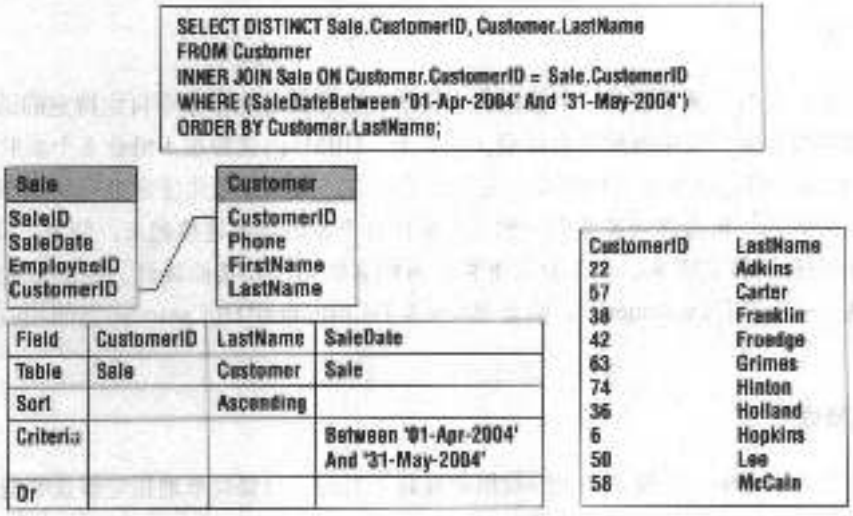


图4-25 连接表运算导致基于JOIN语句指定的列的匹配。可以使用任何一个表的数据。查询问题是：列出在2004年4月1日至2004年5月31日间购物的顾客的姓

在SQL中，用JOIN语句把表连接起来。这个语句在SQL 92的介绍中改变了，可能会遇到很多仍然使用更老版本SQL89语法的查询语句。在SQL89中，JOIN条件是WHERE子句的一部分。大部分版本转换到SQL92语法，所以本书基于该格式。正如第5章指出的，SQL92语法在需要改变连接结构时更为简单易懂。

JOIN的语法如图4-26所示。类似SQL89的非正式语法也列出来了。DBMS不会接受使用非正式语法的语句，但是，如果查询使用了很多表，先写出非正式语法再添加必要的语法细节更容易一些。注意QBE和SQL都必须指定涉及哪些表和哪些列的数据匹配。

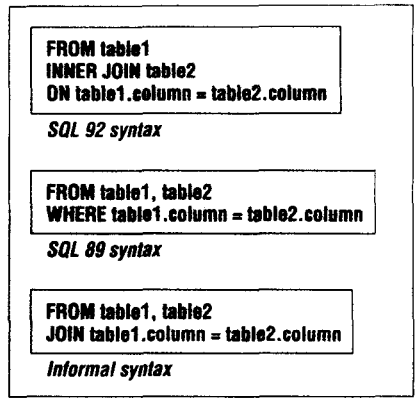


图4-26 SQL92和SQL89的连接表语法。非正式的语法不能在任何DBMS中使用，但是，当需要连接很多表时，这样做更方便阅读

4.10.2 标识不同表中的列

注意SQL JOIN语句中，列是如何指定的。因为列名CustomerID出现在两个表中，写成CustomerID = CustomerID是没有意义的。DBMS无法理解这句话的意思。为了跟踪指定的列，必须指出表名：Sale.CustomerID。实际上，这个语法可以在任何时候指定列。只有当多个表含有相同列名的时候才必须使用完整的table.column格式。如果使用QBE和Microsoft Access，就会发现它总包括表名，这是为了避免添加表时引起混淆。

4.10.3 连接多张表

查询可以使用多张不同表的数据。如果不考虑表的数量，这一过程是类似的。每一个希望增加的表必须和另一张拥有共同数据列的表连接起来。如果不能找到共同的列，要么是规范化没有做好，要么需要找到第三张表包含和这两张表的连接。

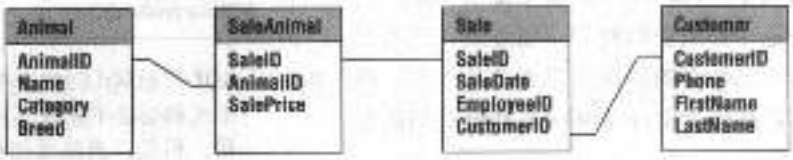
考虑图4-27的例子：列出在给定两个日期之间购买登记过的白色猫的顾客姓名和电话号码。一个重要的步骤是确定需要的表。复杂的问题涉及几张表，最好列出作为输出的列和带有限制的列。在这个例子中，姓名和电话号码是想要得到的，它们在Customer表中。注册状态、颜色、种类（猫）都在Animal表中。SaleDate在Sale表中。但是，连接这三张表，很快会发现Animal表不能和两外两张表连接。由于顾客可以一次购买多个动物，所以这些重复的数据存储在另外一张表中，这个表就是SaleAnimal，包括SaleID和AnimalID列。因此，这个查询使用四张表。

如果数据库包含大量表，那么构造复杂的查询具有相当的挑战性。必须熟悉包含所需列的表。对于大数据库，实体关系图（ERD）或者类图可以显示表是如何连接的。如果数据库建在Access，一定要在建表之前定义关系。第3章阐述了Access如何设置外码关系的参照完整性。当选定表时，Access使用关系自动给QBE添加JOIN运算。还可以使用ERD帮助用户构造查询。

刚开始，连接多于两张表的SQL92语法看起来很复杂。实际上，最好不要强记语法。开始学习SQL时，理解JOIN的含义比语法更为重要。图4-28展示了连接三张表的语法。从最内层的JOIN（括号里）开始阅读或创建类似的语句。然后用相应的ON条件增加表。如果需要其他

的表，继续从内向外增加括号和ON语句。一定要保证新添加的表可以和括号内的某个表连接。图4-28还展示了一种可以速写的简单语法，开始构造查询或者在某种紧急情况下，比如期中考试时可以使用。这种写法类似较老的SQL89（但不完全正确），它的语法是列出所有FROM子句中的表，然后再用WHERE子句连接它们。

```
SELECT DISTINCTROW Customer.LastName, Customer.Phone
FROM Customer INNER JOIN (Sale INNER JOIN (Animal INNER JOIN SaleAnimal
ON Animal.AnimalID = SaleAnimal.AnimalID) ON Sale.SaleID = SaleAnimal.SaleID)
ON Customer.CustomerID = Sale.CustomerID
WHERE ((Animal.Category = 'Cat') AND (Animal.Registered Is Not Null)
AND (Color Like '%White%') AND (SaleDate Between '01-Jun-2004' And '31-Dec-2004'));
```



Field	LastName	Phone	Category	Registered	Color	SaleDate
Table	Customer	Customer	Animal	Animal	Animal	Sale
Sort	Ascending					
Criteria			'Cat'	Is Not Null	Like '%White%'	Between '01-Jun-2004' And '31-Dec-2004'
Or						

图4-27 连接多张表。QBE使得连接多张表相对简单，只要把表列在同一行即可。对于SQL，从两张表开始，并向外扩展。例如，开始用(Animal INNER JOIN SaleAnimal ON Animal.AnimalID = SaleAnimal.AnimalID)，然后用JOIN连接第三张表（Sale）

SQL92连接3张表语法：

```
FROM Table1
  INNER JOIN (Table2 INNER JOIN Table3 ON Table2.ColA = Table3.ColA)
  ON Table1.ColB = Table2.ColB
```

简单但不正确的标记：

```
FROM  Table1, Table2, Table3
JOIN  Table1.ColB = Table2.ColB
      Table2.ColA = Table3.ColA
```

图4-28 连接多张表。按照SQL92语法，首先连接括号内的两张表，然后增加第三张表和JOIN条件。如果集中在如何连接表，使用简单的语法——一定要记住，为了计算机可以理解，必须转换成SQL92语法

4.10.4 表连接提示

表连接和数据规范化关系密切。规范化把数据划分成表以便高效存储和搜索。查询语句和SQL是一个相反的过程：JOIN运算把表中的数据重新连接起来。如果规范化不正确，可能无法连接表。所以当构造查询语句时，要双重检查规范化以保证其正确性。初学者经常在JOIN上遇到麻烦，这一节列出几点提示帮助读者理解潜在的问题。

时刻牢记使用多张表必须连接。有趣的是很多数据库查询系统可以接收没有表连接的查询。甚至可以给出结果。不幸的是结果通常毫无意义。连接起来的表也可以创建一个大的查询。如果没有任何限制，大部分查询系统会产生一个交叉连接（Cross JOIN），一个表的每一行和另一个表的每一行配对。在代数上，交叉连接就是两个集合的笛卡儿积。如果两个表分别有m行和n行，查询结果就有m*n行！

在任何可能的地方双重检查复杂查询的结果，使用样本数据，独立测试用例，可以手工计算结果。也可以一步步地构造复杂查询。开始只要一两张表，并且检查中间结果是否有意义，然后添加新表和其他的限制。最后添加汇总计算（例如Sum、Avg等）。只通过一个数据（总数）判定正确性很困难。相反，观察中间结果列表并且确保它包括所有想要的行，然后添加其他计算请求。

用在JOIN运算中的列经常作为主码列，但是也可以使用任何列连接表。类似地，连接的列可以有不同的名字。例如，可以把Employee.EmployeeID列和Sale.SalesPerson列连接起来。惟一的技术限制是这两个列必须具有相同的数据类型（域）。某些情况下，可以通过函数转换数据降低限制。例如，可以使用Left(ZipCode,5) = ZipCode5把9位的邮政编码字符串转换为5位数字。一定要保证两列数据匹配是有意义的。例如，连接表条件为Animal.AnimalID = Employee.EmployeeID毫无意义。DBMS事实上接受了JOIN（如果两个ID值都是整数），但是JOIN没有任何意义，因为一个Employee永远不是一只Animal（除非是科幻电影）。

避免表之间的多连接。对于Access，如果事先定义了表之间的关系，那么这个问题经常出现。AccessQBE自动使用这些关系连接查询中的表。如果选择如图4-29所示的4张表，去掉所有四个JOIN运算，将无法获得想要的答案。四个JOIN语句返回AnimalOrders，只有Employee设置的订单具有和Supplier相同的CityID！如果只需要Supplier的City，解决方案删除Employee和City的JOIN运算。一般来说，如果查询使用四张表，那么会有三个JOIN运算（比表的数量少一）。

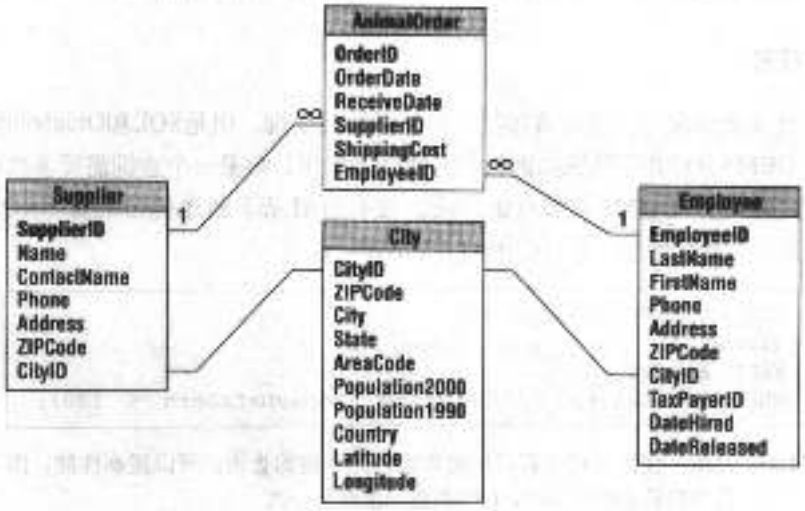


图4-29 这个查询涉及4张表和四个JOIN运算，它返回Employee和Supplier具有相同CityID的行。如果只需要供应商的城市，删除Employee和CityID之间的JOIN即可。如果两个城市都需要，那么增加另一个City表作为第五张表

4.10.5 表的别名

仔细考虑前面的例子。如果想同时显示Supplier的City和Employee的City该怎么办？当然，允许城市不同。答案用到了SQL的一个小技巧：增加第二张City表。第二张“副本”具有不同的名字（例如：City2）。在FROM子句中给表一个新的名字（别名）：FROM City AS City2。如图4-30所示，City表和Supplier表连接起来。City2表连接到Employee表。这样，查询在同一张表上执行了两次独立的JOIN运算——仅仅因为有了不同的名字。




```
SELECT Supplier.SID, Supplier.CityID, City.City, Employee.EID,
Employee.Last Name, Employee.CityID, City2.City
FROM (City INNER JOIN Supplier ON City.CityID = Supplier.CityID) INNER JOIN
((City AS City2 INNER JOIN Employee ON City2.CityID = Employee.CityID)
INNER JOIN AnimalOrderON Employee.EmployeeID = AnimalOrder.EmployeeID)
ON Supplier.SupplierID = AnimalOrder.SupplierID;
```

SID	Supplier.CityID	City.City	EID	LastName	Employee.CityID	City2.City
4	7972	Middlesboro	5	James	7883	Orlando
2	10896	Springfield	1	Reeves	9201	Lincoln
4	7972	Middlesboro	3	Reasoner	8313	Springfield
9	10740	Columbia	8	Carpenter	10592	Philadelphia
6	10893	Smyrna	3	Reasoner	8313	Springfield

图4-30 表的别名。City表使用了两次。第二次指定别名City2，并且作为独立的表对待。因此，可以检索不同城市的Supplier和Employee

4.10.6 创建视图

查询可以作为视图保存。微软直接把它叫做存储的查询，但是SQL和Oracle叫做视图，不论哪种情况，DBMS分析并存储SQL语句，方便以后使用。如果一个查询需要多次运行，就应该作为视图存储，这样DBMS只需分析一次。图4-31展示了创建视图的基本SQL语法。以SELECT语句开头，并且增加一行（CREATE VIEW…）。

```
CREATE VIEW Kittens AS
SELECT *
FROM Animal
WHERE (Category = 'Cat' AND (TodayDateBorn < 180);
```

图4-31 视图，视图是可以在任何时候运行的存储的查询。可以提高性能，因为用户只需输入一次，DBMS也只需分析一次

视图最强大的特征在于它可以和其他查询配合使用。对于多次运行的查询，视图非常有用，也可以创建视图解决复杂的问题。用户可以基于视图创建新的简单的查询，在图4-31的例子中，创建视图（Kittens），显示在最近180天内出生的猫的数据。如图4-32所示，用户可以基于诸如

颜色等其他条件搜索视图Kittens。

```
SELECT Avg(ListPrice)
FROM Kittens
WHERE (Color LIKE '%Black%');
```

图4-32 基于视图的查询。视图可以和其他查询配合使用

如果仅仅使用视图显示数据，那么很简单。但是，如果希望用视图更新数据，那么必须小心。可能无法更新视图中的部分数据列，这取决于如何创建视图。图4-33是一个更新视图的例子。目的是增加新的定购数据项。用户输入OrderID和ItemID。该Item的相应描述自动从Item表搜索得到。

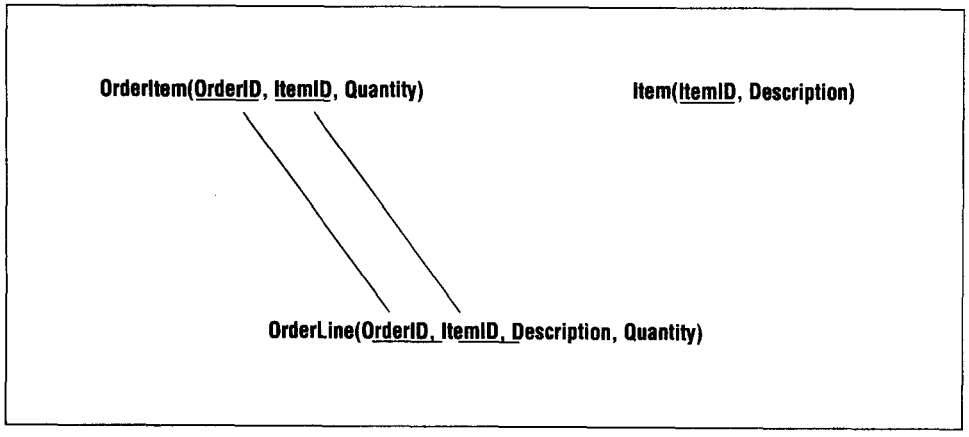


图4-33 更新视图。视图OrderLine设计为在一张表（OrderItem）上更新数据。Item表的Description列用于显示，帮助用户检查ItemID输入的正确性

图4-34举例说明如果草率地选择视图的列会遇到的麻烦。这里OrderLine视图使用Item表（而不是OrderItem表）的ItemID值。这样不能给OrderLine视图增加新的数据。为了理解其中的原因，考虑试图把ItemID从57改为32会发生什么。如果执行该更新，新值存储在Item表，仅仅把猫食的ItemID从57改为32。

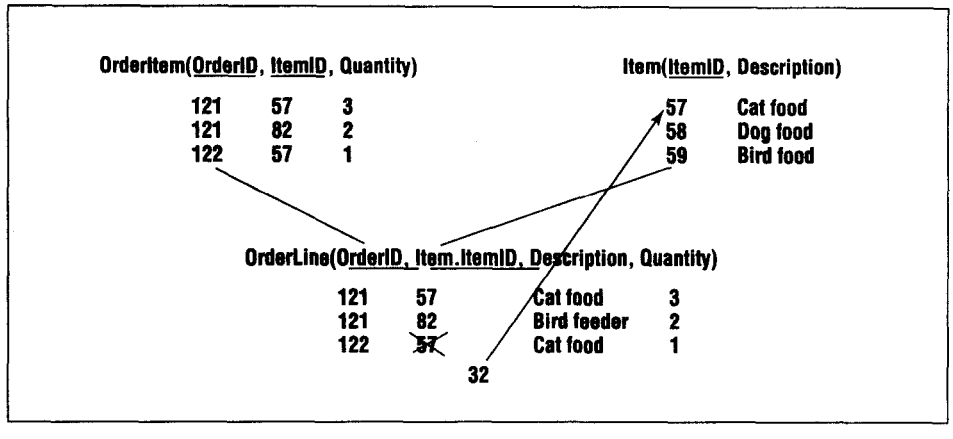


图4-34 不可更新的视图。不要把不同表的主码混在一起。如果这个视图运行，结果并不是想要的。如果把ItemID从57改为32，那么更改的仅仅是猫食的ItemID。不能向OrderItem表增加新数据

为了保证视图是可更新的，必须设计它只改变一张表中的数据。其余数据只用于显示——例如检查用户输入的ItemID是否正确。视图涉及的主码列决不能多于一张表。还有，为了保持可更新性，视图不能使用DISTINCT关键字、GROUP BY和HAVING子句。

视图在数据库中用途广泛。特别是帮助业务管理者和数据库打交道。数据库管理员（DBA）或者信息系统（IS）工作人员可以为业务管理者创建视图，他们只需要查看以视图形式显示的那部分数据。因此，可以隐藏视图的复杂性和大小。最重要的是，可以隐藏构造视图所用的JOIN运算。这样管理者只需面对简单的限制。通过保持视图更新，管理者不再需要原始表。

注意有的数据库系统限制视图可以使用的命令。例如，老版本的Oracle和新版本的SQL Server系统不允许用户在一个存储的视图中使用ORDER BY子句。这样限制的原因是系统可以优化查询提高效率。为了对结果分类，必须对新的查询增加新的ORDER BY语句，这个查询叫做保存的视图。最后，不论如何仔细地用JOIN语句构造视图，DBMS都不允许更新。如果DBMS允许，那么可更新视图可以在建表单时节省一些时间。但是，在其他情况下，必须放弃并且使用简单表单。

小结

创建查询的关键是回答四个问题：（1）想要得到什么？（2）有什么限制？（3）涉及哪些表？（4）这些表如何连接？创建查询的本质就是使用这四个问题获得正确的逻辑。WHERE子句通常是错误的来源。一定要理解查询的目标。用OR和AND连接语句，以及使用德摩根定律化简条件时一定要小心。

要不断地对查询进行测试。构造复杂查询最好的办法就是从简单查询开始，然后增加表。每次增加一个条件，并且检查结果是否正确。最后输入计算请求和GROUP BY子句。当进行计算时，一定要准确理解Sum和Count的区别。记住Count仅仅对行计数；Sum对特定列的值求和。

连接表是直截了当的。一般最好的办法是使用QBE指定连接表的列，然后检查SQL命令的语法。记住JOIN连接的列可以有不同的名称。还有，在没有公共列的情况下，需要增加第三个（或者第四个）表把两张表连接起来。时刻保持类图在手边，有助于确定要使用哪些表以及这些表是如何相互连接的。

开发漫谈

正如Miranda所记下的，SQL和QBE比写程序检索数据简单得多。但是，仍要小心。最危险的就是返回的查询结果并不是查询答案。为了降低风险，一步步地构造查询，每一步都要检查结果。最后使用聚集函数和GROUP BY子句要尤为小心。这样可以检查WHERE子句是否输入正确。如果精心选择列名，更容易看出表如何连接。无论如何，列确实不需要相同的名字才能连接。对于课程项目，要识别常见的查询问题，并且写出查询语句。

关键词

聚集	DESC	示例查询（QBE）
别名	DISTINCT	逐行计算

BETWEEN
布尔代数
数据定义语言（DDL）
数据操纵语言（DML）
德摩根定律
交叉连接（Cross JOIN）
FROM
GROUP BY
HAVING
JOIN
LIKE
NOT
NULL
ORDER BY
SELECT
SQL
TOP
视图
WHERE

复习题

1. 创建查询的四个问题是什么？
2. SQL SELECT命令的基本结构是什么？
3. DISTINCT运算符的目的是什么？
4. 在复杂的（布尔）WHERE子句中使用括号为什么重要？
5. 德摩根定律的内容是什么？它如何化简条件？
6. ORDER BY和GROUP BY命令的区别是什么？
7. SQL的基本算术运算符（+、−等）和聚集函数（SUM等）有何不同？
8. 在SELECT命令中可以使用哪些基本的聚集函数？
9. Count和Sum有何区别？分别举例说明如何使用。
10. WHERE和HAVING子句有何区别？分别举例说明如何使用。
11. SQL连接两张表的语法是什么？

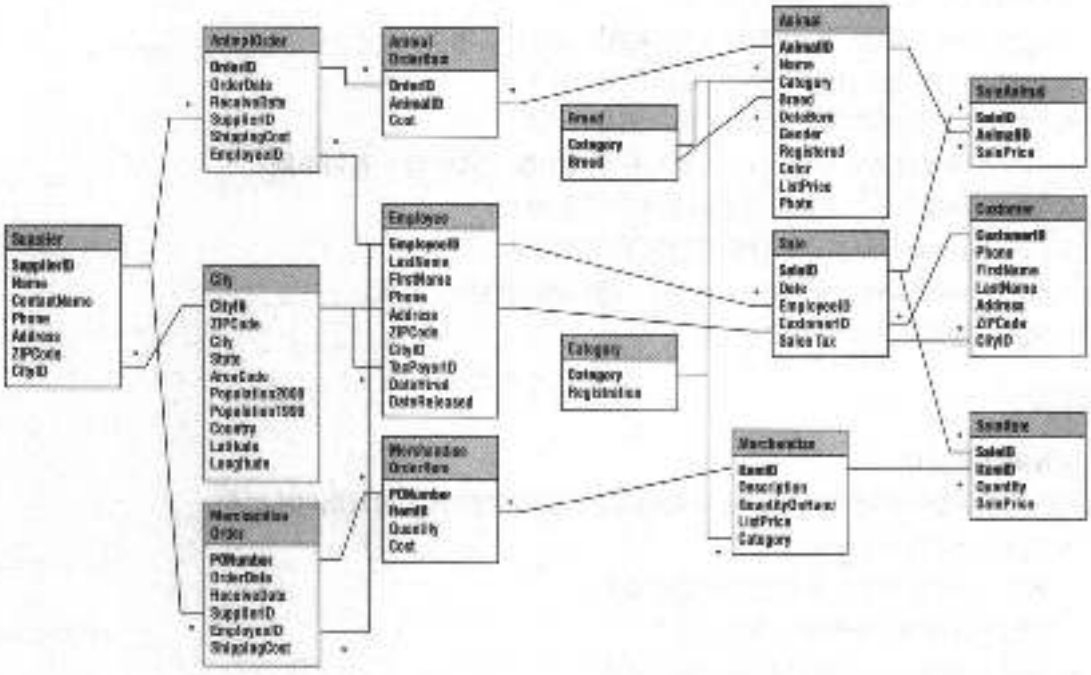
练习

Sally的宠物商店

回答下面编号为1~25的问题，基于宠物商店数据库中的表，请写出SQL语句。

1. 列出在5月份出生的猫。
2. 列出在6月份的前7天在商店购物的顾客。
3. 7月份卖出的最昂贵的一项是什么？
4. 哪些供应商在发出订单后超过10天到达？
5. 列出库存少于10的项。
6. 列出棕色并且花费少于300美元的猫，或者棕色雌性并且价格低于150美元的动物。
7. 12月份卖出动物的总额是多少？
8. 10月份卖出多少只猫？
9. 哪个雇员在4月份订购来自田纳西州供应者的商品？
10. 列出卖鸟给Sally宠物商店的内布拉斯加州供应商。
11. 8月份下了最大订单的是哪个雇员？
12. Sally的顾客来自哪个州的最多？
13. 7月份宠物商店卖给哪个州的商品（按价格计算）最多？
14. 列出向Reasoner提交报表的雇员。

15. 列出从Gibson购买猫的顾客。
16. 商店是否在第一个季度卖出更多的猫或狗?
17. 商店是否在第四个季度卖出更多的雌性或雄性动物?
18. 列出1月份卖出的登记过的白色的狗。
19. 哪个供应商在第一季度卖给宠物商店的货物最多?
20. 哪个雇员在7月和8月卖出的货物最多?
21. 哪些猫被预订了，也就是说在订购前已经卖完?
22. 哪些动物低于成本卖出?
23. 哪些猫的售价至少高于成本50%?
24. 对于每一个商品供应商来说，平均运输成本是多少?
25. 第三季度，在超过5个单元的订单中，哪一项商品订购次数最多?



Rolling Thunder自行车

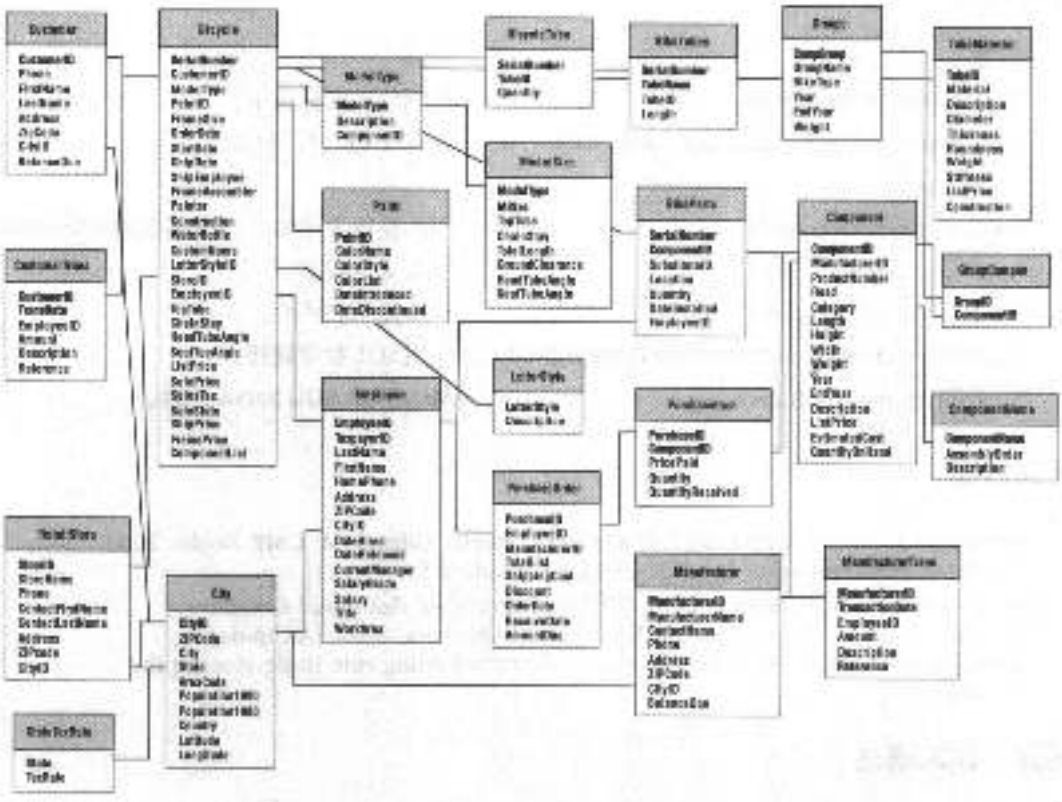
下面编号为26~50的问题是基于Rolling Thunder数据库中的表，写出SQL语句给出答案，并用Access实现查询。

26. 列出2003年9月来自加利福尼亚购买红色山地车的顾客。
27. 列出2001年在没有零售商帮助的情况下，卖出赛车运送到威斯康星州的雇员。
28. 列出2002年佛罗里达州装有后变速器的公路自行车。
29. 谁2004年在乔治亚州购买了最大的（框架尺寸）全避震山地自行车?
30. 哪个厂商2003年在一个订单中给Rolling Thunder以最大的折扣?
31. 自2000年1月1日起，哪个顾客在购买赛车时得到了最大的折扣比例?
32. 哪个雇员在山地自行车上安装了最多的头戴耳机?

33. 哪个非美国公司接受了最多的订单？

34. 在Rolling Thunder库存超过200件的公路自行车部件中，哪个最贵？

35. 按照估计成本，哪种存货项占用了最多的资金？



36. 一个雇员一天最多安装多少个部件？

37. 2003年赛车最流行的字体风格是什么？

38. 哪个顾客在本公司花费最多，2002年这个人买了多少辆自行车？

39. 从2003年到2004年，山地自行车（全避震或无避震）销售增长了还是减少了（按数量计算，而不是价格）？

40. 2003年公司在哪个部件上花费最多？

41. 2003年5月，哪个雇员上红色的赛车最多？

42. 2003年加利福尼亚哪个自行车商店帮助销售了最多的自行车（按价格计算）？

43. 自行车11356所有部件的总重是多少？

44. 2002年Campy Record组所有项目的总价是多少？

45. 2003年，还有其他赛车是用碳质钢或钛质钢制造的吗（以下管为准）？

46. 2001年Shimano XTR后变速器的平均价格是多少？

47. 1999年制造的54cm（框架尺寸）公路自行车的平均上管长度是多少？

48. 平均来看，哪种轮胎的价格更高，公路轮胎还是山地轮胎？
49. 2003年5月，哪些雇员卖出的公路自行车是由他们自己喷漆的？
50. 2002年，着色时旧英语字体更流行吗？

参考网站

网站	描述
http://www.opengroup.org	包括SQL的标准组。
http://www.jtc1sc32.org/sc32/jtc1se32.nsf/Attachments	标准文档，从742开始。
http://thebestweb.com/db	SQL提示参考组，许多其他网站的链接。
http://www.sqlmag.com	SQL重点杂志。
http://www.sqlteam.com	SQL提示和评论。
http://www.vb-bookmark.com/SqlTutorial.html	一般SQL参考链接。
http://msdn.microsoft.com	Microsoft SQL Server注释。

补充读物

Gulutzan, P., and T. Pelzer. *SQL-99 Complete, Really.* Gilroy, CA: CMP Books, 2000. [In-depth presentation of the SQL-99/SQL3 standard.]

Melton, J., and A. R. Simon. *SQL 1999: Understanding Relational Language Components.* San Mateo: Morgan Kaufmann Publishers, 2002. [An in-depth presentation of SQL 1999, by those who played a leading role in developing the standard.]

附录：SQL语法

修改表

```
ALTER TABLE table
      ADD COLUMN column datatype (size)
      DROP COLUMN column
```

提交任务

```
COMMIT WORK
```

创建索引

```
CREATE [UNIQUE] INDEX index
ON table (column1, column2, ...)
WITH {PRIMARY|DISALLOW NULL|IGNORE NULL}
```

创建表

```
CREATE TABLE table
(
    column1    datatype (size) [NOT NULL] [index1],
    column2    datatype (size) [NOT NULL] [index2],
    ...,
    CONSTRAINT pkname PRIMARY KEY (column, ...),
    CONSTRAINT fkname FOREIGN KEY (column)
          REFERENCES existing_table (key_column)
)
```

创建触发器

```
CREATE TRIGGER triggername {BEFORE|AFTER}
    {DELETE|INSERT|UPDATE}
    ON table {FOR EACH ROW}
    {program code block}
```

创建视图

```
CREATE VIEW viewname AS
SELECT ...
```

删除

```
DELETE
FROM table
WHERE condition
```

删除

```
DROP INDEX index ON table

DROP TABLE

DROP TRIGGER

DROP VIEW
```

插入

```
INSERT INTO table (column1, column2, ...)
VALUES (value1, value2, ...)

INSERT INTO newtable (column1, column2, ...)
SELECT ...
```

授权

```
GRANT privilege     privileges
ON object         ALL, ALTER, DELETE, INDEX,
TO user|PUBLIC    INSERT, SELECT, UPDATE
```

收回

```
REVOKE privilege          privileges
ON object           ALL, ALTER, DELETE, INDEX,
FROM user|PUBLIC    INSERT, SELECT, UPDATE
```

回滚

```
SAVEPOINT savepoint

ROLLBACK WORK
     TO savepoint
```

选择

```
SELECT DISTINCT table.column {AS alias}, ...
FROM table/view
INNER JOIN table/view ON T1.ColA = T2.ColB
WHERE (condition)
GROUP BY column
HAVING (group condition)
ORDER BY table.column
{UNION, INTERSECT, EXCEPT, ...}
```

选择创建

```
SELECT column1, column2, ...
INTO newtable
FROM tables
WHERE condition
```

更新

```
UPDATE table
SET column1 = value1, column2 = value2, ...
WHERE condition
```

第 5 章

高级查询和子查询

本章学习内容

- 什么是子查询?
- 如何找到列表中没有的东西?
- 当某些项只在一个表中时，如何连接表?
- 什么时候一个表需要与自己连接?
- SQL可以用来更新和删除数据吗?

5.1 开发漫谈

Ariel： 嘿！Miranda。你看起来好高兴啊！

Miranda： 是啊。查询系统太棒了。它对管理员很有帮助。每次提出请求，它都可以回答它所知道的一切。他们不再每天找我要结果了。而且，我知道查询系统和数据规范化是如何关联的。规范化以后，可以把表分开，以便数据库存储。现在查询系统可以重新连接以获得答案。

Ariel： 也就是说你已经可以创建应用程序了?

Miranda： 差不多，但还没有完全准备好。昨天，我的叔叔问了一个问题，我不知道怎么回答。

Ariel： 真的，我本以为你可以用SQL做任何事情。你遇到什么问题了?

Miranda： 比如查询哪些顾客上个月没有订购任何东西。我试了多次，结果就是不对。

Ariel： 好像不难解决。

Miranda： 我知道。我可以找到顾客列表和除了上个月之外的订单。但是每次连接Customer表和Order表，得到的总是发出订单的顾客。我不知道怎么找到不存在的东西。

5.2 简介

第4章描述了SQL SELECT语句的基本部分。现在可以深入学习更加复杂的查询了。SQLSELECT命令最强大的特征是*子查询*或*嵌套查询*。这个特征使得用户可以提出更加复杂的问题，同时检索不同类型的数据或不同数据源上的数据。SQL不仅仅是一种查询语言。它可以创建整个数据库（数据定义语言）。SQL还有可以改变数据的命令（数据操纵语言）。

学习如何使用子查询关键在于两点：（1）SQL设计对数据集合进行操作——不要按照行考虑问题，（2）可以把嵌套的查询分解成相互独立的部分，分别处理每一部分。

第4章讲过的SQL特征已经非常强大。为什么还要继续学习更多的特征呢？考虑Sally宠物商店这样一个常见问题：哪些动物没有卖出？考虑如何使用现有的SQL知识回答这个问题。第一步选择表：可能选择SaleAnimal和Animal。第二，选择输出列：AnimalID和Name。第三，指定条件。第四，连接表。在这个例子中，最后两步带来了最多的问题。如何找到没有卖出的动物？不能指向任何SaleAnimal表中的动物。因为动物在卖出之前，不会出现在SaleAnimal表中。

实际上，第四步（连接表）会带来更多的问题。假设查询语句如下：SELECT AnimalID, Name FROM Animal INNER JOIN SaleAnimal ON (Animal.AnimalID = SaleAnimal.AnimalID)。这个JOIN条件去掉了所有符合查询请求的动物。JOIN子句限制输出——和WHERE子句类似。在这个例子中，告诉DBMS返回同时在Animal和SaleAnimal表中出现的动物。但是只有卖出的动物才列在SaleAnimal表中，所以这个查询永远不能返回任何没有卖出的动物。

下面几节描述这个问题的两种解决方案：固定JOIN语句，这样它不再作为限制条件；或者使用子查询。

5.3 Sally的宠物商店

图5-1列出了Sally出于业务需要而必须回答的问题。再次考虑如何使用第4章学习的基本SQL语句回答这些问题。这些问题初看上去并不难。但是，甚至最简单的问题——找到高于平均价格卖出的猫——也比看上去要难。所有这些问题需要附加工具：子查询。

2004年10月1日库存多少只猫？
哪些猫的售价高于平均价格？
哪些动物的售价高于该种类的平均价格？
哪些动物没有卖出？
哪些顾客（至少买过一次东西）在2004年11月1日至2004年12月31日之间没有买任何东西？
哪些买过狗的顾客买过猫的配套产品（任何时候）？

图5-1 较难的问题。即使这些问题很少限制，但更为复杂。为了回答这些问题，需要使用子查询或外连接

5.4 子查询

5.4.1 计算或简单查找

也许查看子查询值的最简单方式是考虑相对简单的问题：哪些猫的售价高于猫的平均价格？如果已经知道猫的平均售价（假定为170美元），查询将变得非常简单，如图5-2的上半部所示。

如果不知道猫的平均SalePrice，可以用一个基本查询把它查出来。把结果写在纸上，然后运行原来的查询。但是，使用子查询，可以更进一步：查询结果（平均值）可以直接传送给原来的查询。用图5-2下半部所示的完整的SELECT AVG查询代替值（170美元）。实际上，任何

时候想插入一个值或者比较，都可以使用子查询来代替，甚至可以深入多层，所以子查询可以包含另一个子查询，如此等等。DBMS通常先计算最内层的查询，然后把结果返回给更高层。

◇ 哪些猫的售价高于猫的平均售价？
- 假定已知平均价格是170美元。
- 通常需要先计算平均。

```
SELECT SaleAnimal. AnimalID, Animal.Category, SaleAnimal.SalePrice
FROM Animal
INNER JOIN SaleAnimal ON Animal. AnimalID = SaleAnimal. AnimalID
WHERE ((Animal.Category="Cat") AND (SaleAnimal.SalePrice>170));
```

```
SELECT SaleAnimal. AnimalID, Animal.Category, SaleAnimal.SalePrice
FROM Animal
INNER JOIN SaleAnimal ON Animal. AnimalID = SaleAnimal. AnimalID
WHERE ((Animal.Category="Cat") AND (SaleAnimal.SalePrice>
  ( SELECT AVG(SalePrice)
    FROM Animal
    INNER JOIN SaleAnimal ON Animal. AnimalID = SaleAnimalID
    WHERE (Animal.Category="Cat") ) ) );
```

图5-2 计算子查询。如果不知道猫的平均SalePrice，可以使用查询语句计算该值。使用了查询，可以把计算结果直接放在原查询中（用括号中的子查询代替原查询中的170）

使用子查询时两个有用的经验是：格式上突出子查询，方便阅读；在插入主查询之前对子查询测试。幸运的是，大部分现代数据库系统可以非常简单地创建子查询，然后剪切、粘贴SQL到主查询中。类似地，如果复杂查询出了问题，把内层的子查询拿出来，单独测试。

5.4.2 子查询和数据集合

部分SQL运算符（IN、ALL、ANY、EXISTS）经常和子查询一起使用。为了便于理解，暂且不谈子查询，从简单的数字开始。考虑图5-3相对简单的例子。列出所有购买下列商品之一：1，2，30，32，33的顾客。

```
SELECT Customer.LastName, Customer.FirstName, SaleItem.ItemID
FROM (Customer INNER JOIN Sale ON Customer.CustomerID = Sale.CustomerID)
  INNER JOIN SaleItemON Sale.SaleID = SaleItem.SaleID
WHERE (SaleItem.ItemID In (1,2,30,32,33))
ORDER BY Customer.LastName, Customer.FirstName;
```



Field	LastName	FirstName	ItemID
Table	Customer	Customer	SaleItem
Sort	Ascending	Ascending	
Criteria			In (1,2,30,32,33)'
Or			

图5-3 集合与相关的查询（In）。可以使用OR语句，但是IN运算符更简单。如果ItemID匹配相关列表中的任何ID，那么WHERE条件为真。列出所有购买下列项目（1，2，30，32，33）的顾客

读者可能考虑用基本SQL语句回答这个问题。在WHERE语句中使用许多OR连接符。可能的写法如下：ItemID = 1 OR ItemID =2 OR ItemID = 30。虽然这种办法对于这个简单的例子有效，但是它增加了额外的输入。更好的办法是把该列表看作一个数据集合，使用IN运算符：WHERE ItemID IN (1,2,30,32,33)。如果ItemID匹配集合中任何一个数值，条件为真。

虽然这个例子中IN运算符减少了输入工作，它实际上有更强大的功能。主要在于列表中的数据可以从子查询获得。例如，管理者可能想知道哪些顾客买了和猫相关的商品。虽然可以通过查看所有的ItemID值找到和猫相关的产品，但是让DBMS来做更为合适。只要修改WHERE子句为：ItemID IN (SELECT ItemID FROM Merchandise WHERE Category = 'Cat')。

注意IN运算符的一个关键性特征：列表中的值必须和测试变量具有相同的数据类型（域）。图5-4的例子把SaleItem.ItemID和SELECT ItemID比较。类似JOIN语句，IN运算符比较相近类型的数据。例如，把ItemID和EmployeeID比较得到的结果毫无意义。实际上，IN运算符可以代替JOIN语句——虽然JOIN查询通常快一些。

```
SELECT Customer.LastName, Customer.FirstName, SaleItem.ItemID
FROM (Customer
  INNER JOIN Sale ON Customer.CustomerID = Sale.CustomerID)
  INNER JOIN SaleItem ON Sale.SaleID = SaleItem.SaleID
WHERE (SaleItem.ItemID In
  (SELECT ItemID FROM Merchandise WHERE Category='Cat')
  );
```

图5-4 在子查询中使用IN。列出所有购买和猫相关商品的顾客。子查询生成和猫相关商品的ItemID列表。如果ItemID出现在那个子查询列表中，主查询匹配

5.4.3 带有ANY和ALL的子查询

ANY和ALL运算符用于比较数字和子集合。在前一小节讲过的IN运算符用于比较值和集合项的列表；但IN用于等值比较。测试项必须和列表中的一项精确匹配。ANY、ALL运算符可以和小于（<）或大于（>）运算符一同使用，并且把测试值和一系列值进行比较。

图5-5举例说明带有ANY的查询语句。很难找到必须使用ANY运算符的查询例子。在这个例子中，如果首先找到列表中的最小值，然后作比较也同样简单。但是，有的时候使用ANY运算符更加清晰。SOME可以代替ANY，它们的功能相同。

ALL运算符和ANY相似，但是测试值必须大于列表中的所有值。换句话说，测试值必须大于列表中的最大值。因此，ALL运算符更加严格。ALL运算符是一个强有力的工具——尤其和等值比较（=）一起使用时。例如，如果想测试一个售货员的所有销售是否都在一天完成，WHERE子句可能会包含如下语句：WHERE EmployeeID = ALL (SELECT EmployeeID FROM Sale WHERE SaleDate = Date())。

```
SELECT Animal.AnimalID, Name, SalePrice, ListPrice
FROM Animal
INNER JOIN SaleAnimal ON Animal.AnimalID =
SaleAnimal.AnimalID
WHERE ((SalePrice > Any

  (SELECT 0.80*ListPrice
  FROM Animal
  INNER JOIN SaleAnimal ON Animal.AnimalID =
  SaleAnimal.AnimalID
  WHERE Category = 'Cat'))

AND (Category='Cat');
```

图5-5 带有ANY和ALL的子查询。这个例子计算卖出猫标价的80%。然后找到售价高于那个水平的所有动物。换句话说，列出了售价接近猫的标价的动物

5.5 差：NOT IN

在业务设置中，经常遇到这样的问题，如图5-6所示：哪些动物没有卖出？这个问题具有欺骗性。乍看起来，只需连接Animal表和SaleAnimal表。然后呢？标准的JOIN语句只显示那些同时出现在Animal表和SaleAnimal表的动物。只要输入了JOIN语句，输出列表自动限制为只有卖出的动物。解决这个问题的一个办法是改变JOIN命令的动作，下节将论述这个问题。

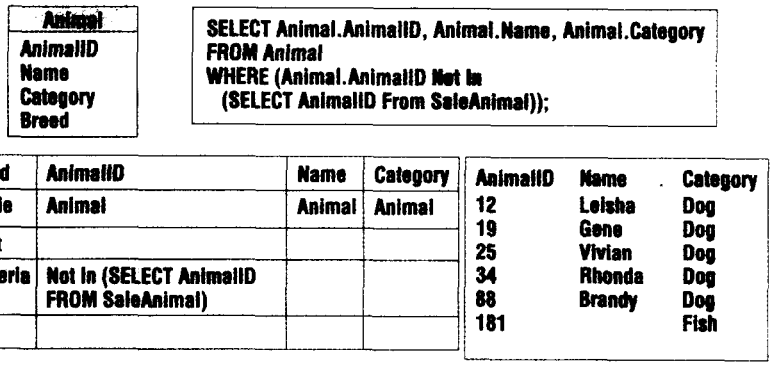


图5-6 使用NOT IN的子查询。哪些动物没有卖出？先列出所有的动物，然后减去卖出的

另一个有效的办法是从数据集合的角度考虑如何解决问题。考虑列表写在纸上该怎么办：一个列出商店购买的所有动物，另一个列出所有已经卖出的动物。如图5-7所示，为了回答这个问题，遍历主Animal表，从中划去已经卖出的动物（出现在SaleAnimal中）。Animal表中剩下的就是没有卖出的动物。

SQL子查询的工作原理与此类似。第一步列出所有的动物——如图5-6中查询主体所示。但是只有不在SaleAnimal表中出现的动物才能列出（子查询）。

SQL NOT IN命令是一个有用的工具。对于涉及多个限制条件的复杂查询更是如此。通过

把相关的限制条件放在子查询中，主查询变得简单易懂。

Animal

ID	Name	Category	Breed
2		Fish	Angel
4	Gary	Dog	Dalmation
5		Fish	Shark
6	Rosie	Cat	Oriental Shorthair
7	Eugene	Cat	Bombay
8	Miranda	Dog	Norfolk Terrier
9		Fish	Guppy
10	Sherri	Dog	Siberian Huskie
11	Susan	Dog	Dalmation
12	Leisha	Dog	Rottweiler

SaleAnimal

ID	SaleID	SalePrice
2	35	$10.80
4	80	$156.66
6	27	$173.99
7	25	$251.59
8	4	$183.38
10	18	$150.11
11	17	$148.47

图5-7　差子查询数据示例。NOT IN语句移除已经卖出的动物，留下没有卖出的

5.6　外连接

截至目前，当涉及多个表时，例子和讨论都集中在INNER JOIN（或等值连接）。这种类型的连接是最常见的，经常在查询中使用。但是，它也有局限性，需要使用另一种JOIN，即外连接（OUTER JOIN）。

为了举例说明OUTER JOIN，考虑前一节的问题：哪些动物没有卖出？如果使用INNER JOIN连接Animal和SaleAnimal表，结果仅仅列出两张表中都有的动物。INNER JOIN命令指示DBMS把Sale Animal表的每一项和Animal表相应的AnimalID相匹配。如果因为某种原因不能匹配，问题中的数据无法显示。

OUTER JOIN改变了两张表的数据匹配方式。特别地，OUTER JOIN描述当一张表中的值在另一张表中不出现时该如何处理。

连接两张表时，必须考虑两种基本情况：（1）左侧表中的值在右侧表没有匹配项，（2）右侧表中的值在左侧表没有匹配项。当然，哪个在左侧哪个在右侧没有实质性区别。但是，列出表之后必须小心，不能混淆。在QBE中，表自然地按照出现的顺序显示。在SQL中，左侧的表先列出。

图5-8的查询是一个典型的左OUTER JOIN（或简写为LEFT JOIN）。在LEFT JOIN中，左侧表的所有行都会在结果中显示。如果右侧表没有匹配值，输出中将填入NULL值。注意LEFT JOIN如何解决识别没有卖出动物的问题。因为查询会列出所有的动物，SaleID为NULL的行表明该动物在Sale表中不存在，即没有卖出。该查询的输出结果如图5-9所示。

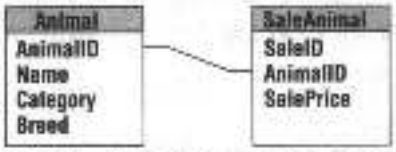
RIGHT JOIN和LEFT JOIN类似。惟一的区别是表的顺序。如果需要右侧表的全部行，就可以使用RIGHT JOIN。为什么不把LEFT JOIN中表的顺序调整一下呢？大部分情况下，可以这么做。但是，如果把几张表连接起来，有时使用RIGHT JOIN可以避免重新对表排序，这样会简单一些。

另外一个连接是全外连接（full OUTER JOIN或FULL JOIN），把左侧表的每一行和右侧表的每一行组合起来。如果行不匹配（根据ON条件），则连接在适当的列插入NULL值。

警告：使用外连接必须小心——尤其是全外连接。两张没有太多公共数据的规模比较大的表的连接结果十分巨大且没有意义。在一个查询中对两张以上表进行OUTER JOIN操作也要小

心。结果依赖于表连接的顺序。

```
SELECT Animal.AnimalID, Animal.Name, Animal.Category
FROM Animal LEFT JOIN SaleAnimal
ON Animal.AnimalID = SaleAnimal.AnimalID
WHERE (SaleAnimal.SaleID Is Null);
```



Field	AnimalID	SaleID	Name	Category
Table	Animal	SaleAnimal	Animal	Animal
Sort				
Criteria		Is Null		
Or				

图5-8 左外连接。哪些动物没有卖出？左外连接包括Animal表（左侧）的所有行，以及SaleAnimal表的所有匹配行。如果动物没有卖出，那么在SaleAnimal表中没有该项，所以相应的项为NULL

ID	Name	Category	Breed	ID	SaleID	SalePrice
2		Fish	Angel	2	35	$10.80
4	Gary	Dog	Dalmatian	4	80	$166.66
5		Fish	Shark	Null	Null	Null
6	Rosie	Cat	Oriental Shorthair	6	27	$173.99
7	Eugene	Cat	Bombay	7	25	$251.59
8	Miranda	Dog	Norfolk Terrier	8	4	$183.38
9		Fish	Guppy	Null	Null	Null
10	Sherri	Dog	Siberian Huskie	10	18	$150.11
11	Susan	Dog	Dalmation	11	17	$148.47
12	Leisha	Dog	Rottweiler	Null	Null	Null

图5-9 左外连接的结果。注意缺失值（Null）即动物没有卖出

最后，注意这些都以SQL92为基础，优点是通俗易懂。但是有的数据库系统不支持这种语法。一种常见的古老的实现OUTER JOIN的技术是使用由星号和等号构成的连接；例如，(*=)表示左外连接，因为星号位于等号的左侧。SQL Server上关于动物的查询如图5-10所示。虽然这个语法和SQL92不同，但作用相同。当阅读老系统开发的查询时注意这种写法。星号（*）容易被忽略，但是它确实改变了查询结果。

旧版Oracle的查询语句使用另外的语法规则表达外连接。图5-10中的WHERE语句变为Animal.AnimalID = (+) SaleAnimal.AnimalID。带括号的加号表示OUTER JOIN，但是注意，加号位于等号的右侧，换句话说，如果需要左外连接，必须把加号放在等号的右侧。这可能和想象的不同，正好和星号标记相反。

```
SELECT ALL
FROM Animal, SaleAnimal
WHERE Animal.AnimalID *= SaleAnimal.AnimalID
And SaleAnimal.SaleID Is Null
```

图5-10 旧版LEFT JOIN语法。哪些动物还没有卖出？注意星号-等号运算符（*=）表示LEFT JOIN。相反地（=*）表示RIGHT JOIN

5.7 关联子查询存在危险

回顾图5-2的例子，哪些猫的售价高于猫的平均价格？这个例子首先使用子查询找到猫的平均售价，然后检查所有猫的售价，如果高于平均价格就显示。有理由把这个查询扩展到关于其他动物的查询问题。管理者可能想找到所有售价高于该动物所在种类的平均售价的动物（例如高于狗的平均价格的狗，鱼和鱼相比，等等）。

虽然这样的业务问题十分必要，但是作为查询语句可能带来一系列严重的问题。开始，如何构造查询并不明显。一种方案是采用图5-2的查询，并且在主查询和子查询中都用“Dog”代替“Cat”，这种方案虽然可行，但必须找到所有的种类，对于每一个种类编辑查询。

为了避免手工输入每个种类，子查询可以从主查询中得到种类，然后计算该种类的平均售价。图5-11是创建这样查询的第一个尝试。难点在于主查询使用了Animal表，子查询也使用Animal表。因此，WHERE限制不起作用。需要指定一张Animal表只用于主查询，另一张只用于子查询。为了做到这一点，需要使用对表重命名（别名），如图5-12所示。

```
SELECT AnimalID, Name, Category, SalePrice
FROM Animal INNER JOIN SaleAnimal ON
Animal.AnimalID = SaleAnimal.AnimalID
WHERE (SaleAnimal.SalePrice>
  (SELECT Avg(SaleAnimal.SalePrice)
  FROM Animal INNER JOIN SaleAnimal ON
  Animal.AnimalID = SaleAnimal.AnimalID
  WHERE (Animal.Category = Animal.Category)) )
ORDER BY SaleAnimal.SalePrice DESC;
```

图5-11 创建关联子查询。列出售价高于该种动物平均价格的动物。子查询需要计算主查询中出现的动物种类的平均售价。但是两张表都叫做“Animal”，这个查询无法进行

```
SELECT A1.AnimalID, A1.Name, A1.Category,
SaleAnimal.SalePrice
FROM Animal As A1 INNER JOIN SaleAnimal ON
A1.AnimalID = SaleAnimal.AnimalID
WHERE (SaleAnimal.SalePrice>
  (SELECT Avg(SaleAnimal.SalePrice)
  FROM Animal As A2 INNER JOIN SaleAnimal ON
  A2.AnimalID = SaleAnimal.AnimalID
  WHERE (A2.Category = A1.Category)) )
ORDER BY SaleAnimal.SalePrice DESC;
```

图5-12 可以运行的关联子查询。注意使用重命名区别两张表。但是永远不要使用这种方法，因为效率极低

图5-12的查询虽然可以运行，但是效率十分低。即使在一个比较快的计算机上，这种查询运行几天也得不到结果！这种查询叫做关联子查询，因为子查询引用了主查询的数据行。对于主查询中的每一条记录，子查询必须重复计算。图5-13列出这个问题。本质上，DBMS从主查询Animal表的最上一行开始。它发现种类是Fish,计算鱼的平均价格（37.78美元）。然后移向下一行，计算狗的平均价格。当DBMS到达第三行，再次碰上Fish，必须重新计算鱼的平均价格（还是37.78美元），对于主查询每一行都要重新计算动物的平均价格十分耗时。为了计算平均值，DBMS必须遍历表中相同种类动物的每一行。考虑一个相对小的查询有100 000行和5种动物。平均起来，每一种有20 000行。每次重新计算平均值，DBMS必须检索100 000*20 000或者2 000 000 000行！

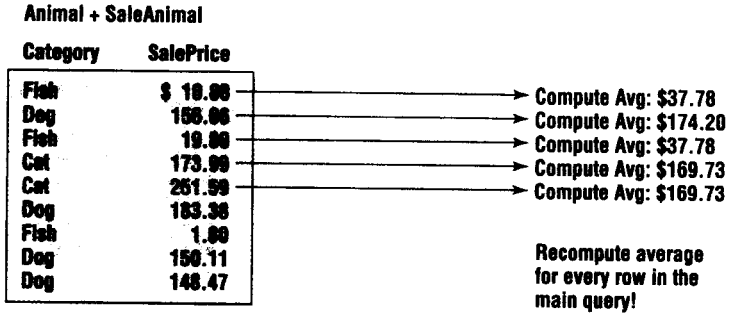


图5-13 关联子查询的问题。对于主查询每一行都要重新计算平均价格。DBMS每次遇到鱼，计算平均价格为37.78美元。这种做法毫无效率（并且十分缓慢），它强制机器每次重复计算平均价格

不幸的是，不能简单地告诉管理者这个重要的业务问题无法回答。存在有效的解决方式吗？问题的解答说明SQL的能力以及写查询语句之前思考问题的重要性。关联子查询的问题在于它不停地重复计算每一种的平均值。考虑手工计算时如何解决这一问题。首先做一张表，把每种动物的平均值列出，然后当需要的时候查找相关项。如图5-14所示，同样的方法可以通过SQL实现。用GROUP BY创建查询计算平均值，保存结果。然后连接Animal和SaleAnimal表作比较。虽然在这个小例子中该方法需要遍历两次查询行，或者读入200 000行，它比相关子查询效率高10 000倍！如果运行新查询需要1分钟，那么关联子查询需要7天才能完成！

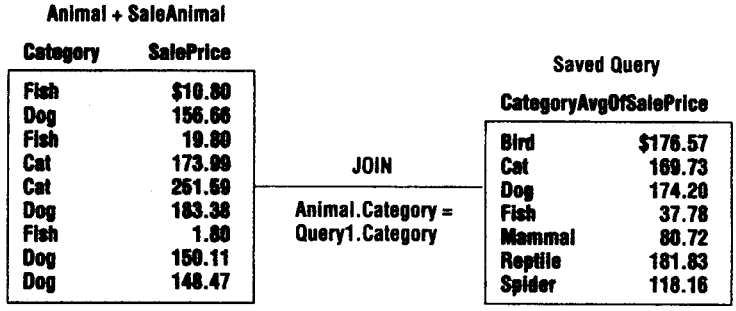


图5-14 更有效的解决方案。用GROUP BY Category创建和存储查询计算平均值。然后连接查询与Animal和SaleAnimal表进行比较

5.8 SQL SELECT的更多特征和技巧

正如读者可能已经注意到的SQL SELECT命令功能强大，并且有很多选项。还有更多的特征和技巧需要知道。业务问题可能很难回答。研究多种多样的案例，对问题和解决方案可以开阔思路。

5.8.1 UNION、INTERSECT和EXCEPT

到目前为止，用到的表包括单独的数据列。JOIN命令把表连起来，使得查询可以显示和比较来自多个表的不同列。偶尔可能遇到另一类问题，即合并相似表的数据行。UNION运算符就是做这件事的。

作为一个例子，假设你在一家公司工作，这家公司在洛杉矶和纽约有办公室。每个办公室维护自己的数据库。每个办公室有包括雇员标准信息的Employee文件。办公室用网络连接，所以可以访问两张表（叫做EmployeeEast和EmployeeWest）。但是公司的管理者经常搜索整个Employee文件——例如，为了决定市场部门的所有雇员工资。一个解决方案是运行两次基本查询（每次一张表）然后手工连接查询结果。

```
SELECT EID, Name, Phone, Salary, 'East' AS Office
FROM EmployeeEast
UNION
SELECT EID, Name, Phone, Salary, 'West' AS Office
FROM EmployeeWest
```

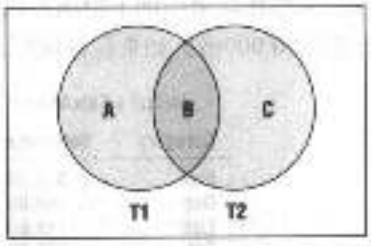
EID	Name	Phone	Salary	Office
352	Jones	3352	45,000	East
876	Inez	8736	47,000	East
372	Stoiko	7632	38,000	East
890	Smythe	9803	62,000	West
361	Kim	7736	73,000	West

图5-15 UNION运算符把两个SELECT语句得到的数据合并在一起。两个SELECT行中的列必须匹配。查询通常保存起来，当管理者需要搜索两张表的时候使用。注意使用一个新的常量列（Office）来表示数据来源

如图5-15所示，更简单的解决方案是使用UNION运算符创建新的查询把两张表的数据合并起来。所有在这个新的查询上进行的搜索和操作把两张表看作一张大表。通过视图合并表，每个办公室可以在自己的系统中修改原始数据；当管理者需要搜索整个公司时，使用保存的查询，可以自动检查来自两张表的当前数据。

创建UNION最重要的是要记住来自两张表的数据必须匹配（例如：EID和EID，Name和Name）。另一个技巧是在SELECT语句中插入常量值。在本例中，这个值表明原始数据来自于哪张表。如果一个查询产生的列在另一个查询中没有而产生问题，这个常量值可以消除这类问题。为了保证两个查询返回相同数目的列，在不包含目标列的查询中插入常量值。确保一个查询中的数据类型和另一个的相同（域必须匹配）。

UNION命令合并两张表中匹配的数据行。默认的命令自动消除重复的数据行。如果希望保留所有的行——即使重复，使用UNION ALL命令。另外两个合并行的选择是EXCEPT和INTERSECT。图5-16表示这三个命令的区别。它们都对数据行操作，维恩图（Venn diagran）表示两张表可能有共



T1 UNION T2	A + B + C
T1 INTERSECT T2	B
T1 EXCEPT T2	A

图5-16 合并两张表中行的运算符。UNION选择所有的行，INTERSECT返回两张表的公共行，EXCEPT返回只出现在一张表中的行

同数据（区域B）。UNION运算符返回在任何一张表中出现的数据，但是两张表的共同行只返回一次。INTERSECT运算符返回两张表都出现的行（区域B）。EXCEPT运算符返回只出现在第一张表的行（区域A）。注意EXCEPT的结果取决于哪张表写在前面。

5.8.2 多JOIN列

有时需要根据多个列的数据连接表。在宠物商店的例子中，每一个动物属于某个类（Cat、Dog、Fish等）。每个类的动物又有不同的品种。例如，Cat可能是Manx、Maine Coon，或者Persian；Dog可能是Retriever、Labrador、或者St. Bernard。类图的一部分在图5-17中重新描绘。注意两条线连接Breed和Animal表。这个关系保证了只有列在Breed表的种类才能赋给Animal的类型。一个实际的商店可能包括一个在Breed表中描述附加的特征（例如注册机构、种类描述或者种类性质）。关键在于表必须通过Category和Breed连接。

在Microsoft Access QBE中，JOIN的构造方法为：标记两列，同时把它们拖拽到Animal表。SQL JOIN命令语法在图5-17中给出。只要在ON语句中列出两列连接就可以了。这种情况下，两个列的集合同时相等，因此语句由AND连接。

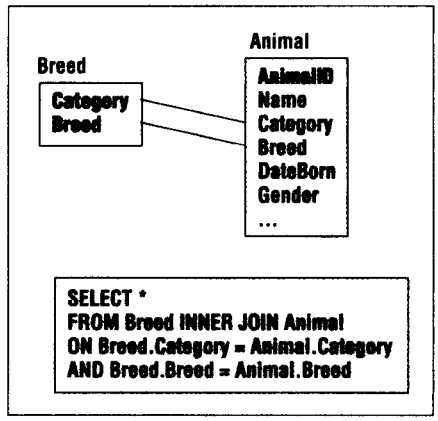


图5-17 多JOIN列。只有种类和品种都匹配的时候，两张表的数据才能连接

5.8.3 自反连接

自反连接或自连接的意思是表自身的连接。表中一列和同一张表的另一列的值匹配。关于Employee表的常见业务问题如图5-18所示。雇员有其管理者。因此管理者的ID在每一个雇员行中存储。表为Employee(EID, Name, Phone, …, Manager)。有趣的特征是管理者也是一个雇员，所以Manager列实际包括EID的值。为了找到管理者相应的名字，需要把Employee表与其自身连接。

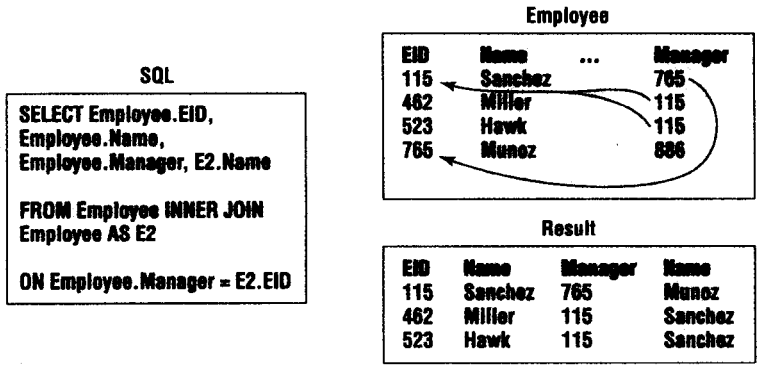


图5-18 Employee表连接自身的自反连接。管理者也是雇员。使用Employee表的第2副本（重命名为E2）获得管理者的名字

操作中惟一需要注意的是ON条件。例如，下面的条件没有意义：ON Employee.Manager = Empoloyee.EID。这个查询返回管理者是自己的雇员，并不是想要的。相反，必须使用两个Empoloyee表的实例，并且对第二副本重命名（假定为E2）。正确的ON条件是ON Employee.Manager = E2.EID。自连接的关键是列包含相同类型的数据，并且为第二副本重命名。

SQL1999中自连接的功能更为强大。考虑雇员的例子，列出所有为某人服务的员工——不仅包括直接的下属，还有为他们服务的人，为该小组工作的人，沿着雇员层次树逐步向下搜索。系统提供了WITH RECURSIVE命令，它有几个搜索数据树的选项。考虑一般的情况，有一张表：Manages(ManagerID, EmployeeID, Title, Salary)。可以从CEO开始列出所有的雇员：

```
WITH RECURSIVE EmployeeList (EmployeeID, Title, Salary) AS
 (SELECT EmployeeID, Title, 0.00
 FROM Manages WHERE Title = "CEO" --starting level
 UNION ALL
 SELECT Manages.EmployeeID, Manages.Title, Manages.Salary
 FROM EmployeeList INNER JOIN Manages
 ON EmployeeList.EmployeeID = Manages.ManagerID)
SELECT EmployeeID, Count(Title), Sum(Salary)
FROM EmployeeList
GROUP BY EmployeeEID;
```

惟一的缺点是现在的DBMS版本不再支持WITH RECURSIVE语句。没有这样功能强大的语句，就需要自己编写代码来完成同样的工作。

5.8.4 CASE函数

SQL92增加CASE函数简化部分类的查询。但是，很多数据库系统没有实现SQL92的所有特征。因此，Microsoft Access 97不支持，Microsoft SQL Server6.5版也不提供这个函数。

也许管理者想根据年龄对Sally的宠物商店中的动物分类。图5-19的SQL语句根据不同的年龄创建了四种类。注意日期计算的方法，用到了今天的日期（Date()）和DataBorn。不论何时执行该查询，都使用当时的日期，把动物分到合适的种类。当然，下一步是针对这个视图运行GROUP BY查询，对每个年龄段的动物计数。

```
Select AnimalID,
  CASE
    WHEN Date()-DateBorn <90 Then "Baby"
    WHEN Date()-DateBorn >= 90
     AND Date()-DateBorn <270 Then "Young"
    WHEN Date()-DateBorn >=270
     AND Date()-DateBorn <365 Then "Grown"
    ELSE "Experienced"
  END
FROM Animal;
```

图5-19 CASE函数把DateBorn转换为年龄段。注意日期的计算方法，生成总是正确的当前值

5.8.5 不等连接

JOIN语句实际上是一个条件。大部分问题直接使用简单的等值条件即可。例如，下面的语句把Customer和Order表连接起来：FROM Customer INNER JOIN Order ON (Customer.CustomerID = Order.CustomerID)。

SQL支持复杂的条件，包括不等连接，比较运算不使用等号，而是不等号（小于、大于）。不等和等值连接共同构成θ连接。

这种类型的连接在某些条件下是有用的。例如，考虑一个常见的业务问题。有一张表AccountsReceivable (TransactionID, CustomerID, Amount, DateDue)。管理者希望对顾客账户分类，按照30、90和120或更长时间确定过期事务的数量。构造这个查询有几种方案。例如，可以写三个独立的查询。但是，如果管理者决定改变业务规则或者建立一个新的分类怎么办？必须有人找到并且修改这三个查询语句。一个更有用的技巧是创建一个新的表记录业务规则和分类方案。如图5-20所示，创建新表LateCategory (Category, MinDays, MaxDays, Charge等)。这张表根据过期时间定义分类。现在使用不等条件连接两张表。首先，使用当前日期计算过期天数（Date() - AR.DateDue)。最后，把过期天数和LateCategory表中的最大最小值比较。

AR(TransactionID, CustomerID, Amount, DateDue)
LateCategory(Category, MinDays, MaxDays, Charge, . . .)

Month	30	90	3%
Quarter	90	120	5%
Overdue	120	9999	10%

```
SELECT *
FROM AR INNER JOIN LateCategory
ON ((Date() - AR.DateDue) >= Category.MinDays)
AND ((Date() - AR.DateDue) < Category.MaxDays)
```

图5-20 不等连接。管理者想把表AccountsReceivable（AR）数据分成三类延期付款。首先，把业务规则/分类存储在一张新表中。把这张表和AR数据通过不等号连接起来

这种做法的本质在于业务规则存储在一个简单的表（LateGategory）中。如果管理者想修改条件或增加新的条件，只需要修改表中的数据。甚至可以创建表单使管理者更容易看到规则，并迅速做出需要的修改。如果使用其他方式，程序员必须重写查询代码。

5.8.6 带有“每一个”的查询需要EXISTS子句

有的查询需要EXISTS条件。考虑这个问题：哪些雇员每一种动物都卖过？这里的关键在于每一种。如果没有计算机该如何回答这个问题？对于每个雇员，列出动物的种类（Bird, Cat, Dog等)。然后浏览AnimalSales表，划掉该雇员卖出的每种动物。完成之后，观察雇员表，哪些雇员的每种动物都划去了（或一张空表)。使用查询语句做同样的事情。

首先，需要使用查询列出某雇员还没有卖出的动物的种类。考虑EmployeeID为5的雇员，如图5-21的查询。注意使用基本的NOT IN查询。还可以使用OUTER JOIN，但是对于3张表，使用NOT IN更容易。

```
SELECT Category
FROM Category
  WHERE (Category <> "Other") And Category NOT IN
    (SELECT Animal.Category
    FROM Animal INNER JOIN (Sale INNER JOIN SaleAnimal
      ON Sale.SaleID = SaleAnimal.SaleID)
      ON Animal.AnimalID = SaleAnimal.AnimalID
    WHERE Sale.EmployeeID = 5)
```

图5-21　列出EmployeeID5的雇员没有卖出的动物种类。使用基本的NOT IN查询

记住，如果这个查询返回结果，那么该雇员没有卖出每一种动物。想要的是那些用这个查询不返回数据的雇员。换句话说，这个查询的结果应该不存在（NOT EXIST）！

下一步检查整个雇员表，看哪些雇员在图5-22的查询中没有返回值。最终的查询如图5-24所示。注意指定的EmployeeID 5已经被和外层循环相匹配的EmployeeID替换。这个查询是一个关联子查询。不幸的是，对于这种问题，无法避免关联子查询。这个查询返回一个雇员(Reasoner)，他每种动物都卖出过。

```
SELECT Employee.EmployeeID, Employee.LastName
FROM Employee
WHERE Not Exists
(SELECT Category
  FROM Category
  WHERE (Category <> "Other") And Category NOT IN
    (SELECT Animal.Category
    FROM Animal INNER JOIN (Sale INNER JOIN SaleAnimal
      ON Sale.SaleID = SaleAnimal.SaleID)
      ON Animal.AnimalID = SaleAnimal.AnimalID
    WHERE Sale.EmployeeID = Employee.EmployeeID)
);
```

图5-22　NOT EXISTS子句的例子。列出卖出每种动物的雇员（“其他”（Other）除外）

图5-22所示的查询类型经常用来回答包括“每一个”字样的查询。有些情况下，一个简单的方法是对每个雇员卖出的动物种类计数。这种方法要求DBMS必须支持Count（DISTINCT）格式。一般来说，这些复杂的查询使用多个查询语句会更好，或者使用第8章描述的数据仓库方法提供的工具。

5.8.7　SQL　SELECT小结

SQL　SELECT命令功能强大，有很多选项。为了帮助读者记住这些选项，图5-23列出了这些选项。每个DBMS的SELECT命令有一个类似的列表，最好参考系统的相关帮助，看看实现细节是否有所不同。记住WHERE子句可以有子查询。

大部分数据库系统对SELECT语句的组成部分的顺序要求非常严格。例如，WHERE语句应该在GROUP BY语句之前。有时这些错误很难发现，所以当遇到令人困惑的错误信息时，注意检查语句顺序的正确性。图5-24也许能够帮助读者记住正确的顺序。还有，一定要逐步建

立查询，这样可以分步测试。例如，如果使用GROUP BY语句，首先检查没有它时选择的结果是否正确。

```
SELECT DISTINCT Table.Column {AS Alias}, . . .
FROM Table/Query
INNER JOIN Table/Query ON T1.ColA = T2.ColB
WHERE (Condition)
GROUP BY Column
HAVING (Group Condition)
ORDER BY Table.Column
{UNION Second Select}
```

图5-23 SQL SELECT选项。记住WHERE语句可以有子查询

Someone	SELECT
From	FROM
Ireland	INNER JOIN
Will	WHERE
Grow	GROUP BY
Horseradish and	HAVING
Onions	ORDER BY

图5-24 记住SELECT运算符正确顺序的方法

5.9 SQL数据定义命令

截至目前，我们仅仅关注数据库的一个方面：查询数据。很清楚，数据库还有很多其他的操作。SQL可以完成所有的常规操作。本节主要讨论数据定义命令，即如何创建和修改数据库及数据库的表。SQL命令实现这些功能比较繁琐。因此，大部分现代数据库系统提供了可视的或菜单的方式来帮助完成这些工作。SQL命令的使用局限于需要自动完成部分工作，或者在独立的程序中修改数据库。

五个最常见的数据定义命令如图5-25所示。当创建新的数据库时，第一步是CREATE a SCHEMA。模式是表的集合。在有的系统中，这个命令等价于创建一个新的数据库。在另一些系统中，它仅仅定义每个用户可以存表的逻辑空间，可能是也可能不是一个物理的数据库。授权部分描述用户并设置密码保证安全。

```
CREATE SCHEMA AUTHORIZATION DBName Password
CREATE TABLE TableName (Column Type, . . .)
ALTER TABLE Table {Add, Column, Constraint, Drop}
DROP {Table TableName | Index IndexName ON TableName}
CREATE INDEX IndexName ON TableName (Column ASC/DESC)
```

图5-25 主要的SQL数据定义命令。大部分情况下，可视的或菜单命令可以代替它们来定义或修改表

CREATE TABLE是主要的SQL数据定义命令之一。用于定义全新的表。基本命令列出表

名和所有列的列名及其数据类型。数据定义命令的格式如图5-26所示。附加的选项包括用DEFAULT命令设置默认值。

```
CREATE TABLE Customer
( CustomerID      INTEGER NOT NULL,
  LastName        VARCHAR(10),
  . . .
);

ALTER TABLE Customer
  DROP COLUMN ZIPCode;

ALTER TABLE Customer
  ADD COLUMN CellPhone VARCHAR(15);
```

图5-26　CREATE TABLE命令定义新表及其包含的所有列。NOT NULL命令通常用于定义表的主码列。ALTER TABLE命令用于对存在的表增加或删除全部列

SQL 92提供了多种标准的数据类型，但是很多数据库版本没有全部实现。SQL92还允许用户使用CREATE DOMAIN命令创建自己的数据类型。例如，为了保持一致性，可以创建一个叫做DomAddress的域，由CHAR(35)构成。任何表用到地址列都可以引用DomAddress。

在SQL 92中，确认主码和外码约束关系。SQL约束是数据库系统的强制规则。为表Order定义主码和外码的语法如图5-27所示。首先，注意每个约束都有一个名字（例如pkorder）。可以使用任何名字，但是最好选择一个容易记忆的以防发生问题。主码约束简单地列出了那些构成主码的列。注意主码的每一列都应该标记为NOT NULL。

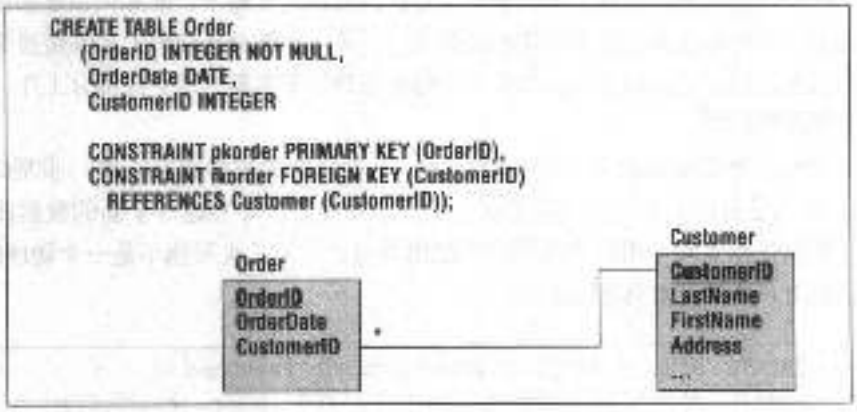


图5-27　在SQL中确定主码和外码。码是DBMS强制定义的约束。主码约束列出了构成主码的列。外码列出了当前表（Order）的列（CustomerID）连接到另一张表（Customer）的列（CustomerID）

如果察看相关类图，更容易理解外码。只有在Customer表中存在的顾客才能下订单，也就是说，Order表中的CustomerID必须存在于Customer表。因此，约束列出了原始的Order表的列，然后指定REFERENCE到Customer表和CustomerID。

ALTER TABLE和DROP TABLE命令用于修改已有表的结构。用DROP命令时要小心，它可能把整个表从数据库中删除，包括数据和结构定义。ALTER TABLE命令相对平和。它用于表中ADD或DELETE的列。很明显，当删除整个列时，存储在该列中的所有数据将被删除。类似地，当增加新列时，所有已有行会包含NULL值。

CREATE INDEX和DROP INDEX命令用于改进数据库的性能。第9章给出使用索引的优点和缺点。一般来说，这些命令对于表是一次性问题，所以通常使用菜单界面更简单。

最后，正如第4章所说的，CREATE VIEW创建并保存新的查询。基本的语法很简单：CREATE VIEW myview AS SELECT…。该命令给出名字并保存任何SELECT语句。而且，这些命令大部分情况下很容易从菜单界面创建并执行。但是，有的时候必须手工创建SQL数据定义语句，所以最好知道该怎么做。

5.10 SQL数据操纵命令

第三类SQL命令体现了SQL的真正功能。SELECT命令检索数据，而数据操纵命令改变表中的数据。基本的命令和语法如图5-28所示。这些命令用于插入数据、删除行、更新（改变）给定单元格的值。使用这些命令时要记住两点：（1）采用了一次一个集合的数据操作方式——而不是一次一行；（2）利用SELECT和WHERE语句前面的用法。

```
INSERT INTO target (column1, column2, . . .)
  VALUES (value1, value2, . . .)

INSERT INTO target (column1, column2, . . .)
  SELECT . . . FROM . . .

DELETE FROM table WHERE condition

UPDATE table
  SET Column1=Value1, Column2=Value2, . . .
  WHERE condition
```

图5-28 常见的SQL命令，用于增加、删除、修改已有表的数据。这些命令对整个数据集合进行操作，利用SELECT、WHERE语句以及子查询

5.10.1 INSERT和DELETE

正如图5-28所示，INSERT命令有两种变化。第一种（VALUES）一次插入一行数据值。除了在某些程序中，这种用法很少使用。大部分数据库系统提供可视的或表格式的数据输入系统，使输入和编辑单行数据非常容易。正如第6章所讲的，还可以构造表单，方便用户输入和编辑单行数据。

INSERT的第二种用法在从一张表向另一张表（目的表）复制数据的时候特别有用。任何SELECT语句都是可以的，包括带有子查询的语句，这样它的功能就比看上去更强。例如，在宠物商店数据库中，可能把较老的Animal文件移到另一台机器上。为了移动所有2004年1月1日之前的动物记录，将会用到图5-29所示的INSERT命令。子查询根据订购日期选定动物。INSERT命令把Animal表中的相关行复制到已有的OldAnimal表中。

```
INSERT INTO OldAnimals
SELECT *
FROM Animal
WHERE AnimalID IN
  (SELECT AnimalOrderItem.AnimalID
  FROM AnimalOrder INNER JOIN AnimalOrderItem
  ON AnimalOrder.OrderID = AnimalOrderItem.OrderID
  WHERE (AnimalOrder.OrderDate= '01-Jan-2004') );
```

图5-29 INSERT命令复制旧的数据行。使用子查询确定需要复制的行

图5-29的查询把部分指定行复制到新表中。下一步是从主要的Animals表中删除这些行以节约空间，提高性能。DELETE命令可以轻松完成这项工作。如图5-30所示，只要把查询的前两行（INSERT和SELECT）换成DELETE即可。注意不要改动子查询。可以使用剪切-粘贴功能来删除已经备份的行。一定要注意DROP和DELETE命令的区别。DROP命令删除整个表。DELETE命令删除表中的行。

```
DELETE
FROM Animal
WHERE AnimalID IN
  (SELECT AnimalOrderItem.AnimalID
  FROM AnimalOrder INNER JOIN AnimalOrderItem
  ON AnimalOrder.OrderID = AnimalOrderItem.OrderID
  WHERE (AnimalOrder.OrderDate<'01-Jan-2004') );
```

图5-30 DELETE命令删除旧的数据。使用剪切和粘贴保证子查询和前一个查询相同

5.10.2 UPDATE

UPDATE命令的语法和INSERT、DELETE命令类似，可以使用WHERE子句的所有功能，包括子查询。UPDATE命令的关键在于它每次对所有行的集合操作。用WHERE子句指定需要修改哪些行。

在图5-31的例子中，管理者希望上调猫和狗的ListPrice。猫的价格上调10%，狗的价格上调20%。因为有两种不同的动物，通常需要两个独立的UPDATE语句。但是，这个例子正好适合CASE函数。可以减少到只须一个UPDATA语句，方法是用CASE语句为Cats选定1.10同时为Dogs选定1.20，代替值1.10和1.20。

```
UPDATE Animal
SET ListPrice = ListPrice * 1.10
WHERE Category = 'Cat';
```

```
UPDATE Animal
SET ListPrice = ListPrice * 1.20
WHERE Category = 'Dog';
```

图5-31 UPDATE命令的例子。如果不用CASE函数，使用两个独立的语句为猫的价格上调10%，为狗的价格上调20%

UPDATE语句还有其他的功能。例如，可以同时修改几个列。只要用逗号把列分开。还可以利用表中的行或查询做计算。例如，动物的标价可以考虑动物的年龄，该命令为SET ListPrice = ListPrice * (1 - 0.001 * (Date() - DateBorn))。这个命令规定了动物自出生起，每天价格下降千分之一。

注意最后一个例子，用内置函数Date（）给出了当天的日期。大部分数据库系统提供多个内置函数可以用在任何计算中。这些函数没有标准化，但是可以从系统的Help命令得到列表(即语法规范)。在Microsoft Access中，可以通过在Help中搜索Funtion和Reference得到完整的列表。Date、String和Format函数尤其有用。

使用UPDATE命令时要记住，计算中用到的所有数据必须在查询中的某一行出现。没有办法引用表中前面一行或后面一行。

5.11 质量：查询检查

复杂查询最大的挑战在于即使有错误也能得到结果。问题是这个结果并不是希望得到的。保证结果正确的惟一办法就是彻底理解SQL，仔细构造并测试查询。

图5-32列出了处理复杂查询的基本步骤。第一步是把复杂的查询分解成几部分，如果查询中包括子查询，这一步尤为重要。对每一个子查询单独检查和测试。对于复杂的布尔条件也可以这么做。从简单的条件开始，检查结果，然后增加新的条件。如果子查询正确，使用剪切和粘贴的办法把它们连在一起构成主查询。一定要避免关联子查询。必要时，把初始的查询保存为视图，然后用一个全新的查询连接视图的结果。第三步建立样本数据检查查询。找到或者创造代表不同情况的数据。

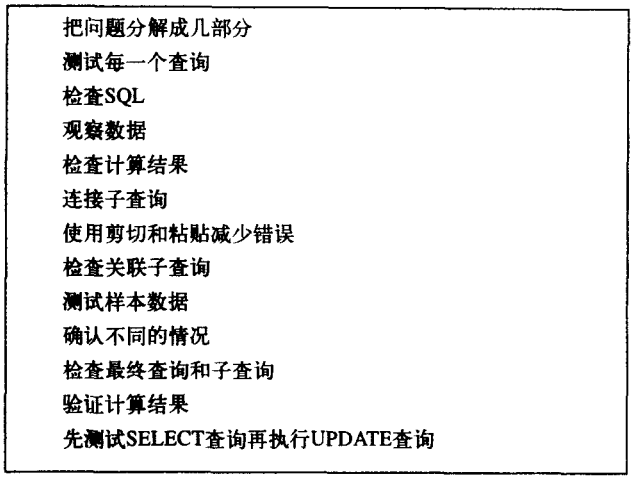


图5-32 构造可靠查询的步骤。执行UPDATE和DELETE查询之前，一定要保证数据库有最新的备份

考虑图5-33的例子：列出买狗和猫产品的顾客。该查询由四种情况构成：

1) 顾客在一次购物中购买了狗和猫产品。

2) 顾客买了狗，在另一个时间买猫产品。

3) 顾客买了狗，从未买猫产品。

4) 顾客从未买狗，但是买猫产品。

```
SELECT DISTINCT Animal.Category, Sale.CustomerID
FROM Sale INNER JOIN (Animal INNER JOIN SaleAnimal
  ON Animal.AnimalID = SaleAnimal.AnimalID)
  ON Sale.SaleID = SaleAnimal.SaleID
WHERE (((Animal.Category)='Dog'))

  AND Sale.CustomerID IN (

  SELECT DISTINCT Sale.CustomerID
  FROM Sale INNER JOIN (Merchandise INNER JOIN SaleItem
    ON Merchandise.ItemID = SaleItem.ItemID)
    ON Sale.SaleID = SaleItem.SaleID
  WHERE (((Merchandise.Category)='Cat'))
);
```

图5-33　查询示例：哪些顾客买了狗也买了猫产品（在任何时间）？每个查询分别构造。然后把它们用SQL连在一起并加到链接上。用样本数据测试结果

因为只有四种情况，可以设置数据分别测试。如果有上千种情况，只能测试可能性较大的情况。

构造查询的最后一步涉及数据操纵查询（例如UPDATE）。首先创建SELECT查询，搜索需要修改的行。检查并测试保证这些行就是要修改的。当查询确认正确后，要有数据库的最新备份——或者至少要有计划修改表的备份。现在可以把SELECT查询改为UPDATE或DELETE语句并执行。

小结

要牢记SQL的操作对象是数据集合。SELECT命令返回符合某种条件的数据集合。UPDATE命令改变数据，DELETE命令删除特定集合的数据行。集合可以由简单的WHERE子句确定，也可以由带有多个子查询和多个表的复杂条件确定。理解SQL的关键在于把WHERE子句看成数据集合的定义。

子查询功能很强。对于外层循环的每一个值，内层循环都要重复一遍的关联子查询必需小心避免。这种情况下，先创建视图，把中间结果保存在一个独立的查询中。还有，在把子查询插入到最终查询之前，要分别进行子查询测试。

在日常情况下，数据保存在一张表中而不是另一张。例如，可能需要最近没有下订单的顾客列表。如果DBMS没有参照完整性也会产生类似的问题——需要找到在顾客表中没有匹配项的顾客的订单。外连接（或者NOT IN子查询）用于这些情况。

每个数据库系统都有内部函数可以用在SQL语句中。不幸的是，这些函数没有标准化，所以每个DBMS使用不同的语法。但是，一些标准的函数通常可以得到并用在业务查询中。有一些处理日期和时间的重要函数。例如，Month函数从一般的日期列中获得月份，允许查询语句计算月度总额。

最重要的要记住，即使构造了错误的查询语句，大部分情况仍然可以执行。不幸的是，结果并不正确。也就是说，必须万分小心地构造查询，并反复检查。从一个小的查询开始，然

后不断地增加元素，直到完成。在构造UPDATE或DELETE查询的过程中，先构造SELECT语句，检查结果，然后改成UPDATE或DELETE。

开发漫谈

Miranda发现有的业务问题比其他的复杂得多。SQL子查询和外连接经常用来回答这些问题。反复练习SQL子查询直到彻底理解，它们可以节省数百小时的工作。考虑一下本章有些问题如果编写代码需要多长时间！作为课程作业，创建几个查询检验一下技能，包括子查询和外连接，构造并测试SQL UPDATE查询来修改数据集合。可以使用SQL创建并修改表。

关键词

ALL	DELETE	LEFT JOIN
ALTER TABLE	DROP TABLE	嵌套查询
ANY	等值连接	OUTER JOIN
CASE	EXCEPT	自反连接
约束	EXISTS	RIGHT JOIN
关联子查询	FULL JOIN	模式
CREATE DOMAIN	IN	自连接
CREATE SCHEMA	不等连接	子查询
CREATE TABLE	INSERT	UNION
CREATE VIEW	INTERSECT	UPDATE

复习题

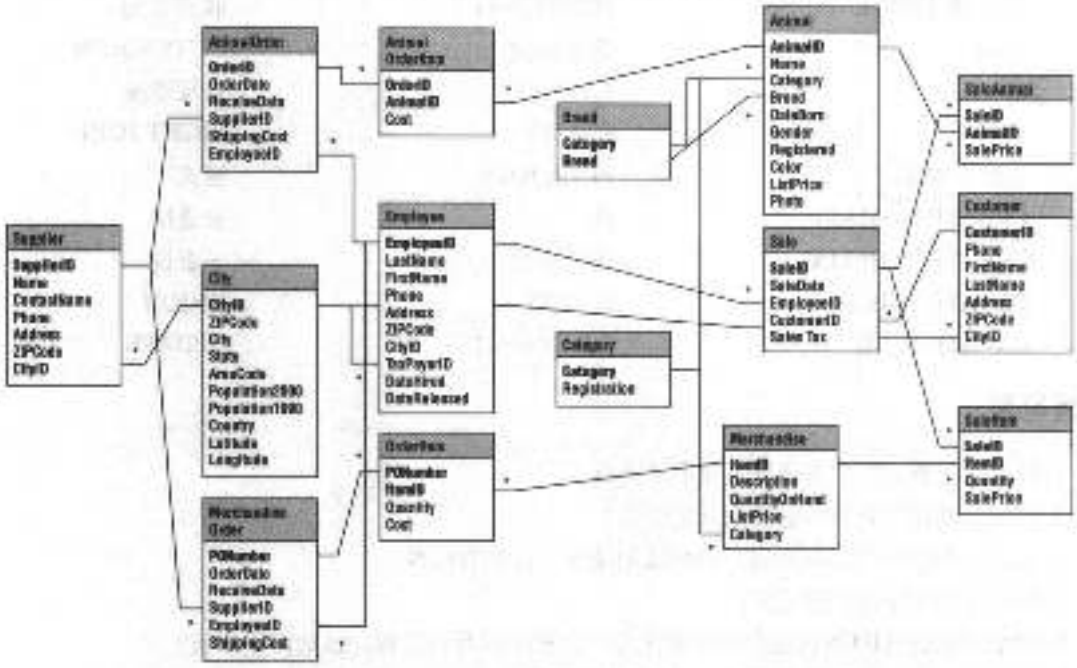
1. 什么是子查询？什么情况下使用子查询？
2. 什么是关联子查询？它的问题是什么？
3. 如何找到不在列表中的项？例如最近没有下订单的顾客。
4. SQL命令可以分成哪三类？
5. 如果一张表的JOIN列包含的数据在另一张表中没有相关列，该如何连接表？
6. 如果两个或多个列需要匹配，该如何连接表？
7. 什么是不等连接？什么情况下使用不等连接？
8. 什么是SQL UNION命令？什么情况下使用该命令？
9. 什么是自反连接？给出可能使用自反连接的例子。
10. SQL CASE函数的目的是什么？
11. 基本的SQL数据定义命令是什么？
12. 基本的SQL数据操纵命令是什么？
13. UPDATE和DELETE命令如何和SELECT语句相似？

练习

Sally的宠物商店

下面编号为1~25的问题基于宠物商店数据库的表，写出SQL语句回答这些问题。在数据库上测试查询。提示：如果把问题分解成多个查询，很多问题会简单些。

1. 列出标价比所有产品平均标价都高的产品。
2. 平均来看，哪个卖出的时间短：雄性猫还是雌性猫？
3. 列出卖出时间比平均时间长的猫。
4. 哪些商品的平均销售价格比平均购买价格高50%以上？
5. 用占全部雇员所有商品销售百分比的形式列出雇员和他们总的商品销售。
6. 平均来看，哪些供应商收取了占商品订单总数最高的运输成本？
7. 哪位顾客消费动物和商品最多？
8. 哪些顾客在5月份购买超过100美元的商品，而在10月份购买超过50美元的商品？

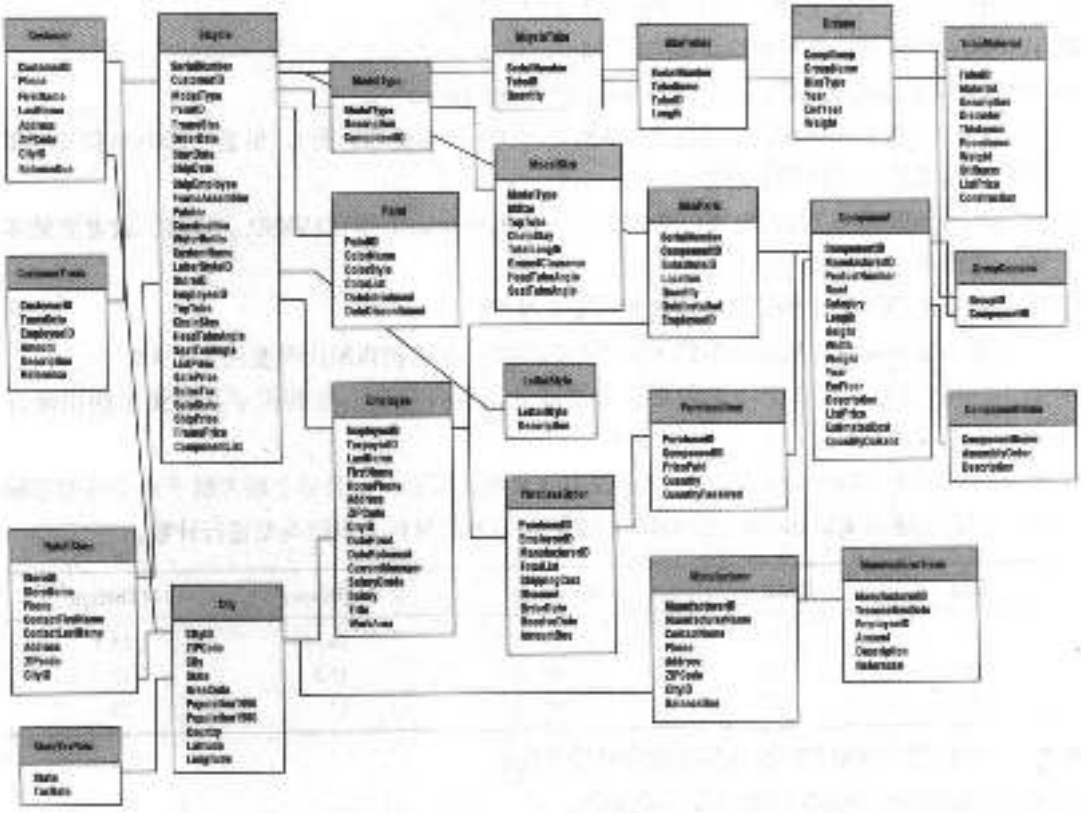


9. 列出在第一季度买狗，并且在第四季度买狗食的顾客。
10. 从1月1日到7月1日价格上涨的罐装狗食净变化量是多少？
11. 7月份哪些标价高于50美元的商品没有卖出？
12. 哪些现存的数量多于100件的商品没有在2004年订购？使用OUTER JOIN回答。
13. 哪些现存的数量多于100件的商品没有在2004年订购？使用子查询回答。
14. 哪些现存多于500件的猫产品没有在7月份卖出？
15. 宠物商店的哪种狗从未售出？使用OUTER JOIN回答。
16. 宠物商店的哪种狗从未售出？使用子查询回答。
17. 列出向Gibson报告的雇员列表。

18. 保存查询回答练习7：每个顾客消费的总金额。如表所示，创建查询根据销售额对顾客分类。构造查询语句列出第一个查询的每位顾客，并显示合适的标签。

类 别	下 限	上 限
差	0	200
良	200	800
优	800	10 000

19. 列出6月份所有宠物商店的供货商。标明他们卖动物还是商品。
20. 列出花在动物上的钱比在商品上的（总数）多七倍的顾客的州。
21. 创建如练习18所示的表，写一个查询。
22. 向练习18所示的表中插入第一行数据，写一个查询。
23. 把练习18所示表第一行的上限修改为400，写一个查询。
24. 删除练习18表的第一行，写一个查询。
25. 为Employee表创建备份，使用删除查询从备份中删除所有的数据。写出查询把原始雇员表复制到新表。复制在11月份没有卖出商品的雇员。



Rolling Thunder自行车

下面编号为26-50的问题是基于Rolling Thunder数据库的表，写出SQL语句回答这些问题，并用Access实现查询。

26. 哪些赛车2003年的销售价格高于2002年的平均价格?
27. 在2002年，哪些自行车相当于平均产量的两倍（以2002年为准）?
28. 根据占总销量的比例按照自行车类型列出2003年销量。
29. 2002年Campy R2齿轮变速器在赛车上的安装百分比是多少?
30. 列出至少购买一辆公路自行车和一辆山地自行车（任何时间）的顾客。使用子查询。
31. 列出至少购买一辆公路自行车和一辆山地自行车（任何时间）的顾客。使用JOIN和保存的查询。与练习30比较查询性能。
32. 列出购买无避震山地自行车，但没有购买全避震山地自行车的顾客。使用子查询。
33. 列出购买无避震山地自行车，但没有购买全避震山地自行车的顾客。使用OUTER JOIN并与练习32比较查询性能。
34. 在2004年，哪个没有销售（安装）的部件库存价值最高（根据成本）?
35. 创建一个厂商联系表，记录所有加利福尼亚的制造商和零售商。只包括VendorName和Phone列。零售商只包括2004年至少卖出一辆自行车的店。
36. 列出所有向Venetiaan报告的雇员。
37. 列出公司至少购买比2000年6月30日之前多25%的部件。
38. 列出购买总值超过当月销售总值的部件和年/月。
39. 在哪些年平均制造时间超过了所有年份的平均制造时间?
40. 列出在1997年至1999年间至少消费7 000美元（不包括税和运费），但是在2003年和2004年却没有购买任何自行车的顾客。
41. 列出所有为2003年7月前三天订购的赛车工作的雇员。工作包括装配、喷漆、或者安装零件。指出每个雇员从事的工作。
42. 哪个雇员从2001年到2002年的销售额增长率最高?
43. 对于每个型号来说，从2002年到2003年和2004年，年度销售额比例变化有多大?
44. 自2000年以来，有多少顾客购买了公路或者比赛自行车，也购买了某种类型的山地自行车?
45. 创建一张新表（CustomerSize），输入表中的数据。写出查询为每个顾客赋予适当的型号标注，然后创建查询对2003年和2004年制造的各型号的自行车根据类型进行计数。

Size	RoadLow	RoadHigh	MTBLow	MTBHigh
小	45	52	14	15.5
中	52	57	15.5	17
大	57	65	17	23

46. 利用CREATE TABLE写查询创建练习45描述的表。
47. 写查询向练习45所示的表插入第一行数据。
48. 写查询把练习47中小型自行车的RoadLow值改为40。
49. 写查询删除练习46表的第一行。
50. 写查询删除练习46创建的整个表。

参考网站

网站	描述
http://www.acm.org/sigmod/	计算机学会——特别兴趣组：数据管理。
http://www.acm.org/dl	ACM数字图书馆包括上千篇全文检索文章。察看你的图书馆是否有订阅。

补充读物

Celko, J. *Joe Celko's SQL Puzzles & Answers.* San Mateo: Morgan Kaufmann, 1997. [Challenging SQL problems with solutions.]

Iseminger, D., ed. *Microsoft SQL Server 2000 Reference Library.* Redmond: Microsoft Press, 2000. [The hardcopy documentation that does not ship with SQL Server.]

Kreines, D., and K. Jacobs. *Oracle SQL: The Essential Reference.* Cambridge, MA: O'Reilly & Associates, 2000. [Reference book for Oracle's version of SQL.]

附录：编程简介

很多书介绍如何学习编写计算机程序。附录的目的是复习编程要点，指出DBMS编程的几个重要特征。如果你是一个编程新手，应该找其他的书理解编程的细节和逻辑。

1 变量和数据

数据库环境下的编程最重要的是处理三种数据：（1）存储在表中的数据；（2）在表单和报表中控制的数据；（3）传统的数据变量，保存临时结果。第3章重点描述存储在表中的数据。第6章描述如何创建表单和数据控件。第8章给出更多的关于这三种数据在构建应用程序时交互作用的细节。现在，学习基本程序变量的基本内容。

程序可以创建变量保存数据。一个程序变量就像一个小箱子：它可以保存将要使用或转移的值。变量有惟一的名字。更重要的是变量具有固定的数据类型。常见的变量类型如图5-1 A所示。一般地，可以分为三种：整数（1，2，−10，…）、实数（1.55，3.14，…）和字符串（'123 Main Street'，'Jose Rojas'，…）。

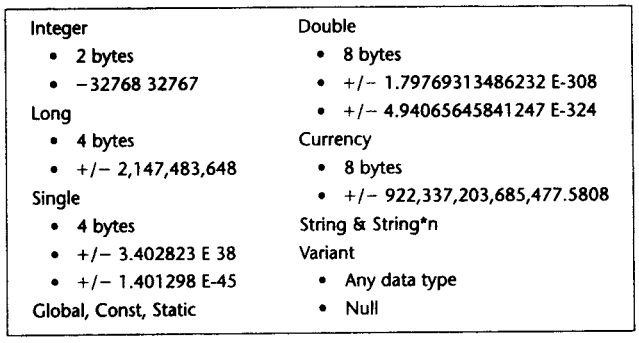


图5-1A 程序变量类型。Microsoft Access的变量规模。注意货币变量有助于防止舍入误差

每一种变量占有固定的存储空间。这个空间决定了变量可以表示的数据的范围。准确的大小依赖于DBMS和操作系统。例如，一个简单的整数占用2个字节空间，即16比特。因此，可

以表示2^{16}个值或者−32 768至32 767之间的整数。实数可以有小数值。通常有两种：单精度和双精度。如果不需要太多的变量，最好选择比较大的变量（长整数和双精度实数）。虽然双精度变量需要更大的空间，处理起来也比较慢，但它们有扩展的空间。如果选择的变量太小，就有可能使程序崩溃或得到错误的结果。例如，用2字节的整数对顾客计数可能发生错误——因为公司很可能拥有超过65 000位顾客。同样的道理，对于货币值应该使用Currency数据类型。不仅可以表示大数，而且可以避免浮点数常见的舍入误差。

Microsoft Access提供可变数据类型，用于转换数据库或输入表单中的数据。可变数据类型有两个独特的属性：（1）可以保存任何类型的数据（包括日期），（2）可以识别数据缺失。如果把变量想象为一个保存值的盒子，就可以发现当盒子为空的时候盒子是有用的。标准的程序变量不检查这个条件。Access提供IsNull函数，它返回一个可变数据变量是否尚未赋值。

2 变量域

变量的域和生存期在编程中十分重要，尤其是在事件驱动的环境里。变量域指在哪些地方可以访问变量，即哪些程序或代码可以访问变量中的数据。生存期指变量何时创建和销毁。这两个属性是相关的，一般是自动设置的。但是，可以通过改变声明变量的方式重载标准程序。

所有的数据变量应该明确声明，在使用前必须确定。最常见的是使用Dim语句。例如，Dim i1 As Integer。在默认的情况下，变量的生存期和域依赖于创建的位置。最常见的，变量在事件函数中创建，即局部变量。函数开始后，局部变量创建。函数内部的代码可以使用该变量。但其他函数的代码不能访问该变量。当函数结束后，局部变量及其数据被销毁。

图5-2 A表示一个表单中有两个按钮。每一个按钮响应一个Click事件，因此，定义了两个函数。每个函数拥有一个叫做i1的变量，但是这两个变量完全无关。实际上，变量在点击按钮之后创建。把函数看作两个房间，当你走进一个房间，只能看见这个房间里的数据。当你离开这个房间，数据即被销毁。

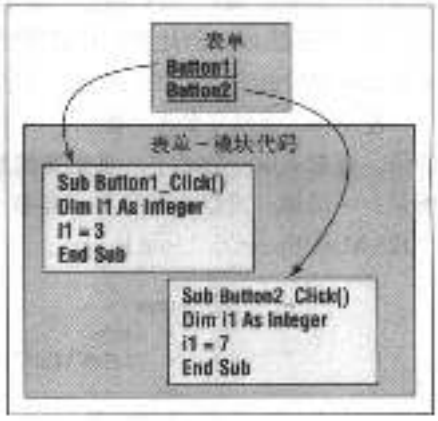


图5-2A 变量域和生存期。每个事件的函数拥有独立的变量。在每次点击按钮时创建和销毁

但是，如果希望代码结束的时候不销毁数据，或者希望从其他函数访问该变量，该怎么办？办法有两个：（1）声明变量为静态类型，改变生存期；（2）在另外的地方声明变量，改变域。应该尽量避免声明静态变量，除非绝对必要（这是非常少见的）。如果变量是静态的，它将一直保持前一次调用的值。在示例中，每次点击按钮，i3保持前一次的值。可以用这种办法对按钮点击次数计数。

更有用的技术是改变变量声明的位置。图5-3 A展示了事件函数在表单或模块中定义，是相关函数的集合。变量i2在整个表单或模块（位于Access的常规部分）中定义。变量生存期由表单决定，也就是说，当表单打开和关闭的时候，变量创建和销毁。表单中的所有函数都在变量的域中，可以看见和修改该变量的值。另一方面，其他表单或模块的函数不知道这个变量的存在。

过程或函数也有域。一张表单中定义的过程可以由这个表单的其他过程调用。如果需要从许多不同的表单或报表访问变量或过程，那么在一个独立的模块中定义，并声明为全局的（或公共的）。

使用全局或公共变量必须小心。程序员试图修改代码时可能不知道变量在其他过程中使用，可能偶然地破坏一个重要的值。在表单中，全局变量的主要目的是把一个事件中的值传递给另一个。例如，需要保持一个文本控件的原始值——在用户修改之前，并把它和新值相比较。这需要一个全局变量，因为两个独立的事件用到了文本控件：(1) 用户先向控件输入，(2) 用户修改数据。

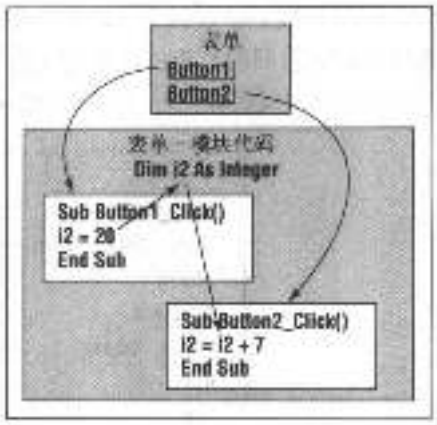


图5-3A 全局变量。在表单的常规部分定义的变量可以被该表单（或模块）的任何函数访问

3 计算

变量最主要的目的之一是执行计算。记住一次计算只涉及单独的变量——一份数据。如果需要在整个表中操作数据，最好使用第5章描述的SQL命令。但是，有时需要更复杂的计算。

标准的算术运算（加、减、乘、除）如图5-4 A所示。这些运算符在大部分程序语言中很常见。一些非标准但有用的运算符包括幂运算（指数，例如：2^3=2*2*2=8），以及整数除法（例如：9\2=4），总是返回整数。模函数返回整除的模数或整除后的余数（例如：15mod 4=3，因为15−12=3）。如果希望知道多少对象可以填入固定的空间时，就可以使用最后两个函数。例如，如果每页有50行，而需要打印185行，那么185\50=3页，第四页还有185 Mod 50=35行。

- 算术运算 + − * /
- 幂 ^
  2^3 = 2*2*2 = 8
- 整数除法
  9 \ 2 = 4
- 模或求余
  15 Mod 4 = 3, or 12 Mod 3 = 15

图5-4A 常见的算术运算符。加（+）、减（−）、乘（*）、除（/）。幂和整数运算在特殊任务中经常使用。例如，整数的算术运算在对对象分组中是有用的

大部分语言支持字符串变量，用于保存基本的文本数据，例如名字，地址或者简短的消息。字符串是字符的集合（或数组）。有时需要进行字符串变量计算。在文本数据上如何进行计算呢？最常见的技术是连接（或加）两个字符串。例如，如果FirstName为"George"，LastName为"Jones"，那么FirstName & LastName为"GeorgeJones"。如果希望在名字间留有空格，那么需要加上它：FirstName &" " & LastName。

图5-5A列出了一些常见的字符串函数。可以从Access的帮助系统中学到更多的字符串函数和语法。常用的函数包括Left、Right、Mid，分别检查字符串的相应部分。例如，可能希望查看字符串左起前5个字母。

4 标准内部函数

回忆数学课学过的，在很多情况下使用的一些常见的函数。如图5-6 A所示，这些函数包括标准的三角函数和对数函数，用于制图和测量相关的过程。还需要函数计算平方根和绝对

值。Int（整数）函数用于去掉数字的小数部分。大部分语言还提供随机数生成器，可以随机地生成0和1之间的数字。如果需要其他范围的数字，可以通过简单的转换得到。例如，为了生成40和90之间的数，可以用下面的函数：y = 40 + (90 - 40) * Rnd。

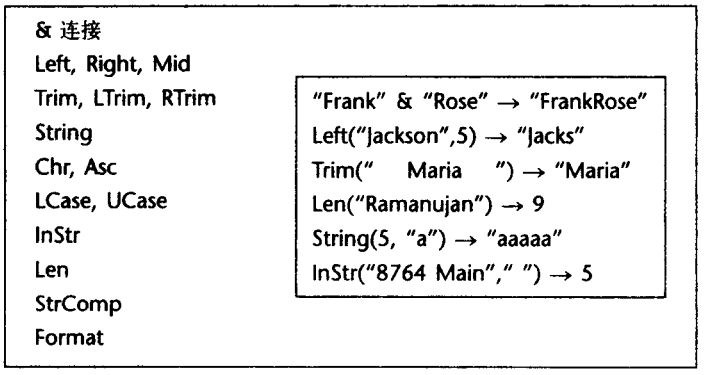


图5-5A　常见的字符串函数包括：连接字符串、截取部分、查找字符、改变大小写、比较两个字符串、将数字格式化为字符串变量

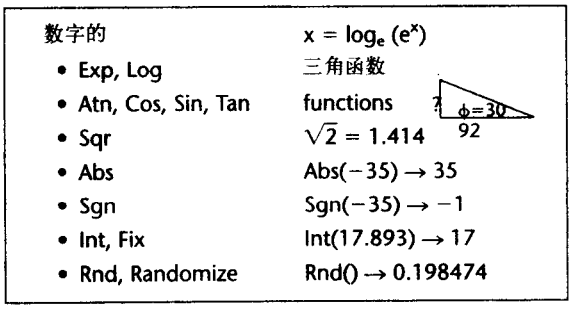


图5-6 A　标准的数学函数。即使在业务应用程序中，数学函数也经常使用

在数据库环境中，经常需要计算和修改日期。提供当前日期（Date）和时间（Now）的函数是有用的。业务处理中两个有用的函数是DateAdd和DataDiff。如图5-7 A所示，DateAdd函数把一段时间和给定的日期相加，得到未来的某个日期。经常需要计算两个不同日期相隔几天。不仅如此，VBA的函数还可以计算月和星期的数目。甚至可以计算两个日期间有多少星期五。

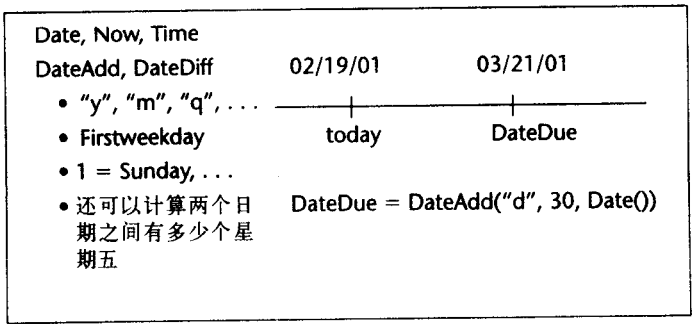


图5-7A　日期和时间函数。业务问题经常需要计算两个日期之间的天数，或者为了计算付款日期把一段日子和一个日期相加

可变数据类型可以包含任何数据类型，比如日期、数字和字符串。有时，需要准确地知道变量的数据类型才能进行操作。有的函数，例如IsDate、IsNumberic和VarType提供类型信息。这些函数可以在使用前检查变量是否具有合适的类型，以避免发生错误。

可变数据类型的另一个功能是测试缺失数据。传统的程序变量不支持这个功能。Microsoft Access为此提供了两个函数：IsNull和IsEmpty。IsFull函数测试数据表值和检查用户是否向数据项控件输入值。

5 输入和输出

在传统的编程中，处理输入和输出是个关键的问题。现在，这些问题仍然重要，但是DBMS可以处理大部分数据处理程序，操作系统处理大部分用户界面。常见的表单、报表（见第6章）用于大部分输入和输出工作。

牢记Windows界面的重要特点是用户控制数据输入流，也就是说，设计者提供表单，用户按照自己的流程工作而不受干扰。偶尔可能打断用户——提供信息或者获取某些数据。一个常见的原因是发出错误消息。这种功能依赖于两个基本函数：MsgBox和InputBox。如图5-8 A所示，消息框包含按钮。按钮常用于提示用户该如何处理问题和错误。

InputBox是一种特别的表单，可以用于输入少量文本或一个数字。用户和开发者都对它没有过多的控制。大部分情况下，最好设计自己的空表单。这样可以使用多个文本框，可以指定和控制按钮。InputBox经常在开发时间有限的情况下作为临时使用。




```
MsgBox ("This is a message box",
        vbYesNoCancel + vbInformation,
        "Sample Box")
```

图5-8A 消息框样例。消息框打断用户，提示用户有限的选择。它通常用于处理错误和问题

6 条件

测试并响应条件的能力是编写自己的过程的一个最常见的原因。基本的条件语句（if … then … else）相对容易理解，其结构如图5-9 A所示。条件可能为真或假。如果为真，执行一组语句，否则，执行第二组。

```
If (Condition1) Then
  statements for true
Else
  statements for false
  If (Condition2) Then
    statements for true
  End If
End If
```

图5-9A 条件。基本的条件简单明了。条件语句的缩进突出了关系

条件可能很复杂，尤其是条件包括多个AND和OR连接符。有的开发者使用NOT语句对条件值取反。写条件语句时要小心。必须保证条件值计算的正确性，保证其他开发者容易读懂代码。

使用括号可以指定计算顺序。对于复杂查询，创建样本数据及测试条件语句，还要缩排代码。对于嵌套条件，缩排尤为重要。在嵌套条件中，一个条件语句包括另一个条件语句。

Select Case语句是一种特殊的条件语句。许多过程需要计算一组相关的条件。例如，考虑使用带有三个按钮（是，否，取消）的消息框的情况。必须针对每个选项测试用户选择。

图5-10 A显示使用嵌套条件代码的样子。

图5-11 A用Select Case语句解决同样问题。注意这份代码可读性强。考虑如果有10个选项会发生什么。If … then 代码变得更糟糕，但是Select Case代码只要在列表的最下端增加新行即可。

```
response = MsgBox (...)
If (response = vbYes) Then
  'statements for Yes
Else
  If (response = vbNo) Then
    'statements for No
  Else
    'statements for Cancel
  End If
End If
```

图5-10A　测试用户响应的嵌套条件。随着条件的增加，代码可读性变差

```
response = MsgBox(...)
Select Case response
  Case vbYes
    'statements for Yes
  Case vbNo
    'statements for No
  Case vbCancel
    'statements for Cancel
End Case
```

图5-11A　Select语句。Select语句用多个条件值测试响应变量。如果响应和某种情况匹配，则执行相应的代码

7 循环

迭代或循环是另一个常见的程序特征。虽然可以尽可能多地使用SQL语句（UPDATE、INSERT等），但有时候需要遍历一张表或一个查询，分别检查每一行。

一些基本的循环格式如图5-12A所示。一般，如果已知循环次数，可以使用For/Next循环。Do循环是一种更加常见的循环。循环的重要特征是能够在循环的开始或结束处测试条件值。考虑一个例子：如果满足（x<=10），那么执行相应的语句。如果初始值x=15会怎样呢？如果在循环的开始处测试条件，那么循环语句不会执行。相反，如果在循环结束处测试条件，那么循环语句仅在测试条件之前执行一次。

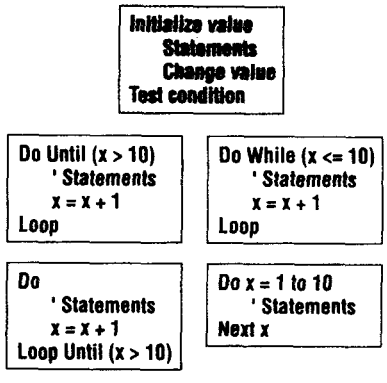


图5-12A　循环。所有的循环遵循共同的格式：初始化计数器值，执行循环体，增加计数器值，测试退出条件。可以在循环开始或结束处测试条件

和条件类似，对循环进行缩排是良好的编程风格。缩排使得别人更容易读懂代码、理解逻辑。如果循环没有问题，可以快速找到循环结束处。

使用循环时需要注意：如果发生错误，计算机可能一直执行循环语句（对于大部分个人计算机，Ctrl+Break可以终止正在执行的循环）。一种常见的错误是忘记更新条件变量（在本例中是x）。在遍历数据查询时，可能忘记跳转到下一行数据，在这种情况下，程序就会反复在同一行数据上进行相同的操作。写循环语句应遵循如下四步编程习惯：（1）写出初始条件；（2）写出结束语句；（3）写出更新条件变量的语句；（4）写出内部代码。前三部分给出框架。首先写出并测试它们，可以保证使用正确的数据。

8 子例程

编程的一个重要概念是把程序分为若干更小子例程和函数。子例程是一段其他程序可以调用的代码。当子例程结束时，控制权返回到调用子例程的代码处。使用子例程的目的是把程序分成较小的片段，更容易理解、测试和修改。

子例程本质上是独立的程序，它可以在程序中多处使用。例如，可以创建一个在屏幕上显示状态消息的子例程。如图5-13A所示，基本的程序只写一次，在需要显示状态消息的时候调用这个子例程。通过把消息传递给子例程，实际的消息可以每次改变。使用子例程的优点是只需要写一次子例程代码。还有，由于子例程指定了位置、风格和颜色，状态消息可以标准化。如果需要改变，只需在子例程中修改少数几行。如果没有子例程，就必须找到并修改每一处显示消息的代码。

```
Main program
...
StatusMessage "Trying to connect."
...
StatusMessage "Verifying access."
...
End main program

Sub StatusMessage (Msg As String)
        ' Display Msg, location, color
End Sub
```

图5-13A 子例程。Status Message子例程可在任何位置调用，当子例程执行完成后，返回调用程序

传递给函数或子例程的数据变量叫做参数。传递参数有两种基本方案：传引用和传值。在Microsoft Access中，默认的方案是传引用。在这种情况下，子例程中的变量和原程序中的是同一个，任何子例程对数据的修改会自动地返回给调用程序。例如，考虑如图5-14 A的两个例子，在子例程中修改变量j2的值，会自动地传回调用程序。但是，如果只传值，那么在子例程中复制数据变量。对子例程中数据的修改不会返回给调用函数。除非确认要修改调用函数中的初始值，最好使用传值方式。使用传引用的子例程经常导致难以发现程序的错误。其他的程序员不会意识到子例程改变了参数值。

```
Main
j = 3
DoSum j
...   ' j is now equal to 8

Subroutine DoSum (j2 As Integer)
      j2 = 8
End Sub
```

By Reference
Changes to data in the subroutine are passed back.

```
Main
j = 3
DoSum j
...   ' j is still equal to 3

Subroutine DoSum (ByVal j2 As Integer)
      j2 = 8
End Sub
```

By Value
Creates a copy of the variable, so changes are not returned.

图5-14A 两种向子例程传递数据的方法。尽可能多地通过传值的方法传递参数，以避免数据的意外修改

大部分语言允许创建新函数。函数和子例程在技术上略有不同，虽然子例程和函数都可以

通过传递引用参数接受或返回数据，但是，函数可以直接返回结果或单个数值至调用程序。例如，主程序可能有这样的语句：*v*1 = Min(*x*, *y*)。函数可以选择两个值中较小的值，并把它返回给主程序，在主程序中，这个值赋给变量*v*1。

9 小结

学习编程的惟一方法是自己写程序。看书、查阅语法文档，阅读别人的代码会有所帮助，但是只有通过练习才能成为程序员。

写程序时，一定要牢记自己（或者别人）以后可能会修改代码。选择具有描述性的变量名称。对于程序中技巧性比较强的部分，以及每部分代码的目的提纲一定要写清楚注释。每个部分和子例程规模要小。测试每个部分，并在文档中记录测试数据和结果。做修改标记，这样就清楚每个部分何时何故修改。

第三部分

应　用

在数据库环境中构造商业应用程序最先遇到的就是表单和报表。大部分数据库管理系统带有构造基本表单和报表的工具。然而，第6章介绍如何设计、修改表单和报表，使得这些工具更为有用，并且对用户友好。第6章还讨论把表单和报表连接成应用程序中的细节问题。

第7章主要关注数据库完整性和事务。探究如何保证数据质量和避免在多个用户同时使用一个大型数据库时所产生的问题。

第8章介绍如何处理事务处理数据库和用于分析的系统之间的冲突，把数据转换为数据仓库的问题，以及大型数据库中的现代交互式数据分析技术。

第 6 章

表单、报表和应用

本章学习内容

- 输入表单的目的是什么？为什么不直接把数据输入到表中？
- 表单上可用控件的主要类型是什么？
- 报表的主要成分是什么？
- 报表的整体结构是什么？
- 什么是创建应用必须的其他特征？

6.1 开发漫谈

Ariel： 你仔细看什么呢？

Miranda： 哦，我才知道怎么回答那些有难度的问题。但是我还是有些担心。很多次，我得到了错误的答案。我用SQL必须十分小心。

Arial： 我相信你能做好。你总是认真检查工作。

Miranda： 我希望越来越好。

Arial： 这是关键。现在你已经开始构造应用程序了？

Miranda： 是的。我看了一些关于表单和报表的信息，比较简单。

Arial： 真的？

Miranda： 当然。你知道最大的好处吗？所有的表单和报表都是以SQL为基础的。要获得所需数据，所有的工作就是构造查询。甚至有向导帮助创建基本的表单和报表。

Arial： 我一直相信总有一天，人们会再次唤起热情。

6.2 简介

表单和报表是数据库应用的一个重要部分。设计者使用它们建立集成的应用，为用户提供工作便利。决策者和职员的日常工作以表单和报表为基础。数年前，表单作为主要的输入方式，报表用于显示结果。但是，由于管理者从联机数据库获益更大，表单愈发重要。报表仍旧用于发布或纸质存储的输出。但是，表单可以以电子形式发布，可以有多种形式的输出。互联网，特别是万维网变得越来越流行，这意味着数据以电子表单的形式发布。数据库表单的设计原则也适用于网络。

图6-1小结如下，表单用于收集数据，显示查询结果，显示分析和执行运算。它们还用作导航或者连接到其他表单和报表。常见的如基于Windows的应用程序，表单用作直接操作对象。图形化界面允许用户拖放对象以示更改。有了这样的表单，用户可以直观地和表单交互。

典型的报表是打印在纸上的，但是越来越多的报表用作直接在屏幕上显示。报表用作格式化数据和显示复杂分析的结果。报表可能很详细，占据多页。一个例子可能是详细存货报表，另一方面，报表还可显示总结性数据，结合图表和汇总。常见的商业例子是周销售报表与过去几周详细销售状况的比较。报表通常是图形化的，占用一页。

- 收集数据
- 显示查询结果
- 显示分析和运算
- 导航其他表单和报表
- 直接操作对象
- 图形
- 拖放

图6-1 数据库表单的基本用途。理解表单的应用很重要，因为数据收集和分析表单的设计不同

应用程序集中了相关的表单和报表。因为所有的用户都通过表单和报表访问数据库，数据库的设计必须符合用户的工作。其他特征如菜单和帮助使用户使用应用程序更为简单。

6.3 报表和表单的有效设计

设计表单和报表时，最重要的是要强调它们主要用于和用户交互。每个表单和报表必须适合特殊情况和商业应用。例如，有的表单可以盲打——录入员在输入过程中不需要看屏幕。其他表单列出了探索性分析，决策者需要检查多种情况。这两种表单的特征、格式和功能完全不同。如果选择了对用户来说是错误的设计，表单（或报表）将完全没用。

有效设计的关键是确定用户的需求。问题在于用户通常不知道自己需要什么。特别地，他们可能并不关心现代DBMS的能力和限制。作为设计者，要和用户交谈，了解他们想要什么。然后根据经验提供表单更为有用的特征。小心找到为用户提供帮助和试图卖给用户不需要的东西的准确界线。

人性化设计研究者已经找到有助于表单设计的几点建议。首先，应用程序（甚至一个企业）中的所有表单和报表应该尽量一致。按键、命令和图标在整个应用中应该用于同一目的。颜色、布局和结构应该协调，这样用户可以理解任何表单和报表中的数据和内容。一组常见任务的基本应用减少了用户学习新应用的时间。基于网络应用的重要性正在增长，在某种意义上简化了设计，因为只有大部分用户理解的有限组工具。人性化设计研究给出设计者在构造表单和报表时应该遵守的要求。

6.3.1 人性化设计

图6-2总结了一些系统设计者在应用程序中应该注意的人性化设计因素。现在的操作系统中最重要的因素是用户——不是程序员，也不是应用程序——应该始终掌握控制权。例如，不要期待（或强迫）用户按照固定的顺序输入数据。相反，建立基本的表单，并且允许用户选择输入的顺序，这样对于用户更简单。用这种方式，用户的选择触发多种事件。应用程序响应这些事件或触发运算、检索和存储数据，提供新的选择。

还有，在任何可能的时候，为用户提供定制的选择。许多用户希望改变显示特征，例如颜

色、字体和大小。类似地，用户可以自己排列结果和包含数据。

人的因素	例　子
用户控制	配合用户任务 响应用户控制和事件 用户定制
一致性	布局、设计和颜色 行为
清晰性	组织 目的 术语
美学	加强艺术 图形 声音
反馈	方法 视觉 文本 声音 使用 接受输入 修改数据 完成工作 事件/激活
宽恕	错误的预报和校正 删除和修改的确认 备份和恢复

图6-2　基本的人性化设计因素。所有的设计都应该遵守这些基本特征

布局（设计和颜色）和期望的行为在应用程序中应该保持一致。在用户行为方面，小心保证基本特征的一致性，例如在输入结束时是否需要按回车键，哪个功能键触发帮助系统，方向键如何使用，每个图标的含义。这些行为在整个应用程序中应该一致。这个原则看上去是显然的，但是实现很有挑战性——尤其当许多设计者和程序员同时建立应用的时候。有两点经验可以保证实现一致：(1) 开始的时候，建立设计标准和基本的模版供所有的设计者使用，(2) 在应用程序接近尾声的时候不断返回检查一致性。

始终为清晰而努力。在许多情况下，清晰意味着保持应用程序简单和良好的组织。如果应用程序有多个表单和报表，根据用户的工作组织它们。应用程序有一个明确的目的，帮助确保设计加强这种目的。使用准确的术语，避免含混的语言，使用企业内部的词汇。如果一个公司把他的员工叫做“会员”，那么使用这个词代替“员工”。

美学在用户界面中扮演重要的角色。目标是使用颜色和设计（有时包括声音）加强表单和报表。避免初学者的错误，即每个表单使用不同的颜色，或者在一页上设置10种不同的字体。虽然设计和艺术带有很大的主观性，但是差的设计相对其他非常明显。如果在设计中没有美学经验，可以考虑选修一两门艺术或设计课程。如果没有机会，学习别人的工作，从中获得灵感，训练自己的艺术灵感，跟随当前的潮流。记住绘图和艺术非常重要。它们为用户提供

有吸引力的和熟悉的环境。

反馈对于大部分人机交互非常关键。人们希望知道当他们按键，选择选项或选择图标的时候，计算机识别他们的行为并且响应。典型的反馈使用包括接受输入、确认修改数据、突出完成工作，或者表示事件的开始或结束。提供反馈的时候可以使用几个选项。视觉上，光标可以改变形状，文本可以高亮显示，按钮可以“按下”或者盒子可以改变颜色。更直接的反馈形式例如在屏幕上显示消息，可以在更复杂的情况下使用。一些系统利用声音反馈，当用户选择一项工作或当计算机完成一项操作时，播放一段音乐或声音。如果决定使用声音反馈，必须给用户提供一个选择——有些人不喜欢“吵闹的”计算机。另一方面，不要草率地放弃使用声音反馈——对于视力差的人这种方式特别有效。类似地，当用户视觉集中在另外的工作上不能看计算机屏幕时，声音响应非常有用。

人类偶尔会犯错误或改变想法。作为设计者，需要明白这些可能性并在应用程序中给他们提供机会。特别地，应用程序应该能够预见和修改错误，删除和主要的更新需要确认——给用户机会确认修改，甚至取消操作。最后，整个应用程序应该包括数据的备份和恢复的机制——以防自然灾难和偶然的删除造成数据丢失。

6.3.2 Windows控件

在Windows下设计应用程序需要对Windows界面的深入了解：一部分因为需要提供标准控件和Windows支持的操作，另外一部分是因为Windows允许在应用程序中轻松提供附加的、强大的功能。

基本的窗口如图6-3所示。窗口由框架、标题栏、控制菜单框和多个标准按钮组成。用户可以调整框架的大小或设为固定尺寸。确定给每个表单提供了简明标题。控制菜单框提供标准的命令，包括移动、调整大小和关闭表单。常见的窗口按钮包括最大化（全屏）和最小化（推到屏幕下方），还有关闭。滚动条允许用户水平和垂直滚动表单。应用程序或DBMS通常自动控制这些行为。自己创建的表单可以删除这些标准控件，但是不要这么做。更重要的是，可以重载这些标准动作提供附加的功能。例如，当关闭表单的时候，可以做一些清理工作。常见的清理工作包括重新计算总数，保存修改的数据，触发其他表单中匹配的改变，或者当一张表的内容变化时同步另一张表的显示。

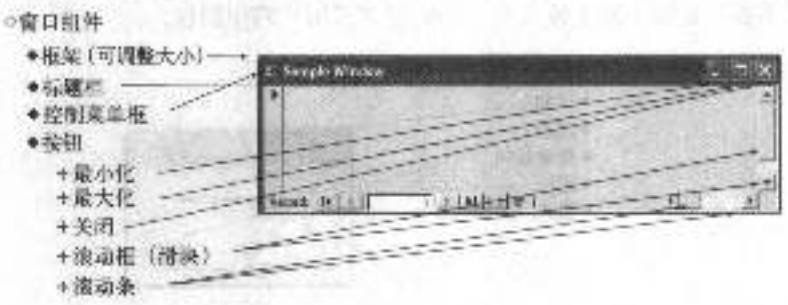


图6-3 Windows界面。窗口包括几个常见的组件。DBMS负责窗口需要的主要操作。但是，可以经常向这些操作添加功能或命令

菜单是应用程序的重要特征。大部分用户（公正地说）期望使用菜单访问每项工作的主要

功能。最常用的菜单是下拉菜单，在屏幕顶端显示，如图6-4。菜单包括一系列动作，通常按照相关命令分组，由分隔线分开。鼠标点击可以激活菜单，或者按下键盘上的加速字母做出选择。加速字母一般是菜单上加下划线的字母（例如，File命令的F），在标准PC上，这些命令以Alt键开始。大部分应用程序还定义了快捷键——尤其对于常见的命令。这些命令一般和Ctrl键联合使用，例如Ctrl+C命令用于复制选中的部分。

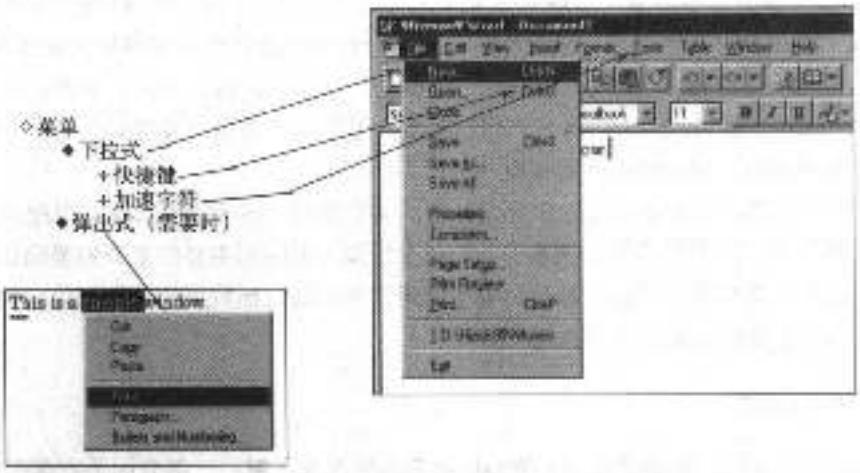


图6-4　窗口菜单。标准的菜单显示在屏幕的顶端，弹出式菜单与上下文相关，随选中内容的不同而不同

偶尔开发人员使用弹出式菜单或快捷键。使用鼠标右键触发弹出式菜单在屏幕上设置各种对象的属性的技术越来越流行。弹出式菜单和标准的固定菜单的最大不同在于，弹出式菜单通常是上下文相关的。上下文相关菜单根据用户选中内容的不同而不同。

很多表单和应用程序的一个有用特征是弹出消息框。如图6-5所示，消息框是一个简单的窗口，几乎没有控件。它用于显示短的、单行消息，用于从用户那里获得简单的反馈（通常为：是/否）。一般来说，最好避免使用消息框，因为它们剥夺了用户的控制权。在极端的情况下，模式框在屏幕上具有优先权，强迫用户处理之后才能继续工作。这种类型的消息框常用于处理错误或建立打印机——这时需要中断。设置模式属性为“是”就可以得到模式框，但是，需要再次重复下面的建议：避免模式框，它们剥夺了用户的控制权。

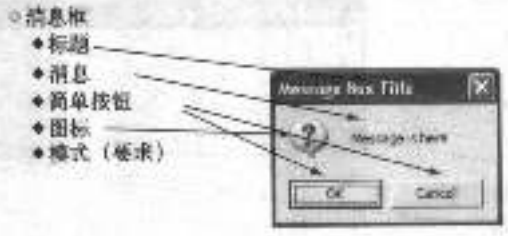


图6-5　消息框。消息框是一个简单的窗口，几乎没有按钮，尽量避免使用它，因为它打断了用户的正常工作，通常用于警告用户发生问题或提供同时选择

6.3.3 用户界面——网络要点

在很多方面，网络环境和标准的Windows环境十分类似。可以控制布局、颜色和数据输入。网页和表单的重要特征是标准化，这样在几乎所有的客户计算机或浏览器中以相同的方式显示。用户理解一些基本的界面特征：点击超链接转向新页面，点击按钮提交表单，下拉菜单用于选择项，等等。

开发网络应用时，最好使用样式表。样式表对于设计一致的网络表单是一种有效的工具。它允许定义标准的布局和颜色用于所有的网页。修改样式表的几个参数可以定义所有网页的修改，创建网站的新样式。

6.3.4 用户界面——访问问题

Windows界面最强大的功能在于其对图形的组织，人们仅需要移动鼠标或在屏幕上做出选择就可以完成复杂的操作。这种界面的缺点在于，系统做到良好的用户可访问比较困难，面临一些实际的挑战。作为设计者，应使应用程序适宜广大用户。首先，应用程序应该可以接收多种输入。不仅依赖鼠标，还要使用键盘，还有越来越多的用户语音。类似地，多种类型的输出对于应用程序也十分有用。考虑集成声音输出会越来越重要。用户必须能够设置输出的颜色和规模。

微软的指导提供了一些制作适宜更多用户的应用程序的建议。详细信息和最新的发展可以在网站上找到。根据其他应用程序总结的经验，提供一些特殊的人性化的建议。例如，不要使用红-绿色组合。大约10%的美国男性在辨识红绿中遇到困难。选择对大部分用户可以区分的对比度高的颜色（例如黑色和白色，黄色、蓝色和红色）。如果有疑问，请人检验所选颜色组合，或者让用户自己选择喜欢的颜色。

第二，避免要求用户做出快速的响应。不要对输入做出时间限制。虽然在游戏中非常有趣，但是很多用户输入速度很慢。一些设计者在输入数据一段时间后使用弹出消息检查用户进度。这类消息通常没有意义且令人厌烦。在现代屏幕保护安全系统中，用户可以设置自己的延迟控件和消息。

第三，避免使屏幕快速闪动的控件。它们会使大部分用户恼怒。最糟糕的是，某种频率的闪动可以在部分人中诱发癫痫发作。在1997年末，日本就发生了一件有趣的事情，当电视播放一系列闪动的图形时，大约700名儿童被送往医院。

第四，尽可能允许用户自定义屏幕。允许他们选择字体、字号和屏幕颜色。这样，用户可以调整屏幕，补偿任何可能具有的视觉缺陷。如果使用声音，允许用户控制音量，如果可能，甚至音调。在许多情况下，Windows环境提供大部分此类功能，关键在于避免重载这些功能。还有，应该在多个计算机上测试应用程序。有的视频系统可能扭曲颜色选择，或者不支持表单显示所要求的分辨率。

6.4 表单布局

独立的表单或者窗口是与应用程序用户交流的主要方式。表单用于收集数据，显示结果，组织整个系统。从数据库的角度看，应用程序从几种标准类型的表单开始构造。从这些基本

布局并开工作，记住可以创建复杂的使用多个不同表单布局的表单。但是，首先应该理解布局和每种独立表单类型的使用。

常用的有四种表单：(1) 表格表单，数据按行列显示；(2) 单行表单，每次显示一行，其中，设计者可以设置屏幕上数值的形式；(3) 子表单，显示来自两张具有一对多关系的表的数据；(4) 导航表单，或菜单，把用户指向应用程序的其他表单或报表。

如图6-6所示，表单具有一些共同点。可以设置它们的属性控制表单的显示风格。另外，表单包含控件，其中包括用于显示基本文本和数据的标签和文本框。表单还可以带有多个事件。例如，打开和关闭对于每个表单都是基本事件。可以通过创建事件响应行为控制表单的动作。

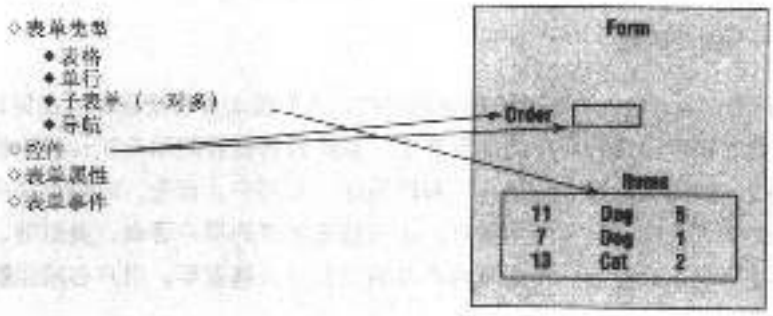


图6-6　表单布局。表单上放置的控件用于显示或收集数据。表单的样式由其属性设置定义。表单的行为由响应表单事件的创建行为控制

虽然表单可以有多个控件（例如文本框），但在某个时刻只能有一个控件具有焦点。具有焦点的控件接受用户的键盘输入。这个控件的边框通常高亮显示或颜色不同。同样的概念适用于屏幕上同一应用程序的多个表单。同一时刻只有一个表单具有焦点。在一个表单中，用户通常可以使用Tab键按照一定的顺序从一个控件移动到下一个，这就是Tab顺序。一定要检查所有表单的Tab顺序，保证其与表单的布局一致。

6.4.1　表格表单

如图6-7所示，最简单的表单是表格表单，可以显示表或查询的行和列。它可以用做子表单，但是很少作为单独表单使用。表格表单和数据表有所不同。表格表单提供更多的控件显示方案——例如三维控件和独立的颜色设置。数据表视图作为子表单比表格表单占用空间更小，更像电子表格。

表格表单的主要特征是显示多行编辑数据。这样，用户可以查看和比较多个数据项。它适用于数据较少的情况。如果要编辑的表太大——行列数太多，可能会出现问题。如果用户需要反复滚动表单才能找到某个特定的数据，表格表单的使用会发生困难。

表格表单常用于子表单，收集并显示与主表单相关的有限数据。例如，订货表单通常含有和OrderItem表相关的子表单，显示当前显示的订单中的订货列表。

6.4.2　单行或单列表单

单行表单每次显示一行数据，其目的是显示每一列。最大的特征是设计者可以在表单的任

何位置显示数据。把表单设计得看起来像传统的纸质表格非常有用。为了方便使用，设计者还可以使用颜色、图形和命令按钮。如图6-8所示，此类表单设计需要导航控件，允许用户前后跳转到其他行。常见的导航控件还包括跳转到第一行、最后一行和某个特定行的按钮。

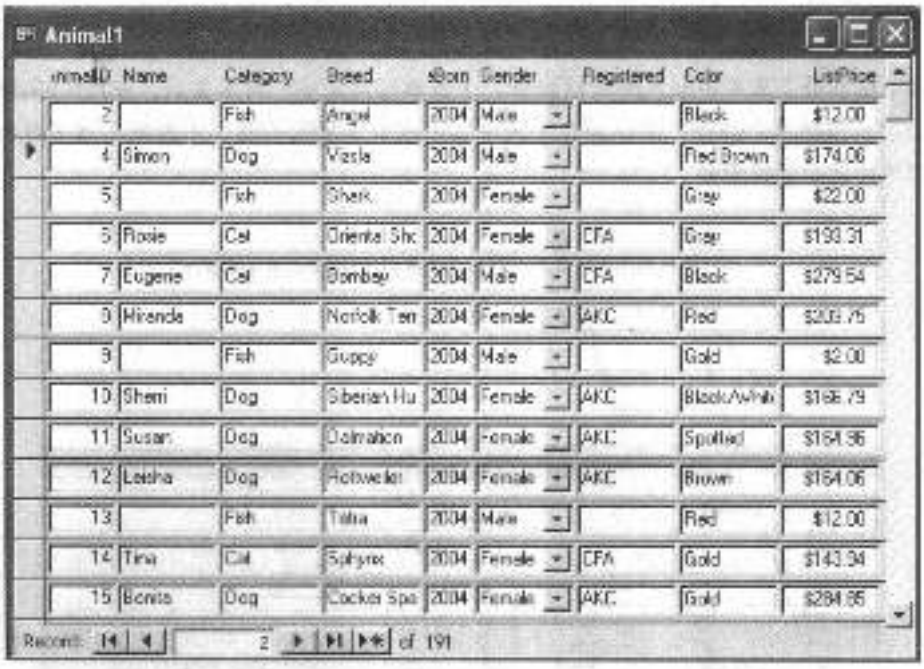


图6-7 表格表单示例。定义一行的控件，DBMS显示查询中所有行的数据。借助滚动条，用户可以看见更多的行（或者数据列）



图6-8 一个简单的单行表单。这个表单每次显示一行数据。通过控件可以设置样式、颜色、图形和命令按钮。底部的导航按钮允许用户显示不同的行

一般而言，需要包括一个查看命令允许用户定位到某个特定的数据行——基于某行的数据。例如，显示顾客数据的表单应该可以按照顾客的名字搜索。类似地，用户经常需要按照不同的顺序对行排序。

单行表单大概是最常用的表单布局。通过精心的设计，它可以显示大量的数据。通过包含子表单，可以突出不同数据的关系，并且方便用户输入数据。还可以包括图表帮助用户进行决策。

6.4.3 子表单

子表单通常是主表单中嵌入的表格表单。子表单经常显示一对多关系。在图6-9的例子中，一条销售记录包括多个动物，所以需要子表单显示这个重复的列表。主表单必须是单行表单，并且子表单应该是表格表单。

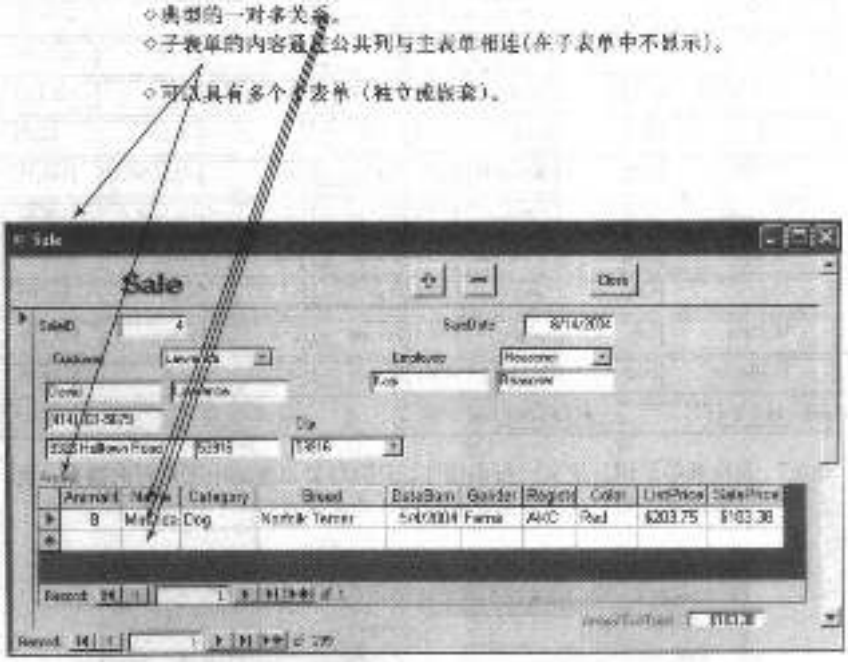


图6-9　子表单示例。主表单基于Sale表，和子表单中使用的SaleAnimal表具有一对多关系，数据表视图用于子表单一次显示多行

如果观察图6-9的表，可以看出SaleID把主表单（基于Sale表）和子表单（基于SaleAnimal表）连接起来。很少在子表单中显示连接列。一般来说，这样做没有意义，因为连接列和主表单中相关的列始终显示相同的值。考虑对于记录这意味着什么。当新的销售记录创建时，Sale表的SaleID由DBMS生成。SaleAnimal表也有SaleID列，每个卖出的动物必须包含相同的来自主表单的SaleID值。记录员对每个卖出动物在子表单中重复输入SaleID值是很痛苦的。通过使用子表单和指定SaleID作为连接（Master和Child属性），DBMS自动向子表单的表中输入主表单的SaleID。

大部分数据库系统支持创建具有多个子表单的表单。子表单可以是独立的——作为主表单的独立框，也可以是嵌套的——每个子表单存在于另一个之中。大部分情况，期望父表单是单行表单。但是，一些系统支持表格表单作为主表单和子表单。大部分应用程序中，用户对此不知所措：他们必须先从父表单种选择一行，再在子表单中匹配。

6.4.4 导航表单

导航表单或菜单表单提供应用程序的全局结构。虽然需要图形化设计的帮助，但是创建它们简单易行。导航表单通常包括图片，设计反映了公司的风格。

从空白表单开始，移去任何滚动条和导航控件。图片可以作为背景插入，或者作为独立控件用作打开另一个表单的按钮。如图6-10所示，命令按钮或连接几乎是导航表单最重要的特征。当用户选择一个按钮，相关表单或报表就打开了。主导航表单使用频繁，应该精心设计。

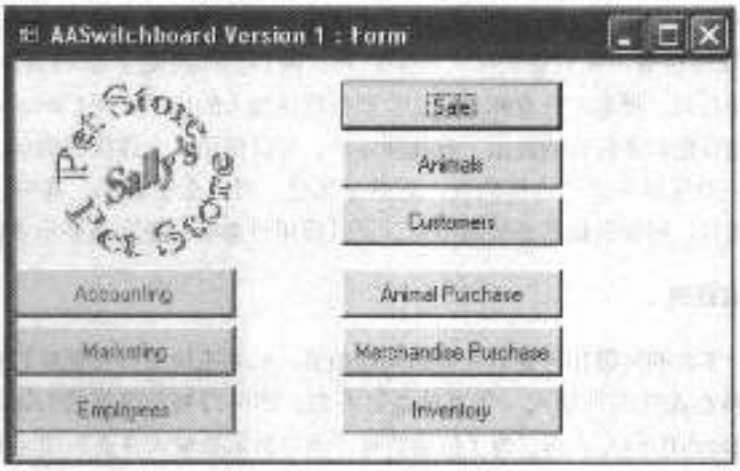


图6-10 导航表单示例。按钮匹配用户的工作

成功的应用程序的关键从导航表单开始，不仅是设计，还包括内容。表单应该满足用户的工作需要。一种做法是首先识别用户，然后提供一组适合他（或她）的工作的选项按钮。考虑一个简单的例子。管理者需要打印每天最畅销的商品的销售日报。每周DBMS必须打印雇员的销售总额清单。公司每月还向最佳顾客发信，提供额外的折扣。秘书负责打印这些报表，所以建立一个简单的菜单列出这些报表。秘书从中选出想要的报表，有的报表可能提出问题，例如使用哪周的数据。秘书输入答案，报表就能打印出来。

创建应用程序的第一步是考虑谁使用。他们如何工作？数据库输入和报表怎样配合工作？目标是设计一个菜单系统反映他们的工作方式。图6-11是两份首菜单的示例。哪个菜单更适合办事员理解？和工作最相关的那个。一旦理解了基本工作，写下相关菜单组。一些菜单选项调用其他的，一些打印报表，另一些激活已建的输入界面。



图6-11 设计用户菜单。秘书更容易理解哪个菜单？设计应用程序时，应该使应用程序配合用户的工作流程

6.5 建立表单

建立表单的第一步是一定要理解表单的目的以及如何使用它。它的用途规定了数据如何显示。一旦知道了需要的数据，就可以从数据库中找到存储相关数据的表。要记住重要的一点：表单应该一次只更新一张表的数据。例如，常见的销售记录表单可能显示这些数据：Sale、Customer、部分SaleItem，或许还有关于Products的详细数据。通过设计过程，这些数据需要存储在四张相关的表中，那么，如何创建表单，只更新一张表呢？

这个问题的答案实际上决定了很多数据库系统的特征。只能把一张表放在一个表单中的原因是多表给表单理解用户究竟想做什么带来困难。例如，如何使主Sales表单包含Sales表和Customer表的所有列，增加新行意味着什么？新行应该加入Sales表还是Customer表？

很多时候需要显示多张表的数据。有几种办法。可以使用多个连接的表单。可能在表单中建立分区，每一分区保存新的连接的表。这些分区之一可以是子表单，其中包含连接到主表单的重复的数据行。根据数据库系统的不同，可以使用可更新的查询来显示多表数据。

6.5.1 可更新的查询

一张表单中多表的问题和可更新查询的问题相关。如果系统支持可更新的多表查询，那么把从多张表中精心选择的列放入一张表单是可能的。图6-12展示常见的Sales Order主表单。Sale表保存CustomerID作为外码。为了记录销售，销售员需要输入合适的顾客ID号。但是记住这些ID号并不容易，在主表单中显示匹配的顾客数据来检验名字和地址非常好。

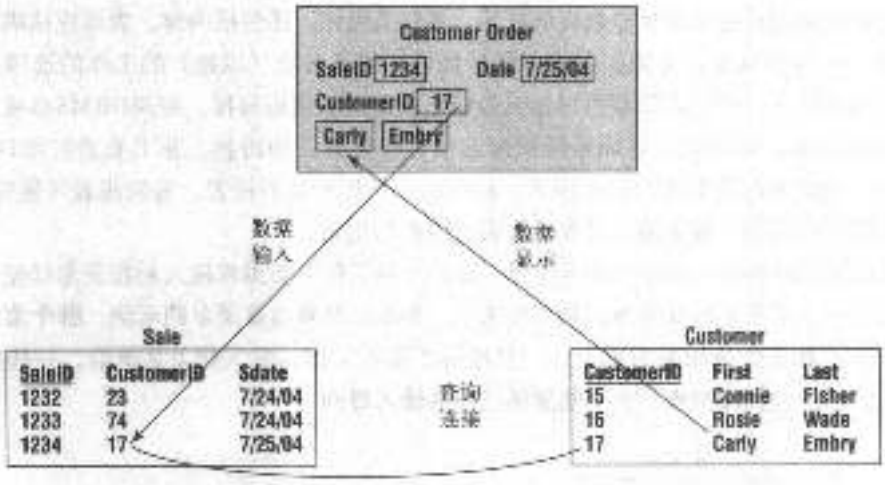


图6-12 Order表单建立在查询之上。查询包括Sale表的所有列和Customer表的部分列。查询不要包括Customer表的CustomerID列，它是用于连接两张表的列

技术上，查询应该包括Sale Order表的所有列以及Customer表的部分描述性列。为了保持可更新性，查询不能包括Customer表的主码（CustomerID）。这样做的一个潜在的缺点是无论何时新的CustomerID插入销售记录，表单必须查询数据库寻找匹配的顾客数据。结果是，一些表单系统不支持这种操作。

6.5.2 连接表单

解决问题的另一个方法是使用连接表单。这些表单可分别显示，或者可能分成几部分显示在一张表单中。后一种方法用于父/子表单，其中子表单包括来自另一张带有连接到父表单的一对多关系的表的数据。结果是，在一张表单中显示相关的表相对在独立窗口中是等价的。主要的区别在于屏幕空间的需求不同。如果把几个分区放在一张表单中，用户会需要大屏幕来查看所有的数据。通过使用多个窗口，用户无法同时看到所有的数据，但是可以在窗口之间切换，察看和编辑数据。

连接表单通过查询把第二张表中显示的数据和原表单的码值匹配。例如，在主Sale Order表单中，CustomerID可以用于显示连接表单中匹配的顾客数据。类似地，SaleItem子表单可以显示和主表单中SaleID匹配的行。每个相关部分的数据可以在分开的表单或区域中显示。理想情况下，表单系统有办法自动地连接并检索匹配的数据；否则，必须定制代码来改变潜在的查询并根据需要刷新数据。

6.5.3 属性和控件

大部分现代软件包使用面向对象（OO）技术。在面向对象设计中，每个对象都有描述它的属性和它能够执行的方法或函数。对象还和事件驱动系统紧密相关，在这样的系统中，用户的行为和修改可以触发各种事件。大部分数据库表单使用这种方法。但是，一般使用的对象已经由DBMS定义。读者只需对属性赋值，编写简短的程序响应各种事件，使应用程序方便用户即可。

图6-13的属性列表突出显示表单和控件的属性可以分为几类。第一类（数据）和数据源相关，在这里可以设置基本表和查询。可以设置过滤器只显示满足某个条件的数据行。还可以直接指定排列顺序，以及基本查询的WHERE子句，这常常是更有效的方法。这一步实际上把控件元素和数据库绑定在一起。

第二组属性指向数据完整性，帮助控制编辑类型和表单允许的修改。例如，可以为每个用户设置属性，这样部分用户使用特殊的表单不可以增加或删除数据。但是，一定要记住在SQL中设置这些条件更为安全，这样应用在数据库而不只是表单。例如，销售记录员不能增加新的供应商。

类　别	属　性
数据	基本表/查询 过滤器 排序
完整性	编辑 增加、删除 锁
格式	标题 滚动条 记录选择器 导航按钮 大小和居中 背景/图片 颜色 顺序
其他	弹出菜单 菜单栏 帮助

图6-13 表单基本属性。至少设置数据源和基本格式属性。附加的属性保证一致性、保护数据和表单方便使用

第三类属性控制表单的显示。从标题到滚动条，表单大小和背景，设置一切显示属性。再强调一遍，记住一致性非常重要。在开始项目之前，选择设计模板和标准；然后所有的表单和控件都要满足标准。

6.5.4　表单上的控件

表单由控件构成。控件包括表单上的所有对象。常见的控件包括简单的文本标签，数据输入的文本框，选项按钮，图片和列表框。更复杂的控件可以从商业软件开发商那里买到。图6-14显示大部分系统常用的控件。标签和文本框使用频繁。

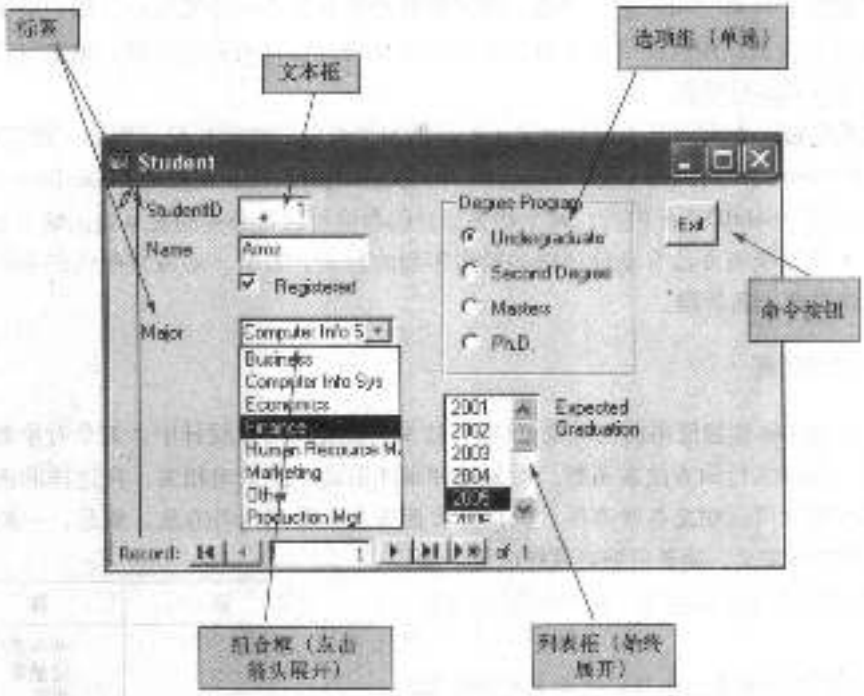


图6-14　示例控件。注意组合框和列表框的区别。组合框占用一行，当用户点击箭头时展开，用户可以选择某一项

所有控件的一个共同属性是名字。每个控件必须有名字。注意，名字一旦设置，很难修改。其他控件和程序代码使用该名字（像变量名一样）在表单上检索和存储数据。创建控件时最好选择有意义的名字。如果以后试图修改，必须找到所有引用这个控件的控件和程序——这是一件耗时易错的事。有的人根据控件的类型命名，例如，标签控件以“lbl”开头，如“lblAddress”。这种命名方式对于以后阅读代码的程序员大有帮助。

标签控件和文本框

最基本的控件是标签、文本框和命令按钮。标签是一段简单的文本，用户不能直接修改。文本框用于显示数据库的数据和输入新值。在表中检索并存储数据的控件叫做绑定控件。因为任何数据的改变都会自动地保存在表中。数据可以输入非绑定控件，但是值不会存储在数据库中，并且在用户浏览表的各个数据行之间也不会改变。标签和文本框的常见属性包括颜色、字体和大小。绑定和非绑定框的区别在于绑定框有控件源（表的列）。在非绑定框中，这个属性是空的。

命令按钮

典型的命令按钮只有一个功能：当用户点击时，触发事件（On_Click事件），执行某个动

作。这个动作有几种选择，包括打开另一个表单，打印报表，或者运行定制程序。命令按钮是导航表单的主要组成部分，用于打开相关表单。

复选框和单选按钮

这些控件允许用户从一组选项中做出选择。例如，办事员从雄性、雌性和未知中指出动物的性别。用户有几种方法做出选择。从技术上，在这三种方法中没有区别。但是，对于常见的用户界面标准，使用时略有区别。

本质上有两种选择方法：互斥和多选。考虑动物性别的例子。三个选项是互斥的。一个动物或者是雄性，或者是雌性，或者不清楚性别情况（中性是独立备选，并不影响一般的性别决定）。单选按钮组（圆型）用于互斥选项。用户知道，当看见一组单选按钮，只选择一个选项。另一方面，顾客可能订购具有这些特征的狗：注册的、褐色或黑色、甚至脾气暴烈的和短毛的。这种情况应该使用带有复选框（方型）的表单，因为可以选中几个选项。选择一个选项不影响其他选项的选择。

图形功能

有时候希望在表单中增加图形。有的控件负责图形，就像简单的线和框一样。线和框常常通过增加阴影或光照的方式给其他控件建立三维效果。

为了像图6-15那样显示表中的图像，必须先在该表中定义对象或图像列。然后绑定对象框和文本框一样在表单上定位和显示图像。

为了使用图像或纹理作为背景，首先使用图形包保证图像足够明亮不会影响其他框读取。然后把表单的图形属性设置为图像文件的名称。在大部分情况下，希望把图像嵌入表单，所以图片直接包含在数据库中。



图6-15 绑定在数据列上的图像。雇员的照片扫描并存储在数据库列中

组合框和列表框

列表框和组合框在大部分数据库应用程序中是有用的控件。这两个控件的目的是显示列表项，允许用户从中选择一个值。组合框和列表框主要有两点区别。在用户点击向下箭头之前，组合框只显示一个选择项。第二，用户可以直接向组合框输入值（组合框的名字表明它是列表框和文本框的联合）。大部分设计者青睐组合框，因为它占用的空间比列表框小。

组合框有两个基本用途。第一是向表中输入值。第二是搜索某一特定数据行。如果决定在应用程序中使用两种组合框，应该用颜色或其他属性区分用途。

如果观察规范化的表，会发现为什么组合框在关系数据库应用如此广泛。许多表通过内部生成的ID连接起来。例如，Sale表用CustomerID识别顾客。在图6-12中，办事员需要输入值17指明购物的顾客。但是办事员如何知道这个值呢？不可能期望员工记住成千上万的顾客代码。类似地，也不能要求顾客记住他们的ID号。在某些情况下（例如音像出租店）可以要求顾客带着包含该信息的ID卡。读者可能想到使用某个共同的ID号，像税号或电话号码。但是很难保证这些号码的惟一性，并且难以说服顾客给出这些号码。

一种比较好的解决方案是让数据库使用内部生成的ID，然后办事员从名字列表中选出该顾客。因此，大部分表单开发系统支持组合框（有时叫做选择框）。组合框显示Customer表中所有的顾客条目（按名字或电话号码排序）。输入顾客名字的前几个字母，组合框就会滚动找出匹配的条目。当办事员选择合适的顾客，相应的ID就输入了Sale表。

复杂的控件

其他的控件对象可以使用多种计算机语言创建或者从商业公司购买。

几种其他控件如图6-16所示。标签和日历控件在商业应用程序中尤为有用。网格控件可以以电子表格的形式显示数据。这些类型的控件不像标准绑定控件那样容易使用，开发者需要写简短的程序向控件下载数据和响应控件事件。

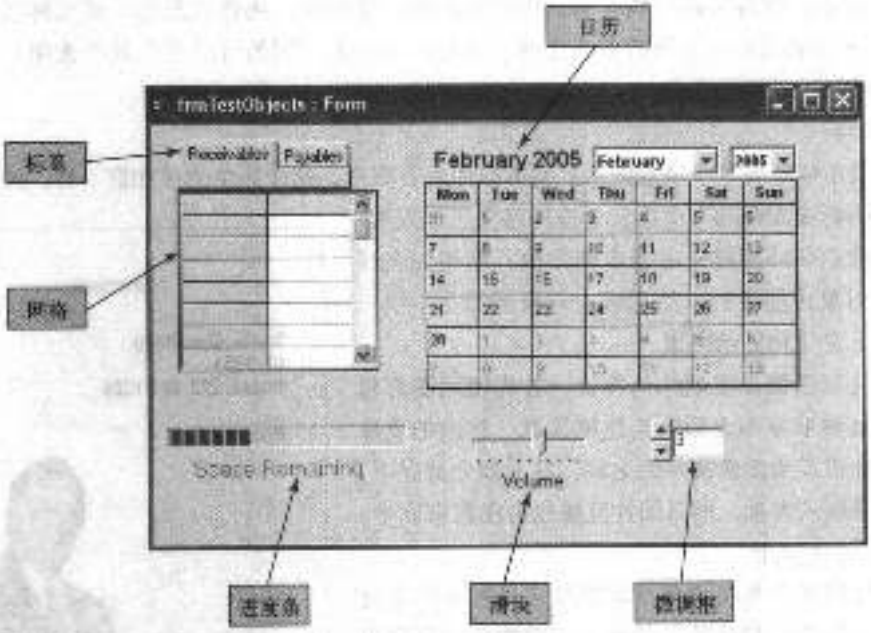


图6-16　附加控件。有成千上万的控件可以改善用户界面或提供专业服务。常见的控件包括标签、网格、日历、进度条、滑块和微调框。读者也可以自己创建定制控件

图表

用于决策的数据库应用程序经常包括图表或图形。图表是另一类可以放在表单（或报表）上的控件。建立有用图表的第一步是和用户讨论需要什么类型的数据和什么类型的图表。然后通常建立新的SQL查询，收集要在图表中显示的数据。图表控件把图表放在表单上，并指定其属性（如图表类型、箭头刻度和颜色）。

在数据库表单和报表中，常用两种类型的图表：（1）显示当前显示行的细节图；（2）显示所有（或部分）行汇总数据的图。这两类图表的区别在于显示数据的级别。细节图随着每行显示数据的改变而改变。汇总图通常从总数或平均值得到。

图6-17列出两种类型的图。它们都可能在宠物商店中用到。每幅图表对比了花在动物和商品上的费用。但是，上面一组图分开显示了每个独立销售，所以图形随着每行Sale查询结果的

不同而不同。下面的图显示了商店的所有销售——虽然也放在展示每行数据的Sale表单上，但它不会改变（除非超时）。为了创建两种类型的图表，主要的区别在于查询。细节图表的查询包含一列（SaleID）连接到表单中显示的数据行（基于SaledID）。汇总图表计算所有销售的总额，没有连接到某个特定的销售。

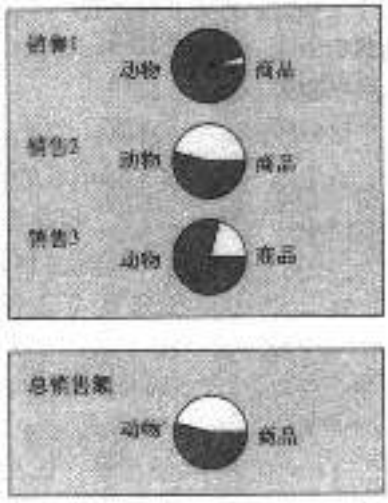


图6-17 表单或报表中的图表。上面的图表显示了每项销售的对比，它随着每行数据改变。下面的图显示总销售的对比，不和某行绑定

6.5.5 多表单

读者可能想到，一个应用程序很快就充满大量的表单。当然，表单应该彼此相连，用户可以通过点击按钮、数值或图片快速地在表单间跳转。导航表单在把表单集成在一块起了重要作用。但是，读者还可以直接把表单连起来。最常见的例子是把子表单放在主表单内。在这种情况下，表单通过设置子表单的Master和Child属性连接起来。然后数据库系统保持数据的同步，当用户选择主表单中新的一行时，子表单自动定位和显示匹配的行。

当表单包括相关数据时，有时候需要在新表单中打开并显示相关的数据行。例如，如果Order表单包括顾客数据，当用户点击Edit按钮（或者双击顾客姓名），应用程序应该打开Customer表单。如图6-18所示，Customer表单应该显示和Order表单中顾客相应的数据。打开表单命令使用连接条件参数完成这一工作。条件限制在新表单中显示的数据。在这个例子中，连接条件指定Customer表单的CustomerID必须和Sales表单的CustomerID一致。

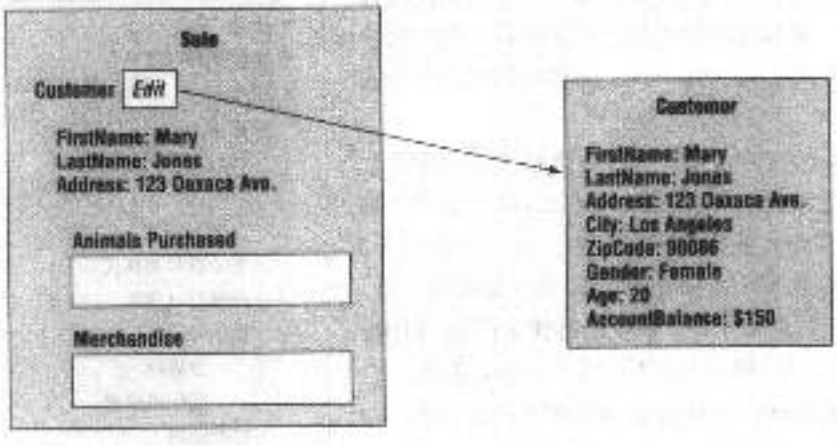


图6-18 连接带有匹配数据的表单。打开Customer表单显示和已经输入到Sale表单的顾客相对应的数据

大部分情况下，用户会关闭Customer表单返回主Sale表单。但是，如果用户保持两个表单同时打开该怎么办？他们希望两个表单的数据同步，这样，如果在Sale表单有新数据显示，

Customer表单相匹配的数据也可以显示。需要写几行代码保证这种同步。从本质上说，当Sale表单改变时，代码获得新的CustomerID，把它传递给Customer表单，再查询数据库，然后用匹配的数据刷新表单。

图6-19显示相关情况。当顾客查看动物数据时，决定买下它。按下Animal表单的Purchase按钮可以迅速转到Sale表单。用按钮把AnimalID复制到Sale表单的合适位置是很方便的。

同上，这里也需要写几行代码才能把AnimalID插入到Sale表单的合适控件中。在某些情况下，可能需要Sale表单把ID值传递回Animal表。

商业应用程序常常需要计算子表单的部分和。如图6-19所示，有的系统把子表单完全当作独立的表单，所以必须先计算子表单的部分和，然后把值复制回主表单。

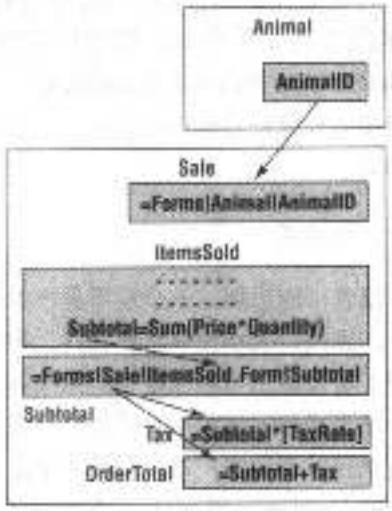


图6-19 从另一个表单中复制数据。默认的AnimalID从Animal表复制到Sale表单。同样的，子表单首先计算部分和，然后复制到主表单

6.5.6 国际属性

设计表单和报表时，需要越来越多地考虑应用程序的国际化设置。一个明显的因素是需要把数据翻译成不同的语言。有的应用程序开发系统通过让开发者在一个位置存储所有的注释（例如窗口名称、标签和错误信息）的办法来解决语言问题。可以利用DBMS创建各种语言的表，保存需要翻译的词汇，达到同样的效果。然后构造表单时根据某个代码值查询合适的词汇。这个独立的表由翻译者修改——不需要修改表单或代码。这样做的缺点是没有编译器，每个表单下载时间更长，因为每个词汇需要分别从表中查询得到。

如图6-20所示，另一个和语言相关的问题是大部分语言使用特殊的字符集。例如，拉丁语言经常包括重音符和其他区分标记。东方语言更加复杂，它们需要成千上万的不同字符图，而不是少量字母。计算机必须能够显示每个国家的特殊字符集。目前，这一问题流行的解决方案是使用Unicode系统，它是一个国际标准，存储和显示每种语言的字符。有的操作系统支持Unicode，例如Windows 95、Windows NT、NetWare和部分UNIX支持2字节字符。一些现代应用程序开发工具支持Unicode。如果编写需要国际支持的应用程序，一定选择能够提供支持的工具。语言和字符的问题还影响数据排序。因



图6-20 国际属性。当创建在国际上使用的表单和报表时，认真考虑字符和数据的不同格式。还有，遵守所有国家的法律——尤其是数据的隐私权

此，数据库允许根据当前字符集设置排列顺序。

日期、时间、电话号码、邮政编码和货币等数据格式也随着国家和地区的不同而不同。大部分可以在Windows环境（控制面板）中设置。但是，把数据从一个国家转换到另一个国家时一定要小心。日期和时间的转换会顺利完成，因为它们以与基础日期和时间的差的形式存储。但是，货币转换会出问题。假设在美国的数据库应用程序中有一列内容是货币，其值以美元记录（例如18.20美元）。如果把数据库复制到使用英镑作为货币标准的伦敦的机器上会发生什么呢？结果是货币符号会改变，而数值不变。在这个例子中，用户会看到18.20英镑，这当然不正确（应该是11.44英镑）。有两种办法：（1）不管货币数据类型，仍然使用美元计价；（2）编写查询把所有的货币数据转换为新货币。通过定义自己的货币类型，第一种情况有可能使用的货币格式，避免货币符号的转换。第二种方法对于很多应用程序可能是最好的，在查询中把原始值乘以转换因子非常方便。当然，转换因子随时间而改变，所以必须注意跟踪转换日期。

电话号码和邮政编码的情况更为复杂。当定义数据和创建表单时，一定要预留足够的空间保存国家代码和长电话号码。还要记住有的国家的邮政编码不仅包括数字，还有字母。程序员习惯性地认为通过把邮政编码限制为5位数字有利于减少输入数据的错误。在国际设置中，不能做这种假设。一种减少数据输入错误的方法是使用组合框，其中包括一张邮政编码的（巨大的）表。更可靠的方法是购买能够自动根据地址检查和更新数据库中邮政编码的软件。

最后，注意遵守所有国家的关于安全和数据访问法律。例如，很多欧洲国家关于隐私权的法律很严格——尤其关于顾客数据和国家ID号码。在有些情况下，把欧洲收集的数据转移到美国是非法的。

6.6 直接操作图形对象

在过去的几年内，应用程序的用户界面发生了变化。图形的大量使用导致对对象的直接操作的重视。用户可以在屏幕上把一个项目从一个地方拖到另一个地方以示更改，而不是输入命令。大部分人已经看到这种方式应用于基本的操作系统命令。例如，在DOS时代，复制文件必须输入下面的命令：COPY MYFILE.DOC A:MYFILE.DOC。现在，点击该文件图标，然后拖到一个磁盘驱动器的图标上就可以了。

6.6.1 Sally的宠物商店示例

图形方式使应用程序更容易使用。但是，它需要读者改变对应用程序的思维方式，和强烈的创造性思维。考虑宠物商店的例子。本章开始时设计的表单容易创建，并且满足应用需求。但是，需要改变整个应用程序的方法。

图6-21是宠物商店例子的部分示意图。比较这张表单和图6-9的传统数据输入表单。传统的方法要求用户向框内输入文本。借助图形，用户可以看见各种动物的照片，然后把它们拖到顾客，表示一笔交易。双击一个项目会提供更多图片或附加说明。类似的方法用于特别订单，使用拖放技术定义动物（种类、颜色等），然后提交订单。

用户不可能完全避免数据输入。有的地方必须收集用户的基本数据（姓名、地址、电话号码等）。当办事员双击用户的图标或照片时，激活传统表单，然后输入数据。也就是说，图6-21

的表单代替了传统的销售表单而不是代替基本顾客表单。当然，一旦顾客数据存储在文件中，就可以在任何顾客访问的时候拖到主表单。



图6-21　直接操作宠物商店的图形对象。无需向框中输入AnimalID，只需把动物的图片拖到顾客，就可以形成交易。双击某个项目显示更多细节或相关图片

6.6.2　因特网

对于图形和对象的直接操作的重视对于因特网上表单的应用极为有用。首先，大部分用户几乎没有操作数据库的经验，对公司也知之甚少。创建公司和它的流程的图形模型有两个重要目的：(1) 由于和用户已知的实际购买方式相匹配，方便使用网站；(2) 把用户的行为限制在事先定义好的范围内。

图形化界面的一个目的是隐藏数据库的使用。是的，所有的基本产品信息、数字和销售数据存储在数据库中，数据库系统具有搜索和保存用户选择的功能。它还可以为管理者提供报表和数据分析。但是，用户并不需要了解数据库本身，用户只需看到存储和产品的图象，他们操作对象以了解更多情况，或者下订单。使用因特网的一个困难在于受到用户浏览器功能的限制。浏览器可以处理大部分简单数据项控件，但是对于拖放操作的支持欠佳。虽然如此，图形搜索方式有助于提高表单的直观性和可用性。

6.6.3　图形方式的复杂性和局限性

基于图形对象直接操作表单可能有潜在的缺点。最重要的是对输入数据无能为力。例如，无法期望码头工人使用拖放表单的办法记录数百个箱子的接收。条码扫描仪可能更有效。类似地，质量控制技术员更喜欢简单的键盘（或语音）系统，这样他（或者她）可以一边工作一边输入数据。

甚至宠物商店销售表单使用拖放方式也存在争议。考虑典型的大型宠物商店的操作。考虑几十个顾客推着装满商品的购物车来到收银台，会发生什么。如果收银员必须使用拖放的方

式，收款的过程就会极为漫长。在这里，条码扫描仪又一次可以加速工作进程。另一方面，如果取消收银员，商店的工作可以改善。考虑如果购物者使用商店的拖放网站来选择商品，取货，等待装好袋子再带走，商店该如何运作？获取数据方式的不同取决于业务操作的不同，以及谁来使用系统。

图形操作方式的另一个缺点是每个应用程序需要数量可观的定制编程工作。传统的方法更为直接。向表单输入数据的常用工具包括文本框、组合框和子表单。这些工具可以用于任何应用程序。另一方面，对象的直接操作要求屏幕上有独立的业务对象，并且和数据相连。每一个用户操作（双击和拖放）必须为应用程序专门定制。以后，可能会有工具辅助编程。但是现在，图形方式比其他方式要求更多的编程工作。

构造因特网上图形化数据库应用程序面临类似的困难。有两个主要限制：传输速度和有限的软件工具。除此以外，巨额的资金和繁重的工作是网络的间接需要。一些行业的很多公司正在寻找解决方案。

6.7 报表

理解了表单，那么报表就比较简单了。相比之下，表单和报表的主要区别在于报表适合打印，而表单适合屏幕显示。还有两点不同：（1）表单用于收集数据；（2）报表一般用于展示汇总数据。因此，报表没有数据收集控件。但是，如果可以打印表单，为什么还需要报表呢？报表的两个主要优势在于（1）方便地控制多页输出（带有连续的页眉和页码）；（2）可以合并细节和汇总数据。第5章举例说明了SQL查询如何通过GROUP BY子句产生相对复杂的结果。尽管如此，简单的SQL查询可以用于显示数据行的细节或者汇总数据——但不能同时显示。好的DBMS报表生成器还提供附加的输出控件，例如使用红色打印负值。

6.7.1 报表设计

如图6-22所总结的，设计报表的过程有几个问题需要注意。在表单的开发过程中，读者和用户需要决定内容和布局。还需要知道典型的报表大小（页数和份数），还有打印的频率。因为涉及物理的操作，打印报表是一个耗时的过程。一份几十页的报表没有问题。但是，如果报表有几百页、上千份，必须仔细计划。首先，需要一个快速的、耐用的打印机。还需要机器和人力来集中分发这些报表。打印长报表时通常需要安排好时间。

纸质报表面临着不同的安全挑战。纸质报表需要更多的传统安全控制，例如书面分发清单、份数和控制数据。如果安全对于企业是一个重要问题，那么设计报表时应该建立这些控制机制。

在设计报表时，涉及一些实质的和美学方面的问题。页面大小，字体和页面的整体设计需要确定。新的DBMS报表生成器相对灵活，这样有利有弊。好处是设计者可以有效地控制报表。坏处是设计者需要更多地理解设计——包括术语。

美学设计和完整的设计问题超出本书的范围。如果读者希望认真地设计（纸质报表、表单或者网页），可以考虑学习图形设计课程。尽管如此，如果能够学习一些基本内容，会有所帮助。

- 报表用途/用户需求
- 报表布局选择
  - 表格
  - 列/分组
  - 图表/曲线
- 页面大小
- 打印机限制
- 报表生成周期如何？
- 触发报表的事件
- 报表的大小
- 份数
- 颜色可用性
- 安全控制
  - 分发清单
  - 唯一编号
  - 隐藏/不打印数据
  - 安全打印机
  - 传输限制
  - 打印队列控制
- 输出事项
  - 字体
  - 可读性
  - 大小
  - 用户限制
  - OCR需求

图6-22　报表设计的基本原则。和用户一起确定报表的内容与布局。估计报表的大小与打印频率。确定安全控制。为了用户的可读性检查字体和大小

6.7.2　术语

很多基本的词汇来源于排版和图形设计。如图6-23的词汇帮助读者理解报表生成器以及产生优质报表。第一步选择页面设置，包括页面大小、打印方向（横向或纵向）和页边距。装订系统的类型会影响页边距，可能必须预留额外的装订距离来满足装订要求。

- ◇页面设置
  - ◆横向或纵向
  - ◆页边距
  - ◆装订线（装订距离）
- ◇字体
  - ◆衬线（Times New Roman）
  - ◆无衬线（Arial）
  - ◆装饰
  - ◆固定宽度
- ◇字号
  - ◆常用：10-12磅
  - ◆72磅约合1英寸
  - ◆四号字（1/6英寸）（12磅）

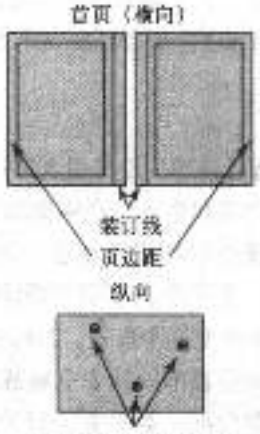


图6-23　基本的出版术语。理解基本的设计术语有助于设计优质报表以及与出版者和排版者交流

下一步是选择字体和字号。一般来说，有衬线的字体容易阅读，但是无衬线字体有更多的空白，在较大或较小字号的情况下更容易阅读。除了封面和标题，避免装饰字体。为保证栏对齐，分栏数通常为固定宽度值。特殊的固定宽度字体（例如Courier），其所有字母使用完全一样的宽度，特别适合于不使用制表符对齐的非数字栏。

字号大小常用磅来衡量，大部分常见的打印字号在10到12磅之间。排指规则有效地指出

了72磅的大写字母的高度大约为1英寸。一些报表系统使用四号字测量大小和距离。一个四号字相当于1/6英寸，或者12磅的高度。

如果报表中包括图形和图像，术语更加复杂。我们知道，位图的质量由原始图像分辨率和输出设备的分辨率来决定。常见的激光打印机的分辨率为每英寸600点（dpi）。打字机的典型分辨率为2 400dpi。在分辨率为600dpi的激光打印机上看起来很好的图像在分辨率为2 400dpi的打字机上可能太小或者太粗糙。

如果报表是彩色的，会遇到更多问题。特别地，屏幕上的颜色可能和打印机的不同。类似地，使用彩色喷墨打印的报表样本可能和提交给打字机的看上去完全不同。Pantone®颜色标准通过为许多标准颜色设置数字来缩小这类问题。在彩色打印中遇到的相关问题需要对所有报表创建颜色分离予以解决。对于提交给打印店的全彩色报表，每一页需要四个相互分离的片。三原色，即青色（蓝色），洋红色（红色）、黄色，以及关键的黑色，合起来记为CMYK。在这种情况下，为保证颜色能够准确地合成，每一页需要高分辨率的校准标记。

设计中最重要的因素之一是保持报表简单和优雅。例如，在一页中坚持使用一种字体和一两种字号。使用大量的空白以突出栏和特征。最重要的是，由于设计风格不断变化，从报纸和杂志的布局中获得新思想和模式。

6.7.3 基本报表类型

从数据格式的角度看，有三种主要的报表类型：表格、分组或部分和、标签。数据类型和报表的用途决定了选择哪一种。

表格报表和标签报表

表格布局如图6-24所示，是最简单的报表设计。它基本上等同于打印数据栏，类似于输出查询结果。和简单查询相比，表格报表的优势在于可以在每页上打印页眉和页码。还增加了关于字号和列宽的控制。表格报表经常用于列出详细信息，例如库存清单报表。注意排列顺序十分关键，因为这些报表将用于搜索指定项目。

Customer

CustomerID	Phone	FirstName	LastName	Address	ZipCode
1		Walkin	Walkin		
2	(606) 801-9830	Brent	Cummings	9197 Hatchet Dri	98815
3	(617) 843-8468	Dwight	Logan	1780 Clearview	02109
4	(502) 807-0907	Shelika	Gilbert	4407 Green Stre	40342
5	(701) 384-5623	Charlotte	Anderson	4333 Highland C	58102
6	(606) 740-3304	Searobe	Hopkins	3183 Highland C	40330
7	(408) 104-9807	Anita	Robinson	8177 Horse Park	95035
8	(606) 688-8141	Cora	Reid	8351 Locust Str	41073
9	(702) 533-3419	Elwood	Henson	4042 West Ridg	89125
10	(302) 701-7398	Kaye	Maynard	5095 Sugar Gro	19901

图6-24 表格报表布局。表格报表功能有限，但是适用于列举细节数据。它们用于列出数据明细单

如图6-25所示，标签也很简单。标签报表的本质在于所有关于一行数据的输出打印在一页

的一“列”中。下一行打印在下一列。**标签标单**的名称来源于标签的预印或预制页。这些报表有时根据实际列的编号命名。图6-25的例子中，一页有三个标签，所以它是三栏报表。在出现报表生成器之前，打印标签报表富有挑战性，因为打印机只能在页的顶端工作。因此，必须写程序代码控制打印三个不同行的数据，然后返回并打印第二行，如此反复。现在打印机更加灵活，报表生成器简化了工作。记住标签报表还有助于完成其他工作——无论何时，需要按行把数据排列到同一页的不同位置。例如，通过插入空白行和改变标签大小，可以创建tic-tac-toe数据模式。它可以对封面或广告单产生有趣的作用，但是不要对数百页数据使用这种模式。

Dwight Parrish 9804 Plum Springs Road Worcester, MA 01613	Dwight Logan 1760 Clearview Street Boston, MA 02105	David Sims 6623 Glenview Drive Boston, MA 02116
Hershel Keen 8124 Industrial Drive Nashua, NH 03060	Reva Kidd 5591 Hilltown Road Bangor, ME 04401	Dan Kennedy 3108 Troon Court Burlington, VT 05401
Sharon Seton 2551 Elementary Drive Barre, VT 05641	Kelly Moore 6116 Clearview Street Middlebury, VT 05753	Casey Tuck 7977 Fairways Drive Clifton, NJ 07015

图6-25　标签表单布局。这个三栏报表常用于打印标签页：它可以用于少量需要打印在一页上的数据

分组或部分和

最常见的报表类型是基于分组和计算部分和。它还提供最灵活的报表项目的格式。常见的例子包括打印收据或账单。很多情况下，报表需要打印多行数据，如图6-26所示的订单表单。每个月的订单打印在一份报表中，但是项目分组显示每个独立订单的部分和。很多人把这样的报表作为控制中断报表。

◇列
◇分组列

MerchandiseOrder

Order	*Order/Receive*	*Employee*	*ShippingCost*	*Supplier*
1	*06-Mar-04*	*4*	*$33.54*	*10*
	08-Mar-04	*Hopkins*		*Rhodes*
		Alan		*Brad*

item	*quantity*	*cost*	*description*	*list price*	*category*	*extended*
27	8	$24.65	Aquarium Filter & Pump	$35.00	Fish	197.20
30	208	$4.42	Flea Collar-Dog-Medium	$7.00	Dog	919.36
				Total		1116.56

图6-26　分组或部分和报表。打印多个订单。每个订单有一个详细的订购项列表。报表可以计算每个订单的部分和，以及整个订单的总和

部分和报表的关键在于它包括细节项列表（订购项、数量、费用等），以及分组或总数

(订购日期、顾客和订购总数)。为了创建这样的报表，首先创建包括需要显示数据的查询。在这个例子中，可能包括Order、OrderItem、Merchandise、Customer、Employee和Supplier表。注意：如果想得到细节数据，不要在查询中使用GROUP BY语句。如果从这个巨大的查询中查找数据，将得到大量的行和列——带有很多重复数据。这样做虽然得到了细节，但正好是用户不希望看到的。报表的目的是清理显示的数据。

为了创建分组报表，请看图6-27所示的报表设计，这个格式页显示了数据的分组中断，并且规定了该页中每个元素的格式。这里的格式又一次由独立的控件确定。更改控件的属性可以修改该控件显示数据的外观。例如，可以设置基本的字体和字号属性。



图6-27 分组报表的布局。基本的报表元素（报表标题等）。还要注意页面布局由数据控件的位置和属性确定。在Merchandise Order的同一份报表中，只定义了一个分组（根据Order号）

报表的基本元素是页眉、页脚、分组中断和详细内容。报表标题包含仅在报表开头显示的数据，例如封面页。类似地，报表尾注（report footer）显示报表最末的数据，例如，汇总统计数据或图表。页眉和页脚在每页上显示，报表标题和尾注页除外。页眉和页脚可以显示列标题、页码、公司标识或安全标识。

定义这种类型报表的报表特征是分组，图6-27的例子定义了一个分组：Merchandise-Order.PO Number。报表会为查询中的每个PONumber中断或创建新的数据组。每个Order包含很多订购项。报表设计规定这些行按照ItemID号排序（在每张订单内部）。每个分组有分组标题和分组尾注。分组标题显示整个订单的相关数据（例如，日期、顾客、店员和供应商）。它还包括详细内容（重复）部分的列标签。分组尾注显示每个分组的部分和。

每个报表元素的常见用法如图6-28所示。所有的元素（除了详细内容）可以成对出现——页眉和页脚。并不一定要一起使用。例如，可以选择在页眉中显示页码，同时删除页脚以获得更多页面空间。

分组代表一对多关系。例如，每个订单的详细部分有很多项。如果数据中有多个一对多（或多对多）关系，可能需要使用多级分组。如图6-29所示，每个分组嵌套在另一个分组内部，详细部分在最内层。

报表部分	用　途
报表标题	整个报表只显示一次的标题页
页眉	每页顶端显示的标题行或页说明
分组标题	分组（例如订单）数据和详细部分标题
详细部分	核心数据
分组尾注	分组部分和
页脚	在每页底部显示一页汇总或页码和注释
报表尾注	报表最末显示一次 总结性注释、总和、整个数据集合的图表

图6-28　报表布局元素的常见用法。大部分元素可以成对使用，但是可以去掉任何不需要的元素



图6-29　嵌套分组。例如，每个顾客可以下多个订单，并且每个订单可以包括多个内容行。使用两个分组：（1）显示每个顾客的订单总数；（2）显示每个订单的总价值

为了创建这份报表，必须建立包括每个显示项的查询。从详细内容开始，然后加入其他的表，直到包括所有需要的列。可以使用通过计算得到的列以减少计算量，例如Price * Quantity。应小心避免使用聚集函数（例如Sum），避免使用GROUP BY语句。惟一可能包括这两个功能的原因是“详细内容”行，实际上就是部分和（或平均值）本身。

分组报表常用于计算，尤其是部分和。一般来说，对一行数据的计算由查询负责。另一方面，聚集函数（Sum、Average等）由报表生成器控制。报表生成器有多种定义操作范围的方法，也就是，哪些数据应该包括在总数之中。

6.7.4　图表

报表中的图表和表单中的类似。第一步是决定对用户来说什么类型的图表最能够表现数

据。第二步是决定图表应该放置在报表的哪个元素中。如果对详细项绘制图表，那么图表应该在详细内容部分；在这里的每行数据都将重复绘制图表。如果图表是总结性的，它属于分组尾注，或者如果汇总整个报表的数据，则其可能位于报表尾注。

一旦已经决定图表的类型和位置，就可以构造查询收集数据了。这个查询和产生整个报表的查询不同，特别地，如果是分组尾注的图表，为了得到图表可能需要在查询中使用聚集函数。报表中一定要包括把图表和数据连接起来的列——即使该列并不在图表中显示。图6-30显示了宠物商店的销售报表中的一条销售记录。图表中的总数由独立的查询计算。

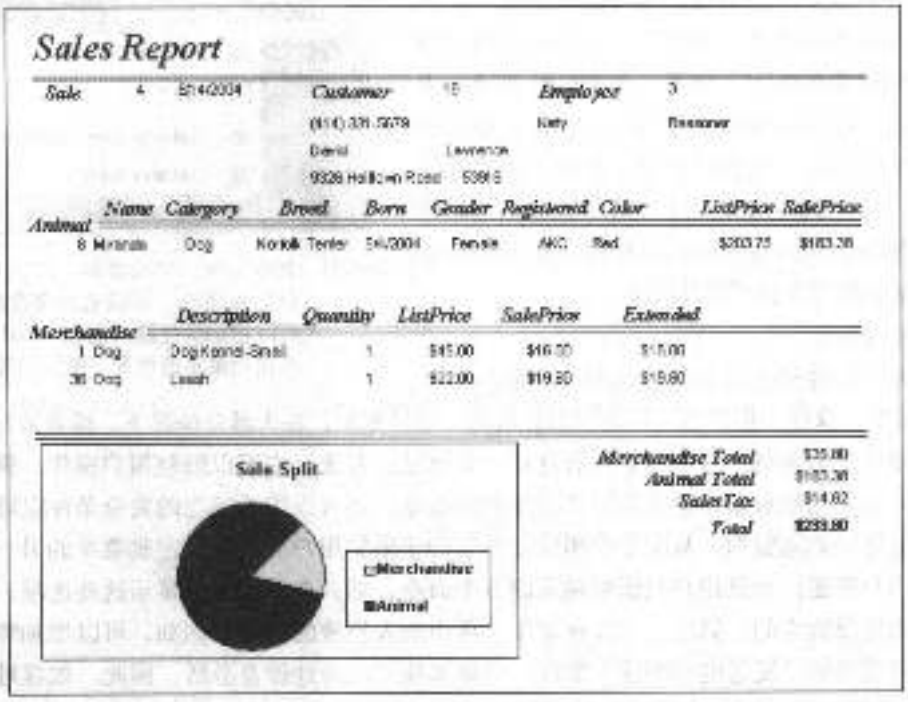


图6-30 销售报表的图表示例。它突出了动物和商品的销售量的比例。该图表和Sale表出现在同一个层次上，而不是在Animal或Merchandise的详细部分，也不是在报表尾注中

6.8 应用软件的功能

现代应用软件包括多个功能，具有标准化的外观，简化读者对应用软件的使用。最重要的三个功能是：菜单、工具栏和帮助系统。

菜单是显示在应用软件顶部的一组功能。主菜单在整个应用中保持不变。因此，菜单集中了任何时刻可以激活的功能。菜单对具有视觉障碍的人和那些喜欢使用键盘而不是指示设备（鼠标）的人有益，因为功能可以通过键盘激活。

工具栏包括一组执行常见任务的图标或按钮。有的应用软件允许用户使用特定的按钮定制工具栏；通常用户还可以重新设置工具栏。当前的开发工具利用这个功能允许用户把菜单放在定制工具栏中。

帮助系统对于任何应用软件都是关键部件。在大部分应用软件中，它代替了纸质手册。理论上，应用软件应该在没有手册的情况下也清晰易用。帮助系统还提供额外的文本或图片指导以及详细信息。

6.8.1　菜单和工具栏

菜单是一系列选项，当用户选中时做出某种动作。大部分菜单是有层次结构的，也就是说，细节选项位于几个关键词之下。Windows界面标准规定菜单显示在应用软件的顶部。但是，用户可以把菜单移动到另一个位置。大部分应用程序使用相似的菜单命令。例如，如图6-31所示，主菜单包括了三个典型的常见命令：文件、编辑和帮助。文件命令一般包括新建、打开、保存和关闭命令。读者的应用程序应尽量与这些标准保持一致。使用同样的布局和词语构造应用程序，用户只需最少的培训就可以理解它。



图6-31　菜单示例。层次结构。加下划线的字母是加速键，可以通过键盘激活。还可以增加快捷键（例如，Ctrl+D），在不使用菜单的情况下激活功能

菜单的功能

读者可能希望在自己的应用程序中使用基本DBMS菜单。这样，用户可以完全控制数据库，尽管如此，在大部分情况下，读者最好为自己的应用程序定制菜单。定制的菜单有这样一些好处。首先，它可以限制用户操作。例如，如果用户不需要删除数据，那么菜单不需要删除命令。还可以设置适当的安全条件以阻止用户通过其他途径删除数据。从菜单中删除命令有助于限制用户的操作。定制菜单的另一个好处是简化用户界面。如果用户只需要输入四五个命令，那么在菜单中仅显示这些选项。这样用户会更快地找到它们。第三，可以在定制菜单中加入特殊的功能。例如，可以增加特殊的帮助命令向支持部门发送电子邮件。第四，菜单选项可以通过键盘激活。因此，依靠触觉的打字员和有视觉障碍的用户可以不看屏幕操作应用软件。

工具栏

定制菜单通常在工具栏上实现。工具栏通常包括一组按钮和菜单项。当用户点击工具栏按钮时，执行预定义的操作。工具栏可以包括传统的按钮，还可以包括文本菜单。大部分工具栏是可停靠的，即用户可以把它们拖拽到应用软件窗口的任何位置。

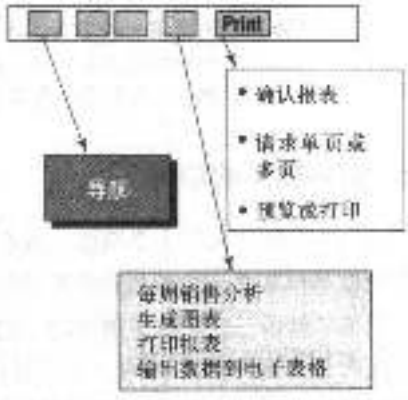


图6-32　工具栏示例。工具栏可以包括按钮和菜单。按钮通常显示图标。当指针移动到它们的上方，显示描述该按钮的简单提示信息。当点击按钮时，执行某个动作或显示菜单

设置工具栏的目的是通过单击操作访问复杂的功能或经常使用的命令。例如，很多工具栏有一个快速保存当前工作的图标。如图6-32所示，可以在工具栏上设置任何图标和命令，可以为每个窗口设置不同的工具栏和菜单，甚至可以有多

个工具栏。例如，一个工具栏包括访问整个应用软件的命令。特殊的工具栏可以在打开每个窗口时自动添加。

创建菜单和工具栏

为了支持标准化和简化菜单的创建，大部分应用程序开发环境有菜单生成功能。具体的步骤依使用的系统而不同；但是创建菜单有三个基本步骤：（1）选择布局或结构；（2）给出每个功能的名称和访问键；（3）定义选中每个选项时的操作。

创建工具栏的步骤基本类似：（1）确定每个窗口需要显示的工具；（2）选择或者创建图标代表相关命令；（3）定义选中时的动作。创建工具栏的一个关键步骤是选择功能的图标。不要假定用户可以识别图标或理解它的含义。大部分系统允许为每个功能定义提示信息。当用户把鼠标放在图标上，就显示提示信息或简要的注释。每个工具栏按钮都要有提示信息。

在当前的应用程序开发系统中，创建工具栏和菜单非常简单。可以对已经存在的工具栏，增加或删除某些功能。类似地，可以创建新的工具栏。按钮图标和菜单可以拖拽到工具栏上。

主要的步骤是设置每个项目的属性。菜单名称应该简短且具有描述性。还要遵循标准的商业软件命名规范。为了指定访问键，在字母前面加上&。例如，&File文本显示为File，Alt+F可以激活相应功能。快捷键（例如，Ctrl+D）可以通过适当设置详细菜单项或按钮命令来实现。

大部分系统允许创建多个工具栏，并且根据不同的用户或应用软件的不同部分激活或阻止工具栏。一般，需要创建多行代码激活或阻止某个工具栏。

6.8.2 定制帮助

在线帮助系统不断发展，已经代替了质纸手册。目的是提供用户有效使用应用软件系统可能需要的背景信息和特殊指导。帮助文件可能包括文本描述、图片和连接相关主题的超链接。帮助信息应该尽量上下文相关。用户可以得到当前特定工作所需的信息。帮助系统还必须具有广泛的搜索引擎，方便用户找到任何主题。图6-33是一个帮助系统的例子。

为了保持一致性，Windows帮助系统几乎自动显示文件、管理链接、表的内容，索引和搜索。对于开发者，可以集中精力创建包括基本信息和必要链接的文件。然后帮助系统编译器把这些数据转化为Windows帮助系统可以显示和搜索的特殊文件。一旦学习了创建页的基本知识，困难的部分在于书写完整的帮助系统所需的上百页的文件。大部分大的开发项目主管会雇用人员专职写帮助文件。

创建Windows的帮助文件

创建帮助文件的第一步也是最重要的一步是理解用户需要什么信息。然后必须写出个性化的文件解释系统的目的以及如何使用它。正如其他交互工程，首先必须理解用户。哪些类型的人会使用这个软件？他们的阅读水平如何？他们一般具有多少使用计算机的经验和训练？他们理解商务运作吗？目的是以用户可以迅速理解的形式提供简明的帮助信息。

理解用户的需求之后，应该书写个性化的帮助文件。创建帮助系统需要五个基本部分：（1）文本信息；（2）图像；（3）主题之间的超链接；（4）描述每页的关键词；（5）主题名称和页码。

从Windows98开始，微软修改了内置帮助系统。旧的系统仍然存在，但是对于新的系统使用超文本标记语言（HTML）创建帮助文件更加容易。有很多创建网页的优秀工具。但是，注

意这些工具：有的，比如Word，创建的复杂代码可能和帮助系统编译器不相容。使用HTML编辑器生成基本的HTML代码，不要使用XML或Javascript。

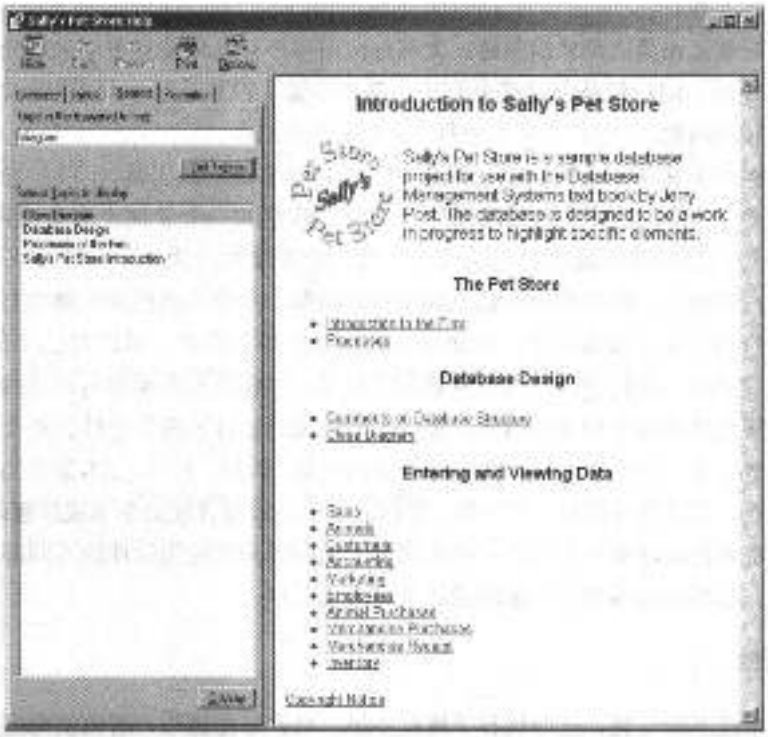


图6-33　帮助窗口示例。Windows帮助系统管理所有的显示和搜索。读者只需要写HTML主题页和指定关键词

从设计的角度讲，首先设计帮助系统的风格并且使用层叠样式表定义风格至关重要。样式表设置字体、字号、颜色和页边距。样式表的功能是在一个地方定义所有的布局选项。每个链接到样式表的页采用其定义的风格。所以，当修改整个帮助文件的布局时，只需对样式表作少量修改，所有的页都会使用新的风格。

每个主题创建为独立的HTML页。用户一次查看一页材料。尽量保持主题简短，可以在一屏内显示。每个帮助页包括指向其他主题的链接。图6-34展示了基本帮助主题的一部分。每页应该有标题（以<TITLE>标记）。页通常包含指向其他主题的链接（使用HTML标准标记<A HREF>）。图像可以是两种格式：JPEG和GIF之一。大部分帮助图像是黑白的，应采用GIF格式。大部分图形包可以以这些格式创建和保存文件。当保存文件时，文件名仅包括字母和数字——不要包括空格。因为最后会有数百页文件，保持每页具有简单的描述主题，并且在最后修改是一个好的办法。

关键词是每个帮助页的重要部分。它们用于为用户创建索引。索引按照字母顺序列出关键词，当用户双击某个单词时，显示相关的帮助页。HTML帮助系统工作组提供两种创建索引的方式：（1）手工列出每个词和相应的主题；（2）在主题页上列出每个关键词。第2种方法更容易，也不容易出错，但是它把关键词分散到整个文档中，很难把它们作为一组编辑。帮助系统工作组有工具可以把关键词放在一个主题页中（编辑/编译器信息/关键词），但是必须把

每页下载到工作组才能使用这种方法。相反，在图6-34中，复制<OBJECT>的所有内容和</OBJECT>标签更容易；然后修改每页列表中的关键词。每个关键词列在单独的<PARAM>标签之中。如果想得到多个层次，可以使用逗号列出层次。例如，三个条目：(1) 销售；(2) 销售，商品；(3) 销售，动物会创建销售的索引项，后面跟着动物和商品两个缩进行。

```
<OBJECT type="application/x-oleobject"
classId="clsid:1e2a7bd0-dab9-11d0-b93a-00c04fc99f9e">
  <PARAM name="Keyword" value="Contents">
  <PARAM name="Keyword" value="Introduction">
  <PARAM name="Keyword" value="Sally's Pet Store">
  <PARAM name="Keyword" value="Management">
</OBJECT>
<HTML><HEAD>
<TITLE>Sally's Pet Store Introduction</TITLE>
<LINK rel="stylesheet" type="text/css"
href="PetHelpStyle.css">
</HEAD><BODY>
<H1>Introduction to Sally's Pet Store</H1>
<TABLE><TR>
<TD><IMG SRC='PetStoreLogo2.gif' border='0'></TD>
<TD>Sally's Pet Store is a sample database project for use
with the Database Management Systems textbook by Jerry
Post. The database is designed to be a work in progress
to highlight specific elements.</TD>
</TR></TABLE>
<H2>The Pet Store</H2>
<UL>
<LI><A HREF='FirmIntroduction.html'>Introduction to the
Firm</A></LI>
<LI><A HREF=`FirmProcesses.html'>Processes</A></LI>
</UL>
</BODY></HTML>
```

图6-34 帮助页部分示例。用HTML把每个主题作为一个独立的网页。标记<A>指向其他页。标记<IMG>下载图像。使用样式表设置字体和设计。使用表格或样式控制布局。在标记<OBJECT>中写入当前页的关键词

上下文相关帮助

和应用软件的Sales表单相关的用户不希望在多个帮助页之间来回切换，也不想思考搜索词句。相反，当他们按下帮助键时，期望看到当前表单的信息。至少，需要为软件的每个表单创建不同的帮助页。但是，现在数据库应用软件有一些方法可以指定每个表单显示哪个帮助页。如图6-35所示，每个表单有一个帮助文件和帮助内容ID属性。Oracle和Visual Basic表单有类似的属性。在帮助文件属性中输入文件名（例如，PetStore.chm）。帮助内容ID需要一个数字。这个数字是长整型的，其范围从1到20亿以上。

注意到应用软件需要主题编号至关重要，但是帮助文件通过文件名而不是数字指向页。为了把这两个系统匹配起来，必须为每个主题页赋予惟一的编号。使用HTML帮助系统，可以创建单独的文本文件（通常叫做Topics.h）记录对应关系。图6-36是一个示例文件。虽然可以选择任何数字，但是分组赋值更容易记住。还有，由于有20亿的空间，可以在不同组数之间留下较大的间隔。例如，计数单位最好是百万千万，而不是个（1、2、3等等）。根据业务对象

赋值是有效的（例如，所有的顾客帮助文件编号从1 000 000到2 000 000）。创建文件之后，使用HtmlHelp API Information按钮（左侧，从上数第四个）告诉帮助工作组包括文件。现在，浏览应用程序的每一个表单，指定文件名和主题编号，避免在帮助文件中修改主题编号；它们难以在应用程序中找到。

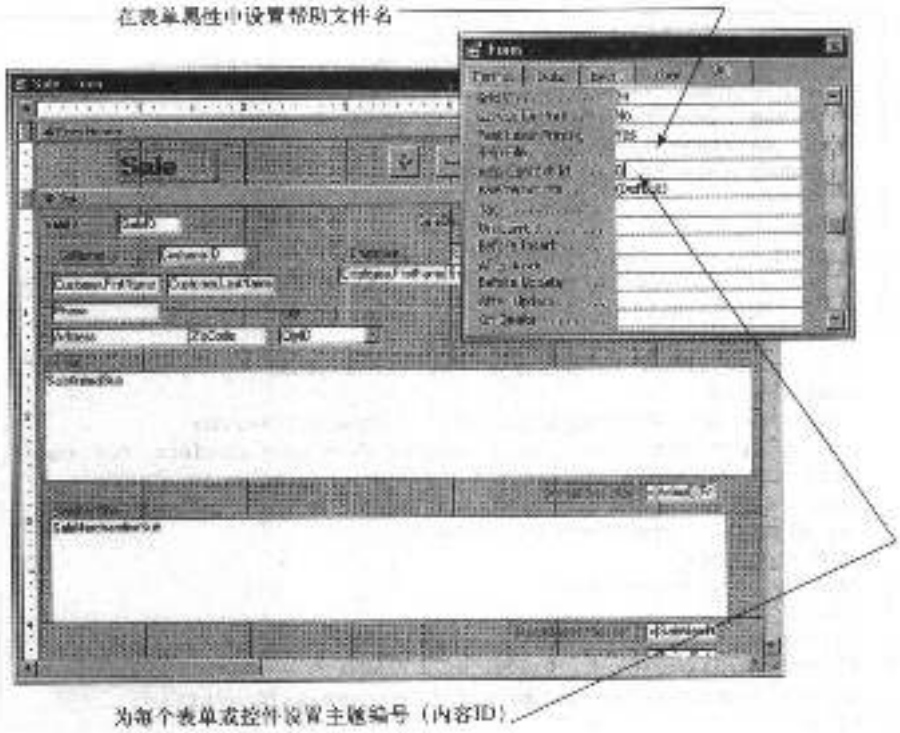


图6-35　设置上下文相关帮助。在每个表单中，输入帮助文件属性中的帮助文件名。然后在帮助内容ID中输入该表单的主题编号。每个控件或子表单还可以有另一个不同的帮助主题——只需输入相应的主题编号

```
#define  PetStoreIntro              100
#define  Accounting               10000
#define  Animal                   20000
#define  AnimalPurchase           30000
#define  ClassDiagram             40000
#define  Copyright                50000
#define  Customer                 60000
#define  DatabaseDesign           70000
#define  Employee                 80000
#define  FirmIntroduction         90000
#define  FirmProcesses           100000
#define  Inventory               110000
#define  Marketing               120000
#define  MerchandisePurchases    130000
#define  MerchandiseReceipt      140000
#define  Sale                    150000
```

图6-36　对应文件。应用程序通过数字指向主题，但是帮助系统使用文件名。对应文件（Topics.h）是一个简单的文本文件，它为每页赋予一个编号

小结

表单设计必须与用户的工作相匹配，这样的应用软件使用起来才容易。为了满足这一点，需要注意设计原则，操作系统指导以及人的缺陷。在任何可能的情况下，构造通过直接操作对象的方式来使用的表单，例如把项目从一个地方拖拽到另一个地方表示移动。

表单基于表或查询。每个表单有单一的目的，并且可以在一张表中保存数据。更复杂的表单可以通过把子表单放在主表单中创建。表单的控件用于向表输入数据，执行查询函数以及操作数据。一些标准的控件可以从Windows环境中获得（例如，文本框、组合框和单选按钮）。可以购买附加的控件完成更复杂的工作，例如用于时间安排的日历和三维图像。

报表常用于打印，它与表单不同，因为报表是为表现数据而设计的，而不是收集数据。有几种类型的表单，但是很多商业表单使用部分和或分组来显示不同层次的数据。例如，销售报表可能根据销售部门分组，或者根据销售员分组，或者同时使用两者。使用查询把所有报表需要的数据项连接起来。使用报表生成器有两点好处：（1）可以直接设置数据格式和排列；（2）报表可以包括详细列表、部分和、总和。

开发漫谈

Miranda的笔记中写道，数据库向导可以创建基本的表单。但是，在使用表单向导和生成数百张小表单之前，考虑用户的工作和整体设计，尝试把最重要的信息放在一个中心表单上，同时有几个次级表单辅助。努力设计一个整洁的、组织良好的界面，少使用颜色和图形增强外观。还要开发设计标准和布局以保持应用程序的一致性。一定要为未来留有空间。课程作业是创建基本的表单和报表。

关键词

可访问性
美学
绑定控件
复选框
清晰性
组合框
命令按钮
一致性
上下文相关菜单
控件
对象的直接操作
可停靠的
工具栏
拖放
反馈
焦点
分组中断
盲打
帮助系统
人性化设计
超文本标记语言（HTML）
列表框
菜单
模式框
单选按钮
页脚
页眉
报表尾注
报表标题
滚动条
单行表单
样式表
子表单
导航表单
Tab顺序
表格表单
工具提示
Unicode

复习题

1. 设计表单时哪些人性化因素应该重点考虑?
2. 如何使得应用程序为更多的用户所使用?
3. 表单的主要类型有哪些?
4. 表单中主要的控件有哪些?
5. 复选框和单选按钮的区别是什么?
6. 子表单的目的是什么?
7. 主要的报表类型有哪些?
8. 报表的主要部分有哪些?
9. 完成应用程序需要哪些功能?
10. 一般来说，菜单的目的是什么？它应该包括哪些项目?
11. 为什么上下文相关帮助如此重要?
12. 创建HTML帮助文件有哪几步?

练习

根据第2章和第3章的练习所描述的数据库创建表、初始表单和报表。

1. 游艇租赁，第2章练习1。
2. 宠物饲养，第2章练习2。
3. 牙医预约，第2章练习3。
4. 定制鞋，第2章练习4。
5. 电台播放清单，第2章练习8。
6. 收费系统，第2章练习9。
7. 网站注释，第3章练习1。
8. 网上销售，第3章练习2。
9. 人力资源福利，第3章练习3。
10. 车库乐队，第3章练习4。
11. 草坪护理，第3章练习5。
12. 宴会设施，第3章练习6。
13. 钟表制造商，第3章练习7。
14. 比萨外卖，第3章练习8。
15. 动画工作室，第3章练习9。
16. 设计师签名的牛仔裤，第3章练习10。
17. 对于已有的数据库，创建主导航表单。把它作为最终版本，包括所有必须的表单和报表的连接——即使还没有创建的。找到潜在的用户（或另一个学生）测试布局和结构。
18. 生成小的帮助文件。创建至少三页带链接的HTML文件，其中一页作为起始页。给每页设置关键词，用H1、H2和H3 标记建立内容表。定义主题编号。使用HTML帮助系统生成编译后的帮助文件并测试它。把它添加到每个表单指向独立页的数据库，再测试它。

19. 建立定制的工具栏和菜单，能够执行简单的任务，例如打开主菜单，显示主要顾客的表单。
20. 复习所用的DBMS文档，确认它提供的支持直接操作对象的功能，例如拖放。

Sally的宠物商店

21. 建立表单管理商品存货，包括表Breed、Category和Merchandise的表单。
22. 创建表单记录来自供应商的动物的订单。
23. 创建表单记录来自供应商的商品的订单和收据。
24. 创建表单按照主要类别显示商品销售的图表。
25. 确认数据库应用程序中所有需要包括的任务，重新设计主导航表单。
26. 创建表单允许管理者根据某种常见条件（例如，购买猫的项目，购物超过某个金额）选择用户，然后建立每个选中顾客的邮件标签报表。
27. 创建报表，根据商品和动物的销售列出雇员，显示图表把每个雇员的销售总数与全部的总数相比较。
28. 创建表单，允许管理者选择某个时间范围，然后绘制图表显示该时段从每个供应商那里购买的所有商品。
29. 创建报表，按类别和星期显示所有商品的销售。其中包括图表按照类别比较全部销售的情况。
30. 创建报表，按州显示总销售情况。

Rolling Thunder自行车

31. 绘图显示Rolling Thunder的表单如何关联。
32. 按照简单的主表单/详细子表单建立自行车表单。和已有的表单比较。优缺点各有哪些？
33. 创建表单，显示在某个指定时间范围内不同型号的自行车销售（价值和数量）的图表。时间范围应该从表单的文本框输入，带有重绘功能。
34. 创建表单，允许会计输入日期，并生成报表，列出每个供应商和截至该日期拖欠该供应商的货款。
35. 创建报表，显示各州、该州的相关商店和零售商店的总销售。包括州总销售。
36. 创建新的应用程序工具栏/菜单。包括根据顾客、制造商和供应商选择表单的菜单。包括指向该类的适当表单和报表的连接。
37. 创建报表，按型号和月份显示自行车销售。包括按型号比较总销售的图表。
38. 创建报表，按雇员和年份显示自行车销售。包括雇员每年的销售比较图表。
39. 创建报表，按月份绘制制造每种型号的自行车的平均时间图表。
40. 创建表单，允许管理者选择某个部件，然后生成图表，显示过去这个部件的购买和销售状况。

参考网站

网站	描述
http://www.microsoft.com/enable	可访问性指导
http://www.unicode.org	Unicode信息的主要网站
http://www.acm.org/sigchi/	美国计算机学会——特别兴趣组：人机交互
http://www.acm.org/sigcaph/	美国计算机学会——特别兴趣组：计算机与身体缺陷

补充读物

Cooper, A. *About Face: The Essentials of User Interface Design*. Foster City, CA: IDG Books, 1997. [A good discussion of various design issues.]

Ivory, M., and M. Hearst. "The State of the Art in Automating Usability." *Communications of the ACM* 33, no. 4 (December 2001), pp. 470–516. [General discussion on evaluating system usability.]

Koletzke, P. *Oracle Developer Advanced Forms and Reports.* Berkely: Osborne/ McGraw-Hill, 2000.

Microsoft Corporation. *The Windows Interface: An Application Design Guide.* Redmond: Microsoft Press, 1992. [An important set of definitions and "standards" that designers should follow.]

O'Reilly, Inc., ed. *The Oracle PL/SQL CD Bookshelf: 7 Best Selling Books on CD ROM*. Cambridge, MA: O'Reilly & Associates, 2000. [A collection of several useful Oracle reference books on CD-ROM.]

Raskin, J. *The Humane Interface: New Directions for Designing Interactive Systems*. Reading, MA: Addison-Wesley, 2000. [The need for a new interface as explained by the creator of the Apple Macintosh project.]

Tsichritzis, D. "Form Management," *Communications of the ACM* 25, no. 7 (July 1982). [Basic concepts of database forms.]

第 7 章

数据库完整性和事务

本章学习内容

- SQL如此强大，为什么还需要程序代码？
- 如何利用数据触发器自动地更新数据？
- DBMS如何保证相关的更新一起完成？
- 如何处理多用户同时更新相同的数据？
- 内部码值是如何生成的？在更新中如何使用？
- 数据库游标的目的是什么？

7.1 开发漫谈

Ariel： 嗯，应用部分结束了吗？

Miranda： 没有。基本表单和报表结束了。但是我还遇到了一些问题。

Ariel： 看来问题总是有的。什么问题呢？

Miranda： 哦，数据有的时候会发生错误。好像是当几个人同时操作同一个数据时发生的。而且应用程序有的时候比较慢。还有……

Ariel： 好。我知道了。但是这些问题看起来很普通。数据库系统还有别的辅助工具吗？

Miranda： 有的。我要开始看一些有关编程的内容和数据触发器。然后，我认为索引可以帮助提高性能。

7.2 简介

业务应用经常发生一些常见问题。例如，多个用户可能同时试图更新同一份数据，或者多个修改需要一起完成，或者需要生成表的新ID。这些问题必须正确处理，才能保证数据的完整性。虽然SQL命令是强有力的工具，但是在这些情况下，需要执行多个语句，或选择执行哪个命令。数据库系统调用过程语言来处理这些情况。

虽然有多种方法实现过程语言，但是把语言嵌入查询系统是有益的。用这种方式，所有的代码和条件仍旧在数据库定义和限制之中，而且可以在程序中自动执行。这些条件通常以数据触发器的形式给出，触发器是一段代码，当某个数据元素被修改时执行。

事务、并发访问和码生成问题是几乎每个事务程序都会遇到的。本章描述相关问题并给出常见的解决方案。

对于大数据集的数据库，性能是一个棘手的问题。涉及多张大表的复杂查询运行很长时间。但是，基于事务的应用程序需要快速处理数据。开发商已经投入了可观的资金和时间来提高性能。常见的解决方案是建立表的索引。读者需要理解基本的索引技术，有更多的方法提高应用程序的性能。

7.3　过程语言

过程语言是传统的程序语言，可以指定一组命令的执行顺序。常见的SQL命令不是过程语言，因为只需要告诉DBMS做什么，而不需要告诉它怎么做。虽然SQL命令强大，有时候需要控制更精确的过程语言。例如，指定一组命令必须按照某个特定顺序执行，并且必须所有的命令都执行完毕事务才算成功，或者，只有在某个外部条件满足时才执行一些命令。在更复杂的情况下，可能需要逐行地在表上进行操作，才能完成某个复杂计算。

7.3.1　代码应该放在哪里

图7-1展示过程语言代码经常出现的三个地方：(1) 查询系统；(2) 表单和报表；(3) 外部程序。一般来说，和数据直接相关的代码应该放在查询系统中。原因是直接了当的：把代码放在距离数据最近的地方，只需要写一次，而不是复制到多个表单中。更重要的是，DBMS可以保证代码执行而不是跳过。考虑一个安全情况，每次修改雇员工资都要向日志表写入记录。如果依靠程序员在表单中实现这段代码，他们可能忘记做，或者做错。还有，有人可以创建全新的表单，或者甚至使用查询语句直接修改数据，而不执行安全代码。把代码放在数据库中，提供了一种机制，保证在任何时候只要数据改变就可以执行，而不论这种修改如何产生。在SQL标准中，存放在数据库中的过程代码叫做持久存储模块（PSM），相关的过程和函数可以存放在开发商定义的模块中。

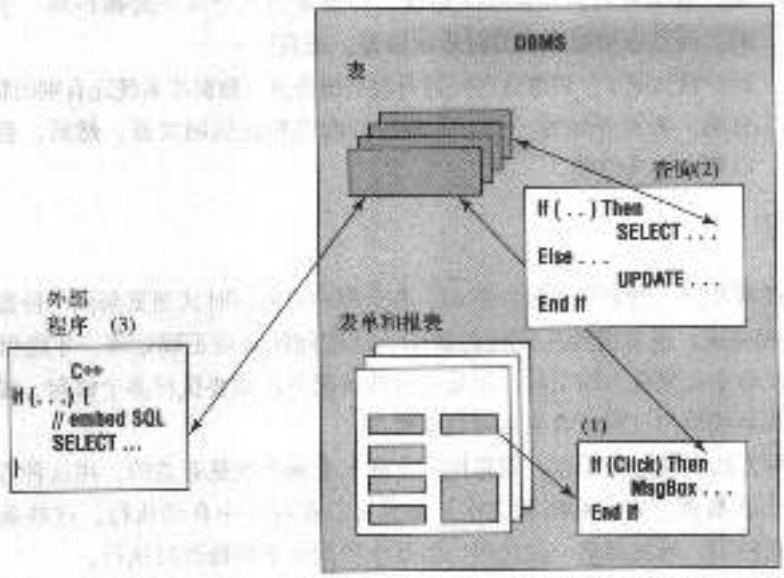


图7-1　过程代码的位置。代码通常位于查询系统、数据库表单或者外部程序中。在可能的情况下，代码应该放在查询系统中，这样不会被跳过

表单中的代码应该集中于处理事件或特定表单的定制问题。另一方面，把代码放在独立的外部文件是第10章介绍的多层客户机/服务器系统经常采用的技术。它具有把业务逻辑固化在某个位置的优势。业务逻辑和DBMS相分离使得必要时更换DBMS更为简便。

7.3.2 用户定义的函数

用户定义的函数是过程代码的最好例证。有时候，需要在多个不同查询中使用计算。即使计算相对简单，把代码放在一个地方，使得以后查找和修改更为简单。可以使用过程代码定义自己的函数名称，进行任何需要的计算。可以以参数的形式传递表的值。图7-2提供一个估算成本的简单函数的例子。实际上，这个函数使用表和查询更好，但是它说明了用户定义函数的基本思想。函数接收值，对这些参数进行计算，向调用程序返回值。还可以创建过程，它和函数的区别在于过程不返回值。但是，几乎所有的情况下，可以使用函数——只要返回错误代码。

```
CREATE FUNCTION EstimateCosts
  (ListPrice Currency, ItemCategory VarChar)
RETURNS Currency
BEGIN
  IF (ItemCategory = 'Clothing')THEN
    RETURN ListPrice * 0.5
  ELSE
    RETURN ListPrice * 0.75
  END IF
END
```

图7-2 用户定义的函数。即使函数不复杂，把业务逻辑放在一个集中的位置也会方便以后查找和修改。这个函数可以使用其他代码片断或使用SELECT语句

图7-3展示一个函数使用输入参数更新数据库。参数可以用在任何SQL语句中。还可以创建局部变量修改参数，然后在SQL语句中使用。如果有必要，函数可以任意复杂。过程语言系统包括任何程序语言的标准元素：变量、条件、循环、子例程。

```
CREATE FUNCTION IncreaseSalary
  (EmpID INTEGER, Amt CURRENCY)
RETURNS CURRENCY
BEGIN
  IF (Amt > 50000) THEN
    RETURN-1    --error flag
  END
  UPDATE Employee SET Salary = Salary + Amt
  WHERE EmployeeID = EmpID;
  RETURN Amt;
END
```

图7-3 更新数据库的函数。输入参数用于给SQL语句赋值。如果需要，可以增加计算，修改参数

7.3.3 查找数据

过程和函数经常需要使用来自表或查询的数据。从单行中获得数据可以直接使用SELECT INTO语句。它和标准的SELECT语句类似，区别在于它把值存在局部变量中，而不是显示值。但是，必须要保证SELECT语句只返回单行数据。如果在WHERE条件中发生错误，返回多行，就会产生错误。

图7-4展示如何使用SELECT INTO语句检索单个值。语句可以用来从多列中检索数据，但是只有一行。只需在SELECT行增加COLUMN INTO VARIABLE，并用逗号把它和已有行分开。注意图7-3和图7-4的区别：新的方法查询表的最大值。这比使用固定值更好，因为可以创建新的表单，允许管理者迅速修改这个值。如果在代码中使用固定值，程序员必须遍历所有模块查找该值。还有，只要有人修改程序代码，就存在引入新错误的巨大风险。如果可能，把重要的值放在表中，需要的时候使用查询程序获得当前值。

```
CREATE FUNCTION IncreaseSalary
  (EmpID INTEGER, Amt CURRENCY)
RETURNS CURRENCY
DECLARE
  CURRENCY MaxAmount;
BEGIN
  SELECT MaxRaise INTO MaxAmount
  FROM CompanyLimits
  WHERE LimitName = 'Raise';

  IF (Amt > 50000) THEN
    RETURN-1    --error flag
  END
  UPDATE Employee SET Salary = Salary + Amt
  WHERE EmployeeID = EmpID;
  RETURN Amt;
END
```

图7-4 查找单行数据元素。SELECT INTO语句可以用于从表或查询的某一行返回数据。结果存储在局部变量（MaxAmount）中，可以在以后的代码或SQL语句中使用

7.4 数据触发器

数据触发器是当数据库中某个事件发生时执行的过程。代码位于查询系统中，保存为数据库的过程或函数。常见的事件是更新、插入和删除，但是有的系统允许对用户或数据库实例相关的事件附加代码。例如，可以创建一个过程，当有人修改Employee表的Salary列时运行。当数据改变时，该过程记录做出修改的人。有了日志，审计员可以返回查看谁做出的修改。工资的例子是数据触发器的一个常见应用，它给数据库增加了特殊的安全或审计功能。它们还可以用于管理事务，例如监视当现有数量低于某个值时，生成电子邮件或EDI订单发送到供应商。

图7-5列出支持触发器的基本SQL命令。在行列上的每个主要数据触发器有两个属性：

BEFORE和AFTER。例如，可以实现一个BEFORE UPDATE的过程，和另一个AFTER UPDATE过程。BEFORE UPDATE事件是在用户试图修改数据，但是数据实际写入数据库之前触发。数据一写入数据库，AFTER UPDATE触发器才触发。根据应用程序想做什么，选择事件。如果想在写入数据库之前检查数据，需要使用BEFORE触发器。例如，想在保存数据之前进行复杂的合法性验证。另一方面，如果想记录数据何时修改，或者需要修改另一个数据，可以使用AFTER触发器。

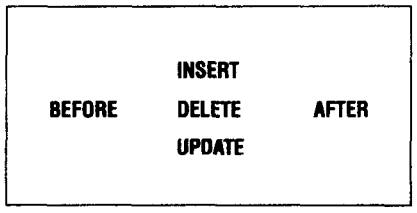


图7-5 数据触发器。可以设置过程在这些操作发生时执行。行事件可以在指定事件发生之前或之后触发

7.4.1 语句与行触发器

SQL标准定义两个层次的触发器：(1) 对于整个表的；(2) 对于要修改的每行数据。图7-6展示不同触发器对于UPDATE命令的工作时间。整个表的触发器首先（BEFORE UPDATE）触发，或者在最后（AFTER UPDATE）触发。单行触发器在检查每行之前或之后触发。对于行触发器，还可以增加条件检查行数据决定是否触发或忽略触发器。例如，在工资表中增加行触发器，只对某个部门的雇员触发。这个条件和原始的UPDATE WHERE 语句完全独立。触发器条件只用于决定是否触发。

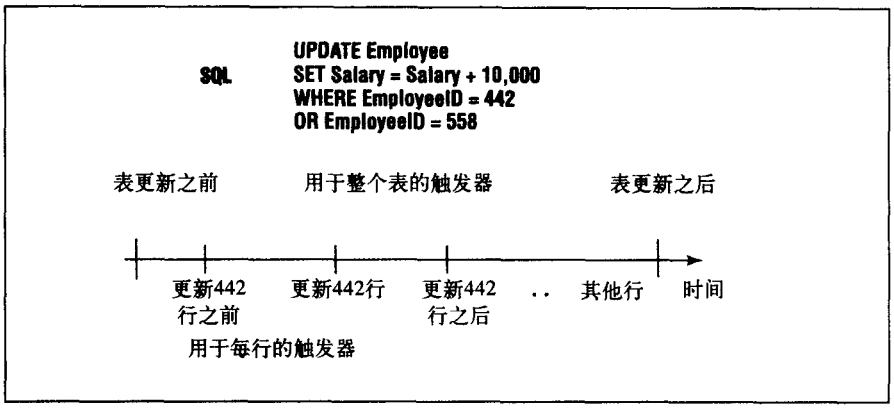


图7-6 更新触发器可以针对整个表，对于整个命令只触发一次，或者每行更新时触发

图7-7是一个触发器的例子，当修改Employee表的行时触发。由FOR EACH ROW语句，这是一个行触发器。这个例子还说明触发器可查询和使用存储在目标表中修改之前（OLD ROW）和修改之后（NEW ROW）的数据。在这种情况下，原始的工资和新的工资同时记录在日志表中。有了这些信息，安全管理者和审计员可以迅速查询日志表，确认主要的工资修改，进一步研究保证修改合法。必须小心使用OLD和NEW数据。例如，在BEFORE UPDATE触发器中，NEW数据还没有创建，所以不能访问。

```
CREATE TRIGGER LogSalaryChanges
AFTER UPDATE OF Salary ON Employee
REFERENCING OLD ROW as oldrow
            NEW ROW AS newrow
FOR EACH ROW
  INSERT INTO SalaryChanges
  (EmpID, ChangeDate, User, OldValue, NewValue)
  VALUES
  (newrow.EmployeeID, CURRENT_TIMESTAMP,
  CURRENT_USER, oldrow.Salary,
newrow.Salary);
```

图7-7　触发器记录修改雇员工资的用户日志。只要工资更新，不论使用何种方法修改数据，触发器就会触发。这是一个有用的敏感数据安全跟踪技术，因为无法避开它，除非是触发器的拥有者

7.4.2　利用触发器取消数据更新

触发器的一个用途是在写入数据库之前仔细检查修改。BEFORE UPDATE和BEFORE INSERT触发器经常用于验证复杂条件。删除数据之前，可能希望更仔细地检查。在这些情况下，触发器的结构非常简单。惟一的不同是需要找到一种方式停止执行原始SQL语句。WHEN条件用于检查即将删除的行。如图7-8所示，SIGNAL语句产生错误条件阻止删除行。实际的信号条件（CANNOT_DELETE_PRESIDENT）可以是任何字符串，但是在整个模块中必须定义为常量。大部分数据库系统不支持SIGNAL关键词，所以实际的语法依赖于你所使用的系统（和版本）。

```
CREATE TRIGGER TestDeletePresident
BEFORE DELETE ON Employee
REFERENCING OLD ROW AS oldrow
FOR EACH ROW
  WHEN (oldrow.Title = 'President')
       SIGNAL CANNOT_DELETE_PRES;
```

图7-8　取消的SQL命令。这个触发器检查要删除的雇员行的数据。公司总是希望保留带有“President”标志的雇员数据。WHEN条件逐行计算。SIGNAL语句产生一个错误阻止执行下面的删除

总的来说，对于简单的检查条件应该尽量避免使用触发器。相反，使用标准的SQL条件（例如，PRIMARY KEY、FOREIGN KEY和CHECK），因为它们更有效，很少引起其他的问题。

7.4.3　级联触发器

触发器的复杂性在于数据库的每个表可以有多个触发器。级联触发器是指当某个修改触发了某张表的触发器导致另一张表的修改，接着又导致第三张表的修改，如此继续。图7-9是一个常见的库存情况。当某个项目卖出去，SaleItem表加入包含卖出数量的新的一行。因为该项已经卖出，Inventory表的当前数量也随之更新。Inventory表的触发器检查QOH是否低于再订购点。如果是，生成新的订单并向供应商发出电子邮件，其结果是向Order和OrderItem表插入新项。

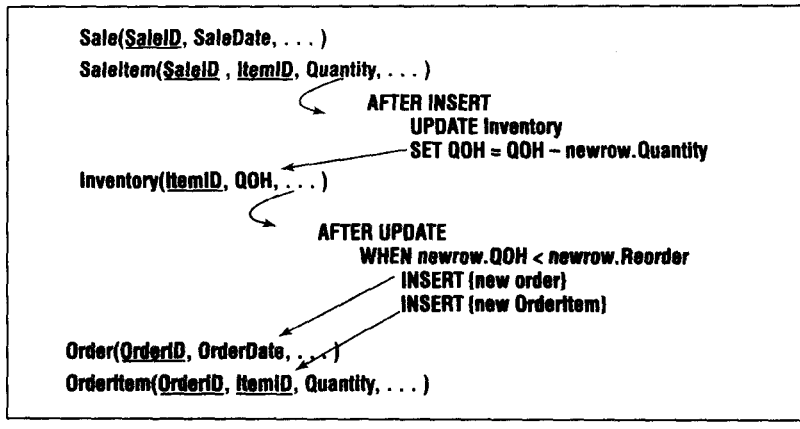


图7-9 级联触发器。当多张表上定义了触发器，一张表（SaleItem）上的修改可以导致其他表的级联修改。在这里，当某个项目卖出，当前数量更新。如果QOH低于再订购点，生成并发送新的订单

级联触发器本身并没有错。但是，过长的更新链可能降低系统速度。它们还给调试系统和查找错误带来困难。在这个例子中，比如查找OrderItem表的错误，这个错误可能由SaleItem表的触发器代码导致。这个链越长，定位问题源就越具有挑战性。

级联触发器潜伏着更复杂的问题。如果这个链构成自身循环会发生什么？图7-10举例说明这个问题。公司已经设置了几条关于支付员工工资的规则。当工资达到某个水平时，员工可以获得奖金。当员工已经得到了一定的奖金时，限制奖金的数量，同时员工得到附加的股权。如果股权达到了某个水平，降低原始工资。但是这导致系统回到开始状态，工资的变化触发新一轮的更新。根据计算结果的不同，循环可能导致发散，即数字越来越大（或反向增长），并且计算无法结束。鉴于此，SQL标准定义禁止触发器循环。遵循这一标准的系统会监视整个更新链，如果遇到循环，取消修改并发出警告。即使系统监控这类循环，读者也应该亲自检查系统避免问题发生。很明显，如果只有有限数量的触发器，系统很容易检查。

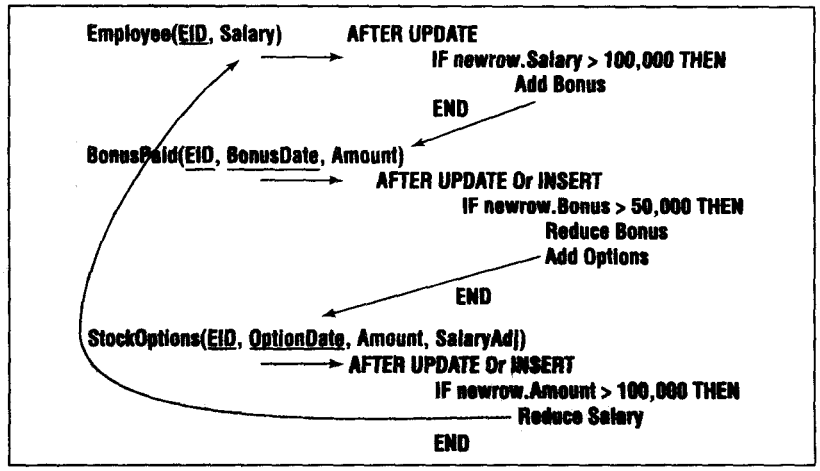


图7-10 触发器循环。考虑级联触发器构成循环会发生什么，即其中一个触发器返回修改了引发最初被修改的表。这个循环可能收敛或发散。即使循环收敛，也会消耗大量资源

7.4.4 INSTEAD OF触发器

有的数据库支持INSTEAD OF触发器，这是一种功能更强大的触发器。标准的触发器运行代码，附加执行基本的函数（DELETE、INSERT或UPDATE）。INSTEAD OF选项完全取代了代码中基本的命令。这样，即使修改应该写入数据库，也必须编写额外的SQL语句执行适当的动作。这是一种使查询可更新的有效技术。连接多张表的查询一般是不可更新的；数据无法加入查询，因为系统无法知道哪些表有了新行。为了解决这个问题，可以在查询中加入INSTEAD OF触发器。这样必要的修改可以用单独的SQL语句插入独立的表中。

7.5 事务

在构造应用程序时，各部分一直正常工作并且从不发生问题的确诱人。诱人，但不一定正确。即使代码正确，也可能发生问题。可能面临电源切断、硬件崩溃、或者可能有人偶然拔错电缆。通过实现备份和恢复程序，在不同驱动器上保存复制数据，安装不间断电源（UPS）使这些问题最小化。但是，不论如何努力，失败总会发生。

7.5.1 事务的例子

在错误时间发生的错误可能导致严重后果。特别地，很多业务操作要求多处修改数据库。事务定义为一组必须一起完成的改变。考虑图7-11的例子。在银行的系统中，某顾客操作ATM把1 000美元从储蓄账户转移到支票账户。这个简单的事务包括两步：（1）储蓄账户余额减去资金；（2）支票账户余额增加资金。创建这个事务的代码要求数据库的两个更新。例如，有两个SQL语句：一个UPDATE命令减少储蓄账户余额，和第二个UPDATE命令增加支票账户余额。

必须考虑如果在两个操作之间机器发生故障会导致什么后果。资金已经从储蓄账户减去，但是还未加入支票账户。资金丢失了。读者可能考虑先给支票账户增加资金，但是顾客会得到多余的资金，而银行遭受损失。关键在于两处修改必须同时成功。

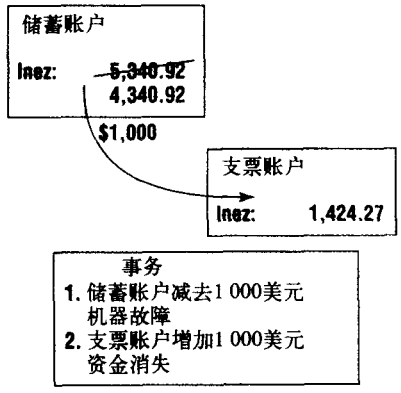


图7-11　涉及数据库多处修改的事务。所有的修改必须同时生效，否则事务出错。例如，为了把资金从储蓄账户转移到支票账户，系统必须减少储蓄账户的资金，同时把减少的资金加入支票账户。如果在减去资金之后、增加支票账户之前机器出现故障，资金发生丢失

7.5.2 事务的开始和结束

如何知道两个操作属于同一事务呢？这是商业规则，或者是资金转移定义。问题是，计算机如何知道两个操作必须同时完成？作为应用程序开发者，必须告诉计算机系统哪些操作属于一个事务。为了做到这一点，需要标记代码中所有事务的开始点和结束点。当计算机看到开始标记时，首先把所有的修改写入日志文件。当它到达结束标志时，对数据表作实际的修改。如果在修改完成之前发生错误，当DBMS重启后，它检查日志文件并完成所有未完成的事

务。从开发者的角度看，DBMS的好处在于自动处理问题。读者需要做的仅仅是标记事务的开始和结束。

事务处理是需要过程语言的一个例子。如图7-12所示，多个UPDATE语句需要保存在一个模块函数或过程中。在这个例子中，两个UPDATE语句必须同时完成或同时失败。START TRANSACTION语句是可选的（SQL99标准），但是突出了事务的开始。如果两个更新成功完成，COMMIT语句执行，告诉DBMS保存所有修改。如果发生了意外的错误，ROLLBACK语句执行，不保存任何修改。大部分系统通过把所有的修改写入中间日志文件管理事务请求。如果事务发生错误，系统可以恢复日志文件，回滚或完成事务。

```
CREATE FUNCTION TransferMoney(Amount Currency,
   AccountFrom Number,AccountTo Number)
  RETURNS NUMBER
curBalance Currency;
BEGIN
  DECLARE HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
    Return-2;            --flag for completion error
  END;
  START TRANSACTION;    --optional
  SELECT CurrentBalance INTO curBalance
  FROM Accounts WHERE (AccountID = AccountFrom);
  IF (curBalance < Amount) THEN
    RETURN-1;            --flag for insufficient funds
  END IF
  UPDATE Accounts
  SET CurrentBalance = CurrentBalance - Amount
  WHERE AccountID = AccountFrom;
  UPDATE Accounts
  SET CurrentBalance = CurrentBalance + Amount
  WHERE AccountID = AccountTo;
  COMMIT;
  RETURN 0;              --flag for success
End;
```

图7-12 转移资金的事务。如果系统在事务结束（提交）之前出错，任何修改都不写入数据库。重启后，修改可能全部回滚，或者事务重启

注意，START TRANSACTION行在SELECT语句开始之前。这看上去没有必要，似乎只有UPDATE命令需要在事务中。把这句话放在前面有语法原因：任何SELECT语句自动开始一个新的事务。但是，在并发控制部分还将给出解释，是在SELECT语句之前开始事务的更好原因。

7.5.3 保存点

有时候，一个事务需要中间点。一些元素比其他的更重要。可能有一些选择性修改需要保存，但是，如果它们失败，仍需要保证提交关键的更新。保存点（SAVEPOINT）技术把事务过程分成多个片断。可以回滚到事务的开始，或者某个保存点。图7-13说明了这一过程，展示了设置保存点和回滚到它的语法。正如图中所示，它可以用于标记一组包括在更新中但却不

是必要的风险步骤。因此，如果风险部分更新失败，可以舍弃这些修改，保持事务开始定义的元素不变。一般来说，使用多个COMMIT语句可以完成同样的工作，但是有时可选的代码可能包括最后结果中需要的计算。如果没有保存点选项，必须重写最后的值。



图7-13　保存点。保存点允许回滚到程序的一个中间点。可以设置多个保存点，并选择回滚多少修改

7.6　多用户与并发访问

数据库最重要的功能之一是多用户或不同进程能够共享数据。在现代商业应用中，这是一个重要的概念：很多人需要同时使用应用程序。但是，这导致了数据完整性的潜在问题：如果两个人同时试图修改同一份数据会发生什么？这种情况就是并发访问。考虑图7-14的邮购系统的例子。公司记录顾客的基本数据，监督顾客的付款和收据。顾客还有未结清的货款，即他们当前的债务。在这个例子中，琼斯欠公司800美元。当琼斯付款之后，办事员收到付款并检查余额（800美元），办事员输入付款数目（200美元），计算机做减法得到新的债务余额（600美元）。新值写入顾客表，代替旧值。至此，没有问题。如果琼斯做出新的购买，过程类似。只要这两个事件不同时发生，就没有问题。



图7-14　并发访问。如果两个程序同时试图修改同一份数据，结果发生错误。在这个例子中，接受付款后的修改被同时提交的新订单覆盖

但是，如果两个事务同时发生会怎么样？考虑下面的情况混合：（1）付款办事员接受付款，计算机查询琼斯的当前欠款余额（800美元）。（2）办事员输入付款200美元。在事务结束之前，琼斯正在给另一个办事员打电话，订购150美元的新商品。（3）这个办事员的计算机也

读取当前欠款余额（800美元），并增加新的购买。现在，在这个事务结束之前，第一个完成了。(4) 付款办事员的计算机得到琼斯目前欠600美元并保存欠款余额。(5) 最后，订货办事员的计算机把新的购物加入欠款余额。(6) 订购计算机保存新的欠款（950美元）。顾客琼斯在接收到下一个账单时注定要心烦意乱了。200美元的付款发生了什么？答案是当新订购修改和接受付款混合在一起时，付款被覆盖（并丢失）了。

7.6.1 悲观锁：串行化

解决并发控制问题的一种方案是强制事务以完全隔离的方式执行，完全禁止并发执行。如图7-15所示，串行化进程强制事务独立运行，这样一个进程甚至无法读取正被另一个进程修改的数据。

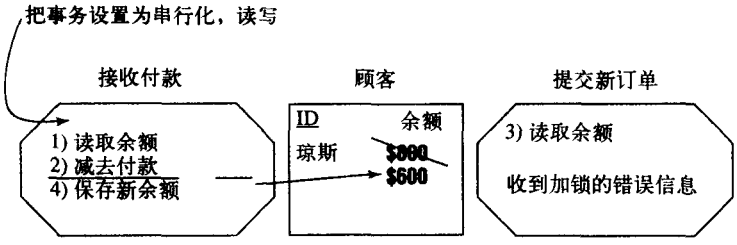


图7-15 串行化。第一个进程对数据加锁，这样第二个进程甚至无法读取。通过强制每个进程等待前一个进程完成，避免并发修改

这种类型的锁机制的使用依赖于DBMS。SQL99定义了标准的指定事务锁的方法，但是没有广泛实现。如图7-16展示基本的逻辑，但是记住不同DBMS的语法可能不同。主要步骤是在SET TRANSACTION语句中指定SERIALIZABLE的隔离等级。然后DBMS给每个可能使用的数据加锁，这样其他的事务在第一个修改提交之前无法读取该数据。但是，所有的事务过程包括错误管理代码很重要。否则，当第二个事务（RecordPurchase与此几乎相同）运行，试图更新或读取数据时，它失败并显示含糊的错误消息。

```
CREATE FUNCTION ReceivePayment(
  AccountID NUMBER, Amount Currency) RETURNS
NUMBER
BEGIN
  DECLARE HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
    RETURN-2;
  END
  SET TRANSACTION SERIALIZABLE, READ WRITE;
  UPDATE Accounts
  SET AccountBalance = AccountBalance - Amount
  WHERE AccountNumber = AccountID;
  COMMIT;
  RETURN 0;
END
```

图7-16 转移资金的事务。如果在事务结束（提交）之前系统出错，没有任何修改写入数据库。重启后，修改可能全部回滚，或者事务重启

串行化的概念是合乎逻辑的，它强调了强迫每个事务完全分离的重要性。但是，它基于悲观锁技术——每个事务都假定并发引用总会发生。每次事务运行，给所有需要的资源加锁。这个技术可以导致下一节描述的另一个严重问题。

7.6.2　多用户数据库：并发访问与死锁

如果在同一时刻，有两个进程试图修改同一个数据，那么会发生并发访问的问题。当两个进程混合起来，出现一个事务丢失和数据不正确的情况。对于大部分数据库操作，DBMS可以自动处理这些问题。例如，如果两个用户打开表单并试图修改同一份数据，DBMS会给出适当的警告并在第一个修改完成之前阻止第二个用户修改。类似地，两个SQL操作（例如，UPDATE）不能同时修改同一数据。

即使编写程序代码，DBMS也不允许两个进程同时修改同一数据。尽管如此，必须理解代码中的数据修改可能不被允许。这种情况经常作为错误处理。

并发问题的解决方案是强制每个数据一次只发生一个修改。如果两个进程试图修改，第二个停止并等待第一个进程结束。这样做产生的延迟可能导致另一个问题：死锁。当两个（或多个）进程已经对数据加锁并希望对其他数据加锁时发生死锁。如图7-17的例子。进程1已经对数据A加锁，进程2已经对数据B加锁。不幸的是，进程1正在等待B释放，而进程2正在等待进程A释放。除非发生其他事件，这会导致长时间的等待。

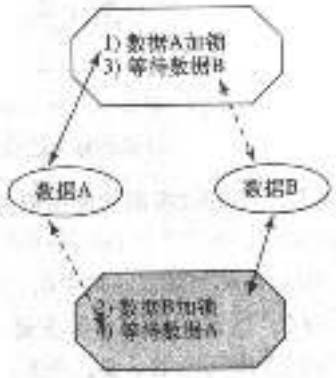


图7-17　死锁。进程1已经对数据A加锁，并等待数据B。进程2已经对数据B加锁，并等待数据A。为了解决这个问题，必须有一个进程放弃并释放它的锁

死锁问题有两个常见的解决方案。第一，当一个进程收到它必须等待某个资源的消息时，该进程随机等待一段时间之后，重新尝试，释放所有已经存在的锁，如果还没有得到资源则重新开始。这个方法之所以有效是因为随机等待。两个死锁的进程其中之一会先尝试放弃，并使用ROLLBACK语句释放所有的锁。释放为其他进程完成工作扫清了道路。这种解决方案很常见，因为它容易编程实现。但是，它的缺点是导致计算机花费大量时间等待——尤其当很多活动进程导致很多冲突的时候。

更好的解决方案是DBMS建立全局锁管理器，如图7-18所示。锁管理器管理每个锁并要求锁（等待）。如果锁管理器发现潜在的死锁，会告知某些进程释放锁，允许其他进程继续，然后重启那些放弃的进程。这种解决方案效率更高，因为进程没有等待时间，另一方面，这种解决方案只有DBMS本身才能实现。锁管理器必须能够监控每个进程及其锁。

对于典型的数据库表单和查询操作，DBMS自动管理并发访问和死锁处理。当通过编写代码修改数据时，DBMS仍试图自动管理，但是，DBMS可能希望程序撤销事务。有的系统在有其他进程试图访问数据时简单地生成错误信息，程序必须捕获错误并处理问题。

	资源A	资源B	资源C	资源D	资源E
进程1		锁		等待	
进程2	等待			锁	
进程3			锁		
进程4	锁				等待
进程5				等待	
进程6		等待			锁
进程7			等待	等待	

图7-18 锁管理器。全局锁管理器跟踪所有加锁的资源和相关进程。如果发现循环，那么死锁存在，然后锁管理器指导进程释放锁，直到问题解决

7.6.3 乐观锁

多年来，悲观锁和死锁管理是很多系统的标准做法。最近，另一种方法受到欢迎。随着计算机系统性能提高，DBMS可以更快地处理事务，其结果是出现并发问题的可能性降低了。乐观锁假设冲突很少或不可能发生。如果确实发生了，再处理也容易。悲观锁和死锁方案需要可观的编程量和计算机资源才能正确管理。尤其在分布式数据库环境中，使用乐观锁更容易，也更快。

理解乐观锁的关键在于认识到它们不是实际的锁；DBMS允许程序读取任何需要的数据。当程序试图修改数据时，DBMS重新读取数据库，并把当前存储的数据和先前获得的进行比较。如果这两个值不同，表明存在并发问题，因为任务完成之前有其他人修改了数据。DBMS产生一个错误，并期望程序处理。总的来说，乐观锁可以提高性能，但是它要求程序员处理潜在的冲突。图7-19列出了基本过程。

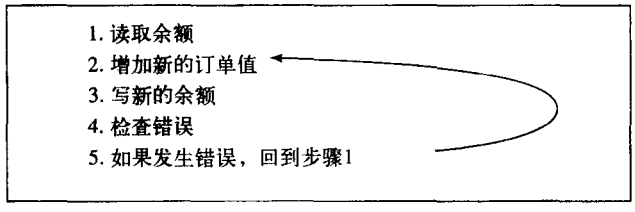


图7-19 乐观锁进程。这些步骤假定并发问题不会发生。如果在该事务结束之前，另一个事务修改了数据，代码收到错误消息并重新启动

前面提到的使用乐观锁处理冲突的方法是撤销已经做出的全部修改，然后重新开始，从数据库读取当前值，重新计算修改，再把新值写入数据库。考虑图7-20的例子。函数首先读取当前余额至内存。在完成一些其他工作之后，转回尝试UPDATE命令。它规定UPDATE命令只能应用于带有给定账号和原始余额的行。如果值被其他事务修改，UPDATE命令不会修改任何行。UPDATE命令后面的错误测试会识别修改成功与否。如果成功，程序完成并退出。如果修改失败，程序员可以完全控制做什么。在这种情况下，应该回到开始，获取修改后的余额并重新

尝试。为了安全起见，应该增加重新尝试的计数。如果这个数太大，那么程序应该放弃，发出错误信息：现在无法更新数据。

```
CREATE FUNCTION ReceivePayment (
  AccountID NUMBER, Amount Currency) RETURNS NUMBER
oldAmount Currency;
testEnd Boolean = FALSE;
BEGIN
  DO UNTIL testEnd = TRUE
  BEGIN
    SELECT Amount INTO oldAmount
    WHERE AccountNumber = AccountID;
    ...
    UPDATE Accounts
    SET AccountBalance = AccountBalance - Amount
    WHERE AccountNumber = AccountID
    AND Amount = oldAmount;
    COMMIT;
    IF SQLCODE = 0 And nrows > 0 THEN
      testEnd = TRUE;
      RETURN 0;
    END IF
    --keep a counter to avoid infinite loops
  END
END
```

图7-20　使用SQL的乐观并发控制。在内存中保存初始值，只有值未改变才做更新。如果在这个事务完成之前，另一个事务修改了数据，返回，得到新值，重新开始

即使涉及多个分布式数据库的事务，这种方式仍然有效。但是，它要求程序员对每个更新编写并验证合适的代码。因此，有必要创建代码库，包含可以为几乎所有事务调用的通用UPDATE命令。

这种做法的另一个优势在于程序代码可以包含相对自动处理通用更新问题更精确的分析。在这个例子中，悲观锁几乎必然停止事务，要求办事员或顾客稍候再试。另一方面，乐观锁知道仅仅获取新的余额重新计算即可。没有交互，也几乎没有延迟。

7.7　事务的ACID特征

完整性是数据库的基本概念。使用数据库的优点之一是DBMS有工具管理常见的问题。在事务方面，很多概念可以总结为缩写的ACID。图7-21展示这个词的含义。*原子性*（Atomicity）代表核心问题，事务的所有操作要么全部成功，要么全部失败。*一致性*（Consistency）的意思是数据库的所有数据必须最终保持一致。即使事务进行中存在暂时的不一致，数据库最终必须回到一致状态。这种情况必须能够用定义的应用程序代码检验。例如，事务结束时必须实现参照完整性。*隔离性*（Isolation）的意思是阻止并发访问问题。一个事务的修改不会导致其他事务的错误。注意事务很少做到完全隔离：它们可能需要处理悲观或乐观锁的消息。*持久性*（Durability）的意思是提交的事务是永久的。一旦事务提交了修改，即保持这个修改。

这个概念在硬件或软件失败时至关重要。对于分布式数据库环境则更难保证。大部分系统通过把修改写入日志文件保持持久性。这样，即使硬件失败中断了更新过程，修改也会在系统重新启动后完成。需要注意的是，如果接受了COMMIT语句，DBMS就无法撤销修改。

- 原子性：所有的修改要么一起完成，要么一起失败
- 一致性：所有的数据保持内部一致性（提交时），可以通过程序合法性检查
- 隔离性：系统给每个事务隔离的感觉没有并发访问问题
- 持久性：事务提交以后，所有的修改永远保存，即使出现硬件或系统故障

图7-21 事务的ACID特征。ACID突出了事务的四个主要的完整性要求

在SQL99中，START TRANSACTION和SET TRANSACTION命令可以用于设置隔离等级。隔离的四个等级为，READ UNCOMMITTED、READ COMMITTED、REPEATABLE READ和SERIAIZABLE。这些等级可以用于阻止不同类型的并发问题，但是很少需要中间等级，所以很多系统只提供第一个和最后一个。

READ UNCOMMITED等级几乎不提供隔离性。它允许程序读取其他程序正在修改、还没有提交的数据。这个问题有时叫做脏读，因为收到的数据可能回滚，数值可能实际上不准确。如果选择这个等级，SQL不允许事务更新任何数据，因为这可能在数据库中传播错误数据。READ COMMITTED等级类似于乐观并发控制。它会阻止事务读取未提交的数据，但是数据仍然可能在第1个事务完成之前被另一个事务修改或删除。

REPEATABLE READ等级阻止正在使用的指定数据被修改或删除。但是没有解决幻影问题。考虑一个事务，如果商品的价格降低但在某个选定区域内，则计算当前的总数量。现在，另一个事务降低了一些其他商品的价格，使得它们降低到选定的范围。重新执行第一个命令或使用同样的选择条件创建另一个命令会导致分析另外的（幻影）数据行。图7-22展示第二个事务的进程可以导致新的行达到要求，所以第一个事务现在操作另一些行。

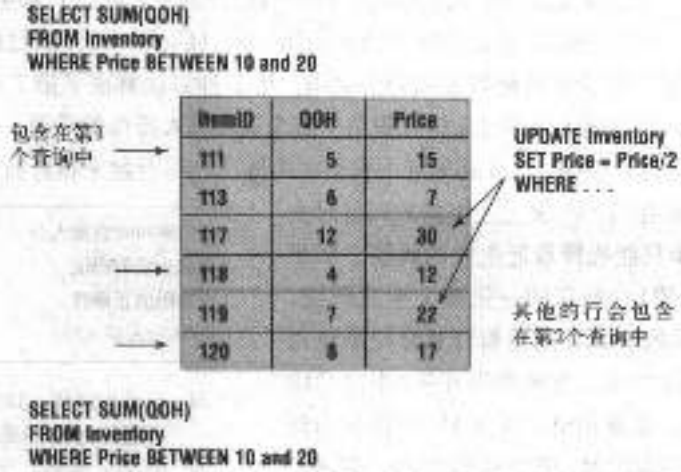


图7-22 幻影行。左边的事务第1次只选中3行数据。第2个事务运行时，其他的行符合要求，这样第2次执行第1个查询时会返回不同的结果

SERIALIZABLE隔离等级通过保证所有的事务如同按照顺序执行一样，阻止幻影行问题。但是，记住这个结果通常是通过使用锁得到的，所以需要数据库资源，而且不保证事务一次执行成功。当一个事务被另一个事务阻塞时，还需要错误管理机制捕获并解决这些问题。

7.8　码生成

现在已经知道，关系数据库与主码关系密切，而主码必须惟一。在商业中，保持主码一直正确的生成存在困难。因此，大部分关系数据库有一个生成惟一的数字码的机制。虽然这些方法对于简单的项目工作良好，但生成码值提出了必须用程序解决的挑战。而且，记住每个DBMS使用不同的机制生成码。

当向表中加入行，然后把相关的码值插入另一个表时，遇到码生成的主要问题。例如，当加入新的顾客时，系统生成CustomerID，需要把这个值插入Order表。图7-23显示基本的问题：为新顾客生成的CustomerID码必须在事务过程中保存，这样该码才能插入Order表。当然这两步必须在一个事务中。尽管如此，不同的生成码的方法使得这一问题更加困难。



图7-23　生成的码。为新的顾客创建订单，要求生成CustomerID码，用于Customer表，同时必须保存它，这样才能在Order表中使用

从逻辑上讲，可以通过两种方式生成码：（1）当新行加入表时，自动生成；（2）独立的码生成程序。第一种方法的优点是把新行加入初始的表（Customer）的过程相对简单，缺点是难以保证得到第二张表中可用的正确的生成码。第二种方法解决了第二个问题，但是使得码生成更加困难，并且要求程序员保证过程位于每个表和插入操作的后面。

如图7-24所示，如果DBMS自动为每个表生成码值，代码看起来相对简单，复杂性在于当两个事务几乎同时在同一个表上生成新的码值时会发生问题。第2步只能选择最近生成的码值。如果第二个事务在步骤1之后在同一张表上生成码值，步骤2会得到错误的值，导致销售订单分配给错误的顾客。为了安全起见，代码需要在第2步之后检验码值正确与否，需要用SELECT INTO语句为新创建的顾客检索顾客数据，然后比较姓名、电话号码和其他属性。另一种方式是忽略DBMS码值，直接使用SELECT语句通过指定每个顾客属性从Customer表中查找码值。使用这种方式必须保证

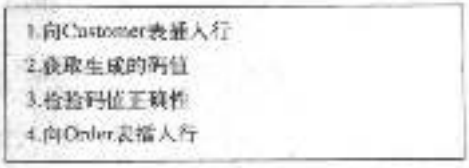


图7-24　自动生成码。DBMS自动生成码值的过程看起来简单。但是，在步骤2如果两个事务几乎同时在同一张表上生成新的码值会发生什么

两个顾客不会有相同的属性。当然，顾客可能重名。但是他们的地址和电话可能也相同吗？不要轻易回答这个问题。居住在一起的父子两人如何？

由于获得自动生成码值的困难性，第二种调用码生成程序的方法很有优势。图7-25表示为新顾客创建订单的基本步骤。这里没有码生成的不确定性。生成程序保证码值惟一——即使两个事务同时请求码值。这种方法的缺点是不够自动。这段代码必须加入应用程序中每个需要码值的地方。

1.生成码作为CustomerID
2.向Customer表插入行
3.生成码值作为OrderID
4.向Order插入行

图7-25 码生成程序。步骤本身不难，但是程序员必须把它们加入到每个表和每个需要插入数据的程序中

7.9 数据库游标

至此，所有的过程和函数与DML语句或单行SELECT语句相关。这些语句或者不返回值，或者只返回一行数据。这个限制简化程序逻辑，使学习SQL的过程语言更容易一些。但是，有些应用程序需要复杂的查询：返回多行数据的SELECT语句。

回忆SQL命令操作数据集合——一次多行。如果希望更精确地控制该怎么办？也许需要一次检查一行执行复杂的计算，比较来自外部设备的数据，或者向用户显示一行并得到响应。也许需要把一行数据和另一行相比较。例如，可能想对两行数据做差。用标准的SQL难以完成这些工作。

7.9.1 游标基础

SQL能够一次一行地遍历数据集合。创建*数据库游标*，它定义了SELECT语句，并且一次指向一行。可以通过循环语句把游标移动到下一行，对于查询返回的每一行反复执行代码。

图7-26展示在Customer表上创建游标和循环计算拖欠资金总数的基本程序结构。当然，这个计算使用简单的SELECT语句做更方便快捷。这里的目的是展示实现数据库游标的主要代码结构。DECLARE CURSOR语句定义检索数据行的SELECT语句。虽然例子中只使用一列，可以使用任何常见的包括多列、WHERE条件和ORDER BY行的SELECT语句。必须打开（OPEN）游标才能使用，最后应该关闭（CLOSE）游标释放数据库资源。当游标第一次打开时，它直接指向第一行数据之前。FETCH语句检索一行数据，然后把该行的数据列存放在程序变量中。循环用于遍历每个匹配选择条件的行。

7.9.2 可滚动的游标

在默认情况下，FETCH命令获取下一行。如果FETCH命令把游标移动到数据集的结尾，产生一个错误条件。可以使用WHENEVER语句捕获这个错误，或者检查SQLSTATE变量，看最后一个SQL语句是否产生错误。5个0的字符串表示最后一个命令成功。

FETCH命令有多个功能选项，把游标移动到不同的行。常见选项有NEXT、PRIOR、FIRST和LAST。它们获取其代表的行。图7-27列出一个游标程序，从最后一行开始，向上移动到第一行。注意，必须用SCROLL关键词把游标声明为可滚动的。当然，把数据按照相反的顺序排列，然后向前移动效率更高；但是这里为了展示另一种移动方向。附加的FETCH滚动选项包括移动到第一行（FETCH FIRST），以及跳转到数据集的特定行。例如，FETCH

ABSOLUTE 5返回数据集的第5行。因为很少准确地知道返回的行号，相对滚动选项更为有用。例如，FETCH RELATIVE−3从当前行向后跳转三行。

```
DECLARE cursor1 CURSOR FOR
  SELECT AccountBalance
  FROM Customer;
sumAccount, balance Currency;
SQLSTATE Char(5);
BEGIN
  sumAccount = 0;
  OPEN cursor1;
  WHILE (SQLSTATE = '00000')
  BEGIN
    FETCH cursor1 INTO balance;
    IF (SQLSTATE = '00000') THEN
      sumAccount = sumAccount + balance;
    END IF
  END
  CLOSE cursor1;
  --display the sumAccount or do a calculation
END
```

图7-26 SQL游标结构。在SQL标准中，DECLARE、OPEN、FETCH和CLOSE是主要的语句

```
DECLARE cursor2 SCROLL CURSOR FOR
SELECT ...
OPEN cursor2;
FETCH LAST FROM cursor2 INTO ...
Loop...
  FETCH PRIOR FROM cursor2 INTO ...
End loop
CLOSE cursor2;
```

图7-27 FETCH选项。可滚动的游标可以向任何方向移动。这段代码移动到最后一行，然后在表中反向移动。其他的FETCH选项包括FIRST、ABSOLUTE和RELATIVE

在行列表中向后移动暴露了另一个事务并发问题。如果在一些行上工作并使用FETCH PRIOR命令会怎样呢？在大部分情况下，会得到当前行的前一行。但是，如果另一个事务在FETCH PRIOR命令执行之前迅速插入新的一行会发生什么？图7-28表示这个问题。代码已经遍历到Carl，但是另一个过程向列表中插入Bob。FETCH PRIOR命令返回Bob的数据，而不是期望的Alice的数据。SQL解决这个问题的标准方法是使数据集对其他修改不敏感。在游标声明中加入关键词即可（DECLARE cursor3 INSENSITIVE CURSOR FOR…）。实际上，DBMS把查询结果复制到不受其他命令影响的临时表里。虽然这种方式可行，但它极大地消耗数据库资源。相反，仔细考虑为什么需要向后移动。在大部分情况下，这并不是必须的。例如，如果希望通过当前行的值减去前一行的值计算差值，只要在内存中保存前一行的值，然后获取下一行，做减法即可。不需要向后移动，还冒出错的风险。

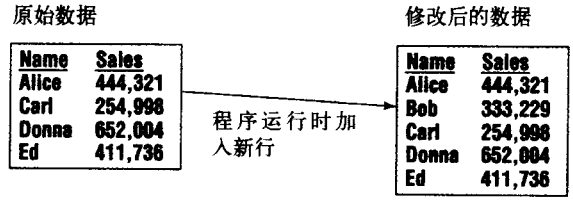


图7-28 游标代码中的并发事务。游标代码已经在数据中遍历到Carl。然后试图使用FETCH PRIOR返回到前一行。但是，如果其间另一个事务插入新的一行（Bob），代码会获得这一行，而不是原来的（Alice）

读者可能注意到没有在查询得到的数据集中找到某一行，并把游标移动到该行的过程（类似SEEK命令）。虽然有的系统提供这个功能，但是很少使用。相反，应该创建WHERE条件只搜索想要的行。

7.9.3 利用游标修改或删除数据

基于游标的程序经常遇到的情况是需要修改或删除当前行的数据。例如，图7-29展示创建的表保存销售数据供分析使用。标准的SELECT命令带有GROUP BY子句可以计算年度销售总额。可以写基于游标的程序计算每年销售的增长（或降低）。要注意的是，需要把计算后的值保存到表中。为了做到这一点，需要指定游标为可更新的，然后用UPDATE语句保存当前游标指向的行的计算值。图7-30表示执行计算需要的主要代码。

年份	销售	增长
2000	151,039	
2001	179,332	
2002	195,453	
2003	221,883	
2004	223,748	

图7-29 销售分析表。标准的SELECT查询可以计算并保存年度总销售。现在需要写基于游标的程序计算与前一年相比的销售增长

```
DECLARE cursor1 CURSOR FOR
SELECT Year, Sales, Gain
FROM SalesTotal
ORDER BY Year
FOR UPDATE OF Gain;
priorSales, curYear, curSales, curGain
BEGIN
  priorSales = 0;
  OPEN cursor1;
  Loop:
    FETCH cursor1 INTO curYear, curSales, curGain
    UPDATE SalesTotal
    SET Gain = Sales - priorSales
    WHERE CURRENT OF cursor1;
    priorSales = curSales;
  Until end of rows
  CLOSE cursor1;
  COMMIT;
END
```

图7-30 更新用游标代码。声明中FOR UPDATE选项允许修改Gain列。WHERE CURRENT OF语句指定游标指向的行

注意，游标声明只有Gain列可更新。这个功能稍稍保护数据库。如果发生错误，或者有人以后修改代码，DBMS只允许修改Gain列。修改Year和Sales列会引发错误。另一个重要元素是WHERE CURRENT OF cursor1语句。这个条件指出，修改当前获得的行，或者游标指向的行。UPDATE语句只能应用于这一行。经典的删除当前行的语句是DELETE FROM SalesTable WHERE CURRENT OF cursor1。

7.9.4 带参数的游标

偶尔需要计算更多动态的查询，即，想根据程序中的某个变量选择一些特定的行。例如，用户可能输入价格，或者程序根据其他的查询计算价格。然后要搜索低于指定价格的行，并在其上执行一些计算。在游标查询中，可以输入局部变量作为参数。图7-31展示参数化游标的基本元素。输入游标的SELECT语句的变量名称。在这个过程中，可以给这个变量赋值。该值可能来自于其他变量的计算，可能由用户输入，甚至可能是从另一个游标或查询中检索到的。当参数化游标打开时，当前值提交给了查询，以便于它只返回符合要求的行。使用游标的参数化查询提供了自动响应其他修改的动态计算数据的强大功能。

```
DECLARE cursor2 CURSOR FOR
SELECT ItemID, Description, Price
FROM Inventory
WHERE Price < :maxPrice;
maxPrice Currency;
BEGIN
  maxPrice = ... --from user or other query
  OPEN cursor2; --runs query with current value
  Loop:
    --Do something with the rows retrieved
  Until end of rows
  CLOSE cursor2;
END
```

图7-31 参数化的游标查询。通过用户输入、计算或其他查询，代码设置maxPrice值。当该游标打开时，值应用于SELECT语句，只返回匹配的行

7.10 Sally的宠物商店的存货清单

管理库存更新是商业数据库应用中一个棘手的问题。在很多情况下，员工需要知道现有某种商品的数量。员工可能查看商品重新排序，或者管理者可能想知道哪些商品库存过多、销售不畅。在数据库系统中决定数量有两种基本方式。第一种，可以写好程序，无论何时，只要有需要，就计算当前库存数量。程序可以查看每个商品的购买和销售，获得当前库存量。在大的应用程序中，这个过程可能很慢。第二种方法是，保持当前库存表的总数不断更新。每当发生商品买进或卖出时，该值必须更新。第二种方式可以快速返回总数，但其缺点是编程稍显复杂。注意，两种方式都需要调整机制对付已经不存在的库存项，用会计的婉转术语即“库存收缩”。

查看Sally的宠物商店的Merchandise表，如图7-32所示，注意到它包含QuantityOnHand列，

这里计划使用第二种计算库存的方式，并为每个商品项保存一个更新的总数。从根本上说，需要三组程序：一组管理商品购买，一组管理商品卖出，一组根据库存实际清点调整库存收缩。调整程序直截了当，但是必须保持用户界面简单易用。购买和销售的程序类似，所以这里只讨论商品销售。

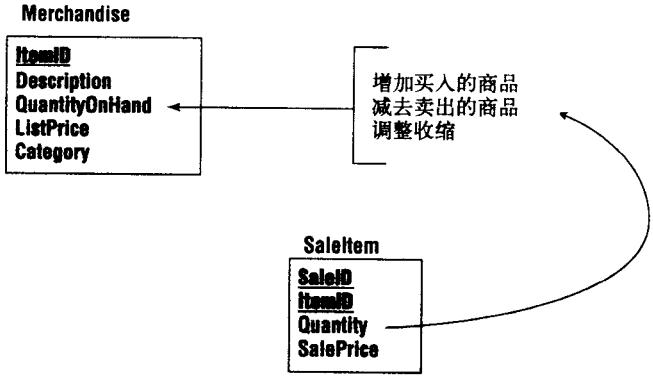


图7-32 处理库存变化。当商品卖出时，卖出的数量输入到SaleItem表。这个值必须从Merchandise表的QuantityOnHand上减去

当某项商品卖出时，以SaleID和ItemID为主码的SaleItem表增加新的一行。该行包括买入的商品数量，例如10罐狗食。当SaleItem表发生变化时，Merchandise表的总数必须调整。图7-33展示SaleItem表可能发生的四个基本变化。这些事件可能并不直接，所以考虑下面这些可能导致它们的商业行为。

1. 新的销售记录在SaleItem表中增加一行，所以QuantityOnHand必须减去卖出的数量。

2. 记事员的错误或顾客想法的改变都会导致销售记录或商品项的撤销，所以从SaleItem表中删去一行。任何已经从QuantityOnHand减去的数量必须重新加入总数。

3. 商品可能退回，或者记事员可能改正数量。数量的调整必须反映到QuantityOnHand总数。

4. 商品项可能存在输入错误，所以记录员修改ItemID。原始ItemID的QuantityOnHand必须重新保存，新ItemID的QuantityOnHand必须相应减少。



图7-33 SaleItem事件。受商业行为驱动，SaltItem表可能有四种情况发生。对于每个事件，Merchandise表的QuantityOnHand必须更新

使用数据库触发器，整个过程不太复杂。如果使用没有触发器的DBMS，必须把相应代码插入表单；这个过程不会很复杂，但是需要验证每个表单，保证其中含有必须的代码。

增加新的一行的第一种情况比较简单。图7-34展示了数据库触发器所需要的逻辑。只需要一个UPDATE语句：从Merchandise表的QuantityOnHand中减去新输入的Quantity。在检查或整理已有系统的代码时，应该确认这个事件已经正确处理了。问题是很多开发者忘记其他事件。

处理删除行的第二个事件不比插入行的代码复杂。图7-35展示了需要的新触发器。从SaleItem删除行表明该商品实际上没有卖出。因此，触发器通过把Quantity加回到Quantity

OnHand抵消销售的影响。

```
CREATE TRIGGER NewSaleItem
AFTER INSERT ON SaleItem
REFERENCING   NEW ROW AS newrow
FOR EACH ROW
  UPDATE Merchandise
  SET QuantityOnHand = QuantityOnHand - newrow.Quantity
  WHERE ItemID = newrow.ItemID;
```

图7-34　新销售触发器插入一个新行就会触发减去最新输入的quantity事件，该Quantity是从QuantityQnHand中卖出的数量

```
CREATE TRIGGER DeleteSaleItem
AFTER DELETE ON SaleItem
REFERENCING   OLD ROW AS oldrow
FOR EACH ROW
  UPDATE Merchandise
  SET QuantityOnHand = QuantityOnHand + oldrow.Quantity
  WHERE ItemID = oldrow.ItemID;
```

图7-35　删除行的触发器。该触发器通过把Quantity加回去抵消原来的减少

修改数据的情况更为复杂。需要考虑当Quantity值改变时意味着什么。假定某个商品项的QuantityOnHand开始为50份。然后，为10的Quantity插入SaleItem行。触发了插入触发器，减去10份，QuantityOnHand剩余40份。记事员现在把Quantity从10修改为8。因为少卖出了2份，所以QuantityOnHand需要调整。如图7-36所示，理解调整代码的最简单方式是认为把原来的10份加回去，然后从新的Quantity中减去8份。最终结果是QuantityOnHand为42份。

```
CREATE TRIGGER UpdateSaleItem
AFTER UPDATE ON SaleItem
REFERENCING   OLD ROW AS oldrow
              NEW ROW AS newrow
FOR EACH ROW
  UPDATE Merchandise
  SET QuantityOnHand = QuantityOnHand
    + oldrow.Quantity - newrow.Quantity
  WHERE ItemID = oldrow.ItemID;
```

图7-36　更新Quantity的触发器。如果Quantity改变，必须把原来的值加回去，再减去新值

第四种修改描述更为复杂。如果记事员修改ItemID值会发生什么？遇到的第一个复杂情况是数据库触发器可能没有独立修改每一列的事件。所以，必须把ItemID的修改与前面处理Quantity修改的代码结合在一起。此外，需要考虑特有的步骤。开始时ItemID为1的QuantityOnHand是50，然后输入出售10份。插入触发器把QuantityOnHand减少到40份。现在记事员把ItemID从1改为11。这意味着ItemID为1的商品实际上没有卖出，所以前述10份必须加回到QuantityOnHand。此外，ItemID为11的QuantityOnHand必须减去10份。如图7-37所示，这

个触发器需要两个独立的UPDATE语句。第一个语句的WHERE子句用oldrow.ItemID，而第二个用newrow.ItemID。还有，仔细观察两个SET语句。第一个加上oldRow.Quantity；第二个减去newRow.Quantity。这个差别为什么重要呢？第一，记事员可能把Quantity和ItemID一起修改，必须保证旧的Quantity用于旧的ItemID。第二，更重要的是，即使ItemID没有变化，这个触发器还可以处理简单的Quantity修改。假设ItemID设置为1并且没有改变。开始QuantityOnHand为50份，先卖出10份，剩下当前QuantityOnHand为40份。浏览代码，看看如果只有Quantity从10改为8份，它是如何工作的。第一，旧的Quantity（10）加回到QuantityOnHand。第二，减去新的Quantity（8），库存变为42份。这个过程和图7-36所示的一样，但是它用两步完成，而不是一步。

```
CREATE TRIGGER UpdateSaleItem
AFTER UPDATE ON SaleItem
REFERENCING   OLD ROW AS oldrow
              NEW ROW AS newrow
FOR EACH ROW
BEGIN
  UPDATE Merchandise
  SET QuantityOnHand = QuantityOnHand + oldRow.Quantity
  WHERE ItemID = oldrow.ItemID;

  UPDATE Merchandise
  SET QuantityOnHand = QuantityOnHand - newRow.Quantity
  WHERE ItemID = newrow.ItemID;
  COMMIT;
END
```

图7-37 最终的更新触发器。如果ItemID改变，必须重新保存原来商品项的总数，并且从新的ItemID的商品项上减去新的数量

现在，可以为购买表（OrderItem）编写同样的代码。逻辑相同。但是，这里的商业规则稍微复杂一些，只有当商品已经到货时才更新QuantityOnHand。如果决定这样做，主要的初始触发器不在OrderItem INSERT事件上，而是在MerchandiseOrder表的UPDATE事件上。让触发器寻找ReceiveDate列的输入，然后更新QuantityOnHand。

小结

虽然SQL命令功能强大，但是，有时候需要过程语言精确控制更新，或连接其他驱动程序或应用程序。根据DBMS的情况，过程代码可以位于模块、表单或外部应用程序。数据库触发器是过程代码的重要应用。这些过程被诸如插入、更新或删除数据等数据库事件触发，或作为响应执行。触发器可以用于增强复杂条件或执行商业规则。例如，附加到Inventory表的QuantityOnHand的触发器可以在值低于某个水平时自动通知供应商。如果一张表的一个修改引起触发器执行，又导致其他表的改变，还可能触发更多的事件，这就是级联触发器。多级触发器难于调试，而且耗费大量的服务器资源。

在大部分商业操作中，事务是必不可少的应用。它们表示一组必须同时成功或同时失败的修改。设置事务的开始点和结束点是应用开发中保护数据完整性的重要步骤。同时有多个用

户试图修改同一个数据所产生的并发访问是数据库完整性的另一个严重威胁。悲观锁经常用串行化的方法保护数据，保证同一时刻只有一个事务可以看见数据。但是，大量的锁消耗资源，而且可能导致死锁问题。乐观锁假设冲突很少，但是必须加入代码管理发生冲突的情况。缩写ACID（原子性、一致性、隔离性和持久性）有助于理解DBMS为保护事务的完整性而必须要保证的主要特征。

在很多关系数据库中，码生成是一个重要步骤，因为很难相信人们可以创建具有惟一性的标识。有两种常见的码生成方法：（1）当一行加入表时，自动创建；（2）根据需要提供独立的函数生成码。两种方法都有复杂性。自动生成码难以获取并在另外的表中使用。码生成函数要求程序员为每张表和每个插入过程编写代码。

数据库游标提供了一种用程序检索查询中多行数据并且一次一行逐行遍历的方法。游标指向的某个当前行可以由程序察看、修改或删除。可滚动的游标允许向前或向后移动，但是，只要有可能，应该尽量沿一个方向移动。程序可以使用可更新的游标修改或删除当前行的数据。程序可以使用带参数的查询动态选择符合其他条件的行。

开发漫谈

Miranda认识到即便好的DBMS也经常需要编程，处理一些复杂的问题。在应用开发中，应该检查所有的商业过程，确认事务元素。还有，确保UPDATE和DELETE过程可以处理并发问题。记住专业的应用应该预见并恰当地处理错误。编写数据触发器或模块代码自动完成基本过程，进行必要的计算。如果需要进行复杂计算，那么另外编写基于游标的程序。

关键词

原子性　　持久性　　悲观锁
级联触发器　　隔离性　　过程语言
并发访问　　隔离等级　　串行化
一致性　　乐观锁　　事务
数据库游标　　持久存储模块（PSM）　　触发器
死锁

复习题

1. 为什么有了SQL还需要过程语言？
2. 数据触发器的目的是什么？
3. 描述主要数据事件的顺序。
4. 表单事件的目的是什么？
5. 某些主要表单事件是什么？
6. 什么是事务？为什么必须由开发者定义事务？
7. 如何开始和结束一个事务？

8. 悲观锁和乐观锁有哪些区别?
9. 为了处理乐观锁的冲突需要加入什么代码?
10. 什么是事务的ACID特征?
11. 码生成的常用方法有哪些?
12. 如何获得所用的DBMS最近生成的码?
13. 什么是数据库游标? 它为什么重要?
14. 使用数据库游标修改数据的程序逻辑是什么?

练习

1. 写一个提高价格的函数，以参数形式接受价格输入。如果价格低于10美元，增加10%并返回。如果价格在10美元和100美元之间，增加5%。高于100美元的增加2%。用查询测试该函数。
2. 创建表，列出商品种类和该种商品的税率。例如，食物（0%）、服装（3%）、娱乐（10%）。写一个以种类和价格作为参数的函数，计算并返回应收的税款。
3. 创建一个数据触发器，当雇员工资改变时把一行写入新的表中。保存日期的变化、雇员、原来的工资和新的工资。
4. 创建一个数据触发器，禁止雇员工资涨幅超过50%。
5. 创建一个数据触发器（如果不能获得触发器，也可以创建表单代码），当商品卖出时，调整当前库存数量。需要SaleItem和Item表。
6. 创建传统银行账户表（AccountID、CustomerID, TransactionDate, AccountType, Amount)。为同一个人添加一个支票账户和一个储蓄账户。编写一个事务程序，安全地把一笔资金从储蓄账户转移到支票账户。
7. 创建一张小表，并且建立表单编辑该表的数据。编写程序修改表中的值。设置表单并编写程序，正确处理表的悲观锁。
8. 创建一张小表，并且建立表单编辑该表的数据。编写程序修改表中的值。设置表单并编写程序，正确处理表的乐观锁。
9. 创建一张小表，使用生成的码（例如，SaleOrder)。创建另一张表，包括该列作为外码（例如，OrderItem)。编写程序实现第一张表输入新的一行，然后用生成的码在第二张表中增加一行。
10. 创建一张小表，列出每个月的销售，其中包括一列PercentChange。编写一个基于游标的程序，在表中循环，计算和前一个月相比的变化率，并把该值保存在当前行。

下面的五个练习用一个数据库样例，创建下面的表：

```
Customer(CustomerID, LastName, FirstName, Phone, City,
AccountBalance)
Payment(PaymentID, CustomerID, DateReceived, Amount)
Item(ItemID, Description, ListPrice, QOH)
Sale(SaleID, CustomerID, SaleDate)
SaleItem(SaleID, ItemID, Quantity, SalePrice)
Shipping(City, Shipping)
```

11. 编写程序获得所有的销售和付款，计算当前每个顾客的AccountBalance，然后把这个数据插入Customer表。
12. 假设所有的商品初始库存为200份。编写程序根据总销售计算并保存每次销售后实际的库存数量。
13. 编写程序计算星期一和星期二的销售差别。
14. 根据下面的百分比编写程序更新Shipping表：

    麦迪逊 2%

    纽约 3%

    旧金山 4%

    芝加哥 2%

    其他 5%
15. 创建表单，允许用户从Shipping表中选择城市，输入百分数修改运输费用，然后点击按钮保存Shipping表的更改。

Sally的宠物商店

16. 创建表单，允许管理者指定当前数量比例和价格下降率。编写程序检查每件商品的销售（全年）。如果当前数量除以该商品的总销售小于指定的比例，按照价格比例下调其订价。
17. 创建表单，通过文本框接受用户输入动物种类和价格上涨率。当用户点击按钮时，用指定的比例更新所有给定种类的动物订价。
18. 使用适当的查询，编写程序计算每种动物按月销售收入的变化。例如，用户希望知道和一月份相比，二月份猫的销售变化率。编写程序计算这种变化，当用户点击按钮时，把结果保存在新的（临时）表中。然后显示结果图表。
19. 编写程序，当购买商品时，尤其是接收日期已经确定的情况下，增加当前数量。一定要使用事务处理，因为当前数量还可能被销售影响。
20. 宠物商店考虑购买扫描器用于结账。这些扫描器获得每件扫描商品的ItemID。假设当扫描商品时，这个数据会触发事件。编写这个事件调用的数据。该函数应该创建新的销售记录，并保存卖出商品的数据。可以通过创建表单模拟扫描触发器，该表单上包含选择ItemID的控件和激活触发器的按钮。

Rolling Thunder自行车

21. 解释Rolling Thunder订单如何工作。特别地，为什么零件选择需要代码？概述用于管理零件的代码逻辑（不要简单地复制代码，用几句话描述每个主要部分）。
22. 描述Rolling Thunder系统如何估计非标准型号自行车（例如，18.5英寸山地自行车）的常见规格（顶管长度、链条等）。
23. 有的州不对运输费征收销售税，而有的则征收。修改StateTaxRate表管理这个问题，并输入样本数据。写一个以自行车价格、运输费和州为参数的函数。该函数返回应收税款的计算结果。
24. 编写适当的代码，当收到新的购货时增加库存零件的数量。一定要管理数值的修改，控制并发修改，因为数据还可能在销售中改变。
25. 创建查询，按月计算每种类型的销售。创建临时表保存该数据和变化率。编写程序执行查

询，把数据放在表中。然后用基于游标的程序计算销售变化率。

26. 编写基于事务的过程，增加一个接收新付款并更新Customer表的BalanceDue的顾客事务。
27. 创建表单，允许管理者指定库存数量值、年份和价格变化率。编写程序降低选中年份之前购买的，库存数量大于指定水平的零件的价格。
28. 编写程序增加某个欠款余额的顾客账户的利息费用。注意处理并发/锁问题。
29. 编写程序，当库存数量低于某个水平时，自动生成新的购买订单。在Component表中增加ReorderPoint列，并输入样本数据。

参考网站

网站	描述
http://www.acm.org/sigplan/	美国计算机学会——程序语言（高级）特别兴趣组
http://support.microsoft.com/support/kb/articles/q115/9/86.asp	避免常见的数据库编程错误

补充读物

Baralis, E., and J. Widom. "An Algebraic Approach to Static Analysis of Active Database Rules." *ACM Transactions on Database Systems (TODS)* 25, no. 3 (September 2000), pp. 269–332. [Issues in database triggers and sequencing, but plenty of algebra.]

Ben-Gan, I., and T. Moreau. *Advanced Transact-SQL for SQL Server 2000,* Berkeley: Apress, 2000. [Discussion and examples of advanced topics for SQL Server.]

Feuerstein, S., B. Pribyl, and D. Russell. *Oracle Pl/SQL Programming,* 2nd ed. Cambridge, MA: O'Reilly & Associates, 1997. [Reference book on programming for Oracle.]

Gray, J., and A. Reuter. *Transaction Processing: Concepts and Techniques.* San Francisco: Morgan Kaufmann Publishers, 1993. [A classic reference on all aspects of transaction processing.]

ISO/IEC 14834. *Information Technology—Distributed Transaction Processing—The XA Specification,* 1996. [A discussion of the common method of handling transactions across multiple systems.]

Sanders, R., and J. Perna. *DB2 Universal Database SQL Developer's Guide.* Burr Ridge, IL: McGraw-Hill, 1999. [Using embedded SQL with IBM's DB2 database.]

Sceppa, D. *Microsoft ADO.NET (Core Reference).* Seattle: Microsoft Press, 2002. [Complete reference on using databases in the .NET framework.]

Sunderic, D., and T. Woodhead. *SQL Server 2000 Stored Procedure Programming.* Berkeley: Osborne McGraw-Hill, 2000. [One of many references providing an introduction to SQL Server programming.]

第 8 章

数据仓库和数据挖掘

本章学习内容

- OLTP系统和OLAP系统有什么区别?
- 什么是数据仓库?
- OLAP查询和传统查询为什么不同，怎样不同?
- 什么是数据挖掘?

8.1 开发漫谈

Miranda：快点儿，快点儿。加油，再快点儿!

Ariel：什么？你在训练马拉松?

Miranda：不是。是他们让我写的这些查询，好像永远也运行不完。我用少量数据测试时还好好的。可现在呢，真搞不懂。

Ariel：或许你只是需要一台更快的计算机。

Miranda：不，我觉得是需要一个不同的系统。这些查询正在获取数据，但这些数据来自很多不同的表。并且那些经理想要所有这些奇怪的小计。

Ariel：哇！总计有很多啊。你怎能让人去读那些东西？我想我在一张纸上看到了四张不同级别的总计!

Miranda：是啊，那只是经理们想要的一部分。我很高兴他们使用这个系统，但我不认为这些报告对他们有什么意义。我想可能需要一个独立的系统来重新组织数据并给经理们生成这些报表。那时他们会想再做一些统计分析。

8.2 简介

关系数据库系统的设计目标是有效存储大量数据，特别是它们可以快速存储和提取事务数据。看Sale和SaleItem表，你会发现数据存放得很紧凑。例如，宠物商店的SaleItem表只有4列，并且都含有简单数字。一笔单独的销售可以快速记录或提取。但是，这种结构却对其他类型的查询带来问题。那些涉及多个表的查询会用JOIN这个可能需要DBMS从数百万行记录中匹配数据值的操作。让DBMS分析数据，通过对若干个不同的因素（如雇员、地域、产品分类和月份）计算部分和。即使在速度很快的硬件上，计算涉及许多因素、多张表及数百万行数据时也可能造成性能问题。数据库系统的卖家试图通过创建表的索引来解决其中一些问题。索

引可以充分提高DBMS在一个表中寻找单独几行的速度，尤其提高了连接运算的性能。尽管如此，还是存在一个折中：给一个表添加索引加速了查询，但是降低了数据更新的速度，因为必须不断地重建索引。这种冲突已经导致了人们集中利用现有的关系数据库系统做联机事务处理（OLTP），而采用含有不同存储和提取系统的新系统做联机分析处理（OLAP）。从OLTP中提取并过滤数据，然后放到数据仓库中。数据仓库是密集索引的，并为提取和分析做了优化。还有一些附加的过程或例程可以用来分析数据、支持经理们的交互式探索以及统计性的查找有意义的关联信息。由于OLAP技术还在发展中，所以几乎没什么标准约束，并且有大量的工具可用。

8.3 索引

尽管报表常常看作行和列的简单排列，可是DBMS却不能简单地把所有数据都存放到顺序文件中。顺序文件的查找时间太长，并需要大量的操作才能给数据中插入新行。观察图8-1中关于雇员的一个简短表格，考虑要找到EmployeeID是7的那一行需要的步骤。DBMS必须顺序地读取每一行并检查ID，直到找到一个合适的匹配。在这个情况下，需要读取7行。平均情况下，如果总共有*N*行，则找到一个匹配大约需要查找*N*/2行。如果有一百万行，一个典型的查找将需要读取500 000行！显然，这种方法对大数据集是行不通的。如果需要保持列表有序，则给数据中插入新行的情况就更加糟糕。系统需要先读取数据的每一行直到给新行找到一个位置，然后再继续读剩下的每一行，把它们复制到下一行去。而删除事实上是比较容易的，因为DBMS并不真移除数据。它只是简单地把一行标记为已删除。随后，数据库可以重新组织或者压缩，移去这些标记的空间。

ID	LastName	FirstName	DateHired
1	Reeves	Keith	1/29/2004
2	Gibson	Bill	3/31/2004
3	Reasoner	Katy	2/17/2004
4	Hopkins	Alan	2/8/2004
5	James	Leisha	1/6/2004
6	Eaton	Anissa	8/23/2004
7	Farris	Dustin	3/28/2004
8	Carpenter	Carlos	12/29/2004
9	O'Connor	Jessica	7/23/2004
10	Shields	Howard	7/13/2004

图8-1 在顺序表中找到某一项。即使知道主码值，系统还是必须从第一行开始，直到找到想要的匹配。平均情况下，如果总共有*N*行，则找到一个特定项需要*N*/2行数据读取

8.3.1 二分查找

观察图8-1中的数据，显然DBMS没有利用到所有的信息。特别地，如果数据行是有序的，则有一个快得多的查找方法。图8-2展示怎么利用这个有序信息。考虑一下查找一个纸质的词

典或者电话本的过程。不是从第一页开始看每一项，而是，先翻到书中间，然后根据你在中间找的名字决定查找书的前一半还是后一半。找到的中间项是Goetz，你就知道Jones在数据的后一半。通过一次查找，你的查找范围立即缩小为原来的二分之一。继续同样的过程，每次把余下的部分分成两半，然后只查找合适的部分。在这个例子中，只需要4次尝试就可以找到Jones。这种二分查找过程持续地把剩下的部分分成两半，直到找到想要的行为止。一般地，如果总共有N行，二分查找将最多需要$m = \log_2(N)$次尝试就可以找到需要的行。理解这个公式的另一种想法就是，每次把列表砍去一半，所以需要砍m次（$2^m = N$）。现在考虑一个一百万行的数据，想找到其中一项最多需要读多少行？这里，m的值是20，显然要比平均500 000次的顺序方法要好得多。

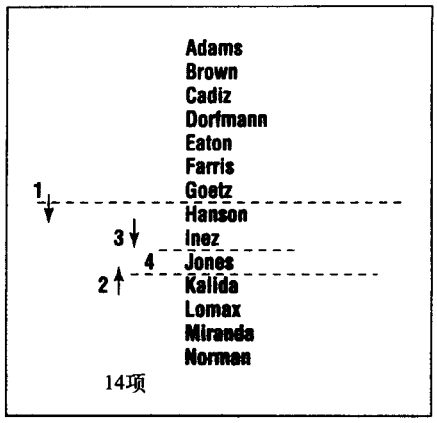


图8-2　二分查找。为了找到Jones这一项，把列表分成两份。Jones落在值（Goetz）的后面，所以把第二部分再分成两部分。Jones又落在了Kalida的前头。继续把剩下的部分分成两半，直到找到匹配的行

8.3.2　指针和索引

当要查找的数据表有序时，二分查找是一个相对不错的方法，所以用主码查找就很说得过去，这在表的连接中很常见。但如果主码是一个数值的CustomerID（客户ID），而想按LastName（姓）来查找，情况又如何？表又怎能按照多种方式排序？答案就在索引和指针中。

数据实际上并不存放在表里。通常数据都是分块存储在特殊文件中。存储时，每一块（也许就是一行）放到一个地方，得到一个地址。这个地址就是能准确告诉操作系统数据在哪里存放的一个指针。这大概就和“指明从一个文件开始的偏移字节”的偏移量一样简单。图8-3表示可以利用被查找的列（ID或LastName）和地址指针来创建索引。这种索引是独立的，并且已被排序，所以可以快速访问。只要找到了合适的索引项，地址指针就会传给操作系统，那么对应的数据也就立即提取出来了。实际上，即使索引也不是顺序存储的。它们通常都会像B树那样成块存储。查找B树至少可以和二分查找一样快，并且插入和删除码值还变得相对简单了。可以在SQL中用CREATE INDEX命令创建索引。

8.3.3　位图索引和统计方法

一些数据库厂商为大的表提供高度压缩的位图索引。在位图索引中，每个数据码都编码成少量的“位”集合。整个索引的位图（二元）图像通常小到能够装进RAM中。高速的位操作用来比较和查找码值。因此，位图索引非常快。对于像次码那种包含大量重复数据的列，位图索引尤其有用。位图索引不应用于包含所有惟一值的列。例如，在一个典型的SaleItem（SaleID，ItemID，Quantity）表中，你可能会考虑为SaleID和ItemID列创建位图索引。但是你大概不会想着给Sale表的SaleID列创建位图索引。

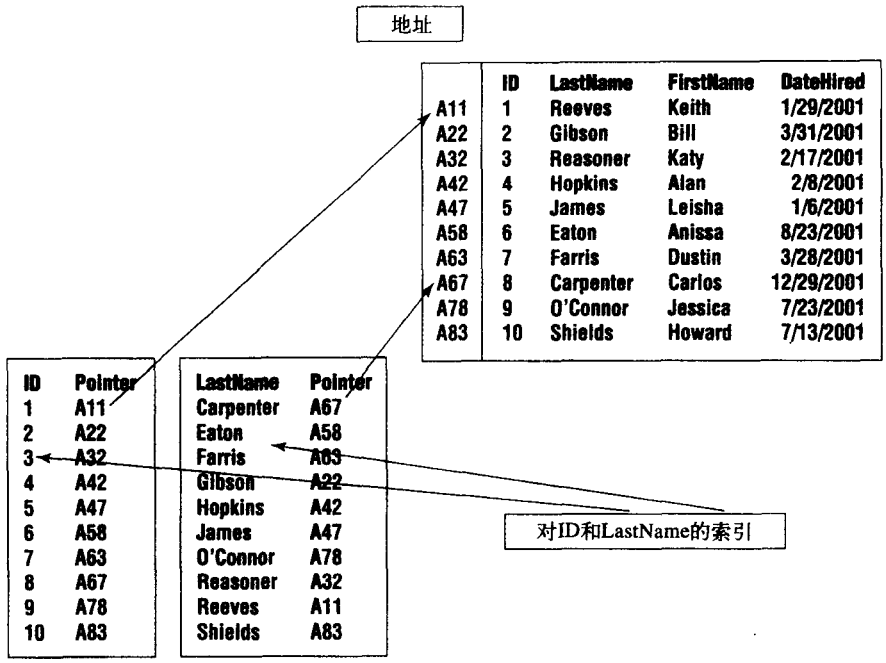


图8-3 指针和索引。每块数据都存到一个有着特定地址的位置。索引包括了要查找的列值以及指向这行其他部分的指针。可以给一个表创建多个索引，以便进行快速查找

一些数据库系统还提供更为复杂的索引方法。这些方法在索引列里检查数据，并存储数据结构方面的统计信息。例如，主码在一个主表中是惟一的。但是外码可能在一个表中出现多次。举个例子，CostomerID中值1173在Costomer表中只出现一次，但在Sale表中出现多次（每个Sale一次）。如果DBMS知道一个表中哪个值出现的最频繁，就可以使用这些统计信息更为有效地查找数据。如果能知道WHERE条件中的某一个条件只会产生一个表中有限的几个匹配，就可以先使用那个条件，再把结果和其他表做连接就简单得多、也快得多了。

8.3.4 索引的问题

考虑一个定义了10个索引的表。当一个新的数据行添加进表中时，每个索引都得改动。最少也需要数据库在10个索引中都插入新行。大部分情况都需要重新组织每个索引。这个问题是决定如何提高应用性能的核心问题。通过创建索引，显著提高查找数据表的能力。但对于创建的每一个索引，每次添加新数据或修改索引列时，应用程序都会变慢。所以，在哪些列创建索引是你的重要抉择。

抉择的第一步是只索引需要随机查找的列。最重要的列是在主要查询中参与表连接的列。可以确定应用中哪些查询最为常用。其次，索引那些JOIN条件中用到的列。然而，为了安全起见，同样需要确定那些要频繁改动的表。把那些表的索引尽可能多地移走。第三步，用大量的样本数据和繁重的查询来测试应用程序。找那些反应较慢的部分或表单。然后检查看有了索引是不是能提高访问速度。测试索引和整个应用看是否和其他部分接口良好。在应用的关键部分，只有当索引清晰、显著地提高了性能时才使用它。第四，或许也是最重要的步骤，

是找一个针对你使用的DBMS的性能分析工具。一个好的分析器可以监控使用，确定瓶颈，并且给出哪些列应该索引的建议。

8.4 数据仓库和联机分析处理

最终，索引的折中与平衡是难以确定的。为了实现复杂的查找，每张表需要很多索引。但是过多的索引又会降低事务处理的速度。考虑一个典型的企业，它在若干个不同的数据库（有时还有一些文件）中存储数据。显然，事务系统需要较高的优先级，否则商业活动将无法运转，但经理们却需要越来越复杂的数据分析。解决方法是：保留事务系统，并为经理们创建一个新的数据库来执行联机分析处理。

渐渐地，相比OLTP系统生成的传统报表，经理们需要更多的信息。他们希望能交互式地检查数据。他们并不总是知道该问哪些问题或者要找些什么，他们需要从多个不同角度快速观察数据。这类查询可能涉及海量的数据和多张表的JOIN运算。幸运的是，这种访问几乎总是只读的，只有极少的数据被修改，而只读查询比更新查询快许多倍。但是涉及几百张表和数百万行记录的数据连接还是需要时间的。而且，像索引那些加速查询的技巧又会减缓事务处理的速度。

8.4.1 数据仓库的目标

许多组织选择通过创建一个数据库的副本来避免这些冲突。*数据仓库*在一个致力于回答管理性查询的特殊数据库中持有一份事务数据的副本。数据可能从多个不同的源而来，但所有数据都被过滤，以保证其一致性并达到参照完整性约束的要求。每张表上建立多个索引，提高JOIN运算的性能。数据仓库还同时包括了特殊功能和查询控制，用以快速创建数据的不同视图。通常，每天一次或两次，数据由事务系统中转换过来，大批量存入数据仓库。另外，数据仓库可能是非规范化的，这样，用数据的重复来改善查询性能。

数据仓库的基本概念由图8-4给出。事务数据库持续地收集数据并生成基本报表，例如库存和销售报表。数据仓库表示数据的一个独立的集合。尽管可能使用同一个DBMS，但也需要新的表。例行的程序是，数据从事务数据库和其他文件中被抽取出来，然后通过检查以确保一致性。例如，所有的码值都必须匹配参照完整性。然后加到数据仓库中，而且通常并不以规范化的表来存储。相反，它是类似于星形模式的特殊结构。这些情况下，数据常常是重复的。比如，同样的城市和州名的结合在数据记录中会出现上千次。

联机分析处理通常和数据仓库相关，但从技术上讲，不使用中间媒介的数据仓库，可以在事务数据库上构建OLAP系统。OLAP是一个比较新的术语，具体包含哪些特性有着不同的说法。更要紧的是每个卖家都提供不同的技术和不同的实现。通常来讲，OLAP包含一系列的工具，令分析和比较数据库中的数据变得更加简单。

数据仓库一旦完成，就有许多查找它的工具。一些人可能使用基本的SQL查询并逐渐改善他的查询。其他人可能使用电子数据表提取和分析数据。更高级的方法使用数据挖掘技术。*数据挖掘*由一些在数据中查找隐含模式的自动工具组成。一些工具包括统计方法（例如回归分析、判定式分析），模式识别（例如神经网络）以及数据库分段（例如，k平均、混合建模、偏离分析）。这些工具通常需要充分的计算能力和极高速的数据提取。即使有当前的高速系统，

分析现存的一些较大的数据集时，这些技术中的许多都需要数天或数周。关键问题是，如果用户要做这种类型的分析，数据库就必须按他们的特殊要求进行配置和调整。

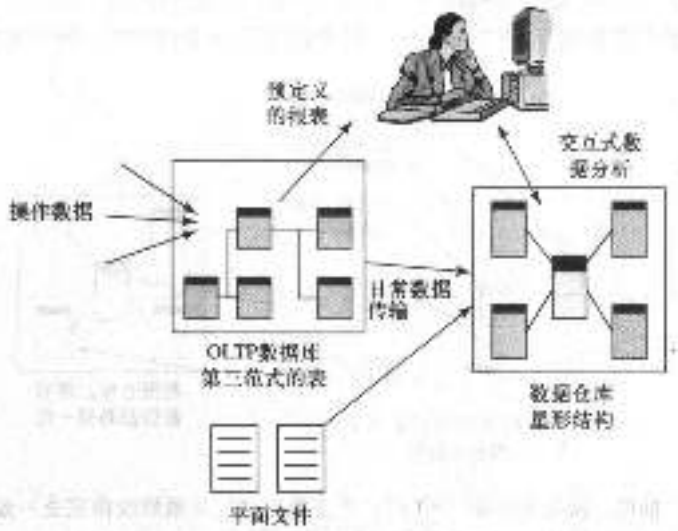


图8-4 数据仓库。例行程序是：从OLTP系统和其他源中得到数据，然后过滤，再转换到数据仓库中去。数据仓库为交互式数据分析做了优化

8.4.2 数据仓库的问题

尽管数据库管理系统在进步，计算机硬件也在不断发展，一些查询还是需要运行很长时间。很多公司都有一些数据是以不同的名字和格式存在不同的数据库中，甚至存在老文件中。数据仓库的目标就是创建一个系统以特定的间隔收集这些数据，进行清洗以确保其一致性，然后存在某一个场地。数据仓库的次要目标是提高OLAP查询的性能。大多数情况下，性能都可以通过数据非规范化得到提升。表连接常常是一个查询中最耗时的部分，所以创建新的数据结构提前做好所有这些连接，并把冗余的数据存储到更少的一些表中。

创建数据仓库有三个主要的挑战：(1) 建立一个收集和清洗数据的传输系统，(2) 当处理数百万或十亿行数据时，设计能够获得最佳查询性能的存储结构，和 (3) 创建数据分析工具，来统计性地分析数据。大多数公司都选择为第三步购买数据挖掘软件。极少数公司会让有经验的程序员编写详细的统计分析程序。一些公司还销售预先打好包的能够用来配置在数据中查找模式的工具。第二个问题——OLAP设计将在下一节讨论。

清洗和传输数据常常是建立一个数据仓库最困难的部分。图8-5表称作抽取、转化和传输(ETT) 的过程。你很快就会发现大多数公司都有许多不同的数据库，含有不同的表和列名以及相同类型数据的不同格式。例如，一个数据库可能有声明为20个字符的列Customers.LastName，而第二个数据库则使用设置为15个字符的Clients.LName。从这些数据源中抽取数据的过程需要尽可能地自动；如果试图手工来清洗数据就很难了，代价也太高。所以常常为从不同的源中合并数据编写复杂的查询。在这个小例子中，很可能导入一个表(例如Customers)，然后运行NOT IN查询来获得在Clients表中但不在Customers表中的名字列

表。这些新的名字就可能加入数据仓库。一些DBMS厂商创建帮助你自动进行数据对比的导入工具，但最终，大多数公司还是以自己写代码来处理这种复杂的过程而告终。这个过程的一个关键要素就是：不打断进行中的操作，从OLTP系统中抽取数据。这一步常常会用到利用了多处理器机器的并行处理过程的专用工具，但其细节取决于DBMS、硬件和数据库配置。

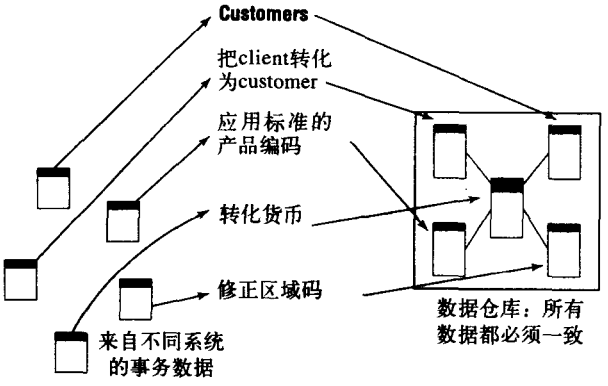


图8-5　抽取、转化和传输（ETT）。事务数据通常必须修改得完全一致。这个过程必须自动，以便不受影响地调度运行

有时我们还有一种方法来降低数据容量，那就是只抽取和传送自上一次传送以来改变了的数据。但是，这个过程需要OLTP系统跟踪所有更改操作的日期和时间。许多老系统都没法针对所有元素记录这些信息。例如，销售数据库必须记录一次销售的日期和时间，但很可能不会记录一个客户地址更改的日期和时间。

转化数据常常意味着替换空值，从文本转向数字，或在连接表中提取值并在基本表中更新值。所有这些操作都可以由SQL语句来处理，还可以创建能在日常事务中执行的模块，用来提取数据、清洗数据，并把它插入到新数据库中。

8.5　OLAP的概念

对OLAP的一个更具体的表述就是绝大部分工具都把数据描述成一个多维立方体。经理们使用不同的工具来检查数据的不同方面。为了展示这个过程，考虑宠物商店数据库的简单例子。经理们对商品销售很感兴趣。特别地，他们希望按日期、按项目的类别（猫、狗等等）、以及按客户的所在地来观察销售。他们想度量的属性就是销售出物品的价值或总量，也就是价格乘以数量。图8-6展示这种小型查询如何刻画成一个三维立方体。OLAP工具使经理们可以检查包含在三维立方体中的任何问题。例如，他们可以快速检查按州、城市、类别或月份统计的总量。他们可以看到每个州内不同种类的产品或细节的小计。这个工具还能提供可以刻画为一个立方体切片的细节项，或者提供任何部分的部分和。

最有用的OLAP工具提供帮助经理们从任何角度观察数据的交互能力。这些工具通常从总计开始，让经理们利用下钻得到细节。例如，一个看到年度总和的经理可以下钻，得到每个季度的值，然后每月，甚至每一天。与下钻相反的是把数据上卷到总和或者平均。与看到每个城市的详细销售不同，经理们可能希望看到整个州的总和。这些术语在图8-7中展示。

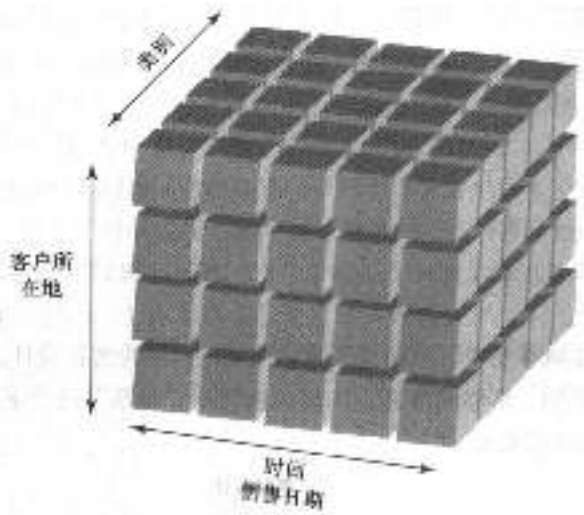


图8-6 有关销售的多维立方体。经理们对于不同维的组合很有兴趣，比如，询在上个季度的销售情况。OLAP工具能快速对涉及这个立方体任何方面的问题做出回答

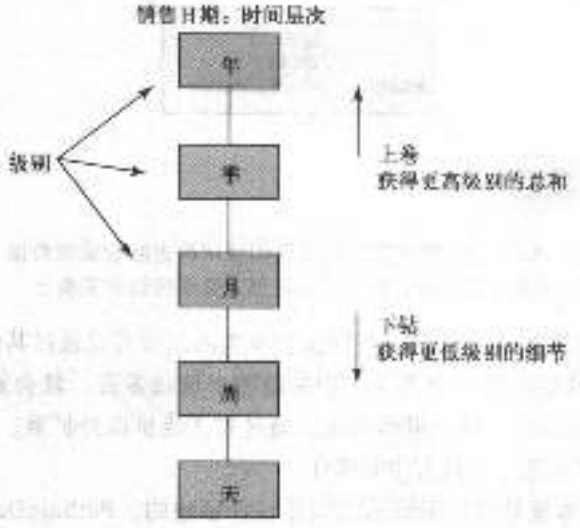


图8-7 下钻和上卷。对给定的维，下钻提供更多细节，上卷聚集子类别中的值

8.6 OLAP数据库设计

OLAP数据库设计与传统的数据库设计不同，一些概念是相似的，但最终，大部分OLAP工具在立方体结构中存储数据，而不是在关系表中。另外，OLAP设计对最终用户隐藏了表连接。管理者只能看到立方体。

如果你理解基本的查询，那么设计OLAP立方体就很简单。但是，需要学习一些新的术语，并理解立方体的基本思想。首先，OLAP立方体中展示的所有数据称作一个度量。度量是一些属性的数值量度，比如销售值或数量。它通常都是从一张表中选出来的详细信息（不带

GROUP BY的SELECT语句），尽管在一些情况下，可能希望创建新的查询来执行某些基本的计算，比如计数。度量来自事实表。事实表通常是数据库里的一个详细的表，宠物商店的例子使用OLAPItem查询当作事实表，它是基于SaleItem表的，后者包括了Quantity（数量）和每一项售出物品的SalePrice（售价）。Amount（总计）列是一个由Quantity * SalePrice计算出来的列，通常，事实表包含一块或者两块经理们想要检查的数据，然后还有一些列是外码。

第二步是选择构成立方体的边的属性。每个属性或边称作一个维。维是从其他表里选出来的属性。这些维表必须和事实表相关，通常，它们都是连接到事实表的外码上的，理论上讲，存在两种OLAP设计：（1）所有的维表都直接连在事实表上——叫做星形设计；或者（2）至少有一个维表在连接到事实表以前通过了第二张表——称为雪花设计。图8-8展示了一个简单的星形设计。如果添加了足够的维表，你就会发现叫“星形”这个名字的原因了。事实表居中，并通过射线与各维表相连。

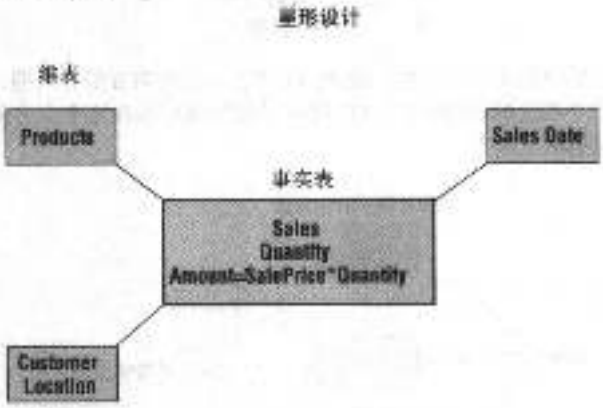


图8-8　星形OLAP设计。事实表保存经理们希望检查的数值型数据。维表保存特性。在星形设计中，所有的维表都直接连接到事实表上

雪花设计有一个更宽松的定义是，在连接到事实表以前可以通过其他一些表连接。图8-9展示了来自宠物商店数据库的一个例子。如果给图中放很多表，就会发现这个名字的原因。事实表居中，这和星形设计一样，但维表可以通过若干级别向外扩展。在星形设计中，所有的维表都直接连接到事实表上，没有中间媒介。

图8-6展示的立方体就是由这里展示的雪花设计创建的。PetSaleDate（宠物销售日期）、Location（地点）和Category（种类）就是用到的三个主维。数量和总计就是从事实表中拿出的量度，用于产生每个维的部分和。PetSaleDate事实上是在图8-7中显示日期的层次结构，这样经理们就可以在不同的级别上检查数据。同样，地点也可以定义成一个包括国家、州和城市的层次结构。

事实表是基于一个计算总计值的查询。如果使用原始的SaleItem表创建立方体，那会怎样？然后你可以把Quantity和SalePrice作为度量，创建一个计算的度量很是诱人：Amount2 = Quantity * SalePrice。但是，这种方法可能导致不正确的结果。理解这两种方法的不同是非常关键的。正确的做法是为每一个需要逐行计算创建查询（Price * Quantity就是一个常见的例子）。如果用OLAP设计立方体来计算度量，那么这个立方体就会（1）把数据切片，（2）对每个度量（Price和Quantity）分开做部分和，然后（3）执行计算：Sum（Price）* Sum（Quantity）。

所以这样的计算会在已经被加起来的数据上进行。图8-10用一个很小的例子展示了这种差异。当使用对事实表的查询计算乘积时，各列先乘再加，给出了正确的总和23美元。如果使用原始的表作为事实表并把立方体计算后的度量作为计算结果，立方体就会首先把数量和价格列加起来，然后执行乘法，给出错误的答案45美元。解决方案就是在查询里逐行计算，并且使用那个查询作为事实表。

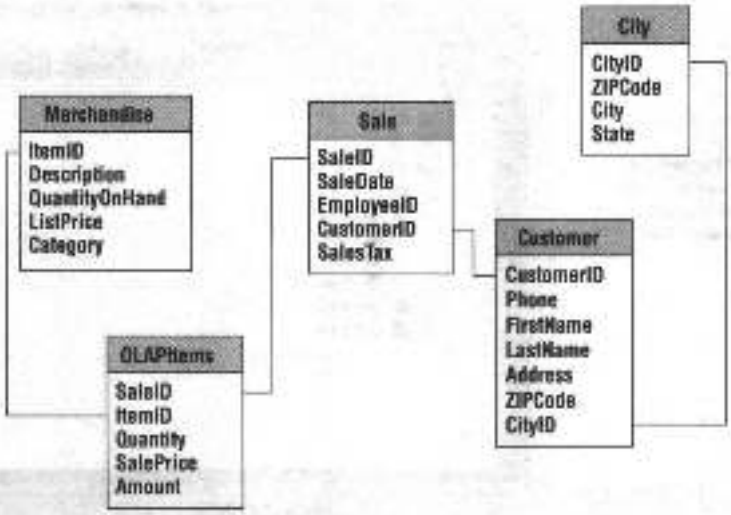


图8-9 雪花设计。它比星形设计宽松，维表在连接到事实表以前，可以连接其他维表

Quantity	Price	Quantity * Price
3	5.00	15.00
2	4.00	8.00
5	9.00	45.00 or 23.00

图8-10 计算顺序。在用于事实表的查询中，乘积运算是先做的，得到正确的总和23美元。在立方体的计算结果下计算，会导致先算总和，然后相乘，得到45美元这个不正确的值

雪花设计中你首先注意到的一件事就是，所做的表连接和执行一个传统查询一样多。一看，这种设计比起基于传统查询的系统并没有什么优势。并且，一些应用中，这种系统还可能使用原始的数据表进行分析。但是，OLAP系统通常都会添加一些项来提高性能，而且事实上数据并不存在独立的表中。例如，数据可能复制多份，比如每个订单都有重复的顾客信息。或者会建立一些内部指针来提供从事实表到维数据的快速连接。要知道那个立方体是设计来提取数据的，并不改变，所以就避免了使用INSERT和DELETE带来的数据不规范化问题。在一些系统中，你可以创建查询的快照来达到同样的效果。

8.6.1 OLAP数据分析

一旦立方体创建并处理，数据就可以拿来分析，并被最终用户浏览。如图8-11所示，大多数系统提供一些类型的立方体浏览器。浏览器可以让经理们简单快速地得到任何维的部分和，

或者执行过滤，只看某一维的部分元素。事实上，在设计立方体时就应该考虑到这个屏幕。



图8-11　OLAP立方体浏览器。在数据表中展示地点维，可以展开，或者点击方格查看部分和。其他两维（Category和SaleDate）在框顶部的下拉框中被选中。这里，时间维是展开的，图中显示的是选中第二季度。当数值改变时，新的总和将立即显示出来

经理可以通过双击相应的级别对数据格点所表示的行进行下钻。也可以选择特定的地点比较部分和。类似地，在下拉框中选择新值就提供了立方体中数据的一个不同的切片。即使是较大的数据集，结果也会立即显示。

在桌面上，微软提供了一个可以运行到Excel内部，甚至交互式Web页面里的强大的立方体浏览器。轴表就是多维立方体的一个交互式界面。轴表在用户的机器上创建，大多数人都会在微软的Excel内部创建轴表。这个工具有很多选项，为用户提供了许多灵活的用途。它可以连接到OLAP立方体和在各种系统中创建的数据库上。

图8-12展示了针对宠物商店例子的轴表报表。通过在一行或一列维上点击，经理们可以看到细节或部分和。他们也可以选择包含在部分和中特定的项。经理们甚至可以灵活地拖动这些维，把它们从列移到行，或者改变聚集的顺序。附加的一些选项提供了其他统计功能，比如平均值。强大的图形选项可以轻易支持通过Excel接口创建图表，这对大多数商务经理都很熟悉。



图8-12 微软的轴表报表。轴工具让经理们可以轻易地从任何角度检查立方体数据，选择数据的子集，执行计算以及生成图表

轴表工具非常强大，并且事实上离开了OLAP或者数据仓库也能使用。但是，和OLAP服务器结合起来，轴表可以运行得非常快。要在Excel中使用OLAP选项创建轴表，只需简单地选择菜单里的Data，然后再选PivotTable Report选项即可。一旦激活，下一步就是获得外部数据。选择OLAP立方体标签，再选择一个新的数据源。从这儿开始，选项就和传统的轴表一样了。把各个维拖到行或列的标题中，然后把事实表（总计）拖到表格中央。从这点开始，有很多选项可以用来自定义布局或是创建图表。选择OLAP立方体标签来从立方体中提取数据，而不是直接从数据库中提取，这一点很重要。把数据从数据库中抽出来意味着轴表已经完成所有数据提取和执行计算的任务了。做完了OLAP立方体就意味着立方体服务器已经完成了绝大部分工作，并且，客户端的轴表只是负责显示和操作数据。

8.6.2 SQL中的OLAP

如果考虑OLAP立方体的概念以及如何使用它，就会认识到那是检查多个GROUP BY语句结果的一种方法。因此，SQL 99标准在SQL里添加了一些计算基本的OLAP结果的特征。注意这些扩展并不改变数据库的结构，你仍需解决和索引相关的冲突。

在宠物商店例子中，如果在两列里使用GROUP BY语句会怎样？图8-13展示了宠物商店例子里在两列（动物类别和销售月份）包含了GROUP BY计算的查询的部分结果。它提供了每个类别元素每个月的部分和。假定所有的动物类型在所有的月份都有出售，你将看到12个针对鸟的值，12个对猫的，12个对狗的，等等。你没得到超聚集的总和，或者整个类别或跨越所有行的总和。例如，鸟这个商品全年的销售总值是多少？

```
SELECT Category, Month(SaleDate) AS Month,
  Sum(Quantity * SalePrice) AS Amount
FROM Sale INNER JOIN (Merchandise
  INNER JOIN SaleItem
ON Merchandise.ItemID = SaleItem.ItemID)
ON Sale.SaleID = SaleItem.SaleID
GROUP BY Category, Month(SaleDate);
```

Category	Month	Amount
Bird	1	$135.00
Bird	2	$45.00
Bird	3	$202.50
Bird	6	$67.50
Bird	7	$90.00
Bird	9	$67.50
Cat	1	$396.00
Cat	2	$113.85
Cat	3	$443.70
Cat	4	$2.25

图8-13 含有两个GROUP BY列的SELECT查询。可以得到每个动物类别每个月的部分和，却看不到跨越整个类别（全年鸟的销售）的合计，也得不到全体的总和

ROLLUP

可以通过使用额外的SELECT语句得到这些超聚集的总和。但是，SQL 99添加了专门计算超聚集总和的ROLLUP选项。图8-14展示了对宠物商店查询的结果。对每个Category列里面的元素都计算了跨越所有月份的总和。这个总和显示出来时，月份是空值。在底部，显示着带有两个空值的全部总和。当然，超聚集总和通常并不用黑体表示，所以可能很难发现。一个更大的问题是，如果一些月份有缺失值（空值）会怎样？在鸟销售有日期缺失的情况下，其显示就会包含两个类似的把鸟作为类别的行，一个针对月份的空值（缺失值）以及某个总计。一个值将是鸟这个产品在那个缺失的月份的销售总和。第二个将是所有日期的超聚集总和。但怎么知道哪个是哪个呢？如果细细观察这些总和数据，你就可能意识到较大的总和应该是超聚集总和的值。但对于其他函数，比如均值，就没有任何方法可以辨认了。

```
SELECT Category, Month ..., Sum ...
FROM ...
GROUP BY ROLLUP (Category, Month ...)
```

Category	Month	Amount
Bird	1	135.00
Bird	2	45.00
. . .		
Bird	**(null)**	**607.50**
Cat	1	396.00
Cat	2	113.85
. . .		
Cat	**(null)**	**1,293.30**
. . .		
(null)	**(null)**	**8,451.79**

图8-14 ROLLUP选项。给GROUP BY语句添加ROLLUP选项将产生超聚集总和。在这个例子里，该查询提供了每个类别元素的总和以及全部合起来的总和。注意，对应的月份是空值

为了标识超聚集的行，SQL标准引入了GROUPING函数。如图8-15所示，这个函数通常返回一个0值。当显示的行是超聚集的计算结果时，显示的值为1。在这个例子中，对于每个类别的产品，跨越月份的总和对GROUPING（Category）函数都会产生1这个值。全部的总和在两个指示列里都显示1值。这个函数也可以用来进行其他计算，甚至在WHERE条件中。例如，你可能希望针对超聚集总和再进行计算。

CUBE

看看结果，很显然ROLLUP选项并不提供经理想要看到的一切信息。例子中，超聚集总和只应用到Category列上。没有相应的针对Month列的总和，或者说没有针对一个给定月份的所有销售的总和。当然，如果你重写查询并且在GROUP BY子句中颠倒Category和Month列的顺序，你也能获得这些总和。

CUBE选项提供了解决方案。CUBE选项和ROLLUP类似，但是对所有的GROUP BY列都计算并显示超聚集。在图8-16中，注意SQL语句的惟一更改就是把ROLLUP关键字换成了CUBE。

结果仍然包含针对每个类别跨越月份的超聚集总和。这些总和在Gc指示列有一个1值。但是查询同时产生了对每个月跨越所有产品类别的超聚集总和。针对三个月的这些值就显示在结果的底部附近。注意Category下的空值，Gm列值为1表示了那是这个月的超聚集总和。

```
SELECT Category, Month ..., Sum ...,
  GROUPING (Category) AS Gc,
  GROUPING (Month) AS Gm
FROM ...
GROUP BY ROLLUP (Category, Month ...)
```

Category	Month	Amount	Gc	Gm
Bird	1	135.00	0	0
Bird	2	45.00	0	0
...				
Bird	(null)	32.00	0	0
Bird	**(null)**	**607.50**	**1**	**0**
Cat	1	396.00	0	0
Cat	2	113.85	0	0
...				
Cat	**(null)**	**1,293.30**	**1**	**0**
...				
(null)	**(null)**	**8,451.79**	**1**	**1**

图8-15 GROUPING函数。对选择的列参数，当显示超聚集行时，GROUPING函数就返回1

```
SELECT Category, Month, Sum, GROUPING (Category)
  AS Gc, GROUPING (Month) AS Gm
FROM ...
GROUP BY CUBE (Category, Month ...)
```

Category	Month	Amount	Gc	Gm
Bird	1	135.00	0	0
Bird	2	45.00	0	0
...				
Bird	(null)	32.00	0	0
Bird	**(null)**	**607.50**	**1**	**0**
Cat	1	396.00	0	0
Cat	2	113.85	0	0
...				
Cat	**(null)**	**1,293.30**	**1**	**0**
(null)	**1**	**1,358.8**	**0**	**1**
(null)	**2**	**1,508.94**	**0**	**1**
(null)	**3**	**2,362.68**	**0**	**1**
...				
(null)	**(null)**	**8,451.79**	**1**	**1**

图8-16 CUBE选项。CUBE选项对GROUP BY语句中的所有列都计算超聚集总和。在底部附近的Gm指示列值为1的行就是各月对所有类别产品的总和

由于这些额外的总和，你很可能使用CUBE比ROLLUP多。但是，如果你给GROUP BY语句再加一些列，你就会得到很多的部分和，以致于你可能希望使用ROLLUP来简化显示。

SQL标准提供了额外的选项，包括在联合多个列的值时创建基于CUBE或ROLLUP的查询。你也可以使用GROUPING SETS隐藏详细的部分和，只显示超聚集总和。如图8-17所示，SQL很容易懂。GROUPING SETS让你可以对每一列指定你希望看到的超聚集。这个例子里，请求3个独立的计算：(1) 跨越月份的类别总和，(2) 跨越类别的月份总和，以及 (3) 圆括号里指定的重要总和。查询的结果是没有显示细节的部分和，而只显示了请求的超聚集总和。

尽管ROLLUP和CUBE选项给SQL带来了新的特性，但其结果很难阅读。在OLAP值这方面，你可能不愿意把这种结果拿给经理们，或者希望他们能够交互地使用这些工具。另一方面，当需要进行更高级的计算而把数据转入到你写的过程中，或把数据转换到电子表格中时，这些选项（ROLLUP等）可能很有用。

```
SELECT Category, Month, Sum
FROM ...
GROUP BY GROUPING SETS
( ROLLUP (Category),
  ROLLUP (Month),
  ()
)
```

Category	Month	Amount
Bird	(null)	607.50
Cat	(null)	1,293.30
...		
(null)	1	1,358.8
(null)	2	1,508.94
(null)	3	2,362.68
...		
(null)	(null)	8,451.79

图8-17　GROUPING SETS选项。隐藏详细的部分和而只显示超聚集总和是可以的。这个例子请求3个总和：按Category，按Month以及空括号里面的全部

8.6.3　SQL分析函数

SQL-99为常见的OLAP分析添加了一些有用的数学函数。例如，标准差的统计函数（STDDEV_POP和STDDEV_SAMP）、方差（VAR_POP和VAR_SAMP）、协方差（COVAR_POP和COVAR_SAMP）、相关系数（CORR）以及线性回归（REGR_SLOPE等）都是标准的一部分。由于大多数数据库系统已经有了这些函数的自主版本，这个冲击也就不那么明显了，但它会帮助厂商采用这些函数的标准名字。

最有意思的是，标准还定义了若干种可以自动生成序号的函数。考虑一个想要在部门内部比较各雇员的销售情况的经理（或者想列出某个体育锦标赛的结果）。图8-18显示了基本的SELECT语句。排序函数是作用在一个范围的值上的，并且每个范围都得排序。同时也要注意RANK和DENSE_RANK函数的区别。两种情况下，相等的值都会得到同样的序号。差异就在于，在相等值后面从哪个值重新开始。RANK函数跳过那些将被分配的值（例子中的第三个序号）。DENSE_RANK函数并不跳过那些值并在下一次有效的序号分配中再次使用它。标准还提供PERCENT_RANK和CUME_DIST函数来计算每个值的百分比和累计百分比的序号。同样有一个ROW_NUMBER函数对查询中每一个显示的行返回标记了数字行。

8.6.4　SQL的OLAP窗口

SQL-99标准为OLAP定义了一些有用的扩展，使得SQL中某些类型的查询更加简单了。尽管大部分数据库还没有实现这个标准，但公司们以后会慢慢添加这些特性的。有两个元素很

有用：(1) 窗口划分和 (2) 针对以前行的计算。

```
SELECT Employee, SalesValue
RANK() OVER (ORDER BY SalesValue DESC) AS rank
DENSE_RANK() OVER (ORDER BY SalesValue DESC) AS dense
FROM Sales
ORDER BY SalesValue DESC, Employee;
```

Employee	SalesValue	Rank	Dense
Jones	18,000	1	1
Smith	16,000	2	2
Black	16,000	2	2
White	14,000	4	3

图8-18 RANK函数。序号函数的排列顺序是分别确定的。相等值获得相同的序号。RANK跳过那些原先将分配给相等值的序号。DENSE_RANK不跳过

计算总和时，划分和分组是相似的，都是指定一列，再把数据划分或分组。其不同在于使用GROUP BY时，你只得到每个组的聚集结果。而使用PARTITION时，你得到的是按选取的列组织的单独的行。图8-19展示这种差异。

```
SELECT Category, SaleMonth, MonthAmount,
 AVG(MonthAmount)
  OVER (PARTITION BY Category
  ORDER BY SaleMonth ASC ROWS 2 PRECEDING)
  AS MA
FROM qryOLAPSQL99
ORDER BY SaleMonth ASC;
```

Category	SaleMonth	MonthAmount	MA
Bird	200101	1,500.00	
Bird	200102	1,700.00	
Bird	200103	2,000.00	1,600.00
Bird	200104	2,500.00	1,850.00
...			
Cat	200101	4,000.00	
Cat	200102	5,000.00	
Cat	200103	6,000.00	4,500.00
Cat	200104	7,000.00	5,500.00
...			

```
SELECT Category, SaleMonth,
  AVG(MonthAmount) AS Average
FROM qryOLAPSQL99
ORDER BY SaleMonth ASC;
```

Category	SaleMonth	Average
Bird	200101	1,925.00
Cat	200101	5,500.00
...		

图8-19 SQL-99 OLAP PARTITION与GROUP BY的对比。窗口的PARTITION语句可以和细节行一起显示聚集数据（均值）。GROUP BY语句只提供聚集数据。同时，PRECEDING语句在划分中计算跨越以前行的数据

用PARTITION语句同样可以跨越选取的一些以前的行来计算聚集。这个例子需要当前行前面两行数据的动态均值。尽管DBMS可以自由使用任何方法来执行计算，我们想想如下步骤的划分过程。首先，提取出数据并且按Category和SaleMonth排序。其次，在每个类别内部，系统检查每一行并计算前面两行数据的均值。

厂商们实现SQL-99的所有这些新特性还需要时间。同时，你应该知道Oracle已经在别的环

境下使用PARTITION关键字多年了，所以要小心使用这个语法。在Oracle中，划分表示相应数据表的不同物理组织。

OVER语句支持指定行的变化范围。它用来执行和当前行相关的一些计算，所以可以向前或向后计算差值或均值。图8-20展示了一些RANGE函数的常用选项。整个查询计算3个总和值。第1个SUM命令计算查询从开始到当前行的总和值。第2个SUM函数做同样的事，但显式地表明了开始和结束行。第3个SUM函数从查询的当前行到最后一行计算总和值。在第2和第3个例子里，注意UNBOUNDED关键字指定开始或结束行的作用。如果只是希望计算前面或后面一些特定编号的行的总和，可以指定编号值来代替它们。

```
SELECT SaleDate, Value
SUM(Value) OVER (ORDER BY SaleDate) AS running_sum,
SUM(Value) OVER (ORDER BY SaleDate RANGE
  BETWEEN UNBOUNDED PRECEDING
  AND CURRENT ROW) AS running_sum2,
SUM (Value) OVER (ORDER BY SaleDate RANGE
  BETWEEN CURRENT ROW
  AND UNBOUNDED FOLLOWING) AS remaining_sum;
FROM ...
```

图8-20　OVER和RANGE函数。第1个SUM函数计算从开始到当前行的总和。第2个SUM函数更清楚地做同样的事。第3个SUM函数累加了从当前行一直到查询结束行的值

绝大部分数据库系统也提供LAG和LEAD函数的使用。这些函数设计成内联函数，用来引用前面或后面一些特定编号的行。例如，LAG函数让引用上一行的值变得简单。图8-21展示了其基本语法以及一个单周期LAG和单周期LEAD的结果。这些函数的强大之处在于其他一些计算中也可以轻易使用LAG或LEAD变量。这些函数并不是官方SQL标准的一部分，所以不同的厂商之间会有一些差异。比如，你或许并不能分配一个默认值，而这在最前（或最后）一些没有定义值的行中比较有用。由于大多数系统都支持这些函数，也因为它们确实有用，所以也值得学习。

```
LAG or LEAD: (Column, # rows, default)

SELECT SaleDate, Value,
  LAG (Value 1,0) OVER (ORDER BY SaleDate) AS prior_day
  LEAD (Value 1,0) OVER (ORDER BY SaleDate) AS next_day
FROM ...
ORDER BY SaleDate
```

SaleDate	Value	prior_day	next_day
1/1/2003	1,000	0	1,500
1/2/2003	1,500	1,000	2,000
1/3/2003	2,000	1,500	2,300
...			
1/31/2003	3,500	3,200	0

图8-21　LAG和LEAD函数。作为内联函数，它们可以很轻易从前一或后一行中返回值。你可以指定向前或向后走多少行

8.7 数据挖掘

数据挖掘的目标就是发现能用来得到更好决策的未知关系。图8-22展示和其他数据提取技术相比，数据挖掘是一个自底向上的方法。高度专业化的工具用来检查数据库，以寻找关系和其他有用的细小信息。这类工具相对比较自动化，可以只需极少指导就能发现模式。其他的需要通过模型构造器多进行一些输入和设定。这些技术中的大部分都是探测性的，因为你不是在验证一个预期值，而是在查找一些未知的关系。一些例程从统计分析而来；其他都很具体，从特定的任务而来。这一节提出更为流行的一些技术概要。详细的统计和编程问题不在这儿涉及，可以在专业教科书中找到。



图8-22 数据挖掘，其目标是识别未知关系。数据挖掘是一种自底向上的模式发现方法。由高度专业化的工具扫描数据，寻找可能有用的信息

图8-23列出了一些常见的数据挖掘技术。有时候，DBMS厂商会在基本的系统中包含一些这类技术。但是，大多数厂商都会把数据挖掘技术作为附加产品进行销售。还有从专门的数据挖掘公司而来的很多其他技术。不管哪种情况，通常都需要建模器工具，帮助构建合适的模型，并解释结果。数据分类和购物篮分析是商业活动中两种常见的数据分析方法，因为它们对许多类型的问题都很有用。地理系统对一些特定问题也是强大的解决方案。通过日志进行时间序列评估的网站分析也变得流行起来，可以评估超大型数据集的新技术和方法也在不断开发中。

- 分类/预测/回归
- 关联规则/购物篮分析
- 聚类
  - 数据点
  - 层次性
- 神经网络
- 偏差检测
- 序列分析
  - 时间序列事件
  - 网站分析
- 空间/地理分析
- 文本分析

图8-23 数据挖掘技术。分类和购物篮分析是商业活动中的流行技术。评估关系的新技术和方法仍在不断开发中

8.7.1 分类

如图8-24所示，许多商业问题都得益于分类分析。已经有很多工具开发出来用于评估可以预测结果的关系。像回归这样的统计方法已经可以直接使用了。尽管如此，统计方法有两个缺点，就是它们趋向于假设存在线性关系，并且这种评估是基于均值的。但最重要的隐含关系常常很细小，以至于很难由均值表示。例如，你可能在查找那些可能因鼓励而回头、并购买更多东西的新顾客。由于是新顾客，你多半并没有足够的数据去创建一个统计性的重要效果。

- 哪些借款人/贷款最可能成功?
- 哪些客户最可能想要一个新的商品?
- 哪些公司最可能进行破产?
- 哪些工人最可能在下面6个月内辞职?
- 哪些正在起步的公司最可能成功?
- 哪些纳税申报单是假的?

图8-24　分类的例子。许多常见的商业问题都能得益于分类分析。每个问题都有一个结果，且其目标就是基于一系列属性把元素分进结果中

可以用分类分析评估的问题都有一个受指示属性集影响的结果。基本目标就是估计每个指示变量效果的好处和它对结果的影响。例如，银行可能在诸如工作稳定性、信用历史和收入这些贷款人属性上有历史数据。以最终结果（是否付出贷款）为目的，数据挖掘系统可以评估这些变量的每一项对结果的影响。这些权重可以用于以后的顾客数据，来帮助确定是否同意贷款，或者影响收取的利率。

图8-25表示一个借贷情况下的小例子。数据可能是类别的（是/否）或连续的（比如收入），一些分类工具在两种类型的数据下都能工作；一些需要转化成类别数据。例如，收入数据可以被转化为二元的，比如低：0~30 000；中：30 000~70 000；高：70 000~120 000，以及富有：120 000以上。当然，你还面临着新的怎样画线来区分这些绝对数值的数据挖掘问题。一些工具同时提供了帮助做此决定的系统。

收入	婚否	信用历史	工作稳定性	成功
50000	Yes	Good	Good	Yes
25000	Yes	Bad	Bad	No
75000	No	Good	Good	No

图8-25　银行贷款分类。指示器属性以某种形式影响结果。数据挖掘软件在一个测试数据集上评估每种属性的分量。生成的结果模型可以用于对未来的数据，预测新贷款潜在的成功或失败概率

常见的分类工具包括各种回归方法、贝叶斯分析、决策树（特别对于层次性数据）、遗传算法以及神经网络。其中，神经网络在典型情况下需要最少的指导，而高级回归技术则依赖于有经验的建模师的技巧。任何分类分析的关键要素就是对现有和新的实例，模型需要预测得多准确。

8.7.2　关联规则/购物篮分析

购物篮分析为促使数据挖掘被人们接受立下了不可磨灭的功劳。最初，这种技术用于分析顾客在便利店的购物情况，因此得到购物篮这个术语。更通用的术语关联规则表示可以用于其他情况的方法学。这些系统回答的基本问题是：顾客最容易同时购买哪些商品？或者，从规则的角度讲，A的存在是否意味着B的存在？典型的情况是，便利店发现购买了尿布的顾客

常常同时也购买啤酒——特别是在星期四和星期五晚上。这个信息的重要性在于经理们可以利用它来增加销售。比如，可以考虑在商店里把这两样东西放得靠近一些，鼓励更多的顾客同时买走两样。同样，制造商也可以使用相同的知识通过提供优惠券或把相关产品打包描述来交叉销售。

购物篮分析需要一系列的事务数据，这些数据要包括一个人购买的所有产品的列表。今天，这种数据可以很轻易地从使用了条码扫描器的超级市场或大型连锁店获得。大多数公司都会把这些数据卖给一些专门的公司，他们再转卖给其他公司。分析软件扫描这些数据并比较每一项和其他项的关系，寻找是否有模式存在。在这个过程中，软件会计算出你用来评估这种关系或规则的3个值。当用于两个项时，其定义比较容易理解，但也适用于多个项的情况。规则的支持度是同时包含了两个项的事务的百分比度量。从统计上讲，这个概率由 $P(A \cap B)$ 表示，由同时包含了两个项的事务数目除以事务总数计算得来。可以对A和B单独计算类似的值，或每个单独的项被购买的次数的百分比。支持度高一些表示两样商品频繁地被同时购买——但这个数字并没有告诉我们是其中一个导致了另一个。而规则（A蕴含B）的置信度是含有A项的事务也含有B项的百分比度量。从统计的角度讲，它是假设A已经被选中，那么B也被选中的概率，记为 $P(B|A)$。根据统计定义，$P(B|A) = P(A \cap B)/P(A)$，所以计算起来相对更容易。同样，置信度高一些就指示了A项的购买导致了B项的购买。大多数数据挖掘工具所声称的第三个统计值是提升度。与没有规则的购买相比，提升度（Lift）是规则的一个潜在的增益属性。如果这个值比1大，增益就是正的。从概念上讲，它表示由关联而引起的销售增加。从统计的角度讲，它可以由 $P(A \cap B)/(P(A)*P(B))$ 或 $P(B|A)/P(B)$ 计算得到。

图8-26展示在尿布和啤酒例子里这些数字是怎么计算出来的。数字都是虚构的但代表了这种情况。注意提升度比1大了很多（1.714），表示这种关联有力地促进了啤酒的销售。数据挖掘软件对所有基本的项目对都计算所有这些值，如果有很多项，这个过程可能运行很长时间。同时，也可以在分析中考虑多个项：床单和枕头的购买是否导致毛巾的销售？尽管如此，把太多维结合起来会导致巨量的计算问题，所以大多数分析都在有限的比较集里进行。

支持度	$P(B \cap D) = .6$　$P(D) = .7$　$P(B) = .8$
置信度	$P(B\|D) = P(B \cap D)/P(D) = 0.857$
提升度	$P(B\|D)/P(B) = 1.714$

图8-26 评估购物篮关联。支持度是指一个事务同时含有两项的百分比。置信度是假定购买了尿布（D），而又购买了啤酒（B）的概率。提升度是指其结果对销售的贡献，且一般应该大于1

在购物篮分析中，很快就会碰到一些问题。首先，购买量较小的一些项可能造成误导的值。如果一项只购买了一两次，那么和它一起购买的所有东西都可以看作与它有关。因此，你必须检查数据，并改变分组以保证大部分项购买的频度差不多。图8-27展示了一个假设的情况。一个销售大量木材的五金店却只销售很少量的钉子和螺丝。为了防止虚假的规则，就需要把钉子和螺丝结合到一个更宽的五金类里，并把木材细分成更多的几个详细定义。如何知道还有没有问题呢？可以用一些额外的查询快速计算一下每个项的销售量。如果原始的计数很难读到，那么新的OLAP函数也可以轻松地计算出百分比来。

在购物篮分析中你可能遇到的其他问题包括你发现的一些规则对于那个行业的任何人来说

都是显而易见的，例如，快餐连锁店无疑会发现肉饼和煎饼相关。在系统返回不合理的或很难解释的规则时还会有一些较微的问题存在，例如，五金连锁店发现马桶圈的销售和新店的开张关联紧密。即使这种关联是真的，你怎么处理它呢？

项	频度
1" nails	2%
2" nails	1%
3" nails	1%
4" nails	2%
Lumber	50%

项	频度
Hardware	15%
Dim. lumber	20%
Plywood	15%
Finish lumber	15%

图8-27　平衡频度。很少购买的项将导致错误的规则。解决方法是定义项时要平衡。在这个例子里，把钉子结合到五金类别去，并把木材分成更小的类

8.7.3　聚类分析

聚类分析用来确定数据（趋向于彼此有关的数据点）分组。它可以用来确定人群的分组，比如把顾客归类。如果你知道顾客特定的组，就可以利用一些顾客的信息来帮你卖更多的东西给同组内的其他人。最有可能的是，同一组的顾客希望买的东西比较类似。例如，书店就可以使用购买哪些图书来把顾客分类，并找出类似顾客购买的书，再把它们推荐给其他买书人。同样，可以使用聚类分析来归类不同部门的雇员技能，然后在雇佣新工人时使用那些信息。

正如图8-28所显示的，聚类在二维情况下比较容易看清。软件的目标就是识别彼此接近（较小的类内距离），而同其他数据点比较远（较大的类间距离）的数据点。不幸的是，大多数数据库没法像这个例子一样有效地显式聚类。但聚类分析仍是很有用的数据探测技术，因为它可以发现用其他工具所不易看到的模式。尽管如此，始终要记得含大量观察结果（行）或多维的数据集极难聚类。就算用相对现代的计算机，评估大型、复杂的问题也会需要几小时到几天时间。所以要开始小心，要用较小的例子试着建立聚类，而且只使用有限的维数。



图8-28　聚类分析。其目标是找到数据点的分组，保证组内很接近但和其他组较远。多维的大型数据集评估起来非常困难，而且很耗时

8.7.4　地理分析

地理信息系统（GIS）显示涉及地点信息的数据。这些系统通常归类为可视化系统。它们可以帮助显示地理关系，给人们展示地点如何影响数据。几乎没有哪个系统真正具有扫描数据以发现模式的数据挖掘能力。虽然如此，它们还是数据分析的重要工具。一些关系当用地图来看时更容易理解，图8-29就展示了美国西部各州简单的销售情况。多用几种颜色还可以表示额外的数据，或者给每个州都放一张图表。

大型的DBMS厂商已经开始把空间和地理信息系统结合到他们的产品中去了。你也可以从

其他厂商那里购买单独的系统。除了绘制地图，真正的GIS系统还有很多方法在地图上显示数据。基本的技术包括阴影与覆盖。常用于按区域显示销售。覆盖在不同的层显示多个项，方便地看到各个项彼此之间以及和地点之间的关系。例如，市场商人可能会比较销售、收入以及不同地理区域的人群。



图8-29 地理分析。基本的地图展示了按州分类的销售情况。由图示可以看到，颜色越深，表示销量越多。如收入等额外数据可以交叠显示，或者用图表进行比较

除了软件，地理信息系统还需要两个重要的部分。首先，需要地图信息。通常，这个数据是随分析系统一起卖的，但有时详细的数据会作为附加的项单独出售。针对美国（以及欧洲大部分）的直至街道的高清晰数据已经出现了，但那是个很大的数据库。其次，需要对数据进行地理编码，并且多半还得购买已经进行了地理编码的人口统计数据。本质上，就是为你的数据存储某种类型的地理标记。最基本地，你大概已经知道国家和州，但你可能还希望添加区域码，或者城市码，甚至经纬度。如果所有的销售数据都来自独立的商店，就可以相对容易从地图或GPS系统中得到每家店的地理位置。随着手机的增长，将来可能还会出现一个有趣的选择。按照联邦应急条例（e-911），手机需要有位置系统。最终，这种信息也会提供给商界，所以事务系统也可以记录销售员的精确地点，甚至还有客户。请关注这些技术造成的隐私问题，但当创建新数据库时，应该考虑把地理编码信息结合到数据表中。一旦数据收集到了，GIS就可以很方便地显示其关系了。

小结

大型数据库是为事务处理而优化的，为了高效地处理日常操作，数据存储在规范化的数据表中。但是，大多数经理需要连接很多表以提取和理解数据。索引加快了数据连接和提取，却降低了事务的效率。这种分歧意味着，创建一个分离的数据仓库来分析数据常常是更好的办法。从事务系统中抽取数据并清洗，然后放到星形或雪花设计中，让经理们可以把焦点放到一个特定事实表周边的维上。

OLAP立方体是一个强大的工具，它让经理们可以快速地详审数据，并从不同的角度检查部分和。不需要写复杂的SQL查询，经理们就可以跨越产品类别、时间，甚至同时跨越多个维来比较值。OLAP立方体浏览器也包含一些简单方法，用来过滤数据到特定的行或立方体的一部分。

已经有许多统计性的数据挖掘工具被开发出来帮助经理们分析数据。它们常常需要训练和人们的专业知识，但能成为理解数据之间关系的强大工具。分类和聚类算法用来把数据划分成组。比较不同的组可能会更好地理解客户并进行市场扩张。关联或购物篮规则在卖很多不同商品的商店里很流行。找出同时购买的商品可能会帮助向其他顾客推荐物品，也可能促进对商店布局和顾客心理的理解。地理系统对于涉及地点的任何问题都很有用。有了专门的工具和人口统计数据，就可以用来查看其存在的地理关系了。

开发漫谈

Miranda发现即使有SQL，一些商业问题也很难回答。在经理们也不是很确定找些什么时，就需要考虑OLAP和数据挖掘方法了。提供OLAP立方体是一个好的开端，因为它可以让经理们很容易索引部分和，并按需要把数据切割到任何细则。更高级的统计性数据挖掘工具已经出现，但通常都需要额外的训练以及知识型用户。要记住由于性能原因，常常需要把OLAP数据移到一个单独的数据仓库中。

关键词

关联规则	二分查找	位图索引
分类分析	聚类分析	置信度
数据挖掘	数据仓库	维
下钻	抽取、转化和传输（ETT）	事实表
经纬度代码	地理信息系统（GIS）	提升度
购物篮分析	度量	联机分析处理
联机事务处理	维表	指针
上卷	雪花设计	星形设计
超聚集	支持度	

复习题

1. 在关系数据库中，为什么索引很重要？
2. 假定关系型DBMS很强大，公司为什么可能还需要数据仓库？
3. 建立数据仓库会碰到哪些主要问题？
4. OLAP查询和传统的SQL查询如何不同？
5. 什么是OLAP立方体？
6. 什么是分层的维，它们如何与上卷和下钻操作相关？
7. SQL（特别是新标准）中定义了哪些基本的分析函数？

8. 数据挖掘的目标是什么？
9. 数据挖掘工具主要分哪些类别？

练习

1. 至少找到两个商用的OLAP工具，并比较其特性。
2. 找一个商用的数据挖掘工具，并列出从典型DBMS中抽取、转换数据的步骤，让它可以被系统所用。
3. 找一个商用的数据挖掘工具，并列出其执行购物篮分析的步骤。

Sally的宠物商店

下面问题的大多数都需要一个OLAP立方体处理器。需要访问SQL、DBMS内部的OLAP浏览器或轴表。对于数据挖掘工具来说，如果没有专门的软件，可以使用Excel来做一些简单的分析。

4. 创建一个按时间、州、雇员和项目类别来浏览商品销售的立方体。
5. 创建一个按时间、类别、品种、性别和登记来浏览动物销售的立方体。
6. 创建一个按时间、州和类别来同时浏览动物和商品销售的立方体。
7. 创建一个基于时间、雇员和地点的浏览从供货商那里购买商品的立方体。在事实表里包括购买值、运费以及从订单到接收之间的延迟。
8. 动物卖出时的年龄和价格有什么关系？
9. 如果你可以访问购物篮分析软件，评估一下销售表，看看是否存在什么关联。
10. 作为一个简单的时间序列分析，按周抽取商品销售，把数据以图的方式展示，然后估计出趋势曲线。
11. 一些类别的项的购买是否比其他的大得多？比如，公司是否销售了足够多的狗？
12. 有没有购买力比其他人大的顾客类别？这是个常见的聚类问题：如果你不能访问专用的软件，可以用回归来测试各属性。
13. 使用产品的周销售表，预测后面3周的销售情况。
14. 有没有关于销售的地理模式存在？是不是一些州或者区域销售得更多？

Rolling Thunder 自行车

15. 创建一个按模型类型、州、时间和售货员来评估销售（价值和数量）的OLAP立方体。
16. 创建一个新的尺寸维，把车架维的尺寸降低到3个（小型、中型和大型）。把自行车按这个维分类。如果可以的话，使用聚类软件；否则，使用均值和标准差。记得公路自行车是用厘米来计量，而山地自行车用英寸。
17. 把练习16里减小了的尺寸维添加到练习15创建的OLAP立方体中。
18. 创建一个按定购日期（时间）、模型类型、月份和装配车架的雇员来评估制造用时的OLAP立方体。
19. 创建一个按时间、制造商、公路或山地车以及部件类别来评估购买组件的OLAP立方体。
20. 各城市的销售和它的人口之间有何关系？同时从自行车的数量和价值两方面来评估。
21. 自行车的尺寸（车架尺寸）和它的价格之间有何关系？
22. 使用按模型类型分类的月销售数据，预测下面6个月的销售情况。

23. 如果有购物篮分析软件，评估各部件的购买情况。看有无模式存在——在定义的组关系以外。
24. 创建一个按油漆类型、字母风格和模型类型来评估销售（数量）的OLAP立方体。
25. 忽略资金消耗但含进薪水，按月评估利润，并预测下6个月的值。

参考网站

网站	描述
http://otn.oracle.com/prlducts/warehouse/owb_calais_new_features/html/module4/04-0110.htm	Oracle 9i OLAP处理
http://www.microsoft.com/sql/evaluation/bi/default.asp	微软SQL Server 2000的分析工具
http://www-3.ibm.com/software/data db2/db2olap/features.html	IBM DB2 OLAP工具
http://www.wintercorp.com/rwintercolumns/SQL_99snewolapfunctions.html	SQL 99 OLAP标准和例子
http://www.datawarehousing.org	数据仓库信息

补充读物

Apte, C., B. Liu, E. Pednault, and P. Smyth. "Business Applications of Data Mining." *Communications of the ACM* 45, no. 8 (August 2002), pp. 49–53. [Some examples of data mining, also part of a special issue on data mining.]

Golfarelli, M., and S. Rizzi. "A Methodological Framework for Data Warehouse Design." *Proceedings of the First ACM International Workshop on Data Warehousing and OLAP*, 1998, ACM Press, pp. 3–9. [Relatively formal definition of facts, dimensions, and hierarchies.]

Han, J., and M. Kamber. *Data Mining: Concepts and Techniques*. San Francisco: Morgan Kaufmann/Academic Press, 2001. [A general introduction to data mining techniques.]

Hastie, T., R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. New York: Springer-Verlag, 2001. [A strong foundation book on the statistics and algorithms of data mining, including all of the math.]

Marakas, George M. *Modern Data Warehousing, Mining, and Visualization*. Upper Saddle River: Prentice Hall, 2003. [An introduction to core concepts in data mining that includes trial software.]

Peterson, T., J. Pinkelman, and B. Pfeiff. *Microsoft OLAP Unleashed*. Indianapolis: Sams/Macmillan, 1999. [Details on OLAP queries and data warehouses in SQL Server.]

Scott, J. "Warehousing over the Web." *Communications of the ACM* 41, no. 9 (September 1998), pp. 64–65. [Brief comments on Comcast using a Web interface for its data warehouse.]

第四部分

数据库管理

大型应用程序需要精心地呵护。绝大多数企业都雇用数据库管理员监控程序性能、访问安全性，并确保数据库完整性。第9章着重讲述数据管理员和数据库管理员的任务，特别是数据库安全。SQL又一次在管理和保护数据库方面扮演了重要的角色。

信息系统（IS）经理比以往更加关注提供场地无关的数据访问问题。网络和Internet提供了多种选择来分布数据和给出贯穿企业的解决方案。第10章将探讨分布式数据库以及在Internet上访问数据库的一些挑战和选择。

第 9 章

数据库管理与安全

本章学习内容

- 数据库应用必须执行哪些管理任务?
- 数据库备份会带来什么复杂性?
- 如何保护数据库使其避免物理灾难?
- 你应该关注什么样的安全威胁?
- 如何在多种安全威胁下保护数据库?

9.1 开发漫谈

Miranda：最后，好像一切都运行得挺好。

Ariel：是不是说你最终拿到薪水了?

Miranda：是啊，他们昨天给了我支票。他们甚至很喜欢我做的事，还给了我一份工作。

Ariel：好极了。你要接受吗?是什么工作?

Miranda：我想是的。他们想让我当数据库管理员。他们说需要我维护这些数据库的正常运行。他们还暗示说想让我帮他们现有的程序员们学习搭建数据库应用。

Ariel：哇！那意味着你将比那些程序员拿更多的钱。

Miranda：大概是吧。不过我必须学些新东西。我确实开始担心安全了。财务经理昨天和我聊了，给了我一些意料之中的有关销售系统问题的想法。

9.2 简介

数据库管理系统（DBMS）的强大在于共享数据的能力。数据可以在多个用户、部门以及应用之间共享。绝大多数企业都建立了不止一个应用也不止一个数据库。大型企业还可能需要多个DBMS。很多公司都同时存在由不同小组开发或修正的多个工程。想像一下如果开发者们可以随意创建数据库、表和应用会怎么样。那些应用几乎不可能在一起协同工作。只用一个DBMS是不够的，想建立集成应用的企业必须要有人控制数据和数据库。

*数据管理*包括从计划、协调需求到定义贯穿公司的一致性数据。必须有人或者小组负责决定应该收集什么样的数据，数据怎么存储，并提示数据如何使用。这个人或者小组对数据完整性负责。

*数据库管理*包括建立和运行数据库的技术。基本的任务就是性能监控、备份和恢复，并且

分配和控制安全。数据库管理员应在安装、配置和运行DBMS的细节方面受过训练。

数据库安全是计算机安全话题的一个子集。尽管如此，由于数据共享的目标，安全在数据库管理中是一个非常重要的问题。同时，所有的应用开发者和数据库管理者都应该学学数据库安全中一些有意思的特性。如果数据库安全合理分配，可以减少许多类型的欺诈。如果数据库安全被忽略或者性能很差，这个公司的主要资产可能被世界上任意一家公司操纵或盗取。理解安全问题并适当处理是非常值得的。

9.3 数据管理员

数据是公司的重要资产。设想一下如果当一个公司的全部计算机受到毁坏或者所有数据丢失，它还能生存多久。一些企业或许能存活几天或一周，而很多将会立即破产，比如银行。

公司应该在丢失任何数据前就意识到数据中包含的信息。如图9-1所示，公司因不同的目的会有很多不同的数据库。随着时间的推移，各企业建立了很多不同的数据库和应用来支持他们可操作的、战术或战略决策。每一个应用自身都很重要，但是，当这些应用和数据库协调工作并交换数据时，管理者才能得到整个企业一个完整的画面。

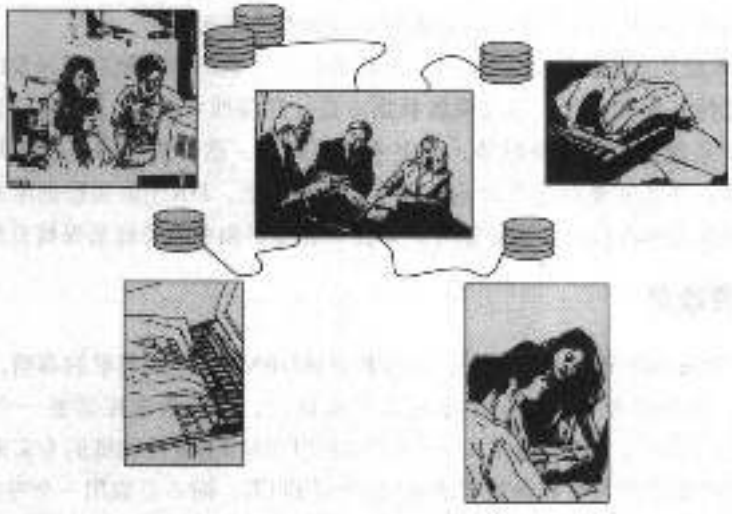


图9-1 数据管理。在众多工程和开发者之间，数据管理员通过协调各项目让数据可以跨应用地集成起来

不管数据库系统多么强大和灵活，不同时间由不同人建立的应用不会自动共享数据。数据集成的关键乃是让某个人管理整个公司的数据资源。绝大多数公司里，数据管理员（DA）充当这个角色。

如图9-2总结的，DA的主要任务就是提供整个企业数据的集中控制。DA建立数据定义标准，保证所有应用都使用一致的格式和命名惯例。DA协调各应用和小组，保证不同工程的数据可以集成到覆盖全公司的信息系统中。如果开发者和管理者之间有争论发生，DA就充当裁判，制定决策保持整个公司的一致性。DA同时监控数据库产业、观察技术趋势，然后给出为了长期利益，考虑应该使用什么数据库系统和工具的建议。

在提议方面DA扮演了一个至关重要的角色。大多数管理者和许多开发者都不知道现代数

据库系统的强大和功能。通过理解管理任务和数据库的能力，DA应该对新应用提出建议，并扩展现有数据的使用范围。

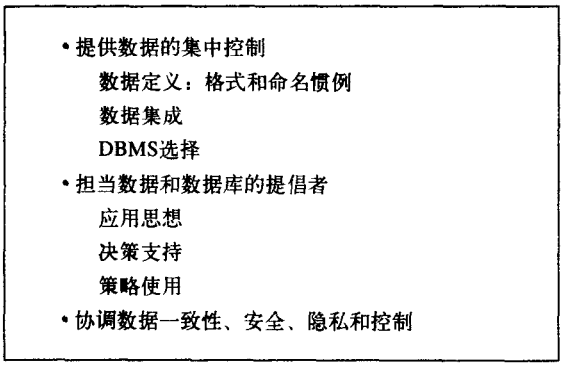


图9-2　数据管理员的任务。DA负责维护数据的质量以及为整个企业集成数据。DA同时还倡导使用数据库，并常常进行安全控制

最终，DA还负责数据的完整性：DBMS中包括的数据真的是公司真实情形的描述吗？公司是否有合适的系统和恰当的控制来保证数据的正确性和适时性？

DA职位更多意义上是份管理工作。DA的任务包括了组织和控制应用开发的设计。控制要靠建立标准、监控开发和变化，以及提供数据库设计的援助来维持。DA同样要花时间和事务管理者评估当前系统，监控事务趋势，确定未来的需求。这个职位的雇员通常都有若干年设计数据库的经验，并且需要对这个公司的细节知识很熟悉。DA还需要数据库的技术知识来理解不同存储对决策意味着什么，还必须能够与技术管理员和事务管理员保持良好的沟通。

9.4　数据库管理员

DBMS是一个复杂的软件包。安装、运行和升级DBMS都不是简单的事情。即使是基于个人计算机系统，这些任务也需要一个专职员工来负责。每个数据库需要一个*数据库管理员*（DBA）来管理。DBA这个职位通常由一个在某特定DBMS上受过训练的专家来充当。在较小的公司里，某个开发带头人可能会被要求履行DBA的职责，而不是雇用一个专家。

DBA的任务更具有技术性。图9-3重点指出，DBA的职责包括安装和升级DBMS。其他任务包括创建用户账户和监控安全，还要负责管理备份。尽管实际的备份任务可能会由系统管理员执行，DBA也要负责设定进度表，确保数据备份的安全。DBA同时监控数据库性能，计划升级等。DBA必须保持和DBMS厂商的联系，跟踪系统问题，系统有变能随时得到通知。如果出现了新的工具、信息，DBA则像一个联络员，收集这些信息并提供给开发者。DBA有着对应用数据完全的访问权。许多企业里，DBA控制着每个数据库的安全。大型公司可能还会任命一个特别的安全员制定方针、程序来帮助监控。尽管如此，DBA通常负责执行数据库安全分配权限的技术细节。

数据分配和存储是DBA日常工作的一项重要任务。一些大型数据库系统需要DBA为每个数据库提前指定一块磁盘空间。一些系统利用创建数据文件和表空间来分配物理空间，这里表空间是数据可以存储空间的逻辑集合。



图9-3 数据库管理员的任务。DBA的任务相当技术，需要每日监控，并对DBMS做出更改

通常数据表、索引和事务日志都分配有独立的空间，DBA必须估计每种组件的大小。如果DBA分配的空间太少了，性能就会受影响，另一面，分配太多空间意味着公司会为不必要的磁盘容量浪费金钱。绝大多数系统都提供追加空间的工具，但最好还是预先估计。第3章讲过的数据量提供了确定空间需求的重要信息。对于表来说，主要的概念就是确定表的平均行宽（按字节），乘以表的期望行数。注意每个DBMS存储数据各有微小差异，有的在存储的每行多加一些字节。文档会提供各DBMS的细节。更精准的办法是建立一个临时数据库，在每个表中创建若干行，然后用实际的平均空间来估计未来的需求。索引和回滚日志需要的空间取决于具体的DBMS和计算机系统。如果需要精确的估计，则必须参考特定DBMS的文档和支持工具。索引和日志的空间需求还取决于应用所定义的事务数目和长度。例如，在一个执行事务处理的数据库中，事务日志将肯定比主要用于决策支持和数据提取的数据库大的多。

9.5 数据库结构

DBA的工作是数据库结构的基础。尽管每种DBMS都有微小的差异，图9-4展示了由SQL标准定义的数据库的一个全面架构。用户由独立的数据库实例定义，并由DBA授予权限。模式是一个容器，作为一个名字空间以避免表的重名。最初，定义后每个用户都有一个独立的空间来建表。两个用户可以创建同样命名为Employee的表而不会引发问题。今天，模式可以为任意目的创建，而不只是为每个用户。目录由SQL99标准提出，通过把模式放到同一个容器里，主要是为了使查找和访问相关模式更加容易。在这点上，并不是任意DBMS都支持目录元素。但是，模式方法相对通用得多。用户和应用分配到默认模式，那个模式下的表和视图都可以直接访问（当然还取决于安全许可）。可是，有时候需要访问不同模式下的表或视图。这种情况下，就需要用全名。全名包括模式名（最终是目录名），举个例子，如果要访问Corporate模式下的Employee表，需要使用SELECT * FROM Corporate.Employee来指示表的全名。如果需要指定目录（比如Main），就需要用Main.Corporate.Employee作为表的全名。像表、视图、触发器这些标准的数据库元素都可用于每个模式。DBA（和DA）的一项任务就是确定何时创建新的模式，尽管没有具体的规则，也



图9-4 数据库结构。模式充当其他元素的容器，以最小化可能的名字冲突

要记住模式的目的是隔离和划分应用。

9.6　元数据

每个厂商都会提供工具帮助DBA完成常见任务。大多会有一个面向图形化的方法来简化使用。另一方面，DBA常常选择使用SQL，通过建立特定过程来完成任务。SQL命令提供操作的详细控制，而且可以写成一次处理数十或上百个操作的形式。例如，图形方法对于增加一个用户来说比较简便，可是如果要增加100个用户，那么写一个从文件或临时表读取用户的SQL过程则更为简单。

在管理上，关系数据库系统强大的方面之一就是：甚至管理用的数据也是存放在表中。这种*元数据*就是数据的数据。例如，一个系统表包含了所有用户表的列表。SQL99标准描述的Information_Schema包含有提供数据库文档信息的一组视图。从技术上讲，Information_Schema视图从Definition_Schema表中提取数据，但是，DBMS厂商可能并不实现Definition_Schema。DBMS厂商已经开发出了自有的系统表存放元数据。这种方法的缺点是没有跨产品的一致性，所以DBA们必须为每种DBMS学习不同的命令。随着厂商们实现新的标准，DBA将会发现工作于来自不同厂商的产品变得容易了。

图9-5显示了一些Information_Schema的通用元素。SQL命令展示了如何基于名字获得表的部分列表。当数据库含有数百个表和视图时，这种命令形式就有用得多了。可以利用SQL的强大而快速找到需要的表，而不需要在数十页的指定表中来回翻动。在这个例子中，必须总是提取出Table_Type和它的名字。表可能是基本类型（真正存放数据的）、视图，或者导出的表。

Schemata Tables Domains Views Table_Privileges Referential_Constraints Check_Constraints Triggers Trigger_Table_Usage Parameters Routines	SELECT Table_Name, Table_Type FROM Information_Schema.Tables WHERE table_name LIKE 'Emp%'

图9-5　Definition_Schema。标准中61种视图中的一部分在左边列出。示例查询显示DBA如何在元数据视图中快速找到特定的项目

9.7　开发阶段的数据库任务

不管你采用什么开发方法（就是说，传统的系统开发生命周期、快速开发、还是原型开发），每一步都有特定的数据库任务。这些任务的大部分都由应用开发者完成。有些需要和DA进行协调。很多任务都需要和DBA交流，获得建议，提供信息，帮助DBA建立数据库。

9.7.1 数据库规划

在可行性和规划阶段，必须对数据存储需求做一个估计。这些最初的估计可能很粗糙，但可以帮助确定支持应用所需的硬件大小和容量。例如，如果只是建一个跟踪5个人资料的简单数据库，那么这个数据库只需要小于100兆字节的存储，运行在个人计算机上。如果最初的大小估计都超过数百兆字节的存储，一个带高速磁盘驱动器的文件服务器可能更为适合。随着数据库估计达到了千兆或者万亿字节，就得采用特殊的数据库硬件和并行处理系统了。

最初的调研还应该提供关于需要的表单和报表数量的要求以及其复杂性。这些数字将用来估计系统开发的时间和成本。一个有经验的DBA可以从类似的工程中估计出空间需求。公司在其他工程上的记录可以提供生成表单和报表的平均时间估计。

9.7.2 数据库设计

设计阶段的基本目标就是了解用户需求，设计出合适的数据表。数据规范化是这个阶段中数据库相关工作的主要活动。最终的表定义也将对存储需求提供更好的估计。

合作协调和项目管理是这个阶段的重要管理任务。如图9-6强调的，合作由DA定义的数据标准所支持。工程可以分成块，分配给各个组成员。标准和交流提供了把块集中成一个完整应用的能力。交流通过共享的数据资源、网络工具、电子邮件以及计算机辅助软件工程（CASE）等手段得到加强。主流的CASE工具包括Oracle Designer/2000，Rational Rose，IEF和IBM的Visual Age。这些工具为包括图表、数据定义和程序编码等所有的工程工作提供了一个集中式的仓库。组成员在属于他们自己那部分工程块上工作时，可以看到工程的其他部分。在面向对象工程中，他们可以使用其他组创建的对象。

- 合作
  - 数据标准
  - 数据仓储
  - 可重用的对象
  - CASE工具
  - 网络和通信
- 细化工程
  - 按阶段陈述
  - 通过用户视图规范化
  - 分配表单和报表

图9-6 管理数据库设计。数据库设计需要合作和标准来确保各个模块能集中成完整的应用。CASE工具和网络通过一个集中式的设计数据的储藏库来加强交流

从数据设计或规范化的观点看，工程常常通过分配表单和划分报表给单独的组成员。然后每个人负责确定业务假设以及定义那些分配的表单所需要的规范化表。各开发者周期性地把他们的工作合到一块儿，创建出一个集中式的将用到数据库中的表清单。这个最终的清单必须遵循DA建立的标准。

9.7.3 数据库实现

数据库实现所需的重点任务由图9-7列出。应用和用户界面的开发是主要步骤。管理和企业的任务很大程度上要求确定应用的总体观感。一旦整体架构确定，编程标准和测试过程将利于合作和保证质量。

另一个重要的管理任务是分配不同数据库的所有权。所有者应该来自业务管理中。数据所有者要负责确定主要的安全规则，并验证数据的准确性。如果DBA对访问权限或者数据的改动有疑问，DBA可以从数据所有者获取额外信息和建议。

必须建立和测试备份和恢复过程。如果任意组件失败，数据库日志应该完全能恢复数据。备份常常用两种方式处理：在预定义的检查点进行全备份，对最后一次全备份后发生的变化进行增量式备份。完全备份更容易恢复，更加安全，但是很耗时间，也需要大量的备份空间。对于小型数据库，全备份不是问题。对于大型、连续变化的事务数据库，或许只有可能一周左右进行一次全备份。

用户和操作员同样需要训练。不管用户界面设计得多么仔细，也总是需要为用户提供一个至少是入门培训的课程。类似地，计算机操作员可能需要在备份和恢复上进行培训。

- 应用程序的编程标准
  - 用户界面
  - 程序结构
  - 程序变量和对象
  - 测试过程
- 数据访问和所有权
- 载入数据库
- 备份和恢复计划
- 用户和操作训练

图9-7　实现管理。必须细心选择用户界面。编程标准和测试过程有助于保证各组件之间的兼容性，并提供质量控制。业务管理者应该分配数据的所有权，便于他们对于安全条件和质量做出最终决定。必须创建和测试备份、恢复计划。对操作员和用户必须创建训练程序

9.7.4　数据库运行和维护

一旦数据库投入运行，DBA将完成绝大部分任务。关键的任务有：（1）监控使用和安全性；（2）进行备份和恢复；（3）支持用户。

监控性能和存储空间是管理一个数据库的紧要因素。监控常常用于调整应用性能以及估算增长并计划将来的需求。安全访问和变化也同样被监控。安全日志可以跟踪关键数据的变化，也可以在出现疑问时跟踪某个用户的使用（包括读和写）。

监控性能和用户问题对系统提供了有用的反馈。如果用户都在某个地方出现问题，就促使开发小组改善那里的表单。类似地，如果一些用户运行的查询花费了很长时间来执行，就该找设计小组来为这些查询创建更加高效的程序。例如，不要期望用户辨认出或更正一个相关子查询。相反，如果DBA看到用户运行需要很长时间运算的复杂查询，小组就该给应用中增加一个新的部分来存储和执行一个更有效的查询。

类似地，如果一些人使用数据库的某个部分非常频繁，那么可能给他们一份那个主要部分的副本更为高效些。如果用户不需要最新的数据，可以在服务器上建一个小数据库，每夜更新。这样用户将最终得到更快的响应时间，因为他们有小数据库且需要更少的通信时间。数据库剩下的部分也将运行更快，因为频繁用户少了。

数据库厂商提供强大的工具来帮助分析查询和数据库性能。利用这些工具，可以分离整个查询过程，精确地观察哪步占用了最多的时间。有了这些知识，开发者就可以换一种方案来避免瓶颈。其他一些工具可以监控死锁和事务问题，也让修正问题变得相对容易。调整一个大型数据库来提高性能是个复杂的问题，且严重依赖于特定DBMS的功能和工具。

9.8　备份和恢复

可能最关键的数据库管理任务就是备份了。不管规划得多好，也不管安全系统多么精密，也总会出错。数据库管理者和开发者有义务为灾难做出计划。计划中最关键的方面则是确保可以很容易访问到数据库的当前副本。任何类型的灾难——火灾、洪水、恐怖袭击、能量不足、

计算机病毒，磁盘毁坏或意外删除——都需要备份数据。假如制作和存储备份的成本不高，就没有理由不在任何时候都有一份可用的当前备份。

如图9-8显示，数据库备份提出了一些有意思的挑战——尤其是当数据库必须每天24小时、每周7天（简称24-7）工作时。基本的问题就是当数据库正在备份时，数据的改变仍在发生。就是说，数据库的每个副本都立即失效——即使正在制作时，一个相关的问题是，有可能数据库中DBMS的副本例行程序需要等待的那部分正在使用中（很可能造成死锁情形）。

幸运的是，较大的数据库系统都提供了很多工具来解决这些问题。大多数情况下DBMS取备表的一个快照（snapshot）。快照表示表在某时刻的状态。然后数据库对最后一次快照后的所有改变都维护一个事务日志。

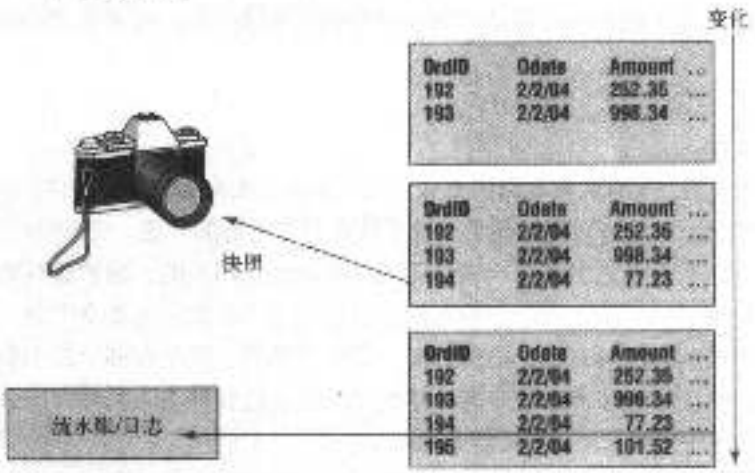


图9-8 变化中的数据库备份。备份程序在某时间点上取一个快照，新的变化都存在流水账或日志中。恢复程序载入快照并增加或删除日志中的变化

未决的是DBA必须决定多长时间进行一次增量式（部分）备份以及多长时间需要进行一次全备份（快照）。部分备份创建得更快，且需要的存储空间更少，但是在灾难中需要更长的时间来恢复。

如果出现问题，数据库就必须从备份磁带中得到恢复。首先，DBMS载入最新的快照数据，然后检查事务。完成事务回滚，变化被重新写回到数据表中。如果备份发生在事务中间，而事务并未完成，DBMS就会回滚或移去最初的改变，并重启事务。要记得事务包含的一系列改变必须全部成功或者一起失败。DBMS依赖于第7章描述事务的应用定义。

备份必须按周期性的时间表来执行。有时候，时间表需要修改——特别当数据库记录改变较多时。要知道自最后一次备份以来的任何更改都存在流水账或事务日志中。DBA必须观察事务日志的空间，如果太满了，备份程序就得提早进行。如果这些非期望的备份发生得太频繁，就该修改时间表了。

备份磁带必须存放在别处。否则，一场大火或其他灾难有可能毁掉建筑内所有存储的数据。快照和流水账日志起码应该每天复制到别处去。如果公司比较大，能支持不止一个地点的计算机工具，那么网络将使传送数据更加容易。不少公司都有抗灾难的储藏室来存放数据磁带和磁盘。在极端情况下，可能都值得拥有双份计算机资源，且有编制的程序让系统自动把主

数据库的改变映像到不同地点的备用计算机上。当出现事故时，备份计算机可以立即接过操作。但是，即使在这种情况下，都必须制作物理备份。

9.9　安全和隐私

计算机安全是当今每个公司都面临的问题，任何计算机应用都面临安全问题。数据库在一个地方聚集了大量数据让人们更加容易地获取和修改。换句话说，数据库是必须保护的关键资源。然而让数据库如此有用的因素同样决定了更难保证其安全性。特别地，数据库的目标就是共享数据。从安全的角度讲，你要控制谁能分享数据以及那些用户能做什么。

计算机安全有两个基本类别：(1) 物理安全，(2) 逻辑安全。物理安全关注物理性地保护计算资源，以及为可能损坏的设备和数据的物理灾难做准备。逻辑安全包括保护数据和控制数据访问。

9.9.1　数据隐私

隐私和安全相关，但有一些细微的差异。公司和政府机构收集客户、供应商和雇员的大量资料。隐私指的是控制这些数据的流传，并尊重这部分人群的意愿。图9-9展示了一些对商业数据的要求，而这些数据是和市场、雇员管理、政府需求相关的。维护这些数据的正确性以及限制谁有权访问它们，这些概念对安全和隐私来说是一致的。在安全方面，每家公司都有保护自身数据安全的自身利益。在隐私方面，至少在美国，很少有哪个公司有私人数据操作的规则或限制。然而，客户和雇员可能希望公司保持自己资料的隐私性，而公司却可能因金融利益的驱动把这些数据交换或者卖给其他公司。



图9-9　隐私，收集和分析数据有很多理由。人们可能会觉得一些理由很有侵犯性。尽管几乎没有隐私法律存在，商家和数据库管理员还是应该在造成影响和使用这些数据中寻求一个平衡

在数据隐私方面，最重要的问题就是，谁拥有数据？绝大多数情况下答案是收集数据的公

司或个人。一部分人，尤其是在欧洲，建议应该考虑把个人作为拥有者。然后公司必须获得许可，或者购买许可，才能使用或者交换个人数据。到目前为止，技术的局限性阻止了大多数这种付费模式的实现。但是，公司必须关注有关隐私的法律。

数据库管理员在数据隐私方面负有道义责任。很多时候你有权限访问到关于客户和其他雇员的私人数据，但你有义务维持这些数据的隐私：你不能把这些数据暴露给其他人。事实上，你应该尽量避免读取这些数据。你同样不该容忍企业内其他工人对这些数据的滥用。如果你监测到隐私（或安全）被其他人侵犯，你应该报告问题，并通知合适的上级主管。

9.9.2 威胁

计算机安全的主要威胁有哪些？什么样的可能事件将造成数据库管理员的噩梦？是你在电影里看到的外面的黑客或者解密高手吗？它是旋风、飓风还是地震（电影里也很常见）？

都不是。任何公司的主要威胁都来自“内部”。公司可以为其他所有威胁制定计划，而且有各种各样的工具来帮助最小化问题。但是，你必须相信你的雇员、顾问和商业伙伴。为了让他们工作，他们需要对你的计算机有物理的访问权，对数据库有逻辑的访问权。一旦你决定赋予权限，再控制他们的行为就难了。并非不可能，而是更困难。

另一面，更多阴险的威胁来自有意破坏数据的程序员。有一项技术，可以在程序里嵌入时间炸弹。时间炸弹需要程序员每天输入一个秘密代码。如果程序员离开（或被解雇），不能再输入这个代码，程序就开始删除文件。还有其他的情况，程序员编制程序故意改变数据或者转移资金到他们的私人账户。这些例子展示了问题的核心。公司必须信任他们的程序员，但是这种信任潜在地带来了相当多的危害或欺诈。这也是公司大都会对信息管理系统的雇员做坏事特别敏感的原因之一。作为一个开发者，必须始终看到一个信任的景象。

9.9.3 物理安全

在物理地保护计算机系统这个问题上，最重要的任务就是确保时刻都有一个当前备份。这种维护备份的方针同样适用于硬件。万一失火或者有其他物理灾难，就需要收集数据磁带，然后找一个计算机来加载和运行。不能等着灾难发生，你确实需要创建一个灾难应急计划。

*灾难应急计划*是当灾难发生时信息系统部门能执行的一个完整的计算机操作步骤的序列。灾难计划，描述了什么人负责，每个人要采取的步骤，列出联系号码，并告知如何让系统恢复并运行起来。找到一台替代计算机有个流行的方法，就是租用灾难应急计划公司的热站。*热站*包括一组含有动力、终端和通信系统的计算机工具，还有一个计算机。只需要每月付费即可在灾难发生时有权使用这些工具。一旦灾难发生，激活相应的灾难应急计划，收集数据磁带，加载系统，装载你的备份磁带，并从热站运行系统。一个稍便宜点儿的选择是租用冷站。*冷站*或Shell site和热站类似，但没有计算机和电信通讯设备。如果灾难发生，打电话给你的硬件厂商，再买一台新计算机。事实上，厂商都会很配合的。关键在于接受和安装新计算机可能需要若干天，你的公司这些天没计算机系统能存活吗？如果只替换较少的计算机，一些灾难恢复公司可以开着卡车，到你的站点去，在你的停车场里运行系统。

在保护私人电脑时一个更有趣的问题出现了。首先，为很多个人计算机备份数据更加困难。如果他们连着网络，数据可以传送到中央文件服务器并从那里复制。如果失火，至少数据能

恢复。可是，计算机呢？如果你丢失了5台或10台计算机，你可以轻易在本地的电脑商城买来替代单元。但如果你丢失了几百台计算机，替代它们可就难多了。一个很有创意的方案是，帮你的雇员购买他们在家里用的计算机。如果发生问题，在你重建中央计算机系统并替换那些个人计算机时，他们还可以在家里工作。

如图9-10总结的，预防是提供物理安全的另一个重要步骤。计算机设施应该有火灾监测和保护系统。类似地，计算机设施应该远离易发洪水的平原、地震危险区、潮汐地区和其他易发生灾难的区域。同时应该限制对计算机、网络设备和私人计算机的物理访问。绝大多数公司都有带电子锁的公司机构证章。还需要控制访客、流动人员和临时雇员的访问权。

- 数据备份
- 硬件备份
- 灾难应急计划和测试
- 预防
  - 地点
  - 火灾监测和控制
  - 控制物理访问权

图9-10　物理安全控制。数据备份是最重要的步骤。找一个移入备份的地点是第二步。灾难应急计划和预防措施可以帮助预防问题和更快地恢复

9.9.4　管理控制

由于数据面临的威胁主要来自公司内部，所以传统的管理控制就在提升安全方面起到了重要的作用。例如，最重要的控制之一首先就从雇佣程序开始。一些公司会对雇员做背景调查以确定其性格和可信赖程度。即使是简单的推荐验证制度都可以使问题最小化。类似地，公司也会在终止雇佣关系上更加谨慎——尤其是有着很多数据库访问权的管理信息系统的雇员。即使是临时解雇，访问权限和密码也会立即收回。

敏感的工作都是分段的。比如说，完成金融事务需要若干名雇员，和更多金钱有关的事务会分配到更高级别的雇员身上。类似地，对外机构如银行也经常暂停，由高级主管检验大型事务。事务通常按时间、地点和执行操作的人来进行监控和记录。

在某些情况下，可以通过对硬件进行物理控制来提升安全性。计算机集中放在加锁的和有人看守的房子里。雇员们也经常被摄像头跟踪。安全印章也常用于跟踪雇员访问这些地点和计算机硬件。

咨询机构和商业联盟也增加了与安全相关的问题。通常，对于选择咨询机构和任意合作的雇员都较少有控制性。尽管你可以在选择咨询公司时加以控制，但对于分配到你处的特定雇员你还是没办法。这些风险可以通过限制他们对数据和物理地点的访问权来控制。有时候，你可能希望给每个顾问都配一个内部雇员。

9.9.5　逻辑安全

逻辑安全的本质就是你想让每个用户都能从某种程度上访问数据，却可以精确控制用户拥有的访问类型。你同样希望监控对数据的访问，用以发现潜在的问题。图9-11显示3种你想要避免的基本问题：未授权的泄漏、未授权的修改和未授权的信息截留（或拒绝服务）。

- 未授权的泄漏
- 未授权的修改
- 未授权的信息截留（拒绝服务）

图9-11　逻辑安全性问题。每种情况都可能给公司带来麻烦，包括金融损失、时间浪费、销售减少或公司解体

某些信息是需要保护的，应该只有特定组的用户才能访问到。例如，必须保护好公司的战略市场计划，以防竞争者们得到这些数据。为了保证安全，

只有公司的若干核心人员才能访问这些计划。

有些信息可以显示给用户看，不过却不应该被用户更改。例如，雇员应该能够查看人力资源文档来验证他（或她）的薪水、剩余假日或价值评估。但是，如果允许雇员修改这些数据，势必造成错误。举个例子，不管雇员多么诚实，允许他们修改自己的薪水都是一个危险的诱惑。

第三个问题很细微但同样危险。考虑一下如果首席金融官需要获取数据来定案一笔银行贷款，而安全系统设置得不正确，拒绝提供这些需要的数据会出现什么样的问题。如果数据没有在那天结束前传送到银行，就会耽误公司的多笔支付，受到消极的评价，损失20%的股票价格，并冒着破产的危险。其关键就是，合法用户被拒绝提供数据可能和把权限赋错了人同样危险。

假设你拥有一个复杂的计算机系统和一个支持安全控制的DBMS，防止这些问题只需要两步。首先，计算机系统必须能够鉴别每个用户。其次，数据的拥有者必须对每块数据分配合适的访问权限。DBA（或安全指挥官）有责任分配和管理用户账户，从而惟一地识别用户。应用设计者和数据拥有者联合确定必要的安全控制以及每位用户的访问权限。

用户鉴别

计算机逻辑安全的主要困难之一就是鉴别用户。人类识别其他人是运用建立在外貌、声音、笔迹等之上的复杂的模式识别技巧。然而即使是人也可能被蒙骗。计算机在模式识别方面很脆弱，所以还需要其他技术。

鉴别用户最常用的方法就是通过用户名和密码。每个人都有一个独立的账户名并选择一个密码。理论上，只有个人用户和计算机系统知道这个密码。当用户输入正确的名字和对应的密码后，计算机就接受认证这个人。

问题是，计算机在记忆密码方面比人强得多。因此，人们常选择费解的密码。一些创建密码的基本规则在图9-12列出。最好的密码都较长，包含非字母字符，和用户没有关系，并且常常改变。这样很好，但当今用户可能轻易就需要10个、20个甚至更多个不同的账户和密码。几乎没人能记住安全所需的每个账户和费解的密码。所以很自然的一个倾向就是，或者把密码写到某个方便的地方（这样可能被其他人发现），或者选择简单的密码（这样可能被猜出）。

- 不要使用“真实的”单词
- 不要使用个人（或宠物）的名字
- 包含非字母的字符
- 至少使用6个字符
- 经常改变密码

图9-12 密码建议。挑选不在字典里并且很难猜的密码。关键是，你是否能记住复杂的密码？

目前密码是最容易实现的系统。已经有一些在中央安全服务器存储密码的工作了（例如Kerberos），就是用户登录主服务器，然后所有其他软件都利用服务器验证用户。另一种方法是使用密码生成卡片。每个用户带一个每分钟都能生成新密码的小卡片。登录时，计算机产生一个和那张卡片同步的密码。一旦密码使用过后，就立即失效，所以即使入侵者看到密码，也没有任何价值。该系统仍然需要用户记住一个短密码以防密码卡被贼偷去。当然，如果用户丢失了密码卡，也就无法访问这台计算机了。加密的软件变种可以装载到笔记本电脑，然后提供对共有网络的访问。

其他方案都正在开发中，目的是抛弃记忆密码的需要。靠度量物理特性的生理测定系统已

经存在并越来越廉价。例如，指纹、手印、虹膜、声音鉴别以及热力图像系统都工作得很好。生理测定方法的优点是用户不需要记忆任何东西，也不需要带任何可能丢失或被盗的设备。主要的缺点还是成本，因为验证设备必须安装到任何雇员可能访问计算机的地方。一个次要问题是尽管这些设备在阻止未授权访问方面表现良好，但仍有很多保持了较高的失败率，拒绝合法用户的访问。

个人用户被鉴别后，大多数系统允许把个人分配到组。组使得分派用户权限更加容易些。例如，通过把100个雇员放到文书组，就可以赋予组权限，这样比分派100次相同权限要快得多。

访问控制

用户被鉴别以后，就可以赋予他们对任何资源的特定许可。从数据库的角度看，必须设定两个级别的访问。首先，必须利用操作系统命令，赋予用户对整个数据库的访问权限。其次，利用数据库安全命令，赋予用户具体的许可。

要记得DBMS也只是运行在计算机环境的另一个软件。它利用受操作系统控制和监视的文件来存储数据。在任何人访问到DBMS控制的数据前，需要计算机赋予这个人访问整个数据库目录和对应文件的许可。这种情况也要求用户被两次鉴别：一次被操作系统，一次被DBMS。在某些系统上（比如Windows 2000），DBMS可能会直接接受操作系统的用户标识。在没有安全设施的操作系统上（如基于个人计算机的Windows），DBMS负责提供所有的安全性。即使有高级的数据库系统，也可能需要为某些应用而在DBMS内部创建用户账户。比如说，基于Web的应用一般会避免使用操作系统的登录机制，并且更为简单地在数据库内部注册用户和密码。

操作系统的许可包括在目录级别（读取、查看、写入、创建和删除）的许可。类似的许可同样适用于独立的文件（读取、写入、编辑和删除）。在大型系统中DBMS以独立的用户登录。DBA保证了这个DBMS用户对他（或她）自己的文件拥有完全的目录、文件控制权。如果你使用的是网络上一台基于个人计算机的系统，必须通过网络操作系统来赋予用户对你自己文件的访问权。例如，用户需要对你存放文件的目录拥有读权限；同时应该给他们在特别文件和加锁文件的读写权限；但是，要避免给他们对任何文件的删除权。一个拥有删除权限的用户若心怀不满，就可能删除你整个数据库——即使他（或她）没法改变这些特定的设置。当然，你有备份，但是找到问题并且重新加载数据库对用户来说是非常烦人的。

当操作系统用户对所有的文件都有权访问时，可以使用DBMS的安全系统来控制对单个表的访问。大多数情况下，DBA必然会在DBMS内部创建用户账户来鉴别每位用户。在某些环境下，这种技术意味着用户得登录两次：一次到操作系统，一次到数据库。所有的数据库系统都有一个DBA用的安全工具，来帮助添加删除用户和更改密码。

- 读数据
- 更新数据
- 插入数据
- 删除数据
- 打开/运行
- 读取设计
- 修改设计
- 管理

图9-13　DBMS特权。这些特权适用于整个表或查询。前三个（读、更新和插入）最为常用。设计特权通常仅仅赋予开发者

图9-13列出的特权可以应用于整个表或者查询。绝大多数你能得到的特权就是读、修改和插入。删除许可意味着用户可以删除整行；它只应该在特殊情况下赋予少数人。更强大的特权可以让用户读取和修改表、表单、报表的设计。这些特权只应该留给那些可信的用户。用户几乎不会想修改原本的

表的设计。这些特权是给开发者的。

在大多数数据库系统里，基本的安全许可都可以通过两条SQL命令设置：GRANT和REVOKE。图9-14表示GRANT命令的标准语法，REVOKE命令类似。SQL 92通过允许在GRANT和REVOKE命令中指定列，提供了一些额外的安全控制。所以现在可以仅仅对一个或两个列分配权限。不过，这种特权会应用于表和查询的每一行。

```
GRANT privileges
ON objects
TO users

REVOKE privileges
ON objects
FROM users
```

图9-14 SQL安全命令，大多数系统同时提供一个可视化工具来赋予和回收访问权限

大多数的数据库系统提供可视化的安全工具来帮助分配访问权限。当执行同时给大量报表或者很多用户分配权限这种批操作时，SQL命令很有用。可视化工具在单一操作中更加好用。同时，你可以轻易地看到很多用户当前已经分配的许可。

GRANT命令还提供一个有时有用的额外选项。作为数据库元素的所有者，当你准予时你可以传播你的权力。如果添加*WITH GRANT OPTION*字句，那么任何你指定接收这些权限的人都可以再把这些权限传递给其他人。例如，你可以准许WITH GRANT OPTION市场部门的头目拥有一个表的SELECT（读）权限。那么，这个人就可以再把读权限赋予市场部的其他人了。

数据库角色

想象一下如果你必须给成千上万个雇员分配许可，你将面临多大的管理难题？每次有雇员到来或离开，就得有人分配或去除可能与数百张表或视图有关的权限。这项任务很耗时间而且非常容易出错。即使你建立SQL过程来帮忙，也几乎需要时时去维护，一个相对特别简单有效的解决方案就是定义数据库角色。角色常常和一组用户相关，并且包含着一些同时分配的权限。如图9-15所示，你创建一个角色，并且把需要的许可分配给角色，而不是分配给个人用户，然后再把角色分配给特定用户。当有新的雇员到来时，把角色添加给用户就提供了所有必要的许可。SQL 99标准有把角色分配给其他角色的功能，例如，你可以定义一些较小的任务集合当作角色（销售条目、添加客户、更新库存），然后把这些任务分配给更大的角色（售货员、库存管理员等）。

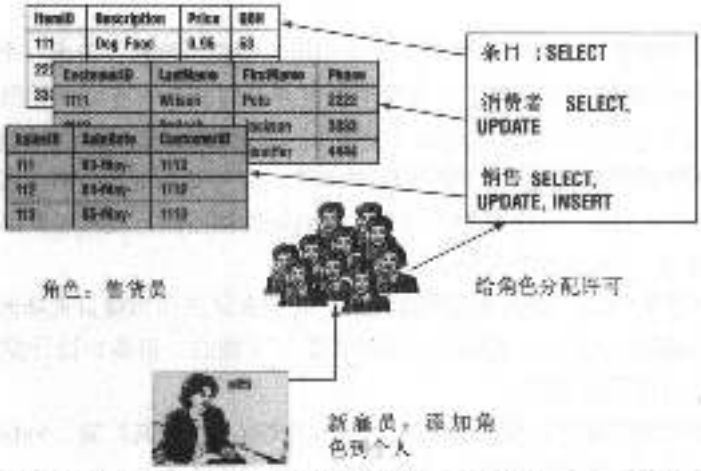


图9-15 数据库角色。创建一个角色并给角色分配许可。添加或删除雇员时，只需简单地分配或移除需要的角色就可以立即提供适当的许可

作为控制的查询

在很多系统里，基本的安全命令很强大，但在有用性方面稍微有些局限。很多系统仅仅能对整个表或查询分发和收回权限。数据库安全系统真正强大的地方在于它允许把权限分配到独立的查询上。看看图9-16的例子。有一个Employee（雇员）表列出了每个工人的姓名、电话号码和薪水。想把这个表当作电话本使用，这样雇员们就可以查询其他工人的电话了。可问题是，不想让雇员们看到薪水值。解决方法就是创建一个仅包含姓名和电话号码的查询。要记得查询并不复制数据，只是简单地从其他表中提取数据。现在，把Phonebook查询的SELECT特权分配给所有雇员，并回收所有雇员在原始的Employee表上的特权。如果你愿意，还可以从Employee表中选择特定的行。比如，你可能不想让高级主管们的电话号码显示出来。

第4章和第5章展示了查询的强大。利用这种强大的功能可以建立事实上你需要的任何级别的安全。实际上，任何用户对数据库的存取权限都是通过查询来实现的。避免任何直接针对表的权限分配。那么，随着商业需求的改变而修改安全条件就容易多了。

基本的过程就是与用户协商，并准确地了解每个用户访问数据时都需要哪种类型的权限。特别地，确定哪些用户有增加或改变数据的需求。然后创建用户并给查询分配合适的安全条件。一定要对每个用户组进行应用的测试。如果安全问题很重要，可以考虑安排一些程序员以不同用户的角度去“攻击”数据库，检查他们是不是能够删除或者改变重要文件。

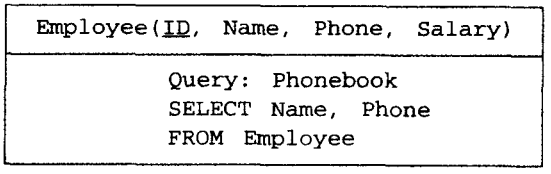


图9-16　使用查询的安全性。你想所有的雇员都能够查询工人的电话号码，但不想让他们看到薪水值。定义一个只包含所需数据的查询，然后把对这个查询（而不是原始表）的权限分配给用户

9.9.6　职责分割

若干年来，安全专家们都在担心公司的工作人员进行偷窃或欺诈。考虑一个似乎年年都会出现的典型情况。一个采购经理创建了一个假的供应商。假装公司有货到并批准了付款。当然，这个经理自己兑现了付款支票。这样的一些欺诈在罪犯被抓住前可能会实施很多年。

避免这种问题的标准做法就是分清所有职员的职责。目标是保证在所有主要的金融事务中都至少涉及2个人。举个例子，一个采购经理可以找到新的供应商，或许兼发出采购请求。另一个人将负责接收请求，然后第3个人批准付款。

划分职责的目标要实现起来还是很有挑战性的。公司会试图利用裁员来降低成本。生意起色了，一些人就会利用这种混乱。消除所有的欺诈是不可能的。但是，设计完好的数据库应用还是能够提供一些有用的控制的。

考虑图9-17里面的采购实例，其基本表包含了Supplier（供应商）表、SupplyItem（供货条目）表、PurchaseOrder（采购订单）表和PurchaseItem（采购条目）表。另外，财务表批准和记录付款。职责分割的关键就是正确地给每个表分配权限。采购经理是惟一有权在Supplier

表上添加新行的用户。采购员是惟一有权在PurchaseOrder表和PurchaseItem表添加新行的用户。接收员是惟一有权记录供应品收据的用户。现在，如果一个采购员想创建假的订单，他（或她）将不能创建新的供应商。因为Order表和Supplier表之间有强制的参照完整性约束，这个职员甚至不能在订单的表单里输入一个假的供应商。同样，付款只会被送到合法的公司，并且采购经理也没法假造到货的收据。数据库安全系统的强大就在于它总是强制执行分配的责任。

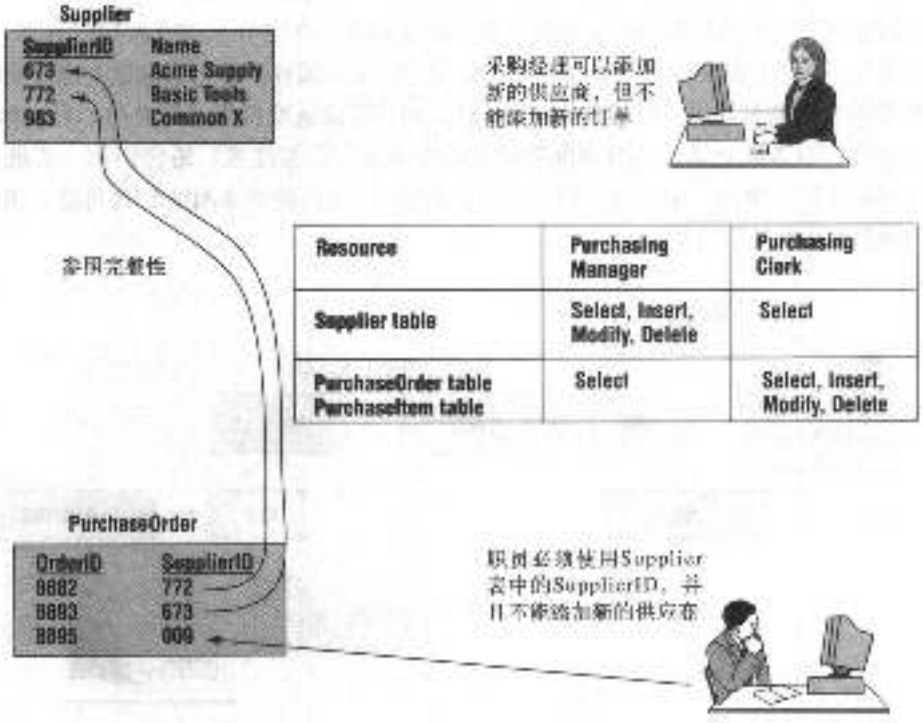


图9-17 职责分割。职员不能创建假的供应商。参照完整性强迫职员输入Supplier表中的SupplierID，并且不能在Supplier表中添加新的行

9.9.7 软件升级

当代的软件都庞大且复杂。软件厂商和其他研究人员常常在软件发行后发现安全问题。一般，公司会有一个评估安全威胁、打补丁修补问题并通知用户升级系统的过程。困难的是，对用户宣告安全瑕疵也意味着潜在的黑客被这个问题所提醒，只要所有的用户能立即给系统打上补丁，这样的宣告就很有效。但是，作为一个数据库管理员，问题并不如此简单。即使你收到一个通知，你仍然需要测试这个补丁并确保它在你的应用里不出问题。这时，黑客们就借助已知的安全漏洞，对没打补丁的公司系统发起了攻击。

因此，数据库管理员的一项主要任务就是关注DBMS和操作系统软件的安全版本发行。这些补丁需要在测试机器上安装，并尽快转移到实际系统中。同时，网络管理员可以屏蔽特定的端口，防止外人访问数据库。DBA和网络管理员还必须细心地监控网络和服务器，查看是否有无赖进程的启动。

9.10　加密

加密是根据一些码来修改原始信息的一种方法，这样只有用户知道解码才可以读出信息。加密可以用来从一台计算机向另一台计算机传输信息。存在一台计算机上的数据也可以加密。没有解密密钥，文件就是乱码。加密对那些不提供用户鉴别和权限控制的基于个人计算机的系统来说非常关键。加密在通过网络（尤其是因特网）传输数据时也非常重要。

当今流行两种基本的加密方式。大部分方式都使用同一个密钥来加密和解密消息。例如，高级加密系统（AES）就使用单独一个密钥。尽管AES是美国标准，但它的版本遍及全球，因为它是基于两个比利时译码者创建的Rijndael的。AES算法速度很快，并支持128、192和256位的密钥长度，有效防止试验所有可能密钥值的穷举攻击。要注意，给密钥加一位就会产生了两倍数目的密钥；例如，使用当代的计算机，从56到128位就得多测试2^{72}种可能。图9-18展示了AES加密方法的基本应用。

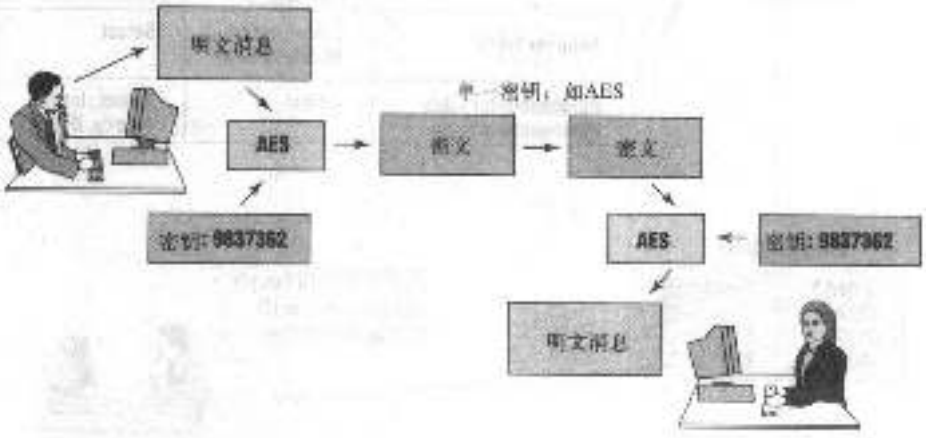


图9-18　单一密钥加密。加密和解密消息使用同一个密钥。当涉及很多用户时，分发和控制密钥的访问权限就成了主要问题

AES的主要缺点是需要双方有同一个加密密钥。另一种方法同时使用一个私有密钥（私钥）和公有密钥（公钥）。不管用哪个密钥加密消息，用另一个才能解开。RSA（Rivest-Shamir-Adelman）算法就是使用双密钥方法的一个实例。RSA在多种计算机上都具有保护作用。RSA加密的实施基于素数的性质。具体讲，就是把两个素数相乘相对很容易，但再把那个很大的结果分解回两个素数，就极其困难。RSA方法的安全性依赖于使用超大数字（128位或更多），这样用当前技术分解的话就需要很多年。

使用双密钥的方法有很多有趣的应用。其诀窍是：人人都知道你的公有密钥，但只有你知道私有密钥。考虑如下情形，Bob想通过因特网给Alice发送一个数据库事务。Bob在字典里查到了Alice的公钥。一旦消息用Alice的公钥加密，就只有她的私钥才能解开；没有其他人可以读取或者改变这条事务消息。但是，消息可能在Alice读到之前就被人破坏。

双密钥系统还有另一种称为认证的应用。我们还是认为Bob想给Alice发送消息。为了确保只有她能够读取，他用她的公钥加密了数据。但是，Bob担心有人冒充他的名字给Alice发送虚假消息，他想保证Alice能知道消息确实来自他本人。如果Bob同时用他的私钥加密消息，消息

就只能被Bob的公钥解开。当Alice接到消息后，用她的私钥和Bob的公钥来解密。如果消息可读，就肯定是Bob发的。这种情形在图9-19中展示。

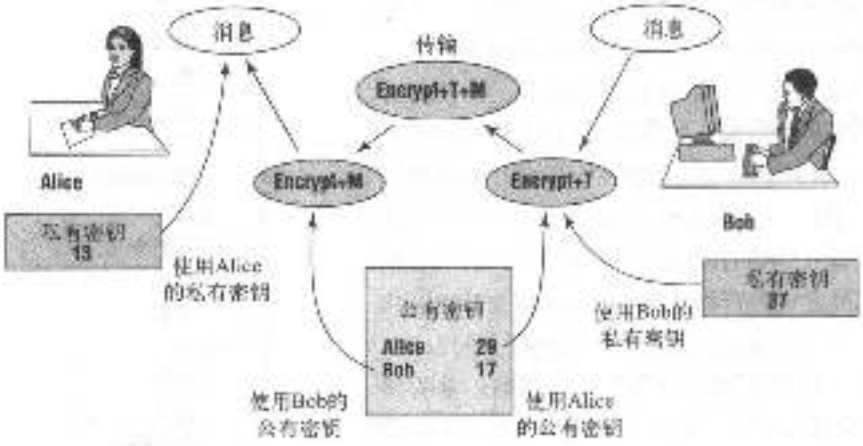


图9-19 双密钥加密。Bob给Alice发送消息。先通过自己的私有密钥，Bob对消息进行认证。然后通过使用Alice的公有密钥加密，就只有Alice能读取消息

当代的加密模式都有强大的工具支持。可以使用它们自动地保证数据的安全存储和传输——即使在因特网这样的开放网络中。要牢记：所有的加密模式都会受到穷举攻击的威胁。随着计算机越来越快，旧的加密模式会更具风险。

双密钥加密系统在信息通信的所有方面都非常有用。但确实有一个复杂的问题：列出所有公有密钥的字典必须准确。想想如果有人假扮了Bob并自己创建了一对私有和公有密钥时会发生什么。这个攻击者将在任何事务被看作是Bob。因此，公有密钥必须由一个可信组织来维护。而且，这个企业必须仔细地检验任何申请密钥的人（个人和集团）的身份。一些公司已经开始提供这种称作认证中心的服务了。Verisign就是其中一个带头的商业公司。

作为内部应用，可以相对容易建立一个公司服务器作为认证中心。通过这种方法，可以获得内部的安全认证，保护雇员之间的传输和内部数据库应用。这种方法比为每个雇员购买一个年度的认证要便宜。当然，你的认证很可能不被公司之外的人接受，所以你仍然需要为那些和外面公司、个人打交道的应用购买商业认证。

9.11 Sally的宠物商店

为Sally的宠物商店分配安全许可的第一步是区分不同的用户组。图9-20展示了初始列表。随着公司的成长，最终还会有其他类别的用户。注意这些都是组，而且每种类别都会有人被分配进来。

第二步是确定不同用户将要执行的操作。设计单独的表单支持每一种活动。图9-21包含了一个主要活动列表的部分内容。

需要在操作系统和DBMS里创建用户和组的账户。创建了表、查询和表单后，DBA应该确保只有DBA才能够读取和修改数据。

现在检查一下每种操作并确定执行操作所需要的查询和表。

- 管理
  - Sally/CEO
- 全体销售
  - 商店经理
  - 销售人员
- 商业联合
  - 会计师
  - 律师
  - 供应商
  - 客户

图9-20 Sally的宠物商店里用户组的初始列表

应该对每个需要完成操作的用户组列出许可。图9-22展示了从供应商订购项目可能需要的许可。注意，只有商店经理（和老板）可以定购新的货物（即MerchandiseOrder和OrderItem表的添加权限）。同样，只有老板可以添加新的供应商。要记得有一个参照完整性约束，强迫MerchandiseOrder表只使用Supplier表中出现的供应商。因此，商店经理将不能捏造出一个假的供应商。同时，你可能希望商店经理在OrderItem表中添加条目，但不让他们修改已经完成的订单。DBMS可能不支持这种约束，并且你很可能已经赋予了经理写入的权限。如果这样，这种差别就会很有用。否则，掌管接收产品的经理就可能窃取一些条目并更改原始的订单数量。如果Sally有足够多的经理，这个问题就可以通过分割职责来最小化，就是让一个经理发出订单而另一个经理记录接收到货。

- 产品
  - 销售
  - 购买
  - 接收货物
- 动物
  - 销售
  - 购买
  - 动物健康保护
- 雇员
  - 雇佣/解雇
  - 时间
  - 薪水
- 账目
  - 支付
  - 收入
  - 管理报表

图9-21　Sally宠物商店里的主要操作。所有这些事务都在数据库里建立表单或报表

订　购	订购查询				订购条目查询	
	货物订单	供应商	雇　员	城　市	订单条目	货　物
Sally/CEO	W/A	W/A	R: ID, Name	R	W/A	W/A
商店经理	W/A	R*	R: ID, Name	R	A	R
销售人员	R	R*	R: ID, Name	R	R	R
会计师	R	R*	R: ID, Name	R	R	R
律师	———	———	———	———	———	———
供应商	R	R*	———	R	R	R
客户	———	———	———	———	———	———

* 基本的供应商数据：ID、名称、地址、电话、区域码、城市ID

R = 读取；W = 写入；A = 添加

图9-22　关于订购的许可。只有老板可以添加新的供应商，并且只有最高级别的经理可以创建新的订单

同样要注意Sally想记录发出订单的雇员身份。为达到这个目的，需要EmployeeID和Name列的读取许可。这种特权可以通过创建一个独立的只从Employee表中提取最少列的EmployeeName查询来建立。然后为订购，可以使用这个查询代替原始的Employee表。

小结

管理一个数据库涉及到若干步骤。DA执行与设计、规划相关的管理任务。其中建立利于共享数据和集成应用的标准有着较高的优先级。DA同时与用户和商务经理一起工作，发现新的应用。相对地，DBA负责安装和维护DBMS软件，定义数据库，确保数据备份，监控性能，并且帮助开发者。

应用开发的每一步都涉及数据管理的不同方面。规划时必须估计规模和大致的开发成本。设计阶段会用到项目管理技巧和团队思想来分解项目，并分配给个人。实现则需要建立和加强开发标准，测试过程、培训以及操作计划。一旦应用运行起来，DBA在空间和处理时间两方面监控性能。还需要调整物理存储参数和其他一些属性，以提高应用性能。

备份和恢复是日常中必须执行的关键管理任务。备份在那些需要持续运行的系统上更具挑战性。DBMS在某个时间点取一个快照并保存数据。所有的改变都存在流水日志里，这个日志也会定期备份。如果需要恢复系统，DBMS就导入快照，并把记录下的变化集成进去。

安全在数据库管理中是个重要问题。物理安全包括涉及到具体设备的问题，比如自然灾害或硬件的物理失窃。数据备份和灾难计划都是提供物理安全的关键。逻辑安全包括保护数据以免发生未授权的泄漏、未授权的修改和未授权的信息截留。提供逻辑安全的第一步是创建能使计算机识别用户身份的系统。然后应用设计者和用户必须确定每个用户所应该得到的访问权限，分配访问权限，强制进行职责分隔。

数据库常常需要利用加密工具进行保护。当操作系统没法保护数据库文件时，加密就显得格外重要。如果数据必须通过网络（尤其是像因特网那样的开放型网络）传输时，加密也比较有用。

开发漫谈

Miranda很快就会发现DBA的任务和开发者并不相同，但是开发者一定要与DBA紧密配合工作。作为开发者，必须理解数据标准的重要性，也需要在计划、实现和维护数据库应用方面同DBA合作。在应用实现以前，需要建立数据库安全权限和控制。对于项目而言，标识所有用户并规定他们的访问权限，使用查询只赋给他们对所需数据的访问权限，测试你的工作。同时，运行任意的性能监控器或者分析工具。

关键词

每周7天，每天24小时
穷举攻击
计算机辅助软件工程（CASE）
数据库管理
加密
逻辑安全
私有密钥
RSA加密
Shell site
调整
高级加密系统（AES）
认证中心
数据管理
数据库管理员（DBA）
GRANT
元数据
公有密钥
角色
快照
WITH GRANT OPTION
认证
冷战
数据管理员（DA）
灾难应急计划
热站
物理安全
REVOKE
模式
表空间

复习题

1. 数据管理员的角色和目的是什么？

2. 数据库管理员执行什么任务？
3. 有哪些监控数据库性能的工具？
4. 应该监控数据库的哪些方面以避免性能问题？
5. 在开发数据库应用中，DA是如何利用团队的？
6. 商业活动中的主要安全威胁是什么？
7. 如何用热站来保护商业应用？
8. 逻辑安全系统面临着哪三个问题？
9. 在安全系统中鉴别用户身份有哪些基本的方法？
10. 赋给用户的基本数据库特权有哪些？
11. 查询如何提供详细的访问控制？
12. 好的DBMS应用如何提供职责分隔？
13. 为什么在安全数据库中加密是一个重要步骤？
14. 双密钥加密系统如何同时提供安全和认证？

练习

1. 扩展图9-16的例子。给Employee表添加Title（头衔）。创建两个电话本。一个可由所有的雇员访问，并让他们看到除了头衔含有单词“Executive”以外的所有人的电话号码。第二个是给主管的，列出所有雇员的电话号码。
2. 当有人绊倒了计算机的电源线时，DBMS正在记录着一些相关更新事务。描述DBMS保护和恢复数据库所采用的步骤。
3. 简要描述在下列问题中，如何保护计算机系统。
   a. 一个刚被解雇的员工偷走了整个客户表并卖给了竞争对手。
   b. CEO生日蛋糕上的蜡烛点燃了他的PDA，然后他把PDA扔进了垃圾桶。接着大火烧光了数据中心。
   c. 磁盘故障。
   d. 汽车从路上飞出，撞翻了主要电源线；恢复设施的电源需要6个小时。
   e. CFO丢失了手机/PDA，然后有人用存储的密码从他的账户偷钱。
   f. 一个检察官指控你的公司违反了1996年的《健康保险可携带性与责任法》隐私法律。
   g. 主管的助手私自提高供应商的付款，并把多余的钱移进了她的银行账户里。
   h. 一个受雇的程序员用软件通过转换数据到他的个人账户偷了250万美元，现在巴西生活。
   i. 检察官以伪造金融报告而依《萨班斯-奥克斯莱法案》控告CEO和CFO。
   j. 一个利用了DBMS软件漏洞而发起的自动攻击刚刚击跨了数据库服务器。
4. 解释为什么对事务数据库进行每周7天，每天24小时持续访问会使备份和安全更加困难？
5. 当前美国有哪些针对数据隐私的法律？当前欧盟又有哪些针对客户隐私的法规？数据库管理员如果要确保数据的隐私性，需要哪些步骤？
6. 你正在为一个电子商务网站创建安全措施。网站应用下面这些表：

```
Customer(CustomerID, Name, Address,...)
Items(ItemID, Description, ListPrice, QOH)
```

```
Sale(SaleID, CustomerID, SaleDate, IPAddress,
CCNumber,...)
SaleItem(SaleID, ItemID, SalePrice, Quantity)
```

为下列用户对每张表定义访问权限：经理、配送员以及使用网站的客户。定义需要的查询来保证安全条件。

7. 为包括下述表的医师办公室建立一个小型收费数据库：

```
Client(ClientID, Name, Address, DOB, Gender,...)
Staff(StaffID, Name, Specialty,...)
Visit(VisitID, ClientID, DateTime, Charge, InsuranceCompany,
InsuranceID)
TreatmentCode(TreatmentCode, Description, Cost)
Treatment(VisitID, TreatmentCode, Comments, Charge)
Insurance(CompanyID, Contact, Phone, EDIAddress)
```

对下列用户为每张表定义访问权限：医师、会计、办事员（预留）、客户（如果你创建网站）、保险公司。定义需要的查询来保证安全条件。

8. 雇员和其他内部人员往往给公司带来最多的安全问题。简要列出为了保护计算机系统而必须实现的联机基本方针和程序。（提示：研究雇员的雇佣程序。）

Sally的宠物商店

9. 为Sally的宠物商店设计一个安全计划。鉴别不同类的用户，并确定每个组需要的访问级别。为提供需要的安全性可以创建任意查询。
10. 如果不存在，就创建一个使用Customer、Sales、SaleItems、Employee和City表中数据的销售查询，产生一个按州排序的所有销售的报表。用查询分析器来评估这个查询，并找出提高它性能的方法。
11. 创建一个用于Sally的宠物商店的备份和恢复计划。确定所用的技术，谁将负责以及备份的频率。解释随着商店和数据库的增长这个处理过程如何变化。
12. 为保护数据库和硬件需要哪些物理安全方面的控制？

Rolling Thunder自行车

13. 为Rolling Thunder自行车设计一个安全计划。鉴别不同类的用户，并确定每个组需要的访问级别。为提供需要的安全性可以创建任意查询。
14. 为Rolling Thunder自行车设计一个备份和恢复计划。确保指定哪些数据需要备份以及备份的频率。简要列出针对公司的一个基本的灾难应急计划。应用中哪里最有可能存在安全问题？如何分割职责来提高安全性？
15. 对DBMS使用性能分析工具，评估表、查询、表单和报表。给出前5个建议的解释。
16. 分析BikeOrderReportQuery，找出该查询中可以提高性能的部分并修改之。
17. 经理们希望切换到无线网络，甚至还想让系统允许他们使用远程PDA或手机连接数据库和检查进展。为保护这些系统的数据库写一个安全计划。
18. 公司计划建一个网站，使客户可以通过因特网进入并跟踪他们的订单。解释额外需要的安全过程。

参考网站

网站	描述
http://www.dama.org	数据管理企业
http://www.aitp.org	信息技术专业学会
http://www.acm.org/sigsac	美国计算机协会：安全、审计和控制特殊兴趣组
http://www.cert.org	追踪安全专题的因特网企业
http://www.benchmarkresources.com	一个和标准测试数据与性能相关的站点
http://www.databasejournal.com	数据库安全和其他数据库管理专题

补充读物

Bertino, E., B. Catania, E. Ferrari, and P. Perlasca. "A Logical Framework for Reasoning about Access Control Models." *ACM Transactions on Information and System Security (TISSEC)* 6, no. 1 (February 2003), pp. 71–127. [A detailed discussion of various security access complications.]

Brown, L., M. Mohun, J. Deep, and E. Kear. *The Complete Oracle DBA Training Course.* Upper Saddle River, NJ: Prentice Hall, 1999. [Introduction to DBA tasks in Oracle.]

Castano, S., ed. *Database Security.* Reading, MA: Addison Wasley, 1994. [Collection of articles from the Association of Computing Machinery.]

Fayyad, U. "Diving into Databases." *Database Programming & Design,* March 1998, pp. 24–31. [Good summary of OLAP, data warehouses, and future uses of databases.]

Loney, K. *Oracle DBA Handbook.* Berkeley: Osborne, 1997. [One of many Osborne books on Oracle.]

Oracle 7 Server Administrator's Guide. Oracle, 1996. [Available on CD-ROM with the Oracle DBMS. Appendix A describes the technique used to estimate space required for tables, indexes, and rollback logs.]

Stuns, D., B. Thomas, and G. Hobbs. *OCP: Oracle8i DBA Architecture & Administration and Backup & Recovery Study Guide.* Alameda, CA: Sybex, 2000. [You need to read many books to become an Oracle DBA or to pass the certification tests.]

Theriault, M. L., R. Carmichael, and J. Judson. *Oracle DBA 101.* Berkeley: Osborne/McGraw-Hill, 1999. [Good introduction to DBA tasks in Oracle.]

———, and W. Heney. *Oracle Security.* Cambridge, MA: O'Reilly & Associates, 1998. [Setting policies and plans for securing Oracle databases.]

第10章

分布式数据库和因特网

本章学习内容

- 为什么公司需要分布式数据库?
- 分布式数据库产生和加剧了什么问题?
- 客户机/服务器应用的优势和缺陷各是什么?
- 在应用中三层的客户机/服务器模式提供了哪些灵活性?
- 如何在Web服务器上使用数据库创建交互式的因特网应用?
- 在分布式的客户机/服务器应用中会出现哪些问题?

10.1 开发漫谈

Ariel: 新工作如何，Miranda?

Miranda: 非常不错！和其他开发者一起工作很有意思。

Ariel: 你还没对这份工作厌烦啊?

Miranda: 没有啊。我觉得这件事压根儿不会发生——万物都在变化之中。现在他们想让我为销售应用建立一个网站。在网站上，客户可以查看订单信息，可能再加入新的订单。

Ariel: 看来很难啊。我知道一些HTML，可我对如何通过网络访问数据库一无所知。

Miranda: 噢，现在有一些好用的工具。利用SQL，再编一点儿程序，应该不会那么难。

Ariel: 是个很好的机会。如果你学会如何建立能访问数据库的站点，你就在哪儿都能找到工作了。

10.2 简介

当今即使是小型商务应用也使用多台计算机，至少也会有多台个人计算机。更真实的情况是，大多数现代企业都会利用计算机网络的优势，在多台计算机上安装部分数据库和应用。当公司在新地方设办事处时，需要跨越更远的距离共享数据。渐渐地，很多公司越来越觉得同世界各地的人分享数据有用且必要。制造业的公司需要同供应商、发行商和客户联系；服务型公司需要同雇员或合作伙伴分享数据。所有这些情况都是分布式数据库的例子。许多应用都可以借助因特网的能力优势和万维网（WWW）的表示标准。

构建跨网络的应用和管理分布式数据库是很复杂的任务，其目标是对用户提供场地透明

性。用户无需知道数据在哪里存放。这些特性需要一个好的DBMS、一个可靠的网络和一个较大的数据库，还有网络和安全的管理技能。尽管困难重重，一个设计良好的分布式数据库却可以使公司更容易地扩展它的业务。

因特网和万维网渐渐用来共享数据。因特网最基本的功能之一就是定义了用户如何连接服务器以及数据如何显示的一系列标准。通过开发运行在网站上的应用，你就获得了最大的系统兼容性和可访问性。构建网上数据库应用和构建一般的数据库应用很类似，只需要多学一些工具和标准就可以了。

10.3　Sally的宠物商店

数据库已经在Sally的宠物商店运行，而且正在赢利。现在她希望扩张，正在计划开第二家商店。同时筹备建立网站，以便用户订购商品、查看新动物以及获得一些照顾宠物的帮助。首先Sally希望将你的工作变成全职的。她需要有人帮她管理计算机的日常事务、修改程序以及培训其他员工。

Sally把数据库扩展到第二个商店的要求引发了很多问题。她需要随时对两个商店的销售数据都能“实时”访问吗？两个商店之间需要彼此分享数据吗？比如，一种产品在一家商店脱销，Sally需要系统自动去检查另一个吗？商店的操作需要某种独立性吗——销售和财务数据由两个商店分别维护——或者数据总是整合成一体？数据需要怎样更新？只有昨天的库存数据可以吗——还是必须按分钟更新？

这些问题的答案将决定一些至关重要的设计思路。特别地，主要的设计问题就是回答是否有一个核心的或单独的、分布式的数据库控制每家商店的销售。答案取决于商店如何管理，需要数据的类型，网络性能和成本以及DBMS的能力。

许多方法最初、也是最经济的做法就是让第二个商店完全独立。这样，除了在一个账务周期结束后检查一些基本的财务数据外，无需共享。这种方法的第二个优点是易于扩展，因为每个新商店都是独立的。类似的情况是，如果某商店的计算机系统出了故障，也不会影响其他商店。

尽管如此，从某些角度考虑，很可能Sally还是希望数据能更紧密地集成。例如，从其他商店检查库存的能力对用户来说很有用，这也就意味着这个应用将需要从存放在不同商店的若干个数据库中提取信息。这些数据库必须通过电信线路联网。物理地连接计算机有很多方法，你需要学习一门电信课程才能理解不同的选项。一旦计算机物理地连接在一起，就需要处理一些关于创建和管理分布式数据库的额外问题。

因特网迅速成为连接客户和共享信息的首选方法。把数据库连接到网络上是一种非常强大的方法，它可以提供实时数据，并使客户能够找到他们需要的确切信息。要理解如何把数据库连接到网站上，首先学习一些分布式数据库的知识是很有用的。

10.4　分布式数据库

*分布式数据库*系统由多个独立的数据库组成，运行在两个或两个以上通过网络连接在一起并共享数据的计算机上。这些数据库通常都处在不同的物理场地。每个数据库都由一个独立的DBMS控制，这个DBMS确保自己数据库的完整性。极端的情况是，各数据库可能安装在不

同的硬件上，使用不同的操作系统，甚至还可能使用不同商家的DBMS软件。最后一种偶然情况是最难处理的。如果所有的环境都运行在同一个商家的DBMS软件上，那么当前的大多数分布式数据库都会表现得很好。

在图10-1的例子里，公司可能在3个不同的国家都有办公室，每个办公室都有自己的计算机和数据库。大部分数据只在各办公室内部。例如，在美国的员工基本不需看到法国员工的日程安排。另一方面，在法国和英国的员工可能都为同一个大型国际项目工作。网络和分布式数据库让他们能够分享数据，就像所有的信息都在一起似地对待项目。

分布式数据库可能组织成若干种结构。当今最流行的方式包括客户机/服务器方法。在客户机/服务器系统中，服务器计算机更加强大，为许多客户机提供数据。客户计算机通常是带有图形用户界面的个人计算机。客户机就是对用户提供界面，收集和显示数据，并且给适当的服务器返回数据。



图10-1 分布式数据库。每个办公室有自己的硬件和数据库。对于国际项目，不同办公室的员工可以轻易地共享数据。员工不需要知道数据存储在不同的地方

10.4.1 目标和规则

创建一个能充分支持分布式数据库的DBMS是很难的（主要的问题将在后面几节中介绍）。事实上，早期的系统面临着各种问题。因此，一些人就想出了一系列的目标或规则，制定了分布式DBMS所应该拥有的特性。同E. F. Codd一起做过关系数据库方法定义的C. J. Date列出了他感觉比较重要的一些规则。这一节我们就来总结Date的规则。

不管是谁对分布式数据库进行定义，最重要的规则就是用户无需知道或者关注数据库是分布式的。例如，用户应该能够像数据库在一台计算机上一样创建和运行简单查询。在屏幕背后，DBMS可能连接3台不同的计算机，收集数据，并格式化结果。但是用户却不知道这些细节。

作为这个规则的一部分，数据应该独立于场地而存放。例如，如果业务改变，应该可以很简单地把数据从一台计算机移出，并放到另一个办公室去。这种转移不应该影响整个应用，只需应用进行极少的修改就能继续运行。系统不应该依赖中央计算机来调节其他计算机，相反，每台计算机都应该能够按照需要同其他计算机联系。这种独立性提升了系统性能，并且，即使有一台计算机或部分网络出了故障，其他办公室还可以继续运行。

还有一些目标就更加理想化了。DBMS应该独立于硬件和操作系统，这样，当需要更新、更快的计算机时，公司就可以方便地把软件和数据转移到新机器上，并且业务操作和过去一样。类似地，如果系统独立于所在的网络也有好处。大多数情况下，网络都是由不同公司的设备和软件构建而成的。优秀的分布式DBMS应该可以跨越不同的网络工作。最后，如果分布式应用不依赖于某商家的DBMS软件，那就更好了。比如，如果两家公司要合并，而他们只需要安装一个网络连接就可以让所有的应用继续工作，那就太棒了——即使两家公司使用不同的

网络，不同的硬件，以及不同商家的数据库软件。这种理想化的境界目前还不存在。但是，像Oracle这样的系统，提供了针对这些目标的许多组件。

需要这些特性是因为公司在不丢弃现有工作的前提下，可以比较轻易地扩展或修改数据库和应用，通过提供硬件、软件和网络组件的混合，这些目标也使得企业可以选择最适合他们需要的组件。

10.4.2　优点和应用

分布式数据库方法的主要长处就是它配合了企业运作的方式。业务操作常常是跨越场地分布的。例如，工作和数据都通过部门分成很多部分，每个部门的员工都同部门内的其他员工共享大部分数据和通信，但也有一些数据需要同公司的其他部门分享，类似地，较大的公司常常在不同的地理区域拥有办公室，同样，收集来的许多数据都是在本区域内部使用的，但也有一些数据需要同不同区域的员工分享。

共享数据有3种基本架构：(1) 一台中心计算机收集和处理所有数据，(2) 每个办公室都有一个与其他部门不共享数据的独立的计算机系统，(3) 分布式数据库系统。

第二个方案是可行的——只要这个办公室很少需要共享数据。许多情况下，它仍然是常用的方法。需要共享的数据通过纸质报表、传真、电话或电子邮件传送。当然，这些共享数据的方法都很低效。

一些早期的计算机系统使用第一种方案，把所有的数据传送到中央计算机的做法有很多缺点。特别地，把数据都传到一个场地代价很高，如果那个计算机发生故障，整个系统就会瘫痪。

图10-2展示分布式数据库方法的一些优点。首先，分布式系统通过更好地协调企业的需求而提供了重要的性能优势。大部分更新和查询都在本地执行。每个办公室都保留了本地控制，并对数据负责。然后系统能够让任何有适当权限的人，在公司的任何地方按需提取和集合数据。

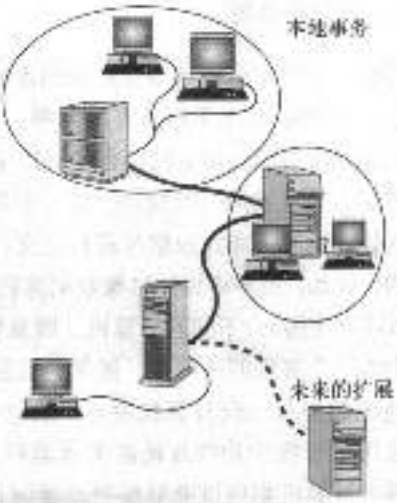


图10-2　分布式数据库的优势。大部分数据都在本地收集和存储，只有需要共享的数据才通过网络进行传输。系统很灵活，因为它可以随着公司的成长慢慢扩展

同集中式系统相比，分布式数据库的第二个优点是更容易扩展。想想如果公司在使用一个大型的、集中式的计算机时会发生什么。如果公司扩展到一个新地区，需要更多的处理能力，那么整个计算机可能被换掉。有了分布式数据库方法，扩展到一个地区只需要增加一台带有支持新操作的数据库的计算机就可以了，所有现存的硬件和应用都能保持原样。通过使用更小的计算机系统，可以更方便和便宜地配合企业变化的需要。

由于分布式数据库方法与任何一家公司的业务布局相匹配，所以有很多应用可以借助它来实现。其中常见的两类就是事务处理和决策支持。在事务处理系统中，每个地区都负责收集它日常用到的详细事务数据。例如，一个制造厂家可能有一个收集和存储购买、人际关系、

产品的数据库。这些数据的大部分被单独的厂家管理。作为共同网络的一部分，每个厂家收集的汇总数据都发回总公司进行分析。另一个例子，考虑一家在多国都有办公室的咨询公司。员工们可以把他们的笔记和注释存在本地数据库里。如果某个国家的客户需要特殊的帮助或遇到独特的问题，本地的合伙人就可以用数据库在世界上其他办公室搜索类似的案例和解决方案。分布式数据库可以让公司的员工分享他们的知识和经验。

10.4.3 创建分布式数据库系统

创建分布式数据库的基本步骤和创建任何数据库应用是类似的。一旦确定了用户需求，开发者们就会通过规范化组织数据，使用SQL创建查询，定义用户接口，并创建整个应用。尽管如此，如图10-3所示，创建分布式数据库还需要一些额外的步骤。需要网络来连接各个场地的计算机。即使网络已经存在，也可能需要经过修改或扩展，才能支持选定的硬件和DBMS软件。

- 设计管理计划
- 选择硬件、DBMS厂商和网络
- 建立网络和DBMS连接
- 选择数据的场地
- 选择复制策略
- 创建备份计划和策略
- 创建本地视图和同义词表
- 执行压力测试：负荷和失败

图10-3 创建分布式数据库的额外步骤。独立的系统和网络安装好以后，需要选择在哪里存放数据。数据可能复制并存放在多个场地。本地视图和同义词表用来提供透明性和安全性。确保对应用进行重度负荷下的压力测试，并且保证它们可以处理网络和远程计算机的失败

另一个关键的步骤就是确定在哪里存储数据。下一节将谈到在分布式数据库上处理查询时可能会碰到的一些问题。现在，只要记得目标是把数据尽可能近得存放到使用最频繁的场地就行了。也有可能把频繁使用的数据复制多份，使得它们可以存储在多台计算机上。当然，接下来你得选择并实现一种策略来确保每份副本都保持最新。

备份和恢复计划在分布式数据库中更加关键。记得不同的计算机可能会运行在不同的场地。每个系统很可能有不同的DBA。然而整个数据库必须进行保护以防失败，所以每个系统都必须有一致性备份和安全计划。开发这些计划需要管理员进行协商，尤其当系统跨越国界或多个时区时。例如，同时备份所有地点的数据事实上是不可能的。

一旦独立的系统安装好并运行起来，每个场地必须创建本地视图、同义词表，存储连接数据库的过程，对适当的用户分发权限，并连接运行在每个系统上的应用。每个独立的连接必须经过测试，整个应用也要在连接性和重度负荷的压力两方面进行测试。同时还应该在网络连接中断或远程计算机失败时测试其行为。

运行和管理一个分布式数据库比管理单独的数据库要困难多了。确定问题的原因就很麻烦。像备份和恢复这样基本的任务都需要所有DBA之间进行协调。一些有用的工具可以使这些工作简化。

不知你是否记得分布式数据库应该对用户透明这个规则？相同的规则却并不适用于DBA或应用开发者。管理员和开发者之间的协调对于使应用更具可访问性是十分关键的。

10.4.4　分布式查询处理

分布式数据库的挑战落到了物理学和经济学。如图10-4所示，存储在本地磁盘上的数据可以以20~60兆字节每秒（或更高）的传输速率传送到CPU。连接到局域网（LAN）存储在服务器上的数据可以以1~10兆字节每秒（10~100兆位每秒）的速率传输，也就是说，局域网的平均传输速率比磁盘直接传输慢10倍。使用连接到广域网（WAN）的公用传输线路只能提供0.01~5兆字节每秒的速率，为获得5兆字节每秒（在T3线路上），你的公司很可能每月至少得付出1万美金。随着技术的进步，这些数字会持续提高，但这种对比关系将保持不变。即，本地磁盘的传输要比局域网快，而局域网比广域网快。尽管如此，注意本地速度的提高已经推动了存储区域网（SAN）的使用。SAN利用千兆位（GB）网络加快了处理机的单独的驱动器速度。尽管存储区域网提供了一些运行服务器的优势，但因为距离有限，还是没能解决分布式数据库的问题。

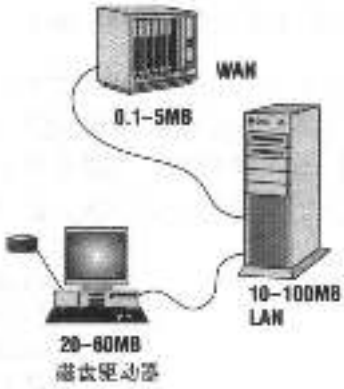


图10-4　网络传输速率。磁盘的传输比局域网快，而局域网又比广域网快。高速的广域网传输要比其他传输方法昂贵得多

分布式处理的目标就是在较慢的网络上最小化数据的传输，降低网络传输的代价。这些目标中的一部分可以通过设计来达到——开发者们必须仔细选择数据在哪里存放。数据应该尽可能近地存放到它使用最频繁的地方。但是，数据需要在多个场地使用时又会有一个折中的问题。

由于查询处理还引发了数据传输的另一个问题。如果一个查询需要从不同的计算机上提取数据，那么传输这些数据并处理查询的时间极大地取决于有多少数据必须传输，以及传输线路的速度如何。因此，其结果取决于DBMS如何从多个表中连接数据。在一些情况下，差异是极其明显的。一种方法可能在数秒内得到结果，同一个查询的另一种不同的方法可能需要处理很多天！理想情况下，DBMS应该对查询、使用的数据库以及传输时间进行估计，确定一个回答查询的最有效的途径。

图10-5表示这个基本的问题。考虑在3个不同数据库上的一些表：（1）在纽约有1百万行记录的表Customer，（2）在洛杉矶有1千万行的表Production以及（3）在芝加哥有2千万行的表Sales。一位在芝加哥的经理要进行下面的查询：

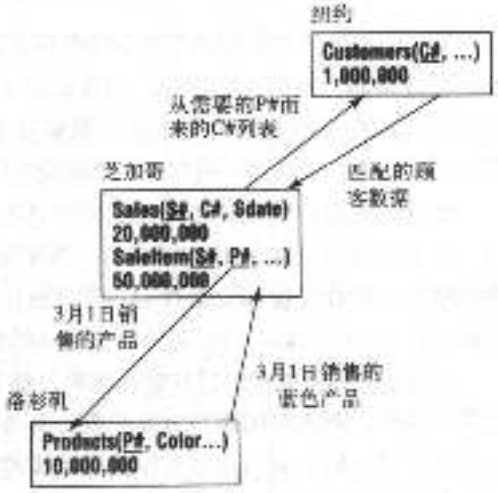


图10-5　分布式数据库查询的例子，列出在3月1日购买了蓝色产品的顾客名单。不好的处理方法把所有的数据都传给芝加哥。优化的目标是限制每一个集合，并传送最少量的数据

列出在3月1日购买了蓝色产品的顾客名单。

这个查询可能会有若干种处理方式。考虑一个坏主意，把所有的数据行都传送到芝加哥；然后连接各表并选择满足查询条件的行。这种方法导致有1100万行数据被传送到芝加哥。即使在相对快速的广域网内，这个查询能在30分钟内得到结果就已经让人吃惊了。

一个好点儿的方法是告诉洛杉矶的数据库找到所有的蓝色产品并把结果行传送到芝加哥。假设仅有一些产品是蓝色的，这种方法就能明显地减少需要传送的数据。性能的提升取决于含有蓝色产品的记录行的百分比。

一个更好的想法是从芝加哥的表中得到3月1日卖出的所有商品列表，这个不需要传输代价。然后把这个结果发给洛杉矶，让那边的数据库确定哪些产品是蓝色的。再把匹配的Customer ID发给纽约的数据库，最后返回对应的顾客数据。

注意，为了最优化查询，DBMS需要知道每张表里数据的一些信息。举个例子，如果在洛杉矶的数据库里有很多蓝色产品而没有多少3月1日的销售数据，数据库就应该从芝加哥发送销售数据到洛杉矶。另一方面，如果没多少蓝色产品，那么从洛杉矶发送产品数据到芝加哥就更为高效了。在一些情况下，系统也需要知道网络连接的传输速度。一个好的DBMS包含着一个查询优化器来检查数据库内容和网络传输速度，以选择回答一个查询的最佳方案。你仍可能需要自己优化一些查询，基本的规则就是尽可能少地传输数据。

10.4.5 数据复制

有时候并没有一个好的办法来最优化查询。当在若干不同场地都需要大的数据集时，复制这些表并在每个场地存储一个备份可能会更加高效。问题在于涉及的数据库必须知道每一份副本。如果一个用户在某地更新了数据，这个改变必须复制到其他所有的副本。DBMS使用复制管理器来确定应该发送哪些改变，并处理每个场地的更新。复制可以在每天特定的时刻自动发送，或者当有人觉得有必要同步数据时手动触发。

开发者和数据库管理员可以通过指定数据库如何复制来调整性能。可以控制改动多久发布一次，以及是分块发送还是随整个表批量传输。最大的困难还是有时候网络连接可能中断或服务器失败。那时DBMS就必须协调数据库，以确保它们都保留表的当前版本，并且不丢失任何改动。

图10-6展示复制的基本概念。每个地方的市场办公室都有来自英国和西班牙的Customer和Sales数据副本。几乎所有的更新都基于本国内部的数据。经理们很可能并不需要其他国家来的实时数据，所以表可以在晚上批量地复制和更新。对于每个地方的经理，数据都是可用的，并且不用考虑传输时间。公司可以通过在非高峰时段执行传输，以最小化国际传输费用。

事务处理数据库通常会记录很多变化，有时每分钟就有几百次。这些应用在处理事务时需要快速的响应时间。通常最好是在本区域内以分布式数据库的方式运行这些系统以提高性能。

另一方面，不同地方的经理常常需要分析事务数据。如果让他们直接访问分布式事务数据库，这种分析查询可能降低事务系统的性能。当前流行的解决方案是把事务数据复制到数据仓库中。例行程序从事务处理系统中提取数据并存放到数据仓库中。经理们运行应用程序并创建查询，从数据仓库中提取并分析数据，制定战术或战略决策。由于经理们几乎不会改变这些数据，数据仓库就非常适合进行复制。原先的事务处理系统保持它的速度，而且不共享

原始数据。经理们分享数据仓库的访问权。

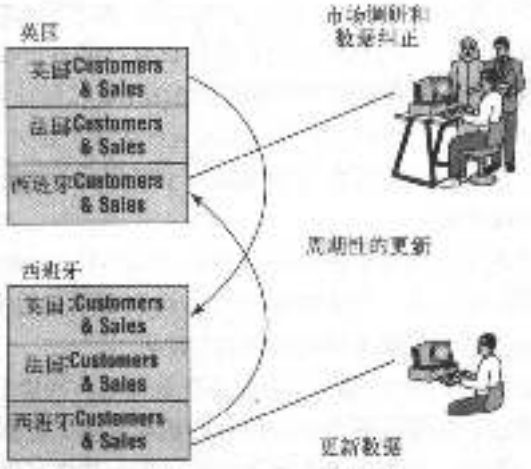


图10-6　复制的数据库。如果经理们不需要其他国家的最新数据，这些表就可以在晚上费用比较低时进行复制和更新传输

自动码生成在数据库复制中是个不小的挑战。如果不同场地的两个人都创建了一个新的客户会如何？如果码生成器不同步，那就很可能造成两个场地获得了相同的码，并且当数据从两个场地进行更新时，冲突就会发生，必须手动来解决。两个常用的方法可以用在分布式数据库里来安全地产生码：（1）随机产生码；（2）场地相关码。如果生成器从一个足够大的码空间里选择码，那么就可以随机生成码，这样两个同时生成的码具有相同值的概率就很小。安全起见，生成器会立即检查刚刚产生的码是否已经存在。第二种方法可以使用序列码或随机码，但依赖于每个分配了一个范围值的场地。例如，一个区域可能分到1~1百万的范围，下个区域1百万~2百万，依此类推。有了场地相关的生成器，必须仔细隔离用于码生成的数据表。例如，你的码表可能包含场地标识符和码的开始或当前值。

10.4.6　并发、锁和事务

并发和死锁在分布式数据库中都是很复杂的问题。当两个人同时试图修改同一份数据时会引发死锁。通过封锁要修改的行可以预防这种情况。如图10-7所示，分布式数据库中出现的问题是应用可能引发涉及独立计算机上不同数据库的死锁。一个用户可能持有一台计算机上某个表的锁，等待另一台计算机上的资源。想象一下，如果死锁涉及到5个场地的5个数据库时会如何。发现死锁问题很困难。当锁发生在一台计算机上时，DBMS可以用锁图来捕获死锁的发生。在分布式数据库环境中，等待资源时DBMS还必须监视延迟。如果延迟太长，

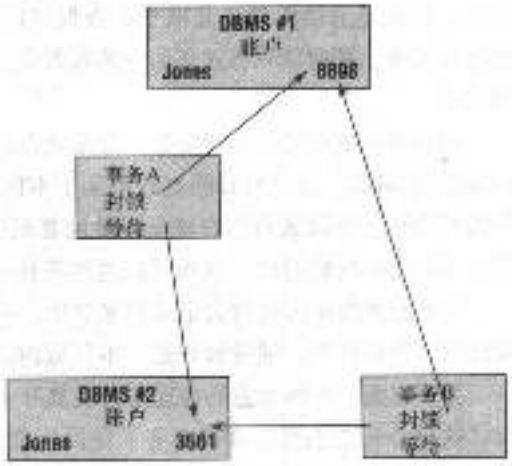


图10-7　并发和死锁在分布式数据库中更为复杂。死锁可以发生在多个不同的数据库中，使得识别和解决死锁变得更加困难

系统假定有死锁发生，并回滚事务。当然，延迟可能缘于很慢的网络连接，所以这个方法也不是很安全。更坏的情况是，消耗在等待上的时间都浪费掉了。在繁忙的系统里，DBMS可能会用更多的时间去等待，而不是处理事务。

处理跨越多个数据库的事务也是一个更加复杂的问题。当更新需要写入若干台计算机时，必须确保所有这些改变都同时成功或失败。到目前为止，检验事务最通用的机制还是两阶段提交过程。图10-8展示了这个过程。发起事务的数据库成为协调者。在第一阶段，它向其他数据库发出更新命令并让它们准备好这个事务。每个数据库都必须针对它自己的状态发送一个应答。每个数据库都必须同意执行整个事务，或按需进行回滚。即使有故障发生，数据库也必须同意进行更新。换句话说，它把更新都写在事务日志中。一旦日志成功创建，远程数据库就同意它处理更新。如果某个数据库碰到了问题并无法完成事务（或许不能锁定某个表），它就发送一个失败信息，然后协调者就会告诉所有的数据库都回滚它们的更新。好的DBMS会自动处理两阶段提交。作为开发者，只需要提交标准的SQL语句，DBMS会处理通信并保证事务成功完成。在一些较弱的系统里，可能还需要把两阶段提交命令嵌入到程序代码中。如果你知道在构建一个使用很多分布式更新的应用，那么为一个能自动处理两阶段提交过程的DBMS准备预算，通常是较好的选择。

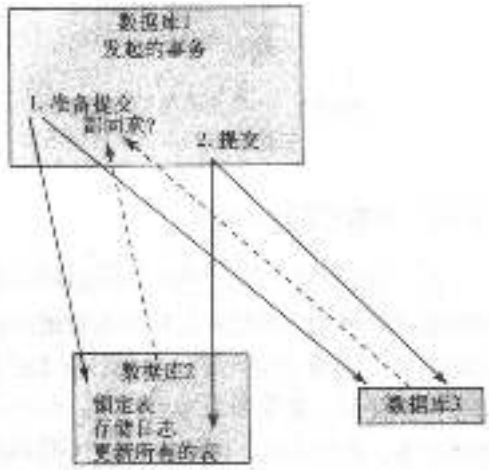


图10-8 两阶段提交。每个数据库必须同意保存所有的更新——即使系统崩溃。当所有系统都准备好时，就会让它们提交更新

注意两阶段提交系统依赖于悲观锁。由于事务的延迟，它可能很大程度上在事务中降低了所有涉及系统的性能——因为它要等待每台计算机的封锁记录。尽管乐观锁在事务的一些方面有所帮助，但当系统或通信链路失败时它还是无能为力。

10.4.7 独立的事务管理器

分布式系统的问题很难得到有效解决，特别当数据库系统来自不同的厂商时。图10-9展示了一个常见方法，使用独立的事务处理监控器或者分布式事务协调器。这个系统是一个独立的软件，协调所有的事务，并基于和本地事务管理器的交互做出提交或放弃的决策。这种方法通常由操作系统厂商提供，而DBMS厂商需要开发和事务管理器通信的接口。主事务管理器可以运行于独立的系统，或者本地事务管理器中的一个也可能提升为协调者。

独立性是事务管理器的主要优势，只要各个厂商提供支持（利用本地资源管理软件），系统就能支持不同的产品。对于程序级别的事务，就是数据库之外还有足够多的代码（比如C++）事务，它也同样有用。通过事务管理器，如果需要，数据库系统可以修改——而不需要重写所有的事务处理元素。

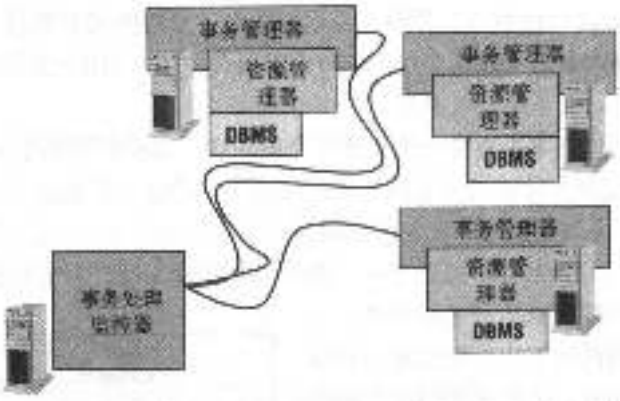


图10-9 分布式事务处理监控器。这个软件处理事务决策，并通过和本地事务管理器的通信来协调各参与系统

10.4.8 分布式设计问题

由于存在传输代价、复制和并发这些问题，分布式数据库需要仔细地进行设计。随着网络传输速率的增加，数据库设计的问题最终会变少。同时，你需要分析你的应用，确定它们该如何分布。图10-10列出了一些设计分布式数据库时可能遇到的问题。关键在于数据库的哪些部分应该复制。如果所有地点的用户都需要数据库保持绝对的一致性，那么复制很可能就是个坏主意。另一方面，你可能有一个对封锁和并发处理得不太好的DBMS。这种情况下，最好复制数据，也比冒着事务不成功时数据被破坏的风险要强。

问题	并发	复制
需要哪个级别的数据一致性?	高	中低
存储器有多贵?	中高	低
什么是权限共享的需求?	全局	本地
多长时间更新一次表?	常常	很少
(事务)要求的更新速度?	快	慢
预测事务时间有多重要?	高	低
DBMS对并发和封锁的支持?	出色	不好
能否避免访问共享?	否	是

图10-10 设计问题。使用这些问题来确定应该复制数据库还是提供通过网络对数据进行并发访问。事务操作通常需要和并发访问配合运行。决策支持系统常常使用复制的数据库。但是，正确选择取决于数据的使用和用户需求

10.5 客户/服务器数据库

当前，在网络上构建用于分布数据（和计算机）的系统时，客户/服务器方法最为流行。在这类系统里，将多用户操作系统强大的计算机作为服务器，而小一点儿的计算机——通常为个人计算机——作为客户端。服务器装有软件和用户共享的数据。客户计算机只保留其使用者

的个人数据。

客户/服务器方法的发展很大程度是由个人计算机操作系统的有限能力所驱使的。早期的操作系统不支持多用户，也不提供安全保护。因此，给处理所有需要共享数据和硬件的服务器安装上强大的操作系统，比起监控数百台PC机，客户/服务器方法比较容易管理和控制。任何需要共享的硬件、软件或数据都存放到一个集中的场地，并由管理信息系统的人员控制。在客户/服务器方法中，要共享的所有数据首先传送到服务器上。

如图10-11所示，客户/服务器数据库以同样的方式运行。实际的数据库驻留在服务器计算机上。独立的组件可以在客户机上运行，但它们在服务器上存储和提取数据。客户端组件通常是和用户交互的前端程序。例如，一种常见的方法就是把数据表存放在服务器上，但在个人计算机上运行表单，表单利用图形化界面处理用户事件，但所有的数据都会传送到服务器上。

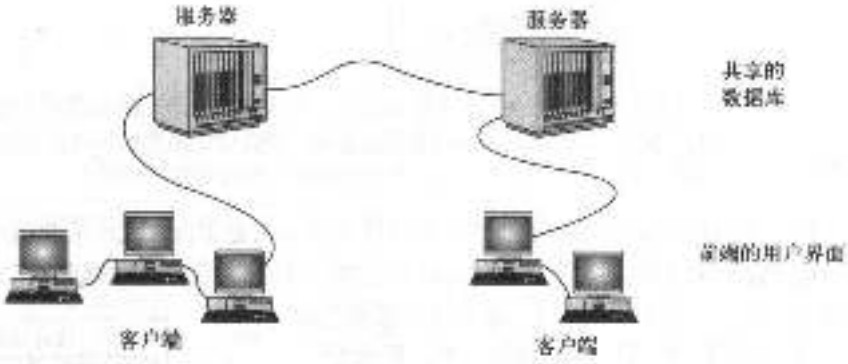


图10-11 客户/服务器系统。客户计算机运行前端的用户界面程序。这些程序在那些运行于服务器计算机上的共享的数据库中存储和提取数据。网络使得客户端可以访问任意服务器上的数据，只要有适当的权限

设计和管理客户/服务器数据库，需要理解一些重要的概念。与任何分布式数据库一样，在哪里存放数据与如何访问可以造就性能完全不同的系统。本节介绍一些现有的用来创建客户/服务器数据库应用的工具。

10.5.1 客户/服务器与文件服务器

为了理解客户/服务器数据库的特性和强大之处，先来看一个不是真正客户/服务器数据库的数据库应用，最初的本地网络是基于文件服务器的。文件服务器是一个用个人计算机共享文件的中央计算机。但是，它不含有数据库。文件服务器存储文件，但对个人计算机来说，它更像一个庞大的、被动的磁盘驱动器。这个服务器惟一的目标就是提供对文件共享的安全访问。客户的个人计算机执行所有的程序处理。

图10-12展示了这个基本的问题。数据库文件存放到文件服务器上，但DBMS自己却运行于客户端，设置了安全许可后，每个用户就有了读和写文件的权限。当运行应用时，表单和你的代码下载到客户计算机上。应用程序运行查询时，问题就出现了。查询的处理是在客户端完成的。这就意味着那台个人计算机必须从服务器上提取数据的每一行，再检查，然后决定是否在计算或显示中使用。如果数据库小，网络连接快，而且如果用户常常希望看到整个表，这种处理就没什么。但是，如果表很大且用户只需要看很小一部分数据，传输整个表到

客户端就是浪费时间和网络带宽了。

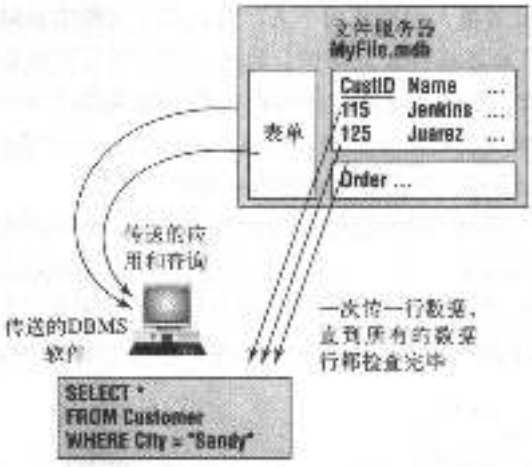


图10-12　文件服务器问题。文件服务器扮演着一个庞大、被动的磁盘驱动器角色。个人计算机执行全部的数据库处理，所以它必须检取和检查数据的每一行。对于较大的表，这个过程很慢，而且浪费网络带宽

问题就在于当应用仅需要部分数据时，文件服务器方法也要传送大量的数据。客户/服务器数据库方法就是设计来解决这个问题的。在客户/服务器数据库中，数据库的二进制代码是运行在服务器上的，如图10-13所示，服务器数据库接收SQL语句，进行处理，然后只返回查询结果。注意它在网络传输上的降低。初始的SQL语句很小，并且只有应用所需的数据才在网络上传输。这个结果对于决策支持系统尤为重要。服务器数据库可能包括几百万行数据。经理正在分析数据并想要些总结性的统计，比如均值。服务器数据库对查询进行优化，计算出结果，然后只给客户端传回一些简单的数字。若没有服务器数据库，几百万行数据将可能通过网络传输。要牢记即使快速局域网的传输速率也比磁盘传输慢很多。

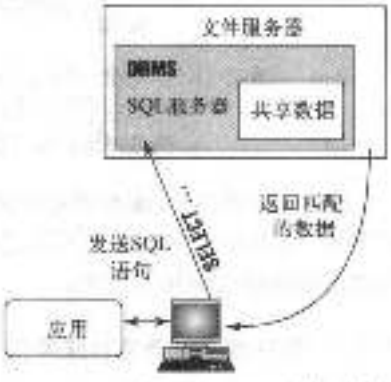


图10-13　数据库服务器。客户计算机发送SQL语句，由服务器进行处理。只给客户端返回结果，降低网络流量

当然，服务器数据库方法也有缺点，即花过多的时间来处理数据。因此，服务器计算机必须配置得可以同时为多个用户有效地执行数据处理。幸运的是，处理器的速度提高比磁盘和网络传输速度的提高要快很多。另一个缺点是这种方法需要购买一个运行在服务器上的强大的DBMS，尽管如此，你几乎没有其他的选择。只有针对少数用户的小型应用才可以不用数据库服务器而运行起来。

10.5.2　三层客户/服务器模型

三层客户/服务器模型的方法已经被认为比两层模型有很多优势。三层方法在客户端和服务器之间加了一层。三层方法在使用若干个数据库并涉及多个应用的系统里尤其有效。这种方法在一些服务器运行遗留的应用时也比较有用。

如图10-14所示，中间层的角色之一就是创建到数据库的连接。如果需要，中间层翻译SQL请求，并从遗留的COBOL程序中提取数据。通过在某地放置访问链接，就可以在不影响客户前端程序的情况下移动或修改服务器数据库。开发者们只需要简单地改变场地指针，或者修改中间件的路由。一些人也把这种方法称作n层方法，因为可以有多台中间级的计算机，每台都确定商业规则中的一个方面。

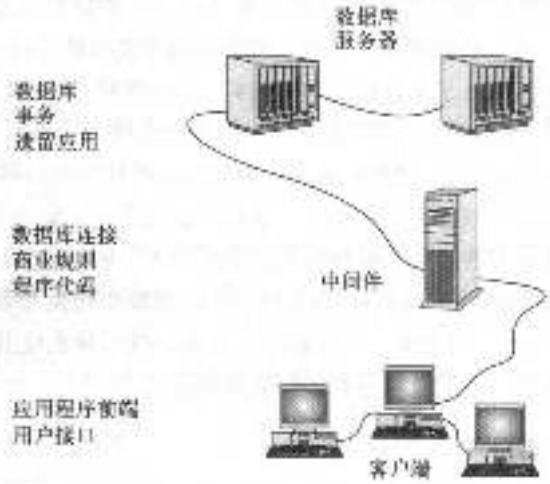


图10-14 三层客户/服务器模型。中间层把数据库和应用里的商业规则和程序代码分离开来。这种独立性可以很容易修改每个组件而不影响其他元素

中间层的另一个重要角色就是维护商业规则。例如，为顾客和产品创建标识应该遵从统一的过程。生成这些数字的例程应该存放在一个场地，然后所有需要它的应用都调用那个函数。类似地，常见的应用函数也可以只写一遍，并存放在中间层服务器上。

这种中间件系统和面向对象的开发方法配合得很好。用于多个商业应用的通用对象可以只写一次，然后存放在中间服务器上。任何应用都可以按需使用那些对象。随着商业规则的变化或系统的更新，开发者可以在不影响客户端应用操作的情况下，修改或更新基类对象。三层方法分离了数据库和应用中的商业规则及程序代码。这种独立性让系统更加灵活，也更容易扩展。近期一些编程工具，比如VB的企业版，推动了中间层的开发。可以用Visual Basic创建中间层对象。它可以从任何服务器上提取数据，或者简单地执行计算，可以创建不以任何形式运行的作为动态链接库（DLL）的组件。把它安装在中间层的计算机上，然后建立客户端应用。有了VB的企业版，你的应用就可以打开和使用任何与之相连的计算机上的组件了——只要设定了合适的权限，可以配置分布式的COM（DCOM）来建立访问权限。

10.5.3 后端：服务器数据库

很多公司都有服务器数据库，而各种产品都有其特定的优缺点。有些必须运行在某种品牌的计算机上；其他可以运行在不同的机器上。幸运的是，大部分主流的DBMS厂商都支持SQL。因此，定义数据库并构造查询可以按你所知的步骤进行。当然，数据库厂商也添加了一些需要学习的新特性。

服务器数据库系统趋向于变得更加复杂，且比基于个人计算机的系统需要更多的管理任

务。服务器环境同时提供了更多的选项，让管理和开发也更加复杂起来。服务器计算机使用更为强大的操作系统来支持多个用户。DBA必须和系统管理员紧密配合，安装软件，定义用户账户以及监控性能。

服务器数据库同时使用触发器过程来定义和确保商业规则。一个更难的设计问题是，必须决定在哪里存储商业规则：是作为数据库触发器和过程存储在后端数据库中，还是使用像C++或Visual Basic这样的低级语言把它们移到中间级服务器上。有时候你会受到可用工具和时间的限制。但只要有可能，应该依据费用、性能和可扩展性考虑各种可行方案。

把过程放到后端数据库保证了不论数据如何访问，所有的规则都由DBMS强制执行。但是，这种方法就把你栓在一个特定的DBMS厂商上了。因为绝大部分系统都有一些非标准的元素，以后想转换到别的DBMS上就难了。把规则放到中间层也使得物理地转移数据库变得更加简单。通常，系统都是使用数据库连接而建立起来的。要转移数据库，只需要简单地改变引用指针。

有一个经验方法，就是为客户计算机编写用户界面代码，同时编写运行在服务器上的数据操作和控制程序。用中间层程序来实现商业规则，并提供数据转换和数据库独立性。其主要目标就是最小化数据在网络上的传输。尽管如此，如果一些计算机比其他的慢得多，为了在快点儿的计算机上执行代码，还需要接受更多的数据传输。

10.5.4　前端：Windows客户端

基于Windows的计算机通常都用作客户机，所以微软已经开发了若干种技术，以提供从PC机到后端数据库的连接。许多不同的工具和厂商都支持这些技术，所以它们也是相对标准化的。随着硬件和网络的提升，应用变得越来越复杂，这些工具也在发展。Visual Basic常常用作前端工具创建表单和报表。程序经过编译，分布到各用户计算机上，可以通过微软的数据组件连接到后台服务器上。PC都有一个网络连接和数据库连接，使得它们可以找到服务器上的中央数据库。VB程序代码只是简单地选择合适的数据库连接。从这点上说，应用已经不再关心数据在哪里存放了——只需把SQL请求传给服务器就行了。

开放式数据库互连（ODBC）是微软早期的用于连接Windows客户端到服务器的技术。一些系统仍然利用这个技术，但它相对较慢。更新的技术是*活动数据对象*（ADO），由于使用较少的层，它更加快速，也提供更多的功能。最新的版本是ADO.NET，含有针对微软.NET环境的一些额外功能。

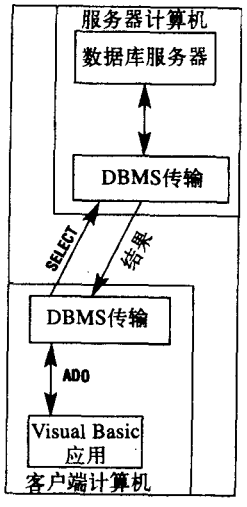


图10-15　活动数据对象连接。在分布式环境中，ADO处理应用程序和基本数据传输层之间的连接。最快的方法就是使用DBMS厂商的数据传输工具，但是没有其他可用机制时也可以使用开放式数据库连接ODBC

ADO运行在各厂商网络系统的顶层；因此它提供对数据更直接的访问。ADO可以利用ODBC来处理文件，也可用于那些没提供自己数据传输方案的系统。图10-15展示了ADO

在分布式环境下的使用，需要安装数据库厂商的数据传输层，然后，应用选择合适的ADO驱动，连接基本的数据库传输服务。ADO把SQL语句传递给传输服务，并对返回数据进行基本的数据传输处理（例如识别日期、整型和金融数据），同时一致地处理数据游标。因此，用ADO工具编写的应用可以访问不同的数据库。

10.5.5 在客户端维护数据库的独立性

分布式数据库最麻烦的地方就是维护数据库的独立性了。当初次建立应用时，它常常与后端的单一的、特定的数据库共同运行。因此，很容易假设那里一直就只有那个数据库，并且使用那个特定系统内可用的工具和快捷键。但以后若有人要改变后端数据库时该如何办？极端情况下，整个应用都要重写。作为一个不成熟的开发者，你可能会问为什么。考虑一下如果有人想在以后改变数据库，他们是否愿意在那时还支付费用？不过，预先只需要很少一些额外付出，应用就可以支持大多数后端常见的数据库，这使以后的更新变得容易。

数据库连接是构建通用程序的一个问题。使用ADO或ODBC这种通用的标准使得改变数据库更加容易。图10-16展示了通过改变连接，应用可以连接到不同的DBMS。当然，绝不是那么简单，但ADO缓存是个重要元素，在很多情况下，你可以动态指定ADO连接串，以方便应用程序在不重写代码的情况下连接不同的DBMS。如果比较仔细，可以创建一个应用，让它能够在任何时候切换DBMS连接。可以使用一个DBMS构建系统，然后让产品系统运行在另一个不同的DBMS上。

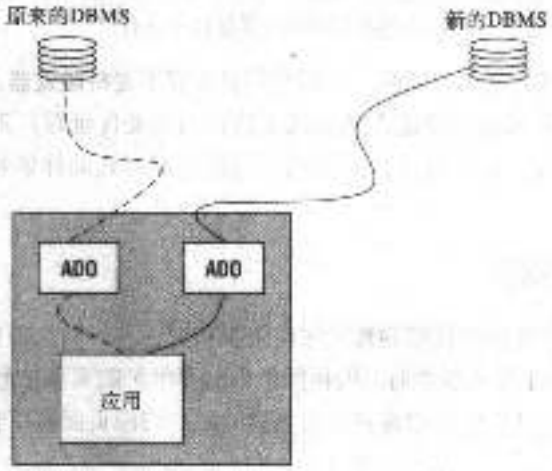


图10-16 数据库独立性。ADO是应用程序和DBMS之间一个有用的缓存。改变连接可以让切换后端DBMS变得相对简单些

在构造一个与DBMS无关的应用时，很重要的一点就是理解连接是惟一的因素。大多数情况下，实际的SQL命令是个更大的问题。DBMS厂商都提供对SQL标准的不同级别的支持。它们也私自增加了一些诱人的命令选项。特别地，厂商在SELECT语句中提供了很多变化。例如，字符串和日期操作在不同的厂商之间严重地不统一。并且，如果使用老版本的Oracle和SQL Server，就不能使用INNER JOIN语法。由于这些差别，让应用独立于DBMS的关键一步乃是把所有对DBMS的查询都存为视图。然后应用只包含从保存的视图中提取数据的简单SELECT

（或者INSERT/UPDATE/DELETE）查询。这些简单的查询应该只使用基本的标准SQL元素。所有与厂商相关的功能都编码成为DBMS内部的查询。要转换到新的DBMS，只需要在新的DBMS上使用厂商指定的工具和语法重新创建查询。这种技术对那些开始较小然后慢慢增长的应用尤为重要。规模较小时，可以使用小型的、便宜的DBMS；但随着用户数的增加和系统需求的增长，需要按比例使用大一些的DBMS。如果细心建立应用查询来保持独立性，转换到新的DBMS就简单多了。图10-17展示了一个使用简单查询来维护DBMS厂商独立性的例子。

```
Generic application query:
SLECT SaleID, SaleDate, CustomerID, CustomerName
FROM SaleCustomer

Saved Oracle query:
SELECT SaleID, SaleDate, CustomerID,
      LastName ||`,'|| FirstName AS CustomerName
From Sale, Customer
WHERE Sale.CustomerID=Customer.CustomerID

Saved SQL Server query:
SELECT SaleID, SaleDate, CustomerID,
      LastName + `,' + FirstName AS CustomerName
FROM Sale INNER JOIN Customer
ON Sale.CustomerID = Customer.CustomerID
```

图10-17　数据库查询独立性。应用只包含了不使用厂商相关功能的简单查询。所有详细的查询都在DBMS内部创建和保存

触发器功能是个更为复杂的问题。一些系统根本就不支持触发器，而那些支持触发器的系统提供的通常也是各不相同的功能。从这点上讲，当前要保证跨厂商兼容性的惟一方法就是避免使用数据库触发器。取而代之，应该把相同的功能写到同样依赖通用查询的中间件代码中去。

10.6　电子商务数据库

在许多方面，电子商务同传统的数据库应用很相似。其不同点来自一些额外的数据和可用性问题。当用户在Web上输入数据时，因特网把你需要维护的重要信息传送到服务器上。例如，你可以保存客户的IP地址、任何把客户导向到你的站点的源页面或广告页面以及支付信息和确认数据。

Web的本质就是让用户作出选择，把他们自己的数据输入到你的表单中。因此，这些表单必须易于使用。这种情况下，标准化就是简易使用的重要方面。随着时间的流逝，网站为处理基本任务已经开发出了标准的格式。你的应用需要追随这些基本格式，这样用户也就可以面对熟悉的任务。

今天，电子商务网站通常包含一个带有价格、描述和图片的产品的数据库。你的应用必须显示商品项的描述，还必须提供帮助顾客找到特定商品的搜索引擎。你同样需要一个购物车，它包含用户临时选中的商品列表。购物车通常开发为一个基本表，它保存着每个客户的商品项列表。你的应用还需要有一个检验页面，使得用户可以对购物车里的商品项进行最终的更

新，收集付款和送货信息，然后记录这个订单。另外，还需要让用户以后再回来，并查看他们订单的状态。

在所有这些用户界面元素上面，网站还需要若干个管理页面。公司里的某人需要更新描述、图片和价格。其他员工需要检查订单，输入送货日期数据，并解决基本的订单问题。市场部希望能监控销售情况，评估广告效益，并检查客户属性。所有这些管理任务都需要额外的表单。并不是从零开始编写一个新的Web应用，事实上，很多公司都选择购买标准的商业软件系统来得到所有这些需要的特性。但对其他很多公司来说，编写客户数据库应用还是很必要的。

由于因特网的快速发展壮大，各种工具仍在开发中。后端数据库的概念和前面章节讨论过的类似。尽管如此，前端的表单和报表都是作为需要连接数据库的Web页面开发的。支持更容易地创建这些页面的一些工具也正在开发中。例如，微软有.NET家族的ASP；IBM有WebSphere产品；Oracle销售OracleWeb工具；还有Sun公司，正在让它的Java和Java数据库连接（JDBC）走向市场。在一本书里覆盖所有这些技术是不可能的，更别说一章了。所以本章只讨论一些通用的概念，和一些基于Web的分布式数据库应用的特殊问题。

10.7 作为客户/服务器系统的Web

万维网（WWW）原来设计为客户/服务器系统。其目标是让人们（物理学家和科研工作者）可以在他们的学院内共享信息。最根本的问题在于每个人都使用不同的硬件和软件——不管是客户端还是服务器——所以解决方案就是定义一系列的标准。这些发展来的标准是Web的核心。它们定义了计算机如何连接，数据如何传输以及如何寻找数据。一些附加的标准定义了数据应该如何存储及显示。客户端运行用于接收和显示数据文件的*浏览器*软件。服务器运行响应请求、找到合适文件并发送所需数据的Web服务器软件。客户端和浏览器都在逐渐变得复杂起来，但图10-18展示了这种方法的本质。

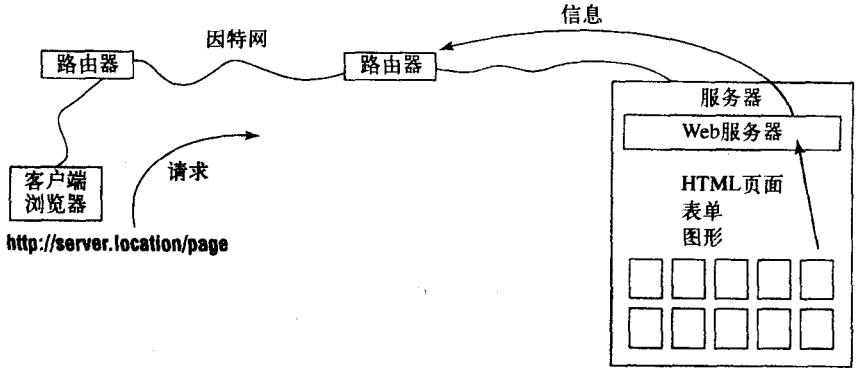


图10-18 Web服务器和客户端浏览器。浏览器是标准化的显示平台。可以从任何浏览器访问服务器

10.7.1 受限的HTML客户端

今天，Web服务器可以绑定到数据库上。数据库能够维护传统数据，或者维护从浏览器接收到的复杂图形和其他对象。要在这个环境下建立应用，必须理解和遵循浏览器显示数据的规则。

这个基本的浏览器规则就是只知道如何显示特定类型的数据，诸如文本和图形。每种数据类型都必须用一种预定义的格式创建。大多数情况下，不允许创建数据的新格式和新类型。浏览器的能力一直在扩展，以便处理新的数据类型，比如声音和视频。但是，应用程序和数据库必须遵循新数据类型的标准。同时，不同浏览器显示信息的方式有很多种。如果希望更多的处于不同环境的人都可以访问你的数据，就必须仔细地避免不兼容问题。

超文本标记语言（HTML）是浏览器显示系统的基础。它是一个简化的页面描述语言。为了最大程度的兼容性，所有发送到浏览器的信息都应该送到HTML页面。在因特网上有很多学习HTML的资源，并且有很多教程手册来帮你学习这门语言。最基本的，就是用标签来控制页面布局和文本属性。**标签**是用尖括号括起来的简短词语，如<B>。HTML使用标签对；例如，要让一个单词成为黑体，可以用一个开始标签和一个结束标签包住它，比如<B>我的文本</B>。一个简单的HTML范例在图10-19显示，其输出见图10-20。

```
<HTML>
<HEAD>
<TITLE>My main page</TITLE></HEAD>
<BODY BACKGROUND="graphics/back0.jpg">
<P>My text goes in paragraphs.</P>
<P>Additional tags set <B>boldface</B> and
<I>Italic</I>
<P>Tables are more complicated and use a set of tags for
rows and columns.</P>
<TABLE BORDER=1>
<TR><TD>First cell</TD><TD>Second cell</TD></TR>
<TR><TD>Next row</TD><TD>Second column</TD>></TR>
</TABLE>
<P>There are form tags to create input forms for
collecting data.
But you need CGI program code to convert and use the
input data</P>
</BODY>
</HTML>
```

图10-19　典型的HTML页面。标签设置页面布局，控制文本格式。数据库应用常常使用标签来创建表格以显示结果。表单标签用于收集和服务器交互所需的数据

My text goes in paragraphs.
Additional tags set **boldface** and *Italic*.
Tables are more complicated and use a set of tags for rows and columns.

First cell	Second cell
Next row	Second column

There are form tags to create input forms for collecting data. But you need CGI program code to convert and use the input data.

图10-20　HTML页面显示

数据库应用常常使用两个特定的标签集：一个创建表，另一个创建表单。表显示数据库的输出，而表单为数据库收集数据，并从用户处获得构建查询的参数。如果希望构建在Web上的数据库应用，应该多学学这些基本的标签。幸运的是，有不少工具可以帮你自动创建HTML页面，所以你不需记住HTML语法。当今大多数文字处理软件都可以存储诸如HTML格式的输入表单这样的文档。

图形的限制更多一点儿。需要把图像存成下列3种格式之一：GIF，JPEG或PNG。现代的图形软件可以处理这种转换。由于在显示图形和数据方面的限制，在开发Web上的应用时，你需要与图形设计者协商。

超文本链接是Web的一个重要特征。每个页面包含到其他页面的引用或链接。链接的基本格式是使用锚标签（<A>）。例如：<A HREF=’newfile.html’>要显示的文本</A>。典型静态页面有着固定的链接，这样每个人都看到了完全一样的页面和链接。数据库使开发者可以创建更为交互的页面和链接。链接（和文本）可以存放到数据库的表里。基于用户的动作，应用可以从表中提取所需的数据，并且针对任意情况都能构建出含链接的页面。一个常见的例子就是订购表单。用户选择一个类别，应用就提取那个类别的商品列表。除了显示列表，每件商品都链接到一个描述页面和一张图片。想看更多信息的用户可以点击那个链接。关键是类别和链接信息都存放在数据库里，这使得检索列表变得很容易，产品经理也能轻易地进行更新。应用只需简单地提取数据并用<A>标签把它格式化就行了。

浏览器的兼容性和标准都在持续地快速发展中。浏览器现在支持你可以用来创建应用的对象。浏览器记录用户事件，让你可以为这些事件绑定代码，这很像一个数据库表单。这种客户端的代码可以记录用户的选择，操纵显示以及与服务器交互。类似地，服务器端代码可以从数据库里提取数据，跟踪用户连接以及维护事务的完整性。当和服务器端DBMS联合时，Web就可以用来创建客户机/服务器应用。使用Web的主要优势在于用户可以从世界上任何地方、使用不同类型的客户计算机来访问应用。

10.7.2 Web服务器数据库基础

连接数据库到Web服务器上没有标准的机制。因此，使用的方法取决于你所安装的特定软件（Web服务器和DBMS）。大多数方法都遵循类似的结构，但在细节方面有所不同。有些现成的工具可以帮你用图形化的设计工具（支持用某种程序设计语言处理数据）来构建表单。然后这些工具会生成发送到客户端浏览器的原始HTML文件。选择工具时要注意的是其中一些工具需要用户为浏览器下载特殊的插件。大多数用户对于下载非标准的组件都很警惕。当你和总公司以外的其他用户打交道时，最好坚持只用那些使用标准浏览器特性的工具。

图10-21展示了连接DBMS到Web服务器的基本过程。里面的数字指示了要发生的3个基本步骤。记得这些步骤有两个方面：客户端和服务器端。（0）首先需要创建表单，这是客户端所要求的。（1）然后用户接收到表单并输入数据。这些数据可能为某个查询条件包含了某些值的限制。例如，用户可能会选择动物的类别和颜色。（2）这个数据会以通用网关接口（CGI）串的形式传回Web服务器。CGI指定了在计算机之间传输数据的格式。这些数据同样告诉Web服务器该打开和传送数据到哪个文件。（3）一个新的页面构造出来并传回给用户。

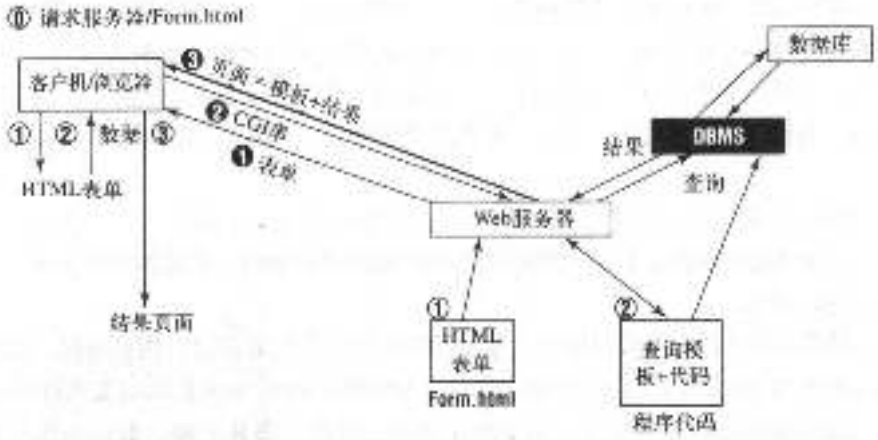


图10-21　Web服务器数据库基础。开发者构建初始表单、查询和显示结果的模板。Web服务器把这些输入数据和查询合并，然后传给DBMS（SQL Server）。查询结果合并到模板，并作为一个新页面发回给用户

客户端方面

在客户端浏览器上用户将看到如图10-22所示表单的简单序列。一旦有人选择了Search选项，那个Animal-Search表单就会显示在浏览器上。用户选择一个类别并输入一种颜色。当点击搜索按钮以后，其选择就发送到服务器上一个新的页面。这个页面提取数据并再格式化成一个新页面。这些数据通常存放在表中。类似于图10-22所示。用户不需要知道关于DBMS的任何事；用户只是看到表单和新页面。每个新页面应该提供额外的选择和到其他页面的链接。



图10-22　客户端方面。客户端在表单里输入数据。点击搜索按钮把数据发送到服务器页面。服务器页面从DBMS里提取匹配的数据并格式化成一个新的HTML页面。这个表返回给用户，还附有一些额外的选项

服务器方面

厂商们忙于创建和改良构建基于Web表单的工具，可以更容易地同数据库交互。选择的工具不同，其细节差异巨大。尽管如此，基本过程都是相似的。服务器从客户表单里得到数据，按你指定的规则进行验证，然后书写SQL查询来插入它们，或更新数据库中的现有行。复习第6章关于构造表单和报表时涉及的所有问题。它与Web之间的主要差异就是，今天的工具通常需要更多的编程技巧，也需要具体情况具体对待。另一方面，基于Web的应用也引起了一些必须解决的潜在难题。数据传输、并发和服务器负荷都是基于Web应用的关键问题。事实上，在厂商工具的重要差异之中，也有一些能从它们解决问题的方式中找到。

10.8 应用中的数据传输问题

乍一看，构建一个客户端/服务器应用好像很简单。只是简单地把数据库移到中央服务器上，并使用网络连接来处理数据传输。但是，客户端/服务器应用可能在数据传输和表单可用性方面有一些颇具挑战性的问题。最难的一个问题就是表单里组合框和选择框的使用。

考虑图10-23展示的一个标准订购表单的主要部分。如果用Access或Visual Basic构建这个表单，并在本地一个快速网络上运行，它会表现得很好。但如果表单运行在一个远程地点，靠50千位每秒的慢速网络连接着，并且有10 000个用户时，又会如何？简化起见，假设每位客户的名字和标识平均有20个字符。所以选择框需要20的10 000倍，或说200 000字节的数据。一个字节是8位，为选择框传输数据需要32秒的时间。任何时候你刷新表单或重新加载组合框时，又得再花32秒。如果你的表单有若干个选择框，表单加载的时间就会更长。如果表单加载超过若干秒，大多数用户都会对性能产生不满。

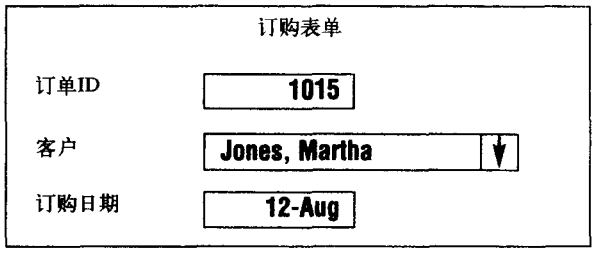


图10-23 表单里的数据传输。如果有10 000个客户时会如何？加载选择框要花多长时间？刷新含有多个选择框的页面需要花多长时间？用户可能如何阅读并滚动这10 000个条目呢？

那么，为什么不干脆移去组合框呢？要理解这个问题，需要记得为什么选择框很有用。在关系数据库里，数据存放在一些独立的表中，而这些表是通过码数据连接的。这个例子里，Order表包含了CustomerID。理论上，实现一次订购简单到只需客户的ID（最终你还会需要商品个体的标识数）。你不能期望你的客户或办事员记住ID号，所以订购表单使用组合框来按字母序列出客户，并为选中的客户返回对应的ID号。如果移去了组合框，就需要重新考虑可用性，并为客户和办事员另找一个输入数据的方法了。

即使没有数据传输问题，当有数千个条目时选择框也不是最好的方案。一些输入框试图

在用户输入名字或产品的前几个字符时，就自动找到匹配的条目，但这种方法仍然需要用户知道前几个字符。因此，一个可能更好的办法是创建更加详细的搜索机制。不用包含所有客户的选择框，用户输入客户姓的前几个字符，点击按钮，并得到一个小点儿的匹配名字列表。

订购条目也显示出类似的问题。有两个常见的解决方案：（1）对于店内的销售，把一个产品ID号绑定到独立商品上（比如条形码）；（2）对于Web销售，让客户搜索商品，在购物车里维持一个选中的ID号集合。

对于仍然想要使用选择框的情况，需要更具有创造性的编程。例如，Oracle不推荐在列表条目多于30时使用选择框。作为替代方案，Oracle建议使用结合值列表（LOV）的标准文本框。值列表可以定义为一个查询，当用户在文本框中输入时，可以从值的列表中选出。这种方法和选择框如何不同呢？主要的区别在表面的背后。LOV以块的形式提取数据，而不是试图一次就传输整个条目集。对用户来说，列表显得是连续的，但通过只传输当前列表显示的选择，LOV减少了传输时间。用户阅读时它也在传输数据，所以时间就更不显眼了。即使你的厂商或工具不直接支持，这种方法也可以使用。在Web表单里，简单地创建第二个表单，它含搜索功能并在多个页面有条目列表。当用户找到特定条目时，可以写一个函数把选中的条目传送到主表单。

由于类似的数据传输原因，Web表单里的并发也是一个问题。在工程术语中，等待时间是系统中的时间延迟。在表单环境中，等待时间是从生成表单到从用户处得到响应之间的时间，如果等待时间很长，就很可能有其他人修改同一个数据元素，所以并发是一个大问题，如图10-24所示，在Web上等待时间一般都比较长，因为传输线路很慢，也因为用户不经意地使用浏览器在提交表单之前还要处理其他任务。因此，基于Web的应用应该避免悲观锁，这样，数据就可以同时为更多的人所用。所以，当数据被另一个进程改变时，你的应用必须测试并处理乐观的并发问题。

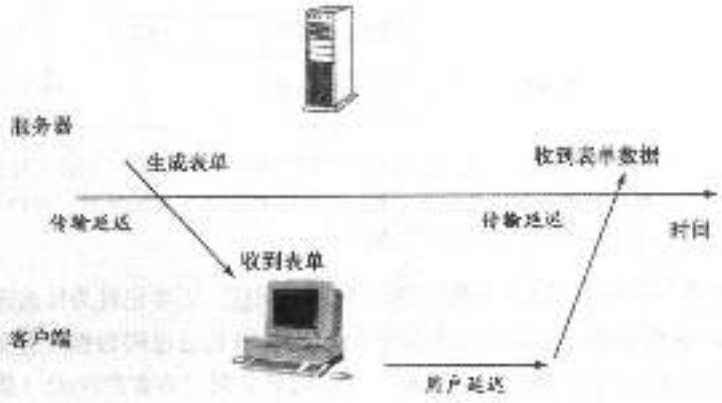


图10-24　等待时间。在Web表单中，传输延迟和用户延迟可能引起较长的等待时间。避免悲观锁，并在基本的数据库改变时进行仔细的测试

10.9 XML：将数据传输到不同的系统

由于因特网的兴起，各企业对于把数据传送给其他公司管理的计算机这个需求增多。例如，一个零售商可能希望使用电子数据交换（EDI）给供应商发送订单。这些传送的挑战在于，两个公司通常拥有不同的硬件、不同的网络以及不同的软件。特定的EDI软件可以处理许多这类问题，但所有的数据传输都必须属于预定义的类别。

把数据传送给一个未知的计算机的挑战首先在于，机器以不同的方式处理原始数据。例如，简单数字的内部二进制存储在基于Intel的个人计算机、基于Motorola的个人计算机（苹果机）和大型的IBM计算机之间差异很大。所以不能传输二进制（比如数据库）文件，希望它们在不同的机器上工作。必须提取出原始数据并把它转化为文本。例如，并不是以二字节的数据元素（0x55 0xE4）传输一个整数，而是把它转换成它的文本表示（21988）。

即使文本传输也会引起问题。比如，和个人计算机用得最多的ASCII码比起来，IBM机器使用不同的字母表/数字系统（EBCDIC）。如果需要传输到其他语言，数据就应该转化为Unicode字符。幸运的是，大多数计算机都可以轻易地在这些标准之间转换文本数据。在建立这些系统来确保每个人都知道文本数据的格式时，只需小心从事就行了。

尽管如此，即使用文本传输数据也不能解决所有问题。简单的文本数据文件并不包含元数据：和文件内容相关的信息。例如，如果你接收到一个数字表，如何知道每一列的意思？列标题可能会有所帮助，但它可能并不包含足够的信息，并且更加复杂的数据传输可能更难以理解。考虑一下图10-25显示的常见订单。它包含单独的订单行，以及为每件订购商品的多个元素。

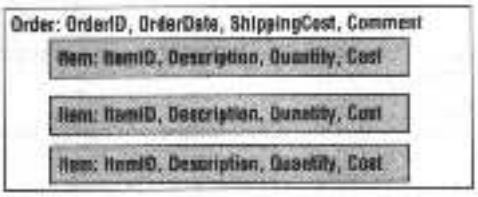


图10-25 层次型的数据文件。注意这个文件的结构：基本的订购日期，然后是订购商品的重复元素，每个订购商品都包含ItemID（商品ID）、Description（描述）、Quantity（数量）和Cost（费用）

这种可扩展标记语言（XML）最初创建来解决这些问题。在某些方面，XML和HTML类似，都是从一个更复杂的叫做标准通用标记语言（SGML）衍生而来的。但是，HTML设计来显示数据，而XML设计来传输数据。XML使用标签来指示数据的作用。XML的强大之处就在于你可以创建所需的任意标签。

当给其他人传输数据时，指定文件的布局，并提供描述数据文件中每一元素的元数据也是很重要的。XML的模式定义（XSD）文件设计来展示XML文件的全局布局。图10-26展示了一个为基本的订单而设计的xsd文件的一部分。这个版本是用微软.NET的xsd.exe工具自动生成的，你可以得到更简明的定义。要注意的是，定义文件展示了数据的布局，并提供了描述每个域元素的元数据。

一个含有宠物商店货物订单示例数据的XML文件如图10-27所示，OrderList是整个文档的

根标签，指明这个文件可能包含不止一个订单，但这里只显示了文件的一部分。每个订单都包含若干个描述订购的元素（OrderDate（订购日期）、ShippingCost（送货费用）和Comment（注释））。每个订单可能包含若干个将要订购商品的列表。每个商品都有自己的标识列表（ItemID（商品ID）、Description（描述）、Quantity（数量）和Cost（费用））。这些商品的数据都包含在文件中，并由适当的开、闭标签括了起来。

可以使用微软的IE浏览器来检查XML文件的内容。如图10-28所示，浏览器提供数据的一个简单层次视图。点击显示内容左边的+/−标记将展开或收缩层次列表。基本的浏览器仅仅提供原始数据的一个简单视图。你绝对不会利用它给用户真实地展现数据。取而代之，可以构建XSL样式单来指定每个元素的布局和格式。但是，这个简易视图是检查要发送到其他公司的XML文件中原始数据的简单方法。

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="OrderList" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft.com:xml-msdata">
 <xs:element name="OrderList" msdata:IsDataSet="true">
  <xs:complexType>
   <xs:choice maxOccurs="unbounded">
    <xs:element name="Order">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="OrderID" type="xs:string" minOccurs="0"/>
       <xs:element name="OrderDate" type="xs:date" minOccurs="0"/>
       <xs:element name="ShippingCost" type="xs:string" minOccurs="0"/>
       <xs:element name="Comment" type="xs:string" minOccurs="0"/>
       <xs:element name="Items" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="ItemID" nillable="true" minOccurs="0"
          maxOccurs="unbounded">
           <xs:complexType>
            <xs:simpleContent msdata:ColumnName="ItemID_Text"
            msdata:Ordinal="0">
             <xs:extension base="xs:string">
             </xs:extension>
            </xs:simpleContent>
           </xs:complexType>
          </xs:element>
          <xs:element name="Description" nillable="true" minOccurs="0"
          maxOccurs="unbounded">
           <xs:complexType>
            <xs:simpleContent msdata:ColumnName="Description_Text"
            msdata:Ordinal="0">
             <xs:extension base="xs:string">
             </xs:extension>
            </xs:simpleContent>
           </xs:complexType>
          </xs:element>
```

图10-26　部分XSD文件。模式定义描述了XML数据文件的布局以及将要传送数据的类型。注意这种层次型的布局和独立的域定义

```
<?xml version="1.0"?>
<!DOCTYPE OrderList SYSTEM "orderlist.dtd">
<OrderList>
<Order>
<OrderID>1</OrderID>
<OrderDate>3/6/2004</OrderDate>
<ShippingCost>$33.54</ShippingCost>
<Comment>Need immediately.</Comment>
<Items>
<ItemID>30</ItemID>
<Description>Flea Collar-Dog-Medium</Description>
<Quantity>208</Quantity>
<Cost>$4.42</Cost>
<ItemID>27</ItemID>
<Description>Aquarium Filter &amp;amp; Pump</Description>
<Quantity>8</Quantity>
<Cost>$24.65</Cost>
</Items>
</Order>
</OrderList>
```

图10-27 XML文件示例。XML文件包含实际的数据。每个元素都由一个开标签和一个闭标签括起来

```
<?xml version="1.0"?>
<!DOCTYPE OrderList (View Source for full doctype...)>
- <OrderList>
  - <Order>
      <OrderID>1</OrderID>
      <OrderDate>3/6/2004</OrderDate>
      <ShippingCost>$33.54</ShippingCost>
      <Comment>Need immediately.</Comment>
    - <Items>
        <ItemID>30</ItemID>
        <Description>Flea Collar-Dog-Medium</Description>
        <Quantity>208</Quantity>
        <Cost>$4.42</Cost>
        <ItemID>27</ItemID>
        <Description>Aquarium Filter & Pump</Description>
        <Quantity>8</Quantity>
        <Cost>$24.65</Cost>
      </Items>
    </Order>
  +<Order>
  +<Order>
</OrderList>
```

图10-28 在IE浏览器里的XML文件。浏览器提供XML文件中原始数据的一个层次视图。也可以构建XSL样式单来改善显示

XML的优势在于可以相对容易地构造出解析器提取数据，并在其他应用中使用。事实上，有些公司提供通用的XML解析器。你的应用代码可以激活解析器，并告诉它自动提取数据的

不同部分。XML另一个强大的方面是有数量庞大的厂商支持。它也是下一版本的SQL标准（200x）要增加的主要内容。因此，许多工具和应用都已经支持从内部数据格式到XML的来回转换。面对越来越多的任务，无需理解XML。只需要简单地告诉某个应用生成XML和XSD文件，把它们传送到其他系统，并让它正确读取数据就行了。在XSD文件的驱使下，所有这些转换都是自动的。

10.10　Java语言和JDBC

Java是由Sun Microsystems公司开发的语言，基于Web的操作系统广泛采用。其中一个目标就是提供一个可以用于任何常见硬件的软件平台。作为这个环境中的一部分，JDBC引入来处理Java和商业数据库之间的交互。解释一下，JDBC并不是只取首字母的缩写，而是一个商标的名字，但人们常常用它来指代Java数据库连接。JDBC是Java 2企业版（J2EE）的重要组件，它是构建数据驱动网站的一种基于服务器的平台技术。

如图10-29中一个小段引用所示，JDBC的结构和诸如ADO等数据库语言类似：代码必须首先建立到数据库的连接。接下来创建包含SQL查询的语句。然后编写一个基于游标的循环来逐行地处理查询结果（ResultSet）。基本的JDBC命令将完成从查询到Java变量中的数据传输。

```
Connection con = DriverManager.getConnection(
                        "jdbc.myDriver:myDBName",
                        "myLogin",
                        "myPassword");
Statement smt = con.CreateStatement();
ResultSet rst = smt.executeQuery(
                  "SELECT AnimalID, Name, Category, Breed
                  FROM Animal");

while (rst.next()){
            int iAnimal = rst.getInt("AnimalID");
            String sName = rst.getString("Name");
            String sCategory = rst.getString("Category");
            String sBreed = rst.getString("Breed");

\\ Now do something with these four variables
}
```

图10-29　JDBC代码的结构。全局的结构和特性与微软的ADO方法类似

当然，需要更多的安装和编程来让这个例子运行。比如，需要安装Java的全部、JDBC以及连接到希望使用的数据库的JDBC驱动程序。这个例子的目的是给你展示其代码的结构和本书其他章节所用的代码结构是相同的。虽然语法和命令不同，但其逻辑是一样的。

小结

随着企业的壮大，分布式数据库变得有用起来。分布式数据库可以让公司扩展部门而不直

接影响其他人。分布式数据库也给了各个部门对他们的数据的更多控制和责任。

尽管如此，带有运行在不同场地的独立数据库引擎的分布式数据库增加了开发和维护应用的复杂性。分布式数据库的一个主要目标就是让数据的场地对用户透明。为了达到这个目标，开发者和DBA需要仔细地定义数据库、网络和应用。

一些分布式数据库引起的复杂性是查询优化，数据复制问题以及对事务、并发控制和死锁解决的支持。这些问题在涉及多个数据库时会变得更加复杂。数据的网络传输比本地磁盘传输慢很多。在广域网上的传输可能又慢又贵钱。这些因素意味着开发者必须仔细设计应用和数据分布策略。应用还必须测试，并接受性能和成本的监控。

设计和控制分布式数据库的一个主要策略是复制数据。与维护一个源不同，在多个场地中复制那些使用频繁的数据常常更加高效。当然，复制需要额外的磁盘空间，以及周期性的升级和把数据改变传到每一份副本上。复制通过提供数据的本地访问节省了时间。它通过降低对全时高速连接的需求而降低了成本。作为替代，大块数据以一个规则的间隔传输——且更趋向于在非高峰通信阶段传输。

客户端/服务器网络和客户端/服务器数据库是设计应用和分布式数据库的常见方法。客户端通常在个人计算机上运行应用，它们的大部分功能是在用户界面上。数据由有限数目的数据库服务器来维护，这比简单的文件服务器传输要高效。在服务器数据库中，客户端发送SQL查询，然后服务器处理这个查询并只返回需要的数据。在文件服务器中，客户端计算机执行所有的处理，而且必须提取和检查所有的数据。

更大的、面向对象的应用一般使用三层的客户机/服务器架构来创建。附加的层在中间，并包含商业规则和运行在服务器上的程序代码（商务对象）。中间层也要负责从数据库服务器中提取数据并根据客户端的使用重新格式化。这三层的分离使得不影响其他元素而修改任意组件变得更容易了。

万维网正在成为创建客户端/服务器应用的流行机制。客户端只有有限的功能，但标准使得每个人访问应用和数据都变得更容易。所有Web工具的功能都迅速增强，让开发者扩展他们的应用更加容易。工具和应用支持XML的话，也会使不同公司、机器和应用间传输数据变得更加简单。

开发漫谈

像Miranda一样，大多数开发者都理解Web的重要性。客户端的标准使分布数据与全世界的用户建立连接变得更加容易。另外，随着应用的扩展，有必要创建一个分布式数据库来提高性能和支持不同的区域。分布式数据库很大程度上也增加了应用开发的难度。首先，确保应用运行在一台计算机上。然后，使用你能负担的最好的软件，尽可能地让服务器数据库执行数据操纵和计算任务，使用客户端计算机显示结果。学习尽可能多的因特网知识：它一直在变，但对你的应用也越来越重要。对于你的课堂项目，你应该找出那个公司可能在什么地方扩展，以及你将把计算机分布到那里来支持它。解释数据库设计在分布式环境中如何进行更新。

关键词

活动数据对象	锚标签	浏览器	客户机/服务器
通用网关接口	分布式数据库	电子数据交换	可扩展标记语言
超文本链接	Java语言	Java 2企业版	JDBC
等待时间	值列表	局域网	开放式数据库互连
复制	复制管理器	三层客户端/服务器	
两阶段提交	广域网	万维网	

复习题

1. 分布式数据库的优势和缺陷各是什么？
2. 让分布式数据库对用户透明需要哪些特性？
3. 为什么在分布式数据库上的查询可能运行很长时间？
4. 在分布式数据库中，什么时候需要复制数据？
5. 为什么并发在分布式数据库中比在单一数据库中问题更大？
6. 两阶段提交过程如何工作？
7. 为什么客户机/服务器数据库比简单的文件服务器上的数据库更为有效？
8. 哪些工具有助于连接不同厂商的数据库？
9. 三层的客户机/服务器方法都有什么优势？
10. 因特网上的Web客户端都有哪些功能？
11. 什么是XML？它在数据传输中有什么作用？

练习

1. 探索那些用于构建三层客户机/服务器应用的软件。描述这些软件的功能。解释各种组件如何分配到每一层上。例如，考虑VB企业版或者J2EE。
2. 你拥有下面的分布式数据库：

地　点	连接速度	表	大　小
波士顿	53kbps	Part(PartID, Description, Size, ListPrice) Order(OrderID, OrderDate, DateShipped, DateReceived) OrderPart(OrderID, PartID, Quantity, SalePrice	1 500 行 300 000 行 3 000 000 行
西雅图	1.544mbps	Part(PartID, Description, Size, ListPrice) Order(OrderID, OrderDate, DateShipped, DateReceived) OrderPart(OrderID, PartID, Quantity, SalePrice)	1 000 行 200 000 行 2 000 000 行
迈阿密	128kbps	Item(BarCode, Description, ListPrice) Sales(SaleID, SaleDate, SalesTax, RegisterID) SaleItem(TransactionID, SaleID, BarCode)	70 000 行 1 000 000 行 40 000 000 行
丹佛 (HQ)	local	Assembly(EmployeeID, BarCodeID, PartID, DateTime) Shipment(ShipID, CustomerID, ShipDate) ShipItem(ShipID, BarCode, Quantity)	1 000 000 行 50 000 行 10 000 行

你为一家供应商在波士顿和西雅图、主要客户在迈阿密的公司工作。供应商的系统可以让你发送XML查询以及当部分到达时提取数据。同样，客户提供对主销售数据库的访

问，这样你就可以检查商品的销售情况。你的CEO注意到一种商品在过去10天显然集中返还给客户了若干次，她想让你找出哪些雇员和供应商与那个特定商品有关。基于通信速度和表的大小，设计回答问题的最好查询。可以通过改变分布式设计来提高数据库和查询的性能吗？

3. 一个公司在同一个州有两个相距100英里的卫星办公室。主要的数据处理是在1 000英里外的一个数据中心进行的。两个办公室都用T1（1.544 mbps）线路连接到主站点。经理想让你开发一个给两个办公室的雇员用的数据库。这个数据库将跟踪客户的交互。每个本地雇员（每个地点大约10个）将跟踪他/她的客户（在任何时间点每个雇员大约有50个活动客户），并且数据每个月经理复查一次。这个数据库最好的分布式设计是什么？
4. 你被一家在若干个不同的州都有工厂的企业聘为顾问。情况比较好，这些工厂大都用1.544 mbps的T1线路连接着，有一些是用128 kbps的ISDN连接着。应用程序维护着每个工厂业务的详细数据，还有价格、规则以及主管创建的过程。这个应用对每个工厂执行复杂的处理，并生成大量的同时被工厂和总部专家使用的数据。此公司希望这个应用运行在Oracle数据库上，并用Visual Basic处理过程。建立这个数据库的基本结构。哪些部分应该分布？哪些部分应该集中？
5. 对于分布式系统来说，把所有数据都存在一个主站点、所有客户端都通过Web浏览器访问的集中式网站有哪些优缺点？

Sally的宠物商店

6. Sally计划增加第二个商店。写一份描述数据如何共享的计划。你如何控制和监视新系统？你将添加哪些工具？
7. Sally想连接到一些饲养站点，以便她可以获得有关动物的实时信息——包括健康和谱系信息。解释你将如何建立系统来实现这种数据共享。
8. 为宠物商店定义一个三层系统。特别是，你将把哪些商业规则和应用存放到中间层？
9. 创建一个网站，这样Sally可以让潜在的客户搜索特定的动物。
10. 扩展数据库，为Sally的好客户们创建网站。当客户每年的购买额超过1 000美元时，他们将收到一个电子邮件信息，指向一个特殊网站，在那里他们可以查看最近的购买，以及订购新的商品。

Rolling Thunder自行车

11. Rolling Thunder计划扩张到跨国的第二个站点。数据库应该如何分布？每张表应该存到哪里？哪些表应该复制以及如何保证数据更新的一致性？
12. Rolling Thunder计划通过向全国不同的自行车商店派出销售代表来进行扩展。他们将拥有笔记本电脑来配置和接收新订单。但大多数自行车商店没有因特网连接，所以系统必须能离线工作。描述一下这种系统将如何工作。需要哪些安全防备？
13. 如果你有一个三层的客户机/服务器系统，描述一下你将在每一地点（客户端、服务器、中间件）存放哪些组件。证实并检查你的选择。
14. 构建一张Web表单，让客户可以检查他们自行车的订购进程。
15. 把与购买相关的表和数据库分开，并把它们移到第二个服务器的一个新数据库里去。修改或重建购买和制造表单，以便它们可以继续在原来的计算机上工作。

参考网站

网站	描述
http://www.w3.org/	Web的标准体
http://www.stardeveloper.com/	数据库的通用信息，特别是关于Java和微软ASP
http://msdn.microsoft.com	搜索微软.NET框架和文档的信息
http://www.xml.org	XML的标准企业
http://www.xml.com	O'Reilly关于XML的站点
http://www.w3.org/TR/xquery	一种XML查询语言
http://java.sun.com/products/jdbc	Java和JDBC的文档及参考资料

补充读物

Burns R., D. Long, and R. Rees. "Consistency and Locking for Distributing Updates to Web Servers Using a File System." *ACM SIGMETRICS Performance Evaluation Review* 28, no. 2 (September 2000), pp. 15–21. [Performance issues in replicated databases.]

Date, C. J. *An Introduction to Database Systems,* 8th ed. Reading, MA: Addison-Wesley, 2003. [In-depth discussion of distributed databases.]

Fortier, P. J. *Database Systems Handbook.* Burr Ridge, IL: McGraw-Hill, 1996. [Technical discussion on building applications using multiple database systems.]

Holzner, S. *Inside XML.* Indianapolis: New Riders/MTP, 2000. [One of several books describing XML.]

Simon, E. *Distributed Information Systems: From Client/Server to Distributed Multimedia.* Burr Ridge, IL: McGraw-Hill, 1996. [General but technical discussion on building distributed systems.]

White, S., M. Fisher, R. Cattell, G. Hamilton, and M. Hapner. *JDBC API Tutorial and Reference,* 2nd ed. Boston: Addison-Wesley, 1999. [Details of Java database access connectivity.]

词 汇 表

24-7：每周7天，每天24小时——应用或数据库每周运行7天，每天运行24小时。由于数据库不可能关闭，所以性能维护很有挑战性。

abstract data type：抽象数据类型——在SQL1999中，具有定义支持继承性存储对象的更复杂数据域的功能。

accessibility：可访问性——应用的设计目标，使应用可以尽可能多地由用户使用，包括身体有残疾的人。解决方案就是支持多种输入、输出方法。

ACID transactions：ACID事务——这个关于事务的首字母缩写词指出安全事务所需的4个要素：原子性、一致性、隔离性和持久性。

active data object（ADO）：活动数据对象——微软用于把程序代码和Web服务器小程序连接到数据库的组件（COM）方法。提供了对任何数据库进行SQL语句和基于行的访问。

active server pages（ASP）：动态服务器页面——微软的Web页面，支持在服务器上运行小程序。有助于对因特网用户提供服务器数据库的访问。

Advanced Encryption System（AES）：高级加密系统——一个用来替代DES，基于Belgian加密系统（Rijndael）的单密钥加密系统。它支持128、192和256位的密钥长度，从而比DES更安全。

aesthetics：美学——应用的设计目标，利用布局、颜色和艺术来改善应用的外观——别不重视。从本质上讲，任何设计的价值都是主观的。

aggregation：聚集（运算）——在若干选中的数据行上进行操作的SQL函数的总称。常见的例子包括SUM、COUNT和AVERAGE。

aggregation association：聚集关联——一种关系，表示个体项成为新类的一个元素。例如，订单包含商品。在UML中，这种关系用一个小的空心菱形在关联端表示。参见Composition association。

alias：别名——表或列的临时名字。常常用于需要多次引用相同的表时，例如自反连接。

ALL——常用于子查询的SQL SELECT子句。如果用于WHERE子句，则匹配列表中的所有项。例如，“Price > All（…）”表示只选择比列表中价格最高的还要高的数据行。

ALTER TABLE——SQL数据定义命令，用于改变表的结构。为了提高性能，一些系统限制了对增加新列的更改。在这些情况下，要做重大的修改，就只能创建新的表，并把老数据复制过来。

anchor tag：锚标签——指示链接的HTML标签。用<A>表示。

ANY——常用于子查询的SQL SELECT子句。如果用于WHERE子句，则至少匹配列表中的一项。例如，“Price > ANY（…）”表示只要价格比列表中的某一项高就是满足条件的数据行。

application：应用（应用程序）——一个执行特定任务集合的完整系统。一般情况下，它包括集成的表单和报表以及菜单和帮助系统。

Application Design Guide：应用设计指导——在构建应用时应该遵循的标准设计概念集合。这些标准使用户操作新应用变得更简单，因为他们在一个系统中学到的技术也适用于另一个系统。

application generator：应用生成器——一个帮助开发者创建完整应用包的DBMS工具。常见的工具包括菜单和工具栏生成器以及集成的上下文帮助系统。

association：关联——类和实体之间的联系。通常，它们表示商业规则。例如，一次订购可以由一个客户完成。区分关联是一对一、一对多还是多对多，非常重要。

association role：关联角色——指UML中关联绑定到类上的那一点。它可以有名字，一般显示重复度、聚集或组合等。

association rules：关联规则——一种数据挖掘技术，用于检查一系列事务来发现哪些项经常被一起购买。

atomicity：原子性——事务的要素之一，它指定事务中所有的操作必须同时成功或同时失败。

attribute：属性——实体的特征或属性。在数据表中属性可能是一列。雇员的属性可能包括姓名、地址、雇佣日期和电话等。

authentication：认证——一个验证系统，用来确定究竟是谁写了消息。常见的系统使用双密钥加密机制。

autonumber：自动标识——一种数据类型域，DBMS自动为每一行新数据都分配一个惟一的标识。用来帮助生成关系表的主码。

B+-tree：B+树——一种索引型的数据存储方法，对很大范围的数据访问比较有效。树的检索提供了稳定的性能，而且不受数据库大小的影响。

base table：基本表——与一个基本实体相关的数据表。它通常不包括外码，所以数据可以不参考其他表而输入进来。例如，客户可能就是一个基本表，而订单就不是。

BETWEEN——SQL比较操作符，用来确定某项是否落到了两个值之间。对日期操作很有用。

binary large object (BLOB)：二进制大对象——大块未定义数据的数据域。BLOB（或简称为二进制大对象）可以支持任何数据类型，但程序员要负责数据的显示、操纵和检索。

binary search：二分搜索——一种针对有序数据的检索技术。从数据的中间值开始，如果检索值比中间值大，就把后面的数据一分为二。如此进行下去，直到找出所要的值。

bitmap index：位图索引——一种紧凑、高速的索引方法，索引码值和条件都压缩到很小的范围，从而可以快速存储和检索数据。

Boolean algebra：布尔代数——创建并操纵用AND、OR和NOT条件连接起来的逻辑查询。

bound control：绑定控件——一种绑定在数据库列上的表单控件。当输入或改变数据时，数据更新就会自动保存到数据表中。

Boyce-Codd normal form (BCNF)：Boyce-Codd范式——所有的依赖都必须由码来显式地指出，不含有非码和码列的隐含依赖。

browser：浏览器——客户计算机上的一个软件包，用于从因特网上访问和显示Web页面。

brute force attack：穷举攻击——一种攻击方法，它试图通过试验口令或密钥的全部组合来突破安全系统。

call-level interface（CLI）：调用级接口——一组动态连接库，它使程序员可以利用DBMS之外的语言（例如C++）和DBMS的特性来工作。DBMS提供通信库，并自己处理大部分数据交换。

cascading delete：级联删除——当表由数据连接起来时，如果在更高级别的表中删除一行，其他表中与之匹配的行就会自动删除。例如，如果删除了1173号客户，则该客户的所有订单也自动删除。

cascading triggers：级联触发器——一个更新操作触发某个表的触发器，而该触发器的操作又造成第二张表的更新，接着又引起了第三张表的更新，依此类推，这一系列更新就可能引发多个触发器事件。

CASE——SQL操作符，被大多数系统所支持。它同时检查多个条件，当找到匹配的条件时就采取相应的行动。

certificate authority：认证中心——一个确保公钥合法性和双密钥加密系统申请人身份的公司。

check box：复选框——一种表示选择的方形按钮。通过设计向导，用户可以用复选框选择多个选项，与单选框只能在多个选项中选取一个不同。

clarity：清晰性——使应用更易使用的目标。通过配合用户任务的优雅设计和组织，使应用的使用对用户更加清晰。

class：类——对有着相似结构、行为和关系的一系列对象的描述符。也就是说，类是商务实体的模型描述。商务模型可能包括雇员类，而一个具体的雇员就是那个类的一个对象。

class diagram：类图——按照关系把很多类连接起来的图。它用来展示模型的静态结构，与实体-关系图类似。

class hierarchy：类层次——突出了类之间继承关系的图。

classification analysis：分类分析——一种数据挖掘技术，它把诸如客户这样的对象分成组，并决定哪些因素是重要的分类变量。

client/server：客户机/服务器——一种组织系统的技术：只有一些计算机存储绝大部分数据，这些数据由使用客户端计算机的个人来提取。

cluster analysis：聚类分析——一种数据挖掘技术，它把数据集的元素分组，常常是基于项与项之间的相似度的。

cold site：冷站——灾难备份专家可供出租的设施。冷站包含动力和通信线路，但没有计算机。灾难发生时，公司会呼叫计算机厂商，请求厂商把首批可用的计算机送往冷站。

collaboration diagram：协作图——一种显示对象间交互关系的UML图。它不把时间作为一个独立的维来显示，用来对过程建模。

combo box：组合框——列表框和文本框的组合，用于输入新数据或从列表项中选择。与列表框相比，组合框节省空间，因为只有当用户选择时列表才会显示。也就是Web表单中的选择框。

command button：命令按钮——一种设计来点击的表单按钮。当点击按钮时，会激活设

计者编写的代码。

common gateway interface (CGI)：通用网关接口——在Web服务器中，CGI是一个用于在因特网中传输数据的预定义系统。当前的脚本语言隐藏其细节，所以你可以轻易地按需得到数据。

composite key：复合码——包括不止一列的主码。表示列之间的多对多关系。

composition association：组合关联——在这种关系里，对象由一组其他对象构成。例如，自行车由零件构成。在UML中，小的实心菱形用来指示关联端。

computer-aided software engineering (CASE)：计算机辅助软件工程——为支持计算机系统的分析和开发而设计的计算机程序。这些程序有助于创建、存储和分享图表和数据定义。某些版本还能分析现有代码并生成新代码。

concatenate：连接——把一个串接到另一个串尾的程序操作。例如，姓&","&名就可能产生"Smith, John"。

concatenated key：连接码——参见Composit key。

concurrent access：并发访问——同时对相同数据执行两个（或多个）操作。DBMS必须把操作顺序化，这样一些改变才不会丢失。

confidence：置信度——在关联规则的数据挖掘中，规则强度的度量由包含A项的事务同时也包含B项的百分比计算得到。也就是A已经在篮子里时，B也在的概率。

consistency, application：（应用程序）一致性——通过一直使用相同的特性、颜色和命令来达到让应用更易使用的目标。现代应用都寻求在共同设计向导下达到一致性。

consistency, transaction：（事务）一致性——事务的要求，它指定了在更改提交时所有的数据都必须内部保持一致，并且可以由应用来检查验证。

constraint：约束——在SQL中，约束是强加在数据上的规则。例如，可能有一些列声明了主码和外码约束，这实际上就限制了输入到表中的数据。其他商业规则也可以形成约束，例如Price>0。

context-sensitive help：上下文相关帮助——被过滤以适合用户正在执行的操作的帮助消息。

context-sensitive menu：上下文相关菜单——随着用户选择的对象不同而不断改变的菜单。

control break：控制中断——包含分组数据的报表使用控制中断来分离各组。这个中断在标识每组成员的码变量上定义。

controls：控件——放置在表单上的小组件的统称。典型的控件包括文本框、组合框和标签。

correlated subquery：关联子查询——必须为主查询的每一行重新求值的子查询，运行起来可能会非常地慢。常常可以避免关联子查询，方法是创建临时表并在子查询中使用。

CREATE DOMAIN——SQL数据定义命令，用来创建由现有数据域组合起来的新数据域。

CREATE SCHEMA——SQL数据定义命令，用来创建表的新逻辑表分组。在一些系统中，它和创建新数据库是一样的。这个命令在Oracle和SQL Server中不可用。

CREATE TABLE——SQL数据定义命令，用来创建新表。这个命令常常由程序生成。

CREATE VIEW——SQL命令，用来创建新视图或保存查询。

Cross JOIN：交叉连接——当没有指定两个表的连接条件时就会发生。第一个表的每一行要与第二个表的每一行进行匹配，也称“笛卡儿积”。应该尽量避免这种操作。

crosstab：交叉表——一种特殊的SQL查询（并不是所有的系统都提供），它基于两组数据创建一个列表的输出。Access使用TRANSFORM命令创建交叉表。

cursor：游标——（1）在图形化环境下，指针的当前位置。（2）跟踪一个表中各行的指针，一次可以激活数据中的一行。

cylinder：柱面——磁盘驱动器划分成表示磁道一部分的柱面（或扇区）。

data administration：数据管理——在整个公司内定义数据一致性所必需的计划和协调。

data administrator（DA）：数据管理员——管理一个公司数据资源的人。DA负责数据完整性、一致性和集成性。

data definition language（DDL）：数据定义语言——用来定义数据的命令集，例如CREATE TABLE。图形界面常常更容易使用，但用程序创建新表时，数据定义命令很有用。

data device：数据设备——为保存数据库的表、索引和回滚数据而分配的存储空间。参见tablespace。

data dictionary：数据字典——保存所有数据表的定义，描述将要存储的数据类型。

data independence：数据独立性——把数据从程序中分离出来，常常可以在不修改程序的情况下更改数据定义。

data integrity：数据完整性——保持数据的正确性，意思是没有错误，而且数据要反映业务的真实状态。DBMS指定约束或规则来维护完整性。例如价格必须总是大于0的。

data manipulation language（DML）：数据操纵语言——用来修改数据的命令集。参见INSERT、DELETE和UPDATE。

data mining：数据挖掘——在数据库中搜索未知模式和信息。其工具包括统计分析、模式匹配技术以及数据分割分析、分类分析、关联规则和聚类分析等。

data normalization：数据规范化——创建表现良好的表集合以有效存储数据、最小化冗余和确保数据完整性的全过程参见第1、第2和第3范式。

data replication：数据复制——分布式系统中，在若干个服务器上放置数据的副本，以减少全局的传输时间和代价。

data type：数据类型——可以用一列保存的一类数据。每个DBMS都有一些事先定义好的系统域（整型、浮点型、字符串等）。一些系统支持由其他数据类型组合而来的用户自定义域。

data volume：数据量——数据库大小的估计值。对每张表来说，通过把估计的行数乘以每行的平均数据长度而计算得到。

data warehouse：数据仓库——为管理查询而优化的专用数据库。数据从联机事务处理系统中抽取出来，经过清洗，然后为搜索和分析进行优化。通常由并行处理机和RAID存储所支持。

database：数据库——用标准格式存储的数据集，设计来为多个用户所共享。也是为特定

商业情况而建的表集合。

database administration：数据库管理——创建和运行数据库的技术。其基本任务是性能监控、备份与恢复以及分配和控制安全性。

database administrator (DBA)：数据库管理员——在管理特定DBMS方面受过训练的专家。DBA要经过安装、配置和操纵DBMS的细节方面的训练。

database cursor：数据库游标——在定义SELECT语句的程序设计语言中创建的变量，一次指向一行数据。该行数据可以用程序语言提取或编辑。

database engine：数据库引擎——DBMS的核心，它负责存储、提取和更新数据。

database management system (DBMS)：数据库管理系统——定义数据库、存储数据、支持查询语言、生成报表以及创建数据输入屏幕的软件。

datasheet：数据表单——显示数据行、列的格状表单，通常用作子表单。数据表单用尽可能少的空间显示数据。

deadlock：死锁——当两个（或多个）进程都封锁另一个所需要的数据时发生的情况。

default value：默认值——自动显示和输入的值，用来在数据输入中节省时间。

DELETE——SQL数据操纵命令，用于删除数据行。它总是随着WHERE子句使用，用WHERE指定哪些行应删除。

deletion anomaly：删除异常——从一个不属于第三范式的表中删除数据时引发的问题。例如，如果所有的客户数据都存放在每个订单里，删除订单时，就丢失了所有相关客户的数据。

DeMorgan's law：德摩根定律——一个代数定律，意思是：要否定含AND或OR连接词的条件，就需要否定每一个子条件并交换连接词。也就是说，AND变成OR，反之亦然。

dependence：依赖——数据规范化中的问题。如果属性A的值的改变受属性B改变的影响，就说A依赖于B。例如，客户的名字依赖于客户的ID（每位雇员都有一个特定的名字）。另一方面，客户的名字并不依赖于订单的ID。每次订购货物时客户的名字并不改变。

derived class：派生类——用另一个类经扩展而得到的类。程序员仅需要定义新的属性和方法。其他所有的内容都从更高级别的类中继承而来。参见inheritance。

DESC——在SQL SELECT ... ORDER BY语句中的修饰词，指定是递减排序（如Z...A）。ASC可以用做递增排序，但它是默认的，所以不需要指定。

dimension：维——OLAP立方体的一个属性，用来分组和检索数据。

direct access：直接存取——一种数据存储方法：物理地址由逻辑码值计算得到。数据可以不经检索地存储和提取。

direct manipulation of objects：对象的直接操作——一种图形化界面的方法，设计来模仿真实世界的行为。例如，通过拖拽图标从一个地方把文件复制到另一个地方。

disaster plan：灾难应急计划——针对突发事件而创建的计划，当灾难袭击计算机系统时使用。计划包括了备份的离线存储、人员通知以及在一个安全站点建立操作。

DISTINCT——在SELECT语句中用于从输出中移除重复行的SQL关键字。

distributed database：分布式数据库——运行在两个或多个利用网络连接起来并共享数据的计算机上的多个独立的数据库。这些数据库通常处在不同的物理场地，每个数据库都由独立的DBMS控制。

dockable toolbar：可停靠的工具栏——用户可以拖到应用窗口任意位置的工具栏。通常可以自定义选项和按钮，以执行特定的任务。

domain-key normal form (DKNF)：域－码范式——设计数据库的最终目标。每张表代表了一个主题，并且所有的商业规则都用域约束和码关系来表达。即，所有的商业规则都用表规则显式描述。

drag-and-drop：拖放——一种图形界面技术，它支持定义这样的动作：即按下鼠标并保持，拖动图标，然后把图标放到一个新对象上。

drill down：下钻——由显示的综合性数据得到更多细节的操作。常用于在数据仓库或OLAP应用中检查数据。参见roll up。

drive head：磁头——在磁盘上读取和写入数据的机械设备。现代驱动器都有好几个磁头。

DROP TABLE——SQL数据定义命令，用于从数据库中完全删除表——包括定义。请保守使用。

dual-key encryption：双密钥加密——使用两个不同密钥的加密技术：一个私钥和一个公钥。公钥公开，任何人都可以得到。要发送加密的消息给某人，就用那个人的公钥。另一面，只有那个人的私钥才能解密消息。先用你的私钥加密消息也可以证明是你写了那条消息。

durability：持久性——事务的要素之一，它指定当事务提交时，所有的更改都必须永久保存，即使有硬件或系统故障。

edit：（微软DAO命令）——微软用来在当前行修改数据的DAO命令。

electronic data interchange (EDI)：电子数据交换——在含有诸如供应商、客户和银行等外部代理的网络上交换数据。

encapsulation：封装——面向对象编程中，在公共类里定义属性和方法的技术。例如，Employee（雇员）类的所有属性和功能放在一起。其他对象可以通过引用Employee对象来使用这些属性和方法。

encryption：加密——用一个密码值将数据编码，使数据变得不可读。当今两种常用的加密方法是：单密钥加密（如DES）和双密钥加密（如RSA）。

entity：实体——现实世界中我们希望识别和跟踪的一个项。

entity-relationship diagram (ERD)：实体－关系图——展示商务实体之间关联（关系）的图。在UML中，类图显示了相似的关系。

equi-join：等值连接——SQL中条件为等式的连接（JOIN）。两张表中的行当列完全匹配时做连接。等值连接是最常见的JOIN条件，跟另一张表没有任何匹配的行将不显示。

EXCEPT——从两个SELECT语句中检查行的SQL操作符。它返回第一个语句的所有行，但除去第二个语句将要返回的那些行。有时候实现为SUBTRACT命令。参见UNION。

EXISTS——用来确定子查询是否返回一些数据行的SQL关键字。

expert system (ES)：专家系统——让新手可以像专家那样做出决策的系统，包含数据和规则的基础知识。

extensible markup language (XML)：可扩展标记语言——一个基于标签的符号系统，用于给数据分配名字和结构。主要是为不同系统之间传输数据而设计的。

extraction, transformation, and transportation (ETT)：抽取、转化和传送——用现有文

件或数据库组装数据仓库的三个步骤。抽取表示选择想要的数据。转化通常是最难的步骤，而且要让数据一致。传送意味着数据必须在网络上物理地移动到数据仓库中。

fact table：事实表——存储将要在OLAP立方体中表达的数据表或查询。

feasibility study：可行性研究——对建议系统的问题、目标及预期成本的快速检查。其目的是确定存在的问题是否能被计算机系统有效地解决。

feedback：反馈——一种设计特性，当任务完成或错误发生时，应用向用户提供信息。反馈可能以多种形式提供（如消息、可见信号或发声提醒等）。

FETCH——在SQL游标编程中用到的命令，目的是把下一行的数据提取到内存中。

first normal form（1NF）：第一范式——一个表处于第一范式，是指当里面没有重复分组时。每一单元格只能包含一个值。例如，有多少项可以放进Order（订单）表？项有重复的话，就必须分成不同的表。

fixed-width storage：定宽存储——每列都用固定的字节来存储数据的每一行。

fixed-with-overflow storage：带溢出的定宽存储——用有限的字节存储数据行的一部分，并把多余的数据移到溢出区去。

focus：焦点——在窗口环境中，当可以接收按键消息时就说表单或控件保持着焦点。通常都是高亮显示的。

For Each ... Next——VBA中的迭代命令，可以自动识别组中对象，并在该集合上应用某个操作。当处理电子数据表格时尤其有用。

foreign key：外码——表中的一列，它是另一张表的主码，但它不必是第一张表的码。例如，在Order（订单）表中，CustomerID（客户ID）就是个外码，因为它是Costomer（客户）表的主码。

forms generator：表单生成器——让你在屏幕上建立输入表单的DBMS工具。

fourth normal form（4NF）：第四范式——不能有码列之间的隐含依赖。当一个码决定着两个不同但独立的属性时，就存在了多值依赖。把表分开，让这两个依赖显形。

FROM——确定查询从哪些表中获取数据的SQL SELECT子句，也用于含JOIN或INNER JOIN语句的连接中。

FULL JOIN——一种连接运算，其运算结果为两个表之间所有匹配的行，加上左表中没有匹配的行和右表中没有匹配的行。这个运算不常使用，参见LEFT JOIN和RIGHT JOIN。

function：函数——用来执行特定计算的过程。函数和子例程的区别在于函数要返回一个特定值（不包括参数）。

generalization association：概括关联——由概要类发起的一种类间关系。更详细的类从它衍生而来，并继承了更高级别类的属性和方法。

geocode：经纬度代码——给数据集指定其地点的经度和纬度坐标。

geographic information system（GIS）：地理信息系统——用来识别和显示商业数据与地点之间关系的系统，是在数据库环境中使用对象的一个很好的例子。

GRANT——赋予某人对特定表或查询以权限的SQL命令。

graphics interchange file（GIF）：图形交换文件——存储图形图像的一种标准方法，通常用于在因特网上共享图片。

group break：分组中断——把数据分成组的报表，分隔点就叫做“中断”，也叫做控制中断。

GROUP BY——对组中每一项计算聚集值的SQL SELECT子句。例如，SELECT Department, SUM (Salary) FROM Employee GROUP BY Department；表示对每一部门计算并列出雇员薪水的总和。

HAVING——随GROUP BY语句使用的SQL子句，它限制只输出那些符合特定条件的组。

heads-down data entry：盲打——触摸式的打字员，只关注输入数据，不看屏幕。为这种任务设计的表单应该最小化按键，并使用声音提示。

Help system：帮助系统——一种用于显示、排列和搜索帮助文档的系统。开发者需要以特定的格式编写帮助文档，然后使用帮助文档编译器来生成最终的帮助文件。

hidden dependency：隐含依赖——由不在表结构中显示的商业规则指定的依赖。通常意味着表应该进一步规范化，并在Boyce-Codd范式或第四范式上存在问题。

hierarchical database：层次数据库——一种过时的DBMS类型，用可以快速地自顶向下搜索的层次结构来组织数据，例如Customer（客户）— Order（订单）— OrderItem（订购项）。

horizontal partition：水平划分——根据数据行把表划分成不同的组。不常用的行移到较慢、较便宜的存储设备中。

hot site：热站——灾难备份专家那里可供出租的设施。热站包含所有的动力、通信设施以及公司运行所必须的计算机。在灾难事件里，公司收集备份数据，通知员工，并把业务操作转移到热站。

human factors design：人性化设计——把计算机系统设计得适合人类使用。

hypertext link：超文本链接——超文本链接（例如Web）文档包含文本和图形，里面有可以得到新页面的链接。点击链接是浏览互联网和获得更多信息的主要手段。

hypertext markup language (HTML)：超文本标记语言——一种显示标准，用于创建在因特网上共享的文档。许多生成器都可以从标准文字处理文件的文件中创建HTML文档。

icon：图标——一些思想或对象的小图片表示。一般情况下，用于图形用户界面来执行命令和操纵基本对象。

IN——SQL WHERE子句的操作符，一般情况下会和子查询一起使用。当选择的项和列表中的某一项匹配时，返回符合这个匹配的数据。例如，WHERE ItemID IN (115, 235, 536) 返回ItemID是115、235或536的数据行。一般来说，括号里经常是另一个SELECT语句。

index：索引——原始表中一组码值的有序列表，含有指向每一行数据的指针。用于加速搜索和数据获取。

indexed sequential access method (ISAM)：索引顺序存取法——一种数据存储方法，和单纯的顺序检索比，依靠索引来更快地检索和提取数据。

inequality join：不等连接——不使用相等操作符，而用不等（大于或小于）来执行比较的SQL连接。根据数据范围把数据分类存放时比较有用。

inheritance：继承——在面向对象设计中，定义源自更高级别类的新类的能力。新类继承所有更高一级的属性和方法，所以程序员只需要定义新类特有的属性和方法。

INNER JOIN——SQL的等值连接条件，两个表中的行在列精确匹配时的连接。这是最常用的连接情况，和另一张表没有匹配的行不显示。

InputBox：输入框——一个预定义的简单Windows表单，可能会用于从用户获取一条数据。但最好不要使用它，而是创建你自己的表单。

INSERT——插入数据到表中的两个SQL命令。一个命令用于一次插入单独一行，另一个命令从某个查询中复制选中的数据并作为新数据行追加到目标表里。

insertion anomaly：插入异常——试图把数据插入到不属于第三范式的表中而引发的问题。例如，如果发现自己在重复输入相同的数据（例如某个客户的地址），那么这个表就很可能需要重新定义。

Internet：因特网——一组松散连接的、在世界范围内交换信息的计算机及其网络。计算机的所有者制作供其他用户使用的文件和信息。

INTERSECT——两个SELECT语句中针对数据行的集合操作，只有两个语句共有的行才会提取。参见UNION。

intranet：内联网——使用互联网技术共享数据的公司内部网络。

isolation：隔离性——事务的性质，即系统必须让每个事务都感觉它是独立运行的，且不存在并发访问问题。

isolation level：隔离等级——在事务中用来指定锁的属性。至少，用来指定乐观锁还是悲观锁。有些系统还支持中间级别。

iteration：迭代——使代码的一部分重复执行，例如循环扫描数据的每一行这种需求。典型的命令包括Do...loop和For...Next。

Java：Java程序设计语言——由Sun Microsystems公司开发的程序设计语言，声称能够不加修改地运行在多种计算机平台上。Java最初设计为嵌入式系统的控制语言，而现在的目标是因特网应用。

Java 2 enterprises edition (J2EE)：Java 2企业版——基于服务器构建复杂应用的后端系统，它基于Java但包含一个完整的编程环境。

JDBC——有时称为Java数据库连接。连接Java代码到数据库的方法集。与ADO的目的相似，但仅适用于Java程序。

JOIN——当从不止一个表中提取数据时，各表必须按一定的数据列连接起来。参见INNER JOIN和LEFT JOIN。

latency：等待时间——系统的时间延迟。在基于Web的系统里，延迟是由慢速的网络连接造成的。长时间的下载造成了较大的等待时间，从而会导致更多的服务器冲突。

LEFT JOIN——包含了“左”表所有的行（即使“右”表中没有匹配的行）的OUTER JOIN。缺失的值显示为Null。参见RIGHT JOIN和INNER JOIN。左表和右表是按表出现的先后顺序定义的，左表是第一个表。

lifetime：生命期——程序变量保持有效的时间长度。例如，在子例程中创建的程序变量，当子例程执行时就创建，而退出时销毁。全局变量对模块内的所有过程一直有效。

lift：提升度——由关联规则贡献的潜在收益（相对于没有规则时的购买而言）。

LIKE——用来比较字符串值的SQL模式匹配操作符。其标准使用百分号（%）匹配任意数

目的字符，下划线（_）匹配单一字符。有些系统（例如Access）使用星号（*）和问号（？）作为替代百分号（%）和下划线（_）的替代符号。

list box：列表框——显示选择列表的表单控件。这个列表一直显示着，并占据固定大小的屏幕空间。

list of values (LOV)：值列表——在Oracle表单上选择数据的重要技术。这种Oracle表单维护了可从文本框选取的数据的一部分且缓存的列表，用来替代组合框或选择框。由于只给用户发送列表的一部分，所以在分布式数据库中尤其有用。

local area network (LAN)：局域网——较小地理区域内的个人计算机的集合，所有的网络组件都由一个公司拥有或控制。

local variable：局部变量——定义在子例程内的变量，只能在子例程内部访问该变量，外部过程不能访问它。

logic：逻辑——定义了程序目的和结构的逻辑表述。可以用独立于程序语法的伪码来描述。逻辑结构包括循环、条件、子例程和输入/输出命令等。

logical security：逻辑安全——确定哪些用户应该对哪些数据有访问权。它防止三种数据问题：(1) 未授权的泄漏，(2) 未授权的更改，以及 (3) 未授权的截留。

loop：循环——每个循环都必须有开始和结束条件，以及递增循环变量值的过程。参见iteration。

market basket analysis：购物篮分析——参见association rules。

master-detail：主从关系——常见于商业表单中的一对多关系，就是主表单（如Order）为显示主组件数据，子表单（如Order Item）显示详细数据。有时也叫父子关系。

measure：度量——OLAP立方体中显示的数值数据。

menu：菜单——一系列分组排列的应用命令，通常在工具栏上。它为经常使用的命令提供引用方法，并且突出应用的结构。

metadata：元数据——有关数据的数据，或者说是数据表和各列的描述。元数据通常保存在数据字典里。

method：方法——类可以执行的函数或操作。例如，Customer（客户）类无论什么时候给数据库中添加新客户对象时，都会调用AddNew方法。

model form：模式框——一个优先出现在屏幕上的框，强制用户在继续执行当前进程前必须先处理它。应该避免使用模式框，因为它妨碍用户的操作。

module (or package)：模块（或包）——子例程的集合，常常和通用的目标有关。

MsgBox：消息框——Windows中一个预定义的方法，它在屏幕上显示简要消息并展示给用户几种选择。由于是对话框且打扰用户，所以应该尽量少用。

multiplicity：重复度——表示关联数量的UML术语。它用一个最小值、一个省略号（...）和一个最大值或表示许多的星号（*）显示在关联线上。例如，客户可以发放0到多个订单，所以重复度是（0...*）。

***n*-ary association**：多重关联——三个或多个类之间的关联。它在UML类图中用菱形表示。这个术语由扩展英文单词而来：unary表示1，binary表示2，ternary表示4；所以N-ary表示许多。

naming conventions：命名惯例——程序团队应该以一致的格式来命名变量和控件。一个常用的方法就是使用3个字母的前缀表示变量类型，然后附上描述名。

nested conditions：嵌套条件——放置在其他条件语句内部的条件语句。例如，If (x > 0) Then ... If (y < 4) Then ...。有些嵌套条件可以用Case语句代替。

nested query：嵌套查询——参见subquery。

network database：网状数据库——一种过时的DBMS类型，它通过支持实体之间的多重连接而扩展了层次型数据库。网状数据库表现为所有的连接都必须被索引支持。

normalization：规范化——参见data normalization。

NOT——SQL的否定操作符。在WHERE子句中用于对一个条件的结果求反。参见DeMorgan's law。

NotInList——一个关联到Access组合框的事件，当用户输入不在列表中的值时就会触发它。常用于给表中添加新数据，例如新客户。

NULL——缺失（或当前未指定）的值。

object：对象——类的实例或特别范例。例如，在Employee（雇员）类中，一个单独的雇员就是一个对象。在关系环境中，类存在表中，表中一个独立的行就是一个对象的数据。

object browser：对象浏览器——微软提供的用来显示可用的对象属性和对象方法的工具。

object-oriented (OO) database management system (OODBMS)：面向对象的数据库管理系统——保存对象（包括属性和方法）的数据库系统，支持对象之间的联系，包括继承。

object-oriented programming：面向对象编程——一种把代码封装在不同对象（或类）定义内部的编程方法。系统由（有希望）可重用的对象构建起来，通过操纵对象属性和调用对象方法来控制系统。

online analytical processing (OLAP)：联机分析处理——用数据库来做数据分析，其关键是数据的提取。主要目标是提供可接受的响应时间、维护安全以及方便用户找到他们所需的数据。

online transaction processing (OLTP)：联机事务处理——用数据库来完成事务处理。它包括很多插入和更新操作，并支持数百的并发访问。数据的高速存储、可靠性以及数据的完整性都是主要目标。具体的实例包括航班预定、在线银行和零售。

open database connectivity (ODBC)：开放式数据库互连——微软提出的标准，旨在使软件可以访问不同的数据库。每个DBMS厂商都提供ODBC驱动程序。要更换DBMS，只需安装并设置合适的ODBC驱动程序，而应用代码只需要编写一次即可。

optimistic lock：乐观锁——不阻塞其他进程的事务锁。如果数据在读、写操作之间被更改，系统就会产生一个必须由代码处理的错误。

option button：单选按钮——用于指示选择的圆形按钮。按照设计向导，单选按钮表示相互排斥的选择，这与选择框不同。

ORDER BY——SQL SELECT语句里的子句，用来将数据列排序输出。修饰符ASC和DESC用于指定是升序还是降序排列。

OUTER JOIN——代表**左连接**和**右连接**的概括性术语。它从表中返回行，即使另一张表

中没有匹配的行。

pack：压缩整理——必须定期在数据库上执行的维护操作，它移去已删除数据的片断。

package：包——UML中把逻辑元素分成一组的机制，在隔离设计组件时很有用。包可以提供整个系统的全局视图，而不需要看到所有的细节。

page footer：页脚——出现在每页底端的报表元素，常用作页码。

page header：页眉——出现在每页顶端的报表元素，常用作列标题和子标题。

parameter：参数——传递给函数或子例程并在计算中使用的变量。

pass-by-reference：传址——可以在子例程内部修改的子例程参数。如果它被修改了，新值就会返回给调用程序。也就是说，子例程可以在代码的其他部分修改变量。通常这是危险的做法。参见pass-by-value。

pass-by-value：传值——不能在子例程内部修改的子例程参数，只传递了它的值。如果子例程改变该值，调用程序中的初始值也不会改变。

pass-through query：直通查询——前端被Access忽略的SQL查询。它们可以让你编写针对特定服务器数据库的复杂SQL查询。

persistent objects：持久对象——面向对象编程中，一种存储对象（文件或数据库中）而后还能把数据提取出来的能力。

persistent stored modules (PSM)：持久存储模块——SQL-99中，一种把方法和对象联起来存储的做法。模块代码将由DBMS自动存储和提取。

pessimistic lock：悲观锁——从开始读取锁定的数据到事务结束前都阻塞其他进程的完全隔离级别。如果数据元素被锁定，程序代码将会接收到一个错误消息。

physical security：物理安全——涉及物理性地保护设备和人员的安全性分支。它包括灾难应急计划、设备的物理访问以及风险分析和防范。

pivot table：主元表——微软给经理们动态检查OLAP数据的工具，典型情况是在Excel电子数据表的内部。数据从数据库或OLAP立方体中抽取出来，然后经理们可以点击按钮来检查得和或详细数据。

pointer：指针——数据的逻辑地址或物理地址。

polymorphism：多态性——在类的层次结构中，每个新类都从上层类中继承方法。通过多态性，你可以重载那些定义，并给新类指定新的方法（用同样的名字）。

primary key：主码——表中标识特殊数据行的列或列集。

private key：私钥——指双钥加密系统中不透露给其他任何人的密钥。用公钥加密的消息只能用匹配的私钥解开。

procedural language：过程语言——一种基于过程的传统编程语言，一般情况是一次执行一条语句。可以和命令直接作用到数据集上的SQL做个对比。

procedure：过程——设计来执行某特定任务的子例程或函数，通常保持过程短小是比较明智的。

property：属性——我们希望跟踪的实体的属性或特征，这个术语常用于面向对象环境中。参见attribute。

prototype：原型——应用的最初轮廓，快速构建起来用于演示和测试应用的不同特性。

常用来帮助用户进行可视化和改善表单、报表。

public key：公钥——指双钥加密系统中公开的密钥，用公钥加密的消息只能用匹配的私钥解开。

query by example (QBE)：QBE语言——用填表的方法设计查询。从一个列表中选择表和列，并为条件和排序填空。它相对容易使用，需要最小的输入技巧，通常随帮助系统提供，对初学者很有用。

rapid application development (RAD)：快速应用开发——试图通过效率和过程交叠来缩短开发时间的系统设计方法学。

referential integrity：参照完整性——数据完整性约束，数据值已经在基本表中时，该数据才可以输入到外码列中。例如，如果1173这个客户ID不在客户表中，那么操作员就没法输入1173号客户的订单。

reflexive association：自反关联——一个类返回自身的关联。最常见于商务中的Employee（雇员）类，因为某些雇员是另一些雇员的经理。

reflexive join：自反连接——当表通过第二个列与自己连接时的情形。例如，表Employee (EmployeeID, ..., ManagerID) 就可能有从ManagerID到EmployeeID的连接。

relational database：关系数据库——最流行的DBMS类型。所有的数据都存储在表（有时也叫关系）中。各表通过它们所持的数据（例如，通过码值）逻辑地联系着。关系数据库应当通过数据规范化来设计。

relationship：关系——两个或多个实体之间的关联。参见association。

repeating groups：重复组——重复数据的组，例如客户购买的各种商品，一个客户的多个电话号码以及分配给某个员工的各项任务。

replicate：复制——制作数据库的一个良好副本，以便可以分布到新的地点，然后迟些时候由主拷贝进行数据同步。

replication manager：复制管理器——在依赖复制的分布式数据库中，管理器是把更改传给数据库的不同副本的自动系统。如果两个人在复制前更改了同一份数据，它就必须处理这些冲突。

report footer：报表尾注——出现在报表结尾的报表元素，常用于总结性的统计或图表。

report header：报表标题——只出现在报表开头的报表元素，常用于标题页和概览。

report writer：报表编写器——让你在屏幕上建立报表来指定各项如何显示及计算的DBMS工具。这些任务中的大多数都通过在屏幕上拖拽数据来执行。

resume：(VBA语句) ——VBA错误处理语句，它告诉进程返回新地点并继续评估代码。Resume自身就返回到导致错误的行，Resume Next返回到错误行的下一行，而Resume <label>则把处理控制转到label指定的新行。

REVOKE——SQL命令，用来删除分发给某些用户的许可。

RIGHT JOIN——包含“右”表所有行（即使“左”表中没有匹配的行）的OUTER JOIN，缺失的值显示为Null。参见LEFT JOIN和INNER JOIN。左表和右表是按表的列出顺序定义的，左表是第一个表。

Rivest-Shamir-Adelman (RSA) encryption：RSA加密——一种由三名数学家持有美国

专利的双钥加密系统。参见dual-key encrytion。

role：角色——按任务或角色来分配权限的安全组，用它替代分配给个人的方案。角色使得改变雇员权限变得简单多了。

roll back：回滚——数据库系统的事务特性。如果错误发生在一个改变序列中，前面的改变可以回滚，然后用正确的数据将数据库重新恢复到一个安全状态。

roll forward：前滚——如果在处理事务中发生错误，数据库可以重启，并从一个已知的检查点进行装载。然后部分完成的事务可以前滚到中断的记录处。

roll up：上卷——聚集细节数据到按类别总计显示的操作，常用在数据仓库或OLAP应用中检查数据。参见drill down。

row-by-row calculations：逐行计算——SQL执行内联计算的方式。例如，语句SELECT price * Quantity AS Extended遍历每一行，并把行中的价格值和对应的数量值相乘得出计算结果。

schema：模式——按共同目的分组的表集合。

scope：作用域——指变量在哪里可以被访问。一般情况下，在子例程内部定义的变量只能被那个子例程内部的代码访问。定义到模块内的变量可以全局地被模块内的任意代码访问。

scroll bars：滚动条——用来进行水平或垂直移动的常见的图形界面元素。

second normal form (2NF)：第二范式——如果每个非码列都依赖于整个码（不只是码的一部分），这个表就属于第二范式。这个问题仅当有连接码（多列）时才会出现。

SELECT——SQL中主要的数据提取命令，其主要组件包括SELECT ... FROM ... INNER JOIN ... WHERE ...。

self-join：自连接——表连接到它本身。参见reflexive join。

serialization：串行化——事务要求，它指定每个事务完全独立对待，就好像没有其他事务一样。

shell site：（见“冷站”）——参见cold site。

single-row form：单行表单——一次只利用表中一行显示数据的输入表单，是最常见的输入表单，因为设计者拥有表单布局的完全控制。

snapshot：快照——在持续变化的数据库中，可以取一个快照来得到数据在某个时间点的一份副本。

snowflake design：雪花设计——用于OLAP的数据库设计。事实表连着维表，维表可以连接其他维表。比星形设计限制得要宽松。

SQL——标准化的数据库语言，用于数据提取（查询）、数据定义和数据操纵。

SQL 1999 (SQL 3)——SQL 1999的设计很大程度上给SQL语言增添了面向对象的特性，并于1999年通过该标准。

SQL 200x——一个几乎已经完成的SQL新版本。它的主要贡献是把XML和多媒体集成到关系数据库的规范化定义里。

star design：星形设计——用于OLAP的数据库设计。事实表连接着提供待分析类别的维表。比雪花设计限制更多一些，所有的维表都直接连接在事实表上。

style sheet：样式表——描述一系列网页所需布局、字体及风格的特殊文件。这是个很强

大的方法，通过对文件进行极少的修改就能建立和改变许多网页的风格。

subform form：子表单——显示在另一个（主）表单内部的表单。子表单里的数据通常连接到主表单中正在显示的行上。

subquery：子查询——在主查询内部使用第二个查询来提取额外的数据。例如，要提取价格比平均值高的所有销售数据，WHERE子句可以使用一个子查询来计算平均价格。

subroutine：子例程——设计来执行特定任务的一个独立的代码片断。子例程可以使用参数和调用它的过程交换数据。

subtable：子表——在SQL 1999中，子表从基本表中继承所有的列。它提供类似于抽象数据类型的继承；但是，所有的数据都存放在独立的列里。

super-aggregate：超聚集——查询中合计的合计，是OLAP查询中由ROLL UP选项提供的重要概念。

support：支持度——关联规则中的数据挖掘的度量，由包含两个项的事务的百分比计算得到。

switchboard form：导航表单——用于指引用户到应用不同部分的表单，常用作显示的第一个表单。表单上的选项应该匹配用户的各项任务。

synonym：同义词——完全数据库路径的简短名字。短名字的优点是很容易记忆，而且用户不需要知道数据在哪里存放。另外，如果每个人都使用同义词，数据库管理员可以很容易地把服务器数据库转移到不同地点，而仅需修改同义词的属性。

syntax：语法——可以在程序中创建命令的特定格式。程序包括逻辑步骤，但每个命令必须按特定的格式给出，以便编译器理解。编译器通常要检查语法并进行消息提示。

tab order：Tab顺序——用户按Tab键或Return键时表单上控件的跟随次序。

table：表——一个类或实体的数据集合，每个属性包含很多列，并且在每行保存一个实体或对象的数据。

tablespace：表空间——Oracle中，表空间是分配来保存表、索引和其他系统数据的磁盘空间。必须首先知道数据库的大致大小。

tabular form：表格表单——在列和行显示数据的输入表单。当没有几列数据或者用户需要同时看到多行时比较有用。

third normal form (3NF)：第三范式——如果每一个非码列都依赖于整个码（而没有其他码），表就满足第三范式。

three-tier client/server：三层客户端/服务器——含有中间层的客户端/服务器系统，中间层保存着定义商业规则和加强对不同事务服务器访问能力的代码。

toggle button：切换按钮——选择框的一个三维变型，用来指示一个选项。

toolbar：工具栏——应用里保存按钮和文本菜单的组件。用户可以点击一两下鼠标来执行命令。用来保持常用的命令，以及用于整个应用的命令，例如打印。

tooltip：工具提示——当用户把鼠标指针移过屏幕上一项时显示出来一个简短消息，对于识别图标的功能非常有用。

TOP——Access提供的SQL SELECT子句，用来限制显示输出的行数。可以直接设置行数或用总行的百分比来表示。

transaction：事务——在数据库应用中，事务是必须一起完行的一系列操作。事务必须由DBMS识别，然后提交或回滚（如果有错误）。例如，从一个银行账户到另一个账户的转帐需要对数据库进行两个更改，这两个更改必须同时成功或同时失败。

transaction processing：事务处理——为记录事务而收集数据。常见的例子包括销售、人力资源管理以及金融计算。

trigger：触发器——导致一个过程执行的事件。例如，点击按钮可以是一个触发器，一个数据值的改变也可以引起触发器执行。

tuning：调整——建立索引、重写查询，并设置和存储其他参数以提高数据库应用的性能。

two-phase commit：两阶段提交——一种在分布式数据库中处理并发和死锁问题的机制。在第一阶段，协调的DBMS发送更新到其他数据库并让它们准备这个事务。一旦它们都同意，协调者发送消息来提交更新。

unicode：——存储和显示不同字符集的标准方法。当今世界几乎所有的字符集都有定义，还包括一些古老的语言。它使用2个字节表示每个字符，使得它可以处理超过65 000个字符或意符。

Unified Modeling Language (UML)：统一建模语言——设计并记录计算机和商务系统的标准化建模语言。

UNION——从两个SELECT语句中合并行的SQL子句。两个查询必须有相同数目、相同域的列。大多数系统也支持INTERSECT和EXCEPT（或SUBTRACT）操作符。

UPDATE——SQL数据操纵命令，用来改变特定列的值。WHERE子句指定哪些行会受影响。

User Interface：用户界面——用户看到应用时的外观和感觉。图形化界面经常采用，用户可以在屏幕上操纵图标和数据来执行任务。

validation table：确认表——一或两列的简单表，包含输入其他表的标准化数据。例如，一个部门的列表可能会存储在确认表中。要给Employee（雇员）表中输入部门的名字，用户只需从确认表中选出那些行就可以了。

VARCHAR——存储字符数据的一种常见方法，它代表可变的字符。数据的每一列都使用准确数目的字节，用这些字节来存储特定数据。

variable：变量——用来存储临时值的内存地点。依赖于在哪里创建和如何定义，变量都有作用域和生命期。它们也有特定的数据类型，尽管VBA中的Variant数据类型可以匹配任何常见的数据类型。

vertical partition：垂直划分——根据数据列把表划分成组。较大的列或不常用的列（例如图片）可以移到较慢、也更便宜的存储设备中。

view：视图——保存的查询。可以从视图中创建提取数据的新查询。视图保存为SQL语句，而不是实实在在的数据。

Visual Basic (VB)：Visual Basic语言——一种微软出售的独立的程序设计语言，用来在Windows环境中开发应用。其专业版支持数据库连接。它的程序可以编译成独立的可执行文件。

Visual Basic for Application (VBA)：VB应用——能用于几乎所有微软工具（包括