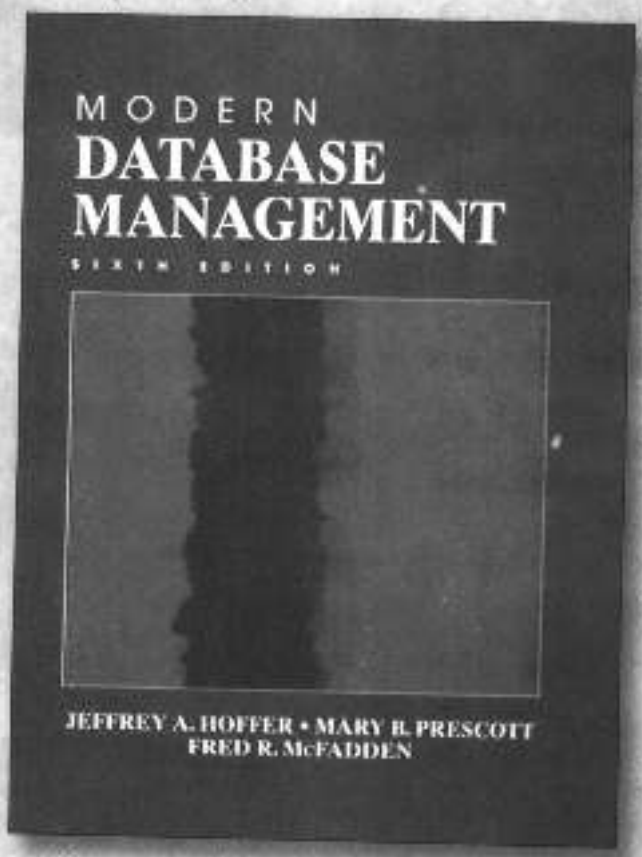
计 算 机 科 学 丛 书

原书第6版

现代数据库管理

(美) Jeffrey A. Hoffer Mary B. Prescott Fred R. McFadden 著 施伯乐 杨卫东 孙未未 等译



Modern Database Management

Sixth Edition

机械工业出版社
China Machine Press

本书是一本数据库管理的教材，内容翔实，示例丰富，由浅入深。本书从数据库管理环境、数据库分析、数据库设计、实现以及高级数据库五个方面全面介绍了数据库的知识。每章之后安排了大量的习题帮助读者梳理知识，掌握基本的概念、原理。本书还包含一个贯穿始终的实例，让读者体验数据库开发的全过程。本书的网站(http://www.prenhall.com/hoffer)中还包含大量相关资源，有助于读者复习所学知识，拓展知识面。本书适合作为相关专业的本科生、研究生的教材，也适合作为从事数据库方面工作的人员和自学者的参考书。

Simplified Chinese edition copyright © 2004 by Pearson Education Asia Limited and China Machine Press.

Original English language title: *Modern Database Management, Sixth Edition* (ISBN: 0-13-033969-5) by Jeffrey A. Hoffer, Mary B. Prescott and Fred R. McFadden, Copyright © 2002.

All rights reserved.

Published by arrangement with the original publisher, Pearson Education, Inc., publishing as Prentice Hall.

本书封面贴有Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

版权所有，侵权必究。

本书版权登记号：图字：01-2002-3644

图书在版编目（CIP）数据

现代数据库管理（原书第6版）/（美）霍弗（Hoffer, J. A.）等著；施伯乐等译. -北京：机械工业出版社，2004.9

（计算机科学丛书）

书名原文：Modern Database Management, Sixth Edition

ISBN 7-111-14517-8

Ⅰ.现… Ⅱ.①霍…②施… Ⅲ.数据库管理系统 Ⅳ.TP311.13

中国版本图书馆CIP数据核字（2004）第052289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：朱 劼

北京中兴印刷有限公司印刷·新华书店北京发行所发行

2004年9月第1版第1次印刷

787 mm × 1092mm 1/16 · 36.75印张

印数：0 001- 4 000册

定价：59.00元

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

译 者 序

数据库管理在信息化建设中具有非常重要的地位，是计算机相关专业的必修课程。计算机专业人员应该扎实地掌握数据库管理的概念、技术和方法。本书作为本科生和研究生的教材，在国际上已经成功地使用了15年。本书内容全面，由浅入深，每一章后面都配有丰富的练习，并且有一个贯穿全书的项目案例。利用每一章后面的练习和这个完整的项目案例，读者可以逐步掌握信息系统开发的理论知识和实践技能。

近十几年来，计算机技术和信息系统领域的技术迅猛发展。本书已经是第6版，在这一版中，作者尽力体现了信息系统的相关技术和方法所发生的变化，如支持Web的信息系统设计和程序设计、大规模数据库和数据仓库、通过深入的系统建模和设计来说明系统需求（如实体-联系模型，扩展的实体-联系模型和面向对象的数据建模）以及与日俱增的联机环境中数据库性能的关键性等。

本书的翻译是在复旦大学施伯乐教授的统一组织和管理下完成的，多位教师参与了本书的翻译工作。陈金海教授（第1章、第2章）、李晓荣博士（第3章）、杨卫东副教授（第4章）、谈子敬博士（第5章、第6章）、孙未未副教授（第7章、第8章、第9章）、刘卉博士（第10章、第11章）、方锦成副教授（第12章、第13章）、丁保康副教授（第14章、第15章、附录C和附录D）承担了主要的翻译工作。此外，杨卫东副教授统稿并翻译了本书的前言、缩略语表、附录A和附录B，孙未未副教授和李晓荣博士参与了部分章节的整理、补译、校对和修改工作。

特别感谢机械工业出版社的编辑杨海玲、朱劼及其同事，他们付出的辛勤劳动和提出的许多意见，尤其是朱劼编辑细致、严谨的工作，这些都是本书顺利出版的保证。

由于译者水平有限，时间紧迫，难免有翻译不当之处，希望读者批评指正。

译 者

前　言

本书可作为数据库管理的入门教材。这门课程是信息系统课程的一部分，可在商学院、计算机技术专业和应用计算机科学系讲授。信息技术专业学会（AITP）、美国计算机学会（ACM）以及国际信息处理联合会（IFIPS）的指导方针（例如IS'97）都概述了这种类型的数据库管理课程。本书以前的版本在研究生和本科生的教学以及管理和专业开发项目中，已成功地使用了15年之久。

本书在第5版的基础上进行了修订。在数据库管理领域，相关技术、管理、方法正以前所未有的速度发生着变化，为了适应这种变化，本书作出了必要的修订。但是，我们也尽量保持前面版本的优点。我们尽所有努力来体现在第4版中介绍的现代数据库管理的特点。

在第6版中，由于Fred McFadden已退休，减少了相应的工作，所以我们改变了作者的署名顺序。Fred对手稿进行了仔细的审校，并在修订方向上提出了中肯的建议。Fred McFadden是本书之父，本书自始至终都体现着他的思想。

第6版的新增内容

由于管理实践、数据库设计工具、方法学和数据库技术飞速发展，因此本书对这些领域的内容进行了更新和扩充。本书反映了信息系统领域的主要发展趋势和现代信息系统专业的学生需要掌握的技巧。

- 在总的客户/服务器体系结构内，设计支持Web的系统并进行程序设计。
- 大规模数据库和数据仓库。
- 通过完全的系统建模和设计明确系统需求。
- 在日益增加的联机环境中，数据库性能的重要性。
- SQL作为数据库查询的标准。

在所有章中，新的屏幕图反映了最新的数据库技术，新添加的“Web资源”小节列出了一些网站，这些网站补充本书涵盖的重要主题，为学生提供最新的数据库发展趋势及其背景。本书在结构上的主要变化包括：

- 介绍面向对象数据库的两章放在一起，并且放在本书的结尾。这样，这些新出现的数据库技术可以完整地呈现给读者，同时不影响其他章的连贯性。
- 本书对客户/服务器部分的改动很大，现在强调了因特网、内部网、外部网作为客户/服务器体系结构的实现。
- SQL的讨论更加深入并用两章加以介绍。
- 重写了数据仓库这一章的大部分内容，并前移到本书的实现部分，体现了这种数据库的爆炸性增长。

下面逐章给出本书的主要变化。每一章的描述说明了该章的目的，以及相对于第5版所作的变化和修订。

第一部分　数据库管理语境

- **第1章**　该章讨论了数据库在组织中的作用，并概述了本书后面各章的主要主题。该章介绍了修正的数据库的分类模式，在第5版中分为四类，即个人、工作组、部门和企业，现

在引入了因特网、内部网、外部网数据库。对企业数据库的介绍进行了扩展，包括作为ERP系统和数据仓库的一部分的数据库。该章更新了数据库技术从数据库前的文件到对象-关系数据库以及支持Web系统的历史的讨论。该章继续给出数据库技术和传统文件处理系统的比较。

- **第2章** 该章在更为宽泛的信息系统开发的语境下，详细讨论了数据库开发的作用，解释了数据库开发的结构化生命周期法和原型法。该章还讨论数据库开发的重要问题，包括数据库开发中的各种人员的管理和理解数据库结构及技术的框架。该章还强调了数据库开发中的信息工程方法学，包括企业数据模型的作用。该章内容与学生在系统分析和设计课程中所学内容是一致的。

第二部分 数据库分析

- **第3章** 该章给出了一个新标题，深入介绍如何用实体-联系模型进行概念数据建模。新的章名反映了该章强调实体-联系模型的动机，即准确描述影响数据库设计的业务规则。该章包括一个全新的小节，介绍业务规则建模的最新方法，该节中的内容详细解释了开发含义明确的实体-联系图的本质部分——如何命名和定义数据模型元素。有一个新的小节解决了很多学生在学习数据建模时所面临的一个问题：是将数据表示为属性还是表示为联系。该章继续从简单的例子过渡到复杂的例子，并以一个易于理解的松谷家具公司的E-R图结束。
- **第4章** 该章讲述几种高级的实体-联系模型的结构。该章的新内容是对实体聚簇的介绍，实体聚簇是表示实体-联系图的简单版本的一种方法。该章基于最新的指导原则，更新了GUIDE业务规则方法的内容，给出了这些指导原则的结构，这些指导原则有利于学生的理解。本章继续深入讨论超类型/子类型联系。

第三部分 数据库设计

- **第5章** 该章论述将概念数据模型转化为关系数据模型的过程。该章对外键特征进行了更详细的讨论，并介绍了非智能企业键的重要概念。该章也强调了企业键（也称为数据仓库的代理键）的概念，因为面向对象的一些概念被移植到关系数据库中。该章继续强调关系数据模型的基本概念和数据库设计人员在逻辑设计过程中的角色。
- **第6章** 该章论述获得有效数据库设计的关键步骤。该章强调改善数据库性能的方法。参考Oracle和其他DBMS使用的改善数据库处理性能的特定技术，对这几节进行了扩充。该章扩展了对索引的讨论，使其包括索引类型的描述（主索引、辅索引、联结索引和散列索引表）以更广泛地适用于数据库技术，并改善查询速度。该章对RAID的讨论作了更新，以反映该重要技术的最新思想。该章继续强调物理设计过程及过程的目标。

第四部分 实现

- **第7章** 该章全面介绍了目前大部分DBMS使用的SQL（SQL-92），并介绍了最新的SQL-99标准中的变化。第6版的主要变化是扩展了整个SQL的内容并在该章和下一章中介绍。该章包括了更多的SQL代码例子，大部分代码使用了SQL-99中的语法，一些代码使用了Oracle 8*i*的语法。本书对视图的描述进行了修改，包括动态视图和物化视图。第7章解释了创建和维护数据库的SQL命令以及编写单表查询的方法。该章继续使用松谷家具公司案例说明各种实际查询和查询结果。
- **第8章** 这是新的一章，通过多表查询、事务完整性、数据字典、触发器、存储过程、在其他编程语言程序中的嵌入式SQL继续解释SQL。该章包括各种形式的OUTER JOIN命令，说明如何在导出表中存储查询结果，如何用CAST命令在不同数据类型之间转换数据

以及在SQL中作条件处理的CASE命令。该章也讨论SQL-99作为数据仓库的数据访问工具所需的联机分析处理（OLAP）特性。和第7章一样，SQL代码的例子大部分使用SQL-99中的语法，一些使用Oracle 8*i*的语法。该章继续清晰地解释SQL中最为复杂和功能强大的概念：子查询和相关子查询。

- **第9章** 该章结合第5版的两章重写了大部分内容，目的是对客户/服务器的体系结构、应用程序、中间件和在当今数据库环境中对客户机数据库的访问进行深入讨论。该章为本书后面的关于因特网的部分提供了技术基础。该章对许多图进行了更新，以便更清楚地说明在多层网络中所能做的选择，包括应用和数据库服务器，不同层次间的数据库分布处理，以及使用浏览器的瘦客户，该章还包括一些重要的新主题，可在Web中使用的数据库的安全性、ODBC和JDBC连接（包括访问JDBC兼容数据库的详细的代码例子）。该章继续讨论三层客户/服务器体系结构，应用分割，大型主机的作用，并行计算机体系结构的使用，中间件以及Microsoft Access 2000的按例查询。同时论述和比较了对称多处理（SMP）和大规模并行处理体系结构。
- **第10章** 这是新的一章，目的是描述使用Web的应用到数据库的连接。该章包括对脚本语言和脚本语言中的嵌入式SQL的讨论，并包括了一个简单购物车应用程序，它用ASP和ColdFusion两种方法实现（在本书的网站中可以找到所有代码）。该章还讨论因特网相关的术语和连接网页和数据库必须理解的概念（如防火墙、代理服务器、静态和动态网页、HTML/SGML/XML/XHTML语言、层叠样式表、公共网关接口以及servlet）。该章解释了Web服务器的作用和数据库连接的服务器端扩展，同时涵盖Web的安全性问题和隐私问题。
- **第11章** 在这一章中，修订了第5版中的大部分内容。目的是描述数据仓库的基本概念。数据仓库被许多企业认为是取得竞争优势的关键因素，是数据仓库中特有的数据库设计活动和结构。该章的主题包括：各种可选的数据仓库体系结构、数据转换和数据一致的技术，以及用于数据仓库的多维数据模型（星型模式）。定义了运作数据存储、依赖数据集市、独立数据集市、逻辑数据集市以及各种形式的联机分析处理。该章最大的变化是讨论数据集市的数据库设计的几节内容，解释和举例说明了代理键、事实表粒度、日期和时间建模、维的一致性，以及帮助者/层次/引用表，也讨论了OLAP和数据挖掘的用户界面。

第五部分 数据库的高级主题

- **第12章** 该章充分讨论数据管理和数据库管理的重要性，并论述执行这些功能会出现的一些关键问题；强调数据管理和数据库管理的方法和改变的角色，着重强调调整数据库和查询，以提高性能。该章充分讨论数据库备份过程、数据安全威胁和响应，并详细讨论数据质量的管理。为更好地了解恢复和并发控制，该章加入了对事务完整性的讨论。该章继续强调在将管理的数据作为企业资源时，数据和数据管理的重要性。
- **第13章** 该章介绍分布式数据库的作用、技术以及数据库设计时机。该章扩充和更新对分布数据库的目标和权衡策略、数据复制策略、选择数据分布策略的因素、分布式数据库供应商和产品。该章及第12章充分讨论了数据库并发访问控制。
- **第14章** 该章采用Booch、Jacobson和Rumbaugh的统一建模语言介绍面向对象建模。该章进行了很大的修改以介绍最新UML表示方法，使用UML提供的行业标准的符号表示类和对象。该章继续强调基本OO概念，诸如继承和聚合，并包括扩展的松谷家具公司的OO数据模型的例子。

- **第15章** 该章的目的是阐述如何将面向对象模型（在第14章中介绍）转换为面向对象数据库管理系统的类、对象、关系和操作定义。该章也介绍了ODBM的标准语言，即对象定义语言（ODL）和对象查询语言（OQL）的最新格式，包含使用ODL描述的松谷家具公司数据库的面向对象数据库定义。该章最后介绍了ODBM的主要供应商和产品。

附录

本书包括四个附录，目的是让希望学习这些主题的读者能够进一步了解相关内容。

- **附录A** 这个新的附录满足了很多读者的需求，即如何将本书使用的E-R表示方法转换为课堂使用的CASE工具或DBMS所用的表示方法。该附录比较了Visible Analyst 7.4、ERwin 3.5.2、Microsoft Access 2000和Oracle Designer 6.0所使用的表示方法。附录中的表和插图说明对于同一结构，每一个流行软件包所使用的表示方法。
- **附录B** 该附录描述（使用例子）Boyce-Codd范式和第四范式。该附录还增加了关于BCNF的例子，说明如何处理交叠候选键。
- **附录C** 该附录描述了数据库实现常用的几种数据结构。主题包括指针、栈、队列、排序表、反向表和树。
- **附录D** 该附录描述了对象-关系数据库管理系统（ORDBMS）。主题包括ORDBMS的特性、扩展的SQL、对象-关系方法的优点，以及ORDBMS供应商和产品的总结。

教学

本书在各章的最后进行一系列补充和改进，为用户提供了更广泛、更丰富的选择。最重要的改进如下:

1) *复习问题* 该节包括与问题和练习中的问题配套的练习。此外，还加入很多新的问题以巩固该章新增加的内容。

2) *问题和练习* 该节在每一章都有所扩展，并包含很多新的问题与练习，以支持更新的章节内容。

3) *应用练习* 该节提供一套小案例，可以分配给每个学生或学生小组来完成。应用练习的范围从指导的应用练习到因特网搜索以及其他类型的研究练习。

4) *项目案例* 以山景社区医院案例作为学生项目，本书已经为新的和扩充的章创建了新的案例。在每一章，项目案例都以与该章内容相关的一个简短的项目描述开始，然后给出一系列需要每个学生或学生小组完成的项目问题和练习。该项目给学生提供了掌握他们学习的概念和工具的有效手段。

5) *Web资源* 每一章都包含补充该章内容的网站的列表和有用的信息。这些网站涵盖了在线的文献、厂商、电子出版物、行业标准组织以及其他资源。这些网站能够使学生和教师发现更新的产品信息，本书出版后出现的革新，深入理解主题的背景信息，并可作为编写研究论文的资源。

我们同时更新了教学特色，以使学生和教师广泛接受第6版。这些特色如下:

1) *学习目标* 出现在每一章的开始以便让学生了解将要从本章学习的主要概念和技巧。学生在准备作业和考试时能够利用学习目标很好地进行复习。

2) *本章介绍和总结* 包含每一章的主要概念和相关章节的链接内容，为学生提供关于课程的全面的概念框架。

3) *本章复习* 包括问题回顾、问题与练习、前面讨论的应用练习，也包含关键术语以测验学生对基本概念、基本事实和重要问题的掌握程度。

4) **不断出现的术语** 在本书的讨论中，会出现许多关键术语的定义。除了在正文中解释它们外，也在本书最后的术语表中给出了这些术语的定义，附录中同时包括数据库管理系统中常用的术语缩写。

组织

我们鼓励教师定制本书内容以满足学生学习的需求。本书的模块化的特点、广泛的内容，详细的说明，以及对高级主题和新问题的涵盖使得定制更为容易。对现有文献和网站的很多引用使教师可以列出补充的读物或在本书提供的资料以外开展讨论。几个高级主题包含在附录中，教师可以根据需要讲授或略过这些主题。

本书的模块化的特点使教师可以略过某些章节或以不同的顺序进行教学。例如，希望重点讲授数据建模的教师可以讲授关于面向对象数据建模的第14章以及第3章和第4章（也可以不讲授这两章）。希望仅教授基本实体-联系概念（但不包括扩展的实体-联系模型和业务规则）的教师可以跳过第4章。

补充包

本书英文原书配有下列教师资源：

教师资源CD-ROM 教师资源CD-ROM具有下列特色：

- Instructor's Resource Manual 提供各章的教学目标、教师思想以及复习问题、问题和练习、应用练习、项目案例问题的答案。Instructor's Resource Manual也可以打印出来或从本书网站上的"Faulty Area"中获取。
- Test Item File或Windows PH Test Manager 包括一套综合的测试问题，其中有多项选择、真假判断，以短问题格式按照问题的难易程度分级，并且按照原书的页码和标题顺序出现。Test Item File可以打印、在IR CD-ROM上以Microsoft Word格式存储并且可用于计算机化Prentice Hall测验管理工具。Test Manager是用于测验和评估的综合的工具集。使教师很容易创建和发布课程的测验，可用通过打印并以传统的方式发布或通过局域网服务器在线发布。Test Manager提供屏幕向导以帮助读者使用该程序，并提供充分的技术支持。
- PowerPoint幻灯片 突出关键的术语和概念。教师可以加入自己的幻灯片或编辑现有的幻灯片对其进行定制。
- 图像库 按章组织的文本。其中包括可以使用的所有的图、表和屏幕快照。
- 伴随的数据库 为本书编写的两个版本的松谷家具公司案例。一个版本匹配于本书的样例。另一个版本中包含用Visual Basic编写的表单、报表和模块。该版本并不完全，这样学生可以创建所缺的表、额外的表、报表和模块。其中也包括山景社区医院的初步版本。数据库中还包括Oracle脚本，用于创建表，插入松谷家具公司和山景社区医院的样例数据。南佛罗里达大学的Robert Lewis为我们创建了数据集和应用。该数据库文件在教师资源CD-ROM中以及本书网站的教师区内可以找到。

本书的网站 在本书的网站（http://www.prenhall.com/hoffer）中包含下列内容：

1) 可用于Access和Oracle的全新的关于松谷家具公司案例和山景社区医院案例的数据集和样例数据库应用。这部分内容在安全的Instructor's Area内提供。

2) 带有多项选择、判断、讨论问题的交互学习指导。学生在回答问题后可得到自动反馈。在学生完成测验后，讨论问题的回答、多项选择的结果、真假问题的判断都可以通过电子邮件发给教师。

3) Web Resources模块包括本书中各章后引用的Web链接，以帮助学生通过Web进一步探索数据库管理的主题。

4) 网站的学生区包括了每一章的PowerPoint演示。

5) 按字母顺序排列和按章节排列的完整的术语表，以及术语缩写。

6) 网站中加入了新的案例研究。一些案例可作为学期的课程设计。其他一些可作为教学案例。随着时间的推移，还会加入新的案例。

致谢

我们感谢众多对本书第6版的编写工作做出贡献的人们。首先，感谢本书的审阅者，感谢他们的详细建议、独特的观点以及独树一帜的教学风格。由于本书在第5版的基础上进行了大量改动，所以他们提供的对内容深度和主题的分析非常关键。他们是:

Cyrus Azarbod，州立曼凯托大学
Willard Baird，Progress Telecom公司
Michael Barrett，宾州克拉里恩大学
Douglas Bock，南伊利诺斯大学爱德华兹维尔分校
Sue Brown，印第安纳大学
Traci Carte，俄克拉荷马大学
I-Shien Chien，克赖顿大学
Kevan Croteau，Francis Marion大学
Sergio Davalos, Portland, OR
Monica Garfield，南佛罗里达大学
Mark Gillenson，孟菲斯大学
Bernard Han，华盛顿州立大学
James Harris, Francis Marion大学
Myron Hatcher，加利福尼亚州立大学弗雷斯诺分校
James Henson, Barry大学
Chang Hsieh，南密西西比大学
Constance Knapp，佩斯大学
William Korn，威斯康星大学欧克莱尔分校
Ram Kumar，北卡罗来纳大学夏洛特分校
Robert Lewis，南佛罗里达大学
Kathleen Moser，艾奥瓦州立大学
Heidi Owens，州立波特兰大学
David Paradice, Bryan, TX
Fred Prose, Phoenix, AZ
John Russo，Wentworth理工学院
Siva Sankaran，加利福尼亚州立大学诺斯里奇分校
Werner Schenk，罗切斯特大学
Richard Segall，阿肯色州立大学
Maureen Thommes，Bemidja州立大学
Heikki Topi，本特雷学院

Chelley Vician，密歇根技术大学
Charles Wertz, Buffalo州立大学
Surya Yadav, Lubbock, TX
Ahmed Zaki，威廉和玛丽学院
Han Zhang，佐治亚理工学院

我们从业内人士那里也获取了极好的建议，他们是Todd Walter、Carrie Ballinger、Rob Armstrong、Dave Schoeff（他们都就职于NCR公司）、Patty Melanson、Kelly Carrigan、Joanna Frankenberry（他们都就职于Catalina Marketing）、Don Berndt（南佛罗里达大学）、Bernadette Lynch（Oracle公司）以及Michael Alexander（Open Access技术公司）。

我们还要感谢华盛顿州立大学的Joe Valacich和佛罗里达州立大学的Joey George，感谢他们对数据库开发与整个信息系统开发之间关系的独特观点给我们带来的帮助。他们的严谨有助于本书与系统分析和设计课程使用的书籍保持一致，如Hoffer、George和Valacich所著的*Modern Systems Analysis and Design*，Valacich、George和Hoffer（Prentice Hall）所著的*Essentials of Systems Analysis and Design*。

我们也感谢威斯康星大学密尔沃基分校的Atish Sinha，他编写了关于面向对象数据库建模和实现的第14章和第15章的原始版本。我们衷心地感谢他所做的工作：使用松谷家具公司和山景社区医院案例将这两章集成到本书中，编写用于比较OO方法、实体-联系和关系的方法的练习，以及为所有章节提供的建议。

本书包含更多的样例数据库应用，相对于前面任何版本，教师和学生更容易利用这些实例进行学习。我们感谢南佛罗里达大学的Robert Lewis编写的松谷家具公司和山景社区医院案例。同时感谢Open Access技术公司的Michael Alexander所编写的因特网购物车的ASP和ColdFusion版本。

南佛罗里达大学的Laura Biasci撰写了附录A中的四个数据建模CASE工具符号的比较。我们感谢他工作到深夜并牺牲周末，在他的计算机上运行四种软件，因此，他能够比较并给出不同的符号约定。

我们也非常感谢Prentice Hall的同事们在整个项目期间的支持和指导。尤其感谢主编Bob Horan整理本书的规划，副主编Lori Cerreto保证每一部分都是完整的，感谢制作编辑Mike Reynolds、高级营销经理Sharon Turkovich，以及营销助理Jason Smith。特别是University Graphics Production Services的Terri O'Prey，他出色的编辑管理是本书出版的保证。

Jeff和Mary为能够与Fred McFadden一起工作并向他学习而兴奋。Fred是本书之父，本书清晰的结构、语言的通俗易懂、精心的组织，以及主流、现代的内容都与他的工作密不可分。Fred以学生为本的观点和严谨的著述是我们的榜样。Fred对第6版帮助极大，并极力支持这次修订，我们非常感谢他。

最后，感谢我们的伴侣。对他们的感谢难以言表，因为他们忍受了许多孤独的夜晚和周末。我们特别感谢Patty Hoffer的奉献，作为本书作者的伴侣，从第1版到第6版，她给予了极大的支持。现在，在本书的两个版本中，Larry Prescott给予了支持。本书的成功应归功于他们的忍耐、鼓励和爱。但是，对本书任何内容的任何错误，我们独自承担责任。

Jeffrey A. Hoffer
Mary B. Prescott
Fred R. McFadden

目　录

译者序
前言

第一部分　数据库管理语境

第1章　数据库环境……2
1.1　学习目标……2
1.2　引言……2
1.3　基本概念和定义……3
1.3.1　数据……3
1.3.2　数据与信息……3
1.3.3　元数据……4
1.4　传统文件处理系统……5
1.4.1　松谷家具公司的文件处理系统……5
1.4.2　文件处理系统的缺点……6
1.5　数据库方法……7
1.6　数据库应用的范围……11
1.6.1　个人数据库……12
1.6.2　工作组数据库……13
1.6.3　部门数据库……13
1.6.4　企业数据库……14
1.6.5　因特网、内部网和外部网数据库……16
1.6.6　数据库应用小结……16
1.7　数据库方法的优点……17
1.7.1　程序-数据独立性……17
1.7.2　数据冗余度最小……17
1.7.3　改善数据一致性……17
1.7.4　改善数据共享……17
1.7.5　提高应用开发的生产率……17
1.7.6　标准的实施……18
1.7.7　改善数据质量……18
1.7.8　改善数据可访问性和响应性……18
1.7.9　减少程序维护……18
1.7.10　关于数据库优点的告诫……18
1.8　数据库方法的成本和风险……18
1.8.1　新的专门人员……19
1.8.2　安装、管理成本和复杂性……19
1.8.3　转换成本……19
1.8.4　需要清晰备份和恢复……19
1.8.5　组织冲突……19
1.9　数据库环境的组成部分……19
1.10　数据库系统的演变……20
1.10.1　20世纪60年代……21
1.10.2　20世纪70年代……21
1.10.3　20世纪80年代……21
1.10.4　20世纪90年代……22
1.10.5　2000年以来……22
本章小结……22
本章复习……23
项目案例：山景社区医院……27
第2章　数据库开发过程……29
2.1　学习目标……29
2.2　引言……29
2.3　信息系统开发中的数据库开发……30
2.3.1　信息系统体系结构……30
2.3.2　信息工程……31
2.3.3　信息系统规划……31
2.4　数据库开发过程……34
2.4.1　系统开发生命周期……35
2.4.2　信息系统开发的其他方法……38
2.4.3　计算机辅助软件工程的作用和信息库……39
2.5　数据库开发中的人员管理……40
2.6　数据库开发的三层模式体系结构……41
2.7　三层数据库定位体系结构……44
2.8　为松谷家具公司开发一个数据库应用……45
2.8.1　匹配用户需求和信息系统体系结构……46
2.8.2　分析数据库需求……48
2.8.3　设计数据库……50
2.8.4　使用数据库……52
2.8.5　管理数据库……54
本章小结……54
本章复习……55

项目案例: 山景社区医院 ……60

第二部分 数据库分析

第3章 组织中的数据建模 ……68
3.1 学习目标 ……68
3.2 引言 ……68
3.3 根据组织中的规则建立数据模型 ……69
3.3.1 业务规则概述 ……70
3.3.2 确定业务规则的范围 ……70
3.3.3 数据命名与定义 ……71
3.4 E-R模型 ……74
3.4.1 E-R图示例 ……74
3.4.2 E-R模型符号 ……76
3.5 实体-联系模型的结构 ……77
3.5.1 实体 ……77
3.5.2 属性 ……81
3.6 联系 ……86
3.6.1 联系的基本概念和定义 ……87
3.6.2 联系的度 ……88
3.6.3 属性还是联系 ……92
3.6.4 基数约束 ……92
3.6.5 建立依赖于时间的数据模型 ……96
3.6.6 实体之间的多种联系 ……98
3.6.7 命名和定义联系 ……99
3.7 E-R建模的例子: 松谷家具公司 ……100
3.8 松谷家具公司的数据库处理 ……101
3.8.1 显示产品信息 ……103
3.8.2 显示顾客信息 ……103
3.8.3 显示顾客订单状态 ……103
3.8.4 显示产品销售 ……105
本章小结 ……105
本章复习 ……106
项目案例: 山景社区医院 ……113
第4章 增强型E-R模型和业务规则 ……115
4.1 学习目标 ……115
4.2 引言 ……115
4.3 超类型和子类型的表示 ……116
4.3.1 基本概念和表示方法 ……116
4.3.2 特化和概化的表示 ……119
4.4 指定超类型/子类型联系之间的约束 …122
4.4.1 指定完备性约束 ……122
4.4.2 指定不相交约束 ……124
4.4.3 定义子类型鉴别符 ……125
4.4.4 定义超类型/子类型层次 ……127
4.5 增强型EER建模示例: 松谷家具公司 …128
4.6 实体聚簇 ……131
4.7 再论业务规则 ……134
4.7.1 业务规则的分类 ……135
4.7.2 陈述结构断言 ……136
4.7.3 陈述动作断言 ……137
4.7.4 表示和强制业务规则执行 ……138
4.7.5 标识和测试业务规则 ……140
本章小结 ……141
本章复习 ……142
项目案例: 山景社区医院 ……146

第三部分 数据库设计

第5章 逻辑数据库设计和关系模型 ……150
5.1 学习目标 ……150
5.2 引言 ……150
5.3 关系数据模型 ……150
5.3.1 基本定义 ……151
5.3.2 数据库示例 ……152
5.4 完整性约束 ……154
5.4.1 域约束 ……154
5.4.2 实体完整性 ……155
5.4.3 参照完整性 ……155
5.4.4 动作断言 ……156
5.4.5 创建关系表 ……156
5.4.6 良构关系 ……157
5.5 将EER图转化为关系 ……158
5.5.1 第1步: 映射常规实体 ……159
5.5.2 第2步: 映射弱实体 ……161
5.5.3 第3步: 映射二元联系 ……162
5.5.4 第4步: 映射关联实体 ……164
5.5.5 第5步: 映射一元联系 ……166
5.5.6 第6步: 映射三元 (多元) 联系 ……168
5.5.7 第7步: 映射超类型/子类型联系 …170
5.6 规范化介绍 ……171
5.6.1 规范化的步骤 ……171
5.6.2 函数依赖和键 ……173
5.7 基本范式 ……174

5.7.1 第一范式 ……174
5.7.2 第二范式 ……174
5.7.3 第三范式 ……175
5.7.4 规范化概要数据 ……177
5.8 合并关系……177
5.8.1 例子 ……178
5.8.2 视图集成产生的问题 ……178
5.9 定义关系键的最后步骤……179
本章小结……180
本章复习……183
项目案例：山景社区医院……189
第6章 物理数据库设计和性能 ……190
6.1 学习目标……190
6.2 引言……190
6.3 物理数据库设计步骤……191
6.4 设计字段……193
6.4.1 选择数据类型 ……193
6.4.2 数据完整性控制 ……194
6.5 设计物理记录和非规范化……195
6.6 设计物理文件……201
6.6.1 指针 ……201
6.6.2 文件组织 ……202
6.6.3 文件组织小结 ……208
6.6.4 聚簇文件 ……209
6.6.5 设计文件控制 ……210
6.7 索引的使用和选择……210
6.7.1 创建惟一键索引 ……210
6.7.2 创建辅键索引 ……210
6.7.3 何时使用索引 ……211
6.8 RAID：通过并行处理来改善文件访问的性能……212
6.9 数据库设计……216
6.10 优化查询性能 ……219
6.10.1 并行查询处理……219
6.10.2 对自动查询优化的重载……220
6.10.3 选择数据块大小……220
6.10.4 在磁盘控制器间平衡I/O操作 ……221
6.10.5 设计良好查询的建议……221
本章小结……223
本章复习……224
项目案例：山景社区医院……229

第四部分 实 现

第7章 SQL ……233
7.1 学习目标……233
7.2 引言……233
7.3 SQL标准的发展……234
7.4 数据库体系结构中SQL的作用……234
7.5 SQL环境……236
7.6 用SQL定义数据库……239
7.6.1 SQL数据库定义 ……239
7.6.2 创建表 ……240
7.6.3 使用和定义视图 ……241
7.6.4 创建数据完整性控制 ……245
7.6.5 修改表的定义 ……246
7.6.6 删除表 ……246
7.7 插入、更新和删除数据……246
7.7.1 批量输入 ……247
7.7.2 删除数据库内容 ……247
7.7.3 修改数据库内容 ……248
7.8 RDBMS中的内模式定义 ……248
7.9 处理单个表……249
7.9.1 SELECT语句的子句 ……249
7.9.2 使用表达式 ……251
7.9.3 使用函数 ……251
7.9.4 使用通配符 ……252
7.9.5 比较运算符 ……253
7.9.6 使用布尔运算符 ……253
7.9.7 范围 ……254
7.9.8 DISTINCT ……255
7.9.9 IN 和 NOT IN列表 ……256
7.9.10 排序结果：ORDER BY子句 ……257
7.9.11 分类结果：GROUP BY子句 ……258
7.9.12 通过分类限定结果：HAVING子句……258
本章小结……259
本章复习……260
项目案例：山景社区医院……266
第8章 高级SQL ……267
8.1 学习目标……267
8.2 引言……267
8.3 处理多表……267

8.3.1 等值联结 ……268
8.3.2 自然联结 ……269
8.3.3 外联结 ……269
8.3.4 并联结 ……270
8.3.5 例子：涉及4张表的多重联结……271
8.3.6 子查询 ……271
8.3.7 相关子查询 ……274
8.3.8 使用导出表 ……275
8.3.9 组合查询 ……276
8.3.10 条件表达式……277
8.4 保证事务完整性……277
8.5 数据字典工具……279
8.6 SQL-99对SQL的增强和扩展 ……280
8.6.1 已建议加入的分析函数 ……280
8.6.2 程序设计能力扩展 ……281
8.7 触发器和例程……282
8.7.1 触发器 ……283
8.7.2 例程 ……283
8.8 嵌入式SQL和动态SQL ……285
本章小结……286
本章复习……286
项目案例：山景社区医院……290
第9章 客户/服务器数据库环境……291
9.1 学习目标……291
9.2 引言……291
9.3 客户/服务器结构 ……292
9.3.1 文件服务器体系结构 ……292
9.3.2 文件服务器的局限性 ……293
9.3.3 数据库服务器体系结构 ……294
9.4 三层体系结构……295
9.5 分割一个应用……296
9.6 大型主机的作用……297
9.7 使用并行计算机体系结构……298
9.7.1 多处理器硬件结构 ……299
9.7.2 与业务有关的SMP和MPP结构的使用 ……301
9.8 使用中间件……302
9.9 建立客户/服务器的安全性 ……303
9.10 客户/服务器的问题 ……305
9.11 客户端应用程序的数据库存取 ……306
9.12 使用按例查询 ……307
9.12.1 QBE的历史和重要性……307
9.12.2 QBE：基本知识……308
9.12.3 选择合格的记录……310
9.12.4 自联结……311
9.12.5 让一个查询基于另一个查询……313
9.12.6 使用SQL传递查询……313
9.13 使用ODBC来链接存储在数据库服务器上的外部表 ……315
9.14 使用JDBC来链接存储在数据库服务器上的外部表 ……317
9.15 在客户端应用程序中使用VBA ……321
本章小结……322
本章复习……323
项目案例：山景社区医院……328
第10章 因特网数据库环境……330
10.1 学习目标 ……330
10.2 引言 ……330
10.3 因特网和数据库连接 ……331
10.3.1 因特网环境……332
10.3.2 术语……333
10.4 常见的因特网体系结构的组成 ……334
10.4.1 与因特网相关的语言……334
10.4.2 服务器端扩展……335
10.4.3 Web服务器接口 ……336
10.4.4 Web服务器 ……337
10.4.5 客户端扩展……337
10.5 Web-数据库工具：ColdFusion与ASP…338
10.5.1 ASP示例……338
10.5.2 ColdFusion示例 ……342
10.5.3 嵌入式SQL……345
10.6 管理Web数据 ……347
10.6.1 Web安全性问题 ……347
10.6.2 隐私问题……349
10.6.3 因特网技术的更新速度问题……350
本章小结……351
本章复习……352
项目案例：山景社区医院……355
第11章 数据仓库 ……356
11.1 学习目标 ……356
11.2 引言 ……356
11.3 数据仓库的基本概念 ……357

11.3.1 数据仓库的历史简介……358
11.3.2 为什么需要数据仓库 ……358
11.4 数据仓库的体系结构 ……361
11.4.1 一般的两层体系结构 ……361
11.4.2 独立数据集市的数据仓库环境 …362
11.4.3 依赖数据集市和运作数据存储的体系结构……363
11.4.4 逻辑数据集市和主动仓库体系结构……365
11.4.5 三层数据体系结构……366
11.5 数据仓库中数据的若干特征 ……368
11.5.1 状态数据与事件数据……368
11.5.2 临时数据与周期数据……368
11.5.3 一个临时数据和周期数据的例子…369
11.6 调和数据层 ……370
11.6.1 进行ETL之后的数据特征 ……371
11.6.2 ETL过程 ……371
11.7 数据转换 ……375
11.7.1 数据转换函数……375
11.7.2 支持数据调和的工具……377
11.8 导出数据层 ……379
11.8.1 导出数据的特征……379
11.8.2 星型模式……380
11.8.3 星型模式的变体……385
11.8.4 维表的规范化……387
11.8.5 缓慢变化的维度……389
11.9 用户界面 ……390
11.9.1 元数据的作用……390
11.9.2 查询工具……391
11.9.3 联机分析处理工具……391
11.9.4 数据挖掘工具……392
11.9.5 数据可视化……394
本章小结……394
本章复习……395
项目案例：山景社区医院……401

第五部分 数据库的高级主题

第12章 数据管理与数据库管理 ……404
12.1 学习目标 ……404
12.2 引言 ……404
12.3 数据管理员与数据库管理员的作用 …405
12.3.1 传统的数据管理……405
12.3.2 传统的数据库管理……405
12.3.3 数据管理与数据库管理方法的演化……405
12.3.4 数据管理方法的演化……407
12.4 企业数据的建模 ……410
12.5 数据库的规划 ……410
12.6 数据安全性的管理 ……411
12.6.1 数据安全性的威胁……411
12.6.2 视图……413
12.6.3 完整性控制……414
12.6.4 授权规则……415
12.6.5 用户自定义过程……416
12.6.6 加密……416
12.6.7 认证模式……417
12.7 数据库的备份 ……417
12.7.1 基本的恢复工具……417
12.7.2 恢复与重启动过程……419
12.7.3 数据库故障的类型……422
12.8 并发访问的控制 ……424
12.8.1 更新丢失的问题……424
12.8.2 可串行性……425
12.8.3 加锁机制……425
12.8.4 版本设置……428
12.9 数据质量的管理 ……429
12.9.1 安全性策略与灾难恢复……430
12.9.2 人员控制……431
12.9.3 物理访问控制……431
12.9.4 维护控制……431
12.9.5 数据保护与私密性……431
12.10 数据词典与信息库 ……431
12.11 数据库性能调整概述 ……433
12.11.1 安装DBMS ……434
12.11.2 内存利用 ……434
12.11.3 输入/输出争用 ……434
12.11.4 CPU利用 ……435
12.11.5 应用软件调整 ……435
本章小结……435
本章复习……436
项目案例：山景社区医院……442

第13章 分布式数据库 ……443
13.1 学习目标 ……443
13.2 引言 ……443
13.3 数据库实施分布式处理的策略 ……447
13.3.1 数据复制……448
13.3.2 水平分割……450
13.3.3 垂直分割……451
13.3.4 操作组合……452
13.3.5 选择正确的数据分布策略……453
13.4 分布式DBMS ……454
13.4.1 位置透明性……455
13.4.2 复制透明性……457
13.4.3 故障透明性……457
13.4.4 提交协议……457
13.4.5 并发透明性……458
13.4.6 查询优化……459
13.4.7 分布式DBMS的发展 ……461
13.5 分布式数据库管理系统产品 ……462
本章小结……463
本章复习……464
项目案例：山景社区医院……469
第14章 面向对象数据建模……470
14.1 学习目标 ……470
14.2 引言 ……470
14.3 统一建模语言 ……472
14.4 面向对象数据建模 ……472
14.4.1 表示对象和类……472
14.4.2 操作的类型……474
14.4.3 表示关联……475
14.4.4 表示关联类……478
14.4.5 表示导出属性、导出关联和导出角色……480
14.4.6 表示概化……480
14.4.7 解释继承和重载……485
14.4.8 表示多重继承……486
14.4.9 表示聚合……486
14.5 业务规则 ……489
14.6 对象建模实例：松谷家具公司 ……489
本章小结……492
本章复习……493
项目案例：山景社区医院……498
第15章 面向对象数据库开发 ……500
15.1 学习目标 ……500
15.2 引言 ……500
15.3 对象定义语言 ……501
15.3.1 定义类……501
15.3.2 定义属性……502
15.3.3 定义用户结构……502
15.3.4 定义操作……503
15.3.5 为属性定义范围……503
15.3.6 定义联系……503
15.3.7 定义以对象标识符作为值的属性…505
15.3.8 定义多对多联系、键和多值属性…506
15.3.9 定义概化……508
15.3.10 定义抽象类 ……509
15.3.11 定义其他用户结构 ……510
15.4 松谷家具公司的OODB设计 ……511
15.5 创建对象实例 ……512
15.6 对象查询语言 ……513
15.6.1 基本的检索命令……514
15.6.2 在select子句中包含操作 ……514
15.6.3 查找不同的值……514
15.6.4 查询多个类……515
15.6.5 编写子查询……515
15.6.6 计算概要值……516
15.6.7 计算分组概要的值……516
15.6.8 在查询中使用集合……517
15.6.9 OQL的小结 ……518
15.7 当今ODBMS产品和它们的应用 ……518
本章小结……519
本章复习……519
项目案例：山景社区医院……522

第六部分 附 录

附录A E-R建模工具和符号 ……523
附录B 高级范式 ……530
附录C 数据结构 ……535
附录D 对象-关系数据库 ……546
术语缩写 ……550
术语表 ……554

第一部分　数据库管理语境

概要

第一部分包含两章，这两章介绍了数据库环境并给出以后将用到的数据库基本概念和定义。在这部分里，我们将数据库管理描绘为一个快速增长、充满挑战而激动人心的领域，它为信息系统专业的学生提供大量就业机会。数据库日益成为日常生活中不可或缺的部分和商务运作的更核心的部分，从个人数字助理（PDA）中存储联系信息的数据库到支撑整个企业信息系统的大型数据库，数据库已经成为几十年前人们所预料的数据存储的中枢。客户关系管理和因特网购物是近几年发展起来的两个依赖于数据库活动的例子。

第1章介绍数据、数据库、元数据、数据仓库、内部网（intranet）、外部网（extranet）和与数据库环境有关的其他术语的定义。我们将数据库和已被取代的老的文件管理系统作比较并描述使用周密规划的数据库能够获得的显著优势。本章描述个人、工作组、部门、企业、因特网/内部网/外部网数据库的典型数据库应用及其特征。企业数据库包括企业资源规划系统和数据仓库。我们还描述数据库环境的主要组成部分，这些组成部分将在以后各章中进行更详细的阐述。最后，我们将简要介绍数据库系统（包括对象-关系数据库系统和支持Web的系统）的发展历史，以及驱动它们发展的动力。这一章引用了松谷家具公司的例子，用来说明数据库管理的许多原理和概念。这个例子作为数据库管理系统应用的一个连续的例子还会在以后各章使用。

第2章描述在数据库的分析、设计、实现和管理中所遵循的一般步骤。这一章也阐述数据库开发过程如何适应整个信息系统开发过程，解释数据库开发的结构化生命周期法和原型法。该章还介绍企业数据建模，它设定组织数据库的范围和一般内容，因此，它常常是数据库开发的第一步。该章同时介绍信息系统体系结构的概念，它可用来作为组织信息系统的蓝图，我们还将描述和解释信息工程，它是用来建立和维护信息系统的面向数据的方法学。数据库开发中的问题，包括数据库开发中的各种人员的管理和理解数据库体系结构及技术的框架也在该章中进行了描述。

在这一章我们描述和解释系统开发生命周期以及另一个迭代开发过程即原型法的应用，阐述在信息系统开发过程中使用计算机辅助软件工程（CASE）工具和信息库的重要性，介绍作为现代数据库系统中主流方法的模式和三层模式体系结构的概念。最后，我们描述数据库开发项目中经常出现的各种人员的角色。

在第一部分中，重要的是清楚地理解所提出的各种不同的概念和定义。这些概念和定义将贯穿本书的其余部分。为了保证充分理解这些概念和定义，应完成每章后的若干复习题和练习，并就不清楚的地方请教老师。

第1章　数据库环境

1.1　学习目标

学完本章后，读者应该具备以下能力：

- 定义下列关键术语：数据库、数据、数据库管理系统、信息、元数据、企业数据模型、企业资源规划系统、外部网、内部网、遗留数据、数据库应用、数据仓库、数据独立性、信息库、用户视图和约束。
- 解释数据库的数量为什么能持续增长。
- 列举出常规文件处理系统的若干局限性。
- 识别五类数据库及必须对每一类数据库作出的关键决定。
- 与传统文件处理相比较，说明数据库方法的至少六个优点。
- 指出数据库方法的若干代价与风险。
- 列出并简要说明典型数据库环境的九个组成部分。
- 简要描述数据库系统的演变。

1.2　引言

在过去20年中，数据库应用的数量和重要性的巨大增长是有目共睹的。包括商业、医疗、教育、政府和图书馆在内的几乎每一类组织都用数据库存储、处理和检索数据。数据库技术一般由个人用在个人计算机上，由工作组用在网络服务器上访问数据库以及由企业范围分布式应用系统的所有员工使用。

在数据库应用的快速增长期之后，对数据库和数据库技术的需求是否将趋于平稳？当然不是。在21世纪初的高度竞争环境中，种种迹象表明数据库技术将更加重要。管理者为了在竞争中获得利益，将寻求使用来源于数据库的知识。例如，可以通过挖掘详细的销售数据库以决定顾客购物模式，并将其作为广告和市场促销的基础。当前许多组织为了这类决策支持应用正在构筑称为“数据仓库”的单独的数据库（Lambert，1996）。

使用数据库来支持客户关系管理、在线购物和员工关系管理显得日益重要。对于当前大多数信息系统，从个人数字助理和信息电器中的小型数据库到支持企业范围信息系统的大型数据库，数据库都是它们的基础。

尽管数据库的未来是确定的，但仍有许多工作要做。在许多组织中，不相容的数据库激增，这些数据库是为满足当时的需要而开发的，但并没有基于良好的规划，也未对其演化过程进行有效的管理。大部分数据限制在更老的“遗留”系统中，且数据质量常常很差。数据仓库的设计也需要新技术，在诸如数据库分析、数据库设计、数据管理、数据库管理领域所用的技术还存在关键性缺陷。本书将陈述这些问题和其他重要问题。

数据库管理课程是现代信息系统全部课程中最重要的课程之一。作为一名专业的信息系统人员，必须在信息系统开发的过程中去分析数据库需求，设计和实现数据库。还必须与终端用户交流，向他们说明如何使用数据库（或数据仓库）去建立能在竞争中获得优势的决策支持系统和高层管理人员信息系统。为了给网站用户返回动态信息，支持网站的数据库已得到广泛应

用，这不仅需要了解数据库如何连接到网络，而且应该了解如何保障这些数据库的安全，即它们的内容可以浏览但不能被外界用户不受约束地获取。

本章介绍数据库和数据库管理系统(DBMS)的概念，阐述传统文件管理系统及它们的一些缺点，并以此引出数据库方法。本章描述数据库应用的范围，从个人计算机和数字助理到工作组、部门和企业数据库。接下来介绍使用数据库方法的益处、代价和风险。最后，本章总结了数据库系统的发展历史及构造、使用和管理数据库的常用技术。本章的内容将在本书以后各章中详细讨论。

1.3 基本概念和定义

数据库（database）是逻辑相关的数据的有组织的集合。一个数据库可以具有任意大小和任意复杂度。例如，推销员可以在他的笔记本电脑上维护关于顾客联系方式的几兆字节数据的小型数据库，一个大公司可以在用于决策支持应用的大型主机上构造拥有几太字节（1太字节 = 1万亿字节）数据的巨型数据库（Winter，1997）。巨型数据仓库包含超过一拍字节（1拍字节 = 1千万亿字节）的数据（全书假设所有数据库都是基于计算机的）。

1.3.1 数据

在历史上，术语*数据*（data）是能记录和存储在计算机媒介上的已知事实。例如，在推销员的数据库中，数据可以包括诸如顾客姓名、地址和电话号码的事实。现在应该扩展这个定义以反映新的现实。除了传统的文本和数值数据外，当今的数据库也可以存储诸如文件、照片、声音甚至视频段等对象。例如，推销员的数据库可以包括顾客的照片，还可以包括最近与顾客联系的录音或视频剪辑。为了反映这个新现实，我们使用以下的扩展定义：**数据**由用户环境中有意义的事实、文本、图形、图像、声音和视频构成。

我们已将数据库定义为相关数据的有组织的集合。所谓*有组织*（organized）是指数据是结构化的，以便用户可以方便地存储、操纵和检索。所谓*相关*（related）是指数据为一组用户描述了他们感兴趣的领域，这组用户能够利用这些数据回答有关该领域的问题。例如，汽车修理商店的数据库包含能够识别顾客（其数据项包括顾客姓名、地址、单位电话、家庭电话及信用卡号）及其汽车（其数据项包括制造商、型号和年份）以及每辆车的维修史（例如，维修日期、维修人员姓名、维修类型和维修费用）的数据。

1.3.2 数据与信息

术语*数据*与*信息*是紧密相关的，事实上它们经常可以互换使用。然而，区分数据和信息通常是有意义的。我们定义*信息*（information）为以能够增加使用者知识的方式处理的数据。例如，考虑下列事实：

Baker, Kenneth D.	324917628
Doyle, Joan E.	476193248
Finkle, Clive R.	548429344
Lewis, John C.	551742186
McFerran, Debra R.	409723145
Sisneros, Michael	392416582

这些事实符合数据的定义，但大多数人认为这种形式的数据是无用的。即使可以猜出这是带有社会安全号的人名列表，这些数据还是无用的，因为不知道这些条目意味着什么。注意，当我们把相同的数据置于某种语境之中（如图1-1a所示）时发生了什么。通过增加少量附加数据项和提供若干结构，我们可以看出这是一门课程的一个班级花名册。它对于一些用户，例如

课程教师和注册办公室工作人员来说是有用的。

将数据转换为信息的另一种方式是汇总这些数据，或者为了人们的解释而处理和描述这些数据。例如，图1-1b显示了以图形信息表示的汇总的注册学生数据。这个信息可作为决定是否增加新课程或雇用新教员的基础。

在实际中，按照我们的定义，目前的数据库可以包括数据或信息（或两者兼有）。例如，一个数据库可以包括如图1-1a所示的班级花名册文档。通常还可以对数据进行预处理，并以汇总形式存储在用于决策支持的数据库中。在本书中，我们使用*数据库*这一术语时不区分其内容是作为数据还是信息。

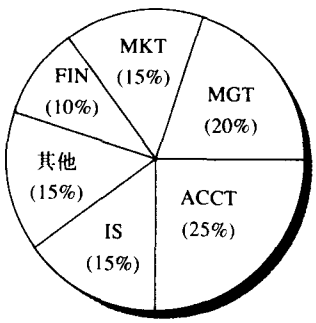
班级花名册

课程：　MGT 500　Business Policy　　学期：Spring 200X

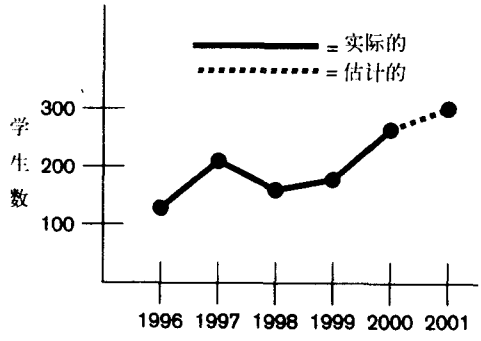
班级：　2

姓名	ID	专业	GPA
Baker, Kenneth D.	324917628	MGT	2.9
Doyle, Joan E.	476193248	MKT	3.4
Finkle, Clive R.	548429344	PRM	2.8
Lewis, John C.	551742186	MGT	3.7
McFerran, Debra R.	409723145	IS	2.9
Sisneros, Michael	392416582	ACCT	3.3

a) 在某种语境下的数据



已注册的专业所占的百分比



历年招生计划

b) 汇总的数据

图1-1　转换数据为信息

1.3.3　元数据

前面说到过，数据只有处在一定的语境中才会有意义。为数据提供语境的主要机制是元数据。**元数据**（metadata）是一种描述其他数据的性质或特征的数据。这些性质包括数据定义、数据结构和规则或约束。

表1-1列出班级花名册（图1-1a）的样本元数据。对于班级花名册中的每一个数据项，元数据显示该数据项的数据名、数据类型、长度、允许的最小值和最大值（在需要的地方给出）以及每个数据项的简要描述。注意数据和元数据之间的区别：元数据是从数据中提取出来的，也

就是说，元数据描述数据的性质但不包括数据。因此，表1-1显示的元数据不包括图1-1a所示的班级花名册的任何样本数据。

表1-1 班级花名册的样本元数据

数据项		长 度	值		说 明
名 称	类 型		最小值	最大值	
Course	字母数字	30			课程ID和名称
Section	整数	1	1	9	班号
Semester	字母数字	10			学期和年份
Name	字母数字	30			学生姓名
ID	整数	9			学生ID(SSN)
Major	字母数字	4			学生专业
GPA	十进制数	3	0.0	4.0	学生学分平均积分点

1.4 传统文件处理系统

基于计算机的数据处理的初期是没有数据库的。塞满了几个大房间的计算机的性能比现在的个人计算机还差得多，它们几乎均专门用于科学和工程计算。计算机逐渐地被引入到商业领域。为了用于商业应用，计算机必须能够存储、操纵和检索大量数据文件。于是，计算机文件处理系统应运而生。几十年来，虽然这些系统有所发展，但它们的基本结构和目标几乎没有改变。

由于商业应用变得更加复杂，传统的文件处理系统所存在的许多缺点和局限性（在后文中介绍）也凸显出来。因此，在当今最关键的商业应用中，数据库处理系统取代了文件处理系统。由于下列原因，对于文件处理系统至少应有一定的了解:

1) 现在，文件处理系统仍在广泛使用，特别是数据库备份系统。

2) 理解文件处理系统中固有的问题和局限性能够帮助我们在设计数据库系统时避免同样的问题。

本节余下部分通过实例描述文件处理系统并讨论它们的局限性。下节用同样的例子介绍数据库处理系统并作比较。

1.4.1 松谷家具公司的文件处理系统

松谷家具公司制造高质量的全木家具并分销到全国范围各零售商店。在该公司的若干产品系列中有计算机桌、娱乐中心用具、餐厅家具、书橱和可在墙壁安装的家具。顾客通过电话、邮件、传真或因特网向松谷家具公司提交订单。该公司目前拥有约100名员工，并正在快速增长。

松谷家具公司的早期计算机应用使用传统的文件处理方法。此时，信息系统的设计重点集中在某个部门的数据处理需求上，而不是评估组织的整体信息需求。信息系统组对于用户对新系统的需求，通常是依靠开发（或获得）特定应用所需的计算机程序来满足的，这些应用包括库存控制、应收账户和人力资源管理。所开发的每个应用程序或系统都要符合特定部门或用户组的需求。因而不存在指导应用增长的全面蓝图、计划或模型。

图1-2显示了基于文件处理方法的三个计算机应用。图中说明的系统是Order Filling（订单填写）、Invoicing（开发票）和Payroll（薪资）。该图也显示了与每个应用有关的主要数据文件。文件是相关记录的集合。例如，订单填写系统有三个文件：顾客主文件、库存主文件和退回订单文件。注意，在三个应用中所有的文件都存在部分重复，这在文件处理系统中是常见的。

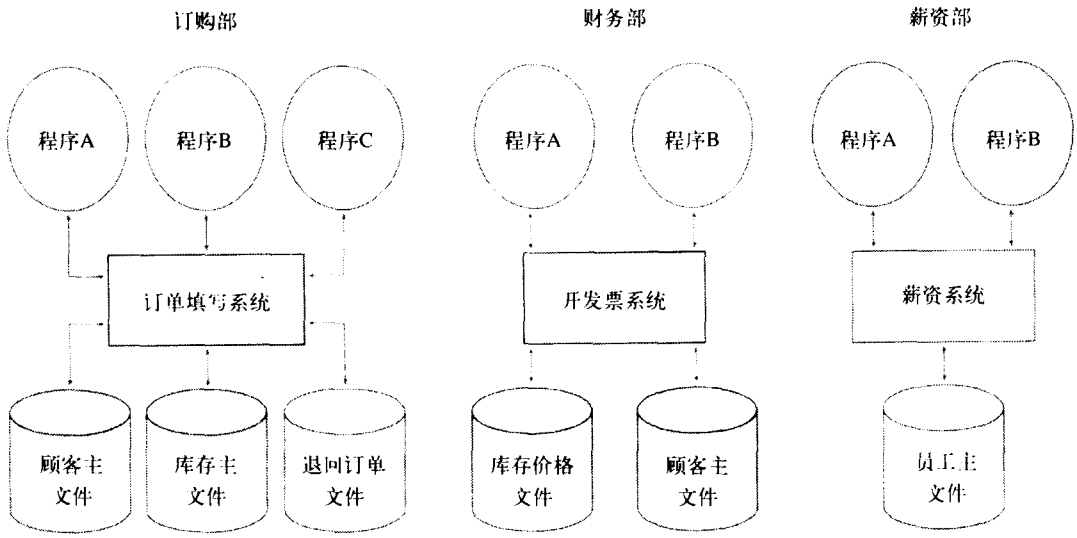


图1-2　松谷家具公司的三个文件处理系统

1.4.2　文件处理系统的缺点

传统的文件处理系统存在若干缺点。表1-2列出了这些缺点，下面将分别说明这些缺点。

1. 程序-数据依赖性

文件描述存储在每一个访问该文件的应用程序中。例如，在图1-2的开发票系统中，程序A不但可以访问库存价格文件，还可以访问顾客主文件。因此，在程序中包含一个关于这两个文件的详细描述。结果，对文件结构作出的任何改变都需要改变访问该文件的所有程序的文件描述。

表1-2　文件处理系统的缺点

程序-数据依赖性
数据重复
数据共享有限
开发时间长
过多的程序维护

注意图1-2，订单填写系统和开发票系统都使用了顾客主文件。假设决定在该文件中将记录的顾客地址字段长度从30个字符改变为40个字符。受影响的每个程序（5个文件）中的文件描述都不得不加以修改。确定受这种改变影响的全部程序是很困难的。更糟的是，进行改变时通常会引入错误。

2. 数据重复

由于在文件处理系统中应用程序常常是独立开发的，所以通常都会出现无计划的重复数据文件。例如，在图1-2的订单填写系统包含了库存主文件，同时，开发票系统包含了库存价格文件，这些文件毫无疑问地包含了描述松谷家具公司产品的数据，诸如产品描述、价格和现有数量。这个重复是多余的，因为它需要附加存储空间，同时，为使所有文件保持最新状态还要增加工作量。更为甚者，重复数据文件常导致数据完整性丢失，这是由于数据格式可以不一致或数据值可以不一致（或两者兼有）。如此无计划和无控制的冗余也导致元数据完整性的丢失。例如，在不同的文件中相同数据项可以有不同的名字或相同名字在不同文件中用于不同数据项。

3. 数据共享有限

在传统文件处理方法中，每个应用都有它自己专用的文件，用户不大可能共享自己应用之外的数据。例如，在图1-2中，财务部的用户可以访问开发票系统及其文件，但他们可能不访问订单填写系统或薪资系统及其文件。如果需要一份报告，那么就要进行大量的编程工作才能

得到不同系统中若干不兼容文件的数据，这将使管理者很头疼。此外，由于不同的组织单位可以有自己不同的文件，所以在管理工作上也要花费较大的精力。

4. 开发时间长

使用传统的文件处理系统，在前一个阶段中的成果不大可能在后面的阶段中得到重用。每个新应用都要求开发者必须从设计新文件的格式和描述开始，然后编写每个新程序的文件访问逻辑。冗长的开发时间与当今快速发展的商业环境极不协调，现在，市场时机（或对于信息系统的产品时机）是关键的商业成功因素。

5. 过多的程序维护

前面所有因素的组合使依赖传统文件处理系统的组织产生了沉重的程序维护负担。事实上，在这样的组织中，整个信息系统开发预算的几乎80%是用在程序维护上。这几乎未留下开发新应用的机会。

需要注意的是，如果一个组织没有适当地应用数据库方法，那么我们提到的文件处理的许多缺点也可能成为数据库的局限。例如，一个组织开发了许多各自独立的没有元数据配合的管理数据库（比如说，为每个部门或业务职能部门创建一个数据库），那么无控制的数据重复、有限的数据共享、冗长的开发时间和过多的程序维护也存在。因而，下一节讲述的数据库方法同样是一种管理组织的数据的方法，它是一组定义、创建、维护和使用数据的技术。

1.5 数据库方法

数据库方法强调整个组织（或至少组织的大部分部门）数据的集成和共享。这个方法要求在思考过程中，需要从顶层管理开始，改变考虑问题的方向或转换考虑问题的角度。这样的改变对于大多数组织来说是困难的，然而许多组织正在作这种转换并认识到信息可以作为一种竞争的武器。

松谷家具公司的数据库方法

20世纪90年代初期，家具制造业的竞争日趋激烈，竞争者对于新商机的反应似乎比松谷家具公司更快。虽然造成这种状况有许多原因，但管理者感到他们曾经使用的（基于传统的文件处理）计算机信息系统已经过时了。公司开始采用数据库方法进行开发。本章简要描述此方法，第2章将作更详细的描述。

1. 企业数据模型

松谷家具公司在转换到数据库方法的第一步是列出支持该公司业务活动的高层实体。实体（entity）是一个对业务十分重要的对象或概念。松谷家具公司涉及的一些高层实体有：CUSTOMER（顾客）、PRODUCT（产品）、EMPLOYEE（员工）、CUSTOMER ORDER（顾客订单）和DEPARTMENT（部门）。

确定和定义了这些实体后，公司继续开发企业数据模型。**企业数据模型**（enterprise data model）是一个显示该组织的高层实体及实体间联系的图形化模型。图1-3展示了包含4个实体和3个有关关联的企业实体模型片段。该模型片段中表示的实体是：

- CUSTOMER 购买或将来可能购买松谷家具公司产品的个人和团体。
- ORDER 顾客购买产品的订单。
- PRODUCT 松谷家具公司制造和销售的产品。
- *ORDER LINE* 某个顾客所购每件产品的明细（例如，数量和价格）。

图中表示的关联（数据库术语中称为联系，即连接实体的3条线）捕获了3个基本业务规则，如下所示：

1) 每个CUSTOMER（顾客）可以提交许多份ORDER（订单）。相反，每份ORDER（订单）只能由一个CUSTOMER（顾客）提交。

2) 每份ORDER（订单）包含许多ORDER LINE（订单明细）。相反，每个ORDER LINE（订单明细）只能包含在一份ORDER（订单）中。

3) 每个PRODUCT（产品）可有许多ORDER LINE（订单明细）。相反，每个ORDER LINE（订单明细）只能针对一个PRODUCT（产品）。

这里的“提交”（Place）、“可有”（Has）和“包含”（Contain）称为一对多联系，因为一个顾客潜在地可提交多份订单，而一份订单只能属于一个顾客。

图1-3所示的这类图称为*实体-联系图*（entity-relationship diagram）。实体-联系（E-R）图在数据库应用中非常重要，第3章和第4章专门介绍这类模型。第3章描述基本E-R图和数据建模（data modeling），第4章描述高级数据建模。这两章还将描述各类业务规则，以及规则的获取方法和建模方法。

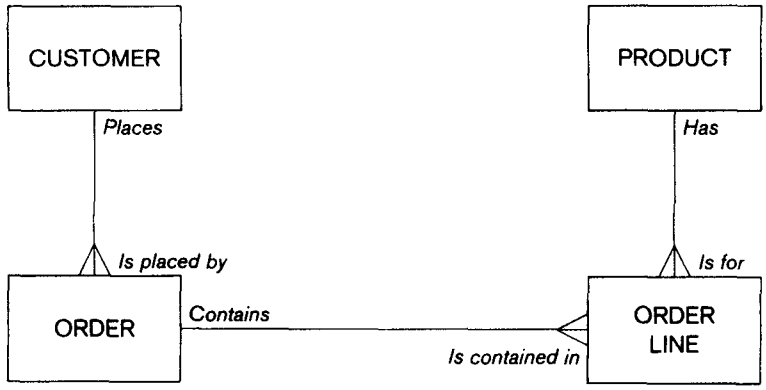


图1-3　企业数据模型片段（松谷家具公司）

注意，企业数据模型有下列特点：

1) 这是一个提供了关于组织功能的有用信息和重要限制的组织模型。

2) 通过关注实体、联系和业务规则，企业数据模型强调数据和处理的*集成*（integration）业务规则。

2. *关系数据库*

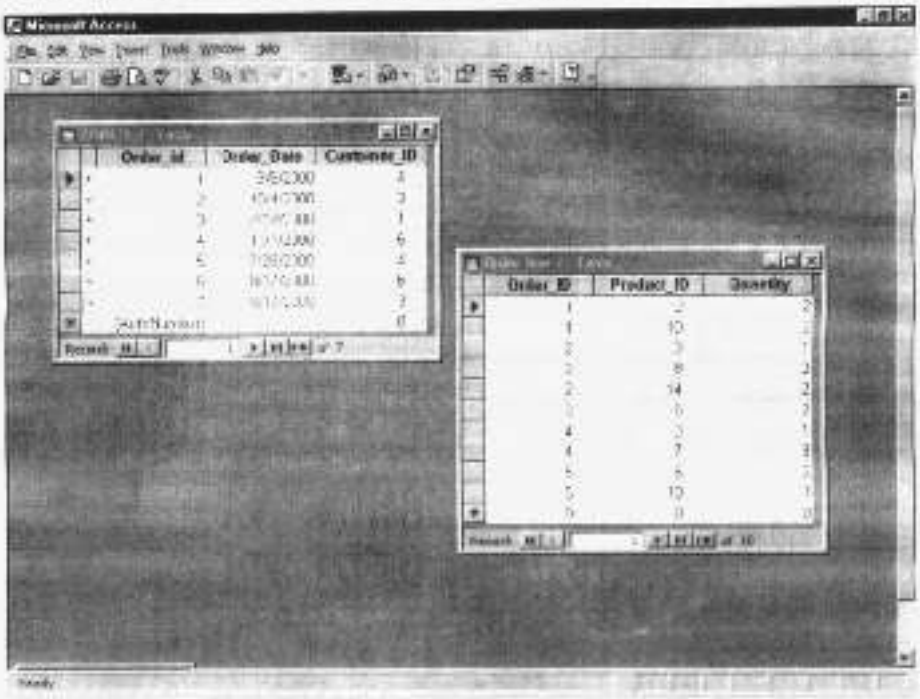
经过初步学习，我们可以确信数据库方法在管理上有许多潜在的优点。该公司决定实现以表格形式查看所有数据的现代关系数据库管理信息系统（第5章讲述关系数据库）。图1-4显示了具有样本数据的四个表：Customer表、Product表、Order表和Order_Line表。这些表也表示企业数据模型（见图1-3）展示的四个实体。

表的每一列表示实体的一个属性（或特征），例如，Customer表的属性有Customer_ID、Customer_Name、Address、City、State和Postal_Code。表的每一行表示实体的一个实例（或取值）。关系模型的一个重要性质是它通过存储在相应表的列的值表示实体间的关系。例如，Customer_ID是Customer表和Order表的一个公共属性。因此，可以容易地将订单与相应的顾客联系起来。再例如，可以确定Order_ID3与Order_ID2有关。能够确定哪个Product_ID与Order_ID2有关呢？以后的各章将介绍如何使用有效的查询语言从这些表中检索数据。

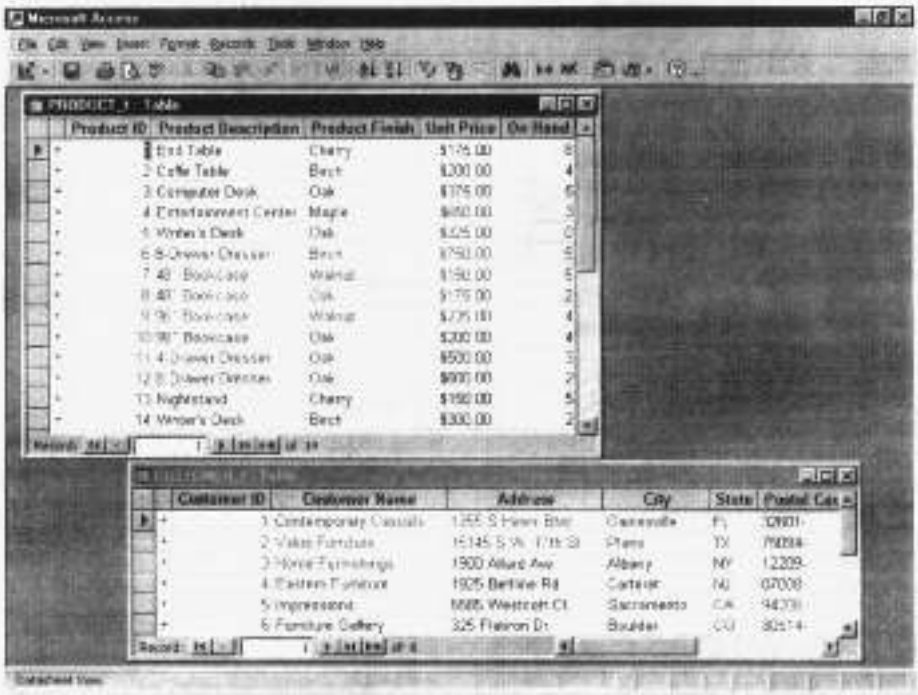
3. *关系数据库的实现*

数据库方法与老的文件处理系统相比提供了许多潜在的优点。然而，如果该组织只是实现

一系列孤立的数据库，那么只能体现其中的少数优点。因为它不允许组织的不同成员共享数据。事实上，孤立的数据库有许多与文件处理系统相同的缺点。



a) Order和Order_Line表



b) Product和Customer表

图1.4 四个关系（松谷家具公司）

为了方便共享数据和信息，松谷家具公司使用了将不同部门员工的工作站链接到数据库服务器的局域网（LAN），如图1-5所示。为了改善企业内部的通信和决策过程，建立了使用基于Web的因特网技术和仅限于企业内部访问的内部网。每个员工的工作站可以作为Web浏览器，快速访问包括电话号码簿、家具设计规格、电子邮件等在内的公司信息。工作站还可以用作个人计算机，并根据需要通过局域网访问数据库服务器。

松谷家具公司期望在它的业务应用中增加一个Web界面，例如，订单录入页面，以便于更多的内部业务活动能够通过内部网进行管理。他们还计划通过因特网使顾客的访问变得更方便，许多销售松谷家具公司家具系列的家具零售商希望通过因特网访问该公司，了解库存家具的货源，然后，零售商可向他们的顾客提供这方面的信息。向零售商显示库存信息就产生了信息安全问题，这将在第12章陈述，第10章也会讨论平台问题。

尽管数据库能够充分支持松谷家具公司的日常操作，然而没过多久，经理们就发现同样的数据库对于决策支持的应用来说常常是不充分的。例如，通过数据库不能容易地回答下列问题：

1）与去年同期相比，今年销售家具的模式是什么？

2）谁是前十位的最大顾客？他们购买的模式是什么？

3）哪类家具趋向于一起销售（例如，桌子与椅子）？



图1-5　松谷家具公司的计算机系统

为了回答这些问题和其他问题，一个组织通常应该建立一个包含历史和汇总信息的单独的数据库。这样的数据库通常称为数据仓库（data warehouse），或有时称为数据集市（data mart）。分析人员需要专门的决策支持工具来查询和分析数据库。用于此目的的一类工具称为联机分析处理（或OLAP）工具。第11章将阐述数据仓库、数据集市和决策支持工具。

4. 数据库应用

数据库应用是一个应用程序（或一组相关程序），它常常代表数据库用户执行一系列活动。每个数据库应用执行下列一些基本操作：

1）**创建**（create） 增加新数据到数据库。

2）**读取**（read） 读取当前数据库数据（常常以有用的形式表现在计算机屏幕或打印好的报表上）。

3）**更新**（update） 更新（或修改）当前数据库数据。

4）**删除**（delete） 从数据库删除当前数据。

松谷家具公司的重要数据库应用之一是创建顾客发票。当订单准备送达顾客时，相应的发票也准备好了，该发票汇总所有的订单项并计算出总额。图1-6显示了一个典型的顾客发票。除非原始的订单发生改变，否则根据数据库（参见图1-4）中的数据自动生成发票。

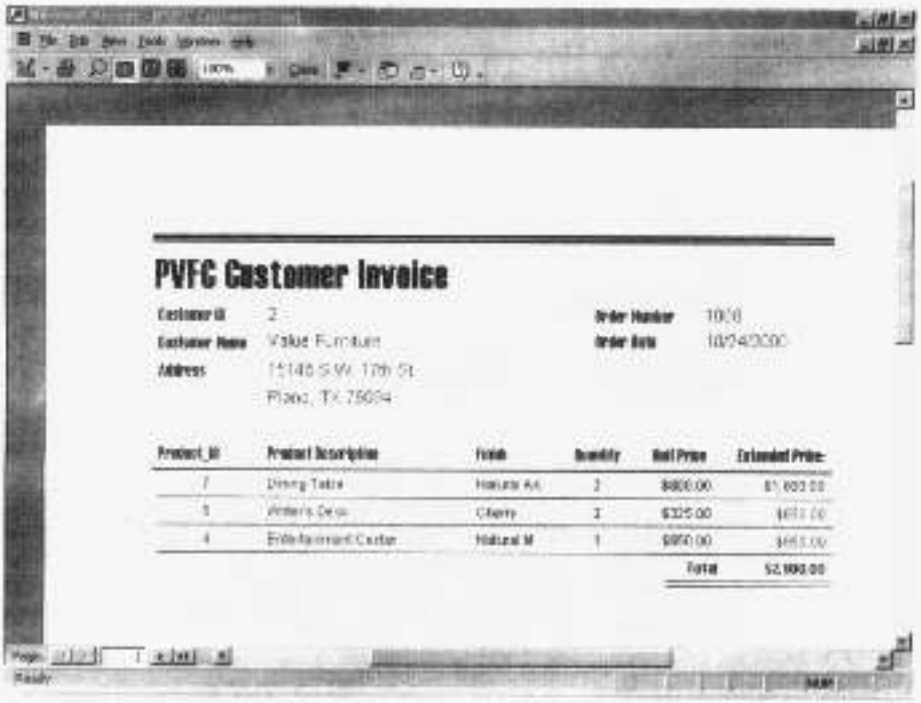


图1-6 顾客发票（松谷家具公司）

1.6 数据库应用的范围

数据库应用范围可涉及从具有台式计算机或个人数字助理的单个用户到拥有数千用户的大型计算机。数据库应用范围可以分为五类：个人数据库、工作组数据库、部门数据库、企业数据库和因特网、内部网和外部网数据库。以下通过典型例子介绍每一类数据库应用。

1.6.1 个人数据库

个人数据库是专用于某一个用户的。个人数据库常驻留于包括笔记本电脑在内的个人计算机（PC）中。近来，个人数字助理（PDA）的出现将个人数据库和手持设备结合起来，不仅具有计算设备的功能，而且可作为蜂窝电话、传真（FAX）机和Web浏览器使用。存有顾客信息和每个顾客详细联系方式的简单数据库应用能够用于PC或PDA，并可容易地从一个设备转移到另一个设备，以便进行备份和其他工作。例如，考虑一个拥有许多销售员的公司，这些销售员访问现有或未来顾客。如果每个销售员还有其他应用，例如，一个包含很多图形的销售演示程序和一个帮助销售员为顾客定货而确定物品类型和数量最佳组合的定价程序，由于存储和性能的需求，应当使用一台笔记本电脑。假如销售员只需要保存他们的联系人列表，那么一台使用小型数据库、具有联系管理应用的PDA可能是最佳的。图1-7显示了顾客联系人列表的典型数据。

顾客

顾客姓名: **Multi Media, Inc.**	
地址: **1000 River Road**	
城市: **San Antonio**	
州: **TX**	
邮编: **76235**	
电话: **(219) 864-2000**	
下次联系日期: **10/17/2000**	时间: **10:30 AM**

与顾客联系的历史记录

日　期	时　间	联系人	事　项
08/04/2000	10:00 AM	Roberts	审阅计划书
08/19/2000	08:00 AM	Roberts	修订时间表
09/10/2000	09:00 AM	Pearson	签署合同
09/21/2000	02:00 PM	Roberts	监督工作进展

图1-7　取自个人数据库的典型数据

在开发个人数据库时，必须作出的若干关键决定是：

1) 该应用是从供应商处购买还是组织自行开发？

2) 如果数据库应用是内部开发，那么它该是由终端用户开发还是由信息系统（IT）部门的专业人员开发？

3) 用户需要什么数据？数据库如何设计？

4) 该应用应使用什么样的商业数据库管理系统（DBMS）产品？

5) 个人数据库中的数据如何与其他数据库中的数据同步？

6) 谁来负责个人数据库中的数据准确性?

个人数据库之所以广泛使用，这是因为它常常能够提高个人生产率。然而，它必然伴有风险：数据不能容易地与其他用户共享。例如，销售经理想要一个顾客联系人的完整视图，而这个视图不能很快地或方便地从某个销售员的数据库中产生。这说明一个非常普通的问题：如果数据对于一个人来说是有用的，那么该数据对于其他人来说也可能（或将可能）是有用的。因此，个人数据库应仅限于在特定情况下使用，即个人数据库用户之间共享数据的需求较少（例如，在非常小的组织中）。

表1-3 工作组数据库示例中的软件对象列表

Object Place,Inc.
400 Magnolia St.
Atlanta, GA 02103

名 称	语 言	描 述	价 格
123Xtender	Visual Basic	电子表格包装器	595
DSSObjects	C++	决策支持生成器	595
ObjectSuite	Smalltalk	6个普通对象集	5000
OrderObject	Smalltalk	普通订单对象	1000
PatientObject	Smalltalk	普通病人对象	1000

1.6.2 工作组数据库

工作组（workgroup）是指在同一项目或应用中，或者一组类似项目或应用中合作的相对较少的一组人员。通常工作组的人数不超过25人。这些人可能从事（例如）建筑项目或开发新的计算机应用。工作组数据库用来支持这组人协同工作。

考虑一个开发标准对象和定制对象（或软件组件）并出售给软件商或终端用户的工作组。表1-3是一份近来开发的软件对象列表。通常是一个或多个人在给定时间内为开发指定的对象或组件而工作。这组人需要一个数据库，该数据库将跟踪开发过程中每一事项并允许工作组成员方便地共享数据。

图1-8显示了在数据库中共享数据的方法。每个工作组成员都有一台笔记本电脑并通过局域网（LAN）连接起来。数据库存储在与局域网连接的称为**数据库服务器**（database server）的中央设备中。因此，每个工作组成员都可以访问共享数据。不同类型的组成员（例如，开发者或项目经理）可以有该共享数据库的不同用户视图。注意，这种安排克服了PC数据库的数据不容易共享（至少，数据不容易在工作组内共享）的主要缺陷。然而，这种安排产生了许多在个人（单用户）数据库中不存在的数据管理问题，例如，数据安全性和并发用户数据更新时的数据完整性。由于一个组织由许多工作组构成，个别人在同一或不同时间可以是许多不同工作组的一部分，所以产生许多数据库是可能的，这和个人数据库的情况相同。

在建立工作组数据库时，组织必须回答建立个人数据库时的相同问题。此外，还出现了下列数据库管理问题:

1) 如何设计数据库才能最优地满足许多工作组成员的信息需求?

2) 不同成员如何能够并发使用数据库而不破坏数据库完整性?

3) 哪些数据库处理操作应在工作站运行? 哪些应在服务器运行?

1.6.3 部门数据库

部门（department）是一个组织内的职能单位。部门的典型例子有人事部、营销部、生产部和财务部。一般说来，部门比工作组规模大（通常有25～100人）并负责更多不同的职能。

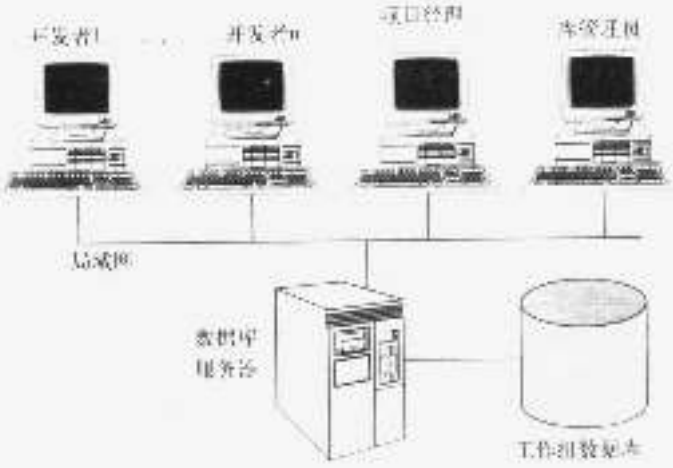


图1-8　局域网上工作组数据库

部门数据库应支持部门的各种不同的职能和活动。它们是本节描述的五类数据库中最常见的一类。例如，考虑一个记录有关员工、工作、技能和工作分配的人事数据库。一旦有关数据存入数据库，那么用户就能够查询数据库并得到下列问题的答案：

1) 对于特定的工作分类（例如，软件工程师），公司里目前存在何种工作机会？

2) 对于同一工作分类，需要哪些工作技能？

3) 特定的员工应该具有什么技能？相反地，哪些员工应该具有特定技能（例如，C++程序设计）？

4) 哪些员工被分配了特定的工作？相反地，某个员工的工作经历是什么？

5) 哪些员工受某个经理管理？

在设计和实现部门数据库时，必须解决的典型问题（除已经描述的外）如下：

1) 假设存在大量用户和用户事务，为达到足够性能，数据库及其环境应如何设计？

2) 为了防止非授权泄露或敏感数据外传，应如何提供足够的安全性？

3) 在这复杂的环境中应该使用什么样的数据库和应用开发工具？

4) 其他部门是否也在维护相同类型的数据，如果是这样，如何能够最好地管理数据冗余、数据一致性和元数据？

5) 是否数据库用户在地理上分散或数据库规模很大，以至于数据必须存储在若干个计算机系统中，于是，必须创建分布式数据库（distributed database）吗？

1.6.4　企业数据库

企业数据库是一个范围包括整个组织或企业（或至少是许多不同部门）的数据库。这样的数据库支持组织范围的运作和决策制定。注意，一个组织可以有若干个企业数据库，因此，这种企业数据库不包括整个组织的数据。由于超大型数据库性能上的局限、不同用户的不同要求和为所有数据库用户实现单一数据（元数据）定义的复杂性，所以单一的、运行的企业数据库对于许多中到大型的组织是不可行的。然而，企业数据库确实应该支持不同部门的信息需求。近十年间企业数据库的演变导致了两方面的主要发展：

1) 企业资源规划系统。

2) 数据仓库实现。

企业资源规划（ERP）系统从20世纪70年代到20世纪80年代的材料需求规划（MRP）和生

产资源规划（MRP-Ⅱ）演变而来。MRP系统安排生产过程的原材料、组件和零部件的需求，而MRP-Ⅱ系统还包括车间调度和产品分发调度。后来，企业职能的扩展促使企业范围的管理系统，即企业资源规划系统的生成。所有ERP系统都强烈地依赖于数据库来存储ERP应用所需要的数据。

ERP系统利用企业当前运作数据进行工作。**数据仓库**（data warehouse）的内容源自包括个人数据库、工作组数据库和部门数据库在内的不同运作数据库。数据仓库使用户有机会利用历史数据去识别模式和趋势并回答商务战略方面的问题。第11章将详细介绍数据仓库。

考虑一个大型医疗组织，它拥有若干医疗中心，包括医院、诊所和疗养院。这些医疗中心都有各自的数据库（或数据库组）以支持该组织的运行，如图1-9所示。这些数据库包括有关病人、医生、医疗服务、商务运作和其他有关事项的数据。

数据库能为每个医疗中心的大多数职能提供足够支持。然而，该组织认识到应该有一个整个组织的数据库的完整视图，如可以查看某个病人或供应商的全部活动。例如，为所有医疗单位集中定货以及跨单位进行人员和服务调度，能够获得高操作效率。ERP系统使得这种方法成为可能。公司制定决策、与外部供应商（例如保险公司）打交道、向不同的代理商报告都需要编辑历史数据和信息。为了满足这些要求，组织创建了在公司总部办公室维护的数据仓库。通过从个别数据库抽取和汇总数据，将数据仓库中的数据定期导出，并利用远程通信网络将这些数据传送到企业的数据仓库。

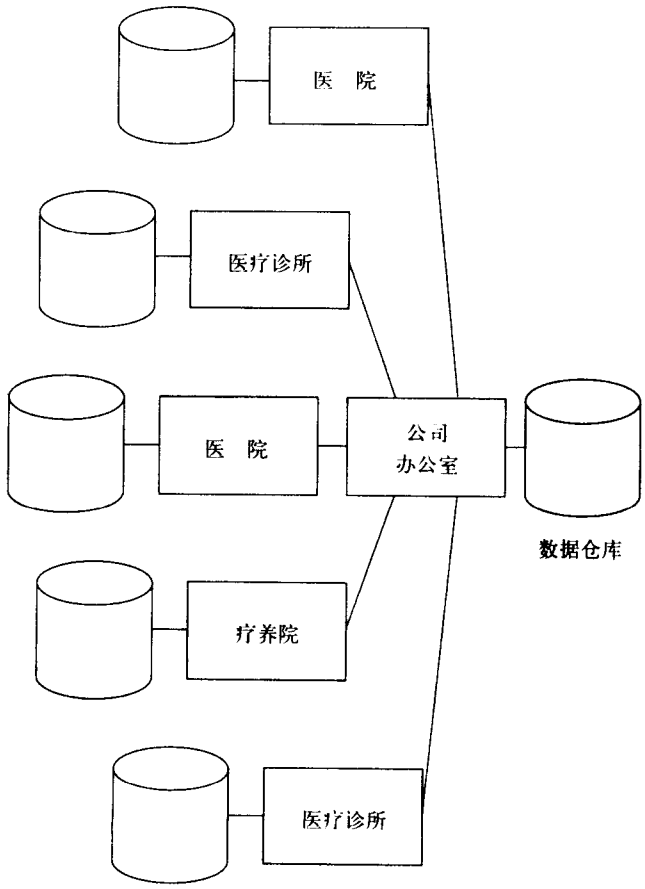


图1-9 一个企业数据仓库

在企业数据库中常产生的若干问题如下：

1) 数据应该如何在公司结构的不同位置传送？

2) 组织如何开发和维护关于数据名字、定义、格式及有关事项的标准？

3) 为了集成包括分析所需的早先系统**遗留数据**（legacy data）的众多系统，必须采取什么行动？

1.6.5 因特网、内部网和外部网数据库

影响数据库环境的最近的变化是因特网的增长，因特网是一个很容易通过Web浏览器界面连接多平台用户的世界范围的网络。在商务上应用因特网使早已建立的商务模式有了重大改变。曾经非常成功的公司受到了新企业的挑战，这些新企业使用因特网以提供改善的顾客信息和服务，代替了传统的营销渠道和传送渠道并实现了员工关系管理。顾客直接向计算机制造商选配和订购他们的最新个人计算机。航空公司机票和收藏品的报价在提交的几分钟后就可以收到，有时可以节省40%以上的时间。关于招聘和公司活动的信息在许多公司内都很容易获得。

这些应用都需要数据库支持，许多应用需要通用访问。多平台的方便连接允许公司重新组织他们的运作，并且更快、更低成本地开发新应用。标准界面允许用户进行较少的培训和较少的支持就可以轻松使用。开发实用的应用程序的核心在于能够从一个与应用连接的数据库中检索当前信息。当一个数据库是支持Web（Web数据库）的，Web浏览器界面允许用户询问惟一的和特殊的问题并基于当前信息获得答案。这些问题的回答是由程序自动完成的，不需要通过电话进行一系列选择或者人工提供帮助。Web数据库对于在线购物站点的开发是必不可少的。为了改善客户关系管理（CRM），公司正在收集有关其顾客的信息（购买模式、站点导航和每个屏幕停留时间等）。

上面所引用的大部分例子反映了B2C关系。因为有些企业的客户是其他企业，所以他们的相互作用通常称为B2B关系。因特网常用于B2C联系，因为顾客对于企业来说必定是外部的，顾客访问企业数据或信息的能力对于顾客与企业之间的成功联系是十分关键的。允许这种外部对企业数据库的访问会为信息系统管理带来新的数据安全性和完整性问题，因为通常数据被严密保护在每个企业中。第10章和第12章将详细讨论这些问题。

企业多年来利用电子数据交换（EDI）取得交易信息。许多企业继续使用EDI系统去指导他们的B2B业务。有些企业，特别是新企业或以前在内部信息交换中未使用EDI的企业，通常建立**外部网**（extranet）来执行B2B交换。外部网使用因特网技术，但是，和所有人都能访问因特网应用不同，外部网不是任何人都能访问的。更确切地说，访问仅限于拥有合法访问权限和使用相互数据和信息协定的企业供应商和顾客。通常，这些供应商和顾客访问**内部网**（intranet）的一部分（以后讨论）。这种访问提供对信息的更快更有效的处理或访问，方便业务关系。

如前所述，许多公司使用因特网技术建立专用网以便为组织进行信息管理。从表面上看，内部网页面与因特网页面类似，但是，对页面的访问仅限于组织内的页面。因此，对于公司数据库的访问是受限制的。内部网也能够建立与因特网的连接，但该连接受防火墙保护，以防止外部用户与内部网连接。

1.6.6 数据库应用小结

本节描述的各类数据库应用的小结如表1-4所示。该表显示各类数据库的典型用户数、典型数据结构（客户/服务器（C/S）结构，在第9章介绍）和典型数据库规模的范围。

表1-4 数据库应用小结（改编自White，1995）

数据库类型	典型用户数	典型结构	数据库典型规模
个人数据库	1	台式/笔记本电脑、PDA	MB级
工作组数据库	5～25	C/S（2层）	MB～GB级
部门数据库	25～100	C/S（3层）	GB级
企业数据库	>100	C/S（分布式或并行服务器）	GB～TB级
因特网	>1000	Web服务器和应用服务器	MB～GB级

1.7 数据库方法的优点

数据库方法相比传统的文件处理系统提供了许多潜在的优点。主要优点见表1-5，下面将这些优点概述如下。

表1-5 数据库方法的优点

程序-数据独立性
数据冗余度最小
改善数据一致性
改善数据共享
提高应用开发的生产率
标准的实施
改善数据质量
改善数据可访问性和响应性
减少程序维护

1.7.1 程序-数据独立性

数据描述（元数据）和使用该数据的应用程序分离称为**数据独立性**（data independence）。利用数据库方法，数据描述存储在系统信息的信息库（repository）中。数据库系统的性质允许在不改变处理数据的应用程序情况下，更新和演变组织的数据（在一定的限度内）。

1.7.2 数据冗余度最小

数据库方法的设计目标是将先前单独（或冗余）的数据文件集成到一个逻辑结构中。每个基本事实记录在数据库的惟一位置。例如，Product_ID 3产品是一台价值375美元的橡木计算机桌这一事实存储在Product表中（参见图1-4b）。数据库方法不会完全消除冗余性，但它允许设计者小心地控制冗余的类型和数量。例如，Order表（图1-4a）中的每份订单包含了Customer_ID以利于建立订单与顾客之间的联系。有时也可包括一些有限的冗余以改善数据库性能，以后章节将看到这种情况。

1.7.3 改善数据一致性

通过消除（或控制）数据冗余，极大地减少了不一致的机会。例如，假如顾客地址只存储一次，那么就不会出现存储值不一致的情况。当每个值仅存储一处时，修改数据值也会大大地简化。最后，避免了冗余数据存储所造成的存储空间浪费。

1.7.4 改善数据共享

数据库是一个共享的、共同的资源。授权的内部和外部用户有权使用数据库，每个用户（或用户组）都可以有一个或多个用户视图以方便使用。**用户视图**（user view）是用户执行一些任务所需要的数据库某些部分的逻辑描述。例如，松谷家具公司数据库的每一个表（见图1-4）都能构成一个用户视图。然而，用户视图常常是这样一个表单或报表，它包含了来自多个表的数据。例如，顾客发票（见图1-6）也是该数据库的一个用户视图。在松谷公司的网站上可获得的产品目录是另一个用户视图。

1.7.5 提高应用开发的生产率

数据库方法的主要优点是大大降低了开发新业务应用的成本和时间。数据库应用比传统文件应用开发速度快得多的两个重要原因是：

1）假定数据库、相关数据的获取和维护应用已经设计和实现，程序员可以集中精力实现新应用需要的特殊功能上，而不必担心文件设计或低层实现细节。

2) 数据库管理系统提供了许多高级的高生产率工具，例如表单和报表生成器，以及能使数据库设计和实现的一些活动自动化的高级语言。在随后的各章中将介绍许多这类语言。

1.7.6 标准的实施

当用数据库方法提供全面的管理支持时，对数据库管理部门的职能应予以单一授权，并责成建立和实施各种数据标准。这些标准包括命名约定、数据质量标准及访问、修改和保护数据的一致性过程。数据信息库为数据库管理员提供一组开发和实施标准的强有力的工具。但是，不能实现强有力的数据库管理功能或许是组织的数据库失败最普遍的根源。第12章将介绍数据库管理（和有关数据的管理）功能。

1.7.7 改善数据质量

数据质量低劣是当今数据库管理的普遍问题（Ballou和Tayi, 1999）。数据库方法提供了许多改善数据质量的工具和过程。最重要的是以下两点：

1) 数据库设计者能够指定由DBMS实施的完整性约束。**约束**（constraint）是数据库用户不能违背的规则。第3章和第4章将介绍多类约束（也称为业务规则）。

2) 数据仓库环境的目标之一是在将数据放入数据仓库之前清洗（或“擦洗”）运作数据（Jordan,1996）。第11章将介绍数据仓库及其改善数据质量的潜力。

1.7.8 改善数据可访问性和响应性

没有程序设计经验的终端用户也通常能够利用关系数据库检索和显示数据，甚至跨越传统的部门界线。例如，员工使用下列查询能够显示松谷家具公司的有关计算机桌的信息：

```
SELECT *
FROM PRODUCT
WHERE Product_Name = "Computer Desk";
```

该查询中使用的语言称为结构化查询语言，简称为SQL（第7章、第8章将详细介绍该语言）。

1.7.9 减少程序维护

由于各种原因，如新数据类型的加入、数据格式的改变等等，存储的数据必须经常地改变。这个问题的著名例子是有名的“2000年”问题，即将常见的2位数字年字段扩展为4位数字以适应从1999年到2000年的转变。

在文件处理环境中，数据的描述和访问数据的逻辑内嵌在某个应用程序中（这是前面所述的程序-数据依赖性问题）。因此，改变数据格式和访问方法不可避免地造成修改应用程序。在数据库环境中，数据更多地独立于使用它的应用程序。在某个限度内，我们可以改变数据或使用数据的应用程序而不必改变其他因素。所以，在现代数据库环境中能够大幅度减少程序维护。

1.7.10 关于数据库优点的告诫

本节介绍了数据库方法的9个主要潜在优点。然而，必须提醒你的是，许多组织在企图实现这些优点时却失败了。例如，由于老式的数据模型和数据库管理软件的限制，数据独立性的目标（和减少程序维护）难以达到。幸好，关系模型（和最新的面向对象模型）为实现这些优点提供了更好的环境。未能成功实现这些优点的另一个原因是组织规划和数据库实现低劣，即使最好的数据管理软件也不能克服这样的不足。因此，我们在下文中强调数据库规划和设计。

1.8 数据库方法的成本和风险

和进行任何商业决策一样，在实现数据库方法时必须认识到和管理某些附加的成本和风险（参见表1-6）。

1.8.1 新的专门人员

采用数据库方法的组织常常需要雇用或培训人员去设计和实现数据库，提供数据库管理服务以及管理新的人员。由于技术的快速更新，这些新员工必须再训练或定期更新知识。这种人员上的增加可能无法由其他生产率的增益来补偿，但是，一个组织不应该将这些有利于获得最大潜在利益所需要的专门技术的需求减到最小。第12章讨论数据库管理的人员需求。

表1-6 数据库方法的成本和风险

新的专门的人员
安装、管理成本和复杂性
转换成本
需要清晰备份和恢复
组织冲突

1.8.2 安装、管理成本和复杂性

多用户数据库管理系统是一组大型、复杂的软件，它具有很高的初始成本，需要训练有素的人员去安装和运行，还需要大量的维护和支持费用。安装这样的系统还可能需要将组织的硬件和数据通信系统升级。为了能跟上新版本和升级，通常需要进行大量培训。为了提供安全性并保证共享数据进行恰当的并发更新，可能需要另外一些更高级和昂贵的数据库软件。

1.8.3 转换成本

遗留系统（legacy system）通常是指组织中老式的应用，它基于文件处理或老式数据库技术。老式系统到现代数据库技术的转换成本——以美元、时间和组织的投入度量——对组织来说常常是一笔很大的负担。如第11章所述，使用数据仓库是一种继续使用老式系统而同时又能开发数据库技术的策略（Ritter，1999）。

1.8.4 需要清晰备份和恢复

一个共享的公司数据库必须在任何时间都是准确的和可用的。为了提供备份数据并在发生毁坏时恢复数据库，就要求开发和使用多个综合过程。通常，现代数据库管理系统完成备份和恢复任务的自动化程度比文件系统更高。第12章将介绍安全性、备份和恢复的过程。

1.8.5 组织冲突

共享数据库要求数据定义和所有权一致，并且要负责准确的数据维护。经验表明，在数据定义、数据格式和编码、更新共享数据的权利及有关问题上的冲突很常见，常常是很难解决的。处理这些问题需要组织在开发数据库时对数据库方法、组织优秀的数据库管理员以及合理的演化方法方面进行巨大的投入。

如果缺乏对数据库方法的强有力的顶级管理支持和投入，单独数据库的终端用户开发可能大幅度增加。这些数据库不遵循我们描述的一般数据库方法，它们不可能提供前面所述的优点。

1.9 数据库环境的组成部分

典型数据库环境及其联系的主要组成部分如图1-10所示。前面已经介绍了一些（不是全部）组成部分。以下对图1-10中的9个组成部分作简要描述。

1) **计算机辅助软件工程（CASE）工具** 使数据库和应用程序设计自动化的工具。本书将描述CASE工具在数据库设计和开发中的应用。

2) **信息库** 集中了全部数据定义、数据联系、屏幕和报表格式以及其他系统组成部分的知识库。信息库包含一个扩展的元数据的集合，它和信息系统的其他部分一样，对于数据库的管理是非常重要的。第12章将介绍信息库。

3) **数据库管理系统（DBMS）** 一个用于定义、建立、维护和提供对数据库及信息库的有控制访问的商业软件（偶尔也包括硬件和固件）系统。第12章和第13章将介绍DBMS的功能。

4) **数据库** 一个逻辑上相关的数据的有组织的集合，通常用于满足组织内多个用户的信息

需要。重要的是区分数据库和信息库。信息库包含数据的定义，而数据库包含数据的具体值。第5章和第6章将介绍数据库的设计活动，第7章～第11章将介绍数据库的实现。

5) **应用程序**　一个用来建立和维护数据库以及为用户提供信息的计算机程序。第7章～第11章将介绍主要的数据库程序设计技巧。

6) **用户界面**　语言、菜单和其他工具，用户借助它们来与不同的系统组成部分（例如，CASE工具、应用程序、DBMS和信息库）交互。用户界面的介绍贯穿全书。

7) **数据管理员**　负责组织中全部信息资源的人员。数据管理员使用CASE工具提高数据库规划和设计的生产率。第12章将详细介绍数据管理的职能。

8) **系统开发人员**　例如系统分析员和设计新应用程序的程序员。系统开发人员常常使用CASE工具进行系统需求分析和程序设计。

9) **终端用户**　整个组织内的人员，他们增加、删除和修改数据库中的数据，并且从中请求或接收信息。用户与数据库的所有交互活动都必须通过DBMS进行。

借助软件，用户界面变得越来越友好。例如，菜单驱动系统、启用Web的系统、鼠标的使用、声音识别系统。这些系统提高了终端用户的计算能力，也就是说，不是计算机专家的用户也能够定义他们自己的报表、显示和简单应用。当然，在这样的环境中，数据库管理必须保证实施足够的安全措施以保护数据库。

图1-10显示的DBMS运行环境是一个集成系统，它包括硬件、软件以及方便存储、检索和控制信息资源及改善组织生产率的人员。

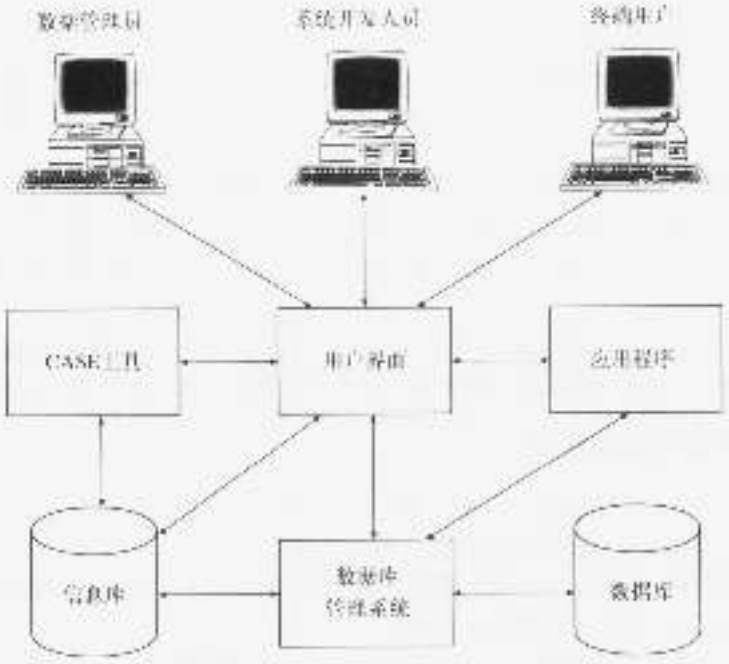


图1-10　数据库环境的组成部分

1.10　数据库系统的演变

数据库管理系统最早出现在20世纪60年代，并在随后的几十年继续发展。图1-11概述了这个演变，着重列出了每十年中趋于前沿的数据库技术。在大多数情况下，技术推广的时期是相

当长的，图中所示的技术在其被采用的十年前就首次提出。例如，关系模型首先由IBM研究员E.F.Codd在1970年出版的论文中定义（Codd，1970）。然而，关系模型直到20世纪80年代才在商业上得到普遍应用。

数据库管理系统的开发克服了前面所述的文件处理系统的局限。总之，以下3个目标总会驱动数据库技术的发展和演变：

1) 要求提供更高的程序和数据间独立性，从而减少维护成本。

2) 管理日益增长的复杂数据类型和结构的要求。

3) 为那些既没有程序设计语言基础，又不十分了解数据库中数据如何存储的用户提供更容易、更方便访问数据的要求。

1.10.1 20世纪60年代

这个时期还是文件处理系统占据主导地位。然而，第一个数据库管理系统在这个时期引入，主要用于大型的和复杂的高风险项目，例如阿波罗登月计划。我们将这个阶段看作实验性的“概念证明”期，验证用DBMS管理大量数据的可行性。在20世纪60年代后期，随着数据库任务组的成立，标准化方面的工作也开始展开。

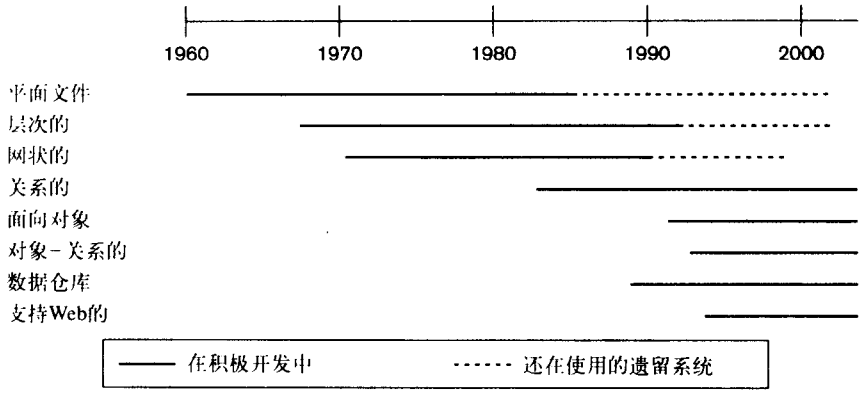


图1-11 数据库技术的演变

1.10.2 20世纪70年代

在这个时期，数据库管理系统的应用成了商业现实。人们开发出大量层次和网状数据库管理系统以应对日益复杂的数据结构，例如，生产中用到的材料单，它用普通的文件处理方法管理是极其困难的。层次和网状模型通常被认为是第一代DBMS。两种方法都在广泛使用，事实上，今天还继续使用着许多这样的系统。然而，它们存在一些主要缺点：

1) 基于导航的一次一条记录的过程使得数据库访问十分困难。其结果是，即使回答简单查询，也必须编写复杂的程序。

2) 数据独立性非常有限，因此，程序与数据的改变密切相关。

3) 和关系数据模型不同，这两种模型都没有广泛公认的理论基础。

1.10.3 20世纪80年代

为了克服这些局限，E.F.Codd和其他人在20世纪70年代开发了关系数据模型。这个模型被认为是第2代DBMS。20世纪80年代，它在商业上的应用得到普遍认可并广泛使用。在关系模型中，所有数据都以表的形式表示。比较简单的第4代语言SQL用于数据查询。所以，关系模型为非程序员提供了方便的访问方法，克服了第一代系统的主要问题。

1.10.4　20世纪90年代

20世纪90年代的这十年迎来了计算的新纪元，首先是出现了客户/服务器（Client/Server）计算，然后数据仓库和因特网应用变得越来越重要。虽然20世纪80年代DBMS管理的数据大部分是结构化的（例如财务数据），但到了20世纪90年代多媒体数据（包括图片、声音、图像和视频）变得越来越普遍。为了处理这些日益复杂的数据，在20世纪80年代后期引入了面向对象数据库（称为第3代）（Grimes，1998）。第14章和第15章将详细介绍面向对象数据库。

由于组织必须管理大量结构化和非结构化数据，关系和面向对象数据库在今天都非常重要。事实上，一些制造商正在开发对象-关系DBMS，这种DBMS能够管理两类数据。附录D将介绍对象-关系数据库。

1.10.5　2000年以来

我们自然会思索：未来十年，数据库技术会朝着什么方向发展？无疑将会出现许多在我们意料之外的重大事件，但我们至少可以预见到以下这些必然的趋势：

1）管理日益复杂的数据类型的能力。这些类型包括在数据仓库应用中已经日益重要的多维数据。第11章将讨论多维数据。

2）“通用服务器”的继续开发。基于对象-关系DBMS，这是一些能够管理各种对用户透明的数据类型的数据库服务器。它们对于因特网应用特别重要。

3）当完全的分布式数据库成为现实时，数据库集中化的当前趋势将继续。随着通信成本随着数据量增加而不断下降，查找和访问集中化数据的成本也将下降。高性能低成本计算也促进了集中化。

4）内容寻址存储将更普遍。利用这种方法，用户只要说明需要什么数据而不必说明如何检索就能够找到所需数据。例如，用户可以扫描一张照片并让计算机查找与该照片最接近的照片。

5）数据库及其他技术（例如，人工智能和类似电视的信息服务）使得数据库访问对于未经训练的用户来说更加容易。例如，用户将能够以更自然的语言请求数据，数据库技术将根据过去的查询和数据库的改变预料用户对数据的需要。

6）开发能够处理超大数据集的数据挖掘算法能够使组织有效地分析他们巨大的数据存储。改善关于顾客、员工、产品和供应商的识别模式、趋势和相互关系的能力将影响他们对战略决策的制定。这个能力也应用于从Web站点活动中获取的大量数据。例如，点击流分析（Riggs，2000）。

7）另一方面，个人数据助理（PDA）的激增将促进小型数据库同步的改善及无线传输率的改善。蓝牙无线标准将大大加速连接因特网的无线PDA的开发。这个发展将强调在日益增长的无线世界中保护数据安全的重要性。

本章小结

过去的20年见证了数据库应用在数量和重要性方面的巨大增长。数据库在各类组织中用于存储、操纵和检索数据。在2000年代高度竞争的环境中，各方面数据都表明数据库技术更为重要。现代数据库管理是信息系统学科中最重要的课程之一。

数据库是相关数据的有组织的集合。数据是用户环境中有意义的事实或对象。术语有组织的意思是数据是结构化的，因此，用户很容易存储、管理和检索数据。术语相关的意思是数据描述了一组用户的关注领域及这些用户能够利用这些数据去回答有关该领域问题。信息是这样一种数据，它被置于一定的语境之中，并能以便于人理解的合适的形式处理和表示。

元数据是描述其他数据性质或特征的数据。数据库管理系统是用于定义、建立、维护和提

供对数据库有控制访问的通用商业软件系统。DBMS将元数据存储在信息库中，该库是所有数据定义、数据联系、屏幕和报表格式及其他系统组成部分的中央仓库。

计算机文件处理系统在计算机早期时代开发，以便计算机能够存储、管理和检索大量的数据文件。这些系统（有的今天还在使用）有许多严重的局限性，例如，程序和数据间的依赖性、数据重复、数据共享有限和开发时间长。数据库方法能够克服这些局限。这种方法强调组织内数据的集成和共享。这个方法的优点包括程序-数据独立性、改善数据共享、最小数据冗余以及提高应用开发的生产率。

数据库应用能够分成以下几类：个人数据库、工作组数据库、部门数据库、企业数据库和因特网数据库。当今的企业数据库包括数据仓库和内容取自不同运作数据库的集成决策支持数据库。企业资源规划（ERP）系统严重依赖于企业数据库。

数据库技术自20世纪60年代问世以来稳步发展。驱动数据库发展的主要目标包括提供更高的程序-数据独立性的要求、管理日益复杂的数据结构的需要和为所有用户提供更快、更方便访问的要求。目前，越来越多的开始人们使用能满足这些目标的面向对象和对象-关系数据库。

本章复习

关键术语

约束	数据库管理系统（DBMS）	遗留数据
数据	企业数据模型	元数据
数据独立性	企业资源规划（ERP）系统	信息库
数据仓库	外部网	用户视图
数据库	信息	
数据库应用	内部网	

复习问题

1. 定义下列术语：

a. 数据　b. 信息　c. 元数据　d. 数据库应用
e. 数据仓库　f. 约束　g. 数据库　h. 遗留数据

2. 将下列术语和定义匹配起来：

_____数据	a. 放置在语境中的数据或概要数据
_____数据库应用	b. 应用程序
_____约束	c. 事实、文本、图形、图像等等
_____信息库	d. 集成的决策支持数据库
_____元数据	e. 相关数据的有组织的集合
_____数据仓库	f. 包括数据定义和约束
_____信息	g. 所有数据定义的中央仓储
_____用户视图	h. 数据描述和程序分离
_____数据独立性	i. 不能违反的规则
_____数据库	j. 部分数据库的逻辑描述

3. 比较下列术语：

a. 数据依赖；数据独立性　b. 数据仓库；数据挖掘
c. 数据；信息　d. 信息库；数据库
e. 实体；企业数据模型　f. 数据仓库；ERP系统

4. 列出并简要描述五类数据库，每类数据库给出一个例子。

5. 数据定义为什么已经延伸到当今的环境中?

6. 列出数据库系统环境的9个主要组成部分。

7. 关系数据库表之间的联系如何表示?

8. 对于下列各类数据库，列出每类数据库的一些通常必须回答的关键问题:

个人数据库;　工作组数据库;　部门数据库;　企业数据库;　因特网数据库

9. 术语数据独立性意味着什么，为什么它是一个重要的目标?

10. 列出数据库方法和文件系统相比的六个潜在优点。

11. 列出与数据库方法有关的五个附加成本或风险。

12. 对于从20世纪60年代至20世纪90年代的这段时间，列出每十年的主导数据库技术。指出每种技术属于第几代。

13. 由于处理大量数据能力的改善，描述有效应用这些巨型数据库的三个业务领域。

问题和练习

1. 对于下列各对对象，指出它们之间是否存在一对多或多对多联系（在典型大学环境下)。然后，用教材中介绍的速记符号为各对关系画图。

a. STUDENT和COURSE （学生注册课程）

b. BOOK和BOOK COPY （书有多个副本）

c. COURSE和SECTION （课程有多个班）

d. SECTION和ROOM （各班被安排在不同教室）

e. INSTRUCTOR和COURSE

2. 参照图1-4，回答下列问题:

a. 对顾客Customer_ID 6，有哪些订单未解决? （给出Order_ID 和 Order_Date）

b. 订单Order_ID 2包括哪些产品? （给出Product_ID 及每项明细的Quantity）

c. 订单Order_ID 2包括哪些产品? （给出Product_ID、Product_Name及每项明细的Unit_Price）

3. 为了与顾客联系，检查个人数据库（参见图1-7)。CUSTOMER 和 CONTACT HISTORY之间的联系是一对多还是多对多? 用教材中介绍的速记符号为该联系画图。

4. 重读本章描述的数据和数据库的定义。近来的数据库管理系统才包含存储和检索数值和文本之外的其他数据的能力。图像、声音、视频和其他高级数据类型需要什么特殊的数据存储、检索和维护能力，而数值和文本数据不需要?

5. 表1-1为一组数据项显示了元数据例子。为这些数据中再安排其他3列（即为已列出的属性再找出其他3个元数据特征)，并且在表1-1中为这3列补充相应的项。

6. 在1.3.2节中，有这样的陈述: 文件处理系统的缺点也可能是数据库的局限，这取决于组织如何管理数据库。首先，组织为什么要建立多个数据库，而不是一个支持全部数据处理需求的数据库? 其次，什么组织和个人因素可能导致组织具有多个、独立管理的数据库（因此，不完全遵循数据库方法）?

7. 假设有一个学生俱乐部或组织，你是其中一个成员。这个单位的数据实体有哪些? 列出并定义每个实体。然后，开发一个企业数据模型（如图1-3所示)，在模型中显示这些实体及其实体间重要联系。

8. 在1.5节中，介绍了5类不同的数据库，并为每类数据库概述了一系列主要决策。以后将了解如何处理这些和其他决策。然而，在阅读完本章后，结合你对计算机技术的全面认识，你

应该能够预料出这些决策在实践中的反应。对于以下几类数据库和主要决策，解释你认为应该作出怎样的选择或在任何组织内作出选择时应该考虑的因素。

a. 个人数据库。谁负责保证个人数据库中数据的准确性?

b. 工作组数据库。在工作站应执行哪些数据库处理操作，服务器上应执行哪些数据库处理操作?

c. 部门数据库。怎样才能提供足够的安全性以防止敏感数据未经授权便泄露或扩散出去?

d. 企业数据库。组织如何开发和维护关于数据命名、定义、格式和有关事项的标准?

e. 因特网、内部网或外部网数据库。借助服务器和浏览器上的applets和代码模块，所有组成部分怎样才能对数据意义有共同理解?

9. 构筑电子商务.com的最大挑战之一是使顾客能够快速收到从Web订购的商品。从本章学过的数据库知识和在Web定货的体验出发，你为什么认为公司应利用数据库解决方案以帮助他们改善供应链管理和加快订单填写和发送?

应用练习

1. 选择一个具有相当密集的信息系统部门和一组信息系统应用的组织。调查该组织是利用传统文件处理方法还是数据库方法去组织数据。该组织有多少不同的数据库?试画一个类似于图1-2的图，描绘该组织的部分或所有文件和数据库。

2. 对于应用练习1中的组织或其他组织，向数据库管理员或设计者咨询。该组织维护哪类数据库元数据?组织为什么选择记录这些元数据而不是其他元数据?它们用什么工具来维护元数据?

3. 对于应用练习1或2中的组织或其他组织，如果可能，确定组织中的数据库属于本章所列5类数据库中的哪一类：个人、工作组、部门、企业和因特网。该组织有数据仓库吗?如果有，该数据仓库如何由其他数据库的数据形成?该组织使用ERP系统吗?

4. 对于以上提到的组织，确定公司使用的是内部网、外部网或支持Web的业务过程。对于每一种方式，确定使用目标和连接网络的数据库管理系统。询问公司明年在他们的业务活动中使用内部网、外部网或Web的计划是什么。为了实现这个计划，公司正在寻求的新技术是什么。

5. 在你学习本书时，应该有一本记录自己对数据库管理的思想和观察的个人日志。使用这本日志记录你听到的评论、你读到的新闻故事概要或读到的专业文章、你创造的原始想法或假设、统一资源定位器和有关数据库的Web站点的评论以及需要进一步分析的问题。应始终关注涉及数据库管理的任何问题。为了提供反馈和反应，你的导师会定期检查你的日志。该日志是一个记录个人想法的无结构的笔记，它将补充你的课堂笔记和激发你思考在课堂上无法涉及的课题。

参考文献

Ballou, D. P. and G. K. Tayi. 1999. "Enhancing Data Quality in Data Warehouse Environments." *Communications of the ACM* 42 (January): 73-78.

Codd, E. F. 1970. "A Relational Model of Data for Large Relational Databases." *Communications of the ACM* 13 (June): 377-87.

Grimes, S. 1998. " Object/ Relational Reality Check." *Database Programming and Design* 11:7 (July): 26-33.

Jordan, A. 1996. "Data Warehouse Integrity: How Long and Bumpy the Road?" *Data Management Review* 6:3 (March): 35-37.

Lambert, B. 1996. "Data Warehousing Fundamentals: What You Need to Succeed." *Data Management Review* 6:3 (March): 24-30.

Riggs, S. 2000. "Collecting Webdata." *Teradata Review* 3:1 (Spring): 12-19.

Ritter, D. 1999. "Don't Neglect Your Legacy." *Intelligent Enterprise* 2:5 (March 30): 70,72.

White, C. 1995. "Database Technology: Sorting Out the Options." *Supplement to Database Programming & Design* 8 (December): 41-44,46.

Winter, R. 1997. "What, After All, Is a Very Large Database?" *Database Programming &Design* 10: 1 (January): 23-26.

进一步阅读

Date, C. J. 1998. "The Birth of the Relational Model, Part 3." *Intelligent Enterprise* 1:4 (December 10): 45-48.

Hoffer, J. A.,J. F. George, and J. S. Valacich. 2002. *Modern Systems Analysis and Design*. 3rd ed. Upper Saddle River, NJ: Prentice Hall.

Ritter, D. 1999. "The Long View." *Intelligent Enterprise* 2:12 (August 24): 58, 63, 67.

Web资源

- www.webopedia.com 计算机术语和因特网技术的在线词典和搜索引擎。
- www.techrepublic.com 信息技术专业人员可以定制以满足其特殊爱好的门户站点。
- www.zdnet.com 能查阅和所选择的信息技术主题有关的最新文章的另一个门户站点。

项目案例：山景社区医院

引言

本案例提供一个应用每章所学概念和技术的机会。我们选择一所医院作为案例，是因为医疗组织在当今社会举足轻重，大多数人应该都会熟悉它。本书每一章的末尾均包含本案例研究的一个片段。每个片段包括与该章内容有关的案例的简要描述，随后是问题和练习。

项目描述

山景社区医院是一个非赢利的、短期的、能够接待急诊的医院。这是一个目前有150个床位的相对较小的医院。根据国家趋势，近年来医疗卫生成本不断上涨，山景社区医院的基本目标是在控制成本的情况下，为周围社区提供优质的医疗服务。

山景社区医院具有支持下列领域的计算机应用：病人医疗管理、临床服务、财务管理和管理性服务。大部分应用是向供应商买来的，少量是自行开发的。计算机应用是基于C/S技术，如图1-5所示。大部分计算机应用是采用关系数据库技术实现的。大部分数据库（及应用）属于本章介绍的部门数据库。

山景社区医院的关系数据库包括许多表。图1-12显示了其中的两个表(具有一些样本数据)。PATIENT（病人）表包含有关当前的或最近的病人数据，而PATIENT CHARGES（病人费用）表包含描述病人承担费用的数据。在本章，这些表被简化了，但在以后各章中将被扩充。

PATIENT

Patient Name	Patient Number	Patient Address
Dimas, Salena	8379	617 Valley Vista
Dolan, Mark	4238	818 River Run
Larreau, Annette	3047	127 Sandhill
Wiggins, Brian	5838	431 Walnut
Thomas, Wendell	6143	928 Logan

PATIENT CHARGES

Item Description	Item Code	Patient Number	Amount
Room Semi-Priv	200	4238	1600
Speech Therapy	350	3047	750
Radiology	275	4238	150
Physical Therapy	409	5838	600
EKG Test	500	8379	200
Room Semi-Priv	200	3047	800
Standard IV	470	8379	150
EEG Test	700	4238	200

图1-12 两张数据库表

项目问题

1) 山景社区医院在使用数据库时应该希望得到哪些重要好处？尽可能多地叙述你对医院环境的意见。

2) 使用数据库时，医院必须小心管理的成本和风险是什么？

3) 当前，山景社区医院正在使用关系数据库技术。虽然这个技术适合结构化数据（例如，病人和财务数据），但它对于复杂数据（如地理数据和图像）不太适合。你能够考虑一些医院维护的、属于复杂数据的数据类型吗？哪一类数据库技术可能比关系技术更适合这些数据类型？

4) PATIENT和PATIENT CHARGES两表中的数据如何相关？即用户如何找到某一个病人的有关费用？

5) 医院使用因特网的途径是什么？

项目练习

1) 使用本章介绍的速记符号，画一张图表示PATIENT 和PATIENT CHARGES之间的联系。

2) 为PATIENT和PATIENT CHARGES表的数据属性开发一个元数据图（参见表1-1）。至少使用表1-1中的那些列，但可以包括其他你认为适于山景社区医院的数据管理的元数据特征。

3) 该医院的重要数据库视图之一是病人账单。以下是这个视图的高度简化的版本。

Patient Name: Dolan, Mark

Patient Number: _____

Patient Address: _____

Item Code	Item Description	Amount

根据图1-12中的数据，为上述视图填写缺少的数据。

4) 使用本章介绍的速记符号，画一张图表示医院环境中的下列联系。

a. HOSPITAL的工作人员中有一个或多个PHYSICIAN。一个PHYSICIAN只能是一个HOSPITAL的工作人员。

b. 一个PHYSICIAN可以诊治一个或多个PATIENT。一个PATIENT只能被一个PHYSICIAN诊治。

c. 每个PATIENT可以发生任意多个CHARGE。一个特定的CHARGE可以发生在任意多个PATIENT身上。

d. 一个HOSPITAL有一个或多个WARD。每个WARD只能位于一个HOSPITAL中。

e. 一个WARD有一个或多个EMPLOYEE。一个EMPLOYEE可以在一个或多个WARD工作。

5) 根据国家趋势，近年来医疗卫生成本不断上涨，山景社区医院的基本目标是在控制成本的情况下，为周围社区提供优质医疗服务。管理良好的数据库如何帮助医院实现这一目标？给出一些使用医院数据库改善医疗服务质量和控制成本的例子。

第2章　数据库开发过程

2.1　学习目标

学完本章后，读者应该具备以下能力：

- 定义下列关键术语：企业数据模型、信息系统体系结构（ISA）、信息工程、自顶向下规划、业务功能、功能分解、系统开发生命周期(SDLC)、原型法、计算机辅助软件工程(CASE)、信息库、项目、递增提交、概念模式、物理模式以及客户/服务器(C/S)体系结构。
- 描述系统开发项目的生命周期，重点强调数据库分析、设计和实现活动的目的。
- 解释数据库和应用开发的原型法。
- 解释设计、实现、使用和管理数据库的人员的角色。
- 解释概念模式、外部模式和物理模式的区别，以及数据库采用三层模式体系结构的原因。
- 解释数据库的三层定位体系结构和数据库的处理。
- 解释数据库设计和开发类项目的范围。
- 绘制显示数据库范围的简单数据模型。

2.2　引言

第1章介绍信息系统的数据库方法和许多组织的数据库应用环境。我们用松谷家具公司的例子来说明信息系统中数据库方法的许多原则和概念。和以后的各章一样，在这一章，你将再一次看到松谷家具公司作为数据库管理应用的一个例子。

第2章概述数据库的分析、设计、实现和管理的主要步骤。因为数据库是信息系统的一部分，所以在本章你将看到数据库开发过程怎样适合整个信息系统的开发过程。本章强调数据库开发与一个完整的信息系统的其他部分的开发相协调的必要性。本章包括松谷家具公司的一个假设的数据库开发过程的重要部分。在这个例子中，本章将介绍在PC机上开发数据库的工具，以及从企业数据库中为单独应用提取数据的步骤。

在这里讨论数据库开发有以下几个原因。第一，尽管你使用过一个数据库管理系统（例如Microsoft Access）的基本功能，但你可能还不理解数据库是怎样被开发的。本书主要的目标是向你介绍许多设计和构建数据库应用的概念和工具。通过简单的例子，本章将简要地说明在利用本书完成一门数据库课程之后你可以作些什么。因此，本章将为在以后各章中详细叙述的主题奠定基础。

第二，许多学生利用充满具体例子的教材能达到最好的学习效果。尽管在本书的所有章节中包含大量的例子、图解和实际的数据库设计和编码，但每一章都关注于数据库管理的一个特定方面。本章的主要目的是帮助你理解数据库管理的各个方面是如何相关的。尽管我们尽量减少技术细节，但是我们希望激发你学习后续各章中详细描述的数据建模和开发技巧的兴趣。

最后，许多教师希望学生在学习数据库课程之初就能尝试简单的数据库开发或者进行一个简单的课程设计。由于本书中的各个主题之间的逻辑顺序，在看到项目的目标之前你将学习很多内容。本章将指导你怎样构建一个足以开始一个课程练习的数据库开发项目。很明显，由于这仅仅是第2章，所以许多例子和符号将比你在其他课外作业中的项目或一个实际的组织所用

到的例子和符号要简单得多。

需要指出的是，你不可能仅仅通过本章的学习就学会如何设计或开发数据库。本章的内容是简单的、入门性质的，其中所用的许多符号与你将在后续章节中学到的符号不完全一样。在第2章，我们的目的是使你大体理解关键步骤和各类技巧，而不是学会特定的技术。然而，这将培养你学习后续章节中所述技巧和知识的学习的热情。

2.3 信息系统开发中的数据库开发

在许多组织中，数据库开发是从**企业数据建模**（enterprise data modeling）开始的，企业数据建模确定了组织数据库的范围和一般内容。这一步骤通常发生在一个组织进行信息系统规划的过程中，它的目的是为组织数据创建一个整体的描述或解释，而不是设计一个特定的数据库。一个特定的数据库为一个或多个信息系统提供数据，而企业数据模型（可能包含许多数据库）描述了由组织维护的数据的范围。在企业数据建模时，你审查当前的系统，分析需要支持的业务领域的本质，描述需要进一步抽象的数据，并且规划一个或多个数据库开发项目。图2-1与图1-3完全相同，它使用第3章和第4章中所使用的符号的一个简化版本显示松谷家具公司的企业数据模型的一个部分。

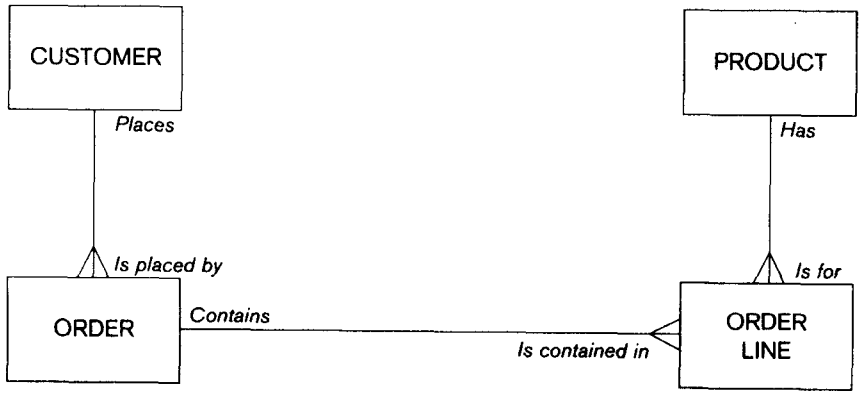


图2-1　企业数据模型的一部分（松谷家具公司）

2.3.1 信息系统体系结构

如图2-1所示，高级的数据模型仅仅是总体**信息系统体系结构**（Information System Architecture，ISA）的一部分或一个组织信息系统的蓝图。在信息系统规划期间，你可以建立一个企业数据模型作为整个信息系统体系结构的一部分。根据Zachman(1987)、Sowa和Zachman(1992)的观点，一个信息系统体系结构由以下6个关键部分组成：

1) 数据（如图2-1所示，但是也有其他的表示方法，其中一些在下一节中进行描述）。

2) 操纵数据的处理（这些可以用数据流图、带方法的对象模型或其他符号表示）。

3) 网络，它在组织内并在组织与它的主要业务伙伴之间传输数据（它可以通过网络连接和拓扑图来显示）。

4) 人，人执行处理并且是数据和信息的来源和接收者（人在过程模型中显示为数据的发送者和接收者）。

5) 执行过程的事件和时间点（它们可以用状态转换图和其他的方式来显示）。

6) 事件的原因和处理数据的规则（经常以文本形式显示，但是也存在一些用于规则的图表工具，如决策表）。

2.3.2 信息工程

信息系统的规划者按照信息系统规划的特定方法开发出信息系统的体系结构。信息工程是一种正式的和流行的方法。**信息工程**（information engineering）是一种面向数据的创建和维护信息系统的方法。因为信息工程是面向数据的，所以当你开始理解数据库是怎样被标识和定义时，信息工程的一种简洁的解释是非常有帮助的。信息工程遵循**自顶向下规划**（top-down planning）的方法，其中，特定的信息系统从对信息需求的广泛理解中推导出来（例如，我们需要关于顾客、产品、供应商、销售员和加工中心的数据），而不是合并许多详尽的信息请求（如一个订单输入屏幕或按照地域报告的销售汇总）。自顶向下规划可使开发人员更全面地规划信息系统，提供一种考虑系统组件集成的方法，增进对信息系统与业务目标的关系的理解，加深对信息系统在整个组织中的影响的理解。

信息工程包括四个步骤：规划、分析、设计和实现。信息工程的规划阶段产生信息系统体系结构，包括企业数据模型。我们将在下一节回顾信息工程的规划阶段。尽管本书在本质上并没有遵循信息工程的方法论，但是余下的各章将讨论信息工程后三个阶段的活动。

2.3.3 信息系统规划

信息系统规划的目标是使信息技术与组织的业务策略紧密结合，这种结合对于从信息系统和技术的投资中获取最大利益是非常重要的。正如表2-1所描述的那样，信息工程方法的规划阶段包括3个步骤，我们将在后续的3个小节中讨论它们。

表2-1 信息工程规划阶段

步 骤	解 释
1	确定关键性的规划因素
	a. 目标
	b. 关键的成功因素
	c. 问题领域
2	确定组织的规划对象
	a. 组织单元
	b. 地点
	c. 业务功能
	d. 实体类型
3	建立企业模型
	a. 功能分解
	b. 实体-联系图
	c. 规划矩阵

1. 确定关键性的规划因素

关键性的规划因素是指组织目标、关键的成功因素和问题领域。确定这些因素的目的是建立规划的环境并且将信息系统规划与战略业务规划联系起来。表2-2显示了松谷家具公司的一些可能的关键规划因素，这些因素有助于信息系统的管理者为新的信息系统和数据库设定优先级以处理需求。例如，考虑到不精确的销售预测这个问题领域，信息系统的管理者可能在组织数据库中存放额外的历史销售数据、新的市场研究数据和新产品的测试数据。

2. 确定组织的规划对象

组织规划对象定义了业务范围，业务范围会限制后来的系统分析和信息系统可能发生改变的地方。五个关键的规划对象如下所示（参见表2-3松谷家具公司的关键规划对象的例子）：

- **组织单元** 组织中的各种部门。

- 组织地点　业务操作的发生地。
- 业务功能　支持组织使命的业务处理的相关组。业务功能不同于组织单元，事实上一个功能可以分配给多个组织单元（例如，产品开发功能可能是销售部和生产部共同的责任）。
- 实体类型　关于组织所管理的人、地点和事物的数据的主要类别。
- 信息系统　处理数据集的应用软件和支持程序。

表2-2　信息工程规划阶段结果的例子（松谷家具公司）

规划因素	例　子
目标	保持每年10%的增长率
	保持15%的税前投资回报
	避免解雇员工
	做一个负责任的公司成员
关键成功因素	高质量的产品
	及时发送成品
	员工的高生产率
问题领域	不准确的销售预测
	竞争的加剧
	成品的库存不足

表2-3　组织规划对象的例子（松谷家具公司）

规划对象	例　子
组织单元	销售部
	订购部
	财务部
	生产部
	加工部
	装配部
	成品部
	采购部
组织地点	公司总部
	Durango工厂
	西区销售处
	木材工厂
业务功能	业务规划
	产品开发
	原材料管理
	市场和销售
	履行订单
	订单发货
	销售汇总
	产品操作
	财务和会计
实体类型	CUSTOMER
	PRODUCT
	RAW MATERIAL
	ORDER
	WORK CENTER

（续）

规划对象	例 子
信息系统	INVOICE EQUIPMENT EMPLOYEE 事务处理系统 订单跟踪 订单处理 工厂调度 工资单 管理信息系统 销售管理 库存控制 生产调度

3. 建立企业模型

一个全面的企业模型包括每个业务功能的功能分解模型、企业数据模型和各种规划矩阵。**功能分解**（functional decomposition）是把组织的功能进行更详细分解的过程，功能分解是在系统分析中为了简化问题、分散注意力和确定组件而使用的经典处理方法。在松谷家具公司中订单履行功能的功能分解的例子如图2-2所示。对于处理业务功能和支持功能（例如表2-3中列出的所有功能和子功能）的全部集合而言，多个数据库是必须的，因此一个特定的数据库可能仅仅对支持功能（如图2-2所示）的一个子集提供支持。为了减少数据冗余和使数据更有意义，拥有完整的、高层次的企业视图是非常有帮助的。

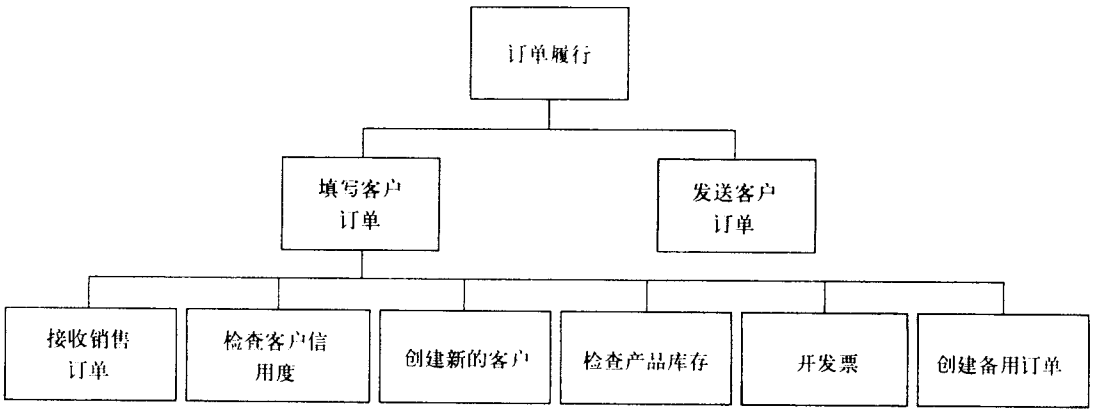


图2-2 订单履行功能的功能分解过程的例子（松谷家具公司）

企业数据模型使用特定的符号来描述。图2-1运用的是一种可应用的一般符号，在第3章和第4章中将解释实体-联系图这种更流行的绘图方法。除了实体类型这种图形描述外，一个完整的企业数据模型还应包括每个实体类型的描述和描述业务操作的提要，即业务规则。业务规则决定数据的有效性，第3章和第4章将定义和说明业务规则。

一个企业数据模型不仅包括实体类型，还包括数据实体间的联系，以及各种规划对象间的其他联系。显示规划对象间联系的一种常见形式是矩阵。由于规划矩阵不需要数据库被明确的建模就可以明确描述业务需求，因此规划矩阵是一种重要的功能。规划矩阵经常从业务规则中导出，它有助于设定开发活动优先级、将开发活动排序和根据自顶向下视图通过一种企业范围

的方法安排这些开发活动。有许多种规划矩阵可供使用，它们的共同之处是：

- *地点-功能*　显示业务功能在哪个业务地点执行。
- *单元-功能*　显示业务功能由哪个业务单元执行或负责。
- *信息系统-数据实体*　解释每个信息系统如何与每个数据实体相互作用（例如，是否每个系统都对每个实体中的数据进行创建、检索、更新和删除）。
- *支持功能-数据实体*　确定每个功能中数据的获取、使用、更新和删除。
- *信息系统-目标*　显示信息系统支持的每个业务目标。

数据实体类型 / 业务功能	顾客	产品	原材料	订单	加工中心	工作订单	发票	设备	员工
业务规划	×	×						×	×
产品开发		×	×		×			×	
材料管理		×	×	×	×	×		×	
订单履行	×	×	×	×	×	×	×	×	×
订单发送	×	×		×	×		×		×
销售汇总	×	×		×			×		×
生产操作		×	×	×	×	×		×	×
财务和会计	×	×	×	×	×		×	×	×
× = 数据实体（列）在业务功能（行）中使用									

图2-3　业务功能-数据实体矩阵的例子

图2-3举例说明了一个可能的功能-数据实体矩阵。这样的矩阵可以用于多种目的，包括以下三个目的：

1) *确定空白实体*　显示哪些数据实体没有被任何功能使用或哪个功能没有使用任何实体。

2) *发现丢失的实体*　每个功能涉及的员工通过检查矩阵能够确认任何可能丢失的实体。

3) *区分开发活动的优先级*　如果一个给定的功能对于系统开发有高优先级（可能因为它与重要的组织目标相关），那么这个领域所使用的实体在数据库开发中拥有高优先级。

Hoffer、George和Valacich(2002)的著作中有关于怎样使用规划矩阵完成信息工程和系统规划的更完整的描述。

2.4　数据库开发过程

基于信息工程的信息系统规划是数据库开发项目的一个来源。这些开发新数据库的项目通常是为了满足组织的战略需求，例如改善客户支持、提高产品和库存管理或进行更精确的销售预测。然而许多数据库开发项目更多的是以自底向上的方式出现的，例如信息系统的用户需要特定的信息来完成他们的工作，从而请求开始一个项目，又如其他信息系统的专家发现组织需要改进数据管理而开始新的项目。即使在自底向上的情况下，建立企业数据模型也是必须的，以便理解现有的数据库是否可以提供所需的数据，否则，新的数据库、数据实体和属性都应该加到当前的组织数据资源中去。

无论是战略需求还是操作信息的需求，每个数据库开发项目通常集中在一个数据库上。一些数据库项目仅仅集中在定义、设计和实现一个数据库，以作为后续信息系统开发的基础。然而在大多数情况下，数据库及其相关信息处理功能是作为一个完整的信息系统开发项目的一部分而被开发的。

2.4.1 系统开发生命周期

指导管理信息系统开发项目的传统过程是**系统开发生命周期**（Systems Development Life Cycle，SDLC）。系统开发生命周期是指一个组织中由数据库设计人员和程序员组成的信息系统专家小组详细说明、开发、维护和替换信息系统的全部步骤。这个过程常被看作一组瀑布式的步骤，如图2-4所示（见Hoffer、George和Valacich，2002）。将这个过程比作瀑布是因为每一步都流到相邻的下一步，即信息系统的规格说明是一块一块地开发出来的，每一块的输出是下一块的输入。然而如图所示，这些步骤并不是纯线性的，每个步骤在时间上有所重叠（因此可以并行地管理步骤），而且当需要重新考虑先前的决策时，还可以回滚到前面某些步骤。（因而水可以在瀑布中倒流！）

图2-4对系统开发生命周期每一阶段的目的和可交付的产品进行了简明注解。系统开发生命周期的每一阶段都包括与数据库开发相关的活动，所以，数据库管理的问题遍布整个系统开发过程。我们在图2-5中重复了系统开发生命周期的七个阶段，并概述了每个阶段常见的数据库开发活动。请注意，系统开发生命周期的阶段和数据库开发步骤之间不存在一一对应的关系，概念数据建模发生在两个系统开发生命周期阶段之间。在本章我们将简要地说明松谷家具公司的数据库开发步骤的每一步。

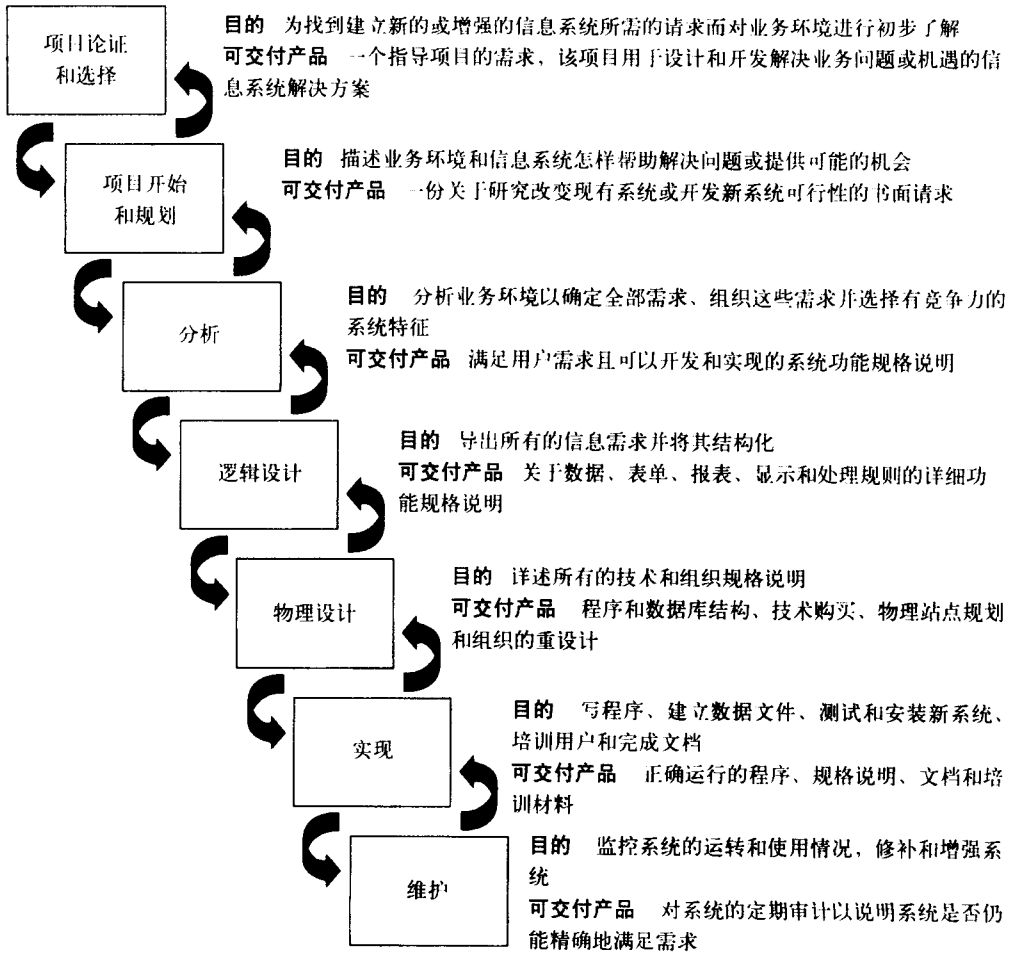


图2-4 系统开发生命周期（SDLC）

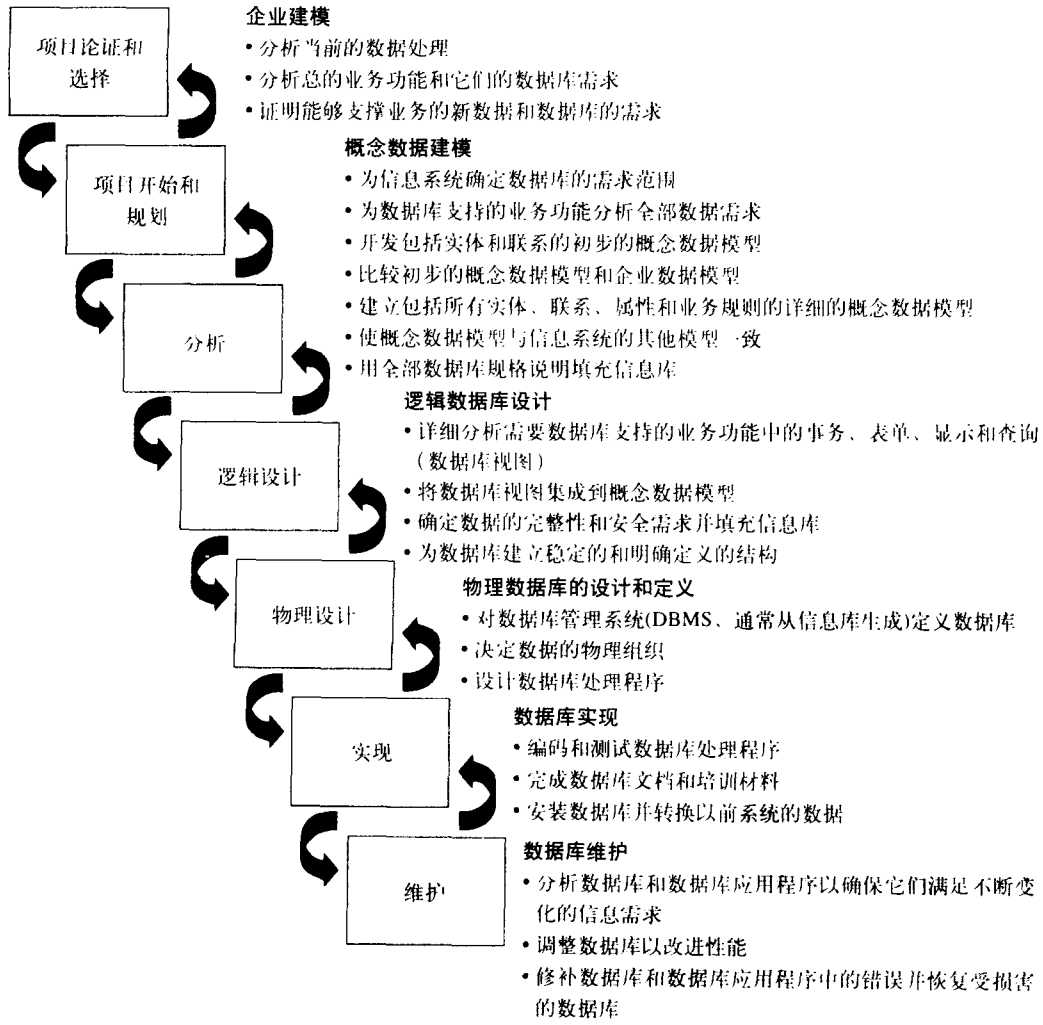


图2-5　系统开发生命周期中的数据库开发活动

1. 企业建模

数据库开发过程从企业建模（系统开发生命周期中项目论证和选择阶段的一部分）开始，设定组织数据库的范围和一般内容。企业建模发生在信息系统规划和其他活动期间，这些活动确定信息系统的哪个部分需要改变和加强并概述出全部组织数据的范围。在这一步中，检查当前数据库和信息系统，分析作为开发项目主体的业务领域的本质，用非常一般的术语描述每个信息系统在开发时所需要的数据。每个项目只有当它达到组织的预期目标时才可以进行下一步。

2. 概念数据建模

对一个已经开始的信息系统项目而言，概念数据建模阶段分析信息系统的全部数据需求。它分为两个阶段。首先，在项目开始和规划阶段建立一张类似于图2-1的图，同时建立其他文档来概述不考虑现存数据库的情况下特定开发项目中所需的数据范围。此时仅仅包括高层类别的数据（实体）和主要联系。然后在系统开发生命周期的分析阶段产生确定信息系统必须管理的全部组织数据的详细数据模型，定义所有数据属性，列出全部数据类别，表示数据实体间所有的业务联系，确定描述数据完整性的全部规则。在分析阶段，还要检查概念数据模型（在本

章后面也称作概念模式）与用来解释目标信息系统其他方面的模型类别的一致性，例如处理步骤、处理数据的规则以及事件的时序。然而，即使是这样详细的概念数据模型也只是初步的，因为后续的信息系统生命周期中的活动在设计事务、报表、显示和查询时可能会发现遗漏的元素或错误。因此，经常说到的概念数据建模是以一种自顶向下的方式完成的，它由业务领域的一般理解所驱动，而不是由特定的信息处理活动所驱动。

3. 逻辑数据库设计

逻辑数据库设计从两个角度进行数据库开发。首先，将概念数据模型变换成基于关系数据库理论的标准表示方法——关系，在第5章你将学会怎样完成这个重要的过程。然后像设计信息系统的每个计算机程序（包括程序的输入和输出格式）那样，对数据库支持的事务、报表、显示和查询进行详细的检查。在这个所谓的自底向上的分析中，精确地验证数据库中需要维护的数据和在每个事务、报表等等中需要的那些数据的性质。

对于每个单独的报表、事务等等的分析都要考虑一个特定的、有限制的但是完全的数据库视图。当报表、事务等被分析时有可能根据需要而改变概念数据模型。尤其在大型的项目中，不同的分析人员和系统开发者的团队可以独立地工作在不同的程序或程序集中，他们所有工作的细节直到逻辑设计阶段才可能会显示出来。在这种情况下，逻辑数据库设计阶段必须将原始的概念数据模型和这些独立的用户视图合并或集成到一个全面的设计中。在进行逻辑信息系统设计时也可以确定额外的信息处理需求，此时这些新的需求必须集成到前面确定的逻辑数据库设计中。

逻辑数据库设计的最后一步是根据为生成结构良好的数据规格说明而确定的规则，将组合的、协商后的数据规格说明转换成基本的或原子的元素。对当今的大部分数据库而言，这些规则来自关系数据库理论和称作规范化的过程，我们将在第5章详细讲述这一过程。这一步的结果是产生管理这些数据的、不引用任何数据库管理系统的完整的数据库描述图。在完成逻辑数据库设计后，开始确定详细的计算机程序的逻辑和维护、报告数据库内容所需的查询。

4. 物理数据库设计和定义

物理数据库设计和定义阶段决定计算机存储器（通常是磁盘）中数据库的组织，定义数据库管理系统的物理结构，概述处理事务的程序，产生期望的管理信息和决策支持的报表。本阶段的目标是设计能够有效、安全地管理所有数据处理的数据库，因此物理数据库设计需紧密结合物理信息系统其他方面的设计，包括程序、计算机硬件、操作系统和数据通信网络。

5. 数据库实现

数据库实现阶段编写、测试和安装处理数据库的程序。设计人员可以使用标准的编程语言（如COBOL、C或Visual Basic）、专用的数据库处理语言（如SQL），或专用的非过程化语言来编程，以产生固定格式的报表、显示结果，可能还包括图表。在实现阶段，还要完成所有的数据库文档，培训用户，为信息系统（和数据库）的用户安装程序。最后一步是利用现存的信息源（遗留应用中的文件和数据库以及现在需要的新数据）加载数据。加载数据的第一步经常是将数据从现存的文件和数据库中转到一种中间的格式（如二进制或文本文件），然后再将这些中间数据加载到新的数据库中。最后，运行数据库以及相关的应用以供实际的用户维护和检索数据。在运转期间，定期备份数据库，并当数据库损坏或受到影响时恢复数据库。

6. 数据库维护

数据库在数据库维护期间逐渐发展。在这一步，为了满足变化的业务条件，为了改正数据库设计的错误，或数据库应用的处理速度而增加、删除或改变数据库的结构特征。当一个程序或计算机发生故障而使数据库受到影响或损坏时也可能应该重建数据库。这一步通常是数据库

开发过程中最长的一步，因为它持续数据库及其相关应用的整个生命周期，每次数据库的发展都可看作一个简略的数据库开发过程，其中会出现概念数据建模、逻辑和物理数据库设计以及数据库实现以处理提出的变化。

2.4.2 信息系统开发的其他方法

系统开发生命周期法或其稍作变化的变体经常用于指导信息系统和数据库的开发。信息系统生命周期（SDLC）是一种方法学，它是高度结构化的方法，它包括许多检查和权衡以确保每一步产生精确的结果，而且新的或替代的信息系统与它必须通信的或数据定义需要一致的现存系统保持一致。系统开发生命周期法经常由于产生一个工作系统需要很长的时间而受到批评，因为工作系统仅仅在整个过程结束时才产生。现在组织越来越多的使用快速应用开发法（RAD），它是一个包含分析、设计和实现步骤的快速重复的迭代过程，直到汇聚到用户所需的系统为止。快速应用开发法在所需的数据库已经存在、增强系统主要是为了检索数据的应用中适用，而不适用于那些生成和修改数据库的应用。

使用最广泛的快速应用开发法之一是**原型法**（prototyping）。原型法是一个系统开发的迭代过程，通过分析员和用户的紧密配合，持续地修改系统而最终将所有需求转换成一个工作系统。图2-6显示原型法的过程。在此图中我们包含了注释，概略地描述了每个原型法阶段的数据库开发活动。一般来说，当信息系统的问题被确定时，仅仅粗略地尝试概念数据建模。在开发最初的原型时，设计用户想要的显示和报表，同时理解任何新的数据库需求并定义一个用于原型的数据库。这通常是一个新的数据库，它复制现存系统的一部分，还可能增加了一些新的内容。当需要新的内容时，这些内容通常来自外部数据源，如市场研究数据、一般的经济指标或行业标准。

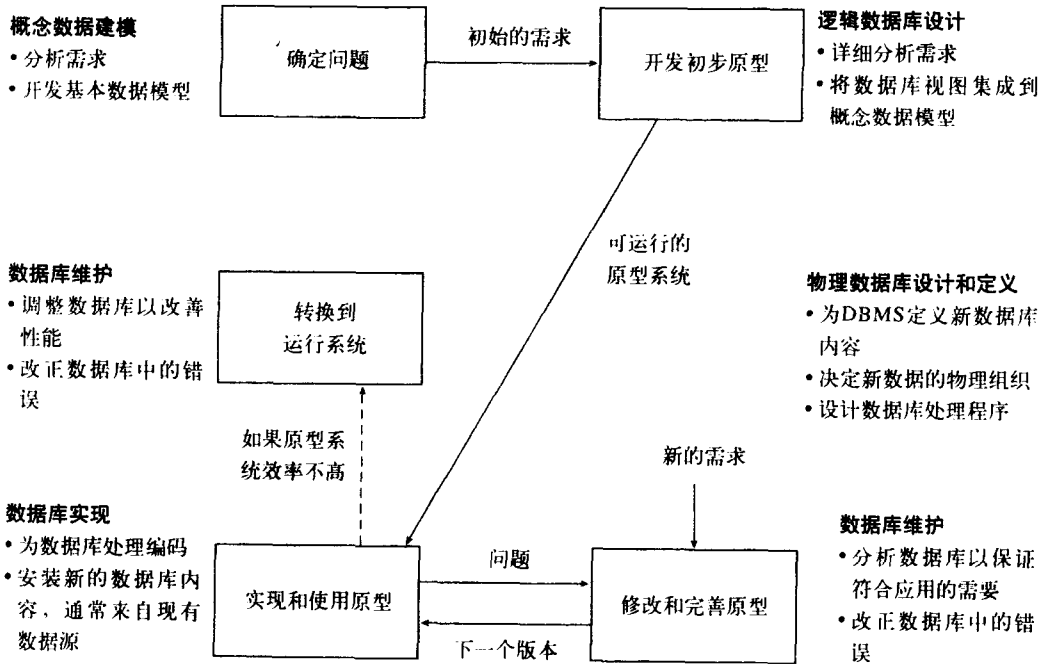


图2-6　原型法和数据库开发过程

当产生原型新的版本时重复数据库的实现和维护活动。通常仅进行最低限度的安全性和完整性控制，因为此时的重点是尽可能快地产生可以使用的原型版本。而且文档管理也延迟到项

目的最后，在交付使用时才进行用户培训。最后，一旦构建了一个可接受的原型，开发者和用户将决定最后的原型和数据库是否能交付使用。如果系统（包括数据库）效率很低，那么系统和数据库将被重新编程和重新组织以达到期望的性能。

随着可视化编程工具（如Visual Basic、Java、Visual C++和第四代语言）越来越流行，利用可视化编程工具可以很方便地修改用户与系统间的界面，原型法正成为可供选择的系统开发方法。使用原型法改变用户报表和显示的内容和布局是相当容易的。在这个过程中，新的数据库需求被确定，因此被发展中的应用使用的现存的数据库应该进行修改。甚至有可能为一个需要新的数据库的系统使用原型法，在这种情况下，当系统需求在迭代的开发过程中不断变化时需要获取样本数据以建造或重建数据库原型。

2.4.3 计算机辅助软件工程的作用和信息库

在前面章节中，我们提到过CASE工具在信息系统开发中的作用。**计算机辅助软件工程**（Computer-Aided Software Engineering，CASE）工具是为系统开发过程的某些部分提供自动支持的软件。我们的目的是学习数据库开发，CASE工具有三个相关的特征。第一是它有能力帮助我们使用实体-联系图或其他的符号（参见图2-1所示的E-R符号一种形式的例子）绘制数据模型。CASE工具绘图的能力是“数据库智能的”，其中每个符号表示特定的数据模型结构，这些符号仅仅能够按照与相关结构的属性相一致的方式使用。数据库绘图工具可用于企业建模、概念数据建模、逻辑数据库设计和物理数据建模。CASE工具能帮助我们确保图与图之间的一致性。例如，CASE工具可以确保每个PRODUCT实体在任何图中使用时含义一致。CASE绘图工具强制使每个数据对象有惟一的名字，当数据对象的特征发生改变时可以自动重画数据模型。

通常，不同的绘图工具和相关的方法用于数据库开发过程的不同阶段。例如，一个工具用于在企业建模时绘制高层图和矩阵，另一个工具可能用于概念数据建模，其他的某个工具用于逻辑数据库设计等等。

CASE工具第二个重要的特性是它可以产生代码。通常，这些代码包含一个给定数据库管理系统的数据库定义命令。在数据库实现阶段，CASE工具将查阅所有的概念、逻辑和物理数据规格说明并生成创建关系表、定义每张表的每个属性和关键索引的SQL命令。某些CASE工具还能够生成C语言或其他语言的代码作为数据库检索和更新程序的基础代码（虽然这是CASE工具不常用的功能）。

有了绘图工具后，许多组织为了生成代码及绘图而使用单独的工具。如果这些工具和方法很有效率，则必须集成它们。特别是这些工具必须能够共享开发过程的每个阶段产生的元数据。但是，共享这些信息的CASE工具还不常见，尤其是不同的供应商所提供的工具。所谓的集成的*CASE*（I-CASE）工具在整个生命周期中提供支持，但是这样的工具使用得相当少，因为它们对系统开发过程的某个特定阶段的支持能力很强，而在其他阶段支持能力却很弱。支持系统开发生命周期从项目认证和选择阶段到物理设计阶段的工具被称作*上层CASE工具*；支持实现和维护阶段的工具被称作*底层CASE工具*。

工具集成取决于一个用来构建和维护信息系统的正式的、详细的体系结构。这个体系结构包括工具间接口的正式定义、工具间的数据模型标准和贯穿整个生命周期的常见控制。这使得CASE工具的第三个特征——信息知识库（或*信息库*）对于我们讨论数据库开发过程很重要。**信息库**（*repository*）存储企业必须能够访问的事实和企业必须成功执行的过程的事实信息的知识库（Moriarty，1991）。因此，例如在信息库中要维护数据库开发的六个阶段中收集的所有信息。从某种意义上说，信息库本身也是一个数据库，它包括产生所有图、表单、报

表定义和其他系统文档所需的信息。信息库帮助系统和数据库分析员实现多个CASE工具间数据的无缝集成。

2.5 数据库开发中的人员管理

如图2-5所暗示的那样，数据库开发是项目的一部分。**项目**（project）是为了达到一个目标（具有开始和结尾）而计划采取的相关活动。项目从项目认证和规划的第一步开始，到实现阶段的最后一步结束。一个高级的系统或数据库分析员将作为项目主管。此人负责创建详细的项目计划、安置人员和监督项目团队。一个好的项目主管应具有领导能力、管理能力、能很好地处理与客户的关系和交流、解决技术问题的能力、冲突管理能力、团队建设能力和管理风险与变化的能力。

项目在项目认证和规划阶段启动并进行规划，在分析、逻辑设计、物理设计和实现阶段执行，在实现阶段结束时结束。项目团队在项目开始时组建，一个系统或数据库开发团队应该包括以下人员：

- *系统分析员*　分析业务环境，确定信息需求和信息服务以解决业务问题或抓住业务机遇。
- *数据库分析员*　专门负责决定信息系统数据库组件的需求和设计。
- *用户*　提供他们对信息需求的评估并监督开发系统是否满足他们的需求。
- *程序员*　设计和编写计算机程序，这些程序包括维护和访问数据库中的数据的命令。
- *数据库管理员和数据管理员*　负责现存的和将来的数据库，确保数据库间的一致性和完整性，且作为数据库技术的专家为其他项目组成员提供咨询和培训。
- *其他技术专家*　例如网络专家、操作系统专家、测试专家和文档专家。

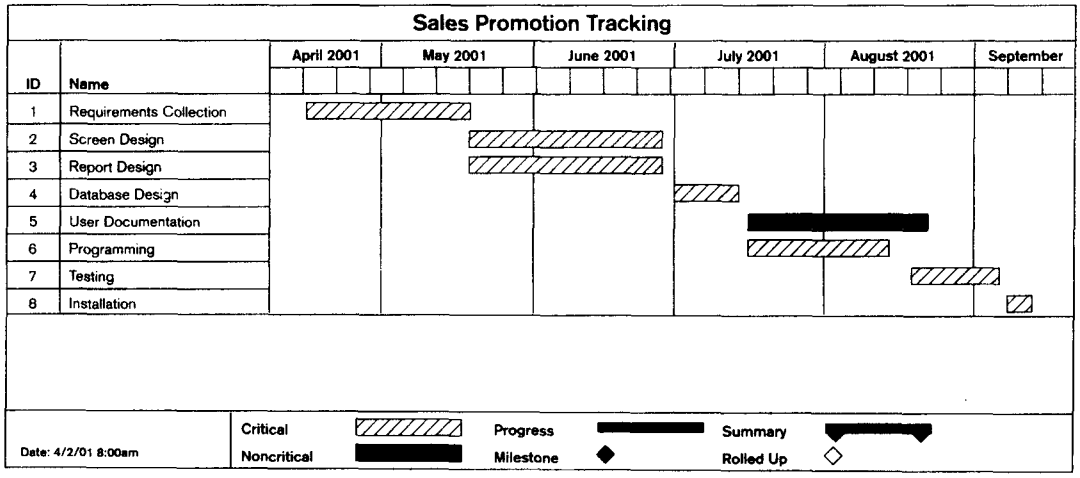
项目主管的责任是选择和管理上述这些人以组成一个有效率的团队。关于怎样管理一个系统开发项目团队，详见Hoffer、George和Valacich（2002）的著作。

项目结束分为自然结束和非自然结束。非自然结束的原因有系统不再有商业价值、系统的性能或开发团队不能被接受、项目耗尽了时间和资金却仍未完成或难以支持等等。为了确定项目进展是否符合计划并且未超过预算，项目主管应该建立项目活动的详细时间表，通常这种时间表用图的形式描述，如图2-7所示的例子图表。这些图表显示项目活动开始和结束的时间、谁负责完成这些活动、每个活动需要多少努力才能完成，以及活动间的前后顺序（即一个活动依赖于哪一个活动的输出）。

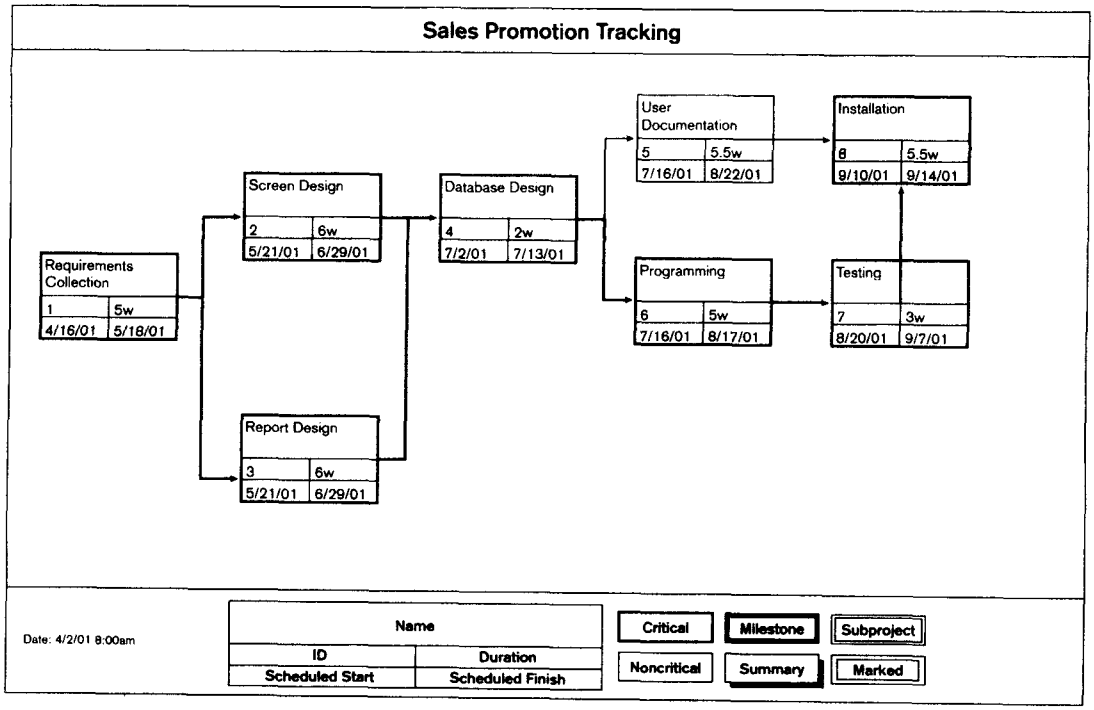
成功的系统开发项目的一个特征是有频繁的检查点，项目组成员在检查点报告到此为止时项目的结果。通常，这些结果要报告给项目组以外的人，包括其他用户、为项目提供资金的人、高级信息系统管理人员，还可能包括管理人员。要频繁设立检查点的原因如下：

- 验证项目的进度是否令人满意。
- 根据每日活动的详细记录回滚到上一步并验证项目的所有部分是否可以合到一起。
- 获取项目各方的新的投入（对持续数月的项目来说尤其重要）。

注意，第三个原因涉及到增量投入的概念。**增量投入**（incremental commitment）是系统开发项目中的一种策略，它在每个阶段后进行检查，在每次检查后调整项目的后续部分。增量投入允许那些对项目感兴趣的人仅仅承诺对项目的下一个阶段进行投入（只需有限的时间和成本），在一些结果显示出来后重新评估是否要对资源（人和金钱）进一步的投入。因此，当系统最初的概念被证实不再有价值时，也不会再浪费重要的资源。增量投入使一个项目可以很容易地改变方向或中止。



a) 甘特图



b) PERT图

图2-7 描述项目计划的图表

2.6 数据库开发的三层模式体系结构

在本章前面关于数据库开发过程的解释中提到了在一个系统开发项目上建立的几个不同的、但是相关的数据库视图或模型:

- 概念模式（在分析阶段建立）。
- 外部模式或用户视图（在分析阶段和逻辑设计阶段建立）。
- 物理模式或内部模式（在物理设计阶段建立）。

图2-8描述了数据库这三个视图之间的关系，重要的是要记住，它们是同一个组织数据库的视图或模型。也就是说，每一个组织数据库都有一个物理模式、一个概念模式以及一个或多个用户视图。因此，三层模式体系结构用观察同一数据集的不同方式定义数据库。

概念模式（conceptual schema）关于全部数据库结构的、与技术无关的规格说明。概念模式定义了整个数据库而不涉及数据怎样存储在计算机的二级存储器中。通常，概念模式用实体-联系（E-R）图或对象建模符号这样的图形格式来描述，我们把这种类型的概念模式称为*数据模型*。另外，概念模式的规格说明作为元数据存储在信息库或数据字典中。

在第1章中，用户视图被描述为用户执行某个任务所需的部分数据库的逻辑描述。因此，用户视图（或外部模式）也是独立于数据库技术的，但是它通常包括与特定的用户或用户组（如一个库存经理或应收款部门）相关的概念模式的一个子集。因为一个程序被特定的用户或用户组使用，所以，用户视图也是用于一个特定程序（例如订单录入程序）的模式的一个版本。因此，可以用逻辑（独立于技术）术语和编程语言术语（与编程语言语法一致）定义用户视图。通常，用户视图的最初描述是计算机屏幕显示、业务事务（如签署更新表单）和报表，因为它们通常描述一个处理显示、事务和报表的程序所需的所有数据。用户视图的一个逻辑版本可以表示成E-R图、对象图或关系。第5章中描述将E-R图翻译成关系，然后将所有关系合并为一个数据库完整关系的描述的过程。

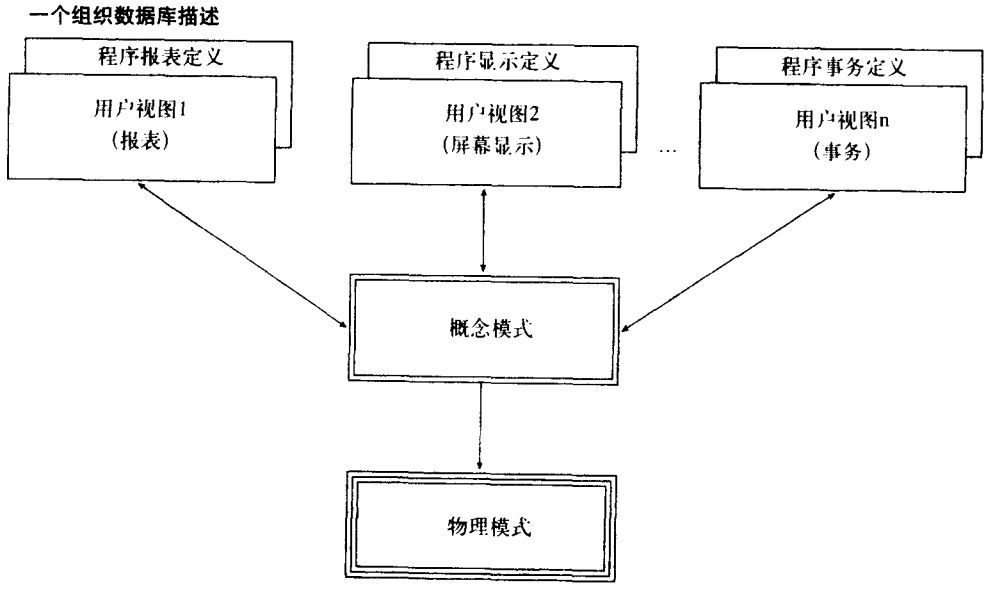


图2-8　三层模式数据库体系结构

物理模式（physical schema）包括概念模式的数据怎样存储在计算机的二级存储器中的规格说明。对数据库分析员和设计人员来说，重要的是物理数据库（物理模式）的定义，它提供了关于分配和管理存储和访问的数据所在的物理二级存储器空间的数据库技术的全部规格说明。

数据库开发和数据库技术是以数据库这三个模式间的区分为基础的。数据库开发项目的一个角色可能仅需处理与这三个视图中的一个相关的工作。例如，一个初学者可能设计用于一个

或多个程序的外部模式，而一个有经验的开发者将设计物理模式或概念模式。数据库设计问题在不同的层次上有很大的不同。为此，本书的组织关注于设计数据库这三种表示的问题：

- 概念模式 第3章和第4章讨论实体-联系建模，第14章讨论面向对象建模。这是描述概念模式或数据模型的两种不同图形符号。
- 外部模式 第5章讨论关系数据库、规范化、将E-R图转换成关系以及合并关系集，所有这些主题有助于外部模式的开发和分析。
- 物理模式 第6章讨论在设计一个物理数据库时必须作出的决定，第7章和第8章讨论怎样用一种定义物理模式和数据库的语言——SQL来详细说明这些决定。第15章讨论面向对象数据库的设计，附录D讨论混合的、对象-关系数据库的设计。

概念模式、外部模式和物理模式合在一起构成了数据库的三层模式体系结构。注意，即使图2-8中显示外部模式在最顶端，外部模式也并不是必须在概念模式之前开发。事实上，技术人员通常会迭代地开发概念模式和外部模式（参见图2-9）。通常，第一级的概念模式的开发是基于对组织的企业数据模型和对项目数据库需求的一般理解。然后开发用于每个事务、报表、屏幕显示和其他系统用途的外部模式（用户视图）。在大多数情况下，对外部模式的分析将产生概念模式中没有显示的新的属性和可能的实体及联系。所以，概念模式随着这些由所谓的自底向上的源确定的需求而增加，从而使得概念模式和外部模式一致。当新的用户视图被确定时，概念模式和外部模式的发展将重复以上过程。

为了开始开发一个数据库以及与它关联的应用程序，需要编写相关的物理模式的规格说明。在设计具有相关的物理模式的物理数据库时，除了概念模式和外部模式以外，还需考虑硬件和软件特征，以及用户对数据库性能的期望。在设计物理数据库时，将可能遇到不一致性或概念模式或用户视图的其他问题，所以，循环回到这些设计步骤是可能的（在图2-9中没有显示）。以后，当其他的用户需求确定下来后，这个过程重新开始。通常新的需求是批量处理的，并且在新的外部模式和概念模式设计的循环中一起考虑，所以，数据库不是经常地改变。周期性地修正数据库发生在系统开发的维护阶段。

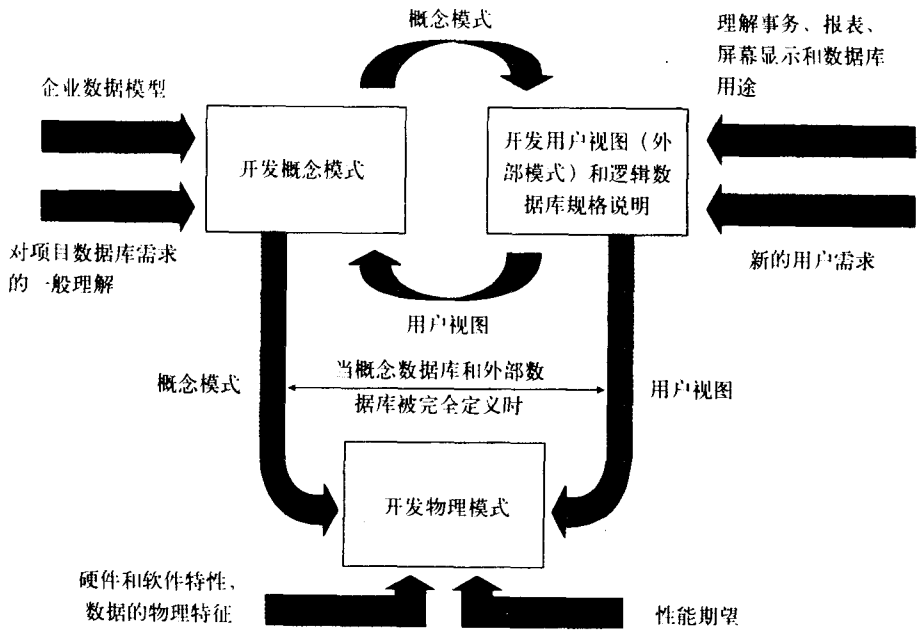


图2-9 开发一个数据库项目三层模式体系结构的过程

2.7　三层数据库定位体系结构

显然，所有数据库中的好的事情都和“三”有关!

当设计一个数据库时，你要选择把数据存放在何处。这个选择在物理数据库设计阶段作出。我们将在第6章中考虑一个更广范围的物理数据库设计决策，但是在这一节我们想概述能够影响数据库开发过程的主要的数据库体系结构的选择。

在第1章中已经介绍过，数据库分为个人数据库、工作组数据库、部门数据库、企业数据库和因特网数据库。个人数据库经常由最终用户自己设计和开发，仅仅由数据库专家给予培训和咨询帮助，它仅包含最终用户个人感兴趣的数据。有时候，个人数据库是从工作组数据库或企业数据库中提取出来的，这种情况下数据库专家经常编写一些提取例程来创建本地数据库。工作组数据库和部门数据库经常被最终用户、业务部门中的系统专家和中心数据库专家一起开发。这些人员的协同工作是必须的，因为在设计共享的数据库时必须权衡大量的问题：处理速度、易于使用、数据定义的差别和其他类似的问题。由于企业数据库和因特网数据库影响广、规模大，所以，通常由在集中的数据库开发小组中受过专业培训的数据库专家来开发。

这种数据库定位的视图对许多现代组织进行了一定的简化，它暗示一个数据库被严格地定位在这四个层次之一。事实上，为了平衡各种组织因素和技术因素，一个给定的信息系统的数据可能分布在多个计算机的位置或层次中。而且，为了利用处理速度、易于使用或易于在不同的平台上编程，一个数据库数据的不同类型的处理可以发生在不同的地点。在公司数据库中广泛使用基于浏览器接口导致了应用服务器市场的增长，应用服务器处理Web浏览器和后端数据库之间的业务软件和事务。大部分应用服务器的供应商，包括Sun、IBM和Oracle已经采用了Java 2企业版(J2EE)作为它们应用服务器的标准。可以考虑四层结构，其中包括客户服务器、应用服务器、Web服务器和数据库服务器，但通常考虑的是三层结构。

1. *客户层*

一个台式计算机或笔记本也称作表示层，它专门管理用户系统界面和本地化数据，在这一层上可以执行Web脚本任务。

2. *应用服务器/Web服务器层*

处理HTTP协议、脚本任务，执行计算和提供数据访问，所以该层称作处理服务层。

3.*企业服务器（小型机或大型机）层*

执行复杂的计算和管理来自组织间多个数据源的数据的合并，也称作数据服务层。

这三层体系结构在图2-10中描述。一个客户/服务器体系结构的有限视图仅仅考虑客户层和一个一般的服务器层。

在一个组织中，数据库和信息系统分层的体系结构与用于分布式计算的客户/服务器体系结构的概念相关。**客户/服务器体系结构**（client/server architecture）基于一个局域网环境，其中服务器上（称作*数据库服务器或数据库引擎*）的数据库软件执行来自客户工作站的数据库命令，每个客户的应用程序专注于它们的用户接口功能。实际上，整个概念数据库（以及访问这些数据库的应用处理例程）作为一个分布式数据库或单独但是相关的物理数据库而分布在本地的PC工作站、中间的服务器（工作组或部门）和一个中心服务器（部门或企业）上。Hoffer、George和Valacich（2002）以及Thompson（1997）的著作中概述了为什么这种体系结构越来越流行。简单地说，使用客户/服务器体系结构的原因是：

- 它可以在多个处理器上同时处理同一个应用，因此改善了应用的响应时间和数据处理速度。

- 它可以利用每个计算机平台最好的数据处理特性（如PC的高级用户界面与小型机和大型机的计算速度）。
- 可以混合使用各种客户端的技术（装配Intel或Motorola处理器的个人计算机、网络计算机、信息站等）和共享公共数据。另外，你可以在任何层改变技术而仅对其他层系统模块的影响很小。
- 能够使处理靠近需处理的数据源，从而改进响应时间并减少网络通信量。
- 它允许和鼓励接受开放系统标准。

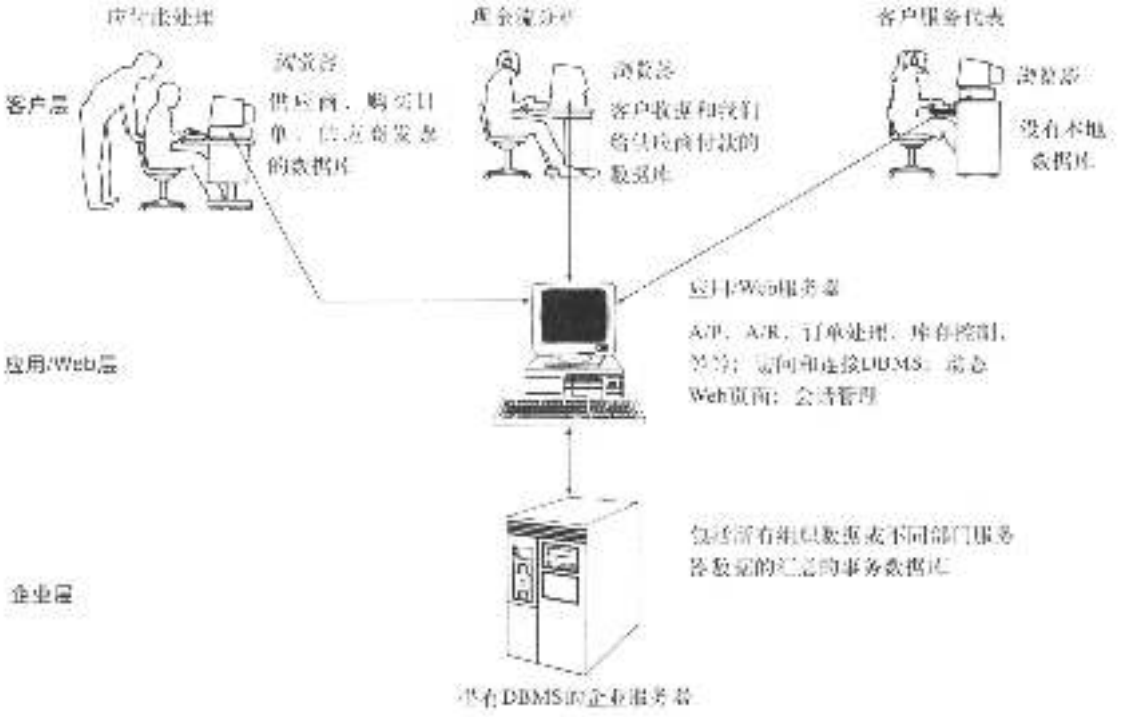


图2-10 三层客户/服务器数据库体系结构

对数据库开发而言，使用一个多层的客户/服务器体系结构开发数据库最有意义之处在于，易于将数据库开发和维护数据库的模块与向最终用户表示数据库内容的信息系统模块分隔开。表示例程能够使用像PowerBuilder、Java和Visual Basic这样的语言来提供易于使用的图形化的用户界面。通过中间件（参见第9章），表示例程能够通过层间相互作用来访问例程。该例程访问所需数据并分析这些数据以形成所需信息。作为一个数据库开发人员和程序员，你可以在这三层中的任何一层工作，开发必需的软件。

我们将在第9章中更详细地讨论客户/服务器体系结构，在该章将略述怎样决定在计算机网络的多个层中如何分布数据。

2.8 为松谷家具公司开发一个数据库应用

在第1章中介绍了松谷家具公司。这个公司自1950年以来一直使用关系数据库管理技术，在个人计算机、部门小型机和公司服务器上具有大量的数据库（参见图1-6松谷家具公司的计算机网络的总图示）。

Helen Jarvis是松谷家具公司的家庭办公室家具的产品经理，她知道这个正在成长的产品系列的竞争越来越激烈。因此对松谷家具公司而言，Helen能够更透彻地分析产品的销售变得越来越重要。这些分析常常是即席的，会被迅速变化和无法预期的业务环境、家具商场经理的评论、贸易行业的观点或经验所驱动。Helen已经请求利用易用的界面直接访问销售数据，这样她能够为各种营销问题寻找答案。

Chris Martin是松谷家具公司信息系统开发领域的系统分析员。Chris已经在松谷家具公司工作了5年，在松谷家具公司的一些业务领域具有丰富的信息系统方面的经验。由于他曾经从事过几个系统开发项目，在西佛罗里达大学接受过信息系统教育并在松谷家具公司接受过全面培训，所以Chris成长为公司最好的系统开发人员之一。Chris擅长数据建模，他熟悉公司内使用的几个关系数据库管理系统。鉴于他的经验、技能和他有空闲时间，所以信息系统的主管分配Chris与Helen一起工作来解决Helen对营销支持系统的需求。

因为松谷家具公司在系统的开发中采取谨慎的态度，尤其因为采用了数据库方法，所以公司已经拥有一个相当完全的信息系统体系结构，包括所有运作的业务功能的数据库。因此，Chris可以从现存的数据库中提取Helen所需的数据。当Helen请求建立单机的数据库时，松谷家具公司信息系统体系结构要求这样的系统，所以，对数据的没有结构的和无法预见的使用将不会干涉访问需要支持有效的事务处理的运行数据库。

再则，因为Helen的需求是数据分析而不是创建和维护，而且是个人的需求，不是机构的需求，所以，Chris决定采用一种原型法和生命周期法的结合的方法来开发Helen所请求的系统。采用结合的方法意味着Chris将按照生命周期的步骤，但是以一种快速的、粗略的方式进行这些步骤，而不是完整地采用原型法。因此，他将迅速地启动项目和进行规划（包括确定这个系统在公司的信息系统体系结构中的位置），然后与Helen紧密的合作，不断地分析、设计和实现以建立一个Helen需要的系统原型。因为这个系统是个人的，而且仅需要一个有限范围的数据库，所以，Chris希望原型最终成为Helen可以实际使用的系统。Chris选择用Microsoft Access来开发系统，Access是松谷家具公司在开发个人数据库时经常使用的技术。然而，在多数情况下，应该说明定义数据库结构所必须的SQL，因为这些命令能用于定义任何一个采用关系数据库管理的数据库。

2.8.1 匹配用户需求和信息系统体系结构

Chris开始开发家庭办公室家具的数据库和相关的营销支持系统的项目时初次会见了Helen。Chris问Helen关于她的业务领域，记录了业务领域目标、业务功能、数据实体类型和其他她要处理的业务对象。此时，Chris听多于说，以便专注于理解Helen的业务领域，他不时地插入问题以确保Helen不会遗漏任何她需要信息系统提供的计算机屏幕显示或报表。Chris问的是非常一般的问题，尽可能地使用业务或营销术语。例如，Chris问Helen她在管理家庭办公室产品时面临哪些问题，什么人、什么地点和事物是她在工作中感兴趣的，她需要分析多久以前的数据，以及在业务中什么事件发生时她将感兴趣。

表2-4总结了Chris从初次会见中了解到的内容。这张表列出了Helen提到的各种业务对象，并且这些对象已经在公司的信息系统体系结构中了。图中仅有少量的实体不在信息系统体系结构中，然而体系结构覆盖Helen提到的数据的主要种类，现有的信息系统管理着这些数据。因此，Chris相信，即使不是全部数据，大部分Helen想要的数据已经存在于公司的数据库中。

表2-4 产品行销支持系统的业务对象

规划对象	家庭办公室产品系列追踪的目标	对象是否已经在松谷家具公司的ISA中
目标	家庭办公室产品年销售增长率至少达16%	否
	家庭办公室产品年利润增长率至少达10%	否
	家庭办公室产品同一顾客反复销售增长至少达5%	否
	每类办公家具成品的销售超过目标	否
	使家庭办公室产品高于公司所有产品的平均水平	否
	将完成办公室产品订单的时间缩减5%	否
	将接收办公室产品发票上最后付款的时间缩减5%	否
组织单元	营销部	是
	办公室家具产品系列管理部	否
	财务部	是
	订购部	是
组织地点	区域销售办公室	是
	公司总部	是
业务功能	接收付款	是
	产品开发	—
	人口统计分析	否
	目标市场分析	是
	营销和销售	—
	履行订单	是
	销售汇总	是
	获取订单	是
实体类型	CUSTOMER	是
	PRODUCT	是
	PRODUCT LINE	是
	ORDER	是
	INVOICE	是
	PAYMENT	是
信息系统	订单处理	是
	销售管理	是

Chris在与Helen再次会谈之前做了两次快速的分析。第一，他确定和Helen所提到的数据实体相关的数据所在的所有数据库。利用这些数据库，Chris列了一个数据实体的所有数据属性的清单，他认为Helen在分析家庭办公室家具市场时可能会对这些属性感兴趣。Chris以前参与开发公司标准销售跟踪和预测系统以及成本会计系统的经历有助于他从表2-4的信息中推测Helen可能需要什么样的数据。例如，每类办公家具成品的销售超过目标意味着Helen在她的系统中需要年销售目标；再如达到至少年销售增长率16%的目标意味着系统需包括每个产品前一年的订单。Chris还总结出，Helen的数据库必须包括所有产品而不仅仅是办公家具系列，因为Helen想把她的产品同其他产品相比较。然而他可以消除每个数据实体的许多数据属性。例如，Helen不需要许多客户数据，如地址、电话号码、联系人、存储尺寸和销售员。尽管如此，他还是包含少数属性（客户类型和邮政编码，在表2-4中未给出），他加入这些属性是因为这些属性在销售预测系统中是非常重要的。

第二，Chris根据这张表绘制了表示带有相关属性的数据实体和实体间主要联系的图形化的数据模型。Chris希望把这张数据模型给Helen看后，能够减少系统开发过程中分析阶段的时间（因此减少概念数据建模的时间）。图2-11是Chris绘制的初步数据库的数据模型图。表2-5列

出了每个实体的数据属性，Chris认为这些属性是Helen需要的系统中应有的。Chris在表2-5中仅仅列出了现存数据库中的基本数据属性，因为Helen为了进行分析将可能用各种方式组合这些数据。

2.8.2 分析数据库需求

Chris与Helen的首次会面仅有45分钟，但是他感觉到对与Helen分享他收集的数据以及他后续的工作产生了许多想法。他计划与Helen进行一次2个小时的会见，以讨论他的想法。在会见前，他发给Helen一个大致的项目进度表，列出了他计划的步骤以及每步所需的时间。由于原型法是一个用户驱动的过程，期间由用户决定什么时候停止在新的原型版本上迭代，所以Chris仅能提供项目某一步持续时间的粗略估计。为此，Chris的上司认为该项目应该与Helen商议所需时间，而不是确定一个固定的时间。

Chris在与Helen的第二次会面中做了很多事。他系统地解释了图2-11中每个数据实体的含义，数据实体每个相关属性（在表2-5中）的含义，以及实体间每条线所表示的业务策略和过程。例如，Chris解释每个订单是根据一张发票付账，而每张发票对应且仅对应于一张订单。Order_Number是每份订单的惟一标识，一张订单属于一个客户。Chris认为Helen可能想知道的关于订单的数据还包括下订单的日期和完成订单的日期（订单上的产品最后发送的日期）。Chris也解释了Payment_Date属性表示某个订单客户最近的支付日期，不管是全部支付还是部分支付。

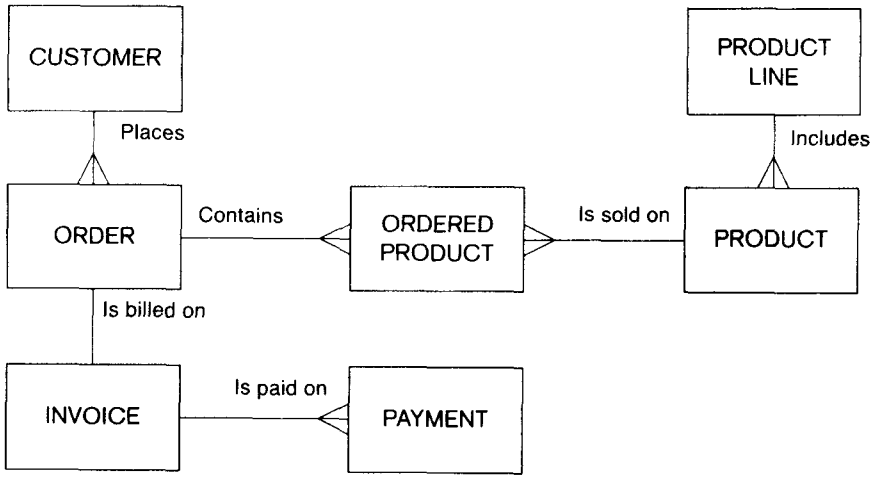


图2-11　产品营销支持系统的初步数据模型

表2-5　初步数据模型中实体的数据属性

实体类型	属　性
CUSTOMER	Customer_Identifier
	Customer_Name
	Customer_Type
	Customer_ZIPCODE
PRODUCT	Product_Identifier
	Product_Finish
	Product_Price
	Product_Cost
	Product_Annual_Sales_Goal

（续）

实体类型	属 性
	Product_Line_Name
PRODUCT LINE	Product_Line_Name
	Product_Line_Annual_Sales_Goal
ORDER	Order_Number
	Order_Placement_Date
	Order_Fulfillment_Date
	Customer_Identifier
ORDERED PRODUCT	Order_Number
	Product_Identifier
	Order_Quantity
INVOICE	Invoice_Number
	Order_Number
	Invoice_Date
PAYMENT	Invoice_Number
	Payment_Date
	Payment_Amount

在讨论期间，Helen告诉Chris她想要的另外一些数据（客户购买松谷家具公司产品的年数和完成每个订单所必需的发货量）。Helen还指出，Chris只提供每类产品一年的销售目标，而她需要去年的销售目标和今年的销售目标。当Helen对数据模型进行反馈时，Chris问她想怎样使用这些数据，此时，Chris并不想完全搞清这些问题，因为他知道Helen还没有象正在开发的信息集那样处理过这样的数据。因此，Helen可能还没有确定需要什么样的数据或对这些数据作些什么。他的目的是为了理解Helen使用数据的一些方式，这样，他可以开发一个最初的原型，包括数据库和一些计算机显示或报表。表2-6中列出的是Helen最后同意的她所需要的属性列表。

表2-6 最终数据模型中实体的数据属性

实体类型	属 性*
CUSTOMER	Customer_Identifier
	Customer_Name
	Customer_Type
	Customer_ZIPCODE
	Customer_Years
PRODUCT	Product_Identifier
	Product_Finish
	Product_Price
	Product_Cost
	Product_Prior_Year_Sales_Goal
	Product_Current_Year_Sales_Goal
	Product_Line_Name
PRODUCT LINE	Product_Line_Name
	Product_Line_Prior_Year_Sales_Goal
	Product_Line_Current_Year_Sales_Goal
ORDER	Order_Number
	Order_Placement_Date
	Order_Fulfillment_Date
	Order_Number_of_Shipments
	Customer_Identifier

（续）

实体类型	属　性*
ORDERED PRODUCT	Order_Number
	Product_Identifier
	Order_Quantity
INVOICE	Invoice_Number
	Order_Number
	Invoice_Date
PAYMENT	Invoice_Number
	Payment_Date
	Payment_Amount

注：*斜体表示与初步设计的列表不同的部分。

2.8.3 设计数据库

因为Chris采用原型法并且在前两次与Helen的面谈中迅速确定了她所需要的数据，所以Chris可以立即建造一个原型。首先，Chris从公司的数据库中提取Helen所提到的数据实体和属性。Chris用SQL查询语言来创建所有这些文件。Helen想要的一些数据要从原始的、操作的数据中计算出来（如Customer_Year），但是利用SQL，Chris可以很容易地指定这些计算。抽取后，可以为每个数据实体产生一个ASCII文件，文件中的每一行包括数据模型中和数据实体相关的所有数据属性，文件中的所有行是它所对应实体的不同实例。例如，PRODUCT Line数据实体的ASCII文件中的每一行包括产品系列名称、去年和今年的年销售目标数据。

第二，Chris将与Helen讨论后得到的最终的数据模型转换成一系列的表，其中表的列表示数据属性，表的行表示属性值的不同集合。表是关系数据库的基本构件，这是Microsoft Access的数据库风格。图2-12与图2-13显示的是Chris创建的PRODUCT LINE表和PRODUCT表（包括与之相关的数据属性）的定义（我们同时将SQL和Access的定义表示在图2-13中）。Chris作这样的转换是为了使每张表都有一个称为**主键**（primary key）的属性，它在表的每行中都不相同。表的其他主要性质包括在每行中，每个属性仅仅有一个值，因此如果我们知道标识符的值，那么其他每个属性都仅能有惟一值。例如，对于任何一个产品系列，其今年的销售目标仅有惟一值。

```
CREATE TABLE PRODUCT_LINE
    (PRODUCT_LINE_NAME          VARCHAR (40)   NOT NULL PRIMARY KEY,
    PL_PRIOR_YEAR_GOAL          DECIMAL,
    PL_CURRENT_YEAR_GOAL        DECIMAL);
```

图2-12　PRODUCT_LINE表的SQL定义

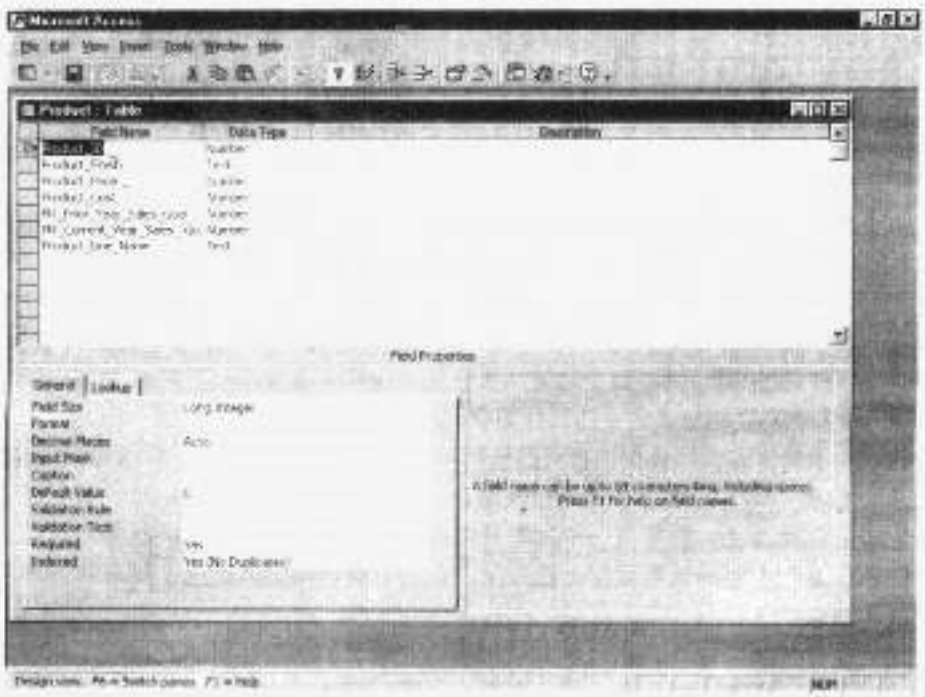
数据库设计包括为每个属性（Access称为属性字段）指定格式或性质。这些设计决定在这个案例中很容易得出，因为大多数属性已经在公司的数据字典中设计好了。在表2-13b中，通过字段名左边的箭头显示Product_ID字段的性质。但是，少数属性（例如ORDER表中的Order_Number_of_Shipment属性）是利用松谷家具公司的数据库中的原始数据计算得来的，所以Chris不得不为这些属性创建规格说明。

Chris关于数据库设计还必须作出的其他主要决策包括在物理上如何组织数据库以便以最快的速度对Helen的查询做出响应。因为数据库用于决策支持的，所以无论是Chris还是Helen都无法预见将来会发生的所有查询，因此Chris必须根据经验做出物理设计的选择，而不是根据使用

数据库的方式的知识来进行选择。Microsoft Access和SQL允许数据库设计人员做的关键的物理数据库设计决定是在哪些属性上创建索引（索引就像图书馆中的卡片目录，通过索引可以很快找到具有共同特征的行）。所有的主键属性（如ORDER表中的Order_Number）是可以进行索引的，因为它们在表中的每一行的值都是惟一的。另外，Chris用了一条经验法则：为任何一个具有超过10个不同的值且Helen可能用它来分割数据库的属性创建索引。例如，Helen表示她使用数据库的一种方式是按产品使用的漆来查看销售情况，因此在PRODUCT表上用Product_Finish属性创建索引可能会有意义。然而，松谷家具公司仅使用六种漆或木料类型，所以，它们不是很好的索引候选者。另一方面，Order_Placement_Date（称作辅码，因为在ORDER表中不只一行具有同一个值）是一个好的索引候选，Helen想用它来分析不同的类型时期的销售。在图2-13a中查找主键，在图2-13b中，靠近屏幕底部有一个Indexed框，查看该框可以发现选择的属性是否被索引，如果已经索引，那么，该索引是否允许重复（主键是不可以重复的）。

```
CREATE TABLE PRODUCT
    (PRODUCT_ID                    INTEGER NOT NULL PRIMARY KEY,
     PRODUCT_FINISH                    VARCHAR (20),
     PRODUCT_PRICE                     DECIMAL,
     PRODUCT_COST                      DECIMAL,
     PR_PRIOR_YEAR_GOAL                DECIMAL,
     PR_CURRENT_YEAR_GOAL              DECIMAL,
     PRODUCT_LINE_NAME                 VARCHAR (40),
FOREIGN KEY (PRODUCT_LINE_NAME) REFERENCES
   PRODUCT_LINE (PRODUCT_LINE_NAME));
```

a) PRODUCT表的SQL定义



b) PRODUCT表的ACCESS定义

图2-13 PRODUCT表的SQL定义或ACCESS定义

图2-14显示了Chris为家庭办公室营销数据库开发的数据模型的原型。每个框代表数据库中的一张表，表中的属性在相关的框中列出。尽管表示关系的符号与图2-11略有不同，但是意义是一样的。例如，在图2-14中，从CUSTOMER实体到ORDER实体有一条线，在靠近CUSTOMER实体一端显示1，而在靠近ORDER一端显示一个表示无穷的符号，这与图2-11中的Place关系具有相同的意义。

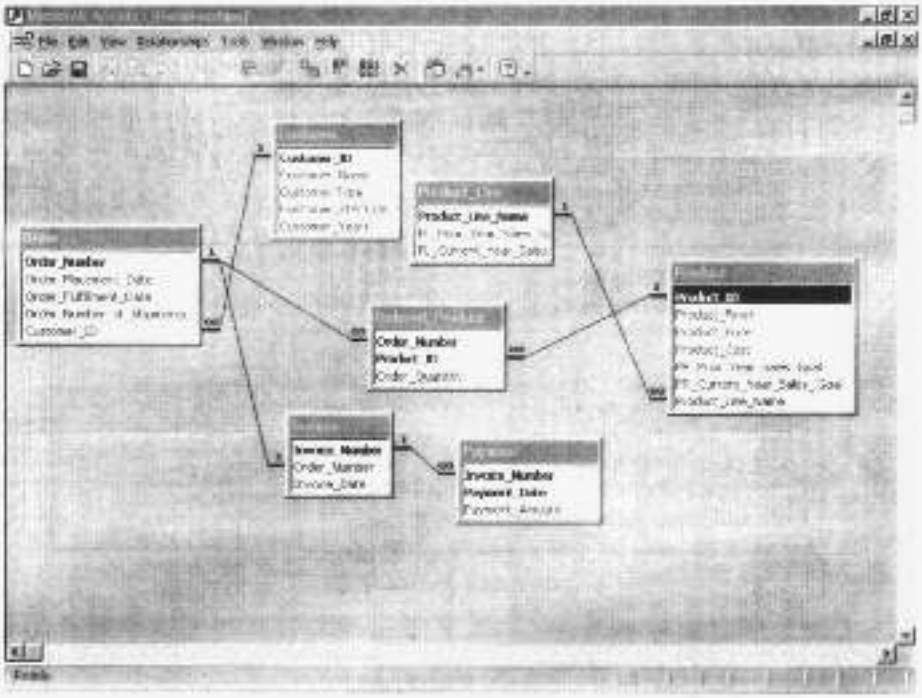


图2-14　家庭办公室产品营销支持系统的数据库定义——Access数据模型原型

2.8.4　使用数据库

Helen要使用Chris为即席问题所建造的数据库。所以，Chris将培训她如何使用Microsoft Access，尤其是数据库查询特性。Chris想等系统设计得更好一些并且Helen从构建原型的过程中学习了数据库和Microsoft Access的一些知识后再培训她。然而，Helen提出她希望定期提出一些标准问题。Microsoft Access提供了开发一些类型的预写例程的功能，这使得Helen可以更容易地回答这些标准问题（所以她不需要从头开始为这些问题编程）：

- *表单*　按预先确定的格式定义的属性集合，全部基于一个数据库记录。因此，一个表单可以包括一个给定顾客的数据，也可以包括一个订单的数据和与此订单相关的顾客的属性。
- *报表*　按预先确定的格式定义的属性集合，它基于许多不相关的记录。报表通常包括每个记录的相同属性，例如，一个报表可以将所有销售低于目标的产品列出产品编号、今年的销售量和今年的销售目标。报表通常包括页码、每页的标题、报表的打印日期和其他描述信息。
- *查询*　一个用特定的查询语言（如按例查询）编写的问题，它从数据库中获取答案。查询的结果是一张表，列是用户想要查看的属性，行是那些属性满足用户条件的属性的不同实例。

在原型法的开发过程中，随着Helen越来越清楚地说明她想要系统完成什么功能，Chris可

以开发出许多这些例程的例子。然而，在开发阶段的早期，Chris想开发创建第一个原型的例程。Helen想要的标准信息集之一是家庭办公室产品系列中每一个产品的清单，其中显示每个产品目前的总销售量与本年度销售目标的比较。一个Access查询就可以产生这个结果。Helen想按照一种固定的格式显示查询的结果，这是一个使用报表的机会，但是现在Chris仅仅教Helen用查询达到这个目的。

图2-15显示的是产生这个产品清单的查询，图2-16是输出的一个例子。图2-15中的顶部显示的是这个查询所需的数据模型中的联系，在底部显示的是按例查询（QBE）的代码。如果选中一个字段，那么这个字段将包括在结果中。Product_Line_Name和Order_Placement_Date这两个字段仅仅用于选择家庭办公室家具和2000年的订单。PR_Current_Year_Goal字段和Product_ID字段上的Group By子句告诉Access对每个产品的销售量进行汇总。这个例子中只有有限的数据，所以图2-16中Total Sales的结果是相当小的，但是格式是图2-15中查询的结果。

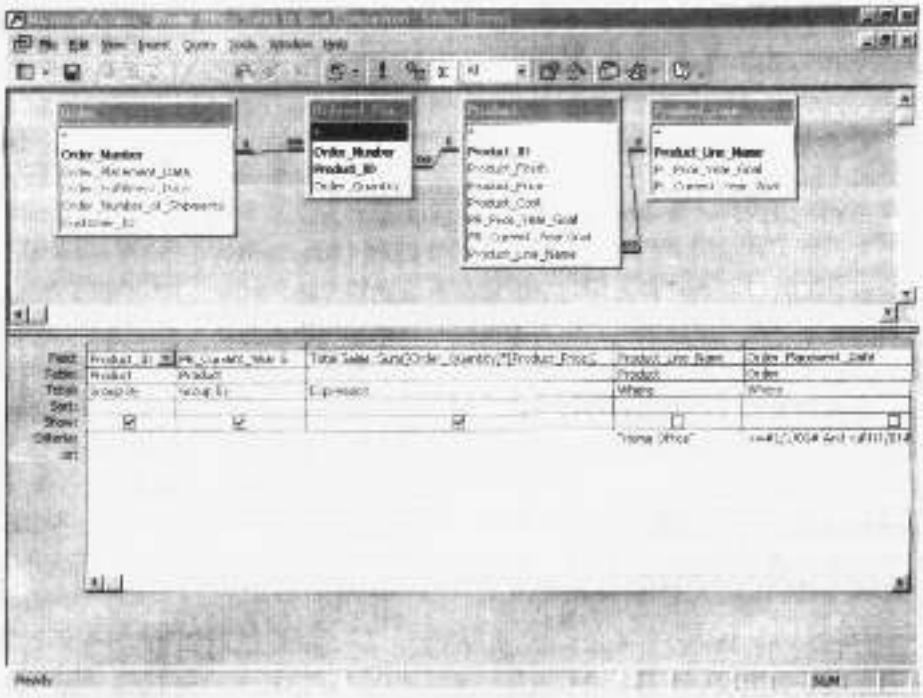


图2-15 家庭办公室产品销售与目标比较的查询

Product_ID	PR_Current_Year_Sales	Total Sales
3	$23,500.00	9375
5	$26,500.00	4550
7	$17,000.00	2250
10	$22,500.00	4400

图2-16 家庭办公室产品系列销售比较

Chris现在准备再次会见Helen，看看原型是否符合她的需求。Chris通过运行Access和显示图2-14～图2-16这样的屏幕向Helen展示了他设计的系统。当Helen提出建议时，Chris能够即时做出少量修改，但是Helen查看后的许多意见要等到他进行更仔细的工作之后才能解决。

由于篇幅限制，我们无法仔细介绍开发家庭办公室家具营销支持系统项目的每一步。在Helen觉得满意，即她所需的所有属性都在数据库中之前，Chris和Helen大约进行了12次面谈，Chris写的标准查询、表单和报表对她很有用，而且，她知道如何为那些无法预期的问题编写查询。当Helen使用系统遇到问题时，包括编写更复杂的查询、表单或报表，Chris将随时为Helen提供咨询支持。Chris和Helen做出的最后决定是判断最终原型的性能是否足够高效，如果是，则无需重写或重新设计原型。Helen现在准备使用这个系统。

2.8.5 管理数据库

家庭办公室产品营销支持系统的管理是相当简单的。Helen决定每周从松谷家具公司的运作数据库下载新的数据到她的Access数据库中。Chris写了一个嵌入了SQL命令的C语言程序以执行必要的提取工作，并且用Visual Basic中写了一个Access程序以从提取结果中重建Access表，他计划在每个星期天的晚上执行这些工作。Chris还要更新公司信息系统体系结构模型以包括家庭办公室产品营销支持系统。这一步非常重要，因为当Helen的系统中包含的数据的格式发生变化时，公司的CASE工具可以警告Chris这些改变可能会影响她的系统。

本章小结

本章讨论了开发数据库及其应用的过程。数据库开发从企业数据建模开始，此时建立组织数据库的范围和一般内容。企业数据建模是为一个组织开发信息系统体系结构（包括数据、过程、网络、人、事件和原因）的全部过程的一部分。为开发信息系统体系结构而采用的一个流行的方法是信息工程，它是一种自顶向下的信息系统规划方法。

信息系统规划必须考虑到组织目标、关键成功因素和组织的问题领域。在信息系统规划期间，数据实体应该与组织的其他规划对象相联系，这些对象包括组织单元、地点、业务功能和信息系统。业务功能通过功能分解过程可以分解以各种详细级别来表示。数据实体间的联系和其他组织规划对象能够表示在更高级别的规划矩阵中，规划矩阵有助于理解联系的模式。

无论是从信息系统规划还是从一个特定的请求（例如Helen Jarvis要求建立一个家庭办公室产品营销支持系统），一旦数据库的需求确定，就可以组成一个项目团队以开发各个部分。项目团队按照某种系统开发过程进行开发，例如系统开发生命周期法或原型法。系统开发生命周期法可以表示为7个步骤：1）项目论证和选择阶段；2）项目开始和规划阶段；3）分析阶段；4）逻辑设计阶段；5）物理设计阶段；6）实现阶段；7）维护阶段。数据库开发活动发生在这些相互重叠的每一个阶段中，而反馈的发生可能导致项目返回到上一个阶段。在原型法中，数据库及其应用是通过系统开发人员和用户的紧密交互而反复改进的。当数据库应用规模很小、单机运行且系统用户数很少时采用原型法更好。

在整个系统开发过程中，CASE工具用于开发数据模型并维护数据库和应用的元数据。信息库维护所有的文档。在数据库开发项目中，许多人可以使用CASE工具和相关的信息库，这些人员包括：系统分析员、数据库分析员、用户、程序员、数据库和数据管理员以及其他技术专家。当项目的一个新的重要的部分完成时，并记录在信息库中时，应该设置一个检查点。在一个检查点上，那些为项目工作的人和为项目提供资金与人力资源的人可以评估项目进度，并在基于递增完成的情况决定是否再进行投入。

数据库开发项目的工作人员为数据库处理三种视图或模式：1）概念模式，它提供完全的、与技术无关的数据库视图；2）物理模式或内部模式，确定完全的存储在计算机二级存储器中的数据库；3）外部模式或用户视图，描述与一个特定用户相关的数据库的子集。

现代的数据库及其应用可能分布在多个计算机上。尽管可以存在许多层次，但是处理数据库

的客户/服务器体系结构分为三层：1）客户层，将数据库内容显示给用户；2）应用/Web服务层，分析数据库的内容和管理用户会话；3）企业服务器层，将组织间的数据合并为一个组织资产。

在本章结尾，我们介绍了假想的松谷家具公司中的数据库开发项目。这个家庭办公室家具产品的营销支持系统说明了个人数据库管理系统的用法，以及Microsoft Access和开发仅用于检索的数据库的SQL编码的用法。这个应用中的数据库包括从企业数据库中提取的数据，然后存储到客户层一个单独的数据库中。开发这个数据库应用采用了原型法，由于用户Helen Jarvis具有相当多非结构化的需求，所以，只有通过开发和精化系统的不断迭代才能更好地了解其需求。

本章复习

关键术语

业务功能	客户/服务器体系结构	计算机辅助软件工程（CASE）
概念模式	企业数据建模	功能分解
增量投入	信息工程	信息系统体系结构（ISA）
物理模式	项目	原型法
信息库	系统开发生命周期（SDLC）	自顶向下规划

复习问题

1. 定义以下术语：
   a. 信息系统体系结构（ISA）　b. 系统开发生命周期（SDLC）
   c. 客户/服务器体系结构　d. 增量投入
   e. 企业数据模型　f. 概念数据建模
2. 将下列术语与定义匹配起来：

_____概念模式	a. 重复分解一个功能到更细的细节
_____业务功能	b. 定期检查系统开发项目点
_____原型法	c. 一种快速的系统开发方法
_____系统开发生命周期法	d. 业务数据的全面描述
_____功能分解	e. 对信息系统需求获得一个广泛的理解
_____自顶向下规划	f. 一组相关的业务过程
_____增量投入	g. 存储在二级存储器中的数据结构
_____物理模式	h. 一个结构化的、逐步的系统开发方法

3. 比较下列术语：
   a. 物理模式；概念模式　b. 系统开发生命周期法；原型法
   c. 自顶向下规划；功能分解　d. 企业数据建模；信息工程
   e. 信息库；计算机辅助软件工程
4. 列出和解释信息工程的四个阶段。
5. 描述信息工程规划阶段的三个步骤。
6. 列出并解释三个信息工程的战略上的规划因素。
7. 列出和定义五个关键的组织的规划目标。
8. 说明在信息系统与数据库开发中进行功能分解的意义。
9. 说明信息系统开发中信息系统规划矩阵的用途。
10. 给传统系统开发生命周期的七个阶段命名，并说明每一阶段的目的与交付产品。
11. SDLC的七个阶段中，哪一阶段出现数据库开发活动？

12. 详细说明用原型法进行系统开发的步骤。哪个数据库开发活动出现在每个原型法步骤中？

13. 说明同一数据库的用户视图、概念模式和物理模式之间的不同。

14. 必须以特定的顺序设计外部模式与概念模式吗？说明原因。

15. 详细说明三层数据库体系结构。

16. 在三层数据库体系结构中，在某一个层上可能没有数据库吗？若不可能，为什么？若可能，请举例。

17. 列出信息系统体系结构（ISA）的六个主要组成部分。其中哪个部分与数据库设计有关？为什么？

18. 将系统开发生命周期法（SDLC）比作瀑布有何意义？

19. 在数据库开发过程中何时绘制实体-联系图？同一数据库的不同图表是如何互相区别的？

20. 实现客户/服务器数据库体系结构的原因是什么？

问题和练习

1. 瀑布比喻能够恰当地形容SDLC吗？说明你对这个比喻的疑问。

2. 按照更有用的顺序重新安排图2-3的行与列。你为什么选择这个新顺序来安排行和列？你现在运用这个重组的矩阵的目的是什么？

3. 列出松谷家具公司的企业数据模型（见图2-1）中可以出现的三个其他的实体。

4. 将你所在的商学院或其他的学院看作一个企业。

   a. 定义若干个功能并且至少功能分解到三个层次。

   b. 定义若干个主要的数据实体类型，并绘制初步的企业数据模型（所用符号与图2-1类似）。

   c. 以练习a的最低层次功能为行，练习b的数据实体类型为列，开发规划矩阵。参照图2-3，填写矩阵单元。

   d. 定义学院的四个重要成功因素（CSF）。

   e. 你的商学院或其他学院是否可以从数据的多层体系结构中受益？说明其中的原因。

5. 考虑你所参加的学生俱乐部。

   a. 定义若干个俱乐部使用的信息系统（人工或自动的）。

   b. 定义若干个主要的数据实体类型，并绘制初步的企业数据模型（使用的符号与图2-1类似）。

   c. 开发一个信息系统-数据实体的规划矩阵。填写每个单元格以表示每一信息系统是如何与每一数据实体相互作用的（单元格用字母C表示该实体创建的新的实例，R表示检索那个实体的数据，U表示更新实体的数据值，D表示删除实体的实例）。

   d. 重组练习c的答案中的行和列，尽你所能来创建一个具有实体的单元沿着主对角线的而空白单元远离主对角线的矩阵。这个重组的矩阵能说明什么问题？

6. 假设一个业务功能-数据实体的规划矩阵。假设通过研究这个规划矩阵，你确定业务功能中的3个功能来提供5个数据实体的大部分使用。这对确定开发用的数据库有什么暗示？

7. 考虑表2-3，根据这张表中的信息开发一个假设的信息系统-实体类型矩阵。假设信息系统中的每个实体可以创建、检索、更新和删除数据，单元格中用字母C表示创建，R表示检索，U表示更新，D表示删除。由于所列的信息系统包括事务处理和管理信息系统，你将遵守什么样的实体使用模式？

8. 考虑表2-3，开发关于产品开发业务功能的一个假设的功能分解，类似于图2-2中显示的

订单履行功能。你的图中有任何与图2-2相同的子功能吗？为什么？

9. 解释企业数据模型和概念数据模型之间的区别。每一个模型代表多少个数据库？每个模型所对应的组织的范围是多少？其他明显的区别是什么？

10. 在物理数据库设计和数据库开发的创建阶段，你是否有可能想返回到逻辑数据库设计阶段？说明其中的原因。如果可能，给出一个例子说明在物理数据库和设计阶段会让你重新考虑先前的概念和外部数据库设计。

11. 比较数据库开发在概念数据建模阶段的自顶向下的本质和逻辑数据库设计阶段的自底向上的本质。在这两个数据库设计阶段考虑的信息类型存在什么主要差别？

12. 用原型法进行系统开发的目标是迅速建造和重建一个信息系统，用户和系统分析员从原型的使用中了解什么特征应该包括在改进后的信息系统中。由于最后的原型不是必须成为一个工作系统，哪里是你认为最理想的开发原型的地点：个人计算机、工作组计算机、部门计算机，还是企业服务器？你的回答基于什么假设？

13. 考虑一个你经常与之打交道的组织，如银行、信用卡公司、大学或保险公司，你从中接受过一些计算机产生的消息，如每月结算表、交易单等等。描述你从组织收到的每个消息中的数据作为它自己的用户视图，用图2-1中的符号表示这些视图。现在，合并所有视图到一个概念数据模型，同样是使用图2-1中的符号。在合并不同的用户视图时，你观察到了什么？不同的用户视图间存在不一致的地方吗？一旦你创建了概念数据模型，你愿意再修改用户视图吗？

14. 考虑图2-10，它描述了一个假设的三层数据库体系结构。确定图中各个数据库间潜在的数据重复。这些重复可能导致什么问题？这些重复违反了第1章中列出的数据库方法的原则了吗？说明其中的原因。

15. 考虑图2-11，解释ORDER和INVOICE间连线的含义，以及INVOICE和PAYMENT间连线的含义。它们是怎样说明松谷家具公司与客户交易的？

16. 回答下列有关图2-12和图2-13的问题：

a. 在PRODUCT表中，Product_Line_Name字段的长度是多少？为什么？

b. 在图2-13b中，PRODUCT表中的Product_ID字段为什么必须被指明？

c. 在图2-13b中，为什么Product_ID与产品表的一个键相邻？

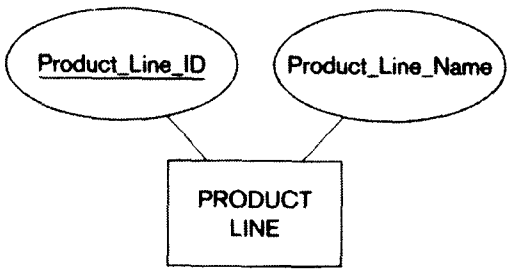
d. 图2-13a中，SQL语句的最后一行有什么作用？

17. 考虑图2-15中的Access查询。

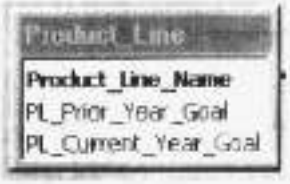
a. 为什么在这个查询中必须包括Order_Placement_Date这个字段，即使它不显示在查询结果中？

b. 如果Helen Jarvis想看到全部产品系列的结果，而不仅仅是家庭办公室产品系列的结果，必须怎样修改查询？

18. 如果观察第3章中的图3-22，你将看到松谷家具公司数据库的部分实体-联系图。PRODUCT LINE实体的图如下：



Chris为Helen准备的草图中关于PRODUCT LINE实体的部分如下：



解释PRODUCT LINE的两种不同定义间的差别。

应用练习

1. 会见不同组织中的系统分析员与数据库分析员，请他们描述他们的系统开发过程。这个过程属于系统开发生命周期法还是原型法呢？他们运用了类似于这种两种方法的方法吗？他们何时运用不同的方法？探索开发一个通过网络使用的应用的方法。他们如何改编方法以适合这个新的系统开发过程？

2. 选择一个你熟悉的组织，它可能是你工作的单位、你的学校或者你朋友工作的单位。针对这个系统，描述它的信息系统结构（ISA）。这个系统有一个正式准备过的体系结构吗，或者你必须创建一个新的体系结构？会见这一组织中的信息系统管理员以找出他们为什么正式认可ISA。

3. 选择一个你熟悉的组织，它可能是你工作的单位、你的学校或者你朋友工作的单位。考虑这个组织中的主要数据库，例如一个支持客户交互的数据库，会计数据库或者生产数据库。这个数据库的体系结构如何？这个组织运用了某种形式的客户/服务器体系结构吗？咨询这个信息系统的管理员以找出他们为这个数据库选择此结构的原因。

4. 选择一个你熟悉的组织，它可能是你工作的单位、你的学校或者你朋友工作的单位。咨询系统分析员和数据库分析员并询问有关信息系统开发团队的典型构成。具体而言，一个数据库分析员在项目组中扮演什么角色？一个数据库分析员服务于整个系统开发过程还是仅仅服务于被选择的点上呢？

5. 选择一个你熟悉的组织，它可能是你工作的单位、你的学校或者你朋友工作的单位。咨询系统分析员和数据库分析员并询问有关该组织在系统开发过程中如何运用CASE工具的问题。把问题集中于CASE工具如何应用于数据建模与数据库设计和CASE工具的信息库如何维护收集的数据、数据特性和数据用法的信息。如果有多个CASE工具应用于一个或多个项目，考察这个组织如何整合数据模型和数据定义。最后，询问系统分析员和数据库分析员对这个支持数据建模和数据库设计的CASE工具的满意度如何。

参考文献

Hoffer, J. A. J. F. George, and J. S. Valacich. 2002. *Modern Systems Analysis and Design*. 3rd ed. Upper Saddle River, NJ: Prentice Hall.

Moriarty, T. 1991. "Framing Your System." *Database Programming & Design* 4 (June): 38-43.

Sowa, J. F., and J. A. Zachman. 1992. "Extending and Formalizing the Framework for Information Systems Architecture." *IBM Systems Journal* 31 (3): 590-616.

Thompson, C. 1997. "Committing to Three-Tier Architecture". *Database Programming & Design* 10 (August): 26-30, 32, 33.

Zachman, J. A. 1987. "A Framework for Information Systems Architecture." *IBM Systems Journal* 26 (March): 276-92.

进一步阅读

Finkelstein, C. 1989. *An Introduction to Information Engineering*. Reading, MA: Addison-Wesley.

Shank, M. E., A. C. Boynton, and R. W. Zmud. 1985. "Critical Success Factor Analysis as a Methodology for IS Planning." *MIS Quarterly* 9 (June): 121-29.

Web资源

- http://netmation.com/docs/bb10.htm 讨论企业范围数据建模的Netmation上的短篇论文。
- http://www.usdoj.gov/jmd/irm/lifecycle/table.htm The Department of Justice Systems Development Life Cycle Guidance Document。这是你可以参考的系统方法论的一个例子。
- http://www-4.ibm.com/software/developer/library/clplatform.html?dwzone=collaboration 由Sean Gallagher作的关于"building a prototyping platform with open-source data stores"的论文。
- http://www.qucis.queensu.ca/Software-Engineering/ USENET新闻组comp.softwareeng的软件工程文档。该站点包括了许多你想浏览的链接。
- http://osiris.sunderland.ac.uk/sst/case2/welcome.html 英国Sunderland大学的计算机辅助软件工程网站。
- http://www.sei.cmu.edu/str/descriptions/clientserver_body.html Client/Server（客户/服务器）软件体系结构——综述。卡耐基·梅隆软件工程学院。
- http://www.acinet.org/acinet/ 美国职业信息网，包含关于职业、展望、需求等的信息。

项目案例：山景社区医院

项目描述

在第1章已介绍过山景社区医院。图2-17表示2001年1月1日山景社区医院的组织图表。像大多数普通医院一样，山景社区医院分为两个主要的组织团体。由Dr. Browne领导的医生组负责病人医疗的质量。Ms. Baker（管理者）协助医生向病人提供护理、疗养以及临床管理方面的服务。

(1) 目标与主要成功因素

正如第1章所述，山景社区医院的基本目标是为其周围社区提供高质量的医疗服务，同时在近几年全国各行业成本呈增长趋势的前提下维持成本不变。山景社区医院为一个人口约有50 000、年增长率为10%的社区服务，由于周边地区吸引了越来越多的退休者，所以这个增长的趋势还将继续下去。因此山景社区医院有一个目标是扩展它的容量（5年内增加50个床位）并成立一个退休人员医疗中心，该中心具有独立的房间和生活设施。目前，在医院的周围有充足的土地可供医院扩建。由于这次扩建，山景社区医院计划增加管理者的人数和医疗人员的人数，成立一个新的由Ms. Baker负责的退休人员生活部和一个新的由Dr. Browne负责的老年内科部。

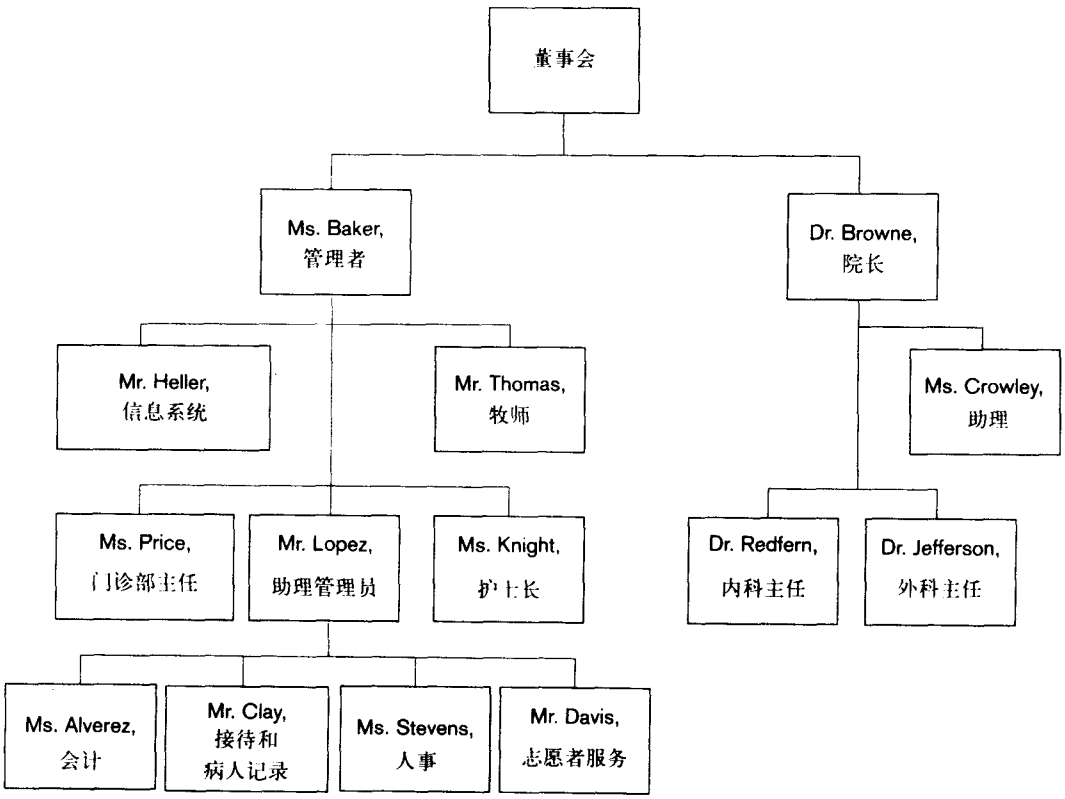


图2-17　2001年1月的组织图

为了适应山景社区医院的稳定成长和扩建，一个由Mr. Heller、Mr. Lopez、Dr. Jefferson和一名顾问组成的特别研究小组已经开发了一个长期的业务规划，包括医院的信息系统规划。他们的工作尚未完成，但是他们已经确定了实现这个规划必须的许多元素。为了满足高质量的医疗服务、保持成本和增加新的服务等目标，特别研究小组总结出医院有4个关键成功因素：医

疗质量、控制运作成本、控制资本成本、招募和留住专业人才（尤其是护士）。改进的信息系统的开发应该有能力处理以上每一个关键成功因素。

小组当前的工作是为每个关键成功因素确定2～4个短期或长期目标。到目前为止，他们为控制运作成本这个关键成功因素开发了以下四个目标：

1) 减少采购成本。

2) 更有效地安排员工。

3) 降低责任保险成本。

4) 扩大志愿者服务。

研究小组更详细地描述了每一个关键成功因素和目标，并将这些描述存入一个信息库，这个信息库由信息系统部门使用的CASE工具来管理。

(2) 企业建模

研究小组还开发了一个业务功能的初步清单，其中描述了医院内的管理活动和医疗活动。这些功能考虑了组织目标和前一小节解释的关键成功因素。此时，研究小组确定了5个主要的业务功能，它们涉及所有的组织单元。这5个业务功能有：

- *病人医疗管理* 管理后勤和病人的医疗记录。
- *临床服务* 提供实验室测试和程序，以及病人监控和显示。
- *病人医疗服务* 为病人提供医疗护理和支持服务。
- *财务管理* 管理财务资源和医院的运作。
- *管理性服务* 提供并非与病人护理直接相关的一般的管理和支持服务。

研究小组把每个高层次的功能分解为更详细的功能（参见图2-18），但是他们知道此时这个清单既不完整，也不是很好。

研究小组最初有一个具有10个实体类型的初步集合，这个集合描述了医院在运作和管理中所需的数据。它们是FACILITY（设施）、PHYSICIAN（医生）、PATIENT（病人）、WARD（病房）、STAFF（职员）、LABORATORY（实验室）、TEST（化验）、MEDICAL/SURGICAL ITEM（内科/外科项目）、SUPPLY ITEM（供应品）和VENDOR（供应商）。与医院员工进行讨论并检查医院的文档和研究现存的信息系统后，研究小组开发了业务规则的一个清单，它描述了医院的政策和管理这些实体间的联系的医运运作的本质。其中一些规则是：

1) 一个FACILITY维护一组LABORATORY：放射

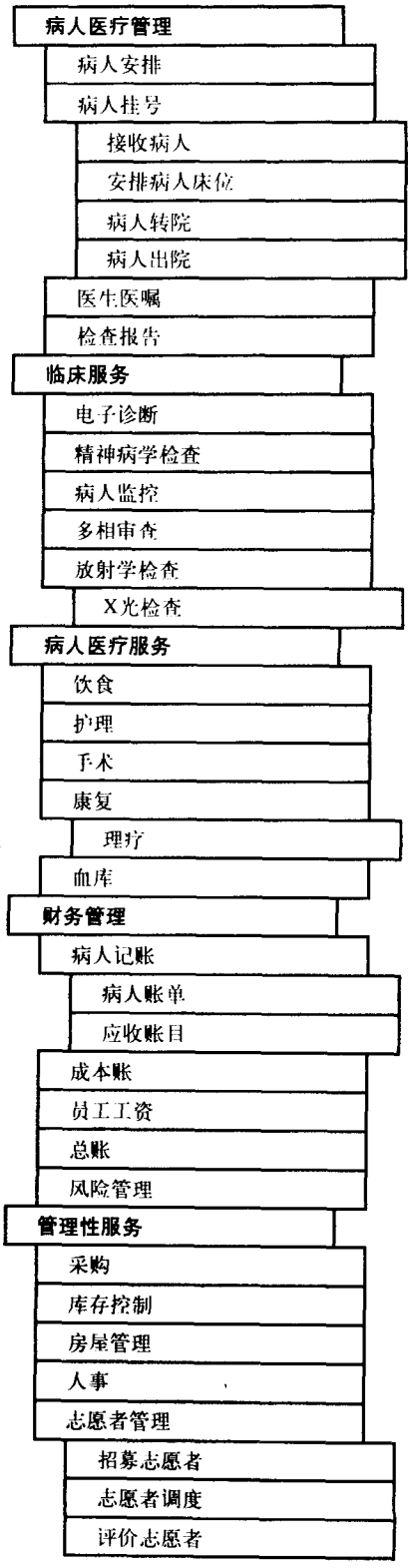


图2-18 业务功能

科、电子诊断、血液学等等。

2) 一个FACILITY包括一组WARD（产房、急诊室、康复室、老年病科等等）。

3) 给每个WARD分配一定数量的STAFF（护士、秘书等等），一个STAFF可能属于多个WARD。

4) 一个FACILITY配备一组PHYSICIAN的医疗小组。一个PHYSICIAN可能为多个FACILITY工作。

5) 一个PHYSICIAN治疗许多PATIENT，一个PATIENT可以被许多的PHYSICIAN治疗。

6) 一个PHYSICIAN诊断许多PATIENT，一个PATIENT可以被许多的PHYSICIAN诊断。

7) 一个PHYSICIAN可能被分配到一个WARD（门诊病人没有被分配WARD）。医院仅关心病人当前的WARD（如果有病房）。

8) 一个PATIENT使用MEDICAL/SURGICAL ITEM，它们由多个VENDOR供应。VENDOR也提供用于内务处理和维护用途的SUPPLY ITEM。

9) 一个LABORATORY对多个PATIENT进行多种TEST。

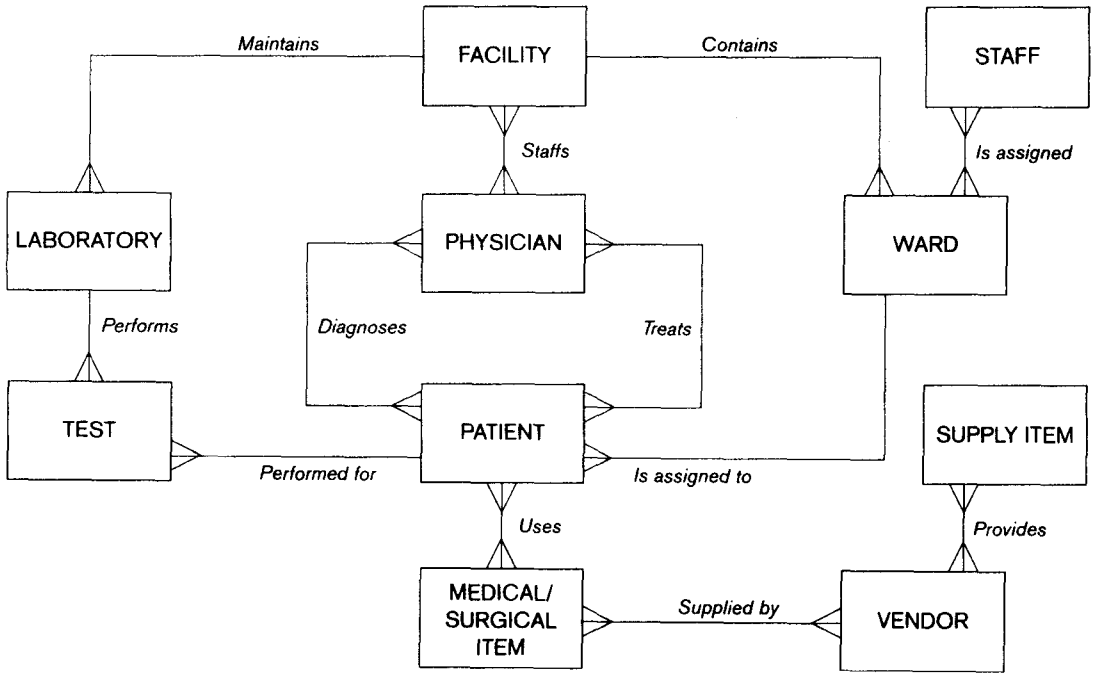


图2-19　初步的企业数据模型

他们认识到某些业务功能，如风险管理和志愿服务，无法用一组数据实体和业务规则表示清楚，但是他们决定以后再处理这些问题和其他问题。研究小组把这些数据实体和业务规则的描述存储到CASE工具的信息库中供以后分析。使用确定的实体和业务规则，研究小组开发了一个初步的企业数据模型（参见图2-19）。因为这个企业数据模型的目的仅仅是给出组织数据的一个概述，它不遵循信息系统部门绘制数据模型使用的所有约定，所以这个数据模型是初步的。

(3) 开发规划矩阵

研究小组使用CASE工具来产生功能-实体类型矩阵的第一个版本（参见图2-20）。这个矩阵将业务功能映射到图2-18中的10个数据实体上。矩阵中每一个单元格编码的含义如下：

M = 功能维护实体的实例（创建、更新和删除）

R = 功能使用关于实体的数据

数据实体类型 / 业务功能	设备	医生	病人	病房	职员	实验室	化验	内科/外科项目	供应品	供应商
病人安排		R	R			R				
病人挂号			M	R						
医生医嘱		R	R				R			
检查报告			R			R	R			
电子诊断							M			
精神病学检查							M			
病人监控			R							
多相审查			R			R	M			
放射学检查						R	M			
饮食			R							
护理			R							
手术		R	R							
康复			R							
血库			R			R				
病人记账			R	R		R	R			
成本账	R		R	R			R			
员工工资		R		R	R					
总账	M			M		M				
风险管理	R	R	R					R		R
采购								M	M	M
库存控制								R	R	
房屋管理									R	
人事		M			M				R	
志愿者管理	R		R	R		R				

M = 数据实体（列）由业务功能（行）维护

R = 数据实体（列）由业务功能（行）使用

图2-20 业务功能-数据实体矩阵

作为矩阵的最初情况，研究小组决定不单独编码数据创建、更新和删除这些活动。

研究小组还使用CASE工具将业务功能与四个关键成功因素相关联（图2-21为工作的结果）。在图2-21中，单元格的含义是：

E = 做好此项功能是实现关键成功因素的基础

D = 做好此项功能对实现关键成功因素有帮助

关键成功因素（CSF） / 业务功能	质量管理	运作成本控制	资本成本控制	员工招募与保留
病人安排	D			
病人挂号				
医生医嘱	D			
检查报告	D			
电子诊断			D	
精神病学检查	D			
病人监控	E			
多相审查	E			
放射学检查	D			
饮食	E			
护理	E	E		E
手术	D	D	E	
康复	E			
血库	D			
病人记账	D	E		
成本账		E		
员工工资				D
总账				
风险管理		E		D
采购		E		
库存管理		E		
房屋管理		D		
人事				E
志愿者管理		E		E

E = 业务功能（行）在实现CSF（列）中是必要的

D = 业务功能（行）在实现CSF（列）中是令人满意的

图2-21　业务功能-关键成功因素矩阵

研究小组正在寻找途径结合图2-20和图2-21中显示的结果，以帮助他们为信息系统开发活动设定优先级。

项目问题

1）哪个信息系统规划矩阵（在这个案例研究中显示的和未显示的矩阵）可能帮助山景社区

医院的研究小组为数据库确定一个层次化位置规划？为什么？

2) 在上面的项目描述中，列出了控制运行成本的目标，控制运行成本是四个关键成功因素之一。根据给定的医院的结构和规划，为其他的关键成功因素列出可能的目标。

3) 在“志愿服务”业务功能中出现了哪些另外的活动？

4) 哪些活动可能出现在“风险管理”业务功能中？

5) 在这个案例中，除了提到的10个实体外，风险管理功能可能需要哪些附加的数据实体？

6) 你认为研究小组通过他们已经完成的分析能够确定所有的数据实体吗？如果不行，还要进行哪些其他的数据库开发步骤？何时做？

7) 你认为业务规划研究小组的成员是依据什么选出的？你会选择不同的人或其他的人吗？

8) 在上面的案例描述中，列出了9个业务规则，研究小组用这些规则开发了图2-19。除了这9个规则外，图2-19还描述或暗示了哪些其他的业务规则？

9) 尽管资本成本控制是一个关键成功因素，但是在图2-21中仅有两个业务功能提到了这个关键成功因素，为什么？需要做什么工作来增强这张图以使得更多的实体表示出业务功能与这个关键成功因素间的联系？

项目练习

1) 重新安排图2-20的行与列，以使其中的项目按对角线排列。分析这个模式，你能得到什么结论？

2) 重新绘制图2-17来反映医院所计划的未来扩建与增长后的医院结构。根据这一扩建和增长计划，图2-18、图2-19和图2-20需要做哪些变动？

3) 以图2-20和图2-21为基础，判断哪些数据实体对于运作成本控制是至关重要的。对于信息系统的开发来说，这一结论暗示着什么？

4) 在项目问题5中，列出了风险管理业务功能中所需要的附加数据实体。更改图2-19以包括这些附加实体和所有数据实体之间的有关联系。

5) 来自“给病人开账单”业务功能的一个重要输出之一是病人的账单。在第1章的项目练习2中介绍了账单的一种简化版本，下面又给出了这个账单。第1章的项目练习2请你加上那些必须出现在账单上的且漏掉的数据。利用你在那个练习中得到的结论，验证图2-19中的企业数据模型包含了产生一个病人账单所必需的数据。解释为完成验证工作你必须做些什么？从分析中你能发现些什么？

Patient Name: Dolan, Mark

Patient Number: _____

Patient Address: _____

Item Code	Item Description	Amount
_____	_____	_____
_____	_____	_____
_____	_____	_____

6) 案例研究中所描述的研究小组的活动与早期的信息系统和数据库的开发有关。请概述接下来要采取的步骤以使得信息系统中目前的系统与数据库和将来医院所需要的信息系统有机联系起来。

7) 风险管理领域的管理者希望利用计算机来支持他的活动。医院面临越来越多的有关玩忽职守的投诉与诉讼，而管理者不相信他必须等到信息系统与数据库计划实施后才能改善信息服务。管理者需要的是一个追踪投诉、法律诉讼、律师、法官、医院员工、投诉赔偿和判决的系统。你将如何设法解决这个改善信息服务的要求？你会运用什么方法来设计他需要的系统和数据库？为什么？

8) 重新考虑项目练习7中风险管理的管理者的要求。他所需要的系统与数据库应该在哪一层上进行开发？为什么？

第二部分　数据库分析

概要

数据库开发的第一步是数据库分析，其任务是确认用户对数据的需求，并开发数据模型以表示用户的需求。概念数据模型是从用户角度描述数据的模型，与实现模型所使用的技术无关。第二部分深入讲述了概念数据建模的主流方法——实体-联系模型。

业务规则是组织中有关业务处理的原则和方法的统称。数据库开发人员根据业务规则进行概念数据建模。在第3章的开始，介绍通过描述业务规则进行建模的观点，说明良好的业务规则应该符合的特征并讨论总结业务规则的过程。在业务规则的讨论中，会涉及数据模型中数据元素命名和定义的一般原则。然后，该章介绍实体-联系模型（E-R模型）的主要特性，包括该模型中实体、联系和属性的表示方法和这些建模技术的主要结构。对于每种特定的结构，该章相应地给出命名和定义模型中元素的原则。该章会讲述强、弱两种实体类型的区别和如何确定联系类型。该章讲述不同类型的属性，如简单属性与复合属性，单值属性与多值属性，导出属性和标识符属性等。该章还讲述联系类型和实例的区别与联系，并介绍关联实体的概念。而且还讲述一元、二元和三元等不同度数的联系。该章还介绍出现在各种建模情况下联系的基数，特别地，该章讨论为与时间有关的数据建模的一般方法。最后，该章描述实体之间具有多种联系的情况。在该章中，以松谷家具公司为例来讲述E-R建模的概念。

在第4章中，讲述实体-联系建模的高级概念，并介绍最新的E-R模型表示法。利用这些表示法能够找到更复杂的业务规则，通常需要额外的建模特性，以应付当今组织遇到的日益复杂的业务环境。

在增强型实体-联系（EER）图中，最重要的新引入的建模结构是超类型/子类型联系。这种新结构允许创建一种通用的实体类型（称为超类型），并将超类型划分成几种特殊的子类型。例如，客车和轿车是汽车的子类型。该章介绍一种简单的表示方法来表示超类型/子类型之间的联系，并介绍概化和特化这两种方法以标识超类型/子类型联系。进一步地，该章还介绍进一步提炼基本的超类型/子类型联系的表示法，应为联系编写良好的说明文档以确保其可理解性，这是非常重要的。该章介绍一种称为实体聚簇的技术，该技术能够简化E-R图的表示以满足各种类型E-R图使用者的需要。

业务规则是定义或约束业务的某些方面的语句。本章对常见的业务规则进行分类，然后介绍各种业务规则的声明方法，并引入同时可用于EER图的业务规则的表示方法。

使用面向对象技术开发的系统中所使用的是另外一种数据建模的表示方法——UML类图。这些内容将在第五部分的第14章进行讲解。如果对UML和E-R图比较感兴趣，可在学完第4章后直接学习第14章。

在第二部分的两章中讲述概念数据建模，为数据库分析和设计打下坚实的基础。对一个数据库分析员来说，应该使用E-R表示法对用户数据和信息的需求进行建模。

第3章　组织中的数据建模

3.1 学习目标

学完本章后，读者应该具备以下能力：

- 简明定义以下列关键术语：业务规则、术语、事实、实体-联系模型（E-R模型）、实体-联系图（E-R图）、实体、实体类型、实体实例、强实体类型、弱实体类型、标识属主、标识联系、属性、复合属性、简单属性、多值属性、导出属性、标识符、复合标识符、联系类型、联系实例、关联实体、联系的度、一元联系、二元联系、三元联系、基数约束、最小/最大基数和时间戳。
- 说明为什么系统大多数开发人员相信数据建模是系统开发过程中最重要的部分。
- 规范地命名和定义实体、联系和属性。
- 区别一元、二元和三元联系，并分别给出一个常见的例子。
- 建立E-R图中以下结构的模型：复合属性、多值属性、导出属性、关联实体、标识联系，以及最小/最大基数约束。
- 绘制表示常见业务情况的E-R图。
- 将一个多对多的联系转换为关联实体类型。
- 在E-R图中应用时间戳建模简单的与时间相关的数据。

3.2 引言

本书前两章通过简单的例子介绍了数据建模和E-R模型。业务规则是描述业务活动的有力工具，本章将基于业务规则的概念形式化数据模型，并进一步详细描述E-R模型。

业务规则来源于政策、过程、事件、功能和其他业务对象，以及在组织中声明的约束。业务规则对于数据建模来说是十分重要的，它们决定了如何处理和存储数据。基本的业务规则就是数据名和数据定义。为了在业务中准确地命名和定义数据对象，本章给出一些基本的指导原则。根据数据的概念模型的要求，必须命名和定义实体类型、属性和联系。其他的业务规则可以在这些数据对象上声明约束。通过对数据模型进行研究，可以发现这些约束，例如，在实体-联系图和相应的文档中都可以发现这些约束。

在应用多年之后，E-R模型仍然是概念数据建模的主流方法。它得到广泛使用的原因如下：使用起来相对容易，具有广泛的CASE工具支持，以及人们认为实体和联系是现实世界中的自然建模概念。

E-R模型经常作为一种交流的工具，用来在数据库开发过程的分析阶段沟通数据库设计者和终端用户（详见第2章）。E-R模型建立了一种概念数据模型，它表示的是独立于具体软件（如数据库管理系统）的数据库结构和约束，以及相关的数据模型。数据库的实现将基于这些模型。

当讨论E-R建模时，一些作者会介绍关系数据模型所特有的术语和概念。特别地，他们建议通过完全解析主键和外键来实现模型的完全规范化。但是，在现有的关系数据模型理论中，这种完全的规范化方法还很不成熟。在目前的数据库环境中，数据库可以用面向对象技术和面

向对象以及对象关系技术的混合来实现。因此，我们将规范化的概念推迟至第5章再讨论。

实体-联系模型是在CHEN所著的一篇关键性的文章（1976）引入的。在文章中，他描述了实体-联系模型的主要结构——实体和联系——以及相关的属性。后来，CHEN和其他人又通过增加额外的结构对这个模型进行扩展，见Teorey、Yang和Fry(1986)以及Storey(1991)的论文。E-R模型继续发展，但由于种种原因，至今E-R模型的表示方法还没有进行标准化。Song、Evans和Park（1995）的著作对10种E-R建模表示方法进行了两两对比，并解释每种方法体系的主要优点和缺点。

许多系统开发人员认为数据建模是系统开发过程中最重要的环节，这种认识来源于以下三个重要原因（Hoffer、George和Valacich，2002）：

1) 在设计数据库、程序以及其他系统组成部分的过程中，通过数据建模发现数据的特征是极为关键的。同时，在信息系统中通过数据建模，还可以发现一些业务中的事实和规则，而这些事实和规则对于保证信息系统的数据完整性是非常重要的。

2) 在许多现代的信息系统中，数据比过程更复杂，所以在构建系统需求时，数据自然而然地成为核心因素。通常，研究数据结构的目的在于提供丰富的数据源，以支持系统中的各类查询、分析和汇总。

3) 与使用数据的业务过程相比，数据更加稳定。因此，基于面向数据的信息系统设计开发比基于面向过程的信息系统具有更长的使用寿命。

本章通过常见的符号和规定，描述E-R建模的主要特征。本章首先给出E-R模型中的基本结构——实体、属性和联系——的定义；随后定义E-R建模中常见的三种类型的实体：强实体、弱实体和关联实体；同时还定义多种重要的属性，包括单值/多值属性，导出属性和复合属性等。然后，本章介绍与联系有关的三个重要概念：联系的度、联系的基数和联系中的参与约束。最后，本章给出松谷家具公司的E-R图的扩展示例。

3.3 根据组织中的规则建立数据模型

在本章和下一章中，将说明如何用数据模型（特别是实体-联系符号）表示组织中的规则与制度。*事实上，数据建模的全部作用就是研究使用数据的组织的规则和制度。*在信息处理和存储系统中，业务规则和制度控制着数据的创建、更新和移除，因此，在描述数据的同时必须描述与之相关的业务规则。例如，有这样一条政策规定“每个在校大学生必须有一位导师”，这条政策规定强制性地将数据库中记录的每个大学生的数据与某个指导教师的数据关联起来。再看下面这条语句：“大学生是这样一类人，他们的入学申请被接受，或者已经接受大学中的某些课程的教育或培训，无论这些课程或培训记入学分与否”，这句话不但定义“大学生”这个概念，而且还说明大学的一项规定（这项规定隐含表示：校友是大学生；而如果认为大学教育展览不是学校所开展的培训项目，那么参加大学教育展览但入学申请没有被接受的中学生不属于大学生的范畴）。

业务规则和制度不是通用的。不同的大学会用不同的制度指导学生，对学生的定义也有所不同。一个组织的规则和制度可能随着时间发生变化（一般比较缓慢）。例如，学校会规定在学生没有选择专业以前不需要指导教师。

数据库分析员的任务如下：

- 确认并理解管理数据的规则。
- 对规则进行描述，使信息系统开发人员和用户可以准确地理解这些规则。
- 用数据库技术实现这些规则。

数据建模在这个过程中起着重要的作用。数据建模的目的是将与数据有关的业务规则记录下来，本章在简要叙述一般的业务规则的基础上，讨论数据建模和实体-联系模型的表示方法。数据模型不可能表示所有的业务规则（也没必要全部表示，因为并非所有的业务规则都要管理数据），数据模型应该与相关的文档和其他信息系统模型（例如，记录数据处理的模型）相结合，才能完整地描述信息系统的业务规则。

3.3.1 业务规则概述

业务规则（business rule）是“一种定义或约束业务中的某些有关方面的语句，主要说明业务的结构以及控制或影响业务的行为……防止、导致或引发某些事件发生的规则等”（GUIDE Business Rules Project，1997）。例如，以下两条语句就是业务规则的常见表示形式，它们描述影响数据处理和存储的业务规则：

- “仅当学生成功完成某门课程的预备课程后，才可以注册某个班级选修该课程”。
- “顾客如果结清余款，可享受10%的折扣”。

在今天的大多数组织中，存在着无数条这样的规则。这些规则在总体上影响着组织的行为方式，并决定着组织对外界环境的反应方式（Gottesdiener，1997）。在组织中发掘和记录业务规则是一项重要而复杂的工作。首先必须对业务规则进行全面整理，然后应用数据库技术在信息系统中进行实现，才能确保信息系统正常工作，以及用户对其输入和看到的信息有正确的理解。

业务规则范型

- 业务规则的概念已经在信息系统中广泛应用，但是，在提到类似的规则时，“完整性约束”这个术语更为常见（特别是在数据库中）。然而，这个术语的使用范围是有限制的，它一般用来指维护数据库中数据值和联系的合法性。

术语“业务规则”具有更广的使用范围，它包括所有组织中对数据库有影响的规则（如上两个例子所示）。事实上，一些研究人员建议用一种新的范型来表示业务规则，这样可以更准确地分析信息系统的需求（von Halle，1997）。这种方法基于以下前提：

- 因为业务规则表示的是企业中各种制度以及这些制度如何指导个人和集体的行为，所以业务规则是企业的核心概念。具有良好结构的业务规则可以采用自然语言的形式展示给最终用户，或采用数据模型的形式展示给系统开发人员。
- 业务规则应用最终用户所熟悉的形式进行表达，这样，最终用户也可以定义并维护他们自己的业务规则。
- 业务规则应该具有良好的可维护性。业务规则应该存储在中心信息库中，每条规则只能表达一次，并可在整个组织中共享。
- 业务规则可以通过软件来自动应用，这些软件可以解释这些规则，并利用数据库管理系统的完整性机制强制应用这些规则。

尽管已经取得很多进步，但业界完全实现上述目标尚需时日。进一步的研究和工具的开发仍在继续（特别是在业务规则的自动执行方面），而且研究人员对业务规则方法寄予厚望。这种方法最大的潜在优势是“业务规则具有极好的可维护性”。业务规则在两方面具有很强的能力，其一是能够将信息系统中的需求作为业务规则的集合进行细化和维护，其二是能够从一个业务规则的信息库中自动生成信息系统。系统的自动生成和维护不但可以简化系统开发过程，而且可以提高所开发系统的质量。

3.3.2 确定业务规则的范围

在本章和下一章中，本书只关注那些对组织的数据库产生影响的业务规则。大多数组织有许多与数据库无关的规则或制度。例如，“员工在周五可以随便着装”是一条很重要的规定，

但它对数据库没有直接的影响。另一方面，“学生只有成功学完某门课程的预备课程，才可以申请注册某个班级选修该课程”这条规则属于本书所考虑的范围，因为该规则在数据库的事务处理中起约束作用。如果试图注册一个没有完成预备课程的学生，在该规则的作用下，将拒绝执行这项事务。一些业务规则不能由一般的数据模型符号表示。自然语言也可以作为一种表示形式，它可以描述这些不能用各种实体-联系图表示的业务规则，其中一些业务规则也可以用关系数据模型表示，在第5章中将详细描述这些问题。

1. 良好的业务规则

无论是用自然语言表述，还是用结构化数据模型表示，或者用其他信息系统文档表示，如果规则符合如上所述的前提条件，则业务规则将具有一些共同的特征。在表3-1中总结了这些特征。如果业务规则由业务人员（而不是技术人员）定义、批准和管理，那么上述这些特征更容易满足。业务人员会主动监督业务规则的执行。数据库分析员的职责就是帮助找到规则，并且将表述有误的规则转化为满足所需特征的规则。

表3-1 良好业务规则的特征

特 征	解 释
声明性	一个业务规则应该是制度内容的语句，而不是描述制度如何执行或监督制度；规则不是在描述过程或实现，而是在描述过程所验证的内容
准确性	规则的含义应该清晰，在某一组织内，其解释是惟一的
原子性	一项业务规则只包含一项陈述，而不是包含几项陈述；业务规则的任何一部分都不能作为一个新的、完整的规则（即规则是不可分的，但却是充分的）
一致性	业务规则应该具有内部的一致性，即不包含相互冲突的语句；并且和其他规则保持一致性
可表述性	业务规则必须能够用自然语言描述，但是业务规则应使用结构化的自然语言描述，以便不会产生歧义
可区分	业务规则不包含冗余信息，但某个业务规则可引用其他的规则（特别是引用定义）
面向业务	业务规则应使用业务人员能够理解的术语来表述。由于业务规则是业务制度的陈述，所以仅有业务人员才能修改或废除一项规则。这就是说，业务规则是属于业务的

注：改编自Gottesdiener（1999）和Plotkin（1999）。

作为数据库分析员，应对不合规范的业务规则进行修改，以满足上述特征。

2. 收集业务规则

业务规则根据组织的职能、事件、制度、单位、风险承担者和其他业务对象来描述。为了确认这些业务规则，应该参与有关的信息系统需求收集讨论会，阅读组织的文档（如人员手册、公司制度、合同、营销手册和技术指导等）以及接触其他信息资源，最后整理成业务规则。常用的确定业务规则的方法是提出并回答这样六个问题：何人（Who）、何事（What）、何时（When）、何地（Where）以及如何操作（How）。业务规则的最初陈述一般是模糊的、不准确的，有人称之为业务泛谈，数据分析员必须坚持澄清关于业务规则的这些初始陈述。通过反复的咨询和整理，形成简洁明确的业务规则。在本章及后面几章中，我们会引入以下问题供数据分析员进行业务规则的检查，这些问题可适应各种数据建模的情况。这些问题包括：“这种情况是否总是存在”，“是否存在导致其他事件发生的特殊环境”，“是否存在不同于那个人的人”，“是仅有一个还是有多个”，“是需要保持历史记录，还是当前数据足以说明问题”。

3.3.3 数据命名与定义

命名和定义数据对象是理解数据结构以及进行数据建模的基础。数据对象必须首先命名和定义，然后才能意义明确地用于组织的数据模型。在实体-联系模型的表示方法中，必须明确命名和定义每个实体、联系和属性。

1. 数据命名

在开发实体-联系数据模型时，我们会特别强调实体、联系和属性的命名，但是要理解命名数据对象的一些常用原则。

- 与业务相关，与具体的软硬件技术无关。例如，Customer（顾客）是一个好的数据对象名，但File10、Bit7或PayrollReportSortKey不是好的名字。
- 命名应具有实际意义。命名本身也是数据对象说明文档的一部分，即命名精炼地体现出数据对象的本质。所以，命名应尽量避免使用过于一般化的词汇，如“has”、“is”、“person”或者“it”。
- 命名必须是惟一的，以区别于其他数据对象。在数据名中应包含能够与其他类似数据对象相区别的词汇（如HomeAddress（家庭地址）和CampusAddress（校园地址））。
- 可读性强。命名的结构应和概念的自然表达方式相一致（如GradePointAverage（班级平均分）是一个好的数据名，而AverageGradeRelativeToA（与A班级有关的平均分）这个名字虽然精确，但却显得过于冗长）。
- 由一个得到认可的词汇表中的词汇组成。每个组织通常选择一些含义恰当的词汇来命名数据，如最好使用maximum（最大值），而不是upper limit（上限）、ceiling（最高限度）或highest（最高）等词汇。另外，数据的别名也应包含在数据库的文档中。用作数据名的词汇也可以是缩略形式，如用CUST表示Customer。在数据名长度受限制时，尽量使用缩略语。
- 可重用。不同的人或者同一个人在不同的时间可能会不约而同地使用同样或相似的数据名，这说明存在一种被人们认可的命名习惯，在命名数据时应尽量遵守这种习惯。如学生的生日被命名为StudentBirthDate，而员工的生日被命名为EmployeeBirthDate等。

Salin（1990）建议采取以下命名方法。

1) 给出一个数据的大体上的定义（定义将在随后讨论）。

2) 去掉不重要和不合法的词汇（大家认为不适合作为名字的词）。注意，在定义中出现AND和OR意味着其中可能包括两个或多个数据对象。如果可以的话，应该分离这些数据对象，并给每个对象分配不同的名称。

3) 将词汇按照含义明确、可重用的形式组织。

4) 给出每个词汇的标准缩略形式。

5) 确认数据名是否已经存在，如果是，则需加上其他的标识性词汇，以确保数据命名的惟一性。

在本章的数据库开发实例中所使用的数据名都可以作为好的数据命名的范例。

2. 数据定义

定义（有时称为结构化断言）可以看做是一种业务规则（GUIDE Business Rules Project, 1997），定义是术语或事实的解释。**术语**（term）是在业务中有特定含义的词或短语，如课程、班级、租用车辆、航班、预约以及乘客等。术语经常可作为构成数据名的关键字。术语的定义必须谨慎、简练。一些常见的名词，如“day”、“month”、“person”或者“television”，因为不会产生歧义，所以不必定义为术语。

事实（fact）是两个或多个术语之间的关联。以下是定义为事实的例子（已定义的术语用下划线表示）。

- “课程是某个特定领域的一个讲授单元”。这个定义将术语“特定领域”和“讲授单元”联系在一起。假设这些术语是很常见的，不需要进一步定义。

- “顾客可能在特定的日期在租车行租赁一辆特定款式的轿车”。这个事实定义了租赁请求模型，其中将四个术语（以下划线表示）关联在一起（GUIDE Business Rules Project, 1997）。其中三个术语都与特定的业务有关，应该单独定义（日期是个常见词汇）。

3. 良好的数据定义

在本章和下一章的实体-联系表示法设计中，读者可以看到良好的定义方式，包括实体、联系和属性等。下面给出几条一般性的原则（Aranow, 1989）。

- 所有的数据定义和各类业务规则都来源于信息系统的需求分析。在信息系统需求分析的研究过程中，系统和数据分析员应寻找数据对象并给出定义。
- 定义一般伴随图形说明，如实体-联系图。定义不必重复已在图中显示的内容，而是对图形进行补充。定义应就以下方面描述数据对象特征：
- 细微之处内涵清晰。
- 特殊和例外条件。
- 实例。
- 何处、何时以及如何在组织中创建或计算数据。
- 数据是静态的，还是可以随时间变化的。
- 数据的原子形式是单数还是复数。
- 谁确定数据值。
- 谁拥有数据，即谁控制数据的定义和使用。
- 数据是否可选，以及是否允许空值（即我们所说的null值）。
- 该数据是否可以分解为若干个原子部分，或它是否常与其他数据合并成更复杂的复合形式或聚合形式。

如果在数据定义中没有包括以上所有特征，则这些特征应和其他元数据存储在同一文档中。

- 数据对象必须先命名和定义，并检查是否与其他数据对象没有冲突，确认后才能加入至数据模型，如实体-联系模型中。但是，数据模型的设计过程也是一个对数据含义理解不断深入的过程，是一个迭代的过程，所以一旦将数据对象置入实体-联系图中，就希望定义不再发生改动。

在数据建模中有一种说法，它体现了良好的数据定义的重要性：“掌握数据的含义，就掌握了数据”。在一个组织中，同时定义所有的术语和事实看起来是一件比较容易的事。但事实并非如此，事实上这可能是数据建模中最困难的问题。在一个组织中，对一些常见术语下多个定义的情况是不常见的，如“customer”和“order”等术语均只有一个明确的定义。

为说明在设计定义时所遇到的问题，考虑大学中的Student（学生）数据对象。Student的定义为“已入校并在去年至少选修一门课程的人。”这个定义所涵盖的内容显然太狭窄。作为一个学生，与学校之间的联系经历如下阶段。

1) 填报志愿：通过一些正式的手续确定感兴趣的学校。

2) 申请人：申请入学。

3) 被大学接受的申请人：获准入学，学生接受的教育可以是学位教育，也可以是其他方式的教育。

4) 被录取的学生：至少注册一门课程。

5) 继续学业：注册参加其他课程的学习（没有实质性区别）。

6) 留级生：在规定时间没有注册有关的课程，重新申请。

7) 毕业生：完成所有的学习过程，可以申请学位，也可以重新申请新的学位。

可以设想，在这种情况下，给出一个得到所有人一致认可的定义很困难的。有一些可供选择的定义方法：

1) 对不同的情况给出不同的定义。但是，若只有一个实体类型，则尽量不要使用这种方法，因为多个定义并不是好的定义，而且会产生较大的歧义。可以针对上述每一种学生的情况创建多种实体类型，但这样这些实体之间的区别就会很小，容易产生混淆，同时数据模型中的结构也将会很复杂。

2) 使用一个尽可能通用的定义涵盖大部分情况。使用这种方法时，必须在每种具体情况下，添加说明特别情况的数据。例如，为了定义学生，必须增加与学生情况有关的数据，如报考人、申请人等。但是，如果一个学生同时出现多种状态，如可以在学习一门专业时报考另一门专业，则这种方法也不适用。

3) 使用多个有联系的数据对象来表示Student。例如，可以创建一个Student的一般实体类型，然后在每种情况下加入相应的特征以表示特定的学生实体类型。在第4章中将介绍这种方法。

3.4 E-R模型

实体-联系模型（E-R模型）是组织或业务领域中数据的详细的、合乎逻辑的表示方法。E-R模型的表述基于业务环境中的实体，实体之间的联系（或关联）以及实体与联系的属性（或性质）。E-R模型通常以**实体-联系图**（E-R图）的形式描述，E-R图是E-R模型的图形化表示。

3.4.1 E-R图示例

图3-1是一个简单的E-R图，这个例子有助于初步理解E-R图。该图表示一个小型的家具制造公司——松谷家具公司。这个公司从不同的供货商处采购标准件，这些供货商将这些标准件运送到公司。这些标准件组装成产品，再销售给已经订货的顾客。在每个订单上会包括一至数行关于所订购产品的说明。

图3-1中的E-R图显示有关这个公司的实体和联系（为简化该图，隐去了属性）。实体用矩形框表示，实体间的联系用矩形框之间的菱形表示，并通过线与有关的实体相连接。图3-1中的实体有：

- CUSTOMER（顾客）：订购或可能订购产品的组织或个人。例如，L.L.Fish 家具公司。
- PRODUCT（产品）：由松谷家具公司制造的多个品种的家具，以供顾客订购。在此，产品类型用产品编号表示，而不是具体的产品。例如，一个6英尺高的5层橡木书架的产品编号是O600，我们称之为产品O600。
- ORDER（订购）：这个事务将一或多个产品的销售与顾客相关联，并在销售记录或会计账目中用一个事务号表示。例如，在2001年9月10日，L.L.Fish 购买一个O600产品和四个O623产品的事件就是一个ORDER。
- ITEM(标准件)：用于组装产品的某种部零件。它可由一或多个供货商提供。同样使用编号来表示部零件。例如，4英寸的球形轴承轮脚的编号是I-27-4375。
- SUPPLIER（供货商）：给松谷家具公司提供标准件的其他公司。例如，Sure Fasteners有限公司。
- SHIPMENT（装运）：装运是一项与松谷家具公司接收标准件有关的事务。来自一个供货商的一次装运的所有标准件都出现在一张送货单上。例如，接收从Sure Fasteners有限公司的300件I-27-4375标准件和200件I-27-4380。

注意，必须准确地定义作为元数据的每个实体。例如，了解到CUSTOMER实体（无论是

组织还是个人）还没有从松谷家具公司购买产品是一条重要的信息。一个组织中的不同部门对于同一术语（同义词）有不同解释是很正常见的。例如，财务部可能只定义那些曾经购买过产品的组织和个人为顾客，但这一定义并没有包括那些潜在的顾客。营销部对顾客的定义如下：任何与他们有联系的人，已经从公司购买过产品的人或从公司的竞争者那里购买过产品的人。一个精确、完整的E-R图必须清晰地定义元数据，否则，该图在不同的人看来可能会产生不同的理解。

在E-R图中，每一条线末端的符号表示联系的基数。参见图3-1，可以看出这些基数符号表示以下业务规则。

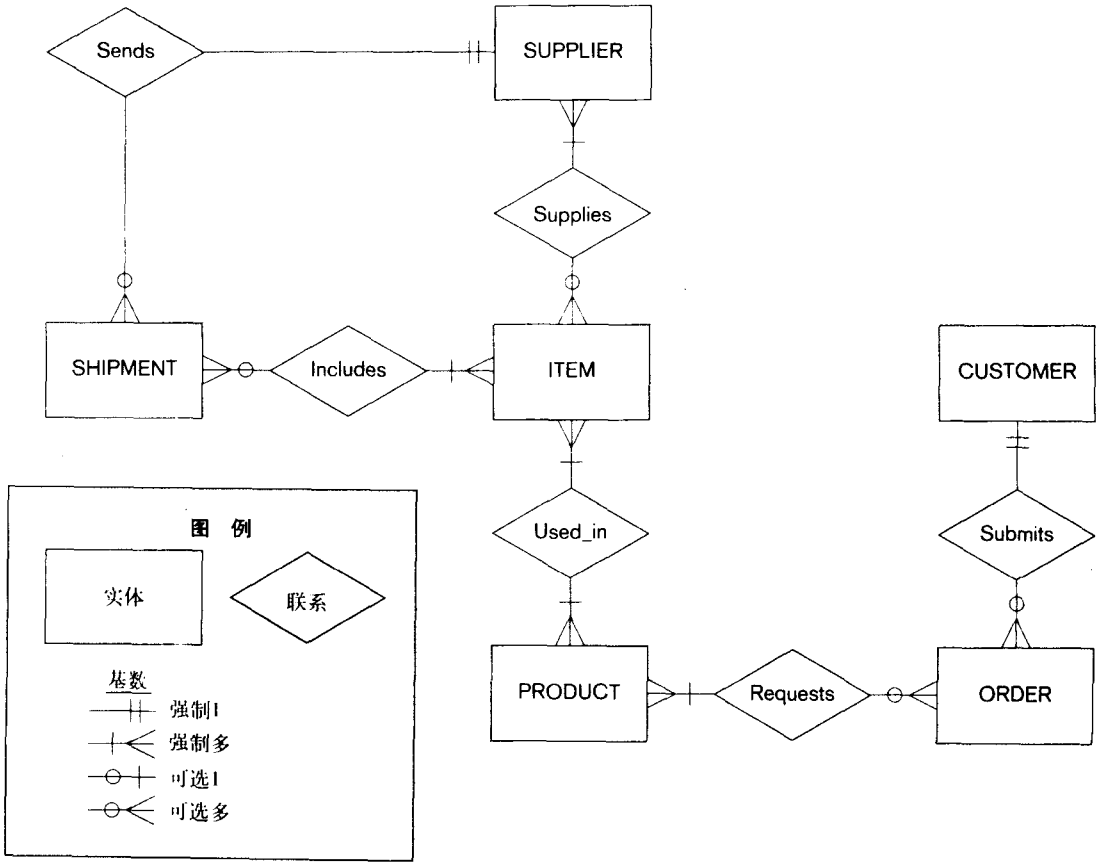


图3-1 E-R模型示例

1) SUPPLIER可能提供多个ITEM（“可能提供”意味着有可能一个也不提供）。每个ITEM都由一些SUPPLIER提供（“提供”意味着ITEM至少出自一家供货商）。

2) 每个ITEM必须用于至少一种PRODUCT的组装，也可以用于多种产品的组装。相反，每种PRODUCT必须使用一至多个ITEM进行组装。

3) 一个SUPPLIER可以发出多次SHIPMENT。另一方面，每个送货单必须由且仅由一个SUPPLIER发出。注意，发货和供货是两个概念：一个SUPPLIER可能能够提供零部件，但可能还没有发出关于零部件的送货单。

4) 一个SHIPMENT必须包含一至多个ITEM。一个ITEM可能包含在几个SHIPMENT中。

5) 一个CUSTOMER可能提交一至多份ORDER。但是，每一份ORDER必须由且仅由一个

CUSTOMER提交。

6) 一个ORDER 必须包括一至多个PRODUCT。某种PRODUCT可能没有ORDER来订购，但也可能会被一至多个ORDER订购。

注意，每一个业务规则大致遵循以下语法:

<实体><最小基数><联系><最大基数><实体>

例如，规则5可表示如下:

<CUSTOMER><可能><提交><任何数字><ORDER>

该语法给出一个将每个联系放在用自然英语描述的业务规则中的标准方法。

3.4.2 E-R模型符号

在图3-2的E-R图中显示了本书使用的基本符号。正如在前一节所指出的，在业界还不存在统一的标准符号（实际上，在第1章和第2章看到的符号稍微进行了简化）。但是，图3-2中的符号已经包括E-R图的常用符号体系的大多数通用特征，并且能够在大多数实际情况下精确地表示模型。在第4章中将介绍用来加强实体-联系模型的一些其他符号（包括类-子类联系等）。

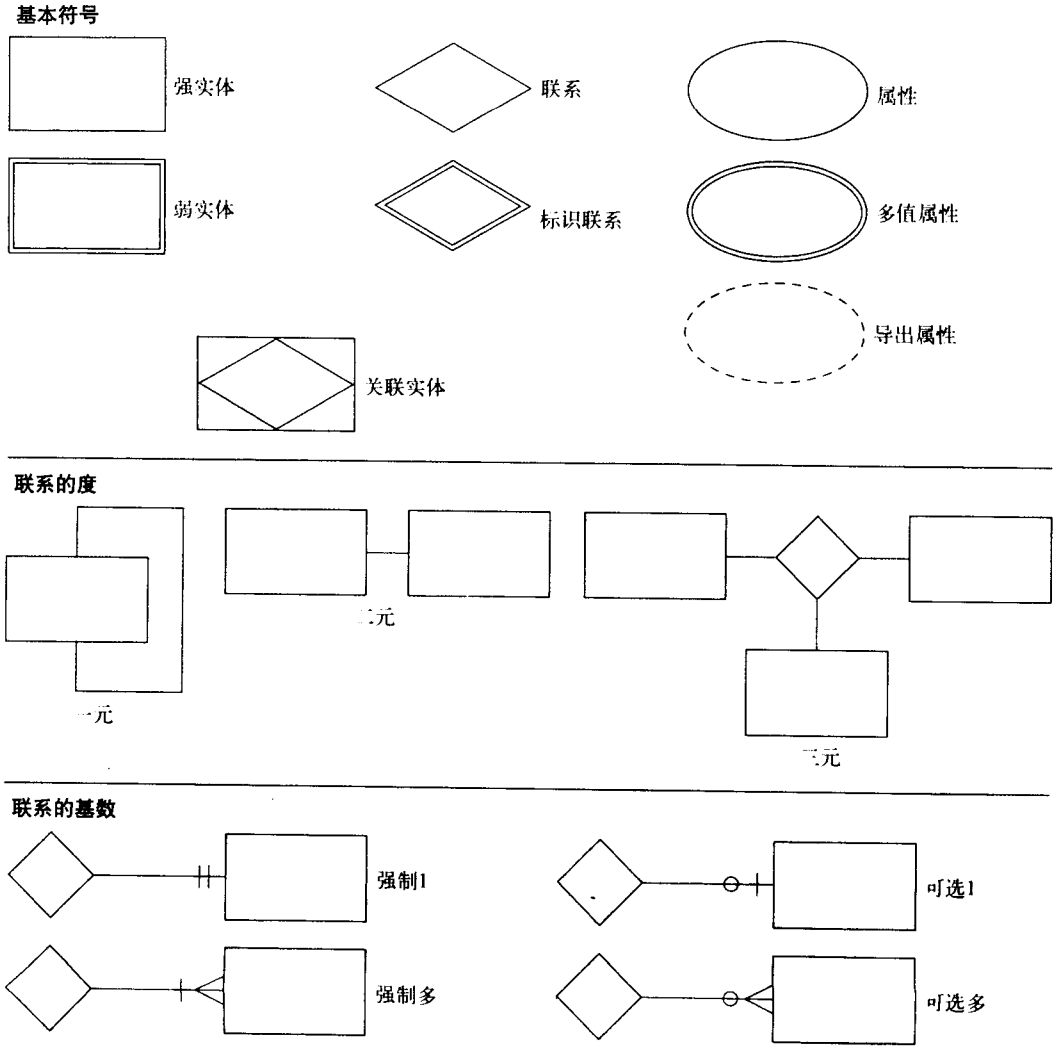


图3-2　基本E-R符号

附录A描述在许多图形工具中使用的E-R符号。这个附录可以帮助读者在书中的符号和实际使用符号之间进行互相转换。

3.5 实体-联系模型的结构

实体-联系模型的基本结构由实体、联系和属性所组成。如图3-2所示，模型允许对每种结构进行扩展。E-R模型的丰富特性使设计者可以对真实世界的各种情况建立准确和充分的模型，这也是该模型广泛使用的原因。

3.5.1 实体

实体（entity）表示组织所期望维护的数据，是指存在于用户环境中的人、地点、对象、事件或概念。下面是各种类型的实体。

人（Person）：EMPLOYEE（员工）、STUDENT（学生）、PATIENT（病人）

地方（Place）：STORE（商店）、WAREHOUSE（仓库）、STATE（州）

对象（Object）：MACHINE（机器）、BUILDING（建筑）、AUTOMOBILE（汽车）

事件（Event）：SALE（销售）、REGISTRATION（注册）、RENEWAL（更新）

概念（Concept）：ACCOUNT（账目）、COURSE（课程）、WORK CENTER（加工中心）

1. 实体类型与实体实例

实体类型与实体实例的区别是十分明显的。**实体类型**（entity type）是具有共同性质和特征的实体集合。E-R模型中的每个实体类型都有一个名字。由于名字表示的是一个特定的集合，所以名字总是单数的。实体类型的名字通常使用大写字母。在E-R图中，实体名字放在代表实体类型的矩形框中（见图3-1）。

实体实例（entity instance）是实体类型集合中的一个特定记录。图3-3描述一个实体类型与其两个实例之间的区别。在数据库中，一个实体类型仅（使用元数据）描述一次，但实体类型的实例则由数据库中的数据表示。例如，在大多数组织中都存在EMPLOYEE这样一种实体类型，但数据库中却存储着数百（甚至数千）个该实体类型的实例。在不引起歧义的情况下，本书通常应用术语“实体”而不是实体类型。

实体类型:EMPLOYEE	
属性:	
EMPLOYEE NUMBER	CHAR (10)
NAME	CHAR (25)
ADDRESS	CHAR (30)
CITY	CHAR (20)
STATE	CHAR (2)
ZIP	CHAR (9)
DATE HIRED	DATE
BIRTHDATE	DATE
EMPLOYEE的两个实例:	
642-17-8360	534-10-1971
Michelle Brady	David Johnson
100 Pacific Avenue	450 Redwood Drive
San Francisco	Redwood City
CA	CA
98173	97142
03-21-1992	08-16-1994
06-19-1968	09-04-1975

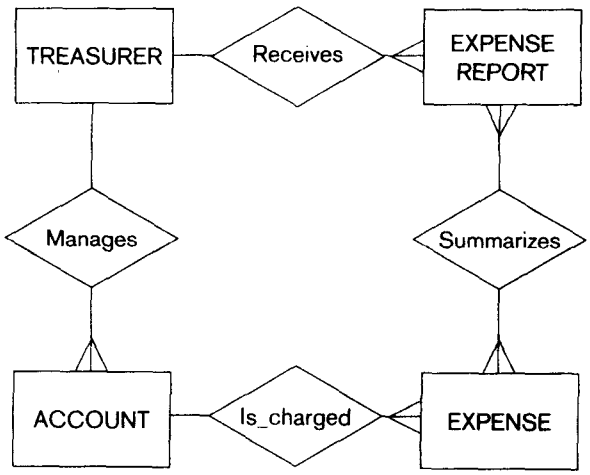
图3-3 实体类型（EMPLOYEE）与其两个实例

2. 实体类型与系统输入、输出或用户

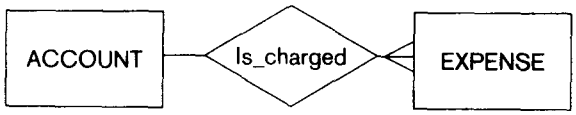
当读者开始学习绘制E-R图时，特别是如果对数据处理模型（如数据流图）已经很清楚的前提下，可能会犯一个常见的错误，那就是将数据实体和整个信息系统模型中的其他元素相混淆。有一个简单的规则可以避免产生这种混淆，即一个真正的数据实体有很多实例，但每个实例都具有一个可以区分的特征，以及一个或多个其他的描述性数据。

如图3-4a所示，该图描述一个存储女生联谊会开支的数据库系统。在这种情况下，女生联谊会的出纳管理账务，接收开支报告，并且记录每一笔账务的开支事项。但是，是否必须记录有关Treasurer（TREASURER实体类型）的每一个数据和她对账务的每一次管理（Manages联系）以及每一次接收报告的详细过程（Receives联系）？Treasurer输入有关账务和开支的数据，并且接收开支报告。也就是说，她是一个数据库的用户。既然只有一个出纳，那么TREASURER数据就不必存储。更进一步，EXPENSE REPORT实体有必要吗？既然开支报告能够由开支事项和账户余额计算出，那么它可由数据库中的数据导出。即使出纳在不同时间收到很多份开支报告，每份报告中的数据都可以由数据库中已有的ACCOUNT和EXPENSE实体类型计算出。

联系Receives和Summariazes名称的本质是另一个有助于我们理解为什么图3-4a中E-R图出错的关键之处。这些联系名指的是转移或转换数据的业务活动，并不仅仅是一种数据到另一种数据的关联。如图3-4b所示，这个简单的E-R图中实体和联系已经足以表示上述女生联谊会的开支系统。



a) 系统用户（Treasurer）和输出（Expense Report）被表示为实体



b) 仅包括必要实体的E-R模型

图3-4　不适当的实体的例子

3. 强实体类型与弱实体类型

大多数在组织中定义的基本实体类型都被归为强实体类型。**强实体类型**（strong entity type）是不依赖于其他实体而独立存在的实体类型。其例子有STUDENT、EMPLOYEE、AUTOMOBILE和COURSE。在强实体类型的一个实例中，总可以找出一个惟一的特征（称为

标识符），也就是区别于该类型的其他实例的一个属性或属性集合。

相比较而言，**弱实体类型**（weak entity type）是依赖于其他实体类型而存在的实体类型。在E-R图中，弱实体类型如果没有它所依赖的实体，则它是没有意义的。被弱实体类型所依赖的实体类型称为**标识属主**（identifying owner）（或简称为**属主**）。一个弱实体类型没有属于它自己的标识符。通常在E-R图中，一个弱实体类型有一个属性，作为一个部分（partial）标识符。到设计的后续阶段（在第5章中描述），通过将其属主的标识符和部分标识符相绑定，会形成一个弱实体类型的完整标识符。

图3-5描述了一个具有标识联系的弱实体类型的例子。EMPLOYEE是一个强实体类型，其标识符为Employee_ID（我们在图中以下划线来表示标识符属性）。DEPENDENT（家属）是一个弱实体类型，在图中用双边框的矩形表示。弱实体类型与其属主之间的联系称为**标识联系**（identifying relationship）。在图3-5中，“Has”是一个标识联系（在图中用双边框的菱形表示）。属性Dependent_Name是一个部分标识符（正如在后面所描述的，Dependent_Name是一个可以分解成更小的组成部分的复合属性）。本书使用双下划线来表示部分标识符。在后面的设计中，Dependent_Name会结合Employee_ID（属主的标识符）形成DEPENDENT的完整的标识符。

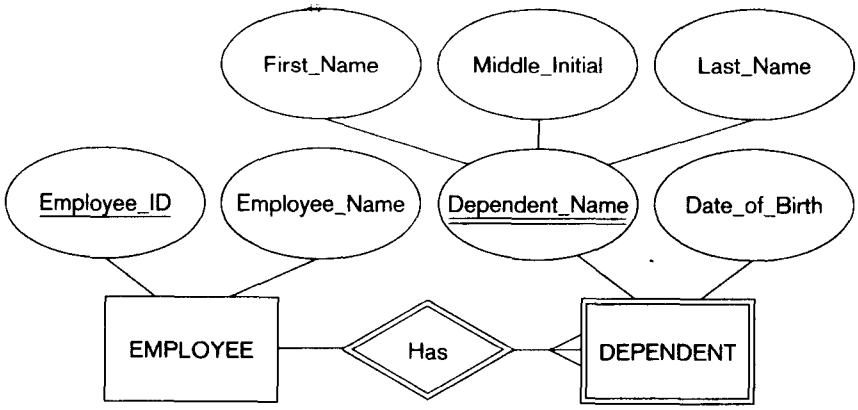


图3-5 弱实体类型与其标识联系的实例

4. 命名和定义实体类型

除了命名和定义数据对象的一般性原则以外，还有一些特殊的命名实体类型的原则如下：

- 实体类型名是一个单数名词（例如CUSTOMER、STUDENT或AUTOMOBILE）；实体是人、地方、对象、事件或概念，即可使用这些名字命名实体类型。实体类型代表实体实例的集合（例如，STUDENT可以代表学生Hank Finley、Jean Krebs等等）。用复数名词来表示实体类型也是常见的（可能在一些附带E-R图的CASE工具信息库中会遇到）。因为有时E-R中的元素最好被解读为复数。例如，在图3-1中，一个SUPPLIER可能提供多个ITEM。注意，英语的复数并不总是由单数名词加s 组成，所以应该使用正确的复数形式。
- 一个实体类型名的应用范围应限于特定的组织。一个组织可以用CUSTOMER作为实体类型名，而另一个组织可以使用术语CLIENT。实体类型的名字对于这个组织来说应该是易于描述和理解的，并且易于与组织中的其他实体类型名相区分。例如，一个和供货商相联系的PURCHASE ORDER和一个与顾客有关的CUSTOMER ORDER很容易区分。但这两种实体类型不能被命名为ORDER。

- 实体类型名应该简洁，使用的单词应尽可能少。例如，在一个大学的数据库中，以REGISTRATION作为学生注册班级这个实体类型的名字就足够了。而短语STUDENT REGISTRATION FOR CLASS尽管十分精确，但如果读者能够理解REGISTRATION的明确含义，那么用这个短语就显得有些啰嗦。
- 实体类型名称应该使用简短的名字或缩略语。在E-R图时中，使用缩略语就足够了。缩略语的命名规则应该和实体的命名规则相同。
- 事件实体类型应该根据事件的结果来命名，而不是根据事件的活动或过程来命名。例如，在一个项目经理安排一个员工去完成一个项目的事件中，会产生ASSIGNMENT（安排）结果；同样，一个学生去联系他的指导教师查一些资料，这个事件产生一个结果CONTACT（联系）。
- 在所有E-R图中，同样的实体类型应具有相同的名字。对于特定的组织，用于表示实体类型的名字也应符合相应标准，组织采用这个标准引用所有同一类型的数据。但是，一些实体类型具有别名，或可替换的名字，它可以作为同义词用于组织中的其他部分。例如，实体类型ITEM可以有一个别名MATREIAL（在生产中）和DRAWING(在设计中)。别名在数据库的相关文件中定义，例如在CASE工具的信息库中指定。

还有一些用于定义实体类型的特殊准则如下：

- 实体类型定义通常以“X是……”开头。这是描述实体类型含义的最直接和最明确的方法。
- 实体类型定义应该包括这样的语句：哪个特征惟一标识了实体类型实例。在许多情况下，声明实体类型的标识符有助于说明实体的含义。例如，在图3-4b中，“支出就是用于购买某些商品或服务的款项。支出由分类账中的账目号惟一确定”。
- 实体类型定义应该清晰地表达该实体类型包含哪些实体实例，以及不包含哪些实体实例。通常，有必要列出所有需要排除的实例种类。例如，“顾客是一个已经下订单购买产品的个人或组织，或者是组织已经联系并利用广告或促销手段向其推广产品的个人和组织。顾客不包括那些通过我们的顾客、分销商和代理机构购买产品的个人或组织。”
- 实体类型定义通常包括这样一个描述：什么时候创建实体类型实例，什么时候删除实体类型实例。例如，在上文中，当个人或组织发提交他的第一份订单时，就隐式地创建一个顾客实例。因为定义中没有说明除此之外会怎样，所以说明顾客实例不会被删除，而只能根据数据库中清理数据的一般规则来删除。关于何时删除实体实例有时也被称为是实体类型的保留周期。如一个删除客户实体类型定义的语句可以是“如果一个顾客超过三年未提交订单，则他不再是顾客”。
- 对于一些实体类型，定义必须指出什么时候一个实例会变为另一个实体类型的实例。例如，考虑一个建筑公司，其投标用的标书可能被潜在的顾客接受而成为合同。在这种情况下，标书可以定义为“公司制定的要为顾客工作的合法的文件，当公司的主管在投标文件上签字后生成标书，在收到由顾客的主管签字的标书副本时，标书转化为一个合同的实例”。这个例子也较好地说明某个定义如何使用其他实体类型名（在上述情况下，标书的定义用到了实体类型名Customer）。
- 对于一些实体类型，定义必须说明应保存实体类型实例哪些方面的历史记录。例如，图3-1中ITEM的特征可能会随着时间发生变化，并且公司应该保留一份有关单价和单价生效时间的完整历史记录。在以后的例子中会看到，这种关于保存历史记录的语句可能会衍生出以下问题：如何在E-R图上表示与实体类型，以及最终如何存储该实体实例的数据。

3.5.2 属性

每一个实体类型都有一组与其相关的属性。**属性**（attribute）是组织所关心的实体类型的一个性质或特征。下面是一些典型的实体类型和与之相联系的属性。

- STUDENT: Student_ID（学生的ID）、Student_Name（学生姓名）、Home_Address（家庭住址）、Phone_Number（电话号码）、Major（专业）。
- AUTOMOBILE: Vehicle_ID（汽车ID）、Color（颜色）、Weight（重量）、Horsepower（马力）。
- EMPLOYEE: Employee_ID（员工ID）、Employee_Name（员工姓名）、Payroll_Address（工资单地址）、Skill（技能）。

在命名属性时，应使用首字母大写、其他字母小写的形式。如果属性名包括两个单词，就使用“_“来连接两个单词，两个单词的首字母都大写。例如，Employee_Name。在E-R图中，使用椭圆来表示属性，属性名写在椭圆中间，椭圆用一条线与其关联实体连接起来。属性也可与联系连接，下面将介绍这种情况。注意，属性必须准确地与相关的实体或联系连接。

注意，在图3-5中，DEPENDENT的所有属性都是员工家属的特征，而不是员工的特征。一个实体类型（无论是弱实体类型还是其他实体类型）不能包括与之相关的实体的属性（称之为外键）。例如，DEPENDENT没有包含该家属与哪个员工相关联的属性，即E-R数据模型中具有属性非冗余的特征。在数据库中，有些数据可以被多个实体共享。这两者是不矛盾的。因为在数据库中，是通过实体类型之间的属性联系实现数据访问的（例如，在一个屏幕上显示Dependent_Name和相应的Employee_Name）。

每个实体（或实体类型的实例）在每个属性上都有一个相应的值。例如，图3-6显示拥有各自属性值的两个实体。从本质上说，一个数据库就是赋予实体的所有属性值的集合。实体可以看作是由一个标识符属性和一个或多个其他属性组合而成的。如果尝试创建只有一个标识符的实体，那么这个实体可能不合法。这样一个数据结构可能只保存一些属性的合法值的列表，这些列表最好保存在数据库之外。

实体1

Student_ID = 638124289
Student_Name = Allison Smith
Home_Address = 482 Walnut St.
Denver CO 86391
Phone = 303-839-1479
Major = Psychology

实体2

Student_ID = 726153972
Student_Name = Cedric Thomas
Home_Address = 944 Maple St.
Houston, TX 51269
Phone = 631-347-9847
Major = English

图3-6 两个实体实例与其属性值

1. 简单属性与复合属性

一些属性可以分解为粒度更小的有明确含义的组成部分。最常见的例子是Address，它可以

分解成如下一些部分：Street_Address（街区）、City（城市）、State（州）和Postal_Code（邮政编码）。**复合属性**（composite address）就是能分解为更小部分的属性（如Address）。图3-7显示在此例中用于表示复合属性的符号。在E-R图中，子属性可能会在复合属性的上方或下方出现。

复合属性为用户提供了相当大的灵活性，用户既可以把复合属性当作一个完整的单元来应用，也可以应用复合属性中的单独部分。例如，一个用户既可以引用Address，也可以引用Address的一部分，如Street_Address。是否将属性分解为更小的部分取决于用户是否需要使用那些单独的属性。当然，设计者必须明确地预计到未来数据库的使用模式。

简单（或原子）属性（simple attribute）是不能分解为更小部分的属性。例如，所有与AUTOMOBILE相联系的属性都是简单属性：Vehicle_ID、Color、Weight和Horsepower。

2. 单值属性与多值属性

图3-6显示两个实体与其各自的属性值。对于每个实体实例，图中的每一个属性都有一个值。经常会有这样的情况，即给定实体实例的一个属性会有一个或多个值。例如，图3-8中EMPLOYEE实体类型有一个名为Skill的属性，其值记录员工的技能。当然，某些员工会具备多种技能（例如，同时是COBOL程序员和C++程序员）。**多值属性**（multivalued attribute）就是在给定实体实例中可以有多个值的属性。E-R图中用双边椭圆来表示多值属性，如图3-8所示，在EMPLOYEE的例子中，Skill属性就是多值属性。

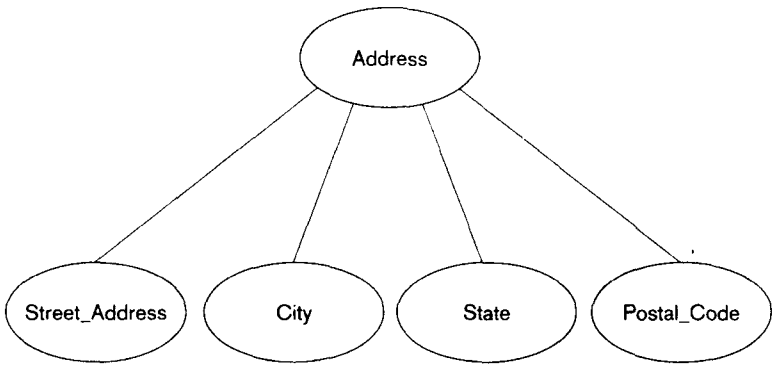


图3-7　复合属性

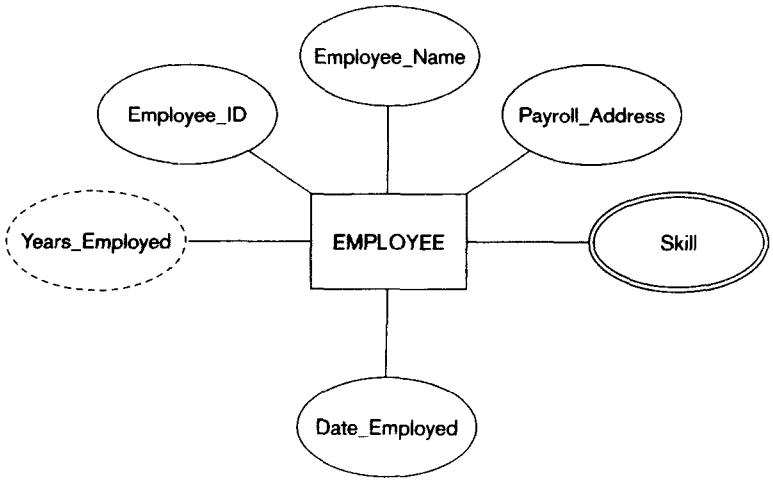


图3-8　具有多值属性（Skill）和导出属性（Years_Employed）的实体

3. 存储属性与导出属性

对于用户来说，他们所感兴趣的一些属性值可以由数据库中现有的相关属性值计算或导出。例如，假设对于一个组织来说，EMPLOYEE实体类型有一个Date_Employed（雇佣日期）属性。如果用户想知道一个员工在此工作了多少年，可以用Date_Employed和当前日期计算出来。**导出属性**（derived attribute）就是一个其值可以由其他相关属性值计算出的属性（参与计算的其他数据也可能不在数据库中，如当前日期，当前时间，或者由系统用户提供的口令等）。在E-R图中使用边为虚线的椭圆表示导出属性，如图3-8所示。

在某些情况下，属性值不能由相关实体的属性导出。考虑松谷家具公司给每一位顾客出具的发票（见图1-6）。Order_Total（订单总价值）是INVOICE实体的属性，它表示顾客应付的总金额。Order_Total属性值可以由发票上列出的各项的Extended_Price属性值的总计得到。这样的数值计算公式也是一种业务规则。

4. 标识符属性

标识符（identifier）是可以惟一标识一个实体类型实例的属性或属性集。在前面介绍的STUDENT实体类型中，其标识符属性为Student_ID，而AUTOMOBILE实体类型的标识符属性为Vehicle_ID。注意，像Student_Name这样的属性不能作为备选的标识符属性，因为学生既有可能重名，也有可能改名。要成为候选的标识符，每一个实体实例必须有该属性的一个值，并且属性必须与实体关联。在E-R图中以下划线表示标识符属性，如图3-9a所示。

对很多实体类型来说，没有哪一个单独（原子）的属性能作为标识符，即没有一个属性能够惟一标识一个实体。但是，两个或更多属性的组合则可以作为标识符来标识实体。在这种情况下，可以用这些属性所组成的一个复合属性作为一个标识符，即**复合标识符**（composite identifier）。图3-9b显示具有复合标识符Flight_ID（航班号）的实体FLIGHT。Flight_ID由子属性Flight_Number和Date组成。对于惟一标识实体FLIGHT来说，这些属性缺一不可。在E-R图中，在复合属性（Flghit_ID）下加下划线表示标识符，而其子属性则没有下划线。

有时实体可能会有多个备选的标识符。在这种情况下，设计者必须选择其一作为标识符属性。Bruce(1992)建议使用以下标准来选择标识符。

1）将每一个实体类型实例的生存周期中其值都不变的属性作为标识符。例如，Employee_Name和Payroll_Address的组合并不是一个好的标识符，因为在EMPLOYEE实体类型中，Employee_Name和Payroll_Address的值在员工的雇佣期间很容易发生改变。

2）选择在每个实体类型的实例中，该属性值都有效且不为空的属性作为标识符。如果标识符是一个复合属性，（如图3-9中的Flight_ID），要确保标识符的每一部分都不为空。

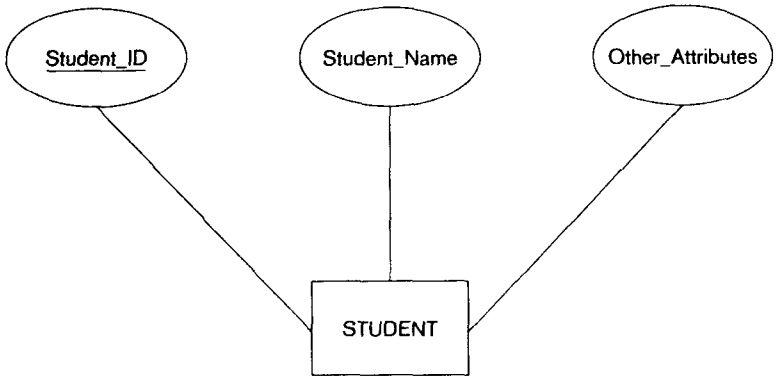
3）避免使用所谓的智能标识符（或键）。这种标识符的结构表示分类、位置等信息。例如，标识符的前两位可能表示仓库的位置。这种代码会随着条件的改变而变化，容易产生标识符非法的情况。

4）尽量使用单值属性作为标识符。例如，在实体类型GAME（比赛）中，应用Game_Number（比赛编号）属性作为标识符，而不用Home_Team（主队）和Visiting_Team（客队）的组合作为标识符。

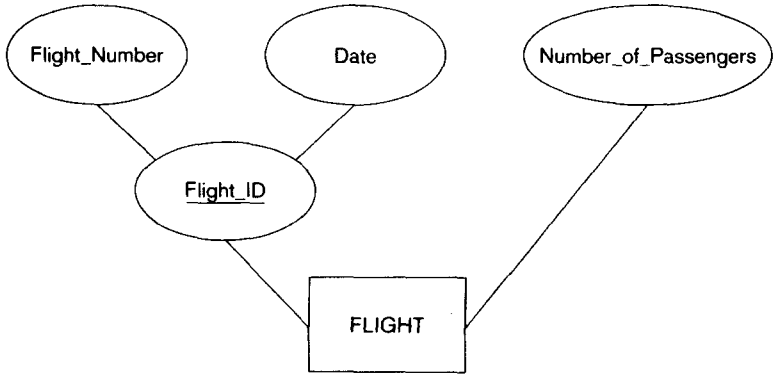
5. 命名与定义属性

除了命名数据对象的一般原则外，以下还给出一些专用于属性命名的原则。

- 属性名是一个名词（例如Customer_ID、Age、Product_Minimum_Price或者Major）。属性通过赋值拥有一个具体的含义，它描述实体的概念或物理特征。在客观世界中，概念或物理特征一般用名词来描述。



a) 简单标识符属性



b) 复合标识符属性

图3-9　简单标识符属性与复合标识符属性

- 属性名应该是惟一确定的。一个实体中不能有两个同名的属性。如果可能的话，在所有的实体中也不要出现同名的属性，以避免混淆。
- 为了使属性名具有惟一性并易于区分，在一个系统的建模中，所有的属性命名应遵守一个标准形式。例如，在大学数据库中，可能以Student_GPA（学生平均积分点）的形式作为一个属性的标准形式，而如果命名为GPA_of_Student就违反了这一标准。每个组织都应该建立这样的标准。常用的形式为：[实体类型名{[_限定符]}_]类别，其中[…]内为可选子句，{…}表明该子句可能重复。实体类型名（entity type name）是与该属性相关的实体类型的名字。实体类型名可以使属性名含义清楚，一般在实体类型的标识符属性（如Customer_ID）中都会出现实体类型名。类别是组织所定义的，描述实体的某类特征，通常为一个短语或其缩略形式。可以作为类别的短语及其缩略形式有：Name（Nm）、Identifier（ID）、Date（Dt）或Amount（Amt）等。很明显，类别是必需的。限定符是组织所定义的，是类别的约束条件。在某些情况下，在属性名中加入一个或多个限定符可以确保属性名的惟一性。例如，限定符可以是：Maximum（Max）、Hourly（Hrly）或State（St）。限定符不是必需的，没有限定符的属性名（如Employee_Age和Student_Major）的含义也同样是清楚的。但有时限定符是必需的。例如，Employee的两个属性名Employee_Birth_Date和Employee_Hire_Date必须使用限定符来加以区别。在有

些情况下，可能需要多个限定符。如属性Employee_Residence_City_Name（或缩略形式Emp_Res_Cty_Nm）表示员工居住的城市名，而属性Employee_Tax_City_Name（或其缩略形式Emp_Tax_Cty_Nm）则表示员工纳税的城市名。

- 在一个组织中，不同实体类型的相同属性应使用相同的限定符和类别。例如，表示教师和学生所在城市的属性名应该分别为Faculty_Residence_City_Name和Student_Residence_City_Name。使用类似的名字可以提示用户这些属性值的值域是相同的。用户可能会在查询中利用域相同这一条件，如查找与其导师住在同一城市的学生时就可以利用这一条件。在这种情况下，如果用户在两个属性名中采用同样的限定符和类别，则用户更加可以认定这一匹配是可行的。

对于属性的定义，也有如下一些专用的原则：

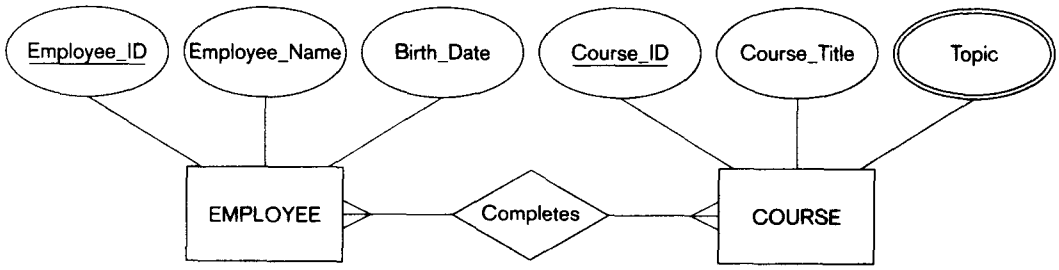
- 属性定义应说明属性的内容和属性之所以重要的原因。属性的定义常与属性的命名是相关的，如Student_Residence_City_Name的定义是“一个学生永久居住的城市名”。
- 属性定义应说明在属性值中所包括的具体内容和不属于属性值的内容。如“Employee_Monthly_Salary_Amount是在员工居住地所在国流通的、按月支付的工资数，其中不包括福利、奖金、退税或其他收入”。
- 属性的别名应在定义中声明，或在其他说明属性的文档中声明，并可以保存在CASE工具的信息库中来维护数据定义。
- 在某些情况下，属性的数值来源也是定义中的重要内容。声明数值来源可以使数据的含义更加明确。例如，“Customer_Standard_Industrial_Code表示企业所从事的业务类型。这个代码的值来自联邦贸易委员会制定的一个标准代码集，并且可以在联邦贸易委员会每年提供的名为SIC的CD中查到”。
- 属性定义（或CASE工具信息库中其他规格说明）也应指出属性值是必需的还是可选的。说明属性这项特征的业务规则对于维护数据完整性是很重要的。在定义中，必须指定实体类型的标识符属性。如果属性值是必需的，那么在创建实体类型的实例时，必须为该属性赋值。“必需”意味着在每个实例中该属性都有一个值，而不是在创建实例时需要值。“可选”则意味着在实体的某些实例中，该属性值可能不存在。可通过说明是否一旦要输入一个属性值，则该值必须总是存在的以便进一步限定可选的值。如定义“Employee_Department_ID是员工所在部门的标识符。当员工被雇佣时，可能还没有分配到一个部门中（这就是该属性可选的原因）。一旦员工被分配到一个具体的部门，该属性值就应是表示该部门的值”。
- 属性定义（或CASE工具信息库中其他规格说明）也需要指出在实例生存期间，一旦赋值，属性值是否是可变的。这条业务规则同样控制着数据完整性。非智能标识符的值不会随时间而改变。若要给一个实体实例分配一个新的非智能标识符，则该实例应先被删除或重新创建。
- 多值属性的定义中应指出，在一个实体实例中属性值的最大和最小个数。例如，“Employee_Skill_Name是一个员工所具备的技能名称。每个员工至少必须具备一种技能，并且每个员工最多可列出他所具备的10种技能。”采用多值属性的原因可能是因为要保存属性的历史记录。例如，“Employee_Yearly_Absent_Days_Number是一年中员工缺勤的次数。如果员工在工作日的工作时间少于规定时间的50%，则认为是缺勤。在该员工为本公司工作的每一年中，都需要保存此属性的值”。
- 属性定义也应指出该属性与其他属性之间的关系。例如，“Employee_Vacation_Days_

Number是员工的带薪休假的天数。如果某个员工的Employee_Type属性的值为‘Exempt’，则Employee_Vacation_Days_Number的属性值需根据该员工的工作年限通过一个公式计算得出”。

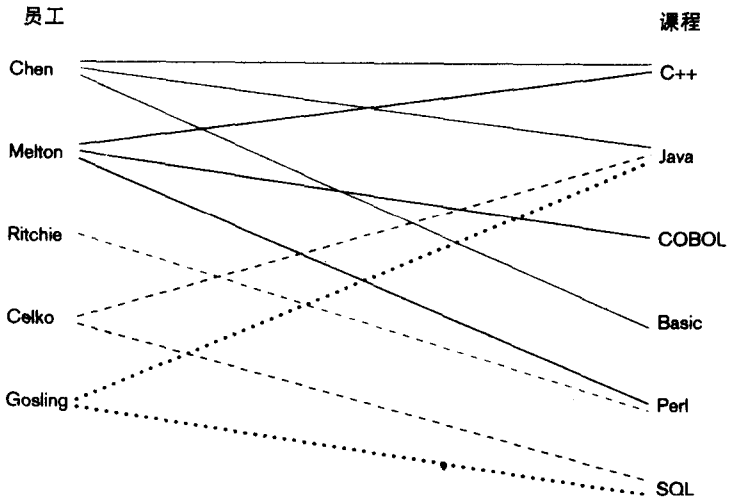
3.6 联系

联系将E-R图中所有部分连接起来。直观地讲，联系（relationship）表示的是组织感兴趣的一或多个实体类型之间的关联。为了更清晰地理解联系的定义，必须分清联系类型和联系实例之间的区别。例如，考虑实体类型EMPLOYEE和COURSE，这里COURSE代表员工培训课程。为了跟踪某个员工已结束的某门课程，在这两个实体类型中定义联系Completes（如图3-10a所示）。这是一个多对多的联系，因为一个员工可以完成多门课程，而一门课程可以由任意数量的员工完成。例如，在图3-10b中，员工Melton已经完成三门课程（C++、COBOL和Perl）。有两个员工（Celko和Gosling）已经完成SQL课程。

在这个例子中，两个实体类型（EMPLOYEE和COURSE）参与名为Completes的联系。通常，任意多个实体类型（从一个到多个）都可能参与到一个联系中。



a) 联系类型（Completes）



b) 联系实例

图3-10　联系类型和联系实例

本章和下一章使用菱形符号和一个动词短语表示联系。因为联系通常因组织事件的发生而

产生，而实体实例就通过这个动作被连接起来，所以动词短语比较合适。这个动词短语应该是描述性的，并使用现在时。还有其他许多方法可以表示联系。在第1章中，应用另一个常见的符号：仅用一条线来表示联系，没有出现菱形，线的两端各写上联系的名称。一些数据建模人员更愿意使用这种方法。这两种方法在结构上是等价的，可以任意选用其中一种。

3.6.1 联系的基本概念和定义

联系类型（relationship type）表示实体类型之间有意义的关联。“有意义的关联”说明联系中包含一些仅用实体类型无法表示的含义。联系类型由包含类型名的菱形符号表示，如图3-10a中的例子所示。建议读者使用尽量短的、描述性的动词短语，采用良好的命名方式对于用户来说是很有意义的（本节后面的部分将详细讨论联系的命名和定义原则）。

联系实例（relationship instance）是实体实例之间的关联，每个联系实例恰包括参与联系的各实体类型中的一个实例（Elmasri和Navathe，1994）。例如，在图3-10b中，图中的每一条线代表一个员工和一门课程之间的联系实例，表示这个员工已经结束这门课程。

1. 联系的属性

正如实体可以用多对多（一对一）联系关联起来一样，属性也可以用多对多（或一对一）的联系关联起来。例如，假设员工结束课程时，组织希望记录完成课程的日期（月、年）。这个联系的属性名为Date_Completed。其他一些例子见表3-2。

那么，在E-R图中Date_Completed属性应放在什么位置？参见图3-10a，可以发现Date_Completed没有和EMPLOYEE或COURSE中的任一个实体发生联系。这是因为Date_Completed是联系Completes本身的性质，而不是任一个实体的性质。这就是说，在每一个联系Completes的实例中，都有一个Date_Completed值。该实例表示这样的语义：员工Melton在2000年6月结束课程C++的学习。

图3-11a是描述这个例子的修正版本的E-R图。图中的Date_Completed连接到连接两个实体的联系符号上。如果需要，也可以在联系中加上其他属性，如Course_Grade、Instructor和Room_Location。

表3-2 Data_Completed联系的实例

Employee_Name	Course_Title	Date_Completed
Chen	C++	06/2000
Chen	Java	09/2000
Chen	Basic	10/2000
Melton	C++	06/2000
Melton	COBOL	02/2001
Melton	SQL	03/2000
Ritchie	Perl	11/2000
Celko	Java	03/2000
Celko	SQL	03/2001
Gosling	Java	09/2000
Gosling	Perl	06/2000

2. 关联实体

联系本身拥有一或多个属性，说明联系可以和实体类型采用同样的处理方式。**关联实体**（associative entity）表示关联一个或多个实体类型的实例之间的实体类型，同时包含特属于实体实例间联系的属性。关联实体CERTIFICATE由矩形框加上菱形的符号表示，如图3-11b所示。在E-R图中，这个特殊符号表示该实体最初被指定为一个联系。因为动词形式的联系名通常要转换

成为名词形式的实体名（动词的ing形式），所以关联实体有时以动名词形式表示。注意，在图3-11b中，在关联实体和强实体的连线中没有用菱形表示的联系。这是因为关联实体表示联系。

在下面四种情况中，需要将联系转化为关联实体类型。

1) 所有参与实体类型的联系都是“多”联系。

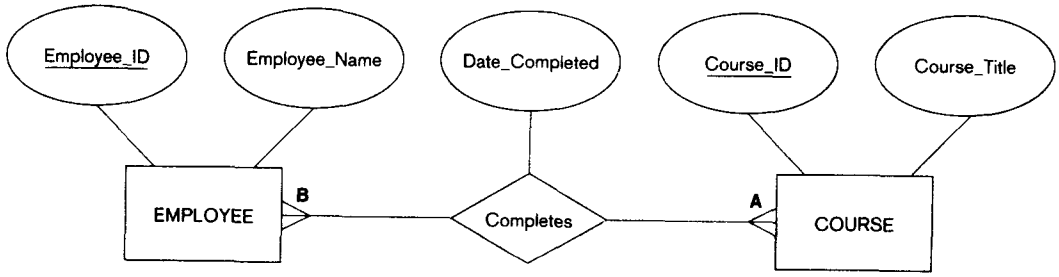
2) 对最终用户来说，关联实体类型具有独立的意义，最好用单值属性作为标识符。

3) 除了标识符外，关联实体还有其他一个或多个属性。

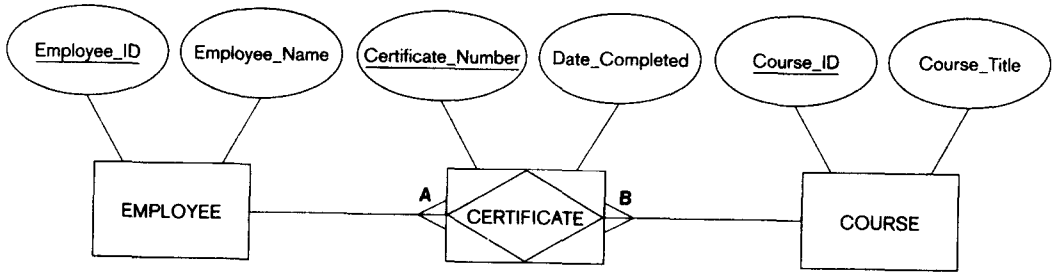
4) 关联实体参与一个到多个联系，这些联系独立于与关联联系相关的实体。

图3-11b表示联系Completes转换为关联实体类型。在这种情况下，公司的培训部将决定给学完课程的员工授予证书。因此，这个实体被命名为CERTIFICATE，该实体对最终用户来说显然具有独立含义。每一个证书以证书编号（Certificate_Number）作为标识符，该实体类型还包括属性Date_Completed（完成日期）。

注意，由于将联系转化为关联实体需移动表示该联系的符号。这就是说，基数“多”在关联实体处中止，而不是在每个参与实体类型处中止。在图3-11中，这表示一个完成一到多门课程（图3-11a中的符号A）的员工可能被授予一至多个证书（图3-11b中的符号A）。同时，既然可以有一至多个员工结束同一门课程（图3-11a中的符号B），那么也可以给一至多个员工授予该门课程的证书（图3-11b中的符号B）。



a) 联系的属性



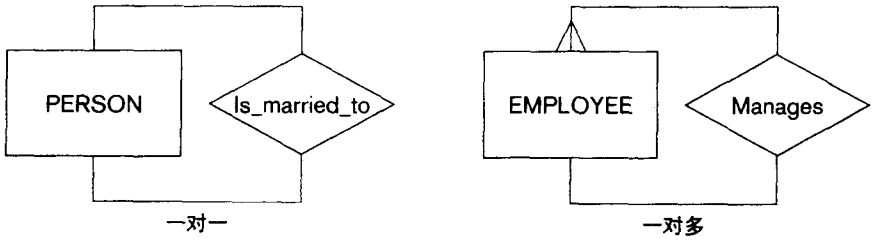
b) 关联实体（CERTIFICATE）

图3-11　关联实体

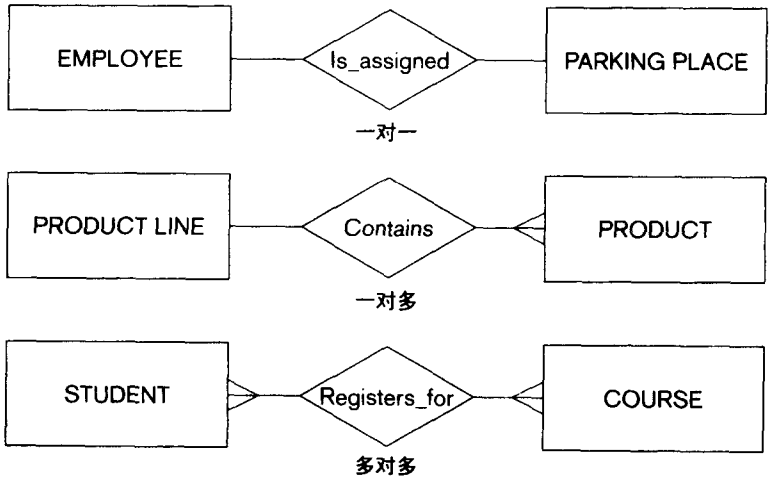
3.6.2 联系的度

度（degree）是参与联系的实体类型的个数。在图3-11b中，因为有两个实体类型（EMPLOYEE和COURSE）参与联系，所以联系Completes的度为2。在E-R中最常见的联系的度有一元（度数为1）、二元（度数为2）和三元（度数为3）。更高度数的联系也是可能的，但在实践中很少见，所以本章仅讨论三元以内的联系。图3-12是一元、二元和三元联系的例子。

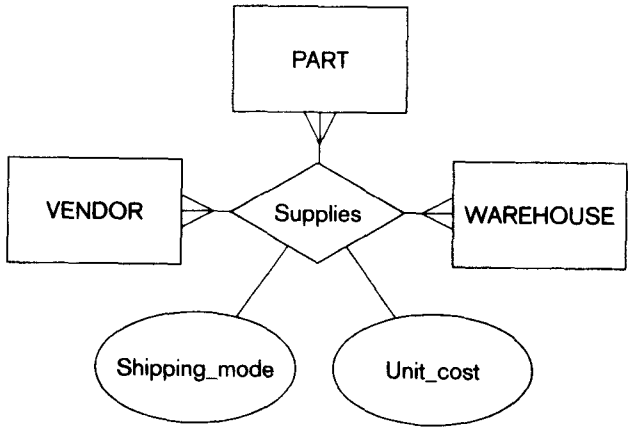
在图3-12中，任何一个数据模型都表示一种特定的情况，而不是一般情况。例如，考察图3-12a中的Manages联系。在某些组织中，一个员工可能会被许多其他员工管理（例如在矩阵型组织中）。根据这些情况开发E-R模型时，必须要理解该组织的业务规则。



a) 一元联系



b) 二元联系



c) 三元联系

图3-12 具有不同度的联系示例

1. 一元联系

一元联系（unary relationship）是指同一实体类型的不同实例之间的联系（一元联系也被

称为递归联系）。图3-12a中是两个一元联系的例子。在第一个例子中，“Is_married_to”（与某人有婚姻关系）表示为实体类型PERSON的两个实例之间的一对一联系。因为是一对一联系，所以该符号只表示一个人当前的婚姻状况应该保存下来。在第二个例子中，“Manages”是实体类型EMPLOYEE的实例之间的一对多联系。利用这个联系，能够确定一个特定管理者管理的全部员工。（注意，如图3-2所示，在这些例子中，均忽略这些联系是否是强制的还是可选择的。本章下一节将介绍这些概念。）

图3-13显示另一个一元联系的例子：*材料单结构*。许多成品由装配件构成，而装配件又由更小的零件和部件组成。在3-13a中，使用多对多的一元联系来表示这个结构。实体类型ITEM用于表示各种类型的组成部分，联系类型的名字是“Has_components”（拥有组件），用来将上一层装配结构和下一层装配结构关联起来。

图3-13b是材料单结构的两个实例。其中的每一个图都显示装配结构中的直接下层组件及其数量。例如，标准件X包括标准件U（数量为3）和标准件V（数量为2）。可以很容易确定其联系是多对多联系。许多装配结构中有多种类型的组件（例如，标准件A有三个直接下层组件，即X、V和Y）。同样，其中一些组件又可用于多个上层装配件的组装。例如，标准件X可同时用于组装标准件A和B。多对多的联系可以确保在标准件X用于构建其他标准件时总会用到X的部件结构。

在联系中出现属性Quantity（数量），这表示分析员将把联系“Has_Components”转换为关联实体。图3-13c中显示实体类型BOM_STRUCTURE，该实体类型构成了ITEM实体类型的不同实例之间的关联。BOM_STRUCTURE中还有一个属性Effective_Date，用以记录该组件第一次用于装配的日期。如需要阀的历史记录时，会需要这个表示生效时间的属性。

2. 二元联系

二元联系（binary relationship）是两个实体类型的实例之间的联系，这是数据建模中最常见的一种联系。图3-12b是三个二元联系的例子。第一个例子（一对一联系）指出给一个员工分配一个停车点，而且一个停车点只能分配给一个员工。第二个例子（一对多联系）指出一个产品系列可以包含许多种产品，而一个产品只属于一个产品系列。第三个例子（多对多联系）显示每个学生可以注册多门课程，而每一门课程都可以由许多学生注册。

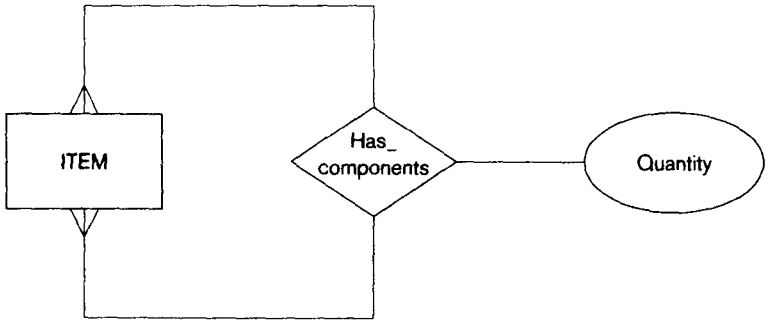
3. 三元联系

三元联系（ternary relationship）是同时存在于三个实体类型的实例之间的联系。一个典型的三元联系例子如图3-12c所示。在这个例子中，供货商能够给仓库提供多种零件。联系Supplies（供应）用来记录这样的业务规则：一个特定的零件由一个给定的供货商提供给一个特定的仓库。在此出现三个实体类型：VENDOR、PART和WAREHOUSE。在联系Supplies中有两个属性：Shipping_Mode（运输方式）和Unit_Cost（单价）。联系Supplies的一个实例记录下述事实：供货商X可以运送零件C到仓库Y，货运方式为隔日航空运输，费用为每单位5美元。

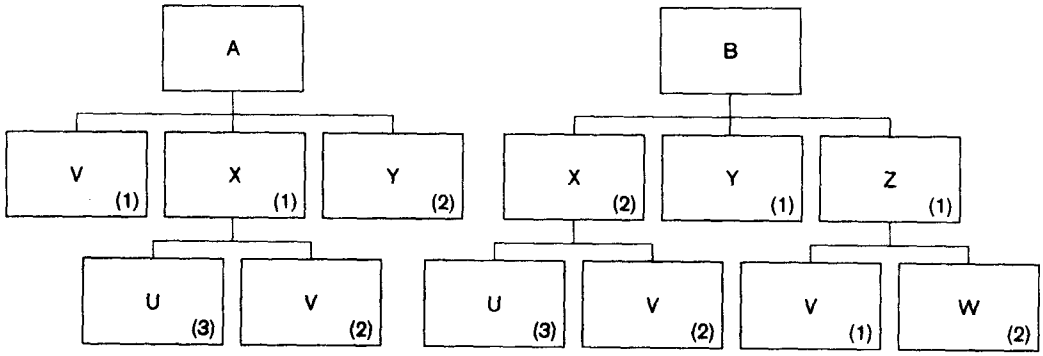
注意，一个三元联系和三个二元联系是不等价的。例如，图3-12c中，Unit_Cost是Supplies联系的一个属性。三个实体类型可以形成三个二元联系，但属性Unit_Cost不能正确地关联到其中任意一个二元联系中（例如在PART和WAREHOUSE之间）。如果我们说供货商X能以每单位8美元的价格运送零件C，但由于没有指出要将零件送至哪一个仓库，所以数据仍然是不完整的。

在图3-12c中，由于在联系Supplies中出现一个属性，所以通常提示设计者应该将该联系转化成一个相关的实体类型。图3-14是和图3-12c等价的另一种三元联系的表示形式。在图3-14中，（关联）实体类型SUPPLY SCHEDULE代替了图3-12c中的联系Supplies。很明显，用户需要独立的实体类型SUPPLY SCHEDULE。然而要注意，SUPPLY SCHEDULE中还没有标识符，这

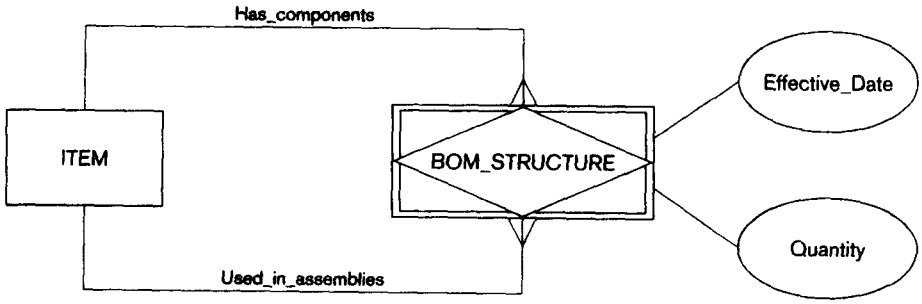
是允许的。如果在E-R建模中，还没有为一个关联实体类型指定标识符，那么在逻辑建模过程中将会指定一个标识符（或键），这将在第6章中讨论。该标识符是一个复合标识符，其组成部分包括每一个参与联系的实体类型的标识符。在本例中，关联实体SUPPLY SCHEDULE的标识符是由实体类型PART、VENDOR和WAREHOUSE的标识符所组成的复合标识符。你还能想出SUPPLY SCHEDULE的其他属性吗？



a) 多对多联系



b) 两个实例



c) 关联实体

图3-13 一份材料单的结构

在图3-14中没有使用菱形符号，而是直接用线连接SUPPLY SCHEDULE和三个实体。这是因为这些线表示的不是二元联系。为了完整、准确地表达图3-12c中三元联系的含义，不能将Supplies联系分解成三个二元联系。

本书强烈建议读者像该例一样，把所有三元（或更高）联系转换为关联实体。Song、Evans和Park（1995）认为使用本章中符号，不能精确地表示三元联系中的参与约束（在下一节讲述）。然而，通过转换为关联实体，就能精确表示约束。另外，包括大多数CASE工具在内，许多E-R图绘制工具不能表示三元联系。所以，尽管在语法上不准确，有时需要用一个关联实体和三个二元联系来表示三元联系。

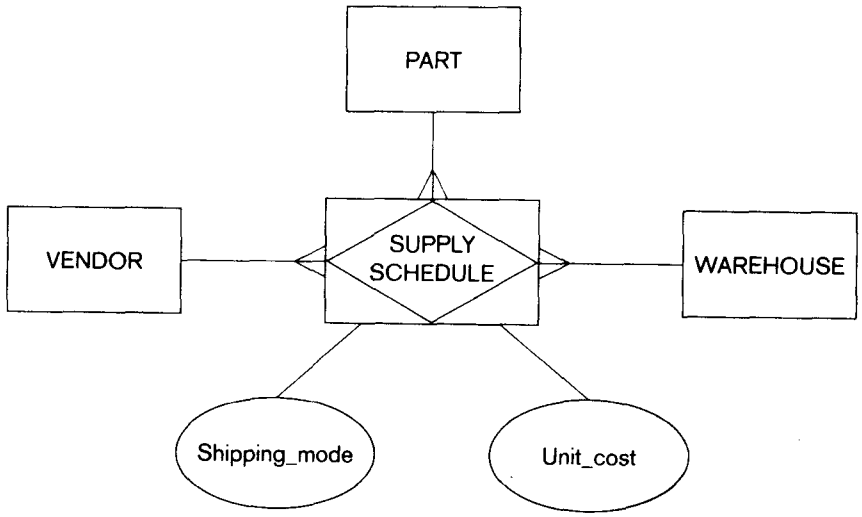


图3-14　关联实体形式的三元联系

3.6.3　属性还是联系

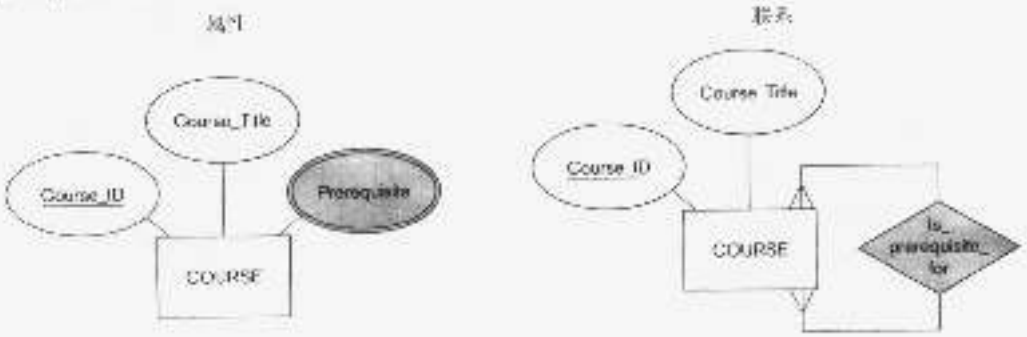
图3-15给出三种通过联系描述属性的情况。在图3-15a中，一门课程的预备课程也是课程的形式，同时一门课程也可以作为其他许多课程的预备课程。所以，预备课程可视为课程之间的一个一元联系，而不是COURSE的多值属性。通过一元联系表示预备课程也意味着，查找一门课程的预备课程和查找一门课程的后续课程都能够以实体类型间联系的方法来处理。如果把预备课程作为COURSE的多值属性，查找一门课程的后续课程方法是：遍历所有COURSE实例，以查找预备课程为该课程的实体实例。在图3-15b中，员工具备技能，但是技能也可以作为一个实体类型，因为技能也是组织所感兴趣的、具有独立含义的数据对象。技能的属性包括技能的简单描述和技能类型，如技术或管理。员工有技能，但不作为属性，而是作为相关实体类型的实例。但是，在有些人看来，在图3-15a和3-15b所示的情况中，多值属性比实体类型更合适，这种做法可以简化E-R图。

那么，在什么情况下，应该将一个属性通过联系链接到实体类型呢？答案是：如果属性是标识符或数据模型中实体类型的其他特征，同时该属性被多个实体实例共享，则需要进行这种转化。图3-15c给出这条规则的例子。在这个例子中，EMPLOYEE有一个复合属性Department（部门）。因为Department表示业务中的一个完整概念，并且有多个员工需共享相同的部门数据，所以应该没有冗余地在实体类型DEPARTMENT中表示部门数据，并通过部门的有关数据与其他实体类型建立联系。通过这种方法，不但不同的员工可以共享相同的部门数据，而且分配给部门的多个项目和部门中的各工作组都可以共享相同的部门数据。

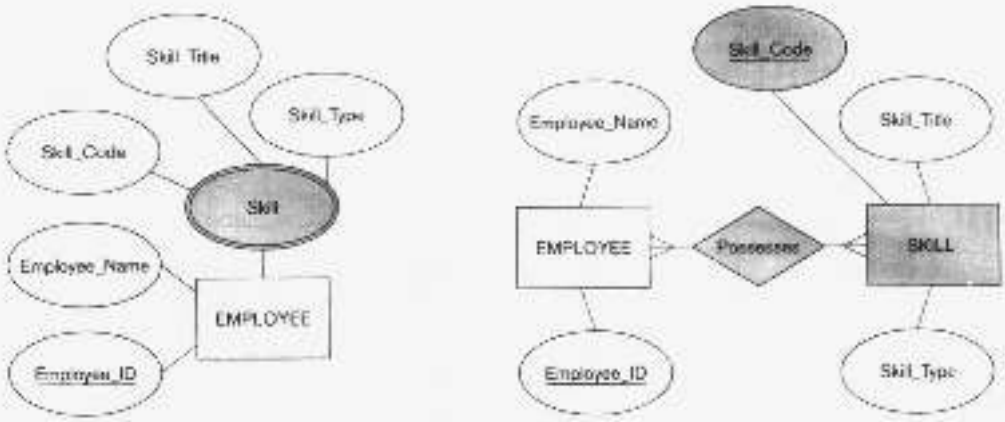
3.6.4　基数约束

假定有两个实体类型A和B，它们通过联系相连接。**基数约束**（cardinality constraint）指定对于实体A的每个实例，有多少个实体B的实例可能（或必须）与之关联。例如，考虑一个出

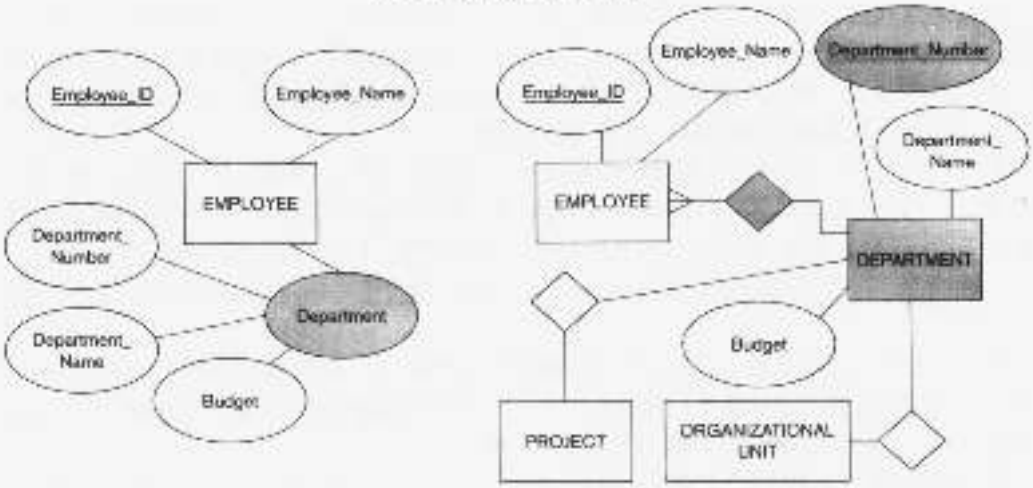
租电影录像带的音像店。这个音像店可以保存一部影片的多盘录像带，即图3-16a所示的“一对多”联系。当然，在某一段时间，店里可能没有一盘该影片的录像带（如所有录像带都被借出）。我们应该引入一种更精确的符号来指出联系中基数的范围。在图3-2中已经出现过这个符号，现在读者需要复习一下。



a) 多值属性与联系



b) 复合属性、多值属性与联系



c) 由其他实体共享的数据项复合属性

图3-15 使用联系链接有关属性

1. 最小基数

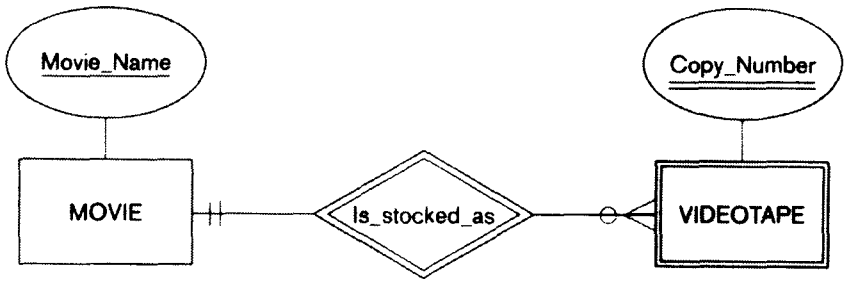
联系的**最小基数**（minimum cardinality）是实体B的实例可以和实体A的每个实例相关联的最小个数。在上面所说的录像带的例子中，对于一部影片来说，录像带的最小数目为0。当参与的最小数目为0时，我们说参与联系的实体B是可选的。本例的“Is_stocked_as”联系中的VIDEOTAPE是可选的。在图3-16b中，从VIDEOTAPE发出的带“0”一个箭头符号表示这个事实。

2. 最大基数

联系的**最大基数**（maximum cardinality）是实体B的实例可以和实体A的一个实例相联系的最大个数。在上面所说的录像带的例子中，对于VIDEOTAPE实体，其最大基数为“多”，即大于一的一个不确定的数字。在图3-16b中，从VIDEOTAPE发出的带有“鸟爪”符号的一个箭头表示这个事实。



a) 基本联系



b) 带有基数约束的联系

图3-16　基数约束

联系是双向的，所以在MOVIE实体旁也有一个基数符号。注意，其最大基数与最小基数都是1，如图3-16b所示，这称为强制1（mandatory one）的基数。也就是说，每盘录像带只能是一部影片的拷贝。参与联系的实体可以是可选的，也可以是强制的。如果最小基数为0，则参与是可选的；如果最小基数为1，则参与是强制的。

如图3-16b所示，每个实体类型中都添加了一些属性。注意，VIDEOTAPE表示为一个弱实体类型。这是因为如果相应的影片不存在，则录像带也不会存在。MOVIE的标识符是Movie_Name（影片名）。VIDEOTAPE没有惟一的标识符，但是Copy_Number（拷贝名）是一个部分标识符，这个部分标识符与Movie_Name组合，即可惟一确定VIDEOTAPE的一个实例。

3. 例子

在图3-17中，显示了最大与最小基数的所有组合的例子。每个例子声明一个基数约束的业务规则，并给出相应的E-R符号。同时还给出一些联系的实例以阐明联系的本质。读者应认真研究每个例子。以下是图3-17中每个例子的业务规则：

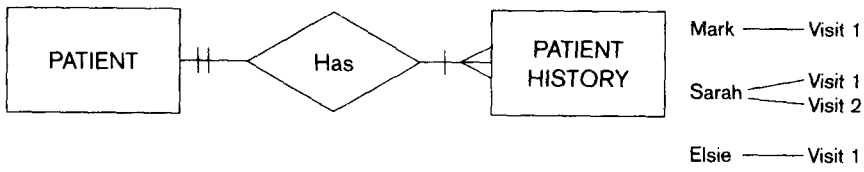
1) PATIENT Has PATIENT HISTORY（参见图3-17a）。每个病人有一个至多个病历（如果病人是第一次看病，将产生他的PATIENT HISTORY的第一个实例）。PATIENT HISTORY的一

个实例仅“属于”一个PATIENT。

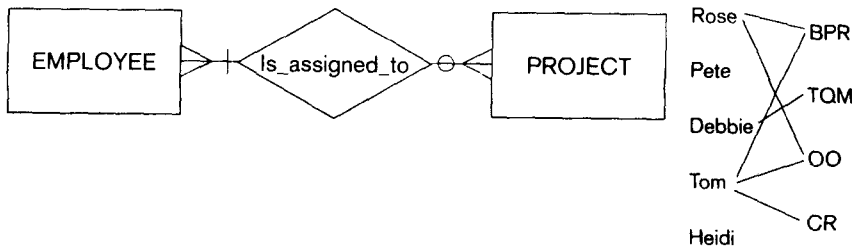
2) EMPLOYEE Is_assigned_to PROJECT（参见图3-17b）。每个PROJECT中至少有一个EMPLOYEE（有些项目可以有多名员工）。每个EMPLOYEE有可能（可选的）分配到一个已有的PROJECT中（例如，员工“Pete”），也可能分配到一个或多个项目中。

3) PERSON Is_married_to PERSON（参见图3-17c）。因为某个人可能已婚也可能未婚，所以该联系的两个方向的基数都是可选的（0或1）。

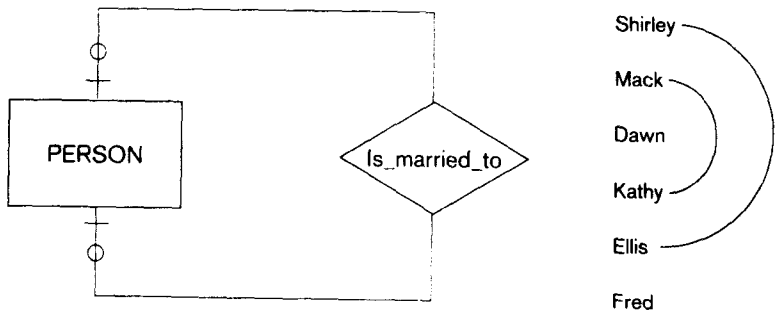
最大基数可能是一个固定的值，而不是一个模糊的“多”值。例如，有一项企业制度规定一个员工最多可以同时在5个项目中工作，那么在图3-17b中，可以通过在PROJECT旁的鸟爪符号的上方或下方写上数字5来表示这个业务规则。



a) 强制基数



b) 一个方向可选，另一个方向为强制的基数



c) 可选基数

图3-17 基数约束的例子

4. 一个三元联系

图3-14中给出一个带有关联实体SUPPLY SCHEDULE的三元联系，本节将基于相关业务规则在该图中加入基数约束。修改后的E-R图和相关的业务规则如图3-18所示。注意，PART和WAREHOUSE强制参与到联系中，而VENDOR则是可选的。因为每个SUPPLY SCHEDULE实例必须与这些参与实体类型的一个实例相联系，所以每个参与实体的基数都强制为1。

如前所述，一个三元联系不等价于三个二元联系。但是，许多CASE工具中不提供绘制三元

联系的机制，所以，读者不得不先将一个三元联系转化为三个二元联系。如果必须这样做，那么注意，在每个二元联系中不要用菱形符号表示联系，并确保三个强实体类型的基数强制为1。

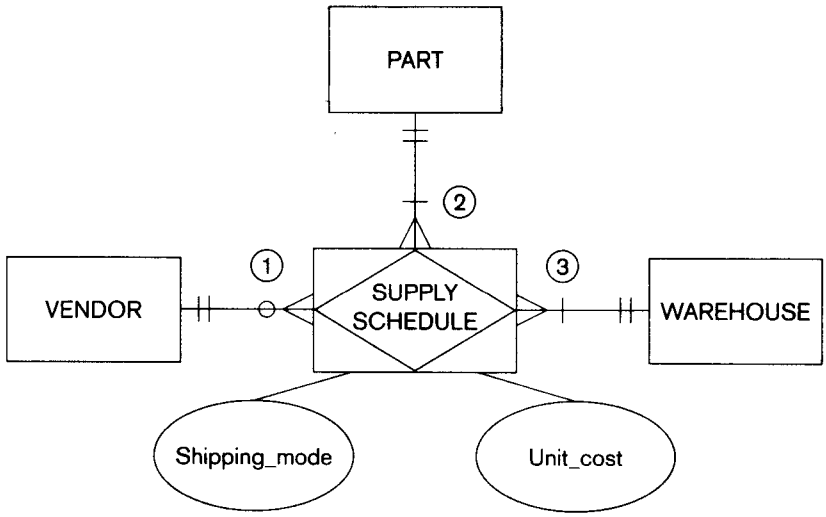


图3-18　三元联系中的基数约束

3.6.5　建立依赖于时间的数据模型

数据库的内容是随时变化的。例如，在数据库中存储着产品信息。因为原材料价格、劳动力的价格以及市场是变化的，所以产品的单价也会发生变化。如果仅需要当前的单价，可在建模时将Price作为一个单值属性。但是，对于会计、账单以及其他方面的要求来说，应该保留单价变化的记录以及变化生效的时间。如图3-19所示，可以将这些需求概念化为一系列价格和价格生效日期。这样做会产生名为Price_History的多值属性，其组成部分有Price和Effective_Date。在这个复合的、多值的属性中，一个重要特征是其组成部分都是具有内在联系的。如图3-19所示，每个Price与相应的Effective_Date成对出现。

在图3-19中，每个Price的属性值都是带有代表其生效日期的时间戳。**时间戳**（time stamp）是一个与数据值相关联的时间值（如日期和时间）。如果需要维护数据值的历史记录，那么可将时间戳与任何随时间变化的数据值相关联。时间戳记录可以指出数据值输入的时间（事务处理时间），值合法存在的时间和变为非法的时间，以及执行某些临界动作的时间（如更新、更正和审计等）。

简单时间戳（如前面的例子）通常可以满足为依赖于时间的数据建模的要求。但是，时间常常会使数据建模变得复杂。例如，图3-20a显示松谷家具公司E-R图的一部分，每个产品都分配到一个产品系列（或相关的产品组）中。公司要年复一年地处理顾客订单，并按月提交产品系列和产品系列中各个产品的汇总。

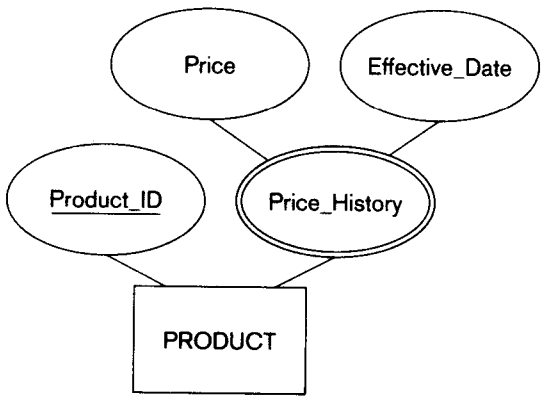
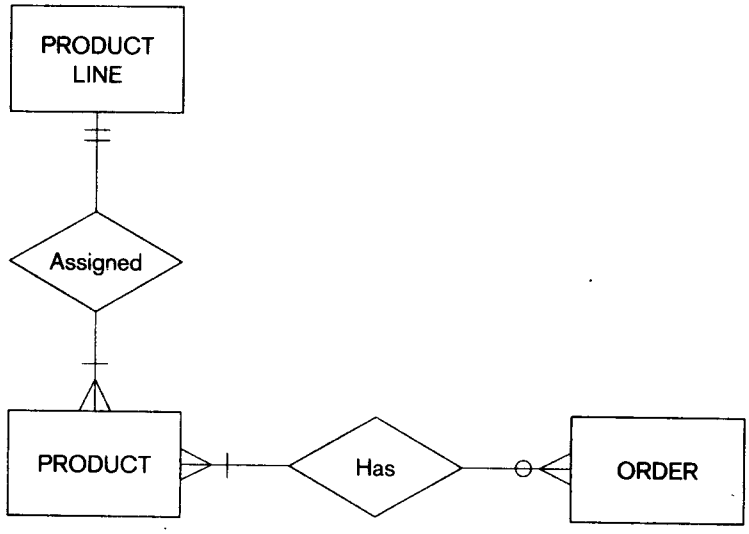
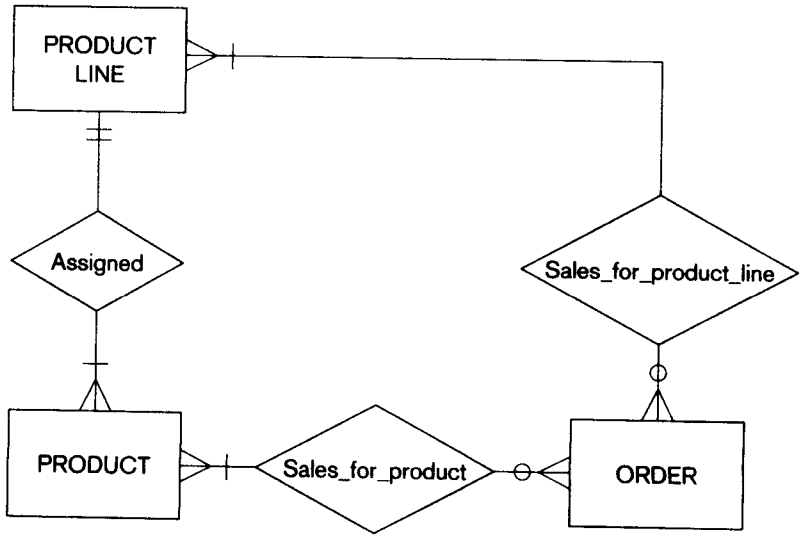


图3-19　简单时间戳的例子



a) 不含产品重新分配因素的E-R图



b) 包含产品重新分配因素的E-R图

图3-20 松谷家具公司的产品数据库

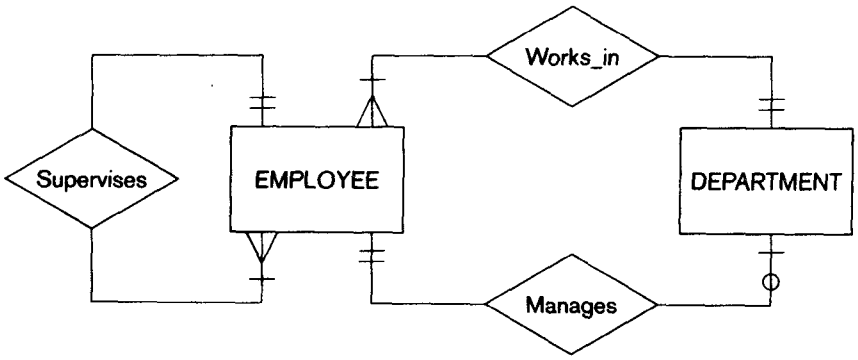
假设在年中，由于需要对销售进行重新组织，所以一些产品被重新分配到其他产品系列上。图3-20a中的模型没有考虑这一因素。于是，所有的销售报告反应的销售合计都是基于产品的当前产品系列，而不是产品销售时的产品系列。例如，在销售报告中可能反应：本年初到目前为止，产品系列B上的某种产品的销售累计为50 000美元，但实际上产品系列B中的该产品销售了40 000美元后，该产品已经重新分配到产品系列A中了。在图3-20a的模型中显然遗漏了这个事实。在图3-20b中，对原有设计进行了一些简单的改动，即可表示这一事实。在ORDER和*PRODUCT LINE*中间增加了一个新的联系（即Sales_for_product_line）。当处理顾客订单时，同时确认订单上的产品在销售时所在的产品系列。

本书作者曾经与许多组织中的管理者讨论关于时间依赖数据的问题。对于数据建模和数据

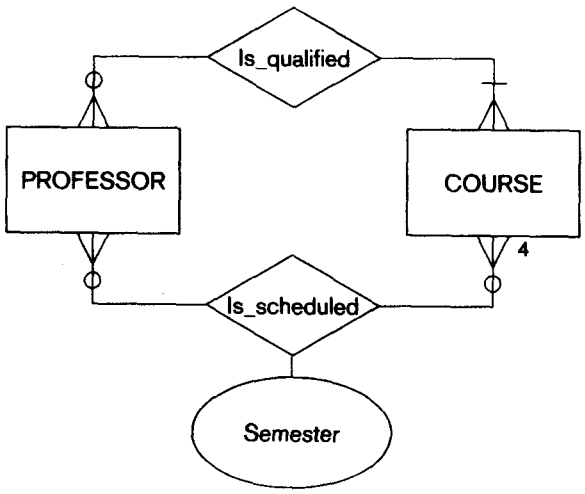
库管理，这些管理者拥有丰富的经验。通过讨论，作者发现，现有的数据模型（以及基于这些模型的数据库管理系统）在处理具有时间依赖关系的数据方面的能力是不够的，在数据建模中，人们一般忽略这个问题，并希望这些不精确之处能够互相抵消。但是，在数据仓库中，通过对依赖时间的数据进行显式地建模，可以消除这些不确定性。在设计依赖时间的数据模型时，读者应对其中的复杂性给予充分重视。

3.6.6 实体之间的多种联系

在某些特殊情况下，需要对同样的实体类型建立多种联系。图3-21是这种情况的两个例子。在图3-21a中，实体类型EMPLOYEE和DEPARTMENT之间有两种联系。联系Works_in将员工与其所在部门关联起来，这个联系是一个一对多联系，并是双向强制的。也就是说，一个部门至少必须有一个员工（可能是经理），而一个员工必须被分配到某一个部门中（只能分配到一个部门）。注意，此处为了说明多个联系的问题而规定这个特定的业务规则。理解业务规则对于设计E-R图是至关重要的。例如，如果EMPLOYEE包括退休员工，则“每个员工必须被分配到一个部门中”这条业务规则就不适用。另外，图3-21a中的E-R模型假定组织只需存储每个EMPLOYEE当前工作的DEPARTMENT，而不需存储人员分配的历史记录。数据模型的结构反映组织应该存储的信息结构。



a) 员工和部门



b) 教授和课程（增加了约束）

图3-21　多种联系的例子

EMPLOYEE和DEPARTMENT之间的第二个联系是“Manages”，它将每个部门和管理部门的员工关联起来。该联系在从DEPARTMENT到EMPLOYEE的方向上是强制为1的关系，表示一个部门必须由且仅由一名经理管理；但从EMPLOYEE到DEPARTMENT方向是可选为1的关系，表示并不是所有的员工都是经理。

图3-21a是一个一元联系Supervises，它将每个员工与他的主管人联系起来，反之亦然。这个联系说明下述业务规则：每个员工有且仅有一名主管人。反过来，一名员工可以主管任意数量的员工，他也可能根本不是主管。

图3-21b中显示实体类型PROFESSOR和COURSE之间的两个联系。联系Is_qualified表示教师有资格讲授某门课程。可能会有多名教师有资格讲授某一门课程，但也可能（可选的）没有一名教师有资格讲授该门课程。如果某门课程是新开设的课程并且刚刚加入学校的教学计划，则可能会出现没有教师有资格讲授这个课程的情况。另一方面，每个教师必须至少有讲授一门课程的资格。（合理要求！）

图中的第二个联系将教师及其在某个学期所要讲授的课程联系起来。注意，在一个学期中，教师讲授课程的最大基数是4。这是一个在E-R图中记录固定约束（最大或最小）的例子。联系Is_scheduled（计划）有一个属性Semester（学期），这是一个由Semester_Name和Year组成的复合属性。

在图3-21b的E-R图中，可以看出基本的基数约束。但在一般情况下，在一个基本的E-R图中寻找业务规则并不容易。考虑图3-21b中的下列规则：如果教师被安排讲授一门课程，则他必须首先具有讲授这门课程的资格。（另一个合理的要求！）这条规则就难以从E-R图中直接找到。在第4章中，将讨论从整个建模过程中寻找业务规则的方法。

3.6.7 命名和定义联系

除了命名数据对象的一般原则外，下面还列出一些专用于命名联系的原则：

- 联系名应是一个动词短语，如Assigned_to、Supplies或Teaches等。联系表示要采取的动作，通常使用现在时。联系名说明要采取的动作，而不是动作的结果（如Assigned_to比Assignment更合适）。联系名说明参与联系的实体类型之间的互相作用的本质，而不是所涉及的过程。（例如，对于Employee与Project之间的联系，联系名Assigned_to比Assigning更合适。）
- 应避免使用含义不清楚的名字，如Has或Is_related_to等。应使用描述性的动词短语，通常可以从联系的定义中找到合适的描述动作的短语。

以下是一些用于定义联系的特定原则：

- 联系的定义应该解释发生什么动作以及这个动作为何重要。在某些情况下，指出实施该动作的人或事件也是比较重要的。但是，没有必要描述动作怎样发生。可以说明联系中涉及的业务对象。但是，因为在E-R图中已经显示联系所涉及的实体类型，而且在其他地方也可以找到实体类型的定义，所以没有必要描述业务对象。
- 在某些情况下，有必要给出阐明动作内容的例子。例如，对于在学生和课程之间的联系Registered_for，应该解释这个联系同时包括现场注册和在线注册，也包括留级学期和复学学期的注册。
- 定义必须解释任何可选参与。设计者应该解释这些问题：在什么情况下会产生零关联实例（基数为0）；是第一次创建实体实例时会产生这种情况，还是任何时候都有可能发生这种情况。例如，“Registered_for联系表示学生与课程之间的关系”。一门课程在未开课之前没有学生注册，但是也可能从未有过学生注册；学生在课程开课之前不能注册，但是也可能不注册任何一门课程（也可能注册了课程，然后未通过某些课程或所有课程）。

- 如果联系中存在显式定义的固定基数，则需在联系定义中进行说明。例如，“Assigned_to联系表示员工和项目之间的关系，根据工会的协议，一个员工同时参加的项目不能超过4个”。这是一种典型的描述基数上界的业务规则，暗示最大基数可能发生变化。在本例中，工会的下一个协议可能改变这个上界。所以，在实现最大基数时，应允许最大基数的改动。
- 联系定义中应解释互斥联系。互斥联系是指下述情况：实体实例可以参与许多联系，但一个实体实例仅能同时参与其中的一个联系。第4章将给出这种情况的例子。现在考虑下面这个例子：“联系Plays_on表示校际运动队与其队员的关系。校际运动队中的学生队员不能同时参与学校中的其他兼职工作。也就是说，一个学生不能通过联系Plays_on与校际运动队联系的同时又通过联系Works_on与学校中的其他工作相连接”。另一个互斥限制的例子是员工实体的两个实例之间不能同时存在联系Supervised_by和联系Married_to。
- 联系定义中应解释联系的参与中的限制关系。除了互斥这种限制外，还有其他许多种限制。例如，“联系Supervised_by表示员工与其下属员工之间的关系以及员工与其主管之间的关系。一个员工不能是自己的主管人；职位等级低于4的员工也不能成为主管人”。
- 联系定义应解释联系中需保留的历史记录的范围。例如，“联系Assigned_to表示病床与病人之间的关系。在系统中只需保存当前的病床分配情况。如果病人没有住院，则不需分配病床；在某个时刻，病床也可能空闲”。另一个描述联系的历史记录的例子是“联系Places表示顾客和订单之间的关系。数据库中仅存储两年内的订单记录。所以，并不是所有的订单都参与联系”。
- 联系定义应指出是否一个联系实例中的实体实例可以转移参与到另一个联系实例中。例如，“联系Places表示顾客和订单之间的关系，同时表示订单与相关顾客之间的关系。订单不能转移到另一个顾客”。另一个例子是“联系Categorized_as表示产品系列与该系列中产品的关系，同时还表示产品和相关产品系列之间的关系。根据组织结构的变化和产品的设计特性，产品可以重新归类到另一个产品系列中。Categorized_as只保存当前的产品系列和产品的关系”。

3.7 E-R建模的例子：松谷家具公司

有两种方法可用于开发实体-联系图。使用自顶向下的方法，设计者从业务的基本描述开始，包括其制度、处理方法和环境。在开发高层E-R图时，由于只包括主要实体和联系以及有限的属性集合（如仅有标识符），所以使用这种方法是最合适的。使用自底向上的方法，设计流程从与用户讨论细节开始，并包括对文件、显示和其他数据源的细节研究。这种方法对于开发一个详细的、“全属性”的E-R图来说是必要的。

在本节中，总体上基于第一种方法为松谷家具公司开发一个高层E-R图。同时，以用户发票（见第1章）为基础并增加一些细节数据，描述如何将这两种方法结合使用。

在对松谷家具公司业务流程的研究中，确定了下列一些实体类型。在每个实体类型中都标出了标识符和一些重要的属性。

- 公司销售不同的家具产品。这些产品分成几个不同的产品系列。产品的标识符是Product_ID，产品系列的标识符是Product_Line_ID。根据图1-6的用户发票，还确定出产品的以下属性：Product_Description、Product_Finish以及Standard_Price[⊖]。产品系列的

⊖ 图1-6中的发票显示了单价，但选择standard price作为属性名。因为将来的单价可能是给顾客打折后的价格，所以要存储标准的产品价格。

另一个属性是Product_Line_Name。一个产品系列可以包含任意多种产品，但是必须至少包含一种产品。每个产品都必须属于且仅属于一个产品系列。

- 顾客提交产品订单。订单的标识符是Order_ID，另一个属性是Order_Date。顾客可以提交任意多个订单，也可以不提交任何订单。每个订单都由且仅由一个顾客提交。顾客的标识符是Customer_ID，其他属性还包括Customer_Name、Customer_Address和Postal_Code。
- 一个已有的订单必须至少包括一种产品，并且对于订单的每一行上只能包括一个产品。松谷家具公司所销售的产品可以不出现在任何订单上，也可能出现在一个或多个订单上。与每个订单条目相关的属性是Ordered_Quantity，表示需要的数量。
- 松谷家具公司已经为其顾客建立了销售区域网络。每个顾客和一个或多个销售区域发生业务联系。销售区域的标识符属性为Territory_ID，还有一个属性Territory_Name。销售区域可以和一个或多个顾客发生业务联系，但也可能不和任何顾客发生业务联系。
- 松谷家具公司有许多销售员。销售员的标识符属性是Salesperson_ID，其他属性还包括Salesperson_Name、Salesperson_Telephone以及Salesperson_Fax。一个销售员仅在一个销售区域工作，而一个销售区域可以有一个或多个销售员在工作。
- 每个产品都是由一定数量的一种或几种原材料组装而成的。原材料实体的标识符是Material_ID，其他属性还包括Unit_of_Measure、Material_Name、Standard_Cost。每种原材料可以参与一种或多种产品的组装，在每种产品中每种原材料的数量都是一定的。
- 原材料由供货商提供。供货商的标识符属性为Vendor_ID，其他属性还包括Vendor_Name和Vendor_Address。每种原材料可以由一个或多个供货商提供。每个供货商可以提供一种或多种原材料，但是某个供货商也可以不给松谷家具公司提供任何原材料。供货商和原材料之间的联系有一个属性为Supply_Unit_Price。
- 松谷家具公司已经建立起多个加工中心。加工中心的标识符属性为Work_Center_ID，还有一个属性为Work_Center_Location。每个产品都可以由一个或多个加工中心生产，而一个加工中心可以生产多种产品，也可以不生产任何产品。
- 公司有100多名员工。员工的标识符属性为Employee_ID，其他属性还包括Employee_Name、Employee_Address和Skill。一个员工可以具备多种技能。每个员工可以在一个或多个加工中心工作。每个加工中心必须至少有一名员工在工作，但是也可以有任意多个员工在工作。
- 除总裁外，每个员工有且仅有一名主管，总裁没有主管。作为主管的员工可以领导任意多个员工，但并不是所有的员工都是主管。（注意，这个业务规则的定义不是很明确，但迄今为止，在E-R图中引入的符号都掩盖了这种二义性。用户不能从规则或图中得知某一个员工是手下暂时无人的主管，还是他根本不是一名主管。在第4章中引入的符号将用于澄清这种区别。）

3.8 松谷家具公司的数据库处理

图3-22中E-R图用于为松谷家具公司提供数据库概念设计。通过与数据库未来的使用者进行交流，检查设计质量对于数据库设计是很重要的，其中一个重要的质量检查类型就是确定E-R图是否能够方便满足用户对于数据或信息的需求。松谷家具公司的员工可能会有许多关于数据检索和报告的需求。在本节中，将描述对于图3-22所示E-R图，如何通过数据库处理来满足一些信息需求。

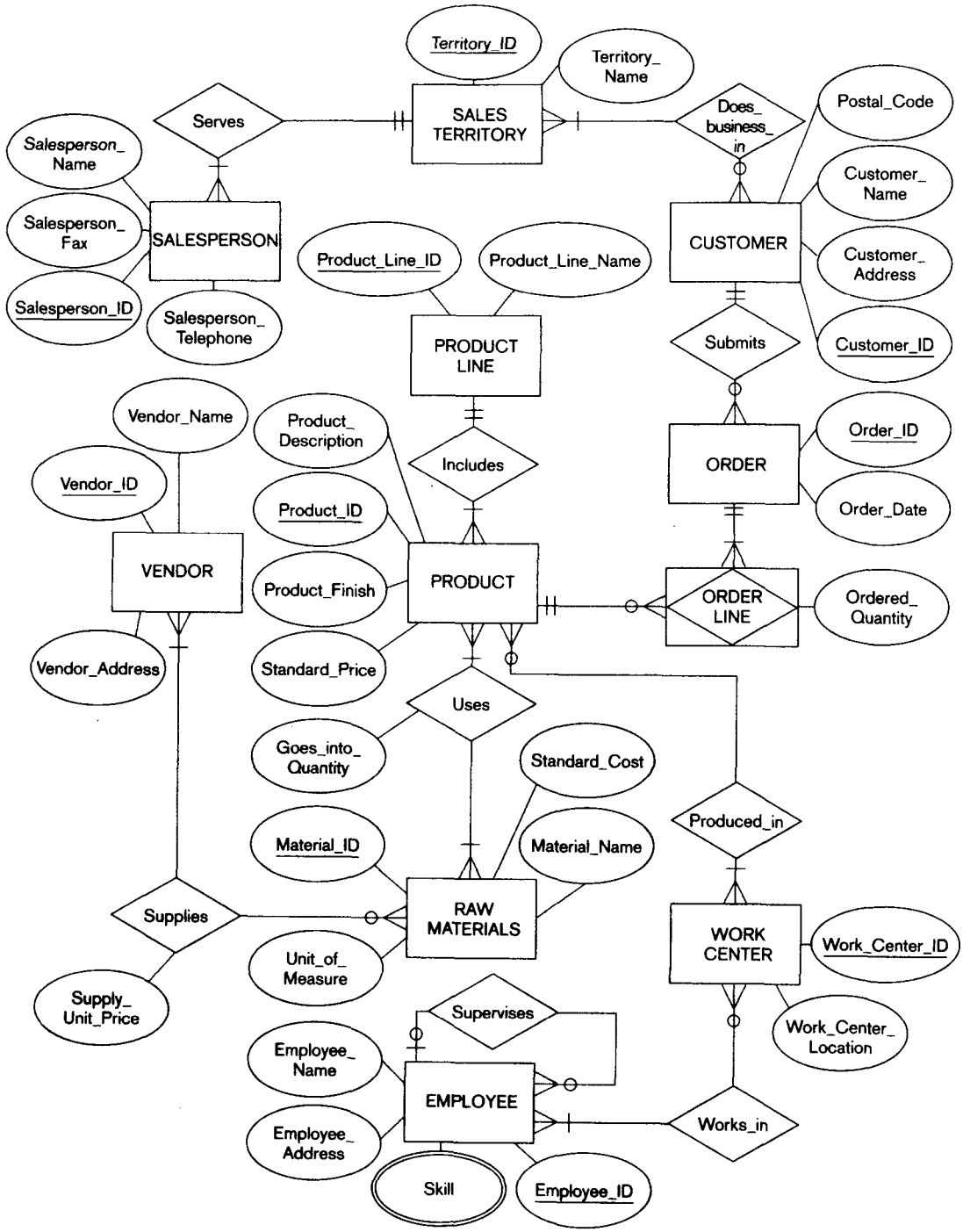


图3-22　松谷家具公司的E-R图

本章使用SQL数据库处理语言（在第7章和第8章中详细介绍）来表述这些查询。为了清楚地理解这些查询，读者应该理解在第5章中介绍的概念。但是，本章中的一些简单查询能够帮助读者理解数据库的集合运算的能力，也为读者理解第5章以及以后各章中所介绍的SQL查询

奠定良好的基础。

3.8.1 显示产品信息

许多不同的数据库用户需要查看松谷家具公司的信息数据，例如，销售员、仓储经理和产品经理。一个典型的情况是：推销员要对顾客关于某类产品的需求及时反应。这种查询的例子如下：

列出库存的各种计算机桌的所有细节信息。

此查询所涉及的数据保存在PRODUCT实体（如图3-22所示）中。该查询遍历这个实体，结果显示包含“Computer Desk”的实体的所有属性。

该查询的SQL代码是：

```
SELECT *
FROM PRODUCT
WHERE Product_Description LIKE "Computer Desk%";
```

该查询的典型的输出结果及形式如下：

Product_ID	Product_Description	Product_Finish	Standard_Price
3	Computer Desk 48"	Oak	375.00
8	Computer Desk 64"	Pine	450.00

SELECT * FROM PRODUCT要求显示PRODUCT实体的所有属性。WHERE子句表示将显示范围限制在其描述以短语“Computer Desk”开头的产品中。

3.8.2 显示顾客信息

在组织中，另一个常用的查询是显示有关松谷家具公司顾客的信息数据。销售区域经理对这些信息特别感兴趣。下面是一个销售区域经理的典型查询：

列出在Northwest销售区域中所有客户顾客的细节。

该查询所涉及的数据保存在CUSTOMER实体中。正如第5章所介绍的，当图3-22所示的E-R图被转化为一个能通过SQL访问的数据库时，属性Territory_ID将被加到CUSTOMER实体中。查询遍历该实体并显示处于所选范围内的顾客的所有属性。

该查询的SQL代码如下：

```
SELECT *
FROM CUSTOMER
WHERE Territory_ID = "Northwest";
```

典型的输出内容和格式如下：

Customer_ID	Customer_Name	Customer_Address	Territory_ID
5	Value Furniture	394 Rainbow Dr., Seattle, WA 97954	Northwest
9	Furniture Gallery	816 Peach Rd., Santa Clara, CA 96915	Northwest

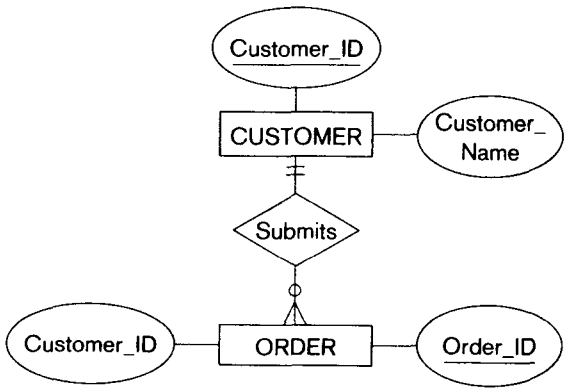
该SQL查询的解释与前一查询相似。

3.8.3 显示顾客订单状态

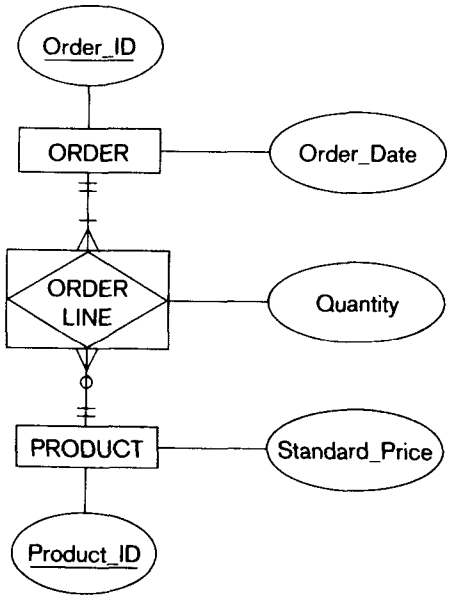
以上的查询相对简单，每个例子仅涉及一张表中的数据。通常，在一个查询中需要用到多张表中的数据。尽管查询很简单，仍然要浏览整个数据库以找出满足查询条件的实体和属性。

为简化查询的编写以及出于其他原因，许多数据库管理系统支持创建有限的、满足特定用户信息需求的数据库视图。对于与用户订单状态相关的查询，松谷家具公司应用如图3-23a所示的用户视图“顾客订单”。这个用户视图仅允许用户看到数据库中的CUSTOMER和ORDER

实体，以及这些实体在图中所显示的属性。正如第5章中所介绍的，属性Customer_ID将被加入到ORDER实体中（如图3-23a）。一个典型的订单状态查询是：



a) 用户视图1：顾客订单



b) 用户视图2：产品订单

图3-23　松谷家具公司的两个用户视图

公司从“Value家具公司”收到了多少个订单？

编写这个查询的SQL代码可以使用许多办法。本章选择的方法是在一个查询中组合另一个查询（即子查询）。这个查询分两步执行。首先，子查询（或称为内部查询）遍历CUSTOMER实体以确定名称为“Value Furniture”的顾客的Customer_ID（从前一个查询可知，该公司的ID为5）；然后，查询（或称为外部查询）遍历ORDER实体并计算出所有该顾客的订单总数。

通过“顾客订单”用户视图进行该查询的SQL代码如下：

```
SELECT COUNT (Order_ID)
FROM ORDER
```

```
WHERE Customer_ID=
  (SELECT Customer_ID
  FROM CUSTOMER
WHERE Customer_Name="Value Furniture");
```

该查询的典型输出为:

```
COUNT(Order_ID)
---------------
              4
```

3.8.4 显示产品销售

销售员、区域经理、产品经理、生产经理以及相关人员都需要知道产品的销售状况。获知哪种产品在某个月的销售比较强劲就是这一类问题。该查询的一般形式如下:

在上个月（2001年6月），哪种产品的销售总额突破25 000美元?

这个查询可以使用图3-23b所示的用户视图“orders for products”编写。该查询涉及的数据源如下:

- ORDER实体中的Order_Date属性（以找出在期望月份中的订单）。
- 在期望月份中，与实体ORDER相联系的关联实体ORDER LINE中每一个订单里每一种产品的数量。
- 与实体ORDER LINE相联系的实体PRODUCT中产品的Standard_Price。

对于在2001年6月订购的每项产品，查询通过将Quantity与Standard_Price相乘得出销售额。通过累计所有订单中的某种产品的销售额，就可以得到这种产品的总销售额。结果中仅显示总销售额超过25 000美元的产品数据。

这个查询的SQL代码已经超出本章的介绍范围，其编写方法需要用到在第5章中介绍的技术。在第5章中，将进一步讨论这个查询的SQL代码。本章使用这个查询的目的只是为了说明图3-22所示的数据库具有从细节数据中查找管理信息的能力。在许多现代的组织中，用户使用Web浏览器来获取上述信息。与Web页有关的程序代码调用相应的SQL命令来获取所需的信息。

本章小结

本章详述了组织中数据建模的基础。在一个组织中，可以从制度、过程、事件、职能和其他业务对象中总结出业务规则，然后由业务规则定义管理组织的约束，从而说明如何处理和存储数据。对于信息系统，特别是数据库应用系统，业务规则是否具有强大的功能来描述系统中的需求，人们尚有争议。业务规则的能力取决于以下条件是否满足：业务规则应该是业务的核心概念；应能够以最终用户熟悉的术语表达；应具有高度的可维护性以及能够用自动的方法（大部分是通过数据库）执行。一个好的业务规则有以下特征：可声明性、精确性、原子性、一致性、可区分性和面向业务性。

基本的业务规则是数据名称和数据定义。本章讲述了在业务中清晰地命名和定义数据对象的方法。在概念数据建模中，必须对实体类型、属性和联系进行命名和定义。其他的业务规则可能在这些数据对象之上定义约束。这些约束可以从数据模型和其他相关的文档中获得。

现在最常用的数据建模符号是实体-联系数据模型。E-R模型表示组织中数据的细节和逻辑关系。E-R模型通常由E-R图表示，E-R图是E-R模型的图形化表示。1976年，Chen首先提出了E-R模型。但迄今为止，E-R模型还没有一套标准的符号体系。

E-R模型中的基本结构为实体类型、联系和有关的属性。实体是用户环境中诸如人、地方、对象、事件和概念等，实体以组织中存储的数据形式表示。实体类型是具有相同性质的实体集

合，而实体实例是实体类型的实例化。强实体类型拥有属于自身的标识符，并且它的存在不依赖于其他实体。弱实体类型的存在依赖于一个强实体类型。弱实体没有属于自身的标识符，通常情况下只有一个部分标识符。一般通过属主实体类型的标识联系来标识弱实体类型。

属性组织感兴趣的实体或联系的性质和特征。属性有许多种类。简单属性不能分解。复合属性可以分解为更小的组成部分。例如，Person_Name可以分解为First_Name、Middle_Name和Last_Name。多值属性在实体的一个实例中有多个值。例如，一个人的College_Degree属性中可能有多个值。导出属性的值可由其他属性值计算得到。例如，Average_Salary（平均工资）可以由全体员工的Salary计算得到。

标识符是可以惟一确定一个实体类型的实例的属性。必须谨慎选择标识符以确保其稳定性和使用方便。标识符可以是简单属性，也可以是复合属性。

联系类型是两个或多个实体类型之间具有明确意义的关联。联系实例是两个或多个实体实例之间具有明确意义的关联。联系的度指参与联系的实体类型的个数。最常见的联系类型有一元联系（度数为1）、二元联系（度数为2）和三元联系（度数为3）。

在开发E-R图时，有时会遇到多对多（或一对一）联系的情况。在这种情况下，联系可能会有属于自身的一个或多个属性，而不是与某个参与实体类型相关的属性。这提示开发者可以将联系转化为一个关联实体。这种实体在多个实体实例之间起关联作用，并且拥有属于自身的属性。关联实体类型的标识符可能是一个属于该实体类型的简单属性，也可能是在逻辑设计过程中分配的一个复合属性。

基数约束声明实体B的实例可能（或必须）与实体A的每个实例相关联的个数。基数约束通常指定实例个数的最大值和最小值。该约束可能是强制为一、强制为多、可选为一、可选为多或者一个特定的数值。最小基数约束也可称为参与约束。最小基数为0指定可选参与约束，最小基数为1指定强制参与约束。

本章复习

关键术语

关联实体	复合属性	实体实例
属性	复合标识符	实例-联系图（E-R图）
二元联系	度	业务规则
导出属性	实体-联系模型（E-R模型）	基数约束
实体	实体类型	最小基数
术语	事实	多值属性
三元联系	标识符	联系实例
时间戳	标识属主	联系类型
一元联系	标识联系	简单属性
弱实体类型	最大基数	强实体类型

复习问题

1. 定义以下术语：

a. 实体类型	b. 实体-联系模型
c. 实体实例	d. 属性
e. 联系类型	f. 标识符
g. 多值属性	h. 关联实体

i. 基数约束　　j. 弱实体
k. 标识联系　　l. 导出属性
m. 多值属性　　n. 业务规则

2. 将下列术语和定义匹配起来。

_____复合属性　　a. 惟一确定实体实例
_____关联实体　　b. 一个实体的实例之间的联系
_____一元联系　　c. 声明实例的最大和最小个数
_____弱实体　　d. 以实体类型的形式建模的联系
_____属性　　e. 表示实体类型之间的关联
_____实体　　f. 相同实体的集合
_____联系类型　　g. 参与联系的实体类型个数
_____基数约束　　h. 实体的性质
_____度　　i. 可以分解为更小的组成部分
_____标识符　　j. 其存在依赖于其他实体类型
_____实体类型　　k. 度数为3的联系
_____三元　　l. 多对多的一元联系
_____材料单　　m. 人、地方、对象、概念和事件等

3. 解释下列术语的区别与联系。

a. 存储属性；导出属性　　b. 实体类型；实体实例
c. 简单属性；复合属性　　d. 实体类型；联系类型
e. 强实体类型；弱实体类型　　f. 度；基数

4. 为什么许多系统设计人员认为数据建模是系统开发过程中最重要的一个环节。给出至少三条理由。

5. 解释为什么业务规则方法被称为系统需求的新的范型。给出至少四条理由。

6. 说明在组织中何处可以发现业务规则。

7. 阐述数据模型中数据对象命名的6条一般原则。

8. 阐述在实体中选择标识符的4条标准。

9. 说明在哪三种情况下，设计人员会将联系建模为关联实体类型。

10. 列出4种基数约束，并分别给出一个例子。

11. 给出一个本章未提及的弱实体类型的例子，指出标识联系存在的必要性。

12. 解释联系的度。列出本章所述的三种类型的联系的度，并分别给出一个例子。

13. 给出本章未提及的下列概念的例子:

a. 导出属性　　b. 多值属性
c. 复合属性

14. 给出本章未提及的下列概念的例子:

a. 三元联系　　b. 一元联系

15. 给出以生效日期作为实体属性的例子。

16. 阐述在什么情况下，应将某个属性从实体类型中取出并置于一个链接的实体类型中。

问题和练习

1. 比较业务规则中“术语”和“事实”这两个概念。给出一个业务规则的例子并指出其中的术语和事实。

2. 在3.6节中详细解释了图3-22。对于解释中的每一条，根据其内容给出图3-22的子集。

3. 对于下列每一种情况，绘制相应的E-R图。如果需要添加其他属性，应就这些属性进行说明。

a. 某公司有多名员工。EMPLOYEE的属性包括Employee_ID（标识符）、Name、Address和Birthdate。该公司有多个项目。PROJECT的属性包括Project_ID（标识符）、Project_Name和Start_Date。每个员工可以分配到一个或多个项目中，也可以不被分配到任一项目中。一个项目中至少有一名员工，也可以有多名员工。某个员工的工资水平根据不同项目而变化，公司希望记录每个员工被分配到某个项目时对应的工资水平（Billing_Rate）。上述描述中出现的属性名是否符合书中所讲的原则？如果不是，给出更好的属性名。

b. 某大学开设许多课程。COURSE的属性包括Course_Number（标识符）、Course_Name和Units。每门课程可能要求先学完一至多门预备课程，也可能不要求学习任何预备课程。同样，某门课程可能是多门课程的预备课程，也可能不是任何课程的预备课程。给出COURSE的定义。并说明该定义为什么是一个良好的定义。

c. 一个实验室有多名化学家，分别从事一个或多个项目。化学家在每个项目中可能用到某些种类的仪器。CHEMIST的属性包括Employee_ID（标识符）、Name和Phone_No（电话号码）。PROJECT的属性包括Project_ID（标识符）和Start_Date（开始日期）。EQUIPMENT（仪器）的属性包括Serial_No（编号）和Cost。组织希望记录Assign_Date，即将某台仪器分配给特定项目中某个特定化学家使用的日期。一个化学家必须至少分配至一个项目中，并至少分配一台仪器。某台仪器可能闲置，某个项目也可能既没有化学家工作也没有分配仪器。给出上述情况下所有联系的定义。

d. 一门大学的课程可能被分为几个班级，也可能不包含任何班级。COURSE的属性包括Course_ID、Course_Name和Units。SECTION的属性包括Section_Number和Semester_ID。Semester_ID由两部分组成：Semester和Year。Section_Number是一个整数（如“1”或“2”），用来区别同一课程中的不同班级，但并不能独立标识一个班级。阐述如何为SECTION建模，并说明为什么采取这种建模方式而不采取其他方式的理由。

e. 某医院中有许多医生。PHYSICIAN的属性包括Physician_ID（标识符）和Specialty。病人由医院中的医生治疗。PATIENT的属性包括Patient_ID（标识符）和Patient_Name。任何病人都必须由且仅由一名医生对其进行诊断。医生可以诊断多名病人。一旦诊断结束，一名病人必须由至少一名医生对其进行治疗。一名医生可能会治疗多名病人，也可能不治疗任何一名病人。医院需要记录任何时候某个医生对某个病人进行治疗的细节（Treatment_Detail）。Treatment_Detail由Date、Time和Results组成。请问：在医生和病人之间是否存在多种联系？请说明原因。

4. 图3-24所示为每学期末发给学生的成绩单。试设计一个E-R图以反映成绩单中的所有数据。假定每门课仅由一名教师讲授。

5. 实体类型STUDENT有下列属性：Student_Name、Address、Phone、Age、Activity和No_of_Years。Activity表示某项学校组织的活动，No_of_Years表示学生参加该活动的年数。一个学生可能参加不止一项活动。绘制E-R图表示这种情况。

6. 为一个房地产公司开发一个与销售有关的E-R图，并给出图中每个实体类型、属性和联系的定义。该公司情况的描述如下：

- 公司在不同的州有许多销售部。销售部的属性包括Office_Number（标识符）和Location。
- 每个销售部中有一名或多名员工。员工的属性包括Employee_ID（标识符）和

Employee_Name。每个员工只能分配到一个销售部工作。

MILLENNIUM COLLEGE
GRADE REPORT
FALL SEMESTER 200X

NAME: Emily Williams ID: 268300458
CAMPUS ADDRESS: 208 Brooks Hall
MAJOR: Information Systems

COURSE ID	TITLE	INSTRUCTOR NAME	INSTRUCTOR LOCATION	GRADE
IS 350	Database Mgt.	Codd	B104	A
IS 465	System Analysis	Parsons	B317	B

图3-24 成绩单

- 每个销售部都有一名经理。一个经理只能管理他所在的销售部。
- 公司要列举出销售的许多房产。这些房产的属性包括Property_ID（标识符）和Location。Location由Address、City、State和Zip_Code组成。
- 一个销售部可能列举出多个销售出的房产，但也可能无法列举出任何一套房产。
- 每套房产均有一个至多个拥有者。拥有者的属性包括Owner_ID（标识符）和Owner_Name。一个拥有者可以拥有一套至多套房产。房产和拥有者间的联系有一个属性Percent_Owned。

7. 在下列图中加入合理的最大和最小基数符号。

a. 图3-5
b. 图3-10a
c. 图3-11b
d. 图3-12（所有部分）
e. 图3-13c
f. 图3-14

8. 假设在学完数据库管理课程以后，有人请你为一个交响乐团设计一个初步的E-R图。通过研究，你发现如下实体类型：

- CONCERT SEASON 在该演出季将举办一系列音乐会。其标识符为Opening_Date，其中包括Month、Day和Year。
- CONCERT 要演奏一个至多个曲目的一次演出。标识符为Concert_Number，另一个重要的属性是Concert_Date（演出日期），这个属性由Month、Day、Year和Time组成。每台编排好的音乐会可以在不同日期演奏多个场次。
- COMPOSITION 在每场音乐会上演奏的曲目。标识符为Compositon_ID，该属性由Composer_Name（作者）和Composition_Name（曲目）组成。另一个重要的属性是Movement_ID（乐章标识），该属性包括两个部分：Movement_Number（乐章号）和Movement_Name（乐章名）。许多曲目（但不是所有曲目）都有多个乐章。
- CONDUCTOR 指挥音乐会的人。标识符为Conductor_ID。另一个属性是Conductor_Name。
- SOLOIST 在某台音乐会上演奏某个曲目的独奏演员。其标识符为Soloist_ID。另一个属性为Soloist_Name。

通过进一步讨论，你还发现以下规则：

- 在一个演出季安排一台至多台音乐会。某台音乐会只安排在一个演出季举行。
- 一台音乐会包含一个或多个节目的表演。一个节目可以在一个或多个音乐会上表演，或

者可能不表演。

- 每台音乐会都有一个指挥。某个指挥可以指挥多台音乐会，也可能不指挥任何一台音乐会。
- 每个曲目可能需要一名至多名独奏演员，也可能不需要独奏演员。独奏演员可以参与一台音乐会上一个至多个曲目的演出，也可以不参加任一曲目的演出。交响乐团希望记录独奏演员参加的最后一个场次演出的日期（Date_Last_Performed）。

绘制E-R图以表示上述内容。根据上述内容确定业务规则，并解释如何使用E-R图为该业务规则建模。

9. 研究一个常见的用户视图，如信用卡账目、电话账单以及其他一些常见的文档。尝试为该文档设计E-R图。

10. 为你熟悉的组织设计E-R图，如男童子军、女童子军以及运动队等。

11. 针对下列情况设计E-R图（见Batra、Hoffer和Bostrom的著作，1988）。同时，选择一组单词作为标识符和类，以便形成属性名，并解释选择这些单词的原因。

Projects Inc. 是一家拥有近500名员工的工程公司。现需设计一个数据库以记录有关全体员工、他们的技能、所从事的项目和所在部门的情况。公司给每个员工分配一个工号，同时还需存储员工的姓名和生日。如果公司内两个员工有婚姻关系，则记录其结婚日期和配偶。但如果员工的配偶不是本公司员工，则不需要记录其婚姻状况。每个员工有一个职务（如工程师、秘书等）。在给定时间内，某个员工只能有一项职务，并且在数据库中只需要存储其当前的职务。

公司内有11个部门，部门的名称是惟一的。每个员工仅隶属于一个部门。每个部门有一个电话号码。

每个部门可以从多个供货商处购得多种设备。一个供货商可以向多个部门提供设备。需要记录每个供货商的名称和地址以及部门和供货商最后一次会谈的日期。

一个项目中可以有多名员工在工作。一名员工可以在多个项目中工作（例如，Southwest Refinery、California Petrochemicals等等），但是一个员工在一个给定的城市中只能从事一个项目。对于每个城市，需要存储其所在的州和人口。一名员工可以有多项技能，如准备材料需求、检查图纸等，但在一个给定的项目中可以只用其中的几项技能。如员工MURPHY在Southwest Refinery项目中做准备材料的工作，而在California Petrochemicals项目中做准备材料和检查图纸的工作。员工利用自己的各项技能参与至少一个项目。每项技能有一个编号，在数据库中需存储关于每一项技能的简单描述。不同的项目有各自的项目号，并应存储大概的项目成本。

12. 针对下列情况设计E-R图。如果必须作出一些假设，才能使图形的含义完整，则给出并说明这些假设。Stillwater古董店收购并销售各类古董，如家具、珠宝、瓷器和衣物等。每一件古董都有一个编号，其他特征还包括描述、开价、条件和自由评论等信息。Stillwater从客户手里购入古董，并卖给其他客户。一些客户只卖出古董，而另外一些客户只买入古董，但其他的客户可能既从这里买入，也向Stillwater卖出他们手里的古董。客户通过客户号进行标识，其他属性还包括客户名和客户地址。当Stillwater卖出一件古董后，需要记录佣金、售价、税金（税金为0表示免税）和售出日期。当Stillwater买入一件古董后，需要记录买入价格、买入日期和当时的条件。

13. 针对下列情况设计E-R图。如果必须作出一些假设，才能使图形的含义完整，则给出并说明这些假设。H.I.Topi School of Business从事国际培训和教育业务，其业务范围遍及欧洲的10个地区。1965年，学校首期毕业9000名学生。学校保存毕业生的以下档案：每个毕业生的学号、在校时的姓名、出生国、现在的国籍、现在的名字、现在的地址以及学生所完成的每个专业（每个学生可以有1至2个专业）。为了加强其校友之间的联系，学校在全世界举办一系列活动。

活动的属性有主题、日期、地点和类型（如招待会、座谈会和讨论会等）。学校应该保存毕业生参加这些活动的记录。当某个毕业生参加一次活动，校方在学生档案中记录关于该生在该活动中的一条说明。学校也同时保存毕业生的联系方式，如邮政地址、电子邮件、电话号码和传真号码等。当举办活动时，学校通过上述方式与毕业生联系。当学校需要与一个毕业生联系时，就会打印出一份报告。报告上会列出近两年历次联系和活动中记录的学生的最新信息。

14. 假定在松谷家具公司的每种产品（其属性有产品号、产品描述和成本）至少包括三种组件（其属性有组件号、组件描述和度量单位），每种组件可用于制造一至多种产品。另外，假定组件可用于制造其他的组件，而且假设原材料也是一种组件。在这两种情况下，需要记录组装时用掉多少组件。针对这些情况设计E-R图，并在图上标出最大和最小基数。

15. 股票经纪人代客户卖出股票，股票价格不断变化。根据这些情况设计E-R图，需要考虑股票价格的变化。

16. 在每个学期，学校给每个学生指定一名导师。导师指导学生如何取得学位，并帮助学生选择课程。学生应在辅导员的帮助下选择课程，但如果指定给该学生的导师不在，学生可以在其他导师的帮助下选择课程。学校要记录以下信息：学生，指定给学生的导师，帮助学生选择课程的导师的姓名。在E-R图中描述上述关于学生和导师的信息。

17. 在图3-22中，考虑每个多对多（M:N）联系。对于每个多对多联系，应该用一个关联实体表示。试解释之。重画E-R图中将多对多联系转化为关联实体的部分。

应用练习

1. 咨询一个数据库分析人员或系统设计人员，了解他们在数据模型中如何命名数据对象。他所在的组织是否有自己的命名约定。如果有，描述该模式；如果没有，询问对方在没有命名约定的前提下，是否曾经出现问题。

2. 访问两个本地的小型企业。一个属于服务业，如干洗店、汽车修理店、宠物医院或书店等；另一个属于制造业，生产实际的产品。与其员工进行交流，找出在这些组织中常见的实体、属性和联系。使用上述信息绘制E-R图。服务业和制造业企业的E-R图有何异同？实体-联系建模技术是否能够较好地处理这两种情况？解释原因。

3. 咨询数据库或系统分析人员。根据他所在的公司的情况，给出一元、二元和三元关系的例子。说明其中哪些例子比较常见，并解释之。

4. 请一个本地企业的数据库或系统分析人员展示该企业某个基本数据库的E-R图。如果有不清楚的地方，你可以向他提问，以便你能够透彻理解每个实体、属性和联系的含义。该组织所使用的E-R符号是否与书中所述的相同？如果不同，描述该企业所使用的E-R符号，并给出每个符号的含义。该企业的E-R图中是否出现关联实体？如果没有，给出其关联实体的建模方法。

5. 对于上题或其他任一数据库所对应的E-R图，考察其中是否存在时间戳或其他时间依赖数据的模型。说明对于用户来说，时间依赖数据存在的必要性。如果不需要描述属性值的历史记录，E-R是图否可以大幅度简化。

参考文献

Aranow, E.B.1989. "Developing Good Date Definitions." *Database Programming & Design* 2(8): 36-39.

Batra, D., J.A.Hoffer, and R.B.Bostrom. 1988. "A Comparison of User Performance Between the Relational and Extended Entity Relationship Model in the Discovery Phase of Database Design." *Proceedings of the Ninth International Conference on Information Systems*. Minneapolis, Nov. 30-Dec. 3: 295-306.

Bruce, T.A. 1992. *Designing Quality Database with IDEF1X Information Models*. New York: Dorset House.

Chen, P.P. -S. 1976. "The Entity-Relationship Model—Toward a Unified View of Data." *ACM Transactions on Database Systems* 1(March): 9-36.

Elmasri, R., and S. B.Navathe. 1994. *Fundamentals of Database Systems*. 2nd ed. Menlo Park, CA: Benjamin /Cummings.

Gottesdiener, E. 1977. "Business Rules Show Power, Promise." *Application Development Trends* 4(3): 36-54.

Gottesdiener, E. 1999. "Turning Rules into Requirements." *Application Development Trends* 6(7): 37-50.

GUIDE, "GUIDE Business Rules Project." Final Report, revision 1.2. October, 1997.

Hoffer, J.A., J.F.George, and J.S. Valacich. 2002. *Modern Systems Analysis and Design*. 3rd ed.Upper Saddle River, NJ:Prentice Hall.

Moriarty, T.2000. "The Right Tool for the Job." *Intelligent Enterprise* 3(9):68,70-71.

Plotkin, D.1999. "Business Rules Everywhere." *Intelligent Enterprise* 2(4):37-44.

Salin, T. "What's in a Name?" *Database Programming & Design* 3(3):55-58.

Song, I.-Y., M.Evans, and E.K.Park. 1995. "A Comparative Analysis of Entity-Relationship Diagrams." *Journal of Computer & Software Engineering* 3(4):427-59.

Storey, V.C.1991. "Relational Database Design Based on the Entity-Relationship Model." *Data and Knowledge Engineering* 7(1991):47-83.

Teorey, T.J., D.Yang, and J.P.Fry.1986. "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model". *Computing Surveys* 18(June): 197-221.

Von Halle, B.1997. "Digging for Business Rules." *Database Programming & Design* 8(11):11-13.

进一步阅读

Batini, C., S.Ceri, and S.B.Navathe.1992. *Conceptual Database Design: An Entity-Relationship Approach*. Menlo Park, CA: Benjamin/Cummings.

Keuffel, W.1996. "Battle of the Modeling Techniques." *DBMS* 9(August):83,84,86,97.

Moody, D.1996. "The Seven Habits of Highly Effective Data Modelers." *Database Programming & Design* 9(October): 57, 58, 60-62, 64.

Teorey, T.1999. *Database Modeling &Design*. 3rd ed.San Francisco,CA:Morgan Kaufman.

Tillman, G.1994. "Should You Model Derived Data?" *DBMS* 7(November): 88,90.

Tillman, G.1995. "Data Modeling Rules of Thumb." *DBMS* 8(August): 70,72,74,76, 80-82,87.

Web资源

- www.adtmag.com *Application Development Trends*杂志，关于信息系统开发实践的主流要出版物。
- www.businessrulesgroup.org 业务规则组，以前是GUIDE国际组织的一部分，负责业务规则的制定、支持标准。
- www.guide.org SHARE是一个志愿者组织，为IT业提供教育、网络方面的信息。SHARE和GUIDE已经合并，两者以前都是IBM公司客户的用户组。
- www.intelligententerprise.com *Intelligent Enterprise*杂志，数据库管理和相关领域的主流出版物。该杂志合并了以前的两本出版物*Database Programming&Design*和*DBMS*。

项目案例：山景社区医院

项目描述

假设在学习完数据库管理课程之后，你在暑假作为一名实习医生受雇于山景社区医院。你的第一项任务是参加一个三人开发小组来开发医院的高层E-R图。你可以通过和医院的许多管理人员以及员工进行交流，以确定医院的关键实体类型。很快，小组确定出以下实体类型：

- Care center（诊疗中心）——医院内部的诊疗中心，如产科和急救中心。每个诊疗中心有一个诊疗中心ID（作为标识符）和一个诊疗中心名称。
- Patient（病人）——来该医院看病的住院或门诊病人。每个病人有一个病人号（作为标识符）和其名字。
- Physician（医生）——为病人诊断的医生和负责为病人进行治疗的医生。每个医生有一个医生ID（作为标识符）和其名字。
- Bed（床位）——分配给一个住院病人的一张病床。每个病床有一个床位号（作为标识符），一个房间号和一个诊疗中心ID。
- Item（治疗项目）——用于治疗病人的任何内科或外科项目。每个治疗项目有项目号（标识符）、描述和单价。
- Employee（员工）——医院雇佣的人员。每个员工有一个名字和员工号。
- Treatment（治疗方案）——医生为病人执行的检查或治疗过程。每个治疗方案有一个治疗ID，它包括两部分：治疗方案号和治疗方案名称。

完成这些任务后，该小组下一步需记录有关联系的信息。

- 医院中的每个员工被分配在一个或多个诊疗中心工作。每个诊疗中心至少有一名员工，也可以有多名员工。医院记录每个员工在某个诊疗中心每周的工作小时数。
- 每个诊疗中心有且仅有一名护士被指定负责管理该诊疗中心。
- 因为存在门诊病人，所以病人可能被分配一个床位，也可能不分配床位。床位可能被分配给病人，也可能空闲。
- 病人被指定由且仅由一名医生负责。医生可以负责多个病人，也可能不负责任何病人。
- 医生可能对任意多个病人执行任意多项治疗，也可能不执行任何治疗。病人也可能由多个医生实施治疗方案。对于每项由一个特定的医生给特定病人实施的治疗，医院记录以下信息：治疗日期、治疗时间和治疗效果。
- 病人可以选择接受任意多项的治疗项目。一个治疗项目可对一个或多个病人实施，也可能没有病人选择接受该项治疗。对于一个病人所接受的一项治疗项目，医院记录以下信息：日期、时间、数量和总费用（可由多个治疗项目的合计得出）。

项目问题

1) 山景社区医院为什么选择实体-联系模型来为数据需求建模？还有其他哪些方法能够建立医院的信息需求模型？

2) 在山景社区医院的数据需求中是否有弱实体类型？如果有，列出这些实体名。

3) 上面列出的第3条联系中，说明“给住院病人分配床位，而不给门诊病人分配床位”。对于山景社区医院的E-R图设计，这条语句有什么意义？

4) 在描述山景社区医院数据需求的模型中，该医院本身是一个数据类型吗？

项目练习

1) 认真研究有关项目的描述。为了更好地理解山景社区医院的数据需求，你还能想出其他

需要医院有关人员给予答复的问题吗?

2) 为山景社区医院开发E-R图，并说明你在开发过程中所作的假设。

3) 在项目的描述文件中，给出一个名为Item的实体类型。根据项目练习2的结果，室内电视是一个病人需付费的项目，该项目是否也可用Item实体类型表示?

4) 假定属性床位号是一个复合属性，该属性由诊疗中心标识、病房号和病房中的床位号组成。设计中的某些部分将因为这个复合属性而发生变化，试根据这个因素重新回答项目练习2的有关部分。

5) 在加入项目练习4的条件后，项目练习2的答案发生变化，山景社区医院的E-R图也随之更新。现再增加一个新的假设：一个诊疗中心有多个病房，每个病房有多个收费项目，试据此假设更新E-R图。

6) 在你所作出的项目练习2的回答中，是否允许多名医生同时对一个病人进行治疗？如果不是，重新回答该问题的相关部分以满足这个条件。试根据这个条件作出你认为合理的任何假设。

7) 在你所作出的项目练习2的回答中，是否允许一名医生多次对一个病人执行同一治疗方案？如果不是，重新回答该问题的相关部分以满足这个条件。试根据这个条件作出你认为合理的任何假设。

第4章　增强型E-R模型和业务规则

4.1　学习目标

学完本章后，读者应该具备以下能力：

- 简明地定义下列每一个关键术语：增强型实体-联系(EER)模型、子类型、超类型、属性继承、概化、特化、完全特化规则、部分特化规则、不相交规则、交叠规则、子类型鉴别符、子类型/超类型层次、实体聚簇、结构断言、动作断言和导出事实。
- 理解在数据建模中，什么时候使用子类型/超类型联系。
- 定义子类型/超类型联系时，使用概化和特化技术。
- 在建模子类型/超类型联系的时候，说明完备性约束和不相交约束。
- 为实际业务情形开发一个超类型/子类型层次。
- 开发一个实体聚簇，以简化实体-联系图的表示。
- 命名不同种类的业务规则。
- 使用图形模型或结构化的语句，定义简单的操作约束。

4.2　引言

第3章所论述的基本的实体-联系模型，是在20世纪70年代中期首次提出的。它适合于建模大部分常见的业务问题，并得到广泛使用。但是，自那时起，业务环境已经发生很大的变化。业务联系更加复杂，因而，业务数据也更加复杂。例如，组织必须准备将市场分割成不同的部分，并定制相应的产品，而这些都对组织的数据库提出了更高的要求。

为了更好地处理这些变化，研究人员一直在增强E-R实体模型，使其能够更精确地表示当今业务环境中遇到的复杂数据。**增强型实体-联系**（EER）**模型**用来标识扩充的模型，即用这些新的建模概念扩充原来的E-R模型所得到的模型。

在EER模型中引入的最重要的概念是超类型/子类型联系。这种方法允许我们先建模一个一般的实体类型（称为*超类型*），然后，将其细分为几个特殊的实体类型（称为*子类型*）。例如，可以将实体类型CAR建模为超类型，将SEDAN、SPORTS CAR、COUPE等作为它的子类型。每一个子类型从其超类型中继承所有的属性，并可能加入自己的特有属性。为超类型/子类型联系的建模增加新的符号，大大提高了基本E-R模型的灵活性。

E-R，尤其是EER图，可能会变得很大而且很复杂，要用多页（或非常小的字体）来显示。一些商用数据库包括了数百个实体。对于大部分指定数据库需求的用户和管理人员来说，如果要理解数据库中他们最感兴趣的一部分，没有必要了解所有的实体、联系和属性。实体聚簇是将实体-联系数据模型的一部分转变成同样数据的更为宏观层次的视图的一种方法。实体聚簇是一种层次分解技术，它能够使实体-联系图更易读。通过将实体和联系分组，你能够对E-R图进行布局，以便集中关注给定的数据建模任务中最重要的模型的细节。

增强型E-R图用于捕获重要的业务规则，如超类型/子类型联系中的约束。但是，大部分组织使用大量的业务规则指导其行为。其中的很多业务规则不能用基本的E-R图，甚至是增强型E-R图表示。目前有望取得成果的研究集中于开发新的表示业务规则的方法，使最终用户经

过某些培训就能够定义他们自己的系统的规则。然后，这些规则会自动变成数据库管理系统强制执行的约束，使数据库保持一致、有效的状态。相对于以前的系统开发方法，该业务规则方法使得组织对变化的业务条件可以更快地做出反映。

4.3 超类型和子类型的表示

回忆一下第3章，实体类型是共享公共的性质或特征的实体的集合。构成实体类型的实体实例都是相似的，我们并不期望它们都是相同的。数据建模中的一个主要挑战是识别并清楚地表示几乎是相同的实体，这里所说的相同的实体就是共享共同性质的实体类型、但同时具有组织所感兴趣的一个或多个不同的性质。

正是由于这个原因，研究人员扩展了实体-联系模型，使其包含超类型/子类型联系。**子类型**（subtype）是对组织有意义的实体类型中实体的分组子群（非空子集——译者注）。例如，STUDENT是大学中的一个实体类型，STUDENT的两个子类型是GRADUATE STUDENT和UNDERGRADUATE STUDENT。在这个例子中，我们将STUDENT称为超类型。**超类型**（supertype）是一般的实体类型，与一个或多个子类型具有联系。

迄今为止，在构造实体-联系图的过程中，我们一直没有提到超类型和子类型。例如，再仔细考虑松谷家具公司的实体-联系模型（参见图3-22）。请注意，并不是所有顾客在一个销售区域内完成其业务。为什么会这样呢？一个可能的原因是有两种类型的顾客：全国性顾客和普通顾客，只有普通顾客在一个销售区域内完成交易。因此，在图3-22中，Does-business-in联系上的与CUSTOMER相邻的可选基数的理由不清楚。明确地画出顾客实体子类型及几个实体子类型有助于使E-R图更有意义。在本章的后面，我们将给出一个修订后的松谷家具公司的E-R图，该图给出几个使图3-22中含糊的地方更为清楚的EER符号。

4.3.1 基本概念和表示方法

图4-1给出了用于表示超类型/子类型联系的基本符号。超类型通过线与一个圆连起来，圆通过连线依次与每一个已定义的子类型连起来。每一个子类型与圆之间的连线上的U状符号表示子类型是超类型的一个子集，也表示了子类型/超类型联系的方向。

被所有实体共享的属性（包括标识符）都与超类型相关联。特定的子类型所独有的属性与该子类型相关联。随着本章的介绍，我们会在这些符号中加入其他的部分，为超类型/子类型联系提供更丰富的含义。

1. 例子

我们使用一个简单但是常用的例子来说明超类型/子类型联系。假定一个组织有三种基本类型的员工：钟点工、受薪员工和顾问。每种类型员工的一些重要属性列举如下：

- 钟点工：Employee_Number、Employee_Name、Address、Date_Hired、Hourly_Rate。
- 受薪员工：Employee_Number、Employee_Name、Address、Date_Hired、Annual_Salary、Stock_Option。
- 顾问：Employee_Number、Employee_Name、Address、Date_Hired、Contract_Number、Billing_Rate。

注意，所有的员工类型都有几个共同的属性：Employee_Number、Employee_Name、Address、Date_Hired。另外，每一个类型都有一个或多个与其他类型属性不同的属性（例如，只有钟点工才有Hourly_Rate属性）。如果在这种情形下开发一个概念的数据模型，你应该考虑下面三种情况：

1) 仅定义一个实体类型，即EMPLOYEE。尽管这种方式在概念上简单，但具有一个缺点，

即EMPLOYEE会包含三种类型员工的所有属性。例如，对一个钟点工的实例来说，不会应用Annual_Salary和Contract_Number这样的属性，这些属性将会是空值（null），或也不会用到。当运用到开发环境的时候，使用该实体类型的程序要处理很多变形，势必会非常复杂。

2) 为三个实体分别定义不同的实体类型。这种方法将不能利用员工的公共性质，并且当用户使用系统时，不得不仔细选择正确的实体类型。

3) 定义一个称为EMPLOYEE的超类型，它具有子类型HOURLYEMPLOYEE、SALARIED EMPLOYEE和CONSULTANT。这种方法利用了所有员工的公共性质，也能识别出每一种类型不同的性质。

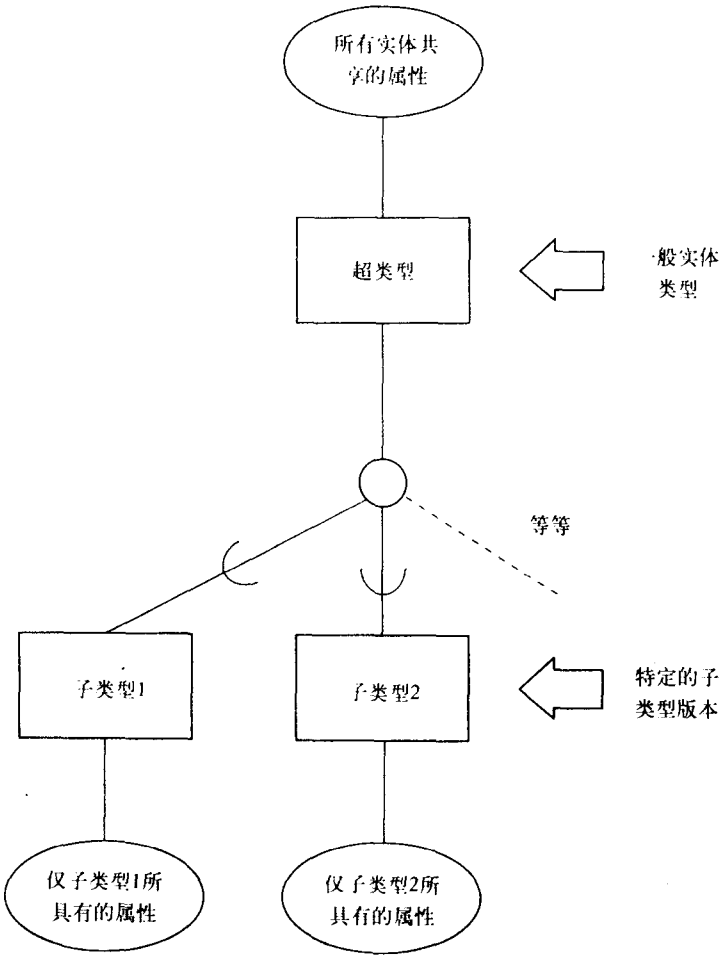


图4-1 子类型/超类型联系的基本符号

图4-2使用增强型实体-联系图符号，给出了EMPLOYEE类型及其三个子类型的表示。被所有员工共享的属性与EMPLOYEE实体类型相关联，每个子类型特有的属性仅被那个子类型所包含。

2. 属性继承

子类型本身是一种实体类型。子类型的实体实例表示超类型中同样的实体实例。例如，如果“Therese Jones”出现在CONSULTANT子类型中，那么这个人必定出现在超类型EMPLOYEE

中。结果，子类型中的实体不仅拥有自己的属性值，也同时拥有超类型中的属性值。

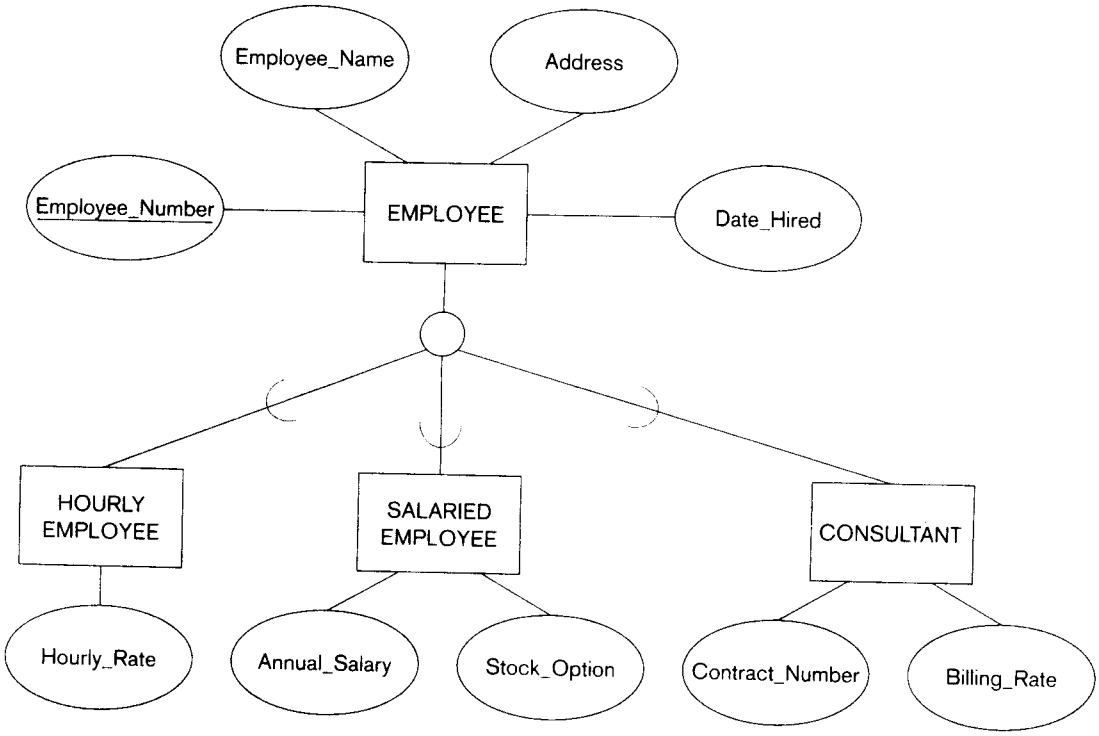


图4-2　具有三个子类型的员工超类型

属性继承（attribute inheritance）是指一种性质，即子类型实体继承超类型所有的属性值。这个重要性质使得子类型没有必要重复包含超类型的属性。例如，Employee_Name是EMPLOYEE的一个属性（见图4-2），而不是EMPLOYEE的子类型的属性。因此，员工的名字“Therese Jones”是从超类型EMPLOYEE继承而来的。但是，该员工的Billing_Rate是其子类型CONSULTANT的一个属性。

我们已经建立起这样的概念：一个子类型的实例必定是一个超类型的实例。反过来是否成立呢？也就是说，超类型的实例是否也是一个（或更多）子类型的实例呢？可能是这样，也可能不是这样，这要取决于具体的业务情形。在本章后面的部分，我们会讨论各种各样的可能性。

3. 什么时候使用超类型/子类型联系

是否使用超类型/子类型联系是数据建模者在每一种情形下必须做出的决定。当下面的任一种（或两种）情况出现时，应当考虑使用子类型。

1) 存在一些属性适用于一些（而不是全部）实体类型的实例。例如，图4-2的EMPLOYEE实体类型。

2) 子类型的实例参与仅属于该子类型的联系。

图4-3是使用子类型联系说明以上两种情况的例子。医院的实体类型PATIENT具有两个子类型：OUTPATIENT和RESIDENT PATIENT（主键是Patient_ID）。所有的病人都具有Admit_Date以及Patient_Name属性。同时，每一个病人都由RESPONSIBLE PHYSICIAN治疗，RESPONSIBLE PHYSICIAN制定病人的治疗计划。

每一个子类型都具有自己特有的属性。例如，门诊病人有Checkback_Date属性，住院病人有Date_Discharged属性。同样，住院病人具有特定的联系：给每一个病人分配床位（注意，这是强制联系）。每一个床位可能分配给病人也可能没有分配给病人。

前面我们讨论了属性继承的性质。因此，每个门诊病人和住院病人继承了超类型PATIENT的属性：Patient_ID、Patient_Name以及Admit_Date。图4-3也说明了联系继承的原理。OUTPATIENT和RESIDENT PATIENT也是PATIENT的实例。因此，每个病人都和RESPONSIBLE PHYSICIAN有Is_cared_for联系。

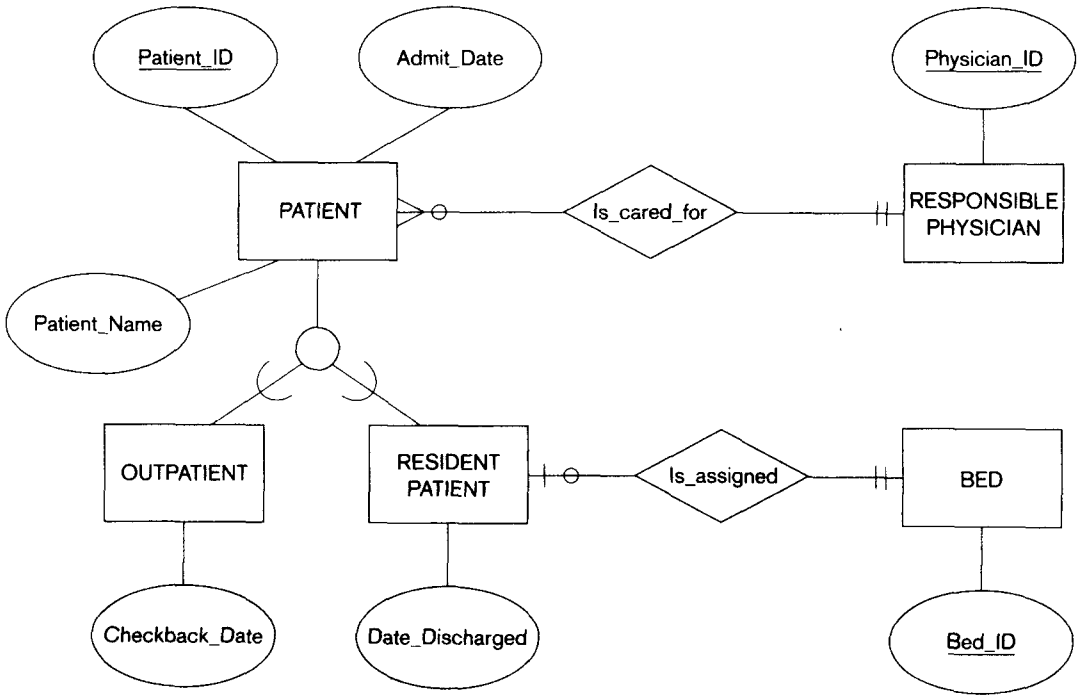


图4-3 医院中的超类型/子类型联系

4.3.2 特化和概化的表示

我们已经描述并说明了超类型/子类型联系的基本原理，包括“好”的子类型的特征。但是在开发现实世界的数据模型时，如何把握利用超类型/子类型联系的时机呢？有两种过程可以作为开发超类型/子类性联系的概念模型——概化和特化。

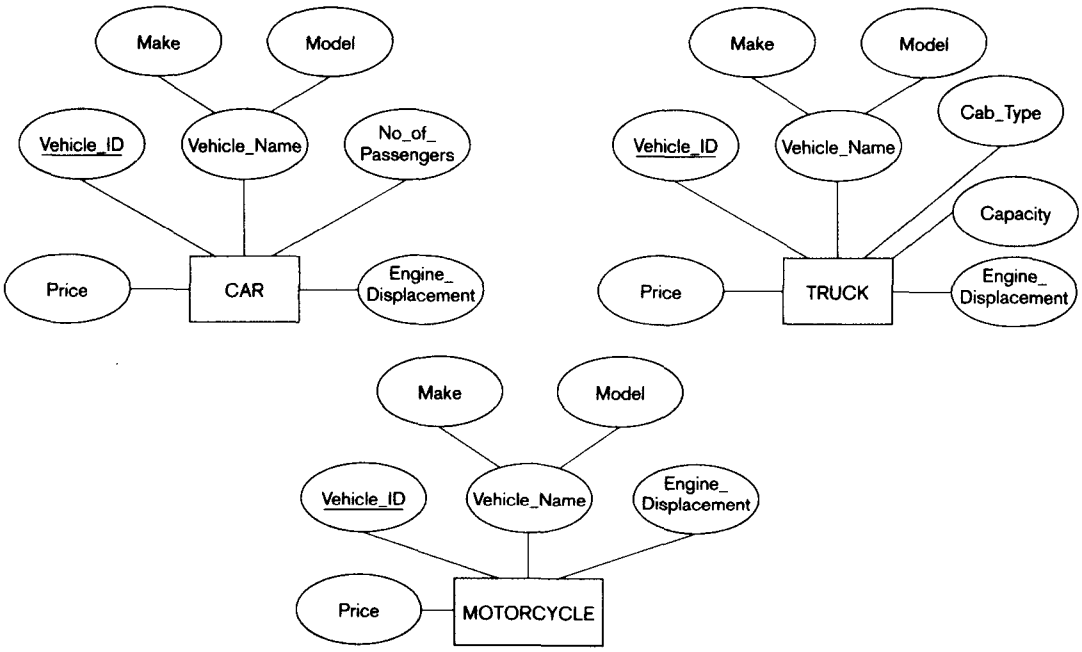
1. 概化

人类智力的独特之处在于具备将对象分类、概括其性质的能力、本能和经验。在数据建模过程中，**概化**（generalization）是从一些更为具体的实体类型中定义更一般实体类型的过程。因此，概化是自底向上的过程。

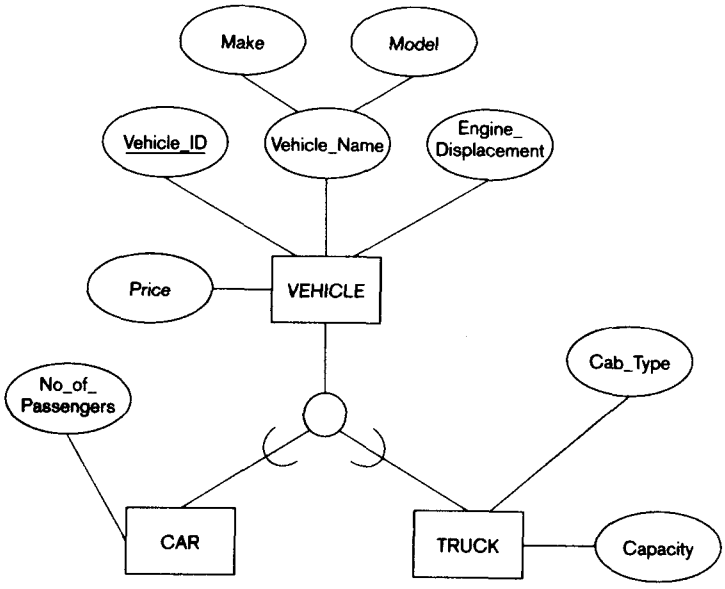
图4-4给出了概化的例子。在图4-4a中，定义了三个实体类型：CAR、TRUCK以及MOTORCYCLE。这时，数据建模人员想在E-R图中单独表示这些实体。但是，经过仔细检查，发现三个实体类型有一些共同的属性：Vehicle_ID（标识符）、Vehicle_Name(包括Make和Model两部分)、Price以及Engine_Displacement。这个事实（由于存在一个公共的标识符而得到支持）表明三个实体类型中的每一个都是更为一般的实体类型的一个特例。

这个更为一般的实体类型（命名为VEHICLE）以及导致的超类型/子类型联系在图4-4b中

给出。实体CAR具有特定的属性No_of_Passengers，TRUCK具有两个特定的属性：Capacity以及Cab_Type。因此，概化允许我们将实体类型及其公共属性组合在一起，同时保持每一个子类型特定的属性。



a）三个实体类型：CAR、TRUCK和MOTORCYCLE



b) 超类型VEHICLE的概化

图4-4　概化的例子

注意，实体类型MOTORCYCLE并没有包含在联系中。仅仅是疏忽吗？不是，这是有意的，

因为它不满足前面讨论的子类型条件。比较图4-4中的a和b，读者会发现MOTORCYCLE的属性正是所有VEHICLE的共有属性；摩托车没有自己特有的属性。再者，MOTORCYCLE与其他实体类型之间没有联系。因此，没有必要创建MOTORCYCLE子类型。

没有MOTORCYCLE子类型的事实表明，一定存在一个VEHICLE的实例不是任何子类型的实例。我们将在说明约束的那一节中讨论这种类型的约束。

2. 特化

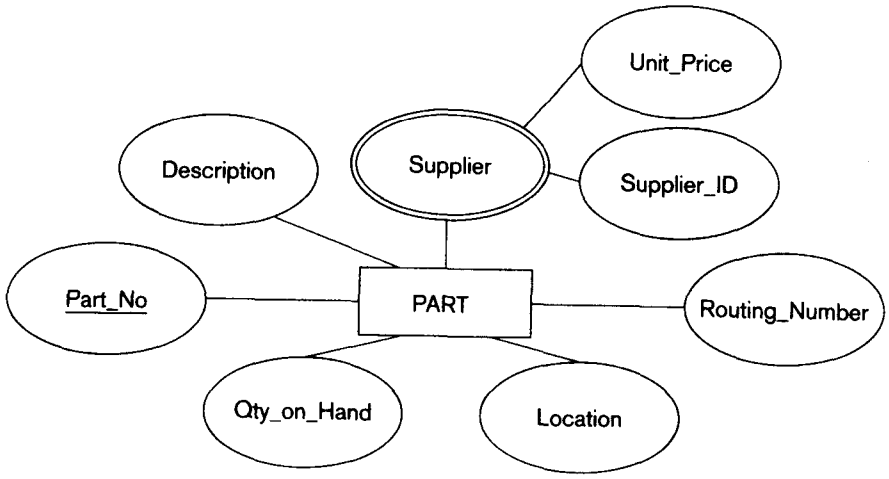
如前所述，概化是自底向上的过程，那么，特化与概化的方向相反，是自顶向下的过程。假如已经定义了实体类型及其属性，**特化**（specialization）是从超类型定义一个或多个超类型的子类型，并形成超类型/子类型联系的过程。每一个子类型都是基于一些不同的特征而形成，如子类型特有的属性或联系。

图4-5给出了特化的一个例子。图4-5a展示了名为PART的实体类型以及它的一些属性。其标识符是Part_No，其他属性包括Description、Unit_Price、Location、Qty_on_Hand、Routing_Number以及Supplier（最后一个属性是多值的，因为一个零件对于不同的供货商可能有不同的单价）。

在与用户讨论的过程中，我们发现零件有两种可能的来源：一些由内部加工，而另一些则从外部的供货商处购买。而且，我们发现一些零件既从外部购买，又有内部加工的。在这种情况下，零件的选择依赖于诸如加工能力、零件单价等因素。

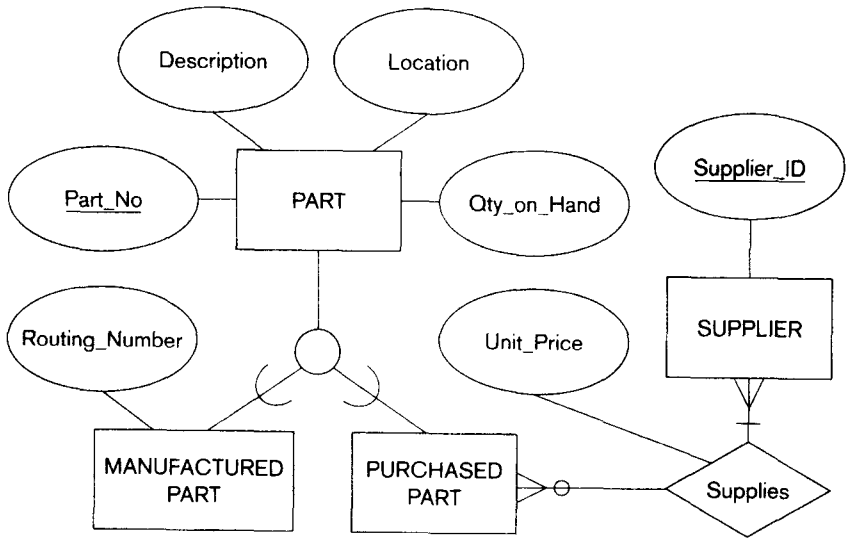
图4-5a中的一些属性适用于所有的零件，而不论其来源如何。但是，其他一些属性则和零件的来源有关。Routing_Number只适用于加工的零件，而Supplier_ID和Unit_Price只适用于购买的零件。这些因素表明，应当通过定义子类型MANUFACTURED PART和PURCHASED PART特化PART（见图4-5b）。

在图4-5b中，Routing_Number与MANUFACTURED PART相关联。数据建模人员起初计划将Supplier_ID和Unit_Price与PURCHASED PART相关联。但是，经过与用户深入讨论，他建议在PURCHASED PART和SUPPLIER之间创建一个新联系。该联系（在图4-5b中命名为Supplies）使用户更容易将购买的零件和其供货商关联起来。注意，属性Unit_Price现在与联系Supplies关联，因此，零件的单价会因供货商的不同而发生变化。在该例中，概化提供了表示问题领域更恰当的方式。



a) 实体类型PART

图4-5 特化的例子



b) 特化为MANUFACTURED PART和 PURCHASED PART

图4-5　（续）

3. 将概化和特化结合起来

特化和概化都是开发超类型/子类型联系的有用技术。在特定的时刻用什么样的技术取决于多个因素，如问题领域的本质，先前的建模工作，个人喜好。读者应该准备同时使用两种技术，并根据前面提到的因素，交替使用这两种技术。

4.4 指定超类型/子类型联系之间的约束

迄今为止，我们已经讨论了超类型/子类型联系的基本概念，并介绍了一些基本的符号来表示这些基本概念。我们也论述了概化和特化的过程，以便帮助数据建模者把握利用这些联系的机会。本节将介绍表示超类型/子类型联系上的约束的其他符号。这些约束使我们能够捕获一些应用这些联系的重要业务规则。本节所论述的两个最重要约束类型是完备性约束和不相交约束（Elmasri和Navathe, 1994）。

4.4.1 指定完备性约束

完备性约束（completeness constraint）解决以下问题：是否一个超类型的实例必须至少是一个子类型的实例。完备性约束有两个可能的规则：完全特化和部分特化。**完全特化规则**（total specialization rule）说明超类型的每一个实体实例必须是联系中一些子类型的一个实例。**部分特化规则**（partial specialization rule）说明了超类型的实体实例可以不属于任何子类型。我们用本章前面的例子来说明这两个规则（见图4-6）。

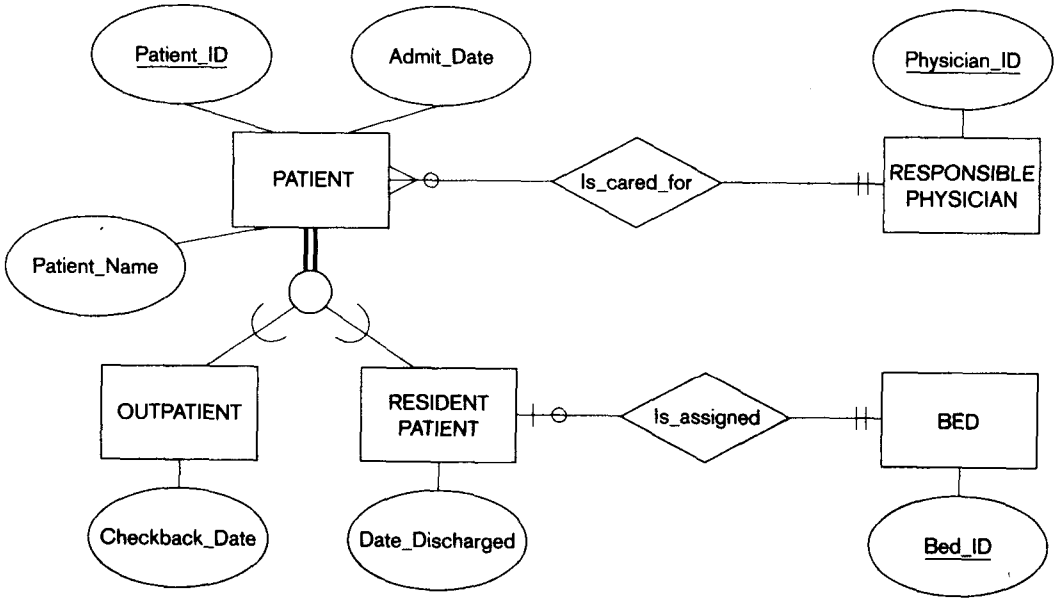
1. 完全特化规则

图4-6a重复了PATIENT例子（见图4-3）并引入完全特化规则的符号。在这个例子中，业务规则如下：一个病人必须是门诊病人或是住院病人（在这个医院中没有其他类型的病人）。完全特化规则使用从PATIENT实体类型连到圆的双线表示。

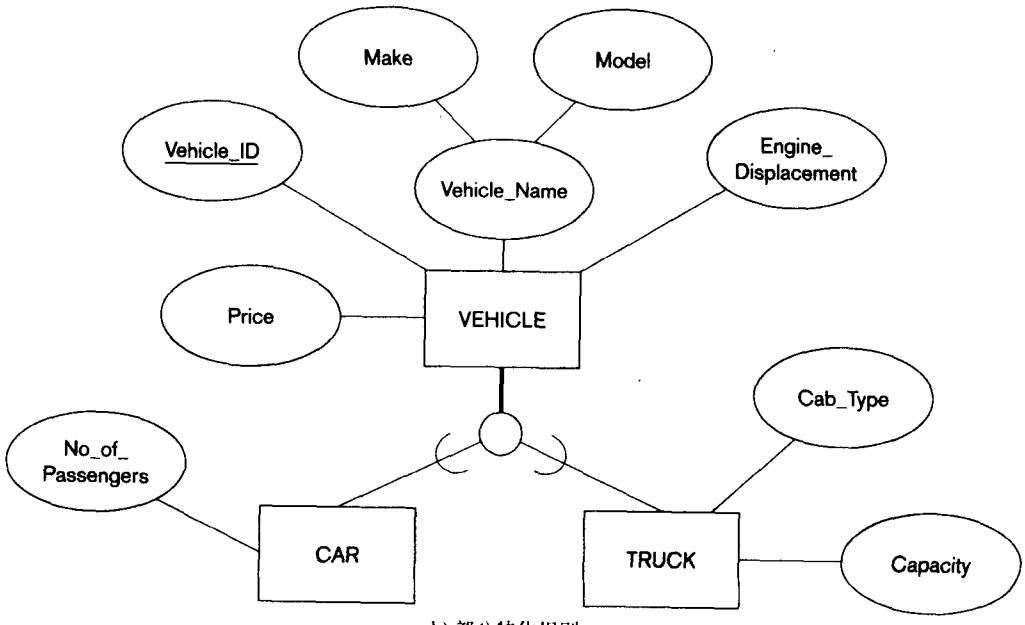
在这个例子中，每次在超类型中插入一个新的PATIENT的实例，就会在OUTPATIENT或RESIDENT PATIENT中插入相应的实例。如果在RESIDENT PATIENT中插入实例，则会创建联系Is_assigned的实例，以便给这个病人分配一个床位。

2. 部分特化规则

图4-6b重复了图4-4中VEHICLE及其子类型CAR和TRUCK的例子。回忆一下，在这个例子中，摩托车是一种交通工具，但是并没有将它表示为数据模型中的一个子类型。因而，如果一个交通工具是小轿车，那么它必是CAR的实例；如果是卡车，那么它必是TRUCK的实例。但是，如果一个交通工具是摩托车，那么它不能作为任何子类型的实例。这是部分特化的例子，由从超类型VEHICLE到圆的单线表示。



a) 完全特化规则



b) 部分特化规则

图4-6 完备性约束示例

4.4.2 指定不相交约束

不相交约束（disjointness constraint）解决这样的问题：一个超类型的实例是否可以同时是两个（或多个）子类型的实例。不相交约束有两个可能的规则：不相交规则和交叠规则。**不相交规则**（disjoint rule）指定如果一个（超类型）的实体实例是一个子类型的实例，那么它不能同时是任何其他子类型的实例。**交叠规则**（overlap rule）指定一个实体实例可以同时是两个（或多个）子类型的实例。图4-7给出了每一种规则的例子。

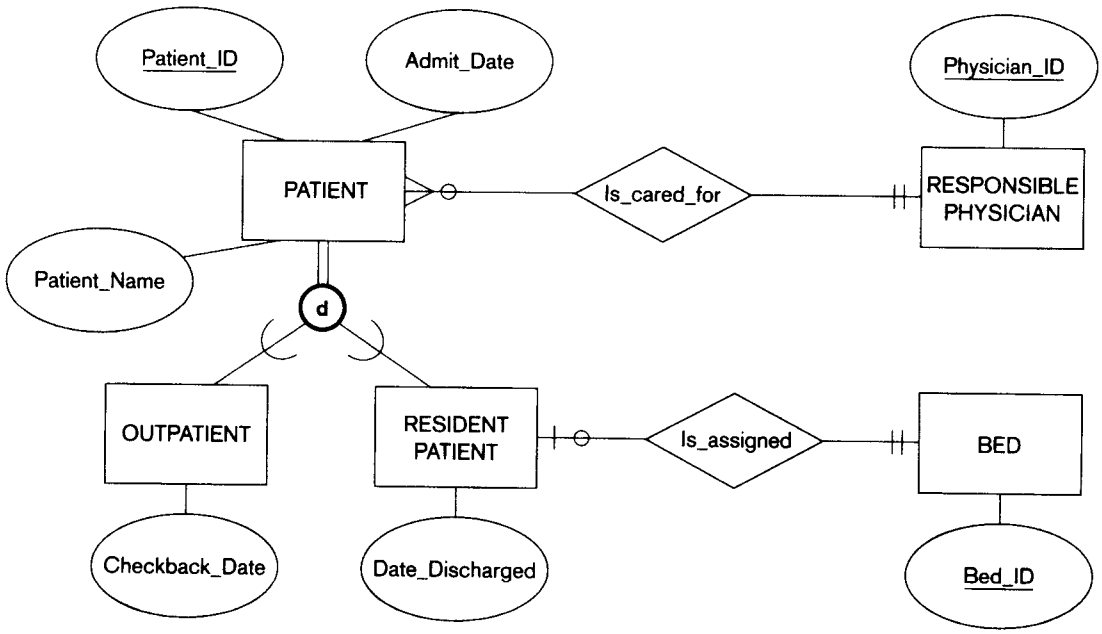
1. 不相交规则

图4-7a显示了图4-6a中的PATIENT的例子。在这种情况下，其业务规则如下：在任何给定时间，对于任何一个病人，要么是门诊病人，要么是住院病人，但不能两者都是。这样的业务规则就是不相交规则，它用写在连接超类型和子类型的圆中的字母“d”表示。注意，在这个图中，PATIENT的子类会随时间而变化，但是，在任何给定的时间，一个PATIENT只能是一种类型。

2. 交叠规则

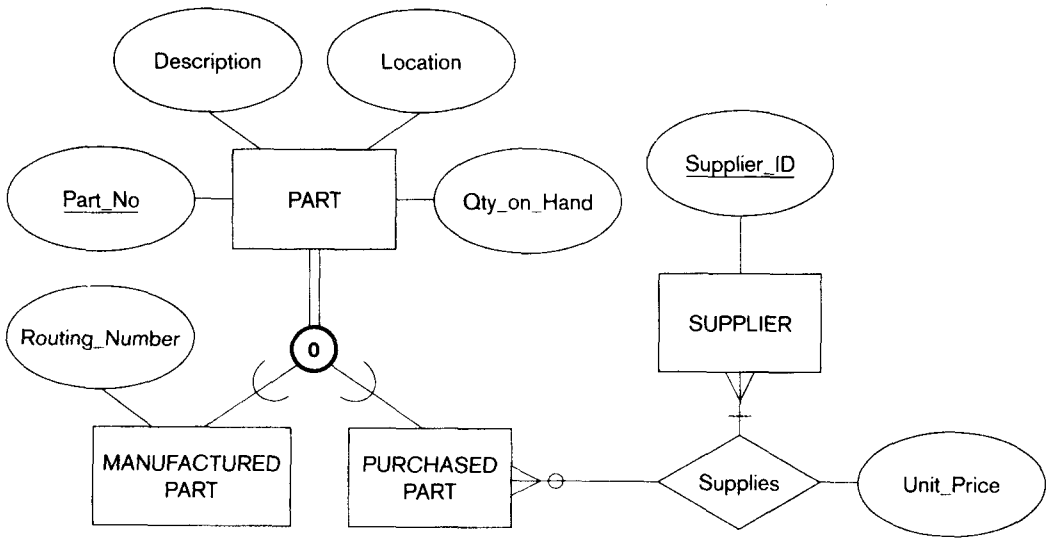
图4-7b说明了实体类型PART及其两个子类型MANUFACTURED PART和PURCHASED PART（见图4-5b）。回忆一下对这个例子所作的讨论，一些零件可以同时是加工零件和外购零件。我们需要对这句话再进行一下说明。在这个例子中，PART的一个实例是特定的零件号（即零件的型号），而不是某个零件（零件通过标识符说明，即Part_No）。例如，考虑零件号4000。在某时刻，这种零件可能有250个，其中，100个是加工的零件，余下的150个是外购的零件。在这种情况下，并不需要关注个别的零件。当需要关注单个零件时，给每一个零件分配一个序列号标识符，并依据它是否存在，将其数量赋为1或0。

将字母o放在圆内来表示交叠规则，如图4-7b所示。注意，该图也说明了完全特化规则（用双线表示）。因此，零件可以是加工零件或外购零件，也可能既是加工零件又是外购零件。



a) 不相交规则

图4-7　不相交约束例子



b) 交叠规则

图4-7 （续）

4.4.3 定义子类型鉴别符

假如有一个子类型/超类型联系，考虑这样的问题：插入一个新的超类型实例。这个实例将插入哪个子类型中呢？我们已经讨论了适用于这种情况的各种规则，我们需要一个简单的机制来实现这些规则（如果存在的话）。通常使用子类型鉴别符来完成这样的任务。**子类型鉴别符**（subtype discriminator）是超类型的一个属性，其值决定了目标类型和子类型。

1. 不相交子类型

图4-8给出使用子类型鉴别符的例子。该例子说明了图4-2介绍的EMPLOYEE超类型及其子类型。注意，下面的约束已经加入该图：完全特化和不相交子类型。因此，每一个员工必须是钟点工、受薪员工或顾问的一种。

超类型中已加入了一个新的属性（Employee_Type）作为子类型鉴别符。当超类型中加入新的员工时，将该属性的值编码为下面三个值中的一个："H"（用来表示钟点工）、"S"（用来表示受薪员工）或"C"（用来表示顾问）。根据这样的编码，将实例指派为合适的子类型。

图4-8也说明了用来表示子类型鉴别符的符号。表达式"Employee_Type="（放在条件语句的左边）放在从超类型到圆的连线的旁边，选择合适子类型的属性值（在该例中，其值是"H"、"S"或"C"）放在指向子类型的连线旁边。因此，对于这个例子，条件"Employee_Type =S"使得实体实例插入到SALARIED EMPLOYEE子类型中。

2. 交叠子类型

当子类型交叠时，必须对子类型鉴别符作稍许修改。其原因是对于一个给定的超类型实例，可能需要在多个子类型中创建实例。

图4-9说明了PART及其交叠子类型的例子。PART中已经加入了名为Part_Type的新属性。Part_Type是一个由Manufactured? 和Purchased? 组成的复合属性，其中的每一个属性都是布尔变量（即只能具有值"Y"或"N"）。当PART中加入一个新的实例时，Part_Type的组成部分的值如下：

部件零件的类型	Manufactured?	Purchased?
仅加工	"Y"	"N"
仅购买	"N"	"Y"
购买且加工	"Y"	"Y"

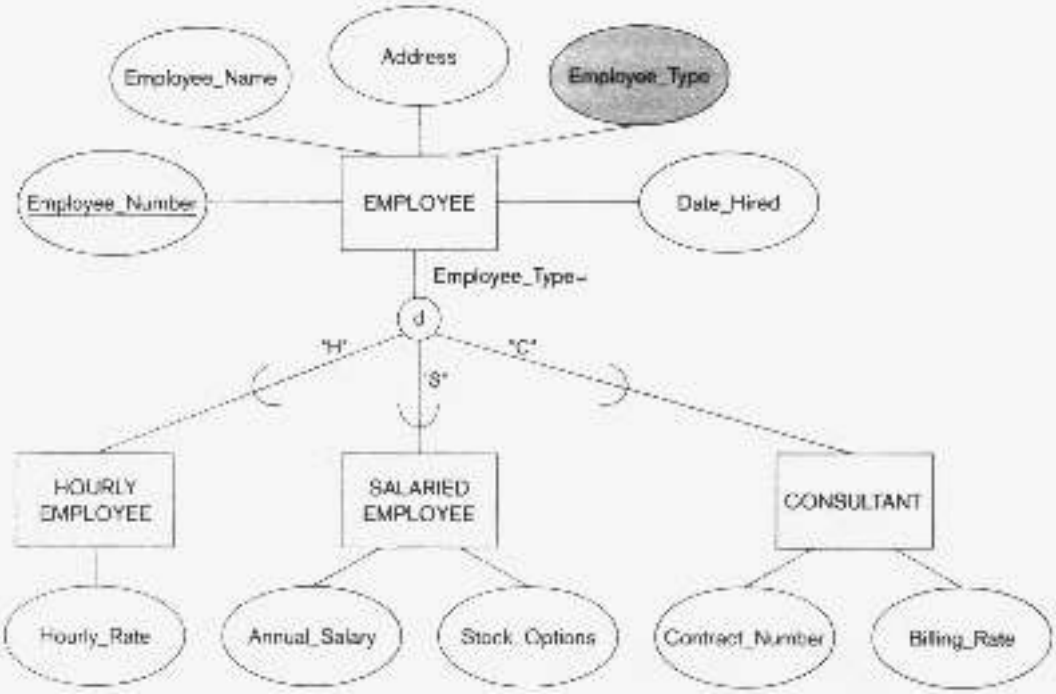


图4-8　引入子类型鉴别符（不相交规则）

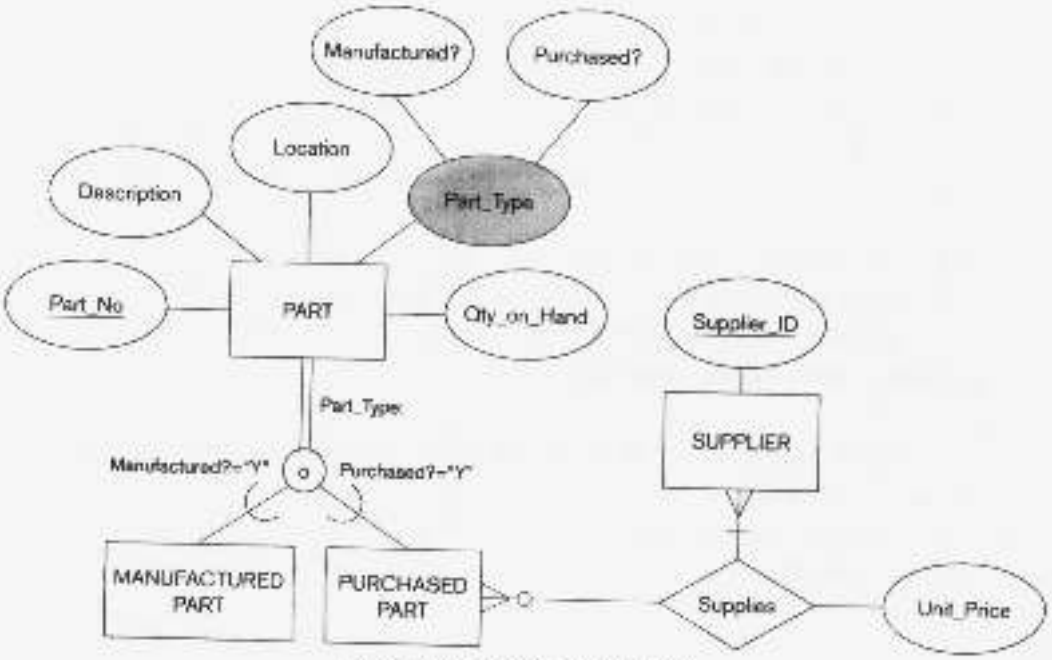


图4-9　子类型鉴别符（交叠规则）

图4-9说明了为这个例子指定子类型鉴别符的方法。注意，该方法可以用于任何数目的交叠子类型。

4.4.4 定义超类型/子类型层次

本章已经讨论了一些超类型/子类型联系的例子。对于这些例子中的任何子类型，都有可能在其上定义其他子类型（在这种情况下，子类型变为新定义的子类型的超类型）。**超类型/子类型层次**（supertype/subtype hierarchy）是指超类型和子类型层次的层次安排，其中，每一个子类型只有一个超类型（Elmasri和Navathe，1994）。

本节给出了一个超类型/子类型层次的例子（见图4-10）。这个例子包含了本章迄今使用的大部分概念和符号。同时也给出了可以在大多数建模情况下使用的方法学（基于特化）。

1. 例子

假定要求你构建一个大学人力资源的模型。使用特化（自顶向下的方法），你可能按照下面过程进行：从层次的顶层开始，先建模最一般的实体类型。在这种情况下，最一般的实体类型是PERSON。列出PERSON所有的属性。图4-10给出的属性有：SSN（标识符）、Name、Address、Gender和Date_of_Birth。有时也将顶层的实体类型称为根（root）。

下一步，定义根的所有主要子类型。在该例中，PERSON的子类型有三个：EMPLOYEE(在大学工作的人)、STUDENT(在大学上课的人)、ALUMNUS(已经毕业的人)。假定没有该大学感兴趣的其他类型的人，完全特化规则的应用如图所示。一个人可能属于多个子类型（例如，EMPLOYEE和ALUMNUS），因此，使用了交叠规则。注意，交叠规则允许任何交叠（一个PERSON可能同时是任何两个或所有三个子类型）。如果不允许进行某些组合，那么意味着要开发更为精细的超类型/子类型层次以消除不允许的结合。

图中表明了特定的属于每一个子类型的属性。因而每一个EMPLOYEE的实例都有Date_Hired和Salary的值。Major_Dept是STUDENT的一个属性，而Degree（具有组件Year、Designation和Date）是ALUMNUS的一个属性。

下一步是评估任何已定义的子类型需要作进一步的特化处理。在这个例子中，将EMPLOYEE分为两个子类型：FACULTY和STAFF。FACULTY具有特定的属性Rank，而STAFF具有特定的属性Position。注意，在这个例子中，子类型EMPLOYEE变成了FACULTY和STAFF的超类型。因为除了教师和职员外，可能会有其他员工类型（如助教），这意味着存在部分特化规则。但是，一个员工不可能同时是教师和职员。因此，在圆中应说明不相交规则（用“d”表示）。

为STUDENT定义了两个子类型：GRADUATE STUDENT和UNDERGRAD STUDENT。UNDERGRAD STUDENT具有属性Class_Standing，GRADUATE STUDENT具有属性Test_Score。注意，指定了完全特化规则和不相交规则，你应该能够叙述这些约束的业务规则。

2. 超类型/子类型层次的小结

注意，图4-10所示层次中包含关于属性约束的两个特性。

1) 在该层次中，尽可能将属性分配在最高逻辑层。例如，因为SSN（即社会安全号）适用于所有的人，所以将其分配到根上。另一方面，Date_Hired属性只适用于员工，因此，仅分配给EMPLOYEE作为属性。这种方法确保尽仅可能多的子类型共享属性。

2) 在该层次中，较低层的子类型不仅从它的直接超类型中继承属性，而且从层次中较高层中的所有超类型中继承属性。例如，FACULTY的一个实例具有下列属性值：SSN、Name、Address、Gender和Date_of_Birth（从PERSON中继承）、Date_Hired和Salary（从EMPLOYEE中继承）以及Rank（从FACULTY中继承的属性）。

在本章最后的案例研究中，会要求你使用本节介绍的过程，开发一个关于山景社区医院的增强型E-R图。

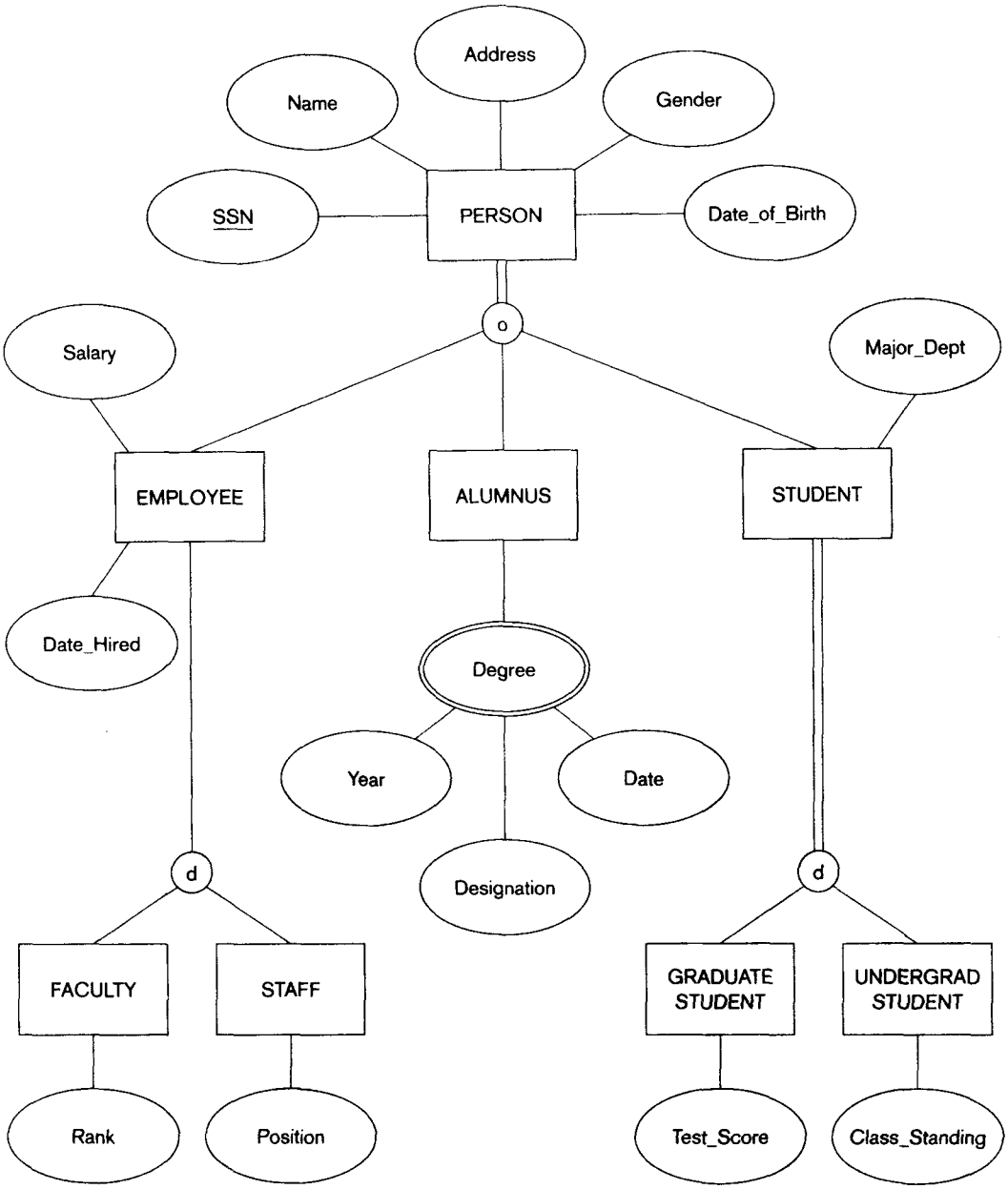


图4-10　超类型/子类型层次的例子

4.5　增强型EER建模示例：松谷家具公司

第3章给出了松谷家具公司E-R图的例子（图4-11重复了该图）。在研究该图之后，可用一些问题来帮助你澄清实体和联系的含义。其中三个问题是：

1）为什么所有的顾客不在一个或多个销售区域内进行交易？

2) 为什么一些员工没有管理其他员工，为什么他们没有受其他员工的管理？为什么一些员工没有在加工中心工作？

3) 为什么并不是所有的供货商都为松谷家具公司提供原材料？

可能还有其他问题，但我们将关注这三个问题，来说明如何用超类型/子类型联系传达更为特定的（语义更为丰富的）数据模型。

对上面问题进行一些研究后，我们发现适用于松谷家具公司完成业务的如下业务规则：

1) 有两种类型的顾客：普通顾客和全国性顾客。只有普通顾客在销售区内进行交易。当且仅当至少一个普通顾客与一个销售区关联时，该销售区域才会存在。每一个全国性顾客与一个账户经理相关联。存在这种可能性：一个顾客既是普通顾客又是全国性顾客。

2) 存在两种特定类型的员工：管理人员和工人。仅工人在加工中心工作，管理人员管理工人。也存在管理人员和工人之外的其他类型员工。工人可以晋升为管理人员，这时，该员工不再是工人。

3) 松谷家具公司会关注很多不同的销售商，并不是所有的销售商都曾为公司提供原材料。一旦销售商变为原材料的正式供货商，便与一个合同号相关联。

上述业务规则用于将图4-11修改为图4-12（除了与所发生的变化有本质关系的属性外，该图省略了大部分的属性）。规则1意味着从CUSTOMER到REGULAR CUSTOMER和NATIONAL ACCOUNT CUSTOMER，存在着完全、交叠特化。CUSTOMER的复合属性Customer_Type用于说明一个顾客实例是普通顾客、全国性顾客还是两者都是。因为，只有普通顾客在销售区域内完成业务，因此，只有普通顾客涉及Does_business_in联系；并且因为一个销售区域要存在，必须至少有一个普通顾客，所以普通顾客旁边的最小基数是1。

规则2意味着存在一个部分的、不相交的特化，将EMPLOYEE特化为MANAGEMENT EMPLOYEE和UNION EMPLOYEE。EMPLOYEE的属性Employee_Type用来区分两种特定类型的员工。特化之所以是部分的，是因为除了这两种类型的员工外，还有其他类型的员工。仅工人涉及到Works_in联系，但是，所有的工人在同一个加工中心工作，因此，WORK CENTER的Works_in的最小基数是强制的。由于一个员工不能同时是管理人员和工人，因此，特化是不相交的。

规则3意味着有一个VENDOR到SUPPLIER的部分特化，因为仅有一些销售商会变成供货商。供货商（而不是销售商）具有合同号。由于VENDOR只有一个子类型，因此没有必要指定不相交或交叠规则。由于所有的供货商供应原材料，RAW MATERIAL在Supplies联系中的最小基数是1。

以上例子表明，一旦理解了实体的概化/特化，就知道如何将E-R图转换为EER图。在数据模型中不仅可以包括超类型和子类型实体，而且还可加入额外的属性（如鉴别符属性），改变最小基数（从可选到强制），以及将联系从超类型移向子类型。

这时刚好强调一下前面讨论过的关于数据建模的一个要点。数据模型是组织所需要数据的概念图。数据模型与实现的数据库元素并不是一一映射关系。例如，读者熟悉的数据库产品Access，尽管SUPPLIER是实体类型而Supplies是联系，但两者在Access数据库中都以表的形式出现。现在，这些细节并不重要。现在的目的是解释控制数据的所有规则，并不是解释如何有效地存储或访问要处理的信息。在随后的章节中，当介绍数据库设计和实现时，将强调技术和效率的问题。

尽管图4-12中的EER图澄清了一些问题，并使图4-11中的模型更为清晰，但它对一些人来讲依然难以理解。一些人并不对所有的数据类型感兴趣，而且一些人可能不需要了解E-R

图中的所有细节。下一节主要介绍如何简化完整的、清晰的数据模型，以呈现给特定的用户组成管理人员。

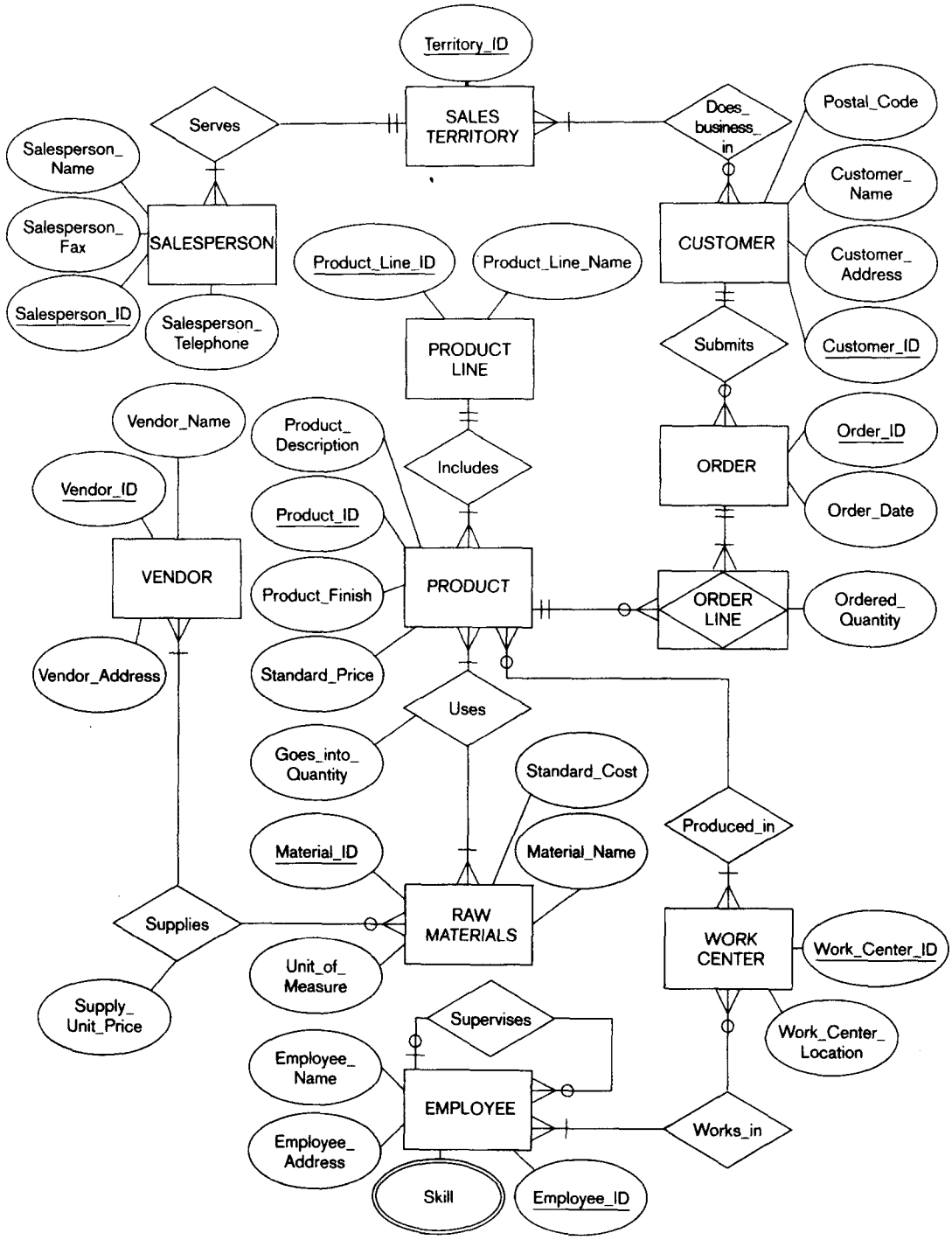


图4-11　松谷家具公司的E-R图

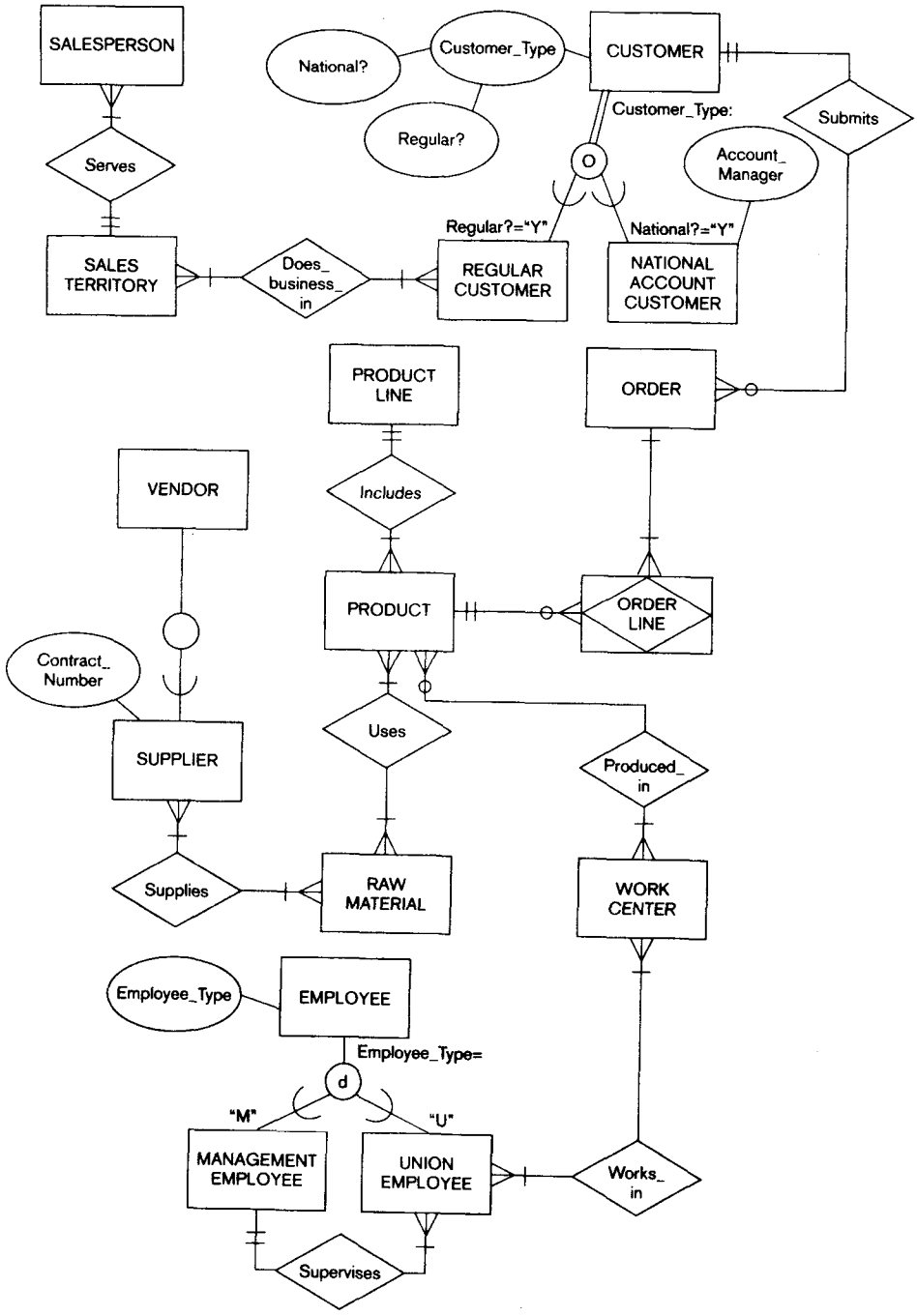
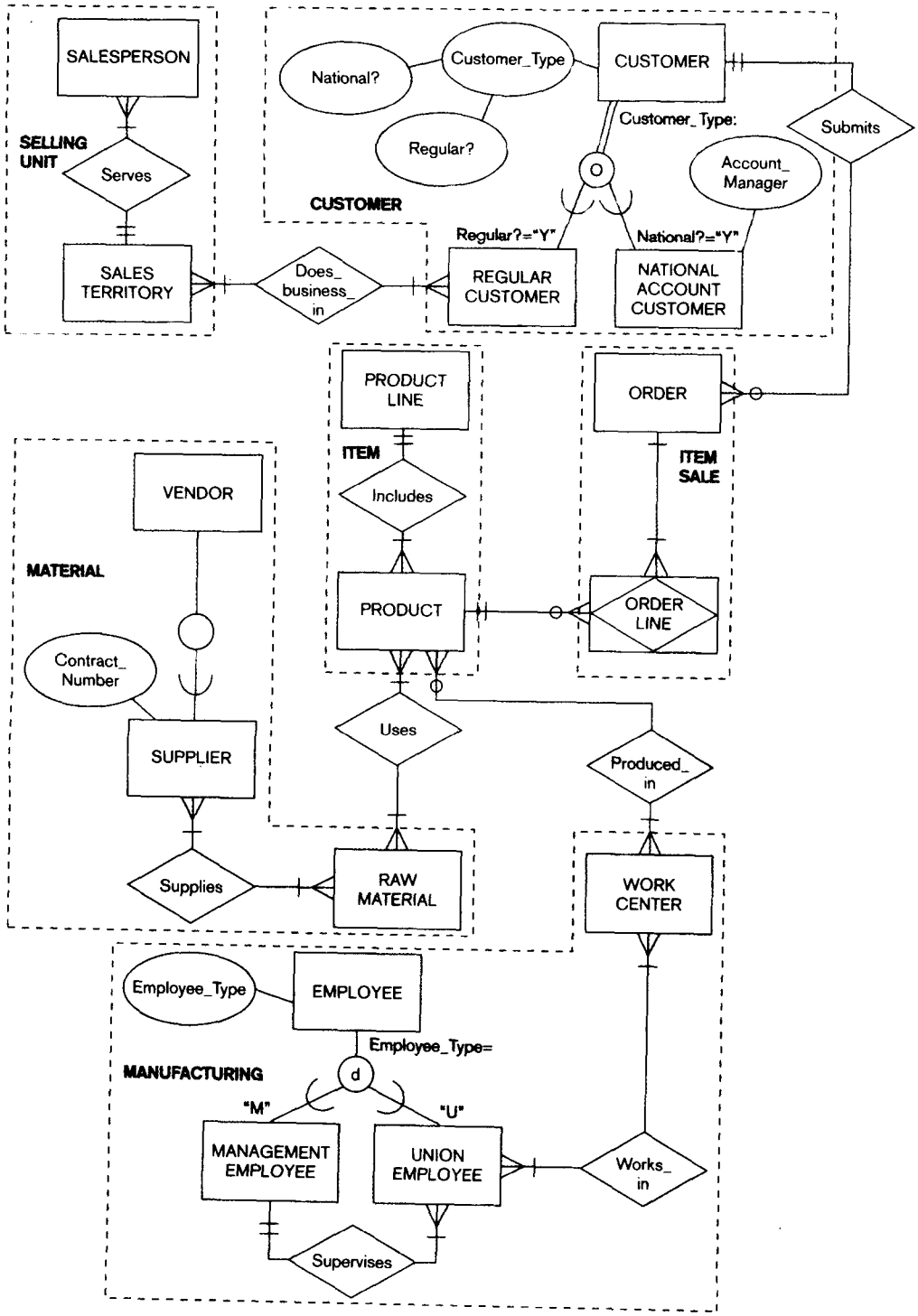


图4-12 松谷家具公司的EER图

4.6 实体聚簇

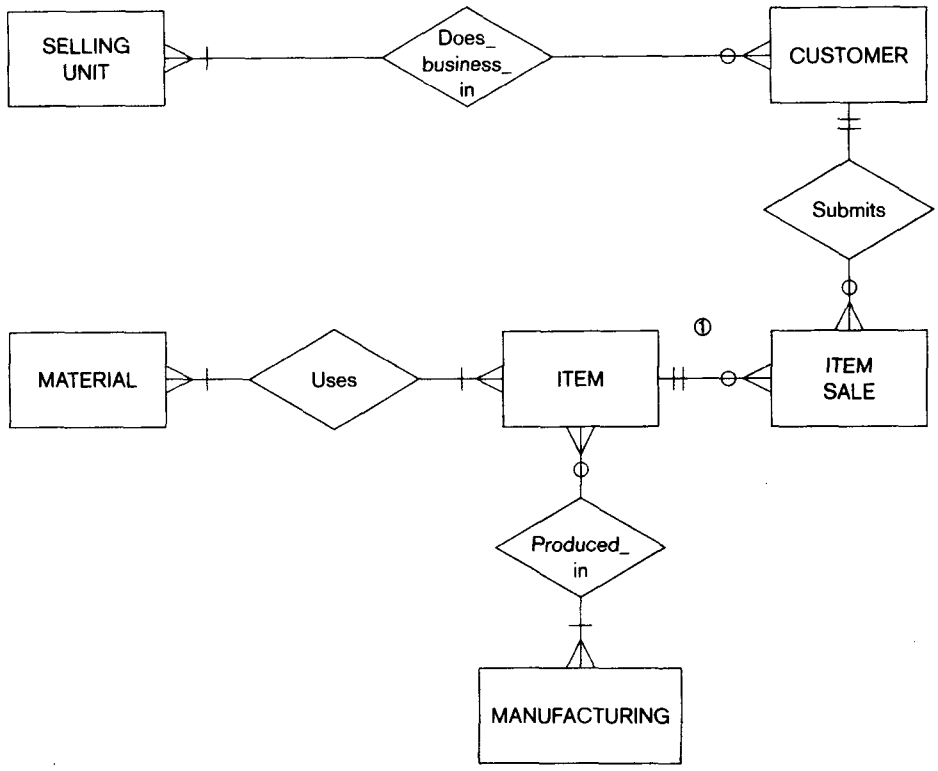
一些企业级的信息系统有1000多个实体类型和联系，如何将这样一个庞大的组织数据图呈现给开发人员和用户？一种方法是创建多个E-R图，每一个图展示数据模型不同片段（可能交

叠）或不同子集合的细节（例如，不同的片段适用于不同的部门、信息系统应用、业务处理或分公司）。这种方法虽然行得通，但是不能展示整个图。



a) 可能的实体聚簇

图4-13　松谷家具公司的实体聚簇



注：①虽然在4-13a中没有任何说明，但此处可以插入表示联系的菱形和名称（如Ordered_on）。

b) 实体聚簇的EER图

图4-13 （续）

实体聚簇（Teorey, 1999）是展现庞大且复杂的组织的数据模型的一个有用的方法。**实体聚簇**（entity cluster）是将一个或多个实体类型及其相关属性组成单个抽象实体类型。因为实体聚簇的行为和实体类型相似，所以实体聚簇和实体类型可以进一步组成更高层的实体聚簇。实体聚簇是一个将数据模型的宏观视图分解成粒度更细的视图的层次分解，最终形成充分的、详细的数据模型。

图4-13说明了图4-12所示的松谷家具公司数据模型的实体聚簇的一个可能的结果。图4-13a在可能的实体聚簇周围画上虚线，来表示完整的数据模型。图4-13b给出了将详细的EER图转换为仅有实体聚簇及其联系的EER图（EER实体联系图可能包含实体聚簇和实体类型，但是该图仅包含实体聚簇）的最终结果。该图中，实体聚簇的含义如下：

- SELLING UNIT 表示SALESPERSON和SALES TERRITORY实体类型和Serves联系。
- CUSTOMER 表示CUSTOMER实体超类型、其子类型以及超类型和子类型之间的联系。
- ITEM SALE 表示ORDER实体类型和关联实体ORDER LINE以及它们之间的联系。
- ITEM 表示PRODUCT LINE和PRODUCT实体类型以及Includes联系。
- MANUFACTURING 表示WORK CENTER、EMPLOYEE超类型实体、它的子类型以及*Works-in*联系和*Supervises*联系、超类型及其子类型之间的联系（图4-14说明了实体聚簇MANUFACTURING到其组成部分的展开）。
- MATERIAL 表示RAW MATERIAL和VENDOR实体类型、SUPPLIER子类型、Supplies

联系以及VENDOR和SUPPLIER之间的超类型/子类型之间的联系。

图4-13和4-14中的实体联系图可以向人们解释装配过程和支持这部分业务所需要的部分的信息的细节。例如，库存管理人员可以通过图4-13了解到，MANUFACTURING的数据与ITEM的数据相关（通过Produced_in联系）。另外，图4-14说明了涉及加工中心和员工在内的生产过程的细节。库存管理人员可能没有必要了解某些细节，例如，嵌入SELLING UNIT实体聚簇的销售结构。

图4-13中的实体聚簇可以通过下列方法得到：1）通过抽取超类型及其子类型（见CUSTOMER实体聚簇）；2）结合直接相关的实体类型和它们之间的联系（见SELLING UNIT、ITEM、MATERIAL和MANUFACTURING实体聚簇）。也可以通过结合强实体及其相关的弱实体类型得到实体聚簇（本图没有相关说明）。由于实体聚簇是层次的，所以如果需要的话，可以将SELLNG UNIT和CUSTOMER实体聚簇以及联系Does_business_in结合起来，画出另一个实体联系图，其原因是它们是直接相关的实体聚簇。

实体聚簇应该关注用户、开发人员和管理人员感兴趣的领域。将哪些实体类型和联系组成实体聚簇取决于你的目的。例如，在关于松谷家具公司数据模型的实体聚簇的例子中，可以将ORDER实体类型放进CUSTOMER实体聚簇，将ORDER LINE实体类型放进ITEM实体聚簇。这样的重组可以去掉ITEM SALE聚簇，因为没有任何一组人员对这个聚簇感兴趣。同样，针对于整个数据模型不同的方面，可以构造几个不同的实体聚簇。

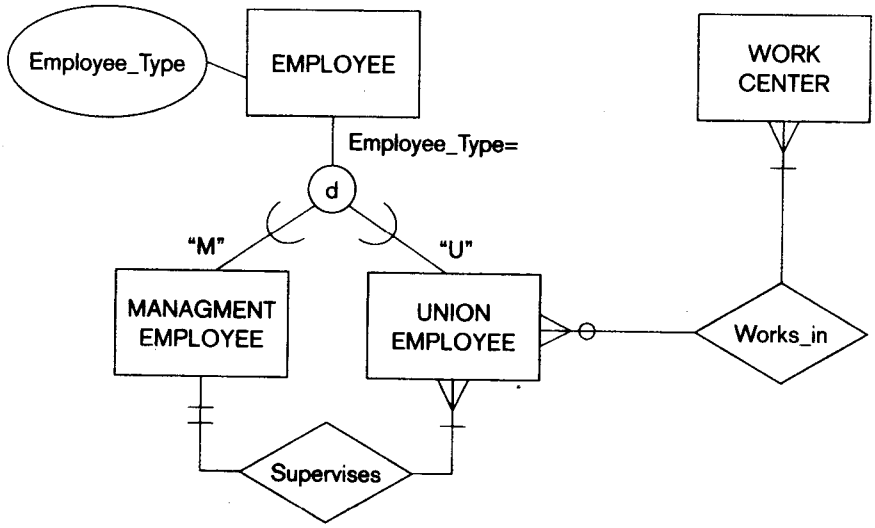


图4-14　MANUFACTURING实体聚簇

4.7　再论业务规则

我们已经了解到，E-R图（以及增强型E-R图）是表达某些业务规则的有效手段。因此，本章所讨论的一些与超类型和子类型相关的参与和不相交约束（disjointness），是与那些联系相关的业务规则的表达式。但是，组织中还有很多其他业务规则不能用前面介绍的符号表达。本节给出一个通用的业务规则框架，并说明如何表达标准的E-R图（和增强型E-R图）符号所不能表示的一些重要类型的规则。

4.7.1 业务规则的分类

有很多不同类型的业务规则，第3章及本章都给出了一些例子，如关于实体间联系的规则（如基数值）、超类型/子类型联系和关于属性和实体类型的事实（例如，定义）。图4-15从GUIDE Business Rules Project改写而来，它给出了给业务规则以及不同类型规则之间潜在联系分类的E-R图。

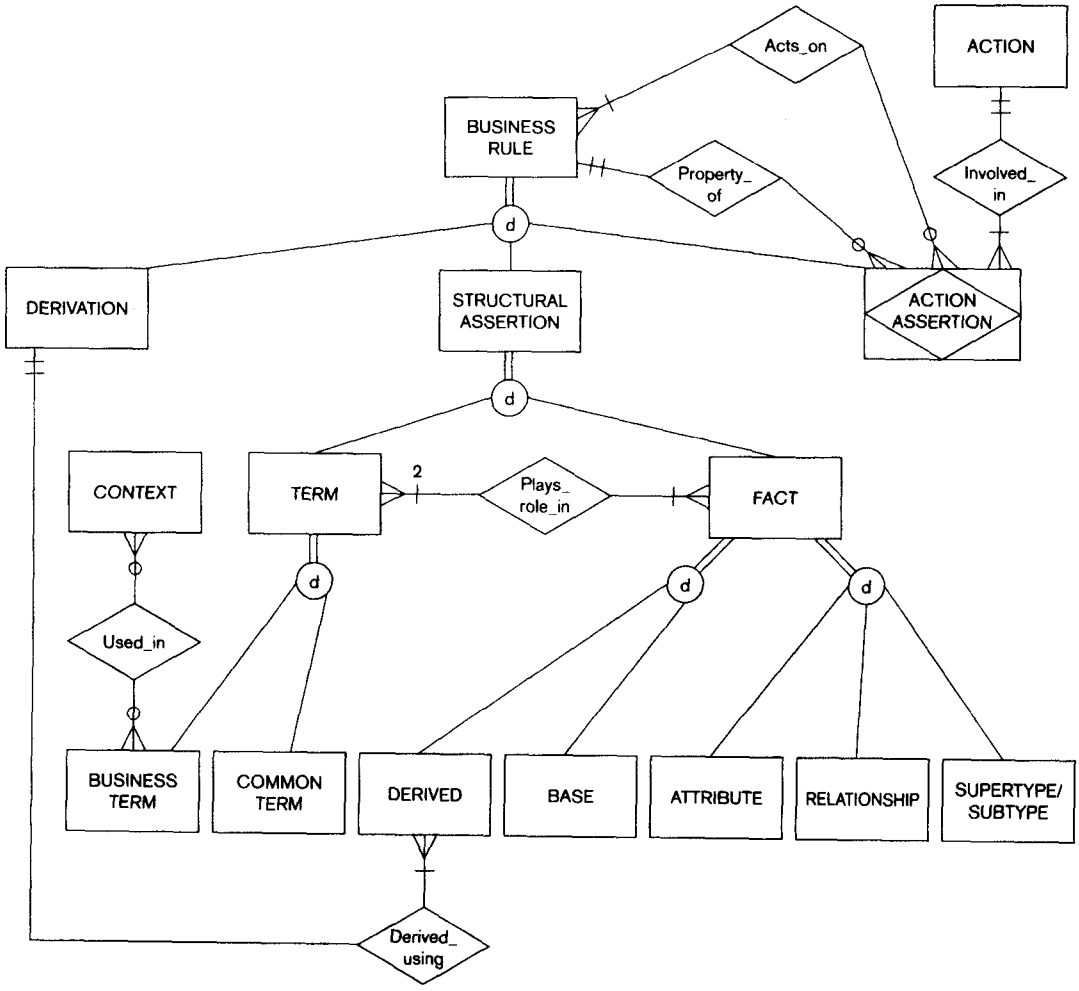


图4-15 描述业务规则的E-R图（改编自GUIDE Business Rules Project, 1997）

有三种主要的业务规则类型：导出、结构断言和动作断言。**导出**（derivation）是指从业务中的其他知识导出的语句。通常，导出是包括文字和事实的数学或逻辑推论。因为在数据建模中，更关心使用导出得到的导出事实，所以，在后面章节中讨论导出事实时才会间接地提到导出。**结构断言**（structural assertion）是表达组织的静态结构的某些方面的语句。E-R图是说明结构断言的常用方法。正如在图4-15中看到的，结构断言可以用术语或事实（术语和事实曾在第3章中定义）来表述。术语在事实的语句中起作用。术语可以是普通的术语，即在通常情况下可以理解；术语也可以是业务术语，即需要在特定的上下文下去理解。**动作断言**（action assertion）是关于组织动作的约束或控制的语句。动作断言是一些业务规则的性质，并说明了在什么条件下，根据什么样的业务规则执行特定的动作（记住，业务规则可以只是实体类型、

联系或属性的定义）。例如，动作断言能够论述在什么条件下，可以创建一个新的顾客或填写一个新的购货订单。另一种常见的动作断言类型说明了特定的属性会有什么样的值（有时称为域约束）。

下面几节论述了结构断言的各种形式，它们通常在E-R图或EER图中说明，并出现在相关数据库文档中。因此，下面几节将论述动作断言。它们可以实现为数据库中的触发器或存储过程，这将在本书的后面讨论。我们使用简单的课程表数据模型来说明这些断言。图4-16给出了该例子的E-R模型。

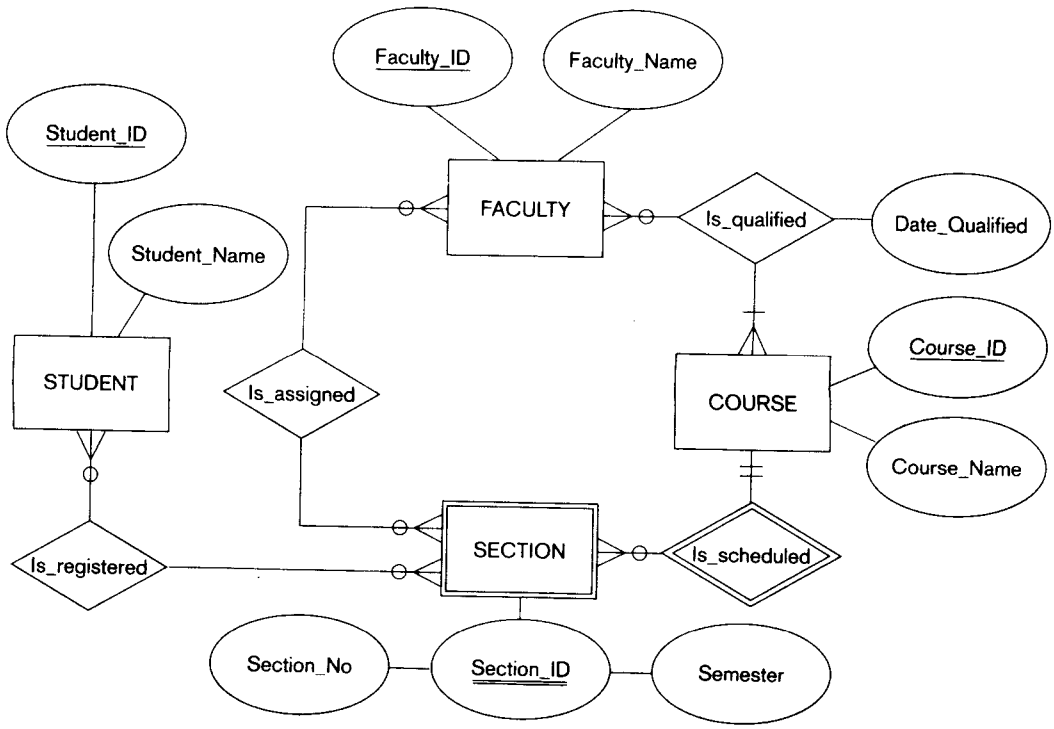


图4-16　课程表的数据模型片段

图4-16中的E-R模型包含4个实体类型：FACULTY、COURSE、SECTION和STUDENT，并且给出了这些实体类型之间的联系。注意，因为SECTION不能离开COURSE实体类型而存在，所以它是弱实体类型。将SECTION的标识联系为Is_scheduled，部分标识符是Section_ID（复合属性）。为了简化该图，仅给出了各实体类型的部分属性。

4.7.2　陈述结构断言

结构断言论述对组织重要的一些事物存在，或存在于与其他感兴趣的事物之间的联系中。结构断言可以只是术语的定义或描述术语之间联系的事实。四个事实的例子是：

- 课程是在特定学科中的一个教学模块。术语定义“课程”与两个术语相关：教学模块和学科。假定这是两个常用术语，不需要进一步在业务上下文中定义。
- 学生姓名是学生的一个属性。这个事实在图4-16中通过Student_Name属性说明。
- 一个学生可以注册多个班，并且一个班可以被多个学生注册。这个事实表明了联系中实体类型的参与。学生和班都是需要定义的业务术语。图4-16通过Is_registered联系说明了这个事实。

- 教员是大学的员工。尽管图4-16中没有给出，这个事实指明了子类型教员及其超类型员工之间的超类型/子类型联系。

导出事实

上面所描述的事实类型称为基事实，即不能从其他术语或事实导出的基本的事实。另一种类型的事实称为导出事实（如图4-16所示）。**导出事实**（derived fact）是使用算法或推理从业务规则中导出的事实。可以像对待基事实那样对待导出事实（例如，因为导出事实是一种业务规则，那么动作断言可能是导出事实的性质）。第3章定义的导出属性就是导出事实的一个例子。下面给出了两个导出事实的例子:

- Student_GPA = Quality_Points/Total_Hours_Taken

其中，Quality_Points=sum[for all courses attempted]
(Credit_Hours*Numerical_Grade)

在这个例子中，Student_GPA是从其他的基事实和导出事实中导出的。Quality_Points和Total_Hours也是导出事实。Student_GPA在图4-16中以连向STUDENT的虚线椭圆给出，Numerical_Grade是Is_registered的属性，Credit_Hours是COURSE的属性。

- 学生由分配给学生注册的班级的老师授课。读者可以看到，根据STUDENT或SECTION的联系Is_registered以及随后的SECTION到FACULTY的联系Is_assigned，如何导出这个事实。

4.7.3 陈述动作断言

结构断言处理的是组织的静态结构，而动作断言处理的是组织的动态方面。动作断言将约束“一定（一定不）”或“应该（不应该）”强加于要处理的数据。动作断言是一些业务规则（称为**锚点对象**）的性质；对于数据处理**动作**（例如，创建、更新、删除、读），它描述了其他业务规则（称为**对应对象**）如何作用于锚点对象。动作断言的一些例子如下:

- 每门课程（锚点对象）必须有一个课程名（对应对象）。在该例中，动作是更新课程的名字性质。
- 每个学生（锚点对象）必须获得2.0及以上的Student_GPA（对应对象）才能毕业（动作）。在该例中，锚点对象是结构断言，但是，锚点对象也可能是另一个动作断言。
- 如果一个班级没有分配有资质的教师（对应对象是联系Is_qualified），则学生不能注册该班级（锚点对象是联系Is_registered）。

动作断言的类型

为了简单起见，图4-15没有给出所有类型的动作断言。有三种方式可以为动作断言分类:

1) 可以根据断言的结果来分类动作断言。考虑这种方式产生的三种动作断言:
   a. 条件 如果某事是真的，则应用另一个业务规则。上面动作断言的第三个例子可以以条件的形式描述“如果一门课程分配了有资质的教师，则学生能够注册开设该课程的班级。”
   b. 完整性约束 一些事情必须永远是真的。上面动作断言的第一个例子说明了完整性约束。还可以给出另一个例子:“教师获得授课资质的日期不能在给该教师分配开设该课程班级的那学期之后。”
   c. 授权 陈述了权限。例如，只有系主任（一种类型的用户）才能授权一个教师教授某门课程。
2) 可以根据断言的形式来分类动作断言。考虑这种方式产生的三种类型的动作断言:
   a. 激活器（Enabler） 陈述如果为真，允许或导致对应对象的存在。激活器的一个例子

是："一旦一名教师有资格教授至少一门课程，就可以创建该教师。"

b. 定时器（Timer）　激活（禁止）或创建（删除）一个动作。定时器断言的一个例子是："如果一个学生的GPA大于2.0，并且总学时数超过125，则该学生可以毕业。"注意，毕业这个动作并不因为该定时器断言而发生，但是，定时器可以激活该动作。

c. 执行器（Executive）　引起一个或多个动作的执行。可以把一个执行器动作断言当成一些动作的触发器。执行器动作断言的一个例子："若一个学生的GPA低于2.0，那么该学生继续留校学习。"该动作断言可能导致学生的一个状态属性更新为值"probation"。

3) 可以根据断言的严格性来分类动作断言。考虑这种方式产生的两种类型的动作断言：

a. 控制（controlling）　陈述了一些一定发生或一定不发生的事情。本书迄今为止使用的例子都属于这一类。

b. 影响（influencing）　一些指导原则或对一定发生的某个通知感兴趣的术语。影响动作断言的一个例子是"当注册某个班级的学生数超过其班容量的90%时，通知负责的系主任。"在这种情况下，没有控制任何事情（学生仍然可以继续注册接近班容量的班级，并不需要创建另外的班级），但是，管理人员想知道某个特定的状况发生了。

4.7.4 表示和强制业务规则执行

大多数组织都有成百上千这样的规则。传统上，动作断言以混入单个应用程序的过程逻辑来实现，这种形式实际上未被承认、不可管理并有不一致性。这种方法给程序员带来很大负担，程序员必须清楚动作可能违反的所有约束，并检查其中每一个约束。程序员的任何疏忽、误解或错误都可能使数据库处于无效的状态。

更为现代的方法是在概念层声明动作断言，而不是说明如何实现规则。因此，需要描述业务规则的规格说明语言。读者已经看到，EER符号能够用于指定多种业务规则。实际上，与更简单的E-R符号相比，EER符号的发明为了使更多业务规则能以图形化方法表示。除了图形化符号之外，另一种方法是结构化语法（诸如英语的某种受限形式）。不论是图形化形式还是语法形式，业务规则语言规格说明需要具备两种良好特性：

1) 应该相对简单，以便最终用户不仅能够理解规则语句，而且能够自己定义规则。

2) 说明语言应该足够结构化，从而能够自动转换为遵守规范的计算机代码。

下面一节论述指定业务规则的图形化和结构化语法方法。本书使用的图形化方法来自Ross (1997)的著作，他是开发业务规则规格说明的一个先驱。

业务规则样例

本节用两个新的业务规则补充图4-16所示的课程表数据模型。在本章后面的问题与练习中，会让读者加入另外的规则。

业务规则1　对于分配教某个班级的一个教师，一定要获得教授该班级课程的授课资格。

该规则涉及图4-16中的三个实体类型：FACULTY、SECTION以及COURSE。问题是：是否能够分配一个教师教授某个班级。因此，锚点对象是联系Is_assigned。在这里，没有约束教师，也没有约束班级。约束的是教师到班级的分派。图4-17显示了从Is_assigned到动作断言符号的虚线。

什么是这个规则中的对应对象？对于分配教授某个班级的教师，有两个必要条件：

1) 教师必须有资格教授课程（该信息由联系Is_qualified记录）。

2) 必须安排该课程的班级（该信息由联系Is_scheduled记录）。

因此，在这种情况下，有两个对应对象。该事实由从动作断言符号到两个联系的虚线

表示。

业务规则2 对分配教某个班级的一个教师，该教师教授班级的总数不能超过三个。

该规则在某时刻教师教授班级的总数上强加了一个限制。由于规则涉及总数，因此需要修改前面的符号

如图4-18所示，锚点对象又是联系Is_assigned。但是，这种情况下，对应对象也是联系Is_assigned。尤其是，它是分配给教师的班级的计数。动作断言符号中的字母“LIM”表示“限制”。从该符号指向带有字母“U”的圆的箭头，字母“U”表示“上限”。第二个圆包含数字3，表示上限为3。因此，约束可读作：“对应对象是分配给教师的班级数，其上限是3。”如果已经给某教师分配了三个班级，那么将会拒绝任何试图给该教师增加班级的事务。

可以使用SQL语言实现上述这些业务规则，并且将规则存储为数据库定义的一部分。使用SQL实现规则的一种方法是使用CREATE ASSERTION语句，它包含在SQL语言的最新版本中(SQL2)。例如，对于业务规则2，下面语句创建了名为“Overload_Protect”的断言：

```
CREATE ASSERTION Overload_Protect
CHECK (SELECT COUNT(*)
FROM ASSIGNED
WHERE Faculty_ID = '12345') <= 3;
```

该语句检查分配给特定教师的班级总数是否少于或等于相应的限制（3）。数据库管理系统负责确保不会违反该约束。

除了本节的几个例子外，还可能有很多其他的业务规则。参见Ross(1997)，它提供了全面的列表和相关的符号。

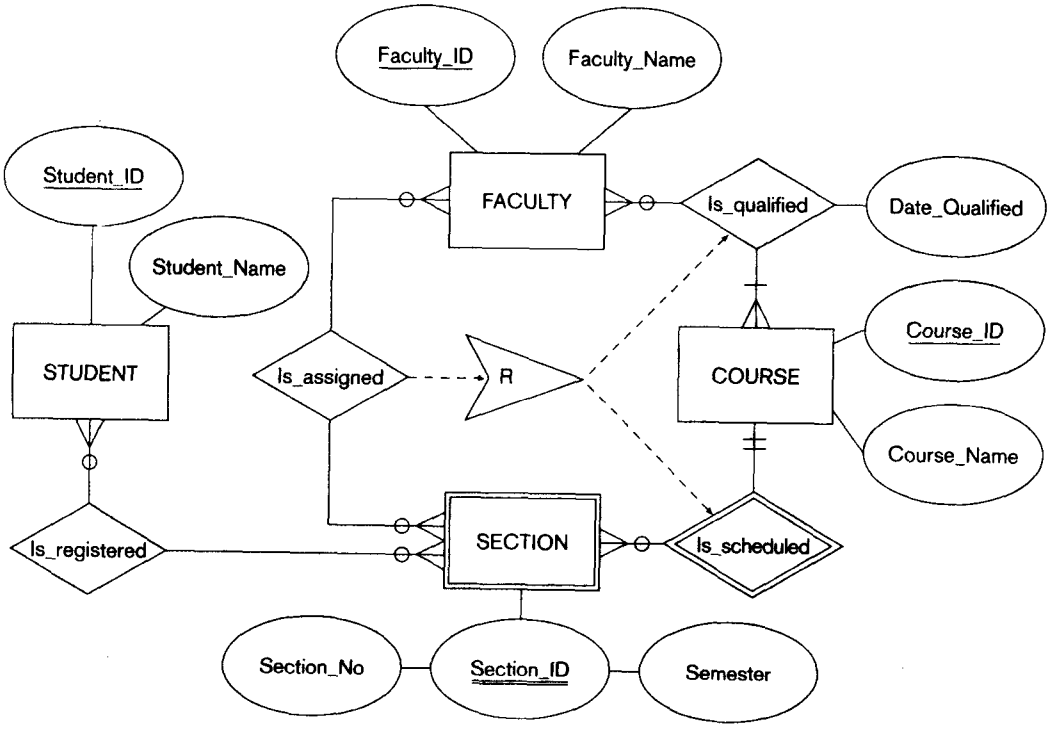


图4-17 业务规则1：对于分配教授某个班级的一个教师，一定要获得教授该班级课程的授课资格（符号中的字母“R”表示“Restricted”，它是很多可选约束中的一个）

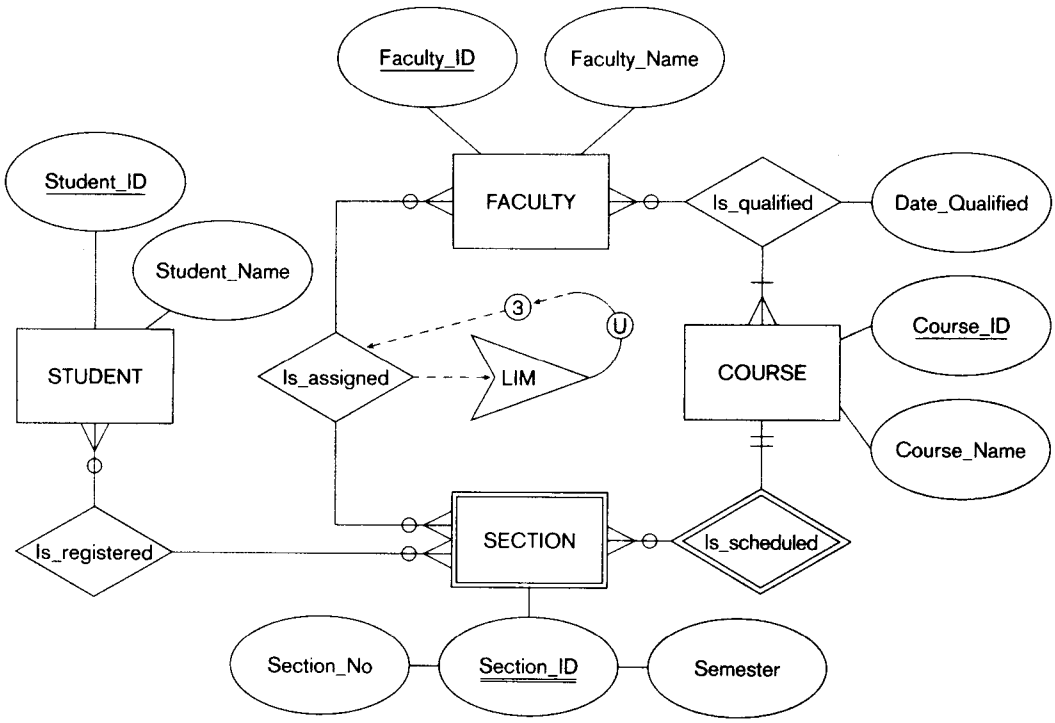


图4-18　业务规则2：对分配教授某个班级的一个教师，该教师教授班级的总数不能超过三个

4.7.5　标识和测试业务规则

本章已经论述了各种业务规则以及使用结构化语法、EER图及其扩展表示业务规则的方法。数据分析人员的任务就是文档化业务规则的规格说明并确保尽可能使用数据库技术强制执行这些业务规则。但是，在说明业务规则之前，必须标识它们。一旦指定了业务规则，应该在实现之前测试它们。

标识和测试业务规则的一个方法是场景（Moriarty,1993）。场景是描述业务在给定情况下如何作出反应的故事或剧本。场景类似于用例的概念，用例是对象建模的一个重要工具（见第14章和第15章）。例如，图4-16中E-R图的场景主要关注和学生相关的事件，包括被录取、注册（或取消）课程、获得成绩和毕业。场景通过追踪这些事件来表现一个人的状态。场景用来标识数据模型的元素（业务规则），或测试处理这些事件的所有可能情况的数据模型（例如，一个学生试图注册不存在的班级，一个试读期学生试图注册，或一个研究生试图注册）。

一个学生场景可能如下："Missi Davies 登录课程注册系统以注册下学期的课程。Missi输入其学生ID和口令以获得选择课程的权限。Missi是MIS专业的学生，Missi想确保她已进入本专业所需的课程，她在MIS课程列表中查找数据库课程。Missi发现，下学期有两个班设置该课程，一个已经停止选课。因此，利用这个在线注册系统，Missi必须选择尚未停止选课的班级。"如果还没有标识业务规则，该场景和其他场景可以用于标识数据模型应该包括的术语、事实和动作断言。如果一些业务规则已经存在，那么场景中的语句可以用于验证处理所有可能情况的业务规则。

本章小结

本章论述了如何扩展基本的E-R模型，使之包括超类型/子类型联系。超类型是与一个或多个子类型有联系的一般实体类型。子类型是对组织有意义的一个实体类型中的一组实体。例如，实体类型PERSON通常建模为超类型，其子类型可能包括EMPLOYEE、VOLUNTEER和CLIENT。子类型继承其超类型的属性和联系。

在数据建模中，当下面两个条件之一（或全部）成立时，通常会考虑超类型/子类型联系。首先，存在一些属性适用于一个实体类型的一些（不是全部）实例；其次，一个子类型的实例参与那个子类型特有的联系。

概化和特化技术是开发超类型/子类型联系的重要方法。概化是自底向上的过程，从一些特定的实体类型定义概括的实体类型。特化是自顶向下的过程，为已定义的超类型定义一个或多个子类型。

使用EER符号能够捕获适用于超类型/子类型联系的重要的业务规则。使用完备性约束能够说明一个超类型的实例是否必须是至少一个子类型的实例。有两种情况：对于完全特化规则，超类型的一个实例必须是至少一个子类型的实例。对于部分特化规则，超类型的一个实例可能是（也可能不是）任何子类型的实例。使用不相交约束能够说明一个超类型的实例是否可能同时是两个或多个子类型的实例。同样，有两种情况：对于不相交规则，在给定的时间，一个实例（超类型的实例）只能是一个子类型的实例；对于交叠规则，一个实体实例可以同时是两个（或多个）子类型的实例。

子类型鉴别符是超类型的属性，其值决定了超类型的实例属于哪一个（或多个）子类型。超类型/子类型层次是超类型和子类型的层次安排，其中每一个子类型仅有一个超类型。

除了超类型/子类型联系外，E-R符号还有其他的扩充。其中一个较有用的扩充是聚合，聚合表示一些实体是另一些实体的一部分（例如，计算机是由磁盘驱动器、RAM、主板等组成）。由于篇幅的限制，在本章没有讨论这些扩展。这些扩展中的大多数，如聚合，也是面向对象数据建模的一部分，这部分内容将在第14章或第15章讨论。

E-R图可能会包含数百个实体，因而变得庞大而且复杂。很多用户和经理为了理解数据库中他们最感兴趣的那一部分，不必看到所有的实体、联系和属性。实体聚簇是解决该问题的一种方法，它将一部分实体-联系数据模型转换成相同数据的宏观视图。实体聚簇将一个或多个实体类型及相关联系的集合组成单个的抽象实体类型。几个实体聚簇及相关联系可以进一步组成更高层的实体聚簇，因此，实体聚簇是一种层次分解技术。通过将实体和联系分组，可以对E-R图进行合理布局，使人们关注数据建模任务中最要紧的那部分模型的细节。

业务规则是定义或约束业务的某些方面的语句。其目的是声明业务结构，或控制、影响商业行为。可以将规则分为三类：导出、结构断言和动作断言。结构断言定义组织的静态结构，而动作断言是约束组织的动态操作的规则。结构断言是术语或事实。术语是关于业务规则概念的定义，事实是涉及两个或多个术语的关联的语句。事实可以是基本的或基础的事实，也可以是从其他事实的数学或逻辑操作中导出的事实。它们可以处理属性、联系或事实之间的超类型/子类型联系。

动作断言论述了一些业务规则之上的动作（如创建、删除、更新和读），这些业务规则称为锚点对象，被其他对应对象所约束。约束可以是“If-Then-Else”条件、完整性规则或授权权限。动作断言允许对应对象的存在、激活或禁止某个动作或引起动作的执行。其他类型的动作断言包括特定条件下触发器通知的简单指导方针。

本章复习

关键术语

动作	动作断言	锚点对象
属性继承	完备性约束	对应对象
导出	导出事实	不相交规则
不相交约束	增强型实体联系(EER)模型	实体聚簇
概化	交叠规则	部分特化规则
特化	结构断言	子类型
子类型鉴别符	超类型	超类型/子类型层次
完全特化规则		

复习问题

1. 定义下列术语。

a. 超类型	b. 子类型
c. 特化	d. 实体聚簇
e. 结构断言	f. 锚点对象
g. 子类型鉴别符	h. 完全特化规则
i. 概化	j. 不相交规则
k. 交叠规则	l. 动作断言

2. 将下列术语与其定义匹配起来。

_____子类型	a. 子类型的集合
_____实体聚簇	b. 属于两个子类型的实体
_____结构断言	c. 子类型获得超类型属性
_____子类型	d. 关于限制哪个动作的规则
_____特化	e. 影响动作的业务规则
_____锚点对象	f. 创建、删除、更新或读
_____动作	g. 使用算法计算
_____子类型鉴别符	h. 概括的实体类型
_____属性继承	i. 组织的静态结构
_____交叠规则	j. 创建一个实体类型的子类型
_____对应对象	k. 一组相关的实体类型和联系
_____导出事实	l. 查找一个实体类型的目标子类型

问题和练习

1. 检查大学EER图的层次（参见图4-10）。作为学生，你是下列子类型的实例：UNDERGRAD STUDENT或GRADUATE STUDENT。列出适合你的所有属性的名字，然后，对每一个属性，给出适合的数据值。

2. 为图4-10中的每一个超类型添加一个子类型鉴别符。给出指定实例给不同子类型的鉴别符的值。使用下面的子类型鉴别符和值：

a. PERSON: Person_Type(Employee?, Alumnus?, Student?)

b. EMPLOYEE: Employee_Type(Faculty, Staff)

c. STUDENT: Student_Type(Grad, Undergrad)

3. 为下面的问题绘制一个EER图。

一个非盈利组织的成功运作，依赖于不同类型的人员。该组织对所有人员的下列属性感兴趣：SSN、Name、Address、City/State/Zip和Telephone。他们对三种类型的人员最感兴趣：员工、志愿者和捐赠者。员工仅有Date_Hired属性；志愿者仅有Skill属性；捐赠者仅有与Item实体类型之间的联系（称为Donates）。捐赠者必须已捐赠一到多项，而某一项可以没有捐赠者，也可以有一个或多个捐赠者。

除了上述几种组织感兴趣的人员外，还有其他一些人员。因此，一个人可能不属于这三种类型的任何一种。另一方面，一个人可能同时属于其中两个或多个类型（例如，员工和捐赠者）。

4. 在问题和练习3创建的E-R图中，加入一个子类型鉴别符（称为Person_type）。

5. 汽车租赁机构将它所租赁的交通工具分为四类：小型、中型、大型及运动型。机构想要记录交通工具的下列数据：Vehicle_ID、Make、Model、Year和Color。这四类交通工具都没有独特的属性。实体类型Vehicle与实体类型Customer有一种联系（称为Rent）。四种交通工具与另外的实体类型都没有专门的联系。你是否考虑要为该问题创建一个超类型/子类型联系？为什么？

6. 在一次周末度假中，实体类型PERSON有三个子类型：CAMPER、BIKER和RUNNER。分别为下面每一种情况画一个EER图片段：

a. 在某一给定时间，某一个人必须仅属于其中一种子类型。

b. 某一个人可能属于（或不属于）这些子类型。另一方面，某一个人可能同时是这些子类型中的任何两个（甚至三个）。

c. 在某一给定时间，某一个人必须至少属于其中一种子类型。

7. 某一银行有三种类型的账户：支票、存款、贷款。下面是每一种账户类型的属性：

CHECKING: Acct_No、Date_Opened、Balance、Service_Charge

SAVINGS: Acct_No、Date_Opened、Balance、Interest_Rate

LOAN: Acct_No、Date_Opened、Balance、Interest_Rate、Payment

假设每一个银行账户必须仅属于其中一种子类型。使用概化给出表示这种情况的EER模型。记住使用子类型鉴别符。

8. 参考图4-2中的员工EER图完成下面的练习。可以做出你认为必要的任何假设。

a. 对图中每个实体类型、属性和联系，给出示范性定义。

b. 为图中所有属性，给出示范性的完整性约束动作断言。

9. 参考图4-16中的课程表的数据模型完成下面练习。在图中加入表达下列业务规则的符号：

a. 对一个要安排课程的班级，必须分配一个有资格教授这门的课程的教师（提示：见图4-17）。

b. 对要注册一个课程的某个班级的学生来说，在一给时间内，这个学生注册的班级不能多于六个（提示：见图4-17）。

10. 参考医院联系的EER图（见图4-3），加入表达下列规则的符号：仅当给某个住院病人分配一个主治医生后，才能给这个住院病人分配一个床位。

在这个例子中，回答下列问题：

a. 哪个（或哪些）是锚点对象？

b. 哪个（或哪些）是对应对象？

11. 考虑下面的业务规则："仅当一个学生完成了他的家庭作业后，他才能去参加音乐会。"

a. 画一个EER图描述这条规则所暗示的实体和联系。

b. 加入合适的符号以表达业务规则。

c. 标识下列对象：（1）锚点对象；（2）对应对象。

12. 参考病人的EER图（见图4-3）完成下列练习。可以做出你认为必要的任何假设。

a. 对图中每一个实体类型、属性和联系给出其定义。

b. 对图中的每个属性，给出其完整性约束动作断言。

13. 图4-13给出了松谷家具公司E-R图的实体聚簇的开发过程。参考图4-13b，回答下面的问题：

a. 为什么Does_businese_in联系中CUSTOMER一端的最小基数是0?

b. ITEM的属性有哪些（参考图3-22）？

c. MATERIAL的属性应该有哪些（参考图3-22）？

14. 参考第3章问题与练习6的答案。开发该图的实体聚簇，并使用实体聚簇重画该图。解释为什么选择你所使用的实体聚簇。

15. 参考第3章问题与练习11的答案。开发该图的实体聚簇，并使用实体聚簇重画该图。解释为什么选择你所使用的实体聚簇。

应用练习

1. 与一个朋友或家庭成员面谈，找到他们可能在工作中接触的关于下列概念常见的例子。

a. *超类型/子类型联系*。向与你面谈的人解释该术语的含义。给出一个常见的例子，如PROPERTY: RESIDENTIAL, COMMERCIAL; 或者BONDS: CORPORATE, MUNICIPAL。利用他所提供的信息，构造一个EER图并给他审阅。如果有必要，由你进行修正，直到你、你的朋友或家庭成员觉得合适为止。

b. *业务规则*。将本章（图4-17和图4-18）提供的业务规则的例子，解释给与你面谈的人。当被调查对象根据他或她的环境给出一个业务规则时，如果需要，遵照本章的语法进行重构。

2. 访问本地的两个企业，一个属于服务业，另一个属于制造业。与这些组织中的员工面谈，找出一些超类型/子类型联系和业务规则的例子（例如，"顾客只有凭有效的销售单才能退货"）。在什么样的环境下才会较容易发现这些构造，为什么?

3. 请一个当地公司的数据库管理员或数据库或系统分析员向你展示其组织一个数据库的EER图（或E-R图）。这些组织建模了超类型/子类型联系吗？公司使用的CASE工具支持这些联系吗？而且，在EER建模阶段包括哪些类型的业务规则？如何表示业务规则？在哪里存储？如何存储?

4. 阅读GUIDE Business Rules Project(1997)发表的关于业务规则的总结以及Gottesdiener(1997)的文章。在Web上搜索另外的关于业务规则的文章。写三页的总结，包括业务规则当前的方向、对系统开发和维护的潜在的影响。

参考文献

Elmasri, R. and S. B. Navathe. 1994. *Fundamentals of Database Systems*. Menlo Park, CA: Benjamin/Cummings.

Gottesdiener, E. 1997. "Business Rules Show Power, Promise." *Application Development Trends* 4(3) (March): 36-54.

"GUIDE Business Rules Project." Final Report, revision, 1.2 October, 1997.

Moriarty, T. "Using Scenarios in Information Modeling: Bringing Business Rules to Life." *Database Programming & Design* 6(8): 65-67.

Ross, R. G. 1997. *The Business Rule Book*. Version 4. Boston: Business Rule Solutions, Inc.

Teorey T. 1999. *Database Modeling & Design*. San Francisco, CA: Morgan Kaufman Publishers.

进一步阅读

Schmidt, B. 1997. "A Taxonomy of Domains." *Database Programming & Design* 10(9) (September): 95, 96, 98, 99.

von Halle, B. 1996. "Object-Oriented Lessons." *Database Programming & Design* 9(1) (January): 13-16.

von Halle, B., and R. Kaplan. 1997. "Is IT Falling Short?" *Database Programming & Design* 10(6) (June): 15-17.

Web资源

- www.adtmag.com *Application Development Trends*杂志，关于信息系统开发实践的重要出版物。
- www.brsolutions.com Business Rules Solutions是Ronald Ross的顾问公司，Ronald Ross是业务规则方法学开发的带头人。
- www.businessrulesgroup.org The Business Rules Group，以前是GUIDE国际组织的一部分，负责业务规则的制定、支持标准。
- www.guide.org SHARE是一个志愿者组织，为IT业提供教育、网络。SHARE和GUIDE已经合并，两者以前都是IBM公司顾客的用户组。
- www.Intelligententerprise.com "*Intelligent Enterprise*" 杂志，数据库管理和相关领域的重要出版物。该杂志将以前的Database Programming&Design和DBMS合并在一起。
- www.kpiusa.com/BusinessRuleProducts.htm 由Knowledge Partners公司维护的网站，业务规则产品的销售商可在该网站上列出信息。

项目案例：山景社区医院

项目描述

山景社区医院作为大型的服务机构，大量人员是其持续成功的保障。医院主要依靠四类人：员工、医生、病人和志愿者。当然，所有这些人共享一些公共的属性：Person_ID（标识符）、Name、Address、City/State/Zip、Birth_Date以及Phone。

其中每一类人至少有一个独特的属性：Employee有Date_Hired，Volunteer有Skill，Physician有Specialty和Pager#（呼机号），Patients有Contact_Date（第一次看病的日期）。

社区医院中也确实有一些人不属于上述四类人（人数相对较少）。但是，有一些特殊的人会同时属于上述四类中的两类或多类（例如，病人和志愿者）。

每一个病人由且仅由一名医生对其负责。在某个时间，有的医生可能不负责病人，也可能负责一个或多个病人。病人分为两类：住院病人和门诊病人。每一个住院病人都有Date_Admitted属性；每一个门诊病人都会安排一次或多次复诊。实体visit有两个属性：Date（部分标识符）和Comments。注意，每一个visit的实例都不能离开门诊病人实体而存在。

员工分为三类：护士、职员和技术人员。只有护士具有属性Certificate，意味着资质（RN，LPN等）；仅有职员具有属性Job_Class；只有技术人员具有属性Skill。每一个护士都被分派到一个或多个诊疗中心。诊疗中心包括Maternity、Emergency和Cardiology。诊疗中心的属性有：Name（标识符）和Location。同时，每个诊疗中心都有一名或多名护士，分配到诊疗中心的护士都约定nurse_in_charge。如果护士没有RN资质，则不能约定nurse_in_charge。

每一个技术人员都被分派到一个或多个实验室。实验室的属性包括Name（标识符）和Location。每一个实验室至少必须分派一个技术人员，也可以有任意数目的技术人员。

诊疗中心可能没有床位，也可能有一个或多个床位（可以有任意数目）。床位的惟一属性是Bed_ID（标识符）。Bed_ID是一个复合属性，它包括两个部分：Bed#和Room#。每一个住院病人必须分配一个床位。在某一个时刻，一个床位可能有也可能没有被分配给住院病人。

项目问题

1）建模超类型/子类型联系的能力在山景社区医院这样的医院环境中是否很重要？说明你的理由。

2）能否将业务规则范型和易于定义、实现、维护业务规则的能力作为山景社区医院这样的医院环境的竞争优势？说明你的理由。

3）在本项目案例的数据需求描述中是否有一些弱实体？如果有，列举出这些弱实体。

4）你能够想到一些可以用于医院环境的业务规则吗？（本项目案例中已给出的那些业务规则除外。）

项目练习

1）绘制一个E-R图以准确地表示这些需求，要利用本章介绍的符号。

2）给出项目练习1的E-R图中的下列对象类型的定义，如果有条件的话，与医院或诊疗中心的人员讨论。否则，基于你自己的知识和经验，做出合理的假设。

a. 实体类型

b. 属性

c. 联系

3）你应当知道语句“除非一个护士有RN证书，否则他或她不能被指派为诊疗中心的

nusrse_in_ charge”是业务规则。回答下列问题:

a. 什么是锚点对象？它是实体、属性、联系还是一些其他对象？

b. 什么是其对应对象（或对象组）？它是实体、属性、联系或一些其他对象？

4) 将你在本章开发的EER图与第3章的项目练习2中开发的E-R图进行比较，这两张图的区别是什么？它们为什么不同？

5) 将项目练习4和第3章项目练习2的两个图合并为一个图，解释在合并过程中你所做的决定。

第三部分　数据库设计

概要

截止到数据库开发过程的分析阶段为止，系统和数据库分析人员对于数据库存储和访问的需求已经有了相对清晰的了解。但在分析阶段里产生的数据模型显式地避免与数据库技术本身存在任何联系。在我们可以实现数据库以前，概念数据模型必须映射成为一种与所使用的数据库管理系统相容的数据模型。

数据库设计的目的是将在数据库分析阶段所产生的对于数据存储的需求转化为具体的规格说明，以指导数据库的实现。有两种形式的规格说明：

1）逻辑规格说明，将概念化的需求映射为与具体数据库管理系统相关的数据模型。

2）物理规格说明，确定所有用于实现的数据存储参数，在这个过程中，实际的数据库系统会使用数据定义语言得到定义。

在第5章中，我们将介绍逻辑数据库设计，重点基于关系模式。逻辑数据库设计是将概念数据模型（第3章和第4章）转化为逻辑数据模型的过程。大多数目前使用的数据库管理系统都使用关系模式，所以我们将它作为讨论逻辑数据库设计的基础。

在本章中，我们首先定义关系模式的重要术语和概念，包括关系、主键、外键、异常、范式、规范化、函数依赖、部分函数依赖和传递依赖。然后讨论如何将E-R模型转化为关系模型。虽然很多CASE工具支持这种转化，但是理解基本的概念和过程仍然非常重要。接下来，我们详细介绍规范化（设计良构关系的过程）的重要概念。最后，我们讨论如何将来自不同源的逻辑设计的关系合并（比如来自同一个项目组的若干个小组），并解决其中常见的问题。

第6章的主题是物理数据库设计，其目的是将数据的逻辑描述转化为存取数据的规格说明。主要目标是设计数据存储，以满足性能要求，同时确保数据完整性、安全性和可恢复性。物理数据库设计所制定的规格说明，会被从事信息系统构建的人员，如程序员等在实现阶段所使用（参见第7章～第11章）。

在第6章中，读者会学习到物理数据库设计的主要术语和概念，如数据类型、页、指针、非规范化、分割、索引文件组织和散列文件组织等，还包括高效完成物理数据库设计的基本步骤。读者会了解到存储属性值的不同方法及如何在这些方法中进行选择，同时会明确为什么规范化的表不一定能组成最好的物理数据文件，以及如何通过非规范化来改善数据存取速度。文件组织和索引对于数据存取效率的提高非常重要，使用廉价磁盘冗余阵列（RAID）可以提供更好的数据库性能和可靠性，该章中也会对此进行介绍。此外，读者还会学习到不同数据库系统体系结构的主要差异和一些用于加快数据访问速度的设计和查询技术。

物理数据库设计必须是非常慎重的，因为在这个阶段所作的决定会对数据可访问性、响应时间、安全性和用户友好性等产生重要的影响。物理数据库设计和数据库管理（第12章）密不可分，在第12章中我们会进一步讨论一些相关的问题。第13章会介绍分布式数据库的设计问题。

第5章 逻辑数据库设计和关系模型

5.1 学习目标

学完本章后，读者应该具备以下能力：

- 掌握以下关键术语：关系、主键、复合键、外键、空值、实体完整性规则、参照完整性约束、良构关系、异常、递归外键、规范化、范式、函数依赖、决定因子、候选键、第一范式、第二范式、部分函数依赖、第三范式、传递依赖、同义词、别名、异义和企业键。
- 列举出关系的五个性质。
- 描述候选键的两个重要性质。
- 用简洁的语言定义以下概念：第一范式、第二范式和第三范式。
- 简要地描述合并关系时可能带来的四个问题。
- 将E-R图（或EER图）转化为逻辑上等价的关系集。
- 创建结合实体完整性和参照完整性约束的关系表。
- 使用规范化将含有异常的关系分解为良构关系。

5.2 引言

在本章中，我们将介绍逻辑数据库设计，重点介绍关系数据模型。逻辑数据库设计是将概念数据模型（在第3章和第4章中介绍过）转化为逻辑数据模型的过程。虽然还存在着其他的数据模型，但是由于以下两个原因，我们仍在本章重点介绍关系数据模型。首先，在现代数据库应用中，关系数据模型使用最广泛。其次，一些关系模型的逻辑数据库设计原理也适用于其他的数据库模型。

前面几章通过一些简单的例子已经非正式地介绍过关系模型。在本章中，我们首先定义关系模型的重要术语和概念（通常使用缩写的术语*关系模型*来代表关系数据模型），然后介绍和讨论将E-R模型转化为关系模型的过程。虽然现在有很多CASE工具可以完成这一转化工作，但是了解其基本的原理和过程仍然非常重要。本章接着将详细介绍规范化的概念。规范化（即设计良构关系的过程）是关系模型的逻辑设计的重要部分。最后，本章介绍如何合并关系，以避免这一过程中一些常见的问题。

逻辑数据库设计的目标是将概念设计（体现某个组织对于数据的需求）转化为可以在特定的数据库管理系统上实现的逻辑数据库。最终的数据库必须满足用户对于数据共享、灵活性和易于访问等方面的需求。本章所介绍的概念对于理解数据库开发的过程非常重要。

5.3 关系数据模型

关系数据模型最早是由IBM公司的E.F.Codd在1970提出的（Codd，1970）。随即启动了两个早期的研究项目来证实关系模型的可行性以及开发原型系统。其中的第一个项目在IBM的San Jose研究实验室进行，并于20世纪70年代后期开发出System R（一个关系DBMS——RDBMS的原型）。第二个项目在加利福尼亚大学伯克利分校开展，开发出一个面向学术研究的关系数据库

管理系统Ingres。大约在1980年，由不同厂商提供的大量商用关系数据库管理系统开始出现（在本书的网站上可以找到RDBMS和其他DBMS提供商的链接）。现在，关系数据库管理系统已成为数据库管理的主流技术，现在有上百种适用于从个人计算机到大型机的关系数据库产品。

5.3.1 基本定义

关系数据模型采用表的方式来存储数据。关系模型以数学理论为基础，因而具有坚实的理论基础。但我们只需要一些简单的概念就能描述关系模型，即使对于那些不熟悉基础理论的人来说，也能轻松理解和使用关系模型。关系数据模型包含以下三个组成部分（Fleming and von Halle,1989）：

1) **数据结构** 数据以具有行和列的表的方式组织。

2) **数据操纵** 利用功能强大的操作（使用SQL语言）来操纵存储在关系中的数据。

3) **数据完整性** 在操纵数据时，包含相应的机制来定义业务规则以维护数据的完整性。

本节主要讨论数据结构和数据完整性，数据操纵则在第7章、第8章和第10章进行讨论。

关系数据结构 **关系**（relation）是一个已命名的二维数据表。每个关系（表）包含一组已命名的列，以及任意数目的未命名的行。与第3章中的定义相同，属性是关系中的一个已命名的列。关系中的每一行对应一条记录，记录中包含一个实体的数据（属性）值。图5-1是一个名为EMPLOYEE1的关系。这个关系包含以下描述员工信息的属性：Emp_ID、Name、Dept_Name和Salary。表中的五行对应于五个员工。值得注意的是，图5-1中的示例数据是用来说明关系EMPLOYEE1的结构的；它们不是关系本身的一部分。即使我们在图中增加一行数据，它仍然是EMPLOYEE1关系，删除一行也同样不会改变这个关系。实际上，即使删除图5-1中的所有行，EMPLOYEE1关系也仍然存在。换句话说，图5-1只是EMPLOYEE1关系的一个实例。

EMPLOYEE1

Emp_ID	Name	Dept_Name	Salary
100	Margaret Simpson	Marketing	48 000
140	Allen Beeton	Accounting	52 000
110	Chris Lucero	Info Systems	43 000
190	Lorenzo Davis	Finance	55 000
150	Susan Martin	Marketing	42 000

图5-1 EMPLOYEE1关系的示例数据

可以使用下面的简略表示方法来表示关系的结构，关系名后面用括号括起属性名列表。EMPLOYEE1关系可用以下方法表示：

```
EMPLOYEE1(Emp_ID, Name, Dept_Name, Salary)
```

关系键 用户必须能够根据行中的数据值来存储或检索关系中的一行数据。为实现这个目的，每个关系必须有一个主键。**主键**(primary key)是可以惟一标识关系中每一行的属性（或属性集）。我们通常在作为主键的属性名加下划线。比如，关系EMPLOYEE1的主键是Emp_ID。请注意，在图5-1中，该属性下加上了下划线。使用简略表示方法，我们可以用如下方式表示该关系：

```
EMPLOYEE1(Emp_ID, Name, Dept_Name, Salary)
```

主键的概念和第3章中定义的术语“标识”是相关的。在E-R模型中作为实体标识符的属性（或属性集）在表示该实体的关系中也可能是主键的组成部分。但是也有一些例外；比如关联实体不一定有标识，弱实体的标识符只是弱实体主键的一部分。此外，一个实体中可以作为关

联关系主键的属性可能有多个。在本章的后面，我们会讨论到这些情况。

复合键(composite key)是由多个属性构成的主键。比如，关系DEPENDENT的主键是由Emp_ID和Dependent_Name构成的。在本章的后面，我们还会再给出一些复合键的例子。

我们往往需要表示两个表或关系之间的联系，这是通过使用外键来实现的。**外键**(foreign key)是一个数据库中关系的属性（可能是复合属性），它是该数据库的另一个关系的主键。比如，考虑关系EMPLOYEE1和DEPARTMENT:

```
EMPLOYEE1(Emp_ID, Name, Dept_Name, Salary)
DEPARTMENT(Dept_Name, Location, Fax)
```

属性Dept_Name就是关系EMPLOYEE1的外键。它使用户可以将某个员工与其所属的部门关联起来。为说明一个属性是外键，某些作者会使用虚线下划线来表示，例如

```
EMPLOYEE1(Emp_ID, Name, Dept_Name, Salary)
```

在本章的后面，我们会介绍很多有关外键的例子，并结合参照完整性来讨论外键的性质。

关系的性质　在前面我们已经定义关系是二维的数据表，但并不是所有的表都是关系。关系具有一些非关系表所不具有的性质。我们将这些性质归纳如下。

1) 数据库内的每个关系（表）的名字是惟一的。

2) 任意行的任一列值是原子的（单值）。在关系内，不允许有多值属性。

3) 每一行是惟一的，同一关系内的任意两行都不能相同。

4) 表中的每个属性（列）的名字是惟一的。

5) 关系中列的顺序（从左到右）是无关的。交换关系的任意两列不影响关系的意义和使用。

6) 关系中行的顺序（从上到下）是无关的。和列的情况一样，关系中的行也可以交换或是以任意顺序存储。

从表中去除多值属性　关系的第二项性质规定关系中不能有多值属性。因此含有一个或多个多值属性的表就不是关系。比如，图5-2a扩展了关系EMPLOYEE1，以包含每个员工所参加的课程信息。由于每个员工所参加的课程不一定惟一，所以属性Course_Title和Data_Completed都是多值属性。例如，Emp_ID为100的员工参加两门课程的学习。如果某个员工没有参加任何课程，那么他的Course_Title和Data_Completed属性就为空值（例如Emp_ID为190的员工）。

在图5-2b中，我们通过将相应的数据填入5-2a中空白的单元来去除多值属性，现在图5-2b中的表只有单值属性，并满足关系的原子性质要求。我们将新的关系命名为EMPLOYEE2，以便与EMPLOYEE1区别开来。但是在后面的讨论中我们会发现，这个新的EMPLOYEE2关系有一些不好的性质。

5.3.2 数据库示例

一个关系数据库中可以包含任意多个关系。数据库的结构是通过概念模式（第2章介绍过）进行描述的，这是对数据库的总体逻辑结构的描述。有两种常用的表示（概念）模式的方法:

1) 短文本语句，其中关系名后跟着以括号括起的属性名（如本章前面定义的关系EMPLOYEE1和DEPARTMENT）。

2) 图形表示，其中每个关系用包含关系属性的矩形来表示。

文本语句的方法比较简单，但是图形表示方法可以更好地表示参照完整性约束（很快就可以看到）。在本节中，我们同时使用这两种方法来表示模式以便进行比较。

在第3章中，我们已经为松谷家具公司构建了E-R图（参见图3-22）。在第1章中，我们给出

了这个数据库的四个关系（参见图1-4）：CUSTOMER、ORDER、ORDER LINE和PRODUCT。在本节中我们给出这四个关系的模式，它们的图形表示见图5-3。

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48 000	SPSS	6/19/200X
				Surveys	10/7/200X
140	Alan Beeton	Accounting	52 000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43 000	SPSS	1/12/200X
				C++	4/22/200X
190	Lorenzo Davis	Finance	55 000		
150	Susan Martin	Marketing	42 000	SPSS	6/16/200X
				Java	8/12/200X

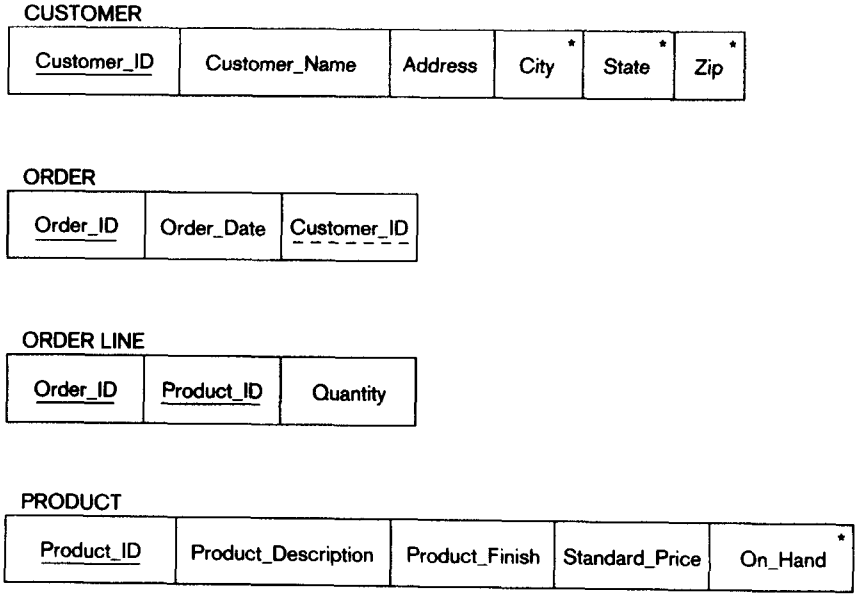
a) 含有重复组的表

EMPLOYEE2

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48 000	SPSS	6/19/200X
100	Margaret Simpson	Marketing	48 000	Surveys	10/7/200X
140	Alan Beeton	Accounting	52 000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43 000	SPSS	1/12/200X
110	Chris Lucero	Info Systems	43 000	C++	4/22/200X
190	Lorenzo Davis	Finance	55 000		
150	Susan Martin	Marketing	42 000	SPSS	6/19/200X
150	Susan Martin	Marketing	42 000	Java	8/12/200X

b) EMPLOYEE2关系

图5-2 消除多值属性



注：*在图3-22的基础上进行了简化。

图5-3 松谷家具公司中四个关系的模式

下面是这些关系的文本形式的表示：

CUSTOMER(Customer_ID, Customer_Name, Address, City, State, Zip)

ORDER(Order_ID, Order_Date, Customer_ID)

ORDERLINE(Order_ID, Product_ID, Quantity)

PRODUCT(Product_ID, Product_Description, Product_Finish, Standard_Price, On_Hand)

注意，ORDER_LINE的主键是复合键，由属性Order_ID和Product_ID构成。另外，Customer_ID是ORDER关系中的外键，这样用户可以将订单和提交该订单的顾客联系起来。ORDER_LINE有两个外键：Order_ID和Product_ID。这两个外键可以使用户将一个订单中的每一行信息和相关的订单及产品联系起来。

图5-4所示是这个数据库的一个实例，图中所示的四个表中都有示例数据。请注意外键是如何帮助我们将不同的表相互关联在一起的。为关系模型创建含有示例数据的实例有以下三个好处：

1) 示例数据提供一种便利的检查设计准确性的方法。

2) 示例数据有利于和用户讨论设计。

3) 可以使用示例数据开发原型应用，并测试查询。

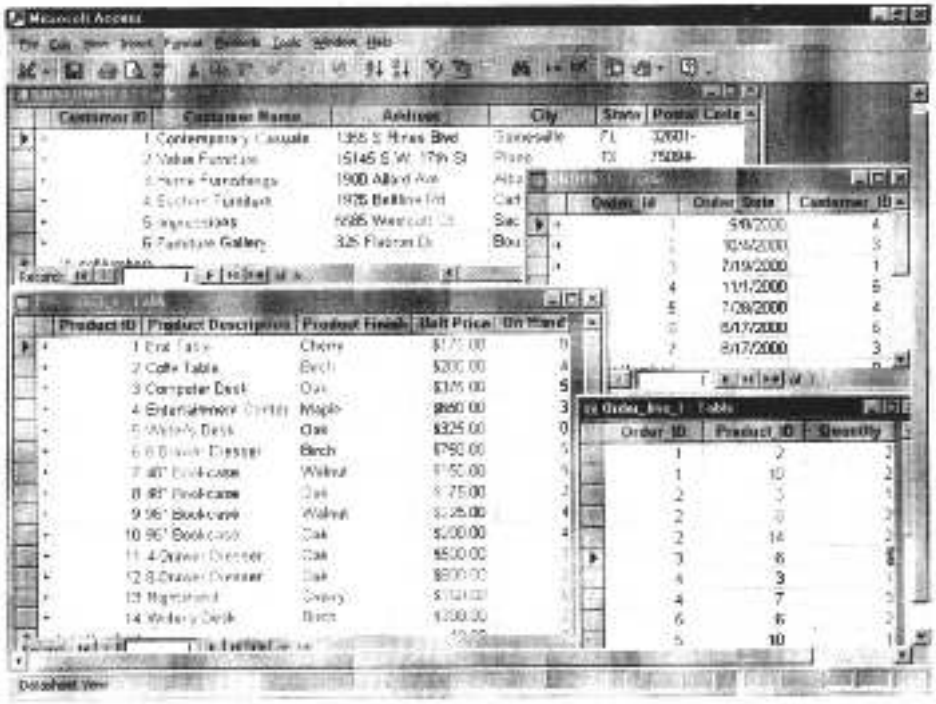


图5-4　关系模式的实例

5.4　完整性约束

关系数据模型包含有多种类型的约束（业务规则），它们用于维护数据库内数据的正确性和完整性。最主要的几种完整性约束包括域约束、实体完整性、参照完整性和动作断言。

5.4.1　域约束

关系中某一列的所有值都必须来源于同一个域。域是值的集合，这些值可以赋给一个属性。

一个域定义通常包含以下部分：域名、含义、数据类型、大小（或长度）、允许的取值或允许的范围（如果适用的话）。表5-1显示了与图5-3里属性相关的域的定义。

表5-1 选定的属性的域定义

属 性	域 名	描 述	域
Customer_ID	Customer_IDs	所有可能的顾客ID	字符：大小为5
Customer_Name	Customer_Names	所有可能的顾客名	字符：大小为25
Customer_Address	Customer_Addresses	所有可能的顾客地址	字符：大小为30
Customer_City	Cities	所有可能的城市	字符：大小为20
Customer_State	States	所有可能的州	字符：大小为2
Customer_Zip	Zips	所有可能的邮编	字符：大小为10
Order_ID	Order_IDs	所有可能的订单ID	字符：大小为5
Order_Date	Order_Dates	所有可能的订单日期	日期格式mm-dd-yy
Product_ID	Product_IDs	所有可能的产品ID	字符：大小为5
Product_Description	Product_Descriptions	所有可能的产品描述	字符：大小为25
Product_Finish	Product_Finishes	产品可能用的木材	字符：大小为12
Standard_Price	Unit_Prices	所有可能的单价	货币值：6位数字
On_Hand	On_Hands	所有可能的现有产品	整数：3位数字

5.4.2 实体完整性

实体完整性规则用来确保每个关系都有一个主键，而且主键的所有数据值都是有效的。特别是，它能保证每个主键的属性值都非空。

在某些情况下，没有办法为某个属性赋值。在以下两种情况中通常会出现这种情形：或者是没有可应用的数据值，或者是在赋值时还不知道数据值。例如，要求员工填写一张含有传真号码的登记表。如果该员工没有传真号，则只能不填，因为不适用于他。又比如，要求该员工填写前任雇主的电话号码，如果他不记得这个号码，则也只能不填，因为这个信息不知道。

利用关系数据模型，我们在碰到上述情况时可以为属性赋空值。**空值**（null）是一个当没有其他值可用，或是可用值未知时，可以赋给属性的值。实际上，null不是一个值，而是表示值的空缺。例如，null和数字0或是空串都不相同。在关系模型中引入null存在着争议，因为在某些情况下它会导致异常的结果（Date，1995）。另一方面，Codd主张使用null来表示缺失的值（Codd，1990）。

人们普遍认同的是，主键的值不可以为null。因此，**实体完整性规则**（entity integrity rule）可以表述为：主键属性（或主键属性的部分）取值不可以为null。

5.4.3 参照完整性

在关系数据模型中，表之间的关联是使用外键来定义的。例如在图5-4中，CUSTOMER和ORDER表的关联是通过在ORDER表中将Customer_ID属性作为外键来实现的。这就要求我们在向ORDER表插入一个新的行时，必须确保下订单的顾客已经存在于CUSTOMER表中。检查图5-4中表ORDER里的行，可以看到每个订单上的顾客号都已经出现在CUSTOMER表里。

参照完整性约束（referential integrity constraint）是在两个关系的行之间维持一致性的规则。这个规则要求，如果一个关系中有外键，则每个外键的取值必须与另一个关系的主键取值匹配，或者必须为null。读者可以检查图5-4中的表，看其是否满足参照完整性约束的要求。

利用关系模式的图形表示，可以方便地表示出参照完整性约束所必须遵从的关联。图5-5显示了图5-3的关系的模式。每个外键都有一个箭头指向其相应的主键。在模式中，每个这样的箭头都必须有对应的参照完整性约束。

如何知道是不是允许外键取null值呢？可以查看与图5-4和图5-5相关联的E-R模型图3-22。在该图中，因为每个订单必须有一个顾客（在Submits关系上CUSTOMER的基数最小为1），所以ORDER关系的外键Customer_ID的值不可以为null。如果最小的基数是0，那么外键的取值就可以为null。在数据库定义时，外键是不是可以取null值必须被定义为外键属性的一个性质。

实际上，外键是不是可以为null的问题比根据E-R图建模要复杂的多，并且是目前最难以决定的。比如，当我们要删除一个已经下订单的顾客时，对订单数据有何影响？我们可能只关心销售情况，而不再关心这个顾客。有三种可能的处理方法：

1) 删除相关的订单（称为级联删除），此时我们不但丢失顾客的信息，还失去它的相关销售记录。

2) 禁止删除顾客信息，直至该顾客的所有订单信息已被先期删除（安全检查）为止。

3) 将外键的值设为null（这是一种例外情况，虽然订单在创建时其Customer_ID的值不可以为null，但是当相应顾客被删除时，Customer_ID的值却可以为null）。

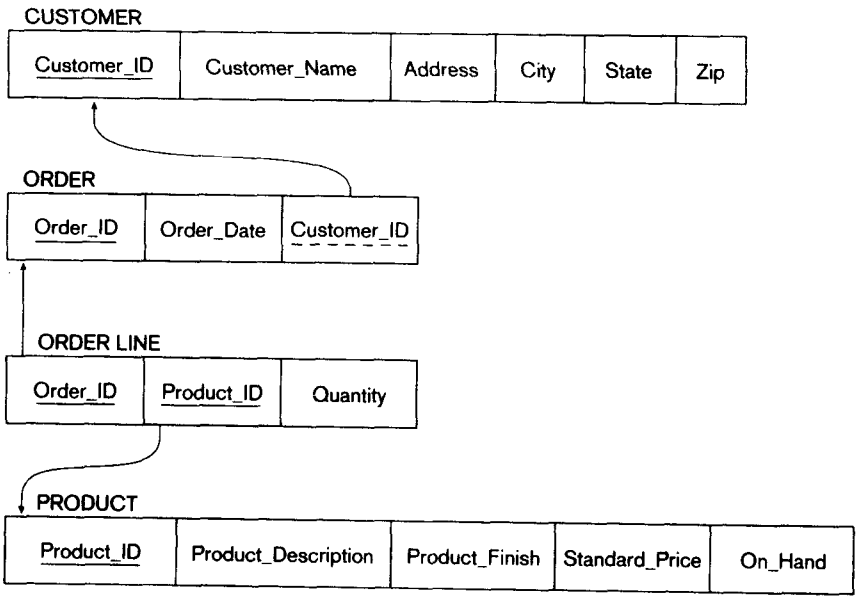


图5-5　参照完整性约束（松谷家具公司）

在第7章中介绍SQL数据库查询语言时，我们会看到这几种处理方法是如何实现的。

5.4.4　动作断言

在第4章中，我们讨论了业务规则，并介绍了一种称为动作断言的业务规则。例如，一个典型的动作断言是："只有当一个人购买了全赛季的票后，他才可以购买全明星赛的票"。定义和实现这种规则的方法有很多，我们会在后面的章节中讨论其中的一些方法。

5.4.5　创建关系表

在这一节中，我们创建图5-5所示的四张表的定义，这些定义是利用SQL数据定义语言中的CREATE TABLE语句创建的。实际上，这些表定义是在数据库开发过程中稍后的实现阶段才创建的。出于连续性考虑，我们在本章中给出这些示例表，同时也说明如何使用SQL语句来实现刚才讨论的完整性约束。

SQL的表定义在图5-6中给出。我们分别创建关系模式中的四个表（参见图5-5），并定义每个表中的属性。注意，属性的数据类型和长度来自前面给出的域定义（参见表5-1）。比如，关

系CUSTOMER中的属性Customer_Name被定义为VARCHAR（可变长字符串）数据类型，其最大长度为25。通过指定NOT NULL，可以防止给每个属性赋null值。

在每个表定义的最后，使用PRIMARY KEY子句为表定义主键。表ORDER_LINE说明如何定义一个复合的主键。在这个例子里，ORDER_LINE的主键是由Order_ID和Product_ID组成的复合主键。四个表中的每个主键属性都使用NOT NULL进行限制，这可以确保前面小节所讨论的实体完整性。同时应该注意，NOT NULL约束也可以用于非主键属性。

使用图5-5所示的图形化模式，可以很容易地定义参照完整性。每个从外键引出的箭头指向其关联关系的主键。在SQL的表定义中，每个FOREIGN KEY REFERENCES语句与这样些箭头相对应。在ORDER表中，外键CUSTOMER_ID引用表CUSTOMER的主键，主键名也是CUSTOMER_ID。虽然在这里主键和外键的名字是相同的，但这并不是必须的。比如我们也可以把外键属性CUSTOMER_ID更名为CUST_NO。但外键和相应的主键取值必须来自同一个域。

ORDER_LINE是一个具有两个外键的表的例子，它的外键分别引用ORDER和PRODUCT表中的主键。

```
CREATE TABLE CUSTOMER
        (CUSTOMER_ID                VARCHAR(5)          NOT NULL,
        CUSTOMER_NAME               VARCHAR(25)         NOT NULL,
        CUSTOMER ADDRESS            VARCHAR(30)         NOT NULL,
        CUSTOMER_CITY               VARCHAR(20)         NOT NULL,
        CUSTOMER_STATE              CHAR(2)             NOT NULL,
        CUSTOMER_ZIP                CHAR(10)            NOT NULL,
PRIMARY KEY (CUSTOMER_ID);

CREATE TABLE ORDER
        (ORDER_ID                   CHAR(5)             NOT NULL,
        ORDER DATE                  DATE                NOT NULL,
        CUSTOMER_ID                 VARCHAR(5)          NOT NULL,
PRIMARY KEY (ORDER_ID),
FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER (CUSTOMER_ID);

CREATE TABLE ORDER_LINE
        (ORDER_ID                   CHAR(5)             NOT NULL,
        PRODUCT_ID                  CHAR(5)             NOT NULL,
        QUANTITY                    INT                 NOT NULL,
PRIMARY KEY (ORDER_ID, PRODUCT_ID),
FOREIGN KEY (ORDER_ID) REFERENCES ORDER (ORDER_ID),
FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT (PRODUCT_ID);

CREATE TABLE PRODUCT
        (PRODUCT_ID                 CHAR(5)             NOT NULL,
        PRODUCT_DESCRIPTION         VARCHAR(25),
        PRODUCT_FINISH              VARCHAR(12),
        STANDARD_PRICE              DECIMAL(8,2)        NOT NULL,
        ON_HAND                     INT                 NOT NULL,
PRIMARY KEY (PRODUCT_ID);
```

图5-6 SQL表定义

5.4.6 良构关系

在讨论规范化问题之前，我们先解决一个问题：什么是良构关系？直观地说，**良构关系**（well-structured relation）应该有最小的冗余，并可以让用户在表中插入、修改和删除表中的行，且不出现错误和不一致性的情况。EMPLOYEE1（参见图5-1）就是这样一个关系。表中的每

一行都是关于一个员工的数据，对于一个员工信息的任何修改(比如修改其薪水)都局限在这一行内。与之相反，EMPLOYEE2（参见图5-2b）不是一个良构关系。如果检查表中的示例数据，会发现其中有大量的数据冗余。比如，员工号为100、110和150的员工，他们的Emp_ID、Name、Dept_Name和Salary都在两行中重复出现。因此如果要修改员工100的薪水，我们就必须同时修改表中的两行（对于某些员工，可能需要修改的行数会更多）。

当用户对表中数据进行更新时，表中数据的冗余可能导致错误或不一致的情况（称为**异常**(anomaly)）。有三种可能的异常：插入异常、删除异常和修改异常。

1. 插入异常

假如我们要为表EMPLOYEE2增加一个新的员工。这个关系的主键由Emp_ID和Course_Title组成（如前所述）。因此，要插入一个新行，用户必须为Emp_ID和Course_Title属性都提供值（因为主键属性不能为null值或是不存在）。这是一种异常，因为用户应该可以在不提供课程信息的情况下输入员工信息。

2. 删除异常

假如从表中删除员工号140的数据，则会导致以下信息的丢失：该员工在12/8/200X完成课程（Tax Acc）。事实上，这样会丢失该课程的某次授课在12/8/200X结束这一信息。

3. 修改异常

假如员工100增加了工资，我们必须在两行中记录这一修改（该员工在图5-2中出现两次），否则就会出现数据不一致的情况。

这些异常说明表EMPLOYEE2不是一个良构关系。这个关系的问题在于，它同时包含实体EMPLOYEE和COURSE的数据。在后面的内容中，我们将使用规范化理论将它分解为两个关系。其中一个关系是EMPLOYEE1（参见图5-1），另一个关系是EMP_COURSE，在图5-7中显示了该关系并带有示例数据。该关系的主键由属性Emp_ID和Course_Title组成，在图5-7中我们使用下划线标注这两个属性。读者可以检查图5-7中的关系EMP_COURSE没有前面提到的异常情况，因而该关系是良构的。

Emp_ID	Course_Title	Date_Completed
100	SPSS	6/19/200X
100	Surveys	10/7/200X
140	Tax Acc	12/8/200X
110	SPSS	1/12/200X
110	C++	4/22/200X
150	SPSS	6/19/200X
150	Java	8/12/200X

图5-7　EMP_COURSE

5.5　将EER图转化为关系

在逻辑设计中，我们可以将在概念设计中创建的E-R（EER）图转化为关系数据模式。这个过程的输入是第3、4章中所研究的E-R（EER）图，输出是在本章前面两节介绍的关系模式。

通过使用一组定义好的规则，可以方便地将E-R图转化（映射）为关系。实际上，许多CASE工具可以自动地完成转化中的许多步骤。但是由于以下的三个原因，还是应该了解这个过程的具体步骤：

1) CASE工具往往不能为更复杂的数据联系建模，如三元联系、超类型/子类型联系等。在

这些情况下，用户必须自行进行转化。

2) 有时候存在多个正确的转化方案，用户需要从中选择特定的一个解决方案。

3) 用户需要对CASE工具的转化结果进行质量检查。

在下面的讨论中，我们利用第3、4章给出的例子介绍转化的步骤。首先回忆一下在这两章中我们定义的三种实体：

1) 常规实体是独立存在的实体，通常代表现实世界中的对象、如人和商品等。常规实体使用单线矩形表示。

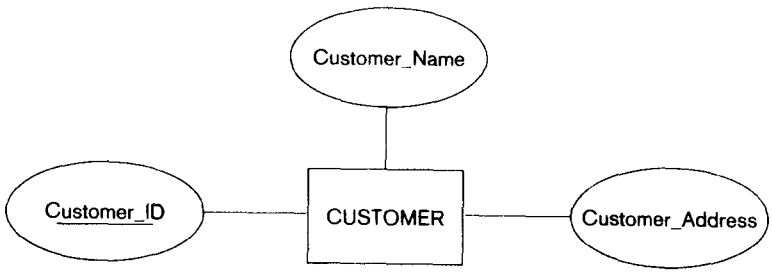
2) 弱实体必须与属主（常规）实体类型建立标识联系才可以存在。弱实体使用双线矩形表示。

3) 关联实体是由其他实体类型间的多对多联系而形成的。关联实体包含菱形联系符号的单线矩形表示。

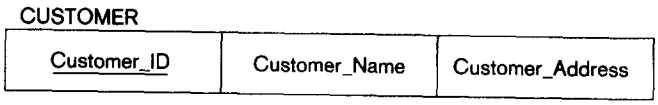
5.5.1 第1步：映射常规实体

ER图中的每一个常规实体可以转化为一个关系，通常关系的名字就是实体类型的名字。实体类型中的每一个简单属性成为关系的一个属性、实体类型的标识符就是相应关系的主键。读者应该验证一下以确定这个主键满足我们在第3章中所列出的标识符的期望性质。

图5-8a表示的是第3章中松谷家具公司的CUSTOMER实体类型（参见图3-22）。相应的CUSTOMER关系的图形表示出现在图5-8b中。为使图简化一些、在这个图和本节的随后的图中我们只给出每个关系的一些重要的属性。



a) CUSTOMER实体类型



b) 关系CUSTOMER

图5-8 映射常规实体CUSTOMER

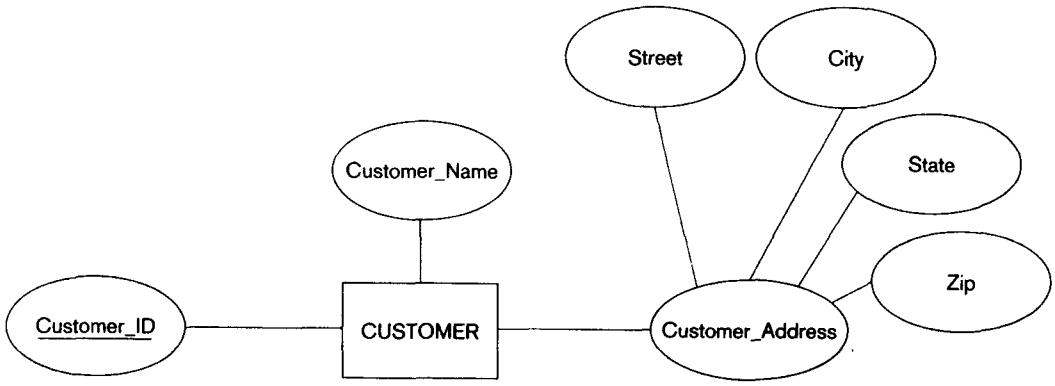
复合属性 当一个常规实体类型具有复合属性时，只有该属性的那些简单的组成属性包含在新的关系中。图5-9显示了图5-8中例子的一个变形，Customer_Address是一个由Street、City、State和Zip构成的复合属性（参见图5-9a）。这个实体被映射成图5-9b中的关系CUSTOMER，它只包含一个简单的地址属性。

多值属性 当一个常规实体类型包含多值属性时，要建立两个新关系（而不是一个）。第一个关系包含实体类型除多值属性以外的所有属性。第二个关系只包含两个属性，这两个属性共同构成第二个关系的主键。其中一个属性是第一个关系的主键，并是第二个关系的外键。另一个就是那个多值的属性。第二个关系的名字应该可以描述该多值属性的含义。

图5-10给出应用这个过程的一个例子，它来源于松谷家具公司的EMPLOYEE实体类型（参见图3-22）。如图5-10a所示，EMPLOYEE有多值属性Skill。图5-10b说明创建了两个关系，

第一个关系（EMPLOYEE）的主键为Employee_ID。第二个关系（EMPLOYEE_SKILL）有两个属性Employee_ID和Skill，它们组成该关系的主键。主键和外键间的联系用图中的箭头表明。

关系EMPLOYEE_SKILL不包含任何非键属性（也称为描述符）。每一行只是记录某个员工所具有的某项技能。此时可以建议用户给关系增加新的属性。比如，属性Years_Experience和Certification_Data就是适合该关系的新值。



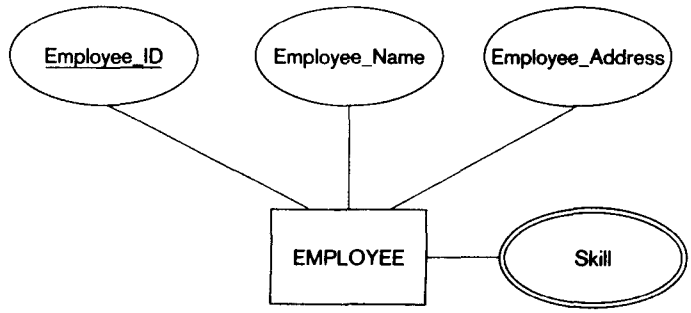
a) 带有复合属性的实体类型CUSTOMER

CUSTOMER

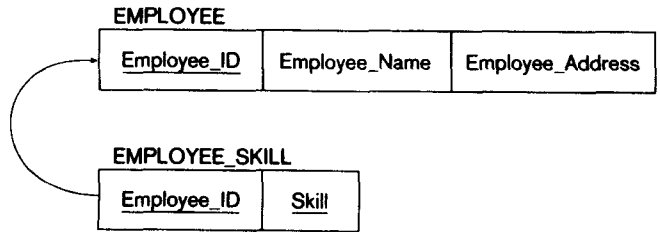
Customer_ID	Customer_Name	Street	City	State	Zip

b) 包含详细地址的关系CUSTOMER

图5-9　映射复合属性



a) 带有多值属性的实体类型Employee



b) 映射多值属性

图5-10　映射带有多值属性的实体

5.5.2 第2步：映射弱实体

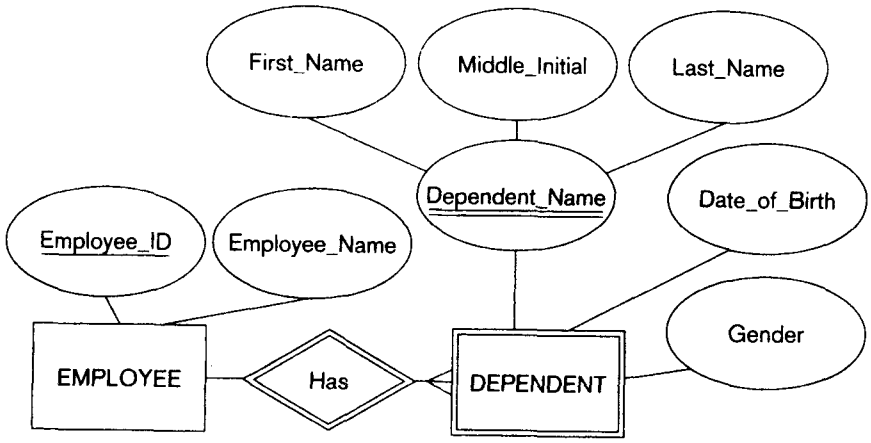
弱实体不具有独立的存在性，它的存在依赖于和一个称为属主的实体建立标识联系。弱实体类型没有完整的标识符，只具有一个称为部分标识符的属性，该属性可以标识属于同一个标识属主实例的弱实体的多次出现。

下面的过程假设已经在步骤1中为标识实体类型创建相应的关系。如果还没有创建，则应该使用步骤1中描述的方法创建关系。

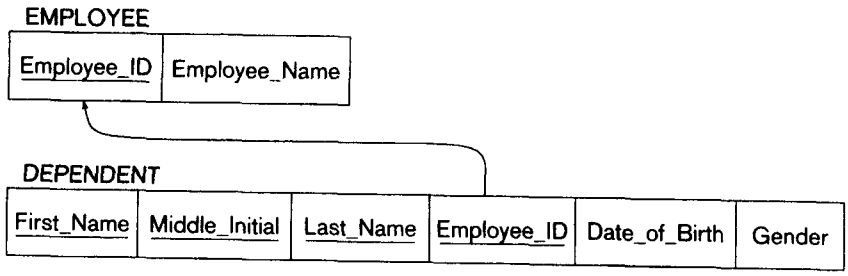
对于每一个弱实体类型，为其创建一个包含所有简单属性（或者复合属性的组成部分）的关系。然后将标识关系的主键作为新关系的外键，新关系的主键是由这个标识实体的主键和弱实体类型的部分标识符所组成的。

图5-11是这个过程的一个例子。图5-11中有弱实体类型DEPENDENT和其相应的标识实体类型EMPLOYEE，它们之间的标识联系为Has（参见图3-5）。请注意，DEPENDENT关系的部分标识符Dependent_Name是由First_Name、Middle_Initial和Last_Name组成的复合属性。我们假定，对于一个给定的员工，这些属性可以惟一标识他的亲属（职业拳击家George Foreman是个例外，他所有的孩子的名字都是和他相同的）。

图5-11b中是映射这个E-R图而得到的两个关系。DEPENDENT的主键由四个属性组成：Employee_ID、First_Name、Middle_Initial和Last_Name。Date_of_Birth和Gender是非键属性。图中使用箭头来表示外键和主键间的联系。



a) 弱实体DEPENDENT



b) 映射弱实体得到的关系

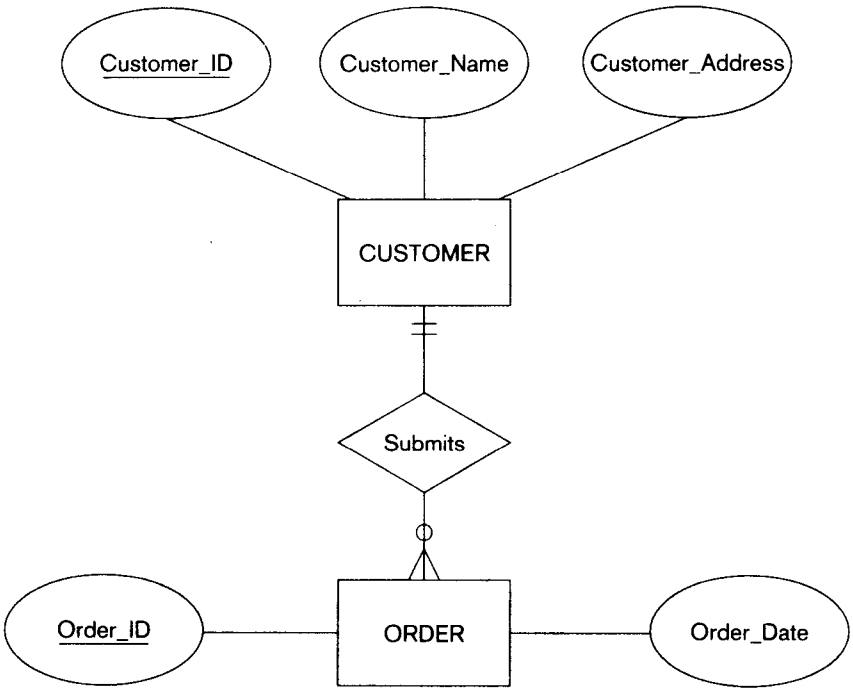
图5-11 映射弱实体的例子

5.5.3 第3步：映射二元联系

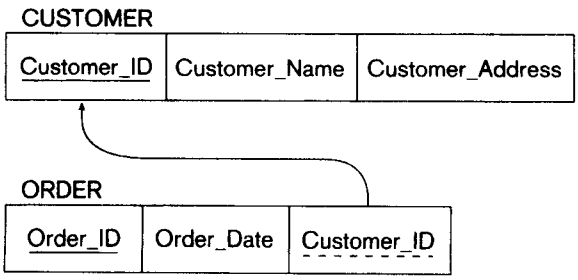
表示联系的过程不但取决于联系的度（一元、二元、三元），还取决于联系的基数。在下面的讨论中，我们分别描述其中比较主要的几种情况。

映射二元的一对多联系　对于每一个二元的1:M联系，首先使用步骤1中所介绍的方法为这两个参与联系的实体类型分别创建关系。接下来，将在联系中作为“一”方的主键属性（属性组）填加到联系中作为“多”方所对应的关系里，作为它的外键（可以这样记住这个规则：主键合并到多的一方）。

为说明这一简单过程，我们使用松谷家具公司（参见图3-22）中在CUSTOMER和ORDER之间的Submits联系，图5-12a显示了这个1:M的联系。图5-12b显示了使用上面的规则对这个带有1:M联系的实体类型映射所得到的结果。关系CUSTOMER的主键Customer_ID（1的一方）作为ORDER（M的一方）的外键，图中使用箭头标出这个外键联系。



a) CUSTOMER和ORDER之间的联系

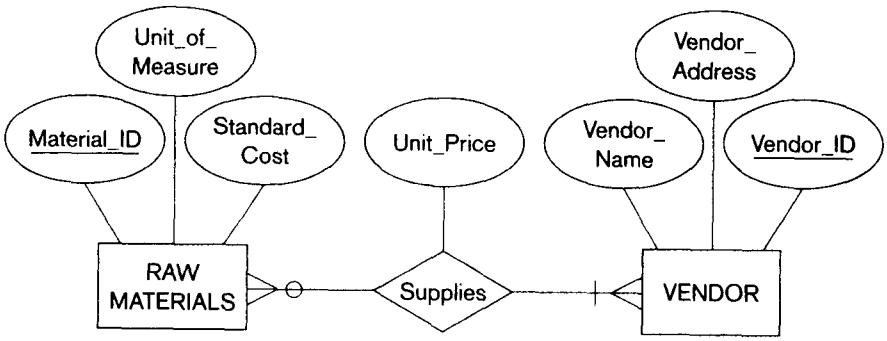


b) 映射联系

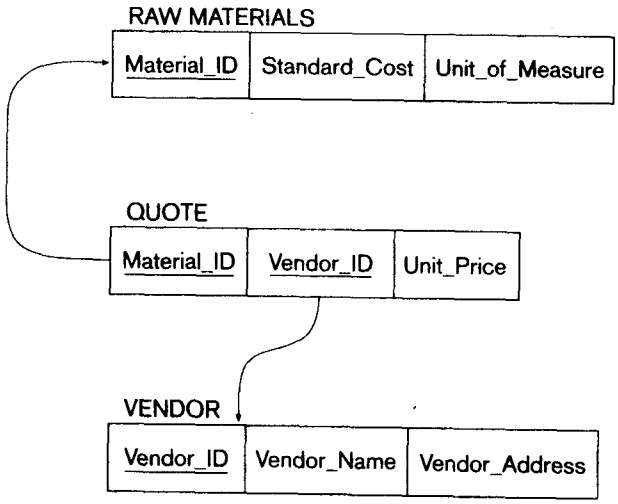
图5-12　映射1:M联系的例子

映射二元的多对多联系 假设在实体类型A和B之间有一个二元的多对多（M:N）联系。对于这个联系，我们创建一个新的关系C。C中包含参与联系的实体类型A 和B的主键属性作为它的外键属性，并组合成为C的主键。C中同时包含所有与这个M:N联系相关的非键属性。

图5-13是应用这个规则的例子。图5-13a是松谷家具公司（参见图3-22）的实体类型VENDOR和RAW MATERIALS之间的联系Supplies。图5-13b中是从这些实体类型和联系Supplies中所生成的三个关系（VENDOR、RAW MATERIALS和QUOTE）。首先为两个常规实体类型VENDOR和RAW MATERIALS分别创建关系，然后为联系Supplies创建关系（即图5-13b中的QUOTE）。关系QUOTE的主键是由Vendor_ID和Material_ID组成的，它们分别是VENDOR和RAW MATERIALS的主键。如图所示，这些属性在QUOTE中是外键，但却是其所在关系的主键。QUOTE同时包含非键属性Unit_Price。



a) 多对多联系（M:N）



b) 映射所得到的三个关系

图5-13 映射M:N联系的例子

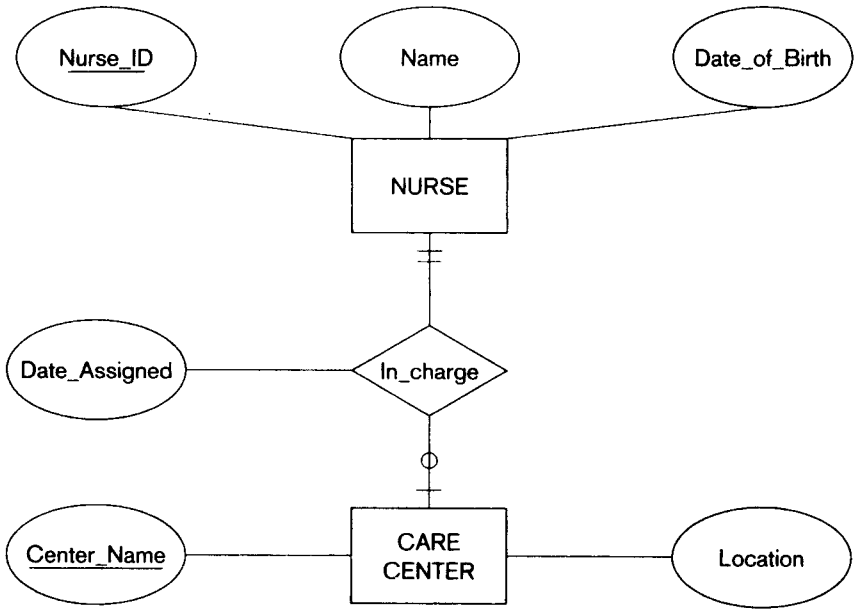
映射二元的一对一联系 二元的1:1联系可以看作是1:M联系的特殊情况。将这种联系映射为关系需要两个步骤。首先创建两个关系，分别针对参与联系的实体类型；然后将其中一个关系的主键加到另一个关系中作为它的外键。

通常来说，在一个1:1联系中，联系的一方总是可选的，而另一方是强制的（可以参考图3-1中对这些术语的表示方法）。应该将联系中必须出现一方的主键加到可选一方的关系中来作

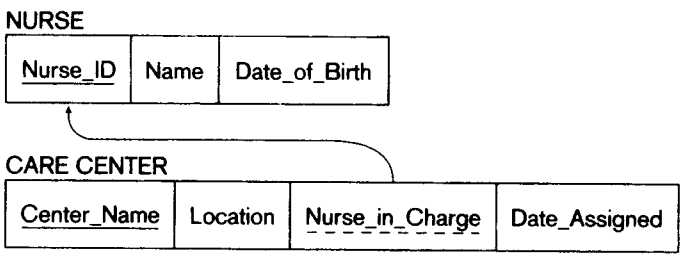
为外键，这样就可以避免在外键属性中存储null值的问题。所有与联系相关的属性也要被添加到外键所在的关系中。

图5-14给出应用这一过程的例子。图5-14a显示了实体类型NURSE和CARE CENTER之间的二元1:1联系。每一个诊疗中心必须有一个负责该中心的护士，所以从CARE CENTER到NURSE的联系是强制的，而从NURSE到CARE CENTER的联系则是可选的（因为某个护士既可能负责该中心也可能不负责该中心）。联系In_Charge中有属性Date_Assigned。

图5-14b给出将这个联系映射为关系的结果。关系NURSE和CARE CENTER分别根据实体类型创建。由于CARE CENTER是可选的联系参与者，所以为它添加外键，这里外键的名称是Nurse_in_Charge。它和Nurse_ID有相同的域，与主键之间的联系也在图中标出。属性Date_Assigned也位于CARE CENTER关系中，而且它不允许有null值。



a) 二元的一对一联系



b) 关系的结果

图5-14　映射二元的一对一联系

5.5.4　第4步：映射关联实体

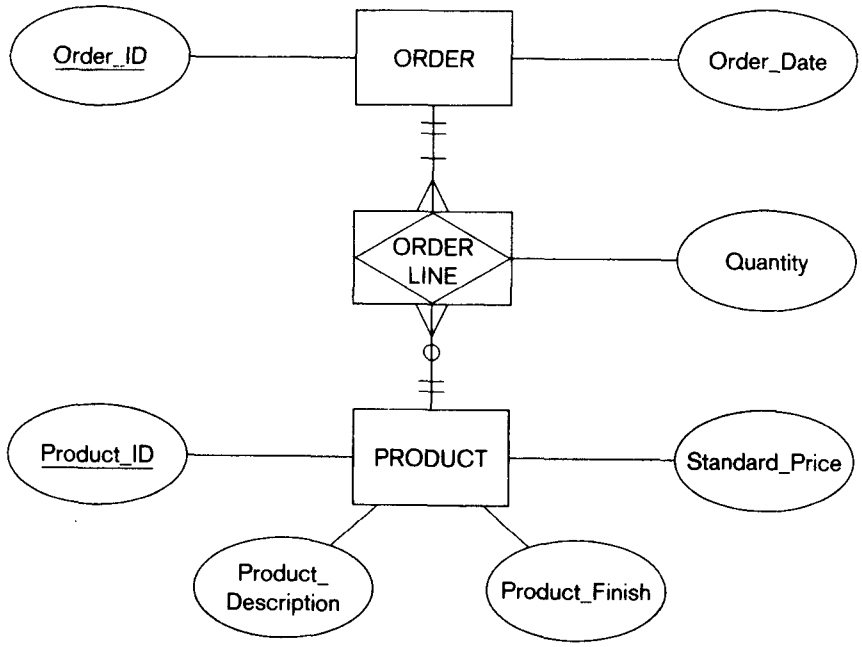
在第3章中曾提到，当数据建模人员遇到多对多联系时，他可以选择在E-R模型中将该联系建模为相应的关联实体。当最终用户更愿意将这种联系看作是实体类型，而不是M:N联系时，这种方法尤为合适。映射关联实体所涉及的步骤和前面步骤3中映射M:N联系的步骤实质上是

类似的。

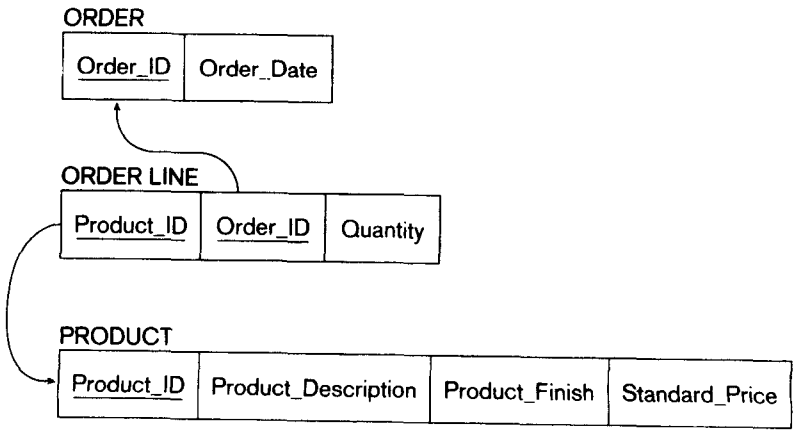
首先创建三个关系：分别针对参与联系的两个实体类型和关联实体本身。我们称对应于关联实体的关系为关联关系（associative relation）。接下来的步骤取决于在E-R模型中是不是为关联实体赋予标识符。

没有赋予标识符 如果没有赋予标识符，则默认情况下关联关系的主键由其他两个关系的主键属性构成，这些属性在关联关系中同时作为外键来引用另外的两个关系。

图5-15是这种情况的一个例子。图5-15a中的关联实体ORDER LINE链接松谷家具公司（参见图3-22）中实体类型ORDER和PRODUCT之间的关联。图5-15b显示映射后生成的三个关系，注意，这个例子的结果和图5-13中M:N映射的结果相类似。



a) 关联实体



b) 得到的三个关系

图5-15 映射关联实体

赋予标识符　有时候，数据建模人员会在E-R模型上为关联实体类型赋予标识符（称为代理标识符或代理键），这通常出于以下两个原因：

1) 关联实体类型有最终用户所熟悉的自然标识符。

2) 默认标识符（由参与联系的实体类型标识符所构成）不能惟一标识关联实体的实例。

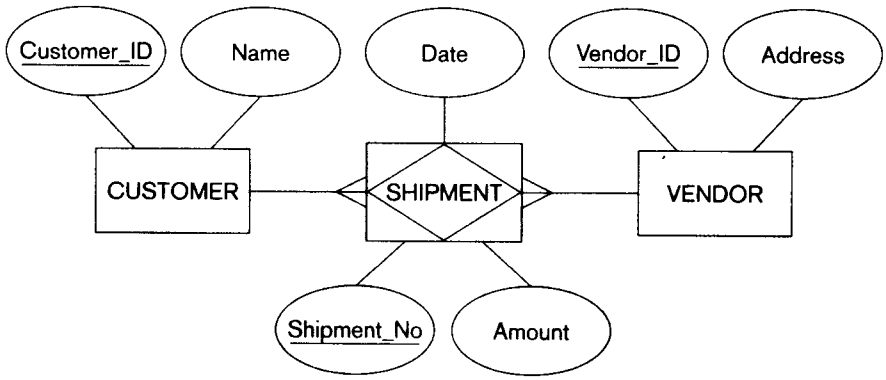
在这种情况下，关联实体的映射要作如下修改。首先，用前面所述的方法创建新的关系（关联关系）来表示关联实体，但应该将E-R图中所赋予的标识符（而不是默认的主键）作为其主键，然后将参与联系的实体类型的主键添加到该关联关系中作为外键。

图5-16是采用这种过程的例子。图5-16a中有关联实体类型SHIPMENT，它链接了实体类型CUSTOMER和VENDOR。用Shipment_No作为SHIPMENT的标识符是由于以下两个原因：

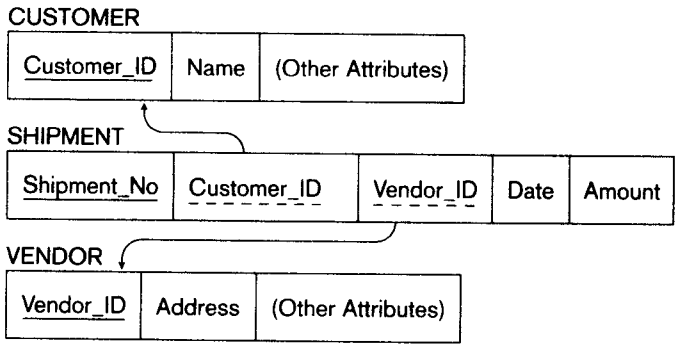
1) 最终用户熟悉Shipment_No这个实体的自然标识符。

2) 由Customer_ID和Vendor_ID所构成的默认标识符不能够惟一标识SHIPMENT实例。这是因为一个供货商可以为一个顾客多次送货，即使再添加属性Date也不能保证其惟一性，因为某个供货商可能在一天内多次供货，而代理键Shipment_No可以惟一标识每一次送货。

图5-16b是映射该实体到关系的结果，新的关联关系名为SHIPMENT。该关系的主键是Shipment_No，同时包含Customer_ID和Vendor_ID作为外键，以及属性Date和Amount是非键属性。



a) 关联实体（SHIPMENT）



b) 得到的三个关系

图5-16　映射带有标识符的关联实体

5.5.5　第5步：映射一元联系

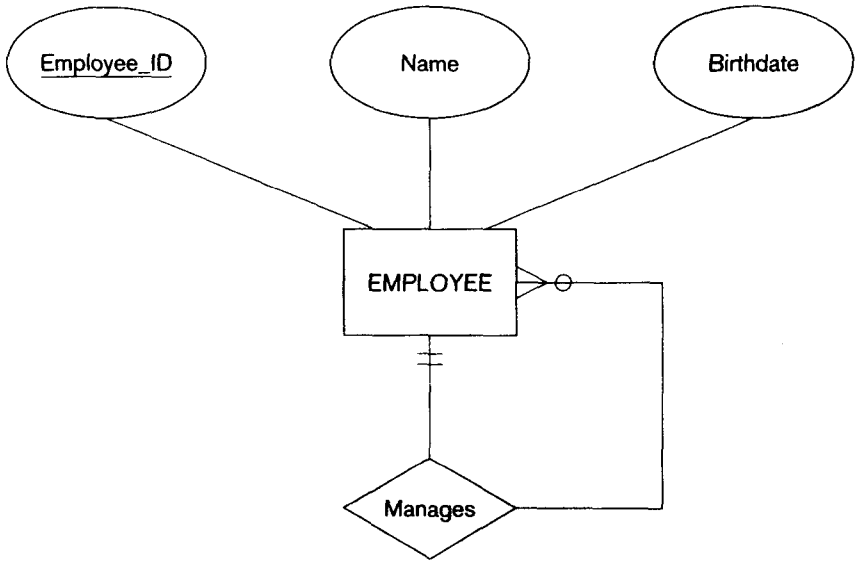
在第3章中，我们定义一元联系是在一个实体类型的实例间的联系，一元联系又称为递归

联系。一元联系最重要的两种形式是一对多和多对多，由于它们的映射方法有所不同，因此下面我们对这两种情况分别进行说明。

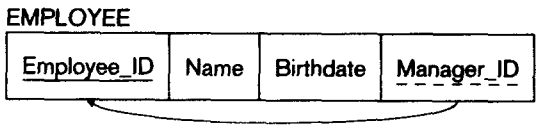
一元的一对多联系 首先使用步骤1的方法将一元联系的实体类型映射到关系，然后在这个关系内添加一个引用主键值的外键属性（这个外键与主键必须有相同的域）。**递归外键**(recursive foreign key)是引用自身所在关系主键值的外键。

图5-17a显示了一个一元的一对多联系Manages，它将组织内的每一个员工和该组织内作为他的经理的另一个员工联系起来。每个员工只有一个经理，而一个员工可以管理零到多个员工。

图5-17b中的EMPLOYEE关系是映射实体和相应联系的结果，其中的递归外键名为Manager_ID，它与关系的主键Employee_ID有相同的域。关系中的每一行存储员工的如下信息：Employee_ID、Name、Birthdate和Manager_ID（就是该员工的经理的Employee_ID）。注意，Manager_ID是外键，它引用关系的主键Employee_ID。



a) 带有Manages联系的实体EMPLOYEE



b) 带有递归外键的关系EMPLOYEE

图5-17 映射一元的一对多联系

一元的多对多联系 对于这种类型的联系，需要创建两个关系：一个表示联系中的实体类型，另一个是关联关系，用于表示这个M:N联系本身。其中，关联关系的主键由两个属性构成，它们（不需要有同样的命名）的取值都来源于另一个关系的主键。联系的非键属性也加到关联关系中。

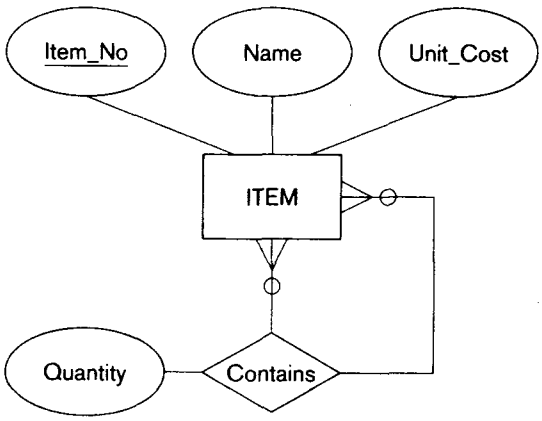
图5-18是映射一元的多对多联系的一个例子。图5-18a表示标准件（ITEM）间的bill-of-materials联系，即每个ITEM由许多个其他ITEM或COMPONENT组成（第3章中介绍过这个结构，图3-13给出过这个例子）。其中的联系（称为Contains）是M:N的，因为每个ITEM可以包

含多个组成它的ITEM，反过来一个ITEM也可以用于组装多个ITEM。

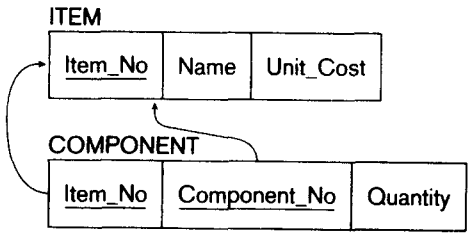
图5-18b是映射这个实体和相应联系后生成的关系。关系ITEM直接由相应的实体类型生成。关联关系COMPONENT的主键由两个属性（Item_No和Component_No）组成。关系中还有一个非键属性Quantity，用于表示组成ITEM的某种其他ITEM的数目。注意，Item_No和Component_No这两个属性都引用关系ITEM的主键（Item_No）。

我们可以很容易地查询以上的关系来决定某个标准件的组成情况。下面的SQL查询会列出Item_No为100的标准件的所有组件（和相应数量）：

```
SELECT Component_No, Quantity
FROM COMPONENT
WHERE Item_No = 100;
```



a) 材料单联系（M:N）



b) 关系ITEM和COMPONENT

图5-18　映射一元的多对多联系

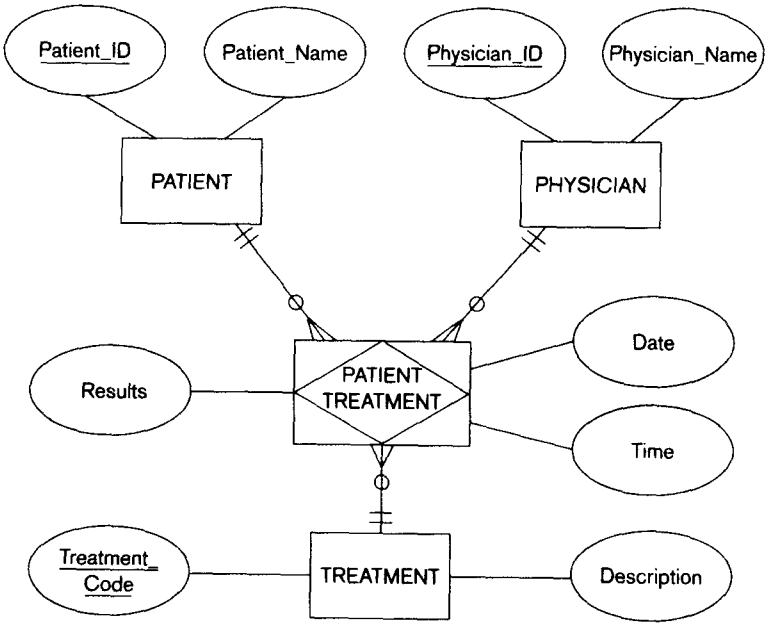
5.5.6　第6步：映射三元（多元）联系

在第3章中，定义三元联系为三个实体类型之间的联系，在该章中我们建议将三元联系转化为相应的关联实体以更准确地表示它们之间的参与约束。

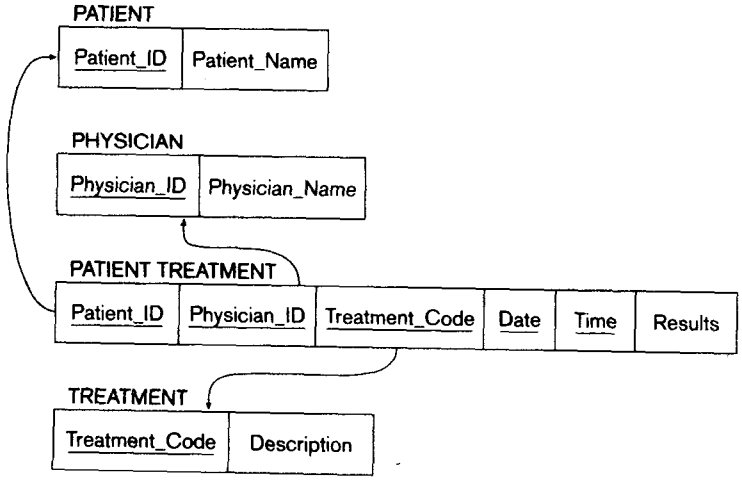
为映射链接三个常规实体联系的关联实体，我们创建新的关联关系。默认情况下，新关系的主键由参与该联系的三个实体类型的主键属性组成（在某些情况下，还需要增加其他属性以保证主键的惟一性）。在新关系中，这些属性作为外键引用相应实体类型的主键。其他属于关联实体类型的属性也要成为新关系中的属性。

图5-19是一个映射三元联系（表示为关联实体类型）的例子。图5-19a是E-R图的片段（或视图），它表示病人从医生处得到治疗。相应的关联实体类型PATIENT TREATMENT有属性Date、Time和Results，该实体类型的每个实例都有这些属性的值。

图5-19b是映射这个视图的结果。关系PATIENT TREATMENT的外键Patient_ID、Physician_ID和Treatment_Code分别是其他关系的主键，并是PATIENT TREATMENT主键的组成部分。需要注意的是，Patient_ID、Physician_ID和Treatment_Code并不能惟一标识某一次治疗，因为一个病人可能在同一个医生处接受过多次相同的治疗。那么将属性Date（或其他的属性）也作为主键的一部分是不是可以保证其惟一性呢？如果某个病人在某一天只从某个医生处接受一次治疗的话是可以的；但情况可能并非如此。例如，一个病人可能在早上接受一次治疗，然后在下午接受一次同样的治疗。为解决这个问题，我们将Time也作为主键的一部分。因此关系PATIENT TREATMENT的主键由图5-19b中的五个属性组成：Patient_ID、Physician_ID、Treatment_Code、Date和Time。关系中惟一的非键属性是Results。



a) 带有关联实体的三元联系



b) 映射三元联系

图5-19 映射三元联系

5.5.7 第7步：映射超类型/子类型联系

关系数据模型并不直接支持超类型/子类型联系。幸运的是，数据库设计人员可以采用多种其他的策略在关系模型中表示这种联系（Chouinard，1989）。出于讨论的需要，我们采用下面的方法，这也是最常使用的一种方法：

1) 为超类型和它的每一个子类型创建单独的关系。

2) 将子类型所共有的属性赋予超类型对应的关系中，包括主键。

3) 子类型对应的关系只包含它特有的属性以及超类型的主键。

4) 将超类型的一个（或多个）属性用作子类型鉴别符（第4章中已经讨论子类型鉴别符的作用）。

图5-20和图5-21是应用这种方法的例子。在图5-20中，EMPLOYEE是超类型，HOURLY EMPLOYEE、SALARIED EMPLOYEE和CONSULTANT是它的子类型（该例子在第4章中介绍过，图5-20再现了图4-8）。其中属性Employee_Number是EMPLOYEE的主键，属性Employee_Type是子类型鉴别符。

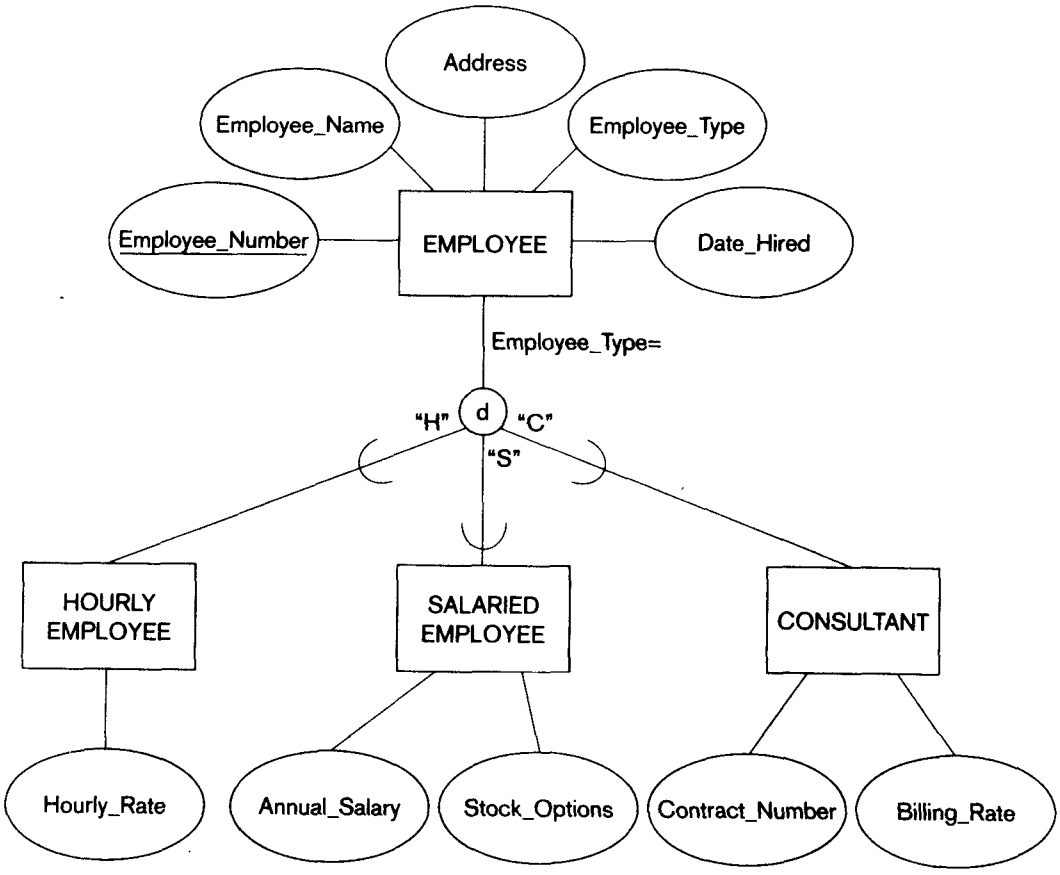


图5-20　超类型/子类型联系

图5-21是应用上述规则将图5-20的联系映射为关系的结果。超类型EMPLOYEE和它的三个子类型都有相应的关系。这四个关系的主键都是Employee_Number，但使用不同的前缀以加以区别。例如，S_Employee_Number是关系SALARIED_EMPLOYEE的主键名。子类型中的主键同时又是外键，它们对超类型主键加以引用（如图中箭头所示）。此外，每个子类型关系中也

包含其所特有的属性。

对于每个子类型，可以通过一个联结它和它的超类型的SQL语句产生一个包含它的所有属性（独有的和继承的）的关系。例如，假设我们想要显示包含SALARIED_EMPLOYEE的所有属性的表，就可以使用下面的语句：

```
SELECT *
FROM EMPLOYEE, SALARIED_EMPLOYEE
WHERE Employee_Number = S_Employee_Number;
```

虽然我们还没有正式学习SQL语句，但是从命令中可以很容易看出，该语句会连接关系EMPLOYEE和SALARIED_EMPLOYEE表，产生包含它们所有属性的结果表。

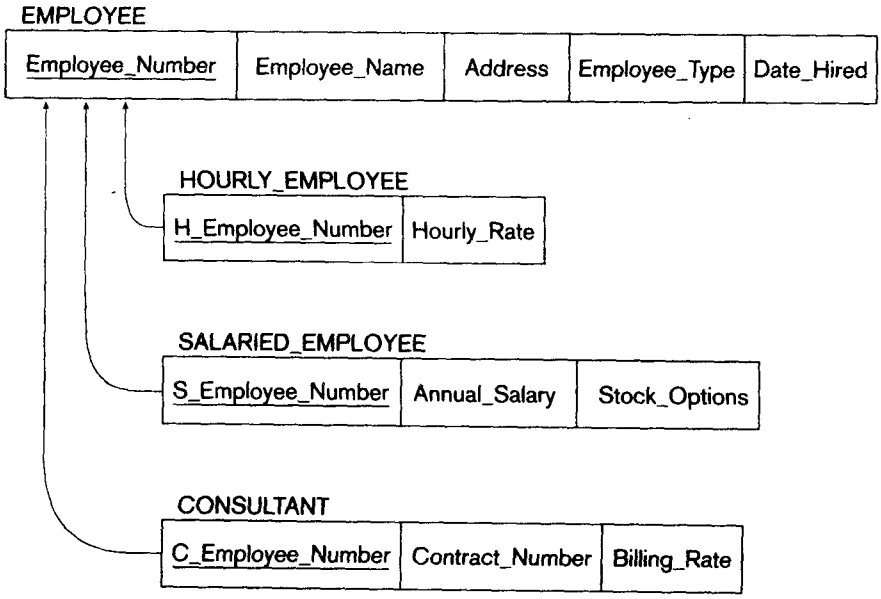


图5-21 将超类型/子类型联系映射成关系

5.6 规范化介绍

规范化是用来确定将哪些属性组合在一个关系中的过程。在第3章和第4章的概念数据建模中，读者都是利用直觉来为实体类型分配属性的。在本章的前面部分，我们学习了如何将E-R模型转化为关系。在进一步进行物理设计以前，我们需要一种方法来确认逻辑设计的有效性。规范化是用来确认和改进逻辑设计的主要工具，它可以确保设计满足特定的约束，以避免数据的无谓冗余。

我们在前面已经讨论过良构关系，但是我们还需要更形式化地定义良构关系，并介绍良构关系的设计步骤。**规范化**(normalization)是将含有异常的关系分解为更小的，良构的关系的过程。例如，我们可以使用规范化的原则将关系EMPLOYEE2（含有冗余）转化为关系EMPLOYEE1（参见图5-1）和关系EMP_COURSE（参见图5-7）。

5.6.1 规范化的步骤

规范化可以分为几个阶段来实现和理解，每一个阶段都对应有一个范式（参见图5-22）。**范式**（normal form）是根据函数依赖（或是属性间联系）对关系应用某些简单准则后所得到的关系的状态。我们在本节对这些范式进行简单的介绍，在以后的小节中会有更详细的说明。

1) 第一范式　任何多值属性（又称重复组）都被去除，表中所有行与列交叉处都只有一个值（容许为null）（如图5-2b所示）。

2) 第二范式　所有部分函数依赖都被去除。

3) 第三范式　所有传递依赖都被去除。

4) Boyce/Codd范式　所有其他的由函数依赖导致的异常都被去除。

5) 第四范式　所有多值依赖都被去除。

6) 第五范式　所有其他的异常都被去除。

在本章中我们讨论第一至第三范式，其他的范式会在附录B中介绍。

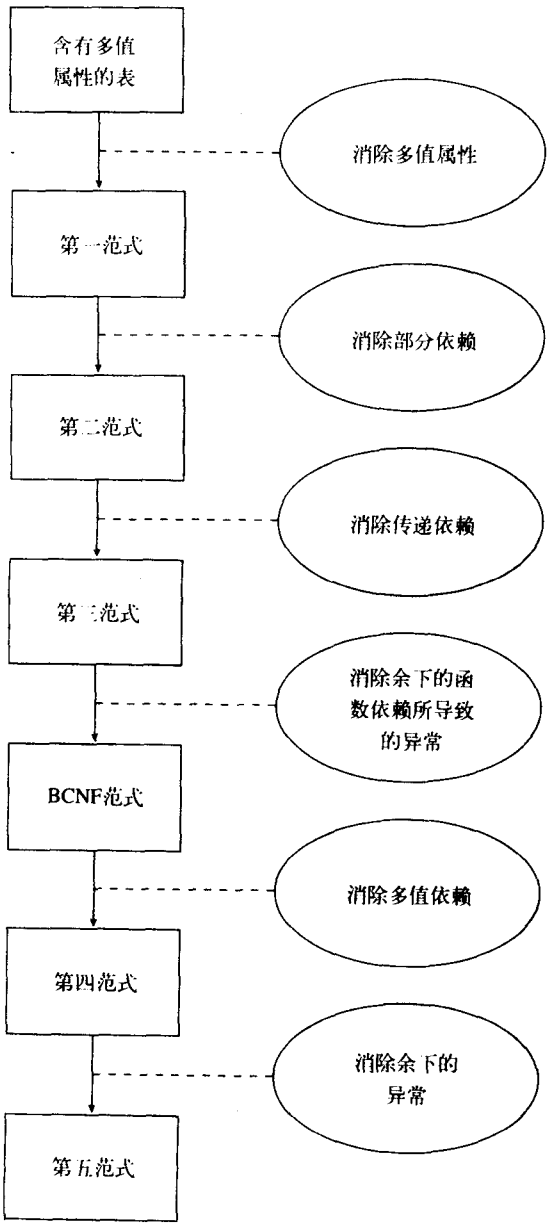


图5-22　规范化的步骤

5.6.2 函数依赖和键

规范化基于对函数依赖的分析。**函数依赖**（functional dependency）是存在于两个属性或两个属性集之间的约束。对任意关系R，当以下条件成立时，我们称属性B函数依赖于属性A：对于任意有效的A的实例，它的值惟一决定属性B的值（Dutka and Hanson，1989）。B对于A的函数依赖可以用箭头表示如下：A→B。一个属性可以函数依赖于两个（或更多）的属性，而不局限于单个属性。例如，考虑图5-7中的关系EMP_COURSE（Emp_ID，Course_Title，Date_Completed），我们将关系中的函数依赖表示如下：

```
Emp_ID, Course_Title→Date_Completed
```

上面的函数依赖表明，一个课程的结束日期完全取决于员工的ID和课程的名称。下面列出其他一些函数依赖的例子：

1) SSN→Name，Address，Birthdate 一个人的社会安全号完全决定他的名字、地址和生日。

2) VIN→Make，Model，Color 车辆的制造商、型号和颜色是由车辆的标识代码决定的。

3) ISBN→Title，First_Author_Name 书名和第一作者名是由书的国际标准书号所决定的。

决定因子 在函数依赖中，箭头左方的属性称为**决定因子**（determinant）。在上面的例子中，SSN，VIN和ISBN都是决定因子。在图5-7的关系EMP_COURSE里，Emp_ID和Course_Title组合起来就是决定因子。

候选键 一个**候选键**（candidate key）是可以惟一标识关系中某一行的属性或属性集。候选键必须满足以下性质（Dutka and Hanson，1989），这是前面所列举的主键6个性质的一部分：

1) 惟一标识 键值必须惟一标识关系中的每一行。这意味着任何非键属性都函数依赖于该键。

2) 非冗余 从键中删除任意属性，都不会破坏键的惟一标识的性质。

使用这个定义，我们在本章中所提到的两个关系里寻找候选键。关系EMPLOYEE1（参见图5-1）的模式为EMPLOYEE1（Emp_ID，Name，Dept_Name，Salary）。Emp_ID是关系中惟一的决定因子，所有其他属性都函数依赖于Emp_ID。所以Emp_ID是候选键，同时也是主键（因为没有其他候选键）。

我们使用图5-23中的符号来表示关系的函数依赖，图5-23a中给出关系EMPLOYEE1的表示。图中的横线描述函数依赖，从主键（Emp_ID）到这条横线有一条竖线。竖线上的箭头指向这些函数依赖于主键的非键属性。

再考查关系EMPLOYEE2（参见图5-2b）。注意，Emp_ID并不能惟一标识关系中的行（不同于EMPLOYEE1）。比如表中有两行的Emp_ID都是100。在这个关系中有两个函数依赖：

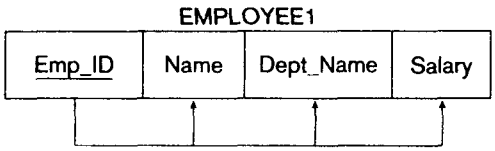
1) Emp_ID→Name，Dept_Name，Salary

2) Emp_ID，Course_Title→Date_Completed

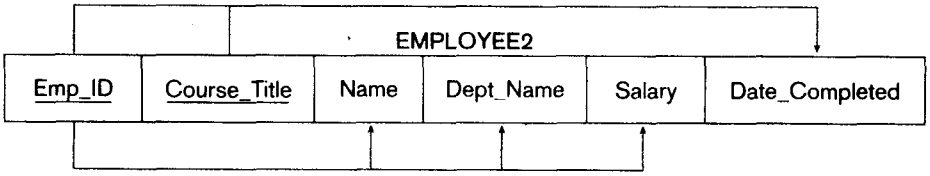
由此可知，Emp_ID和Course_Title的组合是关系EMPLOYEE2的惟一候选键（主键）。换言之，这个关系的主键是一个复合键。Emp_ID或是Course_Title都不能单独标识关系中的一行，根据上面的性质1，它们都不能单独作为候选键。读者可以查看图5-2b中的数据以确认Emp_ID和Course_Title的组合确实可以标识关系中的每一行。我们在图5-23b中表示这个关系里的函数依赖，可以看到Date_Completed是惟一函数依赖于由Emp_ID和Course_Title组成的主键的属性。

我们可以归纳决定因子和候选键之间的联系如下。一个候选键一定是决定因子，而决定因子不一定是候选键。例如，在关系EMPLOYEE2中，Emp_ID是决定因子，但它不是候选键。一个候选键是一个可以标识关系中所有其他（非键）属性的决定因子。一个决定因子可以是候选键（比如EMPLOYEE1中的Emp_ID），是复合候选键的一部分（如EMPLOYEE2中的

Emp_ID），或者是非键属性。我们在后面会给出这种情况的例子。



a) EMPLOYEE1中的函数依赖



b) EMPLOYEE2中的函数依赖

图5-23　函数依赖的表示

5.7　基本范式

在讨论过函数依赖和键后，下面进一步介绍第一到第三范式。我们同时描述信息库中概要数据的规范化问题。

5.7.1　第一范式

一个不含多值属性的关系符合**第一范式**（First Normal Form，1NF）。关系的第一个性质就要求行和列的交叉处必须为原子值，所以包含多值属性或重复组的表不是关系。

在前面小节中讨论如何将E-R模型映射为关系时，已经介绍了去除E-R模型的实体类型中多值属性的过程。所以，如果用户利用将E-R模型转化为关系的方法进行逻辑设计，那么得到的结果中应该不含多值属性。然而一些老的遗留系统，比如一些以COBOL语言编写的系统是支持多值属性的。由于读者可能会参与将这些系统转化为关系数据库的工作，因而了解如何去除多值属性仍是非常重要的。

在前面我们消除图5-2a中的重复组，将其转化为图5-2b中的关系EMPLOYEE2，此时EMPLOYEE2是满足第一范式的。在这个过程中，只需要将每列中由于多值属性而留空的单元用相应的值填上即可。

5.7.2　第二范式

如果一个关系满足第一范式，且任意非键属性都完全函数依赖于主键，那么这个关系符合**第二范式**（Second Normal Form，2NF）。也就是说没有非键属性函数依赖于部分（不是全部）主键。当一个符合第一范式的关系进一步满足以下任一条件时就符合第二范式：

1) 主键只包含一个属性（如关系EMPLOYEE1的属性Emp_ID）。

2) 关系中不含非键属性（关系中所有的属性都是主键的组成部分）。

3) 每一个非键属性都函数依赖于整个主键属性集。

EMPLOYEE2（参见图5-2b）是一个不满足第二范式的关系。关系的主键是由Emp_ID和Course_Title组合而成的，然而非键属性Name、Dept_Name和Salary都函数依赖于主键的部分属性（Emp_ID），而不函数依赖于Course_Title。这些依赖在图5-23b中表示。

部分函数依赖（partial functional dependency）是这样的一个函数依赖，关系中的一个或多个非键属性（如Name）函数依赖于部分（而不是全部）主键。关系EMPLOYEE2中的部分函数依赖导致关系中存在数据冗余，但当用户对表进行修改时，就会产生异常。

为使关系满足第二范式，我们将其分解为满足上面一个(或多个)条件的新关系。EMPLOYEE2被分解为下面的两个关系:

1) EMPLOYEE1（Emp_ID，Name，Dept_Name，Salary）这个关系满足条件1，因而符合第二范式（示例数据显示在图5-1中）。

2) EMP_COURSE(Emp_ID，Course_Title，Date_Completed) 这个关系满足条件3，因而符合第二范式（示例数据在图5-7中）。

读者可以验证这些新关系已经不存在原来EMPLOYEE2中的异常情况。

5.7.3 第三范式

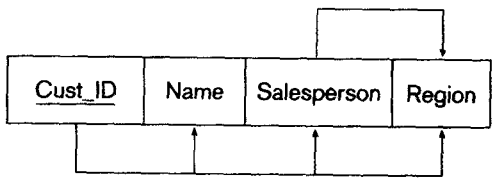
如果关系满足第二范式，且不含传递依赖，则该关系满足第三范式（Third Normal Form，3NF）。关系中的**传递依赖**（transitive dependency）是在两个（或多个）非键属性间的函数依赖。如在下面的例子中:

SALES（Cust_ID，Name，Salesperson，Region）（示例数据在图5-24a中）

SALES

Cust_ID	Name	Salesperson	Region
8023	Anderson	Smith	South
9167	Bancroft	Hicks	West
7924	Hobbs	Smith	South
6837	Tucker	Hernandez	East
8596	Eckersley	Hicks	West
7018	Arnold	Faulb	North

a) 关系SALES的示例数据



b) SALES内的传递依赖

图5-24 含有传递依赖的关系

图5-24b表示关系SALES内的函数依赖。Cust_ID为关系的主键，其他的属性都函数依赖于它。但是在关系内还存在着一个传递依赖（Region函数依赖于Salesperson，Salesperson函数依赖于Cust_ID），这会导致以下的更新异常:

1) *插入异常* 一个新的分配到北区的推销员（Robinson）不能被输入表格，直到给他指定某个顾客为止（因为向表中插入一行时必须为Cust_ID提供值）。

2) *删除异常* 如果顾客6837被删除，就会失去在东区工作的推销员Hernandez的信息。

3) *修改异常* 如果推销员Smith被重新分配到东区，需要修改表中的多行信息以反映这个变化（在图5-24a中有两行）。

这些异常是由于传递依赖所引起的。通过将关系SALES分解为图5-25a中的新关系就可以

消除传递依赖。可以看到，SALES里传递依赖的决定因子Salesperson⊖成为新关系SPERSON的主键，同时是SALES1的外键。通常情况下，图5-25中的规范化的关系在实现时用到的存储空间比图5-24a中所示关系所用的存储空间少。这是由于那些依赖数据（Region和其他Salesperson的数据）不再需要重复出现。由此可见，规范化在消除异常的同时，也可以减少冗余的数据存储。

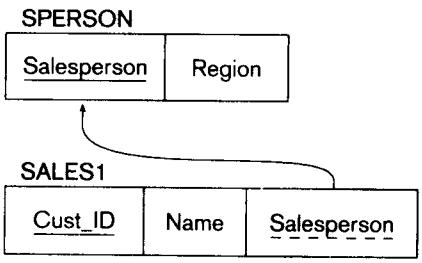
SALES1

Cust_ID	Name	Salesperson
8023	Anderson	Smith
9167	Bancroft	Hicks
7924	Hobbs	Smith
6837	Tucker	Hernandez
8596	Eckersley	Hicks
7018	Arnold	Faulb

SPERSON

Salesperson	Region
Smith	South
Hicks	West
Hernandez	East
Faulb	North

a) 分解SALES关系



b) 满足3NF的关系

图5-25　消除传递依赖

如图5-25b所示，消除传递依赖后的新关系现在满足第三范式。读者可以验证原来SALES中的异常现在在SALES1和SPERSON中已经不存在了。

传递依赖也可能在关系的属性集之间出现。比如，关系SHIPMENT(Snum, Origin, Destination, Distance)记录着运货的信息，包括起点、目的地和距离等信息（Dutka and Hanson, 1989）。图5-26a显示了这个关系的示例数据。图5-26b显示出关系SHIPMENT所包含的函数依赖。SHIPMENT关系中的主键是属性Snum（货号），由此可知它满足第二范式（读者自行考虑原因）。然而在关系中存在着一个传递依赖：属性Distance函数依赖于一对非键属性Origin和Destination，因此在SHIPMENT中存在着异常（作为练习，读者可以自行考虑5-26a来列举出插入异常、删除异常和修改异常的例子）。我们可以通过将SHIPMENT分解为以下的两个关系（都满足3NF）来消除传递依赖：

```
SHIPTO (Snum, Origin, Destination)
DISTANCES(Origin, Destination, Distance)
```

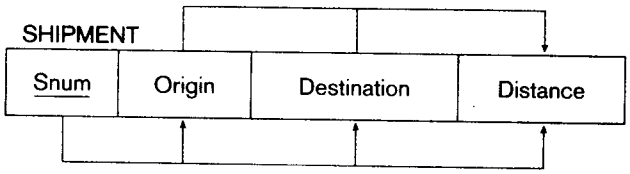
第一个关系分别说明某次运货的起点和目的地，第二个关系说明起点和目的地之间的距离，其示例数据在图5-26c中显示。

⊖ 由于Salesperson是一个智能业务键，所以并不适合作为主键。在第3章中我们对此进行过简要说明，在本章稍后会给予详细的讨论。

SHIPMENT

Snum	Origin	Destination	Distance
409	Seattle	Denver	1,537
618	Chicago	Dallas	1,058
723	Boston	Atlanta	1,214
824	Denver	Los Angeles	1,150
629	Minneapolis	St. Louis	587

a) 关系SHIPMENT及其示例数据



b) SHIPMENT中的函数依赖

SHIPTO

Snum	Origin	Destination
409	Seattle	Denver
618	Chicago	Dallas
723	Boston	Atlanta
824	Denver	Los Angeles
629	Minneapolis	St. Louis

DISTANCES

Origin	Destination	Distance
Seattle	Denver	1,537
Chicago	Dallas	1,058
Boston	Atlanta	1,214
Denver	Los Angeles	1,150
Minneapolis	St. Louis	587

c) 满足3NF的关系

图5-26 另一个传递依赖的例子

5.7.4 规范化概要数据

组织中支持制定管理决策的数据库系统中，通常包含运作型数据库中数据的一个子集或是概要数据。这些用于更高层次管理需求的“数据仓库”通常也需要进行规范化（至少规范化到某种程度）以避免运作型数据库中出现类似的异常问题。我们在第11章中介绍有关数据仓库的数据规范化问题。

5.8 合并关系

在前面的小节中，我们讨论如何将E-R模型转化为关系模型，然后介绍如何检查转化后生成的关系，判定其是否符合第三范式，并在需要的情况下对其进行规范化。

作为逻辑设计过程的一部分，规范化的关系可能由多个独立的E-R模型生成，也可能源自其他的用户视图，所以其中一些关系可能是冗余的，即对应相同的实体。在这种情况下，我们需要合并这些关系以消除冗余。本节讨论的内容是合并关系（又称为视图集成），了解关系合并的方法有以下三个重要的原因：

1) 在大型项目中，多个小组同时参与逻辑设计，所以有合并关系的要求。

2) 将已有的数据库系统与新的信息要求相结合时，往往需要集成多个不同的视图。

3) 在开发的生命周期中，可能产生新的数据需求，因此应该将新生成的关系与已有的关系合并。

5.8.1 例子

假设对一个用户视图建模后产生以下的3NF关系:

EMPLOYEE1 (Employee_ID, Name, Address, Phone)

对第二个用户视图建模后产生以下的关系:

EMPLOYEE2 (Employee_ID, Name, Address, Jobcode, No_Years)

考虑到这两个关系有相同的主键（Employee_ID），它们可能描述的是同一个实体，因而应该合并为一个关系。合并后得到的关系是:

EMPLOYEE (Employee_ID, Name, Address, Phone, Jobcode, No_Years)

注意，在合并后的关系中，前面两个关系共有的属性（如Name）都只出现一次。

5.8.2 视图集成产生的问题

在进行前面所述的关系集成时，数据库分析人员必须了解数据的含义，并准备解决在这一过程中可能产生的问题。在本节，我们简要分析一下在视图集成中可能发生的4个问题：同义词、异义、传递依赖和超类型/子类型联系。

同义词 在某些情况下，两个（或多个）属性名字不同，但是含义相同，因为它们描述的是一个实体的同一个特征。这样的属性称为**同义词**（synonyms）。比如，Employee_ID和Employee_No就可能是同义词。当合并含有同义词的关系时，应该（如果可能的话）和用户在命名上达成共识，使用标准的命名并去除其他同义词（另一种办法是使用其他的名字来替代所有同义词）。例如，考虑下面的关系:

STUDENT1(Student_ID, Name)
STUDENT2(Matriculation_No, Name, Address)

在这种情况下，分析员可能会发现Student_ID和Matriculation_No都是个人的社会安全号，因而是相同的属性（另一种可能性是它们都是候选键，但是只有一个能选为主键）。一个可行的解决办法是将两个属性名称改为标准名称，比如都叫做Student_ID，或者使用一个新的属性名（比如SSN）代替这两个同义词。假设使用后一种方法，合并两个关系后得到新关系:

STUDENT(SSN, Name, Address)

在出现同义词的情况下，用户往往希望使用不同的名字来访问同一个数据。这样他们可以使用熟悉的名称，并与他们的部门所使用的术语保持一致。**别名**（alias）是属性的其他的名字。很多数据库管理系统支持定义别名，它们可以与属性的主名字互换使用。

异义 一个可以有多种含义的属性称为**异义**(homonyms)。比如，术语“account”可以指银行的支票账户、储蓄账户、贷款账户，或者其他类型的账户（这样依据用途的不同，“account”可以代表不同的数据）。

在合并关系时，必须注意异义的情况。考虑下面的例子:

STUDENT1(Student_ID, Name, Address)
STUDENT2(Student_ID, Name, Phone_No, Address)

在与用户讨论后，分析人员发现，在关系STUDENT1中属性Address是指学生的校内地址，

而在关系STUDENT2中Address属性是指学生的永久（家庭）地址。为解决这个矛盾，就需要添加新的属性，所以合并后的关系为:

```
STUDENT(Student_ID, Name, Phone_No, Campus_Address, Permanent_Address)
```

传递依赖 当两个符合第三范式的关系合并成一个关系时，就可能产生传递依赖（本章的前面介绍过传递依赖的概念）。比如，考虑下面的两个关系:

```
STUDENT1(Student_ID, Major)
STUDENT2(Student_ID, Advisor)
```

由于STUDENT1和STUDENT2有相同的主键，所以两个关系可以合并为:

```
STUDENT(Student_ID, Major, Advisor)
```

但是如果每一门专业只有一位指导教师，此时指导教师就函数依赖于专业:

```
Major→Advisor
```

如果存在上面的函数依赖，则合并后的关系STUDENT满足第二范式，但不满足第三范式，因为它包含一个传递依赖。分析人员可以通过消除传递依赖来创建满足第三范式的关系（Major成为STUDENT的外键）:

```
STUDENT(Student_ID, Major)
MAJOR ADVISOR(Major, Advisor)
```

超类型/子类型联系 这些联系可能隐藏在用户视图或关系中。假设我们有以下两个关于医院的关系:

```
PATIENT1(Patient_ID, Name, Address)
PATIENT2(Patient_ID, Room_No)
```

初看起来，这两个关系可以被合并为一个PATIENT关系。但是，分析人员注意到有两种类型的病人：住院病人和门诊病人。关系PATIENT1事实上包含所有病人共有的属性，而PATIENT2的属性（Room_No）是住院病人所独有的特征。在这种情况下，分析人员会为这些实体生成超类型/子类型联系:

```
PATIENT(Patient_ID, Name, Address)
RESIDENT PATIENT(Patient_ID, Room_No)
OUTPATIENT(Patient_ID, Date_Treated)
```

要进一步地了解有关数据库设计中视图集成的问题，可以参阅Navathe、Elmasri和Larson的著作（1986）。

5.9 定义关系键的最后步骤

在第3章中，我们提供了一些选择标识符的准则：标识符的值不随时间变化，必须是惟一的和已知的，非智能的，为复合标识符使用一个属性代理。实际上所有这些准则在数据库实现（标识符成为主键，并被定义为物理数据库的一个字段）以前都不一定要满足。但在关系被定义为数据库的表之前，如果需要的话，应该对关系的主键作适当修改以满足这些准则。

最近，数据库专家（Johnston 2000）强调作为主键的规范准则。专家目前建议，主键在整个数据库范围内是惟一的（称为企业键，enterprise key），而不单单在它所在的关系表中惟一。这个准则使得主键更类似于面向对象数据库系统中的对象标识符（参见第14、15章）。根据这种建议，关系的主键就成为数据库系统的内部值，而不再具有任何业务含义。

一个候选主键（如图5-1的关系EMPLOYEE1中的Emp_ID以及图5-25里关系SPERSON中的Salesperson）只要曾经在组织内部使用过，就称为“业务键”，并可以在关系中作为非键属性。关系EMPLOYEE1和SPERSON（以及数据库内的任何其他关系）都会被赋予一个新的企业键属性（例如叫做Object_ID），该属性不具有任何业务含义。

为什么要添加这个额外的属性呢？使用企业键的一个重要原因是考虑到数据库的可演化性，即在数据库创建以后会再合并进新的关系。例如，考虑下面的两个关系：

EMPLOYEE (Emp_ID, Emp_Name, Dept_Name, Salary)
CUSTOMER (Cust_ID, Cust_Name, Address)

在这个例子中没有企业键，Emp_ID和Cust_ID无论是否具有智能特性，都不一定具有相同的格式、长度和数据类型。假如这个组织提升了其信息系统的处理需求，并意识到员工也可能成为顾客，那可以将员工和顾客都作为新类型PERSON的子类型。此时，系统会有如下的三个关系：

PERSON (Person_ID, Person_Name)
EMPLOYEE (Person_ID, Dept_Name, Salary)
CUSTOMER (Person_ID, Address)

此时，在数据库中的各个地方，同一个人的Person_ID都有同样的值。但是如果在PERSON关系建立以前，就已经为Cust_ID和Emp_ID选定值，那么显然不能够保证这两个值是一致的。此外，如果我们修改Cust_ID和Emp_ID的值，去匹配新的Person_ID，那又如何确保所有已有的Cust_ID和Emp_ID会和那些已经关联Person_ID的employee或customer取值不同呢？更糟糕的是，如果有其他的表，比方说和EMPLOYEE关联，则还必须修改这些表中的外键，从而造成连锁反应。保证关系的主键在整个数据库中惟一的办法是在一开始就为关系建立企业键，这样就永远不需要修改主键。

在我们的例子中，带有企业键的初始数据库（不包括PERSON）的情况如图5-27所示（包括5-27a（关系）和5-27b（示例数据））。在图中，Emp_ID和Cust_ID现在是业务键，而OBJECT是所有其他关系的超类型。OBJECT可以有属性，比如对象的类型名（在例子中为属性Object_Type），创建日期，最近修改日期和其他对象实例应有的系统内部属性。当需要创建新的关系PERSON时，数据库演化为图5-27c（关系）和图5-27d（实例数据）所示的样子。演化成包含PERSON的数据库仍然需要对于原有数据库进行某些修改，但不必修改主键的值、Name属性被移动到PERSON，因为将子类型和外键添加到EMPLOYEE和CUSTOMER以指向同一个Person的实例是很常见的。在后面的第7章中读者会了解到，给表的定义增加或删除非键的列，甚至是外键，都是容易的。相反，大多数数据库管理系统都不容许改变关系的主键，因为引起的外键的连锁反应代价很大。

本章小结

逻辑数据库设计是将概念数据模型转化为逻辑数据模型的过程。由于关系模型在现代数据库系统中的主流地位，所以也成为本章的重点。关系数据模型以称为关系的表的形式来表示数据，一个关系是已命名的二维数据表。关系的一个重要性质是它不能包含多值属性。

在本章中，我们介绍了逻辑数据库设计过程的主要步骤，它基于E-R图到规范化关系的转化。这个过程的三个步骤是：将E-R（EER）图转化为关系，规范化关系和合并关系。这个过程的结果是一组符合第三范式的关系，它们可以使用任何现代的关系数据库管理系统来实现。

E-R图中的每一个实体类型都可以转化为一个和它有相同主键的关系。在表示一对多联系时，给对应联系M一方实体的关系添加外键（它引用联系另一方所对应关系的主键）。通过增加一个新的关系来表示多对多的联系，这个关系的主键是个复合键，由参与联系的各个实体主键构成。

OBJECT (OID, Object_Type)
EMPLOYEE (OID, Emp_ID, Emp_Name, Dept_Name, Salary)
CUSTOMER (OID, Cust_ID, Cust_Name, Address)

a) 带有企业键的关系

OBJECT

OID	Object_Type
1	EMPLOYEE
2	CUSTOMER
3	CUSTOMER
4	EMPLOYEE
5	EMPLOYEE
6	CUSTOMER
7	CUSTOMER

EMPLOYEE

OID	Emp_ID	Emp_Name	Dept_Name	Salary
1	100	Jennings, Fred	Marketing	50000
4	101	Hopkins, Dan	Purchasing	45000
5	102	Huber, Ike	Accounting	45000

CUSTOMER

OID	Cust_ID	Cust_Name	Address
2	100	Fred's Warehouse	Greensboro, NC
3	101	Bargain Bonanza	Moscow, ID
6	102	Jasper's	Tallahassee, FL
7	103	Desks 'R Us	Kettering, OH

b) 企业键的示例数据

OBJECT (OID, Object_Type)
EMPLOYEE (OID, Emp_ID, Dept_Name, Salary, Person_ID)
CUSTOMER (OID, Cust_ID, Address, Person_ID)
PERSON (OID, Name)

c) 增加关系PERSON后的关系集

图5-27 企业键

OBJECT

OID	Object_Type
1	EMPLOYEE
2	CUSTOMER
3	CUSTOMER
4	EMPLOYEE
5	EMPLOYEE
6	CUSTOMER
7	CUSTOMER
8	PERSON
9	PERSON
10	PERSON
11	PERSON
12	PERSON
13	PERSON
14	PERSON

PERSON

OID	Name
8	Jennings, Fred
9	Fred's Warehouse
10	Bargain Bonanza
11	Hopkins, Dan
12	Huber, Ike
13	Jasper's
14	Desks 'R Us

EMPLOYEE

OID	Emp_ID	Dept_Name	Salary	Person_ID
1	100	Marketing	50000	8
4	101	Purchasing	45000	11
5	102	Accounting	45000	12

CUSTOMER

OID	Cust_ID	Address	Person_ID
2	100	Greensboro, NC	9
3	101	Moscow, ID	10
6	102	Tallahassee, FL	13
7	103	Kettering, OH	14

d) 增加关系PERSON后的示例数据

图5-27（续）

关系模型不直接支持超类型/子类型联系，但是我们可以通过为超类型和每个子类型创建单独的表（关系）来为这种联系建模。每个子类型的主键都应该与超类型的主键相同（或至少属于相同的域）。超类型必须拥有一个称为子类型鉴别符的属性，并使用它来区分每个超类型的实例属于哪个子类型。规范化的目的是导出良构关系，这些关系在修改和更新时不会产生异常（不一致或错误）。规范化基于对函数依赖的分析，函数依赖是两个属性（或两个属性集）间的约束。规范化有几个不同的阶段。当关系满足1NF时不含多值属性或重复组，满足2NF时不包含部分依赖，满足3NF时不包含传递依赖。我们可以使用图来表示关系中的函数依赖，以帮助我们在需要的情况下将关系分解为满足3NF的形式。更高层次的范式定义（超出3NF）会在附录B中讨论。

在合并关系的时候，我们必须注意同义词、异义、传递依赖和超类型/子类型联系等问题。

此外，当在数据库管理系统中定义关系之前，所有主键应该被定义为由单一属性构成的非智能键，最好是企业键。

本章复习

关键术语

别名	异常	候选键
复合键	决定因子	企业键
实体完整性规则	第一范式	外键
函数依赖	异义	范式
规范化	Null	部分函数依赖
主键	递归外键	参照完整性约束
关系	第二范式	同义词
第三范式	传递依赖	良构关系

复习问题

1. 定义以下的术语。

a. 决定因子	b. 函数依赖
c. 传递依赖	d. 递归外键
e. 规范化	f. 复合键
g. 关系	h. 范式
i. 部分函数依赖	j. 企业键

2. 将术语和它的定义匹配起来。

_____良构关系	a. 两个属性间的约束
_____异常	b. 非键属性间的函数依赖
_____函数依赖	c. 引用同一关系中的主键
_____决定因子	d. 不含多值属性
_____复合键	e. 不一致或错误
_____1NF	f. 含有少量冗余
_____2NF	g. 包含两个（更多）属性
_____3NF	h. 不含有部分函数依赖
_____递归外键	i. 不含传递依赖
_____关系	j. 函数依赖左侧的属性
_____传递依赖	k. 已命名的二维数据表

3. 比较以下的术语。

a. 范式；规范化	b. 候选键；主键
c. 函数依赖；传递依赖	d. 复合键；递归外键
e. 决定因子；候选键	f. 外键；主键

4. 总结关系的6个重要性质。
5. 指出候选键必须满足的两个性质。
6. 描述表中可能存在的3种异常。
7. 填空。

a. 一个不含部分函数依赖的关系满足_____范式。

b. 一个不含多值属性的关系满足_____范式。

c. 一个不含传递依赖的关系满足_____范式。

8. 什么叫良构关系？为什么良构关系在逻辑数据库设计中很重要？

9. 说明如何将下列E-R图中的元素转化为关系。

a. 常规实体　　b. 联系（1:M）

c. 联系（M:N）　　d. 联系（超类型/子类型）

e. 多值属性　　f. 弱实体

g. 复合属性

10. 简要描述在合并关系时经常出现的4种典型问题，并说明解决这些问题的常用方法。

11. 列举判定一个满足第一范式的关系也满足第二范式的3个条件。

12. 说明如何用SQL CREATE TABLE命令指定以下的完整性约束:

a. 实体完整性　　b. 参照完整性

13. 如何在关系数据模型中表示实体间的联系？

14. 如何在关系数据模型中表示一个一元的1:M联系？

15. 如何在关系数据模型中表示一个三元的M:N联系？

16. 关系的主键和这个关系的所有属性间的函数依赖有什么联系？

17. 什么情况下外键不能取null值？

18. 解释如何定义主键，以避免在数据库演化时可能导致的外键连锁修改？

问题和练习

1. 利用第3章中的E-R图完成下列练习。

I. 将图转化为可以表示参照完整性约束的关系模型（如图5-5所示）。

II. 对于每一个关系，用图表示其中的函数依赖（如图5-23所示）。

III. 如果下列图中所示的关系不满足3NF，将它们转化为满足3NF的关系。

a. 图3-8　　b. 图3-9b

c. 图3-11a　　d. 图3-11b

e. 图3-15a（有联系的版本）　　f. 图3-16

g. 图3-19

2. 利用第4章中的EER图完成下列练习。

I. 将图转化为可以表示参照完整性约束的关系模型（如图5-5所示）。

II. 对于每一个关系，用图表示其中的函数依赖（如图5-23所示）。

III. 如果下列图中所示的关系不满足3NF，将它们转化为满足3NF的情况。

a. 图4-6b　　b. 图4-7a

c. 图4-9　　d. 图4-10

e. 图4-18

3. 指出以下关系所满足的范式。如果关系不满足3NF，则将其分解为满足3NF的形式。函数依赖（除主键表示的以外）在合适的地方给出。

a. CLASS（Course_No，Section_No）

b. CLASS（Course_No，Section_No，Room）

c. CLASS（Course_No，Section_No，Room，Capacity）Room→Capacity

d. CLASS（Course_No，Section_No，Course_Name，Room，Capacity）
Course_No→Course_Name，Room→Capacity

4. 图5-28中是Millennium学院的班级列表，使用企业键将这个用户视图转化为一组满足3NF的关系。我们作出如下假设：

- 每个教师有惟一的办公地点。
- 每个学生有惟一的专业。
- 每门课程有惟一的名字。

MILLENNIUM COLLEGE
CLASS LIST
FALL SEMESTER 200X

COURSE NO.: IS 460
COURSE TITLE: DATABASE
INSTRUCTOR NAME: NORMA L. FORM
INSTRUCTOR LOCATION: B 104

STUDENT NO.	STUDENT NAME	MAJOR	GRADE
38214	Bright	IS	A
40875	Cortez	CS	B
51893	Edwards	IS	A

图5-28 班级列表（Millennium学院）

5. 图5-29是一个简化的信用卡系统的E-R图。有两种类型的卡账户：借记卡和信用卡。信用卡将每次消费的金额累加起来，每次消费都包括消费时间和消费的金额。

a. 开发一个关系模型（类似图5-5）。

b. 表示出所有的函数依赖（类似图5-23）。

c. 使用企业键给出一组满足3NF的关系。

6. 表5-2是关于供货商和他们供应的零件的示例数据。通过对用户的调研，我们发现零件号（而不是描述）惟一标识零件，而供货商的名称惟一地标识供货商。

a. 将这个表转化为满足第一范式的关系（名称为PART SUPPLIER），使用表中的示例数据来说明这个关系。

b. 列出PART SUPPLIER中的函数依赖，并指出其中的候选键。

c. 对于关系PART SUPPLIER，分别找出如下异常：插入异常、删除异常和修改异常。

d. 为关系PART SUPPLIER画出关系模型图（类似图5-23），并表示出函数依赖。

e. 这个关系满足第几范式？

f. 将PART SUPPLIER转化为满足3NF的形式。

表5-2 零件和供货商的示例数据

Part_No.	Description	Vendor_Name	Address	Unit_Cost
1234	Logic chip	Fast Chips	Cupertino	10.00
		Smart Chips	Phoenix	8.00
5678	Memory chip	Fast Chips	Cupertino	3.00
		Quality Chips	Austin	2.00
		Smart Chips	Phoenix	5.00

7. 表5-3中表示了大学里名为GRADE REPORT的关系，下面是这个关系里函数依赖的描述：

- Student_ID → Student_Name, Campus_Address, Major

- Course_ID→Course_Title，Instructor_Name，Instructor_Location
- Student_ID，Course_ID→Grade
- Instructor_Name→Instructor_Location

请根据这些信息完成下列练习：

a. 画出它的关系模型图（类似图5-23），并标出其中的函数依赖。

b. 它满足第几范式？

c. 将GRADE REPORT分解为满足3NF的形式。

d. 为生成的3NF作关系模型图，并标出其中的参照完整性约束。

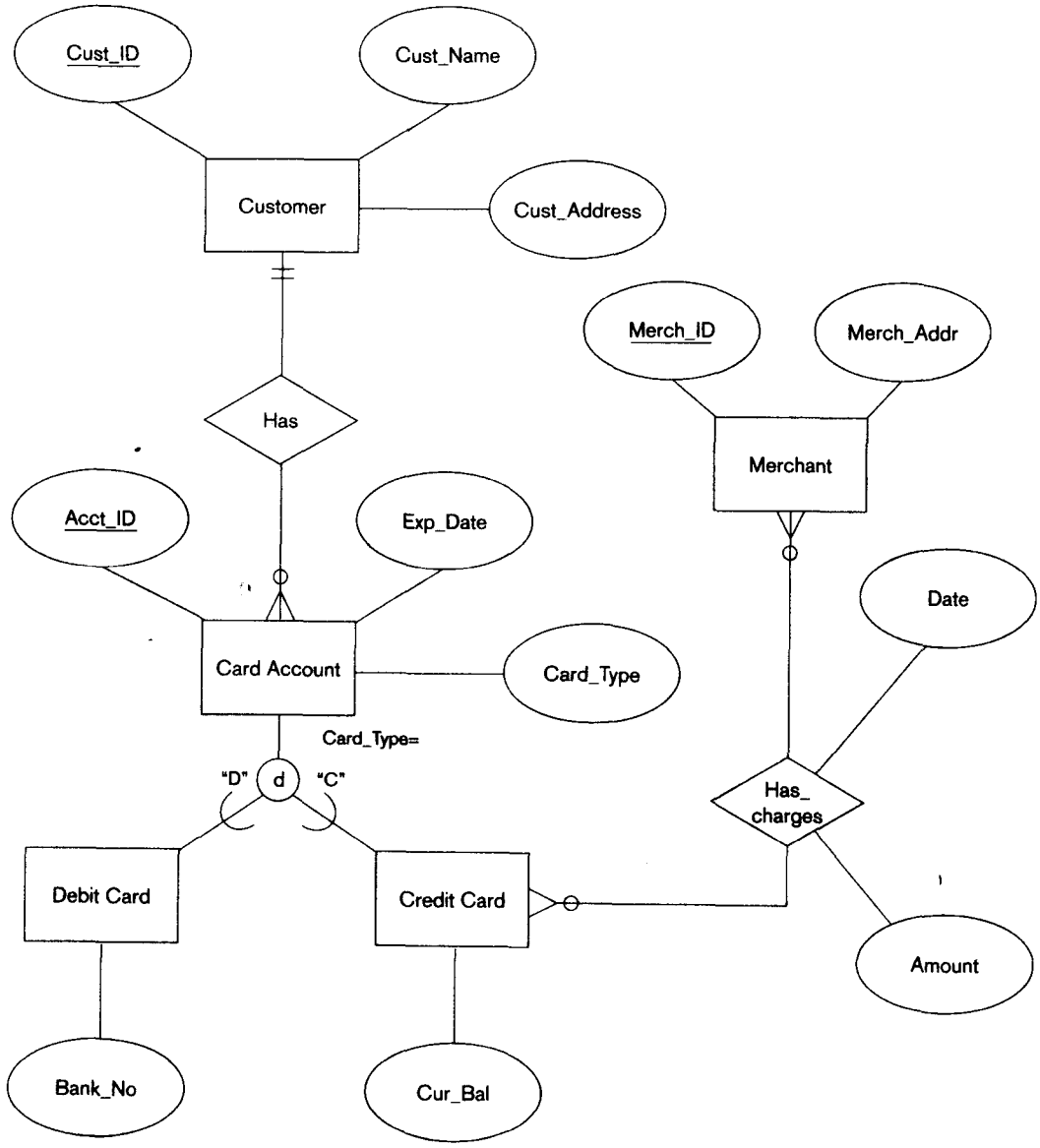


图5-29　银行卡的E-R图

表5-3 关系GRADE REPORT

Student_ID	Student_Name	Campus_Address	Major	Course_ID	Course_Title	Instructor_Name	Instructor_Location	Grade
168300458	Williams	208 Brooks	IS	IS 350	Database Mgt	Codd	B 104	A
268300458	Williams	208 Brooks	IS	IS 465	Systems Analysis	Parsons	B 317	B
543291073	Baker	104 Phillips	Acctg	IS 350	Database Mgt	Codd	B 104	C
543291073	Baker	104 Phillips	Acctg	Acct 201	Fund Acctg	Miller	H 310	B
543291073	Baker	104 Phillips	Acctg	Mktg 300	Intro Mktg	Bennett	B212	A

8. 将图3-15b（属性版本）转化为满足3NF的关系，再将图3-15b（联系版本）转化为满足3NF的关系。将它们和图5-10中的关系进行比较，通过比较这两组不同的3NF关系，你可以得到什么样的结论？

应用练习

1. 咨询企业内的系统和数据库设计人员，询问他们进行逻辑设计的步骤。他们是如何将概念数据模型（比如E-R图）转化为关系模型的？在转换过程中，CASE工具有何作用？他们使用规范化吗？如果使用，能达到什么级别？

2. 收集一些资料，如汽车修理店的销售凭据、顾客发票、信用卡的结账单等，然后完成以下的练习：

a. 列出其中的属性。

b. 列出属性间的函数依赖（作出必要的假设）。

c. 画出这个关系的模式图（类似图5-23），并标出函数依赖。

d. 将它分解为满足3NF的关系，并画出关系模式图（类似图5-5）。

3. 收集3个常见的基于PC的CASE工具的文档，并比较它们表示关系模型的表示方法。读者可以登录以下的网址来获取这些工具的联机信息：

a. Microsoft Access: www.microsoft.com/Visual Tools/

b. Erwin: www.logicworks.com

c. EasyER: www.esti.com

4. 寻找一些商业机构的表单或报表，如结账单、票据，或是你收到的其他文档。为这个表单或报表中的数据作出EER图，然后将其转化为满足3NF的关系。

参考文献

Chouinard, P. 1989. "Supertypes, Subtypes, and DB2." *Database Programming &Design* 2 (October): 50-57.

Codd, E. F. 1970. "A Relational Model of Data for Large Relational Databases." *Communications of the AGM* 13 (June): 77-87.

Codd, E. F. 1990. *The Relational Model for Database Management Version* 2. Reading, MA: *Addison-Wesley.*

Date, C. J. 1995. *An Introduction to Database Systems.* 6th ed. Reading, MA: Addison-Wesley.

Dutka, A. F., and H. H. Hanson. 1989. *Fundamentals of Data Normalization.* Reading, MA:

Addison-Wesley.

Fleming, C. C., and B. von Halle. 1989. *Handbook of Relational Database Design*. Reading, MA: Addison-Wesley.

Johnston, T. 2000. "Primary Key Reengineering Projects: The Problem" and "Primary Key Reengineering Projects: The Solution" available from www.dmreview.com.

Navathe, S., R. Elmasri, and J. Larson. 1986. "Integrating User Views in Database Design." *Computer* (January): 50-62.

进一步阅读

Elmasri, R., and Navathe, S. 1994. *Fundamentals of Database Systems*. 2nd ed. Menlo Park, CA: Benjamin Cummings.

Hoffer, J. A., J. F. George, and J. S. Valacich. 1999. *Modern Systems Analysis and Design*. 2nd ed. Reading, MA: Addison-Wesley.

Storey, V. C. 1991. "Relational Database Design Based on the EntityRelationship Model." *Data and Knowledge Engineering* 7: 47-83.

WEB资源

- http://databases.about.com/compute/databases/msub0045.htm 这是About Network上的一个页面，它介绍规范化的概念。
- http://oracle.com/tools/designer Oracle公司是数据库产品的主流厂商之一，它的产品包括CASE工具Designer。Designer可以帮助数据库分析人员绘制E-R图，并自动地将E-R图转化为满足3NF的关系。读者也可以去搜索一下其他CASE工具供应商的网站。
- www.troubleshooters.com/littslip/Itnorm.html 这是Steve Litt网站上的一个页面，它包含大量排除系统开发和程序设计中错误的小技巧。

项目案例：山景社区医院

在前面的几章中，我们已经介绍了山景社区医院的案例。在本章中，我们继续研究这个案例，并将重点放在对关系数据模型进行逻辑设计上。虽然这个医院还会继续评估新的面向对象和对象-关系技术，但是在未来的几年内，关系技术仍会在它的系统开发中占据主导地位。

项目问题

1) 为什么尽管新技术在不断出现，山景社区医院却仍然使用关系技术进行系统开发？

2) 山景社区医院应该在关系数据库设计中使用规范化吗？为什么？

3) 为什么对于医院来说，实体完整性和参照完整性很重要？

4) 数据规范化会涉及医院内的哪些人员？

项目练习

你的任务是为山景社区医院进行逻辑数据库设计。在第3章和第4章中，你已经开发出它的概念数据模型。具体来说，在前几章里你完成了如下准备工作：

- 为医院中主要实体和实体间联系绘制了E-R图（第3章的项目练习2）。
- 绘制了强调医院内人力资源的EER图（第4章的项目练习1）。

请基于其中的一个（或两个）概念设计进行逻辑设计。对于每一个逻辑设计，需要进行以下的工作：

1) 使用本章所介绍的技术将E-R（或EER）图映射到关系模型，在图中要为所有的主键属性加下划线，包括所有必要的外键，并指明参照完整性约束。

2) 为每个关系绘出函数依赖。

3) 将不满足3NF的关系分解为满足3NF的形式，并据此修改关系模型图。

4) 为每个关系创建企业键，并重新定义所有关系。

5) 为上一步创建的每个关系写出CREATE TABLE命令，对关系中每个属性的数据类型做出合理的假设。

6) 如果你为两章中的E-R图都作过逻辑设计，将它们合并为一个满足3NF的关系集。

第6章　物理数据库设计和性能

6.1 学习目标

学完本章后，读者应该具备以下能力：

- 定义下列关键术语：字段、数据类型、物理记录、页、块因子、非规范化、水平分割、垂直分割、物理文件、表空间、区、指针、文件组织、顺序文件组织、索引文件组织、索引、辅键、位图索引、联结索引、散列文件组织、散列算法、散列索引表、廉价冗余磁盘阵列（RAID）和条。
- 描述物理数据库设计的过程，设计目标和交付产品。
- 为逻辑数据模型中的属性选择存储格式。
- 通过平衡多种重要的设计因素，选择合适的文件组织方法。
- 描述三种主要的文件组织方法。
- 了解设置索引的目的和索引属性的选择要素。
- 将关系数据模式转化为高效率的数据库结构，了解何时和如何对逻辑数据模式进行非规范化。

6.2 引言

从第3章到第5章，我们已经学习了在数据库开发过程的概念数据建模和逻辑数据库设计阶段，如何对数据进行描述和建模。我们已经掌握了如何使用ERR表示方法、关系数据模型和规范化的技术来抽象化组织内的数据，同时表达数据的含义。但是这种表示方法不涉及如何处理和存储数据。物理数据库设计的目的是将数据的逻辑描述转化为存取数据的技术规格说明。设计的目标是设计合理的存储数据的方法，以提供足够的性能，并确保数据库的完整性，安全性和可恢复性。

物理数据库设计不包括实现文件和数据库（比如创建它们并向其中加载数据）。物理数据库设计产生相应的技术规格说明，程序员和其他从事信息系统建设的人员会在实现阶段使用它们，第7章到第11章会介绍实现阶段的内容。

本章会介绍高效的物理数据库设计的基本步骤，本章重点介绍独立、集中式的数据库系统，在第13章中会讨论如何设计存放在多个分布式地点上的数据库。本章会介绍如何估算用户所需的数据库中的数据量和使用数据的具体方式。本章还会介绍存储属性值的不同方法，以及如何根据需要选择合适的方法。读者会了解到为什么规范化的表不一定可以组成最好的物理数据文件，以及如何对数据进行非规范化以加快数据的检索速度。读者会了解不同的文件组织方式和索引的使用，索引对于加快数据检索速度非常重要，本章还会向读者介绍不同数据库体系结构之间的主要差别。

进行物理数据库设计时必须非常仔细，因为在这一阶段所作的决策会对数据库的可访问性、响应时间、安全性、用户界面友好性和其他类似的信息系统设计因素产生重大影响。数据库管理（参见第12章）对于物理数据库设计也有很大影响，我们会在第12章中进一步讨论一些高级的设计问题。

6.3 物理数据库设计步骤

在大多数情况下，一旦选定信息系统所使用的数据库管理技术，许多物理数据库设计的问题就已经被确定下来。因为很多组织对于操作系统、数据库管理系统和数据访问语言都有相应的标准，所以设计人员要讨论一些在可选技术中没有明确解决的问题。因此我们只集中讨论那些设计人员经常需要考虑的问题，以及一些对于某些类型的应用特别关键的问题，比如联机的数据捕捉和检索。

物理数据库设计的最主要目标是对数据的处理能力。在计算技术的单位计算成本（包括运算速度和存储空间）不断下降的今天，如何缩短用户与信息系统交互所花费的时间就显得非常重要。在下面的讨论中，我们重点讨论如何使对物理文件和数据库的处理更高效，而不是空间的使用效率。

物理文件和数据库的设计需要用到一些在前面步骤中所收集和产生的数据。物理文件和数据库设计所需的信息包括如下的一些需求:

- 规范化的关系，包括对量（volume）的估计。
- 每个属性的定义。
- 对于数据使用时间和使用地点的描述：输入、检索、删除和更新（包括频度）。
- 对于数据响应时间、安全性、备份、恢复、保持和完整性的预期及要求。
- 描述实现数据库所使用的技术（数据库管理系统）的信息。

在物理数据库设计中，要作出一些重要的决定，它们会对应用系统的完整性和性能造成影响，这些关键决策包括:

1) 为逻辑数据模型中的每个属性选择存储的格式（称为*数据类型*），以便使存储空间最小而数据完整性最佳。

2) 将逻辑数据模型中的属性分组成为*物理记录*。虽然关系表的列会很自然地当作物理记录的内容，但却不一定是最好的属性分组。

3) 将相似的结构化记录安排在二级存储（主要是磁盘）中，以便单个记录和记录组（称为*文件组织*）可以快速地存储、检索和更新。还需要考虑数据的保护和在发现错误后恢复数据的问题。

4) 选择存储和联接文件的方式（称为*索引和数据库体系结构*），使相关数据的检索效率更高。

5) 准备处理数据库查询的适当的策略，该策略可以优化性能并利用所定义的文件组织和索引。只有当查询和数据库管理系统可以有效地利用某种数据库结构时，这种数据库结构才可能显示出优越性。

数据量和使用分析

前面已经提到，数据量及数据的使用频度是物理数据库设计过程的重要参考因素。所以在逻辑数据库设计的最后阶段或是物理数据库设计的最初阶段，分析人员必须对数据库系统的规模大小及其使用模式进行估计。

通过在对应于逻辑数据库设计所产生的规范化关系的EER图添加符号，可以很容易地表示有关数据量和使用情况的信息。图6-1是松谷家具公司的一个简单库存数据库的EER图（没有标出属性）。这个EER图中是逻辑数据库设计得到的规范化关系，其原始的概念数据模型在图4-7b中显示。图4-7b中的多对多联系Supplies在图6-1中表示为关联实体QUOTATION，它是由这个联系所创建的关系。

图6-1中显示了数据量和访问频度的相关信息。比如，数据库内共有1000个PART。超类型PART有两个子类型，MANUFACTURED（40%的PART是自主生产的）和PURCHASED（70%的PART是购买的，由于某些PART同时具有两种子类型，所以总和超过100%）。松谷家具公司的分析人员估计，大约有50个常联系的SUPPLIER，平均每个供应商提供50个QUOTATION，所以共有2500个QUOTATION。图中的虚线箭头代表访问频度。比如，所有使用该数据库的应用，每小时大约访问PART数据200次。根据数据比例，每小时对PURCHASED PART数据访问140次，此外对PURCHASED PART数据还有另外60次直接访问。在所有对PURCHASE PART的200次访问中，80次同时访问QUOTATION数据；在这80次访问中，又有70次随即使用SUPPLIER数据。对于联机的和基于Web的应用来说，使用图要表示每秒的访问。在一天之内的不同时段，可能需要使用多张使用图来反映不同的使用模式。性能同时会受到网络规格说明的影响。

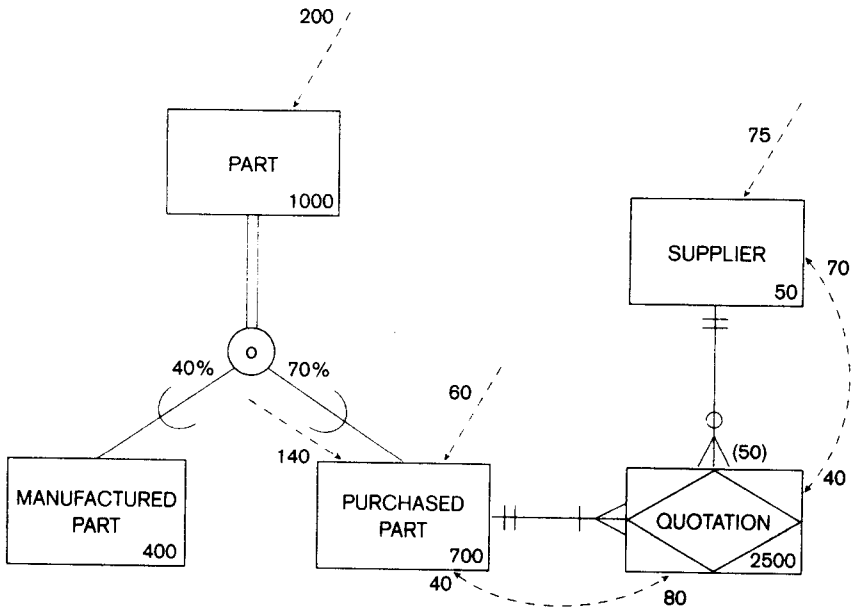


图6-1　复合使用图（松谷家具公司）

分析人员在系统开发过程的系统分析阶段中，通过研究当前和预期的数据处理和业务活动来生成有关数据量及使用频度的数据。数据量的大小代表业务的规模，在估算时至少需要考虑到未来几年可能的业务增长。访问频度的信息则是通过对事件的时间控制、事务量、并发用户的数量，报表及查询活动等进行分析而估算出来的。由于很多数据库系统支持即席访问，这种访问可能随时间的流逝发生很大的变化。在一个月、一周以及一天内已知的数据库访问可能会有高峰和低谷，访问的频度更为不确定，因此对频度的估计往往比对数据量的估计更难进行。幸运的是，在这个过程中通常并不需要精确的数值，更为重要的是数值的相对大小，以提示哪些问题最值得关注以得到可能的最优性能。比如，在图6-1中可以看到每小时对于QUOTATION的40次访问，都同时访问PURCHASED PART。这表明将这两个实体合并成一个数据库表（或文件）会获得更好的性能。这种合并规范化表的操作是非规范化的一个例子，我们会在后面详细讨论。

如果可以分析虚线所表示的访问路径的实质，则会有利于随后的物理数据库设计。比如，

如果知道在对PART数据的200次访问中，150次通过指定主键Part_No的值来寻找一个零件或一组零件（比如访问某个特定零件号的零件），而其余50次是通过属性Qty_on_Hand的值来访问合格的零件的（图6-1中没有给出）。这些更为精确的描述有助于为表选择索引，这是我们后面要讨论的一个重要课题。知道对数据的访问是否会导致数据被创建、检索、更新或删除也很有帮助。这些对于访问频度的精细描述可以用附加的符号标注在像图6-1一样的图上，或是在其他文档中用文本和表格的形式记录下来。

6.4 设计字段

字段（field）是可以被系统软件（如程序设计语言或数据库管理系统）识别的最小的应用数据单元。一个字段对应于逻辑数据模型的简单属性，所以一个字段代表复合属性中的一个属性。

在对字段进行设计时，最基本的决策就是确定代表该字段值的数据类型（或存储类型），数据库系统内建的数据完整性控制，以及DBMS如何处理这个字段可能丢失的值。其他的字段规格说明还包括数据的显示格式等，虽然它们也是整个信息系统规格说明的必要部分，但是我们在这里不予讨论，因为它们通常由程序而不是DBMS来处理。

6.4.1 选择数据类型

数据类型（data type）是系统软件（如DBMS）可识别的用于表示组织数据的具体编码方案。虽然位模式的编码方案通常是对用户透明的，但数据的存储空间和数据访问的速度却是物理数据库设计的重点。使用的特定数据库管理系统往往会决定用户采取哪一种方案。表6-1列举出Oracle 8*i* 数据库管理系统所提供的数据类型，这是一个典型的使用SQL数据定义语言和数据操纵语言的数据库管理系统。在其他数据库管理系统中，可能还提供一些用于货币、声音、图像和用户自定义的数据类型。

表6-1 Oracle 8i中的数据类型

数据类型	描 述
VARCHAR2	可变长的字符数据，最大长度为4000个字符。必须为其输入最大字段长度（比如VARCHAR2（30）表明字段的最大字段长度为30字符）。一个长度小于30字符的值只会占据它需要的空间
CHAR	定长的字符数据，最大长度为2000字符。默认的长度为1（比如CHAR（5）定义一个固定长度为5字符的字段，可以容纳长度为0～5个字符的值）
LONG	可以容纳最大4G字符的变长字符数据字段（例如用于存放用药说明或顾客建议）
NUMBER	从10^{-130}到10^{126}的正数和负数，可以指定精度（小数点两边的位数之和）和数值范围（小数点右边的位数）（比如，NUMBER（5）是一个最多5个数字的整数字段，而NUMBER（5，2）定义一个最多5个数字、小数点右边有2个数字的字段）
DATE	从公元前4712年1月1日到公元4712年12月31日的日期，其中包括世纪、年、月、日、小时、分钟和秒
BLOB	二进制的大对象，可以存储最大可达4G的二进制数据（比如照片或声音剪辑）

数据类型的选择要考虑到以下四个目标，它们在不同应用中的重要程度也不同：

1) 将存储空间最小化。

2) 表示所有可能的值。

3) 改善数据完整性。

4) 支持所有数据操纵。

选择正确的数据类型能够占用最小的空间，表示相关属性所有可能的值（但去除不合法的值），并支持相应的数据操纵（如数字数据类型可进行数学运算，字符类型可进行字符串操作）。概念数据模型中关于属性域约束的信息也有助于为属性选择合适的数据类型。达到数据类型以

上的四个目标需要做一些巧妙的安排。假设在一个DBMS中，某个数据类型的最大长度为2个字节，假设它足以存储表示销售数量的字段。但是当对所有销售数量字段求和时，2个字节可能就不够存储这个和值。如果DBMS使用同样的数据类型来存放其他数学运算的结果，就会产生严重的问题。此外，某些数据类型有其独有的处理功能，比如只有DATE类型上才能进行真正的日期计算。

编码和压缩技术

某些属性的取值是稀疏的，但是范围却又很广，因此会消耗大量的存储空间（大的数据字段意味着数据距离远，这会使数据的处理速度变慢）。可以通过为一个只能在几个可能值内选取的字段进行编码来节省空间。考虑图6-2中产品的Finish字段的例子。松谷家具公司只使用几种有限的木材：Birch、Maple和Oak。通过创建一张编码表或转换表，每个Finish字段值可以用一个编码来代替，该编码以类似于外键的方式交叉引用查找表。这样会节省Finish字段的存储空间，从而节省PRODUCT文件的存储空间。但是查找表需要额外的空间，当查询Finish字段的值时，需要对查找表进行一次额外访问。如果Finish字段很少使用，或是不同的Finish字段值很多，那么额外的开销远远大于带来的益处。需要注意的是，编码表并没有出现在概念或逻辑设计模型中，它只是一个用来提高数据处理性能的物理结构，而不是带有业务值的一组数据。

数据压缩技术往往使用某种形式的编码表，如文件的zip压缩方法。数据压缩技术在数据中寻找模式，然后用较少的位数为常见的模式编码。虽然类似的技术常常用于压缩整个文件，但也可以在一些DBMS中用于压缩特定的字段。与数据压缩技术相关的是加密技术，它将字段转化为一种安全的格式。在压缩和加密后，软件必须知道如何进行相应的操作才能使用户看到实际的字段值。

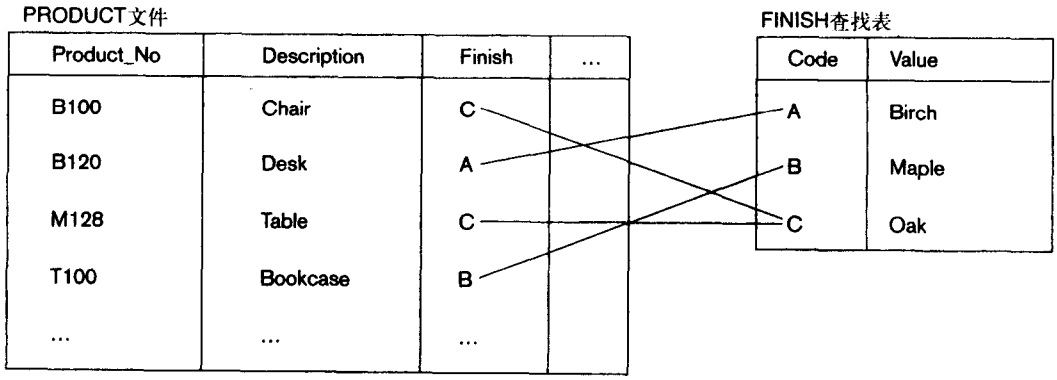


图6-2　代码查找表示例（松谷家具公司）

6.4.2　数据完整性控制

对于很多DBMS，数据完整性控制（如控制一个字段的可能值）可以内建到字段的物理结构中，并由数据库管理系统对这些字段执行控制。数据类型也是数据完整性的一种，因为它限制数据的类型（数字型或字符型）和字段值的长度。DBMS可以提供的其他常见的完整性控制包括：

- **默认值**　当用户没有为字段输入值时，系统就为该字段赋予默认值。为字段赋予默认值可以缩短用户数据输入的时间，因为字段值的输入可以跳过；此外对于常用的值它可以减少数据输入的错误。
- **范围控制**　范围控制限定一个字段允许输入的值，它的范围可以是数字的上界和下界，也可以是一组特定的值。由于范围限制可能随时间变化，所以必须小心地被使用范围控

制。范围控制和编码的结合导致许多组织必须面对2000年问题，因为存储年的字段范围限定为00～99。应该在DBMS内部实现范围控制而不是在程序中实现，因为在程序中实现往往会出现不一致，而且难以查找和修改。

- **空值控制** 在第5章中，我们定义空值。所有的主键必须有完整性控制以避免空值。其他一些所需的字段也应该根据组织策略进行相应的控制以避免空值。比如，某个大学可能禁止将一门课加入到数据库中，除非该课程有课程名和相应的主键值Course_ID。应该只在确实受业务规则限制而不允许空值的字段上使用这种控制，因为很多字段是可以使用空值的。
- **参照完整性** 按照第5章中的定义，参照完整性也可以看作是一种范围控制。因为该字段的值在同一张表或其他表的另一行的字段中存在；换句话说，值的合法范围是由数据库表的某个字段的动态内容来确定的，而不来自于预先定义的值集。需要指出的是，参照完整性保证只取那些已经存在的交叉引用的值，而不能确保取值的正确。

处理缺失的数据

当某个字段的值可以为空时，用户可能就不为它输入值。比如，在某个数据库系统中，顾客的邮政编码字段可以为空，而同时系统中有一个根据月份和邮政编码对销售情况进行汇总的报表。此时的问题是，如何在未知邮政编码的情况下统计销售情况呢？我们前面已经提到两种处理和防止数据缺失的可选方法：提供默认值和不允许为空。通常情况下，空值是不可避免的。根据Babad 和Hoffer（1984）的研究，还有其他一些可行的处理缺失数据的方法：

1) 为缺失数据提供一个估计值。例如，在计算某种产品某月的销售情况时，对于缺失的数据值，可以使用一个包含已有的该产品月度销售数值的平均值和这个月的所有产品的销售量的公式来计算值。类似的估计值需要特别标注，以使用户知道它不是实际的值。

2) 跟踪缺失的数据以产生报表或其他系统信息，使人们尽快地找到相应的数据。这可以通过在数据库上定义中设置一个触发器来实现。触发器是一个例程，它会在某种情况发生或是在某段时间过去后自动执行。当空值或其他丢失值被存储时，一个触发器可以将这个信息记录到文件里；而另一个触发器可以每隔一段时间就为这个日志文件生成报表。

3) 执行敏感性测试，除非在测试中发现它们会对最终的结果产生很大的影响，否则可以忽略缺失的数据。比如一个销售人员的月销售额决定他的薪水，而缺失的数据会影响他的业绩，从而影响他该月的收入。这是到目前为止最复杂的一种方法，需要仔细地编写程序。这些处理缺失值的例程可以在应用程序中编写。很多的现代数据库管理系统具有更强的编程能力，如编写case表达式、用户定义函数和触发器等，所以类似的逻辑可以在数据库内部实现而与特定应用无关。

6.5 设计物理记录和非规范化

在逻辑数据模型中，由于同一个主键决定的属性被放到一个关系里。与之相反，**物理记录**（physical record）是一组在内存里相邻存放的字段，它们是DBMS读写的一个基本单元。设计物理记录时要选择字段在相邻存储空间的顺序，以期达到两个目的：二级存储的有效使用和提高数据处理的速度。

二级存储的有效使用取决于物理记录的大小和二级存储的结构。计算机的操作系统以页而不是物理记录为单元读取数据。**页**（page）是操作系统在二级存储的一次输入或输出操作中所读写的数据量。页的大小是固定的，由系统程序员设定以便所有的应用更有效地使用内存。在某些系统中，物理记录可以跨页，而在某些系统中则不能跨页。因此如果页的大小不是物理记

录大小的整数倍，那么在页中就会出现空间的浪费。一页中物理记录的数目称为**块因子**(blocking factor)。如果存储空间不足且物理记录不能跨页，那么为一个逻辑关系创建多个物理记录会将空间的浪费减到最小。一些数据库管理系统会将多个物理记录合成一个数据块；此时DBMS管理数据块，而操作系统管理页。

非规范化

前面对于物理记录设计的讨论主要强调对存储空间的有效使用，但在大多数情况下，物理记录设计的第二个目标——高效的数据处理在设计过程占主导地位。数据的高效率处理和在图书馆中取书一样，取决于相关数据（书籍）存放的紧密程度。通常情况下，一个关系内的所有属性不会同时被使用，而不同关系的属性常常组合起来以满足查询的要求或是生成报表。因此，虽然规范化的关系可以消除数据维护中的异常，并减少冗余，但直接将规范化的关系对应到物理记录却未必能产生高效率的数据处理。

完全规范化和部分规范化的数据库在处理性能方面的差异有极大的不同，Inmon（1988）对此进行过量化的研究。一个完全规范化的数据库有8个表，每个表有50 000行，另一个部分规范化的数据库有4个表，每个表有25 000行，而另一个部分规范化的数据库有两个表。结果表明部分规范化数据库的性能比完全规范化数据库的性能超出一个数量级。虽然这个结果很大程度上取决于数据库以及对于数据库的具体操作，但仍然表明应该对物理记录和规范化关系的对应问题进行认真考虑。

非规范化　是将规范化的关系转化为非规范化的物理记录规格说明的过程。我们在本节中讨论非规范化的原因和各种形式。概括来说，非规范化可能将一个关系分解为多个物理记录，可能将来自于不同关系的属性组合到一个物理记录里，也可能是两者的结合。正如Finkelstein（1988）指出的，非规范化会增加错误和不一致性出现的机会，当业务规则改变时甚至可能需要重建系统。此外，非规范化以牺牲一些操作的效率为代价，来提高另外一些操作的效率。如果操作发生的频率有改变，则非规范化的益处就不再存在。非规范化几乎总是导致原始数据的存储空间增加，也会导致数据库开销（如索引）需要更多的空间。因此非规范化应该在其他物理设计因素都达不到性能要求时使用，并必须在系统中显著地改善性能。

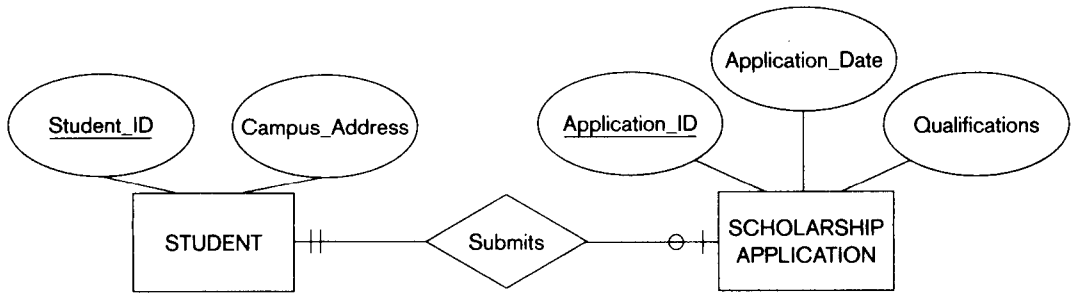
Rogers（1989）介绍了几种常见的适于使用非规范化的情况（图6-3～图6-5给出这三种情况下的规范化和非规范化的关系）：

1）*两个实体具有一对一联系*　即使其中一个实体是可选的参与者，但只要它在大多数情况下存在，就应该考虑将两个实体合并成一个记录（尤其在对它们的访问频率很高时）。图6-3中是学生的数据，和可选的标准学位申请表数据。在这种情况下，可以使用来自于STUDENT和SCHOLARSHIP APPLICATION关系的4个字段组成记录（假设不再需要Application_ID。注意，在这种情况下，来自于可选一方实体的字段值可能为空）。

2）*具有非键属性的多对多联系（关联实体）*　并非将三个文件联结以从关系的两个基本实体中抽取数据，而是将一个实体和关联实体的属性结合成一个记录，这样就可以在许多数据访问模块中将必须的联结操作减少一次。和前面一样，访问次数越多，这种方法的益处就越大。图6-4中是不同供应商对于不同标准件制定的价格信息。此时来自于关系ITEM和PRICE QUOTE的字段可以被结合入一个记录，这样就避免对全部三个文件进行联结（注意，这样可能产生大量的数据冗余。在这个例子里，ITEM中的字段Description会在每个PRICE QUOTE中重复。当该数据变动时，要进行许多的修改）。

3）*引用数据*　引用数据位于一对多联系中处于1的一方的实体中，且该实体不参与任何数据库内的其他联系。此时如果对每个1的一方的实例，M的一方都只有很少的实例与其对应，

那么就应该考虑将两个实体合并为一个记录。在图6-5中，几个ITEM有相同的STORAGE INSTRUCTIONS，并且STORAGE INSTRUCTIONS只和ITEM相关。在这种情况下，可以将STORAGE INSTRUNCTION数据存放在ITEM记录中。当然此时会产生数据冗余和额外的数据维护（不再需要Instr_ID）。



规范化关系

STUDENT

Student_ID	Campus_Address

APPLICATION

Application_ID	Application_Date	Qualifications	Student_ID

非规范化的关系

STUDENT

Student_ID	Campus_Address	Application_Date	Qualifications

Application_Date和Qualifications可以为空值

图6-3 一种可能的非规范化情况：存在一对一联系的两个实体（注意：我们假设当所有字段都存储在一个记录中时，不再需要Application_ID。但如果它是必须的应用数据时，则可以将该字段包括进来）

在上面谈到的情况中，都是将表结合起来以避免联结操作。与之相反，非规范化也可以用于将单个关系分割为多张表，这些分割方法包括水平分割、垂直分割或是两者的结合。**水平分割**（horizontal partitioning）将一个关系分割为多个记录规格说明，方法是将关系的行根据某个列值，分别地存放到不同的记录中。在图书馆里，水平分割类似于将商业杂志放入商业图书馆，将科学书籍放入科学图书馆，等等。每个经分割创建的文件都含有相同的记录布局。比如，顾客关系可以根据字段Region的值而分割为4个区域顾客文件。

当对表中不同类型的行是单独处理时，利用水平分割是很有意义的。例如，在前面提到的顾客表中，或许大多数对于数据的操作在某一时间都是局限在某个区域内的。水平分割也有安全性方面的优点，因为可以使用文件层次的安全性来禁止用户查看某些数据行。此外，对被分割出的文件，可以根据它们各自的使用情况来选择组织方式。恢复一个分割后的文件显然比恢复含有所有行的文件要快得多，而且即使某个文件因为损坏而必须离线恢复，也不会影响用户对于其他文件的继续使用。分割后的文件可以存放在单独的磁盘上以避免对于一个磁盘的争用，这有助于改善数据库系统性能。表6-2中总结了水平分割的优点（实际是各种分割共有的）和缺点。

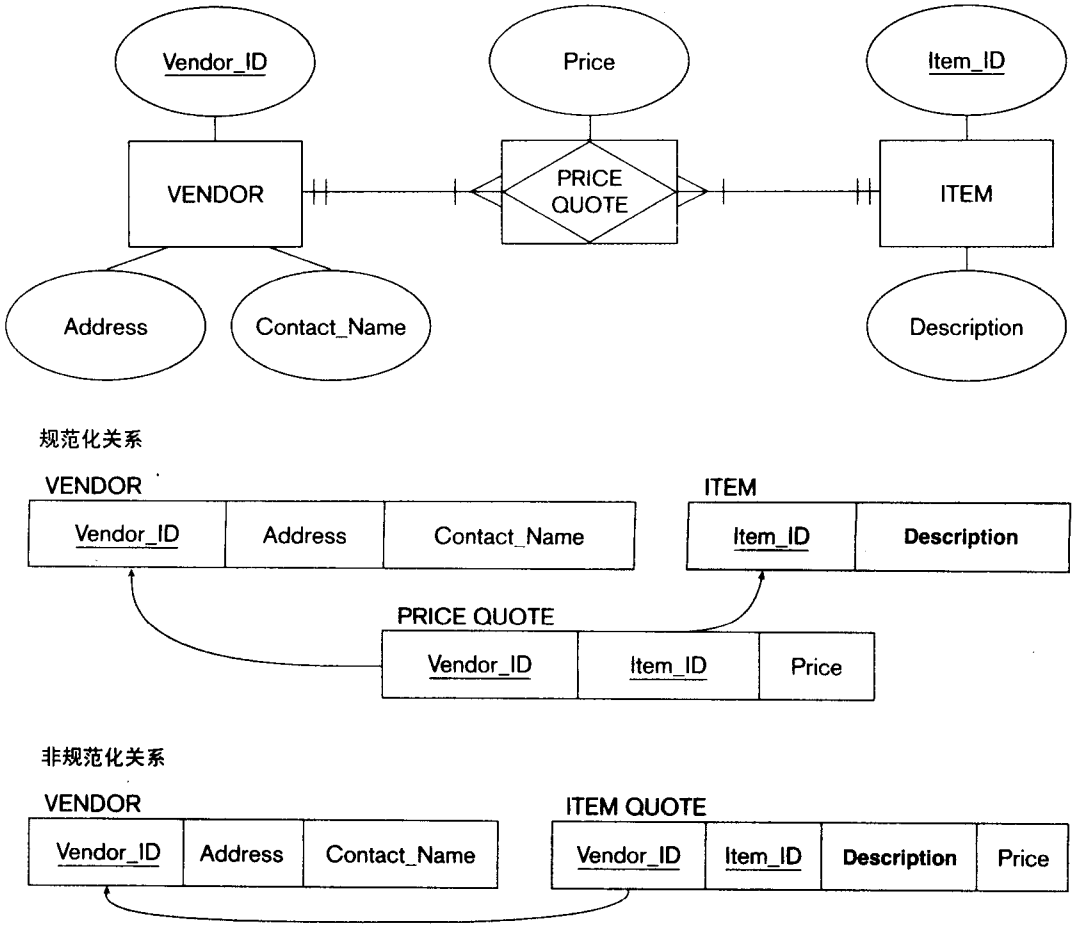


图6-4　一种可能的非规范化情况：带有非键属性的多对多联系

表6-2　数据分割的优缺点

分割的优点
1) 效率：共同使用的数据被存放在一起，不一起使用的数据分开存放
2) 局部优化：某个分割可以根据自身的使用情况存放，以优化性能
3) 安全性：和一组用户不相关的数据可以和他们需要使用的数据分开存放
4) 恢复和正常运行时间：小的文件可以更快地被恢复。即使一个文件损坏，其他文件仍可被访问；损坏的影响被隔离开
5) 负载平衡：文件可以存放在不同的存储区域（磁盘或其他媒介），这可以减少访问对相同存储区域的争用，并允许对不同区域进行并行访问
分割的缺点
1) 不一致的访问速度：不同分割的不同访问速度可能令用户困惑。尤其当必须结合不同分割的数据时，其响应时间可能大大超过不分割的情况
2) 复杂性：分割通常对于程序员是不透明的，在需要结合不同分割的数据时，程序会变得更加复杂
3) 额外的空间和更新时间：数据可能需要在不同分割间重复，这比单独存储规范化的关系浪费空间。一个涉及到多个分割内数据的更新操作会需要更长的时间

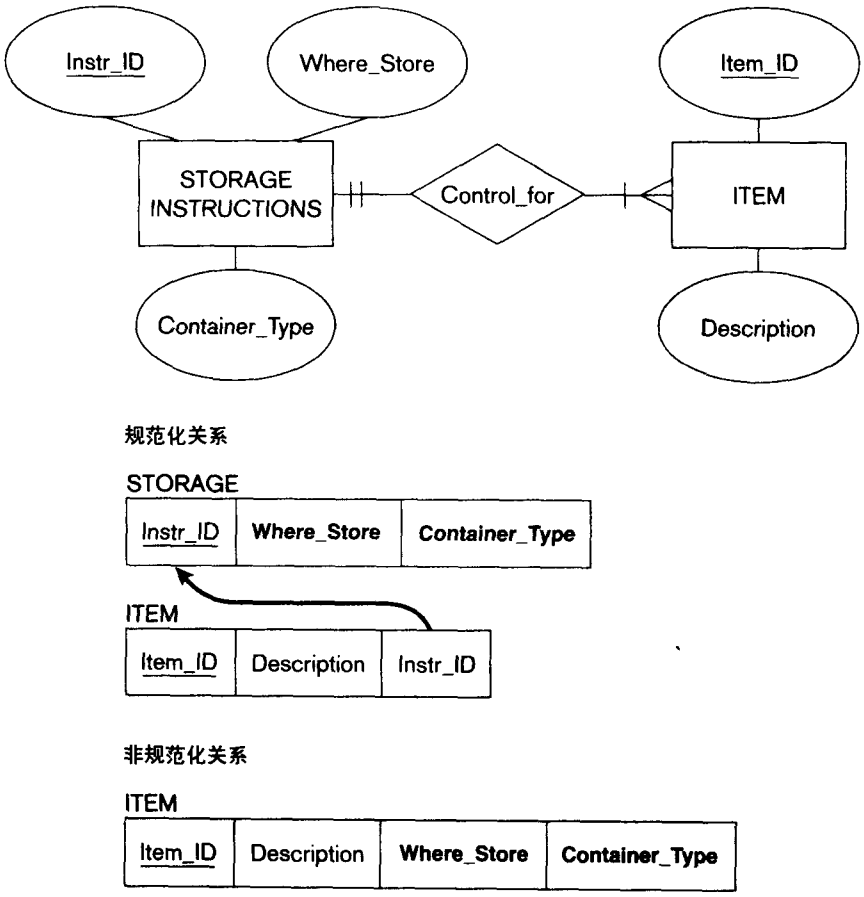


图6-5 一种可能的非规范化情况：引用数据

可以看到，水平分割和创建超类型/子类型联系非常相似，因为不同类型的实体（其中使用子类型鉴别符字段来隔离行）会参与不同的联系，从而导致不同的处理。在面对一个超类型子类型联系时，需要考虑是为每个子类型创建独立的表，还是将它们以不同方式进行组合。如果子类型的使用方式类似，就应该将它们组合起来。而如果子类型在事务、查询和报表中的处理方式不同，就应该考虑将超类型实体分割为多个文件。当一个关系被水平分割时，可以使用SQL UNION操作符（在第7章中介绍）来重建，并得到所有的行。考虑前面的例子，当需要时可以看到所有的顾客数据。

Oracle数据库管理系统支持多种形式的水平分割，这主要是为了处理特别大的表而设计的（Brobst等，1999）。当在DBMS中使用SQL数据定义语言（第7章中会介绍相应的CREATE TABLE命令）来定义表时，可以指定表的分割。在Oracle中，一个表可以有多个分割。Oracle 8*i*提供三种水平分割的方法：

1) **键范围分割** 通过为规范化关系的一个或多个字段指定值的范围（上下界）来定义分割。表的行将会根据它最初的值存放到正确的分割中。考虑到分割键值遵循一定的模式，所以每个分割拥有的行数可能会差别很大。数据库设计人员可以通过指定分割键来平衡行的分布。当键值被更新时，可以限制该行不能在分割间移动。

2) **散列分割** 此时数据被平均地存放在分割中，且不依赖于任何用于分割的键值。散列分

割克服了键范围分割可能产生的行分布不均的问题。

3）复合分割　这是键范围分割和散列分割的结合。对于数据库用户而言，分割可以是透明的（除非用户希望将某个查询局限在某个或某几个分割时，才需要指明）。数据库内用于优化查询的模块，会为查看查询涉及的表的分割定义，并在检索查询结果所需的数据时自动决定是否排除那些特定的分割。这会显著改善查询的性能。

例如，我们在键范围分割中使用事务的日期来定义分割。一个要查看最近事务的查询就只需要在包含最近事务的一个或几个分割上进行查找；而不需要扫描整个表，甚至使用索引在没有分割的表的指定范围中查找行。依照日期定义的分割还可以将新数据的插入局限在一个分割上，这会减少数据库维护的开销；当需要删除旧事务数据时也只需要删除相应的分割。在分割表上也可以使用索引，它能比只使用分割提供更好的性能。Brobst等（1999）详细讨论了使用日期作为键范围分割依据的优缺点。

在散列分割中，行平均地在分割间存储。如果分割可以存放在支持并行处理的存储区域上，那么与对单个表中某个存储区域的数据进行顺序访问相比，会大大提高查询的效率。和键范围分割一样，散列分割相对于查询的程序员来说也可以是透明的。复合分割首先使用键范围来定义分割，然后在得到的分割上再进行散列分割，从而得到子分割，它结合了键分割的优点和散列分割良好的并行处理能力。

垂直分割　将一个关系的列分布到几个单独的物理记录中，此时关系的主键在每个物理记录中重复出现。以零件关系为例，它可以被垂直分割为账目相关、工程相关和销售相关的三个记录规格说明，且零件号必须出现在每个记录中。垂直分割的优缺点和水平分割类似。当所有的零件数据要同时使用时，上面的三个表可以联结起来。和水平分割一样，垂直分割也不影响用户对于原始的整个关系的使用。

水平分割和垂直分割也可以结合使用。这种被称为记录分割（record partitioning）的非规范化形式对于那些文件分布在多个计算机上的数据库而言特别常见，我们会在第13章中对此进行讨论。

单个的物理表可以被逻辑地分割，多个表也可以使用用户视图的概念进行逻辑结合（我们会在第7章中讨论用户视图）。利用用户视图，用户会感到数据库内除物理定义的表以外，还有其他的表；这些逻辑表可以使用水平、垂直分割和其他的非规范化方法创建。然而，任何形式的用户视图，包括通过视图的逻辑分割，只是用来简化查询的编写并使数据库更安全，但并不能提高查询性能。Oracle提供了一种称为分割视图（partition view）的用户视图。在分割视图中，通过使用SQL UNION操作符，可以将物理上分离但是具有类似结构的表逻辑地结合在一起。这种形式的分割有很多局限性。首先，由于存在着多个单独的物理表，所以不能在合并后的行上创建全局索引。其次，每个物理表必须进行单独维护，因此数据维护更加复杂（比如插入新行时，必须指定具体的表）。最后，在创建更高效的查询处理计划时，查询优化器处理分割视图的选项远远少于处理表的分割的选项。

我们介绍的最后一种非规范化是数据复制。在数据复制中，同样的数据被故意地存放在一个数据库的多个地方。以图6-1为例，读者前面已经学习到可以将来自关联实体的数据和组成该关联的简单实体的数据相结合来进行反规范化。在图6-1中，QUOTATION的数据就可以和PURCHASED PART的数据存储在一个扩展的PURCHASED PART物理记录规格说明中。当使用数据复制时，同样的QUOTATION数据可以和与其相关的SUPPLIER数据一起存储在新的扩展SUPPLIER物理记录规格说明中。使用这样的数据复制方法，只要检索SUPPLIER或PURCHASED PART记录，相应的QUOTATION数据就可以应用而不必再次访问二级存储。当

然仅在QUOTATION数据总是和SUPPLIER或PURCHASED PART数据同时使用，且所带来的额外存储和数据维护代价不大时，所获得的性能改善才是值得的。

6.6 设计物理文件

物理文件（physical file）是分配用来存储物理记录的一部分已命名的二级存储（如磁带和硬盘）。有些计算机操作系统将物理文件分成称为区的块来存储。在以后的几节中，我们假设物理文件没有分开，且文件内的每个记录有相同的结构。我们会讨论如何将一个数据库内的关系表的行存储到物理空间中去。为提高数据库系统处理的性能，数据库管理员通常应该了解数据库系统管理物理存储的具体细节。虽然这种技术是与具体DBMS相关的，但是我们在以后几节中所讨论的原理是大多数关系数据库管理系统所应用的物理数据结构的基础。

大多数数据库管理系统将多种类型的数据存放在单个操作系统文件中，我们所说的操作系统文件就是指在磁盘目录列表中显示的有名文件（比如在读者个人电脑的C盘某个目录下的文件）。在Oracle数据库中，物理存储空间的一个重要结构是表空间。一个**表空间**（tablespace）是命名的一组磁盘存储单元，它用来存储一个或多个数据库表的数据。Oracle的一个实例会包含多个表空间，例如一个表空间用来存放系统数据（数据字典或关于数据的数据），一个表空间用于临时工作空间，一个表空间用于数据库恢复，其他一些表空间用来存放用户业务数据。一个或多个表空间可以存放在一个物理操作系统文件中。Oracle负责管理表空间内数据的存储，而操作系统则像管理其他操作系统文件一样为表空间的整体提供管理（处理文件层次安全性、空间分配、处理磁盘读写错误）。

由于一个Oracle的实例通常为多个用户支持多个数据库，所以数据库管理员通常需要创建多个用户表空间，并给用户在各个表空间上赋予不同的权限以保证数据库安全。如上面所介绍的，作为一个操作系统文件，表空间可以分布在多个**区**（extent）上，每个区是磁盘存储空间的连续扇区。当一个表空间需要扩大以存储更多数据时，系统就会为它分配另一个区。每一个数据库的表被存储在一个或多个表空间中（表的一行只存储在一个表空间中，同一个表的不同行可以存储在不同的表空间中）。一个表空间可以存储一个或多个表的数据。在Oracle环境中，管理表空间（或物理的数据库文件）是数据库管理员的重要工作。例如，通过将不同表空间分配到不同的设备上，并将表分布在表空间上，数据库管理员可以将并发用户对磁盘争用减到最少。本书并不是专门讲授Oracle数据库的，所以我们不详细介绍管理表空间的细节。然而在Oracle数据库中设计和管理表空间的基本原则同样适用于其他数据库管理系统的物理存储单元管理。图6-6中的ERR模型表明在Oracle环境下，物理数据库设计所涉及的多个物理和逻辑数据库术语之间的关系。

6.6.1 指针

在文件里联结一块数据和另一块数据从而将所有文件组织起来的两种基本结构是顺序存储和指针。在顺序存储中，一个字段或记录紧接着前一个字段或记录存储。虽然顺序存储易于实现和使用，但有时候它并不是组织数据最有效的方式。**指针**（pointer）是一个数据字段，它用来定位一个相关的数据字段或记录。在大多数情况下，指针包含相关数据的地址或位置。指针可以在多种数据存储结构中使用，我们在本章中会介绍其中的一些存储结构。有兴趣的读者可以在附录C中找到有关指针的更详细的介绍。我们在这里只定义指针，是因为了解指针的概念对于了解文件组织至关重要。通常来说，用户不会直接使用指针，因为数据库管理系统会自动地处理指针的使用和维护。

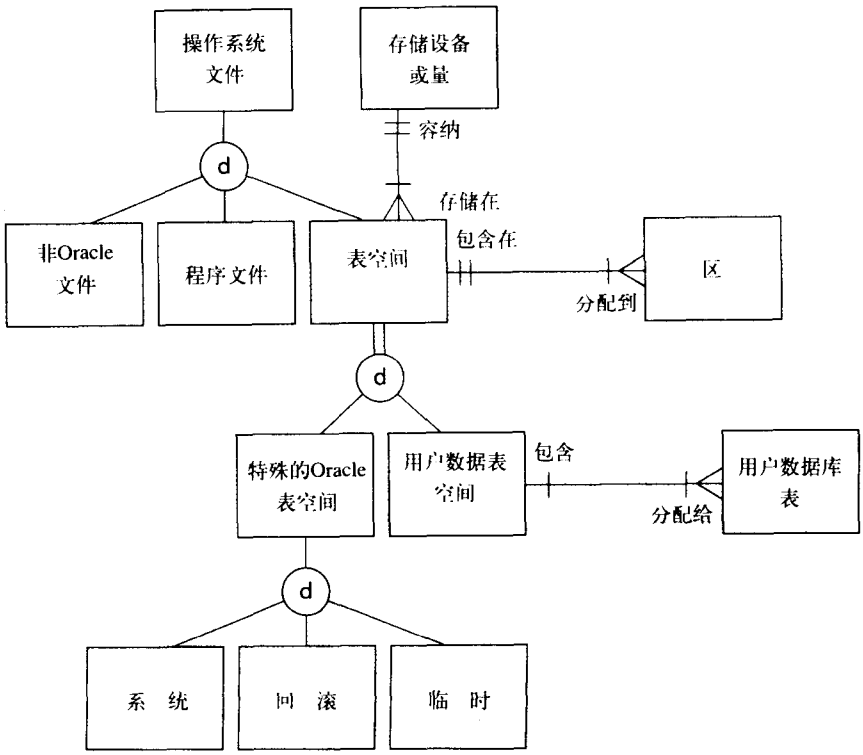


图6-6　Oracle环境中物理文件的术语

6.6.2　文件组织

文件组织（file organization）是在二级存储设备上物理安排文件中记录的技术。在现代关系数据库管理系统中，用户不必自行设计文件组织，而是可以为表或物理文件选择某种组织的形式及其参数。在为数据库内的文件选择文件组织时，应该考虑以下7个重要因素:

1) 快速的数据检索。

2) 处理数据输入和维护事务的高吞吐量。

3) 存储空间的高效使用。

4) 防护失效和数据丢失。

5) 将重新组织的需要最小化。

6) 适应增长。

7) 对非授权访问的安全控制。

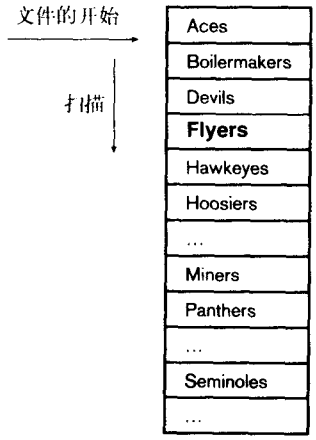
通常这些目标会相互冲突，所以应选择一种文件结构，使得在资源范围内尽量平衡这些因素。在本章中，我们考虑以下几类基本的文件组织方法: 顺序文件组织、索引文件组织和散列文件组织。图6-7以一些大学运动队的昵称为例说明这几种组织方式。

1. 顺序文件组织

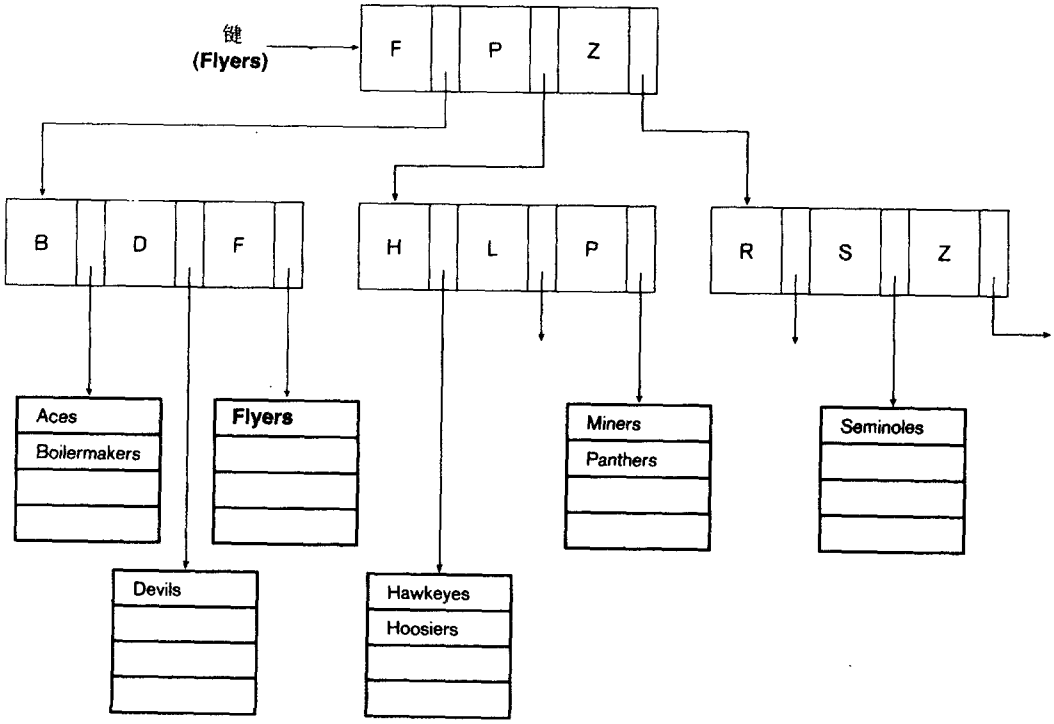
在一个**顺序文件组织**（sequential file organization）中，文件中的记录按照主键值依次存储（参见图6-7a）。要定位某个记录，程序必须从头到尾扫描文件直至找到需要的记录为止。电话簿黄页中字母序人名列表就是一个常见的顺序文件（假设目录中没有其他索引）。表6-3中比较了顺序文件和其他两种文件的功能。由于顺序文件的方式不够灵活，所以数据库系统通常并不使用这种方式；但是它可以应用在数据库备份数据的文件上。

2. 索引文件组织

在索引文件组织（indexed file organization）中，记录可以用顺序或非顺序的方式存储，系统建立索引以使程序可以定位到所需记录（参见图6-7b）。**索引**（index）是一个表，可以用来确定文件里符合某些条件的行的位置，这和图书馆里的卡片目录一样。索引的每一项将一个或多个记录与键值匹配起来。索引可以只定位单个记录（主键索引，如PRODUCT记录的Product_ID字段），也可以定位多个记录。一个允许定位多个记录的索引称为**辅键**（secondary

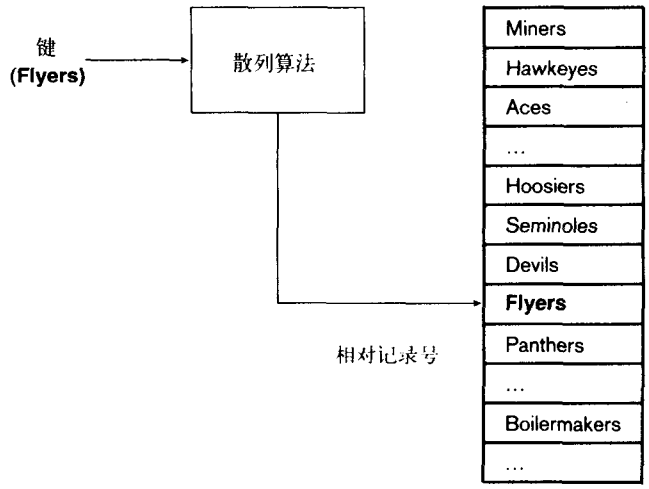


a) 顺序



b) 索引的

图6-7 文件组织的比较



c) 散列的

图6-7（续）

key）索引。辅键索引可以支持许多报表需求，也可以为某些特别的数据检索提供快速响应。辅键索引的一个例子是PRODUCT记录上的Finish字段。

一些索引结构会影响表中行的存储位置，而另一些索引和行的存储位置无关。由于实际的索引结构不影响数据库的设计，对数据库查询的编写也不重要，所以我们不在本章中讨论索引的实际物理结构。因此图6-7b不是索引结构中数据存储的物理视图，而是一个使用索引的逻辑视图。某些关系数据库管理系统使用术语**主索引**（primary index，不同于*主键索引*（primary key index））来表示决定数据物理位置的索引，而术语**辅索引**（secondary index，不同于*辅键索引*（secondary key index））表示和确定数据的存储位置无关的索引。在这些系统中，一个表可以有一个主索引和多个辅索引。主索引可以使用表的主键字段，也可以不使用；辅索引所使用的字段值在行间可以是惟一的，也可以不惟一。当使用术语主索引和辅索引时，有以下4种索引类型：

- **惟一性主索引**（UPI）　基于惟一性的字段的索引，该字段可以是表的主键。惟一性主索引不但可以用来在表中定位具有特定值的行，还被DBMS用来决定行的存储位置。
- **非惟一性主索引**（NUPI）　基于非惟一性的字段的索引。这个索引不单可以用来定位具有特定值的行，还被DBMS用来决定行的存储位置。
- **惟一性辅索引**（USI）　基于惟一性的字段的索引。这个索引只用来定位具有特定值的行。
- **非惟一性辅索引**（NUSI）　基于非惟一性的字段的索引。这个索引只用来定位具有特定值的行。

在图6-7b的例子中，在索引上又创建索引，构成一个有层次的索引集。索引本身也是一个文件，当索引文件很大时，通过为索引作索引也是一种可行的组织方法。图6-7b中的每个索引项都包含一个键值和一个指向其他索引或数据记录的指针。例如，当要寻找具有键值“Flyers”的记录时，文件组织会从顶部的索引开始，指针沿着F项向下寻找。该指针指向其他包括键值开头字母从A～F的索引。程序继续沿着指针在F下层的索引中寻找指针，它会指向键值以E和F开头的记录。最终通过索引，查询会找到所需的记录，或是报告不存在这样的记录。

索引文件组织的一个重要功能是可以创建多个索引。在图书馆里，我们可以依据书名、作者和主题为同样的书和杂志建立多个索引。通过在同一组数据上创建另外的索引，我们构造另

一个文件。在这个文件里，只包含用于索引的字段，而大多数数据都不用被复制。多个索引是可以被使用的。比如，如果使用索引Finish去寻找那些使用白桦制作的产品，而使用索引Cost去寻找那些制造成本少于500美元的产品。通过对这两个查询结果集取交集就可以找到那些使用白桦，且制造成本少于500美元的产品。若在关系数据库中处理查询，那么在一个查询中使用多个索引是非常重要的。逻辑操作AND、OR和NOT都可以对一个索引扫描的结果进行再处理，避免访问不满足所有条件记录所产生的代价。

图6-7b中的层次索引结构被称为树（根结点在上，叶结点在下，这个树是颠倒的）。树状索引的性能主要取决于所使用的树结构的性质，在关系数据库管理系统中常常使用的树的类型是平衡树，即B树，而其中最常见的形式是B+树。在B树中，所有的叶子（包括数据记录或指向数据记录的指针）和树根的距离都是相等的。如在图6-7b中，所有数据记录和根都相差两级。由于是操作系统和DBMS决定使用何种索引树结构，用户通常对索引的结构没有什么选择权，所以树的管理对数据库设计人员和开发人员都是透明的。附录C中将会讨论有关树的数据结构和用户可以控制的一些树的参数。

位图索引是目前使用越来越广泛的一种索引类型。**位图索引**（bitmap index，参见图6-8）使用位图来管理键值。假设表Product有10行，而在这10行中它的Price属性有4个不同的取值。此时在Price字段上的位图索引就拥有4个项，而每个项包含10位。每个位图项中的一位对应Product表中的一行。例如，在price为400美元的位图中，该项第8位的值为1。这就表明Product表的第8行的Price值为400美元（很明显，位图中的每列都只有一个1）。当属性的可能值非常少时，使用位图索引比常规的树索引更加理想。位图索引通常比传统的树索引占用更少的空间（可能只有25%，Schumacher，1997）。当然如果一个属性可以取很多不同的值，那么位图索引就可能比树索引需要更多的空间。如果使用合理，由于对位的处理和扫描非常快速，所以基于位图索引的查询可以比基于常规的树查询快10倍。

	Product表行号									
Price	1	2	3	4	5	6	7	8	9	10
100	0	0	1	0	1	0	0	0	0	0
200	1	0	0	0	0	0	0	0	0	0
300	0	1	0	0	0	0	1	0	0	1
400	0	0	0	1	0	1	0	1	1	0

注：Products 3和5的Price为$100。
Products 1的Price为$200。
Products 2、7和10的Price为$300。
Products 4、6、8和9的Price为$400。

图6-8 Product表中Price属性上的位图索引

位图索引的一个有趣特性是它可以在多个键上使用。比如，我们可以在图6-8的下面根据属性Room的值再添加行。此时，每列就会有两位被置1；一位对应于Price，而另一位对应于Room。如果要查找100美元餐厅使用的产品，就可以将分别对应的两行进行位串的交运算，这样可以找到符合两个条件的产品。这样的位操作速度是非常快的。

事务处理程序需要系统对涉及几张表中少量的相关联行查询的快速响应。比如，要输入一个新的顾客订单，订单输入程序需要在顾客表中找到相应的顾客，在产品表中找到顾客要购买

的产品，可能还要根据顾客的要求查询有关产品特征的行（比如Product的Finish属性）。同时在相关的表中添加顾客订单和发货记录。我们在前面讨论到的索引类型对于这种寻找少量特定行的应用程序有良好的支持。

另一种越来越流行的索引类型是联结索引，在数据仓库和其他决策支持系统中（第11章）这种索引更为流行。在决策支持系统中，往往需要访问与其他表相联系的一个表中的所有行（如从同一家商店购物的所有用户），而这种表的数据量通常非常大。**联结索引**（join index）是在来自于多个表的列上创建的一个索引，这些列有相同的值域。比如，考虑图6-9a中的两张表，Store和Customer，它们都有列City，在City上的联结索引会表示两张表中含有相同City值的行。由于有许多数据仓库，它们都很可能需要查找那些处于同一个城市的商店和顾客的数据（或在多个维之间的类似关联）。图6-9b中是联结索引的另一个可能的应用，它预先计算表Order中外键和表Customer中主键的关联。联结索引预先计算关系中联结（join）操作的结果，该内容在第7章中进行讨论。简而言之，联结索引在同一张表或不同的表里寻找符合某些要求的值所在的行。

Customer

RowID	Cust#	CustName	City	State
10001	C2027	Hadley	Dayton	Ohio
10002	C1026	Hanas	Columbus	Ohio
10003	C0042	Ruskin	Columbus	Ohio
10004	C3861	Davies	Toledo	Ohio
...				

Store

RowID	Store#	City	Size	Manager
20001	S4266	Dayton	K2	E2166
20002	S2654	Columbus	K3	E0245
20003	S3789	Dayton	K4	E3390
20004	S1941	Toledo	K1	E0874
...				

联结索引

CustRowID	StoreRowID	Common Value*
10001	20001	Dayton
10001	20003	Dayton
10002	20002	Columbus
10003	20002	Columbus
10004	20004	Toledo
...		

注：*这一列可以根据需要被包含或不被包含。联结索引可以在三列中的任意一列上建立。某些时候可以创建为个联结索引，一个如上所示。另一个则将两个rowID的列交换一下。

a）常见非键列上的联结索引

图6-9　联结索引

Order

RowID	Order#	Order Date	Cust#(FK)
30001	O5532	10/01/2001	C3861
30002	O3478	10/01/2001	C1062
30003	O8734	10/02/2001	C1062
30004	O9845	10/02/2001	C2027
...			

Customer

RowID	Cust#(PK)	CustName	City	State
10001	C2027	Hadley	Dayton	Ohio
10002	C1026	Barnes	Columbus	Ohio
10003	C0042	Ruskin	Columbus	Ohio
10004	C3861	Davies	Toledo	Ohio
...				

联结索引

CustRowID	OrderRowID	Cust#
10001	30004	C2027
10002	30002	C1062
10002	30003	C1062
10004	30001	C3861
...		

b) 匹配外键和主键的联结索引

图6-9 （续）

联结索引在行被加载入数据库时创建，所以它和前面提到的所有索引一样，都反映数据库的最新信息。如果没有图6-9a中的联结索引，每个对同一个城市内商店和顾客的查询请求都需要重新进行和联结索引相同的计算。由于在两个很大的表之间进行联结操作是相当浪费时间的，所以这会严重影响联机查询的响应速度。在图6-9b中，联结索引可以帮助DBMS找到相关的行信息。联结索引和其他的索引一样，它们都是通过预先查找符合某些条件的数据，来加快查询的效率，其代价是额外的存储空间和索引的维护。Bontempo和Saracco（1996）的著作介绍了如何将联结索引和位图索引结合使用以加快某些特定的联结索引的处理速度。

Bullinger（1999）提出一种联结索引的扩展方法以避免对实际数据表的访问。以图6-9b为例，除Cust#以外，还可以将其他一个或两个表中的字段包含进联结索引中。这样当使用联结索引的查询只需要这些行时，就不再需要访问原来的表。那么什么时候应该使用这种联结索引的扩展呢？答案是数据库是相对静态，一旦创建联结索引，就会有相对长的生命周期。很多数据仓库的环境是满足以上要求的：在这些数据仓库中，已经存在的行通常不会再修改，可能以月为周期进行新数据的添加和废弃数据的删除，在这些修改不频繁，而却经常对大数据表进行复杂查询的数据库中，对联结索引进行扩展就会产生良好的性能改善。

一些新兴的数据库应用（如数据仓库、联机决策支持系统）对于数据库的索引提出新的要求。我们在这里建议读者仔细研究你们所使用的数据库管理系统中的索引功能，弄清楚每种索引的适用条件和如何调整索引结构的性能。

3. 散列文件组织

在一个**散列文件组织**（hashed file organization）中，每个记录的地址是使用散列算法来决定的（参见图6-7c）。**散列算法**（hashing algorithm）是一个将记录的主键值转换为记录地址的程序。虽然有许多散列文件的变体，但是在大多数情况下，使用散列算法的记录不是顺序存放的，也就不适用于顺序的数据处理。

一个典型的散列算法是将每个主键值除以一个素数，并使用余数作为相对的存储位置。比如，假设一个组织中有1000条员工的记录存放在磁盘上。此时合适的素数可以是997，因为它和1000很接近。现在考虑员工12396，我们把这个值除以997后得到的余数是432，因此这条记录就存放在文件的位置432。必须使用其他技术（这里不讨论）来处理重复（溢出）的情况，因为有多个键值取得的余数可能相同（称为散列冲突）。

散列文件组织的一个严重缺陷是，由于数据行的存储位置是由散列算法决定的，因此基于散列进行数据读取时只能应用一个键。散列可以和索引相结合构成散列索引表来解决这一问题。**散列索引表**（hash index table）使用散列算法来将键映射到索引表（有时被称为分散索引表）中的一个位置，该位置存放着一个指向实际对应散列键的数据记录的指针。散列算法的结果是索引的存放地址，而实际的数据存放在另外的位置。由于散列算法的结果是索引表中的位置，表的行可以独立于散列地址而存放，因此就可以使用任何其他的文件组织方法（比如顺序或首个可用空间方法）。和其他索引方法类似，但不同于纯粹的散列方法，此时可以使用多个主键和辅键，并为每一个键定义自己的散列算法和索引表，共享实际的数据表。同时由于索引表比数据表小得多，可以很容易地设计索引以减少键冲突和溢出的可能性。显然此时索引的额外空间和维护成本，会带来更大的灵活性和更快的数据检索速度。在一些使用并行处理的数据仓库中，散列索引表也被利用。此时DBMS将表的数据平均分配在存储设备上，并将任务公平地分配给多个并行处理器，然后使用散列和索引来快速定位所需的数据存放在哪个处理器。

使用索引技术需要注意的问题是数据一旦存储以后的移动数据的困难性。如果需要移动一个行，所有索引中指向它的指针都必须被更新。如果频繁地重新组织数据（这是由于频繁删除行在为数据行所保留的物理文件中产生浪费空间或空洞而造成的），这就会是一个很大的开销。另一方面，纯粹的散列方法不允许移动数据，除非是在处理溢出的情况下。而散列索引表的方法能根据多键值进行数据快速读取和移动数据的能力，其缺点是更新索引的额外开销，尤其当数据空间被重新组织变得很重要时。

正如前面所说，DBMS可以处理所有有关散列文件的管理问题。用户不需要关心如何处理溢出，如何访问索引或是具体的散列算法。作为一个数据库设计人员，重要的是了解不同文件组织的特性，以便为所设计的数据库系统和所开发的程序选择最适当的数据库处理类型。同样，了解DBMS使用的文件组织的性质，可以帮助程序人员书写能够有效地利用文件组织性质的查询语句。正如读者将在第7章和第8章看到的那样，很多查询都可以有多种方法用SQL语句来书写；但是不同的查询语句会导致DBMS使用完全不同的步骤来处理查询。如果用户知道DBMS使用文件组织的方法（使用什么索引，何时、如何使用散列算法），就会设计出更好的数据库，编写出效率更高的查询语句。

6.6.3 文件组织小结

前面介绍的三类文件组织涵盖了用户在进行物理文件和数据库设计时，可以使用的大部分

文件组织形式。虽然可以使用附录C中列举的数据结构构造更复杂的结构，但是通常在DBMS中不使用这种复杂的结构。

表6-3总结了顺序、索引和散列文件组织的特点。读者可以利用图6-7来对表中的特点进行验证。

表6-3 不同文件组织的特点比较

因素	顺序文件组织	索引文件组织	散列文件组织
存储空间	不浪费空间	不增加数据存储空间，但是需要索引的额外存储空间	在最初记录被导入后，需要额外的空间以支持记录的添加和删除
对主键的顺序检索	非常快	中等速度	不能实现，除非使用散列索引
对主键的随机检索	不能实现	中等速度	非常快
对多个键值的检索	可以，但是需要扫描整个文件	使用多个索引时非常快	不能实现，除非使用散列索引
删除记录	可能造成空间浪费，或者需要重新组织	当支持空间动态分配时可以实现，但是需要维护索引	非常容易
添加新记录	需要重写文件	当支持空间动态分配时可以实现，但是需要维护索引	非常容易，但当多个键对应一个地址时需要额外的工作
更新记录	通常需要重写文件	容易，但是需要维护索引	很容易

6.6.4 聚簇文件

一些数据库管理系统支持使用邻接的二级存储来存放不同表的行，这时一个物理文件中的记录结构就可能不同。例如在Oracle数据库中，来自于一个、两个或多个相关表的经常进行联结操作的行可以存储在相同的磁盘区域。一个聚簇（cluster）是通过表和表之间经常进行联结操作的列来定义的。比如表Customer可能经常和表Customer_Order在字段Customer_ID上联结，Customer_Order可能和表Price_Quote（包含从供货商处购买的标准件的价格）在Item_Id上联结。和在磁盘上不同区域存储不同文件的方式相比，聚簇可以显著地加快访问相关记录的速度，这时因为相关的记录在物理上就很接近，自然比将记录存储在不同磁盘区域的不同文件中速度要快得多。将某个表定义到单一的聚簇上，只会影响定义在相同聚簇上的那些表的存取性能。

下面的Oracle数据定义命令说明如何定义聚簇，以及如何将表指定到聚簇上。如下例所示，首先指定聚簇（邻接的磁盘空间）：

```
CREATE CLUSTER ORDERING (CLUSTERKEY CHAR(25));
```

ORDERING是聚簇空间的名字，CLUSTERKEY是必须的但不会再次使用。

当表被创建时，就被指定在聚簇上，如：

```
CREATE TABLE CUSTOMER(
  CUSTOMER_ID       VARCHAR2(25) NOT NULL,
  CUSTOMER_ADDRESS VARCHAR2(15)
  )
  CLUSTER ORDERING(CUSTOMER_ID);

CREATE TABLE ORDER(
  ORDER_ID            VARCHAR2(20)  NOT NULL,
  CUSTOMER_ID         VARCHAR2(25)  NOT NULL,
  ORDER_DATE          DATE
  )
  CLUSTER ORDERING(CUSTOMER_ID);
```

在Oracle中，访问聚簇记录可以通过聚簇键的索引或是聚簇键的散列函数。选择索引或是

选择散列的理由和前面提到的选择索引文件和散列文件类似（表6-3）。最好在记录相对静态的情况下聚簇记录，如果记录经常被添加、删除或是修改，就可能导致空间的浪费，也会使在首次加载后再将相关记录靠近在临近位置的工作变得困难。聚簇是设计人员用来改善那些经常在查询和报表中一起使用的表的性能的一种方式。

6.6.5 设计文件控制

关于数据库文件设计的另一个方面是对于文件的控制，它用来防止文件被破坏或污染，或是在文件被损坏后重构它。数据库文件是以DBMS所专有的文件格式存储的，这是一种基本层次的访问控制。用户可能希望对字段、文件和数据库提供额外的安全控制。在第8章和第12章中，我们会介绍一些可行的方法。简而言之，文件可能被损坏，所以最重要的是快速恢复被损坏的文件。利用备份过程可以提供文件和对文件进行修改的事务的一个副本。当一个文件损坏时，可以利用文件的副本和事务的日志来将它恢复到一个正确的状态。在安全性方面，最有效的策略是加密文件的内容，这样就只有那些知道解密过程的程序才可以看到文件内容。在第8章中我们讨论关系数据操纵语言SQL和在第12章中介绍关于数据活动和数据库管理的内容时，会涉及这些重要的课题。

6.7 索引的使用和选择

大多数的数据库操纵需要定位满足某个条件的一行（或多行），比如需要读取具有某个邮政编码的所有顾客或是特定专业的全体学生。在实际的数据库应用中，由于数据库的表通常很大，所以对表进行扫描以定位需要的行往往需要很长的时间。使用前面所介绍的索引可以大大加快这一过程，因此定义索引是物理数据库设计的重要部分。

在前面几节关于索引的讨论中，读者已经知道索引可以创建在主键或（和）辅键上，通常情况下会为表的主键创建索引。索引自身也是一张表，它包含有两列，键和包含该键值的记录或记录组地址。对于为主键创建的索引而言，每一个键值所对应的记录地址都是惟一的。

6.7.1 创建惟一键索引

前面提到的定义在聚簇上的表Customer有主键Customer_ID，可以使用下面的SQL语句为这个字段创建惟一的键索引：

```
CREATE UNIQUE INDEX CUSTINDEX ON CUSTOMER (CUSTOMER_ID) ;
```

在这个语句中，CUSTINDEX是用来存放索引项的索引文件名；ON子句指定为哪个表创建索引以及组成索引键的列（列组）。当这条命令执行时，所有Customer表中已有的记录都会被索引。如果表中存在重复的Customer_ID值，则该命令会失败。在索引创建后，DBMS会拒绝一切违反Customer_ID惟一性约束的数据插入和修改操作。

ON子句也可以用来定义复合的惟一键，此时需要列出该键的所有组成元素。比如在一个有关顾客订单的表上，Order_ID和Product_ID组成一个复合的惟一键，可以使用下面的SQL语句来为表Order_Line创建索引：

```
CREATE  UNIQUE INDEX LINEINDEX ON ORDER_LINE (ORDER_ID, PRODUCT_ID) ;
```

6.7.2 创建辅键索引

数据库用户经常需要在关系中根据属性值（而非主键值）来读取特定行。例如，在Product表中，用户可能希望选取满足下列条件任意组合的记录：

- 所有的桌子产品（Description＝‘Table’）
- 所有的橡木家具（Finish＝‘Oak’）
- 所有的餐厅家具（Room＝‘DR’）

• 所有价格低于500美元的家具（Price < 500）

为加快类似查询的读取速度，我们可以为使用到的每个属性创建索引。比如，我们可以用以下命令在表Product的Description字段上创建一个非惟一的索引：

```
CREATE INDEX DESCINDX ON PRODUCT (DESCRIPTION);
```

请注意，没有使用UNIQUE，这是因为对于辅键来说，属性的值是可以重复的。和惟一键索引一样，辅键索引也可以创建在属性的组合上。

可以使用类似的命令结构来创建位图索引，用户可以通过以下的语句为Description字段创建位图索引：

```
CREATE BITMAP INDEX DESCBITINDX ON PRODUCT (DESCRIPTION) ;
```

6.7.3 何时使用索引

在物理数据库设计中，必须确定创建索引的属性，这需要在索引带来的性能改善和记录的插入、删除以及修改的额外开销间取得平衡。对于主要工作是数据读取的系统，如决策支持和数据仓库应用等，应该尽量地使用索引。而对于主要工作是支持事务处理，有大量的数据修改请求的系统，应该谨慎地使用索引，因为此时索引会带来额外的开销。

下面是一些在关系数据库中选择索引的经验：

1) 在大的数据表上，索引的效果特别突出。

2) 为每个表的主键创建一个惟一索引。

3) 那些经常出现在SQL查询WHERE子句的列比较适于创建索引，它们可能用来限定行（比如WHERE FINISH = 'Oak'，此时Finish字段上的索引会加快读取速度），也可能用来在表间作联结（比如WHERE PRODUCT.PRODUCT_ID = ORDER_LINE.PRODUCT_ID，此时为表Order_Line在字段Product_ID创建辅键索引，为表Product的主键Product_ID创建主键索引会改善存取的性能）。在后一种情况中，可以看到索引创建在用于和其他表联结的表Order_Line的外键上。

4) 为那些在ORDER BY（排序）和GROUP BY（分类）中使用的属性建立索引。需要特别注意这种情况，要确定DBMS是否会使用子句中列出的这些属性上的索引（比如Oracle会使用ORDER BY 属性上建立的索引，但不会使用GROUP BY 属性上的索引）。

5) 如果某个属性的可选值很多，应该为它创建索引。Oracle的建议是，如果某个属性的不同值少于30个，则索引的用处不大；而如果属性的不同值多于100个，则索引的效果很明显。同样的道理，只有当查询到的记录数不多于文件中所有记录数的20%时，索引才是有益的（Schumacher，1997）。

6) 检查DBMS中可能有的对于一个表上索引数目的限制。很多系统要求索引数不多于16个，并对索引的键值大小有限制（比如每个复合键值不多于2000字节）。如果系统中存在着类似的限制，就必须选择那些对于系统性能改善贡献最大的辅键索引。

7) 当为含有空值的属性创建索引时，需要特别注意。在很多DBMS中，含有空值的行不会在索引中引用（所以它们不会在类似于ATTRIBUTE = NULL的索引搜索中找到）。类似的查询需要扫描文件才能完成。

如何选取索引是物理数据库设计中最重要的决定，但还存在其他可以改善数据库系统性能的方法。其他的方法可以处理以下问题：减少记录重新定位记录的代价，优化文件中空余空间的使用，以及优化查询处理的算法（参见Viehman，1994，有关于使用这些其他方法来改善物理数据库设计的讨论）。我们会在后面的小节中简要地介绍有关查询优化的问题，因为类似的优化可以被用来修正使用数据库系统的一些设计选项的方式，这些设计选项在大多数情况下对于提高数据处理的性能是有益的。

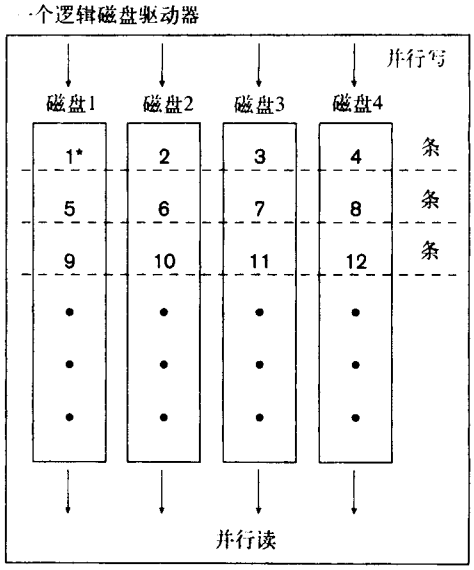
6.8 RAID：通过并行处理来改善文件访问的性能

在本章的前面几节中，读者已经学习了如何通过非规范化和聚簇来让经常一起使用的数据在内存中彼此接近。如果在程序中一起使用的数据可以保存在一个物理的数据页上，那么非规范化和聚簇会有效地减少数据访问所需的时间。对于小记录，类似的要求通常是可以满足的。但是需要一起或连续使用的数据可能被存放在多个单独的页上。在这种情况下，读写这些数据需要多次顺序的磁盘输入/输出操作。此时非规范化和聚簇就无法为改善数据的读写性能提供太大的帮助。

计算机技术发展的两个必然趋势是价格越来越低，所占空间越来越小，这使得计算机组件的冗余和相应的容错在物理上和经济上都变得可行。使用多个小的计算机组件也给数据的并行处理提供帮助。数据并行处理的效果相当显著，例如当使用并行处理时，4次输入/输出操作所耗费的时间相当于1次操作的时间。这种特性对于电子商务这样的联机超大型数据库系统非常重要。

数据库设计人员可以使用硬件或称为廉价冗余磁盘阵列（Redundant Array of Inexpensive Disk，RAID）软件技术来实现并行的数据库处理和容错。在RAID中使用一组物理磁盘，对于数据库用户（以及程序）而言，它们就好像是一个大的逻辑存储单元。因此RAID并不会改变应用程序和数据库查询的逻辑或物理结构。

为了使RAID的输入/输出性能达到最优，就必须使所有的磁盘都投入工作。分条(striping)可以实现工作负载在磁盘上的平衡。如图6-10所示，称为**条**(stripe)的数据段跨越所有的磁盘。逻辑上连续的数据页（例如，一个记录占据的多个页，记录的聚簇，一个文件里逻辑上的顺序记录）在物理磁盘上以循环的方式存储。以图6-10中假定的RAID存储为例，就可能只需要读取1页的时间来读取*逻辑顺序*的4页。在RAID中读取一个存放在多个物理页中的非常大的数据记录(可能包含音频、视频)，可能只需要原来几分之一的时间。分布在多页上的相关记录(聚簇)可以在同时一次读取。



注：*逻辑顺序的页在磁盘上交错。

图6-10　使用4个磁盘和分条的RAID

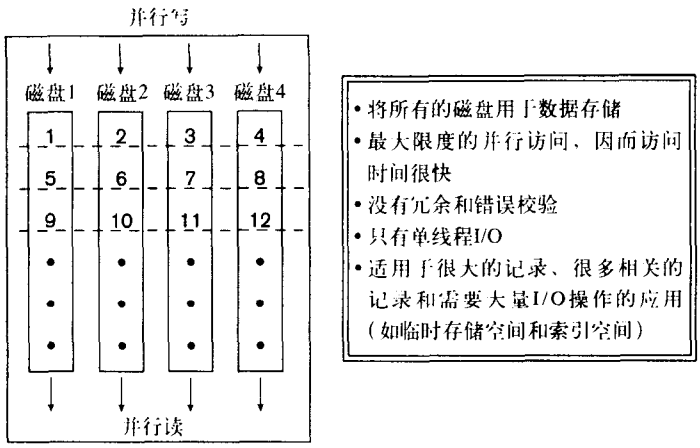
大多数现在的多用户操作系统（如Microsoft NT、UNIX和Novell Netware）都支持某种形式的RAID。这些操作系统可以进行并行的读取操作，并将得到的结果联结成一个逻辑记录返回给需要的程序。这些系统也支持多线程，允许多个不同的用户程序或任务同时发出读/写请求。RAID也可以使用特殊的示配器在硬件中实现，这样就不再需要由操作系统来将输入/输出请求分解为可并行的步骤。

RAID有一个极大的危险：可能增加整个数据库内磁盘失败的机会。如果一个磁盘平均在120万次操作后出现一次错误，则采用4个磁盘的RAID就会在30万次操作后平均出现一次错误。为解决这个问题并使磁盘存储可以容错，很多类型的RAID技术冗余地存储数据，这样就保证至少有一个数据副本是可用的，RAID也常常使用差错校验码来恢复损坏和丢失的数据。

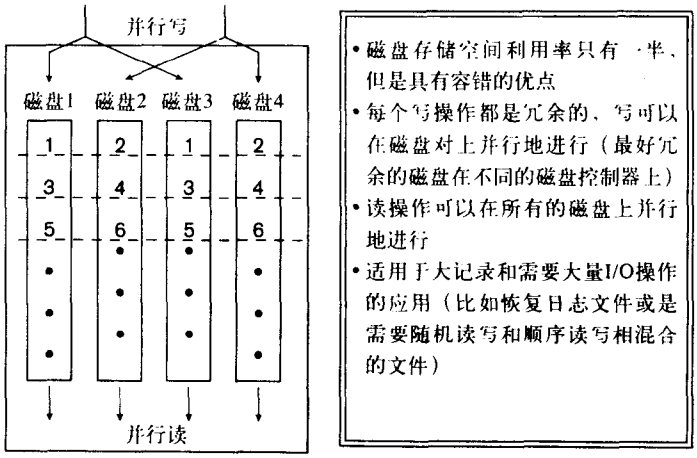
选择RAID的层次

一个文件和数据库的设计者可能拥有六种或更多的可供选择的RAID体系架构，它们提供冗余存储或差错矫正。虽然RAID-0不提供冗余存储，但是它最大限度地利用并行化，可以提供最价的性能。图6-11比较了六种形式的RAID和它们的一些关键特性。不同的磁盘和操作系

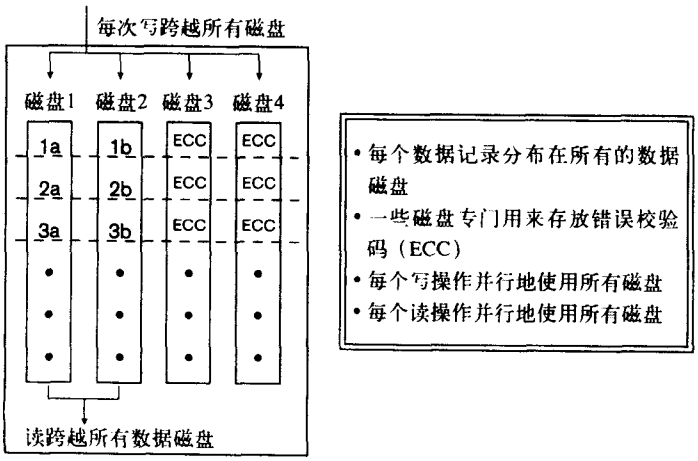
统可能会提供其他的方案。



a）RAID-0（分条）

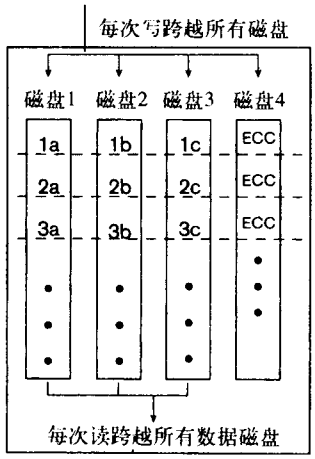


b）带有两对镜像磁盘的RAID-1



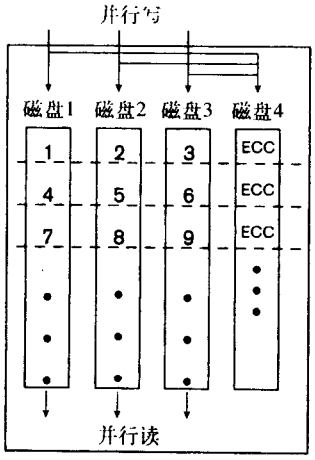
c）一半的磁盘用来存放数据的RAID-2

图6-11 6个层次的RAID（注意：数字是顺序数据块号，字母表示数据块的段）



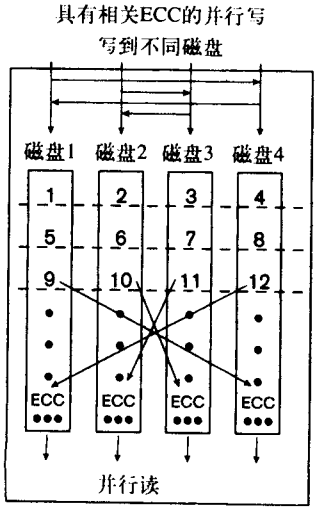
- 每个数据记录可以分布在所有的数据磁盘，或者将单独的记录放在一个条中
- 一个磁盘被专门用来存储错误校验码（比RAID-2有更多的数据存储空间）
- ECC用来恢复条内损坏的数据或磁盘
- 每个写操作并行地使用所有磁盘（可能较慢）
- 每个读操作并行地使用所有磁盘（可以非常快）
- 同一时间只有一个程序可以访问阵列

d）RAID-3



- 在一个条里可以存放多条记录
- 一个磁盘被专门用来存储错误校验码（比RAID-2有更多的数据存储空间），但是当存在频繁的写操作时，这个磁盘会很繁忙
- ECC用来恢复条内损坏的数据或磁盘
- 写操作可以并行地进行
- 读操作可以并行地进行
- 多个程序的访问可以在阵列中并行地进行

e）RAID-4



- 可能在一个条里有多条记录
- 所有的磁盘都用来存放数据和错误校验码，所以没有写ECC的瓶颈
- ECC用来恢复条内损坏的数据或磁盘
- 写操作可以并行地进行
- 读操作可以并行地进行
- 多个程序的访问可以在阵列里并行地进行

f）RAID-5

图6-11　（续）

1. RAID-0

RAID-0（参见图6-11a）是能提供最快的读/写速度和最有效的数据存储的磁盘阵列类型。在RAID-0（通常称为磁盘分条）中，阵列内所有的磁盘都被并行地用于读写应用数据。如果数据库应用需要访问非常大的记录（需要访问多个条），或是一组相关的记录，抑或是读写操作特别频繁，就可以考虑采用RAID-0。由于RAID-0没有采用任何的数据冗余和差错校验码，所以任何一个磁盘的错误都会导致整个数据库崩溃。此外，RAID-0只能同时执行一条I/O命令（不支持多线程I/O）。RAID-0的瓶颈是磁盘控制器，它可能会被大量并发的I/O请求所堵塞；理想情况下，应该在磁盘间使用多个磁盘控制器。

2. RAID-1

改善容错的一个方法是将数据冗余存放，RAID-1（参见图6-11b）是一个完全的冗余存储，也称为磁盘镜像。在这里，每一页都被冗余地存储。虽然此时整个磁盘阵列的存储空间只有原来的一半，但数据库系统有很高的容错能力；因为某一页在两个副本中都损坏的概率是非常小的。一个由两个磁盘构成的RAID-1阵列不使用分条，但是可以增加多个双RAID-1磁盘来组成一个带有分条的大阵列（参考下面的RAID 0+1）。在RAID-1中，每一页的写操作都必须并行地写到两个磁盘上，所以不会有任何的性能改善。但即使是来自于不同应用程序（多线程）的独立数据库查询，两个读操作也可以在每个阵列中并行执行。RAID-1在单用户和多用户环境下都具备良好的性能和容错能力，磁盘价格的不断下跌也使得RAID-1无可争议地成为使用最广泛的RAID技术形式。

3. RAID 0+1

正如前面所看到的，RAID-0通过并行读写提供极高的性能，而RAID-1则通过冗余存储以提供容错能力。通过对一个RAID-0系统镜像，就可以得到RAID-0和RAID-1的所有优点。这种方法称为RAID 0+1，或RAID -10（Compaq）、RAID-6（HP）（图6-11中没有给出RAID 0+1的情况）。RAID 0+1提供极大的数据存储量，高可靠性，并可以防止磁盘失败（Musciano，1999）。但是这种方法非常昂贵，它只能用于需要快速响应时间和极高稳定性的系统中，比如军事的实时命令和控制、空中交通管理、实用过程控制和一些电子商务的应用（如联机的投资活动）。

4. RAID-2

在RAID-2（参见图6-11c）中，一些磁盘使用数据分条以提供并行数据处理（无冗余存储），而另一些磁盘则专门存放错误校验码。这些错误校验码可以用来检测数据条中的错误，并可以在条中的页损坏时重建数据。和RAID-1采用冗余存储来恢复损坏的数据的方法不同，RAID-2（和以后的层次）利用更为有效的错误校验码（或奇偶码）来恢复数据。由于目前的大多数磁盘系统已经在单个的物理页里提供页级的错误校验码或奇偶校验信息，所以RAID-2不如RAID-3理想。

5. RAID-3

和RAID-2一样，RAID-3在磁盘间将数据按条存放，但只使用一个磁盘来存储条级的错误校验码（参见图6-11d）。嵌入在每页里的错误校验码用来检测页级的错误，而ECC页用来恢复损坏的页。假设图6-11d中的磁盘3崩溃，则在每条中的页3都可以从该条中其他数据页和错误校验页中使用一个异或计算得到恢复。为实现这一功能，所有对于某一条内页数据的更新都必须将该条内的所有数据读出以重新计算ECC页。当存在很长的记录，以致一个记录需要一个或多个条时，通常使用RAID-3。此时每一个逻辑的读/写都可以分布到阵列的磁盘上去。也正因为每一个逻辑的读/写操作需要利用所有的磁盘，所以就不能同时处理来自于多个用户或程序的要求。RAID-3比较适宜于在需要大量读取、少量修改的单用户环境中使用。组成RAID-3条

的磁盘越多，磁盘的总体利用率就越高；系统的瓶颈是存放ECC的磁盘。

6. RAID-4

和RAID-3一样，RAID-4也使用一个单独的磁盘来存放错误校验码，但是一个条可以存放多条记录（参见图6-11e）。因此对于不同记录的读操作就可以并行进行，无论它们是来自于同一个程序，还是不同程序。系统的瓶颈是用于存放校验信息的磁盘，因为每当数据有修改时，都必须访问它以便重写错误校验页。对联机事务处理等带有频繁更新的数据库应用来说，这样会造成对奇偶校验磁盘的争用，由此影响并行处理的效能。RAID-5可以避免这种瓶颈情况。注意，有些作者并不区分RAID-3和RAID-4。

7. RAID-5

RAID-5也被称为循环奇偶性阵列（如图6-11f所示），它不使用一个单独的磁盘来存放奇偶校验信息，而是在每个磁盘上同时存放数据和奇偶校验页。读操作可以在所有的磁盘上并行进行，写操作则需要同时访问一个或多个磁盘，以便写入新记录并写入该记录的校验磁盘。出于恢复的考虑，不同的记录将它们的错误校验码存放在不同的磁盘上。这样写操作也可以并行地进行，像RAID-4一样发生阻塞的可能性很小。这是由于随机的记录修改会使随机分布在磁盘间的差错校验码页更新。

8. RAID的性能

在前面的讨论中，RAID-1、RAID-3和RAID-5是最主要的提供容错的RAID系统。由于RAID-3是为单用户和数据密集的环境设计的，因此它通常有其特定的应用范围。对于大多数数据库应用来说，通常在RAID-1、RAID 0+1和RAID-5中进行选择。

RAID的生产商之一Adaptec（http://www.dpt.com）使用不同数目和不同传输率的磁盘组成多种阵列的样式，比较RAID-1和RAID-5的性能。正如所预期的一样，RAID-5在使用存储空间方面比较高效，因为它不是为每个数据都提供副本。在读方面，RAID-5也比RAID-1的性能突出，这是因为当阵列内有相同数目的磁盘时，RAID-5有更多的并行操作。但是对于写操作而言，RAID-5所花费的时间可能是RAID-1所需时间的1.67到3倍（取决于基准程序和阵列的配置）。因此对于易于发生变化的数据而言，RAID-5不如RAID-1。Adaptec同时估计，为使写操作的性能达到只使用一个磁盘的两倍，RAID-1需要一个4个磁盘的阵列，而RAID-5需要7～12个磁盘。

哪种RAID是最好的选择呢？对于容错和数据库维护程序（需要高的稳定运行时间），或当只有两个磁盘控制器时，RAID-1是最好的选择。RAID-1虽然代价较高，但是对各种工作负载都有较好的支持。RAID-5适用于对大数据集的密集读操作，并且需要至少3个（通常是5个）磁盘。随着磁盘不断变大，RAID-5就变的不那么合适。

存储设备是一个快速发展的领域。像存储区域网络（SAN）和网络附加存储（NAS）这样的新技术正在大企业存储环境中不断涌现（Shah，1999）。

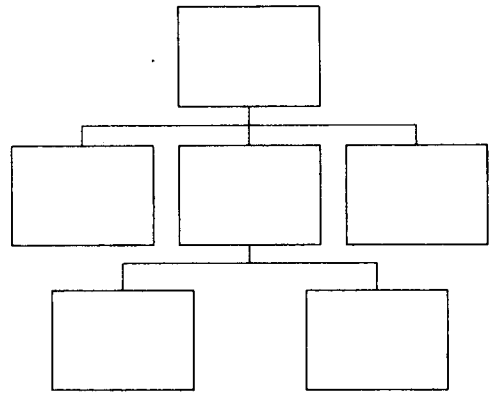
6.9 数据库设计

大多数的现代信息系统包括数据库管理系统或数据仓库系统利用数据库技术来支持数据存储和读取。根据前面的定义，数据库是一组逻辑上相关数据的集合，用于满足一个组织内的多个用户的信息需求。数据库内文件之间的联系是通过概念模型和逻辑模型的联系来定义的，这些联系隐含对数据访问的路径。不同种类的数据库系统支持不同类型的访问路径，所以对于DBMS和数据仓库技术的选择是一种将所需访问路径与数据系技术功能相匹配的功能。

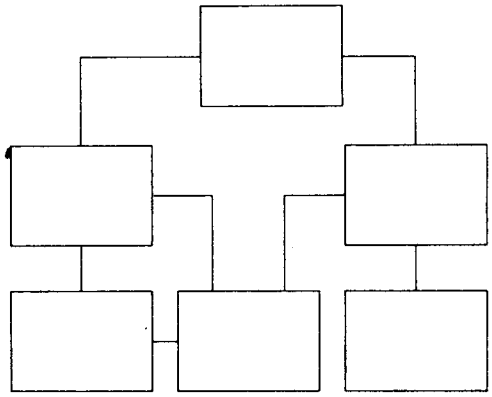
数据库体系结构选择

不同的数据库管理系统和数据仓库系统使用不同的数据定义和构造形式，统称为数据库的

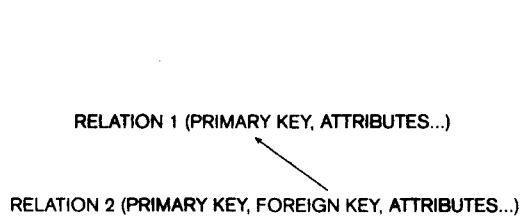
体系结构。确定何种数据库体系结构最适合数据库是数据库设计的基础。数据库或数据仓库系统一般支持5种不同体系结构，图6-12比较了这5种体系结构。



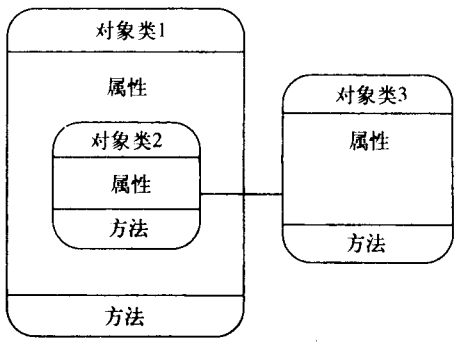
层次数据库模型



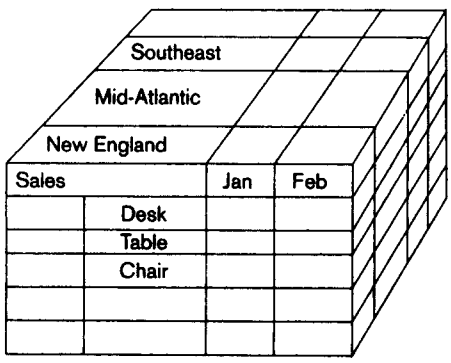
网状数据库模型



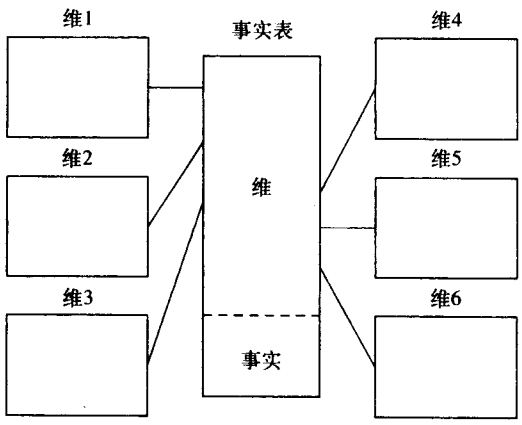
关系数据库模型



面向对象数据库模型



多维数据库模型——多维立方体视图



多维数据库模型——星型模式视图

图6-12 数据库体系结构

1. 层次数据库模型

在这个模型中，文件用类似于树或族谱图的形式以自顶向下的结构来组织。数据之间存在着嵌套和一对多的联系。顶端的文件称为根（root），底端的文件称为叶（leaf），中层的文件

有惟一的父文件（又称属主）和若干个子文件。作为一种最老的数据库体系结构，很多层次化的数据库系统仍然在大型的企业中使用。如果概念数据模型是一棵树，或者大多数的数据访问从根文件开始，那么层次数据库模型就相当适用。层次数据库技术主要应用于数据量很大的事务处理和MIS系统。新数据库系统很少再使用层次化的DBMS，这是由于新型应用所需要的往往不止是简单的事务处理或事务数据的统计。

2. 网状数据库模型

在这个模型里，一个文件可以与任意数目的文件相关联。这个模型很灵活（层次模型只是网状模型的特例），它可以实现任何联系。但是由于这种实现形式往往在不同文件内的相关记录间大量使用指针，因而会导致存储空间的浪费和维护上的困难。通常情况下，网状模型只支持沿着弧的一对多联系，但是某些网状系统也支持多对多联系。在主机系统中，网状模型仍然被广泛使用于大数据量的事务处理应用。由于数据库设计人员对于数据组织有很大的控制权，所以可以设计出高度优化的数据库。例如，每个记录类型（用网络中的盒来表示）可以使用散列算法来组织，或和相关的记录邻接存放（这是早期的聚簇形式）。和层次模型相比，网状模型的系统可以满足更广泛的操作需求。但是网状系统需要复杂的编程和数据库设计技术，因此主要被那些具有相关技术专家的企业所使用。

3. 关系数据库模型

这是最流行的数据库模型，它为每个关系定义简单的表以及多对多联系。表间的交叉引用键用来关联相关的表，表示实体间的联系。主键和辅键索引根据条件对数据进行快速访问。大多数应用是利用关系数据库管理系统构建的，还存在许多关系DBMS产品。第5章中我们详细讨论了关系数据模型的特性，第9章中会介绍一种关系查询语言——按例查询（query-by-example），第7～8章中会介绍使用最为广泛的SQL关系数据操纵语言。

4. 面向对象数据库模型

属性和操作属性的方法封装在称为对象类的结构中。可以通过将一个对象类嵌套或封装在另一个类里来表示类间的关联，新的对象类可以从更一般化的对象类中导出。这种数据模型的一个主要优点是支持如图像、视频和声音等复杂数据类型，就像支持简单数据类型一样容易。这是一种最新的DBMS技术，大型企业有选择地使用它来支持复杂的数据类型和事件驱动的程序。第14章和第15章将介绍如何使用这种模型进行概念数据建模和数据库实现。

5. 多维数据库模型

这是一种在数据仓库中使用的数据库模型，有两种理解它的方法。第一种方法是将数据看做是多维的立方体，其中的每一个单元包含一个或多个简单的属性，维则是用来对原始数据分类的。这些种类（或维）被用户用来对数据进行统计和分片，这样的数据通常包括时间段、地理位置、业务种类和人名等。每个单元所包含的是与它的所有维值都相关的数据。比如，一个单元可能包含涉及特定时间段、地点和销售人员的物品销售的属性。另外一种理解方式是通过星型模式（star schema）。中心是事实表，相当于多维视图的一个单元。这个表包含有所有的原始属性和一个由周围环绕的维表主键构成的复合键。每一个周围的维表定义一种对数据进行分类的方法，比如对每个销售人员的全部描述信息。多维数据库模型设计的关键问题是事先定义数据的最小公分母和用户用来对原始数据进行统计的维或分类。尽管多维数据库模型只是关系数据模型和网状数据的特例，但是已经开发出一些专门为维数据处理进行优化的DBMS。第11章会讨论多维数据模型和数据仓库。

在本书中，我们只详细讨论三种最新的数据库模型：关系数据库模型、面向对象数据库模型和多维数据库模型。如果读者想了解有关层次和网状数据库模型的更多知识，可以自行参阅

本章后面进一步阅读中的相关资料。

6.10 优化查询性能

现在进行物理数据库设计的主要目标是优化数据库处理的性能。数据库处理包括对数据的添加、删除和修改，还包括大量的数据检索活动。对于那些读取操作远多于维护操作的数据库系统来说，优化查询性能（最终用户的联机查询或是脱机的报表生成等）就显得非常重要。在本章前面我们已经介绍了大部分可以用来优化数据库设计以满足查询要求的设计因素（如聚簇、索引和文件组织等）。在本章的最后一节，我们会讨论一些现在很多DBMS所提供的用于数据库设计和处理的其他高级选项。

数据库设计人员优化查询的工作量是和所使用的数据库管理系统密切相关的。由于聘请专业级的数据库开发人员的成本很高，所以数据库和查询的设计工作越少，开发和维护数据库系统的开销就会越少。一些DBMS在处理查询和合理安排数据的物理存储以优化数据读写时，不需要数据库设计人员和查询编写人员进行太多的控制。而另外一些系统则需要应用的开发人员进行很多控制，需要大量的工作来调整数据库设计和查询的结构才可以获得可接受的性能。有时候，数据库系统的工作量变化很大，设计的选项又如此的微妙，因而很难获得良好的性能。如果数据库系统的工作是相对固定的，比如对于数据仓库来说，批量的数据更新很少而主要是数据量很大的查询操作，此时就可以通过使用DBMS中的查询优化器和良好的数据库、查询设计来取得很好的性能。比如，可以调整NCR Teradata DBMS以完成数据仓库环境下的并行处理。在这种情况下，数据库和查询的设计人员就很难再对改善数据库存储和处理数据的性能做出特别的贡献。但是这种情况很少，所以数据库设计人员仍然应该仔细研究那些可以改善数据库处理性能的选项。

6.10.1 并行查询处理

过去几年中，计算机体系结构中一个很大的变化是越来越多地在数据库服务器中使用多处理器。数据库服务器通常使用对称多处理器（SMP）技术（Schumacher，1997）。为利用这种并行处理的功能，一些先进的DBMS可以将查询分解为多个在相关处理器上并行处理的模块。最常见的方法是将查询复制，然后让每个查询副本在一部分数据库上（通常是水平分割（一组行））运行。分割需要预先由数据库设计人员定义。相同的查询在单独的处理器上分别针对数据库的一部分运行，每个处理器产生的中间结果合并结合起来组成最终的结果，就好像查询是在整个数据库上运行的一样。

假设一个Order表中有几百万行记录，这导致查询的性能下降。为确保今后对于该表的扫描能够至少使用3个处理器并行运行，可以使用下面的SQL语句来改变表的结构：

```
ALTER TABLE ORDER PARALLEL 3
```

设计人员要调整每个表使之达到最佳的并行性，通常这种方法并不常见，因为这需要多次对表进行修改才能够达到合理的并行度。

并行查询的处理速度可能是非常惊人的。在Schumacher（1997）报告的一次测试中，相对于普通的表扫描，使用并行处理可以将查询时间缩短一半。由于索引本身也是一个表，因此也可以采用并行结构来设计索引以加快扫描的速度。Schumacher（1997）还报告了一个例子，通过使用并行处理，创建一个索引的时间代价从大约7分钟缩短到5秒！

除表扫描以外，其他的查询操作也可以并行地进行，如某些类型的相关表的联结操作，查询结果的分类，几个查询结果的组合（称为联合），行的排序和计算聚集值等。行的修改、删除和插入操作同样可以并行地处理。此外，一些数据库创建命令的性能也可以通过并行处理得

到改善，包括索引的创建和重建，利用数据库中的数据创建表等。必须使用规格说明预先配置Oracle环境以指定虚拟的并行数据库服务器的数目，此后，查询处理器就会为每个命令选择它认为最好的并行处理方法。

6.10.2 对自动查询优化的重载

有时候，查询编写者知道（或可以了解）查询的一些关键信息，而DBMS中查询优化的模块却可能忽略或是不知道这些信息。当掌握这些关键信息时，查询编写者才能知道查询执行的最优路径。但是，在编写者确定自己的路径为最优路径前，他首先应该知道查询优化器（通常会选择预期查询时间或代价最小的查询处理方法）会如何处理这条查询，尤其是对于那些以前没有执行过的语句更是如此。幸运的是，大多数的关系DBMS可以在实际执行语句以前，告知用户优化器的执行计划。像EXPLAIN或是EXPLAIN PLAN（与数据库系统相关）这样的命令会显示出查询优化器准备如何访问索引、使用并行处理器和对表作联结来产生结果。如果用户在实际的数据库命令前加上EXPLAIN子句，查询处理器就会显示处理查询的逻辑步骤，并在实际访问数据库以前停止。查询优化器基于每张表的统计数据来确定最优的执行路径，比如每行的平均长度或是表中的行数等。在必要时可以让DBMS计算当前数据库最新的统计数据（比如Oracle中的Analyze命令）以得到准切的估计查询代价。用户可以使用多种方式书写一个查询语句，并通过EXPLAIN命令获得系统对于它们性能的预期。然后用户就可以选择预期处理时间最少的查询来执行，或是当觉得代价过高时撤销该查询。

还有其他的改善查询处理性能的方法。在某些DBMS中，用户可以让DBMS使用不同的处理步骤，或是使用DBMS的功能，比如并行服务器等，而不是使用查询优化器所认为的最佳的计划。

例如，用户希望统计某一个销售代表Smith所处理的订单数目。在Oracle数据库中，只有当一个表被扫描时才使用并行处理，而如果通过索引访问表就不使用。此时用户可以指定系统总是对整个表的扫描使用并行处理，完成这个查询的命令如下：

```
SELECT /*+FULL (ORDER) PARALLEL (ORDER, 3) */ COUNT (*)
FROM ORDER
WHERE SALESPERSON = "SMITH";
```

在/* */内的子句是传递给Oracle的提示，它会覆盖Oracle为查询自然生成的查询计划。因此，提示是与查询语句相关的，但在此之前，用户应该修改表的结构以适应并行处理的需要。

6.10.3 选择数据块大小

在本章的前面已经提到，数据在RAM和磁盘之间以块或页为单位进行传输。数据块的大小会对查询的性能产生很大的影响。如果数据块太小，对于表中行的一次或多次访问会导致很多次的物理I/O操作；如果数据块太大，就会浪费时间去传输本不需要的数据。通常，块的最小值为2K字节，最大值则由计算机的操作系统决定，往往是32K或更多。一旦为数据库确定了数据库块的大小，就很难改动；除非先卸载数据，再更改数据库，最后重新装载数据。

要选择合适的块大小，就必须在5个性能因素间进行权衡，这5个因素（Yuhanna，2000）列举如下：

- **块争用**　这是对几个I/O操作需要同时访问一个数据块的可能性的度量，较小的块会减少争用的次数。如果在同一个数据库上运行多个并发作业，例如有许多联机用户，那么这种环境下争用的情况就会比较突出。
- **随机行访问速度**　这个指标衡量访问表中任意一行的速度，此时，小的数据块会有比较好的表现。在联机事务处理中，随机的行访问比较普遍。

- **顺序行访问速度** 这个指标衡量对表进行扫描的速度，此时大的数据块会比较好。如果使用较大的块，一个I/O操作就可以读取许多行高速缓存在内存中，从而减少查询所需的需要读取表中所有行的物理I/O操作的数目。在使用聚簇（本章前面介绍过）的情况下，大的块只需要一个I/O操作就可以将某个聚簇高速缓存起来。在决策支持系统、数据仓库应用和产生事务概要报告的应用中，会有比较多的顺序扫描。
- **行大小** 这是指表中一行所有字段的长度之和，通常情况下块的大小最好等于物理表中的行大小，或是它的整数倍。
- **开销** 这是DBMS为管理产生查询结果和进行其他数据库工作所需的I/O操作所产生的时间代价。小的块会比大的块带来更多的开销。

很难有一个块大小可以满足上面的所有要求。一般来说，小的块适用于联机事务处理应用，而大的块适用于决策支持数据库和数据仓库系统。如果一个系统的工作负载不那么确定，对数据块的调节就会很困难。

6.10.4 在磁盘控制器间平衡I/O操作

一个磁盘控制器会管理附属于它的磁盘所具有的I/O操作。磁盘控制器越多越好（并行度越高，性能越好），但是它的数目会受到经费的限制。多磁盘控制器的优点在它们都工作时才会显示出来，所以设计人员需要合理地分布数据库和DBMS系统文件，以使工作负载平均分配到每个控制器上，最好是在每个磁盘上。

即使加载了数据，表仍然可以容易地从一个磁盘移动到另一个磁盘，所以初始的文件存放配置是可以改变的。为改善查询处理的性能，设计人员必须做好以下工作：

- 了解文件在磁盘上的分布情况以及控制器和磁盘的相互关系。
- 知道在数据库上运行的预定义程序和特别查询的本质（应该关注最重要的程序或是性能特别差的日子或时间）。
- 收集有关磁盘、控制器的使用信息和表（分割）的访问情况统计信息。
- 通过在磁盘和控制器间移动表来重新平衡工作负载。

一般来说，如果不能将一个查询（或是同时运行的多个查询）所需的数据存储在相同的数据块上，那么就最好将它们分布在不同控制器的不同磁盘上以便对数据进行并行的访问。和前面一样，虽然有可能，但是在一个工作负载不断变化的系统中进行优化也是极为困难的。设计人员需要集中关注几个最重要的应用（或每天性能都特别不理想的程序），分析这些应用，重新分配文件的存储为它们提供尽可能最好的性能，同时尽量不对其他程序造成太大的影响。

6.10.5 设计良好查询的建议

本章的前面讨论了许多技术和方法，这些技术和方法用来改进数据库和查询的设计以加快查询处理的速度。很多数据库专家都对于提高查询效率给出了建议，它们中的一些在前面没有提及。读者可以参阅DeLoach（1987）、Holmes（1996）和Kurka（1999）的著作来了解更多有关在不同设置下改善查询效率的建议。下面概要地介绍其中一些普遍适用于各种环境的建议：

- *了解在查询处理中索引的应用原理* 很多DBMS在一次查询中对每张表只使用一个索引。设计人员需要了解DBMS如何选择索引，并监控索引的使用情况，然后就可以删除那些很少使用的索引，由此改善数据库更新操作的性能。一般来说，有相等条件（如WHERE Finish = "Birch" OR "Walnut"）的查询比其他具有更复杂的限定条件的查询（如WHERE Finish NOT = "Walnut"）处理的速度要快，这是因为相等条件可以使用索引的值来进行判定。设计人员需要学习DBMS处理查询中不同类型子句的方法。
- *在查询中为字段和文字使用相容的数据类型* 使用相容的数据类型可以让DBMS不需要在

查询处理时进行类型转化。

- *编写简单的查询*　通常来说，DBMS比较容易处理简单形式的查询。比如由于关系数据库管理系统基于集合的理论，所以可以编写处理行集和文字集的查询。
- *将复杂的查询分解为多个简单的部分*　由于DBMS可能只为每个查询使用一个索引，所以最好将复杂的查询分解为多个比较简单的部分（这样就会为每一个部分都使用索引），然后将这些简单查询的结果结合起来。例如，由于关系DBMS使用集合，所以DBMS将两个独立的简单查询的结果集进行UNION操作是很容易的。
- *不要将查询嵌套*　正如读者在第7、8章中将学习到的，SQL语言支持查询语句的嵌套形式（内部的查询称为子查询）。在产生同等结果的情况下，通常不包含子查询的查询会有更好的性能。
- *不要将表和自己结合在一起*　如果可能的话，避免在一个查询内为表赋予两个（多个）不同的角色（称为自联结，将在第7章中介绍）。通常为表作一个临时的副本，然后将表和副本联结，其性能会好得多。
- *为一组查询创建临时表*　在可能的情况下，应该在一组查询中重用数据。例如，如果一系列查询都引用数据库内的同一个数据集，那么就最好先将这个数据集存放在一个或多个临时表中，然后在查询中使用这些临时表。这样就会避免为每个查询重复进行相同的联结和重复扫描数据库的表。缺点是如果在查询运行的过程中，原始表的数据更新，那么临时表的数据不会改变。
- *结合更新操作*　如果可能，就应该把多个更新操作结合为一个更新操作。这样会减少查询处理的开销，并允许DBMS寻找并行处理的方法。
- *只检索那些需要的数据*　这样会减少数据块的访问和传输。虽然这看起来很明显，但是在一些查询的编写中常常破坏这一准则。例如，SQL语句SELECT * FROM EMP会从表EMP中读取所有行的所有字段。但是，如果用户只是想查看表中的某些列，传输额外的列就会增加查询处理所需要的时间。
- *必须为DBMS的排序操作提供索引*　如果数据要以排序的方式显示，而在需要排序的关键字段上又没有索引，就应该先读取无序的数据，然后在DBMS外进行排序。通常一个用于排序的实用程序会比不使用索引的DBMS排序快。
- *学习！*　跟踪查询的处理时间，使用EXPLAIN语句来评审查询计划，更多地了解DBMS处理查询的方法。参加DBMS厂商举办的特别培训来提高编写高效率查询的能力，这会使你更加了解查询优化器的情况。
- *最后，考虑处理特殊查询所需的全部时间*　这里所说的全部时间包括程序员（或最终用户）编写查询的时间和查询实际执行所需要的时间。在大多数情况下，对于特殊查询，最好让DBMS多作一点额外的工作而不是让用户更快地编写查询。技术的目标不就是要让人们工作效率更高吗？因此不要花费大量的时间去试图为某个特殊查询写出最高效的查询语句。只要书写一个逻辑上正确的查询（能产生正确的查询结果）即可，其他的工作就交给DBMS去作（当然应该预先使用EXPLAIN来确保查询是可行的，以避免其他用户的查询都受到严重延迟）。这就导出一个推论：如果可能，在数据库系统负载较轻的时候运行查询，因为总的查询处理时间是受其他DBMS上运行的工作影响的。

到此为止，我们结束了对优化数据库性能的高级选项的讨论。不是所有的选项都适用于每个DBMS，由于底层设计的不同，每个DBMS通常会有它自己特有的优化选项。读者应该阅读你所使用DBMS的用户手册，以了解可用的特殊优化选项。

本章小结

在物理数据库设计的过程中，设计人员将对数据的逻辑描述转化为对数据存取的技术规格说明。设计的目标是创建一种存储方案，以提供足够的性能，并确保数据库的完整性、安全性和可恢复性。在物理数据库设计中，设计人员通过将关系规范化、估计数据量、定义数据、考虑数据处理的需求和频度、用户预期和数据库技术的特征，来定义字段，设计记录、文件组织和数据库体系结构。

一个字段是应用程序数据的最小单元，相当于逻辑数据模型中的一个属性。设计人员要根据一系列因素为每个字段定义数据类型、完整性控制和如何处理缺失值。数据类型是为表示组织数据而使用的具体的编码方案，可以通过为数据编码和压缩数据来节省存储空间。字段的完整性控制包括为字段定义默认值、允许值的范围、是否可以使用空值和参照完整性。

物理记录是存储在邻接内存区域的一组记录，它是数据读取的基本单位。物理记录通常存储在页（或数据块）中，页是二级存储一次输入或输出操作时读取或写入的数据量。一页中记录的数目称为块因子。出于效率的考虑，一个关系的属性不一定存储在一条物理记录中，而来自于多个关系的属性可以存储在一条物理记录里。非规范化的过程将规范化的关系转化为非规范化的物理记录规格说明。非规范化将在单个I/O操作中经常一起使用的属性存放在一条物理记录里。水平分割是非规范化的一种，它将一个关系分解为多个记录规格说明，根据公共列的值将行存放到不同的记录中去。非规范化还包括垂直分割，此时将一个关系的列分布到不同的文件里，而主键需要出现在每个文件里。

一个物理文件是二级存储中命名的部分，它用来存储物理记录。物理文件里的数据是通过顺序存储和指针的结合来组织的。指针是用来定位相关字段或数据记录的一个数据字段。

文件组织负责在二级存储设备上安排文件中的记录。3种最主要的文件组织类型包括：1）顺序，根据主键值顺序地存放记录；2）索引的，记录以任意的方式存储，使用索引来定位存储记录的位置；3）散列的，记录存储的位置是通过散列算法将主键的值转化为记录地址而确定的。多种类型的物理记录可以存放在一个物理文件的聚簇上，这样就可以使经常一起使用的记录在二级存储中邻接存放。

索引文件组织是目前使用最广泛的方法之一。一个索引可以创建在惟一键上，也可以创建在辅键（非惟一键）上，此时一个键值可以对应多个记录。还可以使用一种新形式的索引，即位图索引，来创建位表，其中的一位表明对应记录具有相关的键值。一个联结索引代表来自两个（多个）不同表，但在某个字段上有相同值的行。散列索引表使得数据的存放和散列算法无关，这样可以通过在在不同字段上分别建立散列函数来访问相同的数据。索引对于加快数据读取非常重要，尤其是应用多个条件来对数据进行选择、排序和联结时更是如此。索引在很多情况下都有显著的作用，这些情况包括使用大的表、经常用来为查询设定条件的列、包含有大量不同值的列和数据读取而不是数据维护占主导的场合。

可以使用廉价冗余磁盘阵列来提高文件访问的效率和可靠性。RAID允许多个程序的数据块在不同的磁盘上并行地读和写，这比传统磁盘上的顺序I/O操作的效率提高很多。多种RAID层次的存在，使得文件和数据库设计人员可以结合访问效率、空间利用率和容错等因素来为数据库应用作出最适当的选择。

目前使用的数据库体系结构包括层次、网状、关系、面向对象和多维。层次和网状数据库系统主要应用在遗留系统中，而关系、面向对象和多维是新系统开发所使用的数据库体系结构。

对于多处理器的数据库服务器的介绍表明，它们将为数据库管理系统提供新的功能。其中

一个主要的新特性是可以将查询分解开来，并行地在表的数据段上运行查询。这种并行查询的处理能力可以极大地提高查询的处理速度。程序员通过提供DBMS执行数据操作的顺序的提示来提高数据库的性能，这些提示可以修正数据库内基于代价的优化器的结果。DBMS和程序员都可以通过对数据库统计数据的研究来确定查询的处理方法，本章中同时介绍了一些如何设计良好查询的建议。

本章总结全书对于数据库设计的讨论。在完成全部的物理数据库设计规格说明以后，读者在后面会开始使用数据库技术来实现数据库。实现意味着定义数据库，编写服务器和客户端程序以处理数据库上的查询、报表和事务。这是后面5章的主要内容，其中涉及到关系数据库在客户平台、服务器平台和客户/服务器环境的实现，以及数据仓库的技术。

本章复习

关键术语

位图索引	块因子	数据类型
非规范化	区	字段
文件组织	散列索引表	散列文件组织
散列算法	水平分割	索引
索引文件组织	联结索引	页
物理文件	物理记录	指针
廉价冗余磁盘阵列（RAID）	辅键	顺序文件组织
条	表空间	垂直分割

复习问题

1. 定义以下术语。

a. 文件组织	b. 顺序文件组织
c. 索引文件组织	d. 散列文件组织
e. 非规范化	f. 索引
g. 辅键	h. 数据类型
i. 位图索引	j. RAID
k. 联结索引	l. 条

2. 将术语和它的定义匹配起来。

_____位图索引	a. 一个I/O操作所读取的数据
_____散列算法	b. 一页中的记录数
_____页	c. 命名的二级存储区域
_____物理记录	d. 值为0或1的表
_____指针	e. 不包含业务数据的字段
_____块因子	f. 将键值转化为地址
_____物理文件	g. 邻接的字段

3. 比较以下的术语。

a. 水平分割；垂直分割	b. 物理文件；表空间
c. 物理记录；物理文件	d. 页；物理记录
e. 辅键；主键	

4. 物理数据库设计的主要输入是什么？

5. 物理数据库设计中的关键决策有哪些?
6. 在复合使用图中有哪些信息?
7. 当开发字段规格说明时，需要做出哪些决策?
8. 为字段选择数据类型的目标是什么?
9. 为什么字段值被编码或压缩?
10. 有哪些用于控制字段完整性的选项?
11. 描述3种处理缺失值的方法。
12. 说明为什么规范化的关系不一定是效率高的物理记录。
13. 列举3种适于通过非规范化来组成物理记录的情况。
14. 水平分割和物理分割的优缺点是什么?
15. 列举选择文件组织要考虑的7个重要因素。
16. 位图索引在什么情况下适用?
17. 散列索引表的优点是什么?
18. 数据存放在聚簇上的目的是什么?
19. 列举选取索引的7条经验。
20. 比较两种看待多维数据库的方法。
21. 如何使用EXPLAIN来编写高效率的查询?
22. 解释4个用来优化查询性能的选项。
23. 哪个层次的RAID最好? 为什么?

问题和练习

1. 考虑下面两个Millennium学院的关系:

```
STUDENT(Student_ID,Student_Name,Campus_Address,GPA)
REGISTRATION(Student_ID,Course_ID,Grade)
```

这是一个针对这些关系的典型查询:

```
SELECT STUDENT.STUDENT_ID, STUDENT_NAME,
COURSE_ID, GRADE
 FROM STUDENT, REGISTRATION
     WHERE STUDENT.STUDENT_ID = REGISTRATION.STUDENT_ID
     AND   GPA > 3.0
 ORDER BY STUDENT_NAME;
```

a. 应该在哪些属性上创建索引以提高查询效率? 说明选择这些属性的理由。
b. 为前面创建的索引写出SQL语句。

2. 参考图6-1中的复合使用图。在一段时间后，对于该图的假设发生以下变化:
a. 每个供货商大约有40个报价（而不是50个）。
b. 自行生产的零件只占到总数的30%，而购买的零件占到75%。
c. 每小时对于购买零件的直接访问增加到75次（而不是60次）。

画出新的使用图以体现数据变化。

3. 为图6-4中的规范化关系属性选择适当的Oracle数据类型。

4. 假设读者要为所在大学的学生记录的年龄字段设置默认值，那么会选择什么值? 为什么选定它? 在考虑到学生的其他特征（如不同的学院或不同的学位）时，默认值会如何变化?

5. 如果学生没有选定专业，校方通常会在专业字段输入“未决定”，此时“未决定”代表

的是默认值还是空值?

6. 考虑以下大型零售连锁店数据库中的规范化关系:

```
STORE (Store_ID, Region, Manager_ID, Square_Feet)
EMPLOYEE (Employee_ID, Where_Work, Employee_Name, Employee_Address)
DEPARTMENT (Department_ID, Manager_ID, Sales_Goal)
SCHEDULE (Department_ID, Employee_ID, Date)
```

当为这个数据库定义物理记录时，存在哪些非规范化这些关系的可能性？在何种情况下，你会考虑创建这种非规范化的记录?

7. 读者有一个硬盘文件，它用来存储最多1000条，每条长240字节的记录。假设一页的长度是4000字节，且记录不可以跨页。那么这个文件需要多少字节的存储空间?

8. 将关系垂直分割可能带来哪些问题？考虑到这些潜在的问题，哪些因素决定着何时进行垂直分割?

9. 是否可能在一个顺序文件上根据多个排列顺序来对数据进行顺序扫描？如果不可以，为什么？如果可以，应怎样操作?

10. 假设文件内的记录按照键序，使用指针和前面、后面的记录联结。这样每条记录的形式可能是:

主键、其他属性、指向前面记录的指针、指向后面记录的指针

a. 和顺序文件组织相比，这种方式有什么优点?

b. 和顺序文件组织相比，这种方式是否可以以多个顺序访问记录？为什么?

11. 假设在大学数据库中的学生文件上，分别在字段Student_ID（主键）以及Major、Age、Marital_Status和Home_Zipcode（都是辅键）上设置索引。如果学校要查询专业为MIS或计算机科学、年龄超过25岁且已婚的学生或是专业为计算机工程、单身且邮政编码是45462的学生，那么应该如何使用索引，使得用户只访问满足这些条件的记录?

12. 再次考虑上题中所提到的学生文件，哪个索引比较适合使用位图索引？要满足什么样的条件使用这个位图索引才有意义？按照图6-8的形式，选择可能创建位图索引的键，并画图。

13. 为图5-4中的表CUSTOMER和ORDER在字段Customer_ID上创建联结索引。

14. 考虑图6-7c，假设这个索引中叶的空行是用来存放新记录的，那么解释一下记录“Sooners”会放在哪里，记录“Flashes”又会放在哪里。当某个叶已经放满，而又有新的记录要加到它上面时会发生什么情况?

15. 考虑图6-12中对于流行的数据库体系结构的特征比较。给出和图中的多维数据模型相等价的关系。

16. 在文件已经加载记录后，还可以对文件进行聚簇操作吗？为什么?

17. 在本章介绍的并行查询处理中，一个查询在多个处理器上运行，每个处理器并行地访问数据库的不同子集。另一种本章中没有提及的并行查询处理形式是，分解查询以使查询的各个部分运行在不同的处理器上，同时允许每个部分访问任何它需要的数据。大多数的查询都包含子句来选择限定的记录。比如，限定子句的形式为:

```
(condition OR condition OR ···) AND (condition OR condition OR ···)AND ···
```

考虑这种一般化的形式，此时应该如何分解查询，以使并行处理器可以分别处理部分查询，并在每个部分的处理结束后再将它们结合起来?

18. 在图6-11f中，假设记录2、3、7和9被4个不同的用户同时更新，哪些记录可以并行更新？给出一个更新操作的调度以使并行最大化。

应用练习

1. 了解学校里提供给学生使用哪种DBMS，以及它们支持何种数据类型。依据对数据类型的支持情况比较不同的DBMS，并判断它们分别适用于哪些应用。

2. 使用本书提供的Web资源和其他因特网信息，调查不同数据库管理系统的并行处理功能。它们的功能有什么不同？

3. 使用本书提供的Web资源和因特网信息，了解面向对象数据库系统的功能。它们的价格是多少？供应商推荐在哪些应用领域使用它们的产品？

4. 和组织内熟悉的数据库设计人员或管理人员交流，了解他们所使用数据库系统支持的文件组织方法，询问他们在选择文件组织类型时要考虑哪些因素。对于索引文件，向他们请教是如何选定索引的，索引是否被删除过？为什么？

5. 咨询组织内熟悉的数据库设计人员或管理人员，他们是否在组织的数据库中使用RAID技术？说明使用或不使用的原因。如果已经使用，他们使用的是哪个层次的RAID？为什么？

参考文献

Babad, Y.M., and J.A.Hoffer. 1984. "Even No Data Has a Value." *Communications of the ACM* 27(August): 748-56.

Ballinger, C. 1998. "Introducing the Join Index." *Teradata Review* 1(Fall): 18-23. Also found at www.teradatareview.com/Fall98/Ballinger.html.

Bontempo, C.J., and C.M.Saracco. 1996. "Accelerating Indexed Searching." *Database Programming&Design* 9(July): 37-43.

Brobst, S, S.Gant, and F.Thompson. 1999. "Partitioning Very Large Database Tables with Oracle8." *Oracle Magazine* (March/April):123-26.

DeLoach, A. 1987. "The path to Writing Efficient Queries in SQL/DS." *Database Programming&Design* 1(January): 26-32.

Finkelstein, R.1988. "Breaking the Rules Has a Price." *Database Programming & Design* 1(June): 11-14.

Holmes, J. "More Paths to Better Performance." *Database Programming & Design* 9(February): 47-48.

Inmon, W.H. 1988. "What Price Normalization." *ComputerWorld* (October 17): 27, 31.

Kurka, A. "Rules for Better SELECTS." *SAP Technical Journal* 1(2): 75-79. Also available at www.saptechjournal.com.

Musciano, C. 1999. "0,1, 0+1...RAID Basics, Part 1." In *UnixInsider*, accessed via Web at www.unixinsider.com/unixinsideronline/swol-06-1999/swol-06-1999-raid1_p.html (address verified December 25, 2000).

Rogers, U. 1989. "Denormalization: Why, What, and How?" *Database Programming & Design* 2(December):46-53.

Schumacher, R. 1997. "Oracle Performance Strategies." *DBMS* 10(May): 89-93.

Shah, R. 1999. "Storage Beyond RAID." In *UnixInsider*, accessed via Web at www.unixinsider.com/unixinsideronline/swol-07-1999/swol-07-connectivity_p.html (address verified December 25, 2000)

Viehman, P.1994."24 Ways to Improve Database Performance." *Database Programming & Design* 7(February):32-41.

Yuhanna, N. 2000. *Oracle8i Database Administration*. Greenwich,CT:Manning Publications.

进一步阅读

Elmasri, R. and S.Navathe. 1994. *Fundamentals of Database Systems*, 2nd ed. Menlo Park, CA:Benjamin/Cummings.

Loney, K., E.Aronoff, and N.Sonawalla. 1996. "Big Tips for Big Tables." *Database Programming&Design* 9 (November): 58-62.

Roti, S. 1996. "Indexing and Access Mechanisms." *DBMS* 9(May): 65-70.

Web资源

- www.networkcomputing.com/605/605buyers.html *RAID Storage Solutions: Which is Right for you*? , J. Milne所著（网址于2000年12月25日验证）。
- www.raid-advisory.com RAID顾问。
- www.teradatareview.com NCR Teradata 数据仓库产品的杂志，包括关于数据库设计的文章。
- www.unixinsider.com/unixinsideronline/swol-07-1999/swol-07-1999-raid2_p.html *RAID Basics, Part 2:Moving on to RAID 3 and RAID 5*, C.Musciano所著（网址于2000年12月25日验证）。
- www.win2000mag.com/Articles/Index.cfm?ArticleID=218 RAID Levels，关于常见的RAID类型的简要描述（网址于2000年12月25日验证）。

项目案例：山景社区医院

在前面几章中，读者已经为山景社区医院开发了数据库系统。基于第3章、第4章、第5章中开发的概念和关系数据库，现在要为它进行相应的物理数据库设计。

项目描述

山景社区医院已经决定使用关系数据库管理系统Oracle来开发数据库，因此物理数据库的设计必须利用Oracle所提供的功能（如果读者对Oracle不熟悉，那么可以假定系统使用你熟悉的DBMS，然后回答下面的问题）。

项目问题

1) 除在第5章项目案例中获得的为第3章中概念数据库所开发的3NF关系以外，还需要哪些信息来进行相应的物理数据库设计？

2) 是否可能在数据库中使用水平和垂直分割？如果无法确定，那么还需要哪些信息才能确切地回答这个问题？

3) 图6-13给出部分山景社区医院数据库的初始复合使用图。在此以后，一些关于数据库使用情况的假设已经发生变化：

- 每个病人大约消费12个（而不是10个）项目。
- 每小时大约访问病人治疗效果数据40次（而不是30次），而且同时会访问相应的治疗办法。

重画图6-13以反映新的信息。

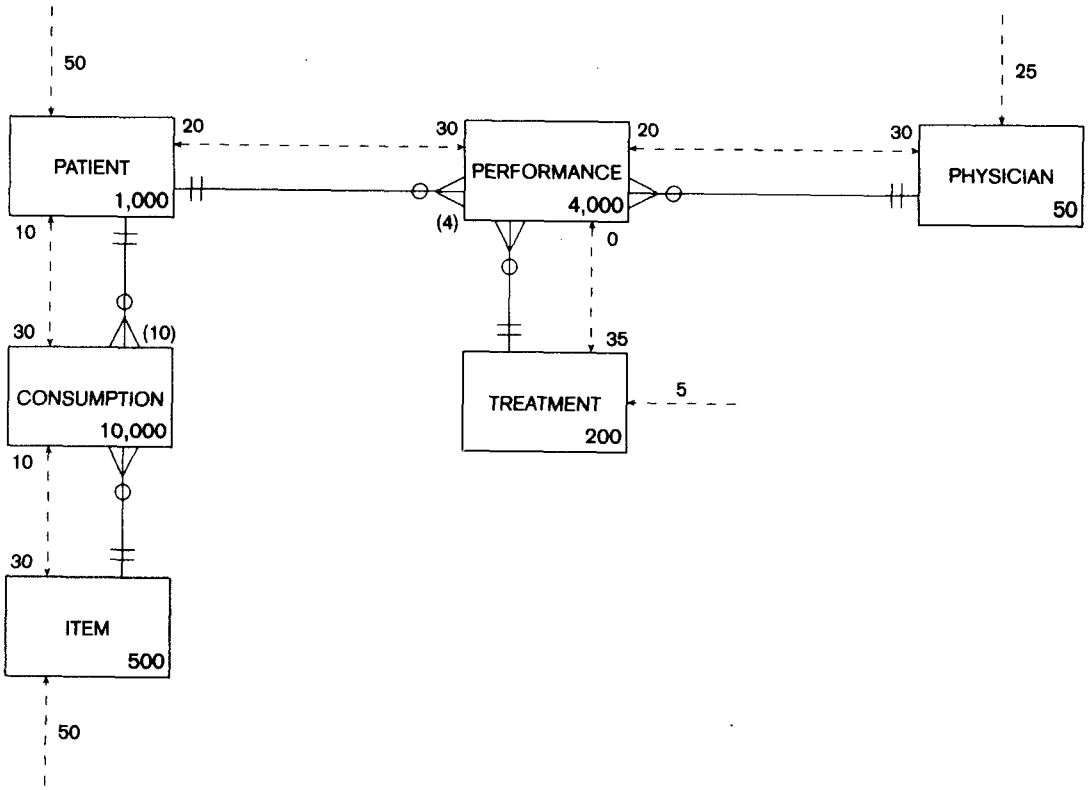


图6-13 部分山景社区医院数据库的复合使用图

项目练习

1) 在第5章的项目练习5中，读者已经为逻辑设计中的每个关系写出了相应的CREATE TABLE命令。在完全了解数据库管理系统（如Oracle）所提供的物理数据库设计选项后，重新考虑前面给出的CREATE TABLE语句，并回答以下问题:

a. 是否会为字段选择新的数据类型？为什么？

b. 是否有应该被编码的字段？如果有，你会使用何种编码方案？

c. 哪些字段可以取空值？

d. 假如进行治疗的日期没有输入，你会采取什么步骤来处理这些缺失的值？是否应该为这个字段提供默认值？为什么？

2) 在项目问题2中，询问读者是否要进行数据分割。除分割以外，是否有进行其他非规范化操作的可能性？如果没有，为什么？如果有，应在哪里和如何进行非规范化？

3) 在项目问题3中，读者已经更新了图6-13中的山景社区医院数据库的复合使用图。观察修正后的图，是否有对来自两个或多个表的行使用聚簇的可能性？为什么？

4) 为山景社区医院数据库的每个表写出创建主键索引的CREATE INDEX语句。

5) 考虑下面山景社区医院数据库上的查询语句：对过去两周的每次治疗效果数据，按照Treatment ID排序显示；若Treatment ID相同，按照日期反向排序，并列出执行治疗的医生（按照治疗分组）和这个医生在当天进行同样治疗的次数。为改善这个查询的性能，请读者建立辅键索引，并在需要时做出合理的假设。

第四部分 实 现

概要

第四部分讨论与实现关系数据库系统相关的主题，包括支持Web的因特网应用，以及数据仓库。正如第2章中所指出的，数据库实现包括：编写和测试数据库处理程序，完成数据库文档和培训材料，安装数据库和（如有必要）从以前的系统中转换数据。现在终于到了系统开发生命周期的关键点，我们一直在为该系统开发生命周期而做准备。我们前面所进行的活动，如企业建模、概念数据建模、逻辑与物理数据设计是必要的前期阶段。在系统实现的后期，我们期望得到满足用户信息需求的功能系统。此后，系统投入实际使用，为使系统正常工作，还要进行必要的系统维护。第四部分中的各章将有助于我们初步认识实现数据库系统的复杂性和挑战性。

第7章讨论SQL（Structured Query Language，结构化查询语言）。SQL已经成为创建和处理关系数据库的标准语言（特别是在数据库服务器中）。本章简要介绍了SQL的发展史，包括对目前大部分DBMS使用的SQL-92的全面介绍，还讨论了当前正在实现的SQL-99标准，以及SQL的语法。该章还介绍了创建数据库的数据定义语言（DDL），用于查询数据库的单个表的数据操纵语言（DML），动态视图（dynamic view）和物化视图（materialized view）的内容也在该章中加以介绍，它们用于约束用户的环境到相关的表上去，这些表对于完成用户工作来说很有必要。

第8章继续解释了更高级的SQL语法和结构，展示了多表查询、子查询和相关子查询，这些功能使SQL的功能更为强大。对事务完整性问题和数据同步构造的解释，将SQL放置在更为广阔的上下文中。其他的一些编程功能，包括触发器、存储过程和用在其他编程语言程序中的嵌入式SQL，更进一步地展示SQL的功能。SQL-99中联机分析处理（OLAP）的特性也在该章中加以介绍，它对数据仓库的访问很有必要。

第9章讨论了客户/服务器的体系结构、应用程序、中间件和在当今数据库环境中对客户机数据库的访问。理解这一章的内容很重要，因为它是理解第10章将要讨论的因特网主题的基础。该章还讨论了多层体系结构，包括应用服务器和数据库服务器，分布在各层的数据库处理选择件，使用浏览器的（瘦）客户。这一章中，支持Web的数据库的安全性以及ODBC和JDBC连接，这些内容为即将讨论的因特网主题奠定了基础。

第10章描述了基于Web的应用到数据库的连接。本章为那些不熟悉Web的人介绍了因特网方面的术语，脚本语言的使用，在脚本中使用嵌入式SQL的内容也在该章中加以介绍。该章包括了一个简单购物车应用程序，它用ASP和ColdFusion两种方法实现。安装和运行购物车的整个代码和文档都可以在本书的网站中找到。要想了解如何建立一个支持Web的数据库应用程序，就应该仔细研究这些代码。该章还介绍了Web服务器的作用，用于数据库连接的服务器端扩展，以及Web的安全性问题。

第11章描述了数据仓库的基本概念，为什么数据仓库被许多企业认为是在竞争取胜的关键，以及数据仓库中特有的数据库设计活动和结构。该章的主题包括：各种数据仓库体系结构，数

据转换和数据调和技术，以及用于数据仓库的多维数据模型（星型模式）。此外还解释和举例说明了用于数据集市的数据库设计，包括代理键、事实表粒度、日期和时间建模、维的一致性、无事实的事实表，以及帮助/层次/引用表。

正如前面对各章介绍所指出的，第四部分既提供了在实现数据库应用中出现的问题的概念理解，也介绍了建立一个数据库原型所必须的过程。我们对最新发展动向的介绍，如客户/服务器、支持Web、数据仓库等可使读者了解到数据库的未来发展方向。

第7章 SQL

7.1 学习目标

学完本章后，读者应该具备以下能力:

- 定义下列关键术语: 关系数据库管理系统（RDBMS）、目录、模式、数据定义语言、数据操纵语言、数据控制语言、基表、动态视图、物化视图、参照完整性、标量聚合以及向量聚合。
- 解释在数据库发展过程中SQL的历史和作用。
- 使用SQL数据定义语言定义数据库。
- 使用SQL命令查询单个表。
- 使用SQL建立参照完整性。
- 讨论SQL-92和SQL-99标准。

7.2 引言

有人把SQL读作“S-Q-L”，也有人把SQL读作“sequel”，但无论如何，SQL已成为创建和查询关系数据库事实上的标准语言。本章的主要目的是深入讨论SQL。SQL是关系系统最常用的语言，它已被美国国家标准化组织（American National Standards Institute，ANSI）接受成为美国标准，同时它也是联邦信息处理标准（Federal Information Processing Standard，FIPS），此外它还被国际标准化组织（ISO）认定为国际标准。

ANSI SQL标准在1986年首次发布，并在1989、1992（SQL-92）和1999（SQL-99）年进行了更新。SQL-92有三个层次: 入门级、中间级和完全级，而SQL-99在SQL-92的基础上进行了重大扩展。SQL-99建立了核心层一致性，核心层一致性必须先于其他一致性得到满足。目前，在SQL-99中指定了八种类型的一致性，包括活动数据库、增强的完整性管理和基本对象支持等。在本书写作之时，大部分数据库管理系统与入门级SQL-92相容，且正在朝着与核心SQL-99相容的目标而努力。例如，Oracle 8*i*支持入门级SQL-92，且基本与核心SQL-99兼容。除标记为Oracle SQL或Microsoft Access SQL，本章的例子都遵循SQL标准。

SQL已经在大型机和个人计算机系统上得到实现，因此本章内容与这两种计算环境有关。虽然许多PC数据库包使用一种按例查询（Query-By-Example，QBE）的接口，它们同时也把SQL作为可选项。例如，在Microsoft Access中，可以在这两个接口之间来回切换，按一下按钮，可以把使用QBE接口建立起来的查询用SQL的形式显示出来，这一特性可以帮助读者学习SQL语法。在客户/服务器体系结构中，SQL命令在服务器上执行，结果返回给客户工作站。

1979年面世的Oracle是第一个支持SQL的商用DBMS。Oracle现在可用于大型机、客户/服务器以及PC平台，适合于各种操作系统，包括OS/390(MVS)、各种UNIX操作系统、Linux、Microsoft Windows和Windows NT、VAX/VMS，以及VM/CMS。IBM的DB2、Informix以及Sybase也适用于这些操作系统。Microsoft SQL Server 2000运行于Windows NT和Windows 2000

上，个人版运行于Windows 98、Windows NT、Windows 2000、Windows Me上。

核心SQL-99包含入门级SQL-92和其他一些功能，因此供应商正在从入门级的SQL-92转入兼容核心SQL-99。已有几个文档发布了相关的标准，它们详细地说明了除了核心SQL-99之外，还可实现的增强级别，这使得SQL除了可用于当前所支持的关系数据库外，还可用于支持对象数据库中持久的、复杂的对象。本章我们使用Oracle 8*i* SQL*Plus，它与入门级SQL-92相容，且正在向核心SQL-99靠拢。在本书写作之时，SQL*Plus不与核心SQL-99相容，但Oracle已声称，它不久将支持核心SQL-99。

7.3 SQL标准的发展

1970年，E. F. Codd在一篇名为“A Relational Model of Data for Large Shared Data Banks”的经典论文中，首次明确提出了关系数据库技术的概念。在加利福尼亚圣约瑟的IBM研究实验室的工作人员承担了System R的开发工作，这个项目是论证在数据库管理系统中实现关系模型的可行性。他们使用了一种称为“Sequel”的语言，这种语言也是由圣约瑟的IBM研究实验室开发的。该项目从1974年开始至1979年结束，期间，“Sequel”被重新命名为SQL。后来，在该项目中所获取的知识被应用到SQL/DS的开发，这是IBM开发的第一个可商用的关系数据库管理系统。1981年，SQL/DS第一次实现，它运行在DOS/VSE操作系统上。在1982年推出了VM版，1983推出了MVS版DB2。

由于System R在所安装的用户结点那里受到好评，所以其他厂商开始开发使用SQL的关系产品。其中的一个产品，来自Relational Software的Oracle，实际上比SQL/DS先上市（1979年）。此外还有一些其他产品，包括来自Relational Technology的INGRES（1981年）、Britton-Lee的IDM（1982年）、Data General Corporation的DG/SQL，以及Sybase公司的Sybase（1986）。为了指导RDBMS开发，ANSI和ISO颁布了一种用于SQL关系查询语言（函数和语法）的标准，该标准常称为SQL/86，它最初是由数据库技术委员会X3H2提出的（数据库技术委员会X3H2, 1986; ISO, 1987）。到了1989年，1986年的标准进行了扩展，它包括一个可选的完整性增强特征（Integrity Enhancement Feature，IEF），人们常称之为SQL/89。同样在1989年，美国还颁布了一个相关的标准，称为数据库语言嵌入式SQL（Database Language Embedded SQL，ESQL）。ISO和ANSI委员会创建了式SQL-92（数据库技术委员会X3H2，1989；ISO，1989，1991），它对SQL/86作了更进一步的扩展。1992年底，SQL-92标准颁布，这就是著名的国际标准ISO/IEC 9075:1992，数据库语言SQL。1994年和1996年SQL-92进行了修正，1999年7月，SQL-99标准颁布。

目前有许多产品支持SQL，它们在从个人计算机到大型主机的各种机器上运行。数据库市场正走向成熟，产品进行重大变革的速度减慢，但它们将继续基于SQL。1999年，Oracle、IBM和Microsoft共占有全部数据库市场73%以上的份额（Gartner Dataquest发布，1999年5月3日）。其中，Oracle和IBM各自控制大约30%的市场。最近，电子商务应用和客户关系管理（Customer Relationship Management，CRM）系统的增长促进了一些规模较大的厂商的发展，但是Gartner Dataquest认为，在一些特定行业的系统和小的应用中，规模较小的厂商仍有发展壮大的机会。当你读这本书的时候，某些即将发布的产品或许会改变数据库管理系统的相对布局，但所有产品还将继续使用SQL，它们在一定程度上遵循后面描述的标准。

7.4 数据库体系结构中SQL的作用

目前，由于有了关系DBMS和应用程序生成器，应用程序用户普遍对数据库体系结构中

SQL的重要性不太清楚，许多用户在根本不了解SQL情况下访问数据库应用程序。例如，Web站点允许用户浏览正在访问的站点的目录（比如在http://www.llbean.com）。某个项目所呈现的信息，如大小、颜色、描述及可用性，都存储在数据库中。这些信息是使用SQL查询检索得来的，但用户此时不必发出SQL命令。

基于SQL的关系数据库应用程序涉及到用户的界面和带有SQL功能的关系数据库管理系统（Relational DBMS，RDBMS）。在RDBMS中，SQL常用于创建表、翻译用户要求、维护数据词典和系统目录、更新和维护表、建立安全机制以及执行备份和恢复过程。**关系DBMS**（RDBMS）是一个实现关系数据模型的数据管理系统，在关系数据模型中，数据存储在一系列表中，数据之间的联系用公共数值表示，而不是由链接表示。第3章的松谷家具数据库系统例子说明这种数据观点。本章SQL查询都将使用这个例子。

SQL标准的最初目的是：

1) 指定SQL数据定义和数据操纵语言的语法和语义。

2) 为设计、访问、维护、控制和保护SQL数据库，定义数据结构和最基本的操作。

3) 为在相容DBMS之间移植数据库定义和应用程序模块而提供相应工具。

4) 规定了最低（第一级）和完全（第二级）两个标准，允许在产品中采用不同的等级标准。

5) 提供了一个初始标准，尽管它不完全，但以后可以增强它，使之包括能够处理参照完整性、事务管理、用户定义函数、等值联结之外的联结运算符以及国家字符集（national character set）（包含在其他功能之中）的规格说明。

SQL-92标准已被广泛接受，大部分新产品支持SQL-92，且逐步支持核心SQL-99。但是，各个厂商的SQL版本都扩展了SQL-92的基本标准，包含了一些新的增强功能、特性和性能。例如，Oracle有一个DESCRIBE命令，它能列出表中所有属性、数据类型和约束。这个特性非常有用，特别适用于那些正在熟悉一个演示数据库的学生，但如果他们使用其他厂商的SQL，比如说MS-Access，那么通常找不到这条命令。因此，SQL标准是一个最小功能集，而不是一个完整的功能集。SQL标准有什么优缺点呢？

这样一个标准化的关系语言的优点如下：

- *减少培训费用* 一个企业可以针对一门语言进行培训。当雇佣新员工时，由于大批的IS专业人员已经接受过这门语言的培训，这样就减少了再培训的需要。
- *提高生产能力* IS专业人员通过长期使用可以深入学习SQL并精通SQL。企业可以投资购买新的工具，从而使IS专业人员的生产效率更高。由于程序员熟悉编写程序所使用的语言，所以他们能够更加迅速地维护现有程序。
- *应用可移植性* 如果每台机器都使用SQL，那么应用软件就能从一台机器移植到另一台机器上。此外，对计算机软件产业来说，当有一种标准语言时，开发现成的应用软件是最经济的。
- *应用持久性* 标准语言往往能长时间保留，因此，重写原有应用软件的压力会很小。并且，当标准语言增强或出现DBMS的新版本时，应用软件可以很容易地进行更新。
- *减少对某个厂商的依赖性* 如果使用一种公共语言，就可以很容易地使用不同厂商的DBMS、培训和教育服务、应用软件及咨询助理。此外，这些厂商的市场竞争会更加激烈，这有助于降低价格，提高服务质量。
- *跨系统通信* 在管理数据和处理用户程序方面，不同DBMS和应用程序可以更加容易地通信和协作。

但另一方面，标准可能会抑制创造力和革新。一种标准无法满足所有的需求，而且某种行

业标准是多方折中的结果，它不可能是最理想的。改变标准是不太容易的（因为许多厂商从中获利），因此改变这种固有的低效性要付出极大的努力。标准的另外一个缺点是，个别厂商在SQL中增加某些特征时，可能会导致丢失标准的一些优点，比如应用的可移植性。

最初的SQL标准遭到普遍的指责，特别是它缺乏参照完整性规则和特定的关系运算符。Date和Darwen（1997）指出，SQL看起来是在不遵照已建立的语言设计原则的情况下设计出来的，“结果，语言中充满了无数的约束，特别的结构，烦人的特殊规则”（p.8）。他们认为标准不够精确，标准SQL实现中的问题仍将继续存在。在本章中，我们可以清楚地看到部分缺陷。

7.5 SQL环境

图7-1是一个SQL环境的简化示意图，它与SQL-92标准是一致的。如图所示，SQL环境包括一个SQL数据库管理系统的实例，其中有DBMS访问的数据库及可以使用该DBMS访问数据库的用户和程序。每个数据库包含在**目录**（catalog）中，目录描述数据库中的任何对象，而不管是哪个用户创造的对象。图7-1有两个目录：DEV_C和PROD_C。大部分公司至少保存他们所使用数据库的两个版本。工作版本（production version，即图中的PROD_C）是正在使用的版本，它收集真实的业务数据，所以必须严密控制和监视这个版本。开发版本（development version，即图中的DEV_C）在数据库构建时使用，在应用到工作数据库之前，它一直作为开发工具、增强和维护的效果可以在该版本中进行彻底测试。通常，由于这样的数据库不包含真实的业务数据，所以该数据库没有被严密地控制或监视。每个数据库有与目录相关的命名的模式。**模式**（schema）是一系列相关的对象集，包括（但不仅限于）基表和视图、域、约束、字符集、触发器、角色等等。

如果多个用户在数据库中创建对象，合并所有用户模式的信息将会形成整个数据库的信息。每个目录也必须包含一个信息模式，其中包含目录中所有模式的描述、表、视图、属性、优先权、约束，以及域，同时还有其他与数据库相关的信息。目录中包含的信息由DBMS维护，它是由用户发出的SQL命令的结果，不要求用户采取有意识地创建命令的行动。句法上简单的SQL命令可引起由DBMS软件执行的复杂数据管理活动，这是SQL语言强大功能的一部分。用户通过使用SQL的选择语句可以浏览目录的内容。

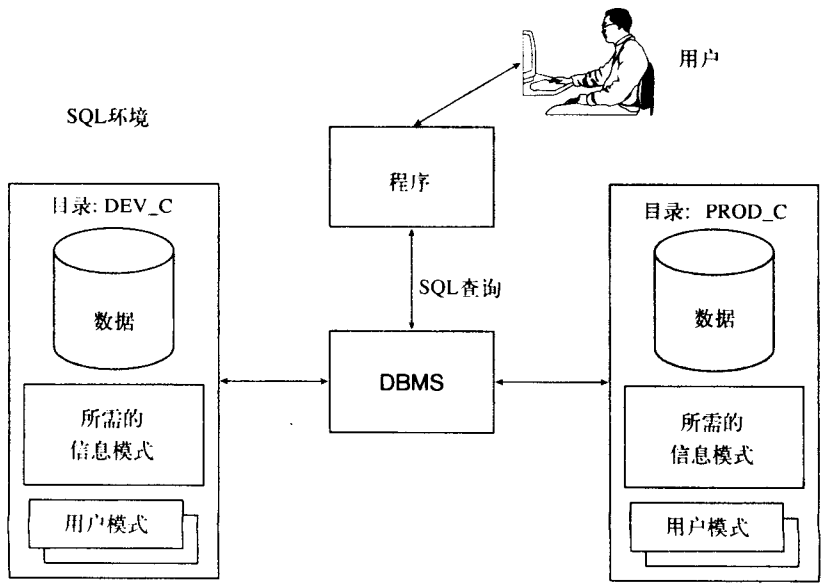


图7-1　SQL-92标准描述的一个典型SQL环境的简化示意图

SQL命令分为三种类型。第一种类型是**数据定义语言**（Data Definition Language，DDL）命令，这些命令用来创建、修改和删除表，本章将首先对此进行介绍。在工作数据库中，为了保护数据库结构不遭受意外修改，通常只能有一个或几个数据库管理员可以使用DDL命令。在开发过程中或在学生数据库中，可以给更多用户授予DDL权限。

```
SELECT [ALL DISTINCT] column_list
FROM table_list
[WHERE conditional expression]
[GROUP BY group_by_column_list]
[HAVING conditional expression]
[ORDER BY order_by_column_list]
```

图7-2 在数据操纵语言中使用SELECT语句的一般语法

第二种类型是**数据操纵语言**（Data Manipulation Language，DML）命令。许多人认为DML命令是SQL的核心命令，这些命令用来更新、插入、修改和查询数据库中的数据。这些命令可以交互地使用，从而在执行语句后，就能立即得到结果。命令也可以包含在用3GL(如C、COBOL）编写的程序中，嵌入式SQL命令可以使得程序员对报告产生的时机、界面外观、错误处理和数据安全性施加更多控制。本章大部分内容以交互的形式讲解基本DML命令。DML中使用的SQL SELECT命令的一般语法见图7-2。

最后一种类型是**数据控制语言**（Data Control Language，DCL）命令。这些命令有助于DBA控制数据库。它们包括这样一些的命令：授予和取消访问数据库或数据库中特定对象的权限，存储和删除对数据库产生影响的事务。

每个DBMS有一个它能处理的数据类型的定义列表。所有的DBMS都包含数字、字符串和日期/时间变量，有些DBMS还包含图形数据类型、空间数据类型和图像数据类型，这些数据类型极大地增加了数据操纵的灵活性。一旦创建了一个表，就必须指定每个属性的数据类型。选择某种数据类型要受需要存储的数据值和数据的预期使用方式的影响。单价应该以数字格式存储，因为通常会用单价进行一些数学操作，例如数量与单价的乘法操作。电话号码应该以字符串数据存储，特别是当这个数据集中包括有外线电话号码时更应如此。即使电话号码只包含数字，对电话号码进行相加或相乘这一类的数学操作也是毫无意义的。由于字符型数据的处理速度快，所以如果数字型数据不需要进行数学计算，也可作为字符型数据存储。选择日期字段而不是字符字段，开发人员就可以利用日期/时间间隔计算功能，而字符型字段则没有此功能。参见表7-1。

随着各种图形和图像数据类型出现，在决定如何存储数据时必须考虑业务需要。例如，颜色（沙滩色或米色）可作为一个描述性字符字段存储起来。但是，这种描述随着厂商的不同而不同，而且它不包含空间数据类型所能包含的信息量，空间数据类型可以包括精确的红绿蓝亮度值。目前，处理数据仓库的通用服务器上都有这些数据类型，RDBMS上也可出现这些数据类型。SQL-92和SQL-99还支持一些没有出现在表7-1中的几种数据类型，包括TIMESTAMP、REAL和INTERVAL，其他一些RDBMS还包括CURRENCY和LOGICAL等数据类型。为了充分利用RDBMS的所有功能，你必须熟悉所用到的每个RDBMS的可用数据字段。

表7-1 Oracle8数据类型示例

字符串型	CHAR(*n*)	定长字符数据，*n*个字符长度。最大长度为2000
	VARCHAR2(*n*)	变长字符数据。最大长度为4000字节
	LONG	变长字符数据。最大长度为4GB 每个表最多可有一个
数字型	NUMBER(*p*, *q*)	带符号十进制数字，共有*p*位数字，小数点后有*q*位
	INTEGER(*p*)	带符号整数，十进制或二进制，共*p*位数字
	FLOAT(*p*)	浮点数，用科学计数法表示，二进制数精度为*p*
日期/时间型	DATE	定长日期和时间数据，采用dd-mm-yy格式

我们准备举例说明SQL命令了。我们将使用的例子数据见图7-3。表名遵循如下命名规则：在每个表的名字后加下划线和字母t，如Order_t或Product_t。查看这些表时，请注意如下几点：

1）每份订单必须有一个有效的顾客号，该顾客号包含在Order_t表中。

2）订单中的每一项既要有一个有效的产品号，又要有Order_Line_t表所示的有效的订单号。

3）这四个表描绘了业务数据库系统中最常用关系集的一个简化版本，即客户对产品的订购。在第3章已经学习了必要的创建Customer表和Order表的SQL命令，这里将对这些SQL命令进行扩充。

Customer_t

Customer_ID	Customer_Name	Customer_Address	City	State	Postal_Code
1	Contemporary Casuals	1355 S Hines Blvd	Gainesville	FL	32601-
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094-
3	Home Furnishings	1900 Allard Ave	Albany	NY	12209
4	Eastern Furniture	1925 Beaver Rd	Carteret	NJ	07008-
5	Impressions	9485 Westcott St	Sacramento	CA	94088-
6	Furniture Gallery	325 Flatiron Dr	Boulder	CO	80514
7	Period Furniture	394 Rainbow Dr	Seattle	WA	97954
8	Cottage Classics	816 Peach Rd	Santa Clara	CA	96915-
9	M & H Casual Furniture	3709 First Street	Clearwater	FL	34620-
10	Seminole Interiors	2400 Rocky Point Dr	Seminole	FL	34646-
11	American Euro Lifestyles	2424 Missouri Ave N	Prospect Park	NJ	07508-
12	Battle Creek Furniture	345 Capitol Ave. SW	Battle Creek	MI	49015
13	Heritage Furnishings	66789 College Ave	Carlisle	PA	17013-
14	Kaneohe Homes	112 Kolowai St.	Kaneohe	HI	96744-
15	Mountain Scenes	4132 Main Street	Ogden	UT	84403-
(AutoNumber)					

Order_Line_t

Order_Id	Product_Id	Ordered_Quan
1001	1	1
1001	2	2
1001	4	1
1002	3	5
1003	3	3
1004	6	2
1004	8	2
1005	4	4
1006	4	1
1006	5	2
1006	7	3
1007	1	3
1007	2	2
1008	3	1
1008	8	3
1009	4	2
1009	7	3
1010	8	10
0	0	0

Order_t

Order_Id	Order_Date	Customer_ID
1001	10/21/2000	1
1002	10/21/2000	8
1003	10/22/2000	15
1004	10/22/2000	8
1005	10/24/2000	3
1006	10/24/2000	2
1007	10/27/2000	11
1008	10/30/2000	12
1009	11/1/2000	4
1010	11/1/2000	1
0		0

Product_t

Product_ID	Product_Description	Product_Finish	Standard_Price	Product_Line_Id
1	End Table	Cherry	$175.00	10001
2	Coffee Table	Natural Ash	$200.00	30001
3	Computer Desk	Natural Ash	$375.00	20001
4	Entertainment Center	Natural Maple	$800.00	30001
5	Writers Desk	Cherry	$325.00	10001
6	8-Drawer Dresser	White Ash	$750.00	20001
7	Dining Table	Natural Ash	$800.00	20001
8	Computer Desk	Walnut	$750.00	30001
(AutoNumber)			$0.00	

图7-3　松谷家具公司数据示例

本章的剩余部分将说明DDL、DML和DCL命令。图7-4概述了在数据库开发的整个过程中，不同类型的命令应在什么地方使用。我们使用下列符号举例说明SQL命令：

1）大写单词表示命令。尽管RDBMS不要求大写，但将命令大写有利于区分。

2）小写单词表示用户必须提供的值。

3）括号里的内容是可选语法。

4) 省略号（…）表示如有必要可以反复执行伴随的子句。

5) 每个SQL命令以分号（；）结束。在交互方式中，当用户按下RETURN键时，SQL命令就会执行。小心使用一些其他的习惯，如键入GO，或者在命令中每一行末使用连续符号，如连字符。为了方便阅读，本书采用间隔和缩排格式，这在标准SQL语法中不是必须的。

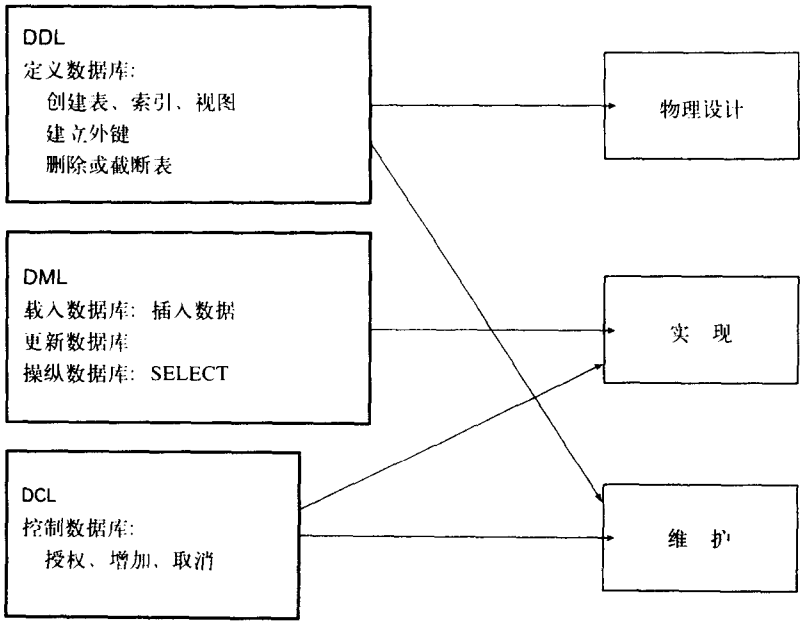


图7-4 DDL、DML、DCL和数据库开发过程

7.6 用SQL定义数据库

创建数据库时，应为大部分系统分配存储空间以包括基表、视图、约束、索引和其他数据库对象。正因为如此，只有数据库管理员才有权限创建数据库，你也可以申请创建数据库。可以为大学中的学生分配一个账号，让他们访问已有数据库，或者允许他们创建自己的数据库。在任何一种情况下，创建数据库的基本语法是:

```
CREATE SCHEMA database_name;
AUTHORIZATION owner_user id;
```

虽然其他指定用户也可以使用该数据库，甚至可以转移数据库的所有权，但数据库归授权用户拥有。数据库的物理存储由硬件和软件环境决定，通常是系统管理员所关心的方面。数据库管理员能够对物理存储施加的控制量是由所使用的RDBMS决定的。当使用MS Access 2000时，系统管理员几乎不能对其进行控制；而Oracle 8*i* 数据库管理员可以对数据布局、控制文件、索引文件等方面施加较多控制，从而提高调节数据库，使其更有效完成任务的能力。

7.6.1 SQL数据库定义

入门级SQL-92包括三个SQL DDL CREATE命令:

- CREATE SCHEMA 用于定义特定用户拥有的数据库部分。模式依赖于目录，它包括基表、视图、域、约束、断言、字符集、校验等模式对象。
- CREATE TABLE 定义一个新表和它的列。这个表可以是基表也可以是导出表。表依赖

于模式。导出表是通过执行用到一个或多个表或视图的查询而创建的。

- CREATE VIEW　从一个或多个表或视图定义一个逻辑表。视图无法被索引。通过视图更新数据有一定的限制。在视图可以被更新的地方，产生的变化会影响到底层的基表——这些基表最初用来创建视图。

上面介绍的每个CREATE命令都可以通过 DROP命令来撤销。因此，DROP TABLE将会删除一张表，包括与之相关的定义、内容、任何约束、视图和索引。通常只有表的创建者可以删除表。DROP SCHEMA和DROP VIEW也可以删除命名的模式和视图。ALTER TABLE 可以改变一个现有基表的定义，它通过增加、删除、改变列或者删除约束来改变基表的定义。

SQL-92标准还有其他五个CREATE命令。它们都不要求满足SQL-92的入门级或中间级标准。

- CREATE CHARACTER SET　允许用户为文本字符串定义字符集，因为可以使用非英语语言，有助于SQL的全球化。每个字符集包含一组字符、内部表示每个字符的方法、这种表示的数据格式以及校验或对字符集分类的方法。
- CREATE COLLATION　一个命名的模式对象，它指定了字符集呈现的顺序。通过操纵现有的校验可以创建新的校验。
- CREATE TRANSLATION　一个命名规则集，为了进行变换或转换，它将字符从源字符集映射到目标字符集。
- CREATE ASSERTION　一个模式对象，它建立CHECK约束，当该约束为假时是非法的。
- CREATE DOMAIN　一个建立了域的模式对象，或属性上的有效值集。数据类型可以指定，如果需要的话，也可以指定一个默认值、校验或其他约束。

7.6.2 创建表

一旦设计好数据模型并进行规范化后，就可以使用SQL CREATE TABLE命令定义各表中所需的列。CREATE TABLE的语法如图7-5所示。以下是创建表时要遵循的一系列步骤。

1) 为每个属性确定合适的数据类型，包括长度、精度和刻度（如有必要）。

2) 指定列能否接受空值，这在第6章中讨论过。当创建表时，同时也创建了列不能为空值的列控制，在输入数据时，对表的每次更新都会强制执行列控制。

3) 确定必须惟一的列。当为某个列建立了UNIQUE的列控制时，对于表中每一行数据，此列的数据必为不同的值（也就是说，没有重复的值）。如果一列或多列指定为UNIQUE，那么该列或多列就是一个候选键（在第5章讨论过）。尽管每个基表可以有多个候选键，但只能有一个候选键可以指定为PRIMARY KEY。当某列被指定为PRIMARY KEY时，即使没有明确说明该列为NOT NULL，系统也会认为它是NOT NULL，UNIQUE和PRIMARY KEY均为列约束。

4) 确定所有的主键-外键对（第5章讨论过）。在创建表时可立即建立外键，或在以后通过修改表来建立外键。对于有父子关系的表，首先应创建父表，这样在创建子表时，它能引用已存在的父表。列约束REFERENCES可用来执行参照完整性。

5) 确定要插入到所有列中的值，可以为各列指定默认值。在SQL-92中，DEFALUT可以用于定义默认值，在输入数据时，若没有数据输入，就将默认值自动地插入表中。在图7-6中，创建ORDER_T表的命令已为DATE属性定义了SYSDATE的默认值（Oracle中当前日期的名称）。

6) 确定所有这样的列，在其上可以声明域规格说明，使得这些列上的约束比由数据类型所建立的列上的约束更多。在SQL-92中使用CHECK作为列约束，对于要插入到数据库中的值，可以为其建立有效性规则。在图7-6中，创建PRODUCT_T表时包括一个CHECK约束，它列举了PRODUCT_FINISH的各种可能值。因此，即使“White Maple”项满足变长字符数据类型约束，但由于“White Maple”不在检查列表中依然会遭到拒绝。

7) 使用CREATE TABLE和CREATE INDEX语句创建表和所有需要的索引（索引是用来提高查询性能的，所以CREATE INDEX命令不是SQL-92标准的一部分，但大多数RDBMS都使用它）。

```
CREATE TABLE tablename
( {column definition  [table constraint] } . . .
[ON COMMIT {DELETE | PRESERVE} ROWS] );

where column definition ::=
column_name
        {domain name | datatype [(size)] }
        [column_constraint_clause . . .]
        [default value]
        [collate clause]

and table constraint ::=
        [CONSTRAINT constraint_name]
        Constraint_type [constraint_attributes]
```

图7-5 数据定义语言中CREATE TABLE语句的一般语法

在第3章和第5章中，我们展示了为松谷家具公司创建Customer表和Order表的SQL数据库定义命令，在CREATE TABLE命令中包含了列约束。在图7-6中，我们展示了使用Oracle 8*i*的数据库定义命令，它包括了其他的列约束，并且为主键和外键命名。例如，Customer表的主键是CUSTOMER_ID，主键约束命名为CUSTOMER_PK。现以Oracle为例，当查看DBA_CONSTRAINTS表时，数据库管理员可以很容易确定Customer表中的主键约束，因为它的名字CUSTOMER_PK是Costraint_Name列的值。如果没有约束名，就会自动分配一个系统标识符，该标识符很难去阅读。

```
CREATE TABLE CUSTOMER_T
            (CUSTOMER_ID                NUMBER(11, 0) NOT NULL,
             CUSTOMER_NAME              VARCHAR2(25) NOT NULL,
             CUSTOMER_ADDRESS           VARCHAR2(30),
             CITY                       VARCHAR2(20),
             STATE                      VARCHAR2(2),
             POSTAL_CODE                VARCHAR2(9),
CONSTRAINT CUSTOMER_PK PRIMARY KEY (CUSTOMER_ID));

CREATE TABLE ORDER_T
            (ORDER_ID                   NUMBER(11, 0) NOT NULL,
             ORDER_DATE                 DATE          DEFAULT SYSDATE,
             CUSTOMER_ID                NUMBER(11, 0),
CONSTRAINT ORDER_PK PRIMARY KEY (ORDER_ID),
CONSTRAINT ORDER_FK FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER_T(CUSTOMER_ID));

CREATE TABLE PRODUCT_T
            (PRODUCT_ID                 INTEGER       NOT NULL,
             PRODUCT_DESCRIPTION        VARCHAR2(50),
             PRODUCT_FINISH             VARCHAR2(20)
                        CHECK (PRODUCT_FINISH IN ('Cherry', 'Natural Ash', 'White Ash',
                          'Red Oak', 'Natural Oak', 'Walnut')),
             STANDARD_PRICE             DECIMAL(6,2),
             PRODUCT_LINE_ID            INTEGER,
CONSTRAINT PRODUCT_PK PRIMARY KEY (PRODUCT_ID));

CREATE TABLE ORDER_LINE_T
            (ORDER_ID                   NUMBER(11,0) NOT NULL,
             PRODUCT_ID                 NUMBER(11,0) NOT NULL,
             ORDERED_QUANTITY           NUMBER(11,0),
CONSTRAINT ORDER_LINE_PK PRIMARY KEY (ORDER_ID, PRODUCT_ID),
CONSTRAINT ORDER_LINE_FK1 FOREIGN KEY(ORDER_ID) REFERENCES ORDER_T(ORDER_ID),
CONSTRAINT ORDER_LINE_FK2 FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT_T(PRODUCT_ID));
```

图7-6 松谷家具公司的SQL数据库定义命令（Oracle 8*i*）

7.6.3 使用和定义视图

图7-6中所显示的SQL语法，说明了在一个数据库模式中如何创建四张**基表**（base table）。

这些表用于数据库中数据的物理存储，对应于概念模式中的实体。使用SQL查询，可以创建虚拟表，即**动态视图**（dynamic view），当引用它们时，它们的内容就被物化。这些视图可以经常利用SQL SELECT查询，用与处理基表相同的方式加以处理。也可以使用**物化视图**（materialized view），它存储在物理磁盘上，在适当间隔内或事件发生时刷新。

视图的作用是简化查询命令，但它也能够为数据库提供有效的数据安全保护，显著地提高程序的生产率。为了着重说明视图的方便之处，考察图1-6中的松谷家具公司发货单。构造发货单需要访问图7-3中松谷家具公司数据库的四张表：CUSTOMER_T、ORDER_T、ORDER_LINE_T和PRODUCT_T。一个数据库的新手在构造这样的多表查询时，也许会出错，也许效率会很低。视图可以使我们预先将这种联系定义到单个虚拟表中，它是数据库的一部分。有了这个视图，只需要顾客发货单数据的用户就不必重建表之间的联系来产生报表或报表的任何子集。表7-2总结了使用视图的优缺点。

视图INVOICE_V通过指定的一条SQL查询（SELECT-FROM-WHERE）来定义，查询的结果就产生了视图。

查询1　为一个客户创建一张发货单需要哪些数据元素？以INVOICE_V为名将此查询保存为视图。

```
CREATE VIEW INVOICE_V AS
  SELECT CUSTOMER_T.CUSTOMER_ID,CUSTOMER_ADDRESS,
  ORDER_T.ORDER_ID,PRODUCT_T.PRODUCT_ID,ORDERED_QUANTITY,
  /*和其他要求的列*/
  FROM CUSTOMER_T, ORDER_T, ORDER_LINE_T, PRODUCT_T
  WHERE CUSTOMER_T.CUSTOMER_ID= ORDER_T. CUSTOMER_ID
        AND ORDER_T.ORDER_ID= ORDER_LINE_T. ORDER_ID
        AND PRODUCT_T.PRODUCT_ID= ORDER_LINE_T. PRODUCT_ID;
```

表7-2　使用动态视图的优缺点

优　点	缺　点
简化查询命令	每次当视图被引用时，重新创建视图会占用处理时间
有助于提高数据安全性和机密性	不一定能直接更新
提高程序员的生产率	
包含大多数当前基表的数据	
占用很少的存储空间	
为用户提供定制的视图	
建立了数据的物理独立性	

SELECT子句指定（或投影）视图表中包含的数据元素（列），FROM子句显示了构建视图时需要的表和视图，WHERE子句指定用来联结CUSTOMER_T、ORDER_T、ORDER_LINE_T和PRODUCT_T时的公共列名。由于视图是一张表，表中有这样一条关系的性质，即行的顺序并不重要，所以视图中的行可以不经过排序。然而，引用视图的查询可按任何指定的顺序显示结果。

通过建立一个订单号为1004订货单的查询，我们可以看到视图的强大功能。不必指定四个表的联结，查询能从视图INVOICE_V中找到所有相关数据元素。

查询2　创建订单号为1004订货单需要哪些必要的数据？

```
SELECT CUSTOMER_ID,CUSTOMER_ADDRESS,PRODUCT_ID,QUANTITY(···其他需要的列)
FROM INVOICE_V
```

```
WHERE ORDER_ID=1004;
```

动态视图是一张虚拟的表，必要时可由DBMS自动创建，而且不必像维护真实数据那样维护它。任何SQL SELECT语句都可以用来创建视图。真实数据存储在基表中，这些表已由CREATE TABLE命令定义。动态视图总是包含最近导出的数据值，所以就最新数据来说，动态视图优于通过几个基表临时建立起来的真实表。而且与临时的真实表相比，视图只占用极小的存储空间。然而，每次访问视图内容时，都必须计算这些内容，因而视图的代价很大。现在可以使用物化视图来解决这方面的不足。

视图可以把多个表或视图联结起来，同时可以包含导出（或虚拟）列。例如，如果松谷家具公司数据库的一个用户只想知道每件家具产品所有订单的总值，那么可以在Oracle SQL*Plus中利用INVOICE_V创建这个视图。

查询3 每件家具产品所有订单的总值是多少?

```
CREATE VIEW ORDER_TOTALS_V AS
SELECT PRODUCT_ID PRODUCT,SUM(UNIT_PRICE*QUANTITY) TOTAL
  FROM INVOICE_V
       GROUP BY PRODUCT_ID;
```

我们可为视图中的列指定与基表或表达式中的列不同的名字（别名）。在本例中，PRODUCT_ID重命名为PRODUCT，该别名仅局限此视图。TOTAL是各产品销售总值表达式的列名。有了这个视图后，随后的查询就能把该表达式当作列一样引用，而不是把它当成一个导出的表达式引用。基于其他视图来定义视图可能会产生一些问题。例如，如果我们重新定义INVOICE_V，其中不包括UNIT_PRICE，那么由于ORDER_TOTAL_V找不到单价，它将不再有效。

视图有助于保证安全性。用户看不到视图中没有包括的表和列。同时，利用GRANT和REVOKE语句限制对视图访问，又增加了一层安全性。例如，可以授权一些用户访问视图中的聚合数据，如平均数，而不允许他们访问基表和详细数据，这样就使得他们无法看到基表中的数据。第12章将进一步解释SQL安全命令。

通过创建视图，限制用户只能使用他完成任务所必须的数据，就能够实现数据的私有性和机密性。如果一个文员需要知道某个员工的地址，但又不能让他知道员工的赔偿率，可以让办事员访问一个不包含赔偿信息的视图。

有些人提倡为每个基表建立视图，即使视图与基表完全相同也是如此。他们之所以这样建议，是因为随着数据库的演化，视图有利于提高编程效率。考虑这样一种情况，有50个程序都使用CUSTOMER_T表。假设进一步补充松谷家具公司的数据库以支持新功能，要求将CUSTOMER_T表重新规范化为两张表。如果这50个程序都使用这张基表上的视图，那么只需重新创建这个视图，这样节约了大量的重新编程的工作。然而，由于每次引用视图时，都要重新创建该视图的虚拟表，所以动态视图需要大量的运行时计算机处理。因此与直接引用基表相比，通过视图引用基表增加了大量的查询处理时间。必须在这种额外操作成本与利用视图避免的潜在重新编程之间进行平衡。

在本章后面部分，我们将创建和使用动态视图，动态视图复制了松谷家具公司的表，用来举例说明SQL语法。下列SQL语句建立了基表的视图:

```
CREATE VIEW CUSTOMER_V AS SELECT * FROM CUSTOMER_T;
CREATE VIEW ORDER_V AS SELECT * FROM ORDER_T;
CREATE VIEW ORDER_LINE_V AS SELECT * FROM ORDER_LINE _T;
CREATE VIEW PRODUCT_V AS SELECT * FROM PRODUCT_T;
```

在一定的限制下，可以直接从视图而不从基表中更新数据是可能的。一般说来，就基表的数据修改来说，只要更新操作是明确的，就允许在视图中对数据进行更新。但如果CREATE VIEW语句包含下列任何情况，视图将不能直接更新。

1) SELECT子句包括关键字DISTINCT（本章后面7.9节中将详细讨论SELECT子句）。

2) SELECT子句包含导出列、聚合、统计函数等的表达式。

3) FROM子句、子查询或者UNION子句引用多个表。

4) FROM子句、子查询引用一个不能更新的其他视图。

5) CREATE VIEW命令包含GROUP BY或者HAVING子句。

可能会发生这样情况，对一个实例更新将导致这个实例从视图中消失。我们创建一个名为EXPENSIVE_STUFF_V视图，视图显示了所有UNIT_PRICE超过$300的家具产品。该视图包含一张PRODUCT_ID是 5的写字台，单价为$325。如果我们通过EXPENSIVE_STUFF_V更新数据，把写字台的单价减至$295，那么由于写字台的单价现在低于$300，所以它不再在EXPENSIVE_STUFF_V虚拟表中出现。如果需要跟踪原来价格超过$300的所有商品，就在CREATE VIEW命令的SELECT子句后增加WITH CHECK OPTION子句。如果UPDATE和INSERT语句使更新或插入的行从视图中删除，那么WITH CHECK OPTION子句将拒绝这样的语句。这个选项仅用于能更新的视图。

下面是用于创建EXPENSIVE_STUFF_V的CREATE VIEW语句。

查询4　显示所有UNIT_PRICE曾经超过$300的家具产品。

```
CREATE VIEW EXPENSIVE_STUFF_V
  AS
  SELECT PRODUCT_ID,PRODUCT_NAME,UNIT_PRICE
          FROM PRODUCT_T
                WHERE UNIT_PRICE>300
                   WITH CHECK OPTION;
```

当试图将写字台的单价更新为$295时，可以使用如下Oracle SQL*Plus语法：

```
UPDATE EXPENSIVE_STUFF_V
  SET UNIT_PRICE=295
      WHERE PRODUCT_ID=5;
```

Oracle将显示出错信息：

```
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation
```

写字台价格提高到$350将会有效，而不会出现出错信息，这是因为视图是可更新的，并且没有违背视图中指定的条件。

视图信息存储在DBMS的系统表中。例如，在Oracle8中，所有视图的文本存储在DBA_VIEWS中，拥有系统权限的用户能找到这条信息。

查询5　EXPENSIVE_STUFF视图有哪些可用的信息？

```
SELECT * FROM DBA_VIEWS WHERE VIEW_NAME='EXPENSIVE_STUFF_V';
```

结果：

```
OWNER            VIEW_NAME            TEXT_LENGTH          TEXT
MPRESCOTT        EXPENSIVE_STUFF_V    103

SELECT PRODUCT_ID, PRODUCT_NAME, UNIT_PRICE
       FROM PRODUCT_T
```

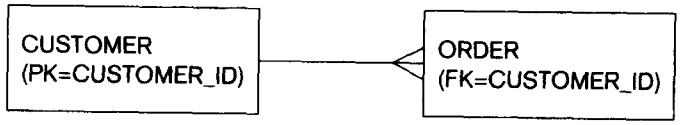
```
WHERE UNIT_PRICE>300
```

物化视图 像动态视图一样，可以用不同方式建立物化视图以满足不同的要求。表可以整体或部分地被复制，可以在预定的时间间隔被刷新，或者在需要访问表时被触发。物化视图建立在对一个或多个表查询的基础之上。在Oracle8*i*中，可以创建基于数据聚合的概要表。使用分布式数据的远程数据拷贝可以以物化视图的形式在本地存储。保持本地视图与远程基表或数据仓库同步将会带来维护开销，但使用物化视图可以提高分布式查询的性能，特别是当物化视图中数据相对来说是静态的，不需要经常被刷新。

7.6.4 创建数据完整性控制

在图7-6中，我们已经看到创建外键的语法。为了在关系数据模型中关系是1：M的两个表之间建立参照完整性，关系中“1”端的表的主键可以被“M”端的表的一列引用。参照完整性意味着“M”端的匹配列中的某个值必须对应于“1”端的表中某个行的主键值，或者为NULL。如果外键的值在它所引用的主键列中不是一个已存在的有效值，SQL REFERENCES子句将不允许增加这样一个值。尽管存在这一条限制，但还是存在其他完整性问题。

如果CUSTOMER_ID的值改变，那么该顾客与该顾客所下订单之间的联系就会被破坏。REFERENCES子句可以防止对外键的值做这样的改变，但没法防止对主键的值做这样的改变。该问题可以通过这样声明来解决：一旦创建了主键的值，就不能再改变它。在这种情形下，在大多数系统中，通过包含ON UPDATE RESTRICT子句，就可以处理对Customer表的更新。这样，试图删除或改变主键值的任何更新都被拒绝，除非在任何子表中都不存在外键引用该值。图7-7中列出了与更新相关的语法。



受限更新：表CUSTOMER中的顾客ID只有在表ORDER中不存在时才可以被删除

```
CREATE TABLE CUSTOMER_T
            (CUSTOMER_ID        INTEGER DEFAULT 'C999'  NOT NULL,
             CUSTOMER_NAME      VARCHAR(40)             NOT NULL,
             ...
CONSTRAINT CUSTOMER_PK PRIMARY KEY (CUSTOMER_ID),
ON UPDATE RESTRICT);
```

级联更新：当表CUSTOMER中的顾客ID改变时，表ORDER中的对应ID也会发生相应改变

```
... ON UPDATE CASCADE);
```

空值更新：当表CUSTOMER中的顾客ID改变时，表ORDER中对应的原ID都设为空值

```
... ON UPDATE SET NULL);
```

默认值更新：当表CUSTOMER中的顾客ID改变时，表ORDER中对应的原ID都设为预先定义的一个默认值

```
... ON UPDATE SET DEFAULT);
```

图7-7 在更新时保证数据完整性

另一种解决方法是通过使用ON UPDATE CASCADE选项，把这种改变传递到子表。如果一个顾客ID号改变，这种改变将级联地传递到子表ORDER_T，并且ORDER_T表中的顾客ID也将更新。

第三种解决方法是允许更新CUSTOMER_T，但要使用ON UPDATE SET NULL选项，将ORDER_T表中相关的CUSTOMER_ID值置为NULL。在这种情形下，使用SET NULL选项将导致订单和顾客之间的联系丢失，这不是我们需要的结果。在这几种方法中，CASCADE是使用最灵活的选项。如果要删除一条顾客记录，可以使用ON DELETE RESTRICT、CASCADE或者SET NULL。使用DELETE RESTRICT，如果在ORDER_T表中有顾客的订单，那么不能删除这条顾客记录；使用DELETE CASCADE，如果删除一个顾客，将会删除ORDRE_T中与这个顾客相关的所有订单记录；使用DELETE SET NULL，在某条顾客记录删除之前，该顾客的订单记录就会置为空值；使用DELETE SET DEFAULT时，在某条顾客记录删除之前，这个顾客的订单记录就会置为一个默认值。DELETE RESTRICT可能用得最为普遍。不是所有的SQL RDBMS都提供主键的参照完整性，在这种情况下，对主键列的更新和删除就不能撤销。

7.6.5 修改表的定义

可以通过修改列的规格说明来修改表的定义。使用ALTER TABLE命令，可以在一张已经存在的表中增加新的列。例如，在CUSTOMER表中增加一个顾客类型的列。

命令　在CUSTOMER表增加一个顾客类型列。

```
ALTER TABLE CUSTOMER_T
  ADD (TYPE VARCHAR(2));
```

ALTER TABLE命令可以包括像ADD、DROP和ALTER之类的关键字，允许修改列名、数据类型、长度和约束。通常来说，当增加新列时，为了便于处理表中已有的数据，它的状态置为NULL。当创建新的列时，表中所有的实例都增加这一列，且将它的值置为NULL是最合理的。由于在使用数据库过程中常常出现需求改变、使用原型、开发方法改进、出现错误等情况，所以为使数据库适应这些不可避免的修改，ALTER命令的引入意义重大。ALTER命令不能用来修改视图。

7.6.6 删除表

表的所有者可以使用DROP TABLE命令从数据库中删除表，通过使用类似的DROP VIEW命令可以删除视图。

命令　从数据库模式中删除一张表。

```
DROP TABLE CUSTOMER_T;
```

该命令将会删除表，并且把即将发生的任何变化保存到数据库中。某人要想删除一张表，他必须拥有这张表，或者被授予DROP TABLE系统权限。删除表将会导致删除相关索引和所授予的相关权限。DROP TABLE命令可以用关键字RESTRICT或CASCADE限定。如果指定了RESTRICT，那么当存在任何当前引用了这张表的依赖对象（如视图、约束）时，该命令会失败，不能删除这张表。当指定CASCADE时，如果删除这张表，则该表的所有依赖对象都将被删除。在Oracle中，可以利用TRUNCATE TABLE命令删除向表中输入的所有数据，但保留表的结构。

7.7 插入、更新和删除数据

一旦创建了表和视图，在编写查询语句之前，必须要向表和视图中输入数据并维护这些数

据。用于填充表的SQL命令是INSERT。如果为表中的每一列输入一个值，可以使用下列命令，它用来给CUSTOMER_T表增加第一行数据。注意，数据值的顺序必须与表中列的顺序一致。

命令 在表中插入一行数据时，为每个属性插入一个值。

```
INSERT INTO CUSTOMER_T VALUES
(001, 'Contemporary Casuals', '1355 S.Himes Blvd.', 'Gainesville', 'FL', 32601);
```

当数据并没有输入到表中的每一列时，可以将空字段置为NULL，要么指定要增加数据的列。在INSERT命令中，数据值的顺序也必须与指定的列的顺序一致。例如，下列语句可用于往PRODUCT_T表中插入一行数据，因为表中不存在产品描述信息。

命令 向表中插入一行数据，其中有一些属性为NULL。

```
INSERT INTO PRODUCT_T(PRODUCT_ID, PRODUCT_DESCRIPTION,
PRODUCT_FINISH, STANDARD_PRICE, PRODUCT_ON_HAND)
VALUES(1, 'End Table', 'Cherry', 175, 8);
```

一般来说，INSERT命令根据语句中提供的值插入一个新行到表中，或者从其他数据库中拷贝一行或多行数据到表中，或者从一个表中提取数据插入到另一个表中。当想要增加数据到CA_CUSTOMER_T表时，可以使用下面的INSERT命令。CA_CUSTOMER_T表与CUSTOMER_T表具有相同结构，但仅只含有松谷家具公司在加利福尼亚州的顾客。

命令 用另一个具有相同结构的表的子集给一个表增加数据。

```
INSERT INTO CA_CUSTOMER_T
  SELECT * FROM CUSTOMER_T
     WHERE STATE='CA';
```

INSERT命令中定义的表可以是一个视图，但是视图必须是可以更新的，这样，插入到视图中的数据也能够插入到基于这些视图的基表中。如果在视图的定义中包括了WITH CHECK OPTION，那么当数据值不能满足WITH CHECK OPTION的规格说明时，试图通过视图来插入数据的要求会被拒绝。

7.7.1 批量输入

INSERT命令用于一次输入一行数据，也可以用于插入作为查询结果的多行数据。某些SQL版本有专用命令或实用程序来批量输入多行数据，即INPUT命令。Oracle中包含SQL*Loader程序，它从命令行运行，能够把一个文件的数据装载到数据库中。使用这个受人欢迎的程序需要点技巧，它不在本书的讨论范围之内。

7.7.2 删除数据库内容

可以逐行删除数据也可以按组删除行。假设松谷家具公司决定不再与夏威夷州的顾客做生意，那么可以通过下面的命令删除CUSTOMER_T表中地址为Hawaii的顾客的所有行。

命令 从CUSTOMER表中删除满足一定条件的行。

```
DELETE FROM CUSTOMER_T
  WHERE STATE='HI';
```

最简单的DELETE形式用于删除表中所有的行。

命令 删除CUSTOMER表中所有的行。

```
DELETE FROM CUSTOMER_T;
```

使用这种形式的命令时应当万分小心!

如果行与多个关系有关，那么删除时也应当小心。例如，在删除与CUSTOMER_T相关的

ORDER_T的行之前，如果我们要删除CUSTOMER_T的行，那么就违反了参照完整性规则（注意：在字段定义中使用ON DELETE子句能缓解这个问题，如果忘记了ON DELETE的用法，可参考本章的7.6.4节）。SQL确实能删除由DELETE命令选定的记录。因此我们常常会先执行一个SELECT命令，显示要删除的记录，从而核实这些行确实是那些想删除的行。

7.7.3 修改数据库内容

为了更新SQL中的数据，我们一定要告知DBMS，在更新中涉及到哪些关系、列、行。如果在PRODUCT_T表中，输入的餐桌价格不正确，那么下列SQL UPDTAE语句可以更正它。

命令　在PRODUCT_T中，把产品号为7的单价修改为775。

```
UPDATE PRODUCT_T
  SET UNIT PRICE=775
    WHERE PRODUCT_ID=7;
```

SET命令也可以把值置为NULL，其语法是："SET *列名* = NULL"。同DELETE命令一样，UPDATE命令中的WHERE子句可以包含一个子查询，但是在子查询中不能引用正在被更新的表。子查询将在第8章中讨论。

7.8 RDBMS中的内模式定义

控制关系数据库的内模式可以提高处理和存储的效率。有一些技术可以用于调整关系数据库内部数据模型的操作性能，这些技术包括：

1) 选择对主键和（或）辅助键索引，从而提高选择行、联结表和排序行的速度，删除索引可以提高表更新的速度。你可以回顾第6章中关于选择索引的一节。

2) 选择基表的文件组织，使它与这些表中的处理活动相匹配（例如，按照一个经常使用的报表排序关键字对表进行物理排序）。

3) 选择索引的文件组织方式（索引也是表），使它适合于索引使用的方式。为索引文件分配额外的空间，这样不用重新组织索引，索引也能够扩展。

4) 聚簇数据，使得频繁联结的表的相关行可以在二级存储器中存放在一起，从而最小化检索时间。

5) 维护对表和索引的统计数据，使得DBMS能够找到执行各种数据库操作的最有效的办法。

并不是所有的SQL系统都会使用这里提到的每一项技术，但是最常用的技术是索引和聚簇，所以下面几节我们将讨论关于索引和聚簇的问题。

创建索引

大部分RDBMS都会创建索引，创建索引可以提高随机和连续访问基表的速度。由于ISO SQL标准一般不处理性能问题，所以它没有包括创建索引的标准语法。这里的例子使用的是Oracle语法，其目的是说明在大多数RDBMS中如何处理索引。值得注意的是，尽管用户在编写每一个SQL命令时并没有直接涉及索引，但是DBMS知道哪个现有的索引能够提高查询性能。通常既可以为主键和辅助键创建索引，又可以对单键和联结键（多列）创建索引。在一些系统中，用户能够将索引中的键按升序和降序排列。

例如，我们在Oracle中对CUSTOMER_T表的CUSTOMER_NAME列按字母顺序创建索引。

命令　在CUSTOMER表中，对顾客名字按字母顺序创建索引。

```
CREATE INDEX NAME_IDX ON CUSTOMER_T(CUSTOMER_NAME);
```

索引可以随时创建或删除。如果在关键字列上已经存在数据，那么对现有数据的索引将自动产生。如果定义索引为UNIQUE（使用CREATE UNIQUE INDEX语法）且现有数据与该条

件相冲突，那么将不能创建该索引。一旦索引创建，则在数据输入、更新或删除时，索引也将被更新。

当不再需要表、视图或索引时，我们可以使用DROP语句将其删除。例如，可以删除NAME_IDX索引。

命令 删除CUSTOMER表中对顾客名的索引。

```
DROP INDEX NAME_IDX;
```

尽管可以对表中每一个列进行索引，但是每当决定创建一个新的索引时，都应当小心。因为每个索引都会消耗额外的存储空间，并且一旦被索引的数据值改变，就会产生维护的开销。因此，这些代价会大大延长检索的响应时间，给在线用户带来延迟。在一个复杂限定条件下，即使对于关键字有多个索引可用，系统也可能只选一个索引。数据库设计者必须确切地知道，特定的RDBMS如何使用索引，从而对索引作出明智的选择。Oracle中包含一个Explain Plan工具，可以用它来查看SQL语句的处理顺序和将要使用的索引。输出结果中包括了一个代价评估，它比较以不同索引来运行语句的代价，从而决定哪个索引是最有效的。

7.9 处理单个表

在SQL中，可以使用四个数据操纵语言命令。我们已经简要讨论了其中的三个命令UPDATE、INSERTE和DELETE，并看到了第四个命令SELECT的几个例子。UPDATE、INSERT和DELETE命令可以修改表中的数据，而带有不同子句的SELECT命令可以查询表中的数据，回答许多不同的问题，以及特定查询。SQL命令的基本结构十分简单且易于掌握。但不要被这个表面现象迷惑，SQL实际上是一个功能强大的工具，它能表示复杂的数据分析过程。然而，由于SQL的基本语法相对来说比较容易学习，所以常会发生这样的事情：写出来的SQL查询语法是正确的，但不能回答需要回答的那个问题。所以在对一个大型工作数据库进行查询之前，应该先在一个小的测试数据集上仔细地测试查询，以确保它们能返回正确的结果。除了人工检查查询结果之外，还可以把查询分为较小部分，检查这些简单查询的结果，然后把这些结果合并起来。这样就能确保查询按照预期的方式工作。我们首先学习只作用于单个表的SQL查询。在第8章中，我们会把表联结起来，使用需要多张表的数据的查询。

7.9.1 SELECT语句的子句

大多数SQL数据检索语句包括以下三种子句：

SELECT 列出要从基表（或视图）投影到命令的结果表中的列（包括列的表达式）。

FROM 指定出现在结果表中列来自哪个表或视图，同时包括用来联结表以处理查询的表和视图。

WHERE 包括在单个表或视图中选择行的条件，以及表或视图之间的联结条件。

前两个子句是必需的，第三个子句仅在检索特定的行，或对多个表进行联结时才需要。本节的例子是从图7-3的数据中提取出来的。在下面这个例子中，我们随时都能从PRODUCT视图中显示标准价格低于$275的所有产品的产品名和数量。

查询6 哪些产品的标准价格低于$275？

```
SELECT PRODUCT_NAME, STANDARD_PRICE
    FROM PRODUCT_V
      WHERE STANDARD_PRICE<275;
```

结果：

PRODUCT_NAME	STANDARD_PRICE
End Table	175
Computer Desk	250
Coffee Table	200

每执行一个SELECT语句，就会返回一个结果表。显示列表时，可以使用两个特殊的关键字：DISTINCT和*。如果用户不希望在结果表中看到重复的行，则可以使用SELECT DISTINCT命令。在上一个例子中，假如松谷家具公司还有其他价格低于$275的计算机桌，查询结果中将出现重复元组。而SELECT DISTINCT PRODUCT_NAME将显示没有重复行的结果表。SELECT *显示FROM子句中所有表和视图的所有列，这里的*是指所有列的通配符。

同时，注意SELECT语句的子句必须按一定顺序排列，否则将出现语法错误，从而导致查询不能执行。根据所使用的SQL版本，限定数据库对象的名字也是很有必要的。如果在SQL命令中有任何模糊之处，就一定要指明所要求的数据来自哪张表和视图。例如，在图7-3中，CUSTOMER_ID这一列在CUSTOMER_T和ORDER_T中都存在。当某人拥有正在使用的数据库（即该用户创建了此表），他要从CUSTOMER_T中得到CUSTOMER_ID，就需要指定CUSTOMER_T.CUSTOMER_ID来说明需要的表。同样，如果要从ORDER_T中得到CUSTOMER_ID，就需要指定ORDER_T.CUSTOMER_ID来说明需要的表。即使他不在意从哪个表得到CUSTOMER_ID，他也必须明确地指定表，因为如果用户不指定，SQL就不能处理模糊语句。当使用他人创建的数据时，用户也必须通过增加表的所有者的用户ID来明确指定表的所有者。一个从CUSTOMER_T表中选择CUSTOMER_ID的语句应该是*OWNER_ID*.CUSTOMER_T. CUSTOMER_ID。本书的例子都假定读者拥有正在使用的表或视图，因为没有限定词的SELECT语句更容易阅读。在必须的地方必须包括限定词，在你认为合适的语句中也可以包括限定词。遗漏限定词可能会产生问题，但包括限定词是不会产生问题的。

如果键入的限定词和列名有些乏味，或者对那些阅读报表的人来说，这些列名没有意义，那么可以为列、表和视图建立别名，这样在查询的剩余部分中就可以使用这些别名了。

查询7　名为Home Furnishings的顾客的地址是什么？为顾客名字建立别名NAME。

```
SELECT CUST.CUSTOMER_NAME AS NAME, CUST.CUSTOMER_ADDRESS
  FROM ownerid.CUSTOMER_V CUST
    WHERE NAME='Home Furnishings';
```

在许多的SQL版本中，这个检索语句可以得到下面的结果。在Oracle的SQL*Plus中，列的别名除了可以在HAVING子句中使用外，不能在SELECT语句的其他子句中使用，所以为了运行该SQL语句，在最后一行中必须使用CUSTOMER_NAME而不是NAME。注意，打印出来列的标题是NAME而不是CUSTOMER_NAME；对于视图的别名，即使它直到FROM子句才定义出来，也可以在SELECT子句中使用。

结果：

NAME	CUSTOMER_ADDRESS
Home Furnishings	1900 Allard Ave.

当使用SELECT子句为结果表挑选列时，可以将这些列重新组织一下，使得它们在结果表中的顺序和在原来的表中的顺序有所不同。事实上，它们显示的顺序和在SELECT语句中定义的顺序相同。再看一下图7-3中的PRODUCT_T，观察一下对于此查询，基表中的顺序和结果表中的顺序的不同。

查询8　列出PRODUCT表中所有产品的单价、产品名和产品ID。

```
SELECT STANDARD_PRICE, PRODUCT_DESCRIPTION, PRODUCT_ID
  FROM PRODUCT_T;
```

结果:

```
STANDARD_PRICE    PRODUCT_DESCRIPTION        PRODUCT_ID
--------------    -------------------        ----------
           175    End Table                           1
           200    Coffee Table                        2
           375    Computer Desk                       3
           650    Entertainment Center                4
           325    Writer's Desk                       5
           750    8-Drawer Desk                       6
           800    Dining Table                        7
           250    Computer Desk                       8
8 rows selected.
```

7.9.2 使用表达式

在某个表中，使用最基本的SELECT-FROM-WHERE子句，还可以处理其他一些事情。你可以创建*表达式*（expression），它是对表中的数据进行的数学运算；也可以利用*存储函数*（stored function），例如SUM或AVG，来处理从表中选定的数据行。也许人们想知道各个清单项目的平均标准价格，为了得到这个平均值，可以使用AVG存储函数。给这个结果表达式取名为AVERAGE。下面是使用SQL*Plus得到的查询和结果。

查询9 清单中各个产品的平均标准价格是多少?

```
SELECT AVG(STANDARD_PRICE) AS AVERAGE
       FROM PRODUCT_V;
```

可以通过使用“+”（加）、“-”（减）、“*”（乘）、“/”（除）来构建数学运算。这些操作符能与任何数字类型的列在一起使用。有些系统还有模（modulo）操作符，通常用“%”表示。模是两个整数相除的余数，例如，14%4=2，因为14/4=3，余数为2。这个标准还支持年-月和天-时间的间隔，从而能够完成日期和时间算术运算。注意，上面查询得到的结果现在显示为表达式AVERAGE。

结果:

```
AVERAGE
-------
440.625
```

复杂表达式中的顺序优先规则与其他程序语言和关系代数中使用的优先规则是一样的。首先计算括号里的表达式；没有括号时，先从左至右计算乘除，然后从左至右计算加减。为了避免混乱，使用括号来明确顺序。当括号嵌套时，最先完成里层括号的计算。

7.9.3 使用函数

在SELECT命令的列的列表中，对于指定列使用像COUNT、MIN、MAX、SUM和AVG之类的函数，可以指定结果表中包含的是聚合数据，而不是行级数据。使用其中的任何一个聚合函数，都将得到一行值。

查询10 1004号订单订购了多少不同的产品?

```
SELECT COUNT(*)
  FROM ORDER_LINE_V
  WHERE ORDER_ID=1004;
```

结果:

```
COUNT(*)
--------
       2
```

似乎通过修改上面的查询，就可以很容易地显示出1004号订单。

查询11　1004号订单订购了多少种不同产品，分别列出这些产品。

```
SELECT PRODUCT_ID, COUNT(*)
  FROM ORDER_LINE_V
       WHERE ORDER_ID=1004;
```

但是在Oracle中，会产生这样的结果。

结果:

```
ERROR at line 1:
ORA-00937:not a single-group group function
```

问题在于，对于选中的两行，PRODUCT_ID返回两个值6和8；而对于ID＝1004的行集合来说，COUNT返回一个聚合值2。在大多数实现中，SQL语句不能返回一行值和一个集合值，我们必须运行两个单独的查询，一个查询返回行信息，另一个查询返回集合信息。

同时，函数COUNT(*)和COUNT很容易混淆。上面所使用的COUNT(*)，不管行中是否包含空值，计算查询选中的所有行；而COUNT只计算有值的行，忽略所有的空值。

SUM和AVG只能用于数字类型的列。COUNT、COUNT(*)、MIN和MAX能与任何数据类型一起使用。例如，在文本类型列中使用MIN，可以找到列中的最小值，这个值的首字母与字母表的开始部分最接近。不同SQL的实现对字母表的顺序有不同的解释。例如，有些系统从A～Z开始，然后是a～z，再下来是0～9和特殊字符；而有些系统不区分大小写；还有些系统以一些特殊字符开始，然后依次是数字、字母和其他特殊字符。在Oracle中，当创建了一个数据库时，语言字符集就确定了，除非重建数据库，否则就无法对该语言字符集进行修改。当客户终端使用不同的字符集时，Oracle自动地转换字符集，或者把数据库字符集转变为客户终端的字符集，或者把客户终端的字符集转换为数据库字符集。当无法直接转换字符集时，就使用替代字符。下面有一个查询：依照字母顺序从PRODUCT_T表中找到第一个PRODUCT_NAME（使用的是Oracle 8*i*中的AMERICAN字符集）。

查询12　在PRODUCT表中，依照字母顺序，查询到的第一个产品名是什么？

```
SELECT MIN(PRODUCT_DESCRIPTION)
   FROM PRODUCT_V;
```

下面给出了结果，从结果可以看出，在这个字符集中，数字排在字母之前。

结果:

```
MIN(PRODUCT_DESCRIPTION)
------------------------
8-Drawer Desk
```

7.9.4　使用通配符

前面我们已经展示了在一条SELECT语句中，将星号（*）作为通配符使用的情形。通配符也可用在不能准确匹配的WHERE子句中。在这里，关键字LIKE与通配符以及字符串在一起配对使用，该字符串包含已经知道的需要匹配的字符。%通配符代表任何字符集，因此在搜索PRODUCT_DESCRIPTION时，使用LIKR '%DESK'就能够找到松谷家具公司生产的不同型号的桌子。下划线（_）作为通配符时仅仅代表一个字符，而不是字符集。因此当搜索PRODUCT_NAME时，使用LIKE '_-drawer'可以找到所有有特定抽屉的产品，比如3-drawer、

5-drawer和8-drawer的梳妆台。

7.9.5 比较运算符

除了本节第一个SQL例子之外，我们在其他例子的WHERE子句中都使用了等于比较运算符。第一个例子使用的是大于（小于）运算符。SQL中最常用的比较运算符在表7-3中列出。在数字数据间通常会使用比较运算符，但在SQL中，用户也可以为字符数据和日期数据使用比较运算符。下面的例子要求查询自2000年10月24日以来所有发出的订单。

表7-3 SQL中的比较运算符

运算符	含 义
=	等于
>	大于
>=	大于等于
<	小于
<=	小于等于
<>	不等于
!=	不等于

查询13 查询2000年10月24日以来所有发出的订单。

```
SELECT ORDER_ID, ORDER_DATE
  FROM ORDER_V
     WHERE ORDER_DATE>'24-OCT-2000';
```

注意，日期用单引号括起，这种日期格式与图7-3所示的格式不同，图7-3取自MS-Access。该查询运行在SQL*Plus中。

结果：

```
ORDER_ID          ORDER_DATE
    1007          27-NOV-00
    1008          30-OCT-00
    1009          05-NOV-00
    1010          05-NOV-00
```

查询14 查询松谷家具公司中不是用cherry（樱桃木）做成的家具。

```
SELECT PRODUCT_DESCRIPTION, PRODUCT_FINISH
FROM PRODUCT_V
      WHERE PRODUCT_FINISH!='Cherry';
```

结果：

```
PRODUCT                PRODUCT_FINISH
Coffee Table           Natural Ash
Computer Desk          Natural Ash
Entertainment Center   Natural Maple
8-Drawer Desk          White Ash
Dining Table           Natural Ash
Computer Desk          Walnut
6 rows selected.
```

7.9.6 使用布尔运算符

通过进一步调整WHERE子句，可以解决更复杂问题。利用布尔（或逻辑）运算符AND、OR和NOT可以达到这一目的。

- AND 联结两个或多个条件，仅当所有条件都为真时才返回结果。
- OR 联结两个或多个条件，当任一条件为真时就能返回结果。
- NOT 对表达式取否。

如果在SQL语句中有多个布尔运算符，那么先判定NOT，然后判定AND，最后判定OR。比如，考察下列查询。

查询15　在PRODUCT视图中，列出所有的办公桌（desk）以及单价大于$300的餐桌（table）的产品名、材料和单价。

```
SELECT PRODUCT_DESCRIPTION, PRODUCT_FINISH, STANDARD_PRICE
  FROM PRODUCT_V
    WHERE PRODUCT_DESCRIPTION LIKE '%Desk'
         OR PRODUCT_DESCRIPTION LIKE '%Table'
         AND UNIT_PRICE>300;
```

这里，列出了所有Desk，甚至包括单价小于$300的Computer Desk。仅有一张Table列出来，单价不到$300的廉价Table没有列出来。看看查询语句，先处理AND，返回所有单价大于$300的Table。然后处理OR，返回所有Desk，而无须考虑价格。

结果:

PRODUCT_DESCRIPTION	PRODUCT_FINISH	STANDARD_PRICE
Computer Desk	Natural Ash	375
Writer's Desk	Cherry	325
8-Drawer Desk	White Ash	750
Dining Table	Natural Ash	800
Computer Desk	Walnut	250

如果我们只想列出单价大于$300的Desk和Table，那么我们应该在WHERE之后、AND之前放上括号。

查询16　在PRODUCT视图中，列出所有单价大于$300的Desk和Table的产品名、材料和单价。

```
SELECT PRODUCT_DESCRIPTION, PRODUCT_FINISH, UNIT_PRICE
  FROM PRODUCT_V
    WHERE (PRODUCT_NAME LIKE '%Desk'
         OR PRODUCT_NAME LIKE '%Table')
         AND UNIT_PRICE>300;
```

现在返回的结果显示如下，仅仅包括那些单价大于$300的产品。

结果:

PRODUCT_DESCRIPTION	PRODUCT_FINISH	STANDARD_PRICE
Computer Desk	Natural Ash	375
Writer's Desk	Cherry	325
8-Drawer Desk	White Ash	750
Dining Table	Natural Ash	800

7.9.7　范围

比较运算符“<”和“>”可用来确定值的范围，也可以使用关键字BETWEEN或NOT BETWEEN来确定值的范围。例如，找出标准价格在$200和$300之间的产品，使用如下查询。

查询17　在PRODUCT视图中，标准价格在$200和$300之间的产品有哪些?

```
SELECT PRODUCT_DESCRIPTION, STANDARD_PRICE
  FROM PRODUCT_V
    WHERE STANDARD_PRICE>199 AND STANDARD_PRICE<301;
```

结果:

```
PRODUCT_NAME          STANDARD_PRICE
Coffee Table                     200
Computer Desk                    250
```

下面这个查询也返回同样的结果。

查询18 在PRODUCT视图中，标准价格在$200和$300之间的产品有哪些?

```
SELECT PRODUCT_DESCRIPTION, STANDARD_PRICE
  FROM PRODUCT_V
    WHERE STANDARD_PRICE BETWEEN 200 AND 300;
```

结果：同前一查询。

在这个查询中的BETWEEN之前增加NOT，将返回PRODUCT_V中所有其他的产品，因为它们的单价小于$200或大于$300。

7.9.8 DISTINCT

有时，当返回不包含主键的行时，可能会出现重复的行。例如，考察下面的查询和它返回的结果。

查询19 包含在ORDER_LINE表中的订单号有哪些?

```
SELECT ORDER_ID
  FROM ORDER_LINE_T;
```

该查询返回18行结果，由于许多订单有多项，所以它们中有许多是重复的。

结果:

```
 ORDER_ID
     1001
     1001
     1001
     1002
     1003
     1004
     1004
     1005
     1006
     1006
     1006
     1007
     1007
     1008
     1008
     1009
     1009
     1010

18 rows selected.
```

然而，如果我们加入关键字DISTINCT，那么每个ORDER_ID在返回时只出现一次，在表中，这10个订单分别只出现一次。

查询20 包含在ORDER_LINE表中的订单号有哪些?

```
SELECT DISTINCT ORDER_ID
  FROM ORDER_LINE_V;
```

结果:

```
ORDER_ID
    1001
    1002
    1003
    1004
    1005
    1006
    1007
    1008
    1009
    1010

10 rows selected.
```

DISTINCT（以及与它相对应的ALL）在SELECT语句中只能使用一次。它出现在SELECT语句之后，所有要列出的列或者表达式之前。如果SELECT语句投影多个列，那么每一列都相同的行就不再显示。因此如果上述语句还包括QUANTITY，将会返回14行，因为现在只有4个重复行而不是8个重复行。例如，ORDER_ID 1004上两个订单的数量都为2，那么就删除第二个1004和2。

查询21　包含在ORDER_LINE表中，不重复的订单号和订单数量组合有哪些?

```
SELECT DISTINCT ORDER_ID, ORDERED_QUANTITY
  FROM ORDER_LINE_V;
```

结果:

```
ORDER_ID          QUANTITY
    1001                 1
    1001                 2
    1002                 5
    1003                 3
    1004                 2
    1005                 4
    1006                 1
    1006                 2
    1007                 2
    1007                 3
    1008                 3
    1009                 2
    1009                 3
    1010                10

14 rows selected.
```

7.9.9　IN和NOT IN列表

使用IN可以匹配一系列值。

查询22　列出所处的州的顾客。

```
SELECT CUSTOMER_NAME, CITY, STATE
  FROM CUSTOMER_V
    WHERE STATE IN ('FL', 'TX', 'CA', 'HI');
```

结果:

CUSTOMER_NAME	CITY	ST
Contemporary Casuals	Gainesville	FL
Value Furniture	Plano	TX
Impressions	Sacramento	CA
California Classics	Santa Clara	CA
M and H Casual Furniture	Clearwater	FL
Seminole Interiors	Seminole	FL
Kaneohe Homes	Kaneohe	HI

7 rows selected.

在使用子查询的SQL语句中，IN特别有用。这将在第8章中讨论。

7.9.10 排序结果：ORDER BY子句

观察前面得到的结果，似乎先显示California顾客，然后显示Florida顾客、Hawaii顾客、Texas顾客才更为合理。这里我们给出SQL语句的其他三个基本部分：

- ORDER BY 将结果中的行按升序或降序排列。
- GROUP BY 对中间结果表中的行进行分组，那些归为一组的行，其值在一列或多列上是相等的。
- HAVING 只能在GROUP BY之后使用，作为一个辅助WHERE子句，仅返回那些满足特定条件的组。

因此，我们可以通过增加ORDER BY子句来对顾客排序。

查询23 在CUSTOMER视图中，列出地址为Florida、Texas、California或Hawaii的所有顾客的CUSTOMER_NAME、CITY和STATE。依字母顺序，先按州，然后按州中的顾客名显示顾客。

```
SELECT CUSTOMER_NAME, CITY, STATE
  FROM CUSTOMER_V
    WHERE STATEIN('FL', 'TX', 'CA', 'HI')
       ORDER BY STATE, CUSTOMER_NAME;
```

现在的结果更容易阅读。

结果:

CUSTOMER_NAME	CITY	ST
California Classics	Santa Clara	CA
Impressions	Sacramento	CA
Contemporary Casuals	Gainesville	FL
M and H Casual Furniture	Clearwater	FL
Seminole Interiors	Seminole	FL
Kaneohe Homes	Kaneohe	HI
Value Furniture	Plano	TX

7 rows selected.

注意，同一个州的所有顾客放在一起，在每个州内，顾客名按字母顺序排列。这种排序是由ORDER BY子句中所列出的列的顺序所决定。在本例中，先将州按字母排列，然后排序顾客名。如果想从高到低排序，那么在排序的列后面放置一个DESC关键字即可。

NULL如何排序呢？SQL-92规定，空值可以放在最前或最后，即有值的列的前面或后面。NULL放在什么地方取决于SQL的具体实现。在SQL*Plus中，NULL排在最后。

7.9.11 分类结果：GROUP BY子句

GROUP BY子句与聚合函数（如SUM或COUNT）一起使用时特别有用。GROUP BY把一个表分为子集（按组），然后使用聚合函数为每个组生成汇总信息。由前面聚合函数例子返回的单个值称为**标量聚合**（scalar aggregate），当在GROUP BY子句使用聚合函数且返回多个值时，则称为**向量聚合**（vector aggregate）。

查询24 统计我们向其发货的、各个州的顾客总数。

```
SELECT STATE, COUNT(STATE)
  FROM CUSTOMER_V
    GROUP BY STATE;
```

结果：

```
ST      COUNT(STATE)
--  ----------------
CA                 2
CO                 1
FL                 3
HI                 1
MI                 1
NJ                 2
NY                 1
PA                 1
TX                 1
UT                 1
WA                 1

11 rows selected.
```

也可以在组内嵌套组；在对多个项排序时，也可使用同样的逻辑。

查询25 统计我们向其发货的、各个城市的顾客数，按州显示城市。

```
SELECT STATE, CITY, COUNT(CITY)
  FROM CUSTOMER_V
    GROUP BY STATE, CITY;
```

GROUP BY子句看似简单，但如果忽略子句的逻辑，就有可能产生不可预料的错误。当有GROUP BY子句时，可以在SELECT子句中指定的列就会受到限制，因为只能包括那些对于每个组只有一个值的列。在上面的查询中，每一组包含了一个城市和城市所在的州。SELECT语句包括“city”和“state”两列。由于每个城市和州的组合只有一个值，所以这条语句有效。但是，如果本节第一个查询的SELECT子句还包含“city”，那么由于GROUP BY子句只含有州，那么这条语句将出错。由于州可能有多个城市，而SELECT子句中的每个值只能对应GROUP BY组中的一个值的要求不能满足，因此SQL不能表达有意义的城市信息。一般而言，SELECT语句中引用的每个列必须在GROUP BY子句中被引用，除非这一列是一个包含在SELECT子句中的聚合函数的参数。

7.9.12 通过分类限定结果：HAVING子句

HAVING子句与WHERE子句类似，但它指定满足条件的组，而不是行，因此通常会看到HAVING子句跟在GROUP BY子句之后。

查询26 只查找有多个顾客的州。

```
SELECT STATE, COUNT(STATE)
  FROM CUSTOMER_V
```

```
GROUP BY STATE
   HAVING COUNT(STATE)>1;
```

该查询返回这样一个结果，它删除前面看到的只有一个顾客的那些州。记住，这里使用WHERE就无法工作，因为WHERE不允许聚合。更进一步说，WHERE限定行，而HAVING限定组。

结果:

ST	COUNT(STATE)
CA	2
FL	3
NJ	2

如果要在HAVING子句中包含多个条件，可以像在WHERE子句中一样使用AND、OR和NOT。为了进行归纳，这里给出一个命令，其中包含了全部六种子句，记住子句必须按这样顺序使用。

查询27 对平均标准价格小于750的材料分组，为每个组列出产品的材料和平均标准价格。

```
SELECT PRODUCT_FINISH, AVG(STANDARD_PRICE)
 FROM PRODUCT_V
    WHERE PRODUCT_FINISH IN('Cherry','Natural Ash','Natural Maple','White Ash')
      GROUP BY PRODUCT_FINISH
        HAVING AVG(STANDARD_PRICE)<750
          ORDER BY PRODUCT_FINISH;
```

结果:

PRODUCT_FINISH	AVG(UNIT_PRICE)
Cherry	250
Natural Ash	458.333333
Natural Maple	650

图7-8显示了SQL处理语句中子句的顺序。箭头表示处理的路径。记住，只有SELECT和FROM子句是必须的。注意，这里的处理顺序与创建语句使用的语法顺序不同。每个子句一旦执行，就会产生一个中间结果表，它供下一个子句中使用。用户不会看到中间结果表，只能看到最终的结果。记住图7-8中的顺序，就可以根据此顺序调试一个查询。先去除可选的子句，然后按照它们处理的顺序一次放回一个子句。通过这种方式，就能够看见中间结果，通常也能发现错误。

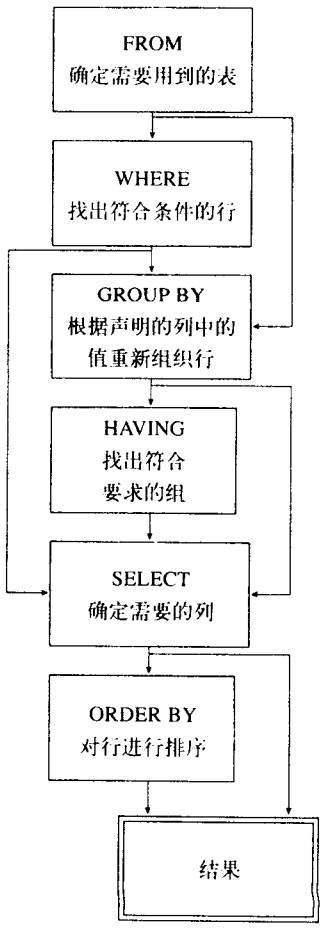


图7-8 SQL语句的处理顺序
（来源：van der Lans, p.100）

本章小结

本章介绍了用于关系数据库定义（DDL）、操纵（DML）和控制（DCL）的SQL语言，它通常用来定义和查询关系数据库管理系统（RDBMS）。这个标准由于存在许多缺陷而遭受批评，为了解决这些缺陷，提高语言的功能，ANSI X3H2委员会和ISO/IEC JTC1/SC21/WG3 DBL对其进行了修改，发布了它的扩展版本，目前的标准被称为SQL-99。

SQL标准的建立和一致性证明测试促使关系系统在当前数据库新的发展中逐渐占据主导地位。使用SQL标准的好处

包括可以减少培训费用，提高生产率、增加应用程序的可移植性和持久性，减少对个别厂商的依赖性，以及提高交叉系统的通信。

SQL环境包括一个SQL DBMS的实例，以及可以访问的数据库及相关的用户和程序。每个数据库包含在目录中，并且有一个描述数据库对象的模式。包含在目录中的信息由DBMS自身维护，而不是由DBMS的用户维护。

SQL的数据定义语言（DDL）命令用来定义一个数据库，包括数据库的创建以及数据库中表、索引和视图的创建。参照完整性也是通过DDL命令建立的。SQL的数据操纵语言（DML）命令通过使用SELECT命令来加载、更新和查询数据库。数据控制语言（DCL）命令用来创建用户对数据库的访问。

SQL命令能直接影响包含原始数据的基表，也能影响已创建的数据库视图。对视图的修改和更新可以（也可以不）传递给基表。SQL SELECT语句的基本语法包括以下关键字：SELECT、FROM、WHERE、ORDER BY和HAVING。SELECT决定在查询结果表中显示哪些属性；FROM决定查询中使用哪些表或视图；WHERE设定查询的条件，包括必要的多个表之间的任意联结；ORDER BY决定显示的结果的顺序；GROUP BY用来对结果分类，且可以返回标量聚合或者向量聚合；HAVING通过分类来限定结果。

理解本章介绍的基本SQL语法，能够使读者尝试有效使用SQL。通过不断的实践，读者可以进一步理解更复杂的查询。SQL的高级主题将在第8章中讨论。

本章复习

关键术语

基表
目录
数据控制语言（DCL）
数据定义语言（DDL）
数据操纵语言（DML）
动态视图
物化视图
参照完整性
关系数据库管理系统（RDBMS）
标量聚合
模式
向量聚合

复习问题

1. 定义下列每个术语。

a. 基表
b. 数据定义语言
c. 数据操纵语言
d. 动态视图
e. 物化视图
f. 参照完整性
g. 关系数据库管理系统（RDBMS）
h. 模式

2. 匹配下列术语及其定义。

_____视图
_____参照完整性
_____动态视图
_____物化视图
_____SQL-99
_____空值
_____标量聚合
_____向量聚合
_____目录
_____模式

a. 值列表
b. 数据库描述
c. 作为使用视图的SQL查询结果的被物化的视图
d. 逻辑表
e. 缺失或不存在的值
f. 对一个数据库中数据对象的描述
g. 嵌入SQL命令的第三代程序语言
h. 在关系数据模型使用外键来建立
i. 作为一张表存在的视图
j. 标准关系查询和定义语言

_____宿主语言　　k. 单个值

3. 对比下列术语。

a. 基表；视图　　b. 动态视图；物化视图

c. 目录；模式

4. 什么是SQL-92和SQL-99？简要描述SQL-99与SQL-92的不同。

5. 描述关系DBMS（RDBMS），包括它的基本数据模型、数据存储结构以及数据联系建立的方式。

6. 列出采用广泛接受的SQL标准所带来的六个潜在的好处。

7. 描述一个典型SQL环境的组件和结构。

8. 描述数据定义命令、数据操纵语言和数据控制语言的区别。

9. 解释在符合SQL-92标准的数据库中建立参照完整性的方法。解释ON UPDATE RESTRICT、ON UPDATE CASCADE和ON UPDATE SETNULL子句有何不同。如果使用ON DELETE CASCADE子句会发生什么？

10. 解释使用SQL创建视图的几种可能目的。特别要解释一下如何利用视图增强数据安全性。

11. 解释当通过视图引用数据时，为什么要限制数据更新操作的种类？

12. 描述使用视图能节省重新编程工作的环境集。

13. 利用前一章所讲的内容，解释在SQL中，决定是否要为一张表建立关键字索引时所需考虑的因素。

14. 解释并至少举出一个例子来解释在SQL中如何限定某人对一张表的所有权。

15. 出现在结果表中的属性的次序是如何改变的？列的标题是如何改变的？

16. 在SQL中，COUNT、COUNT DISTINCT和COUNT(*)有哪些区别？在哪些情况下这三个命令会产生相同的结果？哪些情况下会产生不同的结果？

17. 在SQL命令中，布尔操作符（AND、OR、NOT）的判定次序是什么？如何才能确保操作符是按期望的顺序工作？

18. 如果SQL语句包括GROUP BY子句，SELECT语句中能够请求的属性就会受到限制，请说明这种限制。

问题和练习

问题与练习1～9是基于第4章图4-16所描述的课程计划ERD来完成的。图7-9中多次提到3NF关系与一些示例数据。对于问题4～9，画一张实例图，在图中标记结果表中查询返回的数据。

STUDENT (STUDENT_ID, STUDENT_NAME)

STUDENT_ID	STUDENT_NAME
38214	Letersky
54907	Altvater
66324	Aiken
70542	Marra
...	

IS_QUALIFIED (FACULTY_ID, COURSE_ID, DATE_QUALIFIED)

FACULTY_ID	COURSE_ID	DATE_QUALIFIED
2143	ISM 3112	9/1988
2143	ISM 3113	9/1988
3467	ISM 4212	9/1995
3467	ISM 4930	9/1996
4756	ISM 3113	9/1991
4756	ISM 3112	9/1991
...		

图7-9 班级安排关系
（没有给出IS_ASSIGNED）

FACULTY (FACULTY_ID, FACULTY_NAME)

FACULTY_ID	FACULTY_NAME
2143	Birkin
3487	Berndt
4756	Collins
...	

SECTION (SECTION_ID, COURSE_ID)

SECTION_ID	COURSE_ID
2712	ISM 3113
2713	ISM 3113
2714	ISM 4212
2715	ISM 4930
...	

COURSE (COURSE_ID, COURSE_NAME)

COURSE_ID	COURSE_NAME
ISM 3113	Syst Analysis
ISM 3112	Syst Design
ISM 4212	Database
ISM 4930	Networking
...	

IS_REGISTERED (STUDENT_ID, SECTION_ID, SEMESTER)

STUDENT_ID	SECTION_ID	SEMESTER
38214	2714	I-2001
54907	2714	I-2001
54907	2715	I-2001
66324	2713	I-2001
...		

图7-9 （续）

1. 对于所显示的每个关系，使用SQL DDL写出它们的数据库描述（针对你的SQL版本，必要时可以缩短、简写或者修改数据名字）。假设属性的数据类型如下：

```
STUDENT_ID (integer, primary key)
STUDENT_NAME (25 characters)
FACULTY_ID (integer, primary key)
FACULTY_NAME (25 characters)
COURSE_ID (8 characters, primary key)
COURSE_NAME (15 characters)
DATE_QUALIFIED (date)
SECTION_ID (integer, primary key)
SEMESTER (7 characters)
```

2. 使用SQL定义下列视图：

STUDENT_ID	STUDENT_NAME
38214	Letersky
54907	Altvater
54907	Altvater
66324	Aiken
...	

3. 在向SECTION表中插入任意一行数据之前，准备输入的COURSE_ID必须已经存在于COURSE表中（参照完整性）。写一个实施此约束的SQL断言。

4. 写出下列查询的SQL定义命令。

a. 如何向STUDENT表中增加一个CLASS属性？

b. 如何删除IS_REGISTERED表？

c. 如何把FACULTY_NAME字段从25个字符改为40个字符？

5. a. 创建两种不同形式的INSERT命令，在STUDENT表中增加一个ID为65798，姓为Lopez的学生。
   b. 编写一个从STUDENT表中删除Lopez的命令。
   c. 创建一个SQL命令，把课程ISM 4212的名称从“Database”修改为“Introduction to Relational Database”。
6. 写出回答下列问题的SQL命令：
   a. 哪些学生的ID号小于50 000？
   b. 教职工ID为4756的教职工的姓名是什么？
   c. 2001年第一学期使用的最小Section号是什么？
7. 写出回答下列问题的SQL命令：
   a. 2001年第一学期，Section 2714有多少学生注册？
   b. 自1993年以来哪些教职工已经获得授课资格？列出教职工的ID，所教授的课程和获得授课资格的日期。
8. 写出回答下列问题的SQL命令：
   a. 哪些学生注册了Database和Networking课程？（提示：使用各个班级的SECTION_ID，你就能从IS_REGISTERED表中得到解答。）
   b. 哪些教师既不能教授Syst Analysis又不能教授Syst Design？
9. 写出回答下列问题的SQL命令：
   a. SECTION表中包含了哪些课程？一次列出一门课程。
   b. 按STUDENT_NAME的字母顺序列出所有的学生。
   c. 列出2001年第一学期注册了每门课程的学生，按他们注册的班级分组。
   d. 显示可选的课程。按课程的前缀分组（ISM是所显示的惟一的前缀，但是在大学里还有许多其他的前缀）。

问题和练习10～19要基于整个松谷家具公司数据库进行回答。

10. 通过增加属性QTY_ON_HAND来修改PRODUCT_T表，该属性用于跟踪已完成的货物清单。该字段是有5个字符的整数字段，且只能是正数。

11. 将你自己选择的样本数据输入到PRODUCT_T表的QTY_ON_HAND中。通过把一个产品的库存更新为10 000个，来测试问题与练习10中所作的修改。现在，再一次把产品的库存修改为-10个来进行测试。如果你没有收到错误信息，成功进行了这些修改，就表明在问题与练习10中没有建立合适的约束。

12. 向ORDER_T表中增加一个订单，为订单的每个属性包含一个示例值。
   a. 首先，看一下CUSTOMER_T表中的数据，为任意一个顾客输入订单。
   b. 然后，为一个新的顾客输入订单。如果你没有在CUSTOMER_T表中插入这个新顾客的信息，那么输入的订单数据会被拒绝。如果没有顾客的信息，那么参照完整性约束将会阻止你输入该客户的订单。
13. a. 松谷家具公司有多少加工中心？
   b. 这些加工中心位于什么地方？
14. 哪些产品的材料中含有橡木（oak）？
15. 显示居住在加利福尼亚或华盛顿的顾客，按邮政编码从高到低排序。
16. 确定每个产品系列的平均标准价格。
17. 列出那些姓以“L”开头的员工。

18. 哪些员工在1999年被雇佣?

19. 对于每个被订购的产品，确定订购的总量。列出最受欢迎的产品和最不受欢迎的产品。

应用练习

1. 咨询你所处地区某组织的一个数据库管理员。在采访该数据库管理员时，了解一下该组织目前正在使用的一个应用程序。在采访中要主要询问以下问题：确定终端用户与应用程序的关系，了解终端用户对SQL应熟悉到何种程度。例如，如果终端用户要使用SQL，他们要接受哪些培训？他们在工作中使用交互式的SQL吗？或者他们使用嵌入式的SQL吗？

2. 再一次采访该组织的数据库管理员。这次采访的重点在于了解该组织使用SQL的环境。询问所使用的SQL的版本，确定是否整个组织都使用了相同版本的SQL?如果使用了不同版本的SQL，探讨一下DBA在管理数据库时所遇到的困难。还要询问一下组织中使用了哪些专有的语言，比如Oracle的PL*SQL。了解一下不同部门所使用的版本的可能差异，探讨安装不同版本时会遇到的困难。

3. 同本地组织中的一个数据库开发人员交流。探讨在他的工作中使用SQL的情况。开发人员是使用代码生成器工具还是自己编写所有的SQL代码？开发人员是否有一个供自己支配的CASE工具，如Designer 2000？如果可能的话，请开发人员演示他正在使用的CASE工具。

参考文献

Codd, E. F.1970. "A Relational Model of Data for Large Relational Databases." *Communications of the ACM* 13 (June): 77-87.

Date, C. J., and H Darwen,1997. *A Guide to the SQL Standard*. Reading, MA: Addison-Wesley.

Gruber, M.2000. *Mastering SQL*. Alameda, CA: SYBEX.

Gulutzan, P., and T. Petzer. 1999. *SQL-99 Complete, Really*. Lawrence, KS: R&D Books.

Melton, J. 1997. "A Case for SQL Conformance Testing." *Database Programming & Design* 10 (7): 66-69.

Van der Lans, R. F. 1993. *Introduction to SQL*, 2nd ed. Workingham, England: Addison-Wesley.

进一步阅读

American National Standards Institute. http://www.ansi.org, 1997-2000. [Information on the ANSI federation and latest national and international standards.]

Bowman, J.S., S.L. Emerson, and M. Darnovsky. 1996. *The Practical SQL Handbook*, 3rd ed. Reading, MA: Addison-Wesley.

Celko, J. 1997. *Joe Celko's SQL Puzzles & Answers*. San Francisco: Morgan Kaufmann.

IEEE Standards Systems/Network Staff. *IEEE standards Home Page*, http://standards.ieee.org, 2000.[IEEE Standards Association information.]

ODMG Home Page, http://www.odmg.org/, 1993-2000. [Consortium of object-oriented database management system(ODBMS) vendors and interested parties.]

OMG Home Page, http://www.omg.org/, 1997-2000. [Object Management Group: information about CORBA.]

SW Consulting, SA, *IEC-International Electrotechnical Commission-Home Page (English)*, http://www.iec.ch, 2000.[International standards and conformity assessment body for all fields of electrotechnology.]

Tibbs, A. *Welcome to NCITS*, http:// www.ncits.org/, Undated. (Accessed December 28, 2000.)

[National Committee for Information Technology Standards–used to be Accredited Standard Committee X3.]

Welcome to ISO Online, http://www.iso.ch, December 11, 2000. [Information about the International Organization for Standardization.]

Web资源

- *Cramsessions for Oracle certifications*, http://www.cramsession.com/cramsession/Oracle , 2000年12月28日（为Oracle OCP 考试提供学习帮助。非Oracle站点）。
- Cumming, A. *A Gentle Introduction to SQL*, http://www.dcs.napier.ac.uk/~andrew/ sql, 2000年12月28日（包括一个在线的SQL引擎和来自现实数据的例子）。
- Farland, D, *Exam Facts: Introduction to Oracle:SQL and PL/SQL*, http://www.oraclenotes.com/exam1.htm, 2000(关于OCP考试的信息、大纲、参考文献，讨论组的信息，非Oracle站点)。
- Hoffman, J, *Introduction to Structured Query Language*, http://w3.one.net/ ~jhoffman/sqltut.htm, 2000年12月28日（丰富的ANSI SQL学习资料）。
- *Hot Oracle!*, http://www.hot-oracle.com, 2000年12月28日（Oracle相关站点，包括关于SQL* Plus, PL/SQL及Java和JavaScript的各种教程）。
- Oracle University, *Self Test Software*, http://education.oracle.com/certification/sts.html, 2000年12月28日（关于OCP考试的各种信息，以及免费的测验问题示例，Oracle站点）。
- *SQL Interpreter and Tutorial*, http://www.sqlcourse.com/ ,2000年12月28日（带有实际数据库的ANSI SQL的子集）。

项目案例：山景社区医院

利用第4章中为山景社区医院建立的SQL数据模型，回答下列项目问题并完成项目练习。

项目问题

1) 你将用哪一个版本的SQL来完成项目练习?

2) 哪一个CASE工具可以用来完成项目练习?

项目练习

1) 用SQL语句为在前几章构建的概念数据模型建立山景社区医院数据库。你可利用第3章、第4章、第5章和第6章的项目案例中所提供的信息，选择列的数据类型、长度和索引等。

2) 如果在前一个小题中没有建立主键和外键，那么利用SQL语句建立主键和外键。

3) 选择已建立的一部分数据库，并向其中插入示例数据，例如，你可以使用数据库中医护人员、诊疗中心以及病人的有关部分，也可以利用数据库中与药厂、内科/外科项目、检查以及病人的有关部分。检测你插入的数据，并回答这些实际的数值如何帮助你测试数据库的功能。

4) 编写并测试基于上述例子数据的查询。编写如下查询:

a. 仅从一张表中选择所需的信息。

b. 计算表中某个属性的聚集值。

c. 尝试使用其他的聚合函数如MIN、MAX和AVG等。

第8章 高 级 SQL

8.1 学习目标

学完本章后，读者应该具备以下能力：

- 定义下列关键术语：联结、等值联结、自然联结、外联结、相关子查询、用户自定义数据类型、持久化存储模块（SQL/PSM）、触发器、函数、过程、嵌入式SQL和动态SQL。
- 使用SQL命令编写对单表和多表的查询。
- 理解三种类型的联结命令，并能够使用SQL语句编写这些联结命令。
- 编写非相关和相关子查询，理解其使用场合。
- 使用SQL建立参照完整性。
- 理解数据库触发器和存储过程的使用方法。
- 对SQL-99标准进行讨论并解释它对SQL-92进行增强和扩展的部分。

8.2 引言

在上一章中介绍了SQL，并探讨了SQL在查询单张表时的作用。但是，只有存储并操作许多关联实体的数据对象时，关系模型才真正发挥出它的优势。要想充分利用这种优势，必须建立表之间的关系，同时构造从多个表中得到数据的查询。本章将详细介绍这种多表查询。从多个表中得到查询结果的方法有很多，包括使用子查询、内联结和外联结以及并联结等。

在掌握基本的SQL语法之后，应理解如何在创建应用时使用SQL。触发器是一小段包含SQL命令的代码模块，当预先设定的条件满足时，触发器将自动执行。过程也是一个类似的代码模块，但它必须被显式调用后才能执行。SQL命令经常嵌入到用宿主语言（如C和Java）编写的模块中。动态SQL可以在运行中创建SQL命令，根据需要插入参数值，是Web应用所必需的。本章通过一些实例对以上内容作了简要介绍。同时本章还介绍了SQL-99中对SQL进行增强和扩展的一些内容。

学完本章后，读者应对SQL有比较全面的了解，并掌握一些常见的使用方法。SQL还有许多应用于特殊情况下的附加特性，这些特性不是SQL的基本特性，有兴趣的读者可以参阅一些专门讲解SQL的书籍。利用本章所讲的语法进行实践，学生可以顺利地掌握SQL。

8.3 处理多表

我们已经介绍了SQL在单表操作中的用途，利用SQL对多表进行操作的方法是类似的。RDBMS只有在操纵多张表时才能完全展现出它的功能。如果多个表之间存在联系，那么它们可以通过查询链接起来。回忆第5章中讲过，在需要建立联系时，通过包含每个表的一个公共列（也可能是多个公共的列）来建立联系。这常常通过建立主键-外键联系来实现；外键是对另一张表中主键的引用，它们具有相同的域。两张表之间的联系可以通过查找两个表公共列的相同值来建立。例如在图7-3中，Order_t表中的Customer_ID属性和Customer_t表中的Customer_ID是相对应的。比较这些列和它们的值，可以发现Contemporary Casuals 发出了订单#1001和#1010，这是因为Contemporary Casuals的Customer_ID是1，而Order_t显示Order_ID为

1001和1010的订单是由Customer_ID为1的人发出的。在一个关系系统中，几个相联系表的数据合并成了一个结果表或视图，然后显示出来，或作为表单和报表定义的输入。

在不同类型的关系系统中，链接相关联的表的方法可能是不同的。在SQL中，SELECT命令中的WHERE子句也可以用于多表操作。事实上，在一个SELECT命令中可以引用两张表、三张表或更多的表。下面将要介绍，有两种不同的使用SELECT命令的方法可以合并多张表的数据。

最常用的方法是使用关系操作，该操作从两张或多张表提取数据到一个结果表中，该过程称为**联结**（join）。SQL通过WHERE子句匹配多个表的公共列，来隐含地实现多个表的联结。如果有两张表，一张表中有一个和另一张表中对应列有同样值域的列，则这两张表可以建立联结。联结操作的结果是一张单表，其中包括了表中所有被选择的列。联结操作返回的行都由参加联结的表中对应行的数据组成，这些对应行上的公共列的值相匹配。

在编写联结条件时有一条重要的规则：对每一对联结的表，都必须在一个WHERE子句中编制一个联结条件。因此，如果对两张表进行组合，必须有一个条件；而如果对三张表进行组合（A、B和C），则必须有两个条件，因为存在两对表（A-B、B-C），其他情况依此类推。

在关系数据库查询中，可能会存在多种联结类型，每个SQL实现可能只支持其中的一部分。本章讲述4种联结类型：等值联结、自然联结、外联结和并联结。

8.3.1 等值联结

等值联结（equi-join）的条件是两张表中公共列的值相等。例如，如果想知道下订单的顾客姓名，而这些信息存储在CUSTOMER_T和ORDER_T两张表中（注意，本章的学习基于Oracle数据库，所以使用大写字母来表示表和属性名）。首先对顾客和他们的订单进行匹配，然后根据问题选择相关的信息，最后将顾客姓名和订单号显示在一张表中。

查询1　找出所有已经下订单的顾客姓名。

```
SELECT CUSTOMER_T.CUSTOMER_ID, ORDER_T.CUSTOMER_ID,
CUSTOMER_NAME, ORDER_ID
 FROM CUSTOMER_T, ORDER_T
      WHERE CUSTOMER_T.CUSTOMER_ID = ORDER_T.CUSTOMER_ID;
```

结果：

CUSTOMER_ID	CUSTOMER_ID	CUSTOMER_NAME	ORDER_ID
1	1	Contemporary Casuals	1001
8	8	California Classics	1002
15	15	Mountain Scenes	1003
5	5	Impressions	1004
3	3	Home Furnishings	1005
2	2	Value Furniture	1006
11	11	American Euro Lifestyles	1007
12	12	Battle Creek Furniture	1008
4	4	Eastern Furniture	1009
1	1	Contemporary Casuals	1010

10 rows selected.

CUSTOMER_ID列的值是有重复的，因为两张表中均包含该列，说明所有的客户ID号均被匹配，匹配的结果是每行中有一个不重复的订单号。WHERE子句决不能忽略，否则查询结果将会返回所有顾客和订单的组合，即此两表的笛卡儿积。笛卡儿积返回的行数等于两张表中行数的乘积（10个订单 × 15个客户 = 150行），而这个结果没有任何实际意义。

同样，为了说明Oracle SQL和Access SQL的区别，下面给出了Access SQL产生的查询。

查询2 找出所有已经下订单的顾客名。

```
SELECT Customer_t.Customer_ID, Order_t.Customer_ID,
Customer_t.Customer_name, Order_t.Order_ID
FROM Customer_t INNER JOIN Order_t ON Customer_t.Customer_ID =
Order_t.Customer_ID;
```

结果：与上面查询的结果相同。

在Customer_t和Order_t表之间的这种联结称为内联结（inner join）。在这里，WHERE子句被子句ON Customer_t.Customer_ID = Order_t.Customer_ID代替；INNER JOIN-ON是SQL-92的语法，这种写法将越来越普及。注意，联结语法被移至FROM子句，使WHERE子句仅起过滤器的作用。

8.3.2 自然联结

自然联结（natural join）与等值联结是相同的，惟一的区别是，在使用两张表的同名列进行联结操作后，重复列之一不出现在联结结果中。自然联结是联结操作中最常用的一种。如通过SQL命令查询得到客户名和订单号，略去了一个CUSTOMER_ID。注意，CUSTOMER_ID列必须是被（表名）限定的，因为它有二义性，即CUSTOMER_ID同时存在于CUSTOMER_T表和ORDER_T表中，所以必须告诉SQL *Plus从哪张表中获得CUSTOMER_ID。

查询3 查询下订单的客户的客户姓名和订单号。

```
SELECT CUSTOMER_T.CUSTOMER_ID, CUSTOMER_NAME, ORDER_ID
 FROM CUSTOMER_T , ORDER_T
      WHERE CUSTOMER_T.CUSTOMER_ID = ORDER_T.CUSTOMER_ID;
```

注意，FROM子句中表名的顺序并不重要。DBMS的查询优化器将决定表的处理顺序。在相同列上是否存在索引将影响表的处理顺序，哪张表处于1:M联系的1方或M方也将影响表的处理顺序。如果一个查询所需的时间与FROM子句中的表的排列顺序显著相关，即其查询效率需要用户手工干预，则说明该DBMS不具备一个很好的查询优化器。

8.3.3 外联结

在对两张表作联结操作时，经常可以发现一张表中的某一行在另一张表中没有对应行。例如，在表CUSTOMER_T中的几个CUSTOMER_ID值在ORDER_T中并没有出现（如图7-3所示）。可以猜想这是因为那些客户直至2000年10月21日还没有发出订单，也可能是因为他们的订单没有包括在这个简单的ORDER_T例子表中。所以，上述等值联结和自然联结的结果没有包含CUSTOMER_T中所有的客户名。

企业可能希望确定那些还没有发出订单的潜在客户，希望与这些客户联系并获得新订单，或者想要分析这些客户从而找出没有他们没有下订单的原因。使用**外联结**（outer join）可以产生这些信息：在相同列上没有对应值的行也可以包含在结果表中。在表之间找不到对应值时，在该列上的位置上以空值填充。

大多数厂商的RDBMS产品都提供了外联结功能，但不同厂商提供的用于实现外联结操作的语法却不尽相同。下面的例子所使用的是ANSI标准语法。如果某种DBMS中没有明确定义外联结，可以使用UNION和NOT EXIST（将在本章后面的部分讨论）来实现外联结操作。下面是一个外联结的例子。

查询4 列出CUSTOMER表中所有顾客的姓名，ID号和订单号。包括没有下订单的顾客的ID号和姓名。

```
SELECT CUSTOMER_T.CUSTOMER_ID, CUSTOMER_NAME, ORDER_ID
```

```
FROM CUSTOMER_T LEFT OUTER JOIN ORDER_T
    WHERE CUSTOMER_T.CUSTOMER_ID  = ORDER_T.CUSTOMER_ID;
```

上面使用了LEFT OUTER JOIN语法，这是因为CUSTOMER_T表被放在联结关键字的左边，而期望的结果应包括该表的所有内容，所以无论在ORDER_T表中是否有订单与之匹配，表CUSTOMER_T的所有内容都将出现在结果表中。如果将表的排列顺序倒置，那么使用RIGHT OUT JOIN可以保持查询结果不变。用户也可以使用FULL OUTER JOIN语法，在这种情况下，匹配后返回所有的行，包括那些在另一张表中没有匹配值的行。INNER JOIN比OUTER JOIN更常用，因为外联结只在用户认为使用内联结会遗漏必要信息的情况下使用。

还应注意的是，OUTER JOIN 难以应用在超过两张表的情况下，返回的结果会因DBMS的不同而不同。所以，如果不清楚所用的DBMS是如何对外联结命令进行解释的，应该对涉及两张以上表的外联结命令语法进行测试。

同样，在外联结产生的结果表中，可能以NULL值来表示在第二张表中没有匹配到的值。如果表本身可以使用NULL作为数据值，那么用户就无法分辨这些空值是原有的值还是未匹配成功而填充的空值，除非在基表或视图上运行另一个查询来检验空值。而且，在运行OUTER JOIN所得到的结果表中，被定义为NOT NULL的列可能被赋予一个空值。

结果:

CUSTOMER_ID	CUSTOMER_NAME	ORDER_ID
1	Contemporary Casuals	1001
1	Contemporary Casuals	1010
2	Value Furniture	1006
3	Home Furnishings	1005
4	Eastern Furniture	1009
5	Impressions	1004
6	Furniture Gallery	
7	Period Furnishings	
8	California Classics	1002
9	M & H Casual Furniture	
10	Seminole Interiors	
11	American Euro Lifestyles	1007
12	Battle Creek Furniture	1008
13	Heritage Furnishings	
14	Kaneohe Homes	
15	Mountain Scenes	1003

16 rows selected.

使用外联结的优点是不会产生信息丢失。在上面的表中，返回了所有的客户名，无论这些客户是否下了订单。运行RIGHT OUT JOIN会返回所有订单（既然参照完整性要求每个订单都必须与一个合法的客户ID相关联，则使用右外联结时假定参照完整性已经被强制执行）。

查询5　列出所有与ORDER表中出现的订单相关的顾客名、ID号和订单号。即使某些订单没有顾客名和顾客号与之对应也包括在内。

```
SELECT CUSTOMER_T.CUSTOMER_ID, CUSTOMER_NAME, ORDER_ID
 FROM CUSTOMER_T RIGHT OUTER JOIN ORDER_T ON
     CUSTOMER_T.CUSTOMER_ID = ORDER_T. CUSTOMER_ID;
```

8.3.4 并联结

在SQL-99中还可以使用UNION JOIN（并联结），但这尚未在所有DBMS产品中实现。

UNION JOIN操作的结果返回一张结果表，该表中包含了参与联结的每张表的所有数据。结果表将包含参与联结的表的所有列，以及所有表中每一行数据所组成的记录。因此，CUSTOMER_T表（15个顾客，6个属性）和ORDER_T表（10张订单，3个属性）的UNION JOIN操作返回的结果表中将有25行和9列。假定每个原始表中都不包含空值，则结果表中每个顾客行将包括3个可以赋予空值的属性，而每个订单行将包括6个可以赋予空值的属性。注意不要将UNION JOIN命令和UNION命令混淆，后者用于联结多个SELECT语句，其具体内容将在本章后面部分讲述。

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_ADDRESS	CITY	ST	POSTAL_CODE
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094

ORDER_ID	ORDER-DAT	QUANTITY	PRODUCT_NAME	UNIT_PRICE	(QUANTITY*UNIT_PRICE)
1006	24-OCT-00	1	Entertainment Center	650	650
1006	24-OCT-00	2	Writer's Desk	325	650
1006	24-OCT-00	2	Dining Table	800	1600

图8-1 对四张表进行联结操作的结果（为便于阅读，排版形式有所改变）

8.3.5 例子：涉及4张表的多重联结

关系模型功能的强弱在很大程度上取决于它处理数据库中对象之间关系的能力。设计一个数据库时，使每个对象的数据分别存储在不同的表中，可以简化维护和数据完整性。可通过表的联结操作使对象彼此关联，可以获得关键的业务信息。第7章和第8章中的例子都比较简单，是为了让读者对SQL有基本的了解而构造的；但是读者必须认识到，这些命令能够构造出更加复杂的查询，从而为各种报表和处理提供信息。

下面是一个涉及4张表的联结查询的例子。这个查询产生的结果表为#1006订单创建发货单提供所需的信息。在此过程中需要顾客信息、订单和订单线信息以及产品信息，所以需要联结4张表。

查询6 组合所有必要信息，为订单#1006创建一张发货单。

```
SELECT CUSTOMER_T.CUSTOMER_ID, CUSTOMER_NAME, CUSTOMER_ADDRESS, CITY, STATE,
POSTAL_CODE, ORDER_T.ORDER_ID, ORDER_DATE, QUANTITY, PRODUCT_NAME, UNIT_PRICE,
(QUANTITY*UNIT_PRICE)
 FROM CUSTOMER_T, ORDER_T, ORDER_LINE_T, PRODUCT_T
    WHERE CUSTOMER_T. CUSTOMER_ID=ORDER_T. CUSTOMER_ID
        AND ORDER_T.ORDER_ID = ORDER_LINE_T.ORDER_ID
        AND ORDER_LINE_T.PRODUCT_ID = PRODUCT_T.PRODUCT_ID
        AND ORDER_T.ORDER_ID = 1006;
```

查询的结果如图8-1所示。记住，既然要联结4张表，就应该有3个列联结条件。

8.3.6 子查询

对两张表进行联结的基本方法有两种，前面的SQL例子描述了其中的一个方法，即联结技术。SQL还提供了子查询技术，即在查询的WHERE和HAVING子句中嵌入一个内部查询（SELECT、FROM、WHERE）。外部查询的查询条件中某些值由内部查询提供。这种查询被称为子查询或嵌套子查询，而且可以嵌套多层。

有时候，使用联结操作和子查询技术可以得到相同的结果，不同的人对这两种技术有自己的偏好。但在其他时候，只能使用联结或只能使用子查询才能实现相应的查询。要从多个联系中提取数据并显示，并且联系之间不必嵌套的时候，使用联结技术更方便。比较返回同样结果

的两个查询：查询发出#1008订单的顾客姓名和地址。首先，我们使用联结操作。

查询7　查询发出#1008订单的顾客姓名和地址。

```
SELECT CUSTOMER_NAME, CUSTOMER_ADDRESS, CITY, STATE, POSTAL_CODE
 FROM CUSTOMER_T, ORDER_T
      WHERE CUSTOMER_T.CUSTOMER_ID=ORDER_T.CUSTOMER_ID AND
          ORDER_ID = 1008;
```

使用子查询技术得到的查询与前一查询是等价的。

查询8　查询发出#1008订单的顾客姓名和地址。

```
SELECT CUSTOMER_NAME, CUSTOMER_ADDRESS, CITY, STATE, POSTAL_CODE
 FROM CUSTOMER_T
      WHERE CUSTOMER_T.CUSTOMER_ID =
          (SELECT ORDER_T.CUSTOMER_ID
               FROM ORDER_T
                    WHERE ORDER_ID = 1008);
```

注意，在圆括号中的子查询和前面学过的SQL查询在形式上是相同的，即也可以把它单独取出来形成一个独立的查询。因为在这个查询中只需要显示从外部查询中的表中得到的数据，所以可以使用子查询的方法。ORDER_ID的值没有出现在结果中——它作为外部查询的选择条件。如果要在最终的结果中显示子查询的结果，只能使用联结技术，因为子查询得到的结果不会包含在最终结果中。

可以预先知道在前面的子查询中最多返回一个值，即与ORDER_ID为#1008相关的CUSTOMER_ID。如果不存在这个ID值，则子查询的结果将为空。在子查询中使用关键字IN可以返回值的列表（无论是0条、1条还是多条记录）。例如，可以查询已经下订单的顾客。以下是相应的查询表达式。

查询9　哪些顾客下了订单？

```
SELECT CUSTOMER_NAME
  FROM CUSTOMER_T
       WHERE CUSTOMER_ID IN
           (SELECT DISTINCT CUSTOMER_ID
                FROM ORDER_T);
```

查询结果如下。因为并不关心一个顾客总共下了多少份订单，只关心他是否下了订单，所以在子查询中使用了DISTINCT关键字。对于每一个ORDER_T表中出现的顾客号，在CUSTOMER_T中查找其对应的顾客名并返回（我们将在图8-2a中再次讨论这个查询）。

结果：

```
CUSTOMER_NAME
Contemporary Casuals
Value Furniture
Home Furnishings
Eastern Furniture
Impressions
California Classics
American Euro Lifestyles
Battle Creek Furniture
Mountain Scenes

9 rows selected.
```

限定符NOT、ANY和ALL可用在IN之前或者与逻辑操作符（如=、>和<）结合使用。因为IN可以利用子查询返回的0个、1个或多个值，许多程序员在所有查询中也常常用IN来替代“=”，即使在“=”操作符也能适用的情况下也是如此。下一个例子显示NOT的使用方法，这个例子还说明了在子查询中也能使用联结操作。

查询10 查询没有订购计算机桌的顾客名。

```
SELECT CUSTOMER_NAME
  FROM CUSTOMER_T
       WHERE CUSTOMER_ID NOT IN
           (SELECT CUSTOMER_ID
                FROM ORDER_T,ORDER_LINT_T,PRODUCT_T
                     WHERE ORDER_T.ORDER_ID =
                           ORDER_LINE_T.ORDER_ID AND
                           ORDER_LINE_T.PRODUCT_ID =
                           PRODUCT_T.PRODUCT_ID
                           AND PRODUCT_NAME ='Computer Desk');
```

结果：

```
CUSTOMER_NAME
Value Furniture
Home Furnishings
Eastern Furniture
Furniture Gallery
Period Funiture
M H Casual Furniture
Seminole Interiors
American Euro Lifestyles
Heritage Furnishings
Kaneohe Homes

10 rows selected.
```

结果显示还有10个客户没有订购计算机桌。内层子查询返回了一个已经订购计算机桌的顾客列表。外部查询列出了没有在子查询返回值中列出的顾客姓名。

与使用子查询有关的另外两个条件是EXISTS 和NOT EXISTS。这些关键字在SQL查询中IN所在的位置出现，即子查询开始之前出现。如果子查询返回的中间结果表中包含一行或多行，则EXISTS为TRUE，如果返回结果中没有任何行，则EXISTS为FALSE。相反，NOT EXISTS在返回结果为空时取TRUE，而在返回结果有一行或多行时则取FALSE。考虑下面的SQL命令，其中包括了EXISTS。

查询11 找出所有符合以下要求的订单号，其中至少包含一种使用天然白蜡木（Natural Ash）制造的家具。

```
SELECT DISTINCT ORDER_ID FROM ORDER_LINE_T
WHERE EXISTS
 (SELECT *
      FROM PRODUCT_T
           WHERE PRODUCT_ID = ORDER_LINE_T.PRODUCT_ID
                AND PRODUCT_FINISH = 'Natural Ash');
```

子查询检查订单线上的产品是否是由天然白蜡木制造的。主查询选择出所有包含使用天然

白蜡木制造的家具的订单号。共有7张符合条件的订单，如下面的查询结果所示（在图8-2b中将进一步讨论这个查询）。

结果：

```
ORDER_ID
--------
    1001
    1002
    1003
    1006
    1007
    1008
    1009

7 rows selected.
```

在一个子查询中使用EXISTS 或NOT EXISTS时，子查询的选择列表通常使用“SELECT *”表示选择所有列，因为子查询返回哪些列并不重要。子查询的目的是检查是否有一些行满足条件，而不是返回特定列的值。外部查询确定要显示的列。EXISTS子查询一般是相关子查询（将在随后讨论）。如果不存在满足子查询的行，包含NOT EXISTS关键字的查询将返回一个结果表。

总之，当限定条件是嵌套的或很容易用嵌套方法实现时，应该使用子查询方法。SQL*Plus允许子查询返回多列，但大多数系统只允许在子查询中的某一特定列与外部查询的一列之间建立联结。但是与EXISTS一起使用的子查询例外。最终的结果表中仅显示外部查询中表的数据。典型情况下最多支持16层嵌套。查询按由内到外的次序执行，但另一种子查询——相关子查询——是由外到内处理的。

8.3.7 相关子查询

在第一个子查询的例子中，首先必须确认内部查询已经完成，才能进行外部查询。也就是说，内部查询的结果用来限定外部查询的处理。然而，在**相关子查询**（correlated subquery）中，外部查询的结果决定内部查询的处理。也就是说，对于引用外部查询中每一行的内部查询在某种程度上都是不同的。在这种情况下，内部查询必须根据每个外部查询的行进行计算。而在前面的例子中，内部查询只需计算一次，即可提供所有外部查询中要处理的行。图8-2a和图8-2b描述了在前面各节中子查询的每个例子的不同处理顺序。

下面的查询中列出了PRODUCT_T中产品单价最高的产品信息。

查询12　列出单价最高的产品信息。

```
SELECT PRODUCT_NAME, PRODUCT_FINISH, UNIT_PRICE
  FROM PRODUCT_T PA
       WHERE UNIT_PRICE > ALL
           (SELECT UNIT_PRICE FROM PRODUCT_T PB
                WHERE PB.PRODUCT_ID! = PA.PRODUCT_ID);
```

下面是结果，即单价最高的产品是餐桌。

结果：

```
PRODUCT_NAME      PRODUCT_FINISH     UNIT_PRICE
----------------- ------------------ ----------
Dining Table      Natural Ash               800
```

研究这个SQL语句的逻辑：对于每个产品，子查询都执行一次以确认没有其他产品具有更高的单价。注意，这需要将一个表和它自身相比较。为了完成这项工作，在这里给表起了两个

别名：PA和PB。首先考虑PRODUCT_ID为1的产品。子查询执行时，将返回除外部查询正考察的那个产品以外的每个产品的单价。然后，外部查询会检查正考察产品的单价是否比子查询返回的所有产品的单价都高。如果是，返回该产品作为查询结果。如果不是，再考虑外部查询的下一个值，而内部查询将会又返回除当前正考虑的产品以外的所有产品的单价列表。当外部查询的目标值变化时，内部查询返回的列表也会发生变化。这就构成了一个相关子查询。请读者思考，是否存在一组特别的单价，使得这个查询不能产生期望的结果？

8.3.8 使用导出表

子查询并不仅限于在WHERE子句中使用，也可以出现在FROM子句中，以创建一个在查询中应用的临时导出表。创建一个包含聚合值（如MAX、AVG或MIN）的导出表，就可以在WHERE子句中使用聚合。下面的查询列出超过平均标准价格的那些家具。

```
SELECT CUSTOMER_NAME
        FROM CUSTOMER_T
                WHERE CUSTOMER_ID IN
                        (SELECT DISTINCT CUSTOMER_ID
                        FROM ORDER_T);
```

1. 首先处理子查询（方框中的部分）并创建一个中间结果表：

```
CUSTOMER_ID
-----------
          1
          8
         15
          5
          3
          2
         11
         12
          4
9 rows selected.
```

2. 外部查询输出包含在中间结果表中所需要的每个顾客的信息：

```
CUSTOMER_NAME
-------------------------
Contemporary Casuals
Value Furniture
Home Furnishings
Eastern Furniture
Impressions
California Classics
American Euro Lifestyles
Battle Creek Furniture
Mountain Scenes
9 rows selected.
```

a) 处理一个非相关子查询

图8-2 子查询处理

```
SELECT DISTINCT ORDER_ID FROM ORDER_LINE_T
WHERE EXISTS
        (SELECT *
          FROM PRODUCT_T
                WHERE PRODUCT_ID = ORDER_LINE_T.PRODUCT_ID
                      AND PRODUCT_FINISH = 'Natural Ash');
```



	Product_ID	Product_Description	Product_Finish	Standard_Price	Product_Line_Id
▶		End Table	Cherry	$175.00	10001
	2	Coffee Table	Natural Ash	$200.00	20001
	4	Computer Desk	Natural Ash	$375.00	20001
		4 Entertainment Center	Natural Maple	$650.00	30001
		5 Writer's Desk	Cherry	$325.00	10001
		6 8-Drawer Dresser	White Ash	$750.00	40001
		7 Dining Table	Natural Ash	$800.00	20001
		8 Computer Desk	Walnut	$250.00	30001
*	(AutoNumber)			$0.00	

1. 从表ORDER_LINE_T中选择第一个订单ID：ORDER_ID=1001。
2. 执行子查询来判断订单中是否包含天然白蜡木产品。产品2是天然白蜡木产品并且包含在订单中，因此EXISTS为真，这个订单ID被加入结果表中。
3. 从表ORDER_LINE_T中选择第二个订单ID：ORDER_ID=1002。
4. 执行子查询发现订单中是含有天然白蜡木产品，产品3符合要求，因此EXISTS为真，这个订单ID被加入结果表中。
5. 对每个订单ID进行相同的处理，订单1004、1005和1010没有包括在结果表中。这是因为它们不包含任何天然白蜡木制成的家具。最后生成结果表是本查询[1]的查询结果。

b) 处理一个相关子查询

图8-2　（续）

查询13　哪些家具的标准价格超出了平均标准价格？

```
SELECT PRODUCT_DESCRIPTION, STANDARD_PRICE, AVGPRICE
 FROM
      (SELECT AVG(STANDARD_PRICE) AVGPRICE FROM PRODUCT_T),
      PRODUCT_T
        WHERE STANDARD_PRICE > AVGPRICE;
```

结果：

PRODUCT_DESCRIPTION	STANDARD_PRICE	AVGPRICE
Entertainment Center	650	440.625
8-Drawer Dresser	750	440.625
Dining Table	800	440.625

8.3.9　组合查询

可以使用UNION子句将多个查询所得的结果合并成一张单独的结果表。为了使用UNION子句，每个相关查询结果中的行数必须相同，并且是可以使用UNION的。这要求每个查询结果的对应列的数据类型都应该是兼容的。不同厂商的数据类型的兼容性可能有所不同。执行并

操作时，需要合并同一列上的不同数据类型，即进行数据类型转换；在这种情况下，最安全的方式是由程序员使用CAST命令控制数据类型的转换。例如，ORDER_T表中的数据类型DATE可能应该转换成文本类型。实现这个操作的SQL命令如下所示：

```
SELECT CAST(ORDER_DATE AS CHAR) FROM ORDER_T;
```

下面的例子查询购买松谷家具公司产品数量最多的客户，以及购买数量最小的客户，然后把结果放在同一张表中返回。

查询14

```
SELECT C1.CUSTOMER_ID,CUSTOMER_NAME,ORDERED_QUANTITY,'LARGEST
QUANTITY' QUANTITY
 FROM CUSTOMER_T C1,ORDER_T O1,ORDER_LINE_T Q1
      WHERE C1.CUSTOMER_ID=O1.CUSTOMER_ID
      AND O1.ORDER_ID=Q1.ORDER_ID
      AND ORDERED_QUANTITY=
           (SELECT MAX(ORDERED_QUANTITY)
           FROM ORDER_LINE_T
UNION
SELECT C1.CUSTOMER_ID,CUSTOMER_NAME,ORDERED_QUANTITY,'SMALLEST
QUANTITY'
 FROM CUSTOMER_T C1,ORDER_T O1,ORDER_LINE_T Q1
      WHERE C1.CUSTOMER_ID=O1.CUSTOMER_ID
      AND O1.ORDER_ID=Q1.ORDER_ID
      AND ORDERED_QUANTITY=
           (SELECT MIN(ORDERED_QUANTITY)
           FROM ORDER_LINE_T)
      ORDER BY ORDERED_QUANTITY
```

注意，为了保证可读性，创建了一个名为QUANTITY列，以字符串“Smallest Quantity”或“Largest Quantity”为值。ORDER BY子句用于指定输出行的排列顺序。

结果：

CUSTOMER_ID	CUSTOMER_NAME	ORDERED_QUANTITY	QUANTITY
1	Contemporary Casuals	1	Smallest Quantity
2	Value Furniture	1	Smallest Quantity
1	Contemporary Casuals	10	Largest Quantity

8.3.10 条件表达式

通过在语句中使用CASE关键字，可以在SQL表达式中建立IF-THEN-ELSE逻辑处理。图8-3给出了CASE的语法，实际上它有四种形式。CASE的形式可以是等于某一个值的表达式或谓词。谓词形式基于三值逻辑（真、假、未知），但允许进行更复杂的操作。CASE语句的值表达式形式需要与某个值表达式相匹配。NULLIF和COALESCE是与其他两种CASE表达式有关的关键字。

```
{CASE expression
{WHEN expression
THEN {expression | NULL}} . . .
| {WHEN predicate
THEN {expression | NULL}} . . .
[ELSE {expression  NULL}]
END }
| { NULLIF (expression, expression) }
| { COALESCE (expression . . .) }
```

图8-3 CASE条件语法

8.4 保证事务完整性

关系DBMS与其他数据库管理器有一点是相同的，即它们的基本任务是确保数据维护的正确性和完整性。数据维护在称为事务的工作单元中定义，该单元中包括了一个或多个数据操纵

命令。事务是由一组联系紧密的更新命令组成的完整集合，集合中的命令要么全部执行，要么全部不执行，从而保证数据库的正确性。考察图8-4，例如，当输入一个订单到松谷家具公司的数据库时，订单的所有项都应该同时输入。因此，要么将ORDER_T中的订单的所有ORDER_LINE_T行都输入，要么所有信息都不输入。这里，事务就是完整的一张订单，而不是订单中的部分数据。我们需要一些命令来定义事务范围，提交事务使数据库被永久性地更改，以及在必要时正确中止一个事务的执行。除此以外，还需要有数据恢复服务，以便在事务执行中出现数据库操作异常中止后进行清理。比如，订单是正确的，但是正在输入订单的时候，出现了计算机系统故障或断电。在这种情况下，用户不希望对数据库的修改一部分生效而其他部分无效。如果要保持数据库的正确性，那么在一个事务中对数据库的修改要么全部生效，要么全部撤销。

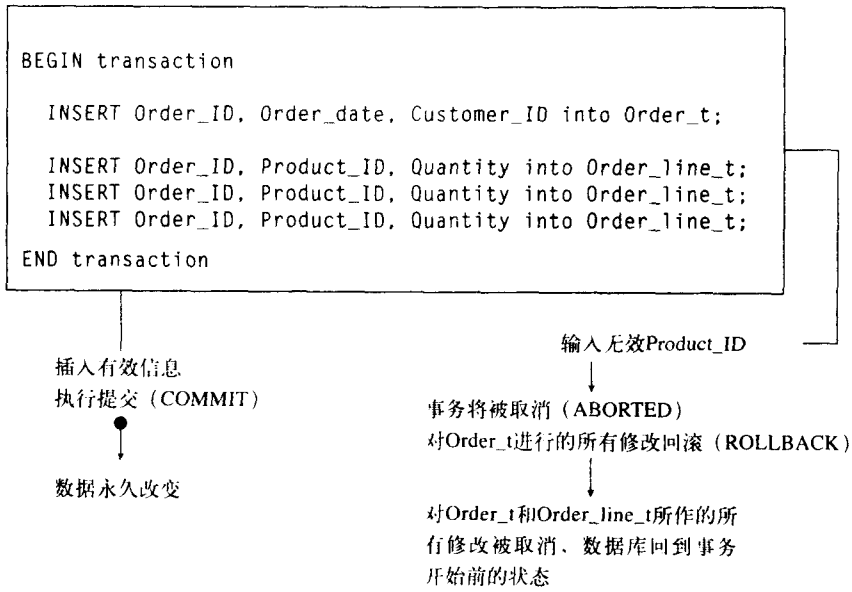


图8-4　一个SQL事务序列（伪代码形式）

当一个单独的SQL命令组成了一个事务时，一些RDBMS在命令执行后自动地提交或回滚。如果有一个用户自定义事务，其中包含多条SQL命令，这些SQL命令要么全部提交，要么全部回滚，那么就需要管理事务的命令。许多系统提供了BEGIN TRANSACTION和END TRANSACTION命令，可以用这两条命令标记一个逻辑单元的事务范围。BEGIN TRANSACTION创建一个日志文件，并开始在该文件中记录所有对数据库的修改（插入、删除和更新）。END TRANSACTION或COMMIT WORK提取出日志文件的内容并应用于数据库中（这样就实现了永久性改变），然后清空日志文件。ROLLBACK WORK提请SQL清空日志文件。有一些RDBMS厂商还提供AUTOCOMMIT(ON/OFF)命令，如果选择ON，表示在每条数据修改命令之后都对数据项做永久性的改变，而选择OFF，则表示需要由COMMIT WORK命令来显式标识对数据库做永久性改变。

用户自定义事务能够改善系统性能，因为将若干事务作为集合处理比逐个处理事务减少了系统负载。当AUTOCOMMIT置为OFF时，除非达到显式指定的事务结束点，否则不会自动地修改数据库。当AUTOCOMMIT置为ON时，在每条SQL语句的执行后，都自动修改数据库；这在必须作为整体被提交或回滚的用户自定义事务中是不允许的。

SET AUTOCOMMIT是一个交互式的命令，所以可以根据适当的完整性标准对一个用户会

话进行动态改变。一般情况下，一条SQL INSERT、UPDATE或DELETE 命令在某一时刻只能对一张表进行操作。而要完成某些数据维护工作则需要更新多张表。因此，这些保持事务完整性的命令就显得很重要，对于一组必须全部做完的修改数据库的操作，可以使用这些命令清晰地把这些修改定义为一个完整的逻辑单元，以保持数据库完整性。

进一步来说，某些SQL系统具有并发控制功能，它解决并发用户对共享数据库的更新。数据库的变化可以在日志中记录下来，所以如果在事务处理过程中出现异常中止的情况，则数据库可以利用日志恢复。这些系统还可以撤销错误的事务。例如，在银行应用中，两个并发用户对同一个银行账号余额的修改应该累积。在SQL中，这样的控制对用户来说是透明的：不需要用户编程来控制数据的并发访问。为了保证一个数据库的完整性，应该注意事务完整性及恢复问题，并确保应用程序开发人员知道应该在何时使用这些命令。

8.5 数据字典工具

RDBMS的数据库定义信息存储在系统创建的表中，我们可以将这些系统表看做数据字典。无论是用户还是数据库管理员，都应该熟悉所使用的RDBMS的系统表，它们能提供很多有用的信息。既然信息存储在表中，用户也可以通过SQL SELECT语句对其进行访问，得到关于系统使用、用户权限和约束等的报表。而且，掌握系统表结构的用户可以扩展现有的表或构建其他表以增强内置特性（例如，存储负责数据完整性的用户的信息）。然而，用户往往不能对系统表结构和内容做直接修改，因为DBMS需要维护这些表，并根据这些表来解释和分析查询。

每个RDBMS包含不同的系统表。在Oracle 8*i* 中，数据库管理员可以使用98个数据字典视图。那些不具有DBA权限的用户也可以使用其中许多视图或DBA视图的子集（与某个用户相关的信息）。这些表的名字以USER或ALL开头，而不是以DBA开头。下面列出了保存有关表、簇、列和安全等信息的表（仅供DBA访问）。还有一些表保存了有关存储、对象、索引、锁、审计、导出和分布式环境的信息。

- DBA_TABLES 描述数据库中的所有表。
- DBA_TAB_COMMENTS 数据库中所有表的注释。
- DBA_CLUSTERS 描述数据库中所有的簇。
- DBA_TAB_COLUMNS 描述所有表、视图和簇的列。
- DBA_COL_PRIVS 包括数据库中对列的所有访问权限。
- DBA_COL_COMMENTS 对表和视图中所有列的注释。
- DBA_CONSTRAINTS 在数据库中所有表的约束的定义。
- DBA_CLU_COLUMNS 将表中的列映射到簇中的列。
- DBA_CONS_COLUMNS 关于所有约束定义中列的信息。
- DBA_USERS 关于数据库中所有用户的信息。
- DBA_SYS_PRIVS 描述授予用户和角色的系统权限。
- DBA_ROLES 描述数据库中所有存在的角色。
- DBA_PROFILES 包括指派给每个预置文件（profile）的资源限制。
- DBA_ROLE_PRIVS 描述授予用户和其他角色的角色。
- DBA_TAB_PRIVS 描述对所有数据库中对象的授权。

为了使用户对系统表中出现的信息类型有所了解，我们以DAB_USERS为例进行介绍。DBA_USERS包括数据库中合法用户的相关信息，包括用户名、用户ID、加密后的口令、默认表空间、临时表空间、创建的日期和分配的预置文件。DBA_TAB_COLUMNS 有15个属性，

包括每张表的所有者、表名、列名、数据类型、数据长度、精度和数值范围等。下面的SQL查询在DBA_TABLES中查找表PRODUCT_T的拥有者。

查询15　谁是PRODUCT_T的拥有者?

```
SELECT OWNER, TABLE_NAME
 FROM DBA_TABLES
   WHERE TABLE_NAME = 'PRODUCT_T';
```

结果:

OWNER	TABLE_NAME
MPRESCOTT	PRODUCT_T

8.6 SQL-99对SQL的增强和扩展

第7章和第8章的前几节已经展示了SQL的简单性和强大功能。然而，对业务分析具有浓厚兴趣的读者想要知道：在SQL中可以使用哪些统计函数。熟悉其他程序设计语言的程序员可能会对以下问题感兴趣：如何定义变量、如何控制程序流程以及如何创建**用户自定义数据类型**（User-Defined Datatype，UDT）等。随着程序设计越来越趋向面向对象，SQL将如何调整呢？SQL-99通过提供更多的程序设计功能对SQL进行了扩展，其中的一个改进是标准化增加的统计函数。SQL将围绕面向对象概念修改相应的标准。

8.6.1 已建议加入的分析函数

在SQL-99的修正案1中加入了一系列分析函数作为对SQL语言的扩展，这些函数称为OLAP（联机分析处理）函数。其中的大多数函数已经在Oracle8*i* R2、DB2 6.2以及Teradata V2R4中实现。在2001年初，这些标准还没有被最后确定，所以这些函数的实现没有完全遵照标准。修正案解决了在数据库引擎中增加分析功能的需求，这是一个重要的发展。线性回归、相关性分析和移动平均数已经不需要把数据从数据库中取出来再计算。经过时间的考验，一旦修正案成为正式的标准，各个厂商的实现将严格按照标准进行，彼此之间也将更加一致。

建议在SQL-99中增加32个新的函数，表8-1列出了这些函数，其中包括了统计函数和数值函数。像ROW_NUMBER和RANK这样的函数将使开发人员更灵活地利用排好序的结果。对于数据库的营销或客户关系管理应用来说，有两种功能是很受欢迎的：其一是在数据库操作中考虑按某个属性排序中最高的n行，另一是按百分比对数据结果再次进行子集划分。在这些函数引入数据库引擎并加以优化后，用户可以进行更高效的数据库处理。一旦这些函数被标准化，应用程序厂商将可以基于这些函数开发应用程序，而不必在数据库之外自行创建实现这些功能的函数。

表8-1　建议在SQL-99中增加的OLAP函数

CEILING	PERCENTILE_CONT	REGR_SLOPE
CORR	PERCENT_RANK	REGR_SXX
COVAR_POP	POWER	REGR_SXY
COVAR_SAMP	RANK	REGR_SYY
CUME_DIST	RANGE	ROW_NUMBER
DENSE_RANK	REGR_AVG	SQRT
EXP	REGR_AVGX	STDDEV_POP
FLOOR	REGR_AVGY	STDDEV_SAMP
LN	REGR_COUNT	VAR_POP
MOVING_AVG	REGR_INTERCEPT	VAR_SAMP
MOVING_SUM	REGR_R2	

注：源自R.Winter所著的“The Extra Mile”，2000，*Intelligent Enterprise* 3(10)，p.63。

如果该修正案被接受，那么SQL语句将引入一个新的子句——WINDOW子句。这个子句由一列窗口定义组成，其中每一项都定义了一个窗口的名字及其规格说明。规格说明包括了分割、排序和聚合分组等操作。

下面的查询示例来自一篇支持该修正案的论文（Zemke等, 1999, p.4）。

```
SELECT SH.TERRITORY, SH.MONTH, SH.SALES,
 AVG(SH.SALES) OVER W1 AS MOVING_AVERAGE
FROM SALES_HISTORY AS SH
WINDOW W1 AS(PARTITION BY (SH.TERRITORY)
             ORDER BY(SH.MONTH ASC)
             ROWS 2 PRECEDING);
```

上例中，窗口名为W1，在FROM子句之后的WINDOW子句中定义。PARTITION子句的作用是在SALES_HISTORY中根据TERRITORY（地区）分割记录行。在每个地区分割中，记录行按月份升序排列。最后，定义聚集分组为当前行和前述两个记录行的划分结果，按ORDER BY子句定义的次序排序。这样，就返回了每个地区销售情况的移动平均数MOVING_AVERAGE。

8.6.2 程序设计能力扩展

SQL-92和更早的标准将SQL作为一个数据检索和处理语言，而不是一个应用程序语言。因此，为了创建面向业务的应用程序、过程或函数，必须将SQL和其他完全的计算性语言（如C和Java）等连接使用。SQL-99通过在Core SQL、SQL/PSM和SQL/OLB中加入新的编程功能，扩展了SQL的程序设计能力。

为了使SQL成为完全的计算性语言，应该扩展程序流程控制功能，如提供IF-THEN、FOR、WHILE语句以及循环，这包含在基本SQL规格说明的一个扩展包中。这个包称为**持久化存储模块**（Persistent Stored Module，SQL/PSM），该包中存储了创建和删除程序模块的功能。持久性（persistent）意味着代码模块在被显式删除以前将一直存储在数据库中，并随时可用于执行用户会话，其持久性正如基表一样，如果没有将其显式删除将永久存储。每个模块作为一个模式对象保存在数据库模式中。一个数据库模式可能没有程序模块，也可能拥有多个程序模块。

每个模块必须有一个名字、一个授权ID、和某一模式的关联、所使用的字符集的声明、所有在模块运行时需要的临时表声明。每个模块必须包括一个或多个SQL过程，这些过程被称为程序，每一个程序在被调用时运行一个SQL语句。每个SQL过程必须包括一个作为状态参数的SQLSTATE声明，该参数表示SQL语句是否能成功运行。

SQL/PSM直接使用SQL数据类型创建应用、过程和函数。SQL/PSM为SQL引入了过程性，这样语句将按顺序进行处理。记住，SQL本身是一种非过程性语言，并不明确指出语句的运行顺序。SQL/PSM包括了如下几个SQL控制语句:

- CASE 使用搜索条件或值表达式，根据值比较的结果或WHEN子句的值，执行不同的SQL语句。其逻辑类似于SQL CASE表达式，但以END CASE而不是END结束，没有与ELSE NULL子句的等价子句。
- IF 如果谓词为TRUE，则执行其后的SQL语句。IF语句以ENDIF结尾，并可包括ELSE和ELSEIF语句来管理不同条件下的流程控制。
- LOOP 重复运行一个语句直至满足跳出循环的条件。
- LEAVE 用于设定退出循环的条件。
- FOR 对于结果集中的每一行都运行一次的语句。
- WHILE 如果特定条件成立，就循环执行一条语句。可与LEAVE语句联合使用。

• REPEAT　与WHILE语句类似，但其循环条件在SQL语句执行之后再检验。

• ITERATE　用于重新开始执行一个循环的语句。

SQL/PSM承诺将解决基本SQL中的几处众所周知的缺点。现在还很难断定程序员更乐于使用SQL/PSM，还是继续使用宿主语言，通过嵌入式SQL或CLI调用SQL。然而，新标准提供了以下功能：

• 可以在SQL中创建过程和函数，可以接收输入和输出参数，并直接返回一个值。

• 在SQL中可以检测和处理错误，而不必通过其他语言处理错误。

• 用DECLARE语句创建变量，这些变量的作用范围是包含它们的过程、方法或函数。

• 成组传递SQL语句而不是逐条传递语句，从而提高了性能。

• 处理阻抗失配问题，即SQL处理数据的集合，而模块中的过程只处理单行的数据。

SQL/PSM还没有完全实现，所以本章也没有给出扩展语法的例子。Oracle的PL/SQL在代码模块方面与新标准有雷同之处，如BEGIN-END、LOOP、WHILE等语句。SQL/PSM还没有广泛流行，但这种局面将很快得到改观。

8.7　触发器和例程

在SQL-99颁布以前，SQL标准不支持用户自定义函数或过程。由于用户对这种功能有需求，所以商用的数据库管理系统中已经实现了这些功能。我们希望这些产品的语法能在将来变得更符合SQL-99的要求，就如同我们希望看到包括SQL/PSM标准一样。

触发器（trigger）和例程是功能很强的数据库对象，因为它们存储在数据库中并由DBMS控制。用于创建触发器和例程的代码被存储在一处并集中管理。这有利于加强数据完整性和一致性。因为它们只存储一次，所以也简化了代码的维护（Mullins, 1995）。

触发器和例程都由过程性代码块所组成。触发器代码存储在数据库中，并在触发器事件（如UPDATE）发生时自动执行。相反，例程不会自动运行，例程是必须被调用执行的代码块（参见图8-5）。

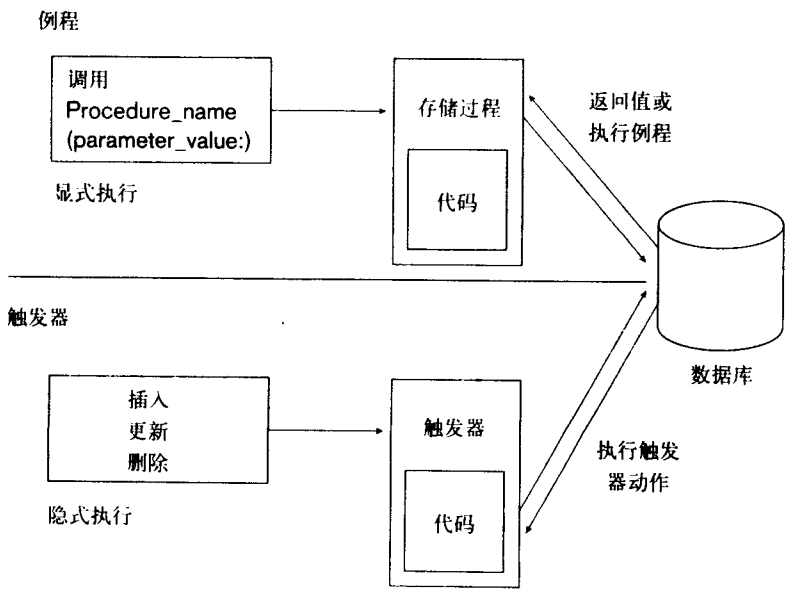


图8-5　触发器与存储过程的比较（改编自Mullins，1995）

8.7.1 触发器

触发器存储在数据库中并在数据库中执行，其执行和所有访问数据库的应用程序有关。触发器可以是级联的，在一个触发器中触发其他触发器。因此，客户端的一个请求可能会导致在服务器上进行一连串的完整性或逻辑检查，但不会引发大量的服务器和客户端之间的网络传输。触发器可以用于保证参照完整性、实施业务规则、创建审计跟踪、复制表或激活过程(Rennhackkamp, 1996)。

约束可以视为一种特殊的触发器，它也能够在数据被修改后自动执行（触发），但其准确的语法由DBMS决定，而且也不像触发器那样灵活。

触发器由三部分组成，即**事件**（event）、**条件**（condition）和**动作**（action），而这些部分也反映在触发器的代码结构中（见图8-6中的触发器语法示例）。下面是一个用PL/SQL书写的简单的触发器例子。

```
CREATE TRIGGER ORDER_ID_BIR
 BEFORE INSERT ON ORDER_T
 FOR EACH ROW
BEGIN
 SELECT ID_SEQUENCE.NEXTVAL
 INTO :NEW.ORDER_ID
 FROM DUAL;
END ORDER_ID_BIR;
```

当插入一张新的订单时，这个触发器将自动插入一个订单号。BIR是触发器命名规则的一部分，代表在插入行以前（Before Insert Row）。触发器既可以在激活触发器的语句执行之前执行，也可以在其后执行。INSERT、UPDATE或DELETE命令都可以激活触发器。触发可以在每一行被改变时触发，也可以在每一条语句运行时触发（无论有多少行被改变）。在上面描述的例子中，触发器将在每张订单插入数据库之前插入订单号。一个只在库存被更新之后才检查其库存水平的触发器，应该用AUR命名标签。上述触发器也需要一个预先定义的名为ID_SEQUENCE序列，该序列中的下一个值将被用作下一个订单号。

在使用触发器时开发人员应该谨慎。因为触发器是自动触发的，除非触发器中包括了显示给用户的信息，否则用户不会意识到触发器已经被触发了。同样，触发器还能够级联触发并导致其他触发器执行。例如，一个BEFORE UPDATE 触发器可能要求在另一张表中插入一行，如果那张表有一个BEFORE INSERT触发器，则第二个触发器也会被触发，这样会导致不希望的结果，甚至有可能产生一个不能结束的触发器循环（即死循环）。因此，使用触发器实施复杂的业务规则、创建复杂的审计日志和建立复杂的安全授权时，必须十分小心。

```
CREATE [OR REPLACE] TRIGGER trigger_name
    {BEFORE|AFTER} {INSERT|DELETE|UPDATE} ON table_name
    [FOR EACH ROW [WHEN (trigger_condition)]]
    trigger_body_here;
```

图8-6 简化的Oracle PL/SQL 触发器语法

8.7.2 例程

SQL中调用的例程可以是过程或函数。此处提到的过程和函数与其他程序设计语言中的过程和函数的使用方式是相同的。**函数**（function）只有一个返回值和输入参数。**过程**（procedure）可以有输入参数、输出参数和既是输入也是输出的参数。可以使用RDBMS的专有代码来声明和命名一个过程性代码单元，也可以调用一个宿主语言的库例程。在SQL-99之前，

SQL厂商已经开发了各自的例程，所以在使用这些产品之前，应该熟悉它们的语法和功能。其中的一些专有语言，如Oracle的PL/SQL，已得到广泛使用，并将继续有效。

SQL例程的优点包括：

- **灵活性**　例程的使用范围比约束或触发器更广，约束和触发器仅限于在数据修改的情况下使用。正如触发器比约束具有更多可选代码，例程比触发器具有更多可选代码。
- **高效性**　例程可以被精心设计并优化，使其运行速度比常规SQL语句更快。
- **共享性**　例程可以高速缓存在服务器上，供所有用户使用，用户不必重新编写。
- **适应性**　例程可以应用于整个数据库，而不仅仅属于一个应用程序。这个优点是从共享性推导出来的。

SQL-99创建过程和函数的语法如图8-7所示。可以看出，语法是很复杂的，这里也不打算深入讨论每一个子句。随后的一个简单的过程将帮助读者理解这些代码是如何工作的。

过程是一个过程性的SQL命令集，在模式中被分配了一个惟一的名字，并存储在数据库中。当需要运行过程时，就通过它的名字来调用。调用过程时，过程中的所有语句都将被执行。过程的这个特征有助于减少网络传输，因为所有的语句将同时传送，而不是逐条传送。

```
{CREATE PROCEDURE | CREATE FUNCTION} routine_name
([parameter [{,parameter} . . .]])
[RETURNS data_type result_cast]   /* for functions only */
[LANGUAGE {ADA | C | COBOL | FORTRAN | MUMPS | PASCAL | PLI | SQL}]
[PARAMETER STYLE {SQL | GENERAL}]
[SPECIFIC specific_name]
[DETERMINISTIC | NOT DETERMINISTIC]
[NO SQL | CONTAINS SQL | READS SQL DATA | MODIFIES SQL DATA]
[RETURN NULL ON NULL INPUT | CALL ON NULL INPUT]
[DYNAMIC RESULT SETS unsigned_integer]      /* for procedures only */
[STATIC DISPATCH]                           /* for functions only */
routine_body
```

图8-7　SQL-99中创建例程的语法

为了创建一个简单的设置销售价格的过程，在已有的PRODUCT_T表中加了一个新列SALE_PRICE，该列用来存放产品的售价。

```
ALTER TABLE PRODUCT_T
ADD(SALE_PRICE DECIMAL(6,2));
```

表已经改变了。

下面这个简单的过程将执行两条SQL语句。STANDARD_PRICE为400美元或价格更高的产品折扣为10%，低于400美元的产品折扣为15%。以下是Oracle代码模块，它创建和存储了名PRODUCT_LINE_SALE的过程。

```
CREATE OR REPLACE PROCEDURE PRODUCT_LINE_SALE
 AS BEGIN
  UPDATE PRODUCT_T
   SET SALE_PRICE = .90*STANDARD_PRICE
   WHERE STANDARD_PRICE >= 400;
  UPDATE PRODUCT_T
   SET SALE_PRICE = .85*STANDARD_PRICE
   WHERE STANDARD_PRICE < 400;
END;
```

如果语法正确，Oracle将返回一个注释："Procedure created"。

可以使用下面的命令在Oracle中执行这个过程。

```
SQL>EXEC PRODUCT_LINE_SALE
```

然后Oracle给出结果如下：

```
PL/SQL procedure successfully completed. Now PRODUCT_T contains:
```

PRODUCT_LINE	PRODUCT_ID	PRODUCT_DESCRIPTION	PRODUCT_FINISH	STANDARD_PRICE	SALE_PRICE
10001	1	End Table	Cherry	175	148.75
20001	2	Coffee Table	Natural Ash	200	170
20001	3	Computer Desk	Natural Ash	375	318.75
30001	4	Entertainment Center	Natural Maple	650	585
10001	5	Writers Desk	Cherry	325	276.25
20001	6	8-Drawer Dresser	White Ash	750	675
20001	7	Dining Table	Natural Ash	800	720
30001	8	Computer Desk	Walnut	250	212.5

8.8 嵌入式SQL和动态SQL

SQL-99中的SQL/PSM将具有一定的编程功能，而这些功能已经在嵌入式SQL和动态SQL中得以实现。在SQL-99之前，我们就可以使用嵌入式SQL和动态SQL来开发数据库应用。开发SQL的最初目的仅仅是为了处理数据库访问，而并不具备流程控制功能或其他创建应用程序所必需的结构。**嵌入式SQL**（embedded SQL）是指在一个使用其他语言（如C、Cobol、Ada或Java）编写的程序中包含硬编码的SQL语句的方法。动态SQL在运行时才生成明确的SQL语句。程序员编写API（应用程序接口，Application Programming Interface）实现两种语言之间的接口。

在编写本书时，使用最广泛的API是开放数据库互连（ODBC）标准，SQL-99标准中包括了SQL调用级接口（SQL/CLI），两者都是用C语言编写的，并且都基于相同的早期标准。Java数据库互连（JDBC）是一种用于从Java连接数据库的行业标准，但还不是ISO标准。

要嵌入SQL，可将SQL语句置入宿主程序的源代码中，并在其前面加上短语EXEC SQL。预编译器将把SQL语句转换为宿主语言能解释的语句，生成一个数据库请求模块（Database Request Module，DBRM）。一旦宿主文件被编译成目标代码，链接器程序将宿主语言中的DBMS调用和DBMS库链接起来。有一个通常被称为BIND的程序，它的任务是处理访问模块中的SQL语句，解析和验证每一条语句，并且产生试图优化语句执行过程的访问计划。对一些DBMS来说，访问计划不在执行时确定，而是根据数据库的当前状态而决定。访问计划以及其他必要的附加信息既可以存储在数据库中，也可以存储在一个外部代码模块中。当数据库中的内存限制了对数据库的访问时，该模块可被任何数据库实例访问和使用。嵌入式SQL比交互式SQL的效率更高，因为它能够更好地利用存储在库中的信息。一旦确定，访问计划和过程调用等对开发者而言都是透明的。

动态SQL（dynamic SQL）用于在应用程序执行时动态地生成SQL代码。大多数程序员在编写数据库应用程序时使用API，如ODBC，这样可以在所有ODBC兼容的系统上移植。动态SQL是大多数Internet应用的核心，这一点我们将在第10章详细描述。由于确切的SQL查询在应用程序执行时才确定，包括传递的参数个数、要访问的表，等等，因此，开发人员能够创建更为灵活的应用。动态SQL在SQL语句的框架将重复使用的情况下非常有用，只需在每次执行时动态改变其参数即可。

随着SQL-99的实现更完整，嵌入式SQL和动态SQL的使用将更加标准化，因为这个标准第一次建立了具备完整计算能力的SQL语言。大多数厂商已经比标准先行一步，他们独立地开发

了这些功能，因此今后几年可能是遵从SQL-99的产品和稳定的老版本产品共存的一段时期。用户应该意识到出现这种局面的可能性，做好应对的准备。

本章小结

本章在第7章介绍SQL语言的基础上，进一步讲述了等值联结、自然联结、外联结和并联结。等值联结基于参与联结的表的公共列的相同值，并返回一个结果表，表中包括了参与联结的各张表中公共列的值。自然联结返回所有查询到的结果，但公共列的值只出现一次。外联结返回联结中包括的一张表的所有值，不管在另一张表中是否有匹配值。并联结返回包含每个参与联结的表中的所有数据。

嵌套子查询，即多个SELECT语句嵌套在一个查询中，对于比较复杂的查询情况是非常有用的。相关子查询是一种特殊的子查询形式，在处理内部查询之前，需要一个来源于外部查询的值。其他的子查询先处理内部查询，返回结果给下一个外部查询，然后再处理外部查询。

其他的高级SQL技术包括嵌入式SQL、触发器和例程的使用。SQL可以包含在许多用第三代语言编写的程序中，如用COBOL、C、Fortran和Ada等编写的程序。使用嵌入式SQL可以开发更加灵活的接口，改善系统性能，以及加强数据库的安全性。在插入、更新和删除记录时，自动运行的用户自定义函数称为触发器。过程也是一种用户定义的代码模块，它通过调用来执行。

在SQL-99的一个修正案中提议加入一系列分析函数。扩展后的SQL-99将使数据库具有更加完整的计算能力，同时在持久化存储模块（SQL/PSM）等规格说明中增加流程控制功能。SQL/PSM可以直接使用SQL数据类型创建应用程序、过程和函数。SQL-99中还包括了可以调用的例程，如触发器、函数和过程。用户应该认识到这些功能已经作为数据库厂商自定义的扩展存在，并将继续存在一段时间。

动态SQL是启用Web的数据库中必不可少的一部分，本书将在第10章详细讲述这部分内容。第8章讲述了SQL的一些复杂的功能，并且引导读者掌握更复杂的扩展SQL功能开发数据库应用程序的方法。

本章复习

关键术语

相关子查询	函数	持久化存储模块（SQL/PSM）
动态SQL	联结	过程
嵌入式SQL	自然联结	触发器
等值联结	外联结	用户自定义数据类型（UDT）

复习问题

1. 给出下列术语的定义。

a. 动态SQL	b. 相关子查询
c. 嵌入式SQL	d. 过程
e. 联结	f. 等值联结
g. 自然联结	h. 外联结
i. 函数	j. 持久化存储模块

2. 匹配下列术语及定义。

_____等值联结	a. 撤销对表的修改
_____自然联结	b. 用户自定义数据类型

_____外联结	c. SQL-99扩展
_____触发器	d. 返回所有指定表的记录
_____过程	e. 保存冗余的列
_____嵌入式SQL	f. 使对表的修改永久生效
_____UDT	g. 在宿主语言中包含SQL语句的过程
_____COMMIT	h. 在运行中能够使应用程序产生SQL代码的过程
_____SQL/PSM	i. 不保存冗余的列
_____动态SQL	j. 在设定条件下执行的SQL语句集合
_____ROLLBACK	k. 被存储和命名的过程化的SQL语句集合

3. 在什么情况下可以使用外联结而不是使用自然联结?

4. 解释相关子查询的执行顺序。

5. 应用SQL解释下面的陈述: 任何可以用子查询方法编写的查询也可以用联结方法编写, 但反过来不可以。

6. SQL中COMMIT命令的作用是什么? 提交和业务事务的相关概念之间有何关系(例如顾客订单的项目或开具客户发货单)?

7. 在给数据库编写触发器时必须非常小心。写出会遇到的一些问题。

8. 解释定义触发器的代码模块的结构。

9. UNION子句在什么情况下使用?

10. 讨论SQL-99的修正案1。

11. 陈述SQL/PSM的作用。

12. 列出SQL例程的四条优点。

13. 在什么情况下使用嵌入式SQL, 什么情况下使用动态SQL?

问题和练习

问题和练习1~5基于第4章的图4-6中所描述的ERD进行回答。带有一些例子数据的3NF关系在图8-8中重复出现。对问题1~5, 画出实例图, 并在期望返回的结果数据上做标记。

1. 给下列每个查询编写SQL检索命令:

   a. 显示所有前缀为ISM的课程ID和课程名。

   b. 显示Berndt教授有资格教授的课程。

   c. 显示一个班级花名册, 包括所有选修ISM 4212的2714班的学生姓名。

2. 编写一个SQL查询回答下面的问题: 那些教师有资格教授课程ISM 3113?

3. 编写一个SQL查询回答下面的问题: 是否有教师有资格教授课程ISM 3113, 但没有资格教授课程ISM 4930?

4. 编写SQL查询回答下面的问题: 在2001年第一学期有多少学生注册了2714班? 在2001年第一学期有多少学生选修了ISM 3113?

5. 编写一个SQL查询回答下面的问题: 在2001年第一学期有哪些学生没有选修任何课程?

问题和练习6~12基于完整的松谷家具公司数据库来回答。

6. 编写一个SQL命令显示订单#1001的订单号、顾客号、订购日期和订购的产品条目。

7. 编写一个SQL命令显示订单#1001中订购的每一项产品, 包括其标准价格和订购的每项产品的总价。

8. 编写一个SQL命令计算订单#1001的总价。

9. 编写一个SQL命令找出所有还没有下订单的顾客。

STUDENT (STUDENT_ID, STUDENT_NAME)

STUDENT_ID	STUDENT_NAME
38214	Letersky
54907	Altvater
66324	Aiken
70542	Marra
...	

IS_QUALIFIED (FACULTY_ID, COURSE_ID, DATE_QUALIFIED)

FACULTY_ID	COURSE_ID	DATE_QUALIFIED
2143	ISM 3112	9/1988
2143	ISM 3113	9/1988
3467	ISM 4212	9/1995
3467	ISM 4930	9/1996
4756	ISM 3113	9/1991
4756	ISM 3112	9/1991
...		

FACULTY (FACULTY_ID, FACULTY_NAME)

FACULTY_ID	FACULTY_NAME
2143	Birkin
3487	Berndt
4756	Collins
...	

SECTION (SECTION_ID, COURSE_ID)

SECTION_ID	COURSE_ID
2712	ISM 3113
2713	ISM 3113
2714	ISM 4212
2715	ISM 4930
...	

COURSE (COURSE_ID, COURSE_NAME)

COURSE_ID	COURSE_NAME
ISM 3113	Syst Analysis
ISM 3112	Syst Design
ISM 4212	Database
ISM 4930	Networking
...	

IS_REGISTERED (STUDENT_ID, SECTION_ID, SEMESTER)

STUDENT_ID	SECTION_ID	SEMESTER
38214	2714	I-2001
54907	2714	I-2001
54907	2715	I-2001
66324	2713	I-2001
...		

图8-8 班级安排关系（没有IS_ASSIGNED）

10. 编写一个SQL查询以产生一个列表，其中包括所有的产品和其被订购的次数。

11. 编写一个SQL查询显示所有顾客及其订单的顾客号、客户名和订单号。

12. 编写一个SQL查询列出所有客户订单中订购数量超过该种产品平均订购数量的订单号和订购数量（提示：使用相关子查询）。

应用练习

1. 在Web上查找一些讨论SQL标准的链接。

2. 比较你曾使用的两个版本的SQL，如微软的Access和Oracle的SQL *Plus。指出它们在使用SQL代码方面至少五个相同之处和三个不同之处。并考虑它们的不同之处是否会导致结果的不同。

参考文献

Codd, E.F. 1970. "A Relational Model of Data for Large Relational Databases."

Communications of the ACM 13(June):77-87.

Date, C.J., and H.Darwen. 1997. *A Guide to the SQL Standard*. Reading, MA:Addison-Wesley.

Melton, J. 1997. "A Case for SQL Conformance Testing." *Database Programming & Design* 10 (7): 66-69.

Mullins, C.S. 1995. "The Procedural DBA." *Database Programming & Design* 8 (12): 40-45.

Rennhackkamp, M. 1996. "Trigger Happy." *DBMS* 9 (5): 89-91, 95.

Van der Lans, R.F. 1993. *Introduction to SQL*, 2nd ed. Workingham, England: Addison-Wesley.

Winter, R. 2000. "SQL-99's New OLAP Functions." *Intelligent Enterprise* 3 (2)(Jan.20): 62, 64-65.

Winter, R. 2000. "The Extra Mile." *Intelligent Enterprise* 3(10)(June 26): 62-64.

Zemke, F., K. Kulkarni, A. Witkowski, and B.Lyle. 1999. "Introduction to OLAP Functions" ISO/IEC JTC1/SC32 WG3: YGS. nnn ANSI NCITS H2-99-154.

进一步阅读

- American National Standards Institute. http://www.ansi.org, 1996。(关于ANSI协会和最新的国际标准方面的信息。)
- ANSI Standards Action, Vol 31, #11, 6/2/2000. p.20 American National Standards Institute, NY。也可见第7章"进一步阅读"中所列内容。
- *Welcome to ISO Online*, http://www.iso.ch 1997。(关于国际标准化组织的信息。)

Web资源

- SQL Archive, Jerry, ece, umassd.edu, University of Massachusetts, Dartmouth. ftp://jerry.ece.umassd.edu (特别地，尝试在目录#99-154处阅读Zemke的文章)。也可见第7章的Web资源中列出的站点。

项目案例：山景社区医院

使用第7章中为山景社区医院实现的SQL数据库设计来完成后面的项目问题和项目练习。

项目问题

1) 你将使用什么版本的SQL来做这个项目练习？

2) 你的这个基于SQL的DBMS是否支持动态SQL、函数、存储过程和UDT？

项目练习

1) 现在，用你的样本数据编写并测试一些查询。编写在第8章中所阐述的更复杂的查询语句：

a. 从两个或多个表中选择信息。

b. 使用子查询语法。

c. 返回一个结果表，它能用来制作一个医院报表，比如分派给每个诊疗中心的护士名单。

d. 使用UNION语句。

2) 编写查询来完成以下工作（显示你感兴趣的列）：

a. 对某个医生来说，这位医生对他所收治入院的每个病人进行了哪些治疗？

b. 对于a中的查询，还要包括那些不是由这名医生收治入院的病人。

c. 对每个病人来说，治疗该病人的医生所进行的平均治疗次数是多少？

d. 对每个护士长来说，在由该护士长负责的诊疗中心内工作的所有员工的总工作时间是多少？（以小时计。）

第9章 客户/服务器数据库环境

9.1 学习目标

学完本章后，读者应该具备以下能力：

- 定义下列这些关键术语：客户/服务器系统、文件服务器、胖客户机、数据库服务器、存储过程、三层体系结构、瘦客户机、应用分割、对称多处理（SMP）、大规模并行处理（MPP）/无共享体系结构、中间件、应用程序接口（API）、按例查询（QBE）、开放数据库互连（ODBC）标准、Visual Basic应用程序（VBA）以及事件驱动。
- 列举与其他计算方法相比，客户/服务器体系结构的若干主要优点。
- 解释应用逻辑的三个组件：数据表示服务、处理服务以及存储服务。
- 指出在不同的客户/服务器体系结构中划分这些服务的可能性范围。
- 区分文件服务器和数据库服务器，以及三层体系结构和n层体系结构。
- 描述中间件，并解释中间件如何为客户/服务器体系结构提供方便。
- 解释QBE接口的功能，它与微软Access 2000的关系以及它与SQL相比有哪些优缺点。
- 解释如何在一个使用ODBC或JDBC的客户/服务器环境中把外部数据表链接到一个应用程序。
- 解释如何在Access 2000中使用VBA来建立客户端的应用程序。

9.2 引言

客户/服务器系统（client/server system）是在一个连网的环境中运行的，它将一个应用程序在前端的客户端和后端的处理器之间进行分割。一般来说，客户端的进程要求拥有某些资源，而这些资源则是由服务器提供给客户端的。客户端和服务器可以驻留在同一台计算机上，也可以驻留在连网的不同计算机上。客户端和服务器都是智能的和可编程的，这样就可以结合两者的计算能力来设计有效和实用的应用程序。

我们很难忽略客户/服务器应用程序在过去10年里所带来的巨大影响。个人计算机技术的进步，图形用户界面（GUI）的迅速发展，计算机连网以及通信等都已经完全改变了企业使用计算系统来满足日益苛刻的应用需求的方式。电子商务要求客户端的浏览器能够存取附属在提供实时信息的数据库上的动态网页。通过支持工作组计算的网络链接起来的个人计算机也越来越常见。大型主机上的应用程序都被重新编写，以便它们能够在客户/服务器环境下运行，并充分利用个人计算机和工作站的网络在成本上的明显优势。适合特定企业环境的战略需求都能通过客户/服务器解决方案来满足，因为客户/服务器解决方案提供了灵活性、可伸缩性（即在不必重新设计的情况下升级一个系统的能力）以及可扩展性（定义新的数据类型和操作的能力）。由于企业在其运作上变得日益全球化，所以他们必须设计分布式的系统（这方面内容将在第13章阐述）。他们的计划中经常会包括客户/服务器结构。

在本章中，我们要回顾一下多用户数据库管理环境的发展，这促进了不同的数据处理的客户/服务器战略的发展。这些战略包括基于局域网的数据库管理系统，客户/服务器数据库管理系统（包括三层体系结构），并行计算体系结构，因特网和企业内部网数据库管理系统以及中间件。

数据库技术的发展趋势的是由各种新的机会和竞争压力来推动的。从某种意义上说，网络就是计算机，因为所有用户通过网络驱动器共享数据，并借助因特网存取他们所需要的信息和数据。公司的重组，包括合并和收购，使得有必要对现有的单机应用程序进行连结、整合和替换。公司缩小规模的一个后果就是，某个经理会拥有更多的控制权，因而有必要扩大其可以访问的数据的范围。应用程序的规模也不断缩减，从昂贵的大型主机变为连网的小型机和工作站，它们具有更友好的用户界面而且有时候具有更高的性价比。处理网络流量成为开发一个成功的客户/服务器应用程序的关键问题，因为在某些体系结构中网络流量可能会过高，尤其是对于那些有关键任务的应用程序放置在一个分布式环境中的组织时更是如此。在集中的和非集中的系统之间建立良好的平衡关系是当前诸多讨论的焦点，因为这些组织力求从客户/服务器和基于大型主机的数据库管理系统中取得最大的效益。

9.3　客户/服务器结构

客户/服务器环境采用局域网来支持个人计算机的网络，其中的每台计算机都拥有自己的存储设备，它们也能共享隶属于这个局域网的公共设备（比如一个硬盘或者打印机）和软件（比如一个数据库管理系统）。局域网上的每台PC和工作站之间的距离一般都在100英尺以内，整个网络线缆的长度不应该超过1英里，至少应该将一台PC指定为文件服务器，在该文件服务器中存储共享的数据库。一个数据库管理系统的局域网模块会添加并发存取控制，可能还会有额外的安全特性以及查询或翻译队列管理，以使能够支持一个共享数据库的多个用户进行并发存取。

几个已经演化发展的客户/服务器体系结构可以通过查看应用逻辑组件在客户端和服务器之间的分布来做出区分。有三个应用逻辑的组件（参见图9-1）。第一个组件是输入/输出（I/O）组件或者表示逻辑组件。这个组件负责在用户的屏幕或者其他输出设备上对数据进行格式化或进行表示，并负责管理用户从键盘或其他输入设备上的输入。第二个组件是处理组件。它控制数据处理逻辑、业务规则逻辑以及数据管理逻辑。数据处理逻辑包括数据确认和处理错误识别之类的动作。在数据库管理系统层次未能编码的业务规则可以在处理组件中进行编码。数据管理逻辑则确定处理事务或查询所必需的数据。第三个组件是存储组件，它负责从与应用程序相连的物理存储设备中取出数据或向其中存储数据。一个数据库管理系统的动作就在存储组件逻辑中发生。



图9-1　应用程序逻辑组件

9.3.1　文件服务器体系结构

第一个开发出的客户/服务器结构就是文件服务器。在一个基本的文件服务器环境（参见图9-2）中，所有的数据操纵都在需要数据的工作站上发生。客户端处理表示逻辑、处理逻辑以及绝大部分存储逻辑（那一部分与某个数据库管理系统相联系）。一个或多个文件服务器连在局域网上。**文件服务器**（file server）是管理文件操作的一种设备，它由每个连接到局域网上的客户端PC所共享。这些文件服务器可作为每一台客户端的PC的一个额外的硬盘。例如，你的个人电脑可能会识别出一个逻辑F驱动器，它实际上是存储在局域网上的某个文件服务器的一个磁盘卷。个人电脑上的程序通过常见的路径指定方法（包括这个驱动器和任何相关的目录以及文件名）查找这个驱动器上的文件。有了文件服务器后，每台客

户端的个人计算机都被称作是一个**胖客户端**（fat client），即绝大部分处理发生在客户端而不是发生在服务器的机器。

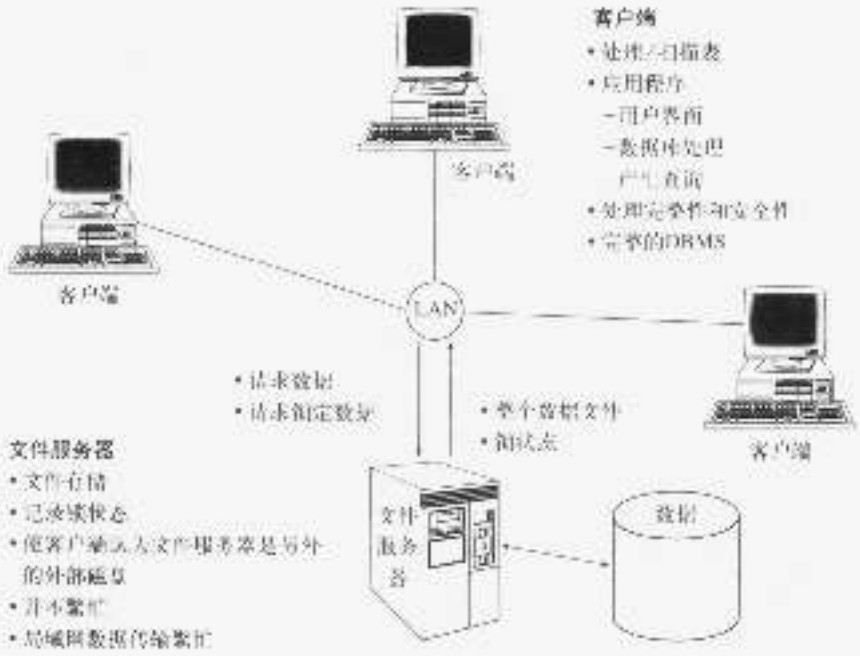


图9-2 文件服务器模式

在一个文件服务器环境中，每台客户端PC被授权使用数据库管理系统（当某个数据库应用程序在那台PC上运行时）。因此，只有一个数据库，但却有许多同时运行的该数据库管理系统的拷贝，每一个活动的PC上都有一个拷贝。文件服务器体系结构的主要特征是，所有的数据操纵都是在客户端的PC上执行的，而不是在文件服务器上执行。文件服务器只是作为一个共享的数据存储设备而存在。文件服务器上的软件将访问需求进行排队，这相当于每台客户端PC上的应用程序利用该台PC上的数据库管理系统的拷贝来处理所有的数据管理功能。例如，在这种环境中，数据安全性检查以及文件和记录锁定都是由客户端PC发起的。

9.3.2 文件服务器的局限性

在局域网上使用文件服务器有三个局限。首先，网络上会发生大量的数据移动。例如，当松谷家具公司内一台客户端PC上的某个应用程序想要存取橡木产品时，整个Product表就要传送到客户端PC，然后在客户端进行扫描以找出所需要的记录。因此，服务器所做的工作甚少，而客户端则忙于完成大量的数据操纵，网络则不断传送着大量的数据。其结果就是，一个基于客户机的局域网会将大量的负担放在客户端的PC上来执行应当在所有客户机上执行的功能，并且会产生很高的网络传输负载。

第二，每个客户端工作站必须为一个完整版本的数据库管理系统会占用每个客户端工作站的内存。这意味着客户端PC上用于应用程序的内存就会减少。同样，增加PC上的RAM将会提高性能，因为它增加了在处理一个事务时可以驻留在PC上的总数据容量。进一步说，由于客户端工作站承担了大部分的工作，每个客户机性能必须足够强大以提供合理的响应时间。相比之下，文件服务器并不一定是很大的RAM，也不一定是一台性能非常强大的PC，因为它所承担的工作甚少。

第二，也可能是最重要的一点，每个工作站上的数据库管理系统拷贝必须保证共享数据库的完整性。此外，每个应用程序必须能够识别诸如锁这样的组件并注意启动适当的锁。因此，应用程序的编写者必须有足够的经验，能够理解在一个多用户数据库环境中可能会发生的各种微妙状况。他们必须知道应用程序如何与数据库管理系统的并发性、恢复和安全控制进行互动，而且有时必须将这样的控制编写进他们的应用程序中去。

9.3.3　数据库服务器体系结构

客户/服务器体系结构的两层方法仍然遵循了文件服务器的方法，在这个系统中，客户端工作站负责管理用户界面，包括表示逻辑、数据处理逻辑以及业务规则逻辑等，而**数据库服务器**（database server）则负责数据的存储、读取以及处理。图9-3显示了一个典型的数据库服务器体系结构。由于数据库管理系统放置在数据库服务器上，所以局域网的传输流量就减少了，因为只有那些符合特定标准的记录才会被传送到客户端工作站上，而不是将全部数据文件都传输到工作站。有人把这种集中的数据库管理系统功能称为后端功能（back-end function），而将客户端PC上的应用程序称为前端程序（front-end program）。

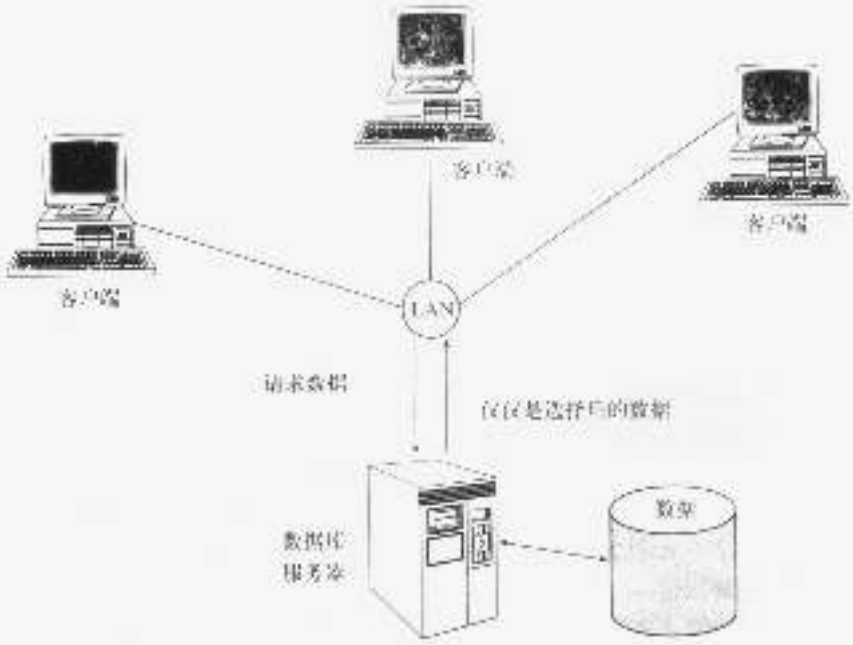


图9-3　数据库服务器体系结构（两层体系结构）

将数据库管理系统移到数据库服务器有以下几个优势。利用这种体系结构，只有数据库服务器需要拥有足够强大的处理能力来处理数据库，而数据库是存储在服务器上而不是存储在客户机上。因此，数据库服务器可以进行优化以实现最好的数据库处理性能。由于在局域网上传送的数据较少，通信负载就减少了。用户授权、完整性检查、数据词典维护以及查询和更新处理都在同一个位置（即数据库服务器）上执行。

采用文件服务器和数据库服务器体系结构的客户/服务器的项目一般是部门级的应用，仅需支持较少数量的用户。这类应用并不是关键任务的，而且绝大部分都是成功的。因为其事务量较少，直接的可用性并不是关键的，安全性也并不是最受关注的。随着各家公司致力于从客户/服务器项目中获取预期的收益，比如获得可伸缩性、灵活性以及较低的成本，他们必须为客户/服务器体系结构找到新的方法。

存储过程（stored procedure）就是包含在数据库服务器中能够实现应用逻辑的代码模块，它的使用推动了数据库服务器体系结构向着能够处理更为关键的业务应用的方向发展（Quinlan，1995）。正如Quinlan所指出的那样，存储过程具有以下优势：

- 提高已编译的SQL语句的性能。
- 随着处理任务从客户端移到服务器，网络流量不断减少。
- 如果存取的是存储过程而不是数据，而且代码转移到了服务器，不会让最终用户直接读取，那么安全性就会提高。
- 数据完整性得到提高，因为多个应用读取的是同一个存储过程。
- 存储过程造成的结果是使用统一的客户机和较胖的服务器。

但是，编写存储过程比使用Visual Basic或者Powerbuilder来开发某个应用程序要花费更多的时间。而且，存储过程的专有性质降低了它们的可移植性，会造成不重新编写这些存储过程就无法改变数据库管理系统。此外，每个客户机必须装载在该客户机上将要使用的应用程序，随着在线用户数量的增加，其性能就会下降。将某个应用程序升级就需要将每个客户端都分别进行升级。数据库服务器体系结构的这些缺陷促使三层体系结构日益流行起来。

9.4 三层体系结构

一般来说，一个**三层体系结构**（three-tier architecture）就是在前面已经提及的客户机和数据库服务器两层之外再包含一个服务器层（参见图9-4）。这样的配置也可称为n层、多层或者是增强型客户/服务器结构。三层体系结构中额外增加的服务器可以用于满足不同的目标。通常应用程序驻留在这个额外的服务器上，在这种情况下该服务器就称为应用服务器。这个额外的服务器也可以保存一个本地的数据库，而另一个服务器则保存企业数据库。以上每一种配置都可以称为三层体系结构，但每一种的功能有所不同，并适用于各种不同的情况。三层体系结构比两层结构更具优势，比如可伸缩性、灵活性、性能以及可重用性均有所提高，这使得它成为因特网应用和以网络为中心的信息系统的流行选择方案。这些优势我们将在下面的内容中详细探讨。

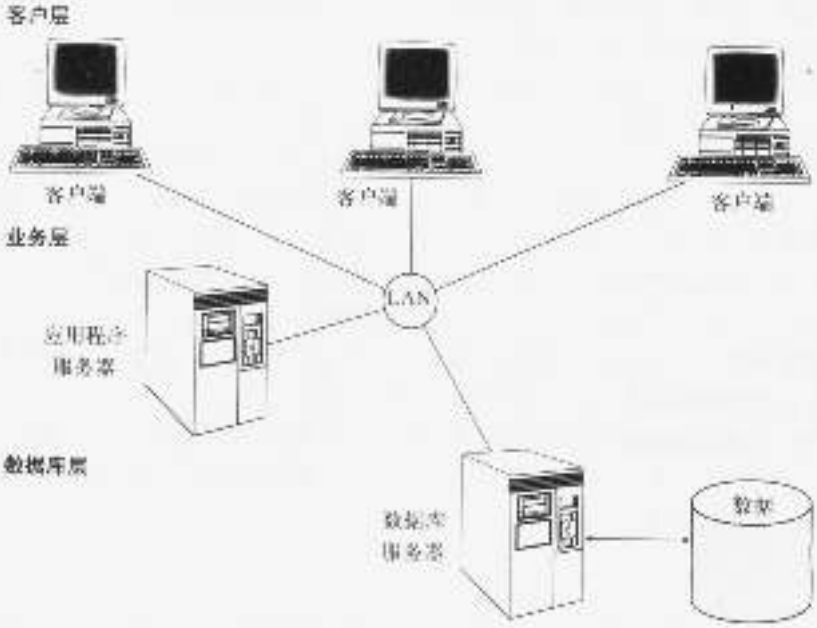


图9-4 三层体系结构

在某些三层体系结构中，大部分应用程序代码都存储在应用服务器上。这种方法所产生的好处与在两层结构中把存储过程放在数据库服务器上完全相同。使用一个应用服务器还能通过使用真正的机器代码提高性能，加强应用程序代码向其他平台的可移植性，以及减少对诸如SQL/PLUS这类专有语言的依赖性（Quinlan,1995）。在许多情况下，绝大多数的业务处理发生在应用服务器上，而不是发生在客户端工作站或者数据库服务器上，因而产生了**瘦客户机**（thin client）。在客户机上使用因特网浏览器来存取Web网页是瘦客户机体系结构的一种最好的范例。现在，驻留在某个服务器上的应用程序就在那个服务器上执行，而不用下载到客户机上的方式越来越普遍。因此，升级应用程序只要求将新版本下载到应用服务器上即可，而不必下载到客户端的工作站上。

三层体系结构能够提供以下好处（Thompson,1997）：

- **可伸缩性**　三层体系结构比两层结构具有更大的可伸缩性。例如，通过使用一个事务处理（TP）监控器来减少与服务器的连接数目，中间层能够用于减少数据库服务器上的负载，并可以增加额外的应用服务器来分布应用处理。TP监控器是在客户机和服务器之间控制数据传输的程序，以便提供一种一致的环境来完成联机事务处理（OLTP）。
- **技术上的灵活性**　尽管触发器和存储过程都需要重新进行编写，但是在三层体系结构中改变数据库管理系统的引擎更为容易。中间层甚至可以迁移到另一个不同的平台上。简化的表示服务使得实现各种所期望的界面（比如Web浏览器或是kiosk）变得更加容易。
- **减少长期成本**　在中间层使用商业现用组件或服务能够降低成本，因为能够在一个应用程序中替换各个模块，而不是替换整个应用程序。
- **使系统与业务需求更好地配合**　可以建立新的模块来支持特定的业务需求，而不用建立更为通用的、完整的应用程序。
- **可以改善客户服务**　在不同客户机上的多种界面可以存取相同的业务进程。
- **竞争优势**　通过改动小的代码模块而不是整个应用程序，可以对业务变化迅速作出反应，从而获取竞争上的优势。
- **减少风险**　此外，迅速实现小的代码模块并将它们与从供应商那里购买的代码结合起来的能力降低了从事大规模开发项目的风险。

三层体系结构和n层结构是客户/服务器方法的最新发展。向一个三层体系结构或者更复杂的环境迁移所涉及的挑战包括以下几项（Thompson,1997）：

- **较高的短期成本**　实现一个三层体系结构要求表示组件与处理组件相分离。较之实现一个两层结构而言，实现这种分离要求使用诸如C这样的第三代语言进行更多的编程工作。
- **工具和培训**　由于三层体系结构是相对较新的技术，实现它们的工具还未得到充分的开发。此外，因为培训项目尚未成熟，公司必须开发出自主实现的技巧。
- **经验**　同样，很少有人拥有构建三层体系结构的经验。
- **不兼容的标准**　对于事务处理监控器来说，还没有提出什么标准。目前尚不清楚会采用针对分布对象提出的若干具有竞争性的标准中的哪一个。
- **缺少与中间层服务相协作的最终用户工具**　目前广泛应用的通用工具，比如最终用户电子数据表和报告工具都不能通过中间层服务运行。这个问题将在本章稍后有关中间件的部分中作更详细的阐述。

9.5　分割一个应用

显然，并不存在一种最优的客户/服务器结构能够解决所有的企业问题。相反，客户/服务

器体系结构中固有的灵活性为组织提供了定制配置以适应他们特定的处理需求的可能性。图9-1描述了必须在客户端和服务器之间进行分配的计算逻辑。表示逻辑驻留在客户端，客户端是用户与系统进行交互的地方。处理逻辑可以在客户端和服务器之间进行分割，就像我们在前面讨论文件服务器、两层和三层客户/服务器体系结构的所指出的那样。存储逻辑通常驻留在数据库服务器，与数据的物理存储位置相邻。数据完整性控制活动，比如约束检查等一般也放在那里。当适当的条件得到满足时就会启动的触发器是与插入、修改、更新和删除命令相关联的。由于这些命令直接影响数据，因此触发器通常也是存储在数据库服务器上的。直接使用数据的存储过程存储在数据库服务器上。那些要使用某个查询结果的命令可以存储在一个应用服务器或是客户机上。根据所处理的业务问题的本质，也可以不遵守这些一般法则，以便实现最优的流量吞吐和性能。

应用分割（application partitioning）有助于实现这种定制。它能让开发人员有机会编写将来既可以放在客户端工作站又可以放在某个服务器上的应用程序代码，这取决于哪个位置能提供最佳的性能。没有必要包含放置被分割进程的代码，也不必编写将会与这个进程建立连接的代码。这些活动都由应用分割工具来处理。

使用面向对象编程所创建的对象非常适合于应用分割。程序员对每个对象的内容拥有很大的控制权，而且很容易将用户界面代码、业务规则以及数据分隔开来。这种分隔支持着今天正在迅速发展的n层系统。企业对因特网和电子商务解决方案的需求强有力地推动应用分割以新的方式更快地向前发展。Web应用必须是多层的和分割的。它们需要可以动态组装起来的组件，并在浏览器要求时调用，而且它们必须与不同的操作系统、用户界面和数据库兼容。有效的应用分割在Web环境下是必不可少的，这样才能在一个无法预期的分布式环境下保持良好的可维护性、数据完整性和安全性的同时达到所期望的性能。

应用程序代码可以在一个客户端工作站上开发和测试，而分割该代码和对它进行放置的决定可以在以后做出。这种能力会有助于提高开发人员的生产效率。应用模块可以在设计阶段的后期放置在某个客户机或服务器上。但是，开发人员必须了解每个进程需要如何以及在哪里运行，以便使每个进程或事务能够在不同的数据库和平台之间做到精确同步。把代码放在应用服务器还是数据库服务器在一定程度上取决于数据库管理系统的功能。例如，对于一个通过放置在数据库服务器上的存储过程和触发器来支持静态SQL（完全预编程的SQL代码）的数据库管理系统来说，如果动态SQL代码（在运行时创建的SQL代码）是放置在应用服务器上的，那么可能会面临一种性能消耗。每条动态SQL语句在被处理时会在数据库服务器上产生一种动态的绑定（或者是与数据库对象的链接）。性能的影响将取决于动态SQL语句使用的频繁程度。究竟是将处理集中于应用服务器还是数据库服务器是开发人员必须做出的一个决定，开发人员了解可以利用的硬件环境，硬件与数据库管理系统软件的交互，以及应用程序的要求。

也可以把事务处理监控器增加到客户/服务器系统中，以便提高性能。只要多个应用服务器和数据库服务器可供利用，TP监控器就能平衡工作负载，引导事务转向并不忙碌的服务器。TP监控器在分布式环境下也很有用，在分布式环境下，来自一个工作单元的分布式事务可以在异构环境下进行管理。

9.6 大型主机的作用

随着分布式客户/服务器系统以及PC计算能力的发展，大型主机的作用在过去十年里变得越来越不确定。正如上文所提到的那样，十年前驻留在大型主机系统中的关键任务的系统趋向于仍然留在大型主机系统内。关键任务程度不高、经常处于工作组层次的系统已经被人们使用

客户/服务器体系结构开发出来。客户/服务器体系结构的流行以及企业在更为复杂的环境下（因为他们的思维角度更加宽泛，更加全球化）利用分布式环境实现更有效的计算的强烈要求导致了他们产生这样的期望，即将关键任务的系统从大型主机中迁移到客户/服务器体系结构中去。

但是，将关键任务的系统从传统的大型主机遗留系统迁移到客户/服务器系统是极具挑战性的工作。成功的分布式计算取决于可用的软件分配、有效的性能管理和实际使用的系统的调整、已经确立的故障排除程序以及主动的代码管理（Hurwitz,1996）。一些企业已经发现，一旦将这些关键任务的应用程序迁移到分布式系统，对它们进行管理就变得复杂得多。在一个分布式处理环境中存在的软件分布问题如下所示：

- 确定哪些代码必须放在哪个工作站上，而哪些代码可通过某个服务器使用。
- 找出代码与其他应用程序可能会发生的冲突。
- 确保在所有位置都能得到充足的资源以处理预期的负载。

除非开发人员能够预见到可伸缩性问题并在他们开发代码时解决它们，否则将关键任务的应用程序迁移到客户/服务器的分布式环境就可能会在从规划阶段向生产过渡的过程中产生严重的问题。这就减缓了关键任务的客户/服务器系统的开发。现在，因特网的应用要求使用大量互操作的应用程序。排除这种交互中的故障是很困难的，而且许多组织并不希望进行这样的故障排除。组件库（共享应用模块的文件）使用的增加会加大对管理相互依赖性的需要，而这种相互依赖性随着组件库在应用程序之间的共享将会不可避免地扩大。

支持分布式环境是一项非常复杂的任务。了解不同位置的环境的动态状况以及创建一个分布式的计算网络是非常复杂的。管理分布式计算环境的困难促使IT经理们重新考虑推动关键任务的系统向客户/服务器解决方案的迁移。功能强大的并行处理设备（在下一节中进行探讨）正在变得越来越容易使用，因此，集中使用某个应用或数据库服务器更具吸引力。

将复杂的、关键任务的应用程序向一种使用客户/服务器结构的分布式环境迁移的困难已经令许多IT经理们感到沮丧，而且有些人已经完全放弃了客户/服务器的改革，退回到大型主机的环境中去。但是，我们期望每个组织在大型主机和客户/服务器平台、在集中的和分布的解决方案之间实现某种平衡，即完全根据其数据的性质以及这些数据的企业用户的位置来作出定制。正如Hurwitz（1996）所建议的那样，不需要经常移动的数据可以集中在一台大型主机上面。用户需要频繁存取的数据、复杂的图形以及用户界面应当尽可能放在接近用户工作站的地方。

企业资源规划（ERP）系统已经面临着诸多挑战。在20世纪90年代，诸如SAP、Baan、Oracle以及J.D. Edwards等公司开发的ERP系统说服了那些处理大量数据的公司，购买集成的财务、人力资源、制造和采购系统将会更有效率、更具性价比。转而使用一个已解决Y2K问题的应用程序套件要比重新编写几百万行现有应用程序的代码对企业来说更具有吸引力。但是，ERP系统被开发成严格控制的封闭系统。整合遗留的系统（组织的历史数据都保存在那里，而且通常是在大型主机上）是开发人员所面临的挑战之一。现在到了21世纪，人们的注意力已经转向实现更有效的供应链管理、客户关系管理（CRM）以及通过电子商务解决方案进行B2B的采购方面。ERP系统为公司的内部管理需要建立了一个集成的基础设施，但它们还必须具有电子商务的功能。经典的ERP系统被重新构建和定位，以便能在更为分布和灵活的环境中进行竞争。此时，就无法预测ERP系统将会如何进展了。

9.7　使用并行计算机体系结构

组织从关键任务的应用程序向客户/服务器体系结构迁移的尝试，包括建立能够处理比最

初的客户/服务器项目更大规模的数据库的系统。涉及超大规模数据库（VLDB）的项目已经从使用并行计算机体系结构中获益。事实上，有些项目之所以可能仅仅是因为数据库的并行处理给出了一个可以接受的快速响应时间。数据仓库已经使用并行性通过在多线程环境下的多个并不昂贵的服务器上来实现更有效的数据提取、转换和装载。

处理高事务量、复杂查询以及新的数据类型的能力在许多单处理器环境下已经证实是有问题的（DeWitt and Gray, 1992）。但是，RDBMS和SQL语言以两种方式帮助实现了一个并行的环境（Ferguson,1994）：

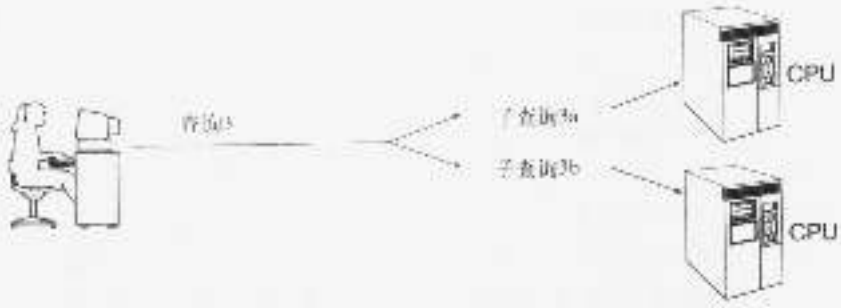
1）在大多数查询中，SQL充当一种非过程性的集合处理语言。因此，查询可以划分成许多部分，其中每一部分都能同时在不同的并行处理器上运行。

2）多重查询可以在并行处理器上以并行方式运行（参见图9-5）。

并行计算机体系结构的有效利用还包括培养对能够在一种并行服务器环境中加以利用的处理能力的认识，而不是把顺序算法改变为并行算法。我们将介绍两个可供利用的多处理器硬件体系结构，其中每一个都是为某种特定的处理而加以优化的。



a) 并行事务



b) 并行查询

图9-5 并行事务和查询（改编自Ferguson，1994）

9.7.1 多处理器硬件结构

紧密耦合的多处理器系统在所有处理器之间拥有一个共享的内存（参见图9-6）。这种体系结构通常称为**对称多处理**（Symmetric Multiprocessing，SMP），它有如下所述的优点（Ferguson,1994）。首先，操作系统的一个拷贝驻留在共享的内存中，因而可以由所有处理器共享。操作系统必须可以在这样一种共享的环境中运行，这样它就能让自己的各个部分并行地在任何一个处理器上运行，而不会使其中某个处理器比其他处理器使用得更为频繁。

第二个优点在于，与单处理器系统相比，瓶颈问题得到了减轻，因为在对称多处理环境中

的所有处理器共同分享所有的任务。但是，当进入共享内存的请求流量较大时，即便是对称多处理系统也会出现I/O瓶颈。为每个处理器配备属于它自己的高速缓存可以减轻I/O瓶颈。然后，通过重新激活正在等待的事务在开始处理它的处理器上运行，就有可能减少对共享内存的请求量。这能够更快地重新激活事务，并且对共享内存的请求更少，并且有可能利用与该事务有关的、仍然保存在特定处理器的内存高速缓冲中的可变信息（Ferguson，1994）。

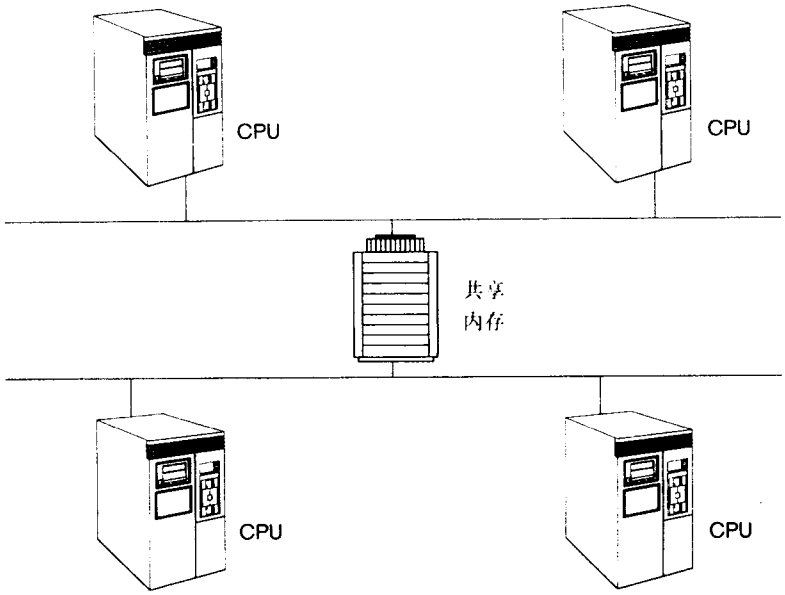


图9-6　紧密结合的多处理器体系结构（改编自Ferguson，1994）

第三，SMP体系结构对于那些数据在处理时必须保留在内存中以达到所期望的性能水平的情况来说是非常有用的。所要处理的问题包括共享内存发生争用的可能性，为了将数据取回到共享内存所需的带宽宽度，以及在单个失败点的所有内存的位置。随着将多个处理器加入到一个共享内存环境中，I/O争用问题是很可能发生的。

与SMP结构相比，松散耦合体系结构（参见图9-7，该结构又称为大规模并行处理（MPP）体系结构或无共享体系结构）可以为每个CPU设置其专用的内存。每个节点可以是一个单处理器或多处理器。因此，MPP系统要求在每个专用的内存中安装一份操作系统的拷贝。由于在各处理器之间没有什么资源需要共享，所以在SMP系统中会发生的内存争用问题则不大可能发生在MPP中，而且在单独的单元内可以增加节点（即另一个处理器和专用的内存）。这种线性的可升级性使得企业能够以较低的成本实现并行处理，并在需要时小规模地增加处理功能。这种可伸缩性在动态的业务环境中创造了一个更加容易管理的系统，而数据仓库正在这种环境中迅速地以指数速度增长。通过在多个处理器之间传递消息来实现并行性，这种系统能够分割大型的任务并将分割后的任务分布在多个处理器上。事实上，正是那些拥有大型任务的应用程序能够从分割任务和同时运行中获得好处，这些都是MPP结构最适合的，而不是那些能够从共享内存的使用中获得好处的应用程序。有很多MPP技术的开发活动，而IBM、NCR、英特尔、Oracle、DEC（康柏公司的分公司）、惠普、Encore、Silicon Graphics以及其他一些公司目前都致力于研究和开发这项技术。由于并行处理系统完全不同于传统的系统，因此考虑采用并行系统的公司有必要重新评估他们的数据库和应用程序设计、功能规划、性能管理以及可操作和调度程序。

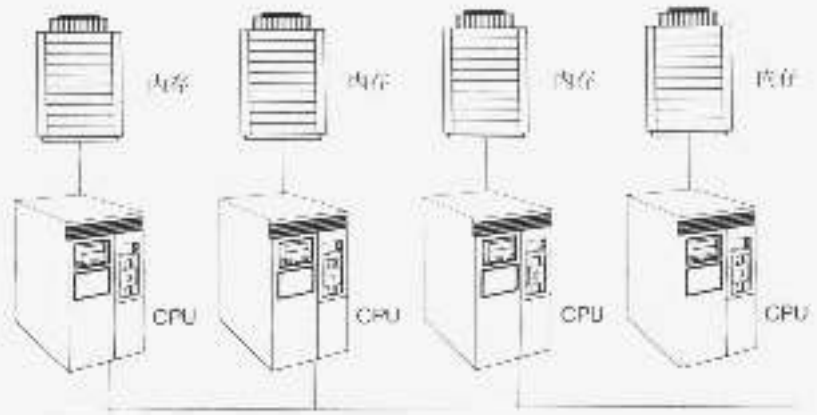


图9-7 松散耦合多处理器结构（改编自Ferguson,1994）

9.7.2 与业务有关的SMP和MPP结构的使用

许多企业都存储了大量历史数据，他们急于利用这些数据，尤其是有关其业务诸多方面的有价值信息，包括客户、产品、库存和销售等。以前，公司还并不拥有足够的计算能力来利用这些存储的数据（Rudin, 1995）。但是现在，并行处理使得企业通过开发数据仓库和数据集市来分析这些庞大的历史数据，并行处理能力的出现为开发数据仓库奠定了基础，关于数据仓库我们将在第11章再讨论。有大量销售数据需要进行分析的零售企业已经开始利用并行处理系统。

企业在采用SMP或MPP系统之前应该考虑以下一些因素（Rudin, 1995）。

- **数据分析需求** 并行系统特别适合解决涉及处理大量数据（250GB或更大）、执行复杂查询的问题或者处理同时有大量用户使用的数据。这些问题只有在它们对企业具有战略意义的情况下才应当加以解决，否则重大的硬件成本、软件开发成本以及再培训成本对组织来说都是不值得付出的。例如，零售应用程序对零售商来说具有战略性的价值。一个很好的例子就是卡特琳娜营销公司的营销一篮子分析系统，它不仅能跟踪顾客购买了哪些产品，还能跟踪顾客会同时购买哪些类别的产品。零售商可以利用这类应用程序得到的结果来安排针对相关产品的协同促销、分配货架空间和布置，以及进行合理定价。
- **成本判定** 要记住并行系统趋向于补充而不是取代现有的操作系统。企业使用并行系统是为了增加收入而不是减少成本。因此，传统的成本判定过程可能并不适合。由于许多应用程序用来改善一个企业的营销战略，因而把并行系统当作一种对营销能力的投资是恰当的。
- **传统的技术** 如果所提议的应用程序能够用传统的、现有的技术在企业内实现，那么继续使用原有的技术是比较可行的。MPP系统还远没有成熟，实现它们对任何企业都是一种挑战。因此，即使你肯定并行处理能够解决企业想要解决的问题时，仍然应当进行一次更成熟的技术可行性分析。在某种单处理器环境下进行并行处理是一种较好的问题解决方案，但在采用某种并行处理方案之前仍然应当先确定是否可以利用传统的技术来实现，在无法利用传统技术的情况下再利用并行处理。

数据仓库和数据挖掘应用程序运用联结和集合操作，处理存取大量数据和元数据的查询。这些查询涉及许多I/O操作，并有可能产生相当大的中间结果集（Frazer, 1998）。目前，还没有规模足够大的对象/关系型（O/R）数据库可当作VLDB系统（可参见附录D来学习对象/关系型数据库）。但是，随着对象/关系技术越来越成熟，人们期望用它来处理至少与目前的关系实现方案相同规模的数据库，其中一部分原因是对象/关系型数据库能够处理存储容量大得多的新

的数据类型。对象/关系型数据库将处理新的数据类型之间实例之间的复杂关系，这种关系已经超出了关系模式那种简单的横行纵列结构。并行处理方法会使这些结构都成为可能。

9.8 使用中间件

中间件经常作为各种客户/服务器应用程序的粘合剂。这个术语经常用来描述在n层体系结构中PC客户机与关系数据库之间的任何软件组件。简单地说，**中间件**（middleware）就是某种能够让一个应用程序与其他软件进行互操作，而不要求用户了解和编码实现这种互操作性的低层操作（Hurwitz, 1998）。中间件已经存在了几十年，例如IBM公司的事务处理中间件CICS，以及BEA System公司用于UNIX的Tuxedo。客户/服务器技术的出现以及现在面向Web的发展刺激了新型的商用中间件的开发。当然，开发通用的中间件，即一种能够集成和连接任何类型系统的神奇软件包是最理想的。但在目前，这样一种中间件软件包并不存在。大多数企业使用若干不同的中间件软件包，有时甚至是在一个应用程序内使用不同的软件包。

另一个需要考虑的因素是所涉及的通信究竟是同步的还是异步的。对同步系统来说，提出请求的系统实时等待对该请求的响应。一个出纳员在兑付支票前检查账户余额的联机银行系统就是一种同步系统。异步系统发送某个请求，但并不实时等待响应。相反，响应只有当它接收到时才会作出。电子邮件就是一个你比较熟悉的异步系统。

Hurwitz（1998）提出了一种包含6个类别的有用的分类系统，它将目前存在的诸多类型的中间件进行了分类。她是基于可伸缩性和可恢复性进行分类的：

- **异步远程过程调用**（RPC）　客户机要求服务，但并不等待响应。它一般会与服务器建立一种点对点的连接，并在等待响应时执行其他处理。如果失去连接，客户机必须重新建立连接并再次发送服务请求。这种类型的中间件具有较高的可伸缩性，但可恢复性较低，而且自1998年以后它已经在很大程度上被同步RPC所取代。
- **同步RPC**　一个使用同步RPC的分布式程序会调用不同计算机上可利用的服务。这种中间件使用方便，不用承担编写一个RPC通常必不可少的详细编码工作。其例子包括微软公司的Transaction Server和IBM的CICS。Java中和RPC功能相当的方法就是远程方法调用（RMI）。
- **发表/订阅**　这种类型的中间件对动作进行监控并把信息送到订阅者那里。它是异步的，也就是说客户端（即订阅者）在接收服务器的通知之间可执行其他动作。订阅者通知信息的发表者它准备接收，而当某个包含这种信息的事件发生时，信息就被发送至订阅者并由它选择是否接收这个信息。例如，你可以在电子书店中输入你所感兴趣的书名的关键字。只要该书店增加了一本书名包含你所输入的关键字的图书，有关该图书的信息就会自动地转发给你。这种类型的中间件在监控特定事件发生时需要采取行动的情况时非常有用。
- **面向消息的中间件**（MOM）　MOM同样也是一种异步软件，它发送消息（这些消息可以收集并储存起来，直到它们被执行为止），而客户机可以继续执行其他处理。像保险单应用这样的工作流应用程序通常会包含多个处理步骤，因而能从MOM中获得好处。存储请求的队列可以在日志中记录下来，因而提供了某种程度的可恢复性。
- **对象请求代理**（ORB）　这种类型的中间件使得应用程序可以在一个面向对象的系统中发送对象和请求服务。ORB跟踪每个对象的位置，并把请求路由给每个对象。目前的ORB是同步的，但异步的ORB正处于开发过程中。
- **面向SQL的数据存取**　通过网络将应用程序与数据库连接起来是通过使用面向SQL的数

据存取中间件来实现的。这种中间件具有将一般的SQL翻译成数据库特定的SQL的能力。已经开发出多数据库访问中间件的数据库供应商和公司在这部分中间件市场中占据主导地位。

尽管前述的分类法使大家对中间件功能有了基本的认识，许多产品综合利用了以上多种功能。例如，面向SQL和RPC的中间件增加了对对象的认识，而ORB则增加了事务、排队和消息服务。对象事务监控器应该很快就能面世，它承诺将结合目前由CORBA或DCOM处理的分布式对象功能和为应用程序提供事务管理并有助于减少I/O瓶颈的事务监控器功能。

在客户/服务器系统中，面向数据库的中间件为访问数据库提供了某种应用程序接口（API）。API是一个应用程序用来指导由计算机的操作系统提供的程序执行的例程集合。例如，在访问一个数据库时，一个API会调用将SQL命令从前端的客户机应用程序路由至数据库服务器的库例程。一个API可以与现有的前端软件（如某种第三代语言或定制报告生成器）协同工作，而且它可以包括自己的建立应用程序的机制。当若干程序开发工具都存在应用程序接口时，你就拥有相当的独立性在最方便的前端编程环境中开发客户机的应用程序，并仍然从某个公共的服务器数据库中获取数据。这样的中间件使得开发人员可以把一个应用程序轻而易举地链接到流行的数据库上。

开放数据库互连（ODBC）是与API相似、但基于Windows的客户/服务器应用程序。它对访问关系数据最为有用，但并不适用于访问其他类型的数据，比如ISAM文件（LaRue,1997）。即使ODBC是难以编程和实现的，但它仍然得到了人们的接受，因为它使得程序员能够与任何销售商的数据库建立链接，而不必学会该数据库使用的专用代码。关于ODBC和建立因特网数据库的连接的更详细讨论，请参见第10章。微软公司的OLE-DB通过提供对多个数据库的单个存取点为ODBC标准增加了价值（Linthicum,1997）。微软正在计划让OLE-DB成为一个通用的数据访问标准，并增加了用于数据挖掘应用和OLAP的OLE-DB。在向客户/服务器迁移的同时访问遗留数据可以通过诸如EDA/SQL这样的产品来实现，它正试图支持许多不同的操作系统、网络和数据库。

Java数据库互连（JDBC）类可用来帮助一个applet访问任何数量的数据库，而不必了解每个数据库的本质特性。JDBC为Java开发定义了一个调用层次接口（CLI），并借用了ODBC的协议。建立一种通用语言来界定不同组件之间的接口并且建立一种协调的机制将促进通用中间件的开发（Keuffel,1997）。1989年成立的对象管理组织（OMG）是一个行业联盟，它产生了为面向对象的通用中间件确立规范的公用对象请求代理体系结构（CORBA）。微软公司则开发了分布式组件对象模型（DCOM），但CORBA是一个更为强壮的规范，因为它已经用来处理许多不同的平台。两个标准之间也开始实现互操作。这样的标准在Web上尤其重要，因为Web上相互连接的平台是截然不同的。

9.9 建立客户/服务器的安全性

客户/服务器数据库计算意味着存在一个网络，它将客户/服务器组件连接起来。在这种分布式的环境中，建立数据库的安全性要比在一个集中式环境中建立安全性更为复杂（可参见第12章了解对数据库安全性的更详细的论述）。通过偷听、未经授权的连接或者对网络上传输的信息包进行未经授权的访问，网络最容易受到破坏安全性的攻击。因此，客户/服务器体系结构要比集中式系统更易受到安全性威胁。

在一个客户/服务器环境中应当采取一定的安全措施，其中不仅应包括确保所有系统安全性的通用措施，而且还应包括确保更为分布化的客户/服务器结构环境安全性的措施

(Bobrowski, 1994):

- **系统层次的口令安全**　用户名和口令通常用于在用户准备与某个多用户的客户/服务器系统进行连接时对用户进行识别和授权。安全标准应当包括口令长度、口令命名规则、口令改变的频度等等方面的指南。口令管理机制应当作为网络和操作系统的一部分。
- **数据库层次的口令安全**　大多数客户/服务器数据库管理系统拥有与系统层次的口令安全相类似的数据库层次的口令安全机制。它还可能通过由操作系统认证能力提供的认证信息。利用通过能力的管理要更容易一些，但外部获得数据存取的尝试也将更容易些，因为使用通过能力把口令的安全层从两个减少为一个。对操作系统认证的依赖不应当提倡。
- **安全的客户/服务器通信**　加密就是把可读数据（明文）转换为不可读的（密文）数据，从而有助于确保安全的客户/服务器通信。大多数客户机发送数据库用户的明文口令给数据库服务器。诸如Oracle、Sybase和Inform这样的大型关系数据库管理系统拥有安全的网络口令传输功能。对网络上传输的所有数据进行加密显然是人们期望的，但加密软件的成本是很高的。加密还会对性能造成负面影响，因为需要有时间来对数据进行加密和解密。

支持Web的数据库的客户/服务器安全性问题

使现有的数据通过因特网连接让浏览者进行访问的Web站点的大量涌现引起了新的问题，这些问题不只是在前一节所述的传统的客户/服务器安全问题。某个数据库动态地创建网页时要求能够访问数据库，而如果数据库没有得到很好的保护，它就会受到用户由于不恰当访问而造成的攻击。这是一个新的易受攻击点，而原先它可以通过严格的数据库访问授权方案或是专门的客户端访问软件来避免。

如果一个组织仅仅是需要静态的HTML页面，那么必须对存储在Web服务器上的HTML文件建立保护。一个静态Web页面是使用传统的客户/服务器工具，比如Visual Basic或者PowerBuilder创建的，因此它们的创建可以通过使用标准的数据库访问控制方法来加以控制。如果某些装载到Web服务器的HTML文件有一定的机密性，那么就将它们放置在使用操作系统安全机制保护的目录内，或者它们在目录内是可读的但并不对外发布。因此，用户必须知道准确的文件名才能访问这个机密的HTML页面。把Web服务器隔离开来并限制其可公开浏览的Web页面内容是常用的方法。机密文件可以保存在另一个服务器上，只能通过企业的内部网才能访问。此外，如果有些用户不应该访问企业内部网上存储的所有文件，那么应该利用上面所说的那些安全措施。用户认证安全机制可以用来控制对机密文件的访问。

用于动态网页创建的安全措施则有所不同。动态网页页面作为一个模板存储，并在与该页面有关的查询开始运行时从数据库插入合适和当前的数据。这意味着Web服务器必须能够访问数据库。为了正常工作，连接通常要求对数据库拥有全部的访问权。因此，对服务器采取足够的安全性措施对保护数据是至关重要的。拥有数据库连接的服务器在物理上应当是安全的，而程序和公共网关接口（CGI）脚本的执行应当是可以控制的。

对数据的访问还可以通过另外一层安全手段，即用户认证安全来加以控制。使用HTML的登录表单可以使数据库管理员确定每个用户的权限。每次会话可通过在客户端存储一段数据（即“cookie”）来进行跟踪。这些信息可以返回给服务器并提供有关登录会话的信息。必须利用CGI脚本或其他手段来执行必要的认证例程。

同样也必须建立会话安全性以确保私有的数据在会话中不会受到破坏，因为信息在网络上进行广播以供某台机器接收，因而也就有可能受到拦截。TCP/IP并不是一个非常安全的协议，

而前面所述的客户/服务器上的加密系统就是必不可少的。许多因特网系统使用一种公共密钥和秘密密钥相结合的加密方法。服务器和客户机都能用他们的私有密钥加密一个秘密密钥。经过加密的数据和他们的公共密钥一起传送，或者他们的公共密钥可以由一个提供这种密钥的Certificate Saver保存。许多开发人员通常用安全套接字层（SSL）这种标准的加密方法来对所有在一次会话中客户端与服务器之间传输的数据进行加密。以https://开头的URL就在传输中使用SSL。

其他一些安全措施（如数字签名、Kerberos服务器以及与供应商相关的安全措施）将在第12章中讨论。

9.10 客户/服务器的问题

无疑，客户/服务器体系结构的建立已经影响了数据库的计算环境。但是，我们经常会听说客户/服务器的实施失败以及用户对此大失所望。为了获得成功，客户/服务器项目必须用定义良好的技术和成本参数来解决某个特定的业务问题。某些领域应当小心地加以处理，以便提高构建一个成功的客户/服务器应用的机会（Linthicum,1996）：

- **准确地分析业务问题** 和其他计算体系结构的情况一样，开发一个可靠的应用程序设计和体系结构对新的客户/服务器系统来说是至关重要的。开发人员选取适当的技术，然后开发适合该技术的应用程序的趋势很明显是过去10年来发生的客户/服务器环境推动的。准确定义问题的范围并确定需求，然后运用该信息来选择技术是更加恰当的。
- **详细地分析结构** 确定客户/服务器体系结构的细节也是同样重要的。建立一个客户/服务器解决方案包括连接许多组件，让这些组件协同工作可能有一定的难度。客户/服务器计算一直宣称接受开放系统方法的能力是其一个极大的优点，但如果选择的异构组件很难连接就会将这个优点变成缺点。除了指定客户端工作站、服务器、网络和数据库管理系统外，分析员还应当确定网络的基础结构、中间件层以及所使用的应用开发工具。在每一个结合点，分析员都应当采取措施确保这些工具能与中间件、数据库和网络等连接起来。
- **避免工具驱动的体系结构** 如上所述，要在选择软件工具之前确定项目要求而不是先选择工具再确定项目需求。首先选择一个工具，然后运用它解决问题，这种做法的风险在于问题和工具之间的配合不会很好。在选择工具时可能是根据设计人员的主观想法来选择的，而不是根据该工具的恰当功能来选择的。
- **实现恰当的可伸缩性** 一个多层的解决方案能使客户/服务器系统进行适当的伸缩以满足不同数目用户的需求，并处理完全不同的负载。但多层解决方案显然更为昂贵，构建起来也更为困难。开发一个多层环境的工具也很有限。除非确有必要，否则架构师应当避免使用多层的解决方案。通常，多层方案在并发用户超过100位、高流量的事务处理系统或者实时处理的环境中比较适用。在规模较小的、事务量不多的环境中利用传统的两层系统经常会更有效率，尤其是用触发器和过程来管理处理时更是如此。
- **服务的恰当配置** 此外，仔细分析所要解决的业务问题在决定如何放置处理服务时是很重要的。转而利用瘦客户机和胖服务器并不总是恰当的解决方案。把应用逻辑转移到服务器从而创建一个胖服务器会影响容量，因为最终用户都要使用现在装载于服务器上的应用程序。有时，把应用程序的处理转移到客户端可以实现更好的伸缩性。胖服务器的确会趋向于减少网络负载，因为处理在接近数据的地方发生，并且胖服务器确实减少了对功能强大的客户机的需要。充分理解业务问题确实有助于架构师恰当地进行逻辑分布。

- **网络分析**　分布式系统最常见的瓶颈仍然是网络。因此，架构师千万不要忽视系统必须使用的网络带宽。如果网络不足以处理必须在客户端和服务器之间传送的信息量，那么响应时间就会受到很大影响，而系统则有可能失败。
- **注意隐蔽的成本**　客户/服务器的实现问题不只包括上面列出的分析、开发以及体系结构问题（Atre,1995）。例如，准备使用现有硬件、网络、操作系统和数据库管理系统的系统经常会受到集成这些异构组件以建立客户/服务器系统的复杂性的阻碍。培训是一项很可观的反复支出的成本，而它经常会被忽视。若环境中的组件来自不同的供应商，那么其复杂性会带来高昂的成本。

如果这些问题都得到恰当的解决，那么利用客户/服务器体系结构会得到以下一些好处（Atre, 1995）：

- 功能可以分阶段交付给最终用户。因此，在项目的第一阶段布署完毕后就可以利用相应的那部分功能。
- 客户/服务器环境中常见的图形用户界面可以使用户利用应用程序的功能。
- 客户/服务器解决方案的灵活性和可伸缩性能促进业务处理的再工程。
- 更多的处理可以在接近数据源的地方执行，因而提高了响应时间并减少了网络流量。
- 客户/服务器结构开发启用Web的应用程序，便于组织内部进行有效沟通并提高了在因特网上执行外部业务的能力。

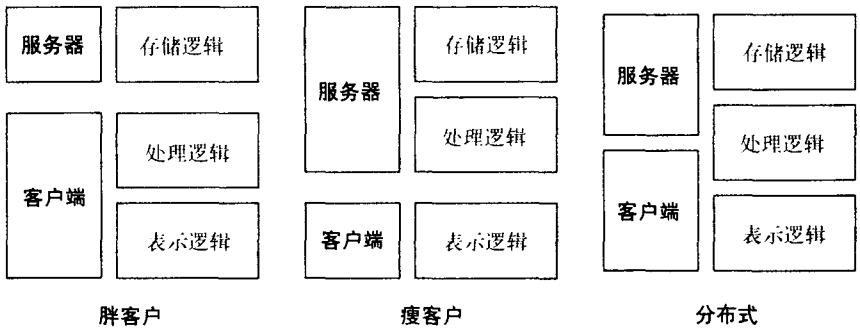
9.11　客户端应用程序的数据库存取

在本章前面，图9-1描述了必须在两层或三层客户/服务器应用环境中分布的程序逻辑组件。分割环境以创建一个两层、三层或是n层体系结构意味着必须作出如何分布处理逻辑的决定。在每一种情况下，存储逻辑（数据库引擎）由服务器处理，而表示逻辑则由客户机处理。

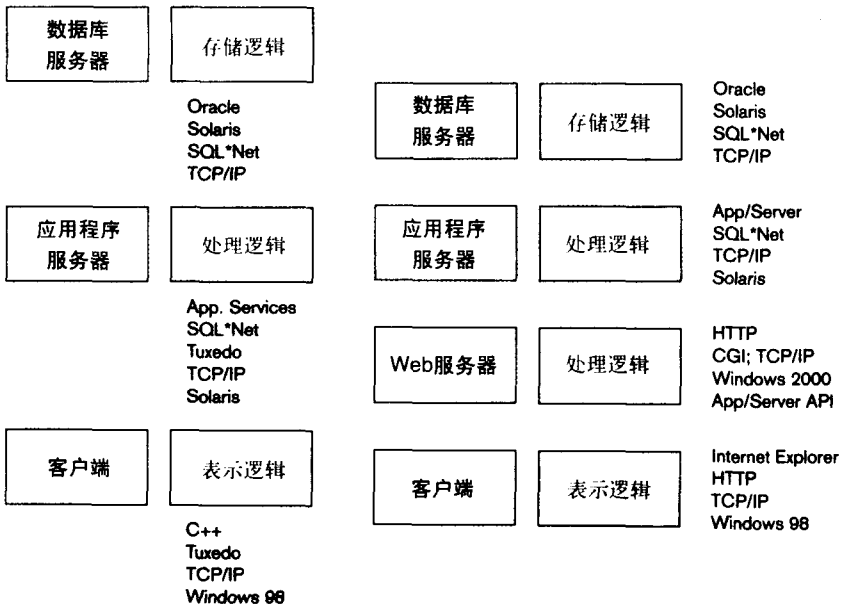
图9-8a描述了可能使用的两层系统，把处理逻辑放在客户端（创建一个胖客户机），或者放在服务器（创建一个瘦客户机），或是在服务器和客户端之间进行分割（一个分布式环境）。在三种情况中所要强调的就是处理逻辑的放置。在胖客户机情况下，应用处理完全在客户端发生；而在瘦客户机情况下，这种处理发生在服务器一端。在分布式环境的例子中，应用处理是在客户机和服务器之间进行分割的。

图9-8b显示了一种典型的三层体系结构和n层体系结构。如果需要的话，某些处理逻辑被放置在客户端。但是，启用Web的客户/服务器环境中的客户机是一种瘦客户机，它使用浏览器来处理表示逻辑。中间层一般是用可移植的语言（如C或Java）来编码的。n层结构方法的灵活性和易管理性是它得以迅速流行的主要原因，虽然管理各层之间的通信增加了其复杂性，但瑕不掩瑜。因特网和电子商务项目的快速响应、分布式和异构的环境也导致了许多n层体系结构的开发。

客户端负责表示逻辑。用户通过他们的浏览器、一个应用程序或者是一个编程界面与系统的其他部分进行交互。这些界面已经完全由图形用户界面（GUI）所主宰。存取数据库的界面也同样利用图形用户界面。虽然SQL语法仍然用来直接与数据库进行通信，但针对查询数据库的按例查询（QBE）方法也已经证明是一种流行的界面。使用的便利以及能够在诸如微软的Access这样的桌面数据库产品中进行编程的能力使许多组织使用这类软件包，作为附属于一个功能强大的后端数据库（如Oracle或Informix）的前端。例如，通过使用传递查询（它在Access中编制但却基于后端的Oracle数据库运行），许多组织大大缩短了开发时间，但又利用了更强壮和功能更强大的关系型数据库管理系统完成他们那些关键任务的应用。



a) 两层客户/服务器环境



b) n层客户/服务器环境（会存在许多种可能性，这些只是一些例子）

图9-8 常见逻辑分布

9.12 使用按例查询

按例查询（Query-by-Example，QBE）是使用最广泛的直接操纵数据库查询语言。虽然QBE不像SQL那样是一种国际标准，但它已经广泛应用了许多年，尤其是在PC关系型数据库管理系统市场应用了许多年。作为一种直接操纵的查询语言，想要对某个数据库进行查询的所有人员都能够轻松地学会它。同样，它的简单性使它成为开发原型的一种流行语言。由于有些数据库系统（比如微软的Access）能把QBE查询翻译成SQL，所以QBE至少可以用作在创建SQL代码时的第一遍扫描。产生SQL代码的基于QBE的系统随后可以用来构建访问服务器数据库的表示层或客户端层的模块。在本节，你将了解QBE的历史以及这种重要的数据库操纵语言的许多查询功能。

9.12.1 QBE的历史和重要性

尽管QBE（同SQL一样）最初是用于大型主机的数据库处理的，但它也在客户/服务器和

个人计算机数据库系统中非常流行。QBE最初是由Zloof（1977）开发，并在IBM大型主机的SQL/DS和DB2数据库系统中首次使用。第一个个人计算机上的关系型数据库管理系统（PC-RDMBS）Paradox是完全基于QBE的，它的成功促使其他产品采用QBE界面作为一种可选方案。大多数当前的系统都包括一个QBE的变化版本。

了解QBE对于理解现代数据库系统是极有必要的。这是因为QBE是一种生产率极高的编程语言。可视化的编程语言，比如Visual Basic、Visual C或者Visual Java已经改变了编写程序的方法。有关研究（例如，Greenblatt and Waxman,1978; Thomas and Gould,1975）已经表明，即使只有很少的培训，学生们发现QBE使用起来要比SQL或是某种关系代数语言容易得多。虽然这些研究已经有20多年历史，但还没有新的数据库查询语言比QBE更加易用。

QBE对于最终用户的数据库编程最为有用。不管执行什么样的数据库任务，这个可视化编程环境让不会编程的用户可以从某种角度看到数据。正如你将会看到的（以及第2章已经表明的）那样，查询是在一个监视器屏幕上以一种类似期望输出的格式完全交互地进行开发的。在大多数程序里，查询和结果是以相同的格式显示的，通常是以一种电子表格的格式显示的。

完整的数据库应用程序可以用QBE来编写，但更常见的是使用QBE进行交互查询或对一个数据库进行更新。也就是说，QBE对于特殊数据库处理特别有用。较之在QBE中开发一个完整的应用程序，更常见的是使用QBE来构建一个应用程序的原型，从而省掉了需要开发的查询。这些省掉的查询随后可以使用应用程序生成器工具，比如屏幕、表单和报表生成器以及通过增加使用与PC-RDBMS有关的数据库语言开发的代码模块来提供定制行为来增强这些查询。

9.12.2 QBE：基本知识

QBE并没有像SQL那样的官方标准（参见第8章关于SQL标准化的讨论）。因此，QBE不存在一个最小的功能集合，查询界面必须满足这个集合才能被当作某种QBE实现。但是，由于QBE是从对数据库查询语言的研究中进化而来的，而且Windows的可视化界面已经标准化了，并且它是一种可视化语言，所以所有的供应商都采用相似方法来实现QBE。

数据检索和数据修改可以通过在一个表格布局的单元内输入关键词、常量和范例数据来完成。由于数据定义存储在内部表中，所以即使是数据定义也可通过一个相似的表格布局界面来完成。在微软的Access 2000（在本章中与Oracle 8*i*一同作为例子）中，点击某个SQL按钮将会显示在构建基于QBE的查询时产生的Access SQL代码。

图9-9描述了最初的微软Access可用性层次结构。金字塔说明了Access可以用于编程能力和复杂性不同的程序。在金字塔的最底层是对象，它允许创建各种表格、查询、表单和报表而不需要具备专门的编程知识。在金字塔的上一层能够使用表达式或函数以执行诸如字段的相乘、数据的确认或者实施某种业务规则这样的简单进程。在第三层——宏中，用户可以利用预存储的Visual Basic应用程序（VBA）代码模块来自动操作他们的应用程序，而且不需要详细了解VBA。在第四层，用户可以编写他们自己的VBA代码以定制他们自己的应用程序。在最顶层，Windows API调用以诸如C、Java或Visual Basic这些语言编写的函数或动态链接库，它们可用来编写和其他程序和数据源交互的接口。

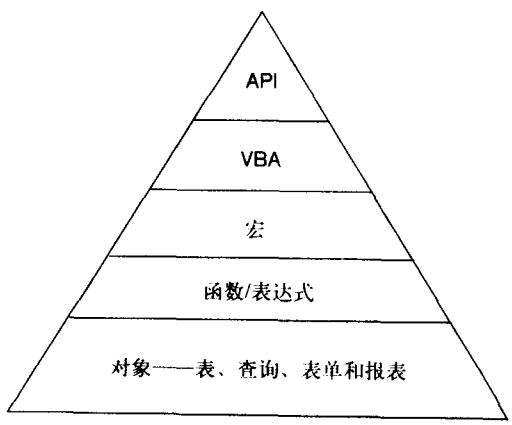


图9-9　Access可用性层次结构
（改编自Prague & Irwin，1997）

QBE提供了一种简单、可视化的方法来确定合格的查询。显示的数据也许仅限于具有期望值的某些列和记录，就像第8章使用SQL时所显示的那样。事实上，传达一个在QBE面板中所构建查询的Access SQL总是能通过点击那个SQL图标来查看。

在Access 2000中，QBE面板（参见图9-10）的上部有一部分工作空间，在工作空间中可以放置一次查询中所涉及的表或查询的数据模型表示。以前建立的联系也会自动显示出来。有时，某个查询中还需要其他的联系，而该联系也能在QBE面板中建立。QBE面板的下部称为查询设计面板或QBE网格，该部分显示一个电子表格形式的表，在那里放置了查询所需的字段以及查询所需的任何排序和限制标准。每个QBE网格中的列包含来自工作空间的一个表或查询的某个字段的有关信息。图9-10显示了一个选择查询的QBE面板，包括要求得出对每个产品在哪次订购完成的有关信息的产品和订购项表格。注意，图9-11所显示的结果被称为动态集合（dynaset）。虽然这个结果看上去像一个表，但它并不像它所基于的那个原始表那样是一个基表。相反，它是一个动态或虚拟的记录集合，而且并不保存在数据库内。图9-12显示了这个查询的Access SQL视图。

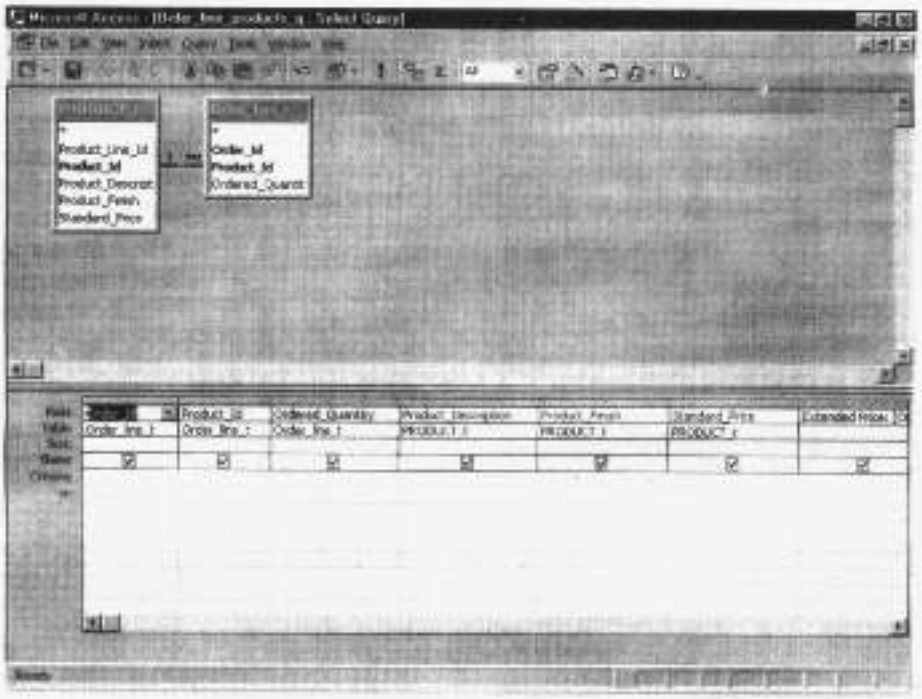


图9-10 设计模式下的MS Access 2000多表查询（松谷家具公司）

动态集合的好处在于，所使用的存储设备只要求较少的存储空间，而且当查询运行时，它会使用数据库的当前版本，即数据库会包括自上次查询以来增加或更新的任何记录。因此，查询看起来就像第8章所讨论的关系视图。如果你需要把查询结果存储进一个新的表，可以使用构造表查询。

迅速回顾一下SQL中（见第8章）的多表查询将会提醒你有必要确定哪些表是在查询中需要的（在FROM语句），并设定在查询中所需的每个表之间的等同性（在WHERE语句）。如果某人打字的速度不快，那么将这些链接在SQL中进行编码就是非常耗时的。而在MS Access 2000 QBE中，当这些联系在Relationship屏幕上明确地显示出来时，这些链接就已经建立。将

所需的表放在QBE面板中时就自动加入了这些链接。因而不必插入每个链接。这在开发查询时能够非常节省时间并能保证准确性。如图9-10所示，查询中所用的两个表已经被添加进QBE面板，而查询结果所需要的字段已经被选取并拖进QBE网格中。图9-12显示了通过点击和拖曳已经建立了多少SQL。注意，即便QBE网格并未包括用来建立两个表之间链接的PRODUCT_t表所含的主键列（Product_ID），SQL中也已经包含了正确的链接。还要注意，Access SQL与第8章所使用的Oracle SQL*Plus有所不同（你也许想写出与图9-12中的代码等价的Oracle SQL代码）。这些差别说明每个供应商都有自己的SQL。



图9-11　MS Access 2000动态集合（从图9-10得出的查询结果）（松谷家具公司）



图9-12　MS Access 2000在SQL视图下的多表查询（松谷家具公司）

9.12.3　选择合格的记录

如果你只是对某些记录感兴趣那该如何呢？图9-13显示了输入限定条件显示所需记录的基本方法。在相关联的列下面放置条件能够完成这一任务。所返回的动态集合如图9-14所示。条件可以是一个不等式，如图9-13中所示的那样。它也可以包含一个范围或等式。例如，你可以在图9-13中的Standard Price列下面输入"Between 350 and 700"，以了解哪些产品处于这个价格范围中。

运算符、函数以及表达式也可在构建查询的标准时使用。数学运算符（比如乘法和除法）

可以在数字字段上使用。关系运算符，比如相等、不等、小于能用于数字字段、日期字段和文本字段。布尔或逻辑运算符则可用于在表达式中设定条件。

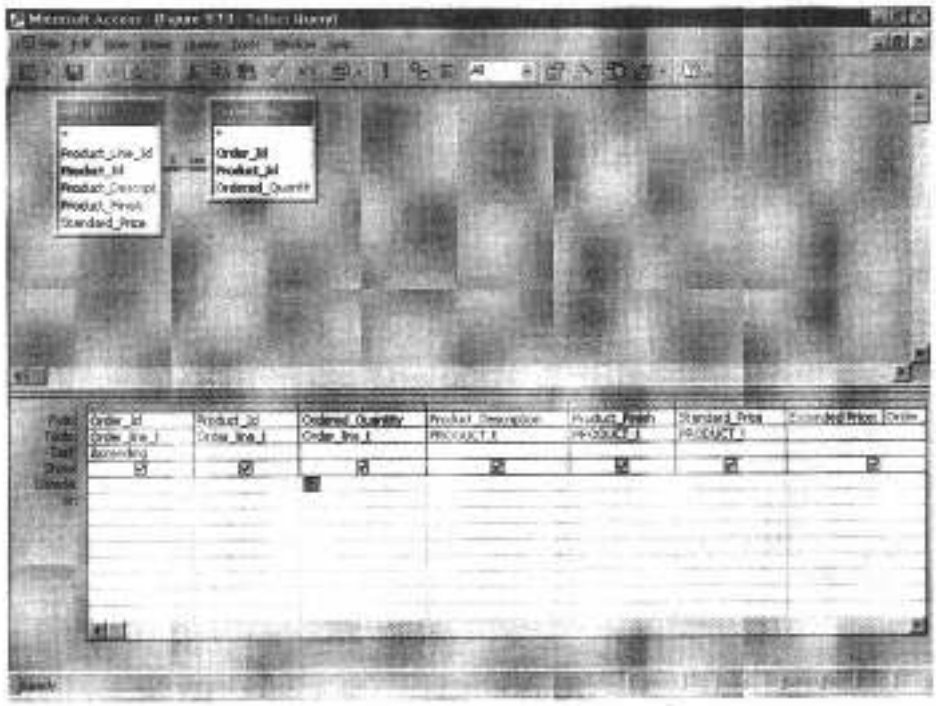


图9-13 选择对某个特定产品订购数量达到三个或超过二个的查询（松谷家具公司）



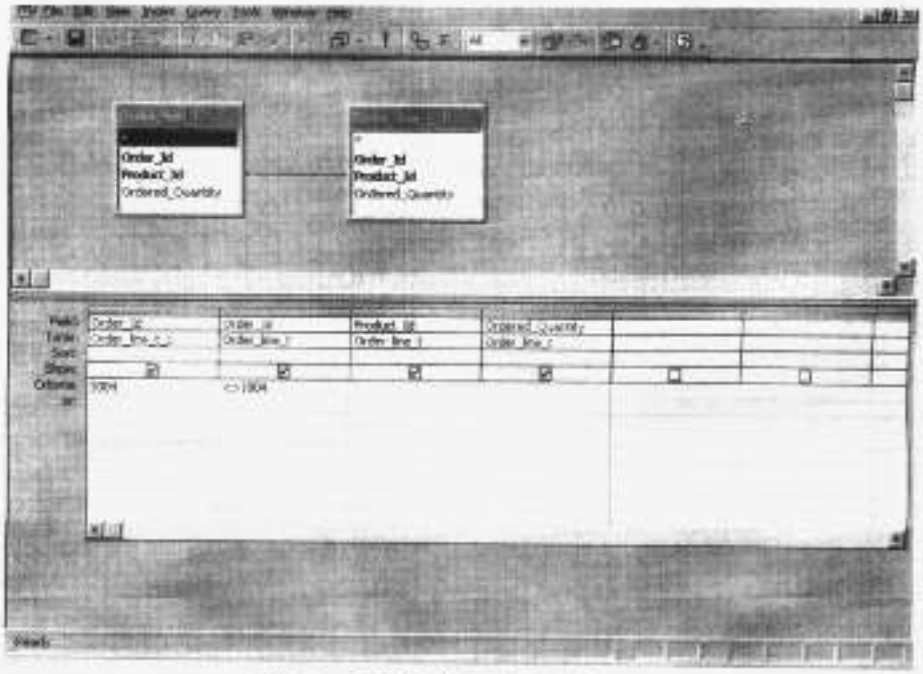
图9-14 图9-13的查询中得出的动态集合（松谷家具公司）

函数是基于函数执行的某个计算或比较而返回某个数值的代码模块。函数可以返回字符串、逻辑或数字值，这取决于该函数的性质。微软公司的Access拥有许多已经存储的常用函数可供使用，比如sum或average，或者用VBA编写一些专门的函数。表达式是由运算符控制的一个条件或一系列条件。通过插入所需的字符（比如引号，如果它们不再使用的话），Access将试图用表达式语法提供帮助。

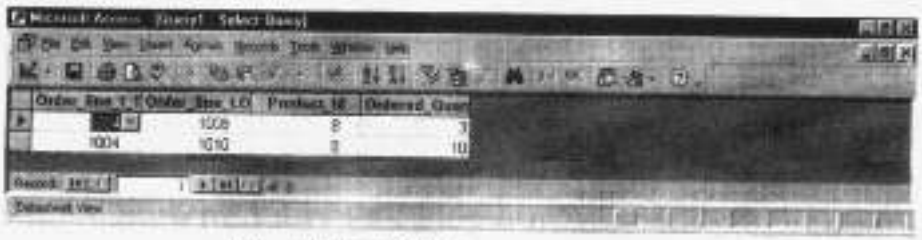
9.12.4 自联结

有些查询用QBE界面来构建要容易得多，因为它的可视化表示比输入一个复杂的语句要清楚得多。例如，当查询要求一个表和它自身联结时用QBE界面就很方便。这种类型的查询称为自联

结（self-join）或者递归联结（recursive join）。例如，假定你想要知道哪些订单会包括与1004号订单相同的产品。进行这样的查询可能是因为该订单的某个包装出了问题，而你需要找出是否其他订单也有相同的问题。图9-15a所示的是回答这个问题的Access 2000的QBE查询。一个自联结可以用以下方法实现，即在QBE面板中放置两份Order_Line_t表的拷贝，设定两个产品ID之间的某种联系（参见连接两个表中的Product_ID字段之间的联系线），并在一个表中确定1004的标准而在另一个表中确定NOT 1004的标准。它们会返回产品ID和数量值。我们可以在图9-15b中看到，仅有8号产品在其他订单中被订购。有三个8号产品在1008号订单中订购，有十个8号产品在1010号订单中订购。图9-15c显示的是SQL（它能在一个本地数据库或服务器数据库上运行）。



a) MS Access 2000来自松谷家具公司的递归联结



b) 从a中的递归联结获得的MS Access 2000动态集合



c) MS Access 2000在SQL视图下的递归联结查询（松谷家具公司）

图9-15　递归联结的例子：哪些订单拥有与1004号订单相同的产品

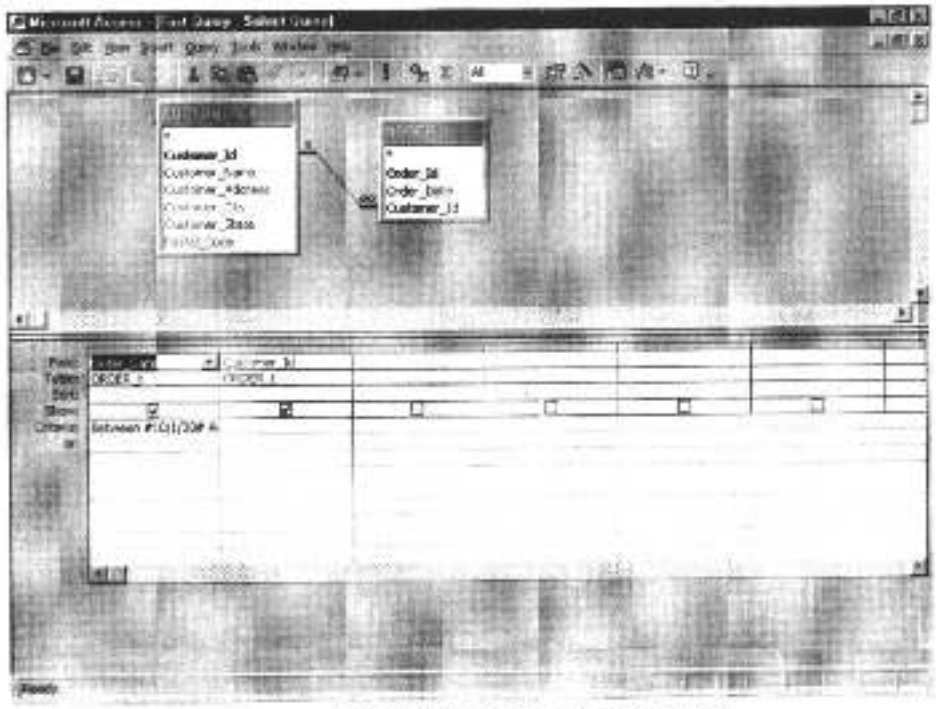
9.12.5 让一个查询基于另一个查询

有时，即使使用QBE也很难在一次查询中回答某个问题。解决这种难度较大的查询的一种方法是把一个查询分解为几个步骤，保存回答某个特定步骤所产生的查询结果，然后下一步的查询是基于所保存的查询结果而不是基表。这种方法有时与使用SQL中的子查询相似。因此，让一个查询基于另一个查询使你可以利用保存的查询所产生的动态集合来计算SUM、COUNT等的值，因而可以解决使用第8章所讨论的记录组的函数所固有的一些困难。

假定你想要找出哪些顾客在2000年10月期间没有从松谷家具公司购买产品。也就是说，你想要知道Custom_t表中的哪些顾客号没有列在2000年10月Order_t表中的记录中。图9-16a～图9-16d说明了如何通过建立一个查询然后再用第一次查询的结果构建第二个查询来回答这个问题。第一次查询（图9-16a中的QBE和图9-16b中的动态集合结果）返回的是在10月份下订单的顾客。这次查询的QBE被保存在一个称为"First Query"的查询中。第二次查询（图9-16c中的QBE和图9-16d中的动态集合结果）随后使用一个外联结来比较所有顾客的列表（表Custom_t）和那些下订单的顾客（由"First Query"所定义的虚拟表），并返回那些未能在第一次查询的动态集合中找到的顾客的动态集合（参见图9-16c中的"Is Null"限定条件）。

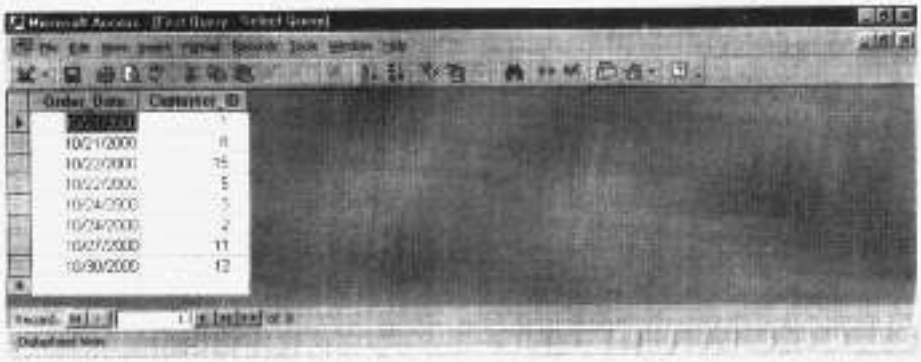
9.12.6 使用SQL传递查询

MS Access之所以成为最常用的客户/服务器应用程序中的客户端界面的原因之一是，人们可以轻松地直接传送命令给任何ODBC（开放数据库互连）数据库服务器。使用一个传递查询（它可以用ODBC数据库服务器的SQL方言而不是MS Access SQL来编写），你就能直接利用服务器的表而不用再链接到它们。这就绕过了缓软的Access Jet数据库引擎，从而使性能更好。所有SQL查询的语法检查、解释和翻译都将在服务器的数据库上进行。网络流量将会减少，因为必须在服务器和客户端之间传送的只有最初的SQL查询和所要返回的记录。

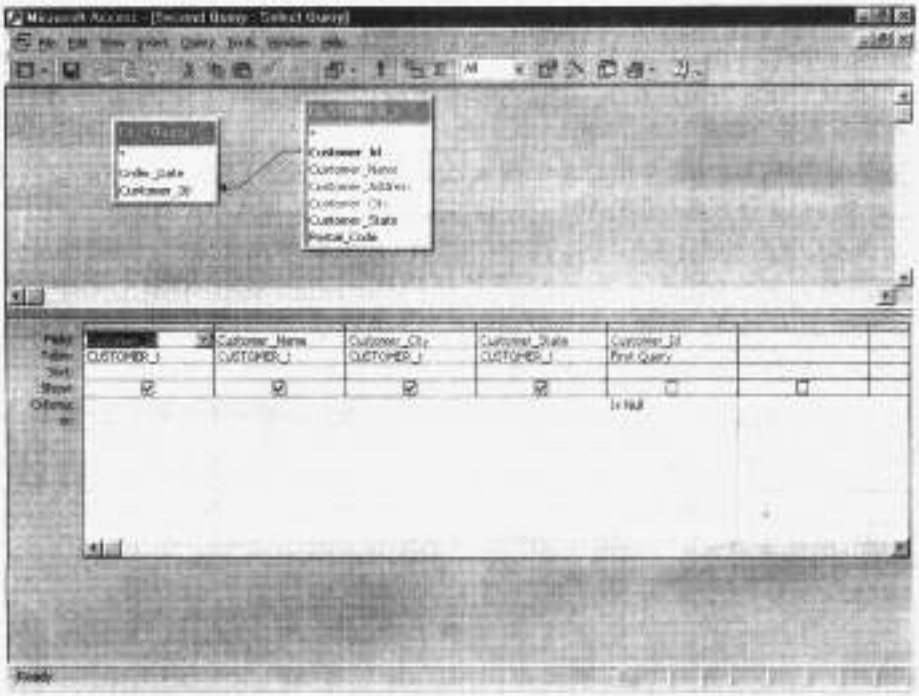


a) MS Access 2000中用于图9-16c的下一次查询的新查询

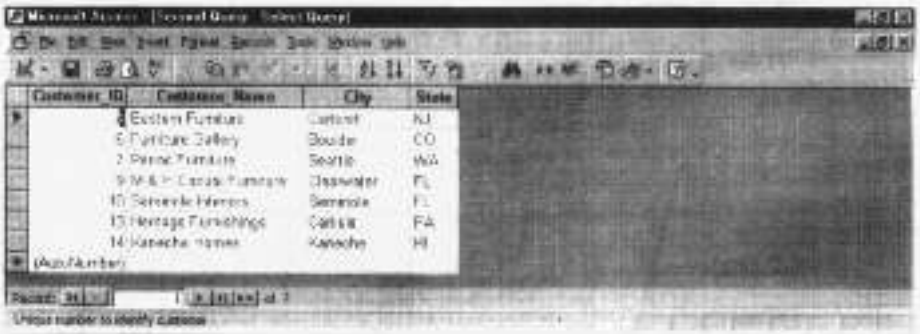
图9-16 根据另一个查询而进行的查询



b) 由图9-16a的查询返回的MS Access 2000动态集合



c) 基于图9-16a的MS Access 2000查询



d) 由图9-16c的查询返回的MS Access 2000动态集合

图9-16 （续）

MS Access传递查询可以用来检索记录、改变数据或者执行位于数据库服务器上的存储过程或触发器。人们甚至能在服务器数据库上创建新表。但要小心，不要执行会影响连接状态的操作，因为这样会造成不可预测的结果。

这种方法并不能让你把ODBC数据库的表链接到MS Access（链接表将在本章稍后部分讨论）。这意味着你不能创建一个基于这些表的可更新记录集合。用户必须熟悉ODBC数据库所使用的SQL方言。因此，尽管能获得较高的性能收益，并且有可能利用大型数据库服务器的强大功能，但用户必须了解所使用的SQL方言，而且必须准备进行更多的人工操作来实现完整的功能。

创建一个MS Access传递查询要求指定一个连接字符串，无论是作为传递查询的一个性质还是在运行查询时都是如此。图9-17显示了Oracle的SQL语法以及包含某条Oracle的SQL语句的一个MS Access 2000传递查询的性质窗口。



图9-17 带有查询性质窗口的MS Access 2000 SQL传递查询

重要的是，要注意每个关系数据库管理系统（比如Oracle、Informix或者SQL服务器）都会拥有自己关于ODBC连接字符串的语法。这个字符串必须插入性质窗口或者在查询运行时输入。例如，这里使用的Oracle连接字符串的语法是：

ODBC;DSN=*ODBC Connection Name*; UID =*User*; PWD=*Password*; SERVER=*Connection Alias from TnSNames.ora*;

9.13 使用ODBC来链接存储在数据库服务器上的外部表

开放数据库互连标准（open database connectivity standard）是在20世纪90年代初期由X/Open和SQL Access Group委员会开发出来的。它提出了关系数据库管理系统能够实现的若干层标准，因此能使任何应用程序使用一个公用的存取和处理应用编程接口（API）来对它们进行访问。这样的RDBMS是与ODBC兼容的。这个标准已经得到了广泛的认可，而最早是由微软公司在他们的产品中实现了ODBC这个标准。ODBC对于互特网应用来说也是非常重要的，因为它能允许开发出访问不同数据库产品的应用程序。为了实现这种功能，ODBC使用了第8章所介绍的ANSI标准类SQL语句，但它不能利用每个供应商赋予其引擎的扩展和特殊特性。

ODBC规范允许驱动程序遵守不同层次的规范，而这样影响了该驱动程序的功能层次。驱动程序本身在编写上的差别可能会影响所要实现的性能。每个想要拥有一个兼容ODBC标准的数据库供应商都提供一个能够安装在Windows机器中的ODBC驱动程序。因此，每个Windows应用程序都能通过恰当的驱动程序，与数据库服务器的某个版本进行通信。例如，一个MS Access应用程序可以与一个Oracle数据库服务器进行连接以执行操作。数据库的表通过ODBC链接被链接到MS Access的应用程序，并且仍然保存在Oracle数据库内。它们并没有存储到MS Access的数据库中。

你也许听说Oracle的数据库服务器通常就称为数据库服务器，但它也可以称为远程服务器、后端服务器或者SQL服务器。正如微软公司的后端服务器被称为SQL服务器一样，提及SQL服务器可能是指某一类型的服务器或是某个特定供应商生产的数据库服务器。这会让人觉得混淆不清。

必须确定5个参数后才能建立一种ODBC连接：

- 需要特定的ODBC**驱动程序**。
- 可连接的后端**服务器**名称。
- 可连接的**数据库**名称。
- 允许访问该数据库的**用户ID**。
- 用户ID的**口令**。

如果需要的话，还可能要提供额外的信息：

- 数据源的名称（DSN）。
- Windows客户端计算机名称。
- 客户端应用程序的可执行名称。

这些参数可以从不同的位置进行确定。它们可以包括在程序中，或者通过DSN确定，或者在提示时由用户提供。在程序中包括以上所有的参数使得程序能够直接与数据库连接，而不用进一步的通信。当然，有时不得不修改程序并插入新的参数值以便把该程序移植到另一个服务器或者关系数据库管理系统。在程序中包括DSN（它包含了某些参数值）能够使本地管理员放置数据库并选择某个关系数据库管理系统。用户可以在登录时提供他的用户ID和口令来使用这个应用程序。

图9-18是一个典型的ODBC体系结构的示意图。客户机应用程序请求与某个数据源建立一种连接。这个请求由微软的驱动程序管理器处理，它能确定恰当的ODBC驱动程序以供使用。记住，这些驱动程序是由供应商提供的，因此可能会有SQL服务器驱动程序、Oracle驱动程序或者是Informix驱动程序等等。初始化请求、格式确认以及ODBC请求管理也都由驱动管理器来处理。所选择的驱动程序将会处理从客户端收到的请求，并向所选择的关系数据库管理系统提交用这种特定RDBMS的SQL语法所编写的查询。创建该查询所要求的处理量取决于关系数据库管理系统被访问的能力。

同样，ODBC的一致性层次将由供应商已经提供的驱动程序的功能来确定。有三种一致性层次：核心API、第一层API和第二层API。这些层次都是由本节前面所提到的标准委员会确定的。大多数驱动程序提供核心API和第一层API，它们包括以下功能：

- 用驱动程序特有的信息与数据源建立连接的能力。
- 准备和执行SQL语句。
- 从一个结果集合检索数据。
- 提交或回滚事务。

- 检索错误和目录信息。
- 发送并接收部分结果。
- 检索有关驱动程序的信息。

第二层API在功能上的一些独有特征包括:

- 浏览潜在的数据源和连接的能力。
- 取回本地的（后端方言版本）SQL。
- 调用某个事务库。
- 处理和显示一个可滚动的游标。

应用程序在调用驱动程序时确定可供利用的API的层次。如果需要第二层API，而应用程序只是第一层的API，然后执行序列就能以一种可控的方式中止，这样就不会发生数据破坏。有些应用程序是既可在第一层执行也可在第二层执行的，这取决于所使用的特定关系数据库管理系统驱动程序的功能。

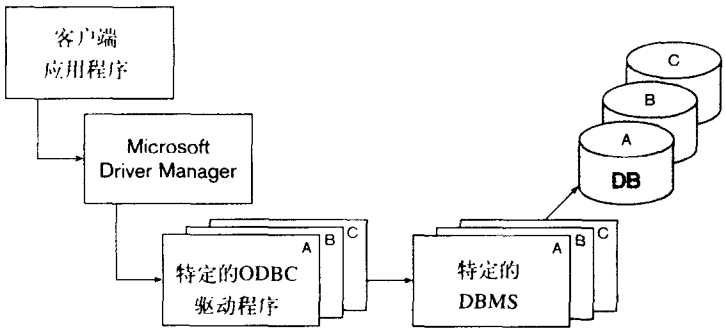


图9-18 开放数据库互连（ODBC）结构

9.14 使用JDBC来链接存储在数据库服务器上的外部表

Java数据库互连（JDBC）API使得Java程序能够执行SQL语句并与数据库服务器进行连接。JDBC类似于ODBC，但它是专门为Java应用程序设计的。ODBC是与语言无关的。Java是一种用于客户/服务器计算的优秀语言，因为它是面向网络的、安全性很强并且可移植。Oracle公司已经采用了Java，而且似乎Oracle的专用语言PL/SQL将会被Java所取代，以便提供所需要的SQL所不具备的编程功能来构建数据库应用程序。

JDBC标准在概念上与微软公司的ODBC相似。基于X/Open的SQL调用层次接口，JDBC包含两个主要的层。一层是JDBC API，它支持从一个Java应用程序到JDBC驱动程序管理器的通信。另一层是JDBC驱动程序API，它支持从JDBC驱动程序管理器直接到JDBC驱动程序，以及到网络驱动程序和基于ODBC的驱动程序的通信。

图9-19包含了一个使用JDBC来存取兼容JDBC的数据库所必需的代码的简单示例。注意，这些代码用四个方框标出，以强调所执行的不同任务。

首先，导入9个预定义的Java包。每个驱动程序为虚拟类提供了类实现，比如java.sql.Connection、java.sql.Statement、java.sql.PreparedStatement以及java.sql.CallableStatement。这些虚拟类描述了特定于每个所连接的数据库产品的API。

图9-19中的例子是与一个Oracle数据库连接，因此下一个代码框包括用于该例的表的非常简单的结构信息，以及可以用来插入下一个主键值的主键序列和触发器。在这种情况下你并不

需要知道触发器的代码，这里的代码只供那些感兴趣的人阅读。

下一步，初始化动作发生，并且诸如用户ID、口令、驱动程序类、驱动类型、数据库连接字符串等等的值都被确定。最大的代码框显示了如何与数据库建立连接（使用buildConnection()），并把传递值插入temp_demo表。

```
//package.com.iteamsolutions.eis.tools.sql.sample;    预定义的Java包
importjava.sql.CallableStatement;
importjava.sql.Connection;
importjava.sql.Driver;
importjava.sql.DriverManager;
importjava.sql.PreparedStatement;
importjava.sql.ResultSet;
importjava.sql.Statement;
importjava.sql.SQLException;
importjava.sql.Types;
/**
*这个类用于简单说明如何通过JDBC来存取兼容JDBC的数据库

Oracle中的表结构、主键序列以及触发器

*The database structure listed below will be used for all examples: <br>
* <pre>
*
TABLE temp_demo
*Name Null?   Type
*
*ID      NOT NULL NUMBER
*NAME          VARCHAR2(20)
*
*Sequence used to populate the primary key
*SEQUENCE temp_demo_seq
*
*—Trigger used to populate the primary key
*
*CREATE OR REPLACE TRIGGER temp_demo_trig
*BEFORE INSERT ON temp_demo
*FOR EACH ROW
*BEGIN
*—This will allow the key to be autopopulated
*—could be retrieved using the RETURNING
*—clause of the INSERT statement
*—IF(:NEW.id IS NULL)
*THEN
*   SELECT temp_demo_seq.nextval
*   INTO :NEW.id
*   FROM dual;
*END IF;
*END;
* </pre>
* <br>
```

图9-19　使用JDBC来访问一个兼容JDBC的数据库

```
                                    初始化
public class OracleSqlTest
{
  /**常量，用于检索表temp_demo中的id列*/
  public static final String ID = "id";

  /**常量，用于检索表temp_demo中的name列*/
  public static final String NAME = "name";

  /**数据库账户的名字*/
  private String NAME = "username";

  /**用户口令*/
  private String m_password = "userpassword";

  /**用于连接数据库的Driver类*/
  private String m_driver = "oracle.jdbc.driver.OracleDriver";

  /**使用的驱动程序类型*/
  private String m_drivertype = "jdbc:oracle:thin:@";

  /**要连接的数据库*/
  private String m_database = "servername:1521:databasename";

  /**插入一条新记录到表temp_demo中*/
  private String m_insert = "INSERT INTO temp_demo (id, name) values (?,?)";

  /**插入一条新记录到temp_demo中*/
  private String m_insertReturning = "BEGIN\n" +
                                     "INSERT INTO temp_demo (name) values (?)\n" +
                                     "RETURNING id INTO ?; \n" +
                                     "END;";

  /**该类完成所有工作所要使用的连接*/
  private Connection m_conn = null;

  /**
  默认的构造器——可以对它进行修改以支持上面提到的那些值、这些参数
  *也可以从一个性质文件中提取出来
  *
  * @throws Exception - any failure is a terminal condition
  */
  public OracleSqlTest()
    throws Exception
  {
    m_conn = buildConnection();
  }

  /**
  * 插入被传递的值到表temp_demo中

  * @param id - number to be used as the primary key in the record
  * @param name - Name to be stored in the name column
  */
  public void insert(int id, String name)
    throws SQLException
  {
    PreparedStatement stmt = null;

    try
     {
       //建立preparedStatement、preparedStatement允许将
       //所传递的String中的值用"?"来代替
       stmt =m_conn.prepareStatement(m_insert);
```

图9-19 （续）

```
    //将实际的值与参数"?"绑定，第一个参数"?"对应第一个值，
    //依此类推
    stmt.setInt(1, id);
    stmt.setString(2, name);
    //exec the statement
    stmt.execute();
  }
  catch (SQLException e)
  {
    throw e;
  }
  finally
  {
    //保证所有的资源被释放，否则由于打开太多的游标
    //会导致数据库产生异常
    if(stmt! = null)
      stmt,close();
  }
}
```

为访问Oracle数据库建立连接对象

```
/**
*为访问Oracle数据库建立连接对象;
*/
private Connection buildConnection()
  throws SQLException
{
  //应该确保与数据库对应的驱动程序被装载，产生的任何异常将被捕
  //获并重新产生一个SQLException用于被客户端捕获
  //
  try
  {
    DriverManager.registerDriver
                ((Driver) Class.forName(m_driver).newInstance());
  }
        catch (InstantiationException e)
  {
    throw new SQLException("Unable to instantiate the Oracle Driver");
  }
        catch (IllegalAccessException e)
  {
    throw new SQLException("IllegalAccessException thrown when attempting to
    load the Oracle Driver class");
  }
    catch (ClassNotFoundException e)
  {
    throw new SQLException("Oracle Driver class not found");
  }
 //代码执行到这里表示驱动程序已经被成功注册了，现在可以返回一个Connection
        return (DriverManager.getConnection (m_drivertype + m_database,
                                m_username,
                                m_password));
  }
```

图9-19　(续)

最后一个代码框包括了建立Connection对象所必需的代码，以便访问Oracle数据库，其中包括如果由于这样或那样的原因没有建立连接时会显示的错误消息。这个JDBC例子的一个更完整的版本可在本书的Web站点找到。它包括了另一种插入主键值的方法，而且还包括了创建一个新的记录以及修改、更新或删除记录的代码。

9.15 在客户端应用程序中使用VBA

Access 2000允许初学的用户创建一个应用程序的原型，其中包括数据库设计、菜单、表单和报表。但是，使用Access的宏所能实现的功能有局限性，开发人员在尝试包含所有期望的功能时会受到阻碍。尽管有些功能要求使用宏并且不能通过VBA来实现，但更多的功能可以通过使用与Access 2000相配套的VBA来实现或增强。如果对Access 2000有了基本的了解，就会了解Access 2000用户为什么需要学习VBA（Smith and Sussman，1997）：

- **复杂的功能**只有通过使用VBA才能实现。显示错误消息，把不应该点击的按钮变灰等等都要求利用VBA编码。
- **错误处理**只能通过使用VBA才能实现。在一个已经完成的应用程序中依赖宏是很危险的，因为如果一个宏崩溃的话，用户是没有办法进行恢复的。
- **更快速的执行**只有在应用程序使用VBA模块而不是宏时才能实现。代码的执行速度要比宏更快；应用程序越是复杂，这种优势就会越明显。
- **维护**会更加简便，因为VBA模块是与表单和报表一起存储的。宏则是与它们相关的表单和报表分开存储的。现在，把应用程序移植到另一个数据库将会更加方便，因为VBA模块将与它们的表单和报表一起迁移（前面说过，VBA模块是与表单和报表存储在一起的）。
- **OLE自动化**可以更加完整地得到使用。
- **更多的程序控制**可以通过使用VBA来实现。宏不能将一个变量作为参数传递给另一个变量，而且不能方便地控制动作序列。
- **阅读**VBA代码要比读宏参数容易，因为你能在Full Module View中看到完整的模块，它是用颜色编码的文本。

必须创建一个自动执行的宏来打开某个数据库，而且宏还必须用来捕获应用程序中的某些击键的操作。除此之外，把一个用宏编写的原型转换为应用程序实用版本的VBA模块将会产生具有更多功能的更为强壮的应用程序。

Windows操作系统和Windows应用程序都是**事件驱动**（event-driven）的。在Access 2000应用程序中发生的所有操作都是对Access 2000所侦测到的某个事件（比如一次鼠标点击）的响应。一个事件发生，它被侦测到，然后就产生对该事件的响应。因此，Access 2000并不是过程化的，即下一个事件可以来自界面屏幕上的任何地方，或者来自任何击键操作。在一个事件驱动环境中的编程包含对象的创建和对其属性的修改，这样它们就会对每个影响该对象的事件作出符合程序员愿望的响应。

图9-20显示了当显示一条新记录时让Product Entry Form的NEXT按钮不可用（变灰）的VBA代码。表单的记录来自Product_t表，而Product_ID则是该表的主键。当前记录的Product_ID值是用Me.Product_ID来确定的。如果当前记录没有主键值，它必定是一个新的记录，而NEXT按钮应当禁用。由于禁用具有焦点的按钮会导致一个运行时错误，因此必须先将焦点转移到其他地方。焦点可以转移到PREVIOUS按钮，然后禁用NEXT按钮。如果当前记录并不是新记录，那么NEXT按钮仍然是有效的。

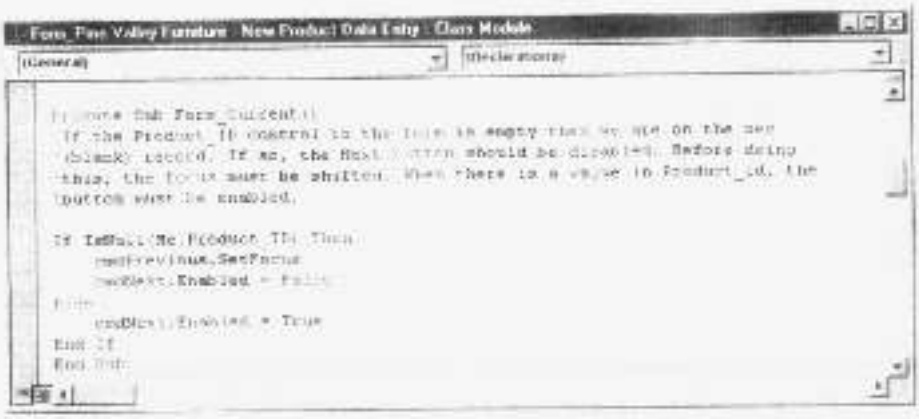


图9-20　VBA模块样本

本章小结

客户/服务器体系结构已经为企业提供了许多机会，以便让他们的计算机系统更好地满足他们的业务需求。在客户/服务器和大型主机数据库管理系统之间建立适当的平衡是当前讨论的焦点。客户/服务器体系结构在提供网络应用程序（包括动态数据访问）方面占据主导地位。本章讨论了几种客户/服务器体系结构，其中包括由文件服务器管理文件操作并由附属于局域网上的每个客户端PC共享的文件服务器结构。文件服务器产生了极大的网络负载，因为在每个客户机上都要有一个完整版本的数据库管理系统，并且要求进行复杂的编程才能管理共享数据库的完整性。

另一种方法，即数据库服务器体系结构，则让客户端管理用户界面而数据库服务器管理数据库的存储和访问。这种体系结构减少了网络流量，降低了对每个客户端的计算能力的要求，并把用户授权、完整性检查、数据字典维护以及查询和更新处理都集中在数据库服务器上。

三层体系结构，即除了在客户端和数据库服务器两层之外再增加额外服务器，允许应用程序代码存储在额外的服务器上。这种体系结构可以在额外的服务器上执行业务处理，从而形成一个瘦客户机。三层体系结构的优势是具有良好的可伸缩性、技术上的灵活性、较低的长期成本，系统与业务需求的更好配合，改进客户服务，提高竞争优势和降低风险。但是，较高的短期成本，复杂的工具和培训，有经验人员的短缺，不兼容的标准以及最终用户工具的缺乏都是日常使用三层体系结构或n层体系结构所面临的问题。

应用分割可以把编写好的部分应用程序代码分配给客户端或服务器分区，以便实现更好的性能和互操作性。应用程序开发人员的生产效率会随着应用分割的使用而得到提高，但开发人员必须深入了解每个流程才能正确地分配代码。

虽然与客户/服务器体系结构的广泛采用有关的大肆宣传使一些人已经预言大型主机会遭到淘汰，但关键任务的应用程序仍趋向于保留在大型主机上。把这些复杂的应用程序转移到分布式客户/服务器环境并不是轻而易举的。并行处理解决方案的应用使得集中使用某个应用程序或数据库服务器更具吸引力。

由于SQL是一种非过程化的集合处理语言，所以已经证明它适合用于并行处理的环境。我们讨论了两种类型的并行处理。对称多处理（SMP）是一个紧密耦合的多处理器系统，所有处理器之间共享公共的内存，当对共享内存存在极多的内存请求时，SMP系统可能会出现I/O瓶颈问题。

松散耦合体系结构（MPP）为每个CPU设置其专用的内存。这种类型的体系结构通常称为**无共享**体系结构。由于在MPP处理器之间没有什么资源共享，所以在SMP系统中可能发生的内存争用问题不复存在，而且有可能在某个单元内增加节点，从而使MPP结构极具可伸缩性。

中间件是允许一个应用程序与其他程序进行互操作而不要求用户了解和编码实现这种互操作性所需的低层代码的某种类型的软件。根据可伸缩性和可恢复性，我们把中间件分为6种类别。这些中间件类型包括异步的远程过程调用（RPC）、发表/订阅、面向消息的中间件（MOM）、对象请求代理（ORB）、面向SQL的数据访问中间件以及同步RPC。面向数据库的中间件的最新发展还包括开放数据库互连（ODBC）、Java数据库互连（JDBC）以及微软的OLE-DB。

把数据库与Web连接，这样浏览器可以通过下订单、访问更新的定价信息等等与Web站点实现交互，这些内容在近几年都得到了人们的关注。Web正在改变数据的分布模式，随着浏览器界面的使用，应用逻辑会迁移到更集中化的服务器上。

安全问题在一个客户/服务器环境中要比在集中式环境中更为复杂，因为除了确保客户端工作站和服务器的安全以外，还必须确保网络安全。在一个客户/服务器环境中包含的安全措施包括系统层次的口令安全、数据库层次的口令安全以及安全的客户/服务器通信。

应当得到解决以便提高建立一个成功的客户/服务器应用程序机会的客户/服务器问题包括：准确的业务问题分析、详细的体系结构分析、避免工具驱动的体系结构、实现恰当的可伸缩性、服务的恰当放置、充分的网络分析以及了解潜在的隐含成本。

转而使用一个客户/服务器环境可以获得的好处包括可以分阶段地交付功能、获得较大的灵活性、可伸缩性、减少网络流量以及开发启用Web的应用程序。

大多数客户/服务器体系结构的表示服务使用按例查询（QBE）界面。QBE提供了一个通常用于查询开发的可视化编程环境。QBE并不存在什么标准，因此不同产品的QBE界面有所不同，但大多数界面会利用将结果布置在电子表格中的一种数据模型方法。微软公司Access 2000的QBE界面就广泛使用了这种电子表格布局，其中包括表和查询定义以及建立数据联系等。

这种不够强壮的关系数据库软件包已经因为它的图形用户界面和QBE界面而流行开来。例如，微软Access 2000使用传递查询来直接向任何兼容开放数据库互连（ODBC）的数据库服务器传送命令。如果数据库兼容ODBC或兼容JDBC，则数据库的表可以链接到一个MS Access 2000的应用程序接口。必须确定若干个参数以便建立一个ODBC或JDBC的连接。这些参数包括所需的特定ODBC或JDBC驱动程序、后端数据库服务器的名称、数据库的名称以及数据库用户的ID和口令。对于ODBC标准有不同的一致性层次，它能让大多数应用程序与期望数据库能至少实现某种连接。

与Access 2000相关联的编程语言是VBA。使用Access 2000的宏（即VBA代码的预存储模块）对构建原型很有作用，但建立一个强壮的应用程序则要求把这些宏转换为VBA模块。VBA还能用来实现更加复杂的功能、处理错误、实现更快速度的执行、更方便的维护、更多的OLE自动处理以及更多的程序控制。

本章复习

关键术语

应用分割	应用程序接口（API）	客户/服务器体系结构
数据库服务器	事件驱动	胖客户机
文件服务器	大规模并行处理（MPP）	中间件
开放数据库互连（ODBC）标准	按例查询（QBE）	存储过程

对称多处理（SMP）　　瘦客户机　　三层体系结构
Visual Basic应用程序（VBA）

复习问题

1. 定义下列术语。
   a. 应用分割　　b. 应用程序接口（API）
   c. 客户/服务器体系结构　　d. 胖客户机
   e. 文件服务器　　f. 中间件
   g. 存储过程　　h. 三层体系结构
   i. 事件驱动　　j. QBE
   k. VBA　　l. JDBC
2. 将下列术语及其定义匹配起来。

_____客户/服务器体系结构	a. 一个负责处理应用逻辑和表示逻辑的客户机
_____应用程序接口（API）	b. 一个用来处理某应用程序的表示层和若干业务逻辑的PC
_____胖客户机	c. 可以提高互操作性、减少程序员编码工作的软件
_____数据库服务器	d. 侦测并对事件作出响应
_____文件服务器	e. 若干处理器共享公共内存的体系结构
_____中间件	f. 负责数据库存储和访问
_____三层体系结构	g. 将应用程序逻辑组件进行分布的系统
_____对称多处理（SMP）	h. 一种包含直接操纵和面向可视化的编程语言
_____大规模并行处理（MPP）	i. 负责管理文件操作；由所有附属的客户端共享
_____瘦客户机	j. 无共享体系结构
_____事件驱动	k. 促进前端程序和后端数据库服务器通信的软件
_____传递查询	l. 使用一种后端RDBMS语法的查询
_____QBE	m. 三层的客户/服务器配置

3. 列出与其他计算方法相比，客户/服务器结构的几个主要优势。
4. 比较下列术语：
   a. 对称多处理（SMP）；无共享体系结构（MPP）
   b. 文件服务器；数据库服务器；三层体系结构
   c. 客户/服务器计算；大型主机计算
   d. 胖客户机；瘦客户机
   e. 按例查询；线性模式界面
5. 描述文件服务器的局限性。
6. 描述数据库服务器的优势和不足。
7. 描述三层体系结构或n层体系结构的优势和不足。
8. 应用程序分割如何帮助开发人员根据某个企业的情况定制一个应用程序？
9. 描述6种类型的中间件。
10. Web如何改变数据的分布模式？
11. 解释QBE与SQL相比的长处和不足之处。
12. ODBC规范的目的是什么？
13. 在建立一个启用Web的数据库应用程序时会出现哪些安全问题？

14. 解释Access 2000与VBA之间的关系。

问题和练习

1. 如果要求你准备一份报告，就处理用于所有分支机构的一个新的客户应用系统的各种可能客户/服务器解决方案作出评估。你将会评估哪些业务特性和哪些技术特性？为什么？

2. 你认为在引入一个新的客户/服务器体系结构时哪些管理问题比较重要？

3. 讨论在一个客户/服务器数据库系统中应当建立哪些不同的安全层次。

4. Web如何影响客户/服务器数据库系统？

5. 为什么ODBC很重要？还开发了哪些其他的连接标准？

6. 从历史来看，哪些类型的应用程序可很快迁移到客户/服务器数据库系统？哪些类型系统的迁移速度较慢？为什么？你认为未来在平衡客户/服务器数据库系统和大型主机数据库系统时会发生什么？

7. 用于连接数据库系统的中间件的优点和不足是什么？

问题和练习8～12要根据松谷家具公司数据库进行回答。这些问题要利用QBE界面来完成。

8. 显示订购产品3或产品8的数量超过四件的订单号、顾客号、产品号、订购日期以及订购数量。

9. 显示所有顾客订单的顾客号、姓名和订单号，并在这个列表中显示数据库中没有下订单的顾客。

10. 编写一个查询显示每个产品以及每个产品的订购数量，包括从未被订购的产品。

11. 显示一个所有产品以及每个产品订购次数的列表。

12. 显示所有顾客订单数量大于该产品平均订购数量的订单号、产品号和订购数量。提示：这可能需要多次查询。

应用练习

1. 调查你的大学所使用的计算体系结构。了解你的大学的计算体系结构的发展历史，并确定这所大学经过了怎样一条道路才达到目前的配置状况。有些大学起步很早，它们拥有大型主机环境，而其他一些大学则在出现个人电脑时才起步。你能否说明你所在大学最初的计算环境是如何影响目前的计算环境的？

2. 我们缩小一下规模，调查你所在大学某个系的计算体系结构。尝试说明目前的系统是如何满足该系的信息处理需求的。

3. 采访本地的某个在其他地方拥有分公司的企业的系统专家。确定该组织的计算体系结构，并找出其目前的系统是如何满足部门的信息处理需求的。

4. 找出三个拥有附属于该站点的交互式数据库的Web站点。评估这些站点的功能，并讨论交互式数据库系统如何影响其功能。如果你不能确定从何处着手，可以先利用以下网址：http://www.amazon.com来分析。

5. 找出你所在地区的两家相似的小企业，但它们使用不同的计算机系统来保存他们的记录。比较两种系统的功能和易用性。

6. 咨询你所熟悉的某个企业的一位数据库分析师或系统开发员。调查该企业个人计算机和服务器数据库系统的使用情况。在选择一个数据库平台时会考虑什么因素？该企业是否使用一种混合的平台，即基于客户端的应用程序模块是用某种语言（比如QBE）编写，而服务器应用程序模块则用另一种语言（比如SQL）编写？为什么选择这种结构？

参考文献

Anderson, G., and B. Armstrong. 1995. "Client/Server: Where Are We Really?" *Health*

Management Technology 16:6 (May): 34, 36, 38,40,44.

Atre, S. 1995. "The Hidden Costs of Client/Server." *DBMS* 8:7 (June): 71, 72, 74.

Bobrowski, S. 1994. "Database Security in a Client/Server World." *DBMS* 7: 10 (October): 48-50, 54, 58.

DeWitt, D., and J. Gray. 1992. "Parallel Database Systems: The Future of High Performance Database Systems." *Communications of the ACM* 35:6 (June): 85-98.

Ferguson, M. 1994. "Parallel Database the Shape of Things to Come." *Database Programming & Design* 7: 10 (October): 32-44.

Frazer, W. D. 1998. "Object/Relational Grows Up." *Database Programming & Design* 11:1 (January): 22-28.

Hurwitz, J. 1996. " Managing Complexity." *DBMS* 9:5 (May): 12, 80.

Hurwitz, J. 1998. " Sorting Out Middleware." *DBMS* 11: 1 (January): 10-12.

Keuffel, W. 1997. " CORBA Masterminds Object Management." *DBMS* 10:3 (March): 42-50, 71.

Greenblatt, D., and J. Waxman. 1978." A Study of Three Database Query Languages." In *Database: Improving Usability and Responsiveness*, ed. B. Schneiderman. New York: Academic Press.

LaRue, M. 1997. "Database Doorways." *Database Programming & Design* 10:12 (December): 62-67.

Linthicum, D. S. 1996. "Client/Server Collapse." *DBMS* 9:13 (December): 24, 26, 28.

Linthicum, D. S. 1997. "Next-Generation Middleware." *DBMS* 10:10 (October): 69-78.

Prague, C. and Irwin, M. 1997. *Access 97 Bible*. Foster City, CA: IDG Books.

Quinlan, T. 1995. "The Second Generation of Client/Server." *Database Programming & Design* 8:5 (May): 31-39.

Rudin, K. 1995. "The Practical Side of Going Parallel." *Database Programming &Design* 8:12 (December): 26-33.

Schneiderman, B. 1983. "Direct Manipulation: A Step Beyond Programming Languages." *IEEE Computer* 16:57-69.

Smith, R. and Sussman, D. 1997. *Beginning Access97 VBA Programming*. Birmingham, UK: Wrox Press.

Thomas, J.C., and J.D. Gould. 1975. "A Psychological Study of Query by Example." *Proceedings of National Computer Conference*. New York: AFIPS Press.

Thompson, C. 1997. "Committing to Three-Tier Architecture." *Database Programming & Design* 10:8 (August): 26-33.

Zloof, M. M. 1977. "Query-by-Example: A Data Base Language." *IBM Systems Journal* 16(4): 324-43.

Web资源

- http://www.orafaq.com/faqodbc.htm Oracle ODBC Connectivity FAQ, Frank Naudé负责，Oracle Underground FAQ的一部分(http://www.ibi.co.za/frank/faq.htm)。
- http://www.javaworld.com/javaworld/jw-05-1996/jw-05-shah.html Integrating Databases

with Java via JDBC, Rawn Shah著, *Internet Systems*, 1997年4月。

- http://www.javacoffeebreak.com/articles/jdbc Getting Started with JDBC, David Reilly著, 1996, ITWorld.com, Inc。
- http://mindprod.com/jdbc.html An extensive list of JDBC interface packages, Roedy Green著, 2001年2月22日更新。Canadian Mind Products。
- http://www.melbpc.org.au/pcupdate/9908/9908article8.htm Database Security: Systems and Methodologies for Identifying and Protecting Weak Spots in Your Web-Enabled Database—Before someone Else Does, Dan Rahmel著, *Internet Systems*, 1997年4月。
- http://www.microsoft.com/technet/ecommerce/MSF3Sec.asp Three-Tier Security in an E-Commerce Environment, Craig Clayton著, 2000年7月28日更新。
- http://www.dbmsmag.com/9805d14.html The Middleware Muddle, David Ritter著, *DBMS*, May 1998。

项目案例：山景社区医院

项目描述

在第2章结尾，我们了解到山景社区医院特别研究小组正在开发一项长期业务计划。规划小组的Heller先生、Lopez先生、Jefferson博士以及一位顾问正在考虑山景社区医院未来的技术需求。目前，病人们必须越过一个包括诸多健康计划、管理人员、医生和诊所的迷宫，就像图9-21所示的迷宫一样。他们想要为山景社区医院设计一个能够集成所有这些数据（来自健康计划、医生和医院系统的各种数据）的系统，这样就能提供准确的实时信息。

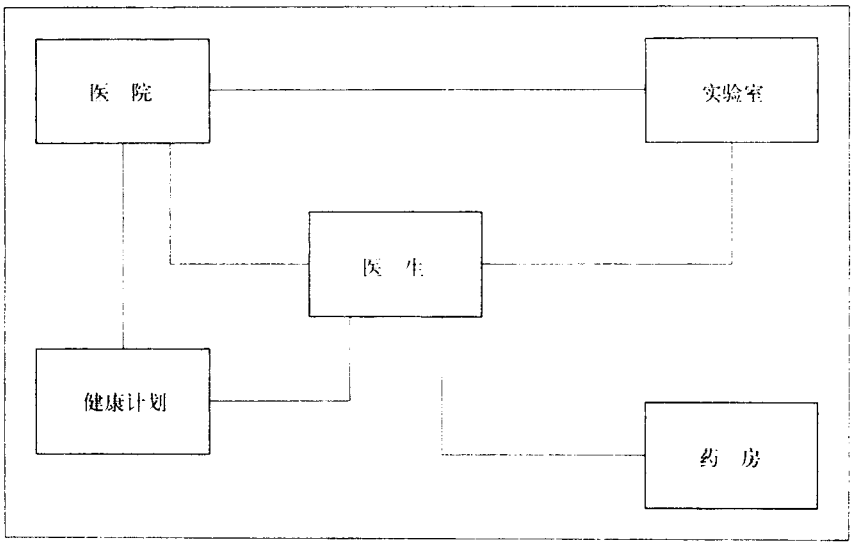


图9-21　山景社区医院目前的技术库（改编自Anderson and Armstrong,1995）

该小组听说许多医院正把他们的关键任务的企业级系统迁移出遗留的大型主机系统，并移植到新的客户/服务器系统中。他们已经听说这类系统能够提供更大的灵活性、集成性和功能。他们想知道这些系统是否能够集成山景社区医院现有的病人账户、医疗管理和保险系统来提高效率和支持改善的医院系统。顾问已经为医院草拟了一份可能的客户/服务器分布式体系结构图，如图9-22所示。为了实现向客户/服务器体系结构的迁移，该小组考虑了如下所述的三种不同战略。

一种方法是购买已有的客户/服务器应用程序软件包，并改编它们以适合山景社区医院的使用。采用这种方法可以使山景社区医院现有的管理信息系统员工能尽快熟悉客户/服务器环境的关键部分，比如网络管理、PC配置管理以及分布式系统管理，而不必在他们已有的工作之外再管理一个成熟的大规模系统开发项目。但是，他们考察过的大多数软件包都要求使用专用的中间件来实现客户/服务器连接。这家医院希望建议的服务器能够继续与遗留系统建立连接，但他们不能肯定有多少程序可供使用以便让他们迁移到希望的环境。

规划小组考虑的另一种方法是定制开发。有些大型医院已经对他们新的客户/服务器系统实施了定制开发。这项任务的风险很大，但较大的医院已经承担了这种风险，以便能在他们需要时得到数据和信息。山景社区医院的规划小组正在考虑承担这样一个项目的风险和由此必需增加MIS员工的需要。

他们考虑的最后一种方法是在现有的系统基础之上构建客户/服务器应用程序。工作流应

用程序可用来促进前端界面的建立。随着该小组逐步实施这个长期的计划，连接一个数据仓库的需要也变得越来越明显，而且该小组现在想要规划基础设施，以便轻松地建立数据仓库。

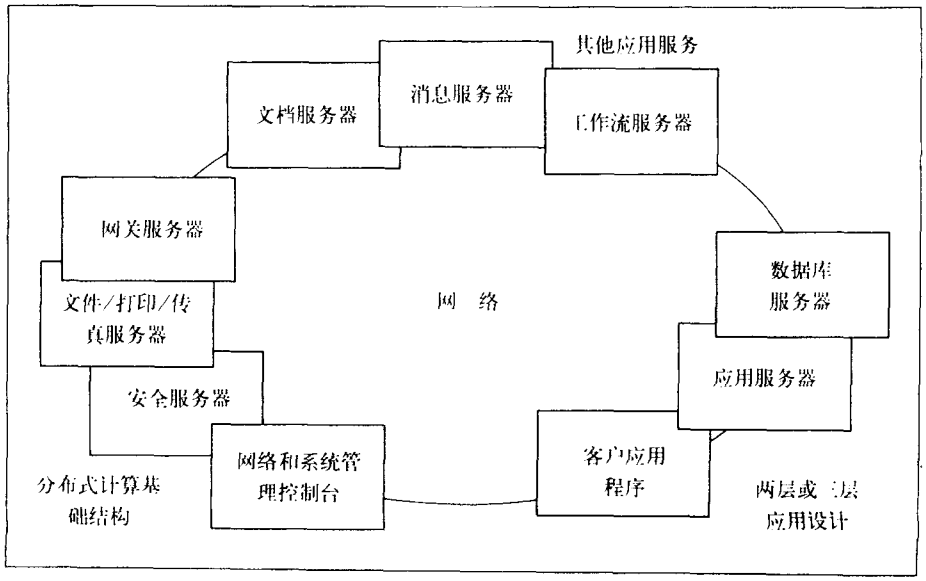


图9-22 可能的客户/服务器分布式体系结构（改编自Anderson and Armstrong,1995）

当然，山景社区医院也可以采用一种基于Web的系统。这样该诊疗中心可以扩大其会员和病人服务的范围，扩大到公司、学校和家庭。医院的会员可以检查他们的保险索赔状态，给服务代表发送消息并查看保险范围。病人通过访问预约时间安排可以自行确定预约时间。该小组的成员们不能肯定这些新服务会在该地区产生什么样的影响，但他们预期其他某些医院将会很快采用并实施这些举措。他们目前正在评估他们的其他选择方案，但在完成进一步的研究之前（即第10章结束之前）会考虑这个最新的选择。

项目问题

1) 为什么山景社区医院的规划小组认为创建他们自己的定制客户/服务器应用程序是很冒风险的？

2) 如果山景社区医院将要迁移到一个客户/服务器环境，那么处理和提供信息会有什么样的可能性发生？

3) 如果山景社区医院决定采用并实施商业现用的客户/服务器应用程序，他们必须规划什么样的活动？选择其中一个活动，比如病人记账系统，然后考虑在选择、安装和使用这样一个系统时会包括哪些内容？

4) 你认为第三种方法，即把客户/服务器应用程序建立在现有系统之上的优势和不足是什么？

5) 你认为山景社区医院在未来两年里不改变他们目前的系统会有哪些优势和不足？

项目练习

概括说明你认为以下四种迁移到某个客户/服务器环境的方法各有什么优缺点：定制商业现用系统（COTS），定制系统开发，在现有基础上建立，以及暂时什么也不做。

第10章　因特网数据库环境

10.1　学习目标

学完本章后，读者应该具备以下能力：

- 简明定义下列关键术语：万维网、电子商务、浏览器、电子业务、B2C、B2B、防火墙、代理服务器、万维网联盟、超文本标记语言、标准通用标记语言、可扩展标记语言、XHTML、Java、JavaScript、VBScript、层叠样式表、服务器端扩展、公共网关接口、Java servlet、DNS平衡、软硬件负载平衡、逆向代理、插件、ActiveX、cookie、ColdFusion标记语言、路由器、非法入侵检测系统以及DCS-1000。
- 解释将数据库连接到Web页面的重要性。
- 描述带有数据库连接的因特网和内部网的基本环境。
- 恰当使用因特网相关术语。
- 解释万维网联盟的目标和已取得的成就。
- 解释服务器端扩展的目的。
- 比较和对比Web服务器接口，包括CGI、API和Java servlet。
- 描述平衡Web服务器负载的三种方式。
- 解释插件。
- 解释如何用ColdFusion或ASP实现简单的购物车应用程序。
- 讨论Web安全性问题。
- 讨论Web保密性问题。

10.2　引言

随着**万维网**（World Wide Web，WWW）的日益普及，数据库的重要性也日益凸现出来。**电子商务**（electronic commerce，e-commerce），也就是在因特网上进行交易，已经成为驱动万维网发展最主要的原因。似乎从AT&T到罗丝花店中的每一项交易都在迎接着一种新的挑战，那就是调整业务以充分利用全球网络（我们称之为因特网）的优势。因特网技术在公司内部的应用，即企业内部网，也已被广泛采用。

公共的因特网和私有的企业内部网可以被看作是一种巨大的客户/服务器（Client/Server，C/S）体系结构，该结构的客户机（浏览器）很瘦而服务器很胖。服务器把信息存储在数据库中，根据需要传送给浏览器。在应用当中，访问数据库的当前信息（例如库存信息）和记录信息（例如与订单有关的数据项）都是非常必要的。然而，如果开发人员没有意识到数据的安全性问题，那么把数据库连接到Web应用上就可能会导致在意想不到的情况下开放对数据库的访问。对于世界各地的客户而言，新的在线购物站点所带来的问题是显而易见的。数据库专业人员必须知道怎样谨慎地建立、操作和管理Web与数据库之间的接口。

本章将介绍数据库与因特网应用程序之间如何配合，包括各种因特网数据库体系结构，阐明在启用Web的数据库中ASP与ColdFusion的使用，并讨论与上述主题相关的管理问题。

10.3 因特网和数据库连接

Web环境具有支持快速采用和实现因特网及企业内部网业务应用的若干特征。首先，**浏览器**（browser）界面简单、功能相似，极大减少了刚开始使用时所遇到的障碍（如复杂性）。就像微软的Windows操作系统使用方便，人们可以轻松学习新的Windows应用程序一样，最流行的浏览器界面在功能上都很相似，这使得用户在浏览器之间或者商业站点之间进行切换非常便利。大多数浏览器窗口的顶部会显示和Windows工具栏功能相似的工具栏，用户可以打印当前页或拷贝/粘贴当前页上的信息。同时，大多数浏览器都支持电子邮件、即时消息、Web站点书签以及Web寻址和搜索。采用本章后面将要介绍的HTML、DHTML、XML和JavaScript使浏览器的信息表达和功能具有一致性，站点之间的切换也变得更加容易。

其次，由于浏览器的软硬件之间相互独立，所以跨平台的信息共享也相对容易，以前许多难以解决的跨平台存取问题也得以解决，特别是使得广泛访问数据库信息也成为可能。借助因特网网络，已经可以做到存储位置互相独立。这样，通过使某些信息可公开访问，而把关键信息放在防火墙后面或者放在另外的服务器上以防止公共存取，公司就可以在本地或远程站点访问他们的数据。如今，借助于因特网移动数据文件和更新数据库已经变得非常普遍了。

最后，开发费用和时间也得以降低。因为可以获得便宜甚至是免费的开发以及开发工具，所以减少了切入该领域的障碍。新生儿可以拥有他们自己的站点，该站点由他们的父母管理。人们期望各行各业都有自己的站点，虽然这些站点的功能性和实时性各不相同，但找不到某个行业站点的机会还是很少的。同时，每个站点的功能和处理速度可能相差很大，而且在因特网上可获得的带宽也在一定程度上决定Web页的显示和装载速度。事务会随着警戒频率而终止，页面显示也会不合时机地冻结。每个站点的导航逻辑不同，并且也不是很明显。

许多站点没有与其相连的数据库，它们所提供的仅仅是静态信息，这些静态信息采用HTML、JavaScript、CGI或其他脚本语言进行编码。然而，许多站点都需要从与其相连的数据库中提取信息或向其中存放信息。有一些站点只是信息的存储库，访问者也可以查询这些存储库。在访问这些站点时，用户可以提出信息请求并阅读所获得的信息，但不能以任何方式修改信息。例如，在佛罗里达州的Hillsborough郡，任何人都可以查到谁拥有该郡的土地，以及土地税率的估计、以往购买价格等有关该土地的详细信息。信息存放在访问者可以查询的数据库中，根据指定的记录、街道地址或者业主姓名等属性进行访问。还有一些站点包含产品库存信息，这些信息也存储在数据库中。这些站点可以根据访问者的查询，提供关于特定产品或产品类别的信息。可以说，任何存储在某个关系数据库中的数据都可以被连接到某个站点上，你可以对Web站点进行探测来验证一下。以数值、字符或图形的方式存储数据已得到广泛的应用。

另外一些站点在用户和数据库之间提供了更多的交互性，用户可以把信息写入与站点相连的数据库中。由于客户可以在线下订单，因此这种功能成为电子商务爆炸式增长的有力支持。客户经常在下订单前，需要确定想要的产品是否有现货，以及收到该产品需要多长时间。所以，电子商务应用必须具有显示和装载与站点相连的数据库的能力。

随着新老业务开始争相采用上述技术，术语**电子业务**（e-business）也应运而生。电子业务描述了一种启用技术的业务，它使用因特网相关技术促进顾客与供应商之间紧密关系的进一步发展。与电子商业的概念相比，电子商务较为严格，它更多地是指电子业务进行过程中所发生的事务。

很多企业认为电子业务的潜在优势包括：改善供应链管理、改善客户服务、缩短产品上市时间、降低成本以及增加销售量。其中，一些目标比另一些目标容易达到。当站点由于订单多

而发生堵塞时，供应链管理问题就会受到很多负面的压力。另外，事实证明，改善客户服务的代价很高，我们中的很多人可能都向商务公司发送过请求，希望得到信息或支持，但没有收到任何回音。所以说，电子业务的获利能力还难以确定。

为了与顾客和供应商之间达到无缝集成，很多公司作了努力，因此出现了电子业务的两个完全不同的类别：B2C（Business to Consumer）和B2B（Business to Business）。它们也被称作零售电子商务和商业电子商务。B2B的接口在某些方面比较简单，主要是以电子方式构建已有的贸易关系。已经证实，通过与商业伙伴建立在线贸易，公司可获得巨大收益。零售电子商务则必须付出更多的努力才能让顾客在线下订单，因为交易的很多方面涉及安全问题，而且客户可能还没做好访问因特网的准备，或者他们更愿意以那种看得见、摸的着的方式购买商品。由于利用因特网可以提高业务效率、改善客户关系，已实现的或预期的效益增长会驱动这个与数据库相关的领域继续发展。

10.3.1 因特网环境

因特网体系结构在后面将会详细介绍，本节只是让读者对因特网环境有初步的理解。图10-1展示建立连接数据库的内部网和因特网所必须的基本环境。右侧的图描述了内部网。从图中可明显看出该结构是典型的C/S体系结构。客户端工作站与Web服务器和数据库服务器之间的连接网络遵循TCP/IP协议。其中，TCP代表传输控制协议，IP代表因特网协议。对于各种因特网传输来说，无论是发送电子邮件、浏览Web，还是用作Web服务器，这两条协议都是必须遵守的。TCP把长的消息分成由小的数据片段组成的多个包，每个包在Web上独立传送，到达目的浏览器后重新装起来并显示。每一台与内部网或外部网相连的计算机都必须有不同的IP地址。如果不需要连接到Web，内部网的IP地址就不需要像对因特网通信网络开放时那样设置为惟一的IP地址。在第9章中描述过多层内部网结构，不过，图10-1所显示的体系结构较为简单，在这种结构中，从客户浏览器发出的请求通过网络传输到Web服务器，而服务器将存储返回的用HTML书写的页面并显示在客户浏览器上。如果所发送的请求需要从数据库中获取数据，则由Web服务器构建一个查询并将其送至数据库服务器，数据库服务器处理查询，然后返回查询结果。同样，输入客户端工作站的数据也可以通过Web服务器被送到数据库中储存起来（Web服务器将数据传送到数据库服务器，数据库服务器再把数据提交到数据库）。

上述处理流程同样适用于与外部网相连的情形，无论这种连接仅仅是针对特定顾客或供应商，还是任何与Web相连的工作站。然而，对外开放Web服务器需要采取一定的数据安全措施。在企业内部，通常由数据库管理系统来控制对数据的访问，由数据库管理员设置员工对数据的访问权限。**防火墙**（firewall）则是用来限制对企业内部数据的外部访问，使企业数据不至外泄。在内部网之外的所有通信都通过**代理服务器**（proxy server）来完成，由它来控制出入内部网的消息以及文件的传输。此外，代理服务器还可以通过高速缓存被频繁访问的页面，这样不必连接Web服务器就可以显示被请求的页面，从而改善站点的性能。

大多数内部网提供如下服务：

- **Web服务器**　处理客户机请求并向客户机返回HTML页面。
- **数据库服务**　提供通过Web服务器和数据库服务器的数据库访问。Web到主机的访问能够访问存储在主机上的遗留数据。
- **目录、安全和认证服务**　防止对内部网进行无认证访问，并提供责任追查能力。
- **电子邮件**　提供内部网、外部网、因特网上计算机之间的传递消息能力。
- **文件传输协议**（File Transfer Protocol, FTP）　提供内部网、外部网、因特网上的计算机之间拷贝文件的能力。用户需要FTP客户端，远程系统需要FTP服务器。

- **防火墙和代理服务器** 提供安全措施防止外界非法入侵公司内部网。
- **新闻组或讨论组** 提供在公众可访问的共享公告板上张贴信息的功能。公告板上的内容包括与某个主题（在其原始位置被标明）相关的一系列讨论。
- **文档搜索** 提供搜索Web站点内容的能力。
- **负载平衡和高速缓存** 分散繁忙的通信流，改善操作性能。

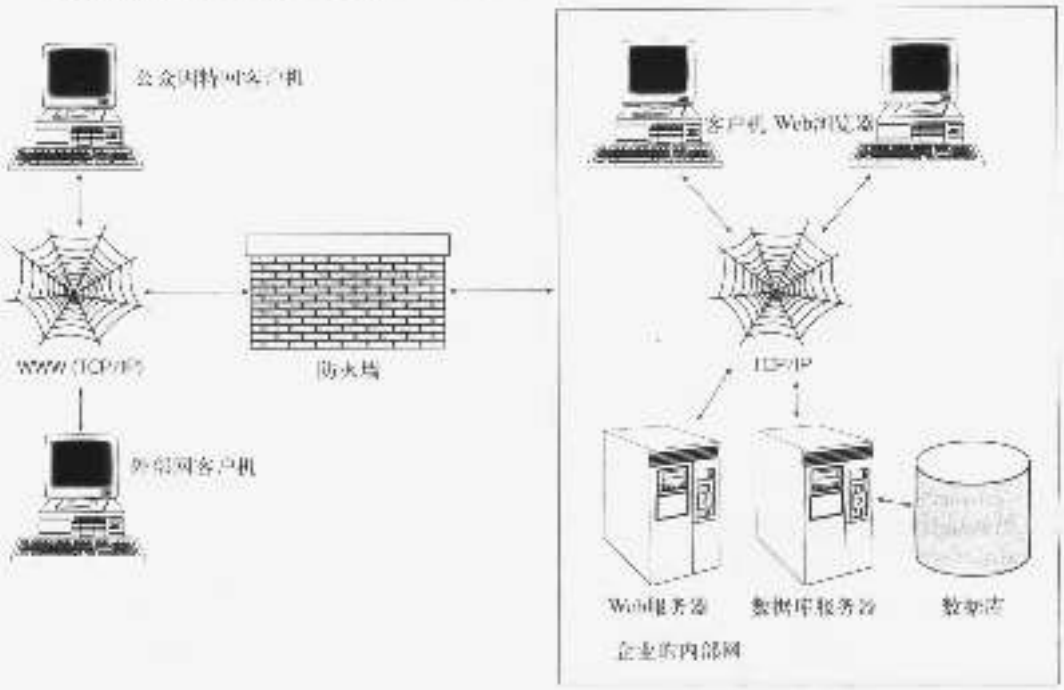


图10-1 带有数据库连接的内部网/因特网环境

10.3.2 术语

信息技术的每一个发展阶段都会产生一系列新的术语和缩写，网络与因特网的引入也不例外。除了前面介绍的有关术语以外，还有一些用户需要熟悉以下一些与Web相关的术语。

1. 与通信有关的术语

- **IP地址** IP地址由四个0～255的数字组成，它们之间用句点（小数点）分隔，例如131.247.152.18。IP地址很难记住，不过每个IP地址都与一个具有实际意义的域名一一对应，这些域名便于记忆，例如www.usf.edu、www.udayton.edu或者www.prenhall.com。IP地址的索引以及与它们相匹配的域名则由域名服务器进行维护。
- **超文本传输协议**（Hypertext Transfer Protocol, HTTP） 这是一个将浏览器的请求传送到Web服务器并且从Web服务器将页面传回浏览器的通信协议。然而，该协议不是特别安全，另外一个协议——HTTPS可提供加密并通过安全端口传输。
- **统一资源定位器**（Uniform Resource Locator, URL） 它是一种指定Web服务器的通信协议及其IP地址的Web地址表示法。作为可选项，还可以同时指定所请求的HTML文件的文件夹路径和文件名。图10-2显示了一个典型的URL。如果在URL中没有指定通信协议，则HTTP就作为默认的通信协议；如果没有指定文件夹路径和文件名，则将Web服务器的根目录作为起始位置。

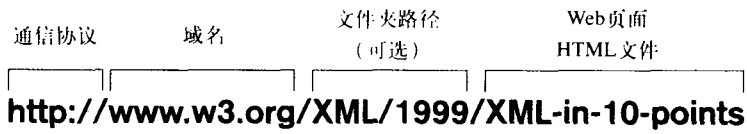


图10-2　一个典型的URL

2. 与Web有关的术语

- **静态Web页面**　内容在写的时候就已经确定的Web页面。任何时候访问该页面都会显示同样的信息。
- **动态Web页面**　显示符合客户端请求/输入的数据的Web页面。通常，产生动态Web页需要数据库通过打开的数据库连接附加在页面上。请参见第9章对ODBC连接的讨论。

10.4　常见的因特网体系结构的组成

一组容易让人混淆的技术和工具（大多用缩写标识）组合在一起就创建出因特网环境。本节讨论该环境中最常见的组件。首先介绍创建Web页面的各种语言，有时还可以结合使用它们。其次，讲述几种作为Web服务器中间件的资源。这些处理请求或者来自Web服务器，或者发送到Web服务器，进而可将其区分为服务器端扩展和Web服务器接口。同时，还讨论Web服务器和客户端扩展。

图10-1只是给出因特网环境的总体结构，并没有在其中包含组件并把它们装配起来。此外，使用冗余工具还能获得各种不同配置。一般而言，同一类的Web技术可以交替使用，一种工具和另外一种工具可以解决同样的问题。比如在本章末尾所举的例子中，分别采用ASP和ColdFusion技术将Web站点与一个数据库连接起来。而具体选择何种工具则由需要遵循企业的开发标准。因此，必须首先了解某类工具的用途，以便知道何时该采用它。其次，还需要一本优秀的产品参考手册，它会提供语法并帮助你解决接口问题。

10.4.1　与因特网相关的语言

万维网联盟（World Wide Web Consortium，W3C）是制定HTTP和HTML标准的主要机构。W3C由Tim Berners-Lee在1994年创建，是一个国际性的公司联盟。W3C致力于发展开放式标准，这些开放式标准促进开发Web规范，使Web文档能够跨平台地一致显示。他们还提出了XML和XHTML规范，最近的草案是2001年3月22日提出的一整套针对XHTML的XML Schema模块，以及一个扩展和修改XHTML的框架。

用于创建Web文档的基本语言是HTML，即**超文本标记语言**（Hypertext Markup Language）的缩写。HTML与**标准通用标记语言**（Standard Generalized Markup Language，SGML）类似，SGML描述文档标记元素的规则，从而使标记能以一种标准方式格式化。HTML的标记规范正是基于SGML规则的。

HTML是一种脚本语言，为了显示而使用各种标记和属性来定义Web文档结构及布局。例如，一个HTML文档以标记<HTML><HEAD>（在这里输入文档主题）</HEAD><BODY>开始，需要显示的信息和其他格式标记紧随其后，最后以标记</BODY></HTML>结束。

XML是一种正在快速发展的脚本语言，它也是基于SGML的，它与XHTML结合使用可能会取代HTML。XML是**可扩展标记语言**（Extensible Markup Language）的缩写，是由W3C开发的另一套规范。XML专门用于Web文档，它允许用户定义自己的标记。这些标记可以跨组织使用，从而使得应用程序之间和组织之间的数据能够被定义、传输、验证和解释。经证明，如果要在Web上加入以前遗留的数据，则XML非常有用，因为当XML标记在遗留数据的存储器

中格式化时，可以用来定义数据，所以避免了重新格式化。

W3C还提出了一种混合脚本语言规范——XHTML，它扩展了HTML编码使之与XML兼容。XHTML采用三种XML命名空间，这三种命名空间与HTML 4.0的三种数据类型定义（DTD）相对应，它们分别是Strict、Transitional和Frameset。同时，由于XHTML所采用的模块也符合某种标准，因此Web文档的布局与表示仍然是跨平台一致的。W3C的希望使用XHTML取代HTML成为标准脚本语言，你可以通过访问W3C的站点来了解他们的进展。

前面所提到的语言都属于脚本/标记语言，它们的主要功能是处理文档的布局和显示，而不能编程实现某种功能或活动。Java则是一种非常适于在Web上使用的通用编程语言。Java applet是一种Java小程序，可以将其从Web服务器下载到客户机，在与Java兼容的Web浏览器（如Netscape Navigator或Microsoft Internet Explorer）上运行它。

Java是一种面向对象的语言，它比C++简单。该语言是由Sun Microsystems设计的，最初的想法是设计一种较少导致常见程序错误的语言。Java的源代码文件会被编译成一种叫做*字节码*（bytecode）的格式，由Java解释程序执行。而Java的解释程序和运行时环境，即Java*虚拟机*（VM），在大多数操作系统中都存在，因此编译后的Java代码可以在大多数系统中运行。字节码可以直接转换为机器语言指令。

Netscape独立开发了JavaScript，它的许多特性和结构与纯Java语言类似。使用JavaScript编写Web文档可以获得交互性，还可以引入动态内容。例如，鼠标滚动、内容被更新的自动提示、通过嵌入在HTML代码中的JavaScript完成错误处理，等等。在事件发生时，如将鼠标滚动到某个按钮上，就会激活JavaScript。JavaScript是一种开放式语言，不需要许可证，当前的Netscape和微软浏览器都支持它。

VBScript与JavaScript类似和JavaScript基于Java但比Java简单一样，VBScript也基于VB但比VB简单。VBScript可以在Web页面上添加按钮、滚动条和其他交互式控制元素。这种脚本语言由微软开发并且可以在微软的IE浏览器上使用。

层叠样式表（Cascading Style Sheet，CSS）是另外一种由W3C开发并添加到HTML中的特性。利用CSS，设计人员和用户可以创建样式表，定义不同元素的外观，如报头和链接的外观。定义完毕之后，该样式表就可以应用到任何Web页面上。其实，每个Web页面上还可以应用多个样式表。目前已经发布了两种W3C推荐规范（CSS1和CSS2）。这两种规范在大多数浏览器上都已经得以实现，只是实现的程度有所不同。

10.4.2 服务器端扩展

由于Web服务器只能理解HTML格式的页面，所以为了处理来自客户端对数据库的访问请求，必须提高或扩展Web服务器的功能。**服务器端扩展**（server-side extension）其实是一个程序，它直接与Web服务器交互，以处理各种请求。例如，可以扩展Web服务器功能以支持数据库请求，图10-3显示了该过程。首先，浏览器的信息请求通过Web提交到Web服务器；脚本中包含SQL查询，但是Web服务器无法直接解释查询，而是通过Web-数据库中间件识别查询，并准备把查询传递到数据库管理系统和存储数据项的数据库；查询的结果返回到中间件并在那里进行转换，这样，当Web页面返回客户端浏览器时才能正确显示出来。

服务器端扩展具有极大的灵活性。网络管理员可以选择任何Web服务器（可以是Netscape Fast Track Server也可以是Microsoft Internet Information Server），也可以采用任何有SQL功能的兼容ODBC的数据库（如Oracle、Sybase或MS Access），中间件则与两者相连（ColdFusion和Netscape Application Server就是这种中间件）。使用ColdFusion和ASP的详细例子将在本章末尾介绍。

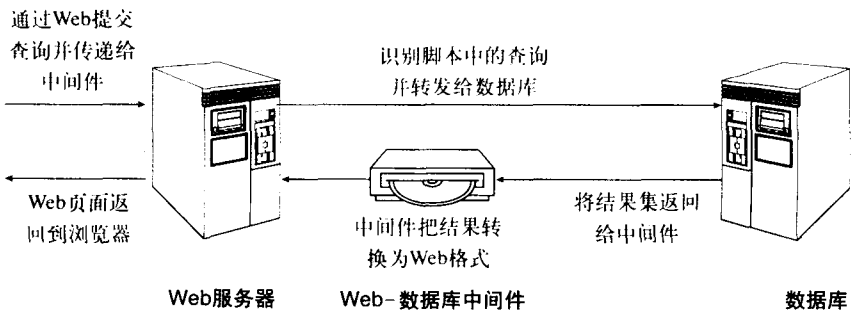


图10-3　Web-数据库中间件

10.4.3　Web服务器接口

为了使Web服务器与外部程序进行交互，必须建立适当的接口机制。无论使用的术语是接口、互操作性还是交互性，Web服务器与客户机（或数据库）之间都必须有通信能力。静态Web页面不一定需要具备这种类型的接口，因为所有要显示的信息都已存储在HTML文档中。然而，动态Web页面只有在客户端浏览器请求某个页面时，才能决定将要显示的内容。例如，显示某些被更新的信息（如产品当前的库存量）时就需要Web服务器接口。常见的Web服务器接口有以下两种：

- 公共网关接口（Common Gateway Interface，CGI）。
- 应用程序接口。

CGI指定Web服务器和CGI程序之间的信息传输模式。CGI程序用来接收和返回数据，它可以用任何能产生可执行文件的语言编写，这些语言包括C或C++、Perl、Java或Visual Basic等。不过，CGI程序所处理的数据必须遵循CGI规范。通常，浏览器上会显示一些表单，由用户填写，而CGI程序用来接收这些表单中的数据。当然，CGI程序也可以用来接收遗留系统中的数据，例如某个已有数据库或其他的文档等。需要注意的是，网关程序属于可执行程序，它能够在不同的信息服务器下交替运行。

CGI程序是Web服务器与用户请求之间进行动态交互的常用方式，还有其他一些客户端方法会在后面的内容中陆续讲到。CGI脚本存储在Web服务器上，并且在用户发出使用该CGI脚本的请求时执行。如果很多用户同时发出使用CGI脚本的请求，则服务器性能就会发生显著下降。为解决这个问题，相继出现了一些服务器端解决方案，如Java脚本和applet、ActiveX控件等，这些内容在后面的部分中将会一一讨论。

除了CGI程序以外，Java servlet是Web服务器接口的另一个选择。与applet相同，servlet不是在操作系统而是在其他应用程序中执行的程序，但它们存储在服务器上而不是和应用程序一起存储在客户端。servlet允许客户端程序向服务器上载额外的程序代码，并且在服务器上执行代码。因为servlet的文件尺寸很小，而且可以跨平台兼容，所以作为浏览器的因特网小应用程序非常理想。Java servlet是持续的，一旦被启用，它们就驻留在活动内存中，并且能够完成多个请求。而CGI程序在运行后关闭，所以Java sevlet更加有效，而且因为它是持续的，所以服务器性能也不大会受其影响。

应用程序接口（Application Programming Interface，API）也比CGI脚本更加有效，一般将API实现为共享代码或动态链接库（DLL）。这就意味着代码驻留在内存中，并可以根据需要动态调用。对每个请求而言，不需要执行任何外部程序。一个API是一组例程、协议和工具，应用程序用它来控制操作系统过程的性能。API也可以使用共享的数据库连接，而不需要在每次请求连接时都建立一个新链接。不过，API也有一些缺点。因为它驻留在Web服务器上，所以

一个API错误可能会导致服务器崩溃。此外，API对于其驻留的操作系统和Web服务器而言是特定的，如果要在其他系统上运行就必须重写API。

10.4.4 Web服务器

这里介绍一个有趣的Web站点：www.netcraft.com/survey/，该站点每个月对所能识别的因特网Web服务器都进行一次调查。1995年，它第一次调查了3428个站点；到了2001年4月，这个数字已经达到28 669 939个。站点数目的指数级的增长取决于两个方面的能力：一是与数据库通信的能力，二是提供包含当前信息的动态页面的能力。这些站点都能提供HTTP服务，该协议允许Web浏览器和Web服务器之间进行通信。HTTP是一个相对简单的协议，通过TCP连接传递纯文本。起初，对每个需要下载到浏览器的对象，都需要建立一个新的HTTP连接。现在，新版本的HTTP支持持续连接，因此，多个对象就能够以包的形式通过单个TCP连接进行传输。

借助于多线程/多处理技术，Web服务器能够同时为多个用户提供服务。运行在Unix之上的Web服务器通常采用多处理技术，而其他服务器则采用多线程技术、多处理技术或二者的混合。多处理技术通过串联多个处理器达到快速处理的目的。一个线程是一个较大程序的一部分，多线程技术就是同时运行多个线程的技术。

流行Web站点被点击的频率往往会超过单个服务器的管理能力，因此必须安装多台服务器。此时，如何平衡负载以充分利用各服务器就成为一个关键问题。一些站点采用**域名服务器平衡**（domain name server balancing，DNS 平衡）处理大量的点击，在分开但却相同的物理服务器上放置站点的多个拷贝。站点主机名的DNS服务器返回站点的多个IP地址，可以对该主机名返回多个IP地址，也可以对所收到的每个DNS请求返回不同的IP地址。这种方法虽然简单，但因为IP地址的选择不一定平衡，所以也不能确保服务器上的负载是平衡分配的。

软硬件负载平衡（software and hardware load balancing）能够更为平均地在Web服务器之间分配请求。在这种情况下，站点仅有一个公开的IP地址，对该IP地址的请求在TCP/IP路由层被分配到驻留站点的各个服务器中。采用这种方法所达到的负载平衡通常比DNS方法好。还有一些负载平衡器检测到池中的Web服务器停机时，可以把请求动态重定向到另外一台Web服务器上。

第三种方法通过中途截取来自客户端的请求并且在将响应返回客户端的Web服务器中高速缓存响应来减少负载。这种方法叫做**逆向代理**（reverse proxy），即由代理在自己的本地高速缓存中处理请求，而不与Web服务器连接，从而减少Web服务器的负载。

某些公司采用全局负载平衡技术，这种技术在全球范围内对数据文件的发送进行分配。通常，他们同时采用DNS负载平衡技术和软硬件负载平衡技术，通过判断客户端的位置，把距离用户最近的文件提供给用户。

现在，Web服务器有时也提供XML数据和HTML数据，起着应用服务器的作用，因此，Web服务器与应用服务器之间的差别也变得不那么清晰了。另一方面，应用服务器也可以配置为一个简单的Web服务器，但它还是作为一个应用服务器来使用。如何配置Web站点取决于该站点的应用情况。如果某个Web站点需要从一个应用中传送大量数据，那么它最好同时具备一个Web服务器和一个应用服务器；而对于那些传递不需要频繁依赖于某个应用的HTML页面的站点，有一个具有应用服务器功能的Web服务器就足够了。

10.4.5 客户端扩展

所谓客户端扩展就是向浏览器增加功能。本节简要介绍几种最流行的方式，包括插件、ActiveX控件、cookie等，以帮助读者理解什么是客户端扩展。

插件（Plug-ins）是指为了扩展浏览器的最初功能而添加某种特征或服务的硬件/软件模块。通过把某个插件下载到浏览器的插件文件夹并进行安装，就可以获得客户端扩展。例如，利用

RealAudio插件能够收听Web上的音频广播，利用Shockwave插件能够播放动画文件，还有掌上电脑与PC同步、无线访问、加密等插件。尽管插件最初是遵循Netscape Navigator规范而设计的，但它也同样适用于Microsoft IE和其他浏览器。不过，Microsoft IE除了支持大多数Netscape Navigator插件以外，也开发了一个自己的软件标准——ActiveX，用来取代插件。ActiveX将在后面的内容中介绍。同样，Netscape Navigator也支持一些ActiveX控件。

大多数插件都是免费的，而且它们的尺寸很小，便于下载。尽管如此，有时用户还是会拒绝添加插件，因为插件可能会使浏览器或PC以不期望的方式运行。基于这种情况，同时考虑到插件可能会妨碍搜索引擎选择公司站点，所以公司可能不会在其站点上使用插件。此外，与其他外部开发的功能一样，最流行的插件会集成到浏览器的未来版本中。

ActiveX是由微软开发的一套松散定义的技术，它来源于其他两项微软技术——OLE（Object Linking and Embedding）和COM（Component Object Model）。ActiveX控件对浏览器进行扩展，并且可以操纵浏览器的内部数据。它们是微软发布的OLE控件的第3版，通过使用COM技术，可以提供与其他类型COM组件及服务进行协同工作的能力。例如，允许用户在运行控件之前明确控件的作者，这是重要的Web安全特性。COM体系结构可以在所有微软产品上工作，产生具有标准接口的对象，因此可以用在任何兼容COM的程序当中。通常使用C++或Visual Basic编写ActiveX控件。

当用户重新访问某个Web站点时，cookie可以用来验证用户身份。一般而言，Web站点会要求访问者填写一个表单，内容包括姓名、电子邮件和其他一些信息（邮政地址、电话、与站点相关的兴趣爱好等）。用户提供的数据存储在cookie中并发送到Web浏览器。当用户再次访问该站点时，cookie的内容就会被送回服务器并返回定制的Web页面。由于Cookie可以在浏览器上存储相当长时间，所以它是持久的。拒绝提供相关信息的用户可以选择不在他们的Web浏览器上存储cookie。

10.5 Web-数据库工具：ColdFusion与ASP

现在，数据库与Web应用之间的连接可以用很多中间件应用程序来实现，其中，ColdFusion和ASP（Active Server Page，微软的活动服务器页面）最为流行。还有其他一些产品在2001年的中期开始受到关注，如Cocoon、元素构造集（Element Construction Set，ECS）、可扩展标记语言编译器（eXtensible Markup Language Compiler，XMLC）、可扩展样式表语言转换（eXtensible Stylesheet Language Transformation，XSLT）和Java服务器页面（Java Server Page，JSP）。以下主要从三个方面讨论上述各种产品的品质：所需编码的数量与性质、可移植性、编译器需求。

首先，为了帮助读者理解数据库与Web页面的连接，下面引入一个简单的购物车示例。在这个例子中既可以使用ASP也可以使用ColdFusion 4.0。该购物车应用程序允许在Web站点上选择项目并下订单，其中附带一些理解连接过程所必须的代码。使用ASP或ColdFusion实现购物车应用程序的完整脚本文件和指令可以从以下Web站点获取：www.prenhall.com/huffer。需要说明的一点是，该示例仅仅展示如何从连接数据库中获取数据，而并没有存储订单信息或接受信用卡信息。

ASP和ColdFusion都允许在HTML文件中自定义标记，这些定制标记使得服务器端能够产生动态Web页面。ColdFusion页面以扩展名为.cfm的脚本文件格式存储，ASP则以扩展名为.asp的脚本文件格式存储。

10.5.1 ASP示例

ASP由文本文件组成，其中包含文本、HTML和脚本语言命令（通常是JavaScript或VBScript）。由于这些文件在服务器上执行，客户端平台只是显示服务器端处理的结果，所以

程序员不必关心客户端平台。ASP通过它们的扩展名.asp来确定。与其他URL一样，ASP可以通过Web浏览器访问，如通过微软的IE或Netscape Navigator来访问。

请求ASP文件会使服务器处理所有嵌入到页面中的脚本命令，生成HTML文档，并把该HTML文档返回到客户端加以显示。

这里所列举的简单的ASP购物车示例包含一个全局文件和6个.asp文件：

- global.asa——每个ASP应用程序必须包含一个global.asa文件。该文件既管理应用程序，也管理从客户端开始的任何会话。
- cart.asp——显示购物车的内容，并负责添加或删减购物车中的项目。
- checkout.asp——接受购物者信息并完成订单，但没有考虑订单存储和信用卡处理。
- item.asp——显示特定项目的信息（某件家具），提供可以链接到搜索框和主区域的选项。
- line.asp——显示在某次购物结束时得到的所有产品的列表，允许访问搜索框。
- search.asp——显示与搜索参数Product_t.Product_Description相匹配的产品列表，搜索参数由表格中的字段serarchval决定。
- store.asp——松谷家具公司ASP购物车演示程序的主页。可以使用搜索工具查找特定项目，并且列出所有不同的、可以购买的成品系列。

在文件global.asa中指定事件脚本，并且声明ASP应用程序中各个页面可以访问的对话和应用程序对象。图10-4说明在购物车应用程序中使用的global.asa文件，该文件必须存放在ASP应用程序的根目录下。每个ASP应用程序只能有一个global.asa文件，其中可放置开始/结束应用程序和会话的指令。在本例中，开始一个新会话就意味着开始一个响应客户请求的购物车，其中包含一个可以跟踪定货数量的变量。

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>

'You can add special event handlers in this file that will get run automatically when
'special Active Server Pages events occur. To create these handlers, just create a
'subroutine with a name from the list below that corresponds to the event you want to
'use. For example, to create an event handler for Session_OnStart, you would put the
'following code into this file (without the comments):

Sub Session_OnStart
'**Put your code here **
    Session("Cart") ' Tracks what they want to order
    Session("Count") ' Tracks the quantity that they want
End Sub

'EventName          Description
'Session_OnStart    Runs the first time a user runs any page in your application
'Session_OnEnd      Runs when a user's session times out or quits your application
'Application_OnStart Runs once when the first page of your application is run for the first
                    time by any user
'Application_OnEnd  Runs once when the web server shuts down

</SCRIPT>
```

图10-4 global.asa

图10-5列出购物车应用程序6个.asp文件中的store.asp，其余的5个文件以及在PC上的安装指令可以登录站点www.prenhall.com/hoffer获取。store.asp是松谷家具公司购物车的主页，它展

示出很多不同的ASP功能，可以帮助读者理解ASP应用程序如何把一个数据库连接到应用程序上。如果读者使用运行Windows操作系统的PC，那么可以采用个人Web服务器（Personal Web Server）安装购物车程序运行的测试环境。

Box A

```
<%@ language=VBScript %>
```

Box B

```
<%
option explicit
REM designed by: Michael Alexander
REM Project: Pine Valley Furniture Demo, ASP
REM Purpose: Store Entrance

REM This page provides entrance to the Pine Valley Furniture Store
REM asp demo version. It provides the user with a search tool to
REM search for a specific item. It also has a listing of all the
REM different Finish "Lines" available for purchase.
%>
```

Box C

```
<%
Dim conn
Dim rsRes
Dim strSQL
REM Set up Database connection
set conn = server.CreateObject ("adodb.connection")
conn.Open "PVFMDB"
%>
```

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE>Pine Valley Furniture Company Web Store</TITLE>
</HEAD>
<BODY bgcolor=#ffffcc text=#00008b>
<TABLE align=center>
        <TR>
            <TD><STRONG><FONT size=6>Welcome to the Pine Valley Furniture Store
                    </FONT></STRONG></TD>
        </TR>
</TABLE>
<HR><BR><BR>
<TABLE>
        <TR>
            <TD valign=top align=center>
                <B>Search for a product such as bookcase, nightstand, or table:</B><BR>

                <FORM action="search.asp" method=post>
                        <INPUT Name="searchval" type=text width=20><BR><BR>
                        <INPUT Value="SEARCH!" type=submit><BR>
                </FORM>
```

Box D

```
                <% If Session("Cart") <> "" Then %>

                <FORM method=post action=cart.asp>
                        <INPUT type=Hidden name=Quantity value=0>
                        <INPUT type=Hidden name=item value=0>
                        <INPUT type=Hidden name=none value=0>
                        <INPUT type=Submit value="Go to Cart">
                </FORM><BR>
                <% End If %>
```

图10-5　store.asp

```
</TD>
<TD align = center valign=top width=100>
<B>Or, search by product finish:</B><BR><BR>
```

Box E

```
<%
REM Get a list of Finishes
strSQL = "SELECT Product_Finish FROM PRODUCT_t GROUP BY Product_Finish;"
set rsRes = conn.Execute(strSQL)
%>
```

```
<TABLE>
```

Box F

```
<%
REM Display the list of Finishes
While not rsRes.EOF
%>
        <TR>
                <TD align=center valign=top>
                        <%=rsRes("Product_Finish")%></TD>
                <TD>
                        <FORM method=post action="line.asp">
                        <INPUT type=Hidden name=line
value="<%=rsRes("Product_Finish")%>">
                        <INPUT type=submit Value=GO!>
                        </FORM>
                </TD>
        </TR>
<%
        rsRes.MoveNext WEnd
%>
```

```
</TABLE>
    </TD>
  </TR>
</TABLE>

</BODY>
</HTML>
```

Box G

```
<% rsRes.Close %>
```

图10-5 （续）

下面对图10-5作一些说明。

Box A使用ASP标记（<% %>）向服务器指明ASP应用程序将使用VBScript，并且应该进行相应的解释。Box B包含一条命令“option explicit”，这意味将通过Dim语句显式声明所有变量。使用Option Explicit很容易发现拼写错误的变量名。Box C显式声明变量并建立数据库连接。注意，数据库名为“PVFMDB”，在建立ODBC连接时，必须使用该名字。

Box D表明ASP能够处理条件逻辑。在本例中，如果客户选中某种商品，则将其提交至购物车以便于跟踪，直至购物完成。Box E展示ASP如何与数据库连接并发送查询。在本例中，查询将会选择家具成品，并形成某一系列的产品。Box F说明在客户机上显示Box E的查询结果所必须的HTML和ASP的组合。Box G关闭变量rsRes，它用来遍历形成某个产品系列的所有成品。

这个简单的ASP编码示例说明，任何一个有HTML和VBScript/JavaScript经验的人都能很快地用ASP建立一个动态Web站点。而那些对VBScript/JavaScript不太熟悉的人可能会发现ColdFusion更容易理解和快速使用。本书建议读者安装这两种购物车的版本，以便在自己的机器上比较它们的优劣。

10.5.2 ColdFusion示例

ColdFusion要求使用服务器端标记语言——**ColdFusion标记语言**（ColdFusion Markup Language，CFML）产生ColdFusion应用页面脚本，这种脚本的扩展名为.cfm。当客户机浏览器向Web服务器发送一个.cfm页面请求时，该请求会被传递到ColdFusion应用服务器，在那里执行脚本，执行的结果会转化为HTML格式并返回Web服务器，Web服务器进一步将结果返回客户机并进行显示。

CFML以HTML为模板构成，它包含读和更新数据库表的标记，可以产生并检索电子邮件消息，并且能执行HTTP和FTP操作，进行信用卡核实和身份验证、读写客户端cookie等操作。借助于这些功能就可以实现动态页面的创建、用真实数据填充表单、处理表单提交的能力以及与本地文件的交互。

简单的ColdFusion购物车包含7个.cfm文件：

- application.cfm——该文件设置应用程序、客户机和会话范围变量的状态以及激活时间的长短。这个文件自动由ColdFusion服务器包括在每个页面的顶端。
- cart.cfm——显示购物车内容，允许增加或减少购物车中的项目。
- checkout.cfm——接收购物者信息并完成订单，但尚不具有存储订单和处理信用卡的功能。
- item.cfm——显示有关某项目（某件家具）的信息，并提供链接到搜索框和主区域的选项。
- line.cfm——显示在某次购物完成时得到的所有产品的列表，允许访问搜索框。
- search.cfm——显示与搜索参数Product_t.Product_Description匹配的所有产品的列表，搜索参数由表格中的字段serarchval决定。
- store.cfm——松谷家具公司ColdFusion购物车演示程序的主页。可以使用搜索工具查找特定项目，并且列出所有不同的、可以购买的成品系列。

图10-6显示了完整的application.cfm文件。其中，<CFAPPLICATION>标记指定应用名称为PVFC。客户机与会话管理都设置为“TRUE”。激活客户机与会话管理使得Web应用能够对每个客户机保持一种虚拟状态，这对于记住每个客户购物车中的内容非常必要。无论是否使用cookie，这些选项可以保证会话之间彼此区分并且保证安全，而且虚拟地使用所有浏览器。

将要被访问的数据库也在此处命名，在机器上安装购物车演示程序时将会用到该数据库。本例中，ODBC数据源被命名为“PVFMDB”。<CFIF>标记表示条件逻辑。本例中，如果没有定义*session.cart*，客户机就开始一个新的会话，此时应该定义*session.cart*；如果已经定义了*session.cart*，就不采取任何行动。这里，可以为应用范围变量和对话范围变量设置一个容许的时间间隔，以便能自动删除被放弃的购物车（图10-6所示的简单代码中没有提及）。

```
<!---Name application--->
<CFAPPLICATION NAME="PVFC" CLIENTMANAGEMENT="TRUE" SESSIONMANAGEMENT="TRUE">
<!---Set application constants--->
<CFSET ODBC_DataSource = "PVFMDB">
<CFSET BG_Color = "ffffcc">
<CFSET Text = "00008b">
<!---Set application variables--->
<CFIF not isDefined("Session.cart")>
      <CFSET Session.cart=ArrayNew(2)>
</CFIF>
```

图10-6　application.cfm

另外一个文件item.cfm如图10-7所示。图中所使用每个CFML标记都用黑体加粗并且用方框括了起来。因此，我们很容易看出CFML标记在概念上与HTML标记的相似之处。每遇到一

个CFML标记，Web服务器就会与ColdFusion服务器联系，以获得对该标记的解释。在这里，我们将逐个介绍5对CFML标记，以帮助读者初步理解CFML标记的本质。ColdFusion购物车示例所包含的其他5个文件也需要一一研究，但这里就不再赘述。

在图10-7中，Box A包含几个CFML标记。首先，<CFPARAM>设定所需要的参数或参数默认值。这里，*item*参数的默认值设为“”。下一个CFML标记<CFIF>设置一个条件逻辑循环，以*item*是否取<CFPARAM>中设置的默认值之外的值作为判断条件。当*item*的值不是默认值“”时，那么就显示一种产品，并使用CFML标记<CFSET>将其存入变量*strItem*中，<CFSET>的作用是给变量赋值。

<CFIF>是建立条件逻辑的一组标记中的一个，并且也是最常用的一个。每个<CFIF>标记必须有与其相匹配的</CFIF>标记。用于比较的标记还有<CFELSEIF>和<CFELSE>。在一个<CFIF>循环中，根据需要可以使用多个<CFELSEIF>标记，但是只能使用一个<CFELSE>标记。Box C包含另外一个条件逻辑的例子，该条件逻辑把所需要项目的信息移入购物车（*cart.cfm*）。

Box A中还有一个标记<CFLOCATION>，它用来把浏览器重定向到一个不同的URL。本例中，浏览器被重定向到应用程序的主页。

Box B包含控制显示的CFML标记<CFOUTPUT>，BODY语句就放置在CFOUTPUT代码块中。因此，每个被声明的变量值（“#”包围的值）都将会被使用。而在*application.cfm*文件中，这些变量的值都被设置为应用程序的常数。

Box D显示了<CFQUERY>的用法，它与数据库连接并选择所需要的某个项目的信息。需要注意的是，代码中指定的数据源是在*application.cfm*中确定的，而*item*的值则是在*search.cfm*中确定的。这个代码块中包括SQL查询，结果返回需要显示的产品信息。由*search.cfm*返回的*item*值赋给变量*strItem*。查询输出作为Box E代码模块的结果显示出来。名为*Item*的查询的结果由<CFOUTPUT>标记格式化，模块中使用HTML代码来格式化查询的输出。

上述关于ColdFusion编码的简短介绍表明：CFML标记与HTML标记非常相似，将它们结合可以很容易创建与数据库连接的动态Web应用程序。目前有将近100个CFML标记可支持传统的编程结构和技术。不过，CFML不需要像传统编程语言那样编码，只需理解HTML就可以了。编写基于Web的定制应用程序有一些非常简便而快速的工具，ColdFusion就是其中之一。

从前面两部分的介绍中，还同时展现了在其他语言中嵌入使用SQL代码的例子。参见图10-5的Box E和图10-7的Box D，我们可以发现在与数据库连接的ASP Web应用程序中嵌入VBScript的SQL语句，以及在与数据库连接的Coldfusion Web应用程序的CFML标记中嵌入的SQL语句。在这两个例子里，SQL代码或者通过ASP服务器或者通过ColdFusion服务器传送到数据库，在那里解释执行SQL语句并返回结果。然而，这种“嵌入式SQL”模式并不需要使用与第三代语言（Third Generation Language，3GL，如C语言）中嵌入式SQL类似的复杂语法。更完整的在3GL中使用嵌入式SQL的内容请参见下节。

```
<!---
Designed by: Michael Alexander
Project: Pine Valley Furniture Demo, ASP
Purpose: Display a specific Product.

This page displays a specific item in its main area.
It also provides an option to link to the search box, and the main area.

Check to see if the user has gotten to this page correctly.
--->
```

图10-7 item.cfm

Box A

```
<CFPARAM NAME="item" DEFAULT="">
<CFIF item IS NOT "">
<!--- We do have a Product, so put it into strItem.--->
        <CFSET #strItem# = #item#>
<CFELSE>
<!---We don't have a Product, so kick back out to main store page.--->
        <CFLOCATION url="store.cfm" addtoken="NO">
</CFIF>
```

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE>Pine Valley Furniture Company Web Store</TITLE>
</HEAD>
```

Box B

```
<CFOUTPUT>
<BODY bgcolor=#BG_Color# text=#Text# link=#Text# vlink=#Text# alink=#Text# >
</CFOUTPUT>
```

```
<TABLE align=center>
        <TR>
                <TD><STRONG><FONT size=6>Pine Valley Furniture Store
                        </FONT></STRONG></TD>
        </TR>
</TABLE>
<HR>
<TABLE>
        <TR align=right>
                <TD valign=top align=center>
                        <B>Search for a product such as bookcase, nightstand, or table:</B><BR>

                        <FORM action="search.cfm" method=post>
                                <INPUT Name="searchval" type=text width=20><BR><BR>
                                <INPUT Value="SEARCH!" type=submit><BR>
                        </FORM>
                        <BR>
                        <A href="store.cfm">Return to Store Front</A><BR>
```

Box C

```
<CFIF not ArrayIsEmpty(Session.cart)>
                <FORM method=post action=cart.cfm>
                        <INPUT type=Hidden name=Quantity value=0>
                        <INPUT type=Hidden name=item value=0>
                        <INPUT type=Hidden name=none value=0>
                        <INPUT type=Submit value="Go to Cart">
                </FORM><BR>
</CFIF>
```

```
                </TD>
                <TD valign=top width=75%>
                        <TABLE align=center>
                        <!---REM Get the information about the Product--->
```

图10-7　(续)

Box D

```
<CFQUERY DATASOURCE=#ODBC_DataSource# Name="Item">
SELECT *
FROM PRODUCT_t
WHERE Product_ID = #strItem#
</CFQUERY>
```

Box E

```
        <CFOUTPUT query="Item">
        <tr>
                <td>Product: </td>
                <td width=75% align=left>#Product_Description#</td>
        </tr>
        <tr>
                <td>Finish: </td>
                <td width=75% align=left>#Product_Finish#</td>
        </tr>
        <tr>
                <td>Cost: </td>
                <td width=75%
                align=left>#DecimalFormat (Standard_Price) #</td>
        </tr>
        <tr>
                <FORM method=post action="cart.cfm">
                <INPUT type=Hidden Name=Item Value=#Product_Id#>
                <td colspan=2 align=center>
                        <INPUT type=Submit Value="Buy It!">
                </td>
                </FORM>
        </tr>
        <tr>
                <FORM method=post action="line.cfm">
                        <INPUT type=Hidden Name=line
                        Value="#Product_Finish#">
                <td colspan=2 align=center>
                        <INPUT type=Submit Value="Go to
                        #Product_Finish# Finish">
                </td>
                </FORM>
        </tr>
        </CFOUTPUT>
                        </TABLE>
                </TD>
        </TR>
</TABLE>

</BODY>
</HTML>
```

图10-7 （续）

10.5.3 嵌入式SQL

我们在第7章和第8章介绍了交互和直接的SQL形式。在前述例子中，输入一条SQL命令并一次执行，即SQL命令构成作业或事务的一个逻辑单元。一些维护数据库有效性所必须的命令

（如ROLLBACK和COMMIT）在大多数交互式SQL状态下对于用户而言是透明的。而另一种在创建客户端和服务器端应用程序时广泛使用的SQL形式称为嵌入式SQL。在3GL中嵌入SQL就是把SQL命令放在3GL宿主程序中的合适位置。所谓3GL是指第一代语言，如ADA、COBOL、C、Fortran、M、Pascal、PL/I等。Oracle提供PL/SQL，即过程化语言SQL（Procedural Language SQL）。它是一种扩展后的SQL的专用语言，其中添加了一些过程化语言的特性，如变量、类型、控制结构（包括IF-THEN-ELSE循环）、函数和过程等。PL/SQL代码块也可以嵌入3GL程序。

我们考虑在3GL中嵌入SQL有以下几个原因。首先是为了创建一种更灵活的、易访问的用户界面。有效使用交互式SQL要求用户对SQL本身、数据库结构有较好的理解，而用户通常都不具备这种能力。许多RDBMS都包含表单、报表和应用生成程序（或者由附加软件提供），但使用这些工具很难完成开发者通常所期望的功能。使用3GL会比较容易达到他们的期望，因为3GL具有对SQL数据库进行嵌入式SQL调用的能力，所以能更好地编写那些需要访问关系数据库的大型复杂程序。

其二是为了通过使用嵌入式SQL改善性能。交互式SQL在每次处理查询时都需要将其转换为可执行的机器代码。但直接式SQL所自动运行的查询优化器可能并没有成功地优化查询，因而导致执行速度缓慢。有了嵌入式SQL，开发人员可以更好地控制数据库访问，因此极大地改善性能。同时依赖SQL转换器和优化器，同时通过程序控制取决于问题的性质，并且必须借助经验和测试来确定最终的解决方案。

使用嵌入式SQL的第三个原因是提高数据库的安全性。DBA通过使用SQL提供的GRANT和REVOKE命令以及创建视图对访问进行限制。相同的限制在嵌入式SQL应用程序中也可以调用，从而提供了另外一层保护。此外，在嵌入式SQL中，更容易完成复杂的数据完整性检查，包括交叉字段一致性检查。

使用嵌入式SQL的程序包含由C或COBOL等3GL编写的宿主程序，以及需穿其中的SQL代码。每个SQL代码块都以关键词EXEC SQL开始，在程序预编译时，该关键词表明嵌入式SQL命令将被转换为宿主源代码。不同的宿主语言需要不同的预编译器，而且必须确保3GL编译器与RDBMS用于每种语言的预编译器兼容。嵌入式SQL程序的处理步骤如图10-8所示。

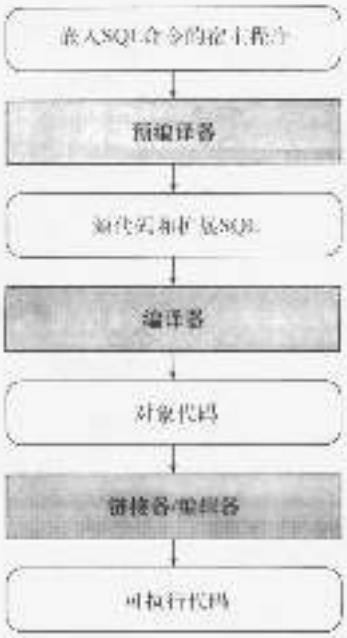


图10-8　嵌入式SQL程序的处理步骤

预编译器碰到EXEC SQL语句时，就把SQL命令翻译成宿主程序语言。有些预编译器会检查SQL语法的正确性并产生错误消息直到SQL语句试图实际执行为止。某些产品的预编译器（DB2、SQL/DS、Ingres）会为SQL语句创建一个单独文件，然后由一个称为联编程序的单独应用程序对它进行处理，这个联编程序确定引用对象是否存在，用户是否拥有运行SQL语句的权限以及将要使用的处理方法等。其他一些产品（Oracle、Informix）则是在运行时解释SQL语句，而并不编译它们。这两种情况下，最后所得到的程序都包含对DMS例程的调用，链接器/编辑器会把它们链接进来。

下面的程序是一个如何在程序中嵌入SQL的简单示例，该程序使用C语言作为宿主语言，

本例中包含一个准备好的SQL语句，叫做“getcust”，它将被编译和存储为数据库的可执行代码。Cust_ID是顾客表的主键。该SQL语句返回和某个订单号相匹配的顾客信息（c_name、c_address、city、state、postcode）。SQL语句里的订单信息使用一个占位符表示，意为输入参数。顾客信息是SQL查询的输出并且通过into子句存入宿主程序的变量中。本例假定查询仅返回一行结果。如果要返回多行结果，必须采用游标写一个循环程序，使得每次返回一个元组并保存起来。游标帮助消除SQL的一次设定处理与过程语言的一次记录处理之间的阻抗失配。

```
exec sql prepare getcust from
    "select c_name, c_address, city, state, postcode
    from customer_t, order_t
    where customer_t.cust_id=order_t.cust_id and order_id=?";
    .
    .
    ./* code to get proper value in theOrder */
exec sql execute getcust into :c_name, :c_address, :city, :state,
:postcode using theOrder;
    .
    .
    .
```

10.6 管理Web数据

数据库对于因特网的重要性不言而喻。随着数据库对于因特网应用程序而言越来越重要，一些传统的数据库问题也日益重要起来。首先就是数据库的安全性问题，Web应用程序与公司外部的用户建立连接并可供该用户访问，这种方式甚至在几年前都未曾听说过。其次是隐私问题，公司可以收集那些访问他们Web站点的访问者的相关信息。如果他们从事的是电子商务活动，在Web上出售产品，那么他们可以收集对其业务有价值的顾客信息。如果公司在顾客不知情的前提下泄露顾客信息，或者顾客认为可能发生了类似情况，都将会引起关于道德和隐私权的争论。最后是关于因特网技术更新速度的问题。技术变化的步伐在不断加快，数据库管理员也面临着巨大挑战：他们必须频繁更新现有产品，并评估和确定新产品对于业务运作的重要性。以下将分别讨论这些问题。

10.6.1 Web安全性问题

用户对网上交易的安全性缺乏信任是抑制电子商务发展的最重要的因素之一。必须解决与Web相关的数据库安全性问题，以保护敏感信息，控制泄露数据的风险。将商业应用程序与公众网络（如因特网）链接起来时要求必须妥善处理所有已经出现的安全隐患。在处理一个需要返回数据库信息并加以显示的客户端请求时，必须严格限制所能得到的信息。通过因特网连接发送请求的客户端不可以直接访问数据库。

整个系统都必须在网络级、操作系统级、Web服务器级以及数据库级等各个级别采取安全性措施。在各个区域都应该保持警惕，如果要访问敏感数据则必须通过所有的安全性检查。同时，由于业务/安全环境的频繁改变可能会使得先前安全的站点也受到非法入侵的攻击，所以定期监控以及进行安全测试也必不可少。图10-9展示硬件的安全措施，它可以放在除了防火墙以外的任何位置，作为抵御非法入侵的第一道防线。

图10-9中除了包含先前讨论过的防火墙以及图10-1所述的安全措施之外，还添加了**路由器**（router）符号与**非法入侵检测系统**（Intrusion Detection System, IDS）符号。路由器是放置在通信网络中的中间设备，用来传输消息包，并以最有效的路径将消息包转发到正确的目的地。

而非法入侵检测系统则试图识别非法进入计算机系统或误用计算机系统的企图。IDS可以监控网络上传输的消息包，监控系统文件、日志文件，还可以设置欺骗系统，通过众所周知的安全黑洞尝试诱捕黑客。除此之外，还应注意在图10-9中，Web服务器与业务或内部主机是相互隔离的，同样，数据库服务器与Web服务器之间也应该相互隔离。

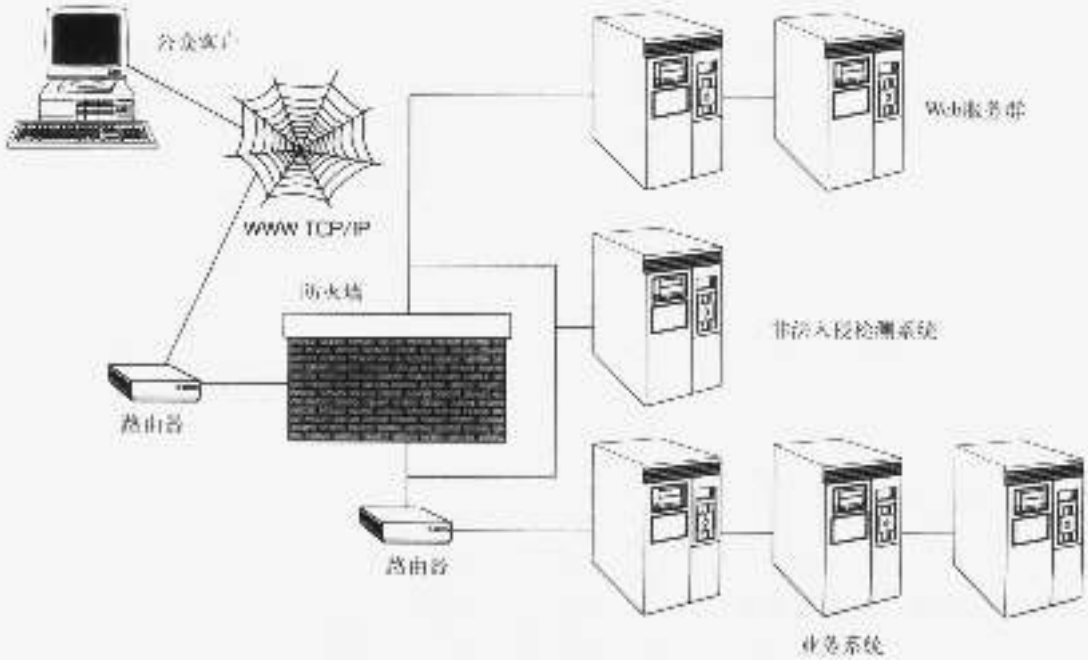


图10-9　建立因特网安全措施

1. Web安全计划

通过执行一个正式的、书面的、被管理人员所认可的风险评估，为启用Web的数据库应用程序建立安全机制。风险评估至少应考虑以下因素：

- 风险的本质。
- 风险的可能性。
- 对组织在运作、财务及法律上潜在的影响。
- 存在某些人有可能破坏安全措施的情况。

在没有从其他相关人员那里获得信息的前提下，数据库管理员自己不应进行风险评估。风险评估应委派给那些有能力成功完成它的人去进行，而且他们还能够将风险评估的内容传达给相应的管理层。这样做的结果是由管理层而不是由数据库管理员负责处理风险，从而可以起到保护数据库管理员的作用。

2. 网络级安全

充分保护因特网环境里的数据库还需要保护与之通信的网络。

- 如果必须与内部系统相连，则Web服务器和数据库服务器应处在与其他业务系统不同的局域网中，而且它们之间应该设置防火墙，防火墙设置在Web服务器和数据库服务器之间以及它们与业务系统之间，防火墙仅仅行使防火墙的功能，并且受口令保护。
- 应尽量减少连网服务器中硬盘的共享。
- 应记录网络和防火墙日志，使任何改动都不易发生；并且要对它们进行定期监控，以发

现任何未经授权或不寻常的举动。

- 安装并定期监控用于检测来自外部的探查和攻击的软件。执行该监控任务的人员应定期访问软件厂商的站点和安全性Web站点，以便发现新的安全性漏洞并尽快修补。

3. 操作系统级安全

正如为了保护数据库和内部业务系统而建立网络级安全一样，还必须建立充分的操作系统级安全。

- 通过及时访问软件厂商的站点和安全性Web站点，对新发现的漏洞应用补丁，从而修补所有已知操作系统的漏洞。
- 安装反病毒软件，在机器启动、下载文件或接收电子邮件时，自动检查病毒。
- 监控服务器日志，同监控网络日志一样对对未经授权的活动进行监测。安装和监控IDS软件，以检测外部的探查和攻击。
- 在因特网环境里，禁止任何不必要但可能会向未经授权的用户提供访问的服务。例如，发送邮件服务，虽然它已被新的电子邮件程序取代，但仍然嵌入在操作系统中，因此应该被禁止。

4. Web服务器安全

每个Web服务器都应进行良好的配置以使非法访问难以进行，最好是不发生非法访问。不过，层出不穷的新病毒和黑客使大家认识到要想达到完全的系统安全是不可能的。因此，要注意设置Web服务器以尽可能地限制访问。

- 尽可能地限制Web服务器上用户的数量，以及超级用户或管理员的数量。只有授权用户才能安装软件、编辑或添加文件。
- 限制对Web服务器的访问，尽量减少开放的端口。使打开的端口数量最少，最好只开放http和https。
- 卸载任何在安装服务器时自动安装但并不需要的程序。比如演示程序就可能向黑客提供他想要的访问。编译器与解释器（如Perl）不要放在可从因特网直接访问的路径上。CGI有潜在的安全性问题，所以最好在必要时再运行它，并且应把所有的CGI脚本限制在一个子目录当中。
- 如果Web服务器运行在UNIX之上，则必须确保不使用root安装默认的操作系统，而是从仅有最小权限运行Web服务器软件的用户账号下安装。

数据库级安全将会在第12章中详细讨论。

10.6.2 隐私问题

使用因特网时保护个人隐私已经成为一个重要问题。电子邮件、电子商务与营销，以及其他在线资源都可以产生新的以计算机作为媒介的交流方式，而新的与个人隐私相冲突的问题也随之产生了。许多组织都对人们的因特网行为感兴趣，包括雇主、政府机关以及企业。返回个性化回答的应用程序需要收集个人信息，但同时必须尊重员工、市民以及顾客的隐私和尊严。这是一个简单但难以操作的概念。

用户必须保护自己的隐私权，因此需要了解所使用工具会对个人隐私造成何种影响。例如，在使用浏览器时，用户可以选择接受或者拒绝在他们的机器中放置cookie。为了作出符合自己意志的选择，用户必须了解几件事情。首先，他们必须知道cookie，理解它们的概念，从而能够判断自己是否接收定制化信息，并学会如何设置机器来接受或拒绝cookie。浏览器和Web站点还不具备帮助用户快速理解上述知识的能力。随着利用Web进行交流、购物以及其他应用的发展，隐私的滥用（比如出售由cookie收集得来的客户信息）促使我们开始深刻认识隐私问题

的重要性。

在工作中，用户必须认识到通过雇主的机器和网络所进行的通信并不是保密的。法院支持雇主有权监控所有职员的电子通信。

因特网本身并不保证通信隐私。常用软件中所采用的产品加密、邮件匿名发送、内置安全机制等措施有助于保护隐私。目前，保护私人拥有和运行的计算机网络已成为信息基础设施的一个非常关键的部分，它对于发展Web上的电子交易、银行、医疗和传输应用有重要的意义。

W3C提出一个数据隐私标准（Platform for Privacy Preferences, P3P），它传达Web站点所规定的隐私政策并将其与用户自身的政策偏好作比较。具体地说，P3P在Web站点服务器上采用的XML编码可以被任何配备P3P的浏览器或插件自动取走。然后，客户浏览器或插件比较站点的隐私政策和用户的隐私偏好，并把差异或矛盾告知用户。P3P主要处理如下几方面的在线隐私：

- 谁在收集数据？
- 收集哪些信息、为了何种目的而收集这些信息？
- 哪些信息可以与他人共享，共享信息的人是谁？
- 用户可以更改收集者使用他们数据的方式吗？
- 如何解决争端？
- 保留数据应该制订什么政策？
- 在哪里可以找到站点详细政策？

匿名（anonymity）是在压力下出现的因特网通信的另一个重要方面。美国法律保护匿名权利，但还是要求聊天室和电子邮件论坛显示用户名，即便他匿名张贴消息。1995年，欧洲议会表示将切断与任何缺乏充分隐私保护国家之间的数据交换，从而使美国也达成协议，将对欧洲客户提供同欧洲企业一样的隐私保护。这可能会促使美国国会建立比先前颁布的法案更具保护性的新法案。

FBI的电子邮件监控设备DCS-1000（原来叫做Carnivore）安装在电子邮件服务器上，它可以跟踪电子邮件的标题但并不跟踪电子邮件的内容。大家发现，DCS-1000具有安全黑洞，会破坏公司的技术基础设施或者危及隐私。美国公民自由联盟正试图使美国国会注意，现有隐私法对于因特网通信来说已经过时。尽管政府修订针对因特网通信的隐私条款的速度非常缓慢，但个人和企业在建立所需的隐私级别时，应确保自己理解当前的权利和责任。

10.6.3　因特网技术的更新速度问题

从本章可以看到，硬件、软件与电信技术的更新在信息时代发展的道路上起着重要作用。在过去的20年里，很少有公司只实现一个信息系统并长期依赖于这个信息。变化的速度和广度已经使许多公司难以招架。由于设计周期缩短，在这样一种动态环境中竞争变得非常具有挑战性，公司也不得不彻底改造自身，并尝试用环境、经验、和技能装备他们的员工以便在不断变换的经济和文化中使公司繁荣发展。

在过去的20年中，信息技术的变化速度不断加快。要实现概念简单而视觉优美的、完整而分布的系统是非常困难的。要时刻牢记“细节才是罪魁祸首”这句俗语。信息技术的规划已经从独立的战术活动演变为组织的策略规划的中心。员工和顾客对相关的、易用性系统的期望都呈现极大的上升趋势。在市场中，有效的IT系统已成为成功的关键因素。而数据库技术是满足这些期望并支持业务策略的最基本要素。

在数据库取代了平面文件之后，创建一个更为集成的数据管理环境的工作就开始了。关系数据库技术解决了在一个位置记录数据而在不同的应用程序中使用这些数据的问题。面向对象数据库改善了响应时间，因为请求对象时不需要重新构造这些对象。

现在，期盼已久的计算技术与电信技术的结合正在转变人们获取及共享信息的方式。传统的技术也不再像以前那样各行其是。蜂窝电话和个人数字助理（Personal Digital Assistant, PDA）都可以与因特网相连。人们可以通过寻呼机答复电子邮件，即使在开会时也不受影响。电视通过无线电波、电缆和卫星传送电视节目。这些传统上相互独立的技术通过结合创造了崭新的商业机会，同时也需要出台新的政策和法规来控制公共服务，如无线电波和公众网络等。随着数据库访问的新方法及其定位的建立，数据库安全问题也开始涌现出来。

同时，组织的关系变化也促使数据库管理员想办法处理新的业务形式。合作模型是竞争模型的有益补充。当几个公司为完成同一个项目而协同工作、在项目完成后才分开时，访问共享数据库的问题也变得日益重要起来。

商业的全球化速度也以前所未有的速度在加快。各大公司有策略地在全球范围内部署自己的团队，保证至少有一支团队在工作。IT经理也被赋予全球管理的责任，如管理距离较远的纽约小组、墨尔本小组和伦敦小组。此时也不能为数据库备份和调整而停机，而是要求能够一直访问数据库。

面对不断加快的更新速度，数据库管理员必须主动参与调整IT结构使之符合企业战略的活动。因此除了理解技术基础结构和数据库体系结构外，数据库管理员还必须对商业有一定了解，还要培养自己的领导技能（包括交流与聆听技巧）、对上层的影响能力以及对员工的管理能力等。成功的IT经理不再是只有较深技术背景的人，而且要有把握商业环境中的快速变化的能力。

本章小结

本章意在阐明数据库与因特网应用之间的配合，内容包括各种因特网数据库体系结构，解释了针对Web数据库的ASP和ColdFusion的使用，并讨论与之相关的管理问题。Web环境的若干特征（包括浏览器界面的简洁性及功能的相似性）支持因特网与企业内部网的商务应用得以快速采纳和实现。浏览器软硬件的独立性也使跨平台的信息共享变得容易起来。开发成本与开发时间都有所减少。

因特网环境包括与客户端工作站相连的网络、Web服务器和数据库服务器，以及它们遵循的TCP/IP协议。每台与因特网连接的计算机必须有各自不同的IP地址。在简单的体系结构中，客户端浏览器发出的请求通过网络传送到Web服务器。如果该请求需要从数据库中获取数据，那么由Web服务器构造一个查询并将其传送至数据库服务器，数据库服务器处理查询并返回结果集。通常采用防火墙限制外部用户对公司内部数据的访问。

因特网体系结构的常用组件有：某种编程或标记语言、Web服务器中间件（服务器端扩展和Web服务器界面）、Web服务器、客户端扩展。为了帮助读者理解如何将数据库连接到Web页面，本章给出一个简单的购物车应用程序，分别采用ASP和ColdFusion来实现这些应用。

某些传统的数据库问题又引起人们的极大关注，因为数据库对于因特网应用而言变得越来越重要。这些问题包括安全问题、隐私问题、以及处理商务和技术两方面的快速更新问题。必须在网络层、操作系统层、Web服务器层和数据库层采取相应的安全措施。用户必须保护自己的隐私权，而且需要了解自己所使用的工具在隐私方面有哪些规定。匿名问题也仍需要加以解决。

最后，信息技术的变化的速度日益加快，我们所有人都必须努力使技术的变化与业务实践的变化相匹配。

虽然在某些方面数据库技术已相当成熟，但数据库对于因特网和电子商务的开发有极为重要的作用，因为数据库总使得它们处于技术变化的前沿。本章仅介绍了Web数据库的基本概念，

有兴趣的读者可以找到更多的资料以全面了解当前该领域的最新动态。

本章复习

关键术语

ActiveX	电子业务	代理服务器
浏览器	电子商务	逆向代理
B2B	可扩展标记语言(XML)	路由器
B2C	防火墙	服务器端扩展
层叠样式表(CSS)	超文本标记语言(HTML)	软硬件负载平衡
ColdFusion标记语言(CFML)	非法入侵检测系统(IDS)	标准通用标记语言(SGML)
公共网关接口(CGI)	Java	VBScript
cookie	Java servlet	万维网(WWW)
DCS-1000 (Carnivore)	JavaScript	万维网联盟(W3C)
域名服务器平衡	插件	XHTML

复习问题

1. 定义下列关键术语。

a. ColdFusion标记语言（CFML）　b. cookie
c. DNS平衡　d. 电子业务
e. 防火墙　f. 服务器端扩展
g. W3C

2. 将下列术语和定义匹配起来。

_____浏览器　a. 用于多Web服务器的负载平衡方法
_____电子业务　b. 万维网联盟
_____防火墙　c. 限制对公司内部数据访问的软件或硬件
_____URL　d. 显示HTML文档的软件
_____W3C　e. 在其他应用程序中执行并储存在服务器上的小程序
_____XHTML　f. 确定Web服务器的IP地址和通信协议的Web地址
_____servlet　g. 采用因特网技术的技术业务
_____逆向代理　h. 扩展HTML语言使之与XML兼容的脚本语言

3. 比较下列术语。

a. 电子商务；电子业务　b. 因特网；企业内部网；企业外部网
c. HTML；XML；XHTML　d. DNS平衡；软硬件负载平衡；逆向代理
e. HTML；SGML

4. 解释为什么将数据库与Web页面相连对于促进电子业务非常重要。
5. 建立因特网数据库连接所必须的环境组件有哪些?
6. 阐述万维网联盟的宗旨和业绩。
7. 数据库中间件（如ColdFusion和ASP）的作用是什么?
8. 比较和对比Web服务器界面的差异，包括CGI、API以及Java servlet。
9. 描述三种平衡Web服务器负载的方式。
10. 解释什么叫cookie，用作因特网术语时，它是如何使用的?
11. 阐述如何使用ASP或ColdFusion。

12. 描述在网络层、操作系统层和Web服务器层为确保Web数据库安全所应采取的安全措施。

13. 简要描述一下W3C所提出的数据隐私标准（P3P）。

问题和练习

1. 阐述静态Web站点与动态Web站点之间的区别。动态Web站点的哪些特征能够更好地支持电子业务的发展？

2. 本章列举了九种常见的内部网服务。请你指出哪些服务是任何企业内部网所必须的，哪些是可选的，并说明理由。

3. IP地址的每个组成部分的含意是什么？以24.24.27.15为例加以说明。（注意：该题的详细答案未在本章中介绍。）

4. 列举与因特网应用程序开发有关的语言，并根据它们的功能进行分类。（注意：不一定要与本章所采用的分类方案相同。）

5. 在学习和测试了本章所提供的ASP和ColdFusion购物车案例后，根据自己的安装和使用经验对ASP和ColdFusion做一个简单的比较。在比较时，要考虑到自己已有的编程背景。

6. 当Web与数据库相连时，为什么需要在四个层次（网络、操作系统、Web服务器、数据库）上采取安全措施？

7. 讨论因特网上用户匿名的优点和缺点。

应用练习

1. 以自己的PC为例，找到已经下载的插件，并说明它们的用途。这些插件是否与操纵或显示来自数据库的数据有关？

2. 找到cookie下载后的存储位置，按目录组织它们。如果你决定删除某个cookie，请描述你的决定过程。你是不是对自己机器上的cookie数量感到惊讶？

3. 如果你要在自己的Web站点上或者在你的公司的对外Web站点上使用ASP或ColdFusion，那么你应该做些什么？

4. 假如你要连接一个数据库到Web站点上，请为你的Web站点做一个风险评估。

5. 如果将数据库连接到公共站点上，根据你的工作列出一个进行风险评估所要采取的措施。如果可能的话，实际实现该风险评估。

6. 根据个人兴趣，使用ColdFusion或ASP连接一个数据库到你的个人Web站点上。首先进行本地测试，然后将其移至公共站点。

7. 获得一份你所处工作位置的计算机隐私策略的复印件，为了符合该策略的要求，确定在你使用计算机工作时是否有需要改变的地方。

参考文献

Forta, B. 1998. *The ColdFusion 4.0 Web Application Construction Kit*. Indianapolis, IN: Que.

Merkow, M. S., and j. Breithaupt. 2000. *The Complete Guide to Internet Security*. Detroit, MI: AMACON Books.

Morrison, M., and J. Morrison. 2000. *Database-Driven Web Sites*. Cambridge, MA: Course Technology.

Prague, C. N., and M. R. Irwin. 1999. *Microsoft Access 2000 Bible*. Foster City, CA: IDG Books Worldwide, Inc.

Schneier, B. 2000. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd Edition. New York: John Wiley & Sons.

Zwicky, E. D., D. Russell, S. Cooper, and D. B. Chapman. 2000. *Building Internet Firewalls*,

2nd Edition. Sebastopol, CA: O'Reilly & Associates.

Web资源

- http://hoohoo.ncsa.uiuc.edu/cgi/ 完整的来自NCSA的CGI规范的信息。
- http://www.w3.org/CGI/ W3C关于CGI的页面。
- http://www.w3.org/MarkUp/ W3C关于HTML和XHTML的主页。
- http://www.netcraft.com/survey/ Netcraft Web Server Survey跟踪Web服务器的市场份额以及SSL站点操作系统。
- http://webservercompare.internet.com/webbasics/index.html 关于Web服务器的教程。
- http://staff.plugged.net.au/dwood/xmlc/index.html XMLC的教程，XMLC是一种基于Java的编译器，可以从HTML或XML文档中创建Java类，从而可以重新创建文档。
- http://xml.apache.org/cocoon/ Java Web发布的框架，它将文档的内容、形式和逻辑分离，各部分可以独立地设计、创建和管理。
- http://www.vbxml.com/xsl/tutorials/intro/default.asp XSLT教程，XSLT可以将XML文件转换成HTML文件或其他基于文本的格式。
- http://www.cve.mitre.org/ Common Vulnerabilities and Exposures（CVE）是一个已知漏洞及其他信息安全隐患的标准名字列表，这些常见漏洞由CVE Editorial Board确认，并由MITRE公司监控。CVE的目标是对所有已知的普遍漏洞和安全隐患的名字进行标准化。
- http://www.computerprivacy.org/who/ Americans for Computer Privacy (ACP)是一个由100多家公司和40多个协会组成的机构，这些公司和协会代表金融服务业、制造业、电信业、高科技和运输业，以及法律实施、公民自由、pro-family和纳税人组织，他们共同关心计算机隐私问题。
- http://www.cpsr.org/ Computer Professionals for Social Responsibility (CPSR)是一个由计算机科学家和其他关心计算机技术对社会影响的人们所组成的公众联盟。
- http://www.epic.org/ Electronic Privacy Information Center（EPIC）。

项目案例：山景社区医院

医疗机构采用电子形式维护它们的记录的速度比其他行业要慢得多。不过，现在通过计算机处理所产生的常见数据，比如化验结果，已经可以通过计算机访问了。然而，对于大多数医院而言，许多信息仍然不能以电子形式获取。例如，体检结果、医嘱和护士记录以及病人和家族病史等信息在大多数医院不能以电子形式获取。不同医疗机构和供应商所开发的获取数据的信息系统把这些信息分隔成了碎片，阻挠了本来很容易达到的系统共享的开发进程。在哪里开发医疗系统，就能在哪里看到效益。比如一个对处方进行记录的信息系统将由错误用药、错误剂量或药物反应所引起的问题减少了55%。

项目描述

山景社区医院的规划小组认为医院可以通过采用基于Web的解决方案改善其日常运作。他们去了解了一些相关的系统，这些系统的作用在局部安装后已得到证实。但是，他们同时发现这些系统并不真正适用于自己的医院，因为系统的性价比很难让人接受。

在考虑如何为山景社区医院的医疗系统开发一个浏览器界面时，应该解决以下几个问题：

- 首先，安全性是第一个要关注的问题。病人的健康信息由于其敏感性需要高度保密。
- 数据的录入问题也很重要。医生、护士以及其他医疗人员必须能够而且也愿意将数据录入所提供的信息系统。
- 如果决定要实现一个Web系统，系统的可用性也很关键。
- 将基于浏览器的系统与现有系统集成起来有多大困难？
- 山景社区医院如何向医疗机构的财务部门证明所提出的系统的性价比较高？
- 山景社区医院如何预测在工作中可能会发生的模式变化？
- 随着系统的变化，组织的政策和程序需要作出何种相应的变化或更改？

项目问题

1) 请描述在计划委员会作出任何关于在Web上提供更多对病人而言至关重要的信息的决定时，应考虑的隐私问题的范围和性质。

2) 请描述计划委员会应考虑的数据录入问题。确保数据录入由医生、护士和其他医疗人员完成，提出能解决这些问题的可能方法。

3) 医疗中心的专家希望病人能更加积极地参与自己的医疗中心的建设。请找到至少五个可访问的医疗Web站点，比较每个站点所提供的信息和服务。提出服务建议，通过在山景社区医院的Web站点加入这种服务，使病人更加积极地关注医疗中心的建设。

4) 利用计算机记录病人的病历有哪些优点和缺点？请列出你所认为的优点和缺点。

项目练习

山景社区医院计划委员会正在考虑三种可以放到在线界面上的业务功能。具体如下：

1) 安全性和保密性方面：谁可以访问这些数据，如何限制访问，所提出的安全系统如何在两者之间折中？

2) 数据录入需求：哪些工作人员录入数据，让每种人员录入数据的阻力有多大，如何处理这种阻力？

3) 山景社区医院所期望的收益：成本是多少。

下面考虑有三种可能性：

1) 在线提交保险索赔。

2) 在线提供病人的临床信息。

3) 在线实现供应链管理。

这三种可能性里你认为应该最先实现哪一条？为什么？

第11章 数 据 仓 库

11.1 学习目标

学完本章后，读者应该具备以下能力:

- 定义下列关键术语: 数据仓库、运作系统、信息系统、数据集市、依赖性数据集市、独立数据集市、企业数据仓库、运作数据存储、主动数据仓库、调和数据、导出数据、事件、临时数据、周期数据、静态抽取、增量抽取、数据清洗、刷新模式、更新模式、数据转换、选择、联结、聚合、星型模式、粒度、购物篮分析、一致性维度、雪花模式、OLAP、ROLAP、MOLAP、数据挖掘和数据可视化。
- 为什么往往在信息管理者的需求和可得到的信息之间存在“信息鸿沟”，给出两条重要原因。
- 为什么当今大多数组织都需要数据仓库，列举两条主要原因。
- 命名并简要描述数据仓库的三层体系结构。
- 列出数据调和的四个主要步骤。
- 描述星型模式的两个主要组成部分。
- 给定合理的数据库维度，估算一个事实表的行数和总的字节数。
- 设计一个数据集市，使用不同的方案对维度进行规范化和非规范化，说明事实的历史记录并改变维度的属性值。

11.2 引言

众所周知，易于获得的高质量的信息在当今的业务中起着至关重要的作用。让我们看看两个数据管理专家的有关陈述:

> 在今天的业务环境中，信息是最为关键的。成功取决于及早而果断地使用信息。缺乏信息就意味着失败。迅速变化的业务运作环境要求对数据进行更为及时的访问(Devlin, 1997)。

> 许多企业正在积极寻求新技术，以帮助他们获得更高的利润并且更具竞争力。要想获得竞争力，就要求这些公司加快他们制定决策的速度，以便于对变化作出快速反应。加快决策速度的关键就是在正确的时间拥有易于访问的正确信息(Poe, 1996)。

根据上面所强调的信息的作用和近来信息技术的发展，你可能以为大多数组织都拥有开发完善的系统并且能够向管理者和其他用户提供信息。然而，这种情况却并不多见。实际上，尽管有海量的数据和众多的数据库，但组织只掌握他们所需要信息的一小部分。管理者常常为不能访问或使用他们所需要的数据和信息而倍感困扰。

现代组织淹没在数据当中但却渴望信息。尽管这句话只是一个比喻，但却非常形象地描述了许多组织的现状。为什么企业会陷入这种境地呢? 有两条重要原因导致大多数组织出现信息鸿沟。

出现信息鸿沟的第一条原因是，多年以来，大多数组织一直以分散的方式开发信息系统及其支持系统的数据库。本书强调系统开发应该在仔细规划的结构化方法基础之上展开，并产生

一个集成的数据库集合。然而，在现实中，由于时间和资源的约束，大多数组织采取“一次做一件事情”的方式开发信息系统的一部分。这种方法必然会产生一个不协调的并且经常是不一致的数据库。通常，数据库基于不同的软硬件平台。在这种环境中，对于管理者而言，要想准确地定位并使用信息几乎是不可能的，必须跨越不同的记录系统进行组合。

产生信息鸿沟的第二条原因是，大多数系统是为了支持运作处理而开发的，很少或没有考虑到制定决策所需的信息或分析工具。**运作处理**（operational processing）也叫做事务处理，用来获取、存储并操纵支持组织日常运作的数据。而**信息处理**（informational processing）则是分析数据或其他信息形式以支持决策的制定。这两种类型处理所需要的数据类型和所执行的处理截然不同，下一节将会详细解释这两种处理的区别。大多数自行开发的系统或直接从软件供应商那里购买的系统都是为支持运作处理而设计的，很少会考虑到信息处理的需求。

数据仓库（data warehouse）是弥补信息鸿沟的桥梁，它合并和集成来自许多内部和外部资源中的信息，以作出准确的业务决策为目的来安排它们（Martin, 1997a）。通过利用诸如趋势分析、目标营销、竞争性分析、客户关系管理等应用程序，它们能够帮助高级管理人员、经理以及业务分析人员作出复杂的业务决策。数据仓库已经朝着满足上述需要但并不打乱现有运作处理的方向发展。

由于数据仓库是一个涵盖诸多内容的主题，要想全面解释它需要一本书的篇幅，所以本章仅对其作简要介绍。事实上，大多数关于该主题的著作都只是论述其中的某一个方面，比如数据仓库设计或管理。本章的内容主要集中在与数据库管理相关的两个方面：数据仓库体系结构和数据库设计。在本章中，首先将学习数据仓库是如何与现有的运作系统相关联的。接着将介绍三层数据体系结构，这种结构对于大多数数据仓库环境都非常合适。然后，我们着重介绍从现有运作系统抽取数据的问题，以及如何将这些数据装入数据仓库。接下来，我们讨论在数据仓库中经常用到的数据库设计元素。最后，阐述用户如何与数据仓库交互，内容包括联机分析处理、数据挖掘和数据可视化。

11.3 数据仓库的基本概念

数据仓库（Data Warehouse, DW）是一种面向主题的、集成的、随时间变化的、不可更新的数据集合，用于支持决策制定过程管理和业务的智能化（Immon和Hackathorn, 1994）。该定义中的关键术语的含义如下：

1) *面向主题* 数据仓库围绕企业的关键主题（或高级实体）而组织，这些主题包括客户、病人、学生、产品和时间等主题。

2) *集成* 数据仓库中存放的数据采用统一的命名规则、格式和编码结构进行定义，相关特征从若干内部的记录系统及组织外部的资源中收集得来。

3) *随时间变化* 数据仓库里的数据包含一个时间维度，因此可用来研究趋势和变化。

4) *不可更新* 数据仓库里的数据从运作系统中装载和更新，而不能由终端用户来更新。

数据仓库不仅仅只是组织中所有运作系统的合并。因为它主要关注业务的智能化、外部数据及随时间变化的数据（不只是当前状态）上，所以数据仓库是一种独特的数据库。

数据仓库是一个过程，组织通过使用数据仓库从所拥有的信息资源中抽取其含义，并最终制定出决策（Barquin, 1996）。数据仓库是信息技术中的新成员。虽然它在15年前才出现，但其飞速发展已使其成为信息系统的热点领域之一。研究显示，超过90%的大型企业或者拥有数据仓库或者正在创建数据仓库。1996年出台的一份对62个数据仓库项目的研究显示，项目的平均投资回报率为321%，平均投资回收期为2.73年。

11.3.1 数据仓库的历史简介

作为信息系统领域在过去几十年里不断发展的结果，数据仓库出现了。其中有如下几个关键的进步：

- 数据库技术的进步，尤其是关系数据模型和关系数据库管理系统（RDBMS）的发展。
- 计算机硬件的进步，尤其是大容量存储器和并行计算机体系结构的出现。
- 终端用户计算的出现以及与之相辅的功能强大的、直观的计算机界面和工具。
- 中间件产品的进步，使跨异构平台的企业数据库连接成为可能。

对运作（或事务处理）系统（有时也叫做记录系统，因为它们的作用就是维持组织中官方的、合法的记录）和信息（或决策支持）系统之间本质区别的认知（以及随后的定义）是促进数据仓库发展的关键发现。基于这种差异，Devlin和Murphy在1988年发表了第一篇描述数据仓库体系结构的文章。1992年，Inmon出版了第一本描述数据仓库的书，后来他成为该领域最多产的作家。

11.3.2 为什么需要数据仓库

大多数组织需要数据仓库的原因主要有以下两个：

1) 在业务上需要一种集成的、拥有高质量信息的公司范围的视图。

2) 为了显著改善管理公司数据时的性能，信息系统部门必须将信息系统从运作系统中剥离出来。

1. 需要公司范围的视图

通常情况下，运作系统中的数据是分割开并且不一致的。而且，它们还分布在各种不相兼容的软硬件平台上。例如，一个包含顾客数据的文件可能存放在一台基于Unix的运行Oracle DBMS的服务器上，而另一个文件则位于一台运行DB2 DBMS的IBM主机上。但是为了进行决策，有必要提供一个联合的信息视图。

为了便于理解获得一个联合的视图的难度，请参见图11-1所示的简单例子。图中显示来自三个不同记录系统的三张表，每张表都包含一些学生数据。表STUDENT_DATA来自班级注册系统，表STUDENT_EMPLOYEE来自个人系统，而表STUDENT_HEALTH来自健康中心系统。每张表都包含有一些与学生相关的惟一的数据。然而，每张表都包含的公共数据（如学生姓名）却以不同格式存储。

假设要为每个学生构造一张简表，在其中以一种文件格式合并所有的数据，那么需要解决如下问题：

- **不一致的键结构**　前两张表的主键是学生的社会安全号，而第三张表的主键是学生的姓名（Student_Name）。
- **同义字**　在表STUDENT_DATA中，主键被命名为Student_No，而在表STUDENT_EMPLOYEE中，主键被命名为Student_ID（本书第5章论述了如何处理同义字）。
- **自由形式字段与结构化字段的差别**　在表STUDENT_HEALTH中，Student_Name是一个单独的字段，而在表STUDENT_DATA中，学生名（复合属性）被分解为几个组成部分：Last_Name、MI和First_Name。
- **不一致数据值**　Elaine Smith在表STUDENT_DATA中有一个电话号码，在表STUDENT_HEALTH中又有一个不同的号码。这是个错误，还是他本来就有两个电话号码呢？
- **缺失数据**　Elaine Smith在表STUDENT_HEALTH中Insurance字段的值被遗漏了（或者是空值）。该如何确定这个值呢？

STUDENT_DATA

Student_No	Last_Name	MI	First_Name	Telephone	Status	···
123-45-6789	Enright	T	Mark	483-1967	Soph	
389-21-4062	Smith	R	Elaine	283-4195	Jr	

STUDENT_EMPLOYEE

Student_ID	Address	Dept	Hours	···
123-45-6789	1218 Elk Drive, Phoenix, AZ 91304	Soc	8	
389-21-4062	134 Mesa Road, Tempe, AZ 90142	Math	10	

STUDENT_HEALTH

Name	Telephone	Insurance	ID	···
Mark T. Enright	483-1967	Blue Cross	123-45-6789	
Elaine R. Smith	555-7828	?	389-21-4062	

图11-1 异构数据示例

这个简单的示例说明在开发一个联合视图时，未能抓住该任务的复杂性。在现实生活中，我们常常会遇到几十（甚至数百）个文件和成千上万条（甚至是几百万条）记录。在本章后面部分还要详细讲述复杂度和尺寸问题。

为什么组织要把不同的记录系统中的数据集合到一起呢？当然，其根本原因是希望提高利润、更富有竞争性或者通过增加顾客价值而使企业增长。可以通过加快决策的速度和灵活性、改善业务过程或者通过更清楚地理解顾客行为而达到上述目的。对于前面的学生例子而言，大学的管理者可能想调查学生的健康或在校园里的学习时间是否与他们的学业成绩有关，选修某门课程是否与学生的健康有关，或者成绩不佳者是否需要更多支持，比如，由于增加的卫生保健费用以及其他费用。一般而言，组织的某种趋势会促进对数据仓库的需求。这些趋势包括以下几个方面:

- **没有记录系统** 几乎没有哪一个组织只有一个数据库。好像很奇怪，不是吗？还记得我们在第1章里讨论过的关于比较数据库与单独的文件处理系统的原因吗？因为在不同的运作设置下需要异构的数据，因为公司合并和收购，或者纯粹是因为组织的规模较大，所以并存着多个运作数据库。
- **多个系统没有同步** 要使单独的数据库达到一致是极其困难的，既使是由同一个数据管理员控制的同样的元数据（参见第13章），相同属性的数据值也不会一致。这是因为各数据库的更新周期不同，而且是在不同的位置为每个系统捕获同样的数据。因此，要想得到一个组织视图，必须在一个附加数据库中定期地合并和同步来自不同系统的数据。实

际上，我们将可能看到两个这样的联合数据库——一个称为运作数据存储，另一个则称为企业数据仓库，两者都包含在数据仓库的主题下。

- **组织希望以一种平衡方式来分析某些活动**　许多组织已经实现了某种形式的平衡记分卡——同时展示该组织在财务、人力资源、客户满意度、产品质量及其他指标的成果的度量手段。为了确保这个组织的多维视图能够显示一致的结果，必须有数据仓库。平衡记分卡出现问题时，与数据仓库协同工作的分析软件可用来以“下钻”（Drill-Down）、“切片和切块”、可视化以及其他方式挖掘业务智能。
- **客户关系管理**　各行各业的组织都认识到拥有一个与客户在所有接触点上交互的完整视图是非常有价值的。不同的接触点（比如，对银行来说，这些接触点包括ATM、出纳员、资金电子转账、投资证券组合管理以及贷款等）由不同的运作系统提供支持。因此，如果没有数据仓库，当银行出纳员的屏幕上自动显示出一大笔反常的存款业务时，他可能不知道去尝试越区向客户推销一种银行的共有基金。或者，一家信息技术公司为了保障客户利益，给所有寻求帮助的电话区分优先级；销售人员也想为了保障客户利益建立客户联系计划。在给定客户的前提下，获得公司所有活动的完整视图需要从不同的运作系统中合并数据。
- **供应商关系管理**　对许多组织而言，供应链管理也已成为降低成本和提高产品质量的至关重要的因素。这些组织希望在他们与供应商之间活动的完整视图的基础上，从开账单到确定交货日期、质量控制、价格、售后支持等方面建立战略性的供应商合作关系。这些不同活动的数据可能锁定在不同的运作系统当中（如应付款，发货和收货，生产进度安排，维护）。企业资源规划（Enterprise Resource Planning, ERP）系统通过把这些数据放进一个数据库来改善这种状况。然而，ERP系统主要用于优化运作，而不是进行信息或分析处理。

2. 需要区分运作系统和信息系统

运作系统（operational system）基于当前数据，实时执行某种业务。运作系统的例子有销售订单处理、预约系统、病人登记系统等。运作系统必须处理大量相对简单的读/写事务，同时作出快速反应。运作系统也称为记录系统。

信息系统（information system）则是在历史数据和预测数据的基础上，用来进行决策支持的系统。它们也可用于复杂查询或数据挖掘应用。信息系统的例子有：销售趋势分析、顾客划分、人力资源规划等。

信息系统和运作系统之间的主要差异见表11-1。这两种类型的处理几乎在每个比较类别中都具有截然不同的特征。特别需要注意的是：这两种处理的用户群截然不同。运作系统主要由那些负责处理日常事务的职员、管理员、销售人员及其他人员使用。而信息系统则是由那些搜索状态信息和制定决策的经理、高层管理人员、业务分析人员以及某些顾客使用。

表11-1　运作系统和信息系统的比较

特　征	运作系统	信息系统
主要目的	在当前基础上执行业务	支持管理者制定决策
数据类型	当前业务状态的表示	历史的时间点（快照）和预测
主要用户	职员、销售人员、管理员	经理、业务分析人员、顾客
使用范围	范围较小的、事先规划好的、简单的更新和查询	范围广泛的、特殊的、复杂的查询和分析
设计目标	性能吞吐量，实用性	灵活的访问和使用
容量	在一个或几个表格行上进行大量的、固定的更新和查询	定期的成批更新和查询，需要很多甚至全部的表格行

区分运作系统与信息系统主要基于以下三个因素：

1）数据仓库对散布在各不相同的运作系统中的数据加以集中，并将它们用于决策支持。

2）一个良好设计的数据仓库通过改善数据质量和数据一致性，增加数据价值。

3）一个单独的数据仓库可以消除许多由于混淆信息应用和运作处理而产生的对于资源的争用。

11.4 数据仓库的体系结构

经常用到的数据仓库基本体系结构有三种。首先，是一种一般的、适合初级数据仓库应用的两层物理体系结构；其次，是一种扩展的，适用于较复杂环境的三层体系结构；最后，是一种与三层物理体系结构相关联三层数据体系结构。

11.4.1 一般的两层体系结构

一般的数据仓库体系结构如图11-2所示。建立一个这样的体系结构需要以下四个基本步骤。

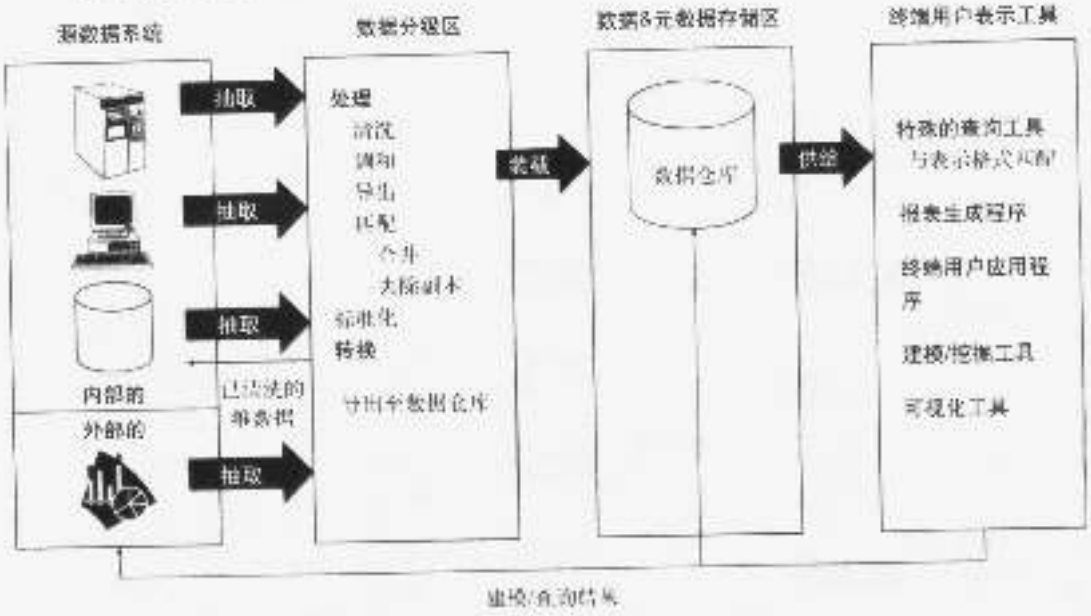


图11-2 一般的两层数据仓库体系结构

1）从各种内部和外部的源系统文件及数据库中抽取数据。在大型企业中，可能有成百上千个这样的文件和数据库。

2）来自不同源系统的数据在载入数据仓库之前必须经过转换和集成。某些事务可能会返回源系统以校正在数据分级时发现的错误。

3）数据仓库是为了支持决策制定而组织的数据库。它包含详细数据和汇总数据。

4）用户通过多种查询语言和分析工具访问数据仓库。访问结果（如预测或预报）可以反馈至数据仓库和运作数据库。

在后面的几节里我们还会详细讨论从源系统向数据仓库抽取、转换、装载（ETL）数据的重要步骤，以及各种终端用户表示工具。

抽取和装载要以一定的周期进行，比如一天、一周或一个月。因此，数据仓库常常不具有，也不需要具有当前的数据。请记住，数据仓库并不支持运作事务处理，尽管它可能包含有事务

性数据（不过，更多的是事务的汇总和状态变量的快照，如账目结算和库存水平等）。对大多数数据仓库应用而言，用户并不是希望发现对单个事务的反应，而是希望在数据仓库的一个大的子集上寻找组织状态的趋势和模式。如果数据仓库中保存了至少五个财政年度的数据，那么至少可以了解年度趋势和模式。旧的数据可以被清洗或存档。后面我们将看到一种先进的数据仓库体系结构——主动数据仓库，它基于对当前数据需求的不同假设而生成。

11.4.2　独立数据集市的数据仓库环境

与截止到目前本章所讨论的许多原则相反，某些组织并不总是创建一个数据仓库，而是创建若干独立的数据集市，每个数据集市都基于数据仓库和数据库技术，但并不进行事务处理。**数据集市**（data mart）是限定在一定范围内的数据仓库。它的内容可以来自于独立的ETL过程，如图11-3所示的**独立数据集市**（independent data mart）；也可以来自数据仓库，相关内容将在下面两节讨论。数据集市是为了特定的终端用户组的决策应用而专门定制的。因此，数据集市用于优化良好定义和可预测用途的性能，有时甚至只有一条或几条查询。例如，为了支持特定目的分析处理，某个组织可以建立一个营销数据集市、一个财务数据集市、一个供应链数据集市等等。

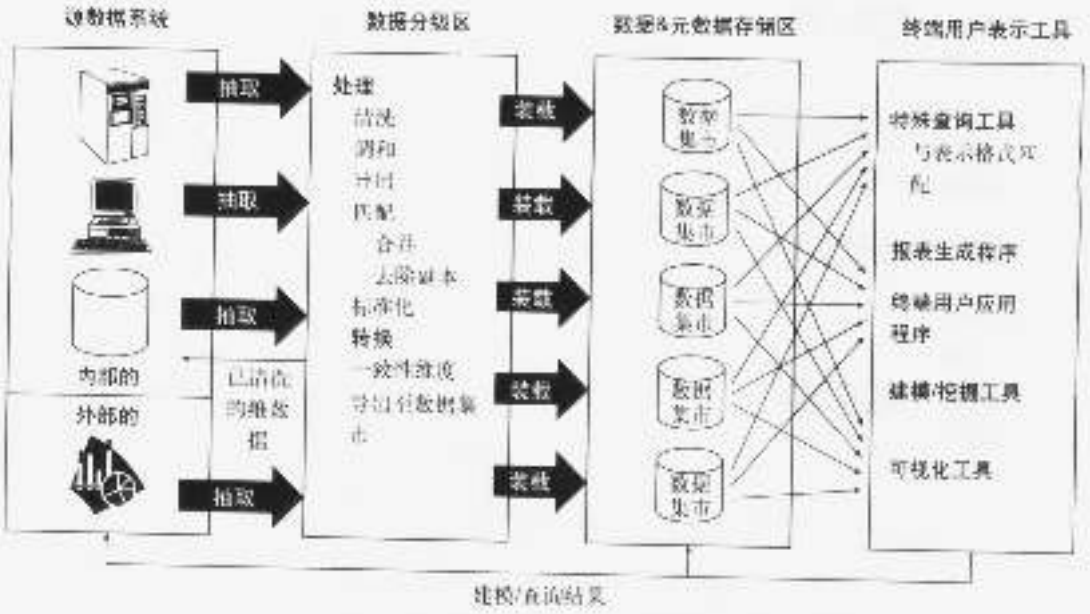


图11-3　独立数据集市的数据仓库体系结构

后面我们将对各种数据仓库的体系结构进行比较，不过，目前已经可以看到独立数据集市策略的一个明显特征：当终端用户需要访问单独数据集市的数据时比较复杂（从连接所有数据集市和终端用户表示工具之间互相交叉的连线就可以证明这一点）。这种复杂性不仅源于不得不访问单独的数据集市数据库，而且还源于多个数据集市可能产生新的数据系统不一致。如果在所有的数据集市之上只有一组元数据，并且如果在数据分级区的活动中所有的数据集市保持一致的话（例如，通过图11-3所示数据分级区中的"一致性维度"），那么对于终端用户而言的复杂性就降低了。在图11-3中，ETL过程的复杂性不太明显，因为针对每个独立数据集市都需要建立单独的转换和装载过程。

组织采用这种体系结构的原因只是为了方便，因为建立一些小的、独立的数据仓库总要比

让所有部门都与一个中心数据仓库的组织视图保持一致，在组织和策略上更简单一些。而且，某些数据仓库技术在所支持的数据仓库大小上有限制——我们在后面将其称为可伸缩性问题。因此，如果在了解数据仓库需求之前，就把自己限定在一个特定的数据仓库技术集中，那么数据仓库的体系结构可能就是由技术而不是由业务来决定的。在下一节里，通过和与其最有竞争力的体系结构进行比较，来讨论独立数据集市体系结构的优劣。

11.4.3 依赖数据集市和运作数据存储的体系结构

图11-3所示的独立数据集市的体系结构有如下几个较大的局限性（Meyer, 1997）：

1) 为每个数据集市都必须开发单独的ETL过程，这将导致大量的冗余数据和极大的开发工作量。

2) 数据集市之间可能并不一致，因此不能提供一个包括顾客、供应商和产品等重要主题的、清晰的企业级数据视图。

3) 不能从其他数据集市或共享的数据信息库中抽取更多的细节或相关事实，所以，给分析造成了一些难度（比如，不能为不同数据集市在不同平台上进行联结操作）。

关于独立数据集市的价值问题已经引起了极大的争议。Kimball（1997）强烈支持以独立数据集市应作为决策支持系统阶段性发展的可行策略。Armstrong（1997）和Inmon（1997与2000）认为这种看法是错误的。其实，围绕独立数据集市主要在两个方面存在争论：

1) 关于实现数据仓库环境的阶段性方法的性质。争论的焦点是：数据集市是否应该逐渐进化为决策支持数据的企业级定义的子集。

2) 关于何种数据库体系结构适用于分析处理。争论的焦点是：数据集市的数据库应规范化到什么程度。

上述两个争论的焦点将贯穿本章。而且，在本章末尾有一个练习专门深入地探究这两个争论。

解决独立数据集市的局限性的最普遍的方法就是使用**依赖数据集市**（dependent data mart）和**运作数据存储**（Operational Data Store，ODS）结构（如图11-4所示）。通过从**企业数据仓库**（Enterprise Data Warehouse，EDW）载入依赖数据集市可以解决上面的第一和第二条限制。所谓企业数据仓库，就是一个集成的中心数据仓库，它是决策支持应用的终端用户可以得到的控制点和惟一的“事实版本”。依赖数据集市也是为制定决策的用户群提供一个简化的、高性能的环境。用户可以访问自己的数据集市，在需要其他数据时，也可以访问EDW。依赖数据集市之间的冗余可以进行规划，这些冗余数据是一致的，因为每个数据集市都是从同一个公共数据源同步载入的。数据集成是管理企业数据仓库的IT人员的职责，不应该让终端用户为了每条查询或应用不得不跨越若干独立数据集市而进行数据集成。依赖数据集市和运作数据的存储体系结构通常称为“中央辐射式”方法，其中，EDW是中心，而源数据系统和数据集市分别位于输入和输出辐射端。

第三条限制可以通过让运作数据存储提供一个所有运作数据的集成源来加以解决。**运作数据存储**（Operational Data Store，ODS）是一种集成的、面向主题的、可更新的、具有当前值的详细数据库，它可以向作决策支持处理的终端用户提供服务（Imhoff，1998；Inmon，1998）。ODS通常是关系数据库，和记录系统中的数据库一样进行规范化，不过是为决策制定应用作了调整。例如，可以为了检索更广泛数据集的查询，而不是为了事务处理或者查询单个的以及直接相关记录（如顾客订单号），ODS可以对索引及其他一些关系数据库的设计元素进行调整。ODS包含所有活动的、企业级的当前视图。因为它拥有不断变化的当前数据，所以在不同时间对ODS进行相同查询也可能会得到不同结果。ODS通常不包含历史数据，而EDW却掌

操着某一时刻组织状态的快照的历史数据。ODS的数据可能来自于ERP应用数据库，但是因为大多数组织不仅仅只有一个ERP数据库，而且也不可能在一个ERP上执行所有操作，所以通常一个ODS并不等同于一个ERP数据库。另外，ODS并不是用来处理大量事务性工作，而是用于决策支持。ODS同时起数据分级区的作用，将数据载入EDW。ODS可以立即从记录系统接收数据，也可以有所延迟，这种特性对于它所支持的决策制定需求来说是实用的，而且也是可以接受的。

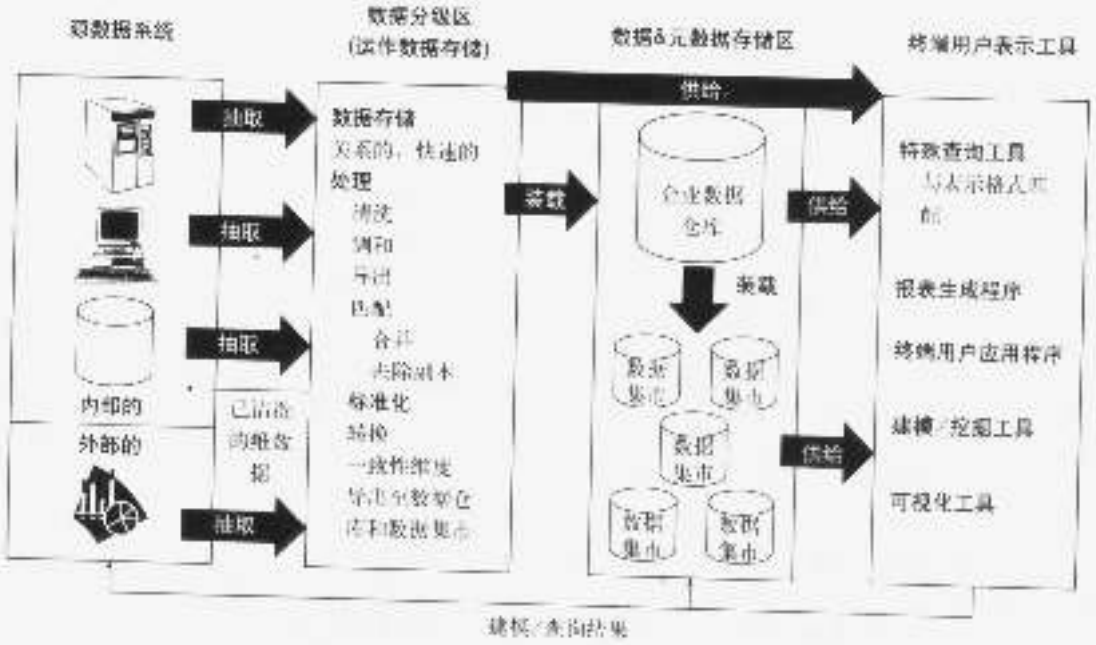


图11.4　依赖数据集市和运作数据的存储体系结构

依赖数据集市和运作数据存储的体系结构也被称为企业信息工厂（Corporate Information Factory，CIF）（见Imhoff，1999），我们可以把它看作是一个支持用户所有数据需求的组织数据的综合视图。

即便是本领域的权威人士，他们所赞成的数据仓库方法也非常不同。支持独立数据集市的权威人士认为，这种方法有两个重大优点：

1）该方法可以使数据仓库的概念通过一系列小项目得以验证。

2）因为组织不必等到所有的数据都集中起来后再工作，所以从数据仓库中得到收益的时间也得以缩短。

而CIF的拥护者则提出了独立数据集市方法所面临的严峻问题。这些问题如下所示（Inmon，1999；Armstrong，2000）：

1）数据集市之间的冗余。

2）用户自身需要多个数据集市之上的集成的组织视图。

3）过多ETL过程所导致的额外工作。

4）数据集市由于技术限制或每个数据集市采用不同的技术，都未能达到EDW的规模，它们合理使用数据仓库技术的可能性不大。

5）维护数据集市同步的代价高昂。

Armstrong（2000）等人还认为，独立数据集市的拥护者声称的所谓好处实际是采取阶段性方法开发数据仓库的好处。而阶段性方法也可以在CIF架构中实现，并最终实现为数据仓库体系结构。这是我们在下节中将要讲述到的。

11.4.4 逻辑数据集市和主动仓库体系结构

逻辑数据集市和主动仓库体系结构仅适用于中等规模的数据仓库或采用高性能数据仓库技术（如NCR Teradata系统）的情况。如图11-5所示，这种体系结构有如下特征：

1）**逻辑数据集市**（logical data mart）不是物理上分开的数据库，而是某个稍不规范的、物理的关系数据仓库的不同关系视图（参见第7章中视图的概念）。

2）数据不经过独立的分级区就可直接载入数据仓库，从而利用数据仓库技术的高性能计算能力执行数据的清洗和转换步骤。

3）由于不需要建立物理数据库，也不需要精通相应的数据库技术，又无需编写装载例程，因此可以非常快速地创建新的数据集市。

4）因为只有在需要时才创建视图，所以数据集市的数据都是最新的。如果用户对同一个数据集市的实例要进行一连串的查询和分析，那么也可以将视图物化。

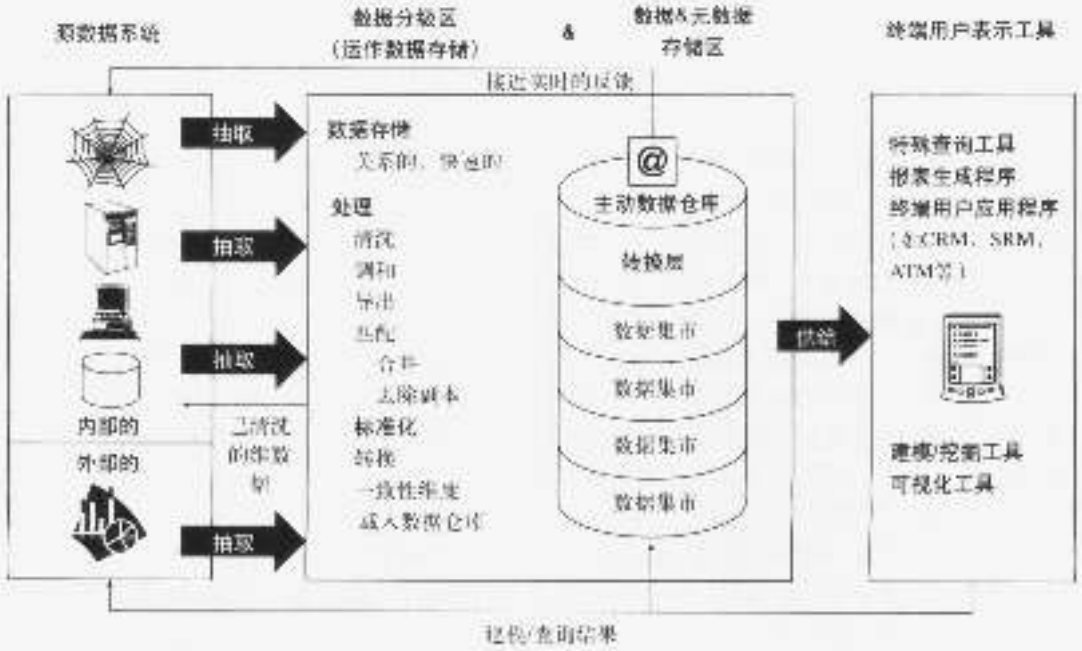


图11-5 逻辑数据集市和主动仓库体系结构

无论是逻辑的还是物理的，数据集市和数据仓库都在数据仓库环境中扮演着不同的角色，表11-2对这些不同之处作了总结。尽管在规模上有限制，数据集市也不一定很小。因此，可伸缩技术非常关键。尽管数据集市的规模受到限制，建立一个数据集市的复杂性较低，但其仍然面临着ETL问题的挑战。而且，当用户自己需要在不同的物理数据集市之上集成数据的时候（这是极有可能发生的），就会面临巨大的负担和成本。通过不断增加数据集市的方法，可以分阶段建立数据仓库。要想做到这一点，最简单的方法莫过于采用逻辑数据集市和主动仓库体系结构了。

表11-2 数据仓库与数据集市的差别

数据仓库	数据集市
范围	**范围**
• 独立于应用	• 特定的DSS应用
• 集中的、可能是企业范围的	• 不集中，随用户而定
• 事先规划的	• 有机的，可能并没有规划
数据	**数据**
• 历史的、详细的、汇总的	• 部分历史的、详细的、汇总的
• 稍微不规范的	• 高度不规范的
主题	**主题**
• 多主题	• 用户关心的某个主题
源	**源**
• 许多内部源与外部源	• 几个内部源与外部源
其他特征	**其他特征**
• 灵活的	• 限制的
• 面向数据的	• 面向项目的
• 生命期长	• 生命期短
• 大型	• 开始小，逐渐变大
• 单一复杂结构	• 多种的、半复杂结构

注：改编自Strange（1997）。

从图11-5所示体系结构中的**主动数据仓库**（active data warehouse）部分可以看到：因为需要对组织的当前复杂视图做出快速反应，所以源数据系统和数据仓库之间的数据交换是以接近实时的速度进行的。例如，回答问题并记录问题的客服人员必须持有顾客最近与销售人员接触的情况、账单和付款业务、维修活动以及订单情况的总体视图。有了这些信息，客服部门的系统就可以用类似的特征文件，自动地为专业人员产生维修合同、升级产品、或客户购买的其他产品的销售脚本。关键事件（如输入新的产品订单）可以被立即考虑，这样，企业至少能做到像顾客了解自己一样了解自己与顾客的关系。最终方向是使每个事件（比方说是客户）都成为一种用户化和个人化交流的潜在机会，这种交流是基于如何利用特定的特征文件响应客户的战略性决策。受益于主动数据仓库的应用包括：

- 基于最新的库存量实现变更交付的及时运送。
- 电子商务应用。比如，在用户注销之前，被放弃的购物车可以触发电子邮件提示消息。
- 信用卡事务中的欺诈检测，其中某个异常的事务模式可能警告售货员或联机购物车例程采取额外的防范措施。

上述应用通常都需要处于联机状态、365天每天24小时向用户提供访问。对于任何一种数据仓库体系结构来说，用户可能是员工、顾客或者合伙人。

若利用高性能计算机和数据仓库技术，可能并不需要将ODS与企业数据仓库分离。当ODS和EDW合二为一时，用户就很容易在解决一系列彼此相关的特殊问题时下钻和上挖（Drill-Up）。同时，由于消除了依赖数据集市和运作数据存储体系结构层，所以这也是一种比较简单的体系结构。

11.4.5 三层数据体系结构

图11-6显示的是数据仓库的三层数据体系结构：

1）运作数据存储存在于整个企业的不同运作记录系统中（有时也存储在外部系统中）。

2）调和数据是保存在企业数据仓库和运作数据存储中的数据类型。

3) 导出数据是存储在每个数据集市中的数据类型。

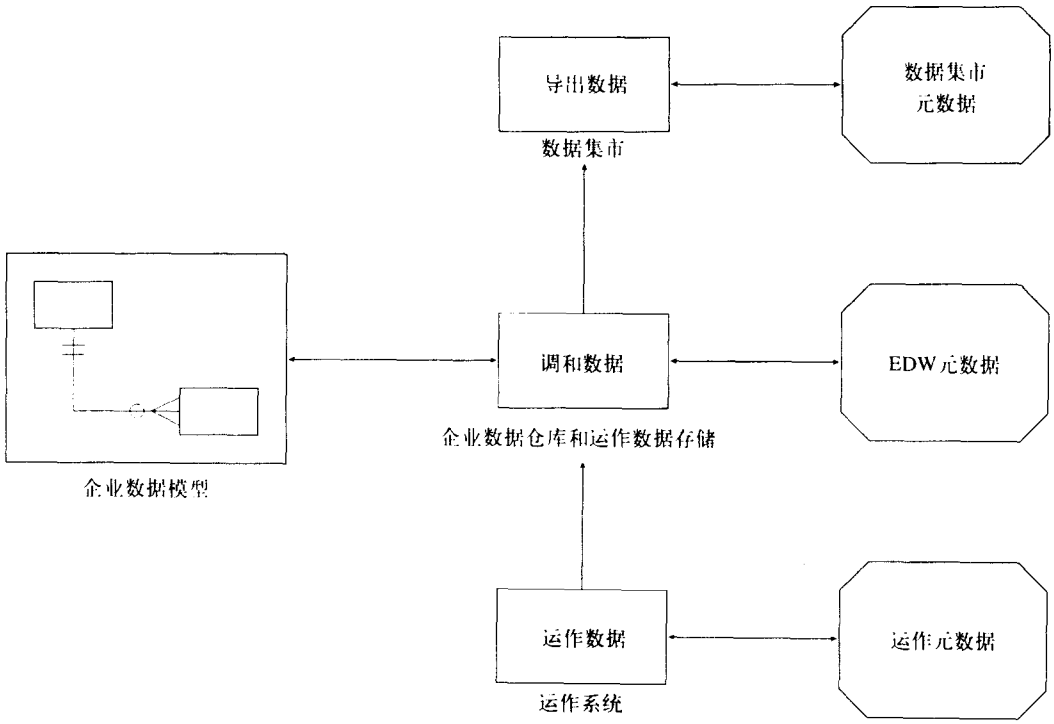


图11-6 三层数据体系结构

调和数据（reconciled data）是详细的当前数据，并且是所有决策支持应用的惟一的、权威的数据来源。**导出数据**（derived data）是为了终端用户的决策支持应用系统而选择、格式化和聚集的数据。

以下两节将讨论调和数据和导出数据。还有两种在图11-6所示体系结构中非常关键的组件，它们是企业数据模型和元数据。

1. 企业数据模型的作用

图11-6还显示出与企业数据模型相链接的调和数据层。回忆一下第2章提到过，企业数据模型代表了一个解释企业所需数据的完整视图。如果调和数据层是决策支持所需要的所有数据的权威来源，那么它就必须与企业数据模型所指定的设计相一致。因此，企业数据模型控制数据仓库的阶段性发展。通常，企业数据模型在解决新问题和决策应用系统的过程中不断演化。一次性开发企业数据模型要花费很长时间，同时，在数据仓库建好之前，决策需求可能会动态地发生变化。

2. 元数据的作用

图11-6还显示出与三个数据层相链接的元数据层。回忆一下第1章提到过，所谓元数据就是描述数据的性质和特征的数据。以下是图11-6所示的三种类型元数据的简单描述。

1) 运作元数据（operational metadata）描述流入EDW的不同运作系统（也包括外部数据）中的数据。通常情况下，运作元数据有各种不同的存在格式，但质量较差。

2) EDW元数据（EDW metadata）由企业数据模型衍生而来（或至少与其相一致）。EDW元数据既描述调和数据层，也描述将运作数据转换为调和数据的规则。

3) 数据集市元数据（data mart metadata）描述导出数据层，以及将调和数据转换为导出数

据的规则。

11.5 数据仓库中数据的若干特征

为了更好地理解各层中的数据，读者需要学习数据储存在数据仓库中时的一些基本特征。

11.5.1 状态数据与事件数据

状态数据与事件数据之间的区别如图11-7所示。图中显示一个典型的由数据库管理系统记录的日志条目，它记录了某个银行应用系统所处理的一次业务。该日志条目既包括状态数据也包括事件数据："前象"和"后象"分别代表银行账户在提款之前和提款之后的状态。图的中部显示的是代表提款（或更新事件）的数据。

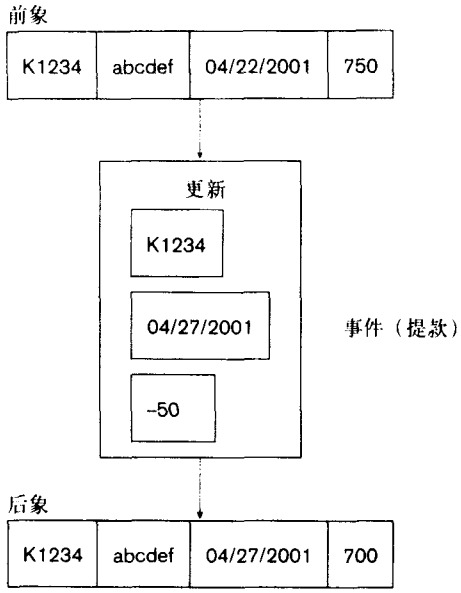


图11-7 DBMS日志条目举例

事务即业务活动，它导致在数据库级发生一个或多个业务事件（Devlin，1997），在第12章里我们还会进一步讨论事务。**事件**（event）则是由事务产生的数据库动作（创建、更新或删除）。注意，一个事务可能引起一个或多个事件。图11-7中的提款事务产生了一个事件，即将账户余额从750减至700。另一方面，把钱从一个账户转到另一个账户将会引发两个事件：一个是提款事件，另一个是存款事件。有时，一些非事务性活动也都是重要事件，比如，一个被放弃的联机购物车、忙信号或断开的网络连接、把某项目放入购物车但又在结账前取出来。

状态数据和事件数据都可以存储到数据库当中。但在实际应用中，数据库（包括数据仓库）中存储的数据大多数都是状态数据。一个数据仓库可能包含状态数据快照的历史记录，也可能包含事务的汇总（比如，每小时统计一次总数）或事件数据。代表事务的事件数据可以定义其存储时间，但是在用后需将这些数据删除或存档以节省存储空间。状态数据和事件数据一般都存储在数据库日志中（如图11-7所示），以便在备份和恢复数据时使用。下面将会解释数据库日志在填充数据仓库时所起的重要作用。

11.5.2 临时数据与周期数据

在数据仓库中，常常需要维护一份过去所发生事件的记录。例如，将现阶段的销售水平或库存水平与去年同一日期或同一时期的历史数据进行比较。

大多数运作系统都基于**临时数据**（transient data）的使用。在临时数据中，对现存数据的修改直接在前面数据的基础上进行，因此前面的数据内容会遭到破坏。在不保存记录的先前内容的情况下直接删除记录的情况也时有发生。

参照图11-7，可以很形象地理解临时数据：假如后象在前象之上修改，那么前象（包含以前的余额）就没有了。不过，由于图11-7显示的是数据库日志，所以前象、后象通常都会保存。

周期数据（periodic data）一旦存入就不会被物理改变或删除。图11-7中的前象和后象就代表周期数据。注意，每条记录包含一个时间戳（第3章已经介绍了时间戳的使用），指明最近更新事件所发生的日期（如果需要的话，还可显示时间）。

11.5.3 一个临时数据和周期数据的例子

图11-8和11-9显示一个更为详细的比较临时数据和周期数据的例子。

Table X (10/01)

Key	A	B
001	a	b
002	c	d
003	e	f
004	g	h

Table X (10/02)

Key	A	B
001	a	b
002	r	d
003	e	f
004	y	h
005	m	n

Table X (10/03)

Key	A	B
001	a	b
002	r	d
003	e	t
005	m	n

图11-8 临时运作数据

Table X (10/01)

Key	Date	A	B	Action
001	10/01	a	b	C
002	10/01	c	d	C
003	10/01	e	f	C
004	10/01	g	h	C

Table X (10/02)

Key	Date	A	B	Action
001	10/01	a	b	C
002	10/01	c	d	C
002	10/02	r	d	U
003	10/01	e	f	C
004	10/01	g	h	C
004	10/02	y	h	U
005	10/02	m	n	C

Table X (10/03)

Key	Date	A	B	Action
001	10/01	a	b	C
002	10/01	c	d	C
002	10/02	r	d	U
003	10/01	e	f	C
003	10/03	e	t	U
004	10/01	g	h	C
004	10/02	y	h	U
004	10/03	y	h	D
005	10/02	m	n	C

图11-9 周期仓库数据

1. 临时数据

图11-8显示了一个包含四条初始记录的关系（Table X）。该表有三个属性：一个主键和两个非主键属性A和B。每个属性在日期10/01的值在表中都有记录。例如，该日期001记录中属性A的值是a。

在日期10/02中，这个表发生三个变化（用箭头在表的左边指出发生变化的行）。002记录被更新，属性A的值从c变成r。004记录也被更新，属性A的值从g变成y。最后，向表中插入一条新记录（主键为005）。

注意，002记录和004记录得到更新，而且新记录取代了先前的记录。因此，先前的值丢失了，因为没有这些值的历史记录。这就是临时数据的特征。

在日期10/03中，记录的变化更多（为了简化讨论，我们假定在某一天对指定记录只能进行一次修改）。003记录得到更新，004记录被删除。注意，没有任何记录表明004记录曾经在数据库中保存过。图11-7所示的数据处理方式是运作系统中临时数据的典型特征。

2. 周期数据

数据仓库的目标之一就是维护关键事件的历史数据或针对诸如销售额的特殊变量创建时间序列。为实现这一目标，就要求存储周期数据而不是临时数据。图11-9和图11-8使用的是同一张数据表，只不过是作了一些修改使其表示周期数据。图11-9作了如下修改：

1) 表X中新增了两列。

a) 名为Date的列代表时间戳，它存储表记录更改的最近日期。

b) 名为Action的列记录数据更改的类型，该属性可以取的值包括C（Create）、U（Update）和D（Delete）。

2) 一旦一条记录存入表X，这条记录就不会改变。在改变一条记录时，前像和后像都会存入数据表。尽管一条记录可以在逻辑上被删除，但是被删除记录的历史版本仍然保存在数据库中，保存的时间取决于趋势分析的需要（至少五个季度）。

现在让我们来分析图11-9所发生的一组动作。假定所有的四条记录都在10/01创建，如第一个表所示。

在第二个表中（10/02），002记录和004记录得到更新。此时，表X中既包含旧的记录版本（10/01）也包含新的记录版本（10/02）。同时还增加了一条在10/02产生的记录005。

第三张表（10/03）对003记录进行了更新，也保留了旧记录和新记录。004记录在这张表中被删除了，那么此时表X包含三个版本的004记录：初始版本（10/01）、更新版本（10/02）与删除版本（10/03）。004记录的最后一行中的D指明这条记录已经在逻辑上删除了，因此对于用户或他们的应用程序而言，该记录不可访问。

经过对图11-9的研究分析，就可以明白为什么数据仓库的增长如此迅速。存储周期数据需要大量存储空间。因此，得出一个必然结论：用户必须非常仔细地选择需要以这种方式进行处理的关键数据。

11.6 调和数据层

如图11-6所示，**调和数据**（reconciled data）意指与运作数据存储和企业数据仓库相关联的数据层。IBM在1993年的一篇论文中使用了这个术语，用来描述数据仓库体系结构。尽管该术语并没有被广泛使用，但是它准确地描述了出现在企业数据仓库中数据的本质以及获得它们的方式。通常，调和数据是作为ETL过程的结果被提及。EDW或ODS通常是规范化的关系数据库，因为它们都需要一定的灵活性来支持各种各样的决策需求。

11.6.1 进行ETL之后的数据特征

抽取、转换和装载都是为了给用于决策支持的数据提供一个惟一的权威来源。理想情况下，这个数据层是详细的、历史的、规范化的、全面的、及时的，并且其质量也能够得到良好的控制。

1) **详细的** 数据是详细的（而不是概要的），可以为不同的用户群体提供最大的灵活性，以构建符合用户需求的数据。

2) **历史的** 数据是周期的（或者说是某个时间点上的），因此可以提供历史视图。

3) **规范化的** 数据是完全规范化的（即，符合第三范式或更高级别的范式，我们在第5章里已经讨论过规范化问题）。规范化数据比非规范化数据提供更大的完整性和灵活性。由于通常情况下使用批处理方法周期性地访问调和数据，所以非规范化不是提高性能所必须的。不过，后面我们将会看到，一些流行的数据仓库的数据结构就是非规范化的。

4) **全面的** 调和数据反映企业级视图，它的设计与企业数据模型一致。

5) **及时的** 除了主动仓库以外，数据不必是实时的。然而，数据必须足够新，以便作出及时的决策反应。

6) **良好的质量控制** 调和数据不能有任何质量和完整性问题，以便在数据集市中汇总并用于决策制定。

注意，调和数据的这些特征与衍生出它们的典型的运作数据的特征大相径庭。运作数据也是详细的，但是在其他四个方面存在很大差异：

1) 运作数据是临时的，而不是历史的。

2) 运作数据不是规范化的。就其核心而言，运作数据可能从来不会被规范化，甚至出于性能考虑而进行非规范化。

3) 运作数据通常为了某个特定的应用而限制于一定范围内，因此它们不会是全面的。

4) 运作数据的质量一般不太好，可能存在各种不一致错误和其他错误。

数据调和过程负责把运作数据转换成调和数据。由于这两种数据之间存在巨大差异，因此，明确的数据调和在构建数据仓库时是最困难也是技术上最富有挑战性的一个部分。值得庆幸的是，已经有几种完善的软件产品可以辅助完成数据调和过程。

11.6.2 ETL过程

在填充EDW的过程中，数据调和在以下两个阶段发生：

1) 在创建EDW之初的初始装载阶段。

2) 为了保证EDW内容最新或对其进行扩展，而随后进行的更新（一般是定期进行）。

数据调和过程如图11-10所示，其中包括四个步骤：**获取、清洗、转换、装载和索引**。实际上，这些步骤可以用不同方式进行合并。例如，数据获取和清洗可以合并成一个过程，或者将清洗和转换合并为一个过程。下面首先讨论获取、清洗、装载和索引，然后彻底讨论转换。

1. 抽取

从源文件和数据库中获取用来填充EDW的相关数据的过程叫做抽取。一般来说，并不需要所有运作系统中的数据，所需要的只是其中的一个子集。在对源系统和目标系统详尽分析的基础上方能抽取数据子集，这个工作最好由数据管理部门领导的小组来完成，该小组由终端用户和数据仓库专业人员组成。

数据抽取分为两种类型：静态抽取和增量抽取。静态抽取用于初始填充数据仓库，而增量抽取则用于数据仓库的日常维护。

静态抽取（static extraction）是一种获取所需要的源数据在某个时间点的快照的方法。源

数据视图被创建时是独立于时间的。

增量抽取（incremental extraction）只获取自上次抽取后源数据中发生变化的部分。最常见的方法就是日志获取。因为数据库日志中记载的是数据库记录最近变化的后象（参见图11-7），所以日志获取就是从日志中选择自上次抽取后所记载的后象。

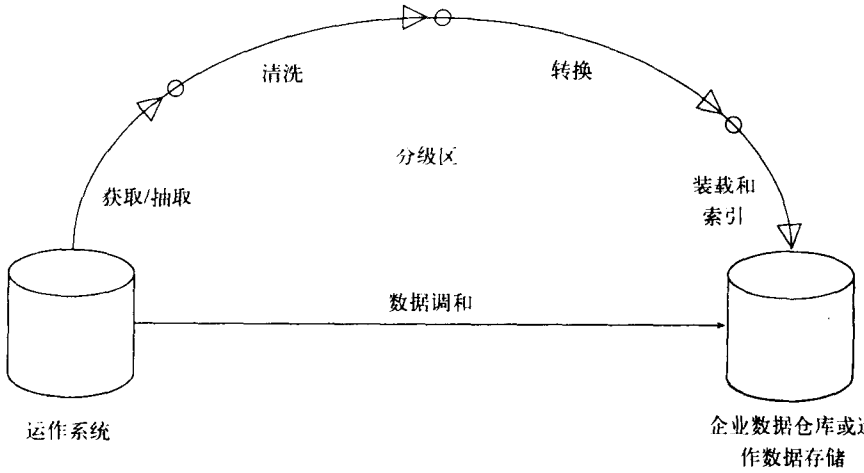


图11-10　数据调和的步骤

English（1999）和White（2000）详细地提出了使记录系统和其他源数据具有抽取到分级区域资格的必要步骤。主要标准是源系统数据的质量。主要依据如下几点判断质量:

- 数据命名的清晰度，便于仓库设计者确切知道源系统中都存在哪些数据。
- 源系统执行的业务规则的完备性和准确性，这一点直接影响数据的准确度。同时，源系统的业务规则应与数据仓库中使用的规则相匹配。
- 数据格式（跨数据源的公共格式有助于匹配相关数据）。

与源系统的所有者达成协议也很重要，这样，他们就会在源系统的元数据发生变化时通知数据仓库管理员。由于事务系统会频繁更改以满足新的业务以及利用新的、更好的软硬件技术，因此，如何管理源系统中的更改是抽取过程所面临的最大挑战之一。一旦源系统发生变化，数据质量以及抽取和转换数据过程就需要进行重新评估。这些过程将源系统的数据映射为目标数据仓库（或数据集市）的数据。映射表明数据仓库的各个数据是由哪个源系统的哪个数据导出来的；而转换规则（在其他的小节中讨论）则描述如何进行导出。对于定制构建的客户源系统，数据仓库管理员必须开发定制的映射和抽取例程。而一些打包的应用软件（比如ERP系统）可以购买预先定义的映射模板。

可以采用与源系统相关的工具（如数据导出工具）编写抽取例程。通常，用一种中性的数据格式来抽取数据，如采用逗号分隔的ANSI格式。有时，也使用SQL命令SELECT-INTO来创建载入分级区的表。

一旦选定数据源并编写了抽取例程，就可以把数据转移到分级区，在那里开始清洗过程。

2. 清洗

运作系统中数据的质量通常很差，其中一些典型的数据错误和数据不一致包括:

1) 名字和地址拼写错误。

2) 不可能的或错误的出生日期。

3) 某些字段的使用与最初的使用目的不符合。

4) 地址和邮政编码错误匹配。

5) 缺失数据。

6) 重复的数据。

7) 源数据之间的不一致（如，不同的地址）。

这里可以再举出一些错误的例子。比如，在顾客文件中，顾客的名字经常作为主键或检索的关键字。然而，这些名字却经常被拼错或在不同的文件中以不同方式拼写。例如，“The Coca-Cola Company”是可口可乐公司名字的正确拼法，但在顾客记录中可能会写成“Coca-Cola”、“Coca Cola”、“TCCC”等各种不同的方式。研究表明，仅“McDonald's”一词就可能有100种不同的拼写方式!

另外一种数据污染就是某个字段的使用与最初的目的不符。例如，在某银行数据库中，设计了一个字段用来记录电话号码。然而，某个部门经理并不需要这个字段的内容，所以取而代之存入的是利率。另外一个例子更加奇怪，是由一家著名的英国银行报道的：数据清洗程序在文件里发现了一个顾客，他的职业栏中填写着“泰坦尼克上的乘务员”！（Devlin，1997）

你可能想知道为什么运作数据中的错误如此普遍。运作数据的质量在很大程度上由负责收集它们的组织对数据的估计所决定。但是，数据收集组织经常对一些重要数据估计过低，而这些数据的准确性对于下游的应用系统（如数据仓库）则是非常关键的。

在错误经常发生的前提下，最坏的情形就是公司仅仅把运作数据拷贝到数据仓库中，而较好的情形是利用一种叫做数据清洗的技术来改善源数据的质量。**数据清洗**（data scrubbing，也称为data cleansing）就是在转换数据并将其载入数据仓库之前，使用模式识别和其他技术提升原始数据质量的一种技术。

成功的数据仓库要求实现一个符合全面质量管理（Total Quality Management，TQM）的正式程序。TQM致力于防止错误，而不是纠正错误。尽管数据清洗有助于改善数据质量，但它并不是一个解决数据质量问题的长期方案。English（1999）对在数据管理中应用TQM有精妙的论述。

所需的数据清洗类型取决于源系统数据的质量。除了能解决前面所提到的问题外，清洗还可以完成如下一些任务：

- 解码数据，使数据仓库应用可以理解这些数据。
- 重新格式化和改变数据类型并执行其他函数，把来自每个源系统的数据变成数据仓库的标准格式，为转换做好准备。
- 加上时间戳以区分相同属性在不同时间的值。
- 在不同度量单位之间进行转换。
- 为表的每一行产生主键（在本章后面，我们将讨论数据仓库中表的主键和外键的形成）。
- 把不同的抽取内容匹配和融合，构成一张表或一个文件，并把数据一一匹配到这张表的相应行中（当不同的源系统采用不同的键，或者不同的源系统命名惯例不同以及源系统中的数据包含错误时，这是一个非常困难的过程）。
- 在日志中记录所检测到的错误，更正那些错误，并再次处理已修正的数据而不需要创建冗余项。
- 找寻缺失的数据，保证接下来要进行的装载过程数据完整。

对不同数据源的处理顺序也是要考虑的一个方面。例如，先处理来自销售系统的顾客数据，然后新的来自于外部系统的顾客统计数据就能够与之匹配。

一旦分级区的数据清洗完毕，则进行转换的数据也就准备好了。在我们详细讨论转换过程

之前，首先简要地回顾一下把数据载入数据仓库或数据集市的过程。

3. 装载和索引

填充EDW的最后一步就是把选择好的数据装入目标数据仓库并创建必要的索引。装载数据到目标EDW有两种基本模式：刷新模式和更新模式。

刷新模式（refresh mode）是一种应用定期重写目标数据的填充数据仓库的方法。也就是说，起初目标数据被写入仓库，随后定期重写仓库数据，替换先前的内容。这种模式如今已经不大流行了。

更新模式（update mode）是一种只向数据仓库中写入源系统数据变化的装载方法。为了支持仓库数据的周期性本质，新记录往往被写入数据仓库，而不需要覆盖或删除先前的记录（参见图11-9）。

像我们所期望的那样，刷新模式通常用于在数据仓库创建时填充仓库，更新模式则是用于目标仓库的日常维护。刷新模式与静态数据捕获联合使用，而更新模式与增量数据捕获联合使用。

无论是刷新模式还是更新模式，都必须创建和维护用来管理仓库数据的索引。数据仓库环境经常采用一种叫做*位映射索引*（bit-mapped indexing）的索引类型（参见第6章）。本章后面讲述针对数据仓库和数据集市的数据库设计时，还会讨论索引问题。

基于广为认知并且非常成功的Wal-Mart公司的数据仓库，Westernman（2001）讨论了在决定以怎样的频率更新数据仓库时应考虑的因素。他的原则是根据实际应用更新数据仓库。更新不够频繁会导致装载数量过大，而且还会导致用户等待新数据。另一方面，几乎实时的装载虽然对于主动数据仓库非常必要，但对于大多数数据挖掘和分析应用系统而言并不必要，并且可能会导致效率低下。对于大多数组织而言，Westerman建议每日更新就足够了。不过，每日更新不能对某些条件的变化作出迅速反应，例如改变某些滞销产品的定价或购买订单。Wal-Mart实行的是持续更新数据仓库，由于他们采用海量并行的数据仓库技术，所以这种方法是实际可行的。

把数据载入仓库通常意味着向仓库中的表追加新的行，也可能意味着用新数据更新现有行（例如，从额外的数据源获取数据填充仓库表里缺失的值），还可能意味着从仓库里清除指定数据，这些数据可能因为时间过长而废弃，或者在上一次装载操作时没有正确装载。数据可以利用以下方法从分级区载入仓库：

- SQL命令（例如INSERT或UPDATE）。
- 由数据仓库或第三方提供的特殊装载程序。
- 由仓库管理员编写的定制例程。

上述任何一种方法都要求装载例程不仅能够更新数据仓库，而且还要产生错误报告来显示被拒绝的数据（例如，尝试追加键值重复的行，或者更新仓库表中根本不存在的行）。

装载程序能够以批处理模式或连续模式进行工作。利用装载程序，用户可以通过编写脚本来定义分级区中数据的格式，以及分级区中的数据分别与数据仓库中的哪个字段对应。装载程序可以将分级区某个字段中的数据类型转换为仓库中目标字段的数据类型，还可以执行IF-THEN-ELSE逻辑处理不同格式的分级区数据，或者指导将数据输入数据仓库的不同表里。程序还可以在数据装载（刷新模式）或追加新行（更新模式）之前清理仓库表中的所有数据（DROP TABLE）。装载程序还能够将输入数据分类，以便于表的行在被更新之前先被追加。此外，装载程序可以根据需要运行任何DBMS存储进程；理想情况下，还可以运行所有DBMS的并发控制；一旦在装载过程中遇到DBMS故障，重启和恢复将会奏效。因为执行装载过程会非

常耗时，所以如果DBMS在装载过程中发生崩溃，则装载能够从检查点开始继续装载是极为重要的。本书第12章将完整地讨论数据库的重启和恢复。

11.7 数据转换

数据转换（data transformation）是数据调和过程的中心，它是数据调和的组成部分，目的是把源运作系统的数据格式转换为EDW的数据格式。数据转换接受来源于数据捕获组件的数据（在数据清洗之后），然后把数据映射为调和数据层的格式，再将数据传递至装载和索引组件。

数据转换的范围覆盖从数据格式（或表示）的简单变化到实现高度复杂的数据集成。下面列举了说明这个范围的三个例子：

1）一位推销员要求从主机数据库中把顾客数据下载到自己的笔记本电脑中。在这种情况下，所需要的转换仅仅是将EBCDIC型数据转换为ASCII表示，借助商业现用软件可以很容易地完成这种转换。

2）一家制造公司的产品数据存储在三个不同的遗留系统中，这三个系统分别为制造系统、营销系统和工程应用系统。公司需要为这些产品数据开发一个联合视图。数据转换涉及几个不同的功能，其中包括解决不同的键结构问题，转换为一组公共编码，集成来自不同源的数据。这些功能非常直观，可以直接用标准的、带有图形界面的商业软件包来实现。

3）一家大型的健康医疗组织管理着一组地理上分散的医院、诊所及其他诊疗中心。因为这些单位是通过收购而归入这家医疗组织的，所以其中的数据是异构和不一致的。出于许多重要原因，该组织需要开发一个数据仓库以提供一个企业级整体视图。这就要用到如下描述的所有转换函数，其中包括一些定制的软件开发在内。

数据清洗所执行的功能和数据转换所执行的功能有可能发生混淆。一般而言，数据清洗的目标是校正源数据中数据值的错误，而数据转换的目标是把源数据的格式转换为目标系统的格式。注意，在数据转换前必须进行数据清洗，因为如果在转换前数据就有错误的话，那么在数据转换后错误仍然存在。

11.7.1 数据转换函数

数据转换包含多种不同的函数。这些函数从广义上可以划分为两类：记录级函数和字段级函数。在大多数数据仓库应用系统中，可能需要联合应用其中一些甚至是所有的函数。

1. 记录级函数

操作一组记录（如文件和表）的最为重要的记录级函数包括：选择、联结、规范化和聚合。

选择（selection，也叫做subsetting）就是根据预先定义的标准划分数据的过程。对数据仓库应用而言，选择就是从将要填充数据仓库的源系统中抽取相关数据的过程中。事实上，选择是前面所讨论的捕获函数的一部分。

当源数据是关系数据时，SQL的SELECT语句可以用于选择（详情参见第7章）。例如，从上一次捕获时已经创建的数据库日志中选择后象就可以实现增量捕获。典型的后象如图11-7所示。假定该应用的后象储存在一个名为ACCOUNT_HISTORY的表中，那么在12/31/2001之后所创建的后象就可以用如下语句来选择：

```
SELECT *
FROM ACCOUNT_HISTORY
WHERE Create_Date>12/31/2001;
```

联结（joining）把不同来源的数据合并为一张表或视图。联结数据在数据仓库应用中是一

个重要的功能，因为合并来自不同源的数据是非常必要的。例如，一家保险公司的客户数据可能分布在许多不同的文件或数据库中。当源数据是关系数据时，SQL语句可以用来执行联结操作（详情参见第7章）。

由于存在以下因素，联结有时非常复杂:

1) 源数据通常不是关系型数据，在这种情况下，就不能使用SQL语句了，取而代之的是必须编写过程语言语句。

2) 即使是关系型数据，欲联结的表的主键通常来自不同的域（例如，工程零件编号与目录编号）。这些键必须在执行SQL联结之前进行调和。

3) 源数据可能包含错误，这使得联结操作也可能产生错误。

规范化（normalization）过程可以分解不规则的关系，以产生较小的、结构良好的关系（详情参见第5章）。正如前面所指出的，运作系统中的源数据通常不是规范化的（或者一点也不规范）。因此，这些数据必须在数据转换中被规范化。

聚合（aggregation）是将数据从细节级转换到汇总级的过程。例如，在零售业务中，可以根据商场、产品、日期等汇总单个销售事务以产生总的销售统计。因为在我们的模型中，EDW只包含详细数据，聚合一般与这个部分无关。然而，在填充数据集市时，聚合非常重要，这一点会在下面进行解释。

因为可以用SQL语句实现选择、联结和聚合，所以一些数据仓库专家主张转换应该在运作数据存储或数据仓库里进行。

2. *字段级函数*

字段级函数将源记录中给定格式的数据转换为目标记录中不同格式的数据。字段级函数分为两种: 单字段和多字段。

单字段（single-field）转换将一个源字段数据转换为一个目标字段数据。图11-11a是这种类型转换的基本表示（由图表中的字母T指明）。例如，把某个度量从英制表示转换为公制表示就是单字段转换。

如图11-11b和11-11c所示，有两种基本方法可以实现单字段转换，即算法和表查找。算法转换是借助于公式或逻辑表达式来实现的。图11-11b显示了利用公式将华氏温度转换为摄氏温度的过程。当简单的算法不能奏效时，可以使用查找表的方法。图11-11c显示了使用查找表将状态代码转换为状态名的过程（这种类型的转换在数据仓库应用系统中非常普遍）。

多字段（multifield）转换把一个或多个源字段中的数据转换为一个或多个目标字段的数据。这种类型的转换在数据仓库应用中也很普遍。图11-12展示了两个多字段转换的例子。

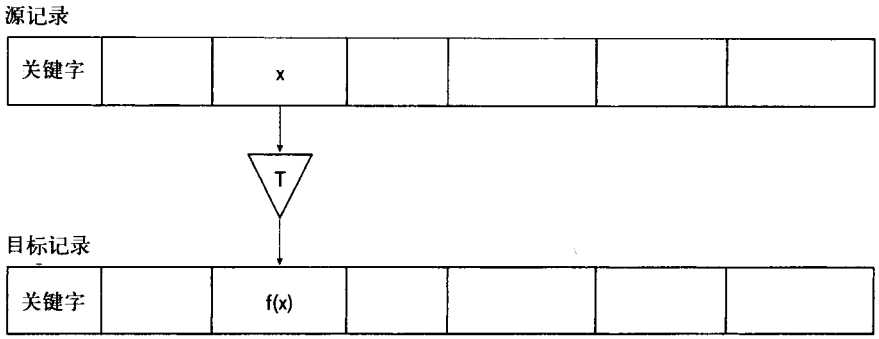
图11-12a是一个多对一的转换的例子（在本例中，将两个源字段映射到一个目标字段上）。在源记录中，员工姓名和电话号码共同作为主键。这种联合主键比较笨拙，而且可能发生不能惟一标识某个人的情况。因此，在创建目标记录时，联合主键就被映射到惟一的员工ID字段上（Emp_ID）。此时，必须创建查找表以支持该转换。同时，还可以使用数据清洗程序确定源数据中的重复部分。

图11-12b是一个一对多的转换的例子（在本例中，将一个源字段转换为两个目标字段）。在源记录中，用产品代码对品牌名和产品名编码（这种编码方法常常用于运作数据）。然而，在目标记录中，需要完整地描述产品和品牌名。此时，同样需要采用查找表来达到这一目的。

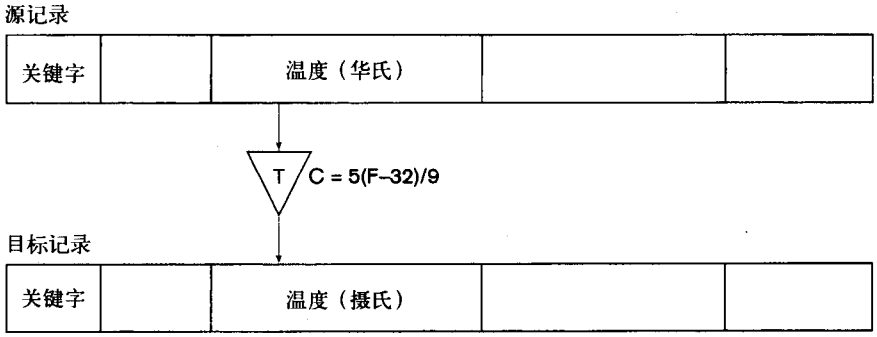
3. *更复杂的转换*

在图11-12中，多字段转换只涉及一个源记录和一个目标记录的情况。但是，多字段转换常常涉及多个源记录和/或多个目标记录。最复杂的情形下，这些记录甚至来自于不同的运作

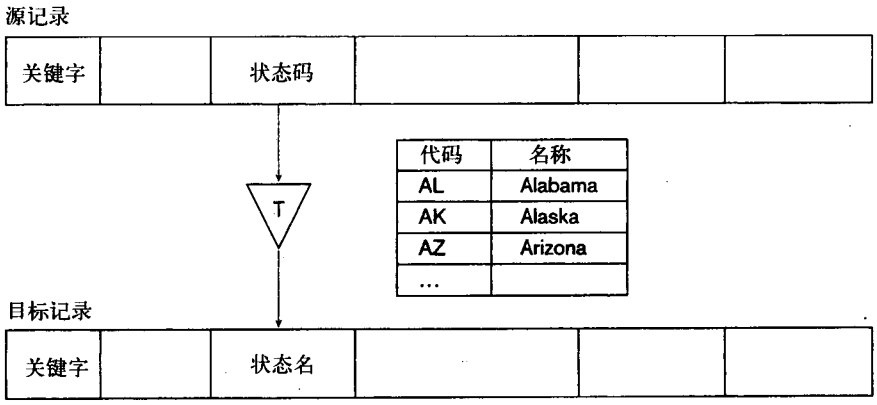
系统和不同的时区（Devlin，1997）。



a) 基本表示



b) 算法转换

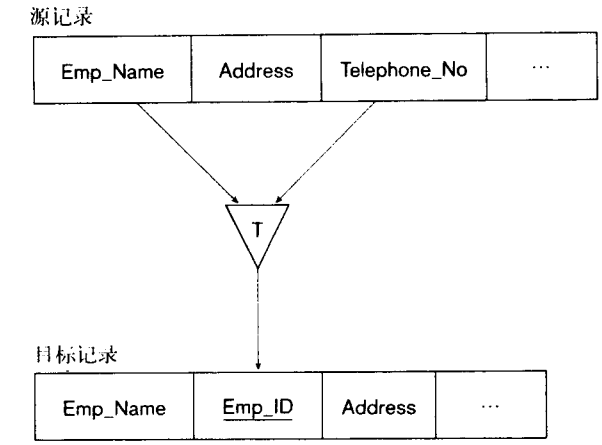


c) 表查找

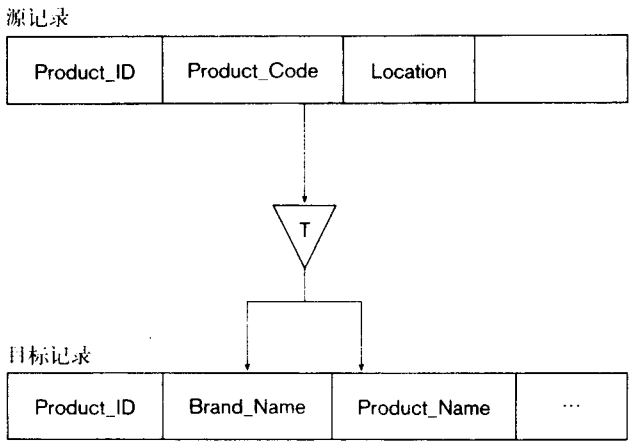
图11-11 单字段转换

11.7.2 支持数据调和的工具

前面已经介绍过，数据调和是一个特别复杂和具有挑战性的过程。必须开发（或者获取）各种紧密集成的软件应用来支持这个过程。值得庆幸的是，已经有许多功能强大的软件工具可以帮助组织开发这些应用系统。本节描述三种这样的工具：数据质量、数据转化和数据清洗（如果想了解更多情况，请参见Williams，1997）。表11-3对这三种工具作了总结。



a) M:1联系



b) 1:M联系

图11-12　多字段转换

表11-3　支持数据调和的工具

产品名称	公　司	描　述
WizRules	WizSoft, Inc. www.wizsoft.com	规则发现
Extract	Evolutionary Technologies International www.evtech.com	抽取、转换、装载和索引
InfoRefiner	Computer Associates www.ca.com	抽取、转换、装载和索引
DataBridge	Taurus Software, Inc. www.taurus.com	抽取、转换、装载和索引
Genio Suite	Hummingbird, Ltd. www.hummingbird.com	抽取、转换、浓缩、装载
Power Mart	Informatica www.informatica.com	抽取、转换、装载和索引

（续）

产品名称	公 司	描 述
Trillium	Harte-Hanks www.harte-hanks.com	质量分析和数据清洗
Centric	Firstlogic www.firstlogic.com	质量分析和数据清洗（参见Information Quality Demo）
Integrity	Vality Technology, Inc. www.vality.com	用于质量分析和数据清洗的产品系列

1. 数据质量工具

这些工具的作用是评估现有系统数据的质量，并且将结果与数据仓库的要求相比较。所以，在仓库开发初期，这些工具是很有用的。表11-3列出了一个这样的工具——WizRules（WizSoft, Inc.），它是一个发现规则的工具，通过搜索现有表中的记录并发现与数据相关的规则。例如，可能有这样一条规则“如果Customer_Name是Able或Baker，那么City就是San Diego，可能性是0.95，该规则在50条记录中存在”。该产品也可以确定从已建立规则中导出的记录。

2. 数据转换工具

表11-3中显示的另外一组工具是数据转换工具。这些工具通常实现三种主要功能：抽取、转换、装载和索引。表11-3中列出的这一类工具包括Extract（Evolutionary Technologies International）、InfoRefiner（Computer Associates）、DataBridge（Taurus Software, Inc.）、Genio Suite（Hummingbird Ltd.）以及Power Mart（Informatica）。

这一类工具基本上是程序生成工具，它们作为源和目标文件的模式（或文件描述）输入，以及准备用作数据转换的业务规则。典型的业务规则包括公式、算法和查找表（如图11-11所示）。接着，这些工具产生必要的程序代码，以便持续执行转换功能。一些工具（如Genio MetaLink for SAP R/3，它是Genio Suite的一部分）提供从流行的应用软件包中进行抽取和转换的模板。

3. 数据清洗工具

最后一类包含一些专用于数据清洗及相关功能的工具。例如，Integrity（Vality Technology, Inc.）、Trillium（Harte-Hanks）和Centric（Firstlogic）。它们用来进行数据质量分析、数据清洗和数据重建（即在数据项中发现业务规则和联系）。

11.8 导出数据层

现在让我们来研究一下导出数据层，它是与逻辑或物理数据集市相关的数据层（参见图11-6），用户通常与这一层进行交互，以进行决策。理想情况下，应首先设计调和数据层作为导出数据层的基础（无论数据集市是依赖的、独立的还是虚拟的）。在本节中，我们首先讨论导出数据层的特征以及从调和数据层中导出数据的方式。然后，介绍星型模式（或多维模型），它是目前用来实现导出数据层的一种常用数据模型。特别是星型模式用来对关系数据模型进行非规范化。虽然我们强调导出数据层可以采用规范化关系，但是大多数组织会构建很多数据集市，每个数据集市都是围绕着一个用户组感兴趣的某一个业务目标而进行优化的。

11.8.1 导出数据的特征

以前，我们将导出数据定义如下：为终端用户的决策支持应用而选择、格式化且聚合的数据。如图11-6所示，导出数据的源是前面讨论过的调和数据。数据集市里的导出数据一般是出于特定用户群（如某个部门、工作小组或者个人）的需要而进行优化的。常见的操作方式是：

从每日运行的EDW中选择相关数据，根据需要将其格式化并进行聚合，然后向目标数据集市装载和索引这些数据。

导出数据的目标与调和数据的目标非常不同。导出数据的目标包括:

1) 为决策支持系统提供方便。

2) 为预定义的用户查询或信息请求提供快速响应。

3) 为特定的目标用户群定制数据。

4) 支持即席查询和数据挖掘应用。

为了满足上述需要，导出数据通常具有如下特征:

1) 包含详细数据和统计数据。

a. 详细数据通常（但并不一直）是周期性的。也就是说，它们提供历史记录。

b. 聚合数据被格式化为能够对预先定义（或一般）的查询作出快速响应。

2) 数据分散在部门服务器上。

3) 数据集市最经常使用的数据模型是星型模式，星型模式与关系模型类似，有时也使用专有模型。

11.8.2 星型模式

星型模式（star schema）是一种简单的数据库设计模式（特别适用于即席查询）。在这种模式下，维度数据（描述通常如何聚合数据）与事实或事件数据相分离（描述单个业务事务）(Devlin，1997)，另外一个经常使用的名字是多维模型（Kimball，1996)。尽管星型模式非常适用于即席查询（和其他形式的信息处理)，但它并不适合联机事务处理，因此在运作系统、运作系统存储或EDW中并没有被广泛采用。

1. 事实表和维表

星型模式包括两种类型的表: 事实表和维表。事实表（fact table）包含实际的、与业务有关的定量数据，例如销售量、预订的订单等。维表（dimension table）则包含关于业务主题的描述性数据。维表通常是在各种查询、报表或图表中用来对事实加以限定、分类或汇总的属性源。最简单的星型模式包括一个事实表和围绕它的几个维表。典型的业务维度（主题）包括Product、Customer和Period。Period或者Time总是维度之一。这种结构如图11-13所示，图中包含四个维表。

每个维表与中心的事实表之间是一对多的联系。一般来说，每个维表有一个主键和若干非主键属性，而主键又是事实表的外键（如图11-13所示)。事实表的主键则是由所有外键串联形成的复合键（图11-13里有四个键)，再加上其他与维表没有对应关系的组件。每个维表与事实表之间的联系为我们提供了一条联结路径，允许用户利用SQL语句方便地查询数据库，这些查询或者是事先定义的或者是即席的。非键属性通常称作数据列，如图11-13所示。

2. 星型模式举例

图11-14给出一个简单的星型模式例子。该例包含三个维表PRODUCT、PERIOD和STORE，以及一个事实表SALES。事实表记录了三个业务事实: 总销售量、总销售额、总成本。这些汇总记录了产品、时期和商店的每个可能值。

该模式的一些样本数据如图11-15所示。对Period为002的110号产品，我们从事实表里发现如下事实:

1) 在商店S1售出30个该产品，总销售额是1500美元，总成本是1200美元。

2) 在商店S3售出40个该产品，总销售额是2000美元，总成本是1200美元。

与维度有关的其他细节可以从维表里得到。例如，在PERIOD中，我们发现002代表2001

年第一季度的5月份。可以尝试以同样方式跟踪其他维度。

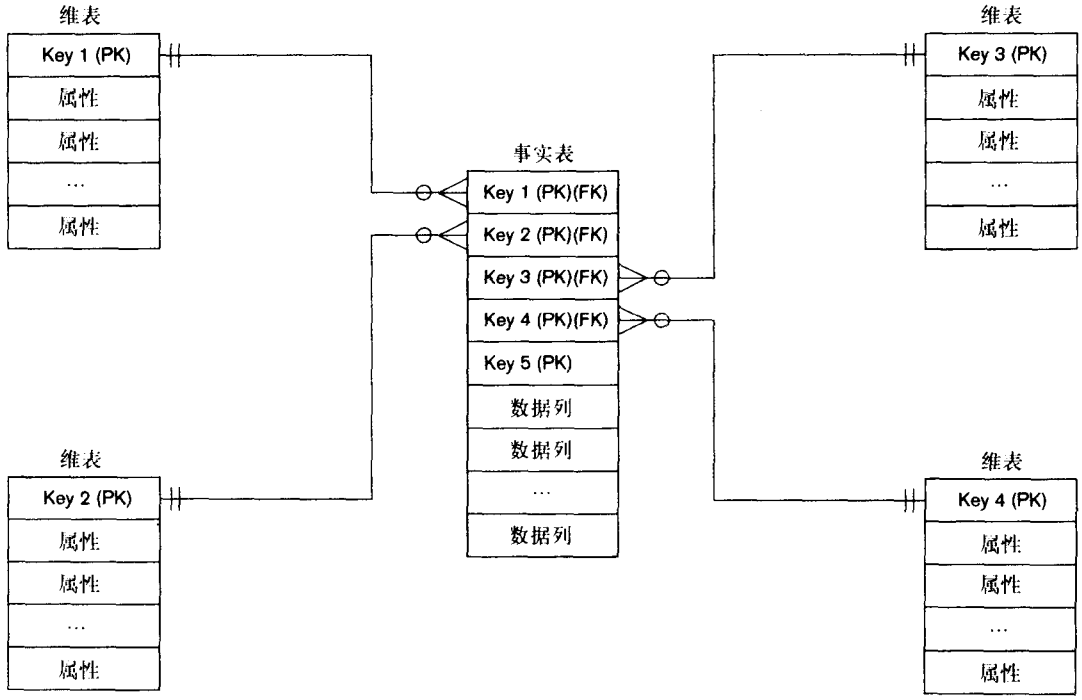


图11-13 星型模式的组成

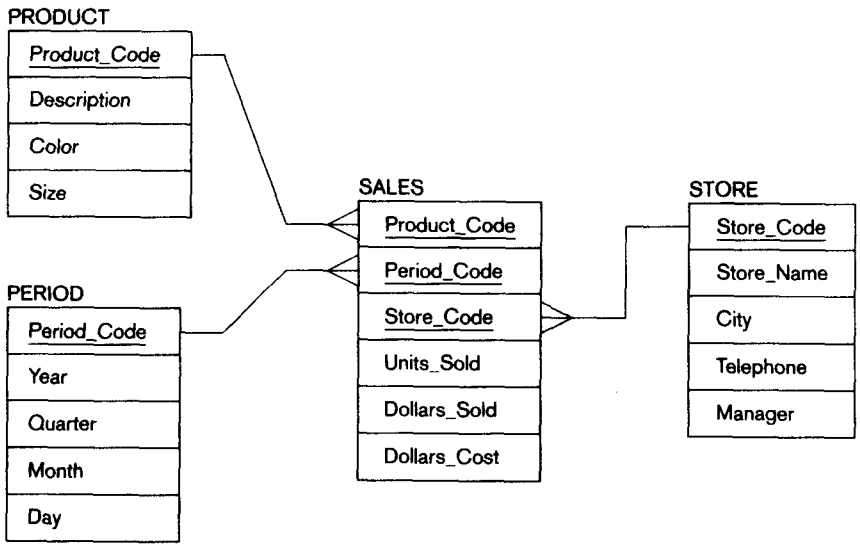


图11-14 星型模式举例

3. 替代键

用来联结事实表与维表的每个键都应该是一个替代键（非智能的或由系统指定的），而不是一个真正采用业务值的键（有时也叫智能键或产品键）。也就是说，图11-15中，事实表和维表里的Product_Code、Store_Code和Period_Code都是替代键。举例来说，如果必须知道一种产

品的产品目录编号、工程编号或库存项目编号，那么这些属性就会作为产品维表的属性与Description、Color和Size一起存储。采用替代键的主要原因如下（Kimball，1998）：

- 通常业务键会随时间缓慢变化，因此我们应该记住同一个业务对象的业务键的旧值与新值。在后面介绍减缓变化维度的内容中我们将看到，利用替代键可以方便地处理变化键和未知键。
- 使用替代键也便于我们跟踪相同产品随时间变化而不同的非键属性值。因此，如果某个产品的包装尺寸发生变化，我们可以把一个产品键与几个不同的替代键联系起来，每个替代键对应不同的包装尺寸。
- 替代键通常比较简单，也比较短，尤其当产品键是复合键时。
- 所有的替代键可以有相同的长度和格式，无论其对应数据库的哪个业务维度，甚至是日期也不例外。

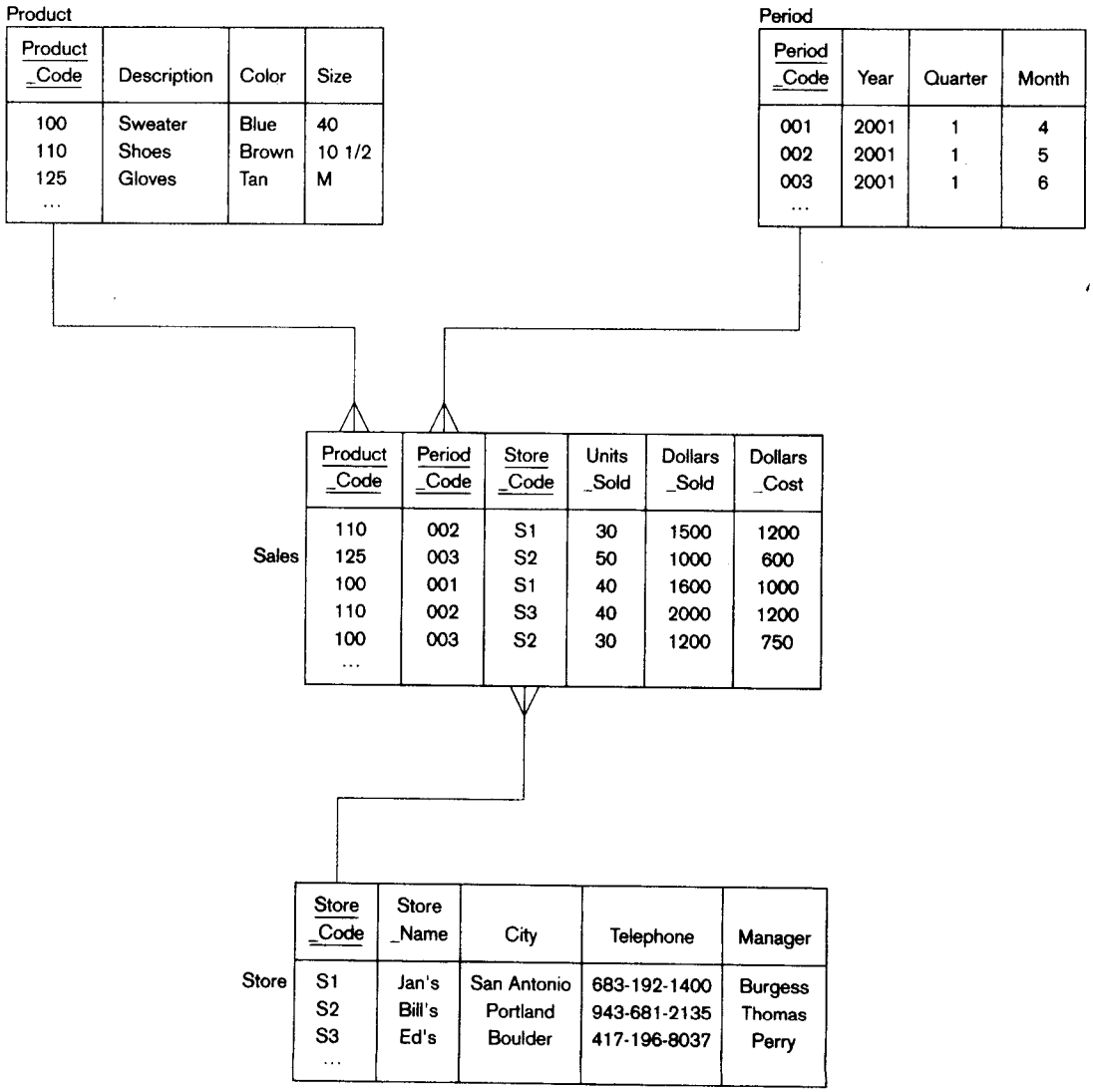


图11-15　带有样本数据的星型模式

4. 事实表的粒度

星型模式的原始数据保存在事实表中。事实表明，决定所存储的详细事实数据的最低级别是数据集市设计过程中最为重要的一步。数据的详细级别由组成事实表主键的所有部分的交集决定。主键的交集称为事实表的**粒度**（grain）。确定粒度是非常关键的一个方面，而且从业务决策的角度而言也必须确定粒度。通过利用维度属性进行聚合是汇总事实数据的一个方法；但是在数据集市里没有办法在小于事实表粒度的细节级别上理解业务活动。

可能的最小粒度即为每个业务事务，如产品销售单据上的某一项、一条职员变动命令、材料收据上的某一项、关于保险策略的一项声明、一个ATM交易等。事务级粒度允许用户执行**购物篮分析**（market basket analysis），购物篮分析可以对某个顾客的购买行为进行研究。高于事务级别的粒度可以是某天的产品销售量、某月在某个特定仓库的所有原材料收据或者是某个ATM会话上所有ATM事务的净效应。事实表的粒度越小，存在的维度越多，存在的事实行也越多，通常情况下，数据集市模型与运作数据存储的数据模型也越接近。

在数据集市技术有一定局限性的情况下，Kimball（2001）和其他专家推荐采用尽可能小的粒度。尽管最初数据集市的用户可能只需要某个聚合级别的粒度，然而在使用一段时间后，用户可能会问一些更加详细的问题（下钻）以解释为什么存在某种聚合模式。而那时，已经不可能再“下钻”到事实表的粒度之下（不包括转向其他的数据源，如EDW、ODS或者最初的源系统，因为分析起来相当困难）。

5. 事实表的尺寸

和我们所预期一样，事实表的粒度对事实表的尺寸有直接的影响。我们可以用下面的方法估计事实表的行数：

1) 为每个与事实表相关的维度估计可能值的数目（换句话说，就是事实表中每个外键可能值的数目）。

2) 在作出任何必要的调整后，把1所得到的所有值相乘。

现在，把上述方法应用到图11-15所示的星型模式中。假定维度采用如下值：

商店总量：1000

产品总量：10 000

时间总量：24（两年的月数据）

虽然产品总量为10 000，但某个月内可能只记录了一部分产品的销售情况。因为事实表只记录某个月的销售情况，所以我们需要调整这个数字。假定每个月记录了大概50%（或者说5000）产品的销售情况，那么，事实表的行数可以估算如下：

总行数＝1000（商店）×5000（活动产品）×24（月）

＝120 000 000（行）

因此，在这个简单的例子中，包含两年的月数据的事实表的行数超过1亿行。本例清楚地说明，事实表的尺寸是维表的很多倍。比如，STORE表只有1000行，PRODUCT表有10 000行，而PERIOD表只有24行。

如果我们知道事实表每个字段的尺寸，就能够进一步估算以字节为单位的表的尺寸。图11-15中事实表（名为SALES）有六个字段。如果每个字段的平均长度为四个字节，则事实表总的尺寸就可以用如下方法估算：

总尺寸＝120 000 000（行）×6（字段）×4（字节/字段）

＝2 880 000 000或2.88G（字节）

事实表的尺寸取决于维度的个数和事实表的粒度。假定在使用一段时间图11-15所示的数据库后，营销部门要求在事实表中累加每日的销售总数（这是一种典型的数据集市演变）。随着表的粒度变为每日项目总计，事实表的行数也可以用如下方法估算：

总行数＝1000（商店）×2000（活动产品）×720（天）
　　＝1 440 000 000（行）

在上述计算中，我们假定每天大约有20%的产品有销售记录。此时，数据库行数将超过10亿，而数据库的尺寸则变为：

总尺寸＝1 440 000 000（行）×6（字段）×4（字节/字段）
　　＝34 560 000 000或34.56G（字节）

现在，许多大的零售商（如沃尔玛、Kmart、Sears）和电子商务站点（如Travelocity.com和MatchLogic.com）都拥有自己的数据仓库（或数据集市），这些数据仓库中大多数的尺寸都在几个TB左右。而且，随着营销人员不断要求事实表具有更多的维度和更小的粒度，这些数据仓库的尺寸还会快速增长。

6. 对日期和时间建模

因为数据集市随时间变化记录不同维度的事实，所以日期和时间（此后简称日期）总是一个维表，而日期替代键也一直都是任何事实表的主键的组成部分。由于用户可能想聚合日期的许多不同方面的事实，因此日期维度可以有许多非键属性。同样，因为日期的某些特征是与特定地区或是特定事件有关的（例如，某个日期是假期或者在某天会发生某些标准事件，比如过年或举行足球赛等），对日期维度建模会比目前所阐述的更为复杂。

图11-16显示了日期维度的一种典型设计。正如我们前面所看到的，日期替代键作为事实表主键的一部分出现，并且是日期维表的主键。日期维表的非键属性包括日期的所有特征，用户利用这些日期对不随地点或事件而改变的事实进行分类、汇总和分组。对于一个在不同国家（或不同的地理位置，在那些地方日期的特征有所不同）开展业务的组织而言，我们可以添加一个Country Calendar表，该表用于记录在每个地方的每个日期的特征。因此，Date键是Country Calendar表的一个外键。通过组合Date键和Country，Country Calendar表的每一行都是唯一的。所以，Date和Country形成该表的复合主键。同样，对于某个日期发生某个特别事件的情况而言（为了简单起见，我们假定某个给定日期所发生的特殊事件不超过一个），也可以通过创建事件表规范化Event数据，因此每个事件的描述性数据（如“Strawberry Festival”或“Homecoming Game”）只需存储一次。



图11-16　对日期建模

11.8.3 星型模式的变体

上面所介绍的简单星型模式对许多应用而言已经足够了。但是，为了处理更复杂的建模问题，通常需要对该模式进行各种扩展。在本节中，我们将简要地介绍两种扩展：多事实表和雪花模式。想要更多了解其他扩展和变体，可以参考Poe（1996）及www.ralphkimball.com。

1. 多事实表

为了改善性能或其他原因，经常需要在给定星型模式中定义多个事实表。例如，假设不同用户需要不同级别的聚合（换句话说，有不同的表粒度），通过为每种聚合级别定义不同的事实表有助于改善性能。这种做法的明显缺点是：随着新表的增加，存储要求也会急剧升高。更为一般的情况是，需要多事实表存储用于不同维度组合的事实。

图11-17说明了多事实表的一种典型情况。该图包含两个相关的星型模式，另外还包含两个事实表分别处于两个星型模式的中心位置：

1) Sales——关于某天在某个商店对某个顾客销售某个商品的事实。

2) Receipts——关于某天某个供应商的商品入库的事实。

在这种情形下出现的一个共同问题是：有关一个或多个业务主题的数据（本例中为Product和Date）应该被存储到每个事实表的维表中。在设计中可以采用两种方法处理维表的共享问题。一种情况是，因为Sales表和Receipts表中对商品的描述差异很大，所以创建两个商品维表；另外一种情况是，由于用户需要相同的日期描述，所以使用同一个日期维表。在这两种情况下，都要求创建**一致性维度**（conformed dimension），也就是说，对每个事实表，维度的意义都相同，因此，维表可以采用相同的替代主键。即使两个星型模式分别存储在不同的物理数据集市中，但只要维度是一致的，就可以跨数据集市提出问题。（例如，某个供应商的产品销售的更快吗？他们能在较短时间内补货吗？）简单来说，一致性维度允许用户：

- 共享非键维度数据。
- 跨事实表的一致性查询。
- 在意义统一的事实和业务主题上工作。

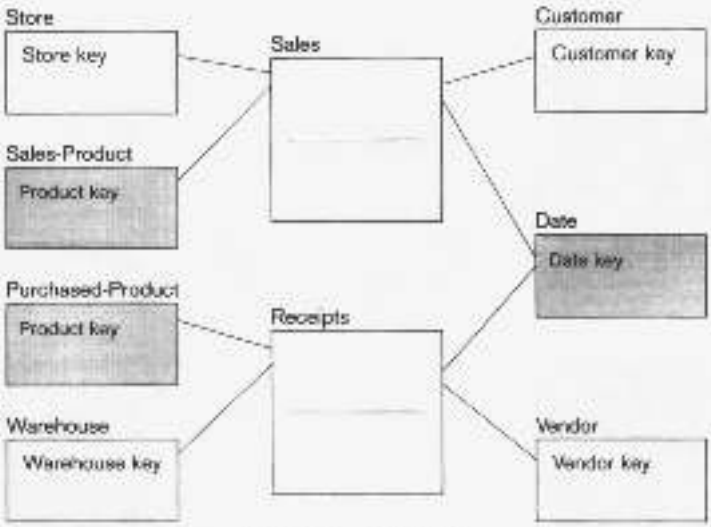
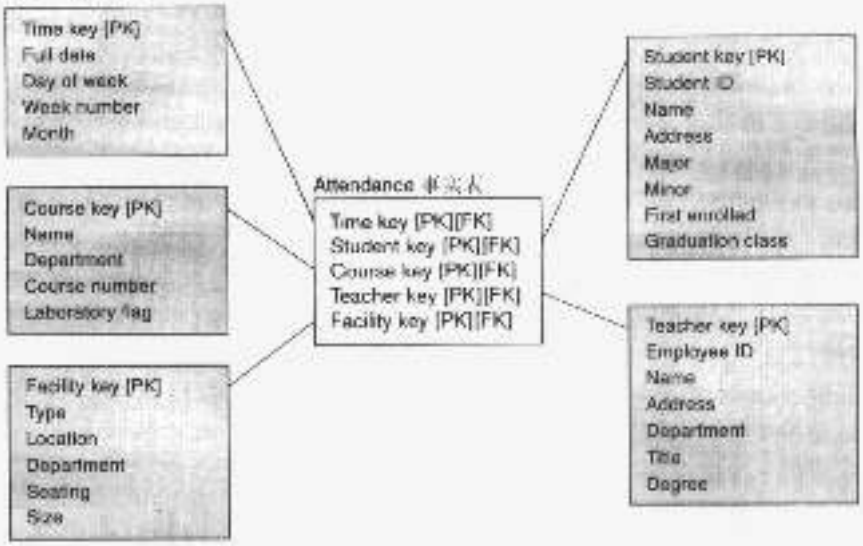


图11-17 一致性维度

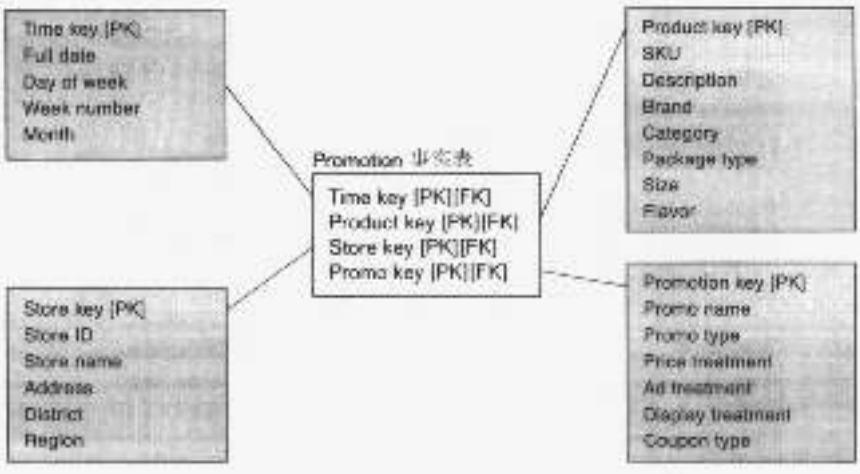
2. 无事实的事实表

有一些应用的事实表没有非键数据，而只包含与维有关的外键。无事实的事实表通常在以

下两种情况中应用：跟踪事件（如图11-18a所示），列出可能发生的事件集合（称为覆盖）（参见图11-18b）。图11-18a中的星型模式跟踪某个学生在某个时间利用某种教学仪器参加某个教师教授的某门课程。应该知道的所有事实就是是否发生该事件，答案由五个外键的交集表示。图11-18b所示的星型模式则显示某个商店在某个时间利用某种促销方式时，某种商品可能的销售情况。图11-18b没有显示另外一张销售事实表，除了相同的维度组合（与Promotion事实表相同的四个外键）以外，这张表还包含销售数和销售额（非键事实）。利用这两个事实表和四个一致性维度，通过在促销事实表中发现四个键值的某个组合在销售事实表中并不存在，就能够得出某种商品在某个时间在某个商店的某次促销活动里没有卖出去（即销售额为零）的结论。如果只有销售事实表则不足以回答这个问题，因为在销售事实表里不可能有四个键值的组合对应销售额为零的一行记录。



a）显示发生某个事件的无事实事实表



b）显示覆盖的无事实事实表

图11-18　无事实的事实表

11.8.4 维表的规范化

事实表是完全规范化的，因为每个事实依赖且仅依赖于复合主键。而另一方面，维表可能不是规范化的。大多数数据仓库专家认为，可以将数据集市进行优化和简化以适应某个用户群体，这样，所有的维度数据只是远离相关事实的一个联结。有时，正如任何关系数据库一样，非规范化维表的异常常常会在进行添加、更新和删除时产生问题。在本节中，我们会讨论有必要对维表做进一步规范的各种情形。

1. 多值维度

对于相同的业务主题，有必要把事实的取值限定在一定范围内。图11-19展示了一个有关医院的例子。本例中，医院在某天对某个病人的收费和付费（比如说，Finances表的所有外键）与一次或多次诊断有关（通过连接Diagnosis表和Finances表之间M:N联系的虚线表明了这一点）。可以用最重要的一次诊断作为Finances表主键的组成部分，但这就意味着我们有可能失去与该行事实相关的其他几次诊断的重要信息。或者，可以在Finances表中设计固定数目的诊断键，该数目大于与Finances表的一行数据相关的可能诊断次数，但这又会产生很多主键为空的行数据，这种情况是违反关系数据库性质的。最佳方法（规范化方法）就是创建一个表作为Diagnosis表和Finances表之间的关联实体。在数据仓库的数据库领域里，这种表被称为"助手表"。在后面几节中我们还会陆续看到更多助手表的例子。助手表可以有（任何关联实体表都可以有的）非键属性，例如图11-19所示"Diagnosis group"表里的weight factor（加权因子），它表明在某个诊断组里各次诊断所起的作用，一个诊断组所有诊断的加权因子的和被归一化为100%。另外还要注意，Finances表里可能会有多行与同一个诊断组相关联。

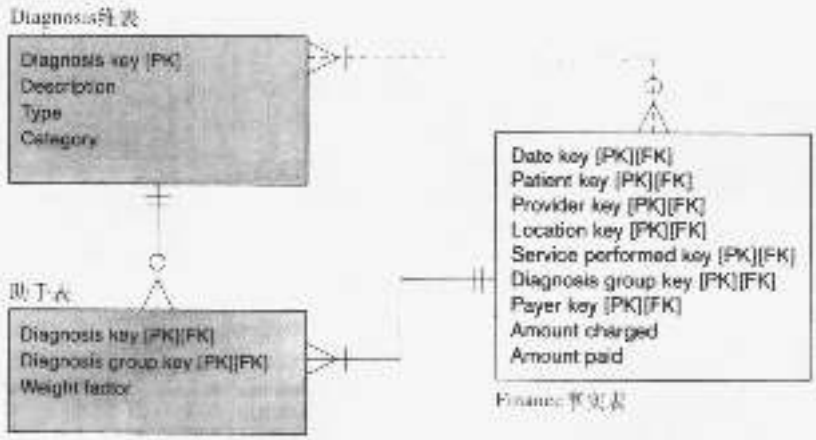


图11-19 多值维度

2. 层次结构

很多时候，星型模式的某个维度会自然形成层次结构。例如，地理层次结构（市场-州-地区-国家），商品层次结构（小包装-商品-捆-箱）。当某个维度构成层次结构时，数据库设计者可以有以下两种选择：

1) 在一个单独的非规范化的维表里包含层次结构中每一层的所有信息，这种方法会产生相当大的冗余和更新异常。

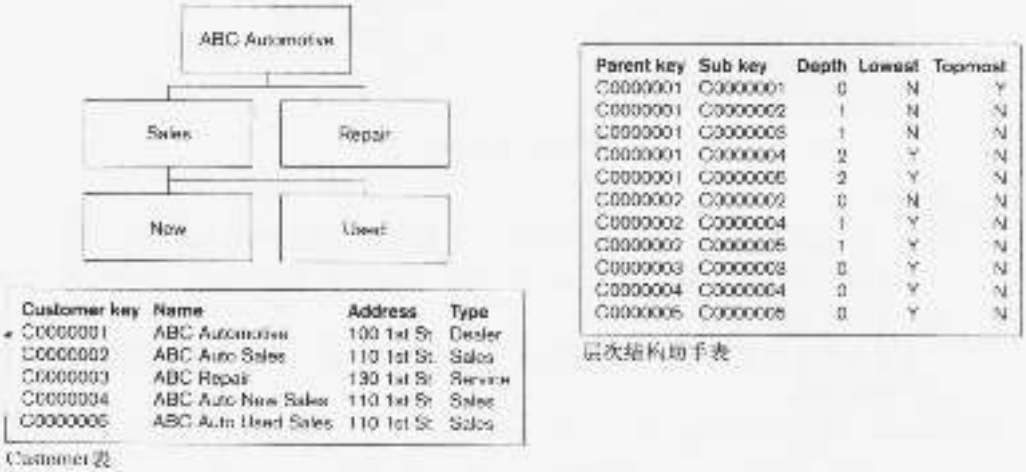
2) 把该维度规范化为一组嵌套的表，表与表之间呈1:M联系。此时，聚合层次结构中任何层次上的事实数据仍然是可能的，但用户必须沿着层次路线进行嵌套联结，或者向用户提供一个预先联结的层次视图。

现在，让我们考虑一个典型的咨询公司的例子，该公司为顾客开具特定时间段关于项目的发票。在这种情况下，一张收入事实表可以显示每张发票记入了多少小时的多少收入。发票中记录了特定的时间段、特定的顾客、特定的服务、特定的员工和特定的项目。因为可以为相同组织的不同部门而进行这种咨询工作，所以如果我们想知道在顾客组织的任何级别上咨询所起的总体作用，就需要一个顾客层次结构。组织各部门之间的这种层次结构就是递归关系，如图5-17所示。在规范化数据库中表示该层次结构的标准方式就是在公司行中放入一个外键，用以代表其父母的单位。

用上述方式实现的递归联系对于进行决策的终端用户来说是非常困难的，因为确定在某个层次上如何进行聚合要求编写复杂的SQL程序。另外一种较为简单的方法如图11-20所示。图11-20a显示如何在数据仓库中利用助手表对层次结构建模的方法（Kimball，1998b；Chisholm，2000）。咨询公司所服务的组织的每个部门都被分配了一个不同的客户替代键和Customer维表中的一行，该客户替代键在Revenue事实表中被用作外键。在一个任意深度的递归联系中进行联结所面临的问题是：用户不得不编写代码进行任意次数的联结（每个附属级别都要进行一次联结），而数据仓库的这些联结，因为其尺寸过大，可能会非常耗时（除非利用某些高性能的使用了并行处理的数据仓库技术）。为了避免该问题，助手表通过在一行中记录每个部门的子部门和包括其本身在内的上级部门（所有通往客户组织顶层的路径）把层次关系扁平化了。该助手表的每一行都有三个描述符——该行子部门距离上级部门的层数，该子部门是否为最低层的标志，该子部门是否为最高层的标志。图11-20b描述一个客户组织层次结构的实例以及助手表中代表整个组织的行（其他客户组织的子部门-父部门关系在助手表的其他行中记录）。



a) 助手表的使用



b) 带有客户表和助手表的层次结构实例

图11-20　在一个维度中表示层次关系

图11-20a中的Revenue事实表有一个非键属性：Invoice Number。该属性就是所谓的生成维度（generative dimension）的一个实例，它没有感兴趣的维度属性（所以不存在相应的维表，Invoice Number也不是表的主键的组成部分），而且也不是被用于聚合的事实。但是，如果需要开发一个能发现关于发票事务其他细节的ODS或源系统，则该属性是有用的。

当维表通过利用助手表（有时也称为桥表或参照表）而得到进一步规范化时，简单的星型模式就演变成**雪花模式**（snowflake schema）。雪花模式类似于一个ODS或源数据库的片段，以汇总为事实表的事务表为中心，所有其他表直接或间接地与这些事务表相关联。很多数据仓库专家并不鼓励使用雪花模式，因为这种模式对用户而言过于复杂，并且需要更多的联结操作才能把结果放在一个表中。然而，如果规范化可以节省大量的冗余空间（例如，有很多冗余的、长文本的属性时）或者用户认为浏览规范化表非常有用时，雪花模式也是值得使用的。

11.8.5 缓慢变化的维度

数据仓库和数据集市随时间流逝一直在跟踪业务活动。业务不会随着时间流逝而保持静止——商品会改变尺寸和重量，顾客也会发生变化，商店会改变布局，销售人员也会到不同的地方开展工作。大多数记录系统仅仅记录业务主题的当前值（比如，当前的顾客地址）。但是在数据仓库或数据集市中，我们应该了解历史值，以便把历史事实与事实发生时正确的维度描述匹配起来。例如，我们应该在发生某个销售事实的时间段内将该销售事实与相关顾客描述之间关联起来，这个描述可能并不是对客户的当前描述。当然，业务主题的变化与大多数运作事务相比而言是较为缓慢的。因此，我们说维度数据是不断变化的，只是变化速度较慢。

可以用如下三种方式来处理变化着的维度属性（Kimball，1996b和1999）：

1）用新值重写当前值。但由于这种方式删除了解释历史事实时所需的描述，所以是不可行的。

2）为每个变化的维度属性建立一个当前值字段和若干旧值字段（例如，有固定数量事件的多值属性）。如果在数据仓库的某段历史时间里可以预测维度变化的数量，那么这种方案是可行的（比如，我们保持24个月的历史，并且某个属性的值最多一个月变化一次）。但是，这种方法只能在受约束的假定下奏效，而不适用于任何缓慢变化的维度属性。

3）每次维度对象改变时都建立一个新的维表行（有一个新键），这个新行包含所有的维度特征。这个新键与事实行相关联，键的属性在事实所发生的时候应用。还可以存储变化发生的日期/时间及变化发生的原因代码。这种方法使我们可以根据需要建立维度对象变化。不过，如果行频繁改变或者行很长，那么这种方法就显得很笨拙。

第三种方案是处理缓慢变化维度的一种常用方法。利用这种方案，也可以在维度行里为初始对象存储替代键值，从而可以把所有变化与它们的初始值联系起来。

不过，需要注意的是，当维度对象频繁改变或维度行较大时，维表的行数可能会变得非常大。图11-21展示了处理这种情况的一种方法——维度分段，该方法还能处理一种更为普遍的情况——维度属性的子集以不同的频率发生变化。在这个例子中，Customer维度被分成两个维表：一张表保存几乎不变化或变化速度非常缓慢的维度，而另一张表保存变化速度较快的属性聚簇，相同聚簇的属性同时发生变化。这些变化较快的属性通常被数据仓库设计者称为“热”属性。

分割的另一个方面是，对于热属性，可以把一个单值维度属性改变为范围维度属性，如把顾客收入从$75 400/年变成$60 000 ~ $89 999/年。范围可以由用户根据需要定义，可以大一些也可以小一些。当然，这种方法会失去一些精确性，但范围的使用使热属性变得不是那么“热”了，因为如果属性的变化处于该范围内，那么就不会导致向维表写入一个新行。不过，这种设计对用户而言非常复杂，因为根据分析，他们也许不得不将事实与多个维度片段联结起来。

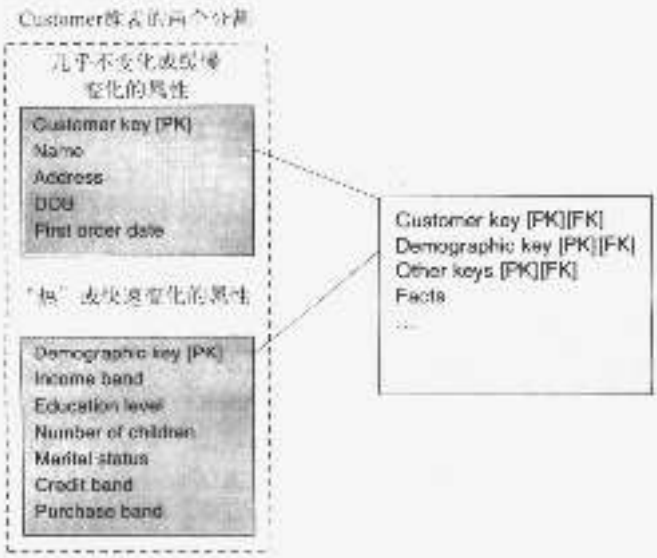


图11-21　维度分割

11.9　用户界面

“有产品就有市场”策略在某些场合也许奏效。但对于数据仓库和数据集市而言可能不一定适用。即使是一个设计良好的数据集市，并且装载了相关数据，也可能无法使用，除非提供给用户一个功能强大的直观的界面，使得他们能够很容易地访问和分析这些数据。在本节中，我们将对现代数据仓库的界面进行简要介绍。

目前有各种工具可以用来查询和分析数据仓库或数据集市中存储的数据。这些工具大体可分为如下几类：

1）传统的查询和报表工具。

2）联机分析处理（OLAP）、MOLAP和ROLAP工具。

3）数据挖掘工具。

4）数据可视化工具。

传统的查询和报表工具包括电子表格、PC数据库、报表编写程序和生成程序。由于篇幅所限（也因为在相关的参考书里已经包含了这部分内容），我们在这里不再介绍这些工具。本节在讨论元数据的作用之后，主要讲述其他三类工具。

11.9.1　元数据的作用

建立一个友好用户界面的首要条件就是有一组元数据。它们用便于用户理解的业务术语描述数据集市中的数据。图11-6展示了在三层数据体系结构中元数据与数据集市的关联。

与数据集市相关的元数据通常称为“数据目录”（data catalog或data directory）。元数据对于数据集市中的数据起到一种类似“黄页”的作用。有了元数据，用户就能很容易地回答下面的问题：

1）数据集市描述什么主题？（典型的主题有：顾客、病人、学生、产品、课程等等。）

2）数据集市包括哪些维和事实？事实表的粒度是什么？

3）数据集市的数据如何从EDW的数据中导出？在导出时使用了什么规则？

4）EDW的数据如何从运作数据中导出？在导出中使用了什么规则？

5) 为了查看数据，可以使用那些报表和预定义的查询？

6) 可以使用哪些“下钻”和其他数据分析技术？

7) 谁负责数据集市中数据的质量？向谁提出变化请求？

11.9.2 查询工具

SQL作为使用最普遍的数据库查询语言（参见第7章和第8章），常常被扩展以支持数据仓库环境下某些类型的计算和查询。但是，一般来说，SQL不是一种分析语言（Mundy，2001）。SQL-99包含了一些数据仓库扩展。因为很多数据仓库操作都会处理某种类型的对象（比如说按时间排序的对象），所以SQL-99包含一个可以定义一组动态行的WINDOW子句。例如，可以用一个WINDOW子句定义相邻的三天，从而计算移动平均值（可以想像成一扇窗户在窗格的低端和顶端之间移动，提供数据行的滑动视图）。WINDOW子句的PARTITION与GROUP BY作用类似。PARTITION告诉WINDOW子句每组的基础。ORDER BY子句对一组元素排序。ROWS子句求得用于计算的一个序列包含多少行。RANK窗口函数能够计算某些在标准SQL中难以计算的内容，即那些基于某些准则的、在某个特定相对位置的表的行（例如，在某个时间段内销售额排在第二位的顾客）。WINDOW子句的实例可参见8.6.1节。SQL-99并不是一个功能全面的数据仓库查询和分析工具，但它在认识决策支持系统的特殊查询需要方面的一个开始。

11.9.3 联机分析处理工具

目前已经开发了一种向用户提供数据的多维视图的特殊工具，然后利用图形界面就能够很容易地分析数据。在最简单的情形下，可将数据看作一个简单的三维立方体。

联机分析处理（On-Line Analytical Processing，OLAP）就是使用一组图形工具向用户提供数据的多维视图，并允许用户利用简单的窗口技术对数据进行分析。之所以称之为“联机分析处理”，是为了与传统的术语“联机事务处理”（OLTP）相对比。这两种处理类型之间的区别已经在表11-1中作了总结。术语“多维分析”常用作OLAP的同义词。

图11-22展示了一个典型的OLAP“数据立方体”（或多维视图），该视图与图11-14中介绍的星型模式非常接近。图11-22中的两个维度与图11-14中的两个维表（PRODUCT和PERIOD）相对应，而第三个维（名为Measures）与图11-14中的事实表（名为SALES）的数据相对应。

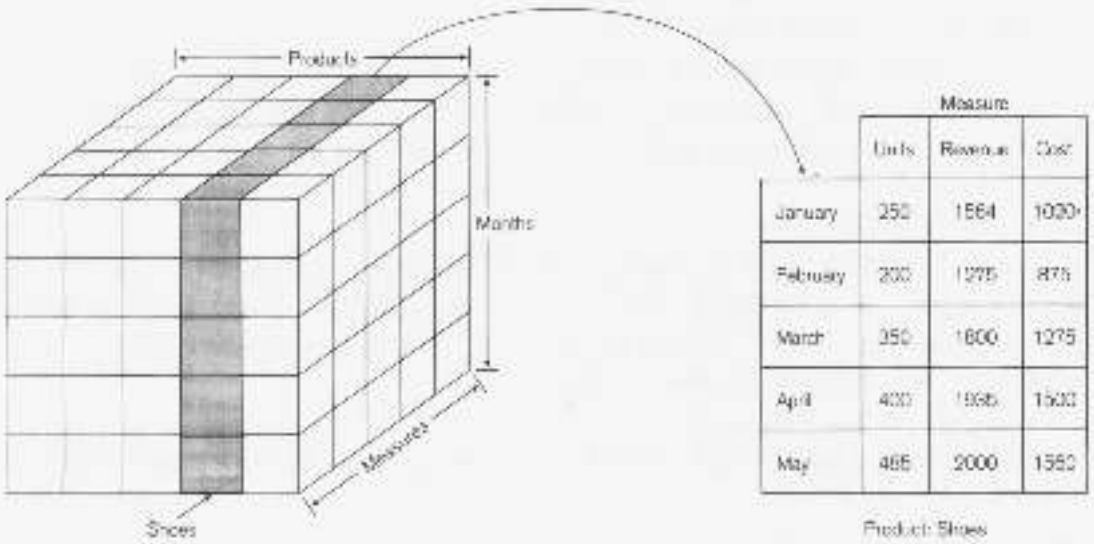


图11-22 数据立方体的切片

OLAP实际上是几类数据仓库和数据集市访问工具的总称（Dyché，2000）。**关系OLAP**（Relational OLAP，ROLAP）工具采用SQL的变体并把数据库看作传统的关系数据库，无论它是星型模式还是其他的规范化/非规范化表集合。ROLAP工具直接访问数据仓库或数据集市。**多维OLAP**（Multidimensional OLAP，MOLAP）则是把数据装入一个中间结构，通常是三维或更高维的数组。因为MOLAP的使用比较普遍，所以我们将在下面几节中详细介绍这种工具。使用MOLAP时必须注意，数据不仅仅被看作多维数组（三维立方体），而且还通过从数据仓库或数据集市中抽取数据，然后把数据保存在一个特殊的、单独的数据存储里来创建MOLAP数据集市，只有借助多维结构才能看见这些数据。此外，还有两种不太常用的OLAP工具：一种是DOLAP（Database OLAP），这种工具是在DBMS查询语言中嵌入OLAP功能（由非ANSI的、专用的标准SQL系统实现）；另一种为HOLAP（Hybrid OLAP），这种工具既允许利用多维立方体也允许利用关系查询语言对数据进行访问。

1. 对立方体切片

图11-22还展示了一个典型的MOLAP操作——将数据立方体切片以产生简单的二维表或视图。在图11-22中，进行这种切片操作是为了获得鞋类商品的信息，结果数据表显示该类商品在某段时间（月份）内有三种度量（售出量、收入和成本）。凭借简单的“拖放”操作，用户可以很容易地开发其他视图。这种操作通常称为对立方体的“切片或切块”。

另外一种与切片和切块紧密相关的操作是数据换轴（data pivoting）。这个术语指的是以某一个数据点旋转视图，以从另外一个角度来查看数据。例如，图11-22显示4月份鞋类的销售数目是400，分析者可以旋转该视图获得4月份各个商店的鞋类销售量。

2. 下钻

下钻（drill-down）是另外一种在多维分析中经常运用的操作，即在更详细的级别上分析给定的数据集合。图11-23显示了一个下钻的例子，图11-23a是一个指定品牌纸巾的三种包装尺寸（2卷装、3卷装和6卷装）的总体销售额汇总报表。由于这些纸巾的颜色也各不相同，所以分析人员想进行更详细的分类，即在上述包装尺寸下再将纸巾按颜色作进一步分类。使用OLAP工具，通过鼠标点击方法可以很容易得到这种更为细化的分类。

下钻的结果如图11-23b所示。注意，下钻结果的表示相当于向初始报表中添加新的一列（本例中，添加了一列名为“Color”的属性）。

执行下钻操作（正如本例所示）可能要求OLAP工具具有“回访”数据仓库的能力以获得下钻所必须的细节数据。仅当OLAP工具可以得到一组集成的元数据时，这种类型的操作可以由该工具执行（用户不参与）。如果某个查询需要的话，某些工具甚至允许OLAP工具回访运作数据。

11.9.4 数据挖掘工具

有了联机分析处理，用户可以搜索所提问题的答案。例如，对于未婚者和已婚者，谁的健康医疗费用更高？而利用数据挖掘，用户可以在事实或观察的结果中寻找模式或趋势。**数据挖掘**（data mining）是一种混合了多种技术的知识发现，这些技术包括传统的统计学技术、人工智能技术和计算机图形技术（Weldon，1996）。

数据挖掘可达到以下三个目标：

1）**解释性**　为了解释一些观察到的事件或状况，例如，为什么科罗拉多的敞篷小型载货卡车的销售量增加了？

2）**确定性**　为了确定某些假设，例如：双收入家庭是否比单收入家庭更有可能购买家庭医疗保险？

3) **探查性** 为了新的或预期之外的联系而分析数据，例如：有哪些消费模式可能伴随着信用卡欺诈?

Brand	Package size	Sales
SofTowel	2-pack	$75
SofTowel	3-pack	$100
SofTowel	6-pack	$50

a) 汇总报表

Brand	Package size	Color	Sales
SofTowel	2-pack	White	$30
SofTowel	2-pack	Yellow	$25
SofTowel	2-pack	Pink	$20
SofTowel	3-pack	White	$50
SofTowel	3-pack	Green	$25
SofTowel	3-pack	Yellow	$25
SofTowel	6-pack	White	$30
SofTowel	6-pack	Yellow	$20

b) 加入颜色属性后的下钻结果

图11-23 下钻的例子

1. 数据挖掘技术

数据挖掘通常使用几种不同的技术，表11-4对一些最常用的技术作了总结。如何选择合适的技术取决于将要被分析的数据的性质以及数据集合的大小。在数据集市或EDW中都可以进行数据挖掘。

表11-4 数据挖掘技术

技 术	功 能
实例推理（法）	从现实世界的实例中推导规则
规则发现	在大型数据集合中寻找模式和关联
信号处理	以相同的特征识别一组观察结果
神经网络	基于人脑的建模原理开发预测模型
分形	无损压缩大型数据库

2. 数据挖掘应用

数据挖掘技术已经被成功应用到现实世界当中，表11-5对一些典型的应用类型作了总结，每种类型都给出了相应的例子。数据挖掘应用迅速增长的主要原因如下:

1) 数据仓库和数据集市中的数据量呈指数级增长，因此用户需要数据挖掘所提供的自动化技术以从这些数据中挖掘知识。

2) 拥有扩展功能的新的数据挖掘工具不断被引入。

3) 不断增长的竞争压力迫使公司更好地利用数据中所包含的信息和知识。

表11-5　典型的数据挖掘应用（摘自Zaitz, 1997及Dyché, 2000）

应用类型	举　例
人口建模	为价值很高的顾客、信贷风险和信用卡欺诈开发概要文件
业务趋势分析	确定以高于（或低于）平均水平的速度增长的市场
目标营销	确定促销活动面向的顾客或顾客群
可用性分析	确定产品和服务的使用模式
营销活动的效果	比较营销活动策略的效果
产品亲和性	确定同时被购买的产品，或者购买某组产品的购买者特征
顾客的维护和流失	为了防止顾客继续流失到竞争者那边去而进行的顾客行为测试
赢利分析	给定顾客与组织之间全部活动的集合，决定哪些顾客是可获利顾客
顾客价值分析	确定在不同的年龄阶段，哪些是有价值的顾客
追赶销售	基于关键事件和生活方式的变化，确定给顾客提供的新产品或服务

11.9.5　数据可视化

一般而言，用图形方式表现数据更便于人们用眼睛来分辨模式。**数据可视化**（data visualization）就是为了便于人类分析而用图形和多媒体的方式表示数据。数据可视化的优势包括：能更好地观察趋势和模式、识别关联和聚簇。数据可视化经常与数据挖掘和其他分析技术同时使用。

本章小结

当前，尽管组织收集了大量数据，但大多数管理者在获得制定决策所需的信息时还是存在困难。这种“信息鸿沟”主要是由以下两个因素造成的：首先，数据通常是异构和不一致的，这是通常是因为采用分步系统开发方法所造成的。其次，系统主要是为了满足运作目标而开发的，几乎不能给予管理者所需的信息。

在运作系统和信息系统以及它们所拥有的数据之间存在着很大的差异。运作系统用来在当前基础上执行业务，其主要的目标是向用户提供高性能的事务处理和数据库更新。信息系统则是用来支持管理者制定决策，其目标是为信息工作者提供便捷的数据访问和使用。

数据仓库的目的是合并和集成来自各种源系统的数据，并根据需要改变它们的格式以作出正确的业务决策。也就是说，数据仓库是一个集成的、一致的、面向主题的数据存储，其中的数据来自于各种源系统，并格式化为某种有意义的格式以支持组织制定决策。

今天，大多数数据仓库使用三层体系结构。第一层由分散在不同运作系统中的数据组成；第二层是一个EDW，它是一个集中化的、集成的企业数据仓库，也是决策支持应用的终端用户可以使用的所有数据的控制点和惟一来源；第三层是一系列数据集市。数据集市是为了满足一个特定用户群制定决策的需求而限制在某个范围内的数据形成的数据仓库。数据集市可以是独立于EDW的，也可以是从EDW中导出出来的，或者是EDW的逻辑子集。

EDW的数据层称为调和数据层。理想情况下，该数据层的特征是：详细的、历史的、规范化的、全面的和可以控制质量的。通过用各种运作系统填充EDW或运作数据存储可以获得调和数据。对数据进行调和需要四个步骤：从源系统中捕获数据；清洗数据（消除不一致性）；转换数据（把数据转换为数据仓库所需要的格式）；向数据仓库装载和索引数据。调和数据通常不能由终端用户直接访问。

数据集市的数据层被称为导出数据层。终端用户为了制定决策可以访问这些数据。

数据集市经常使用星型模式（多维模型）存储数据，它是关系模型的变体。星型模式是一种简单的数据库设计方案，在这种模式中，维度数据与事实或事件数据相分离。星型模式包含

两种类型的表：维表和事实表。事实表的大小在一定程度上依赖于表的粒度（即细节级别）。目前，数据仓库应用系统中事实表超过10亿行的情况已经非常普遍了。星型模式有几种变体，包括多事实表模型和雪花模式，后者是在一个或多个维度存在层次结构的情况下提出的。

目前有多种访问和分析决策支持数据的终端用户界面。联机分析处理（OLAP）采用一组图形工具向用户提供数据的多维视图（通常用一个立方体来观察数据）。OLAP使数据分析操作变得很简便，这些数据分析操作包括：切片和切块，数据换轴以及下钻。数据挖掘则是运用复杂的技术进行知识发现的一种形式，这些技术包括传统的统计学技术、人工智能技术和计算机图形技术。

本章复习

关键术语

聚合
一致性维度
数据集市
数据挖掘
数据清洗
数据转换
数据可视化
数据仓库
依赖数据集市
导出数据
企业数据仓库（EDW）
事件
粒度
增量抽取
独立数据集市
信息系统
联结
逻辑数据集市
购物篮分析
多维OLAP
联机分析处理（OLAP）
运作数据存储（ODS）
运作系统
周期数据
调和数据
刷新模式
关系OLAP
选择
雪花模式
星型模式
静态抽取
临时数据
更新模式
主动仓库

复习问题

1. 定义下列术语：

a. 数据仓库
b. 数据集市
c. 调和数据
d. 导出数据
e. 联机分析处理
f. 数据挖掘
g. 星型模式
h. 雪花模式
i. 粒度
j. 静态抽取
k. 增量抽取
l. 事件

2. 将下列术语和定义匹配起来：

_____事件
_____周期数据
_____数据集市
_____星型模式
_____数据挖掘
_____调和数据
_____依赖数据集市
_____数据可视化
_____临时数据

a. 先前的数据内容丢失
b. 详细的历史数据
c. 转换数据格式
d. 校正源数据错误
e. 数据没有被改变或删除
f. 一种数据库动作（如创建）
g. 限制在一定范围内的数据仓库
h. 维和事实表
i. 知识发现的方式

_____雪花模式　　j. 从数据仓库中填充
_____数据转换　　k. 由层次维度产生
_____数据清洗　　l. 以图形方式表示数据

3. 比较下列术语:

a. 静态抽取; 增量抽取　　b. 临时数据; 周期数据
c. 数据仓库; 数据集市; 运作数据存储　　d. 数据清洗; 数据转换
e. 调和数据; 导出数据　　f. 事实表; 维表
g. 星型模式; 雪花模式　　h. 独立数据集市; 依赖数据集市; 逻辑数据集市

4. 指出目前在许多组织中使数据仓库成为必然的五大主要趋势。
5. 简要描述数据仓库体系结构的主要组成部分。
6. 列举三种在三层数据仓库体系结构中出现的元数据，并简要描述每种类型元数据的功能。
7. 列举调和数据的五条典型特征。
8. 列举并简要描述数据调和过程的四个步骤。
9. 列举在运作数据中普遍存在的五种错误和不一致性。
10. 简要描述利用OLAP工具很容易执行的三种类型操作。
11. 解释抽取、转换和装载与数据调和过程的关系。
12. 解释独立数据集市的优缺点。
13. 解释数据仓库变更与运作系统数据库变更之间的区别。
14. 阐述逻辑数据集市的优缺点。
15. 列举数据清洗中的常见任务。
16. 描述在数据仓库或数据集市中应用的替代键的特征。
17. 为什么时间始终是数据仓库或数据集市中的维度之一?
18. 对于相同数据仓库环境下不同的星型模式，使用一致性维度的目的是什么?
19. 事实表可以没有非键属性吗? 为什么?
20. 在什么方式下，维表通常不需要进行规范化?
21. 什么是与维表相关的层次结构?
22. 解释“缓慢变化的维度”的含意?
23. 阐明处理缓慢变化的维度最常用的方法。

问题和练习

1. 研究图11-1所示的三张学生数据表。另外设计一张表，这张表可以保存三张表所有的数据并且没有冗余。选择你认为最适合这些数据的列名称。

2. 下面这张表显示了06/20/2001的一些简单的学生数据:

键	名　字	专　业
001	Amy	Music
002	Tom	Business
003	Sue	Art
004	Joe	Math
005	Ann	Engineering

下列事务发生在06/21/2001:

1) 004号学生把专业从Math变为Business。
2) 从文件中删除005号学生。

3) 新的006号学生被添加到文件当中，他的名字是Jim，专业是Phys Ed。

下列事务发生在06/22/2001:

1) 003号学生把专业从Art变为History。

2) 006号学生把专业从Phys Ed变为Basket Weaving。

你有以下两项主要任务:

a. 为06/21/2001和06/22/2001构建反映上述事务的临时数据表（参见图11-8）。

b. 为06/21/2001和06/22/2001构建反映上述事务的周期数据表（参见图11-9）。

3. Millennium College希望你为他们设计一个记录学生成绩的星型模式。其中有四个维表，属性如下:

- Course_Section。其属性有: Course_ID、Section_Number、Course_Name、Units、Room_ID、Room_Capacity。在某个学期，学校提供平均500个班。
- Professor。其属性有: Prof_ID、Prof_Name、Title、Department_ID、Department_Name。
- Student。其属性有: Student_ID、Student _Name、Major。每个班平均有40个学生。
- Period。其属性有: Semester_ID、Year。数据库将包含30个时间段（总共10年）的数据。

事实表中记录的惟一事实是Course_Grade。

a. 为本题设计一个星型模式，所遵循的格式可以参见图11-14。

b. 使用上述给定的假设估算事实表的行数。

c. 估算事实表的总体大小（以字节为单位），假定每个字段平均长度为5字节。

d. 班级、教授和学生的特征都会随时间发生改变，在你设计星型模式时如何考虑这些改变？为什么？

4. 在掌握了第5章讲述的规范化法则的基础上，读者应该能立即看出上题中为Millennium College设计的星型模式不符合第三范式。请应用规范化法则将星型模式转变为雪花模式。这种转换会影响事实表的大小吗？

5. 请你为Simplified Automobile Insurance Company构建一个星型模式（参见kimball，1996，里面有更为真实的例子）。有关的维度、维度属性和维度大小的规定如下所示:

- Insured Party。其属性包括: Insured_Party_ID、Name。每张保单和保险项目平均有两个投保方。
- Coverage Item。其属性包括: Coverage_Key、Description。每张保单平均有10个保险项目。
- Agent。其属性包括: Agent_ID、Agent_Name。每张保单和保险项目有一个经纪人。
- Policy。其属性包括: Policy_ID、Type。目前公司大概有100万张保单。
- Period。其属性包括: Date_Ke、Fiscal_Period。

对这些维度所记录的事实有: Policy_Premium、Deductible、Number_of_Transactions。

a. 为本题设计一个星型模式，遵循的格式可参见图11-14。

b. 应用上述假定，估计事实表的行数。

c. 估计事实表的大小（以字节为单位），假定每个字段平均占5个字节。

6. Simplified Automobile Insurance Company想在星型模式中添加一个Claims维度。其属性包括: Claim_ID、Claim_Description以及Claim_Type。事实表目前的属性有: Policy_Premium、Deductible、Monthly_Claim_Total。

a. 扩展问题与练习5中的星型模式以包含上述新数据。

b. 估计事实表的行数，假定公司每个月处理平均2000笔赔偿。

7. Millennium学院（参见问题与练习3）现在想向班级里添加几个新数据：提供该课程的系，系所属的学院，以及分配给该系的预算。修改问题与练习3的回答，以包含新的数据。解释为什么采用你所选择的实现方法。

8. 在本章提到过，Kimaball（1997）、Inmon（1997，2000）以及Armstrong（2000）对独立和依赖数据集市以及规范及非规范化的数据集市之间的优劣进行了争论。请从图书馆或网络资源上找到这些论文，并对争论双方的观点作一总结。

应用练习

1. 参观某个已开发数据仓库的组织，与数据管理员或其他主要参与者面谈。讨论以下问题：
   a. 用户对数据仓库是否满意？它在哪些方面改善了用户的决策制定过程？
   b. 数据仓库采用的是两层还是三层体系结构？
   c. 该体系结构是否有一个或多个数据集市？如果有，那么这些数据集市是依赖的还是独立的？
   d. 终端用户采用何种工具？是否采用了数据挖掘工具？
   e. 在开发数据仓库环境时必须克服哪些障碍和困难？

2. 访问如下因特网站点。浏览这些站点以了解关于数据仓库的其他信息，其中包括数据仓库实现的实例、最新的数据仓库产品的描述，有关会议和其他事件的通告。
   a. International Data Warehouse Association: http://www. idwa.com
   b. The Data Warehousing Institute: http://www. dw-institute.com
   c. Knowledge Discovery Mine: http://www.kdnuggets.com
   d. Data Mining Institute: http://www. datamining.org
   e. Data Warehousing Knowledge Center: http://www. datawarehousing.org
   f. 一份数据仓库的电子期刊: http:// www.tdan.com

参考文献

Armstrong, R. 1997. "A Rebuttal to the Dimensional Modeling Manifesto." A white paper produced by NCR Corporation.

Armstrong, R. 2000. "Avoiding Data Mart Traps." *Teradata Review* (Summer): 32-37.

Barquin, R. 1996. "On the First Issue of *The Journal of Data Warehousing*." *The Journal of Data Warehousing* 1 (July): 2-6.

Chisholm, M. 2000. "A New Understanding of Reference Data." *DM Review* (October): 60, 84-85.

Devlin, B. 1997. *Data Warehouse: From Architecture to Implementation*. Reading, MA: Addison Wesley Longman.

Devlin , B., and P. Murphy. 1988. "An Architecture for a Business Information System." *IBM Systems Journal* 27 (No. 1): 60-80.

Dyché, J. 2000. *e-Data: Turning Data into Information with Data Warehousing*. Reading, MA: Addison-Wesley.

English, L. P. 1999. *Improving Data Warehouse and Business Information Quality*. New York: John Wiley.

Griffin, J. 1997. "Fast Payback Projects." *DM Review* 7 (December): 10,11.

Hackathorn, R. 1993. *Enterprise Database Connectivity*. New York: John Wiley & Sons.

IBM. 1993. "Information Warehouse Architecture." SC26-324-00. White Plains, NY IBM

Corp.

Imhoff, C. 1998. "The Operational Data Store: Hamering Away." *DM Review* (July): available from www.dmreview.com.

Imhoff, C. 1999. "The Corporate Information Factory." *DM Review* (December): available from www.dmreview.com.

Inmon, B. 1997. "Iterative Development in the Data Warehouse." *DM Review* 7 (November): 16, 17.

Inmon, W. 1992. *Building the Data Warehouse*. Wellesley, MA: QED Information Sciences.

Inmon, W. 1998. "The Operational Data Store: Designing the Operational Data Store." *DM Review* (July): available from www.dmreview.com.

Inmon, W. 1999. "What Happens When You Have Built the Data Mart First?" in TDAN, an electronic journal found at www.tdan.com.

Inmon, W. 2000. "The Problem with Dimensional Modeling." *DM Review* (May): 68-70.

Inmon, W. H., and R. D. Hackathorn. 1994. *Using the Data Warehouse*. New York: John Wiley & Sons.

Kimball, R. 1996a. *The Data Warehouse Toolkit*. New York: John Wiley & Sons.

Kimball, R. 1996b. "Slowly Changing Dimensions." *DBMS* (April): 18-20.

Kimball, R. 1997. "A Dimensional Modeling Manifesto." *DBMS* (August): available from www.dbmsmag.com.

Kimball, R. 1998a. "Pipelining Your Surrogates." *DBMS* (June): 18-22.

Kimball, R. 1998b. "Help for Hierarchies." *DBMS* (September) 12-16.

Kimball, R. 1999. "When a Slowly Changing Dimension Speeds Up." *Intelligent Enterprise* (August 3): 60-62.

Kimball, R. 2001. "Declaring the Grain." from Kimball University Design Tip #21 at www.ralphkimball.com, verified March 21, 2001.

Martin, J. 1997a. "New Tools for Decision Making." *DM Review* 7 (September): 80.

Martin, J. 1997b. "Web Warehousing." *DM Review* 7 (November): 10.

Meyer, A. 1997. "The Case for Dependent Data Marts." *DM Review* 7 (July/August): 17-24.

Moriarity, T. 1995. "Modeling Data Warehouses." *Database Programming & Design*, 8 (August): 61-65.

Mundy, J. 2001. "Smarter Data Warehouses." *Intelligent Enterprise* (February 16): 24-29.

Poe, V. 1996. *Building a Data Warehouse for Decision Support*. Upper Saddle River, NJ: Prentice-Hall.

Strange, K. 1997. "Can Data Marts Grow?" *CIO* (July): 34, 36.

Weldon, J. L. 1996. "Data Mining and Visualization." *Database Programming & Design* 9 (May): 21-24.

Westerman, P. 2001. *Data Warehousing: Using the Wal-Mart Model*. San Francisco, CA: Morgan Kaufmann.

White, C. 2000. "First Analysis." *Intelligent Enterprise* (June): available from www.intelligententerprise.com.

Williams, J. 1997. "Tools for Traveling Data." *DBMS* 10 (June): 69-76.

Zaitz, J. 1997. "Data Mining Neural Clusters." *DM Review* 7 (July/August): 91, 92.

进一步阅读

Bischoff, J., and T. Alexander. 1997. *Data Warehouse: Practical Advice from the Experts*. Upper Saddle River, NJ: Prentice-Hall.

Goodhue, D., M. Mybo, and L. Kirsch. 1992. "The Impact of Data Integration on the Costs and Benefits of Information Systems." *MIS Quarterly* 16 (September): 293-311.

Jenks, B. 1997. "Tiered Data Warehouse." *DM Review* 7 (October): 54-77.

Web资源

- www.teradatareview.com 这是一份叫做*Teradata Review*的杂志，它包含关于NCR Teradata数据仓库系统的技术和应用的文章。(最近，这份杂志改了名字，新旧名字下的文章都可以在www.teradatamagazine.com找到。)
- www.dmreview.com *DM Review*是一份商业杂志月刊，包含关于数据仓库的文章和栏目。
- www.tdan.com 一份关于数据仓库的电子期刊。
- www.billinmon.com Bill Inmon是数据管理和数据仓库方面的权威。
- www.ralphkimball.com Ralph Kimball是数据仓库方面的权威。
- www.dw-institute.com Data Warehousing Institute，一家致力于数据仓库方法和应用的行业组织。
- www.datawarehousing.org Data Warehousing Knowledge Center，包含与许多数据仓库产品供应商的链接。
- www.olapreport.com 关于OLAP产品和应用系统的详细信息。

项目案例：山景社区医院

项目描述

在很多方面，山景社区医院遵循仔细的规划方法进行了信息系统的设计、选择和安装。该组织已经开发了一个企业数据模型来指导它的数据库开发（参见第3章和第4章）。而且，医院已经安装了计算机系统支持组织的日常运作。例如，医院已经有病人账目系统、管理服务系统和财务管理系统。这些系统大多是经过仔细的考察最终从外部供应商那里购买的。

尽管经过仔细的规划，管理层还是发现现有的医院信息系统存在一些不足和限制。已经引起注意的两个典型问题是：

1) 数据在不同的文件和数据库中经常以不同的格式甚至在不同的媒介上发生重复。例如，由于记账的需要，一组病人数据保存在基于关系数据库的病人账目系统中，但同时，大多数病人的病历又保存在一个人工系统中（每个病人一个文件夹）。

2) 系统主要用来支持运作（事务）处理，但随着现代医院在管理方面需求的增加，该系统不能再提供管理信息或支持分析研究。

通常，医疗行业（包括像山景社区医院这样的医院）是靠获得更好和更多的对其临床、运作和财务信息的集中化访问来推动的（Griffin，1997）。典型的医院数据仓库可能包含四种类型的数据：病人记录；医生、门诊和医院记录；药品及制药公司记录；HMO和保险公司记录（Martin，1997b）。

为了解决上述问题，山景社区医院的管理者想调查一下数据仓库技术是否能成功地应用于他们的组织。因为医院比较小，所以他们认为开发一个大型数据仓库并不合适。不过，他们还是想开发一个小的原型来证明这个想法是否可以实现。在进行了一些调研之后，他们决定开发两个规模较小的原型数据集市：

1) 一个数据集市用来记录医院里医生所执行的检查和步骤的汇总信息。

2) 另外一个比较详细的数据集市记录医生对某个病人所执行的检查和步骤的细节。

项目问题

1) 山景社区医院利用数据仓库或数据集市可以获得哪些好处？

2) 假定医院决定采用星型模式开发一个数据集市。请列举几种你认为在该模式中应该具有的维度。

3) 正如上面所提到的，山景社区医院在没有建好数据仓库的前提下考虑开发数据集市，你认为这种方式合适吗？这样做有哪些风险？你认为一个组织在没有建好数据仓库的前提下，能够通过开发原型数据集市来验证某种想法吗？讨论该方法的优点和缺点。

项目练习

(1) 汇总数据集市

a. 设计一个星型模式：

Physician维度：

Physician_ID (pk): 5字节

Physician_Name: 10字节

Specialty: 5字节

Physician_Address: 10字节

Physician_Telephone: 5字节

Treatment维度：

Treatment_ID (pk): 3字节

Treatment_Description: 6字节

Period维度:

Period_ID (pk): 2字节

Month: 1字节

Year: 2字节

Treatment（事实）表:

粒度：按医生和治疗对每月的治疗与平均治疗成本进行汇总。

Monthly_Total: 3字节。

Average_Cost: 5字节。

b. 估算事实表的行数和大小（以字节为单位）。假定:

1) Treatment：医院大概执行500种不同的治疗。在一个月内，大概执行了30%（150种）的治疗种类。

2) Physicians：每次治疗由一位医生进行。

3) Periods：如果开发一个全面的数据集市，可以考虑聚合36个时间段（3年）的数据。

(2) 详细数据集市

a. 设计一个星型模式。

Physician维度：同上。

Treatment维度：同上。

Period维度:

Period_ID (pk): 3字节

Date: 5字节

Patient维度:

Patient_ID (pk): 5字节

Patient_Name: 10字节

Patient_Address: 10字节

Patient_Telephone: 5字节

Treatment（事实）表:

粒度：一位医生对一位病人所做的每次治疗。

Treatment _Cost: 4字节

Treatment _Result: 20字节。

Doctor_Order: 10字节。

b. 估算事实表的行数和大小（以字节为单位）。假定:

1) Treament。一天中针对病人进行平均200种治疗（每天统计的平均病人是100名，并且每个病人每天平均做两种治疗）。

2) Patients。每次对一位病人进行治疗。为了简单起见，假定在某一天里一种指定的治疗只能由一位医生对一位病人进行。

3) Physicians。每次治疗由一位医生进行。

4) Periods：如果开发一个全面的数据集市，可以考虑收集大概1000天（3年）的数据。

c. 为什么必须假定在某一天里一种治疗只能由一位医生对一位病人进行一次？请提出解决这种限制的方法。

第五部分　数据库的高级主题

概要

第一部分～第四部分已为读者开发高效实用的数据库奠定了基础，第五部分则补充介绍一些重要的数据库设计与管理问题。这些问题包括数据库安全性、备份与恢复、对数据并发访问的控制、数据质量的维护、数据库性能调整的高级主题，以及当代最热门的两个数据库主题——分布式数据库和面向对象数据库。第五部分后面的四个附录包括其他E-R表示方法、高级范式（补充第5章）、数据结构（补充第6章）和对象关系数据模型（补充第5章与第14章）。

读过本书，读者会发现，和人力、物质资源以及财务资源都是企业资产一样，数据也是一种企业资产。因此，数据和信息都是不能随便处理的极有价值的资源。在第12章里，读者将学到数据与数据库管理的关键性信息资源管理活动，还将学到并发性控制、死锁、加密、信息库、加锁、恢复、系统目录、事务和版本设置的意义，还将学到下列两类角色的作用：

- **数据管理员**——总体负责数据、元数据及数据使用策略的人员。
- **数据库管理员**　负责物理数据库设计和处理诸如实施安全性、数据库性能、备份与恢复以及与管理数据库有联系的技术问题的人员。

数据管理与数据库管理活动贯穿于数据库开发的全过程。数据管理员在数据库开发的早期阶段以及数据资源的整体规划中起到重要的作用，而数据库管理员则在物理数据库的设计、实现与操作中起着重要作用。

在一些大型的组织里，数据库可能分布于多台计算机和多个位置。当一个组织机构试图以单个数据库方式而不是以多个分散、单独的数据库方式管理分布数据时，就会出现一些特殊的问题。在第13章里，读者将学到同构或异构分布式数据库、分布式数据库的目标与权衡，以及分布式数据库的几种可用的体系结构。除此之外，读者还将学到数据复制、数据分区以及如何使一个分布式数据库上的同一个数据的多个实例同步等重要概念。还将学到包括分布式事务控制（如提交协议）在内的分布式数据库管理系统的特点。另外，还将介绍分布式数据库管理系统的演变和分布式数据库管理系统产品范围。

第14章引入了另一种E-R建模技术。面向对象数据模型和另一些系统方向越来越受欢迎，因为它们可以通过高度相关的建模表示方法来表达复杂的思想。本章采用统一建模语言（Unified Modeling Language，UML），它是该领域的一种标准。在UML里，所谓对象就是一个实体，它包含有三个属性：状态、行为与标识。一个对象的行为是通过封装在该对象里的一个或多个操作来加以确定的。关联、概化、继承与多态都是重要的概念。本章对第3章的松谷家具公司案例，采用类图形式来描述其面向对象的版本。

第15章阐述将类图转换为模式的方法。它可以利用对象数据库管理系统（ODBMS）来实现。该章还介绍了对象定义语言（ODL）和对象查询语言（OQL）。读者将学到如何在ODL里创建面向对象数据库定义，包括如何定义对象、属性、操作与联系；并将学到与SQL类似的OQL初步知识，包括单表或多表查询。

第12章　数据管理与数据库管理

12.1　学习目标

学完本章后，读者应该具备以下能力：

- 定义下列关键术语：数据管理、数据库管理、数据库安全性、加密、数据库恢复、事务、并行控制、加锁、死锁、版本设置、系统目录和信息库。
- 列举数据管理与数据库管理的一些主要功能。
- 描述在现代业务环境下数据管理员和数据库管理员作用的变化。
- 描述数据词典和信息库的作用以及如何在数据管理中利用它们。
- 比较并发性控制的乐观系统与悲观系统。
- 描述数据库的安全性问题，并列举加强安全性常用的五种技术。
- 描述数据库恢复问题，并列举在DBMS中恢复数据库常用的四种基本工具。
- 描述调整数据库以达到较好性能的问题，并列举在调整数据库时可能发生变更的五个方面。

12.2　引言

现在很多人已经认识到数据对于组织有极为重要的作用。正如人力、物质资源与财务资源都是企业资产一样，数据也是企业的一项资产。因此，数据与信息是极有价值的资源，不能轻率地处理它们。信息技术的开发极大地提高了管理企业数据的效率，人们已经开发出有助于组织达到有效管理数据目标的数据管理活动。有效的数据管理可以为组织中各部门制定管理性决策提供支持。

另一方面，数据管理的低效性将导致数据利用率低，可以用许多组织中最常见的下列状况来刻画低效的数据管理：

1) 同一数据项的多重定义或不同数据库中同一数据元素的不一致表示（使得集成不同数据库时产生较大的风险）。

2) 关键数据元素缺失（这一缺失会使现有数据失去价值）。

3) 使用不合适的数据源或者从一个系统向另一个系统传输数据的时机不当会引起数据质量水平低下（这会降低数据的可靠性）。

4) 对现有数据缺乏了解，包括不熟悉数据所在位置和所存储数据的意义（这会降低利用数据作出有效战略、规划决策的能力）。

5) 差的及不可容忍的查询响应时间、频繁的数据库停机、没有正确地控制数据的保密性与安全性。

许多组织已经采取各种策略来处理这些数据利用问题。有些公司利用一种称为**数据管理**（data administration）的功能。负责实现这一功能的人被称为数据管理员或者信息资源经理，他们负责数据资源的总体管理。第二种功能即**数据库管理**（database administration），主要负责物理数据库设计和处理诸如实施安全性、数据库性能、备份与恢复以及与数据库管理有关的各种技术问题。有些组织则结合使用数据管理与数据库管理功能。随着商业步伐的加快，数据

管理员（DA）与数据库管理员（DBA）的作用也在发生改变，这就是我们要在下一节讨论的内容。

12.3 数据管理员与数据库管理员的作用

有两个因素驱使数据管理与数据库管理的作用不断发生变化：一是许多组织已经可以利用更多的技术与平台，但它们必须并发地进行管理；二是业务变革的速度不断加快(见Quinlan, 1996)。面对这些变化，理解数据管理和数据库管理的作用至关重要。这将有助于理解具有不同信息技术体系结构的各个组织，是通过怎样的途径来综合利用这些角色的。

12.3.1 传统的数据管理

数据库是属于整个企业的共享资源，它们并非是组织内某个人或者某个部门的私有财产。正如财务总监是财务资源的监管人一样，数据管理就是组织中数据的监管人。与财务总监相仿，数据管理员必须开发出各种过程来检测与控制数据资源。此外，数据管理必须解决数据由用户集中使用或者共享时可能会出现的各种问题；还必须决定数据存储与管理的方案。数据管理是负责组织里数据资源的总体管理的一种高级功能，其中包括维护全公司的数据定义与标准。

数据管理的职责范围相当大，包括数据库的规划、分析、设计、实现、维护与保护。数据管理还负责建立改善数据库性能的过程，并向用户提供教育、培训及咨询服务的支持。数据管理员必须要同高级管理层、用户以及计算机应用专家打交道。

选择数据管理员并组织好其管理职能极其重要。一个组织在进行较大的变革时通常会遇到各种问题，数据管理员必须是能够解决这些问题的经验丰富的管理人员。数据管理员应该是从组织内选拔出来的有一定威望的高级管理人员，不提倡聘用只懂技术的计算机专家或者新手来担任此职。

数据管理部门的经理既要有丰富的管理经验，又要有相当的技术能力。一方面，他必须能够协调拒绝将其私有数据提供出来作为共享数据的用户；并必须使这些用户确信，对于访问数据库来说，遵循标准的定义与过程会带来极大的好处。另一方面，数据管理员又必须能够管理那些负责处理查询优化和并发控制等问题的技术人员。

12.3.2 传统的数据库管理

通常情况下，数据库管理角色往往更为直接地在物理上接触一个或多个数据库。数据库管理是一种技术性职能，它负责物理数据库的设计，并处理诸如实施安全性、数据库性能以及备份与恢复等技术问题。数据库管理员应当参与到数据库开发的每一个阶段里，从数据库规划到分析、设计与实现阶段，并完成具体的操作、维护和修改。DBA必须贯彻执行由数据管理员所制定的标准和步骤，包括强制性编程标准、数据标准、策略及步骤。有些组织里，数据库管理这一职能并没有完全独立出来，DBA还必须承担数据管理员的职责。

与数据管理员一样，DBA也要有丰富的工作经验。最主要的是要有扎实的技术背景，包括当前流行的硬件体系结构与功能的渊博知识，以及对数据处理有深入的理解。理解包括传统方法和原型方法在内的数据库开发生命周期，也是很有必要的。在概念、逻辑和物理层次上，都需要有较强的设计和数据建模能力。然而，管理性技能也是很重要的，因为DBA在分析、设计和实现数据库时必须管理其他信息系统人员，而且DBA还必须面对许多最终用户，并向他们提供有关数据库的设计和使用方面的服务支持。

12.3.3 数据管理与数据库管理方法的演化

目前还没有所有人普遍接受的数据管理和数据库管理结构，而且各个组织的数据管理方法也不尽相同。随着业务实践改变，其角色作用在组织内也会发生改变。然而，不管组织结构如

何，总会存在一组核心的数据管理与数据库管理功能，它们在任何组织内都会存在。这些功能很可能由各个数据管理员和数据库管理员来负责，或者在极端情况下，所有这些功能可能仅是由某一个人来处理。其中一些功能是组织层次的功能，人们倾向于把它们归类为组织的数据管理功能，以示数据管理和数据库管理之间有所区别。另一些功能对于数据库或者应用软件来说更为专门些，因此把它们归类于数据库管理功能。不过，要进行有效的数据与数据库管理，必须要处理以下一些功能：

- **数据的策略、过程与标准**　每个数据库应用软件都需要通过实施一致的数据策略、过程和标准来建立保护措施。数据策略是明确数据管理目标的语句，如“每个用户必须有一个口令”。数据过程就是编写为完成特定的活动所采取行为的概述。例如，备份与恢复过程应该与所有相关员工进行通信。数据标准是一些必须遵循的明确的约定和行为，它们可以用于帮助评估数据库质量。例如，对于程序员来说，数据库对象的命名规则应当是标准化的。同时，外部数据源使用的增加和来自组织外部对组织数据库访问的增加，也促使员工认识到理解数据策略、过程和标准的重要性。
- **规划**　涉及开发组织的信息体系结构的一项关键性管理功能。有效的管理既需要理解组织对信息的要求，又要有开发符合组织的众多要求的信息体系结构的能力。完成数据和数据库管理的关键是，理解该组织的信息需求，并使信息体系结构的开发与组织需求相吻合。
- **数据冲突的化解**　数据库应该是共享的，而且通常都是涉及该组织多个不同的部门。在任何组织里，确定数据的所有权往往是一个棘手的问题（至少有时候是这样）。在数据和数据库管理中，要善于化解数据所有权问题，因为它们并非只同某个部门相关联。化解这类冲突时，最重要就是建立相应的过程。只要在化解冲突时，管理功能能得到充分的授权来进行调解和强化，那么数据和数据库管理在处理这类问题时是非常有效的。
- **内部营销**　尽管组织已经充分意识到数据和信息对于它们的重要性，然而却未必会意识到与其有关的数据标准的重要性。以下确立的过程和政策的重要性，必须预先通过数据和数据库管理员告知整个机构。有效的内部营销可以减少变革的阻力和数据所有权问题。
- **信息库的管理**　信息库包含描述组织数据和数据处理源的元数据。在很多组织里，信息库用来替代数据词典。然而，数据词典只是简单的数据元素文档工具，可是信息库却能被数据管理员和其他的信息专家用来管理整个信息处理环境。信息库是数据和数据库管理的一种基本工具，并在整个数据库系统生命周期中应用。信息库可以用作为下述对象的信息源：

1) 用户（必须理解数据定义、业务规则和数据对象之间的联系）。
2) 自动化CASE工具（用来规定与开发信息系统）。
3) 应用软件（访问或者操纵企业数据库中的数据或业务信息）。
4) 数据库管理系统（维护该信息库并更新系统权限、口令、对象定义等）。

- **硬件与软件的选择**　硬件与软件的评估与选择对于一个组织的成功至关重要。新型软硬件开发速度的加快和组织内的快速变革，使选择的标准日渐细化。在选择平台时，某个供货商的数据库似乎是最合适的产品；但一旦其竞争对手发行了一种不同的但具有更加合适的功能的新版本，那么原先的选择就显得极不适用。现行的供货商甚至于可能已经脱离了业务实际，导致生产数据库中的维护工作简直就像是一场噩梦。人们越来越盼望数据管理员和数据库管理员能够了解更多的硬件体系结构，并且既能管理自行开发出来的应用软件，也能管理商业现用的应用软件，因为组织总是力求更加快速地满足自己的信息要求。
- **DBMS的安装与升级**　一旦选定了DBMS，就必须进行安装。在安装之前，应当在DBMS

供货商所提供的计算机上运行其数据库工作负载的特征指标。在实际安装时必须考虑到特征指标的期望问题（在选择软硬件时也可以用特征指标来比较竞争对手的产品）。DBMS安装是一个很复杂的过程，安装时要确保各个模块的正确的版本都正确安装，所有的设备驱动程序已正确安装，以及DBMS能够与任何第三方软件产品协同工作。DBMS供货商会定期地更新其软件包模块；制定测试计划并安装升级，以确保现有的应用软件仍然能正确地工作，这项工作可能很花费时间并且很复杂。一旦安装好DBMS，就必须要创建并维护好用户的账户。

- **数据库性能调整** 由于数据库是动态的，因此，希望数据库的最初设计就能确保它在整个生命期内都能达到最佳处理性能是不现实的。应该对数据库的性能（查询与更新的处理时间以及数据存储利用率）持续进行监控。数据库的设计必须不断变更，以迎合新的需求并克服多次内容更新所带来的退化效应。数据库应当定期地加以重建、重组织和重索引，以便恢复浪费的空间、校正不良的数据分配和存储设备的分段、为并发处理分配不同的缓冲空间以及重新设定参数来定义最佳的恢复过程。DBMS所需要的不包含业务数据的各种系统表，也需要定期地加以重建，以便能与数据库的新的使用和规模相一致。
- **数据库查询处理性能的改善** 一个数据库的工作负载绝大多数是会随时间不断增加的，因为越来越多的用户找到了更多的办法在数据库中使用不断增长的数据。因此，有些查询原先在一个小型数据库上运行得很快，但在某个有许多用户使用的数据库上，要想获得满意的运行时间，就必须将其改写成一种更有效的形式。可能需要追加或删除索引，以便平衡所有查询的性能。也可能要将数据重新定位到其他设备上，以便更快地执行查询与更新的并发处理。有争议的是，一个DBA是否需要花费大量的时间来调整数据库的性能和改善数据库查询的处理时间。
- **数据安全性、私密性与完整性的管理** 数据库管理还要负责组织数据库的安全性、私密性与完整性。关于确保私密性、安全性和完整性的方法的内容将在本章后面部分介绍。这里重要的是理解由于因特网和企业内部网的出现，再加上将数据和数据库分布到多个站点的可能性，使得数据安全性、私密性和完整性的管理变得更加复杂。
- **数据备份与恢复** DBA必须确保已经确立备份过程，以便可以恢复所有必要的数据，不至于在出现应用软件故障、硬件故障、物理或电气灾难、人为错误或者渎职犯罪时丢失数据。本章后面也将会讨论常用的备份和恢复的策略。这些策略必须以一定的时间间隔进行充分的测试和评估。

了解这些数据和数据库管理功能，可以使读者了解在组织层次和项目层次进行数据管理的重要性。不采取这些真正的步骤，就可能会大大降低组织的有效运作的能力，甚至于可能造成业务失败。对于减少应用软件开发时间的要求始终必须经过审查，以便在时间急迫时仍然不至于忘记确保必要的质量；对于这类捷径，最好要有严格的复审。图12-1对数据管理和数据库管理功能按照典型的系统开发阶段进行介绍。

12.3.4 数据管理方法的演化

许多组织现在都把数据管理和数据库管理的角色作用结合在一起。这些组织关注的是快速建立数据库的能力，调整数据库使其达到最大性能以及能够在问题出现时迅速解决它的能力。这些数据库很可能就是部门性的客户/服务器数据库，它们可以采用原型法之类较新的开发方法快速开发出来，这些方法可以作出某些快速的改变。将数据管理和数据库管理作用结合起来也意味着，这些组织中的DBA必须能够创建和实施数据标准与策略。在这种环境下，要求DBA迅速交付高质量、强壮的系统。在组织中，一方面要求DBA维护数据库使之保持所需的质量水平，

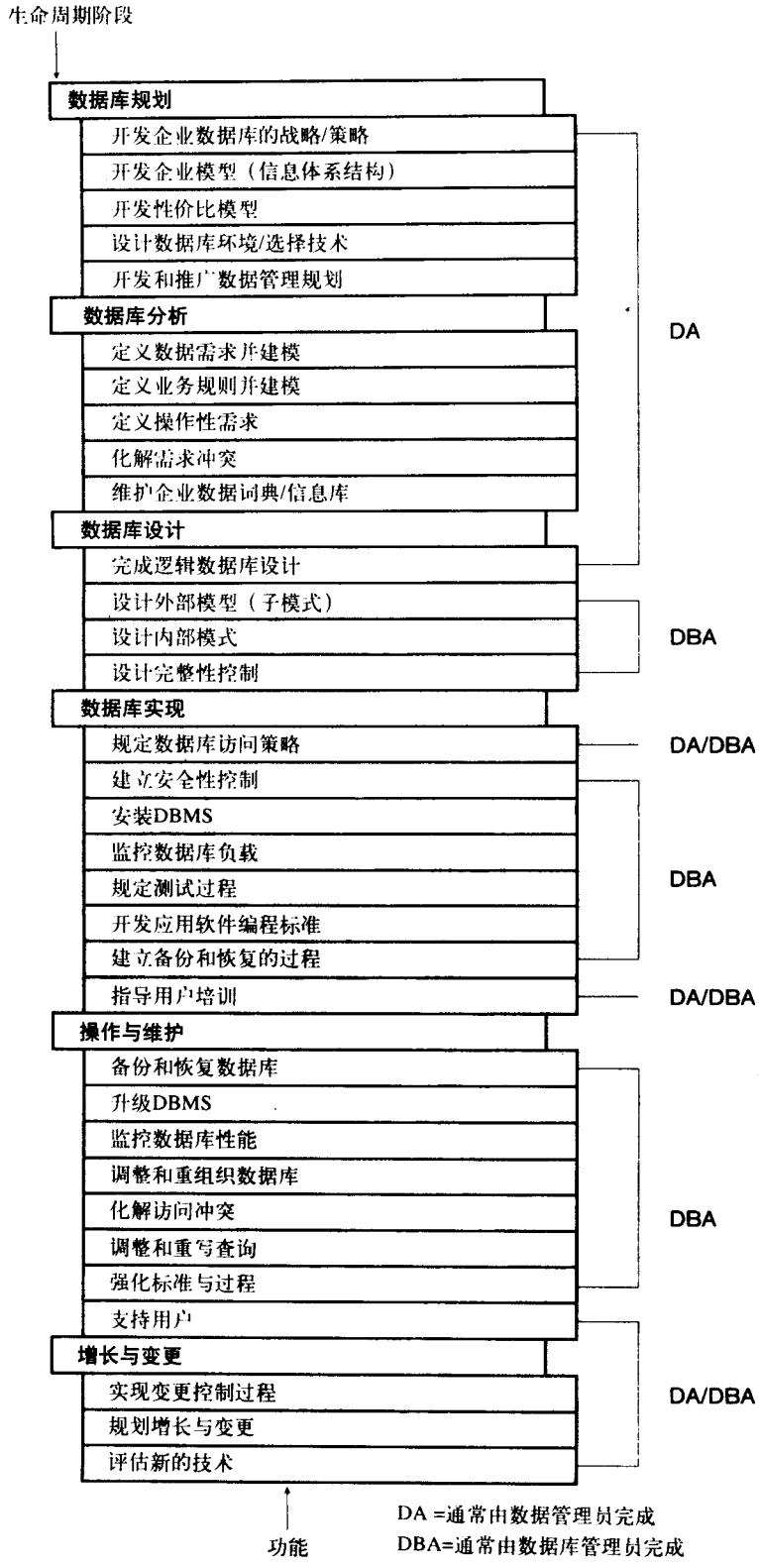


图12-1　数据管理和数据库管理的功能

另一方面要以更短的时间构建可靠的系统。Quinlan(1996)建议对数据管理和数据库管理实践作某些改变，这些改变可以在传统数据库开发生命周期的每一个阶段完成，以实现上述目标:

- **数据库规划** 通过有选择地评估可能产品的方式来改进对技术的选择。要确保考虑了与企业数据模型相符合的每种技术，在数据库规划阶段仔细选择所用技术可以搜索寻找出减少以后阶段所需要时间的途径。
- **数据库分析** 尝试让物理设计与逻辑、物理模型的开发并行完成。现在，利用应用软件原型方法可以顺利地在开发过程中对原先的逻辑、物理数据模型作变更。
- **数据库设计** 可以按照量、重要性和复杂性对应用软件事务的优先级排序。对于应用软件来说，这些事务变得越来越关键。在开发这些事务的时候，应当迅速地审查一下它们的规格说明书。逻辑数据建模、物理数据建模和原型化可以并行地进行。DBA应该力求为数据库环境提供最合适的控制，同时允许开发者有实验的空间和机会。
- **数据库实现** 建立数据库变更控制过程，以支持而不是拖延开发和实现。只要有可能，就将模型分割为模块，这样可以更快地进行分析和实现。寻找在不损害质量的前提下能更快地测试系统的办法。测试可以在开发的早期阶段进行；可以利用测试和变更控制工具来建立和管理测试与生产环境。
- **操作与维护** 应当审查所有节省时间的手段，以确保数据库质量不受损害。要尽可能考虑采用第三方工具和实用程序，以节省工作量；其他的工具（如Lotus Notes）可以减少会议时间，从而也就节省了工作时间。

DBA的作用还将会继续演化。DBA可能会变得更加专业化，成为各个领域的专家，如分布式数据库/网络容量规划DBA、服务器编程DBA、商业现用软件定制DBA或数据仓库DBA等(Dowgiallo等，1997)。利用多个数据库、通信协议及操作系统的能力具有很高的价值。那些拥有丰富的开发经验，并能够快速适应不断改变环境的DBA们将会有较多的机会。现在的一些DBA活动（如调整）将被通过分析用户的使用模式来调整系统的决策支持系统所取代。在大型企业中继续使用超大规模数据库（VLDB）的机会以及在小型、中型企业中管理台式或中型服务器的机会依然很多。

数据仓库管理

最近五年里，数据仓库迅速增长（见第11章），于是一个新的角色——数据仓库管理员(DWA)也应运而生。Inmon(1999)在其几篇论文中（在其站点上可以找到）概述过DWA的职责。正如所预期的，关于DWA角色有两个普遍特征:

1) DWA在数据仓库中起到的许多作用与DA和DBA相同，数据集市数据库用于支持决策应用软件（而非典型的DA与DBA处理的是事务处理应用软件）。

2) DWA主要关注来自许多数据源的元数据和数据(抽取协议、操作性数据存储、企业数据仓库)的集成和协调，这些分散管理的数据源上的数据未必是标准化的，而且可能超出了DWA的控制范围。

Inmon还提议，DWA应该拥有一些专门的特权:

- 建立与管理一个能支持决策制定应用软件的环境。因而，DWA特别关心制定决策所花费的时间，而不关心查询响应的时间。
- 为数据仓库（即企业信息工厂）建立一个稳定的体系结构。因而，DWA更加关心数据仓库增长（指数据量和用户数的扩展）的影响，而不关心重新设计现有的应用软件的影响。
- 同数据仓库的数据供应者和消费者签订服务层协议。因而，DWA的工作与最终用户和运作系统管理员的密切程度要胜过DA和DBA，以便协调极为不同的目标，并监督新的应用

软件的开发（数据集市、ETL过程和分析服务）。

这些职责是数据管理员或数据库管理员的常规职责之外的职责，数据管理员和数据库管理员的常规职责包括选择技术、与用户就数据需求进行交流、制定性能与容量的决策以及对数据仓库需求编制预算和规划。

Inmon估计，EDW每100GB数据就需要有一位DWA。还可以按照EDW中一位DWA每年需要维护的数据量进行度量。用于抽取/变换/负载的定制工具，通常会增加所需要的DWA的数目。

数据仓库管理员通常是归组织的IT部门管理的，但是他同营销部门及其他的业务部门也有较密切的关系，这些部门依靠企业数据仓库的应用软件，比如客户或供货商关系管理、销售分析、渠道管理以及其他分析性应用软件。DWA不应当像DBA那样是传统的系统开发组织的一部分，因为数据仓库应用软件的开发与运作系统不一样，应该将数据仓库应用看成是与任何特定的运作系统相独立的。另一种做法是将DWA安排在EDW的最基本的最终用户组织中，不过，这可能会导致创建过多的数据仓库或数据集市，而不是创建一个真正的可伸缩的EDW。

12.4　企业数据的建模

在现代业务环境下，常常要求DBA迅速提供高质量的系统来适应变化。正如第2章所介绍的，实现这一目标的关键在于开发一个企业体系结构或者信息系统体系结构(ISA)。ISA的作者（见Zachman，1997）认为信息系统的开发与其他复杂产品的开发和制造过程相似。他仔细研究了这些过程，以便把它们的原理和过程应用到信息系统企业体系结构的开发过程之中。建立企业的体系结构可以使组织更有效地完成从业务策略到实现的步骤。Zachman（1997，p.48）把企业体系结构定义为："与描述某个企业相关的描述性表示（即模型）的集合，这使得它（指企业）能够符合管理者的要求（质量），并能够在其有效的生命周期（变化）中进行维护"。

在系统开发中经常会看到，由于开发者关注建立企业系统的片段而忽视了企业视图。当把这些片段集成到一个正常运行的企业系统时，往往会出现问题。有些系统在完成时就不能正常工作，或者后来不能正常工作。系统维护的成本在一个新系统总体成本中占据很大一部分。像千年虫（据Garnter估计，它大约在全球造成了400亿美元的损失）这样的问题就是由于过去没有很好地处理企业体系结构内的数据元素定义造成的。

理想情况下，任何数据库开发项目都应当在其企业体系结构里完成。这就意味着，首先开发与业务要求相一致的概念数据模型，然后将它转换成逻辑数据模型、物理数据模型以及被实现的系统。这种一致性得到极大的重视，并成为任何数据库设计生命周期的一部分。但是，概念数据模型也必须要与Zachman框架相吻合，后者是由业务过程模型、业务网络逻辑、企业内的工作流模型、企业总日程计划以及企业业务计划等组成的。其中的每一个部分（例如概念数据模型）都有其固有的体系结构，但它也必须与该框架的其他部分的体系结构相协调。必须考虑到每个部件的执行给整个企业所带来的影响，并要解决各种问题和不连续性。

凡是建立企业体系结构并开发标准及过程以确保系统开发项目符合该企业体系结构的组织，都应该能体会到这样做的好处。代码可以重用，系统是有机地组合在一起而不是彼此冲突的，业务目标也是匹配的。当业务规则的匹配和协作变得更加复杂的时候，这些建立企业体系结构的组织应当能够比没有开发企业体系结构的组织，更快和更一致地实现那些业务规则。在今天竞争激烈的环境下，这可以使组织获得相当大的战略上的收益。

12.5　数据库的规划

数据库开发生命周期的初始阶段是规划。在这段时间里，应当从企业体系结构的角度仔细

考虑所提议的系统。数据库设计应该在企业信息系统的约束下进行。在规划阶段，对所提议的数据库系统作出初步的评估。该提议系统应当与作为企业体系结构的一部分的业务目标和战略相吻合。企业体系结构中Scope层的其他部分也是很重要的：

- 会包含哪些数据？预先标识它们对于业务是否很重要？
- 系统将完成哪些处理过程？是否需要把这些过程标识为重要的组织过程？
- 系统会影响哪些业务位置？所标识的位置是否应当受到这类系统的影响，或者是否标识出某些无意间受到影响的位置？
- 该系统会影响哪些部门与个人，或者哪些部门及个人会应用该系统？这些结果的特点是什么？哪些影响是部分在请求新系统时没有预料到的？
- 该系统是如何处理重大业务事件的？新系统能否适应现有的时间安排？
- 该数据库会保存多少数据？为了支持应用软件，对硬件的大小与容量有何要求？候选的DBMS在数据库安装后增长时，是否很容易进行扩展（例如，在不对数据库进行规模较大的重设计或变更硬件的情况下，DBMS能否支持更多的用户、更多的应用软件、更大型文件等）？

初始评估也包括考虑继续使用现有系统、修改现有系统或者替换现有系统。留待进一步解决的决策还有，现有系统存在哪些缺陷以及改正这些缺陷的可能性。

倘若有可能建立新系统，那就应该进行可行性研究。可行性研究应当解决该项目在技术方面的问题，包括硬件和软件的选购或开发成本、运行环境、系统的大小与复杂性以及与相似系统的体验程度。此刻不必考虑特定的供应商或者产品，但应该对实现的类别（如PC还是主机、数据库模型和编程的语言）进行考虑。可行性研究也应当对该项目的成本进行估算，包括有形的与无形的。成本应当区分出与项目相关的一次性成本和贯穿系统整个生命期的重复成本。还应当确定运作可行性，即评估该项目实现其期望目标的可能程度。还应当评估在预期时间之内完成项目的可能程度。此刻，还应当确定出所有可能的法律与合同类的问题。应当考虑到该项目的政治性后果。注意，支持该项目的主要投资人会极大地美化其成功的前景。

一旦收集好可行性信息，就可以举行包含所有相关方在内的项目的正式评审。这次评审的主要目标是，检验早在项目开始之前就准备好的基线计划里的所有信息和假设。在论证项目之后，规划阶段所确定的问题将在项目的分析阶段里再作进一步的研究。

12.6 数据安全性的管理

数据库安全性（database security）的目的是保护数据，使其免遭可能会破坏其完整性的有意或无意的威胁及访问。数据库环境已经变得相当复杂，具有位于客户/服务器体系结构上的分布式数据库，而非主机。通过因特网和内部网，访问变得更加开放。其结果是，有效地管理数据安全性已变得更加困难，更加耗费时间。我们在第9章已经介绍过一些针对客户/服务器和基于Web系统的安全性过程。数据管理负责开发保护数据的总体策略和过程。以下将讨论在确立充分的数据安全性时数据管理员应当采用的一些工具，不过重要的是首先了解一下数据安全性的潜在威胁。

12.6.1 数据安全性的威胁

对数据安全性的威胁可以是对数据库的直接威胁。例如，一些未授权用户非法访问某个数据库，然后就可能会浏览、改变或者偷窃他们已经非法访问的数据。然而，只注重数据库的安全性，并不能确保获得安全的数据库。必须要确保系统的所有部分都是安全的，包括数据库、网络、操作系统、安装该数据库的建筑物以及任何有机会访问该系统的人士。图12-2描绘

了可能威胁数据安全性的众多位置。要实现这一层次上的安全性，需要仔细地观察，确立安全性过程与策略，以及实施和强制这些过程与策略。每一份全面的数据安全性计划都必须解决下面提到的问题。

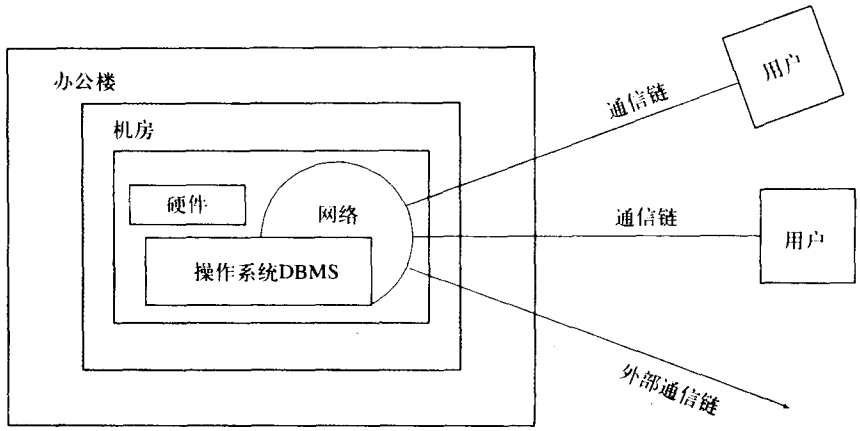


图12-2　威胁数据安全性的可能位置

- **偶然的损失，包括人为的错误、软件与硬件所引起的漏洞**　建立诸如用户授权、统一的软件安装过程和硬件维护日程安排等操作过程，这些都是可以用来避免偶然损失威胁的操作。不管怎么努力，由于这涉及到人类的本性，有些损失是不可避免的，但倘若能采取一些合理的策略和过程，应该是可以减少损失的总量与严重程度。有可能造成更严重后果的则是非偶然性的威胁。
- **偷窃和欺诈**　这些活动是由人来实施的犯罪活动，很可能利用电子手段，也有可能改动或者不改动数据。此时，应当关注图12-2中的每一个可能位置上。例如，应当确立起物理安全性的控制，使得未得到授权的人没有机会进入机房。在运行工资单这样的机密文件之前，要启用限制改变数据的数据访问策略，这样有助于保护数据安全。建立防火墙，以防止非授权用户通过外部通信链接访问数据库中不允许访问的部分是安全性过程的另一个例子，这会阻止企图实施偷窃或欺诈的人士。
- **私密性或机密性的损失**　私密性的损失通常是指个人数据的损失，而机密性损失通常则是指关键性组织数据（它们对于组织往往可能具有战略价值）的损失。私密性信息的泄露会导致黑色邮件、行贿受贿、公众丑闻或用户口令的丢失。机密性信息的丢失可能会导致企业丧失竞争力。现在，（美国）的某些州和联邦已经出台相应的法律，要求某类组织创建与交流确保顾客和客户私密性数据的策略。安全性机制必须强制执行这些策略，否则会在资金和声誉方面造成巨大的损失。
- **数据完整性的损失**　在数据完整性遭到破坏时，数据就会变得无效或者错误百出。除非数据完整性能够通过建立备份与恢复过程加以复原，否则，组织将会蒙受严重的损失，或者根据无效的数据作出不正确与费用昂贵的决策。
- **可用性的损失**　对于用户来说，硬件、网络或应用软件若遭到破坏，有可能造成数据无法使用，这又将会再次导致严重的运作困难。

一份全面的数据安全性计划还包括建立管理性策略与过程、物理保护以及数据管理软件保护。数据管理软件的最重要的安全性特性如下：

1）视图或子模式，它们将限制用户对数据库的查看。

2) 域、断言、检查以及定义成数据库对象的其他完整性控制，由DBMS在数据库查询与更新时执行。

3) 授权规则，标识用户并限制用户可以对数据库采取的操作。

4) 用户自定义过程，在使用数据库时定义补充的约束与限制。

5) 加密过程，把数据编码成不可识别的形式。

6) 认证方案，明确地标识出试图访问数据库的人。

7) 备份、日志记录和检测点功能，这些功能有助于完成恢复。

12.6.2 视图

在第7章里，我们把视图定义为展示给一个或多个用户的数据库的子集。创建视图首先要查询一个或多个基表，然后在用户发出请求时产生一个动态的结果表。因此，视图总是建立在其赖以构建的基表的当前数据上的。视图的优势在于，它能够仅仅显示用户需要访问的数据，有效地防止了用户查看其他一些可能具有私密性或机密的数据。可以授予用户访问视图的权限，但不允许该用户访问此视图所赖以建立的基表。从而，相对于允许访问基表的用户来说，某些用户只能局限于视图的规定，可能是更严格的限制。

例如，我们可以为松谷家具公司的员工构建一个视图，该视图提供制造一个松谷家具公司产品所需要材料的信息，但不提供其他(诸如单价之类)与员工工作无关的信息。创建列举所需要的木材和每个产品可使用的木材的视图的命令如下所示:

```
CREATE VIEW MATERIALS_V
    AS
    SELECT PRODUCT_T.PRODUCT_ID, PRODUCT_NAME, FOOTAGE,
    FOOTAGE_ON_HAND
       FROM PRODUCT_T, RAW_MATERIALS_T, USES_T
            WHERE PRODUCT_T.PRODUCT_ID=USES_T.PRODUCT_ID
            AND RAW_MATERIALS_T.MATERIAL_ID=
            USES_T.MATERIAL_ID;
```

所创建视图的内容在每次访问该视图时，都将进行更新，下面是该视图的当前内容（可以使用SQL命令来访问)。

```
SELECT*FROM MATERIALS_V;
```

PRODUCT_ID	PRODUCT_NAME	FOOTAGE	FOOTAGE_ON_HAND
1	End Table	4	1
2	Coffee Table	6	11
3	Computer Desk	15	11
4	Entertainment Center	20	84
5	Writer's Desk	13	68
6	8-Drawer Desk	16	66
7	Dining Table	16	11
8	Computer Desk	15	9

```
8 rows selected.
```

用户可以针对视图编写SELECT语句，即可以将视图视为一张表。尽管视图通过限制用户对数据的访问，提高了安全性，但它们并不是最充分的安全措施，因为未授权人士可能会了解或者访问特定的视图。其次，几个人可以共享某个特定的视图，其中每个人都有权读取数据，但只有其中的几个人有权更新其数据。最后，利用高级查询语言，未授权人士通过简单的试验就可能访问数据。因此，我们还需要更为复杂的安全措施。

12.6.3 完整性控制

完整性控制保护数据免遭未经授权的使用与更新。通常，完整性控制限制某个字段所能保存的值，限制在数据上所能完成的操作，或者触发执行某个过程，如在某个日志上设置一项，以便记载哪个用户对哪些数据做了哪些操作。

完整性控制的形式之一便是域。实际上，域是创建用户自定义数据类型的一种途径。一旦定义了某个域，某个字段就能够将该域作为其数据类型。例如，下列用SQL定义的PriceChange域就可以作为任何数据库字段（如PriceIncrease和PriceDiscount）的数据类型，以限制在某个事务中可能增加的总量标准价格：

```
CREATE DOMAIN PriceChange AS DECIMAL
 CHECK(VALUE BETWEEN .001 and .15);
```

然后，比如说，在某个制定价格的事务表定义里，我们就可以定义：

```
PriceIncrease PriceChange NOT NULL,
```

域的好处之一是，倘若它改变，那么可以只在一个地方，即域定义的位置作出改变，处于该域的所有字段便会自动地改变。换一种做法，也可以在PriceIncrease字段和PriceDiscount字段上的约束中包括相同的CHECK子句。但是，此时如果CHECK的限制需要改变，那么DBA就必须找出这个完整性控制的每一个实例，并分别在每一处进行修改。

断言是强制实施一定的数据库条件的强有力的约束手段。断言是由DBMS在运行存在此断言的表或字段时自动地进行检查的。例如，假设某个员工表包含字段EmpID、EmpName、SupervisorID和SpouseID。假如公司的一条制度是，任何员工都不可以管理自己的配偶。下列断言会强制实施这一规则：

```
CREATE ASSERTION SpousalSupervision
     CHECK(SupervisorID<>SpouseID);
```

如果该断言不满足，DBMS将会产生一个错误消息。

断言可能会变得相当复杂。假设松谷家具公司有一条制度是，两个推销员不得同时被分配相同的销售区域。假如Salesperson表包含字段SalespersonID和TerritoryID。这一断言可以采用一个相关的子查询写为：

```
CREATE ASSERTION TerritoryAssignment
     CHECK(NOT EXISTS
         (SELECT*FROM Salesperson SP WHERE SP.TerritoryID IN
              (SELECT SPP.TerritoryID FROM Salesperson SSP WHERE
                SSP.SalespersonID<>SP.SalespersonID)));
```

最后，触发器也可以用于保证安全性。在第8章里定义并说明过触发器。触发器中可以包含事件、条件和动作。它要比一个断言更为复杂。例如，一个触发器可以完成以下任务：

- 禁止不适宜的动作（例如，在正常工作日之外改变某个工资值）。
- 引起特殊的处理过程的执行（例如，倘若在规定日期之后才收到顾客的支付票据，那么可以对该顾客的账户追加罚金）。
- 导致在某个日志文件上写入一行，以反映有关用户和涉及敏感数据的事务的重要信息，这样，该日志可以通过人工或者自动化的过程来审阅，来确定是否可能存在不适宜行为（例如，日志可以记载某个用户对某个员工进行了工资修改）。

和采用域一样，触发器的优势在于，可以像任何存储过程那样，使得DBMS强制实施对所有用户和所有数据库活动实行这种控制。该控制不必编码到每一个查询或程序中。因此，对于

个人用户和程序，就无须施加必要的控制。

12.6.4 授权规则

授权规则（authorization rule）就是指包含在数据库管理系统中，限制对数据的访问，并限制人们在访问数据时所能采取的操作的各种控制。例如，提供某个特定口令的人可能被授权读取数据库里的任何记录，但他不一定可以对那些记录作任何修改。

Fernandez、Summers和Wood(1981)曾经开发过一个数据库安全性的概念模型。该模型以包含主体、对象、动作和约束的一个表（或矩阵）形式来表示授权规则。该表的每一行表示被授权主体可以（或许受到某些约束）对该数据库的某个对象实施某种动作。图12-3显示了这种授权矩阵的一个例子。这张表包含几个项，这些项涉及会计数据库中的记录。例如，表的第1行表示，销售部的任何人都有权在数据库里插入一个新的顾客记录（只要该顾客的信用限额不超过$5000即可。最后一行表示，程序AR4有权修改订单记录而无任何约束。数据管理负责确定和实现在数据库级别上实现的授权规则。授权方案也可以在操作系统级别或者应用软件级别上实现。

主 体	对 象	操作	约 束
销售部	顾客记录	插入	信用限制低于$5 000
订单传送	顾客记录	读取	无
终端12	顾客记录	修改	仅限于账户余额
会计部	订单记录	删除	无
Ann Walker	订单记录	插入	订单总额大于$2 000
程序AR4	订单记录	修改	无

图12-3 授权矩阵

大多数现代数据库管理系统并不像图12-3那样实现授权矩阵，它们通常采用简化的版本。主要存在两种类型：用于主体的授权表和对于对象的授权表。图12-4给出了每种类型的一个例子。例如，在图12-4a里我们看到，推销员可以修改顾客记录，但不能删除这些记录。在图12-4b里，订单录入或者会计用户可以修改订单记录，但推销人员则不可以修改订单记录。某个DBMS产品可能提供其中任一种工具，也可能两种工具同时兼有。

图12-4所示的授权表是一个组织的数据及其环境的属性，因此，应当将它们看成是元数据。该表应该在信息库中存储并维护。由于授权表包含高度敏感的数据，所以它们本身也应当用严格的安全性规则进行保护。通常，在数据管理中只有经过严格选拔出的人才有权访问和修改这些表。

	顾客记录	订单记录
读取	Y	Y
插入	Y	Y
修改	Y	N
删除	N	N

a) 对于主体(推销员)的授权规则

	销售人员 (口令BATMAN)	订单录入 (口令JOKER)	会计 (口令TRACY)
读取	Y	Y	Y
插入	N	Y	N
修改	N	Y	Y
删除	N	N	Y

b) 对于对象(订单记录)的授权规则

图12-4 授权规则的实现

例如，在Oracle 8*i*中，可以在数据库或表级授予用户图12-5所包括的特权。INSERT与UPDATE权限可以在列级授予。如果许多用户（例如从事同一种工作的用户）需要相同的权限，就可以创建一个包含这些权限的角色，然后就可以通过对该角色授权而达到对用户赋予相应的

权限。为了向登录ID为smith的用户授予读取产品表和更新价格的权限，可以采用下列SQL命令：

```
GRANT SELECT,UPDATE(unit_price)ON PRODUCT_T TO SMITH;
```

有八个数据词典视图包含着有关可以赋予的权限的信息。在本例中，DBA_TAB_PRIVS包含用户以及用户的对象（用户有操作该对象的权限，如表）。DBA_COL_PRIVS则包含在表的列上赋有权限的用户。

权　限	功　能
SELECT	查询对象
INSERT	将记录插入表或视图中。可以针对特定列进行操作
UPDATE	更新表或视图中的记录。可以针对特定列进行操作
DELETE	从表或视图中删除记录
ALTER	改动表
INDEX	创建表的索引
REFERENCES	创建参照该表的外键
EXECUTE	执行过程、软件包或函数

图12-5　Oracle 8*i* 的权限

12.6.5　用户自定义过程

除了前面讨论的授权规则以外，某些DBMS产品还提供用户出口（或界面），允许系统设计人员或者用户创建他们自己的**用户自定义过程**（user-defined procedure）以保证安全性。例如，可以设计一个用户过程来提供明确的用户ID。在试图登录计算机时，用户除了提供口令以外，可能还要提供过程的名称。如果口令有效，并且提供了过程名称，则系统将会调用该过程，询问用户一系列其答案应该只有该口令的持有者才知道的问题（比如母亲娘家的姓）。

12.6.6　加密

对于公司财务数据这种高度敏感性数据，不妨可以采用加密。**加密**（encryption）就是对数据进行编码，这样人们不能读出它们。有些DBMS产品本身带有加密例程，在敏感数据存储或者在通信信道上进行传输时，系统自动地对其进行编码。例如，在电子资金转账（EFT）系统里普遍采用加密。其他一些DBMS产品则提供允许用户编写自己的加密例程的界面。

提供加密工具的系统，必须同时提供配套的数据解码例程。这些解码例程必须由相应安全性措施进行严格保护，否则加密的优势就丧失了，因此，它们也需要相当多的计算资源。

现在有两种常用的加密形式：单密钥方法和双密钥方法。采用单密钥（one-key）方法（又称为数据加密标准(DES)），发送方和接收方两者都需要知道用来对传输或存储的数据进行编码的密钥。双密钥（two-key）方法（又称为对偶密钥方法、公众密钥方法或者非对称加密），使用一把私钥和一把公钥。双密钥方法为信用卡号码之类的支付数据提供了安全性传输与数据库存储，所以在电子商务应用软件中特别受欢迎。采用这种方法，所有的用户都知道彼此的公钥，但只有每个人才知道其私钥。首先，用户从受托系统（例如DBMS或与DBMS一起工作的安全性系统）分配到一对密钥。一条消息（比如更新数据库的一个事务），先分别用发送方的私钥和接收方的公钥进行加密；这条消息只能用接收方的私钥和发送方的公钥来解密。这里的发送方和接收方，可以是两个用户（或用户程序），也可以是DBMS和一个用户(或用户程序)。双密钥方法取决于其公钥目录是安全的，不可能被破坏。因而，该公钥必须在如第三方这样的高度安全的环境里进行维护。目前已经有像Verisign这样的专门从事维护公钥和检验新密钥的公司。数字签名是双密钥方法的一种变形，它为电子商务提供证据、形式化、确认、有效性和认证（关于数字签名和公钥验证的详细说明，请参见Web资源http://abanet.org/scitech/ec/isc/

dsg-tutorial.html)。

12.6.7 认证模式

计算机界一个长期令人困扰的问题是，如何正确地标识出试图访问计算机或其资源的人士？第一道防线是利用口令，口令能提供一定程度的保护。然而，人们把口令分配给各种设备后，很快又要设计记住这些口令的方法，这些方法会降低口令方式的有效性。口令被写下来，其他人可能会发现它们。其他用户共享这些口令，整个部门共同使用一个访问口令的情况并不少见。让口令包括在自动登录脚本里，这虽然省去了记忆与键入口令的麻烦，但同时也消除了口令的有效性。其次，口令通常是以明文而非加密的形式遍历网络的，一旦被窃听，是很容易被解释出来的。此外，口令本身并不能确保计算机及其数据库的安全性，因为它们无法指出是谁在试图进行访问。当前的分布式与网络式环境需要更进一步的安全性措施。有各种方法及方法的组合正在设法解决这一问题。

首先，业界开发出了一些设备和技术，以明确地标识出任何可能的用户。其中前景最被看好的似乎是**生物设备**（biometric device），它是一种度量或者检测个人特征的技术，这些特征包括指纹、声波纹、眼球图或者动态签名等。由于用来获取视网膜影像的激光有可能会伤害到眼睛，所以利用视网膜影像的方法已经逐渐淡出。为了实现生物方法，有好几家公司已经开发出了一种其上永久性地嵌有个人惟一生物数据（指纹之类）的智能卡。为了访问计算机，用户将该卡插入读取设备（一种生物设备）来读取此人的指纹或者其他特征。然后将实际的生物数据同数据库里的数据进行比较，对于访问计算机的用户来说，两者必须匹配。丢失或者偷来的卡，对于别人是没有用处的，因为生物数据不会相匹配。这类智能卡对于ATM、信用卡消费甚至于电子业务来说都是非常有用的。

另一种方法是利用第三方作中介的认证系统，它通过Kerberos之类受信任的认证代理来确定用户的可靠性。Kerberos由MIT开发，主要在应用软件级协议，如TELNET或FTP中使用，以提供用户到主机的安全性。向用户提供一种密钥（Kerberos卡）的Kerberos套件，然后就可能被嵌入到任何其他的网络协议里。于是，实现了这种协议的任何过程，都可以确定该请求来源。Sun公司也有一种认证机制，称为DES认证，该认证机制基于发送消息时由发送方加密盖上一个时间戳，然后根据接收方的内部时钟进行核对。这就要求与代理保持当前时刻的一致性，而且两者都必须采用相同的加密密钥。不利因素是要使用第三方认证方式需要获得广泛的认同。也有人采用一种建立在公钥认证基础上的认证方式，适合于团体性标识认证，它们则可以在不进一步涉及第三方的情况下进行交换。这类数字认证在涉及信用卡或数字现金消费的电子业务事务中得到广泛应用。

最后一个认证问题是建立不可否认性。也就是说，在发送了一个消息之后，用户不能否认曾经发过该消息，或者说是根本没有发送过任何消息。生物设备与消息发送配合使用就可以确立不可否认性。

12.7 数据库的备份

数据库恢复（database recovery）是数据管理对于Murphy定律的反应。数据库不可避免地会由于某种可能是由人为错误、硬件故障、不正确或无效的数据、程序错误、计算机病毒、网络故障、冲突事务或者自然灾难所引起的系统问题而遭受破坏或者丢失。由于组织非常依赖于其数据库，所以数据库管理系统在发生损失或破坏后，必须提供迅速而精确地复原数据库的机制。

12.7.1 基本的恢复工具

一个数据库管理系统应当提供四种基本的数据库备份与恢复的工具:

1) **备份工具**（backup facility）　周期性地提供数据库整体或者部分备份的拷贝的工具。

2) **日志记录工具**（journalizing facility）　维持事务和数据库变更的一份审计追踪的工具。

3) **检查点工具**　DBMS定期地用该工具挂起所有的处理，与控制文件同步化，并作日志记录。

4) **恢复管理器**　允许DBMS把数据库复原至正确的条件，并再启动处理事务。

1. 备份工具

DBMS应当提供备份工具，用以产生整个数据库以及控制文件和日志的一个备份拷贝(或保存本)。通常，备份拷贝至少每天生成一次。该拷贝应当存放在一个安全的场所，以防止遭到丢失或破坏。该备份拷贝用于在发生硬件故障、灾难性损失或破坏事件发生时复原数据库。有些DBMS配备有备份实用程序，供DBA进行备份；另外一些系统是假设DBA会利用操作系统的命令、导出命令或SELECT-INTO SQL命令来完成备份操作。由于在每天夜里都需要重复性地备份某个数据库，所以预先创建一个可以自动地进行有规律的备份的脚本将会节省时间，而且很少产生备份错误。

对于大型数据库来说，进行有规律的备份是不切实际的，因为完成该备份所需的时间可能相当长。或者，某个数据库可能是一个关键性的系统，它必须始终保持可以使用的状态；故而采用需要关闭数据库的冷备份是不切实际的。此时有规律地备份动态数据（即所谓热备份，其中仅有一部分数据库被关闭使用），而不备份通常是不变的静态数据，往往能减少占用的时间。可以以一定的时间间隔进行增量型备份（仅仅记录自上一次完整备份以来的变化，但完成一次完整备份需要很长时间），这样，两次完整备份之间的时间间隔就会延长。因此，确定备份的策略时必须以对于数据库系统的要求为依据。

2. 日志记录工具

DBMS必须配备日志记录工具，以产生一份**事务**（transaction）和数据库变更的审计追踪。一旦出现故障，就可以利用该日志中的记录以及最近的完整备份，重新建立一致的数据库状态。如图12-6所示，有两种基本的日志或日记。第一种就是**事务日志**（transaction log），它包含着针对该数据库所处理的每一项事务的基本数据的记录。为每一项事务所记录的数据包括事务代码或ID、事务操作或类型(如insert)、事务的时间、终端编号或用户ID、输入数据值、所访问的表或记录、所修改记录以及新旧字段值。

第二类日志是**数据库变更日志**（database change log），它包含被事务修改的记录的前象和后象。**前象**（before image）指记录在被修改前的一个拷贝，而**后象**（after image）则指被修改后的同一记录的一个拷贝。

有些系统还包含一个安全性日志，一旦出现或试图违反任何安全性规则时，它可以警告DBA。

恢复管理器利用这些日志来实施undo（撤销）和redo（重做）操作，这些内容将在本章后面部分再作阐述。这些日志可能保存在磁盘或磁带上；因为它们对于恢复是很重要的，所以也必须对其加以备份。

3. 检查点工具

检查点工具（checkpoint facility）在DBMS里定期地拒绝接收任何新的事务。首先完成所有进行中的事务，然后更新日志文件。此刻，系统处在一种安静的状态，而数据库和事务日志是同步的。DBMS将一条特殊的记录（称为检查点记录）写入该日志文件，这就像是数据库状态的一个快照。该检查点记录包含重新启动系统所必需的信息。任何脏数据块（指包含还没有写入磁盘修改的内存页面）从内存写入磁盘存储器，从而确保在该检查点之前作出的一切改变都已被写入长期存储器。

DBMS可以自动地（最好是这样）或者以响应用户应用软件命令的形式来设置检查点。检查点应当经常设置（比如说，每小时好几次）。一旦出现故障，通常可以从最近一次检查点复原，开始重新处理。因此，只有几分钟的处理工作需要重新完成，而不必完全重做当天好几小时的处理工作。

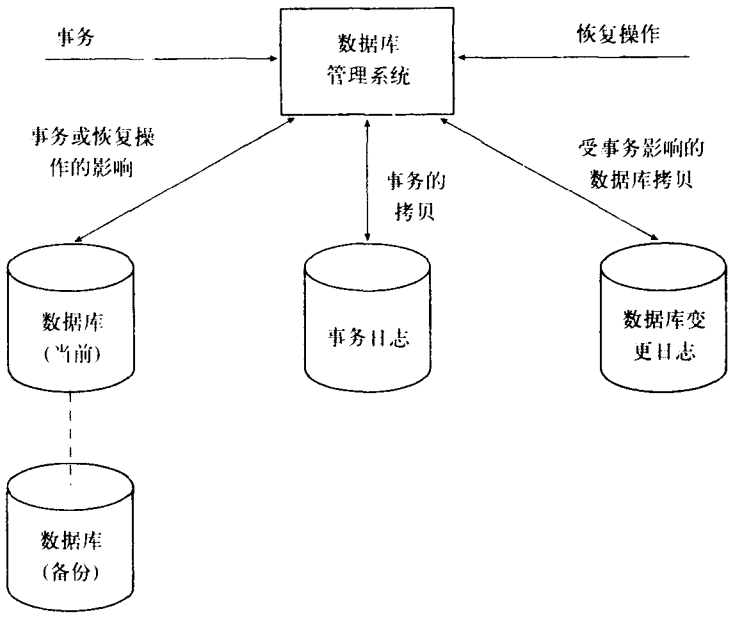


图12-6 数据库审计追踪

4. 恢复管理器

恢复管理器（recovery manager）是DBMS的一个模块，它在出现故障时，将数据库复原成正确的条件，并重新开始处理用户请求。所采用的重新启动类型取决于故障的本质。恢复管理器是利用图12-6所示的日志（倘若必要时，再加上备份拷贝）来复原数据库的。

12.7.2 恢复与重启动过程

在给定情况下，所采用的恢复过程的类型取决于故障的本质、DBMS恢复工具的精细程度以及运作策略与过程。下面讨论最常用的一些技术。

1. 切换

为了能够切换某数据库的某个现有拷贝，该数据库必须是镜像的。也就是说，数据库至少必须有两个拷贝，并且同时更新。一旦出现故障，处理就切换到该数据库的另一个拷贝。这种策略可以实现最快的恢复，并由于长期存储器价格不断下调而大受欢迎。1级RAID系统可用来实现镜像。数据库是分布在几个较小的、廉价的磁盘上，并镜像到一组相同的磁盘上而不是存储在一个大的驱动器上。当一个磁盘出现故障时，系统便直接切换到该镜像磁盘上。有问题或受损的磁盘就可以移除，并用一个新的磁盘代替它。该磁盘可以利用镜像磁盘进行重建，而且不会中断对用户的服务。此类磁盘称为是热交换的。然而，这种策略并不能防止掉电或者数据库的灾难性损害。

2. 复原/重运行

复原/重运行（restore/rerun）技术涉及在要恢复的数据库或部分数据库的备份拷贝上重新处理当天（直到故障点为止）的事务。首先关闭数据库，接着安装要恢复的数据库或文件的最

近（比如说，前一天的）拷贝，然后重新运行自该拷贝以来出现的所有事务(它们存放在事务日志里)。这可能也是制作事务日志的备份拷贝和清除或重做事务日志的最好时机。

复原/重运行的好处是其简洁性。DBMS不需要创建数据库变更日志，也不需要任何特殊的重新启动的过程。然而，它有两点不利因素。首先，重新处理事务的时间可能令人无法接受。根据制作备份拷贝的频率，重新处理可能需要好几个小时。处理新的事务应当推迟到恢复完成后进行，倘若系统负载很重，也许就不能采用它。其次，事务的顺序与其最初的处理顺序经常是不一样的，这可能会产生完全不同的结果。例如，按原来的运行顺序，在取款之前先显示顾客的存款。而在重运行时，也许先进行取款操作，可能会导致向顾客发送资金不足的通知。因此，复原/重运行不是一个充分的恢复过程，一般，它仅仅是作为数据库处理不得已而采用的手段。

3. 事务完整性

数据库往往通过处理其结果是变更一个或多个数据库记录的事务而进行更新。如果在处理事务时出现错误，数据库可能会遭到损害，因而需要某种形式的数据库恢复。因此，为了理解数据库恢复，我们首先必须先理解事务完整性概念。

一个业务事务就是构造某些定义良好的业务活动的一系列步骤。医院的“接纳病人”和制造公司的“输入顾客订单”都是业务事务。通常，一个业务事务包含多个数据库操作。例如，考虑事务“输入顾客订单”。当输入某个新顾客的订单时，可能需要通过某个应用软件程序完成以下步骤:

1) 输入订单数据(由用户键入)。

2) 读取CUSTOMER记录（倘若是新顾客，就插入记录)。

3) 接受或者拒绝该记录。如果Balance Due加上Order Amount不超出Credit Limit，就接受订单；否则就拒绝它。

4) 如果订单被接受，将Order Amount加到Balance Due里。存储更新后的CUSTOMER记录。将已接受的ORDER记录插入数据库里。

在处理事务时，DBMS必须确保该事务具备称之为ACID性质的以下四个广为接受的性质:

- Atomic(原子性)，意味着该事务不能再分割。因此，它要么整体被处理，要么完全不处理。一旦整个事务被处理，我们就称变更已提交。如果该事务在中途出现故障，我们就称它中止。例如，假设程序正在接受一份新顾客的订单，就加上Balance Due，并存储此更新后的CUSTOMER记录。然而，假如新ORDER记录插入不成功（也许存在重复的Order Number键，也许物理文件空间不足)。在这种情况下，我们希望该事务的任何部分都不会影响数据库。
- Consistent（一致性)，意味着在该事务处理前为真的数据库约束在该事务处理后也应该为真。例如，如果手头现有的余额恰好是总收入减去总支出之差，那么它在订单事务发生之前之后都应该为真，它就会扣去手头余额，以满足该订单。
- Isolated（隔离性)，意味着对于数据库的变更直到该事务提交前是不向用户展示的。例如，这一性质表示，直到库存事务完成之前，其他用户是不知道现有库存的。因此，这一性质意味着禁止其他用户同时进行更新，甚至于不能读取正在更新的进程里的数据。我们以后将在并发控制和加锁中更详细地进行讨论这方面的内容。事务彼此相互隔离的后果是，并发事务(即处于某种部分完成状态的若干事务)就仿佛它们是以串行方式交给DBMS那样，将全部影响数据库。
- Durable（持久性)，意味着变更是永久性的。因此，一旦某个事务被提交，随后的任何

数据库故障，都不可能逆转其对该事务的影响。

为了维持事务的完整性，DBMS必须向用户或应用软件程序提供工具来定义**事务边界**(transaction boundary)，即事务的逻辑起点和终点。在SQL里，BEGIN TRANSACTION语句放在该事务内第一个SQL命令之前的，而COMMIT命令则放在该事务的最后。在这两个命令之间，可以放入任意多个SQL命令；前面说过，这些命令就是完成某个定义良好的事务活动的数据库处理步骤。如果在执行BEGIN TRANSACTION之后、执行COMMIT之前，需要处理某个命令(比如ROLLBACK)，则DBMS就会中止该事务，并撤销该事务边界内所处理的SQL语句的影响。应用软件就像是编好了程序，仿佛在该事务中途完成UPDATE或INSERT命令中，DBMS产生了一个出错信息时，执行ROLLBACK那样。因此，DBMS对于成功的事务（它们到达了COMMIT语句）提交（使之持久）变更，而对于中止的事务（它们遇到了一个ROLLBACK）则拒绝作出变更。在COMMIT或ROLLBACK之后，并且在BEGIN TRANSACTION之前遇到的任何SQL语句，将其作为一个语句事务来执行，如果执行中没有错误，就自动地提交，而如果执行中出现任何错误，就被中止。

尽管在概念上事务是业务工作的逻辑单位（比如，一份顾客订单或是接收供货商的新库存），但仍然可能为了完成数据库处理而决定把该业务工作单位再分割成几个数据库事务。例如，因为隔离性，一个包含许多命令并需要很长处理时间的事务，会禁止其他用户在同样的时间内使用同样的数据，从而延误了其他重要的（可能只是读取）工作。有些数据库要频繁地使用，所以尽可能快地完成这些在所谓热点数据上的事务工作变得极其重要。例如，某个主键及其银行账号索引很可能是每个ATM事务都需要访问的，所以必须使数据库事务能迅速使用和释放这些数据。回忆一下，在事务的边界之间的所有命令都是必须执行的，哪怕这些命令是搜索某个在线用户的输入。倘若这个用户很慢地响应在事务边界内的输入请求，其他用户就可能会遭到极大的延迟。因而，只要可能，就要在事务开始之前收集所有的用户输入。此外，为了使事务的长度最小化，在事务之前尽可能早地检查可能有的错误，比如重复的键或者不足的账户余额，这样，万一该事务可能中止，这样就能够尽快地释放出这部分数据库供其他用户使用。有些约束（如收到事项单位数目与库存退还数目的平衡）是不能在许多数据库命令执行之前检查的，所以事务必须有一定长度，以确保数据库完整性。因此，生成数据库事务的指导原则是，只要仍然保持数据库的完整性，就尽可能地短。

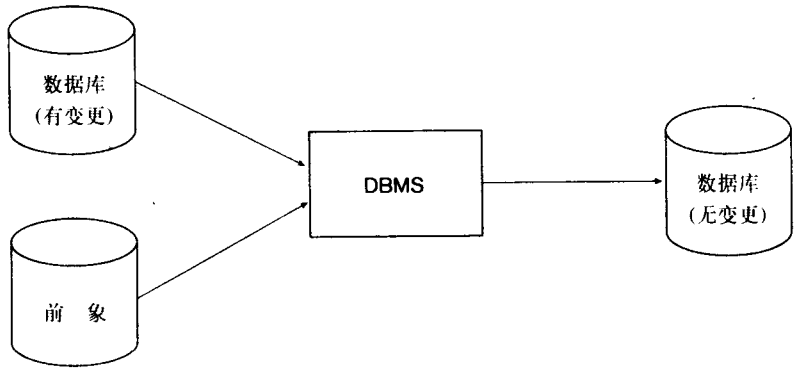
4. 后向恢复

DBMS通过**后向恢复**（backward recovery，也称为**回滚**）来退出或撤销不想要的数据库变更。正如图12-7a所示，把已经变更的记录的前象应用于数据库。其结果是，数据库又回复到早先的状态，不想要的变更被消除了。

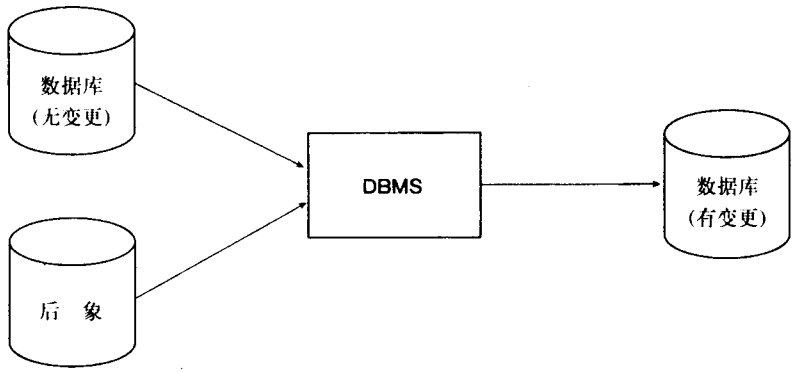
后向恢复常用于恢复由被中止或者异常中断的事务所作出的变更上。为了详细说明后向恢复(或UNDO)的必要性，假设某项银行事务是将$100从顾客A的账户转账到顾客B的账户。这一过程的步骤如下所示:

1) 程序读取顾客A的记录，并从其账户余额中减去$100。

2) 然后程序再读取顾客B的记录，并在其账户余额中加上$100。现在，程序把顾客A的更新记录写入数据库。然而，当试图把顾客B的记录写入数据库时，该程序遇到了某个错误条件(比如磁盘故障)，因而不能写入记录。现在数据库是不一致的（记录A已经更新，而记录B却还没有），该事务必须中止。UNDO命令会使恢复管理器应用记录A的前象，把账户余额复原到其原来的值。(恢复管理器可能再重新启动该事务，再作另一次尝试。)



a) 回滚



b) 前滚

图12-7　基本恢复技术

5. 前向恢复

DBMS通过**前向恢复**（forward recovery，也称为**前滚**）启用该数据库的较早的一个拷贝。利用（作为良好事务结果的）后象，迅速使数据库移向某个后续的状态(参见图12-7b)。基于下列理由，前向恢复要比复原/重运行更快更精确:

1) 不必重复重新处理每个事务的费时逻辑。

2) 只需要应用最近的后象。一个数据库记录可能会有一系列的后象（作为一系列更新的结果），但仅有最近的“良好”后象才是前滚所需要的。

事务的不同顺序的问题被回避了，因为使用的是应用事务的结果（而不是事务本身）。

12.7.3 数据库故障的类型

在处理数据库的时候，有可能会出现各种故障，包括输入某个不正确的数据，直到数据库崩溃或完全丢失。四种最常见的问题是，中止事务、不正确的数据、系统故障以及数据库丢失或崩溃。以下各节将分别描述各种类型的故障，并指出其可能的恢复过程（参见表12-1）。

1. 中止的事务

正如以前所述，完成一个事务常常要执行一系列的处理步骤。**中止事务**（aborted transaction）就是异常的中断。造成这类故障的原因包括人为错误、输入不正确数据、硬件故障和死锁（在下一节介绍）。硬件故障的常见类型是在事务进行过程中出现通信链路的传输丢失。

表12-1 对数据库故障的响应

故障类型	恢复技术
中止事务	回滚(首选)
	前滚/重运行事务直至中止前的状态
不正确的数据	回滚(首选)
(更新不精确)	重新处理带不精确数据更新的事务
	赔偿事务
系统故障	切换至备份数据库(首选)
(未影响数据库)	回滚
	从检查点重启动
数据库崩溃	切换至备份数据库(首选)
	前滚
	重新处理事务

当某个事务中止时，我们需要“退出”该事务，并消除对于数据库已经作出(尚未提交)的任何变更。恢复管理器是通过后向恢复（对问题事务应用前象）来实现这一点的。这个功能应当由DBMS自动地实现，然后通知用户修正并重新提交该项事务。也可以应用其他的过程（如前滚或事务重新处理）使数据库回到中止之前的状态，但回滚是这种情况下首选采用的过程。

2. 不正确的数据

当数据库被合法但不正确的数据更新时，情况就比较复杂一点。例如，为学生记录了一个不正确的成绩，或者对某个顾客应付款输入了不正确的总额。

不正确的数据是很难检测出来的，经常会导致混乱。检测错误并改正数据库记录会消耗大量时间。就在此时，也许已经有很多其他用户使用过这些错误数据，于是，由于各种应用软件都可能使用了不正确的数据，因而会出现一系列错误的反应。此外，基于这些不正确数据而产生的事务输出（比如文档或消息）可能已经传递给了用户。例如，给学生发送了一份不正确的分数报告，或者给顾客发送了一段不正确的陈述。

在已经导入不正确的数据时，数据库可以采取以下办法来进行恢复:

1) 如果错误发现得比较早，就可以采用后向恢复。(然而，必须小心确保所有的后续错误都已被纠正。)

2) 如果仅仅出现为数不多的错误，就可以通过人为的干预引入一系列赔偿事务，以校正错误。

3) 如果前面两种办法都不合适，那就只能从出现错误之前的最近一个检查点开始重新启动，随后的事务在处理时就不会产生错误。

凡是由错误性事务所产生的任何错误消息或报告，都应当通过适当的人为干预（解释信、打电话等等）加以纠正。

3. 系统故障

在系统故障中，该系统的某个组件失效，但数据库并没有遭到破害。系统故障的原因包括掉电、操作错误、通信传输损失以及系统软件故障等。

当系统崩溃时，可能正在处理某些事务。恢复的第一步就是利用前象（后向恢复）退出这些事务。然后，倘若系统已经进行镜像，就可以切换到其镜像数据，并在一个新的磁盘上重新建立被破坏的数据。倘若该系统没有被镜像，那么因为在主存里的状态信息已丢失或损坏，所以无法重新启动。最安全的做法是从系统故障前的最近一个检查点开始重新启动。对于检查点后需要处理的所有事务，数据库是应用后象进行前滚的。

4. 数据库崩溃

在**数据库崩溃**（database destruction）的情况下，数据库本身已经丢失、被破坏，或者不能读取了。数据库崩溃的典型的原因是磁盘驱动器故障(或磁头毁坏)。

对于这类事件的恢复，其首选策略也是利用数据库的镜像拷贝。倘若没有镜像拷贝，则需要一个数据库的备份拷贝。前向恢复用来将数据库复原到出现丢失前的状态。在数据库丢失时正处于进行中的任何事务都可以被重新启动。

12.8 并发访问的控制

数据库是共享的资源。数据库管理必须为同时有多个用户试图访问和加工处理数据作好规划。由于涉及到更新的并发处理，所以一个数据库要是没有**并发性控制**（concurrency control），则将会由于用户之间的干扰而麻烦不堪。并发性控制有两种实现办法，一种是悲观的方法(利用加锁)，另一种是乐观的方法（利用版本设置）。我们将在以后的内容中介绍这两种方法。

绝大多数DBMS是在多用户环境下运行的，用户希望在这种环境下能够共享包含在数据库里数据。倘若用户只是读取数据，就不会遇到任何数据完整性问题，因为数据库并没有任何的变化。然而，如果有一个或多个用户在更新数据，那么就会遇到维护数据完整性的问题。当同时处理多个数据库的事务时，该项事务被看成是并发的。用于确保维持数据完整性的操作称为并发性控制操作。回想一下，CPU每次只能处理一条指令。当提交某个新事务时还有其他也是针对该数据库的处理，那么通过CPU在事务之间的切换，使得CPU轮流访问每个事务，从而完成每个事务的某个部分。因为CPU能够非常快地在事务之间切换，所以绝大多数用户都不会感觉到他们是在和其他用户分享着CPU时间。

12.8.1 更新丢失的问题

在没有足够的并发性控制条件下，多个用户试图更新某个数据库所遇到的最普遍的问题就是更新的丢失。图12-8显示了一种很常见的情景。John和Marsha拥有一个联名支票账户，两个

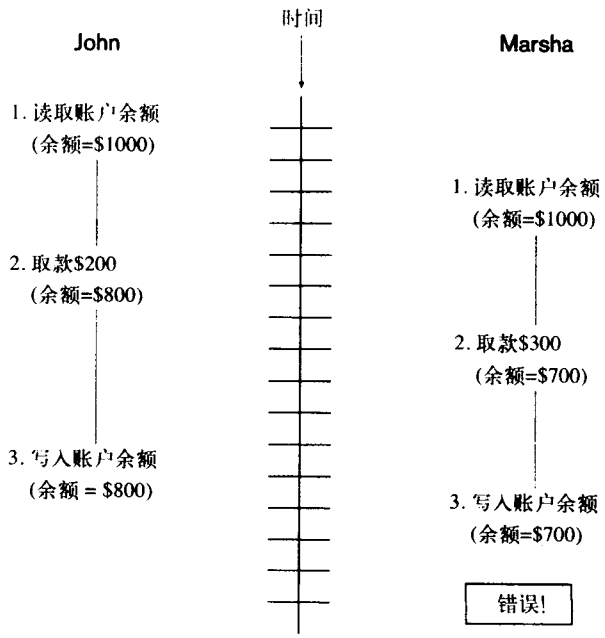


图12-8　更新的丢失(没有任何并发性控制的结果)

人同时想要提取一些现金，他们在不同的地点使用ATM终端。图12-8显示了在缺少并发性控制机制时，可能会出现的事件序列。John的事务读取账户余额（为$1000），然后他取款$200。在该事务写入新账户余额（$800）前，Marsha的事务读取了账户余额（仍是$1000）。然后她取款$300，此时余额为$700。她的事务然后写入这个账户余额，这替换了John事务所作的写入操作。John的更新被丢失的结果是由于事务之间的干预造成的，银行当然不希望出现这种情况。

在不建立并发性控制时，可能会出现的另一个类似问题是**不一致读问题**（inconsistent read problem）。这个问题在一个用户读取已经被其他用户部分地更新的数据时出现。这种读取是不正确的，所以有时也称为脏读或者不可重复读。当DBMS没有隔离事务（事务的ACID性质的一部分）时，就会遇到更新丢失和不一致读问题。

12.8.2 可串行性

并发的事务应该按隔离方式处理，使得它们不会彼此相互干扰。如果在处理其他事务之前可以完整地处理完前一个事务，那么就不会发生干扰。处理事务时，其结果与前面所描述的情景一样，那么该过程被称为是可串行性的。利用某个可串行性方案来处理事务，其结果就像是一个紧接着另一个处理的。只要计划方案设计得事务不会彼此干扰，那么事务仍然可以并行地运作。例如，从数据库的不同表中请求数据的事务就不会彼此冲突，它们可以并发地运作，而不会产生数据完整性问题。可串行性可以通过不同的手段实现，但加锁机制是最常见的并发性控制机制。利用**加锁**（locking）机制，用户为了更新而检索的任何数据，必须加以锁定，即直到更新完成或中止之前，拒绝其他的用户。数据加锁就像是从图书馆里查出一本书，在其借阅者归还之前，其他人是无法得到的。

12.8.3 加锁机制

图12-9显示了采用记录锁来维持数据完整性的情况。John启动从某个ATM提款的事务。由于John的事务需要更新其记录，应用软件程序在将其读入主存之前，先锁定该记录。John继续提款$200，然后计算出新的余额($800)。在John的事务开始不久，Marsha启动了一个提款事务，但她的事务在John事务把更新记录归还给数据库，并对该记录解锁之前是无法访问该账户记录的。因此，加锁机制强制实现了一个有序的更新过程，防止错误的更新。

1. 加锁等级

实现并发性控制的一个重要因素是选取加锁等级。**加锁等级**（locking level，又称为**粒度**（granularity））就是每次锁定所包括的数据库资源的范围。绝大多数商用产品都在下述某个级别上进行加锁：

1) 数据库 整个数据库被锁定，其他用户都不能使用该数据库。这一级别的应用软件很少，例如在对整个数据库作备份时所用的软件（见Rodgers,1989）。

2) 表 锁定包含所请求记录的整个表。这一级别主要适合于更新整个表的批量更新，例如将所有的员工薪水增加5%之类。

3) 块或页 锁定包含所请求记录的物理存储块(或页)。这是最常实现的加锁级别。页有固定的大小（4K、8K等），并能包含多种类型的记录。

4) 记录级 仅锁定所请求的一个记录（或一行）。其他的记录（即使是该表内的），都可以供其他用户使用。当一次更新涉及几个记录时，在运行时要强制一些负载。

5) 字段级 仅仅锁定所请求记录的特定的字段(或列)。当大部分更新最多影响记录里的一个或两个字段时，这一级锁定可能是最合适的。例如在库存控制应用软件中，现有数量字段会经常修改，但其他字段（比如描述字段或存储箱位置字段）不经常被更新。字段级锁定需要相当大的开销，因此很少使用。

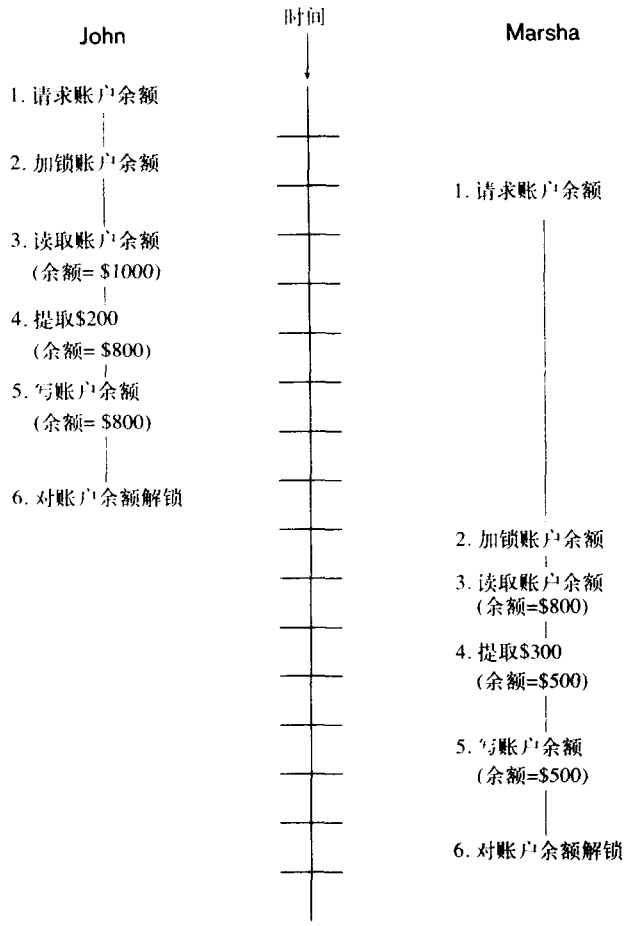


图12-9　采用加锁的更新(并发性控制)

2. 锁的类型

迄今为止，我们仅仅讨论了防止对锁定项目进行访问的加锁。其实，数据库管理员一般可以在两种类型的锁中进行选择：共享锁和排他锁。

1) **共享锁**　共享锁（也称为S锁或读锁）允许其他事务读取（但不是更新）记录或者其他资源。当一个事务仅仅是读取而不是更新一条记录或数据资源的时候，应当在其上放置一个共享锁。在一条记录上放置共享锁防止了另外的用户在其上放置排他锁（而不是共享锁）。

2) **排他锁**　排他锁（也称为X锁或写锁）能防止其他事务读取（包括更新）记录，直到解锁为止。当准备更新记录时，事务就应当在记录上设置排他锁（见Descollonges, 1993）。在一个记录上设置排他锁能够防止其他用户对该记录再设置其他类型的锁。

图12-10显示了将共享锁和排他锁应用于检查账户的例子。当John启动其事务时，程序在他的账户记录上设置一个读锁，因为他正在读取记录，以检查账户余额。当John请求提款时，程序试图在该记录上设置一个排他锁（写锁），因为这是一个更新操作。然而，如图所示，Marsha已经启动她的事务，该事务在同一个记录上设置一个读锁。结果，John的请求被拒绝了。请记住，倘若有一个记录是读锁，则其他用户就不能获得写锁。

3. 死锁

加锁解决了错误更新的问题，但又可能会导致另外一个所谓**死锁**（deadlock）的问题，即

两个或更多的事务都锁定某个共同的资源，而每个事务都在等待他人释放该资源的一种僵局。图12-10显示了简单的死锁例子。John的事务在等待Marsha事务消除对该账户记录的读锁，反之亦然。尽管余额大大超出实际所需，但他们两人都无法提款。

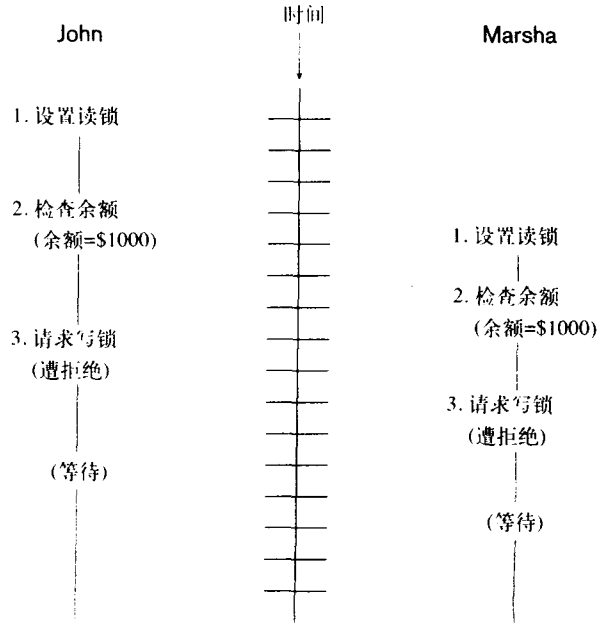


图12-10　死锁的例子

图12-11显示了一个稍微复杂的死锁的例子。其中，用户A锁定记录X，而用户B锁定记录Y。然后，用户A请求记录Y（打算更新），而用户B则请求记录X（也是打算更新）。两个请求都遭到拒绝，因为所请求的记录已经被锁住。除非DBMS干预，否则两个用户都将无限地等待下去。

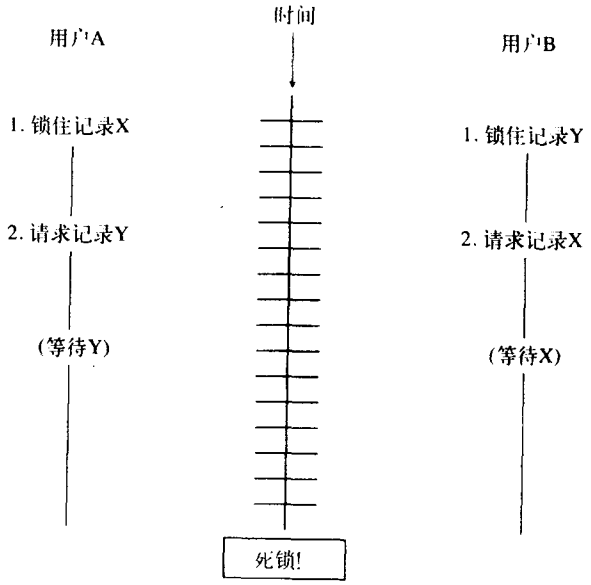


图12-11　另一个死锁的例子

4. 死锁的管理

解决死锁有两种基本方法，即死锁预防和死锁化解。当采用**死锁预防**（deadlock prevention）时，用户程序必须在事务一开始，锁定它所需要的所有记录，而不是每次锁定一条记录。在图12-11里，用户A应当在处理事务前先锁定记录X和Y。倘若两条记录都已被锁定，程序就必须等待，直到它们被释放为止。凡是在任何资源被解锁之前，又出现需要加锁操作的地方，就要应用一个**两阶段加锁协议**（two-phase locking protocol）。一旦释放了该事务的任何锁，就不需要再加锁。从而，在两阶段加锁协议里的阶段，通常一个称为增长阶段（其时需求着所有必要的锁），而另外一个称为收缩阶段（其时所有的锁在释放着）。锁不应该被同时请求。经常出现的情况是，请求某些锁，进行处理，然后又请求所需的其他锁。

在事务一开始时就锁定所有需要的记录（称为保守的两阶段加锁）可以预防死锁。但是，通常很难预测在处理某个事务时下一步会需要哪些记录。典型的程序都有许多处理部分，并可能以不同的顺序调用其他程序。其结果是，死锁预防并不实用的。

在两阶段加锁中，每个事务都必须按相同的顺序请求记录（即对资源串行化），这样也能够预防死锁，但这也并不实用。

第二种较常用的方法是允许死锁出现，但在DBMS里建立检测和打破死锁的机制。实质上，**死锁化解**（deadlock resolution）机制的工作原理如下。DBMS维护着一个资源使用矩阵，它能在某个瞬间指出哪些主体（用户）在使用哪些对象（资源）。通过扫描这个矩阵，计算机就能够检测到是否出现了死锁。DBMS然后便通过“撤销”某个被死锁的事务，来化解该死锁。该项事务直到死锁前所作出的任何变更都将消除，等到其请求的资源变为可用时，再重新启动该项事务。我们后面将很快再详细描述这个过程。

12.8.4 版本设置

这里所描述的加锁通常被称为悲观（pessimistic）并发性控制机制，因为每次只请求一个记录，DBMS采取密切注意被锁定记录的方法，使得其他程序不能再使用它。其实，在大多数情况下，其他用户不会去请求同一个文档，或者它们可能只是需要读取一下，那是没有问题的（见Celko,1992）。因此，冲突是很少的，一种较新的并发性控制方法（被称为**版本设置**，versioning）则是采取乐观（optimistic）的方法，大多数时间里其他用户是不会需要同一个记录的，即便他们要求该记录，那也只是要求读取（并非更新）此记录。利用版本设置，就不再需要任何形式的加锁。在事务开始时，每个事务都约束限制于数据库的一个视图，在某个事务修改记录时，DBMS就创建一个新的记录版本，覆盖替代旧的记录。

理解版本设置的最好办法就是将数据库看作一个中央记录室（见Celko,1992）。该记录室有一个服务窗口。用户（对应于事务）到达该窗口，并请求文档（对应于数据库记录）。然而，原始的文档从不离开该记录室。相反，职员（对应于DBMS）为所请求文档制做拷贝，并给它盖上时间戳。用户随后取走其私人的文档拷贝（或版本），回到其工作地进行阅读或修改。当完成时，他们把打过标记的拷贝归还给中央数据库。在没有任何冲突的情况下（例如，只有一个用户对一组数据库记录作过变更），该用户所作的变更就会直接合并到公共（或中央）数据库里去。

假设存在冲突。例如，两个用户对其私人数据库拷贝作出了相互矛盾的变更。此时，其中一个用户作出的变更已经提交给数据库了（请记住，事务都是盖有时间戳的，早的事务能够优先提交）。另一个用户必须被告知存在冲突，他的工作不能被提交（或者不能合并到中央数据库）。他必须找出数据记录的另一个拷贝，并重复原先的工作。不过，这类重复工作是极少数的异常情况，而不是经常发生的。

图12-12显示了利用版本来检查账户的简单例子。John读取包含账户余额的记录，成功地提取了$200，而新的余额（$800）通过COMMIT语句发送到账户。其间，Marsha也已读取该账户余额，并请求提款，该请求发送到她账户记录的本地版本。然而，当此事务试图COMMIT时发现了更新冲突，于是她的事务中止（或许出现“不能在此时完成事务”之消息）。她然后从正确的新初始余额$800重新启动其事务。

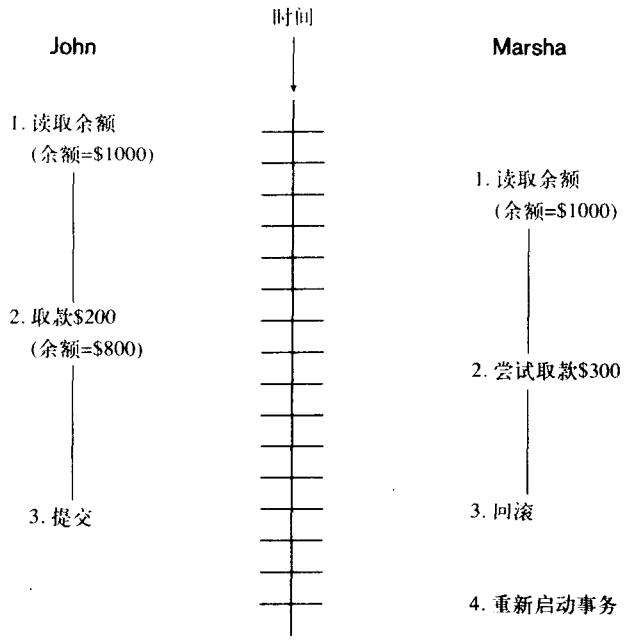


图12-12 版本设置的用法

版本设置相对于加锁的主要优势在于性能上的改善。只读事务可以与更新事务并发地运行，而不会损失数据库的一致性。

12.9 数据质量的管理

今天，组织管理的基础是高质量的数据，即精确的、一致的并能及时使用的数据。组织必须尽可能标识出适宜于其决策制定的数据，开发确保数据精确性与完备性的业务策略与规程，并提供全企业范围的数据共享。

建立一个由代表每个主要业务部门的有权制定业务策略的人员所组成的业务信息咨询委员会，对于建立高质量的数据是非常作用的（见Moriarty，1996）。这些委员会成员是IT和其业务部门之间的联系人，他们考虑的不仅是本职能部门的信息需求，而且包括全企业范围的数据需求。委员会的成员必须具有把信息作为企业资源进行管理的强烈兴趣，深刻理解该组织的业务，并具有良好的协调能力。这些成员有时候可称为**数据管家**（data steward），即负责确保组织内的应用软件正确支持该组织企业目标的人员。他们也必须确保所获得的数据都是精确的，在整个组织内是一致的，这样，整个组织的用户都能够依靠其数据。

建立企业范围内的数据质量标准并不容易，甚至于还会有某些业务机构和规程会影响数据质量（见Moriarty，1996）。在设立彼此竞争的战略性业务部门的组织内，数据共享可能是很困难的，因为每个战略性业务部门都力求保护其竞争地位。有些正规的产业（如银行和通信业）可以合法地限制能够共享的数据量。维护高质量数据的另一种威胁可能是：数据记录会受到奖

励计划的引诱，或是建立在完成工作量基础上的定额系统。那时，实现高速数据输入的需求可能会压倒对于数据精确性的要求。凡是从一个以上业务部门收集来的需要共享的数据，其不精确性和不一致性的比例是相当惊人的。必须要开发出处理上述问题的策略。

除了处理上述问题之外，任何组织都需要处理关系到实现高质量数据的五个方面：

1) 安全性策略和灾难恢复。

2) 人员控制。

3) 物理访问控制。

4) 维护控制。

5) 数据保护和私密性。

12.9.1 安全性策略与灾难恢复

每个组织（无论大小）都应当建立起全面的安全性策略，并制定详细的灾难恢复计划。安全性策略将决定组织如何维持一个安全系统。灾难恢复计划将确定组织在遇到洪水或火灾之类的紧急情况时，应当如何继续执行其职能。

应当编写安全性策略以提供关于哪些地方需要有安全性措施的指导。应当根据已建立的策略确定员工的职责和义务，并应当指出不遵循策略所带来的后果。最后，还应当指出实施这些指导所需要采取的一般过程。这些一般过程将通过更专门的过程来实现，当开发使用新的技术或者开发出加强安全性环境的新产品时，这些过程是策略中最可能需要随着时间推移而变更的部分。

灾难恢复计划应当在可能预见的任何灾难出现以前就详细地制定出来，并尽可能地加以测试。例如，1993年Andrew飓风袭击了迈阿密地区，Tampa Bay公司启动了他们的飓风灾难恢复计划。结果，有些组织发现，他们利用航班把数据磁带送往安全地点的计划根本不现实，于是他们利用这个机会改进了飓风灾难恢复计划。应急计划必须考虑到向其他地点的转移过程和安全措施中的数据、设备、人员以及管理问题。计划的拷贝应当保存在多个地方，并定期加以查看。

表12-2　数据库保护过程

备份和开发环境
磁带备份　为找出备份磁带设定正式的过程
磁盘备份　最大限度防止未授权使用数据库所需要的磁盘备份和冗余(RAID)拷贝
开发数据库　如果它们包含实际数据，则必须对程序员、开发者、可接受的测试者以及与系统开发有关用户进行控制，使得数据不向没有相应权限或不需要知道的人员展示
系统与专用账户
这些账户(如由DBMS提供的系统管理账户SYSMANAGER和DBA，以及用于系统测试的账户)往往比典型的用户和开发者账户拥有更高的权限。这些用户ID和口令必须加以保护(也必须加以记录)，使得它们只能授予有相应权限的人员
数据库名与位置
建议对所有的物理数据库对象都使用别名，以尽可能地隐藏数据的实际位置。这一点也在本章前面讨论的用户视图中得到强调
系统文件
这些文件包含敏感的数据库数据，比如用户ID、数据位置、字段描述以及其他的数据词典内容等等。必须保护这些内容，防止它们被破坏或非授权使用。一种策略是，将这些文件存储到与实际数据库不同的设备上，可以由操作系统采取额外级别的保护，并设定对这些设备的物理访问
口令控制
设置口令使其经过一段时间后失效，从而强制所有的数据库用户定期地变更口令。这可以大大地限制失窃或不恰当共享口令的责任
隔离数据库
将不同的生产以及生产与测试数据库分别置于单独的环境中，并对每一个数据库采用单独的服务器和操作系统级安全性

注：本表摘自Loney(2000)。

大型的组织可能都安排有空间（一个冷场所），以便在发生灾难事件时使用，该地点可以和组织位于同样的物理地区，也可以位于与组织所在地相隔遥远的某个地区。有时候，一个组织会在某个暖场所投资，其中包含建立计算机操作所必需的所有设备。也有的组织维持某个热场所，其中运作是按双份配备的，一旦一个系统遇到问题，立即可以切换到另外一个系统。

上述讨论强调，在安全性受损或者灾难事件中，为了保护数据库环境所必需的几个重要的过程和策略。表12-2总结了一些专门过程，它们配合DBMS所提供的安全性命令（如视图、权限、用户账户、加密以及生物设备）使用，从而产生一份全面的数据库保护计划。

12.9.2 人员控制

必须开发和遵循完全的人员控制策略，对业务安全性的最大威胁通常是来自内部而不是外部。除了本章早些时候讨论的安全性授权和认证过程以外，组织还应当开发相应的过程来确保一个有选择的雇佣程序能够验证员工关于背景和能力的表现。要监控以确保人员遵循着已建立的规程、享用正规的假期、能够与其他员工一起工作等等。员工应当接受与其职位相关的安全性和质量方面的培训，鼓励他们熟悉并遵守标准的安全性和数据质量控制措施。应当强制实施标准的作业控制。例如分离责任，这样使得没有任何个人要对整个业务过程负责，或者使应用软件开发人员访问生产系统。如果一个员工离开，那么应该执行一系列过程来撤销该员工的授权和认证，并把状态的变化告知其他员工。

12.9.3 物理访问控制

限制进出办公楼的特定区域，往往也是控制物理访问的一部分。包括硬件和外围设备在内的敏感设备，如打印机（它也能用来打印分类报表），可以通过将其放在安全区域而加以控制。其他的材料可以锁在抽屉或柜子里，或者可以安装一个报警器。备份的数据磁带应当保存在防火的数据保险箱里或者保存在其他的安全地点。清晰标记移动媒介和处置媒介的日程计划，并对存放的所有物品做好标记和索引，诸如此类的过程也必须确立。

计算机屏幕放在适当的位置，使得从办公楼外无法看到它可能也是很重要的。还应当开发一个对办公楼外区域进行控制的过程。公司经常利用保安人员对进出办公楼的人员加以控制，或者采用一个卡片扫描系统或指纹识别系统使员工可以自由进出办公楼。可以给来访者发放标识卡，并由人陪同通过大楼。

12.9.4 维护控制

一个有助于维护数据质量，但却往往被忽略的控制领域，就是维护控制。组织应当审查它们所使用的所有硬件和软件的外部维护协议，确保达到合适的响应速率，以维护数据质量。考虑与所有关键的软件开发者达成协议，使得组织得以能够访问源代码也是很重要的，以防开发者结束业务或者停止对程序的支持。实现上述目标的办法之一是，让某个第三方持有源代码，并订立协议，一旦出现这类开发情况，便可发布该程序。

12.9.5 数据保护与私密性

随着越来越多的人开始熟悉计算机以及计算机通信的不断增加，人们越来越关心不得随意收集和散布私人信息方面的个人权利。信息私密性法规通常规定个人有权了解收集了他们的哪些数据，并校正这些数据中的任何错误。因为被交换的数据量在不断地增长，开发合适的数据保护措施的需求也在不断增长。适当的提供将数据用于合法用途的许可，这也是很重要的，这样组织中需要这些数据的人士能够访问它们，并相信它们的质量。

12.10 数据词典与信息库

RDBMS的一个不可分割的部分就是**数据词典**（data dictionary），它用来存储元数据，即

关于数据库的信息，包括该数据库中每个表的属性名和定义。数据词典通常是为每个数据库所生成的**系统目录**（system catalog）的一部分。该系统目录描述所有的数据库对象，包括诸如表名、表创建者或拥有者、列名与数据类型、外键与主键、索引文件、授权用户、用户访问权限之类的与表有关的数据。系统目录是由数据库管理系统创建的，并且信息存储在系统表里，这些系统表可以采用像任何其他数据表一样的方式进行查询，只要用户拥有相应的访问权限即可。

数据词典可以是主动型或者被动型的。主动型数据词典是由数据库管理软件自动地进行管理的。主动型系统总是与数据库的当前结构和定义相一致的，因为它们是由系统本身来维护的。绝大多数关系数据库管理系统现在都包含着主动型数据词典，这些数据词典可以直接从它们的系统目录里派生出来。被动型数据词典是由该系统的用户(们)管理的，并在数据库结构改变时加以修改。由于这种修改必须由用户手工完成，因此，与数据库的当前结构相比，它可能不是最新的。然而，被动型数据词典可以在某个单独的数据库里进行维护。这在设计阶段可能是很理想的，因为它允许开发者尽可能长久地与采用特定的RDBMS保持独立性。而且，被动型数据词典可以不局限于其数据库管理系统所能够辨别的信息。因为被动型数据词典是由用户来维护的，所以可以扩展它们以包含未被计算机化的组织数据的信息。

信息库

虽然数据词典只是简单的数据元素记录工具，然而。数据管理员和其他信息专家可以用信息库来管理整个信息处理环境。**信息库**（information repository）既是开发环境的基本组件，也是生产环境的基本组件。在应用软件开发环境中，人们（既可以是信息专家，也可以是最终用户）使用CASE工具、高级语言以及其他工具来开发新的应用软件。CASE工具可以自动连接到信息库。在生产环境中，人们使用应用软件来建立数据库、保持数据为最新的以及从数据库中抽取数据。所有这些活动都需要访问信息库，并且保证信息库是最新的。

如前所述，CASE工具通常产生的信息应当是信息库的一部分，因为它本身可作为文档编制工具、项目管理工具以及数据库管理软件。当它们在第一次开发时，由这些产品所记录的信息是不容易集成起来的。然而，现在已经出现了许多使这些信息更容易访问和共享的尝试。**信息库词典系统**（Information Repository Dictionary System，IRDS）就是用来管理和控制对信息库的访问的一种计算机软件工具。它提供了记录、存储和处理某个组织的重要数据和数据处理资源的工具（见Lefkovitz,1985）。当系统兼容IRDS后，就可以在由各种产品所产生的数据词典中间传递数据定义。IRDS已被国际标准组织（ISO）采纳为一种标准（1990），它包含存储数据词典信息和访问这些信息的一组规则。

图12-13显示了一种信息库系统体系结构的三个组件（见Bernstein,1996）。第一个组件是信息模型。这个模型是存储在信息库里的信息的一种模式，与该数据库相联系的工具可以用该模型来解释信息库的内容。第二个组件是信息库引擎，它管理着信息库对象。其中包括读取与写入信息库对象、浏览以及扩展信息模型等服务。第三个组件就是信息库数据库，信息库对象就存储在其中。请注意，信息库引擎支持五种核心功能(见Bernstein,1996):

- *对象管理*　面向对象信息库存储有关对象的信息。随着数据库面向对象程度的不断提高，开发者就可以利用在信息库里存放的关于对象的信息。信息库能够基于一个面向对象的数据库，或者附加支持对象的功能。
- *联系管理*　信息库引擎包含着关于对象联系的信息，这些联系有助于使用依附于该数据库的软件工具。
- *动态可扩展性*　信息库信息模型定义类型，它应当很容易进行扩展，即追加新的类型或者

扩展已有的某种类型的定义。这一功能使它很容易将某个新的软件工具集成到开发过程里。

- **版本管理** 开发时，建立版本控制是很重要的。信息库能够用来为软件设计工具提供版本控制功能。对象的版本控制要比文件的版本控制更加困难，因为在应用软件里的对象往往比文件的对象更多，而且一个对象的每个版本又可能有许多联系。
- **配置管理** 将版本化的对象加入到也是版本化的代表整个系统的配置里也很有必要。它可以帮助我们把配置看作与文件目录相类似的东西，只是配置是可以版本化的，而且它们包含的是对象而不是文件。信息库经常用检出系统来管理对象、版本和配置。希望使用某个对象的开发者将该对象检出，作出适当的变更，再将该对象检入。此时，创建了对象的一个新版本，并且其他开发者又可以使用该对象了。

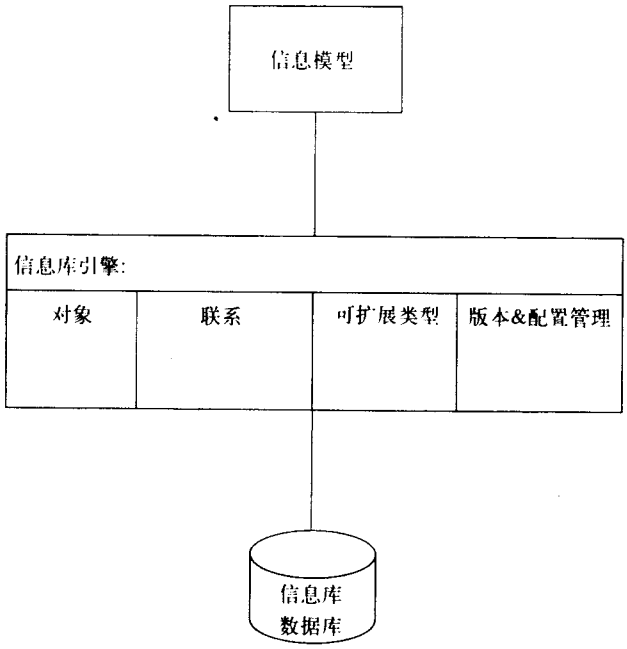


图12-13 信息库系统体系结构的三个组件(改编自Bernstein,1996)

随着面向对象数据库管理系统的可用性越来越高，关系数据库相联系的面向对象编程的增长，信息库的重要性也在不断增长。因为面向对象开发需要利用包含在信息库里的元数据。此外，在IRDS标准已经被普遍接受的今天，由不同的软件工具所产生的元数据和应用软件信息可以很容易地集成到信息库里。虽然信息库已经被包括在企业级开发工具里，但由于对面向对象开发的关注仍然不断增长，这将会导致信息库的用途更加广泛。

12.11 数据库性能调整概述

有效的数据库支持可产生可靠的数据库，其性能不会受到硬件、软件或用户应用软件的干扰，并能达到其最佳性能。调整数据库并非仅仅是在DBMS安装时或者实现某个新的应用软件时所做的一项工作，然后就听之任之。相反，性能分析和调整是所有数据库管理持续存在的一部分，因为硬件和软件的配置在改变，而用户的活动也在改变。当尝试维护一个调整良好的数据库时，就应当解决DBMS管理的五个方面的问题，即DBMS的安装、内存利用、输入/输出争

用、CPU利用以及应用软件调整。数据库管理员对上述每个方面影响随着DBMS产品的不同会有很大的区别。我们在本节所采用的DBMS是Oracle 8*i*，不过应当指出，每个产品都具有其固有的调整功能集合。

调整数据库应用软件需要熟悉其系统环境、DBMS、应用软件以及该软件所使用的数据。这种场合正是检验数据库管理员的功力的地方。要达到一种安静的环境，也就是一种可靠的、允许用户有把握及时得到所需信息的环境，所需要的技术能力和经验，只有通过多年的数据库工作实践方可获得。以下所讨论的方面都是十分一般的，主要让用户对调整数据库所涉及活动的范围有初步理解，而不是详细介绍调整某个特定数据库应用软件的细节。

12.11.1 安装DBMS

DBMS产品的正确安装对于任何环境来说都是必需的。产品中通常都会包括一个README文件，该文件中包含详细的安装指示、过程的修订、安装时必须增加磁盘空间的注意事项等等。快速浏览README文件可以节省安装时间，并产生较好的安装效果。一般性的安装指示也应当浏览一下，否则有可能会在安装过程中产生默认值，而这些默认值未必是该用户的最佳选择。这里列举一下某些可能的注意事项。

在安装开始前，数据库管理员应当确保可以提供充足的磁盘空间。你应该参考特定DBMS的手册，以便将逻辑数据库大小参数（比如字段长度、表的行数和估计的增长）转化为实际的物理空间需求。也许推荐的空间需求是很低的，但由于要对DBMS作出改变，所以空间应该大一些，不过，这个文档不能反映其变化。为了保险起见，一般至少要比所建议的空间多分配20%。在安装之后，要查看在安装过程中所产生的所有日志文件。它们的内容能够反映出没有注意到的安装问题，或者提供安装过程如预期那样顺利的保证。

还需要考虑为数据库分配的磁盘空间。例如，某些UNIX备份系统在处理大小超过1GB的数据文件时会出现问题。保持数据文件都小于1GB就能避免可能出现的麻烦。按标准大小分配数据文件，能够很容易平衡I/O，因为数据文件位置的交换比较容易，这应当是化解瓶颈所需要的。

12.11.2 内存利用

内存的有效利用，涉及到理解DBMS如何使用内存，正在使用哪些缓冲，以及在内存中的程序有哪些要求。例如，Oracle在数据库运行某个数据库管理功能时，有许多驻留在内存中的后台过程。某些操作系统需要一段连续的内存块才能装载Oracle，而内存不足的系统应该首先释放内存空间。Oracle在内存中维护一个数据词典高速缓冲，理想情况下，它的大小应该使得至少90%的数据词典请求可以容纳其中，而无须检索磁盘信息。这里的每种情况，都是在调整数据库时应当考虑的典型的内存管理问题。

12.11.3 输入/输出争用

达到最大的I/O性能是调整数据库最重要的方面之一。数据库应用软件是I/O密集型的——生产数据库在其工作时通常会向磁盘读写大量的数据。尽管CPU的时钟速度急剧增长，但I/O速度却没有相应的增长，而且增加了复杂性的分布式数据库系统则会有更为复杂的I/O功能。

理解最终用户如何访问数据对于管理I/O争用极其重要。当开发热点（即反复访问的物理磁盘位置），理解引起热点的活动的本质为数据库管理员提供更多的机会来减少I/O争用。Oracle允许DBA控制包含数据文件的表空间的位置。DBA对用户活动的深刻理解有助于他通过分隔开同时访问的数据文件来减少I/O争用。只要有可能，就应当将需要并发性访问的大型数据库对象从磁盘上剥去，以减少I/O争用，并改善性能。分布式I/O活动的总体目标均匀地分布在磁盘和驱动控制器上，应当能指导DBA调整I/O。

12.11.4 CPU利用

绝大多数数据库操作正在趋向于要求CPU工作活动。因此，在调整数据库时，对CPU利用的评估就很重要。采用多CPU，就能在CPU并行工作时，共享查询处理，而且性能也可以得到极大提高。DBA应该使现有的CPU性能达到最大，尽管规划的收益可以通过每次CPU的更新换代来达到。

监控CPU负载，从而了解24小时的典型负载，这可以向DBA提供开始重新平衡CPU负载所需的基本信息。在各种环境下都应该重新调整联机与后台处理的混合。例如，建立“凡是可以在工作时间以外的时间运行的所有作业，必须在工作时间以外的时间里运行”这样一条规则，将有助于减轻机器在高峰工作时间的负载。确立带有有限空间的用户账户，也有助于管理CPU负载。

12.11.5 应用软件调整

前面几节集中讨论了调整DBMS的活动。考察最终用户在数据库中使用的应用软件，也可能会提高性能。尽管规范化至少要达到3NF是许多采用关系数据模型的组织所期待的，仔细计划的非规范化(见第6章)仍然可能会改善性能，这通常在运行某个SQL查询时，通过减少必须联结的表的数目来实现。

考察和修改应用软件中的SQL代码也可以改善性能。例如，扫描整个表的查询应当尽量避免，因为它们不是选择性的，也不可能长时间地驻留在内存里。这就要求从长期存储器里进行更多的检索。凡是在使用DBMS的时候，应当对多表联结主动加以管理，因为联结的类型可能极大地影响性能，尤其是需要联结整个表的那些联结。

类似地，包含视图和包含子查询的语句也应当加以审查。应当调整这些语句，以最有效的方式分解这些组件，可能会使性能显著提高。第6章讨论过一种技术，DBA可以用来调整应用软件的处理速度和磁盘空间利用率(比如重新索引、覆盖自动查询计划、改变数据块大小、在存储设备上再分配文件以及有效查询设计的指南)。DBA在向程序员和开发者推荐最有效的技术方面起着很重要的作用。

对数据库性能可能会受到影响的那些方面进行简要描述，有助于读者了解有效的数据库管理和调整的重要性。当DBA对由他负责的DBMS和应用软件有了较深的理解后，调整数据库性能的重要性就变得很明显了。希望本节关于数据库调整的简要描述，能够激起读者进一步学习更多的数据库产品，提高调整的能力的欲望。

本章小结

本章重点介绍了对数据进行管理的重要性。数据管理负责数据资源的总体管理，其功能包括开发保护与控制数据的过程，分解数据的权属与使用问题以及开发和维护全公司范围的数据定义与标准。另一方面，数据库管理功能则与直接管理一个或者多个数据库有关，包括DBMS的安装与升级、数据库的设计问题与技术问题，如安全性的强制实施、数据库性能以及备份与恢复等。在目前的业务环境下，面对既要快速建立高性能的系统又要保证数据质量的巨大压力，数据管理与数据库管理的作用也在不断地变化。达到这些期望的关键是开发一个企业的体系结构(ISA)，并制定合适的数据库规划。

对于数据安全性的威胁包括偶然的损失、偷窃与欺诈、私密性受损、数据完整性受损以及可用性受损等。要解决这些潜在的威胁需要一份全面的数据安全性规划，可以通过建立视图、授权规则、用户自定义过程和加密过程消除部分威胁。

数据库恢复与备份过程是另一类基本的数据管理活动。必须具备的基本恢复工具包括备份

工具、日志记录工具、检查点工具和恢复管理器。根据问题的类型，可能还需要后向恢复(回滚)或前向恢复(前滚)。

多用户环境中的并发性访问问题也必须加以考虑。DBMS必须确保数据库事务具有ACID特性，即原子性（atomic）、一致性（consistent）、隔离性（isolated）与持久性(durable)。必须选择合适的事务边界以便在可接受的性能上达到这些特性。倘若事务没有并发性控制，就有可能出现更新丢失的情况，这将影响数据完整性。可以采用包括共享锁与排他锁在内的加锁机制。在多用户环境下还可能出现死锁现象，这可以通过各种手段，包括应用两阶段加锁协议或者其他的死锁化解机制来加以管理。版本设置是并发性控制的一种理想措施。

维护高质量数据涉及五个方面：安全性策略与灾难恢复、人员控制、物理访问控制、维护控制以及数据保护与私密性。

数据词典往往是绝大多数关系数据库管理系统的系统目录的一部分，对数据词典和信息库的管理有助于DBA维护高质量的数据与高性能的数据库。信息库词典系统(IRDS)标准的建立，有利于开发信息库的信息，这些信息集成了来自DBMS、CASE工具和软件开发工具等多种来源的信息。

实现高效的数据管理是不容易的，有可能碰到上述所有方面的问题。一方面，对于面向对象开发方法和快速开发技术的日益重视，正在改变着数据管理的功能；另一方面，实现有效管理和数据库调整的优秀工具的可用性也越来越高。

本章复习

关键术语

数据管理	数据库管理	数据库安全性
授权规则	用户自定义过程	加密
生物设备	数据库恢复	备份工具
日志记录工具	事务	事务日志
数据库变更日志	前象	后象
检查点工具	恢复管理器	复原/重运行
事务边界	后向恢复（回滚）	前向恢复（前滚）
中止事务	数据库崩溃	并发性控制
不一致读问题	加锁	加锁级别（粒度）
共享锁(S锁或读锁)	排他锁（X锁或写锁）	死锁
死锁预防	两阶段加锁协议	死锁化解
版本设置	数据管家	数据词典
系统目录	信息库	信息库词典系统(IRDS)

复习问题

1. 定义以下术语：

a. 数据管理	b. 数据库管理	c. 数据管家
d. 信息库	e. 加锁	f. 版本设置
g. 死锁	h. 事务	i. 加密

2. 将下列术语及其定义匹配起来：

_____备份工具　　a. 保护数据免遭损坏或误用

_____生物设备	b. 异常或者中止事务的恢复
_____检查点工具	c. 描述所有数据库对象
_____数据库恢复	d. 自动地产生整个数据库的一个拷贝
_____数据库安全性	e. 后象应用
_____粒度	f. 可以分析你的签名
_____恢复管理器	g. 在受损后复原数据库
_____回滚	h. 在故障后复原数据库的DBMS模块
_____前滚	i. 数据库需要对事务加锁的级别
_____系统目录	j. 同步记录数据库状态

3. 比较和对比下列术语:

a. 数据管理; 数据库管理	b. 信息库; 数据词典
c. 死锁预防; 死锁化解	d. 后向恢复; 前向恢复
e. 主动型数据词典; 被动型数据词典	f. 乐观并发性控制; 悲观并发性控制
g. 共享锁; 排他锁	h. 前象; 后象
i. 两阶段加锁协议; 版本设置	j. 授权; 认证

4. 数据管家的功能是什么?

5. 简要描述典型的数据库系统生命周期的六个阶段。

6. 描述在当前业务环境下数据管理员与数据库管理员作用的变化。

7. 列举造成数据管理低效的四个常见问题。

8. 列举系统数据管理或数据管理员所必需的四种职业技能。列举项目数据管理或数据库管理员所必需的四种职业技能。

9. 列出并描述实现有效的数据库管理所应当具备的11种功能。

10. 为了快速交付高质量、功能强大的系统，可以在传统的数据库开发生命周期每一阶段的数据管理中做哪些改变?

11. 列出并讨论可能会对数据安全性构成威胁的五个方面。

12. 阐述如何创建提高数据安全性的视图，解释为什么人们不应该完全依靠视图来强制实施数据安全性。

13. 列出并简要说明如何应用完整性控制来保障数据库安全性。

14. 认证模式与授权模式之间有什么区别?

15. 与悲观并发性控制相比，乐观并发性控制有哪些好处?

16. 共享锁与排他锁之间的区别是什么?

17. 死锁预防与死锁化解之间有什么区别?

18. 简要描述数据库备份与恢复所需要的四种DBMS工具。

19. 什么是事务完整性? 为什么它如此重要?

20. 列出并描述数据库故障的四种常见类型。

21. 列出并描述为了获得高质量数据所应当解决的五个方面。

22. 什么是信息库词典系统(IRDS)?

23. 列出并简要说明数据库事务的ACID特性。

24. 阐述加密的两种常见形式。

25. 概述六种数据库保护过程。

问题和练习

对仓库管理职员的授权

	库存记录	应收款记录	工资单记录	顾客记录
读取				
插入				
修改				
删除				

对库存记录的授权

	推销员	会计出纳	仓库管理职员	木工
读取				
插入				
修改				
删除				

1. 根据下列假设为松谷家具公司填写前面的两张表格。
   (1) 推销员、经理和木工可以读取库存记录，但不能对这些记录执行其他操作。
   (2) 会计出纳可以读取或更新（插入、修改、删除）应收款记录和顾客记录。
   (3) 仓库管理职员可以读取或更新（修改、删除）库存记录。他们不可以查看应收款记录或工资单。他们可以读取但不能修改顾客记录（输入Y代表yes、N代表no）。
2. 以下列出五种恢复技术，针对下列的每种情形，确定最适宜使用哪种恢复技术。
   (1) 后向恢复　　(2) 前向恢复(从最近检查点)
   (3) 前向恢复(采用数据库备份拷贝)　　(4) 重新处理事务
   (5) 切换
   a. 当用户正在进入某个事务时，出现了一次电话断线。
   b. 在正常操作时，出现某个磁盘驱动器故障。
   c. 由于突然的暴风雨引起的电源故障。
   d. 学生学费金额输入不正确，并且已经显示出来。该错误在几个星期内都没有发现。
   e. 当数据库崩溃时，数据录入员已经在进行完全的数据库备份之后又用两个小时输入事务。这时发现自从备份以后，数据库的日志记录工具根本没有工作。
3. Whitlock百货店在局域网文件服务器上运行多用户的DBMS。但是，该DBMS现在还不能实现并发性控制。Whitlock的某个顾客账户有一笔$250.00的欠账应付，此时有三笔与该顾客有关的事务要同时处理:
   (1) 支付$250.00　　(2) 赊购$100.00　　(3) 退货(信用)$50.00
   这三笔事务都要在账户余额为$250.00时（即在其他事务完成之前）读取该客户的记录。更新后的顾客记录以上述顺序返回数据库。回答以下问题:
   (1) 在最后一笔事务完成之后，该顾客的账户将会有多少余额?
   (2) 在这三笔事务完成之后，该顾客的账户应该有多少余额?
4. 针对下面描述的每种情况，指出最适合使用下述哪种安全性措施:
   (1) 授权规则　　(2) 加密　　(3) 认证模式
   a. 一家全国性的经纪公司采用电子资金转账(EFT)系统在各地传输敏感的财务数据。
   b. 某个组织正在建设一座基于计算机的培训中心离散站点。该组织希望只有授权的员工能够访问该站点。由于每个员工并非经常使用该中心，所以不希望为员工配备中的钥匙。
   c. 某个制造企业采用简单的口令系统来保护其数据库，但后来发现需要一个更加全面的系统来为不同的用户授予不同的权限（如读取但不能创建或更新）。
   d. 一所大学发现有大量非授权用户利用从合法用户那里窃取的口令访问各种文件和数据库，这让管理人员很头痛。

5. Metro Marketers公司打算建造一个数据仓库，用来存储顾客信息以进行市场分析。所建造的数据仓库要比他们原先所要求的容量更大、处理能力更强，因此他们考虑分别以Oracle和Red Brick作为其数据库和数据仓库产品。作为实施计划的一部分，Metro公司开始配置数据管理职能。目前，数据管理员这个职位有四位候选人：

a. Monica Lopez,拥有五年Oracle数据库管理经验的高级数据库管理员，曾为一家国际银行管理过财务数据库，但没有任何数据仓库方面的经验。

b. Gerald Bruester,拥有六年Informix数据库管理经验的高级数据库管理员，曾为一家在《财富》杂志上排名前1000名的食品生产企业管理过面向市场的数据库。Gerald在去年12个月里参加了多个数据仓库研讨班，对数据仓库有着浓厚的兴趣。

c. Jim Reedy，目前是Metro Marketers的项目经理。Jim非常熟悉Metro现行的系统环境，并获得其同事的好感。他参与过Metro当前的数据库系统的建设，但还没有任何数据仓库方面的经验。

d. Marie Weber，拥有两年经验的数据仓库管理员，曾经利用基于Red Brick的应用软件为一家汽车保险公司跟踪过事故信息。

基于上述信息，对这四位数据管理员职位候选人进行排序，并陈述排序的理由。

6. 参照问题和练习5，对这四位数据仓库管理员职位候选人进行排序，并陈述排序的理由。

7. 参照问题和练习5，对这四位数据库管理员职位候选人进行排序，并陈述排序的理由。

8. 倘若你应聘某个数据库管理员的工作，同时发现数据库用户在每天早晨上班时采用一个公共的口令进入数据库，你会作何考虑？你又得知他们甚至在离开机器时仍然让工作站整天连接到工作站上，你又作何考虑？

9. 某个组织拥有一个带三台磁盘设备的数据库服务器，会计和工资软件共享其中的一个磁盘设备，但总是发现有性能的问题。请你来研究这个问题并调整数据库。在减少I/O争用方面，你会提出什么建议？

10. 假设你在某个拥有遍布全球的分布式数据库的组织里担任数据库管理员。你分析了数据库的性能，在分析中你发现，所有的区域性月销售报表都是在该公司总部完成的。公司运营分成五个区域：美国东部、美国西部、加拿大、南美洲和墨西哥。在每个区域总部的服务器里都保存着各自区域的全部数据。你会如何改善制作月销售报表所需的时间？

11. 当你阅读本书时，假设某个流行的DBMS（如微软的Access）发行了新的版本，你要从某一旧版本升级安装该产品。此时你会面对什么样的问题？你是否需要转换数据库？为什么？假设数据库已经进行过转换，它是比前一版本的数据库占用的空间更多还是更少？这又是为什么？为前一版本编写的查询和程序现在是否仍然能够工作？为什么？能否在同一台计算机上同时运行两种版本？为什么？

12. 回顾第6章问题和练习5的答案，是否有机会为该数据库创建某个域？如果可以，请描述这个域。此外，是否有可能为控制这个数据库的完整性定义一个断言？

13. 回顾第6章问题和练习5的答案。

a. 概述为现有的顾客补发一张新卡这一业务事务的步骤。试为一个或多个需要创建ACID事务的数据库事务定义边界，并检验答案的正确性。

b. 概述为新顾客发一张新卡这一业务事务的步骤。试为一个或多个需要创建ACID事务的数据库事务定义边界，并检验答案的正确性。

应用练习

1. 参观某个已经实施数据库方法的组织，并评估如下几个方面：

a. 系统数据管理与项目数据管理在该组织中的设置。

b. 系统数据管理与项目数据管理的职责分配。

c. 数据管理负责人的背景与经验。

d. 信息库的状态与利用(被动型、主动型设计或者主动型运作)。

2. 参观某个已经实施数据库方法的组织，并与接触过灾难恢复计划的MIS部门员工面谈，了解该组织的灾难恢复计划。在面谈前，先仔细考虑该组织会遇到各种灾难的相对可能性。例如，该地区是否会有地震或其他自然灾难？厂房有可能受到何种破坏？使用这个系统的员工具有怎样的背景、接受过何种培训？找出该组织的灾难恢复计划，并特别询问你所能想到的各种潜在的问题。

3. 参观某个已经实施数据库方法的组织，通过面谈了解他们平时所采取的安全性措施。对如下方面进行评估:

a. 数据库安全性措施　　b. 网络安全性措施　　c. 操作系统安全性措施

d. 厂房安全性措施　　e. 个人安全性措施

4. 确定某个处理大量零星突发性数据负载的组织。例如，已实施数据仓库的组织，因为它既然使用数据仓库，就可能会有较大的数据负载。试确定该组织在其容量规划中是采用什么措施来处理这样大量的数据负载的?

5. 随着时间推移，数据库一般总是越来越大，因为新的事务数据不断增加。请至少与三家数据库不断增大的公司面谈，并确定他们决定清除或者归档旧数据的准则与过程。确定它们隔多久清除数据，清除的又是哪些类型数据？同时找出每个组织归档的数据，这些数据又保留多长时间?

6. 访问http://tpc.org网站。选择一篇与本章内容有关的技术论文，并概述至少涉及本章的一个主题的新思想，将其写成一篇报告。

参考文献

Bernstein, P.A. 1996. "The Repository:A Modern Vision." *Database Programming & Design* 9:12 (December): 28-35.

Celko, J. 1992. "An Introduction to Concurrency Control." *DBMS* 5:9 (September): 70-83.

Descollonges, M. 1993. "Concurrency for Complex Processing." *Database Programming & Design* 6:1 (January): 66-71.

Dowgiallo, E., H.Fosdick, Y.Lirov, A.Langer, T.Quinlan,and C.Young. 1997. "DBA of the Future." *Database Programming & Design* (June):33-41.

Fernandez,E.B., R.C.Summers, and C.Wood. 1981. *Database Security and Integrity*. Reading, MA:Addison-Wesley.

Inmon, W.H. 1999. "Data Warehouse Administration." www.billinmon.com/library/other /dwadmin.asp, verified July 25, 2001.

Lefkovitz, H.C. 1985. *Proposed American National Standards Information Resource Dictionary System*. Wellesley, MA:QED Information Sciences.

Loney, K. 2000. "Protecting Your Database." *Oracle Magazine* (May/June) : 101-106.

Moriarty, T. 1996. "Better Business Practices." *Database Programming & Design* 9: 7 (September.) : 59-61.

Rodgers, U. 1989. "Multiuser DBMS Under UNIX." *Database Programming & Design* 2:10 (October.) : 30-37.

Quinlan,T. 1996. "Time to Reengineer the DBA?" *Database Programming & Design* 9: 3 (March) : 29-34.

Zachman, J.A. 1987. "A Framework for Information Systems Architecture." *IBM Systems Journal* 26 : 3 (March) : 276-292.

Zachman, J.A. 1997. "Enterprise Architecture:The Issue of the Century." *Database Programming & Design* 10: 3 (March) : 44-53.

Web资源

- http://developer.netscape.com/docs/manuals/security/pkin/contents.htm 该站点有各种现代加密方法的详尽说明，除包括本章讨论过的方法外，还有许多其他的方法。
- http://www.bionetrix.com, http://www.identicator.com, http://www.keyware.com BioNetrix Systems、Identicator Technology和Keyware Technologies都是生物技术的先驱，它们的Web站点阐述了各种可以用于不同应用领域的生物设备。
- http://www.isi.edu/gost/brian/security/kerberos.html 这份文档是用户认证的Kerberos方法的一个指南。
- http://www.abanet.org/scitech/ec/isc/dsg-tutorial.html 这份数字签名的详细说明是American Bar Association Section of Science and Technology以及Information Security Committee准备的。
- http://tpc.org 创办非盈利组织TPC的宗旨是制定事务处理与数据库基准，并向业界传播客观、可验证的TPC性能数据。这是一个优秀的站点，通过阅读数据库基准的技术论文，可以学到许多与评估的DBMS和数据库设计有关的知识。

项目案例：山景社区医院

在第2章结尾，我们已经看到山景社区医院特别研究小组正在开发一个长期的业务规划。这个小组的成员有Heller先生、Lopez先生、Jefferson博士及一个顾问。他们尝试确定一个规划，以实现医院的提供高质量医疗、成本控制和加入诸如Browne博士的预防衰老医学部等新型服务。

信息系统主管Heller先生经常阅读《Computerworld》、《Health Management Technology》、《Healthcare Information》等期刊、杂志。通过阅读，他了解到在大城市的大型医院里实现基于计算机的病人记录(CPR)系统这一趋势，并且想要知道山景社区医院是否现在就应当规划以便未来实现CPR。通过阅读，他归纳出关于CPR存在两种看法。直到最近，CPR被当成替代现有的病历记录的手工系统的手段，只不过术语的用法改变了。现在有许多论文把CPR看成是一个集成的病人医疗信息系统，该系统可以由医院管理员、医生、护理师和医疗管理组织访问。CPR系统可以很好地集成来自病床终端、实验室系统和管理性系统等不同系统的医疗信息。集成性和可访问性达到如此程度的系统不可能在几年内面世。

这类系统在收集和利用医疗信息的方法上会有显著的改变。正在建立的CPR将向医生提供一个基础结构，使他能够访问其病人的所有医疗记录，甚至是分布在各种医生、医院、实验室等许多位置的记录以及保险记录。最后，倘若CPR成功集成的话，连病人日常生活中的一些健康信息也可以加以利用。

项目分析

1) CPR是否可以帮助山景社区医院达到其高质量医疗和成本控制的目标？举例说明其理由。

2) 确立病历的安全性始终是医疗记录的主要需求。一旦决定CPR系统可以由社区医生、实验室、医疗组织访问，山景社区医院会遇到哪些数据安全性问题？

3) 美国医疗消费中70%以上是用于临床诊断。CPR以什么方式帮助临床诊断，从而帮助控制医疗消费？

4) 如果CPR系统在几年内不能广泛使用，Heller先生是否现在就要致力于规划这类系统？为什么？

5) 集成的CPR是否是适合山景社区医院采用，或者它更适合于更大型的社区和更大型的医院？在山景社区医院中采用CPR系统之前，是否要达到某种环境要求？例如，是否医院的医生都要同意安装CPR？

项目练习

1) 倘若在山景社区医院安装一个CPR，试列举需要授权使用该系统的所有可能的用户。包括医院外希望包含在集成CPR系统里的用户团体。

2) 对于上列用户，试指出他们所能具备的权限（读取、插入、删除、修改）。

3) 这时Heller先生开始着手计划本章所描述的典型数据管理任务。简要概述你认为他应当考虑的问题。

4) CPR系统并没有回答诸如数据所有权、认证和资格性之类的法律问题。如果医院决定实现CPR系统，那么为了处理这些问题，他们应当采取哪些步骤？如果这些问题没有解决，是否仍然有办法实现CPR？这些问题中是否有哪个问题不如其他问题重要？为什么？

第13章　分布式数据库

13.1　学习目标

学完本章后，读者应该具备以下能力：

- 定义下列关键术语：分布式数据库、分散型数据库、位置透明性、本地自治、同步分布式数据库、异步分布式数据库、本地事务、全局事务、复制透明性、事务管理器、故障透明性、提交协议、两阶段提交、并发透明性、盖时间戳和半联结。
- 阐述在企业中作为应用分布式数据库的驱动力的业务条件。
- 描述分布式数据库环境多样化的优点。
- 阐述分布式数据库的好处与风险。
- 阐述分布式数据库设计的四种策略，每种策略中的可选项以及选择策略时应考虑的因素。
- 说明作为分布式数据库设计的三种主要手段的同步数据复制、异步数据复制和数据分区的好处。
- 概述在分布式数据库中处理查询所涉及的步骤，以及优化分布式查询处理常用的几种手段。
- 阐述几种分布式数据库管理系统的优点。

13.2　引言

若组织在地理上分布于不同位置，它可以选择将数据库存放在某个中央计算机里，或是将其分布于各个本地计算机上（或者两者的组合）。**分布式数据库**（distributed database）就是物理上虽然分布在多个地点的计算机内，但通过数据通信链连接起来的逻辑数据库。我们强调的是，分布式数据库确实是一个数据库，而不是松散的文件集合。分布式数据库仍然作为一种企业资源进行集中式管理，只不过提供局部灵活性和定制方案。网络必须允许用户共享数据。因此，在A地的用户（或程序）必须能够访问（或者更新）放在B地的数据。一个分布式系统的站点可能遍布于某个很大区域（比如美国或全世界），也可能仅分布在一个很小的区域中（比如一栋大楼或一所校园）。计算机也可以是从微型机到大型机甚至超级计算机的各种计算机。

分布式数据库需要有多个数据库管理系统，每个远程站点运行一个。不同的分布式数据库环境的区别就在于，这些不同的DBMS合作工作的程度，以及是否存在某个主站点来协调涉及多个站点数据的请求。

重要的是把分布式数据库与分散型数据库区别开来。一个**分散型数据库**（decentralized database）也是存放在多个位置的计算机上的。然而，这些计算机并没有通过网络彼此互连，也没有数据库软件使得数据仿佛是处在一个逻辑数据库里。因此，各个站点上的用户不能共享数据。最好将分散型数据库看成是许多独立数据库的汇总，而不是地理上分布的单一数据库。

各种业务条件都鼓励采用分布式数据库：

- *业务单位的分布与自治*　现代组织机构的分公司、部门与设施通常都是地理上分散(甚至跨国)分布的。一般每个单位都有权创建自己的信息系统，而且每个单位都需要可以对之进行控制的本地数据。商业并购经常会产生这样的环境。

- *数据共享* 即使不太复杂的业务决策都会要求共享各业务单位上的数据，因此，必须能很方便地根据需要合并各本地数据库上的数据。
- *数据通信成本与可靠性* 通过某个通信网络传输大量的数据或者从远程数据源处理大量的事务，其成本是很高的。通常，比较经济的做法是，尽可能将所需要的数据和应用软件本地化。此外，依赖通信网络会存在一定的风险，因此，在本地保存数据的拷贝或片段是实现快速访问该组织数据的一种可靠方法。
- *多应用软件供货商环境* 今天有许多组织从不同的供货商处采购成套的应用软件。每个“最佳解决方案”软件包都是专门为其固有的数据库而设计的，并可能与不同的数据库管理系统一起工作。人们可以定义一个分布式数据库，它提供的功能远远超出单独的应用软件。
- *数据库恢复* 将数据复制到单独的计算机上是使受损数据库迅速复原，并使用户在主站点正在恢复时仍可以访问数据的一种比较保险的策略。而将数据复制到多个计算机站点正是分布式数据库的一种自然形式。
- *满意的事务与分析处理* 正如第11章所介绍的，数据库管理的需求随着OLTP和OLAP应用软件的不同而有所区别。即使支持每种类型应用软件的两个数据库的数据相同也是如此。分布式数据库技术有助于跨OLTP、OLAP平台实现数据的同步。

创建分布式数据库的功能，至少从20世纪80年代开始就已具备。正如所期待的，存在各种分布式数据库备选方案(见Bell和Grimson,1992)。图13-1概括了分布式数据库环境的范围。这些环境简要阐述如下：

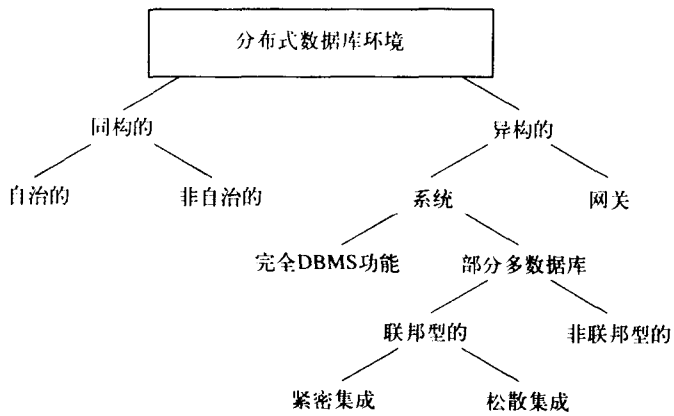


图13-1　分布式数据库环境(摘自Bell和Grimson, 1992)

I *同构的* 每个结点采用相同的DBMS。

A *自治的* 每个DBMS独立工作，来回传递消息，以共享数据更新。

B *非自治的* 某个中央或主DBMS协调数据库在结点上的访问和更新。

II *异构的* 每个结点可以采用不同的DBMS。

A *系统* 支持一个逻辑数据库的某些或全部功能。

i *完全DBMS功能* 正如本章后面将要讨论的，支持分布式数据库的全部功能。

ii *部分多数据库* 正如本章后面将要讨论的，支持分布式数据库的一部分功能。

a *联邦型的* 对于惟一的数据请求，支持本地数据库。

- *松散集成* 对于每个本地数据库存在多种模式，而每个本地DBMS必须与所有的本地模式通信。
- *紧密集成* 存在一种全局模式，该模式定义了所有本地数据库上的所有数据。

h 非联邦型的 要求所有的访问都需通过某个中央的协调模块。

B 网关 创建通向其他数据库的简单路径，但没有单一逻辑数据库的优势。

图13-2描绘了一个同构分布式数据库环境。这个环境可以通过下列特征(与上述的非自治范畴相关)加以定义：

- 数据分布在所有的结点上。
- 每个位置使用相同的DBMS。
- 所有的数据都由该分布式DBMS管理(因此不存在任何专门的本地数据)。
- 所有用户都通过一个全局模式或数据库定义来访问数据库。
- 该全局模式是所有本地的数据库模式的合并。

绝大多数组织机构都很难强制实施某个同构环境，而异构环境更加难以管理。

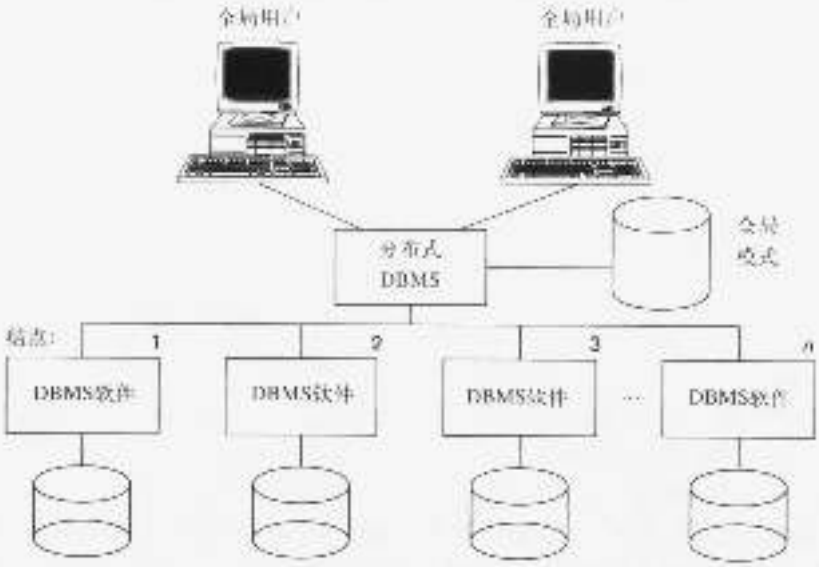


图13-2 同构分布式数据库环境(摘自Bell和Grimson,1992)

正如以上所列出的那样，异构分布式数据库环境存在很多变体。然而，在本章的后面，将采用下列特征定义异构分布式数据库环境（如图13-3所示）：

- 数据分布在所有的结点上。
- 每个结点都可以使用不同的DBMS。
- 某些用户只要求对数据库进行本地访问，这可以只采用本地的DBMS和模式实现。
- 存在一种全局模式，它允许本地用户访问远程数据。

目标与权衡

分布式数据库的一个主要目标就是使用户在不同的位置可以很容易地访问数据。为了达到这一目标，分布式数据库系统必须提供所谓的**位置透明性**（location transparency），也就是说，用户（或用户程序）为了查询或更新而使用数据时，无须知道数据的位置。来自任何站点的检索或更新数据的所有请求，都会由系统自动地转发到与处理该请求有关的站点。理想情况下，用户不必担心数据的分布性，网络里的所有数据就仿佛处于一个站点上的单一逻辑数据库中。在这种理想情况下，只用一个查询就可以将处在多个站点的表里的数据联结起来，就像数据都在一个站点里一样。

分布式数据库的第二个目标是**本地自治**（local autonomy），也就是在与其他结点连接失败

时，独立管理和操作本地数据库的能力（见Date，1995）。利用本地自治，每个站点都有能力控制本地数据，管理安全性，记录事务并在出现本地故障时进行恢复，还能在任何中央或协调站点无法正常运行时，由本地用户对本地用户进行全面访问。此时，数据是本地持有和管理的，甚至能从远程站点访问这些数据。这意味着不需要依靠中央站点。

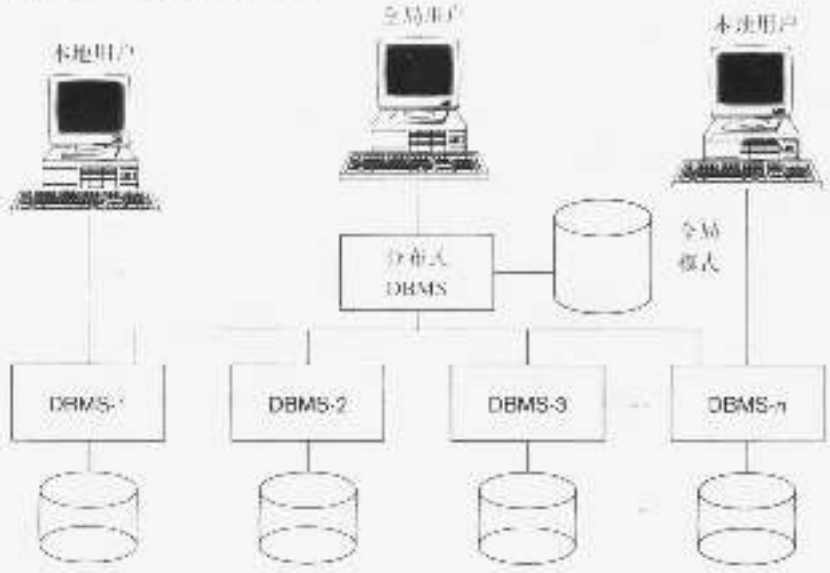


图13-3　异构分布式数据库环境(摘自Bell和Grimson，1992)

设计分布式数据库环境时需要作出的一个重要权衡是，到底使用同步分布式技术还是异步分布式技术？采用**同步分布式数据库**（synchronous distributed database）技术，网络上的所有数据一直保持为最新的数据，这样任何站点上的用户在任何时候都能够访问网络上任何位置的数据，并总是得出相同的结果。采用同步技术，如果某个数据项的任何拷贝在网络的某个地方被更新，则同样的更新会立即应用到其他拷贝上，或者被中止。同步技术确保了数据完整性，并使得获得数据最新拷贝的存放地点的复杂性降到最低。同步技术的响应时间可能不尽如人意，因为分布式DBMS要花费相当多时间来检查更新是否正确而完全地在网络上进行传播。

异步分布式数据库（asynchronous distributed database）技术在其他的结点里保存复制数据的拷贝，这样本地服务器无须通过网络就能访问数据。采用异步技术时，在远程数据库上传播数据更新时通常会有一定的延迟，因此，有一定程度的暂时不一致性是容许的。异步技术具有可以接受的响应时间，因为更新在本地发生，而数据拷贝是成批地或者以事先预定的间隔进行同步的。但要确保结点上的数据完整性和一致性处于正确的水平，其计划和设计是比较复杂的。

与中央数据库相比，两种分布式数据库都有许多优势，最重要的优势有以下一些：

- **提高可靠性和可用性**　当集中式系统发生故障时，数据库就不能供所有用户使用。然而，一个分布式系统即使在某个部件失效时，都将会以稍低一些的水平继续提供其功能。可靠性与可用性取决于数据是如何分布的。（这将在下一节讨论。）
- **本地控制**　将数据进行分布有利于本地的工作小组对“它们的”数据有更多的控制权力，这促进了对数据完整性和管理的改进。同时，用户在必要时能够访问非本地的数据。可以为本地站点选择硬件，使它与本地而不是全局的数据处理工作匹配。
- **模块性增长**　假设组织扩展到某个新的地区或者增加了某个新的工作机构，那么向其分布式网络增加一个本地计算机以及与其相关的数据往往要比扩展一个大型的中央计算机更

加容易、更加经济。而且，破坏现有的用户的机会也会比修改或扩展中央计算机系统的情况少得多。

- 较低的通信成本 采用分布式系统，数据可以存放在与用户距离较近的地点。与中央系统相比较，这种方式能降低通信成本。
- 较快的响应 根据数据的分布情况，在某个站点的用户的大多数数据请求是能够利用该站点所存放的数据来满足的。这样可以加快查询处理的速度，因为通信和中央计算机的延迟能够达到最小。也可以把复杂的查询分解成能够在几个站点并行处理的子查询，以提供更快的响应。

分布式数据库系统也会面临一定的成本和不利因素:

- 软件成本和复杂性 分布式数据库环境需要比较复杂的软件(尤其是DBMS)。

我们将在本章后面讨论这种软件。

- 处理开销 各个站点必须交换消息并执行附加的计算，以确保不同站点的数据之间真正协调。
- 数据完整性 增加复杂性以及协调需求的一个副作用是，数据完整性的不正确更新和其他问题会暴露出来。
- 响应慢 如果数据按其使用没有正确分布，或者查询没有正确地编写，那么对于数据请求的响应可能就特别慢。这些问题将在本章稍后的内容中加以讨论。

13.3 数据库实施分布式处理的策略

一个数据库应当如何在网络的站点（或结点）之间进行分布呢？我们在第6章里讨论过物理数据库设计的重要问题，其中介绍了一个分析过程，用来评估可选的分布式策略。在本章中，我们注意到，分布式数据库存在四种基本策略:

1) 数据复制。

2) 水平分割。

3) 垂直分割。

4) 以上的组合。

我们以关系数据库为例来说明这些方法。同样的概念也可以应用（稍许有些变化）到其他的数据模型，比如层次数据模型或网状数据模型。

假设一家银行在某个州有许多支行。该银行数据库的基本关系之一是Customer关系。图13-4显示了该关系的一个简化版本。为简单起见，关系中的样本数据只应用到两个支行(Lakeview和Valley)。这个关系的主键是账号(Acct_Number)。顾客开户的支行名称为Branch_Name（因此假设他们是在该支行完成其大部分事务的)。

Acct_Number	Customer_Name	Branch_Name	Balance
200	Jones	Lakeview	1000
324	Smith	Valley	250
153	Gray	Valley	38
426	Dorman	Lakeview	796
500	Green	Valley	168
683	McIntyre	Lakeview	1500
252	Elmore	Lakeview	330

图13-4 银行的Customer关系

13.3.1 数据复制

数据分布一个日益流行的可选项是每两个（或多个）站点就存放一份单独的数据库拷贝。复制允许IS组织将数据库从中央主机移动到更加接近用户的、花费较少的部门或专用位置服务器上（见Koop,1995）。复制既能采用同步分布式数据库技术，也能采用异步分布式数据库技术，不过异步技术在复制环境里更为常见。例如，图13-4中的Customer关系可以存储在Lakeview或者Valley中。倘若在每一个站点都存放一个拷贝，那么我们就进行了完全复制（除非只有少数相关的数据库，否则这也是不实际的）。

数据复制有五个好处:

1) *可靠性*　如果包含该关系（或数据库）的一个站点发生故障，那么无须网络传输延迟也能够在其他站点上找到一个拷贝。而且，所使用的拷贝完全可以在事务发生时快速进行更新。一旦它们归还服务器，未使用的结点也会得到更新。

2) *快速响应*　具有完全拷贝的每个站点能在本地处理查询，因而查询处理的速度很快。

3) *可避免复杂的分布式事务完整性例程*　复制后的数据库通常按预定的时间间隔刷新，因此，数据库拷贝的同步要求比较宽松时，可以采用各种复制方式。

4) *结点无须耦合*　每个事务在进行时无须在网络上协调。因此，如果结点故障、忙碌或不能连通时（比如在移动式个人计算机中），事务可以在用户希望的时候进行处理。代替更新的实时同步，某个幕后的过程在协调着所有的数据拷贝。

5) *降低高峰时间的网络流量*　数据更新经常会发生在业务高峰时间段内，也就是网络流最高、快速响应的需求也最大的时候。具有延迟更新拷贝数据的复制，就可以将发送更新的网络流量发送到其他结点，在非高峰时间段处理。

复制有两个主要的不利因素:

1) *存储需求*　具有完全拷贝的每个站点必须具备和数据集中存放一样多的存储容量。每个拷贝都需要存储空间（其成本是不断下降的），而更新每个结点的拷贝也需要处理时间。

2) *更新的复杂性与成本*　每当更新一个关系时，拥有拷贝的每个站点也必须（最终）被更新。正如以后在讲述提交协议时会看到的，接近实时的同步更新需要小心地进行协调。

基于以上理由，数据复制在绝大多数处理请求仅为读取、数据相对静态的场合（比如目录、电话簿、列车时刻表等）很受欢迎。CD-ROM和DVD有望成为一种复制数据库的经济实用的载体。复制是用于“非捆绑型数据”的，即一个位置上的数据不需要实时更新别的位置上保存的数据（见Thé,1994）。在这些应用软件中，数据最终会进行同步、尽量快是比较实用的。复制对于联机应用软件，比如机票预约、自动提款机事务以及其他金融活动是不可行的，因为其中每一个用户所需要的都是相同的非共享数据。

1. *快照复制*

更新数据拷贝有各种模式。有些应用软件，如决策支持和数据仓库或挖掘应用（它们不需要当前数据），只利用简单的表拷贝或周期性快照。其工作过程如下。假定多个站点正在更新相同的数据。首先，把来自所有要复制的站点的更新周期性地汇总在某个主机或主站点上，其中所有的更新形成了所有变更的合并记录。利用某个分布式DBMS，把这些变更清单汇总在快照日志里，该日志是要进入快照的记录的行标识符的一个表。然后，在此主站点，产生该数据库复制部分的只读快照。最后，将此快照发送到存在拷贝的每个站点（即其他站点“订阅”主站点所拥有的数据）。这也称为数据库的完全刷新（见Edelstein, 1995a; Buretta, 1997）。另一种情况是，自从上一次快照以来，只发送变更的页面，这称为微分刷新或增量刷新。在这种情况下，每个复制表的快照日志是与关联的库联结在一起的，以形成需要发送到复制站点的有变更行的集合。

有些形式的复制管理允许数据的所有权动态变化，其中更新要复制数据的权限是一个站点一个站点地移动的，但在任何时刻都仅有一个站点有更新权限。动态所有权比较适合于业务活动随着时间而变动的情况，或者其中数据处理遵循由其他数据库服务器所支持的业务单位上工作流的场合。

复制管理的最后一种形式是允许数据所有权的共享。共享更新会引起管理站点上更新冲突管理的严重问题。例如，当两个支行的提款机试图同时更新顾客的地址时会怎么样？异步技术允许冲突暂时存在。只要更新对于业务操作并不关键，而且这类冲突可以被检测出来，并在造成实际业务问题之前加以解决，就依然可以使用这种方法。

完成快照刷新的成本取决于该快照是简单还是复杂。所谓简单快照就是仅仅参照一个表的全部或一部分。复杂的快照涉及多个表，通常来自涉及联结的事务（比如输入顾客订单和相关的产品项目）。在某些分布式DBMS里，简单快照可以通过微分型刷新来处理，而复杂快照却需要比较费时的完全刷新。有些DBMS仅仅支持简单快照。

2. 接近实时复制

为了满足接近实时的需求，为每个已完成事务存储和转发消息可以通过网络进行广播，通知所有的结点尽快地更新数据，在发端结点的数据库更新之前，并不强迫初始结点的某个配置(就像下面将要讨论的利用协调的提交协议的情形，见Schussel，1994)。产生此类消息的一个办法是使用触发器(在第8章讨论过)。触发器可能存放在每个本地数据库里，每当更新了一段复制数据，触发器便执行相应的更新命令以响应远程的数据库复制(见Edelstein，1993)。采用触发器，每一个数据库更新事件都可以对程序和用户单独、透明地进行处理。如果网络与某个结点的链接断开或结点繁忙，则通知结点更新其数据库的这些消息就被放入一个队列等待处理。

3. 下拉式复制

以上对于复制品同步化所阐述的模式全都是下推式策略的例子。也存在下拉式的策略。在下拉式策略里，当更新本地数据库时，控制的是目标结点而非原始结点。采用下拉式策略，本地数据库确定何时需要刷新，并请求一个快照或者清空某个更新消息队列。下拉式策略的好处是，它所要求的是本地站点控制，并能处理更新。从而，同步化很少混乱，仅在每个站点都要求时才会出现，而并非是主站点自认为最好更新的时候。

4. 复制的数据库完整性

无论是周期性复制还是接近实时的复制，都需要力求其分布的、复制的数据库是一致的。无论是延迟还是接近实时，管理复制数据库的DBMS仍然必须确保数据库的完整性。决策支持应用软件允许基于表-表进行同步，而接近实时应用软件却要求事务-事务之间的同步。然而，在这两种情况下，DBMS都必须确保每个应用软件需求的拷贝都同步。

采用复制数据库处理更新的难度也取决于可能进行更新的结点的数目(见Froemming, 1996)。在单一更新者环境里，经常是通过周期性地发送要更新的数据库片段的只读数据库快照到非更新者结点来处理的。这时，对于只读站点，多重更新的效果可以有效地批量化。比如产品目录、价格清单以及汽车销售力的其他参考数据就属于这种情况。在有多个更新者的环境里，最明显的问题是数据冲突。在每个独立运行更新的结点都试图在同一时间更新同样的数据时，就可能会遇到数据冲突。此时，DBMS必须包括一种机制来检测和处理数据冲突。例如，DBMS必须规定，如果结点在处理时发生冲突，那么该处理应当挂起，直到数据冲突被化解为止。

5. 何时使用复制

复制是否是分布式数据库可行的设计选项，这取决于以下几个因素（见Froemming，1996)：

1) 数据及时性 容许过期数据（无论是几秒钟还是几小时）的应用软件是复制的最佳候

选项。

.2) DBMS功能　DBMS一个重要功能是，它是否支持需要参照多个结点数据的查询？如果不支持，那么复制就是比下一节要讲的分割模式更好的备选项。

3) 性能内涵　复制意味着每个结点周期性地进行刷新。当进行刷新时，分布的结点可能在非常忙碌地处理大量的更新。如果刷新的出现是由事件触发的（例如，积累了一定数量的变更时候），那么当远程结点正在忙于完成本地工作时，刷新可能就出现一次。

4) 网络中的异构性　如果不同的结点使用不同的操作系统、DBMS或者更常见的是，使用不同的数据库设计时，复制可能会比较复杂。将变更从一个站点映射到n个站点可能意味着有n个不同的例程用来把发端结点的变更转化成其他结点处理可接受的模式。

5) 通信网络功能　数据通信网络的传输速度和容量可能会禁止非常大的表的频繁完全刷新。然而，复制根本不需要专用的通信连接。因此，廉价的共享网络就能用于数据库快照的传输。

13.3.2　水平分割

利用水平分割（见第6章对各种形式的表分割的描述），可以把一个表（或关系）的某些行放入一个站点的基关系中，而将其他行放入别的站点的基关系中。一般说来，一个关系的行是分布于许多站点的。

图13-5显示了将Customer关系进行水平分割的结果。每一行现在都位于其开户支行。如果顾客的事务通常在开户支行完成，那么事务就可以在本地进行处理，响应时间也最少。当顾客在别的支行处理其事务，那么为了处理就要把该事务传输到开户支行，并把其响应传输回其处理支行（这就是使用ATM的常见模式）。如果顾客的使用模式改变（或许是由于搬家缘故），系统可以检测出这种变化，并动态地将记录移到处理绝大多数事务的位置。归纳起来，分布式数据库的水平分割具有四大好处:

1) 效率　数据存放在最接近它们使用地的地方，并与其他用户或应用软件所使用的数据相分离。

2) 本地优化　数据可以存储起来，以优化本地访问的性能。

3) 安全性　在某个站点与使用无关的数据不可用。

4) 查询方便　很容易将各水平分割中的数据组合起来，因为各分割中的数据通过并操作就可以合并起来。

Acct_Number	Customer_Name	Branch_Name	Balance
200	Jones	Lakeview	1000
426	Dorman	Lakeview	796
683	Mclntyre	Lakeview	1500
252	Elmore	Lakeview	330

a) Lakeview支行

Acct_Number	Customer_Name	Branch_Name	Balance
324	Smith	Valley	250
153	Gray	Valley	38
500	Green	Valley	168

b) Valley支行

图13-5　水平分割

因此，当组织功能为分布式的时候，经常采用水平分割，但每个站点仅仅关心实体事例（频繁地基于地理位置）的一个子集。

水平分割也有两个不利因素:

1) *不一致的访问速度* 当要求使用几个分割中的数据时，其访问时间可能与仅仅完成本地数据访问的速度相去甚远。

2) *备份脆弱性* 由于数据没有复制，所以一旦数据在一个站点变得无法访问或遭到破坏，则不能切换到其他存在拷贝的站点；倘若没有在每个站点完成真正的备份，则数据就会丢失。

13.3.3 垂直分割

采用垂直分割（请参见第6章），某个关系的一些列被投影到一个站点的基关系，而其他列则被投影到另一些站点的基关系（更常见的是，列被投影到几个站点上）。每个站点的关系必须共享一个公共域以便能够重新构造原始表。

为了说明垂直分割，我们利用图13-6所示的某个制造公司应用软件。图13-7显示了以Part_Number为主键的Part关系。其中一部分数据供制造部使用，而另一部分则主要由工程部使用。数据采用垂直分割方法分布于相应的部门计算机中，如图13-8所示。图13-8中的每个分割是通过对原始关系作投影（即选定列）而获得的。而原始关系则可以通过对所产生的分割进行自然联结而获得。

归纳起来，除了在垂直分割上进行数据组合要比在水平分割上更为困难以外，水平分割与垂直分割的利弊完全相同。这一困难主要是由于需要匹配主键或其他条件，以便联结分割上的行而造成的。水平分割支持其功能通常是按地域为基础加以复制的组织型设计，而垂直分区则通常应用于经过合理分割的数据需求之组织型功能。

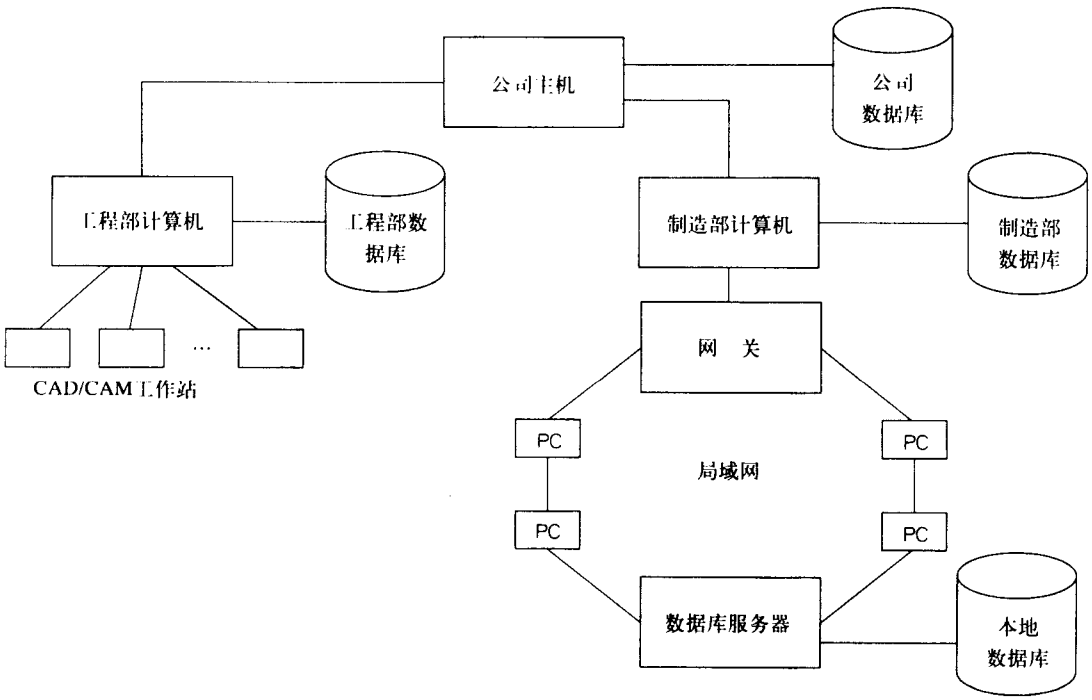


图13-6 某个制造公司的分布式处理系统

Part_Number	Name	Cost	Drawing_Number	Qty_on_Hand
P2	Widget	100	123-7	20
P7	Gizmo	550	621-0	100
P3	Thing	48	174-3	0
P1	Whatsit	220	416-2	16
P8	Thumzer	16	321-0	50
P9	Bobbit	75	400-1	0
P6	Nailit	125	129-4	200

图13-7　Part关系

Part_Number	Drawing_Number
P2	123-7
P7	621-0
P3	174-3
P1	416-2
P8	321-0
P9	400-1
P6	129-4

a) 工程部

Part_Number	Name	Cost	Qty_On_Hand
P2	Widget	100	20
P7	Gizmo	550	100
P3	Thing	48	0
P1	Whatsit	220	16
P8	Thumzer	16	50
P9	Bobbit	75	0
P6	Nailit	125	200

b) 制造部

图13-8　垂直分区

13.3.4　操作组合

前述策略几乎有无限多个组合，所以有时情况更为复杂。有些数据可以是集中存放的，而另外一些数据被复制到各个站点里。而且，对于某个给定关系，水平分割或垂直分割都可能适合于数据分布。图13-9就是组合策略的一个例子：

1) Engineering Parts、Accounting和Customer数据都集中存放在不同的地点。

2) 标准零件数据在三个位置上进行(水平)分割。

3) 在三个位置上都复制有Standard Price List。

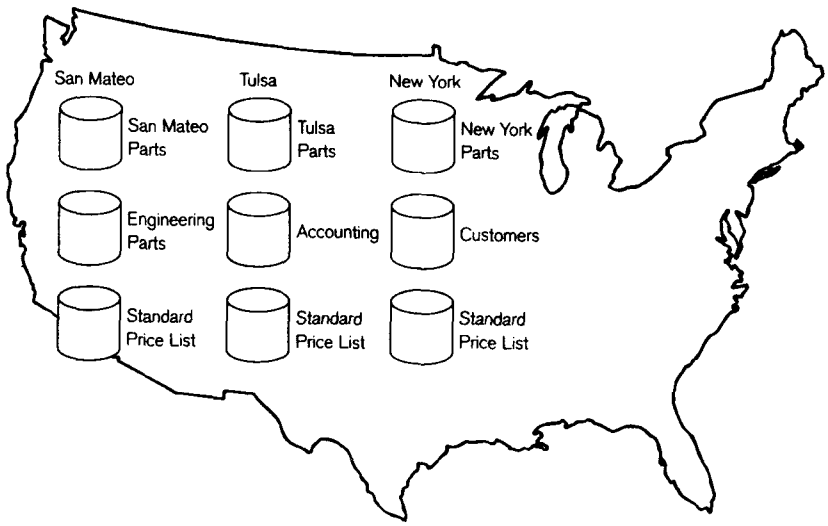


图13-9　数据分布的混合策略

(经Miller Freeman Publications允许，摘自“*Database Programming & Design*”，April 1989, Vol.2, No.4)

分布式数据库设计的原则是，数据应当存放在最频繁访问它们的地方（虽然其他因素，如安全性、数据完整性以及成本等也很重要）。在组织分布式数据库，决定其成为分布式而不是分散型的问题上，起着关键和中心作用的是数据管理员。

13.3.5 选择正确的数据分布策略

基于前几节的讨论，可以按照五种独特的方式组织一个分布式数据库：

1) 完全集中存放在一个位置，但从许多地理上分布的站点进行访问。

2) 地理上分布的站点上部分或完全复制，且采用快照周期性地更新每份拷贝。

3) 地理上分布的站点上部分或完全复制，且采用接近实时的方式同步更新。

4) 按地理上分布的不同站点分割成片段，但仍然处在一个逻辑数据库和一个分布式DBMS内。

5) 分割成独立的、非集成的散布于多台计算机和数据库软件上的片段。

以上五种方法没有哪个绝对是最好的。表13-1从可靠性、增加结点的可扩展性、通信开销或对通信网络的需求、可管理性以及数据一致性方面比较以上五种方法。分布式数据库设计者应该平衡这些因素以便选择适合给定分布式数据库环境的最佳策略。在给定情景下选择哪种策略取决于以下因素。

- *组织的能力* 可用的资金、组织中各单位的自治程度和安全性需求。
- *频度和局部性或数据参照的冲突性* 一般来说，数据位置应当接近使用这些数据的应用软件。

表13-1 分布式数据库设计策略的比较

策 略	可靠性	可扩展性	通信开销	可管理性	数据一致性
集中式	**差**： 高度依赖中央服务器	**差**： 有限性成为性能的壁垒	**很高**： 一个站点高流量	**很好**： 一个站点需要小的协调	**优**： 所有用户总是具有相同的数据
用快照复制	**好**： 冗余性和可容忍的延迟	**很好**： 增加拷贝成本可能小于线性	**低～中等**： 不固定，但周期性快照可能造成网络流量突增	**很好**： 每个拷贝都非常相似	**中等**： 只要业务能够容忍一定的延迟
同步复制	**优秀**： 冗余性和最小延迟	**很好**： 增加拷贝的成本低，同步化工作仅线性	**中等**： 消息固定，但允许有些延迟	**中等**： 冲突会给管理增加一些复杂性	**中等～很好**： 接近精确一致性
集成分割	**很好**： 有效利用分割和冗余性	**很好**： 新结点只获得所需的数据，无须变更总体数据库设计	**低～中等**： 查询多为本地，但需要多站点数据的查询可能引起暂时性负载	**难**： 要求分布式表和更新必须紧密协调的查询尤其困难	**很差**： 需要付出相当的努力，不一致性相当严重
采用独立分割的分散式	**好**： 只依赖本地数据库可用性	**好**： 新站点独立于已有的站点	**低**： 即便需要在网络上发送数据或查询，其负载也很小(如果存在)	**很好**： 每个站点都很容易管理，除非存在共享站点数据的要求	**低**： 不保证一致性，实际多半不一致

- *对于增长和扩展的要求* 网络上处理器的可用性将影响到数据可能存放的位置和应用软件运行的位置，并指示出网络需要扩展的要求。
- *技术能力* 必须考虑到每个结点的功能，对于DBMS来说还要考虑获得和管理该技术的成

本。存储成本不断降低，但管理复杂技术的成本却可能会增加。

- *可靠服务的要求*　以任务为主的应用软件和频繁需要的数据应该使用复制模式。

13.4　分布式DBMS

为了拥有分布式数据库，必须有一个数据库管理系统来协调各个站点上的数据访问。我们将这类系统称为*分布式*DBMS。虽然每个站点都有一个DBMS在管理本站点的本地数据库，但分布式DBMS则将完成下列功能(见Elmasri和Navathe, 1989; Buretta, 1997):

1) 在分布式数据词典里维护数据存放的位置。这意味着，向开发者和用户部分地表示一个逻辑数据库和模式。

2) 无须开发者或用户进行任何专门操作，就能确定所请求数据的位置，以及处理分布式查询每一部分所在的位置。

3) 倘若必要，把一个结点上采用本地DBMS的请求翻译成正确的其他站点上采用不同的DBMS和数据模型的请求，并以发出请求的结点所能接受的格式将数据返回。

4) 提供诸如安全性、并发性与死锁控制、全局查询优化以及自动故障记录与恢复等数据管理功能。

5) 提供远程站点上数据拷贝的一致性(例如，采用多阶段提交协议)。

6) 表现为单一逻辑数据库，它在物理上是分布的。这种数据视图的衍生物之一就是全局主键控制，即关于同一业务对象的数据总是与相同主键相联系的，而与分布式数据库里数据的存储位置无关，不同的对象总是与不同的主键相联系。

7) 可伸缩。可伸缩性就是规模可以增大或者减小，当业务需求改变时，可伸缩性就变得更加异构了。因此，分布式数据库必须是动态的，并且能够在合理的范围内变化，而无须重新设计。可伸缩性也意味着有了更简便的办法来添加（或订阅）新站点以及初始化新站点(如采用复制数据)。

8) 复制分布式数据库各结点上的数据和存储过程。与分布数据的理由相同，也需要对存储过程进行分布。

9) 透明地利用剩余的计算机功能来改善数据库处理的性能。例如，这意味着同一个数据库查询，若在不同的时间提交，就可以用不同的方式在不同的站点进行处理，这取决于查询提交时分布式数据库上的工作负载。

10) 允许不同的结点运行不同的DBMS。分布式DBMS和每个本地DBMS可以使用中间件(见第9章)来弥补查询语言和本地数据的细微差别。

11) 允许在分布式数据库的不同结点上驻留不同版本的应用软件代码。在具有多个分布式服务器的大型组织里，让每个服务器/结点都运行相同版本的软件显得不太实际。

并非所有的分布式DBMS都能完成上述的所有功能。几乎每个可行的分布式DBMS里都具备前6项功能。而其余的功能则按其重要性程度以及在现代技术中使用的频率，以近似降序排列出来。

从概念上说，每个本地站点都可能运行各种不同的DBMS，采用一个主DBMS来控制数据库部分的交互。正如本章前面所定义的，这类环境称为*异构分布式数据库*。目前，尽管理想、完全的异构还无法实现，但当每个DBMS具备相同的数据体系结构（例如关系）时，某些产品还是在一定程度上具有异构性的。

图13-10显示了具有分布式DBMS功能的计算机系统的一种流行的体系结构。每个站点都有一个本地DBMS，用于管理存放在本站点的数据库。而且，每个站点都有该分布式DBMS的

一个拷贝和相关的分布式数据词典/目录(DD/D)。该分布式DD/D包含网络中所有数据的位置以及数据定义。用户或应用软件对数据的请求首先由分布式DBMS处理，它会判断出所请求的事务究竟是局部的还是全局的。**本地事务**（local transaction）就是其请求的数据全部存放在本地站点的事务，而**全局事务**（global transaction）则需要引用一个以上的非本地站点的数据才能满足其请求。对于本地事务，分布式DBMS将请求发送到本地DBMS；而对于全局事务，分布式DBMS则按要求将请求传送到其他的站点。根据协调事务处理的需要，分布式DBMS在各参与站点交换消息，直到其完成（或必要时被中止）为止。可以看到，这个过程是十分复杂的。

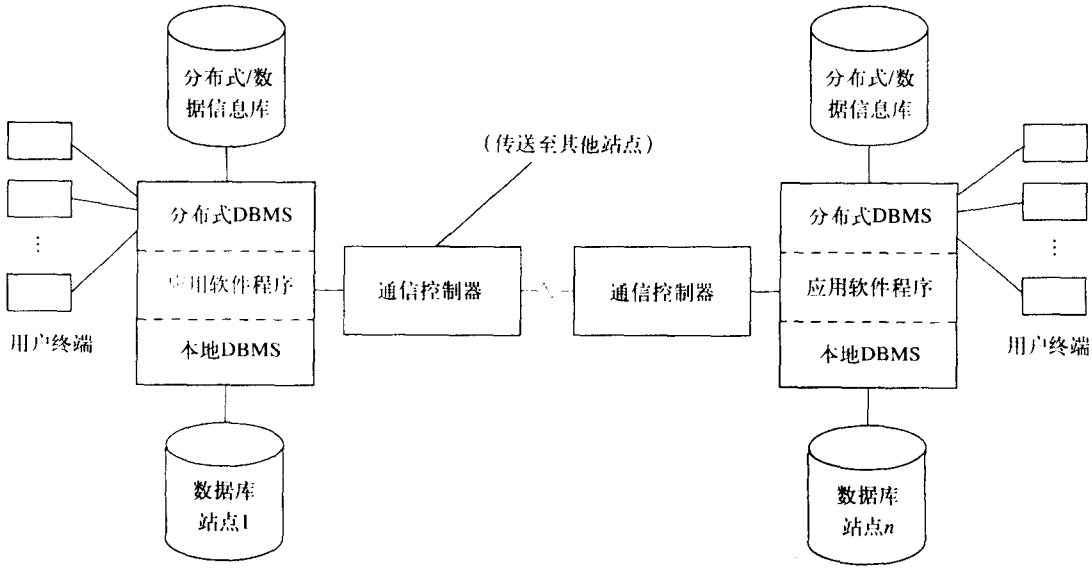


图13-10 分布式DBMS体系结构

一个站点上的DBMS（及其数据模型）可能与其他站点的DBMS是不同的。例如，站点A可能具有关系DBMS，而站点B具有网状DBMS。这时，分布式DBMS就必须对请求进行翻译，使它能够被各本地DBMS处理。处理混合型DBMS和数据模型的功能是一种刚刚在某些商用DBMS中出现的先进技术。

在分布式系统体系结构的讨论中（参见图13-10），我们假设每个站点都存在分布式DBMS和DD/D的拷贝（因此DD/D本身就是数据复制的一个例子）。另一种情况是将分布式DBMS和DD/D放置在中央站点，还可能有其他的策略。然而，集中式解决方案极易发生故障，因而不太理想。

分布式DBMS应当使用户与分布式数据库管理系统的复杂性尽可能分隔开来。换句话说，分布式DBMS应当让数据在网络中的位置以及分布式数据库其他特性变得透明。当分布式DBMS达到以下四个目标时，就可以很容易地构造程序并检索分布式系统里的数据。这些目标包括：位置透明性、复制透明性、故障透明性以及并发透明性。为了深入理解故障透明性和并发透明性，我们还讨论了提交协议的概念。最后描述查询优化，这也是分布式DBMS的重要功能之一。

13.4.1 位置透明性

尽管数据在地理上分散分布，并能随着位置的变化而转移，但利用位置透明性，用户（包括程序员）却能够将所有数据看作存放在单一结点那样进行操作。为了解释位置透明性，不妨

考虑图13-9所示的分布式数据库。该公司在加里弗尼亚州的San Mateo、俄克拉荷马州的Tulsa和纽约市都有仓库和相关的采购功能。公司的工程部在San Mateo，而销售部在纽约市。假设在加里弗尼亚州San Mateo的营销经理需要一份总采购量小于$100 000的所有顾客的清单。由于具有位置透明性，所以该经理可以从San Mateo的终端输入下列请求:

```
SELECT*
  FROM CUSTOMER
  WHERE TOTAL_SALES<100,000;
```

请注意，这一SQL请求不要求用户知道数据的物理存储位置。本地站点（San Mateo）的分布式DBMS将会咨询分布式DD/D，并确定这一请求必须发送到纽约。当选定的数据在San Mateo传输与显示时，该站点的用户觉得数据是在本地检索到的。(除非存在较长时间的通信延迟！)

现在考虑一个较为复杂、需要检索多个站点的数据的请求。例如，考虑图13-9里的Part逻辑文件，它是按地理位置分割，存放在靠近其各自仓库位置的计算机上的物理分布数据库文件: San Mateo Part、Tulsa Part和NewYork Part。假设Tulsa的库存经理希望得到一份有关桔黄色零件的清单（不管其位置）。这位经理可以采用下列查询来组合来自三个站点的信息:

```
SELECT DISTINCT PART_NUMBER,PART_NAME
  FROM PART
  WHERE COLOR='Orange'
  ORDER BY PART_NO;
```

在编写查询时，用户不必考虑零件数据存放在各个站点（假设具备位置透明性），因此，这是一个全局事务。若没有位置透明性，用户就应当分别引用每个站点的零件数据，然后组合数据(可能要使用UNION操作)，以产生所要的结果。

如果DBMS不直接支持位置透明性，那么数据库管理员可以通过创建视图（见第8章对使用SQL的视图的讨论），为用户实现虚拟的位置透明性。对于图13-9所示的分布式数据库，下列视图虚拟地在一张表里合并零件记录:

```
CREATE VIEW ALL_PART AS
  (SELECT PART_NUMBER,PART_NAME FROM SAN_MATEO_PART
  UNION
  SELECT PART_NUMBER,PART_NAME FROM TULSA_PART
  UNION
  SELECT PART_NUMBER,PART_NAME FROM NEW_YORK_PART);
```

其中三个零件表名字是三个远程站点的表的同义词。

前面的例子是关于只读事务的。本地用户是否也可以更新某个(或几个)远程站点的数据？利用现代的分布式DBMS产品，用户当然可以更新存放在某个远程站点的数据，比如本例中的Customer数据。因此，位于Tulsa的用户能够更新存放在San Mateo的BOM（原材料清单）数据。当更新存储在多个站点的数据（比如Vendor文件）的时候，会遇到比较复杂的问题，我们将在下一节讨论这个问题。

为了实现位置透明性，分布式DBMS必须能够访问某个精确的、当前的数据词典/目录，以指示网络中所有数据的位置。若该目录是分布的（如图13-9所示的体系结构），它们必须是同步的，这个目录的每一份拷贝都反映关于数据位置的相同的信息。尽管这方面取得了很大的进展，但在绝大多数现代系统里，仍然还未能提供真正的位置透明性。

13.4.2 复制透明性

虽然一个给定的数据项可以在网络的多个结点上复制，但利用**复制透明性**（replication transparency，有时也称为*片段透明性*），程序员（或者其他用户）仍然可以把该数据项当成是位于某一结点上的单个数据项进行处理。

为了说明复制透明性，让我们来看一下Standard Price List文件（参见图13-9）。在所有三个结点上都有一个与该文件相同的拷贝(完全复制)。首先考虑在每个结点处读取部分（或全部）文件的问题。分布式DBMS将会访问数据词典，并确定它是一个本地事务（即可以只利用本地站点的数据来完成的事务）。因此，用户不必担心要在其他站点上存储的相同数据。

现在假设只在部分（而非全部）站点上复制数据（部分复制）。如果这个读请求是从某个不包含被请求数据的站点发出的，那么此请求就应当传输到其他站点。此时，分布式DBMS应当选择能提供最快响应的远程站点。站点的选择取决于网络的当前条件（比如通信线路的可用性）。因此，分布式DBMS（与其他网络工具配合使用）应当动态地选择最佳路径。利用复制透明性，发出请求的用户就不用担心它是全局（非本地）事务。

当一个或多个用户试图更新复制的数据时，就会出现更为复杂的问题。例如，假设纽约的某个经理想到改变某个零件的单价。这一改变必须精确地、并发地在所有三个站点上实施，否则数据会不一致。利用复制透明性，纽约的经理就可以像本地事务那样输入数据，也不必担心在所有三个站点上完成相同的更新。然而，为了确保数据完整性，系统还必须提供我们将在下一节讨论的并发透明性和故障透明性。

13.4.3 故障透明性

分布式系统的每个站点（或结点）都有可能会遇到集中式系统会遇到的故障（错误的数据、磁头损坏等等）。而且，还要考虑通信链故障（或消息丢失）的风险。为了保证系统的健壮性，就必须能够*检测故障、重新配置系统使得计算可以继续进行，并且在修复处理器或通信链路后复原*。

错误检测和系统重新配置可能是通信控制器或处理器而非DBMS的功能。然而，分布式DBMS在出现故障时负责数据库恢复。每个站点的分布式DBMS都具有一个称为**事务管理器**（transaction manager）的组件，它执行下列功能:

1) 维持一个事务和数据库前象、后象的日志。

2) 维持适当的并发性控制模式，以确保该站点的事务并行执行时的数据完整性。

对于全局事务，每个参与站点中的事务管理器相互协调，以确保更新操作同步。没有这样的协调，在出现故障时就可能会丢失数据完整性。为了解释这个工作过程，假设（与早先一样）纽约的经理想要改变Standard Price List文件中某个零件单价（参见图13-9）。这个事务是全局的，因为必须更新（三个站点上）该零件记录的每份拷贝。假设在纽约和Tulsa已经成功地更新了单价表记录。然而，由于传输故障，San Mateo的单价表记录未被更新。现在，这个零件的数据记录是不一致的，员工有可能会存取到不正确的零件单价。

利用**故障透明性**（failure transparency），要么一个事务的所有操作都被提交，要么这些操作都不提交。一旦发生某个事务，则其影响远甚于硬件与软件故障。在供货商例子里，当一个站点的事务失败，则该事务的结果不会提交给其他站点。因此，直到该事务成功地完成之前，所有的站点始终保持着供货商旧的标称值。

13.4.4 提交协议

为了确保实时、分布式更新操作的数据完整性，需要协调事务管理器执行一个**提交协议**（commit protocol），这是一个定义完好的过程（涉及消息的交换），以确保全局事务要么在每

个站点都成功地完成，要么中止。使用最广泛的协议称为**两阶段提交**（two-phase commit）。一个两阶段提交协议确保多个站点上的并发事务在处理时就像是在所有站点上按相同的串行顺序执行的。一个两阶段提交有点像在许多人之间安排一次会议。首先，发出该全局事务的站点或者某个总体协调站点（就像是安排会议日程的人员），将向参与部分事务处理的每个站点发送一个请求。在安排会议时，其消息可以是“在某个日期或时间，您是否有空？”。每个站点都处理其子事务（倘若可能的话），但不立即把结果提交（或存储到）本地数据库。相反，是把结果存储到某个临时性文件里。若以开会作比喻，就是每个人都用笔在其日历上写下了会议安排。然而，每个站点都锁定了（禁止其他更新）其将要更新的那部分数据库（就像每个人都不会在预定的会议时间里安排其他约会）。每个站点在其完成子事务时就会给发端站点发送通知。一旦所有的站点都响应后，发端站点便启动两阶段提交协议：

1) 向每个参与站点广播一个消息，询问该站点是否要提交其站点所处理的那部分事务。每个站点回复“OK”或“Not OK”消息。这就像每个人回复能或不能出席会议。这个阶段通常称为预备阶段。“OK”代表该远程站点答应允许发出的请求在此远程数据库上管理事务。

2) 发端站点收集所有站点的消息。如果全部都是“OK”，就向所有站点广播一个消息，让它们提交各站点处理的事务部分。如果有一个或多个站点响应的是“Not OK”，它就向所有的站点广播一个消息，以中止该事务。这个阶段通常称为提交阶段。这就像会议安排者会根据每个人的回答来决定召开或取消会议计划。也有这种可能，即虽然预备阶段顺利通过，但在提交阶段（即在远程站点的提交之间）却出现事务失败，此时，称该事务为被废弃。一个废弃事务可以利用超时或轮询来加以识别。利用超时（在规定的时间内没有确认提交）无法区别是忙碌还是故障站点，而轮询则在网络负载和处理时间方面代价较高。

上面只对两阶段提交协议进行了简单描述。欲知其更详细的讨论和其他的协议，请参见Date(1995)。

对于分布式数据的同步化，采用两阶段提交策略，提交事务要比发端的位置能够单独工作的情况慢一些。两阶段提交这一传统方法的最新改进是，致力于降低由这一方法固有的过多协调所引起的延迟。目前已经开发了三种改进策略（见McGovern,1993）：

- *只读提交优化*　这一方法识别出事务中的只读部分，并消去对这些部分的确认消息之要求。例如，某个事务可能包含在输入新订单之前的检查库存余额的要求。在事务边界内读取库存余额可以无须回复确认。
- *惰性提交优化*　这一方法允许能够更新的站点先行更新，而其他不能立即更新的站点可以以后再“更新”。
- *线性提交优化*　这一方法使事务的每个部分依次提交，而不是在处理的子事务被延迟时挂起整个事务。

13.4.5 并发透明性

在第12章已经深入讨论过一个（集中型）数据库的并发性控制问题。当多个用户访问或者更新一个数据库时，除非采用加锁机制来防止数据受到并发性更新的影响，否则就有可能丢失数据完整性。在分布式数据库中，并发性控制问题比较复杂，因为多个用户分布于多个站点之间，而且数据通常又是复制到许多站点的。

并发性管理的目标很容易定义，但实际上却往往难以实现。虽然分布式系统并发运行许多事务，但**并发透明性**（concurrency transparency）却使每个事务就像系统中的惟一活动那样处理。因此，当几个事务并发地处理时，其结果必须与每个事务以串行顺序处理的结果相同。

必须协调每个站点上的（前面介绍的）事务管理器，以提供分布式数据库的并发性控制。

可以采用三种基本方法：加锁、版本设置以及盖时间戳，前面两种方法在第12章讨论数据库环境中的并发性控制方法时介绍过。在Date(1995)里讨论了分布式数据库中关于加锁的一些特殊方面。下面将讨论盖时间戳方法。

盖时间戳

采用这一方法，每个事务都会拥有一个全局惟一的时间戳，它一般由该事务发生时的系统时钟时间和站点ID所组成。**盖时间戳**（stamping）可以确保两个事件即便同时在不同的站点发生，它们都仍然具有惟一的时间戳。

盖时间戳的目标是确保事务按串行顺序处理，从而避免使用锁（因此也避免了产生死锁的可能性）。数据库里的每一条记录都载有最近一次更新它的事务的时间戳。如果某个新事务试图更新该记录，而它的时间戳比该记录上载有的时间戳还早，则该事务就需要分配一个新的时间戳并重新启动。因为一个事务不可能对时间戳比它晚的记录进行处理，所以它不可能去干涉其他的事务。

为了说明盖时间戳，假设某个数据库记录载有时间戳168，这表示带时间戳168的事务是最近成功地更新过该记录的事务。某个带时间戳170的新事务试图更新这个记录。这个更新是允许的，因为事务的时间戳比记录的当前时间戳更晚。当更新被提交时，记录的时间戳将重新设置成170。现在假设时间戳为165的记录试图更新该记录。这一更新是不允许的，因为它的时间戳比该记录所载有的时间戳还早。相反地，该事务时间戳将重新设置成记录所载有的168，并且重新再启动该事务。

盖时间戳的主要好处是避免了加锁和死锁检测（以及相关开销）。主要缺点是该方法过于保守，即使在与其他事务没有任何冲突的时候，事务有时也会重新启动。

13.4.6 查询优化

采用分布式数据库，对于查询的响应可能要求DBMS组合来自几个不同站点的数据（虽然具备位置透明性，用户不必操心这些要求）。DBMS的主要决策是如何处理查询，这既受到用户编写查询的方式的影响，又受到分布式DBMS开发处理查询的敏感计划的智能化程度的影响。Date(1983)为这个问题提出了一个既明确又简单的例子。考虑摘自Date的下列情景。一个简化的采购（关系）数据库具有下列三个关系：

```
SUPPLIER(SUPPLIER_NUMBER,CITY)              10 000条记录，存放在Detroit
PART(PART_NUMBER,COLOR)                    100 000条记录，存放在Chicago
SHIPMENT(SUPPLIER_NUMBER,PART_NUMBER)  1 000 000条记录，存放在Detroit
```

下面是用SQL编写的一个查询，以列出红色零件的Cleveland供货商的供货商编号：

```
SELECT   SUPPLIER.SUPPLIER_NUMBER
FROM     SUPPLIER,SHIPMENT,PART
 WHERE   SUPPLIER.CITY='Cleveland'
   AND   SHIPMENT.PART_NUMBER=PART.PART_NUMBER
   AND   PART.COLOR='Red';
```

每个关系中的每条记录长100个字符，有10个红色零件，Cleveland的送货记录有100 000次，查询的计算时间与通信时间相比可以忽略不计。而且，通信系统的传输速率为10 000字符/秒，消息从一个结点发送到另一个结点时有1秒钟的访问延迟。

Date针对这种情况点出了六种比较可行的查询处理策略，并研究了与其相关的通信时间。这些策略和时间在表13-2中进行了总结。随着策略的不同，满足查询所需要的时间从1秒钟直到2.3天！虽然最后的策略是最好的，但第四种策略也是可以接受的。

表13-2　分布式数据库环境下的查询处理策略(摘自Date,1983)

方　法	时　间
把PART关系移到Detroit、整个查询在Detroit计算机上处理	16.7分钟
把SUPPLIER和SHIPMENT关系移到Chicago、整个查询在Chicago计算机上处理	28小时
在Detroit计算机上连接SUPPLIER和SHIPMENT、将其投影到供货商为Cleveland的元组上、然后在Chicago计算机上逐个检查确定相应的PART是否为红色	2.3天
在Chicago计算机上投影PART到红色零件、并在Detroit计算机上逐个检查找出与该PART相应的SHIPMENT和SUPPLIER为Cleveland者	20秒
在Detroit计算机上连接SUPPLIER和SHIPMENT、仅对SUPPLIER为Cleveland者投影SUPPLIER_NUMBER和PART_NUMBER、再将匹配的投影移到Chicago计算机上以匹配红色PART	16.7分钟
在Chicago计算机选择红色PART、将结果移到Detroit计算机匹配SUPPLIER为Cleveland者	1秒

一般来说，这个例子表明通常建议将分布式数据库环境中的查询分解成由不同的站点隔离开来的片段，然后确定哪个站点最能产生最少的符合条件的记录，再将此结果移到其他站点，以完成另外的工作。显然，两个以上站点需要更复杂的分析和更复杂的探索来指导查询处理。

分布式DBMS通常采用下列三个步骤开发查询处理计划（见Özsu&Valduriez,1992）：

1) *查询分解*　此时，查询被简化，并改写成一种结构化的关系代数的形式。

2) *数据本地化*　此时，查询从原来以数据库处在一个位置上，而查询参照网络上的数据，变换成一个或多个片段，其中每一个片段只参照一个站点的数据。

3) *全局优化*　最后一步，决定执行查询片段的顺序，数据如何在站点之间移动，以及执行查询的哪些部分。

可以肯定，数据库的设计与分布式DBMS的复杂性会影响查询的性能。设计分布式数据库要取决于数据是在何处及如何使用。然而，对于给定的数据库设计（它将数据分配到一个或多个站点），所有的查询（无论期望与否）都必须尽可能有效地进行处理。

有一种更加有效地处理分布式查询的技术，称之为**半联结**（semijoin）操作(见Elmasri和Navathe 1989)。在半联结里，仅将正在连接的属性从一个站点移动到另一个站点，然后仅返回所要求的行。如果参与联结的仅是很少一部分行，那么需要传输的数据量是极少的。

站点1		站点2	
Customer表		Order表	
Cust_No	10字节	Order_No	10字节
Cust_Name	50字节	Cust_No	10字节
Zip_Code	10字节	Order_Date	4字节
SIC	5字节	Order_Amount	6字节
10 000行		400 000行	

图13-11　分布式数据库，两站点各一个表

例如，考虑图13-11所示的分布式数据库。假设站点1的查询要求显示在某个特定的Zip_Code范围而且Order_Amount超过给定界限的所有顾客的Cust_Name、SIC和Order_Date。假设处于该Zip_Code范围内有10%的顾客，而有2%的订单超过给定界限，则可以构造以下半联结：

1) 在站点1执行一个查询，以创建处于预期的Zip_Code范围内的Cust_No值列表。故有10 000顾客 × 0.1，即1000行满足该Zip_Code限制条件。因此，需要发送到站点2的有1000行Cust_No属性（连接中属性），每行10字节，共10 000字节。

2) 在站点2执行一个查询，以创建Cust_No和Order_Date的值列表发回站点1，从而形成最后的结果。如果我们假设每个顾客都有相同数量的订单，则与从站点1发来的顾客数相对应的Order表约为40 000行。如果假设每个顾客的订单在限值以上的同样多，则该查询涉及的Order表行数为800(40 000 × 0.02)。对于每一行，Cust_No和Order_Date需要发到站点1，即14字节 × 800行，共11 200字节。

采用上述的半联结，需要传输的数据总量仅有21 200字节。我们把这个总量和从一个站点到另一个站点所需要的每个表的子集作一个比较:

- 对于从站点1到站点2的数据发送，需要发送占Customer表1000行的Cust_No、Cust_Name和SIC（65字节1000行共65 000字节）发往站点2。
- 对于从站点2到站点1的数据发送，需要发送占Order表8000行的Cust_No和Order_Date（14字节，8000行共112 000字节）发往站点1。

显然，半连接方法减少了网络流量，这是响应用户查询的总体时间的主要因素。

分布式DBMS采用一个成本模型来预测可选执行计划的执行时间（数据处理和传输）。成本模型是根据一般网络条件在执行查询之前完成的；因而实际成本可能会有所偏差，这取决于实际的网络与结点负载、数据库重构以及其他动态因素。因此，随着网络中一般条件（比如本地数据库的重新设计、网络路径的变更以及本地站点DBMS的替换等）的改变，成本模型的参数应当定期地加以更新。

13.4.7 分布式DBMS的发展

分布式数据库管理仍然是一门新兴的而非已成熟的技术。目前发布的分布式DBMS产品并没有提供前面各节所描述的全部特性。例如，有些产品为只读事务提供位置透明性，但不能支持全局更新。为了描述分布式DBMS产品的发展，我们简单地把这一发展过程分割为三个阶段:远程作业单位、分布式作业单位和分布式请求。然后，在下一节里再归纳主流分布式DBMS（在编写本书时出现的这些软件包）的主要特点。

在下面的讨论里，术语作业单位指处理事务所需要的一系列指令，即从“开始事务”操作开始，直到“提交”或“回滚”操作为止的指令集合。

1. 远程作业单位

第一阶段允许在一个位置上发出多条SQL语句，并使其像是在一个远程DBMS上的一个作业单位那样执行。无论发送端还是接收端计算机，都必须运行相同的DBMS。发送端计算机并不需要访问数据目录来定位包含远程作业单位的选定表的站点。相反，发送端的应用软件必须先知道数据驻留的位置，并在连接到远程作业单位之前先与远程DBMS相连。因此，远程作业单位概念并不支持位置透明性。

一个远程作业单位（也称为远程事务）允许在一个远程计算机上进行更新。在作业单位内的所有更新都是尝试性的，直到对它们作出提交使其变成永久性的，或者回滚撤销它们为止。因此，要为远程站点维护事务完整性的。然而，当操作涉及多个远程位置时，事务完整性就不能得到保证。参考图13-9的数据库，San Mateo的应用软件可以更新Tulsa的Part文件，并保持事务完整性。但是，应用软件不能同时更新两个以上位置上的Part文件，却仍然可以维持事务完整性。因此，远程作业单位也不提供故障透明性。

2. 分布式作业单位

分布式作业单位允许作业单位内的各个语句引用多个远程DBMS。这个方法支持一定程度上的位置透明性，因为它能访问数据目录，对包含每个语句中选择表的DBMS进行定位。但是，在一个SQL语句里的所有表都必须处于同一个位置。因此，计划用来组合图13-9所示的三个站

点的零件信息的下列查询，是不能用于远程作业单位的:

```
SELECT DISTINCT      PART_NUMBER,PART_NAME
FROM                 PART
WHERE                COLOR='ORANGE'
ORDER BY             PART_NUMBER
```

类似地，分布式作业单位也不允许用一条SQL语句更新多个位置上的数据。例如，下列SQL语句打算更新三个位置上的Part文件:

```
UPDATE      PART
SET         UNIT_PRICE=127.49
WHERE       PART_NUMBER=12345
```

这一更新（倘若执行的话）将会在Tulsa、San Mateo和New York（参见图13-9）设置零件号12345的单价为$127.49。然而，这个语句是不能作为分布式作业单位的，因为该SQL语句要引用多个位置上的数据。分布式作业单位能支持涉及多个站点的受保护的更新，只要每个SQL语句引用的仅是一个站点上的一个（或多个）表即可。例如，在图13-9中，假设我们需要增加Tulsa的零件号为12345的库存量，同时减少在纽约的同种零件的库存量（可以反映库存的调整）。此时可以采用下列SQL语句:

```
UPDATE      PART
SET         BALANCE=BALANCE-50
WHERE       PART_NUMBER=12345 AND LOCATION='TULSA'
UPDATE      PART
SET         BALANCE=BALANCE+50
WHERE       PART_NUMBER=12345 AND LOCATION='NEW YORK';
```

在分布式作业单位概念下，要么这个更新在两个位置都提交了，要么它回滚并（或许）重新尝试。从这个例子可以看出，分布式作业单位在一定程度上支持本节前面所讨论的透明性。

3. 分布式请求

分布式请求克服了分布式作业单位的主要限制，它允许一条SQL语句引用多个远程DBMS表。分布式请求支持真正的位置透明性，因为一条SQL语句能够引用多个站点的表。然而，分布式请求可以（或不可以）支持复制透明性和故障透明性。有可能在真正的分布式DBMS之前的某个时刻，市场上会出现支持前面所描述的所有透明性的产品。

13.5　分布式数据库管理系统产品

数据库管理系统的绝大多数主要供货商都有分布式版本产品。在大多数情况下，为了尽量利用分布式数据库的功能，必须在每个结点上运行同一个供货商的DBMS（同构分布式数据库环境）。客户/服务器形式的分布式数据库虽有争议，但实际上却是今天最普遍的形式。在客户服务器环境里（参见第9章关于客户/服务器数据库的阐述），很容易在局域网或广域网的多个结点上定义具有表的数据库。一旦用户程序确立了与每个远程站点的链接，并装载好适当的数据库中间件，就可以达到完全的位置透明性。所以，在客户/服务器数据库中，分布式数据库（甚至异构DBMS）都是很容易用于任何信息系统开发者。

尽管它们的实现方法在不断改变，但还是可以概述一下不同的供货商是如何解决分布式数据库管理的。用户最感兴趣的方面是产品之间的差异。这些差异（归纳在表13-3里）说明了选择分布式DBMS产品的难度，因为DBMS的确切功能必须与组织的需求尽可能匹配。此外，既有如此多的可选项，每个产品处理的分布式数据又各不相同，因此，几乎不可能列举出管理分

布式数据库的一般原理。在进行任何分布式数据库的设计时都要求仔细地分析业务的需求和DBMS的复杂性。Thompson(1997)也建议，应当只在真正需要分布式DBMS的地方才采用分布式DBMS产品。不要用分布式DBMS来为以任务关键的应用软件创建备份数据库；对于简单需求则有比较简单的解决办法，比如RAID(见第6章)。

表13-3 分布式DBMS

供货商	产品	重要特性
IBM	DataPropagator Relational (DPropR)	• 与DB2一起工作 • 主站点和异步更新 • 只读站点订阅主站点 • 订阅子集与查询结果
	Distributed Relational Database Architecture (DRDA)	• 异构数据库
	DataJoiner	• 访问非IBM数据库的中间件
Sybase	Replication Server	• 主站点和分布式只读站点 • 只读站点更新为一个事务 • 层次型复制 • 复制数据和存储过程
	SQL Anywhere	• 可移动数据库
	OmniSQL	• 异构数据库
Oracle	Table Snapshot Option	• 周期性快照发往只读站点
	Symmetric Replication option	• 带多重可更新拷贝的异步、同步和从某个结点到其他任何结点的复制(双向) • 微分刷新 • DBA控制复制 • 两阶段提交
Computer Associates	CA-Ingres/Replicator	• 可更新所有的数据库拷贝 • 层次型复制 • DBA寄存复制数据和其他站点的数据 • 主/从形式允许从属于Ingres数据库
	Ingres/Net 和 Ingres/Star	• 将查询分解到分布的、同构的站点 • 两阶段提交 • 也可用于非Ingres数据库
Microsoft	SQL Server	• 主站点和分布式只读站点 • 公布和订阅文章、出版物 • 一个数据库能够把发布的拷贝送往其他站点 • 可移动数据库

本章小结

本章介绍了有关分布式数据库的各种问题与技术。我们已经看到分布式数据库是遍布于多个位置的计算机上，并通过一个数据通信网络连接起来的单个逻辑数据库。分布式数据库与分散型数据库不同，分散型数据库中的数据并没有互相连接。在分布式数据库中，其网络必须允许用户尽可能透明地共享数据，还必须允许每个结点自治地运作，在网络链接断开或者某个结点出故障时，这一点尤其重要。当前的业务环境鼓励用户采用分布式数据库，这些业务环境包括业务单位的分散化与自治（包括组织的全球化）、数据共享的需求以及数据通信的成本与可靠性等等。分布式数据库环境可以是同构的，即每个结点都使用相同的DBMS；也可以是异构的，

即不同的结点可以使用不同的DBMS。此外，一个分布式数据库可以用直接同步方式保存数据与相关数据的全部拷贝，也可以容忍采用异步方法进行数据更新所带来的预料中的延迟。

分布式数据库有诸多优势。最重要的优势包括：提高数据的可靠性与可用性、用户对其数据的本地控制、模块式（或增量式）增长、减少通信成本以及快速响应数据请求。分布式数据库也会带来一些开销与不利因素，其中包括软件成本较高且复杂、处理开销增加、维护数据完整性更为困难，而且倘若数据分布得不当，对于数据请求的响应可能会很慢。

在一个网络中分布数据可以有多种选择：数据复制、水平分割、垂直分割以及这些方法的组合。采用数据复制，每两个（或多个）站点就要保存该数据库（或部分数据库）的一份拷贝。数据复制可以改善可靠性与加快响应速度，在一定的场合下又能够很简单地完成，它允许结点彼此更独立(当然也还需要协调)地运作，从而减小了网络流量。然而，它需要额外的存储容量，要在每个站点直接进行更新比较困难。复制后的数据可以通过对数据的正式记录作周期性快照，并把快照发送到复制站点进行更新。这些快照可以包含所有的数据，也可以仅仅包含自最近一次快照以来有变化的数据。采用水平分割时，一个联系的某些行放在某个站点里，而别的行则放在另一个(或者几个)站点里。另一方面，垂直分割则是在不同的站点分布一个联系的列。数据分割的目标是改善性能和安全性，人们经常采用把数据复制与水平分割、垂直分割组合使用。在选择数据分布式设计时，主要考虑的因素包括：组织性因素、查询与事务的频率与位置、数据与结点的可能增长、技术水平以及对可靠性的要求。

有了分布式数据库，还必须有一个分布式DBMS对各个结点的数据访问进行协调。来自用户或应用软件的数据请求，首先由分布式DBMS处理，以确定该事务是本地事务（可以由该站点处理）、远程事务（可以由某个别的站点处理）还是全局事务（需要在几个非本地站点访问数据）。对于全局事务，该分布式DBMS会去访问数据目录，并在需要时传输部分请求，然后把远程站点的结果汇集起来。

一个分布式DBMS应当使用户远离分布式数据库在管理上的复杂性。位置透明性是指尽管数据在地理上是分布的，但对用户来说仿佛所有的数据都处在一个结点上。复制透明性是指尽管数据项可能存放在几个不同的结点上，但用户可以把它当成是单个结点上的单一数据项那样地处理。故障透明性是指要么一个事务的所有操作都在每个站点上完成，要么它们都不被提交。分布式数据库可以在直接同步化不是必要的时候允许结点之间有暂时的不一致。并发透明性是指每个事务都像是该系统里仅有的一项活动。故障透明性和并发透明性可以通过提交协议来加以管理，该协议协调着结点间的更新、数据加锁以及盖时间戳。

由分布式DBMS做出的一项重要决策就是如何处理全局查询。处理一个全局查询的时间可能会从几秒到几个小时，这取决于该DBMS在产生一个有效的查询处理计划时的智能程度。查询处理计划把该查询分解成一系列结构化步骤，这些步骤是通过该分布式数据库不同的结点的本地数据来确定的，并最终选择出执行这些步骤的顺序与位置。

多数(如果不是全部的话)分布式DBMS产品都提供各种形式的透明性、各种形式的数据复制与分割，以及在分布式查询处理上同样级别的智能性。然而，由于分布式系统的增长所形成的业务压力，这些产品都在迅速地改善。关系数据库产品的主流供应商已经引入了分布式版本，以及帮助数据库管理员设计与管理分布式数据库的工具。

本章复习

关键术语

分布式数据库　　分散型数据库　　位置透明性

本地自治　同步分布式数据库　异步分布式数据库
本地事务　全局事务　复制透明性
事务管理器　故障透明性　提交协议
两阶段提交　并发透明性　盖时间戳
半联结

复习问题

1. 定义下列术语:
   a. 分布式数据库　b. 位置透明性
   c. 两阶段提交　d. 全局事务
   e. 本地自治　f. 盖时间戳
   g. 事务管理器
2. 匹配下列术语及其定义:
   _____复制透明性　a. 确保某个分布式数据库上的事务或者全部更新，或者都不更新
   _____作业单位　b. 表现为给定事务是惟一一个在分布式数据库上运行的事务
   _____全局事务　c. 将数据拷贝视为仅有一个拷贝
   _____复制　d. 引用不止一个位置的数据
   _____并发透明性　e. 处理某个事务所需要的指令的顺序
   _____故障透明性　f. 对于只读型数据的一种较好的数据库分布策略
3. 比较下列术语:
   a. 分布式数据库; 分散型数据库　b. 同构分布式数据库; 异构分布式数据库
   c. 位置透明性; 本地自治　d. 异步分布式数据库; 同步分布式数据库
   e. 水平分割; 垂直分割　f. 发布; 订阅
   g. 完全刷新; 微分刷新　h. 下推复制; 下拉复制
   i. 本地事务; 全局事务
4. 简要描述可以使用分布式数据库的六个业务条件。
5. 阐述同构分布式数据库的两种类型。
6. 简要描述同构分布式数据库的五个主要特征。
7. 简要描述异构分布式数据库的四个主要特征。
8. 简要描述分布式数据库与分散型数据库相比的五个优势。
9. 简要描述分布式数据库的四个开销与不利之处。
10. 简要描述分布式数据库数据复制的五个好处。
11. 简要描述分布式数据库数据复制的两个不利之处。
12. 阐述最适合采用快照复制方法的场合。
13. 阐述最适合采用接近实时复制方法的场合。
14. 简要描述五个影响数据复制作为某个应用软件分布式数据库设计的策略的因素。
15. 阐述分布式数据库水平分割的利弊。
16. 阐述分布式数据库垂直分割的利弊。
17. 简要描述五个影响分布式数据库设计策略选择的因素。
18. 简要描述六个分布式数据库管理系统所独有的功能。
19. 简要说明位置透明性对一个特定的数据库查询的作者的影响。
20. 简要说明复制透明性对一个特定的数据库查询的作者的影响。

21. 简要说明在什么场合下两阶段提交仍然无法创建完全一致的分布式数据库。
22. 简要描述对于两阶段提交协议的三种改进。
23. 简要描述分布式查询处理的三个步骤。
24. 简要说明采用半联结会加快分布式查询处理速度的条件。

问题和练习

问题和练习1～3参考图13-9所示的分布式数据库。

1. 说出以下每一种情况所应用的透明性。
   a. New York与Tulsa的最终用户同时要对San Mateo的Engineering Part数据库进行更新，两个用户都未意识到对方正在访问数据，然而系统会避免由于干扰而造成更新丢失。
   b. Tulsa的最终用户删除了本地Standard Price List中的某数据项。他不知道该分布式DBMS还从San Mateo与New York的Standard Price List中里删除了此数据项。
   c. San Mateo的某个用户启动一项事务，即删除San Mateo零件库的某个零件，同时把它追加到New York零件库里去。该事务在San Mateo中完成，但由于传输故障在New York没有完成。该分布式DBMS便自动地恢复了San Mateo的事务，并通知用户重试此事务。
   d. New York的某个最终用户请求本地查看编号为33445的零件的库存。该用户不知道此零件的记录的存放位置。分布式DBMS去访问数据目录，并把此项请求传输给San Mateo。
2. 考虑图13-9中的Standard Price List:
   a. 写出使Part_Number 56789的Unit_Price增加10%的SQL语句。
   b. 指出上述语句在下列协议中是否可以接受:
      (1) 远程作业单位　(2) 分布式作业单位　(3) 分布式请求
3. 考虑图13-9中的四个Part数据库:
   a. 写出将San Mateo Part库中Part_Number 56789的库存增加10%的SQL语句，以及将New York Part库中Part_Number 12345的库存减少10%的SQL语句。
   b. 指出上述语句在下列协议中是否可以接受:
      (1) 远程作业单位　(2) 分布式作业单位　(3) 分布式请求
4. 说明为什么难以实现真正的异构分布式数据库环境。此环境里存在哪些特殊困难。
5. 阐述表13-2所列出的六种查询处理策略会产生截然不同的结果的主要原因。
6. 表13-2所列的六种查询处理策略中是否使用了半联结？若有，请阐述半联结是如何用的。若没有，请阐述如何使用半联结来创建有效的查询处理策略，或者为什么在这种情况下使用半联结无济于事。
7. 考虑本章13.3.6节提到的分布式数据库和SUPPLIER、PART与SHIPMENT关系:
   a. 写出全局SQL查询（在Chicago提交），显示不是由Columbus供应商提供的任何零件之Part_Number和Color。
   b. 为上述查询设计三个可能的查询处理策略。
   c. 为比较上述三个策略的处理时间，开发一个与表13-2类似的表。
   d. 三种策略中哪个最佳？为什么？
   e. 能否利用数据库的数据复制、水平分割和垂直分割创建一种更加有效的查询处理策略？说明原因。
8. 考虑一家大型连锁店的下述规范化的数据库关系:

STORE (Store_ID, Region, Manager_ID, Square_Feet)
EMPLOYEE (Employee_ID, Where_Work, Employee_Name, Employee_Address)
DEPARTMENT (Department_ID, Manager_ID, Sales_Goal)
SCHEDULE (Department_ID, Employee_ID, Date)

假设有一个数据通信网络把公司总部的计算机与每个零售批发店的计算机链接起来，连锁店中包括50家商店，平均每家商店有75名员工，而每家商店有10个部门。需要保存5个月（前两个月、当月以及后两个月）的每日进度安排表。进一步进行如下假设：

- 每个商店经理大约每小时更新5次其员工的工作进度安排表。
- 公司生成每家商店每个员工的全部工资表清单、员工通知以及其他邮件。
- 公司每个月要为每个部门制定新的销售目标。
- 公司有权雇用与解雇各个商店经理，并控制商店经理的全部信息，而商店经理有权雇用与解雇其店内的所有员工，并控制有关他们的全部信息。

a. 你会为该连锁商店推荐分布式数据库、分散型数据库还是一组分散型数据库？
b. 假设决定采用某种分布式数据库，你建议该连锁店采用怎样的数据分布策略？

应用练习

1. 参观一个已经安装了分布式数据库管理系统的组织。研究下列问题：
   a. 该组织是否拥有真正的分布式数据库？若有，其数据是如何分布的？是采用数据复制、水平分割还是垂直分割？
   b. 采用的是哪种商业分布式数据库产品？该组织为什么选择这些产品？该组织使用这些产品后，是否发现过什么问题或者限制？
   c. 该系统是否提供了以下的各种扩展功能：
      i. 位置透明性　　ii. 复制透明性
      iii. 并发透明性　　iv. 故障透明性
      v.查询优化
   d. 该组织对于分布式数据库未来的发展有何打算？
   e. 与该组织的数据库管理员交谈，以了解如何确定网络中数据的位置。在决策中主要考虑哪些因素？有没有采用过分析性工具？若有，数据库管理员是否满意于此项工具对其提高查询处理效率的帮助？
2. 在因特网上查找研究本章讨论到的DBMS供应商提供的最新的分布式数据库产品。更新其中描述的一个分布式DBMS产品的特点。试查找别的供应商的分布式DBMS产品并列出关于这些产品的信息。
3. 参观某个安装有客户/服务器数据库环境的组织。寻找下述问题的答案：
   a. 该客户/服务器DBMS在使用时提供哪些分布式数据库特性？
   b. 该组织是否尝试达到本章对分布式数据库所指出的客户/服务器环境的相同好处？有哪些好处达到了？又有哪些是客户/服务器技术不可能达到的？

参考文献

Bell, D. and J. Grimson. 1992. *Distributed Database Systems*. Reading, MA : Addison-Wesley.

Buretta, M. 1997. *Data Replication : Tools and Techniques for Managing Distributed Information*. New York: John Wiley & Sons, Inc.

Date, C. J. 1983. *An Introduction to Database Systems*. Vol. 2. Reading, MA: Addison-Wesley.

Date, C. J. 1995. *An Introduction to Database Systems*. 6 th ed. Reading, MA: Addison-Wesley.

Edelstein, H. 1993. "Replicating Data." *DBMS* 6 (June) : 59-64.

Edelstein, H. 1995 a. "The Challenge of Replication, Part I." *DBMS* 8 (March) : 46-52.

Edelstein, H. 1995 b. "The Challenge of Replication, Part II." *DBMS* 8 (April) : 62-70, 103.

Elmasri, R., and S. B. Navathe. 1989. *Fundationals of Database Systems*. Menlo Park, CA: Benjamin /Cummings.

Froemming, G. 1996. "Design and Replication : Issues with Mobile Applications—Part 1." *DBMS* 9 (March) : 48-56.

Koop, P. 1995. "Replication at Work." *DBMS* 8 (March) : 54-60.

McGovern, D. 1993. "Two-Phased Commit or Replication." *Database Programming & Design* 6 (May) : 35-44.

Özsu, M. T., and P. Valduriez. 1992. "Distributed Database Systems: Where Were We?" *Database Programming & Design* 5 (April) : 49-55.

Schussel, G. 1994. "Database Replication : Watch the Data Fly." *Client/Server Today* (October) : 80-90.

Thé, L. 1994. "Distribute Data Without Choking the Net." *Datamation* (January 7) : 35-38.

Thompson, C. 1997. "Database Replication: Comparing Three Leading DBMS Vendors' Approaches to Replication." *DBMS* 10 (May) : 76-84.

Web资源

- http://dis.sema.es/projects/WIDE/ 该WIDE项目主要研究开发同扩展分布式与主动数据库技术有关的课题，以便为实现工作流的高级面向应用的软件产品提供附加价值。这些问题已经由来自西班牙、意大利与新西兰的专业人士所组成的组织进行研究。
- http://www.compapp.dcu.ie/databases/f449.html 该Web网络站是Dublin City大学维护的，比较关注数据库与多媒体。该网站有对分布式数据库几个重要方面的通俗易懂的教程。
- http://databases.about.com/compute/databases/ 该Web网站包含有众多的关于各种数据库技术（包括分布式数据库）的新闻与评论。

项目案例：山景社区医院

在第6章山景社区医院的案例里，我们曾为一个集中型数据库作过一些物理数据库设计的决策。请参照该部分的项目问题和项目练习的答案。

项目描述

山景社区医院将从几个方面进行扩展。首先，医院将在Golf Estates郊区开设一家附属外科门诊部。其次，医院正在获取社区内几个医生团体的意见，并向这些办公室和门诊所提供所有的病历记录和账单服务。因为有这些显著变化，医院信息系统的员工正在考虑采用分布式计算机和数据库。

项目问题

1) 除了第6章中制定物理数据库设计决策时收集的信息之外，还需要哪些信息才能决定山景社区医院应当采用分布式数据库技术？

2) 除了第6章中制定物理数据库设计决策时收集的信息之外，还需要哪些类型的信息，才能决定为山景社区医院设计分布式数据库？

3) 是否有机会把分布式数据库的数据复制、水平分割和垂直分割应用于山景社区医院？倘若不能确定，你还需要哪些信息才能回答这个问题？

项目练习

对于山景社区医院来说，绝大多数数据维护都是在病人活动所在的站点本地完成的。例如，以第6章中图6-13表示的数据为例，80%以上的变化是在医生办公室作出的，10%是在医院总部作出的，还有10%是在附属外科诊所里作出的。而且，绝大多数查询都是本地的（例如，医生办公室里的医生通常在病人在其办公室时，查阅对此病人进行过哪些治疗）。不过，由于医疗服务问题通常要求快速的响应，所以医生很关心分布式数据库的性能。

另一方面，管理性的查询几乎都集中在医院总部的管理办公室里。产生结论的查询、向州卫生机构汇报治疗统计信息的总结、医生效率和效益等问题，全部是在医院总部由管理人员发出的。

山景社区医院分布式数据库设计的一个建议是将医生、病人、过程和消费数据进行水平分割，并把它们分布于本地医生计算机里。例如，一个科的所有医生的记录，以这些医生为主治医生的所有病人，以及与此相关的所有活动的数据都存放在医生办公室的计算机里。Treatment和Item数据在各站点上加以复制。医院总部的计算机和外科诊所的计算机保存所有数据的拷贝，因为当病人正在医院或外科诊所就诊时，可能需要访问所有的数据，而通向办公室的数据链路被关闭。

1) 你会建议这个分布式数据库采用何种形式的数据复制完整性控制？为什么？

2) 对于以下每个查询，你建议采取怎样的全局查询优化计划？

a. 上个月每项Item总共消费多少？

b. 上个月中，医生办公室、外科诊所以及医院总部中每个医生诊治多少名病人？

c. 按医疗科别分类，每个医生在每一科别中开了多少个处方？

第14章　面向对象数据建模

14.1 学习目标

学完本章[⊖]后，读者应该具备以下能力：

- 定义下列关键术语：对象、状态、行为、对象类、类图、对象图、操作、封装、构造器操作、查询操作、更新操作、范围操作、关联、关联角色、多重性、关联类、抽象类、具体类、类范围属性、抽象操作、方法、多态、重载、多重分类、聚合和复合。
- 描述面向对象开发生命周期的不同阶段中的活动。
- 与结构化方法相比，列举面向对象建模的优点。
- 比较面向对象模型与ER和EER模型的区别。
- 使用UML（统一建模语言）类图来建模现实世界的应用。
- 使用UML的对象图，提供系统在某时间点上详细状态的快照。
- 确定何时使用概化、聚合和复合联系。
- 说明类图中不同类型的业务规则。

14.2 引言

在第3章和第4章，我们已经学过如何使用ER模型和EER模型进行数据建模。在那些章中已学过如何使用实体、属性和各种联系来对组织的数据需求进行建模。本章将介绍日趋流行的面向对象模型，这种模型能完全表示复杂的联系，并且以一致的符号表示数据和数据处理。以前学的大部分概念与面向对象建模中的概念是一致的，但面向对象模型比EER模型有更多的特性。

在第3章和第4章中已提到，数据模型是对现实世界的抽象。它能通过关注一个组织需要的数据的最本质和最基本的特性来处理现实世界问题所固有的复杂性。ER模型是建立在实体的基础上，而面向对象模型是建立在对象基础上。但是，我们后面将会看到，对象把数据和行为封装在一起，也就是说，面向对象方法不仅可以用于数据建模，也可以用于处理建模。为了完整地建模现实世界的应用，不仅要建模数据，而且还要建模数据的处理（这在第2章有关规划对象的信息中已讨论过）。由于面向对象方法能够以一种通用的方式表示数据和处理，而且提供了诸如继承和代码重用等优点，因此，为开发复杂系统时提供了功能强大的环境。

和第2章解释的系统开发生命周期概念类似，面向对象的开发生命周期按进度可以分成三个阶段——分析、设计和实现（见图14-1）。在开发的早期阶段中，开发的模型是抽象的，主要关注系统的外部性质。随着模型的逐步演化，它越来越详细，关注点转移到系统怎样建立和系统有怎样的功能上。建模的重点应该是在分析和设计阶段，关注前端的概念问题，而不是后端的实现问题，没有必要限制设计的选择（Rumbaugh等，1991）。

在分析阶段，应开发一个表示现实世界应用的重要性质的模型。这个模型从应用领域中抽象出概念，并且描述所开发的系统应该做些什么（what）事情，而不是描述这个系统怎么（how）完成这些事情。这个模型应该说明系统的功能性行为，这与系统最后实现的环境是无关的。你必须花足够的时间去清楚地理解问题的需求。这个分析模型应该完整、准确地捕捉问

⊖ 本章的最初版本由Wisconsin-Milwaukee大学的Atish P.Sinha教授编写。

题的需求。应该记住，在分析阶段修改或修正错误要比在以后的阶段完成这些工作容易得多，且花费少得多。

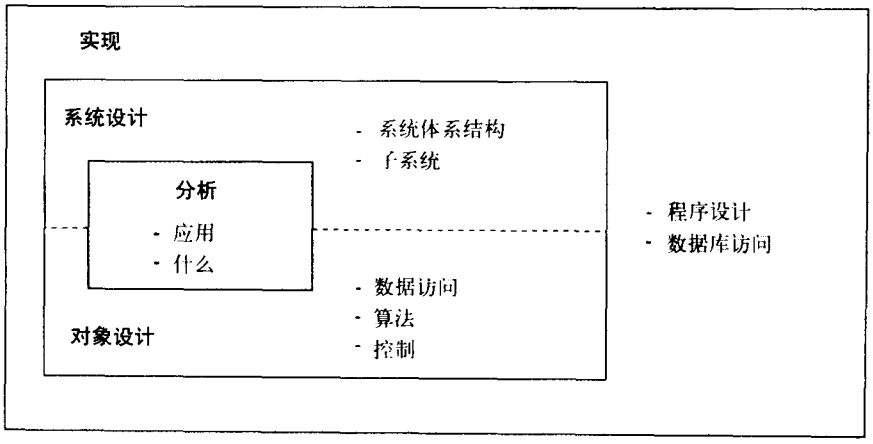


图14-1 面向对象系统开发生命周期的阶段

在面向对象设计阶段，要定义面向应用的分析模型怎样在实现环境中实现。Jacobson等人在1992年列举了使用面向对象设计的三个理由。这三个理由如下所述:

1) 分析模型不足以直接用程序设计语言去实现。为了能无缝地转换成源代码，还需要通过下列决策对对象作进一步的精化: 一个对象将提供哪些操作，对象之间应该如何通信，它们之间传递哪些消息，等等。

2) 实际系统必须适应系统将要实现的环境。要达到这个目标，在将分析模型转换成设计模型时，必须考虑到不同的因素，例如性能需求、实时需求和并发性、目标硬件和系统软件、相应的DBMS和程序设计语言等。

3) 使用面向对象设计能够确认分析的结果。在设计阶段，可以验证来自分析的结论是否适合于建立这个系统，是否要对分析模型做出必要的修改。

为了开发设计模型，你必须确认和审查实现环境对设计将产生哪些影响（Jacobson等，1992)。应该做出所有策略性的设计决定，例如怎样整合DBMS，进程间通信和错误处理怎样完成，重用哪些组件库等。接下来，应该将这些决定整合到最初的设计模型中以适应实现环境。最后，应该对设计模型形式化，以描述在各种可能的场景下（用例）对象相互之间如何交互。

Rumbaugh等人在1991年把设计活动分成两个阶段: 系统设计和对象设计。作为系统设计人员，应该提出一个总的系统体系结构，按不同的组件（称为子系统）组织系统，并给出做出决定的相关语境，例如确定并发性；分配子系统到处理器或任务中；处理对全局资源的访问；在软件中选择一个控制的实现等等。

在对象设计阶段，按照系统设计阶段所确定的策略，通过在分析模型中加入实现细节来建立设计模型。这些细节有: 为了改善有效性而重新构造类，实现每个类的内部数据结构和算法，控制的实现，关联的实现，以及打包成物理模块。分析模型中应用领域的对象类仍应保留，但应该增加计算机领域的成分，以优化重要的性能指标。

设计阶段完成后就进入实现阶段。在实现阶段，要使用程序设计语言或DBMS实现设计。把设计转换成程序代码是比较容易的过程，因为设计模型已经整合了程序设计语言和DBMS的细微之处。

Coad和Yourdon（1991b）给出了面向对象建模的一些动机和益处，如下所述:

- 处理更具挑战性的问题领域的能力。
- 改善了用户、分析者、设计者和程序员之间的交流。
- 提高了分析、设计和程序设计活动之间一致性。
- 明确地表示了系统组件之间的共性。
- 系统的健壮性。
- 分析、设计和程序设计结果的可重用性。
- 提高了在面向对象分析、设计和程序设计阶段开发的所有模型的一致性。

最后一点需进一步说明。在结构化分析和设计的建模方法（第2章中介绍的）中，开发的模型（例如，分析阶段的数据流图和设计阶段的结构图）缺乏通用的表示，因此很难联系起来。与这些方法所固有的不连续和不相交的转化相比，面向对象方法提供了从分析到设计再到实现的连续的表示（Coad和Yourdon，1991a），形成了从一个模型到另一个模型无缝的转化（Jacobson等，1992）。例如，从面向对象分析转向面向对象设计需要用与实现有关的细节扩展分析模型，但不用开发整个新的表示。

本章介绍的面向对象数据建模是作为高级概念活动提出的。正如在第15章中将要介绍的一个好的概念模型对于设计和实现面向对象数据库应用的作用是无法估价的。

14.3 统一建模语言

统一建模语言（Unified Modeling Language，UML）是“一种用于说明、可视化和构造软件系统制品的语言，这种语言也能用于商业建模”（*UML Document Set*，1997）。Grady Booch、Ivar Jacobson和James Rumbaugh等三位一流专家通过努力，定义了一种面向对象建模语言，他们希望该语言在不久的将来能成为一个行业标准。UML建立在Booch（Booch，1994）、OOSE（Jacobson等，1992）和OMT（Rumbaugh等，1991）等方法以及其他一些先进方法的语义和概念之上，并且把它们统一起来。

UML符号对于用图形表示面向对象分析或设计模型是很有用的。UML不仅能说明系统的需求并捕捉设计决策，而且能够促进开发工作中关键人员之间的交流。开发者能够使用UML符号表示的分析或设计模型来与领域专家、用户和其他项目相关人员进行交流。

为了有效地表示一个复杂的系统，所开发的模型必须具有不同的视点或角度。通过提供不同类型的图，UML能够从多种角度表示系统，这些图包括用例图、类图、状态图、交互图、组件图和部署图。基本的模型集成了这些不同的视图，从而能以完整、一致的方式来加以分析、设计和实现系统。

由于本书是介绍数据库，所以将只论述类图，它只解决了系统的数据和某些行为方面的问题。而其他一些图提供的角度不直接与数据库系统有关，例如有关系统的动态方面，所以本书就不介绍了。但是应注意，数据库系统通常只是整个系统中的一部分，而整个系统的模型应该包括系统的各个方面。有关其他UML图的讨论，请看Hoffer、George和Valacich（2002）等人的著作。

14.4 面向对象数据建模

本节将介绍面向对象数据建模。在面向对象建模中涉及到的主要概念和技术有：对象和类；属性和操作的封装；关联、概化和聚合联系；基数和其他类型约束；多态以及继承。下面将介绍如何使用UML符号来开发类图，以提供被建模的系统的概念视图。

14.4.1 表示对象和类

在面向对象方法中，用对象来建模现实世界。在把这个方法应用到现实世界问题之前，应

该先理解什么是对象。**对象**（object）是一个实体，在应用领域内有定义明确的角色、状态、行为和标识。一个对象是一个概念、抽象，或在应用语境中有意义的事物（Rumbaugh等，1991）。一个对象可以是一个可触及或可见的实体（例如一个人、位置或东西），也可以是一个概念或事件（例如部门、性能、婚姻、注册等），还可以是设计过程的制品（例如用户界面、控制器、调度者等）。

你可能想知道，对象与第3章研究的ER模型中的实体有什么不同。显然，ER模型中的实体可以用对象模型中的对象来表示。但是除了存储状态（信息）之外，对象还通过能检查或影响状态的操作表现它的行为。

对象的**状态**（state）包含它的性质（属性和联系）以及这些性质具有的值。对象的**行为**（behavior）表示对象怎样动作和如何响应（Booch，1994）。对象的状态是由它的属性值以及与其他对象的链接决定的。对象的行为取决于它的状态和执行的操作。一个操作就是一个动作，表示一个对象为了获得响应而在另一个对象上执行的动作，也可以将操作看作一个对象（供应者）提供给他的客户的一个服务。客户发送消息给供应者，而供应者通过执行相应的操作提供所需的服务。

考虑一个有关学生的例子，Mary Jones可以看成一个对象。这个对象的状态用属性和其当前值来刻画，其中属性有姓名、出生日期、年级、地址和电话，这些属性都有相应的值。例如，其姓名是“Mary Jones”，年级是“junior”（三年级）等等。对象的行为是通过操作表示的，例如操作calc-gpa用来计算一个学生的当前成绩绩点的平均值。因此Mary Jones对象把她的状态和行为封装在一起。

所有的对象都有一个标识，也就是说，没有两个对象是相同的。例如，有两个学生有同样的姓名和出生日期，但他们本质上仍是两个不同的对象。即使他们的所有属性都有相同的值，但仍有不同的标识。同时，每个对象在它的生命周期内保持自己的标识不变。例如，如果Mary Jones结婚了，其姓名、地址和电话改变了，但她仍然是同一个对象。

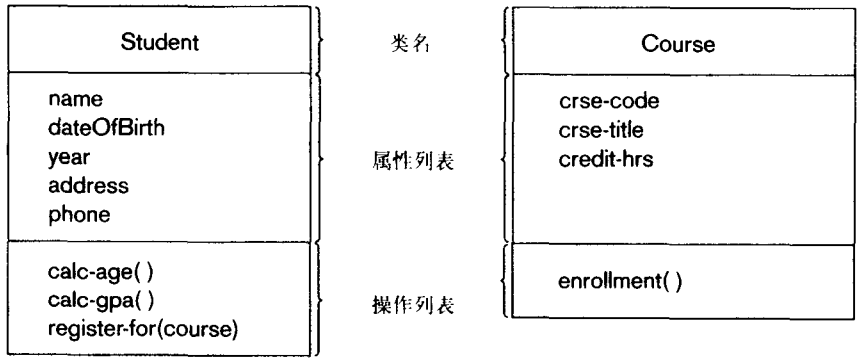
术语“对象”有时指的是一组对象，而不是单个的对象。这可以从语境中来区分。但严格地说，“对象”是指单个对象，而不是一类对象。下面我们再对此作解释。如果要消除可能的混淆，那么可使用“对象实例”来表示单个的实体，而用**对象类**（object class，或直接用**类**）表示共享公共结构和公共行为的一组对象（就像第3章中的实体类型和实体实例）。在上面的例子中，Mary Jones是一个对象实例，而所有学生（Student）是一个对象类。

可以用图14-2a的类图来描述类。**类图**（class diagram）表示面向对象模型的静态结构，其中包括对象类，对象类的内部结构和对象类参与的联系。在UML中，一个类用一个矩形框表示，矩形框用水平线分隔成三个部分。类的名字出现在最上面一部分，属性列在中间一部分，操作放在最下面一部分。这个图表示了两个类（Student和Course）以及它们的属性和操作。

Student类是共享公共结构和公共行为的一组Student对象。所有学生有公共的性质：姓名、出生日期、年级、地址和电话。它们还通过共享calc_age、calc_gpa和register_for（course）操作表示公共的行为。因此，一个类为类实例提供了模板或模式。每个对象知道它应属于哪个类。例如，Mary Jones对象知道它应属于Student类。属于同一个类的对象可以参与与其他对象类似的联系。例如，所有学生都要选修课程，因此Student类要参与与Course类的联系“register_for”（参见14.4.3节）。

对象图（*object diagram*）也称为实例图，它是与给定类图相兼容的实例图。图14-2b表示有两个实例的对象，分别表示图14-2a中出现的两个类的一个实例。如图所示，一个静态对象图是类图的一个实例，提供了系统在某一时刻的详细状态的快照（*UML Notation Guide*，1997）。

这里提供的只是静态对象图的例子，而不是动态对象图（称为协作图）的例子。



a) 表示两个类的类图



b) 有两个实例的对象图

图14-2　UML类图和对象图

在对象图中，每个对象用分隔成两部分的矩形框表示。对象的名字和对象所属的类被加上下划线，放在矩形框的顶部，其句法如下:

对象名: 类名

对象的属性和值放在第二部分。例如，有一个对象称为Mary Jones，它属于Student类。姓名、出生日期和年级等属性值也在图中标出。不感兴趣的属性值可以不标出。例如，Mary Jones的地址、电话属性就未标出。如果对对象的所有属性都不感兴趣，那么整个第二部分可被取消。对象的名字也可省去，但冒号和类名仍应保留，如图中Course的实例所示。如果标出了对象名，那么其后的冒号以及类名也可省去。

操作（operation）是由类的所有实例提供的函数或服务，例如Student中的calc_gpa（参见图14-2a）。只有通过这样的操作，其他对象才能访问或操纵存储在这个对象中的信息。因此，操作给类提供了外部接口；这个接口表示类的外部视图，而不显示它的内部结构或它的操作如何实现。这种在外部视图中隐藏对象的内部实现细节的技术称为**封装**（encapsulation）或信息隐蔽（Booch，1994；Rumbaugh等，1991）。在我们提供公共行为的抽象给接口中类的所有实例时，在类之间封装了类的结构和应有行为的秘密。

14.4.2　操作的类型

根据客户需要的服务种类，操作可以分成四种类型：构造器（constructor）、查询（query）、更新（update）和范围（scope）（*UML Notation Guide*，1997）。**构造器操作**（constructor *operation*）*创建类的一个新实例*。例如，在Student中有create-student操作，用于创建一个新的学生并初始化其状态。这种构造器操作对所有类都是适用的，因此在类图中就不必明确地标出。

查询操作（query operation）是没有任何副作用的操作。这个操作能访问一个对象的状态，

但不能改变其状态（Flowler，2000；Rumbaugh等，1991）。例如，Student类有一个操作get-year（未标出），用于检索查询中指定的Student对象的年级（新生、二年级、三年级、四年级）。由于这个操作检索基属性的独立值，因此在类图中不必明确表示get-year这个查询操作。再考察Student中的calc-age操作，这也是一个没有任何副作用的查询操作。这个查询惟一的参数是目标Student对象。这样的查询可以表示成导出属性（Rumbaugh等，1991）。例如，可以将“age”表示为Student的导出属性。由于目标对象总是一个隐式操作变量，因此在操作声明中不必明确表示出来。

更新操作（update operation）有副作用，它要改变对象的状态。例如，考虑Student的操作promote-student（未标出）。这个操作是让学生升到新的年级，例如从三年级升到四年级，因而改变了Student对象的状态（年级属性的值）。另外一个更新操作例子是register-for（course），这个操作建立Student对象与特定的Course对象之间的连接。要注意，这个操作除了把目标Student对象作为隐式变量外，还有一个显式变量“course”，它指定学生想要注册的课程。显示变量用括号括起来。

范围操作（scope operation）是应用于类而不是应用于对象实例的操作。例如，适用于Student类的avg_gpa（在图14-2中，这个类的操作中未标出）计算所有学生的平均gpa值（用下划线标出操作名表示它是一个范围操作）。

14.4.3 表示关联

与ER模型中联系的定义一样，**关联**（association）是指两个或多个对象类的实例之间的已命名的联系。与ER模型一样，一个关联联系的度数可以是一元、二元、三元或多元（n元）。图14-3使用了图3-12的例子来说明怎样用面向对象模型来表示不同度数的关联联系。一个关联用参与类之间的实线表示。关联端（也就是与类相连的端）称为**关联角色**（association role，*UML Notation Guide*，1997）。每个关联有两个或多个角色。每个角色可以在靠近关联端的地方用标号来显示命名（参见图14-3a中的“manager”角色）。角色名指出这个类在关联中所起的作用。角色名的使用是可选择的。你也可以在关联名处指定角色名。

图14-3a表示两个一元关联，即Is-married-to（婚姻）和Manages（管理）。在Manages联系的一端，有名为“manager”的角色，表示一个职员可以担任经理的角色。而其他角色没有命名，但是对关联已命名了。在角色名没有出现时，可以把与端部相连的类的名字作为角色名。（Fowler，2000）。例如，在图14-3b中，Is-assigned联系右端的角色就以类名Parking Place来称呼。

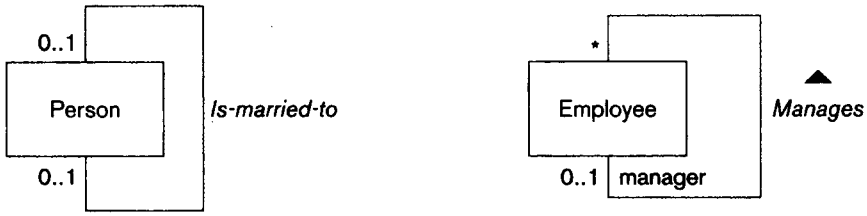
每个角色都有**多重性**（multiplicity），多重性是指在一个给定的联系中有多少对象参与。在类图中，多重性规范以用整数区间表示的文本串表示，其格式如下：

下界..上界

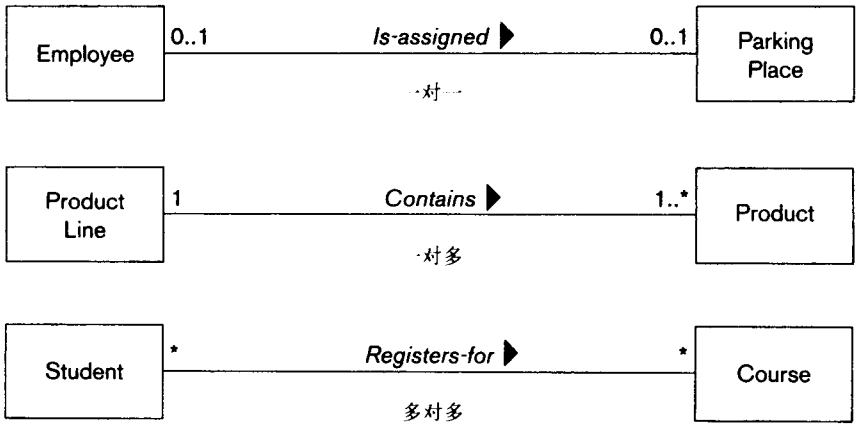
这个区间是一个闭区间，表示范围既包括上界也包括下界。例如，多重性2..5表示在一个给定的联系中可参与对象的数目的最小值为2，最大值为5。因此，多重性只不过是第3章中讨论过的基数约束。多重性的上界除了整数以外，还可以是星号“*”，该符号表示一个无穷大的上界。如果只指定一个整数值，那么表示范围只能取这个整数值。

实际上，最常用的多重性是0..1、*和1。多重性0..1表示最小值是0和最大值是1（可选一），而*（或0..*）表示范围从0到无穷大（可选多）。而单个1代表1..1，表示联系中参与的对象数目恰好是1（强制1）。

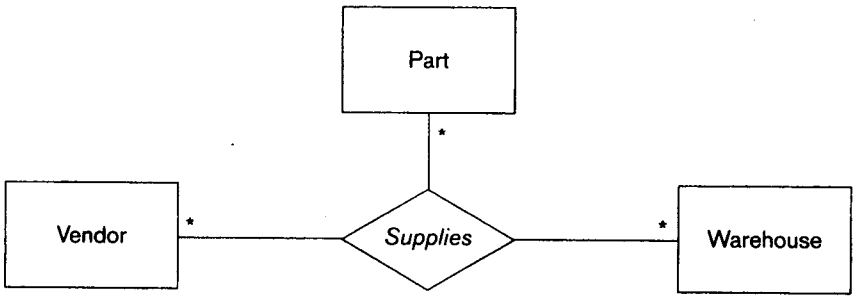
在Is-married-to联系中，任一角色的多重性都是0..1，表示每个人可以是单身，也可以与另一个人结婚。对于Manages联系中的manager角色的多重性是0..1，而另一个角色是*，表示每个职员只能由一个经理管理，但一个经理可管理多个职员。



a) 一元联系



b) 二元联系



c) 三元联系

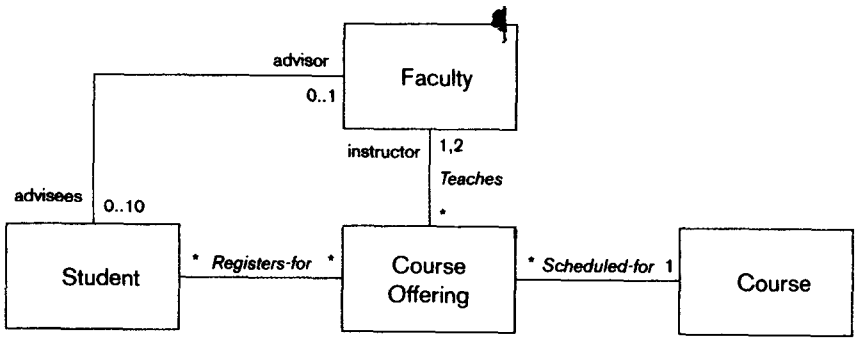
图14-3　不同度数的关联联系的例子

图14-3b表示三个二元联系，即Is-assigned（一对一）、Contains（一对多）和Registers-for（多对多）。虽然在类图中，关联名可以沿着一个方向读，但是二元关联是固有的双向联系（Rumbaugh等，1991）。例如，Contains关联可读成从Product Line到Product的关联（注意，在此例中，可以在关联名上加个实心三角形明确表示方向）。但是，Contains隐含着一个相反的遍历Belongs-to，它表示一个产品属于某个产品系列。这两个遍历的方向提供了相同的基本关联；关联名可直接在一个方向上建立。

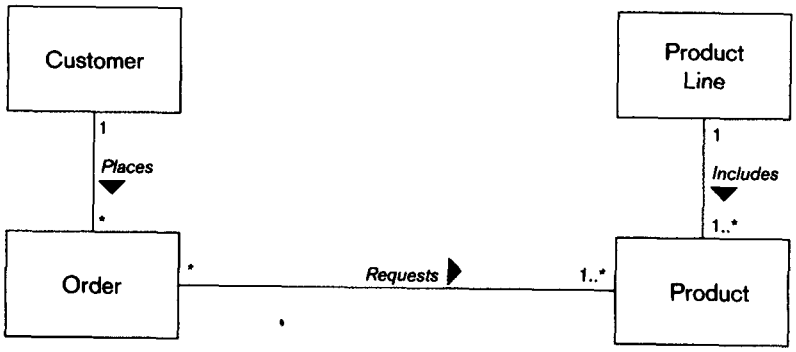
Is-assigned联系的图表示给每个员工分配一个停车位或根本不分配停车位（可选一）。从另一个方向读，可以说每个停车位被分配给一个员工或者根本不分配给员工（也是可选一）。类似地，每个产品系列中可以有多个产品，但至少有一个产品，反之，每个产品只能属于一个产品系列（强制一）。第三个二元关联图说明每个学生可注册多门课程，但也可能根本不注册课程，而一般每门课程有许多学生注册（两个方向都为可选多个）。

在图14-3c中，在Vendor、Part和Warehouse之间存在着一个三元联系Supplies。与ER图一样，用菱形符号表示三元联系，并在其中填上联系的名字。与第3章讨论的一样，联系是多对多对多，并且不可以用三个二元联系替换，否则势必造成信息丢失。

图14-4a中的类图表示如下二元关联：Student和Faculty之间的关联，Course和Course Offering之间的关联，Student和Course Offering之间的关联，以及Faculty和Course Offering之间的关联。这个类图表示一个学生可以有一个导师，而一个导师可指导多达10个学生。另外，一门课程可以有多个设置，而一个课程设置恰好对应一门课程。UML可以用数值来说明任何多重性。例如，图中表示每个课程设置可以由1名或2名讲师（1，2）授课。可以使用一个数值（例如用2表示桥牌队的成员数目）、范围（例如用11..14表示参与足球比赛的人数）或数值与范围的离散集（例如用3、5、7表示委员会成员的人数，用20..32、35..40表示公司里每个职员每周用小时计的工作量）。



a) 大学的例子



b) 顾客订单的例子

图14-4　二元关联联系的例子

图14-4a还表示教师扮演讲师的角色，也扮演着导师的角色。导师（advisor）角色标识着与Student对象关联的Faculty对象，指导（advisees）标识着与一个Faculty对象关联的一个Student对象集。也可以命名这个关联为Advises，但此时，该角色名能充分表示联系的语义。

图14-4b表示顾客订单的另一个类图。图14-5是相应的对象图，图中显示了类的若干实例及其实例之间的链接情况（注意，每个实例属于一个类，每个链接属于一个联系）。在本例中，两个顾客Joe和Jane下了若干订单。Joe下了2个订单：Ord 20和Ord 56。在Ord 20中，Joe订购运动产品系列中的产品P93。在Ord 56中，他再次订购同样的运动产品以及硬件产品系列中的产品P50。Jane订购了和Joe同样的硬件产品，还有两个化妆品产品系列的产品（P9和P10）。

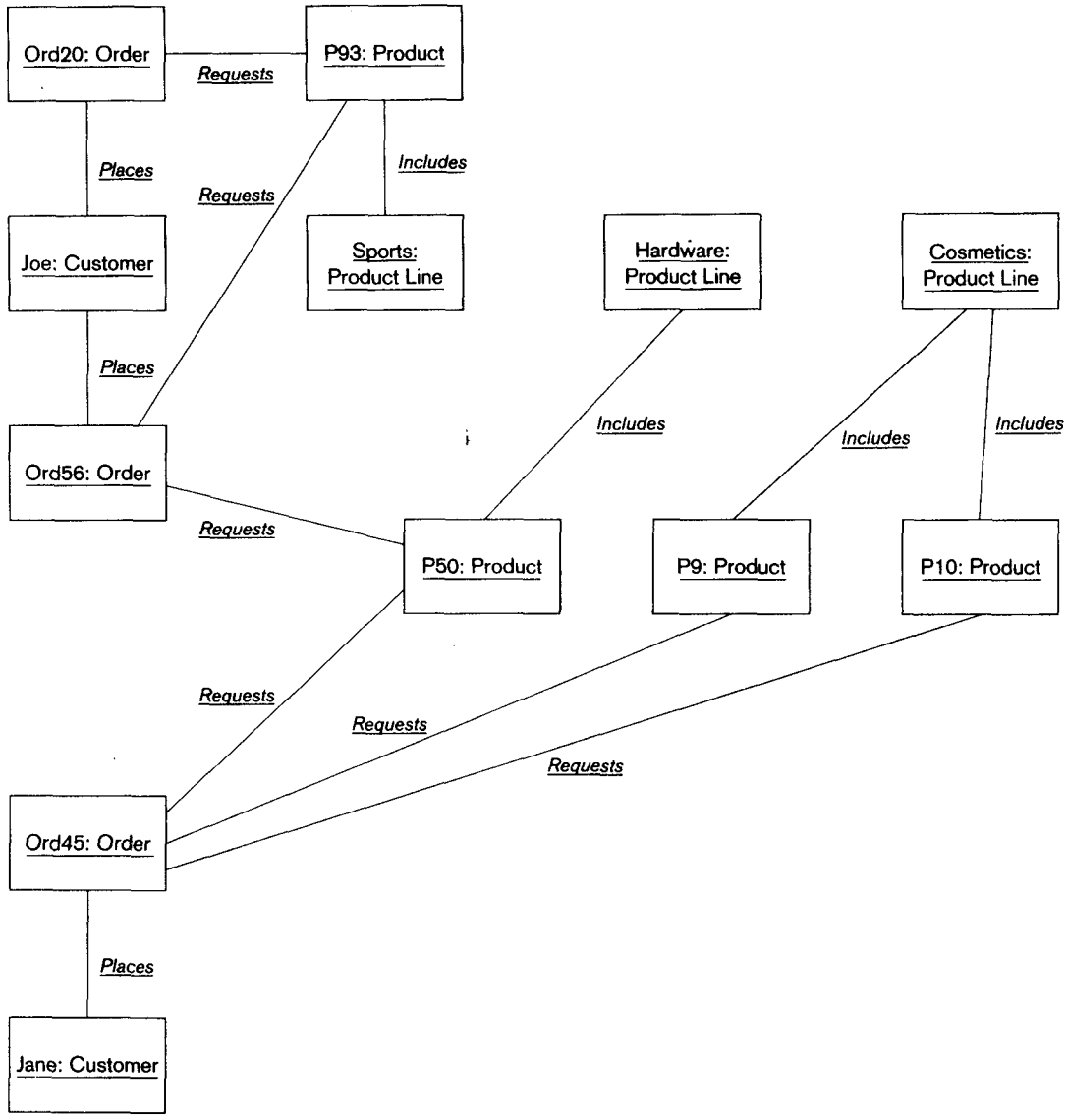
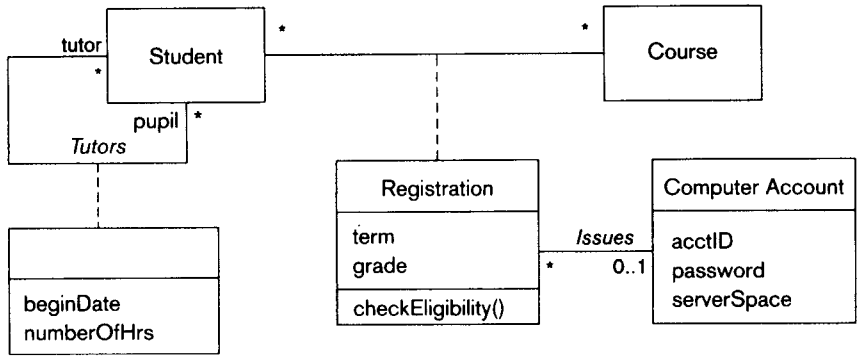


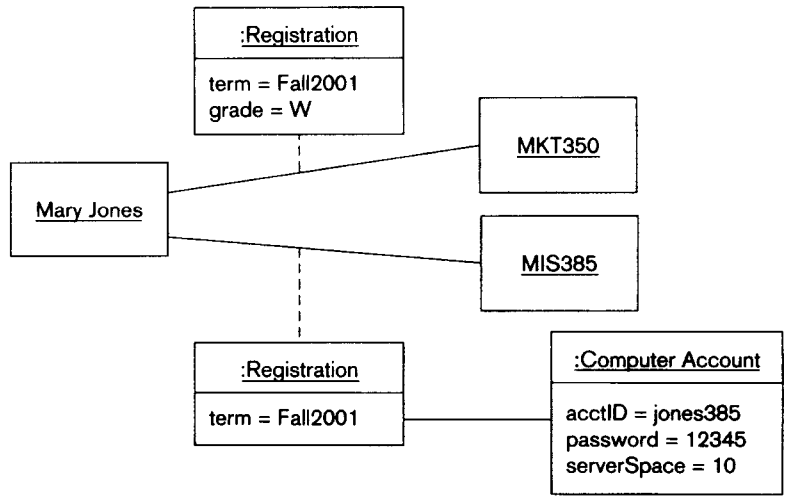
图14-5　顾客订单例子的对象图

14.4.4　表示关联类

当关联本身也有属性或自己的操作，或者关联参与其他类的联系时，把关联建模成**关联类**（association class）是很有用的（就像第3章中使用的“关联实体”）。例如，在图14-6a中，属性term和grade实际上属于Student和Course之间多对多关联。除非学生和课程都已知，否则学生某门课程的成绩是不能确定的。类似地，要找学生在哪个学期选修该门课程，也必须知道学生和课程。checkEligibility操作用于测试学生是否有资格注册给定的课程，这个操作属于关联，但不属于参与这个关联的两个类中任何一个类。我们还发现，对于某门课程注册，会给学生提供一个计算机账号。因此，可以把Registration建模成一个关联类，这个关联有它自己的一组特性，并与其他类（Computer Account）有一个关联。类似地，对于一元关联Tutors、beginData和numberOfHrs（授课时数）实际上属于这个关联，因此应出现在单独的关联类中。



a) 表示关联类的类图



b) 表示链接对象的对象图

图14-6 关联类和链接对象

关联类的名字可以在关联路径上表示，也可用类符号表示，也可以两种方法都使用。当关联只有一个属性但没有任何操作，也不参与其他关联时，可在关联路径上标出名字，并在关联类符号中省略属性，以便强调它的“关联本质”（*UML Notation Guide*，1997）。这就像Tutors关联所表示的那样。另一方面，Registration关联有两个属性和一个它自己的操作，还有一个与Computer Account的关联Issues，这个关联的名字在类矩形框中表示，以强调它的“类本质”。

图14-6b表示的是学生Mary Jones及她在2001年第一学期注册的课程MKT350和MIS385的对象图的一部分。与类图中关联类相对应，在对象图中也存在链接对象。在这个例子中，对于Registration关联类有两个有链接对象（表示成:Registration），有两门课程注册。这张图还显示，对于MIS385课程，Mary Jones得到带有账号、口令和指定服务器空间的计算机账户。她还没取得MIS385这门课的成绩，但对于MKT350课程，由于她取消了这门课因此获得了成绩W。

图14-7表示的是Student、Software和Course这三个类之间的一个三元联系。它表示这样一个事实，学生对于不同的课程使用了各种软件工具。例如，需要存储这样的信息，Mary Jones对于Database Management这门课程使用了Microsoft Access和Oracle软件，对于Object-Oriented Modeling课程使用Rational Rose和Visual C++软件，对于Expert System课程使用了Level5 Object

软件。假定我们现在想要知道Mary每周在Database Management课程中使用Oracle软件花费的小时数。这个处理实际上属于三元关联，而不属于某个类。因此，可创建一个称为Log的关联类，在这个关联类中再声明一个称为estimateUsage的操作。除了这个操作外，还指定属于这个关联的三个属性：beginData、expiryData和hoursLogged。

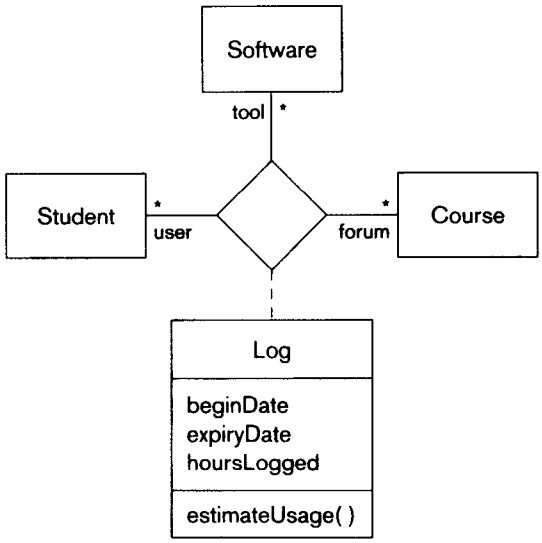


图14-7　带有关联类的三元联系

14.4.5　表示导出属性、导出关联和导出角色

导出属性、导出关联和导出角色是分别可从其他属性、关联和角色计算或导出的（导出属性的概念在第3章已介绍过）。导出元素（属性、关联和角色）在元素名前加一个斜线（/）来表示。例如在图14-8中，由于年龄可利用出生日期和当前日期计算出来，因此age是Student的一个导出属性。由于计算是在对象类上的一个约束，因此该计算可以在图中Student对象类的上方用{ }标出。还有，Student和Course之间的Takes联系也是可导出的，这是因为Takes联系可以从Registers-for和Scheduled-for这两个联系推导出来。同样，参与也是可从其他角色导出的角色。

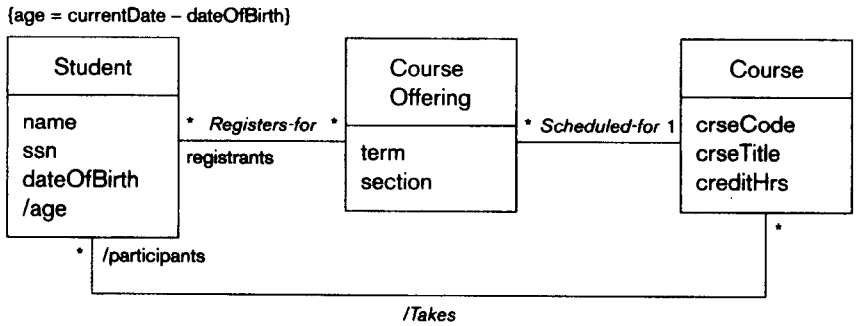


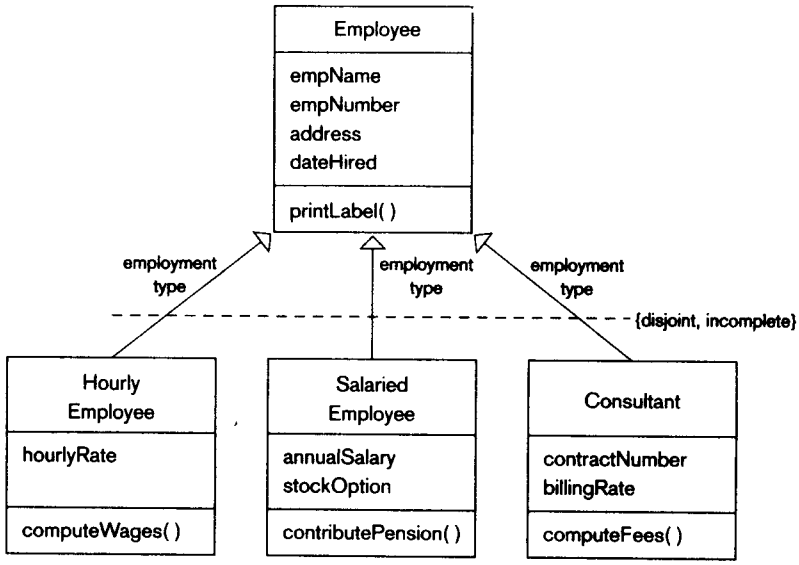
图14-8　导出属性、关联和角色

14.4.6　表示概化

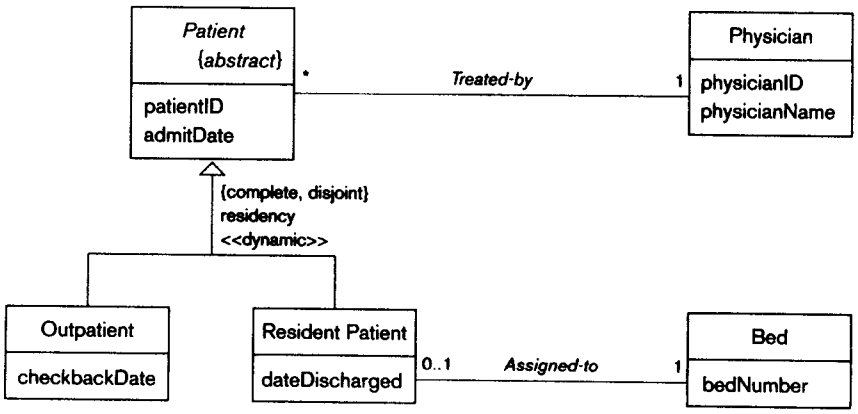
第4章已介绍过概化和特化的概念。使用扩展的ER模型，我们已经学习了怎样把两个或多个实体的公共属性以及这些实体参与的公共联系抽象成更一般的实体超类型，同时把不是公共的属性和联系仍留在本身的实体（子类型）中。在面向对象模型中，我们可以利用这些表示法，

但有一点不同。在将一个对象类集概化成更一般的类时，我们不仅要抽象公共的属性和联系，还要抽象公共的操作。一个类的属性和操作被称为类的特性。被概化的类称为子类，而概化后的类称为超类，这与EER图中的超类型和子类型完全对应。

考察图14-9a的例子（参见图4-8中相应的EER图）。有三种员工：钟点工、受薪员工和顾问。所有员工都共享的特性有empName、empNumber、address、dateHired和printLabel，这些属性都存储在Employee超类中，而特定员工特有的特性存储在相应的子类中（例如Hourly Employee的hourlyRate和computeWages）。在表示概化路径时，从子类到超类画一条实线，在超类的一端画一个带空心的三角形（指向超类）。也可以将给定超类的一组概化路径表示成一棵与各个子类相连接的多分支树，共享部分用指向超类的空心三角形表示。例如，在图14-9b（对应图4-3）中，从Outpatient到Patient和从Resident Patient到Patient的两条概化路径合并成带指向Patient的三角形的共享部分。我们还可以指定这个概化是动态的，表示一个对象有可能改变子类型。



a) 带有三个子类的Employee超类



b) 带有两个子类的抽象Patient类

图14-9 概化、继承和约束的例子

可以在路径旁设置一个鉴别符，指出概化的基础。鉴别符（对应于第4章中定义的子类型鉴别符）指出通过特定的概化联系抽象出对象类的哪一个特性（Rumbaugh等，1991）。每次只能鉴别一个特性。例如，在图14-9a中，可以根据员工类型（小时工、受薪员工、顾问）鉴别出员工类。为了鉴别图14-9b中的一组概化联系，只需指定一次鉴别符。虽然可以把patient类分成Outpatient和Resident Patient两个子类，但是基于是否住院，在共享线旁边只需显示一次鉴别符标号。

子类的实例也是其超类的实例。例如，在图14-9b中，Outpatient实例也是一个Patient的实例。因此，概化称为一个Is-a联系。而且，子类继承了其超类的所有特性。在图14-9a中，Hourly Employee子类除了它自己特性hourlyRate和computeWages以外，还从Employee类中继承了empName、empNumber、address、dateHired和printLabel等属性。Hourly Employee的实例将存储Employee和Hourly Employee的属性值，在需要时，也可使用printLabel和computeWages操作。

概化和继承可以在超类/子类层次结构中的不同等级间转化的。例如，Consultant有一个子类Computer Consultant（计算机顾问），这个子类继承了Employee和Consultant的特性。Computer Consultant的实例也是Consultant的一个实例，因此也是Employee的一个实例。Employee是Computer Consultant的祖先，而Computer Consultant是Employee的子孙。使用这些术语是因为类的概化是跨越多个层次而实现的（Rumbaugh等，1991）。

继承是面向对象模型的另一个主要优点。继承可以使代码具有重用性，即程序员不必编写已在超类中编写过的代码。程序员只须编写对那些已存在类的新的、被精炼的子类来说具有惟一性的代码。实际上，面向对象程序员们先要访问他们各自的领域中的大型类库，从这些类库中找到可以被重用、被精化的类以满足新应用的需求。支持面向对象模型的人声称代码的可重用性可将生产率提高几个数量级。

注意，图14-9b中Patient类是斜体的，这表示它是一个**抽象类**（abstract class）。抽象类是一种没有直接实例，但它的子孙可以有直接实例的类（Booch，1994；Rumbaugh等，1991）。注意，也可以在类名下面用一对花括号，在括号中写上abstract字样。这在手工生成类图时特别有用。一个类也可有直接的实例（例如Outpatient或Resident Patient），这样的类称为**具体类**（concrete class）。在这个例子中，Outpatient和Resident Patient都可以有直接实例，但Patient没有它自己直接的实例。

Patient抽象类参与了一个与Physician类有关的联系Treated-by，这个联系表示所有的病人（包括门诊病人和住院病人）都要由医生来治疗。Resident Patient类除了这个继承的联系外，还有它自己专有的与Bed类相关的联系，这个联系称为Assigned-to，这表示住院病人都被分配了某张病床。这样，子类不仅要精化类的属性和操作，还要特化它参与的联系。

在图14-9a和14-9b中，“complete”、“incomplete”和“disjoint”放在花括号内，与概化相邻。这些词表示子类之间的语义约束（complete对应于EER表示法中的完全特化，而incomplete相应于部分特化）。在UML中，用逗号隔开的关键字表可像图14-9b那样放在共享的三角形旁边，也可像图14-9a那样放在与所有概括线相交的虚线旁边（*UML Notation Guide*，1997）。可使用下列UML关键字：overlapping、disjoint、complete和incomplete。根据UML Notation Guide（1997），这些术语的意义如下所述：

- *overlapping*（重叠）：子孙可以由多个子类转变而成（与EER图形表示法中的重叠规则一样）。
- *disjoint*（不相交）：子孙不可以由多个子类转变而来（与EER图形表示法中的不相交规则一样）。

- *Complete*（完备）：必须指定所有子类（不管是否标出）。不期望有其他的子类（与EER图形表示法中的完全特化规则一样）。
- *incomplete*（非完备）：某些子类被指定，然而是不完全的。有些子类在模型中未出现（与EER图形表示法中的部分特化规则一样）。

在图14-9a和14-9b中的概化都是不相交的。一个员工可以是一个小时工、受薪员工或顾问，但不可以同时兼任，例如同时既是受薪员工又是顾问。同样，一个病人可以是一个门诊病人或住院病人，但不可能同时是两者。图14-9a中的概化是非完备的（与图4-8中的显示相违背的），表示一个员工可以不属于这三种类型中的任何一种。此时，这个员工可以作为具体类Employee的实例存储。相反，图14-9b中的概化是完备的，表示一个病人必须是门诊病人或住院病人，不能有其他类型。因此，Patient被指定为一个抽象类。

图14-10是一个重叠约束的例子。这个图表示科研助教和教学助教都是研究生。重叠约束表示一个研究生可能既是科研助教又是教学助教。例如，研究生Sean Bailey每周要花12小时担任科研助教，要花8小时担任教学助教。还要注意，Graduate Student应指定成具体类，以便表示不担任助教的研究生。“level”鉴别符的概化线下面的省略号（…）不表示非完备约束。它表示图中未标出的模型的其他子类。例如，虽然模型中有Undergrad Student（本科生），但由于重点是在助教方面，因此该子类在图中省略了。还可以在空间有限时使用省略号。

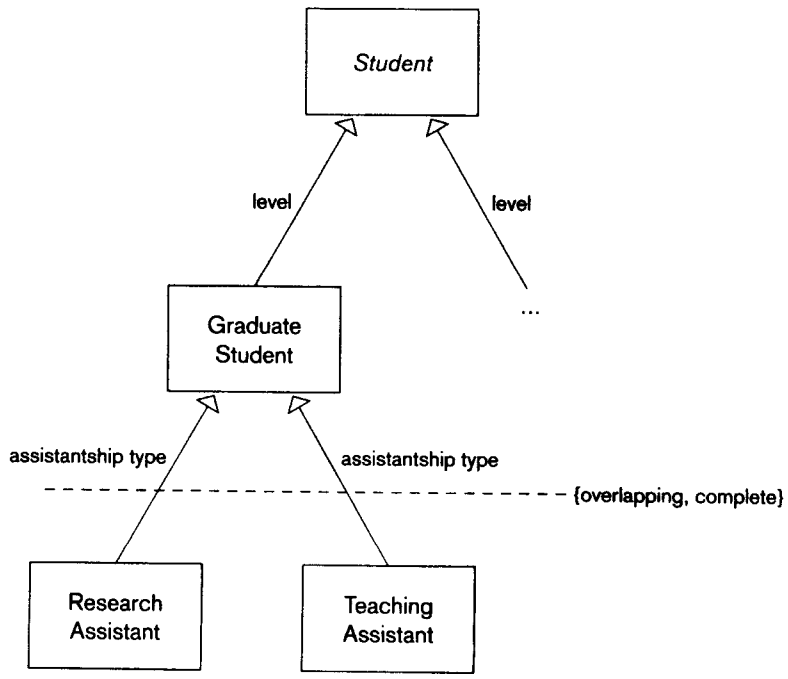


图14-10 重叠约束的例子

图14-11表示了在为学生记账建模开发的模型中表示的研究生和本科生。calc-tuition操作用于计算一个学生应该付的学费，这个总和取决于每贷款小时的学费（tuitionPerCred）、选修的课程以及每一门课程的贷款学时数目（credtHrs）。每贷款小时的学费还取决于学生是研究生还是本科生。在这个例子中，研究生的贷款小时学费是$300，而所有本科生的贷款小时学费是$250。应指出，用下划线标出了两个子类的tuitionPerCred属性与它的值。这样的属性称为**类范围属性**（class-scope attribute），指定一个值在整个类中都应一样，而不必为每个实例去指定一

个值（Rumbaugh等，1991）。

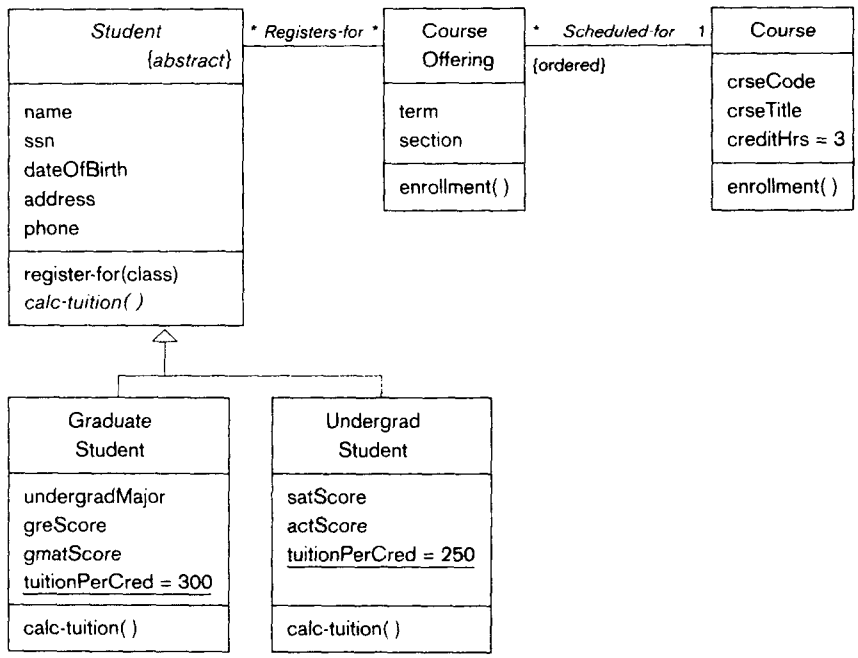


图14-11　多态、抽象操作、类范围属性和排序

也可以在属性名后使用“=”号来指定一个属性的初始默认值。这是新创建的对象实例的初始属性值。例如，在图14-11中，creditHrs属性的初始值为3，表示在创建新的Course实例时，其creditHrs值被置为默认值3。通过编写一个显式的构造器操作可以修改初始的默认值。也可以在以后通过其他操作来修改这个值。初始值规范与类范围属性之间的区别在于前者允许类的实例有不同的属性值，而后者强制所有实例共享一个共同的值。

除了指定关联角色的多重性以外，还可以指定其他性质。例如起该作用的对象是否排序等。在图中，Scheduled-for联系的Course Offering端放置了关键字约束“{ordered}”，用来表示对于给定课程的设置被排序成表，例如根据学期和班级来排序。很显然，在角色的多重性大于1时，排序是有意义的。在一个角色上默认的约束是“{unordered}”。也就是说，如果在角色边不指定关键字“{ordered}”，那么假定有关的元素形成一个未排序的集合。例如，课程设置与学生无关，学生能以任何顺序来注册设置的课程。

Graduate Student子类通过添加四个属性和精化继承的calc-tuition操作特化了抽象的Student类，这四个属性是undergradMajor、greScore、gmatScore和tuitionPerCred。注意，calc-tuition操作在Student类中用斜体表示，这意味着该操作是一个抽象的操作。**抽象操作**（abstract operation）定义了操作的形式或协议，但不是它的实现（Rumbaugh等，1991）。在本例中，Student类定义了calc-tuition操作的协议，但是不必提供相应**方法**（method，操作的实际实现）。协议包含了参数的数目和类型、结果类型和操作的预期的语义。这两个具体子类Graduate Student和Undergrad uate Student提供了calc-tuition操作本身的实现。注意，由于这些类是具体的，因此它们不能存储抽象操作。

还应指出，虽然Graduate Student和Undergraduate Student类共享同一个calc-tuition操作，但它们可能以不同的方式实现这个操作。例如，对研究生来说，实现这个操作的方法可能是对研

究生每选修一门课程就要添加一个专门的研究生费用。同样的操作可以以不同方式应用到两个或多个类上的现象称为**多态**（polymorphism），多态是面向对象系统中的关键概念（Booch，1994；Rumbaugh等，1991）。图14-11中的enrollment操作是多态的另一个例子。Course Offering中的enrollment操作计算特定的课程设置或班级的注册人数时，在Course之中的同名操作也计算给定课程的所有班级的注册人数。

14.4.7 解释继承和重载

我们已经看到，一个子类怎样增加从它祖先那里继承的特性。此时，称子类使用了*扩展的继承*。另一方面，如果一个子类约束了祖先的属性或操作，那么称子类使用了*约束的继承*（Booch，1994；Rumbaugh等，1991）。例如，名为Tax-Exempt Company的子类可以取消或屏蔽从其超类Company继承的操作compute-tax。

一个操作的实现可以是重载的。**重载**（overriding）是用子类中更特定的方法的实现来替换从超类中继承的方法的过程。重载的原因涉及扩展、约束和优化（Rumbaugh等，1991）。新操作的名字和所继承的操作的名字相同，但必须在子类中明确指出这个操作是重载的。

*在扩展重载*中，通过加入某些行为（代码）扩展子类从超类继承的操作。例如，Foreign Company是Company的一个子类，它继承了称为compute-tax的操作，但通过加入外国增付费用来计算总的税额而扩展了继承的行为。

*在约束重载*中，子类中新操作的协议是受约束的。例如，Student中有一个称为place-student（job）操作，在其子类International Student中通过对参数job限制而受到了约束（参见图14-12）。一般的学生在夏季可安排所有类型的工作，而留学生由于受到限制只能从事校园内的工作。新的操作通过对job参数加以限制重载了继承的操作，限制job值只能是所有可能工作的一个比较小的子集。这个例子还说明了多个鉴别符的使用方法。在概化集的基础是学生的“level”（研究生或本科生）时，另一个概化集的基础是他或她的“residency”状态（美国的或外国的）。

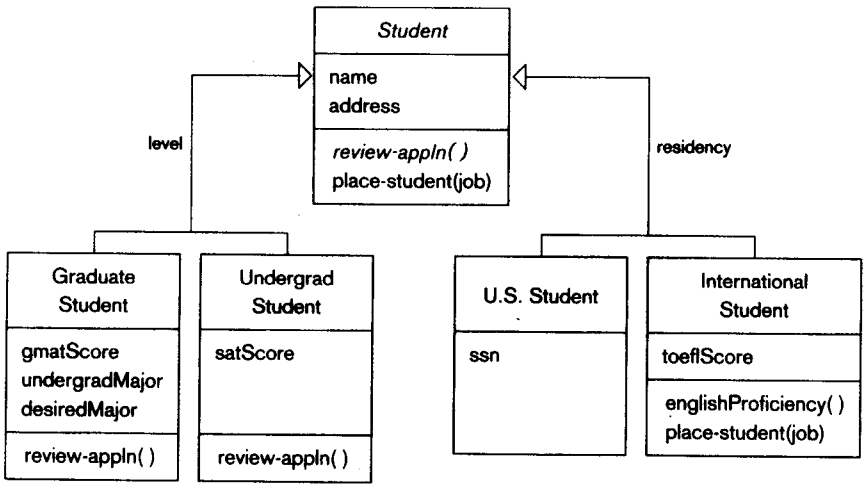


图14-12 重载继承

*在优化重载*中，通过利用由子类拥有的约束，新操作与改进的代码一起实现。例如，称为Dean's List Student的Student的子类表示所有处于Dean's List中的学生。为了有资格进入Dean's List，学生的绩点平均值必须大于等于3.50。假设Student有一个操作mailScholApps，该操作把优等和中等奖学金发给绩点平均值大于等于3.00，并且家庭总收入低于20 000美元的学生。

Student中这个操作的方法必须检查这个条件。另一方面，Dean's List Studen子类中同样操作的方法通过从源代码中消除第一个条件而改进执行的速度。考虑称为findMinGpa的另一个操作，该操作在学生之间找最小的绩点平均值（gpa）。假定Dean's List Student类以gpa升序方式排序，但Student类不是这样。Student中findMinGpa的方法必须在所有学生中执行顺序搜索。相比之下，Dean's List Student中的同样操作可以用列表中检索第一个学生的gpa的方法实现，因此不再需要费时的搜索。

14.4.8 表示多重继承

前面已研究了单重继承，即一个类只从一个超类处继承。正如在科研助教和教学助教的例子中看到的那样，有时一个对象可能是多个类的实例。这种情况称为**多重分类**（multiple classification，参见Fowler，1997；*UML Notation Guide*，1997）。例如，Sean Bailey是具有两种身份的助教，他有两个类别：一个是Research Assistant的实例，另一个是Teaching Assistant的实例。但是专家不主张使用多重分类。普通的UML语义和许多面向对象语言都不支持多重分类。

为了避免这个问题，可以使用多重继承，该方法允许一个类从多个超类那里继承特性。例如，在图14-13中，可创建Research&Teaching Assistant类，它既是Research Assistant也是Teaching Assistant的子类。既是科研助教又是教学助教的学生可以存储在这个新类下。现在可以把Sean Bailey表示成属于Research&Teaching Assistant类的一个对象，它从两个父亲那里继承了特性，例如从Research Assistant那里继承了researchHrs和assisgnProject（proj）属性，并且从Teaching Assistant那里继承了teachingHrs和assignCourse（crse）属性。

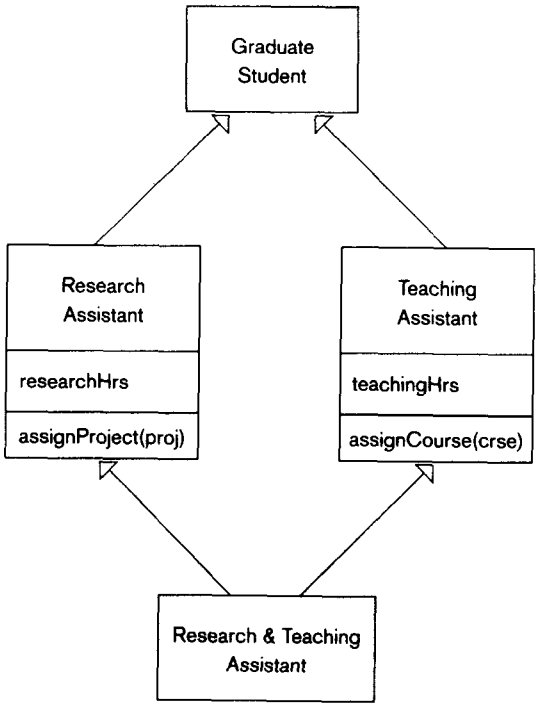


图14-13　多重继承

14.4.9 表示聚合

聚合（aggregation）表示了组件对象和聚合对象之间的Part-of（一部分）联系。这是一种

较强形式的关联联系（具有附加的“part-of”语义），在聚合的一端用空的菱形表示。例如在图14-14中，个人计算机是CPU（多处理器可达4个）、硬盘、显示器、键盘和其他对象的聚合。注意，聚合涉及一组不同对象实例，每个聚合包含了其他实例，或由其他实例组合而成。例如，Personal Computer对象与CPU对象有关，CPU对象是Personal Computer对象的一部分。相比之下，概化与对象类有关，即一个对象（例如Mary Jones）同时也是它所在类（例如Undergrad Student）的一个实例，以及它的超类（例如Student）的一个实例。只有一个对象（例如Mary Jones）包含在概化联系中。这就是为什么在聚合线的端点处要指出多重性，而概化联系中不必指出多重性的原因。

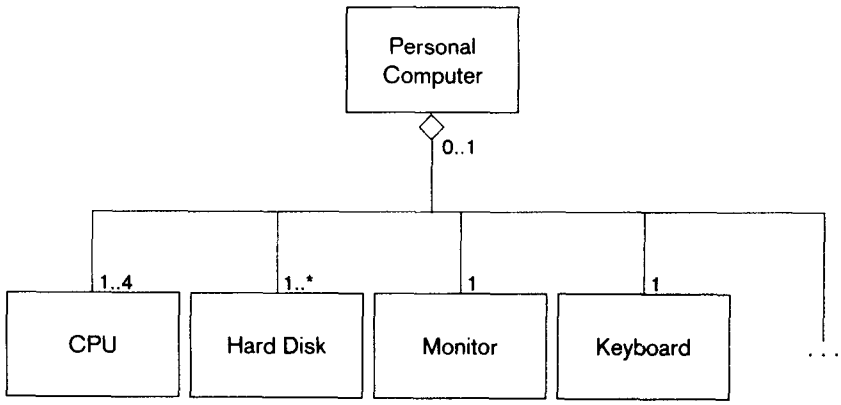


图14-14 聚合的例子

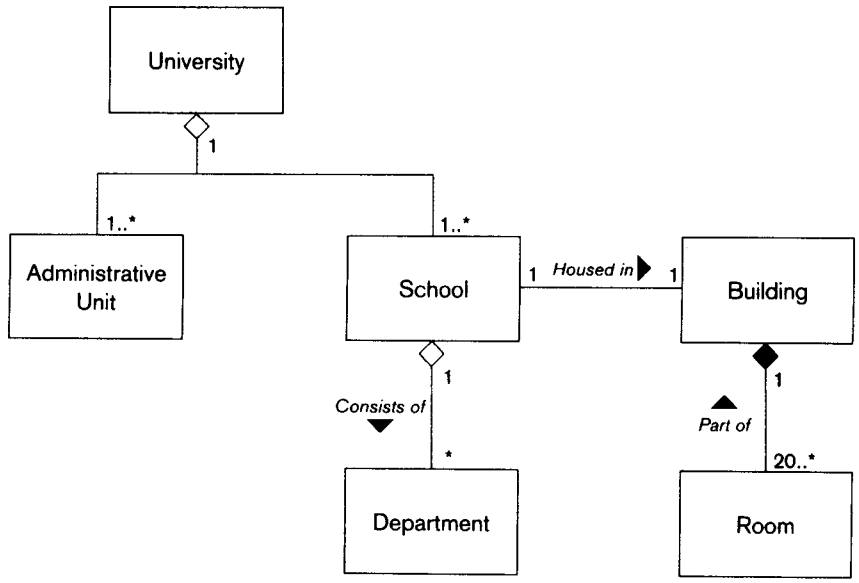
图14-15a表示了大学的聚合结构。图14-15b的对象图表示University的对象实例Riverside University怎样与它的组件对象（表示它的管理单位，例如Admissions、Human Resources等）和学院（例如Arts&Science、Business等）联系。一个学院对象（例如Business）也可由几个系对象组成（例如Accounting、Finance等）。

注意，在Building和Room之间联系的一端的菱形不是空心的，而是实心的。实心的菱形表示一种较强形式的聚合，称为复合（Fowler，2000；*UML Notation Guide*，1997）。在**复合**（composition）中，一部分对象只属于一个整体对象。例如，一个房间只是一幢大楼中的一部分。因此，聚合端的多重性可以超过1。部分可以在创建整个对象后才创建。例如，房间可以附加到已存在的大楼中。但是，一旦创建复合的一部分以后，这一部分就与整体共存亡；删除聚合对象会将它的组件一起删除。如果一幢大楼被摧毁，那么所有房间也随之消失。但是，在聚合删除前就有可能将其中一部分删掉，就像没有摧毁大楼但却摧毁一个房间。

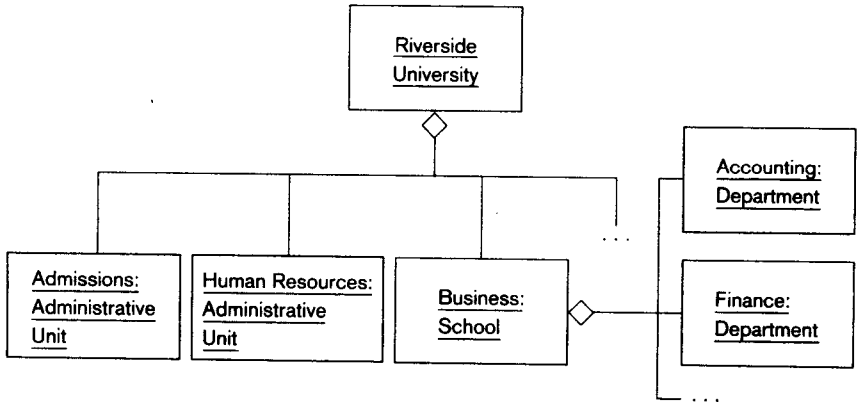
考虑另一个聚合例子，即在第3章就已提到的材料单结构。许多制造出的产品由部件制成，这些部件又由子部件和零件组成，等等。我们已看到，在ER图（参见图3-13）中这种类型的结构可以表示成多对多的一元联系（称为Has_components）。若联系也有它自己的属性，例如Quantity，那么联系也可以转换成一个关联实体。虽然材料单结构本质上是一种聚合，但由于ER模型在语义上不支持较强的聚合概念，因此只能将该结构表示成一个关联。在面向对象模型中，我们能明确地显示这个聚合。

图14-16表示了材料单结构。为了区别部件和元件（不能再分），我们创建两个类Assembly和Simple Part，这两个类都是Part类的子类。这个图表示一个产品由多个零件组成，而零件本身又是由其他零件组成，等等；这是一个*递归*聚合的例子。由于Part被表示成抽象类，因此一个零件可以是一个部件也可以是一个元件。一个Assembly对象是Part超类的实例的聚合，表示

它由其他部件（可选择的）和元件组合而成的。注意，我们可以轻松地捕捉附加到聚合联系的关联类的属性，如一个组件中零件的数量。



a) 类图



b) 对象图

图14-15　聚合和复合

在不能确定两个对象之间的联系是一个关联还是一个聚合时，应弄清楚一个对象是否真正是另一个对象的一部分。也就是说，存在整体/部分联系吗？注意，一个聚合不一定表示物理包含，例如Personal Computer和CPU。整体/部分联系可以是概念性的，例如组合基金和属于该基金的一部分的股票之间的联系。在聚合中，可能存在（也可能不存在）独立于聚合对象的对象。例如，一个股票是否存在与它是否是组合基金的一部分无关，而一个部门的不存在也与一个组织无关。而且，一个对象可能是几个聚合对象的一部分（例如许多组合基金可以包含IBM股票）。但是应记住，在聚合中，复合不允许一个对象是多个聚合对象的一部分。

聚合的另一个特点是整体上的某些操作可自动地应用于它的部分上。例如，在Personal

Computer对象类上有一个称为ship（发货）的操作，它也可以应用于CPU、Hard Disk和Monitor等，这是因为只要计算机发货，其零件也随之发送出去。也可以说，在Personal Computer上的ship操作被传播到它的零件（Rumbaugh等，1991）。

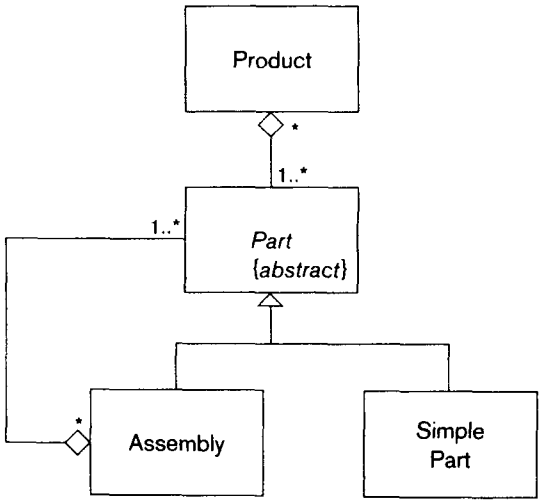


图14-16 递归聚合

14.5 业务规则

在第3和第4章就曾详细地讨论过业务规则。前面已讨论过怎样用ER图表示不同类型的规则。在本章的例子中，我们已看到许多业务规则，隐式或显式地作为类、实例、属性、操作和联系等的约束。例如，怎样在关联角色上确定基数约束和排序的约束。还看到怎样在子类之间表示语义约束（例如重叠、不相交等）。本章已讨论过的许多约束在使用时都用到了花括号，其中包含UML关键字，例如{disjoint，complete}和{ordered}，它们被放在使用约束的元素的旁边。例如图14-11中表示了对给定课程排序的业务规则。但是如果你使用预定义的UML约束不能表示一个业务规则，那么你可以用英语或形式逻辑那样的语言来定义规则。

在确定涉及到两个图形符号（例如表示两个类或两个关联）的业务规则时，可以用从一个元素到另一个元素的虚线箭头表示，并在花括号中标上约束名来表示（*UML Notation Guide*，1997）。图14-17表示这样一个业务规则：系主任必须是系中的一个成员，这是通过指定Chair-of关联是Member-of关联的子集实现的。

当业务规则涉及到三个或更多的图形符号时，可以用一个注释表示这个约束，并把这个注释用虚线连到每个符号上（*UML Notation Guide*，1997）。图14-17表示了这样一个业务规则："被指派教一门课程的某个班级的教师必须有资格教授这门课程"，并且该规则标记在注释符号中。由于这个约束涉及到这三个关联联系，因此要把注释连接到这三个关联路径上去。

14.6 对象建模实例：松谷家具公司

在第3章，我们已学过怎样开发松谷家具公司的高级ER图（参见图3-22）。通过对公司业务过程的研究，确定了实体类型、键和其他重要属性。现在说明如何使用面向对象模型来开发同样应用的类图。类图在图14-18中表示。我们讨论这张类图和ER图之间的相同和不同之处。

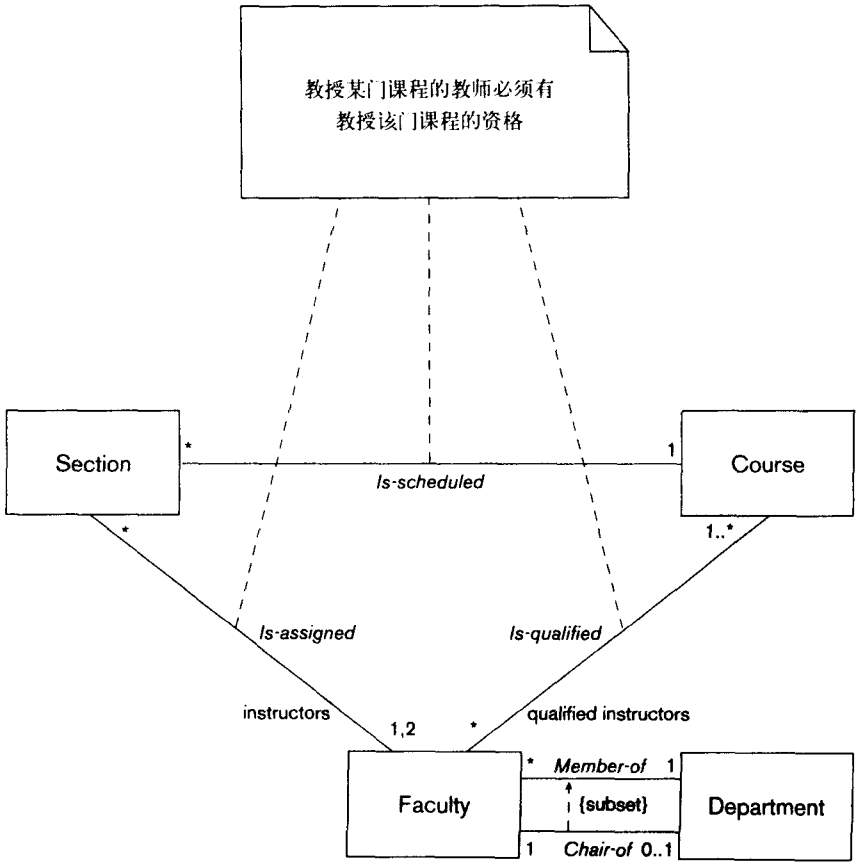


图14-17　表示业务规则

正如期望的那样，实体类型表示成对象类，所有属性都在类中表示。但是应注意，不能用主键形式来表示显式标识符，这是因为根据定义，每一个对象有它自己的标识。由于ER模型以及关系数据模型（参见第5章）没有其他支持标识的概念，因此需要显式地说明标识符。在面向对象模型中，应该表示的标识符是现实世界中有意义的属性，例如salespersonID、customerID、orderID和productID，这些在类图中都表示成键。注意，在Product Line中未标出标识，因为假定Product_Line_ID被作为内部标识符包含在ER图中，而不是像orderID或productID那样作为现实世界属性。如果松谷家具公司实际上不使用vendorID或任何其他属性来支持它的业务处理，那么在类图中就不包含这个属性。为此，也不必为Vendor、Order Line和Skill等类标出标识符。

在ER图中，“Skill”是Employee的多值属性。在类图中，Skill被表示成独立的对象类，这是因为根据Fowler（2000）的观点，在概念面向对象模型中，一个属性总是应该是单值的。Order Line被表示成一个关联类，即“Supplies”。在类符号中显示Order Line联系的名称来强调“类本质”时，可以在关联路径上显示Supplies关联的命名来强调它的“关联本质”。

图14-18的类图揭示了用操作表示的处理，而这些处理在ER图中是无法表示的。例如，Customer有一个mailInvoice操作，在执行时，把发货单邮寄给已发出订单的顾客，并以美元为单位说明订单总额，并根据这个数量增加顾客的未付款的额度。一旦收到顾客的付款后，receivePayment根据收到的数额调整这个额度。类OrderLine中的orderlineTotal操作计算给定订

单的某一行总的美元数，而Order中的OrderTotal操作计算整个订单的总的金额（也就是所有订单行的金额的总和）。

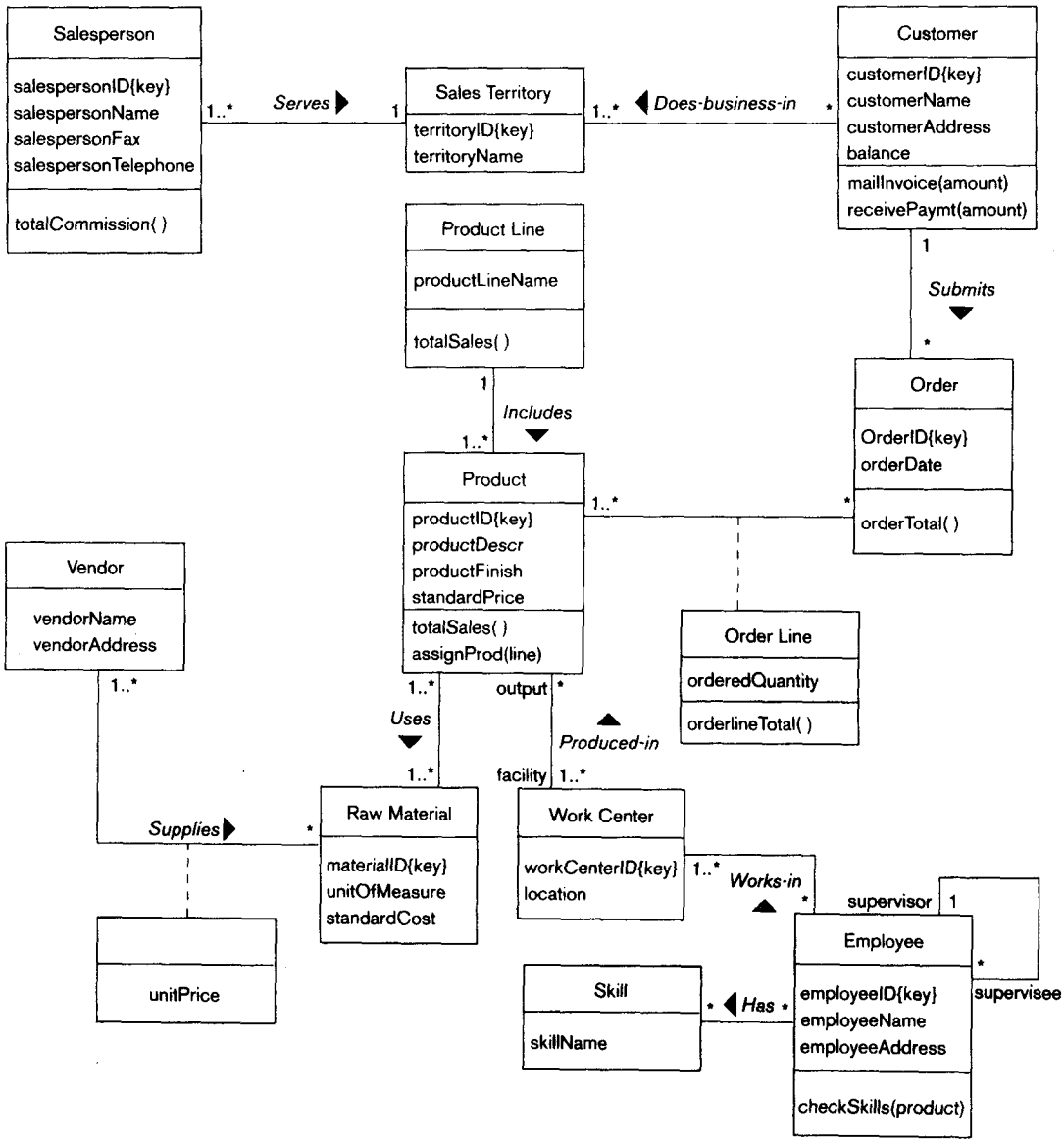


图14-18 松谷家具公司的类图

图14-18还说明了多态。totalSales操作在类Product和类Product Line中都出现过，但是是使用两种不同方法实现的。在Product中的方法计算给定产品的总的销售量，而ProductkLine中的方法计算属于给定产品系列的所有产品的总销售量。

totalSales、totalCommission、orderTotal、orderlineTotal和checkSkills等操作都是查询操作，这是因为这些操作都是不产生副面影响。而mailInvoice、receivePaymt和assignProd等都是更新操作，因为这些操作修改了某些对象的状态。例如，assignProd操作分配一个新的产品给用“line”变量指定的产品系列，因而改变产品（被分配的产品）和产品系列（包含这个产品的产

品系列）的状态。

本章小结

本章介绍了日趋流行的面向对象建模方法，这是一种使用公共的基本表示方法来支持现实世界应用的有效表示，其中包括数据和处理两个方面。本章描述了面向对象开发生命周期的不同阶段涉及到的活动，并强调了面向对象模型在从分析、设计到实现各个阶段转化的无缝的本质，这与其他建模方法形成了鲜明的对比。例如结构化分析和设计等方法缺少公共的基本表示，因此在转化过程中会出现不连续和不相交的情况。

面向对象建模是一个高级概念活动，特别适用于数据分析。本章中引入了对象和类的概念，并且讨论了对象标识和封装。本章还开发了几个使用UML表示法的类图来说明怎样建模各类情况。我们还学到了怎样绘制与给定类图相对应的对象图。对象图是类图的一个实例，提供了系统在某时间点上存在的实际的对象和链接的一个快照。

我们还看到了怎样使用操作来为基本应用进行建模的处理。操作有三种类型：构造器、查询和更新。ER模型（以及扩展的ER模型）不能应用这些处理，只能对一个组织的数据需求建模。本章强调了ER模型和面向对象模型之间的一些相似性，但同时也强调使后者比前者功能更强大的特性。

我们还看到了在类图中怎样表示不同程度的关联联系——一元联系、二元联系和三元联系。一个关联可以扮演两个或多个角色；每个角色有一个多重性，指出参与该联系的对象的数目。在关联角色中还可以指定其他约束，如形成有序的对象集。在关联本身具有自己的属性或操作时，或在关联参与其他关联时，关联可以被建模成一个类，这样的类称为关联类。对象图中的链接和链接对象分别与类图中的关联和关联类相对应。导出属性、导出联系和导出角色也能够在类图中表示。

与扩展ER模型中的超类型和子类型类似，面向对象模型使用超类和子类表示概化联系。概化路径的依据可以用紧靠概化路径的鉴别符标号来表示。子类中的语义约束可以用UML关键字（overlapping、disjoint、complete和incomplete）来指定。在一个类没有任何直接的实例时，它可以被建模成一个抽象类。一个抽象类可以提供一个抽象操作，但这个操作不是方法。

在概化联系中，一个子类继承它的超类的特性，并利用传递性继承了它的所有祖先类的特性。在面向对象系统中继承支持代码重用，因此继承是一个功能强大的机制。我们还讨论了使用继承特性的方法，以及在子类中使用重载继承操作的理由。我们还介绍了面向对象建模中另一个关键的概念——多态，这意味着一个操作可以以不同方式使用在不同的类上。面向对象建模中的封装、继承和多态概念可以使系统开发者利用功能强大的机制来开发复杂的、健壮的、灵活的和可维护的业务系统。

面向对象模型支持聚合，而ER或扩展ER模型不支持聚合。聚合是一种语义比关联还要强的形式，表示组件对象和聚合对象之间的part-of联系。我们对聚合和概化加以区分，并提供了在表示联系时选择关联还是聚合的准则。我们还讨论了比聚合还要强的一种形式，该形式称为复合。复合中每一部分对象属于惟一的整体对象，并与之同存亡。

本章还阐述了在类图中怎样隐式或显式地叙述业务规则。UML提供了几个关键字，它们可用于对类、属性、联系等的约束上。此外，也可以使用用户定义约束来表示业务规则。在一个业务规则涉及到两个或多个元素时，可以使用注解符号表示类图中的规则。在本章结束时，通过开发松谷家具公司的类图，说明了怎样利用面向对象方法来建模现实世界业务问题的数据和处理。

本章复习

关键术语

抽象类	抽象操作	聚合
关联	关联类	关联角色
行为	类图	类范围属性
复合	具体类	构造器操作
封装	方法	多重分类
多重性	对象	对象类（类）
对象图	操作	重载
多态	查询操作	范围操作
状态	更新操作	

复习问题

1. 定义下面每一个术语：

a. 对象类	b. 状态	c. 行为
d. 封装	e. 操作	f. 方法
g. 构造器操作	h. 查询操作	i. 更新操作
j. 抽象类	k. 具体类	l. 抽象操作
m. 多重性	n. 类范围属性	o. 关联类
p. 多态	q. 重载	r. 多重分类
s. 复合	t. 递归聚合	

2. 匹配下列术语及其定义：

_____具体类	a. 以不同方式使用的操作
_____抽象操作	b. 形式，不是实现
_____聚合	c. 直接实例
_____重载	d. 只属于一个完整的对象
_____多态	e. 方法替换
_____关联类	f. part-of联系
_____复合	g. 一组对象
_____对象类	h. 等价于关联实体

3. 比较下列术语

a. 类；对象	b. 属性；操作	c. 状态；行为
d. 操作；方法	e. 查询操作；更新操作	f. 抽象类；具体类
g. 类图；对象图	h. 关联；聚合	i. 概化；聚合
j. 聚合；复合	k. 扩展的重载；限制的重载	

4. 试叙述在面向对象开发生命周期的以下三个阶段涉及到的活动：
面向对象分析，面向对象设计，面向对象实现
5. 试比较面向对象模型和扩展的ER模型。
6. 试叙述将一个关联联系建模成一个关联类所依据的条件。
7. 试用类图来说明下面的联系类型：一元、二元和三元联系。说明这些联系的多重性。
8. 在复习问题7的例子中，试为关联联系添加角色名。

9. 在复习问题7的例子中，试为类添加操作。

10. 试给出一个概化的例子。这个例子应至少包含一个超类和三个子类，对于每个类至少给出一个属性和一个操作，指出鉴别符并说明子类之间的语义约束。

11. 如果为复习问题10开发的图中不包含抽象类，请扩展这张图，添加一个至少包含一个抽象操作的抽象类。还应指出，一个类的哪些特性被其他类继承。

12. 使用（或在必要时扩展）复习问题11的图，给出多态的一个例子。

13. 试给出一个聚合的例子。这个例子应至少包含一个聚合对象和三个组件对象，并说明在所有聚合联系的每个端的多重性。

问题和练习

1. 为下面列出的每种情况绘制类图，表示出有关的类、属性、操作和联系（如果你认为有必要做出额外的假设，那么说明对每种情况作出的假设）：

a. 一个公司有若干员工。员工（Employee）的属性有员工工号（employeeID，主键）、姓名（name）、地址（address）和出生日期（birthdate）。这个公司还有若干项目（Project）。项目的属性有项目名（projectName）和开始日期（startDate）。每个员工可以被指派到一个或多个项目中，也可以不指派到任何项目中。一个项目至少要指派一个员工，也可以指派任意数目的员工。一个员工的工资等级随着项目而变化，公司希望记录每个员工指派到特定项目时所适用的工资等级。在月末，公司给每个在这个月参加项目工作的员工发一个支票。支票的数额是根据员工参加项目的时间和工资等级给出的。

b. 一个大学在课程目录中设置了大量课程（Course）。课程的属性包括课程编号（courseNumber，主键）、课程名（courseName）和单元（units）。每门课程可以有一门或多门不同课程作为预备课程，也可以没有预备课程（prerequisites）。类似地，一门课程也可以是若干门课程的预备课程，也可以不是任何课程的预备课程。只有在课程的导师（director）做出申请时，大学才可以为这门课程加入或撤销一门预备课程。

c. 一个实验室有若干化学家，而每个化学家可在一个或多个项目上工作。化学家在进行每个项目可以使用若干设备。化学家（Chemist）的属性包括姓名（name）和电话号码（phoneNo）。项目（Project）的属性包括项目名（projectName）和开始日期（startDate）。设备（Equipment）的属性包括序号（serialNo）和价格（cost）。组织希望记录指派日期（assignDate）和总时数（totalHours），指派日期是指把一个给定的设备指派给从事某个特定项目的某个化学家的日期，总时数是指化学家使用该设备从事其项目的总时数。组织也要求追踪化学家使用每一种设备的情况。需要计算化学家在所有指派的项目中使用设备的平均时数。一个化学家必须被指派到至少一个项目，并至少为他分配一个设备。一个给定的设备不必分配给化学家，而且一个给定的项目也未必能分配给某个化学家或某个设备。

d. 一个学院的课程可以有一个或多个班级，也可以没有班级。课程（Course）的属性有课程标识（courseID）、课程名（courseName）和单元（units）。班（Section）的属性有班号（sectionNumber）和学期（semester）。班号的值是一个整数（例如“1”或“2”），用于区别同一门课程的不同班，但是不能惟一标识一个班。操作findNumSections用于查找在某学期某门课程所安排的班数。

e. 一个医院有许多注册的医生。医生（Physician）的属性有医生标识（physicianID，主键）和专长（specialty）。病人被医生收入医院治病。病人（Patiant）的属性有病人标识（patiantID，主键）和病人姓名（patiantName）。任何住院的病人必须被一位医生收入医

院。一个医生可以收入任何数目的病人。一旦病人住院，必须由至少一个医生治疗。一个医生可以治疗任何数目的病人，也可以不治疗任何病人。在病人被医生治疗期间，医院应记录治疗的详细情况（details），包括治疗的日期（date）、次数（time）和结果（results）。

2. 学生的属性有学生名（studentName）、地址（address）、电话（phone）和年龄（age），学生可以参加多种校园活动。学校要记录学生参加活动的年份，同时在每学年结束时发给学生一份参加各种活动的活动报告。试为上述情况绘制一个类图。

3. 试为一个房地产公司设计一个类图以列出销售的房产。下面是对这个组织的描述：

- 公司在许多州（states）有若干销售部（sales offices）；地点（location）是销售部的一个属性。
- 每个销售部有一名或多名员工。员工的属性包括员工标识（employeeID）、员工名（employeeName）。一个员工只能在一个销售部工作。
- 在每个销售部中，有一名员工被指定为经理。一个员工只可以管理他所在的销售部。
- 公司所销售的房产（property）列表。房产的属性有房产名（propertyName）和地点（location）。
- 每一处房产必须由其中一个（只有一个）销售部负责。一个销售部可以负责若干个房产，也可能不负责任何房产。
- 每一处房产可以有一个或多个拥有者（owners）。拥有者的属性包括拥有者名字（ownerName）和地址（address）。一个拥有者可以拥有一个或多个单位房产。对于拥有者拥有的每个房产，称为percentOwned的属性指出拥有者拥有的房产的百分比。

试在类图中确定的两个关联中加入一个子集约束。

4. 试为你熟悉的某个组织绘制类图，例如男童子军/女童子军、运动队等。在图中至少应为四个关联角色指出名字。

5. 试为下列情况绘制类图（同时说明为了开发一个完整的图而做出的假设）：

Stillwater古董店购买和销售各种的古董（例如家具、珠宝、瓷器和衣物）。每一项惟一地用项目号标识，并且用描述、开价、条件和自由评论来刻画。Stillwater古董店有许多不同的客户，这些客户在古董店销售或购买古董。某些客户只向古董店销售古董，某些客户只购买古董，还有些客户既销售也购买。每个客户用客户编号标识，用客户名和客户地址来描述。当古董店以现货形式销售给客户一件古董时，拥有者要求记录支付的佣金、实际销售价格、销售税（零税表示免税销售）和销售日期。当古董店购买客户的货物时，拥有者要求记录购买价值、购买日期和购买条件。

6. 试为下列情况绘制一个类图（同时应说明为了开发一个完整的图而做出的假设）：

H.I.Topi商业学校的业务遍及欧州的10个地点。学校在1965年已有第一批毕业生9000人。学校记载了每个毕业生的学生学号、姓名、出生国、现在的国籍、现在的姓名、现在的地址和学生的专业名称（每个学生有1至2个专业）。为了保持与毕业生的密切联系，学校设置了各种事件。每个事件有标题、日期、位置和类型（例如招待会、餐会或研讨会）。学校要记载哪些学生参加了哪些事件。对于事件中参加的学生，应记载校方人员从事件中学生那里听来的信息。学校还应该通过邮件、电子邮件、电话和电传等与学生保持联系。伴随这些事件，学校还记载了来自这些联系的学生传来的信息。在校方人员知道他或她将与学生见面或谈话时，要产生一个报告来显示关于学生的最近信息，以及最近两年来自所有联系的学生与学生参加的事件方面的信息。

7. 假定在松谷家具公司中，每个产品（用产品编号、说明和价值来描述）至少由三个部件

（用部件名、说明和计量单位描述）组合而成，并且这些部件也可用于制造一个或多个产品。另外，假设部件可以用来制造其他部件，这样，原材料也可视为部件。在这两种情况下，我们需要记载有多少部件用于制造某个产品。试为上述情况画一个类图，并指出图中所标识的联系的多重性。

8. 试为下述问题画一个类图。一个非盈利性组织依赖于若干不同类型人来成功完成操作。这个组织对这些人的下列属性感兴趣：社会安全号（ssn）、姓名（name）、地址（address）和电话（phone）。最感兴趣的是以下三种类型的人：员工（employees）、志愿者（volunteers）和捐赠者（donors）。每个人具有这些属性以外，员工还有一个称为聘用日期（dateHired）的属性，志愿者还有一个称为技能（skill）的属性，捐赠者是已经捐赠了一件或多件物品（items）给组织的人。物品用名称（name）说明，该物品可能没有捐赠者，也可能有一个或多个捐赠者。在捐赠物品时，组织将记载它的价值（price），以便在年末时能确定前十位捐赠者。

有一种人不属于员工、志愿者、捐赠者，但对该组织很关心，因而这种人不属于上述三种类型。另一方面，在某一时刻，一个人可能同时属于两种类型或多种类型（例如员工和捐赠者）。

9. 银行有三种账户：支票（checking）、存款（savings）、贷款（loan）。下面是每种账户的属性：

CHECKING：Acct_No、Date_Opened、Balance、Service_Charge

SAVINGS：Acct_No、Date_Opened、Balance、Interest_Rate

LOAN：Acct_No、Date_Opened、Balance、Interest_Rate、Payment

假设每个银行账户必须是这三种账户中的一种。在每个月末银行计算每个账户的余额，并发邮件给持有该账户的顾客。余额的计算取决于这个账户的类型。例如，支票账户余额可能影响服务收费，反之存款账户余额可能包含利息金额。试绘制一个类图表示上述情况。图中应包含一个抽象类，以及计算余额的一个抽象操作。

10. 再一次考察关于医院联系的类图（参见图14-9b）。试添加符号来表示下列业务规则：只有在病人被指派了他的主治医生时，这个病人才有可能分配到一个床位。

11. 现委托一个组织开发一个保存某个州所注册的所有车辆的注册和牌照信息的系统。对于注册的车辆，系统必须存储车主的姓名、地址、电话号码，以及注册的开始和结束日期、牌照信息（发牌者、年份、型号和编号）、操纵杆（年份、型号和编号）和注册费。此外，还应保存有关车辆本身的信息：编号、年份、制造者、型号、车身样式、重量、乘客数、柴油机动力（是/否）、颜色、价值和行驶里程。如果车辆是拖车，那么参数柴油机动力和乘客数就无关紧要了。对于旅游车，必须知道车身的编号和长度。系统必须保存客车的行李容量、卡车的最大载货容量和最大拖带能力、摩托车的马力等信息。在车主注册两个月后，系统将发出注册通知。在车主重新注册时，系统将更新有关车辆的注册信息。

a. 绘制类图来开发一个面向对象模型来表示所有的对象类、属性、操作、联系和多重性。对于每个操作，列出其参数。

b. 每一辆车有驱动装置，驱动装置包括发动机和变速器（忽视拖车未必有发动机和变速器这个事实）。假设对于每一辆车，系统必须保存下列信息：发动机汽缸的尺寸和数量，变速器的型号和重量。试在类图中加一些类、属性和联系来表示这些信息。

c. 试给出一个在子类或子类之间重载的操作的例子（也可以创建一个例子）。把这个操作加到类图中合适的地方，并讨论重载的理由。

应用练习

1. 访问一位朋友或家庭成员，找出一些有超类/子类联系的例子。还必须解释一些术语的

意义并提供一些常见的例子，例如:

PROPERTY: RESIDENTIAL, COMMERCIAL

BONUS: CORPORATE, .MUNICIPAL

利用这些信息来构造一个类图，并把类图反馈给朋友。不断修改这张类图，直到你和你的朋友或家庭成员觉得合适为止。

2. 参观两个本地的小企业，一个属于服务业，另一个属于制造业。访问这些组织中的员工，以获得超类/子类联系和操作性业务规则（例如“只有顾客具有有效的销售凭证时才能退货”）的例子。在什么环境下才能容易地发现这些构造的例子？为什么？

3. 咨询本地企业的数据库管理员或者数据库和系统分析人员，为企业的主数据库设计一个EER（或ER）图。把这张图转换成一张类图。

4. 访问本地公司中使用过面向对象程序设计语言或系统开发工具的系统分析员。就已经绘制出的有关数据库和应用的分析图和设计图，向他请教。试将这些图与本章的图进行比较。你认为有什么差别？使用了哪些附加的特性与符号？其意图是什么？

参考文献

Booch, G. 1994. *Object-Oriented Analysis and Design with Applications*, 2nd ed. Redwood City, CA: Benjamin/Cummings.

Coad, P., and E. Yourdon. 1991a. *Object-Oriented Analysis*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall.

Coad, P., and E. Yourdon. 1991b. *Object-Oriented Design*. Englewood Cliffs, NJ: Prentice-Hall.

Fowler, M. 2000. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 2nd ed. Reading, MA: Addison Wesley Longman.

Jacobson, I, M. Christerson, P. Jonsson, and G. Overgaard. 1992. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Reading, MA: Addison-Wesley.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. 1991. *Object-Oriented Modeling and Design*. Englewood Cliffs, NJ: Prentice-Hall.

UML Document Set. 1997. Version 1.0 (January). Santa Clara, CA: Rational Software Corp.

UML Notation Guide. 1997. Version 1.0 (January). Santa Clara, CA: Rational Software Corp.

Web资源

- www.omg.org OMG的网站，OMG是关心面向对象分析和设计的一个主要的行业协会。

项目案例：山景社区医院

项目描述

这个项目描述类似于第4章中的项目描述。在你完成这个项目后，你可以对第3、4章开发的数据模型进行比较。

山景社区医院是一个很大的服务性组织，由于许多人的工作才能保证其正常运转。医院主要依靠四种人：员工（employees）、医生（physicians）、病人（patiants）和志愿者（volunteers）。当然，这些人都有公共的属性：个人标识（personID）、姓名（name）、地址（address）、邮政编码（city-state-zip）、生日（dateOfBirth）和电话（phone）。

其中每一种人都至少有一个惟一的属性。员工有聘用日期（dateHired），志愿者有技能（skill），医生有特长（speciality）和呼机号（pagerNumber），病人有就诊日期（ofContact，指第一次上医院的日期）和calc-age。每一种人都还有惟一的方法。员工有calc-benefits，志愿者有assign-to-Cneter，医生有treats，病人有calc和assignToLocation。也是病人的员工或医生还有一个属性specialService。

在医院里还有其他一些人，这些人不属于上述四种人（这些人实际上很少）。另一方面，某一个人也可以在某一时刻属于其中的两种（或多种）人（例如病人和志愿者）。

每个病人有一个（且只有一个）主治医生。一个医生在某时刻可能不作为任何病人的主治医生，也可能是一个或多个病人的主治医生。病人分成两类：住院病人（resident）和出院病人（outpatient）。每个住院病人有属性dateOfAdmission和一个方法assignToBed，这个方法重载了所有病人的方法assignToLocation。每个出院病人可安排零次或多次复诊。实体visit有两个属性：date和comments。如果不存在一个出院病人的实体，那么回复的实例也不可能存在。

员工分成三类：护士（nurse）、职工（staff）和技术人员（technician）。只有护士才有表示其熟练程度（RN、LPN等）的属性certificate。只有职工才有属性jobClass，而只有技术人员才有skill属性。每个护士被分配到一个（且只有一个）治疗中心（care center）。例如，Maternity（产科）、Emergency（急诊）和Cardiology（即病房）等治疗中心。治疗中心的属性有name（标识符）和location。一个治疗中心可以有一个或多个护士。还有，对于每个治疗中心，其中一个护士被指定为主管护士（nurse_in_charge）；但是只有获得RN证书的护士才能指定为主管护士。治疗中心在某幢大楼中，因而大楼可用大楼编号（buildingNumber）、名称（buildingName）和代码（buildingCode）描述。

每一个技术人员被分配到一个或多个实验室。实验室的属性有name（标识符）和location。一个实验室至少有一个技术人员，也可以有任意多个技术人员。一个实验室只能位于一幢大楼内，但该大楼内可以有许多实验室。实验室里有许多设备和在该实验室工作的员工，因而实验室还有一个方法numberOfEmployees。

可以不给治疗中心分配病床，但可以给治疗中心分配一个或多个（直至任意数目）病床。病床只有一个属性bedID（标识符）和一个方法utilization。bedID是一个复合属性，由bedNumber和roomNumber组成。每个住院病人必须有一个病床。一个病床在某个时刻可能分配（或不分配）一个住院病人。

项目问题

1）在像山景社区医院这样的环境中建模超类/子类联系的功能是否很重要？为什么？

2）在这个项目数据需求的描述中，出现过链接对象吗？如果出现过，列举出这些链接对象。

3) 在这个医院的描述中，是否存在抽象对象类？为什么？

项目练习

1) 试画一个类图，遵循本章的表示法准确地表示这个需求集。

2) 试为项目练习1的类图中的下面每一种类型对象进行定义。你可以咨询医院或医疗机构中的人员；也可以基于你的知识和经验作出合理的假设。

a. 类　　b. 属性

c. 联系　　d. 方法

3) “只有获得RN证书的护士才能指定为主管护士”，这是一条业务规则。这条规则与第4章项目练习有所不同，现在你怎样建模这条规则？什么是约束对象？它是一个实体、一个属性、一个联系还是某个其他对象？

4) 比较本章开发的类图与第4章开发的EER图和第3章开发的ER图。这几张图之间区别是什么？为什么有这些区别？

5) 在本章的项目描述中，试找出聚合和复合的例子，并说明你的理由。

第15章　面向对象数据库开发

15.1　学习目标

学完本章[⊖]后，读者应该具备以下能力：

- 简要地定义下列每一个关键术语：原子文字、集合文字、集、包、列表、数组、词典、结构化文字和外延（extent）。
- 使用对象定义语言（ODL）创建面向逻辑对象的数据库模式。
- 通过映射类（抽象的和具体的）、属性、操作（抽象的和具体的）、关联联系（一对一、一对多、多对多）和概化联系，把概念UML类图转化为逻辑ODL模式。
- 为属性、操作变量和操作返回值确定类型规格说明。
- 创建对象并指定对象的属性值。
- 理解在实现面向对象数据库时设计的步骤。
- 理解对象查询语言（OQL）的句法和语义。
- 使用OQL命令编写各种类型的查询。
- 理解应用面向对象数据库的应用的类型。

15.2　引言

在第14章中已介绍了面向对象数据建模技术。我们已学过怎样使用UML类图来概念性地进行数据库建模。本章将阐述怎样把概念面向对象模型转换成能使用对象数据库管理系统（ODBMS）直接实现的逻辑模式。

正如后面将提到的，虽然关系数据库在传统业务应用方面非常有效，但在存储和操纵复杂数据和联系时有很严重的局限性（在程序设计需求和DBMS性能方面）。本章将说明在面向对象数据库环境中怎样实现应用。

本章采用对象数据库管理组织（ODMG）建议的对象模型（Object Model）来定义和查询面向对象数据库（OODB）。在开发逻辑模式时，会专门使用对象定义语义（ODL），这是在ODMG 3.0标准中为OODB规定的一种数据定义语言（Cattle等，2000）。ODMG是在1991年由OODB软件商为了创建标准而组成的一个组织，以便使OODB更加可行。就像SQL数据定义语言（DDL）模式能在兼容SQL的关系DBMS中实现（参见第7章）一样，用ODL创建的逻辑模式能在兼容ODMG的ODBMS中实现。

我们将使用类似于第14章中的例子来说明怎样把概念UML类图变换为逻辑ODL模式。我们将学到怎样把类（抽象的和具体的）、属性、操作（抽象的和具体的）、关联联系（一元和二元）和概化联系从UML类图映射到相应的ODL结构。

在第7章中，SQL中除了DDL成分外，还包括数据操纵语言（DML）成分，DML允许用户查询或操纵关系数据库中的数据。有一种与其类似的语言称为对象查询语言（OQL），ODMG 3.0标准指定用OQL来查询对象数据库（Cattell等，2000）。本章将阐述怎样使用OQL来编写各种类型的查询。ODMG组织已规定了C++、Smalltalk和Java版本的ODL和OQL。在本章的例子

⊖ 本章的最初版本由Wisconsin-Milwaukee大学的Atish P.Sinha教授编写。

中，我们使用比较常见的版本。

在第14章已给出松谷家具公司公司的一个概念面向对象模型。在本章中，将把这个概念模型转换成逻辑ODL模式。最后，将讨论适用ODBMS的应用类型并简单描述使用已有的ODBMS产品开发的某些应用。

15.3 对象定义语言

在第7章已学习过怎样使用SQL数据定义语言（DDL）来为关系数据库指定逻辑模式。类似地，可以使用对象定义语言（ODL）来为面向对象数据库指定逻辑模式。ODL是一个用于定义OODB模式的与程序设计语言无关的说明性语言。就像SQL DDL模式可移植到兼容SQL的关系DBMS上一样，ODL模式也可以移植到任何兼容ODMG的ODBMS上。

15.3.1 定义类

图15-1显示了大学数据库的一个概念UML类图（可参阅第14章对这个类图的解释），这是一个所有读者都熟悉的例子。现在，我们重点关注一下Student和Course类，以及它们的属性。在ODL中，用关键字class指定类，用关键字attribute指定属性。Student和Course类用如下形式定义：

```
class Student {
  attribute string name;
  attribute Date dateOfBirth;
  attribute string address;
  attribute string phone;
//加入联系和操作…
};
class Course {
  attribute string crse_code;
  attribute string crse_title;
  attribute short credit_hrs;
//加入联系和操作…
};
```

我们用黑体强调了ODL的关键字。此外，还添加了注释（前面用符号“//”标出）指出在稍后阶段还需把联系和操作（参见图15-1）加到类中。在attribute关键字的后面指定属性的类型，后面跟属性名。

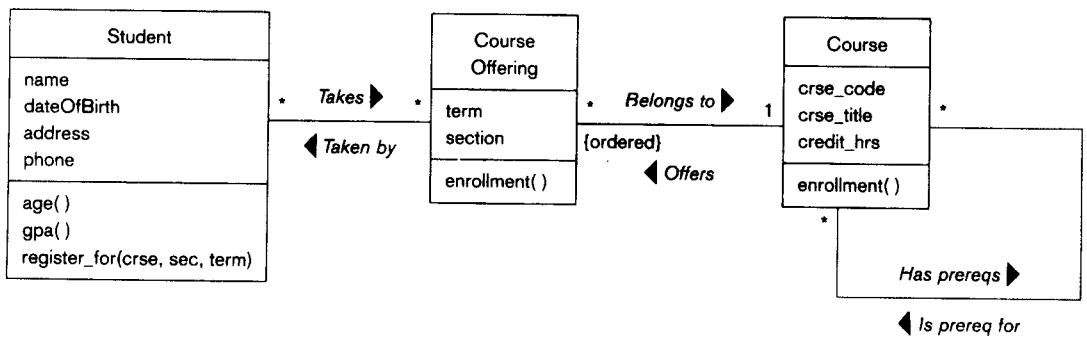


图15-1 大学数据库的UML类图

15.3.2 定义属性

属性的值可以是一个**文字**（literal），也可以是一个对象标识符。在第14章已讨论过，每个对象都有一个惟一的标识符。由于对象在它的生存期内保留着标识符，所以即使对象的状态改变了，但对象仍保持同样的标识符。相反，文字没有标识符，因此不可能像对象那样被单独引用。文字是嵌入在对象里的。可以将文字值看作常量。例如，字符串Mary Jones、字符C和整数20等都是文字值。

对象模型（Object Model）支持不同的文字类型，包括原子文字、集合文字和结构化文字。**原子文字**（atomic literal）类型的例子有string、char（字符）、boolean（真或假）、float（实数）、short（短整数）和long（长整数）。

集合文字（collection literal）是元素的集合，这些元素本身可以是任意的文字或对象类型。ODMG对象模型支持的集合文字类型有*set*（集）、*bag*（包）、*list*（列表）、*array*（数组）和*dictiondry*（词典）五种。**集**（set）是同样类型的不允许有重复元素的无序集合。**包**（bag）是同样类型的允许有重复元素的无序集合。与集合和包不一样，**列表**（list）是同类型元素的有序集合。**数组**（array）是能用位置定位的元素的有序集合，并且其大小可以动态变化。**词典**（dictionary）是由键-值对组成的不带重复元素的无序序列。

也称为结构的**结构化文字**（structured literal）由固定数目的已命名元素组成，每一个元素可以是文字或对象类型。对象模型支持下列预定义的结构：Date（日期）、Interval（时间间隔）、Time（时间）和Timestamp（时间戳）。对象模型还支持用户定义的结构，其例子将在后面给出。

现在再查看大学数据库的ODL模式。Student的属性name、address、phone以及课程的属性crse_code和crse_title都是字符串类型。字符串属性能存储一串用双引号界定的字母数字字符。Course的属性credit_hrs是短整数，这是因为它的值总是小于2^{16}的一个整数。除了这些原子文字类型以外，模式还为Student的属性dateOfBirth指定结构化文字类型Date。

15.3.3 定义用户结构

除了ODL提供的标准数据类型以外，用户还可以使用关键字*struct*定义结构。例如，可以定义称为Address（地址）的结构，这个结构有四个部分——street_adress（街道）、city（城市）、state（州）和zip（邮编），这四个部分都是字符串属性。

```
struct Address{
    string street_address;
    string city;
    string state;
    string zip;
};
```

类似地，也可以定义phone（电话号码）为由area_code（区号）和personal_number（个人号码）组成的结构。注意，personal_number被指定成long是因为它要超过2^{16}。

```
struct Phone{
    short area_code;
    long personal_number;
};
```

有些结构能够用来作为其他结构的元素。例如，如果学生有多个电话号码，那么phone属性可定义为下列形式：

```
attribute set <phone> phones;
```

15.3.4 定义操作

我们也能够为这两个类定义操作。在ODL中，可以在操作名称后使用括号来指定操作。Student的ODL定义可用下列形式表示:

```
class Student {
   attribute string name;
   attribute Date dateOfBirth;
//用户定义的结构化属性
  attribute Address address;
  attribute Phone phone;
//加入联系
//操作
  short age ();
  float gpa ();
  boolean register_for (string crse, short sec, string term);
};
```

我们已经定义了图15-1中所示的三个操作: age、gpa和register_for。前两个操作是查询操作。register_for操作是一个更新操作，注册一个学生在给定学期（term）选中某门课程（crse）的某个班（sec）。括号内的每一变量的前面都注明了它的类型[⊖]。还必须指定每个操作的返回类型。例如，age和gpa的返回类型分别是short（短整型）和float（实数）。register_for操作的返回类型是boolean（真或假），用来说明注册是否成功完成。如果一个操作不返回任何值，则返回类型可以声明成void。

每一个对象类型也可以有预定义的操作。例如，一个集对象有一个预定义的操作“is_subset_of”，一个数据对象（属性）有一个预定义的布尔操作“days_in_year”。Cattell等（2000）的著作详细介绍了预定义对象操作。

15.3.5 为属性定义范围

如果已经知道一个属性的所有可能取值，那么可以用ODL枚举这些值。例如，如果已知一门课程的最大班号为8，那么在CourseOffering类中的属性名前加关键字enum，然后在属性名后添上这些可能的值即实现了枚举。

```
class CourseOffering {
   attribute string term;
   attribute enum section{1, 2, 3, 4, 5, 6, 7, 8};
//操作
   short enrollment ();
};
```

15.3.6 定义联系

最后，把图15-1中显示的联系加到ODL模式中。ODMG对象模型只支持一元和二元联系。在图15-1中有两个二元联系和一个一元联系。在第14章中已讨论过，每个联系都是双向联系。例如，在Student和CourseOffering之间命名联系时，从前者到后者的联系命名为Takes，从后者到前者的联系可命名为Taken_by。可以使用ODL关键字relationship来指定这个联系。

```
class Student {
  attribute string name;
  attribute Date dateOfBirth;
```

⊖ 如果遵循严格的OCL语法，参数类型的前面必须有关键字in、out或inout，它们分别将参数说明为输入、输出或输入/输出。为了简单起见，我们在此省去这些说明。

```
  attribute Address address;
  attribute Phone phone;
//在Student和CourseOffering之间的联系
  relationship set <CourseOffering> takes inverse CourseOffering::taken_by;
//操作
  short age();
  float gpa();
  boolean register_for (string crse, short sec, string term);
};
```

在Student类中，用关键字relationship定义“takes”联系。联系名称的前面是联系的目标CourseOffering。由于一个学生可以选择多种课程设置，因此使用关键字“set”来说明Student对象与CourseOffering对象集（并且这个集是无序的）有联系。这个联系规范表示了从Student到CourseOffering的一条遍历路径。

ODMG对象模型要求将联系指定成双向的。在ODL中，用关键字inverse来指定相反方向上的联系。“takes”的逆是从CourseOffering到Student的“taken_by”。在Student的类定义中，已经命名了这条遍历路径（taken_by），在这条路径名称的前面是这条路径起源类的名字（CourseOffering）和双冒号（::）。在下面显示的CourseOffering的类定义中，指定为“taken_by”的联系上有来自Student的逆联系“takes”。由于一门课程可以由许多学生选定，因此联系把一个Student对象集链接到一个给定的CourseOffering对象。对于这种多对多联系，必须在两边都指定对象的集合（集、列表、包或数组）。

```
class CourseOffering {
  attribute string term;
  attribute enum section{1、2、3、4、5、6、7、8};
//在CourseOffering和Student之间的多对多联系
  relationship set <Student> taken_by inverse Student::takes;
//在CourseOffering和Course之间的一对多联系
  relationship Course belongs_to inverse Course::offers;
//操作
  short enrollment ();
};
```

ODBMS能自动强制执行在ODL模式中指定的联系的*参照完整性规则*（Bertino和Martino，1993；Cattell等，2000）。例如，如果你要删除一个Student对象，那么ODBMS将自动解除与所有CourseOffering对象的链接，也解除从所有CourseOffering对象返回到这个Student对象的链接。如果你把一个Student对象与一个CourseOffering对象集链接起来，那么ODBMS将自动创建逆链接。也就是说，创建从每一个CourseOffering对象返回到这个Student对象的链接。

在CourseOffering类中，还指定了另一个联系“belongs_to”，其逆是来自Course的“offers”。由于一个课程设置只能属于一门课程，因此belongs_to遍历路径的目的地是Course，这也意味着CourseOffering对象只能链接到一个Course对象。在关联联系的“一“边指定时，可以简单地指定目的地对象类型，而不是对象类型的集合（例如集）。

下面说明怎样指定Course的联系和操作。前面已经说过Course中的联系offers的逆是belongs_to。由于课程设置在课程中是有序的（根据某个学期的班号），因此不用set，而是用list来指定集合中的顺序。图15-1中显示的一元联系的两个方向上，has_prereqs和is_prereqs_for的起点和终点都在Course类上，Course的定义如下：

```
class Course {
  attribute string crse_code;
```

```
  attribute string crse_title;
  attribute short credit_hrs;
//预备课程的一元联系
  relationship set <Course> has_prereqs inverse Course::is_prereq_for;
  relationship set <Course> is_prereq_for inverse Course::has_prereqs;
//Course和CourseOffering之间的二元联系
  relationship list <CourseOffering> offers inverse CourseOffering::belongs_to;
//操作
  short enrollment ();
};
```

大学数据库完整的模式显示在图15-2中。注意，其中还引入了ODL关键字extent来指定类的外延。类的**外延**（extent）是数据库中类的所有实例的集合（Cattell等，2000）。例如，称为Student的外延就是数据库中所有的Student实例。

```
class Student {
(   extent students)
    attribute string name;
    attribute Date dateOfBirth;
    attribute Address address;
    attribute Phone phone;
    relationship set ⟨CourseOffering⟩ takes inverse CourseOffering::taken_by;
    short age( );
    float gpa( );
    boolean register_for(string crse, short sec, string term);
};

class CourseOffering {
(   extent courseofferings)
    attribute string term;
    attribute enum section {1, 2, 3, 4, 5, 6, 7, 8};
    relationship set ⟨Student⟩ taken_by inverse Student::takes;
    relationship Course belongs_to inverse Course::offers;
    short enrollment( );
};

class Course {
(   extent courses)
    attribute string crse_code;
    attribute string crse_title;
    attribute short credit_hrs;
    relationship set ⟨Course⟩ has_prereqs inverse Course::is_prereq_for;
    relationship set ⟨Course⟩ is_prereq_for inverse Course::has_prereqs;
    relationship list ⟨CourseOffering⟩ offers inverse CourseOffering::belongs_to;
    short enrollment( );
};
```

图15-2 大学数据库的ODL模式

15.3.7 定义以对象标识符作为值的属性

迄今看到的所有例子中，属性值都是文字。前面已提过，属性值也可以是一个对象标识符。例如，Course中有一个称为“dept”的属性，该属性表示某个系设置了这门课程。这个属性不存储系名，只存储Department对象的标识符。因此，必须作如下改变：

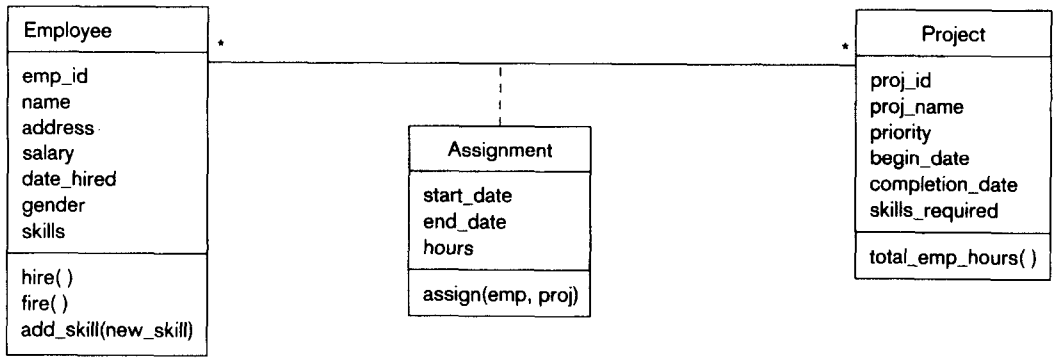
```
class Course {
```

```
//属性dept的值是一个对象标识符
  attribute Department dept;
//其他属性、操作和联系…
};
class Department {
  attribute short dept_number;
  attribute string dept_name;
  attribute string office_address;
};
```

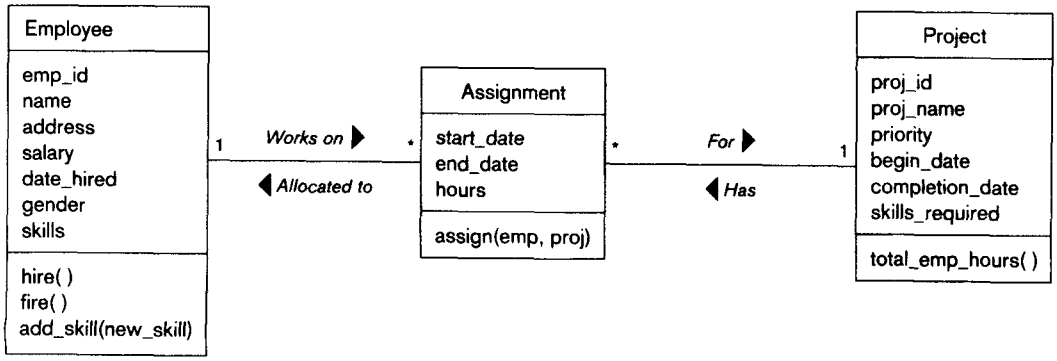
dept属性的类型是Department，表示属性值是Department的实例的对象标识符。这类似于从Course到Department的一个单向联系。但ODBMS并不会自动维护这种单向联系的参照完整性。如果用户查询中大多数引用是从Course到Department（例如，查找设置给定课程的系的名称），而不是从Department到Course，并且没有设置参照完整性，那么使用属性参照在一个方向上的这种联系的表示提供了一种更有效的候选方案。

15.3.8 定义多对多联系、键和多值属性

图15-3a显示了Employee和Project之间的多对多联系。为了完成一次分配，必须有一个Employee对象，以及一个Project对象。这里把这个Assignment建模成带有自己特性的关联类。这些特性有start_date、end_date和hours等属性，以及assign操作。在图15-3b中，把Employee到Project之间的多对多联系拆分成两个一对多联系，一个是从Employee到Assignment的联系，另一个是从Project到Assignment的联系。虽然用ODL可直接确定多对多联系，但不分解成两个一对多联系就不能抓住属于Assignment的特性。



a) 关联类的多对多联系



b) 拆分成两个一对多联系的多对多联系

图15-3　员工项目数据库的UML类图

现在可以把图15-3b中的类图转换成下列ODL模式:

```
class Employee {
( extent employees
//emp_id是Employee的主键
  key emp_id)
  attribute short emp_id;
  attribute string name;
  attribute Address address;
  attribute float salary;
  attribute Date date_hired;
  attribute enum gender {male,female};
//多值属性
  attribute set <string> skills;
  relationship set <Assignment> works_on inverse Assignment::allocated_to;
//下列操作不返回任何值
  void hire();
  void fire();
  void add_skill (string new_skill);
};

class Assignment {
( extent assignments)
  attribute Date start_date;
  attribute Date end_date;
  attribute short hours;
  relationship Employee allocated_to inverse Employee::works_on;
  relationship Project for inverse Project::has;
//下列操作给项目指派一个员工
  void assign (short emp,string proj);
};

class Project {
( extent projects
//proj_id是Project的主键
  key proj_id)
  attribute string proj_id;
  attribute string proj_name;
  attribute enum priority{low, medium, high};
  attribute Date begin_date;
  attribute Date completion_date;
//多值属性
  attribute set <string> skills_required;
  relationship set <Assignment> has inverse Assignment:: for;
  long total_emp_hours();
};
```

在ODL模式中，使用关键字keys为Employee和Project指定了候选键。注意，在对象数据库中，每个Employee或Project实例都是惟一的。也就是说，需要明确的标识符来强制实现对象的

惟一性。但是，指定键能确保属于一个类的两个对象没有同样的键属性值。惟一性的作用域被限制在类的外延中。emp_id属性对每个Employee对象必须有惟一的值，Project的proj_id也要有同样的性质。ODL还支持复合键，也就是键中的属性可有多个。例如，如果员工用name和address来惟一标识，那么可用下列语句指定键:

```
keys{name,address}
```

上面的模式还说明了怎样定义一个多值属性，即一个属性在给定时刻可以有多个值。Employee的skill属性和Project的skills_required属性都被指定成字符串值的集合。

15.3.9 定义概化

ODL支持一元和二元关联联系，但不支持更高元数的联系。关键字extend可以用来表示概化联系。图15-4显示的UML类图在第14章中也已出现过。三个子类被概化成一个称为Employee的超类，三个子类是Hourly Employee、Salaried Employee和Consultant。与这个类图相应的ODL模式如下:

```
class Employee {
(   extent employees)
    attribute short emp_number;
    attribute string name;
    attribute Address address;
    attribute float salary;
    attribute Date date_hired;
    void print_label();
};

class HourlyEmployee extends Employee {
(   extent hrly_emps)
    attribute float hourly_rate;
    float wages();
};

class SalariedEmployee extends Employee {
(   extent salaried_emps)
    attribute float annual_salary;
    attribute boolean stock_option;
    void contribute_pension();
};

class Consultant extends Employee {
(   extent consultants)
    attribute short contract_number;
    attribute float billing_rate;
    float fees();
};
```

子类HourlyEmployee、SalariedEmployee和Consultant通过引进新的特性扩充了更一般的Employee类。例如，HourlyEmployee除了有从Employee继承的公共特性以外，还有hourly_rate和wages两个特性。所有这些类（包括Employee）都是具体类，即它们都有直接的实例。由于子类是非完备的，因此Employee是一个具体类。

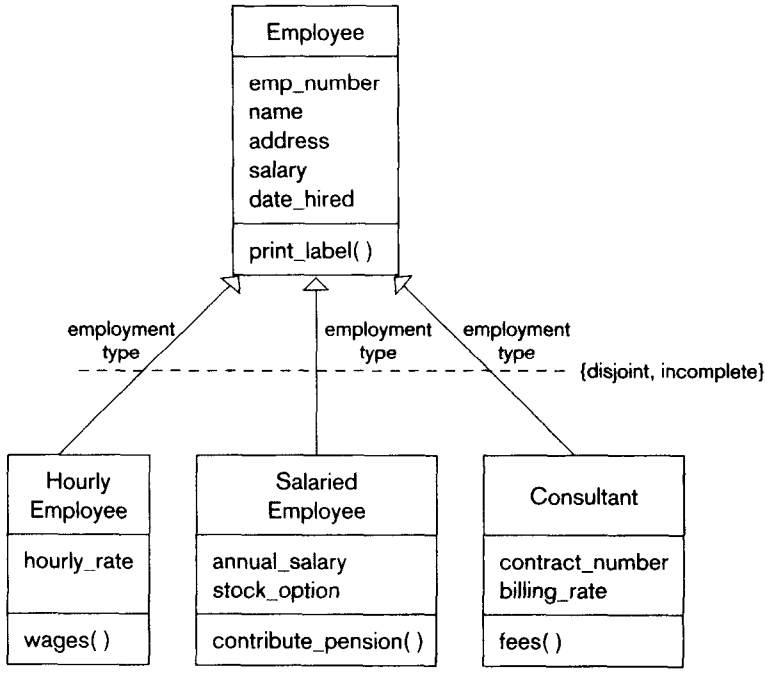


图15-4 表示员工概化的UML类图

15.3.10 定义抽象类

图15-5是称为Student的一个抽象类，它可以没有任何直接的实例。也就是说，一个学生必须是Graduate Student或Undergraduate Student的实例（应注意子类之间的“完备”约束）。在逻辑模式中，可以用如下方法指定Student为一个抽象类:

```
abstract class Student {
( extent students
  key stu_number)
  attribute long stu_number;
  attribute string name;
  attribute Date dateOfBirth;
  attribute Address address;
  attribute Phone phone;
  relationship set <CourseOffering> takes inverse CourseOffering::taken_by;
  boolean register_for (string crse, short section, string term);
//抽象操作
  abstract float calc_tuition();
};
```

Student的calc_suition操作是抽象操作，也就是说，此时指定的是操作形式而不是它的实现。我们用关键字abstract来指定抽象类和抽象操作[⊖]。子类用下面的方法定义:

```
class GraduateStudent extends Student {
( extent grads)
  attribute char undergrad_major;
  attribute GRE gre_score;
```

⊖ ODL当前并不支持类和操作的“抽象”关键字。本书使用的语法类似Java，Java语法明确地区分了抽象类/操作和具体类/操作。

```
  attribute GMAT gmat_score;
  float calc_tuition();
};

class UndergradStudent extends Student {
( extent undergrads)
  attribute SAT sat_score;
  attribute ACT act_score;
  float calc_tuition();
};
```

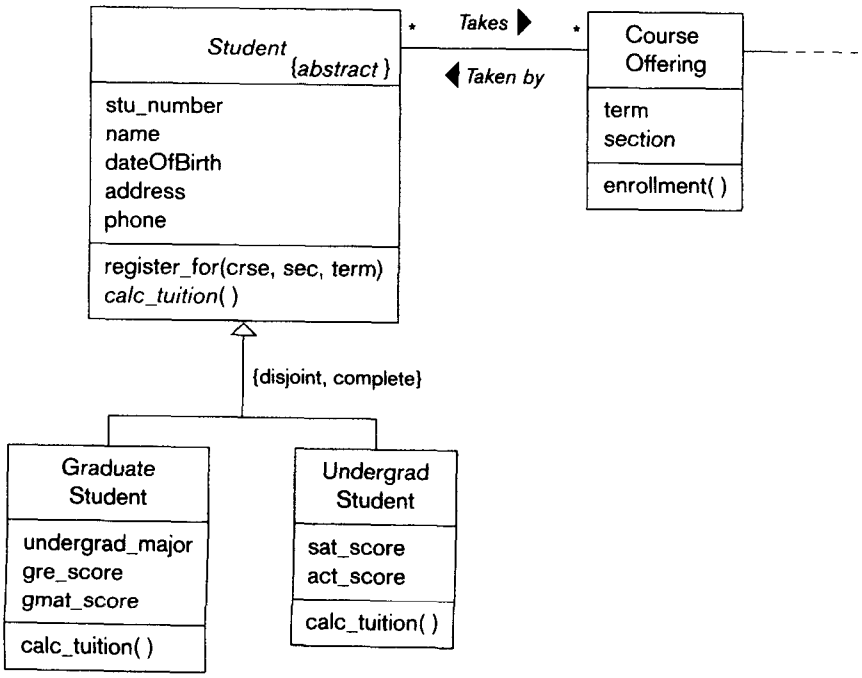


图15-5　表示学生概化的UML类图

由于这两个子类都是具体类，因此这两个子类中的calc_tuition操作都必须是具体操作。因此，虽然每个子类继承来自Employee的操作形式，但它们仍必须提供具体方法。calc_tuition操作分别在每个子类中单独指定，因此体现出多态性。因为没有使用关键字abstract，所以它们是具体的。

15.3.11　定义其他用户结构

这个模式定义还包含用户定义的结构，例如GRE、GMAT、SAT和ACT，它们都指定了各种考试得分的类型。在属性规范中可以使用Date和Time这种预定义结构，你也能定义用于相同目的的其他结构。用于考试得分的结构的定义如下所述：

```
struct GRE {
    Score verbal_score;
    Score quant_score;
    Score analytical_score;
};

struct GMAT {
    Score verbal_score;
```

```
    Score quant_score;
};
struct SCORE {
    short scaled_score;
    short percentile_score;
};
struct SAT {...};
struct ACT {...};
```

GRE结构由文字测试、资格测试和分析测试三部分组成，而GMAT结构没有分析测试。每个考试得分的类型都是称为SCORE的结构，它由定标得分（例如680）和百分比得分（例如95%）组成。

15.4 松谷家具公司的OODB设计

在第14章已为松谷家具公司以类图形式（参见图14-18）开发了一个概念面向对象模型。现在把这个概念模型转化为逻辑ODL模式，这个模式可以用来为公司实现面向对象数据库系统。

图15-6显示了这个ODL模式。现在可以清楚地理解类图中每个类、属性、操作和联系怎样映射为逻辑模式中等价的ODL结构。但是有一点要说明一下。由于Address和Phone的结构定义在前面已给出，因此在图中未显示出来。SalespersonFax（推销员）的Salesperson属性的类型是Phone，表示这个属性与推销员的Telephone的类型相同。还有，虽然类图将任何集合指定为有序的，但是也可以在ODL模式中对它们进行排序。例如，Order对象中的订单行的集合包含在一个指定为一个列表，表示订单行对象是有序的或被排序过的。

```
class Salesperson (
(   extent salespersons
    key salespersonID)
    attribute string salespersonID;
    attribute string salespersonName;
    attribute Phone salespersonFax;
    attribute Phone salespersonTelephone;
    relationship SalesTerritory serves inverse
        SalesTerritory::represented_by;
    float totalCommission( );
};

class SalesTerritory (
(   extent salesterritories
    key territoryID)
    attribute char territoryID;
    attribute char territoryName;
    relationship set<Salesperson> represented_by
        inverse Salesperson::serves;
    relationship set<Customer> consists_of inverse
        Customer::does_business_in;
};

class Customer (
(   extent customers
    key customerID)
    attribute string customerID;
    attribute string customerName;
    attribute Address customerAddress;
```

```
class OrderLine (
(   extent orderlines)
    attribute short orderedQuantity;
    relationship Order contained_in inverse
        Order::contains;
    relationship Product specifies inverse
        Product::specified_in;
    long orderlineTotal( );
};

class Product (
(   extent products
    key productID)
    attribute string productID;
    attribute string productDescr;
    attribute char productFinish;
    attribute float standardPrice;
    relationship ProductLine belongs_to inverse
        ProductLine::includes;
    relationship set<OrderLine> specified_in
        inverse OrderLine::specifies;
    relationship set<WorkCenter> produced_at
        inverse WorkCenter::produces;
    relationship set<RawMaterial> uses inverse
        RawMaterial::used_in;
    float totalSales( );
    boolean assignProd(string line);
};
```

图15-6 松谷家具公司数据库的ODL模式

```
    attribute float balance;
    relationship set<SalesTerritory>
        does_business_in inverse
        SalesTerritory::consists_of;
    relationship list<Order> submits inverse
        Order::submitted_by;
    void mailInvoice(float amount);
    void receivePaymt(float amount);
};

class Order {
(   extent orders
    key orderID)
    attribute string orderID;
    attribute Date orderDate;
    relationship Customer submitted_by inverse
        Customer::submits;
    relationship list<OrderLine> contains inverse
        OrderLine::contained_in;
    float orderTotal( );
};

class RawMaterial {
(   extent rawmaterials
    key materialID)
    attribute string materialID;
    attribute enum unitOfMeasure {piece, box,
        carton, lb, oz, gallon, litre};
    attribute float standardCost;
    relationship set<Product> used_in inverse Product::uses;
    relationship set<Supply> listed_in inverse Supply::lists;
};

class Supply {
(   extent supplies)
    attribute float unitPrice;
    relationship RawMaterial lists inverse
        RawMaterial::listed_in;
    relationship Vendor provided_by inverse
        Vendor::provides;
};

class Vendor {
(   extent vendors)
    attribute string vendorName;
    attribute Address vendorAddress;
    relationship set<Supply> provides inverse
        Supply::provided_by;
};
```

```
class ProductLine {
(   extent productlines)
    attribute string productLineName;
    relationship list<Product> includes inverse
        Product::belongs_to;
    float totalSales( );
};

class WorkCenter {
(   extent materialID
    key workCenterID)
    attribute char workCenterID;
    attribute string location;
    relationship set<Product> produces inverse
        Product::produced_by;
    relationship list<Employee> employs inverse
        Employee::works_in;
};

class Employee {
(   extent employees
    key employeeID)
    attribute string employeeID;
    attribute string employeeName;
    attribute Address employeeAddress;
    relationship set<WorkCenter> works_in inverse
        WorkCenter::employs;
    relationship set<skill> has inverse Skill::possessed_by;
    relationship Employee supervised_by inverse
        Employee::supervises;
    relationship set<Employee> supervises inverse
        Employee::supervised_by;
    boolean checkSkills(string product);
};

class Skill {
(   extent skills);
    attribute string skillName;
    relationship set<Employee> possessed_by inverse
        Employee::has;
};
```

图15-6　（续）

还应注意，两个带关联类的多对多联系怎样映射为两个一对多联系。第一个多对多联系是Order和Product之间的联系，带一个称为OrderLine关联类。在逻辑模式中，定义参与两个一对多联系的OrderLine类，一个与Order联系，另一个与Product联系。另一个多对多联系是带关联类Supplies的联系，它也可映射为两个一对多联系，一个是Vendor和Supply之间的联系，另一个是RawMaterial和Supply之间的联系。

15.5　创建对象实例

在创建一个类的新的实例时，也指定了惟一的对象标识符。可以用一个或多个惟一的对象

标识符。例如，可以用下列方法创建一个新的课程对象MBA 669:

```
MBA669 course();
```

这就创建了Course的一个新实例。对象标记名为MBA669，可以用来引用这个对象。此时还未指定这个对象的属性值。假设要创建一个新的学生对象，并且初始化某些属性。

```
Cheryl student (name:"Cheryl Davis", dateOfBirth: 4/5/77);
```

这就创建了一个新的学生对象，标记名为Cheryl，并且初始化了两个属性值。也可以为结构中的属性指定值，方法如下:

```
Jack student(
   name:"Jack Warner",dateOfBirth:2/12/74,
   address:{street_address"310 College Rd",city"Dayton",state"Ohio",zip 45468};
   phone:{area_code 937,personal_number 228-2252});
```

对于多值属性，可以指定值的集合。例如，可以为员工Dan Bellon指定其技能（skill），方法如下:

```
Dan employee(emp_id: 3678, name:"Dan Bellon",
             Skills:{"Database design", "OO Modelling"});
```

还可以很容易地为给定的联系中的对象之间建立链接。假定要存储Cheryl在1999年秋季选修三门课这个事实。可以编写以下语句:

```
Cheryl student(takes:{OOAD99F,Telecom99F, Java99F});
```

这里的OOAD99F、Telecom99F和Java99F是三个课程设置对象的标记名。这个定义创建了三个从标记为Cheryl的对象到每个课程设置对象的“takes”联系的链接。

考虑另一个例子。指派Dan参与TQM项目，可以编写以下语句:

```
assignment(start_date: 2/15/2001,allocated_to:Dan,for  TQM);
```

应注意，还没有说明指派对象的标记名。这些对象将由系统产生的对象标识符来标识。指派对象有一个到员工对象（Dan）的链接，还有一个到项目对象（TQM）的链接。

在对象创建时就被指定了生命周期，可以是暂时的或持久的。暂时对象只在某个程序或会话正在操作时存在。而持久对象一直存在到它被明确地删除为止。数据库对象总是持久的。

15.6 对象查询语言

现在介绍对象查询语言（OQL），OQL类似于SQL92，并且已被ODMG采纳为查询OODB的标准。OQL在编写查询时提供了许多灵活性。你可以写一个简单的查询:

```
Jack. dateOfBirth
```

它返回Jack的生日，结果是一个文字值。又如，

```
Jack. address
```

它返回一个带有街道地址、城市、州和邮码值的结构。如果只是要查找Jack居住的城市，那么可以用如下方法编写:

```
Jack. address. city
```

和SQL一样，OQL也使用Select-From-Where结构来编写更复杂的查询。例如，考虑图15-2给出的大学数据库的ODL模式。我们来观察怎样为这个数据库编写OQL查询。由于SQL和OQL之间极为相似，因此下面一节的解释是相当概括的。为了进一步解释，可以先回顾第7、8章关

于SQL和第14章关于对象建模的内容。感兴趣的读者可以参考Cattell等人（2000）关于OQL的介绍。

15.6.1 基本的检索命令

假设我们想要查找MBA664的名称（title）和每学时收款数（credit_hrs）。与SQL一样，这些属性在select子句中指定，具有这些属性的类外延（extent）在from子句中指定。在where子句中，必须指定要满足的条件。在下面的查询中，Course类的外延是courses，在from子句中被约束成称为c的变量。在select子句中说明了外延（即数据库中所有Course实例的集合）的属性crse_title和credit_hrs，并在where子句中列出了条件c.crse_code= "MBA664"。

```
select c.crse_title,c.credit_hrs
from courses c
where c.crse_code= "MBA 664"
```

由于只需处理一个外延，省略变量c也不会损失清晰度。但是，和SQL一样，如果涉及到具有公共属性多个类时，必须把外延绑定到变量以便系统能清楚地为所选择的属性标识类。这个查询的结果是一个带有两个属性的包。

15.6.2 在select子句中包含操作

我们也可以用指定属性的方法调用OQL查询中的操作。例如，要查找学生John Marsh的年龄，就在select子句中调用“age”操作。

```
select s.age
from students  s
where s.name= "John Marsh"
```

如果假定只有一个学生名为John Marsh，那么这个查询返回一个整数值。除了文字值以外，查询还可以返回带有标识的对象。例如，查询

```
select s
from students s
where s.gpa>=3.0
```

返回一个学生对象集合（包），这些学生的gpa值大于等于3.0。注意，我们在where子句中使用了gpa操作。

如果想要编写同样的查询，但限制学生不居住在Dayton，那么可以像在SQL中一样使用not操作符:

```
select s
from students s
where s.gpa>=3.0
and not (s.address.city= "Dayton")
```

如果不使用“not”，也可以用下列形式指定新的条件:

```
s. address. city != "Dayton"
```

这里“！”是不等操作符。

假定我们想查找gpa小于3.0的所有学生的年龄。这个查询编写如下:

```
select s.age
from students s
where s.gpa<3.0
```

15.6.3 查找不同的值

前面的查询将返回整数集合。有可能有多个学生年龄相同。如果想删去重复的值，那么可

以使用关键字distinct来重新编写这个查询:

```
select distinct s.age
from students s
where s.gpa<3.0
```

15.6.4 查询多个类

在OQL查询中，也可以像SQL那样在where子句联结多个类。在构成联结的基础的联系在对象数据模型中未定义时，这是必需的。当联系已经定义时，可以遍历在模式中定义的联系的路径。下面的查询查找2001年秋季开设的所有课程的课程代码。

```
select distinct y.crse_code
from courseofferings x,
    x.belongs_to y
where x.term= "Fall 2001"
```

由于一门课程在某个学期有多个设置，因此在select子句中使用了distinct。在from子句中，使用从CourseOffering对象到Course对象之间的“belongs_to”联系指定了一条路径。变量“y”被绑定在Course对象中，Course对象是用x.belongs_to表示的路径的终点。

假设我们想要寻找在MBA664课程第1班注册的学生的数目。注册操作在CourseOffering中是有效的，而课程代码在Course中也是有效的。下面的查询使用belongs_to联系遍历了从CourseOffering到Course这条路径。变量“y”表示x.belongs_to路径的终点对象。

```
select x.enrollment
from courseofferings x,
    x.belongs_to y
where y.crse_code= "MBA 664"
and x.section= 1
```

下面的查询遍历了两条路径，一条使用“takes”联系，而另一条使用“belongs_to”联系，用来寻找Mary Johns选修的所有课程的代码和名称。

```
select c.crse_code, c.crse_title
from students s
    s.takes x,
    x.belongs_to c
where s.name= "Mary Jones"
```

我们也可以选择一种由多个组件组成的结构。例如，下面的查询返回以age和gpa作为它的属性的一个结构。

```
select distinct struct(name: s.name, gpa:s.gpa)
from students s
where s.name= "Mary Jones"
```

15.6.5 编写子查询

在select语句中还可以使用一个select语句。为了选择注册人数小于20的课程代码、课程名称和课程设置，可以编写下面的OQL命令（参见图15-2的数据库设计）:

```
select distinct struct (code: c.crse_code,title: c_crse_title,
                        (select x
                        from c.offers x
                        where x.enrollment<20))
from courses c
```

enrollment是CourseOffering对象的一个操作，并且Course有一个与CourseOffering的1:M联系offers。这个查询返回distinct结构的一个集合，其中每一个结构包含课程代码、课程名称的字符串值以及注册人数小于20的CourseOffering对象的一个对象标识符。

还可以在from子句中使用select语句。在下面例子中，编写了检索年龄超过30、gpa值大于3.0的学生的姓名、地址和gpa值。

```
select x.name, x.addrss, x.gpa
from (select s from students s where s.gpa>=3.0)as x
where x.age>30
```

这里x是from子句中的select语句创建的外延的别名。

15.6.6 计算概要值

OQL支持SQL提供的所有聚合操作：计数（count）、求和（sum）、求平均值（avg）、求最大值（max）和求最小值（min）。例如，可以使用count操作来计算大学里学生的人数：

```
count (students)
```

也可以写成下面的查询形式：

```
select count (*)
from students s
```

现在考虑员工-项目数据库的模式。假设想要计算公司中女性员工的平均工资，可使用avg函数来完成这个查询：

```
select avg_salary_female:avg (e.salary)
from employees e
where e.gender=female
```

为了查找付给员工的最大工资，可用max函数：

```
max(select salary from employees)
```

为了计算所有员工的工资总和，可用sum函数：

```
sum(select salary from employees)
```

15.6.7 计算分组概要的值

像在SQL中一样，也可以把一个查询响应分成不同的组。在下面查询中。我们可以使用group命令按性别male和female分成两组。这个查询计算这两组中每一组人的最小工资。

```
select min(e.salary)
from employees e
group by e.gender
```

如果想按照项目的优先级对它们分组，可以编写下面的查询。

```
select *
from projects p
group by      low:      priority=low,
              medium:   priority=medium,
              high:     priority=high
```

这个查询返回三组项目对象，每个组都标有优先级：low、medium和high。

对分组的限定

像SQL一样，我们可以使用having命令把一个条件加在分组上，或在分组上进行过滤。例

如，在下面查询中，我们选择只在那些登录超过50小时的组。

```
select *
from projects p
group by      low:     priority=low,
              medium:  priority=medium,
              high:    priority=high
having sum (select x.hours from p.has x)>50
```

15.6.8 在查询中使用集合

有时需要判断一个元素是否属于某个集合。要完成这个工作，应该使用关键字in。假定我们想要查找具有数据库设计和面向对象建模技能的员工的标识（ID）和姓名。技能是多值属性，即它是一个值的集合。在where子句中，可以使用“in”来检查数据库设计或面向对象建模是否是员工技能集合的一个元素。

```
select emp_id,name
from employees
where "Database Design" in skills
or "OO Modeling" in skills
```

类似地，可以寻找在ID为TQM9的项目中工作的员工。

```
select e.emp_id,e.name
from employees e,
     e.works_on a,
     a.for p
where "TQM9" in p.proj_id
```

为了寻找不需要C++程序设计技术的项目，可以编写下列形式的查询:

```
select *
from projects p
where not ( "C++ Programming" in p.skills_required)
```

最后，也可以使用存在量词exists和全称量词for all。下面的查询用来寻找至少被指派到一个项目的员工。

```
select e.emp_id,e.name
from employees e
where exists e in(select x from assignment y,
                  y.allocated_to x)
```

where子句中的select语句返回所有被指派的员工对象（即他们的标识符）的集合。然后存在量词检查被限定在from子句中员工对象是否存在于这个集合中。如果在，那么在响应中就应包含这个员工的ID和姓名，否则就不包含。

如果想要寻找只在2001年开始的项目中工作的员工，那么可使用for all量词编写以下查询:

```
select e.emp_id,e.name
from employees e,
     e.works_on a
where for all a: a.start_date>= 1/1/2001
```

在from子句中，查询寻找通过“works_on”联系链接的员工对象被指派的对象的集合。在where子句中，把条件（start_data>=1/1/2001）应用到所有使用for all量词的集合中所有对象上。只有满足条件的对象，查询响应中才包含该员工的ID和姓名。

15.6.9 OQL的小结

本节介绍的内容只是OQL功能的一个子集。可参考Cattell等（2000）、Chaudhri和Zicart（2001）的著作来了解有关更标准的OQL的特性以及OQL在各种ODBMS中如何实现。

15.7 当今ODBMS产品和它们的应用

在从计算机辅助设计和制造（CAD/CAM）到地理信息系统以及多媒体的应用中，组织对存储和管理复杂数据（如图形、声音、图像）的需求日益增长，ODBMS产品得到了普及。但是，业内分析家认为因特网和基于Web的应用使大家突然重新开始对ODBMS感兴趣（King，1997；Watterson，1998）。虽然ODBMS未必能超过RDBMS，但ODBMS仍然是极有前景的应用产品。

ODBMS允许组织存储各种各样与它们的Web站点有关的组件（对象，参见Watterson，1998）。Web上复杂数据类型激增以及存储、索引、搜索和操纵这些数据的需求的迅速增长，给ODBMS技术提供了超过其他数据库技术的机缘。为了与这个新出现的技术抗衡，像Oracle、Informix、IBM和Sybase等主流的关系数据库厂商提出了通用数据库（universal database）概念，也称为对象关系DBMS（ORDBMS），这是一种可能的备选方案。ORDBMS是一种混合型的关系DBMS，设法将复杂数据作为对象来处理（King，1997；可参见附录D）。但是，这些系统引起了对性能和规模的关注。此外，关系技术和对象技术之间本质上的不匹配可能引起许多公司去采用纯ODBMS方案。

适合于ODBMS使用的应用的类型包括物材料单数据（参见图14-16）、支持导航访问的电信数据、医疗、工程设计、金融和贸易、多媒体和地理信息系统。现在可以买到若干商用ODBMS产品。例如，ObjectStore（当今市场主流产品）、Versant、GemStone、Objectivity和Jasmine（参见表15-1）。

表15-1　ODBMS产品

公　司	产　品	Web网站
Computer Associates	Jasmine	http://www.cai.com/products/jasmine.htm
Franz	AllegroSCL	http://www.franz.com
Gemstone Systems	GemStone	http://www.gemstone.com
neoLogic	neoAccess	http://neologic.com
Object Design	ObjectStore	http://www.odi.com
Objectivity	Objectivity/DB	http://www.objectivity.com
POET Software	POET Object Server	http://www.poet.com
Versant	Versant ODBMS	http://www.versant.com
与ODBMS产品有关的其他链接		
Barry & Associates		http://www.odbmsfacts.com
Doug Barry's *The Object Database Handbook*		http://wiley.com
Object database newsgroup		news://comp.databases. object
Rick Cattell's *The Object Database Standard ODMG* 3.0		http://www.mkp.com
Object Database Management Group		http://www.odmg.org
Chaudhri and and Zicari's *Succeeding with Object Databases*		http://www.wiley.com/compbooks/chaudhri

Watterson（1998）和Barry&Association（www.odbmsfacts.com/faq.htm）提供了ODBMS的现实世界应用的例子。Lucent技术公司的客户支持分公司使用GemStore共享遍及全球的顾客交换机上的信息。摩托罗拉公司使用Objectivity为其中一个卫星网络存储复杂的天文信息。法国

的退休计划管理公司Groupe Paradis使用O_2来访问分散在几个数据库中的超过100GB的数据。GTE、Southwest Airlines和Time-Warnet等公司使用ObjectStore开发动态Web应用，用于收集来自空中各个源的信息。芝加哥股票交易所使用Versant ODBMS用于其基于因特网的交易系统。

业内专家预言ODBMS是新兴数据库系统中最有前途的。虽然关系DBMS仍然可能在传统的业务应用方面保持它们占有的市场份额，但是前面提到的许多应用的数据还不能很容易地放置到二维数据库表中。而且，从各个表中访问数据需要执行联结操作，这是一个非常慢的操作。

本章小结

本章介绍了怎样使用对象定义语言实现一个面向对象数据库系统。我们介绍了ODL的句法和语义，介绍了怎样把用UML类图形式表示的概念模式转换成用ODL结构定义的逻辑模式。而且，学习了怎样通过创建新实例和为这些实例指定属性值来形成OODB。本章还介绍了OQL，这是一种为查询OODB而设计的语言。对于怎样使用OQL来写各种OODB查询也作了介绍。

本章还讨论了ODBMS适用的应用的类型。简要描述了当今使用的ODBMS适用的某些应用。

第14章提供了OODB设计的概念化框架，而本章提供了使用兼容ODMG的ODBMS实现面向对象数据库系统所需要的知识。

本章复习

关键术语

数组	原子文字	包
集合文字	词典	外延
列表	集	结构化文字

复习问题

1. 定义下面每一个术语:
   a. 对象类　　b. 联系
   c. 外延　　d. 原子文字
   e. 结构化文字
2. 比较下列术语:
   a. 列表；包；词典　　b. 集合；数组　　c. 集合文字；结构化文字
3. 试解释对象标识符的概念。对象标识符与关系系统的主键有什么区别?
4. ODL中关键字struct的用途是什么?
5. ODL中关键字enum的用途是什么?
6. 试解释图15-2的用于许多联系的术语relationship set（联系集）的意义。
7. 试分析通过暗指属性值是对象标识符而不是通过使用联系子句来表示ODL中的联系的危险性。
8. ODL中关键字extends的意义是什么?
9. 试解释SQL和OQL的相同和不同之处?

问题和练习

1. 试为下列问题开发相应的ODL模式。学生（student）的属性包括学生名（studentName）、地址（address）、电话（phone）和年龄（age），并可参与多项校园活动。大学（university）应记载一个学生参与某活动的年数，并且在每学年末发活动报告给学生，展示他参与各种活动的

情况。

2. 为一个地产公司（estate firm）列出其销售的房产而开发一个ODL模式。下面是对这个组织的描述：

- 公司在许多州（states）有若干销售部（sales offices）。地点（location）是销售部的一个属性。
- 每个销售部有一名或多名员工。员工（employee）的属性包括员工标识（employeeID）和员工名（employeeName）。一个员工只能在一个销售部工作。
- 在每个销售部中，有一名员工被指定为经理。一个员工只能管理他所在的销售部。
- 公司列出所销售的房产（property）。房产的属性有房产名（propertyName）和地址（address）。
- 每一处房产必须由其中一个（只有一个）销售部销售。一个销售部可以销售若干个房产，也可能没有任何房产。
- 每一处房产可以有一个或多个拥有者（owners）。拥有者的属性包括拥有者名（ownerName）和地址（address）。一个拥有者可以拥有一个或多个单位房产。对于拥有者拥有的每个房产，用percentOwned属性指出拥有者拥有的房产的百分比。

3. 试为你熟悉的某个组织开发一个ODL模式，例如男童子军/女童子军、运动队等。该模式中应至少包含四个关联联系。

4. 试为下列情况开发一个ODL模式（同时描述为了开发一个完整的图而需作出的假设）：

Stillwater古董店购买和销售各式各样的古董（例如家具、珠宝、瓷器和衣物）。每一项用项目号惟一地标识，并且用描述、开价、条件和自由评论来刻画。古董店有许多不同的客户。这些客户通过古董店销售或购买古董。某些客户只销售古董，某些客户只购买古董，而有些客户既销售也购买。每个客户用客户编号标识，用客户名和客户地址来描述。在古董店以现货形式将古董销售给客户时，拥有者应记录支付的佣金、实际销售价格、销售税（税金为零表示免税销售）和销售日期。在古董店购买客户的古董时，拥有者要求记录购买价值、购买日期和购买条件。

下面的第5~14题都要参考图15-2中的ODL模式。为这些问题编写OQL查询。

5. 查找在2001年秋季只选修一门课程的学生的姓名和电话号码。

6. 查找在2001年冬季和2001年秋季都设置的课程的代码和名称。

7. 查找在2002年冬季设置的课程MBA664的所有班的总的注册人数。

8. 查找MIS385课程的预备课程（代码和名称）。

9. 查找居住在辛辛那提并且在2001秋季选修MBA665课程的所有学生。

10. 查找MIS465课程的所有预备课程的总的付款时数。

11. 按学生生活的城市分组查找学生的平均年龄和平均gpa值。

12. 查找2002年冬季Sean Chen选修的课程的代码和名称。

13. 查找选修2002冬季MBA665课程的一班和二班的所有学生的姓名和电话号码。

14. 查找在2002年冬季设置的每小时收费为3美元的课程的注册人数最少的班级。回答应根据课程分组，并且分组标识应该是crse_code。

应用练习

1. 访问你所熟悉的公司中的一位数据库管理员。请他解释采用ODBMS为这个公司带来的好处。他们计划使用ODBMS吗？为什么？

2. 访问本教材中列出的Web网站。访问ODBMS供应商的几个站点。为你找到的一个产品

写一份概要。它的DDL与本章中的ODL有什么区别？它的查询语言与本章中的OQL有什么区别？供应商声称他们的产品与其他ODBMS产品或关系型产品相比有什么优点？

参考文献

Bertino, E., and L. Martino. 1993. *Object-Oriented Database Systems: Concepts and Architectures*. Wokingham, England: Addison-Wesley.

Cattell, R. G. G., et al. (Eds.) 2000. *The Object Database Standard: ODMG 3.0*. San Francisco: Morgan Kaufmann.

Chaudhri, A. B., and R. Zicari. 2001. *Succeeding with Object Databases*. New York: Wiley.

King, N. H. 1997, "Object DBMSs: Now or Never." *DBMS* (July): 42-99.

Watterson, K. 1998. "When It Comes to Choosing a Database, the Object Is Value." *Datamation* (December/January): 100-107.

Web资源

参见表15-1。

项目案例：山景社区医院

回顾第14章中项目案例的项目描述，参照该描述及项目练习的答案回答下列问题。

项目练习

1. 针对第14章项目练习1的答案开发一个ODL模式。
2. 写出使用这个模式的应用的三个典型的问题，并为这些查询编写OQL命令。

第六部分 附 录

附录A E-R建模工具和符号

第3章给出了表示数据模型的符号，第4章扩展了这些数据模型，并在全书范围内使用。所使用的各种描述模型的软件工具不同，应用这些符号的方法也不同。就像业务规则和策略不是通用的一样，各种数据建模工具所用的符号也不同。使用不同的图形结构和方法有可能能够（也有可能不能够）表示特定的业务规则。

本附录旨在帮助读者对建模工具中的符号和书中的符号进行比较。四种常用的工具包括：Visible Analyst 7.4、Platinum的ERwin3.5.2、Microsoft Access 2000以及Oracle Designer 6.0。表A-1a和A-1b显示了这四种工具表示实体、联系、属性、规则、约束等等的符号。另外，图A-3到图A-6包含了每一个工具的屏幕显示。

图3-22所示的松谷家具公司（PVFC）的E-R图是本附录例子的基础。该图是使用本书介绍的符号系统，根据第3章包含的该公司的业务规则画出的数据模型。表A-1给出了四种软件工具使用这些表示方法的比较。

A.1 比较E-R建模规则

如表A-1所示，创建表示逻辑数据模型的E-R图时所用的符号有很大不同。这里并不想对各种工具进行深入比较，下面的论述使用图3-22中描述的PVFC ERD来分析工具之间的区别。特别注意在描述下面概念时的区别：多对多联系、基数或可选性、外键、超类型/子类型联系。

A.1.1 Visible Analyst 符号

该工具很灵活，给用户提供用图形描述数据模型的几个选择。有5种可供选择的符号，包括IDEFIX、Bachman、Crow's Foot、Arrow和UML。表A-1使用Crow's Foot，并遵守Gane&Sarson规则。选择任何不同选项会有不同的表示功能。

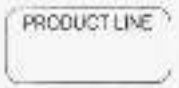
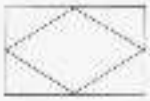
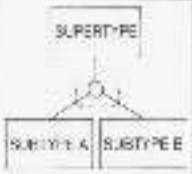
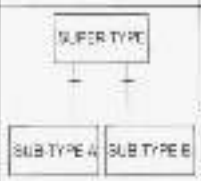
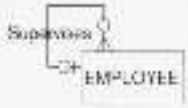
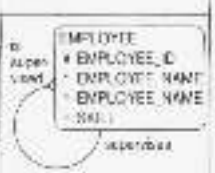
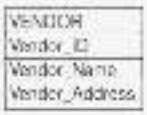
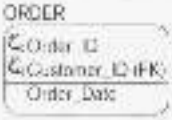
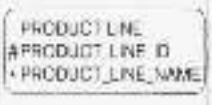
实体 有三种实体类型符号：基本、关联和属性。图的细节可以灵活地显示，例如，仅显示实体名，或实体名和主键，也可以包括所有属性。当在关联实体上放置一个属性时，例如，表A-1中的Order_Line，它的符号会自动改变，实体名的旁边会出现指示符〔AS〕，如图A-3所示。

联系 可以将线标记为单向或双向，并且可以有多个线段，可以选择使起点和终点相互远离。根据所选择的符号，联系线能够显示基数，存储一个实体与另一实体相关的特定数目的实例。联系线可以表示零对一、一对一、零对多、一对多或多对多。使用对话框能够改变已有联系的基数。递归联系表明父亲和孩子是同一个实体。表A-1a中的例子表明，Employee可以管理很多员工，但并不是所有的员工都是管理人员；该符号表明允许使用空值。父亲实体可能拥有多个孩子，但一个孩子只能有一个父亲。不能表示这样的规则：一个员工只能有一个管理者，因为可选符号说明员工可能没有管理者。

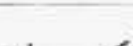
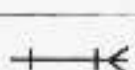
A.1.2 Platinum ERwin符号

这里可以选择EDEFIX、IE（Information Engineering）或DM（Dimensional Modeling）符号。这里使用IE为例。

表A-1a Hoffer、Prescott和McFadden利用4种软件工具的建模符号（常用建模工具符号）

	Hoffer Prescott McFadden Notation	Visible Analyst 7.4	Platinum ERwin 3.5.2	Microsoft Access 2000	Oracle Designer 6.0
基本实体					
关联实体				（没有特殊符号，使用常规实体的符号）	（没有特殊符号，使用常规实体的符号）
子类型				（没有特定符号）	
递归实体					
其他					

表A-1b Hoffer、Prescott和McFadden利用4种软件工具的建模符号（常用建模工具的基数/可选符号）

	Hoffer-Prescott McFadden Notation	Visible Analyst 7.4	Platinum ERwin 3.5.2	Microsoft Access 2000	Oracle Designer 6.0
1:1		（若没有基数则不可用）	（若没有基数则不可用）		
1:M		（若没有基数则不可用）	（若没有基数则不可用）		
M:N		（若没有基数则不可用）		（不完整）	
标识 1:1				（没有可选符号）	
标识 1:M				（没有可选符号）	
可选 1:M				（没有可选符号）	

实体 如果在标识的联系中，实体是一个孩子（弱实体），则该实体为依赖实体，用圆角矩形框表示。关联实体也用该方式表示。用户选择使键迁移到主键区域或非键区域。

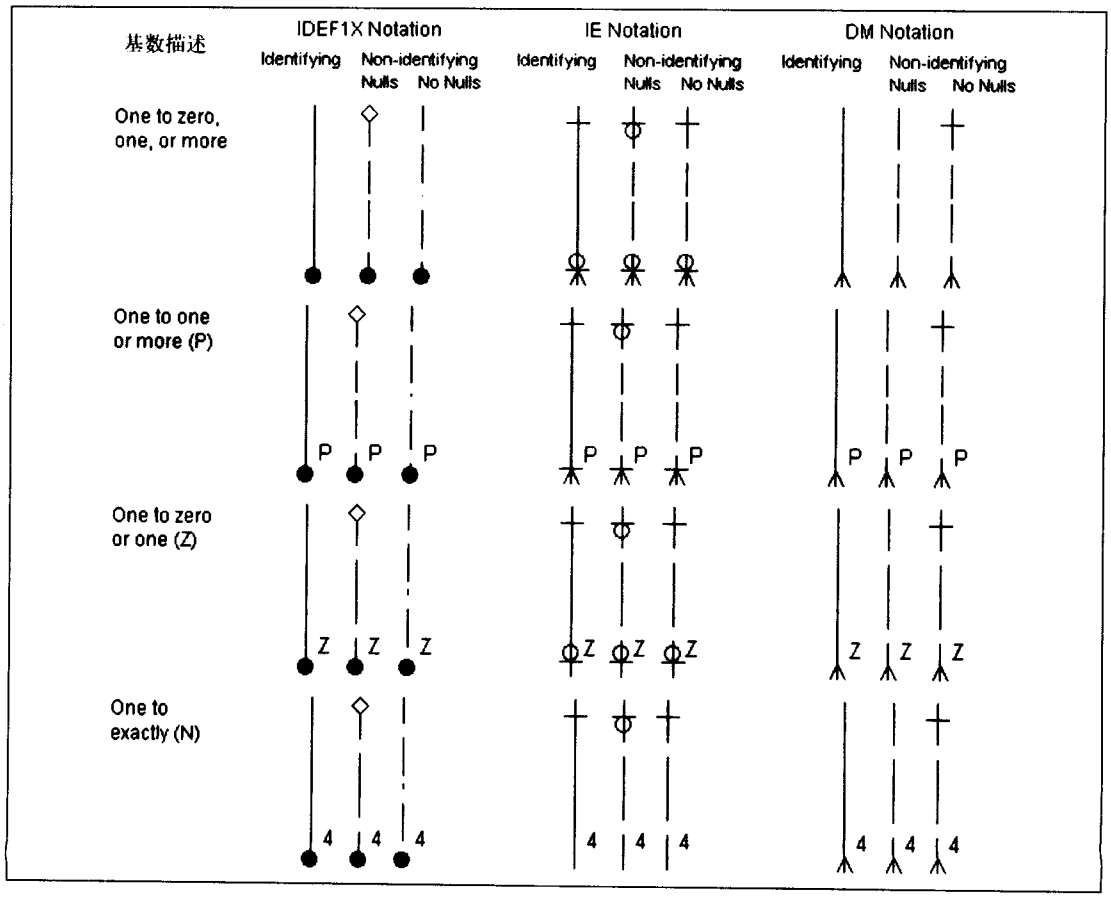
联系 基数选择灵活且含义明确。一个父亲可以连接到“Zero, One, or More”，用空格表示；“one, or More,”用P表示；“Zero, one”由Z表示；“Exactly”表示出现在ERD中的实例数目。多对多联系自动解决，在逻辑模型上显示一个关联实体，并且多对多联系被删除。父亲和孩子的递归联系表示为同一个实体，一个员工（一个Supervisor）可以管理很多员工，但并不是所有的员工都是管理者。该符号表示可以使用空值。一个父亲可以有任意数目的孩子，但是一个孩子只能有一个父亲。当建立联系时，键自动迁移，并且外键表示为“(FK)”。

该图是从ERwin的在线帮助抓取的，并在图A-1中给出。该图描述了该产品使用的表示基数范围的符号。

A.1.3 Microsoft Access 2000符号

Access不像画图工具那样可以在设计数据库之前建立逻辑模型。为了在数据模型中描述具有属性的表，以及建立实体之间的联系，必须在数据库中定义这些带有属性的表。但是，当用户了解这样的限制后，可以在关系屏幕上定义表，MS Access 2000用户应该能够利用这样的功能建立联系。

实体 因为Access关注的是数据库设计而不是数据建模，所以不存在不同实体类型的特定表示方法，这些实体类型包括关联实体、超类型/子类型。



图A-1 ERwin基数/可选性符号

联系 联系描述为线，通过连接已有表的属性画出联系。Access在安排实体使其显示更清楚这方面的功能有限。可以编辑联系线以改变基数，但是没有提供表示方法以区别可选性。当联系建立起来时，和ERwin不同，没有键的自动迁移。必须定义属性以使用于链接。一对一或一对多的递归联系通过以下方式描述：在工作空间的两个表中放置表的两个拷贝，并建立该表的两个拷贝之间的链接。在表A-1a中显示了Access自动标为EMPLOYEE_t_1和EMPLOYEE_t的EMPLOYEE的两个拷贝。递归联系显示为一个链接，它链接EMPLOYEE_t_1表的主键和EMPLOYEE_t中的"Employee-supervisor"属性。递归联系的1对多特性显示在EMPLOYEE_t_1和EMPLOYEE_t之间的链接之上。关联实体（例如Order Line）使用与其他实体相同的符号表示，如图3-22所示。Order_Line以大小写混合的方式命名，以区别于别的实体（以大写字母来命名）。PRODUCT和ORDER之间的多对多的联系通过创建具有复合键Order_ID和Product_ID的关联实体Order_Line来解决。

A.1.4 Oracle Designer 6.0符号

使用ERD工具来画图时，可以作如下设置：只显示实体名，显示实体名和主键，或者显示实体名和所有的属性。

实体 不存在不同实体类型的特定符号，如关联实体、超类型/子类型。所有实体都表示为圆角矩形框，属性显示在方框内。惟一标识符通过符号"#"表示，并且是强制的。强制属性标记为"~"，可选属性标记为" "。

联系 线必须标为双向，而不只是单向。不要有多个线段，这样难以操纵和对齐。联系线描述了基数和可选性。通过首先选择所需的基数和可选性将其画出。在对话框中可以对它们进行修改。可选性使用表示强制联系的实线和表示可选联系的虚线来描述。在一个二元联系中，根据联系的方向，可选性会有所不同。两个实体之间的连线对视为实线和虚线。可以从一个实体开始顺着其上的连线理解联系。因此，在图A-6中，一个产品必须在Work Center，但是，Work Center不必为了存在而制作一个产品。基数通过连在其他实体上的基数符号来理解。因此，一个顾客可以下订单也可以不下订单，但如果已经下了一个订单，就必须与特定的顾客相关联。考虑EMPLOYEE实体，递归的管理联系由实体上的"猪耳"形符号描述。它表明一个Employee可能管理一个或多个员工，并且那个员工必须被一个员工管理。至于多重基数是0、1还是多，其含义是模糊的。图A-2给出了在Designer中建立联系所使用的工具条按钮。

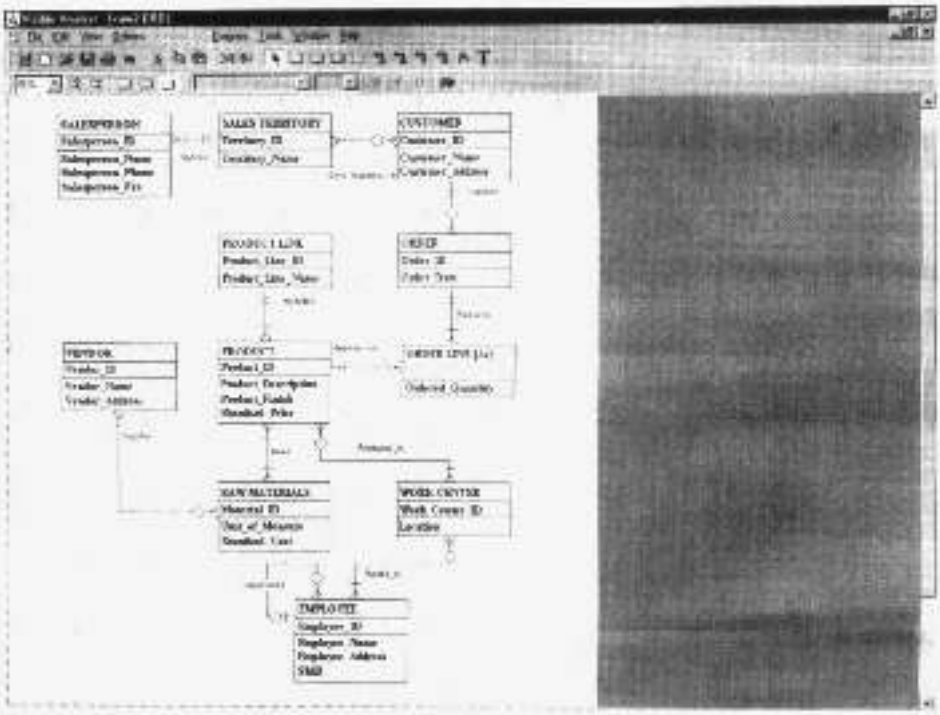


图A-2 Oracle Designer 6.0联系工具条

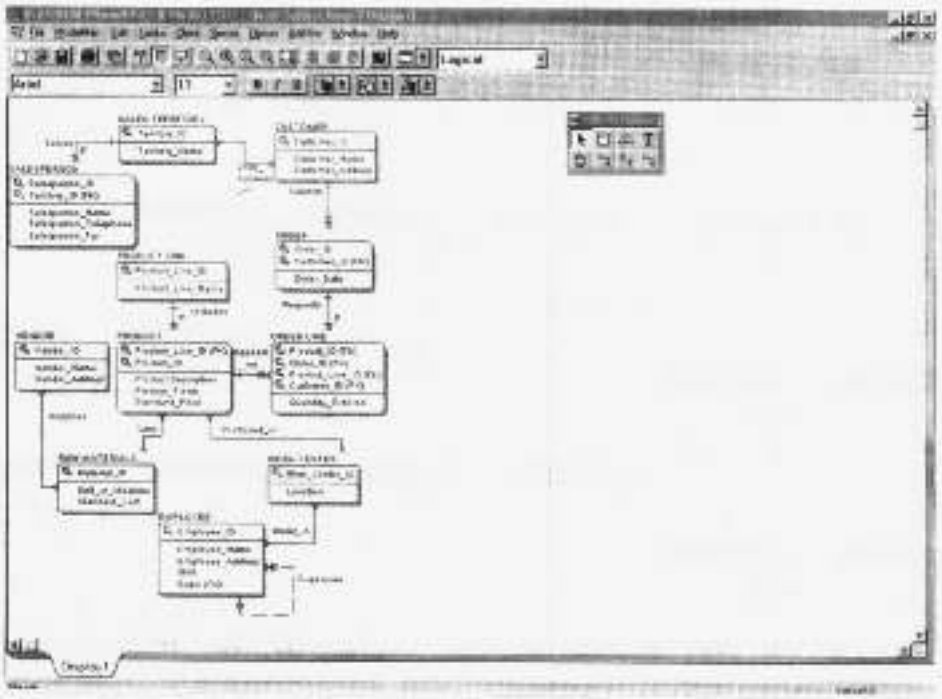
当使用Oracle Designer时，在使用工具前仔细和完整地勾画出数据模型非常重要。编辑模型很具挑战性，从图中删除一个对象并不会将其从信息库中自动删除。

A.2 工具界面和E-R图的比较

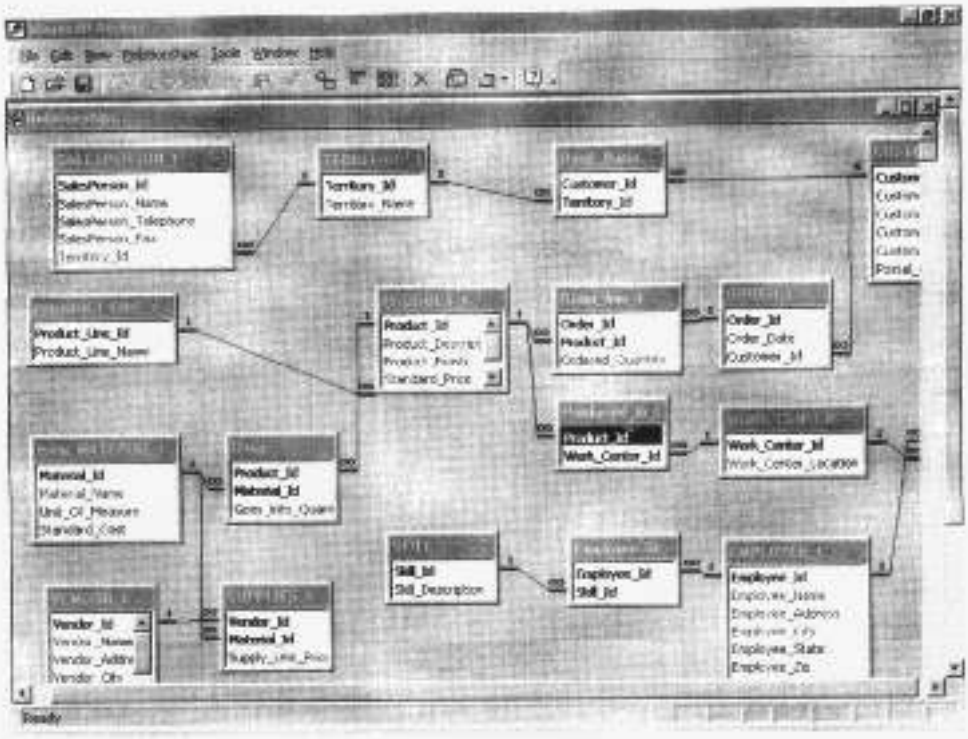
利用表A-1a中提到的每一种建模工具，给出了图3-22中一部分数据模型的屏幕抓取。通过这些图，读者能够很好地理解这些符号如何在实际中使用。图A-3用Visible Analyst 7.4画出，带有Crow's Foot和Gane&Sarson选项。图A-4用Platinum的ERwin 3.5.2画出，并选择IE选项。参照完整性可以可选地显示，但不包括在该图中。图A-5a是Microsoft Access 2000联系屏幕图，图A-5b是使用File菜单下的"Print Relationships..."而生成的。图A-6用Oracle Designer 6.0画出，并选择IE选项。



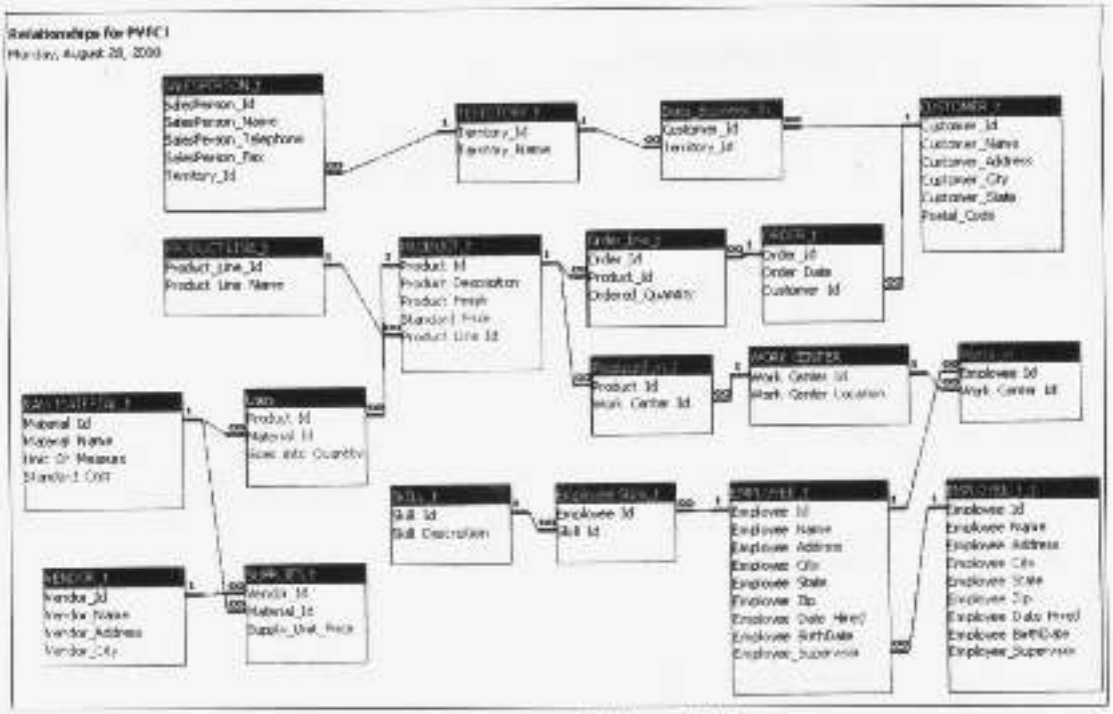
图A-3 Visible Analyst 7.4界面



图A-4 Platinum ERwin 3.5.2界面

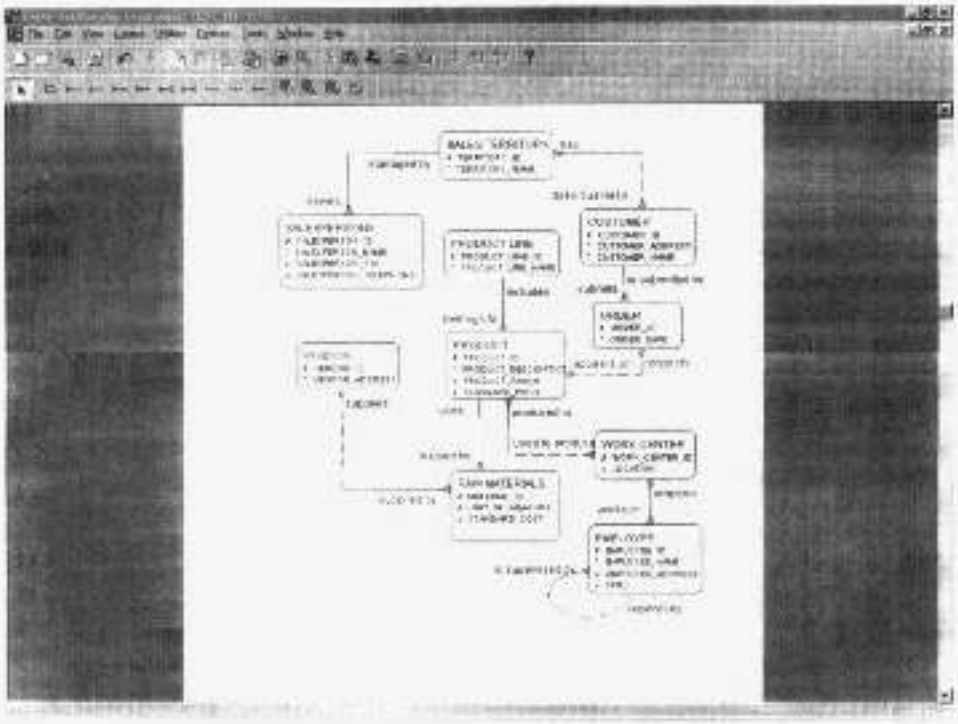


a) Microsoft Access 2000界面——联系屏幕



b) Microsoft Access 2000界面——联系报表

图A-5 Microsoft Access 2000界面



图A-6 Oracle Designer 6.0界面

附录B 高级范式

在第5章中介绍了范式并首先详细论述第一到第三范式。对大多数数据库应用来讲，第三范式（3NF）关系已经足够。但是，3NF并不能保证消除所有异常。第5章中说过，可以使用其他几个范式去除这些异常：Boyce-Codd范式、第四范式和第五范式（参见图5-22）。在本附录中，主要讨论Boyce-Codd范式和第四范式。

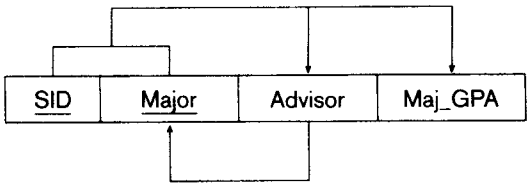
B.1 Boyce-Codd范式

当一个关系有多于一个的候选键时，处于第三范式的关系也会导致异常。例如，考虑图B-1中的关系STUDENT_ADVISOR。该关系具有下列属性：SID（学生ID）、Major、Advisor和Maj_GPA。图B-1a给出了该关系的样例数据，图B-1b说明了函数依赖。

STUDENT_ADVISOR

SID	Major	Advisor	Maj_GPA
123	Physics	Hawking	4.0
123	Music	Mahler	3.3
456	Literature	Michener	3.2
789	Music	Bach	3.7
678	Physics	Hawking	3.5

a) 具有样例数据的关系



b) STUDENT_ADVISOR中的函数依赖

图B-1 第三范式关系，但不是BCNF

如同图B-1b所示，该关系的主键是由SID和Major组成的复合键。因此，属性Advisor和Maj_GPA函数依赖于该键。这反映了如下约束：尽管一个学生可能有多个专业，对于每一个专业，一个学生只有一个指导老师和一个GPA。

该关系中的第二个函数依赖为：Major函数依赖于Advisor。即，每一个指导老师只指导一个专业。注意，它不是传递依赖。在第5章中，本书将传递依赖定义为两个非键属性之间的函数依赖。相比之下，在该例子中，一个键属性（Major）函数依赖于一个非键属性（Advisor）。

B.1.1 STUDENT_ADVISOR中的异常

很明显，STUDENT_ADVISOR关系符合第三范式，因为不存在部分函数依赖和传递依赖。然而，由于Major和Advisor之间的函数依赖，该关系中仍然存在异常。考虑下面的例子：

1) 假定将物理学指导教师由Hawking替换为Einstein，则必须在表的两行（或多行）进行修改（更新异常）。

2) 假定要在表中插入一行这样的信息：Babbage指导Computer Science这门课程。当然，直到至少一个专业为Computer Science的学生指定由Babbage指导后，才能插入这行信息（插入异常）。

3) 最后，如果学生号为789的学生退出学校，则丢失Bach指导Music这条信息（删除异常）。

B.1.2 Boyce-Codd范式的定义

STUDENT_ADVISOR中的异常源自这样的事实：有一个决定因子（Advisor）不是关系中候选键。R. F. Boyce和E. F. Codd标识出这样的缺陷，并给出一个比3NF更强的定义来矫正这样的问题。当且仅当关系中的每一个决定因子都是候选键时，我们说关系符合**Boyce-Codd范式**（Boyce-Codd Normal Form，BCNF）。STUDENT_ADVISOR不符合BCNF，因为尽管属性Advisor是一个决定因子，但它不是一个候选键（只有Major函数依赖于Advisor）。

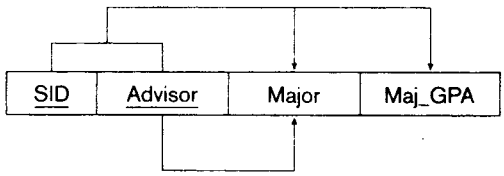
B.1.3 将一个关系转换为BCNF

使用简单的两步过程，可以把符合3NF的关系（但不是BCNF）转换为符合BCNF的关系。图B-2说明了这样的过程。

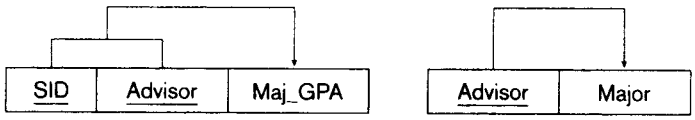
第一步，修改关系使关系中不是候选键的决定因子转化为修订后关系的主键的一部分。函数依赖于那个决定因子的属性变为一个非键属性。由于函数依赖，所以对原始表的重构是合理的。

对STUDENT_ADVISOR应用该规则的结果如图B-2a所示。决定因子Advisor变为复合主键的一部分。函数依赖于Advisor的属性Major变为一个非键属性。

如果读者查看图B-2a，会发现新的关系有一个部分函数依赖（Major函数依赖于Advisor，Advisor是主键的一部分）。从而，新的关系符合第一范式（但不符合第二范式）。



a) 修订的STUDENT_ADVISOR关系（2NF）



b) 符合BCNF的两个关系

STUDENT

SID	Advisor	Maj_GPA
123	Hawking	4.0
123	Mahler	3.3
456	Michener	3.2
789	Bach	3.7
678	Hawking	3.5

ADVISOR

Advisor	Major
Hawking	Physics
Mahler	Music
Michener	Literature
Bach	Music

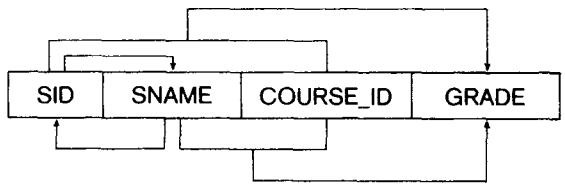
c) 具有样例数据的关系

图B-2 转换一个关系为BCNF

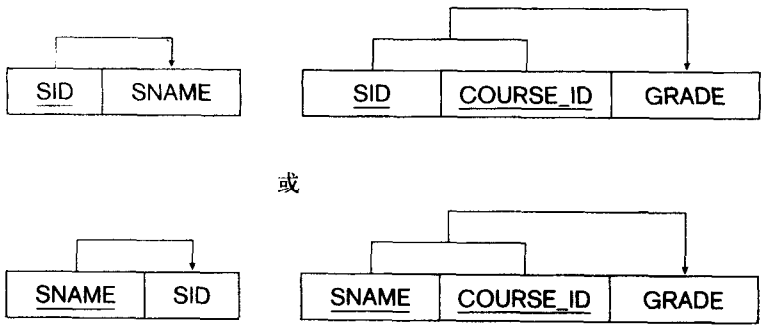
转换过程的第二步是分解关系以消除部分函数依赖，这在第5章介绍过。这导致两个关系，如图B-2b所示。这些关系符合3NF。事实上，这些关系也符合BCNF，因为每一个关系中只有一个候选键（主键），因此3NF和BCNF是等价的。

图B-2c给出了带有样例数据的两个关系（现在名为STUDENT和ADVISOR）。应该可以验证，这些关系没有STUDENT_ADVISOR中的异常。也可以验证，通过联结关系STUDENT和ADVISOR，可以重新创建STUDENT_ADVISOR关系。

另一个常见的违反BCNF的情况是当关系中有两个（或多个）交叠的候选键。考虑图B-3a中的关系。在这个例子中，有两个候选键（SID，COURSE_ID）和（SNAME，COURSE_ID）。其中，COURSE_ID出现在两个候选键中。该联系的问题是，除非学生已经开始学习某个课程，否则我们不能记录学生数据（SIC和SNAME）。图B-3b说明了两种可能的解决方法，每一种方法都创建了符合BCNF的两个关系。



a) 带有交叠候选键的关系



b) 符合BCNF的一对关系

图B-3　将带有交叠候选键的关系转换为BCNF

B.2　第四范式

当一个关系符合BCNF时，就不再有任何函数依赖导致的异常。但是，可能会存在多值依赖（在下面定义）导致的异常。例如，考虑图B-4a所示的表。这个用户视图显示了每一个课程，讲授某门课的教师以及所使用的教材。在该表中，假定下面假设成立：

1) 每一门课程都有一个定义良好的教师集合（例如，Management有三个教师）。

2) 每一门课程都有一个定义良好的教材的集合（例如，Finance有两本教材）。

3) 一门给定课程使用的教材不依赖于该门课程的教师。例如，不管三名教师中谁来教授Management，都使用同样的两本教材。

在图B-4b中，通过在每个空单元中填入值可以将这个表转换为关系。该关系（OFFERING）符合第一范式。因此，对于每一门课程，教师和教材的所有可能的组合都出现在OFFERING中。

注意，该关系的主键由三个属性组成：Course、Instructor和Textbook。因为除了主键外没有其他决定因子，所以该关系实际符合BCNF。然而，它确实包含了很多冗余数据，很容易导致更新异常。例如，假如要在Management课程中加入第三本教材（作者为Middleton）。完成这个修改需要在图B-4b的关系中新加入三行，即为每一个教师加入一行（否则该教材只能应用于某些教师）。

OFFERING

Course	Instructor	Textbook
Management	White Green Black	Drucker Peters
Finance	Gray	Jones Chang

a) 课程、教师、教材的表

OFFERING

Course	Instructor	Textbook
Management	White	Drucker
Management	White	Peters
Management	Green	Drucker
Management	Green	Peters
Management	Black	Drucker
Management	Black	Peters
Finance	Gray	Jones
Finance	Gray	Chang

b) 符合BCNF的关系

图B-4 多值依赖的数据

多值依赖

这个例子中所示的依赖类型称为**多值依赖**（multivalued dependency），当一个关系至少有三个属性时（例如，A、B和C）才会发生，并且对于A的每一个值，有一个定义良好的A值集合和B值集合。但是，B值集合不依赖于C值集合，反之亦然。

为了从一个关系中消除多值依赖，我们将该关系分解为两个新关系。其中每一个表包含两个属性，它们在原始表中具有多值联系。图B-5说明了对图B-4b中OFFERING关系的分解结果。注意，称为TEACHER的关系包含Course和Instructor属性，因为对于每一个课程，都有一个定义良好的教师集合。由于同样的原因，TEXT包含属性Course和Textbook。但是，没有关系包含Instructor和Textbook，因为这些属性相互独立。

如果一个关系符合BCNF并且不包含多值依赖，则它符合**第四范式**（Fourth Normal Form，4NF）。读者很容易验证图B-5中的两个关系符合4NF，并且不存在前面描述的异常。同样能够验证，可以通过联结这两个关系重构原始表（OFFERING）。另外，注意图B-5中的数据少于图B-4中的数据。为了简单起见，假定Course、Instructor和Textbook都是等长的。因为图B-4b中有24个数据单元，图B-5中有16个数据单元，所以4NF表节省了25%的空间。

TEACHER

Course	Instructor
Management	White
Management	Green
Management	Black
Finance	Gray

TEXT

Course	Textbook
Management	Drucker
Management	Peters
Finance	Jones
Finance	Chang

图B-5 符合4NF的关系

B.3 更高级范式

至少定义了两个更高级范式，即第五范式（5NF）和域键范式（DKNF）。第五范式处理称为“无损联结”的特性。根据Elmasri和Navathe（2000），“这些情况极少发生，而且实际上很难检测出来。”因此（也因为5NF的定义非常复杂），本书并不论述5NF。

域键范式（DKNF）试图定义一个考虑到所有可能的依赖类型和约束的“终极范式”（Elmasri和Navathe，2000）。尽管DKNF的定义非常简单，据作者所言“其实际用途非常有限”。因此，本书并不论述DKNF。

更多的信息参见Elmasri和Navathe（2000）以及Dutka和Hanson（1989）的著作。

B.4 参考文献

Dutka, A., and H. Hanson. 1989. *Fundamentals of Data Normalization*. Reading, MA: Addison-Wesley.

Elmasri, R., and S. Navathe. 2000. *Fundamentals of Database Systems*. 3rd ed. Reading, MA: Addison-Wesley, pp.440, 443.

附录C 数 据 结 构

数据结构是任何物理数据库体系结构的基本构造块。无论使用什么样的文件组织或DBMS，都要使用数据结构连接相关的数据。虽然许多现代DBMS隐藏了基本的数据结构，但是，调整物理数据库仍然需要理解数据库设计人员在数据结构方面所能作出的选择。本附录介绍所有数据结构的基本元素，并概述存储和查找物理数据元素的常用模式。

C.1 指针

在第6章已介绍过指针。如第6章所述，指针用于引用其他数据的地址。实际上，有三类指针（如图C-1所示）。

1. 物理地址指针

这种指针应包含被引用数据的实际的、完全解析的磁盘地址（设备、柱面、磁道和块）。物理指针是定位其他数据的最快的方式，但局限性也最大，因为如果被引用数据的地址改变，那么指向这个数据的所有指针也必须改变。物理指针通常在具有网状和层次数据库体系结构的遗留数据库应用中使用。

2. 相对地址指针

这种指针应包含关联数据与某个基地址（即开始点）的相对位置（或“偏移量”）。相对地址可以是字节位置、记录或行的数目。相对指针的优点是，在整个数据结构改变位置时，对这个结构的所有相对引用可保持不变。大多数DBMS都使用相对指针。它常常用于索引中，其中索引键与具有该键值的记录的行标识符（相对指针的一种类型）相匹配。

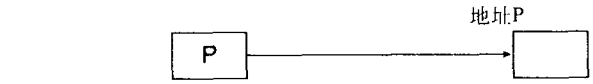
3. 逻辑键指针

这种指针应包含关于关联数据元素的有意义的数据。逻辑指针必须通过表查找、索引搜索或数学计算转换成物理指针或相对指针，以便真正找到被引用的数据。关系数据库中的外键通常是逻辑键指针。

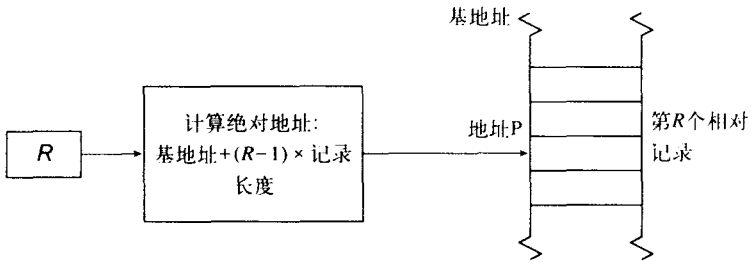
表C-1总结了这三类指针的主要特点。在数据库中，数据库设计人员可以根据不同的情况选择不同类型的指针。例如，关系中的外键能用这三类指针中的任何一类来实现。此外，在数据库被破坏时，了解使用哪种类型指针的数据库管理员能够很快重建数据库内容之间被破坏的链接。

表C-1 各类指针的比较

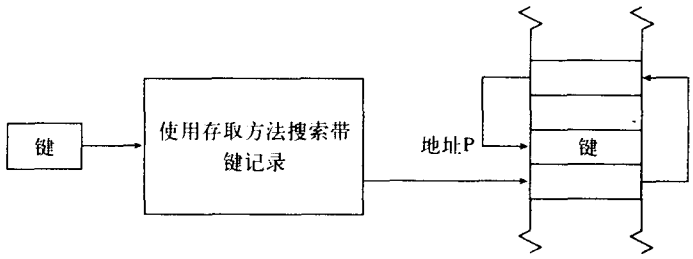
特 征	指针的类型		
	物理指针	相对地址指针	逻辑键指针
形式	实际辅存（磁盘）地址	与参照点（文件开始处）的偏移量	有意义的业务数据
访问速度	最快	中等	最慢
对数据移动的敏感性	最强	只对相对位置改变比较敏感	最小
对破坏的敏感性	非常敏感	非常敏感	通常能很容易地重构
空间需求	固定、通常较少	变化、通常最少	变化、通常最多



a) 物理地址指针



b) 文件中第R个记录的相对地址指针



c) 带键记录的逻辑键指针

图C-1　指针的类型

C.2　数据结构的构造块

所有的数据结构都由连接和定位数据的几个可选的基本构造块(building block)构成。连接方法允许在相关的数据元素之间移动。定位方法允许结构内的数据首先被放置或存储，然后被找到。

对于数据的连接元素只有两种基本方法:

1. 地址顺序连接

一个后继(或相关)元素被放在与当前元素紧邻的物理存储空间中(如图C-2a和C-2c所示)。若读取整个数据集或按存储顺序读下一个记录时，地址顺序连接的性能最好。反之，对于检索任意记录和数据更新(添加、删除和修改)操作，地址顺序连接的效率很低。由于必须保持物理顺序，一般要求重新组织整个数据集，因此更新操作也很低效。

2. 指针顺序连接

为了标识后继(或相关)数据元素的位置，必须在一个数据元素内存储一个(或多个)指针(如图C-2b和C-2d所示)。对于数据更新操作，指针顺序是比较有效的，这是因为只要保持相关数据之间的链接就能查找任何位置的数据。指针顺序方式的另一个主要特点是在同样的数据集内通过设置不同的指针保持许多不同的顺序链接。我们简单地回顾一下指针顺序方式的几种常用的形式(线性数据结构)。

此外，与连接机制有关的数据放置有两种基本方法:

1. 数据直接放置

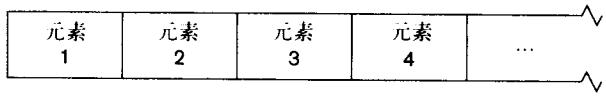
这种连接机制把一个数据项直接与其后继(或相关)数据项相链接(如图C-2a和C-2b所示)。

直接放置具有在遍历一个连接时可快速找到数据的优点。其缺点是实际的数据分散在大部分的磁盘存储器空间内，这是因为实际的空间必须在连接元素之间分配。

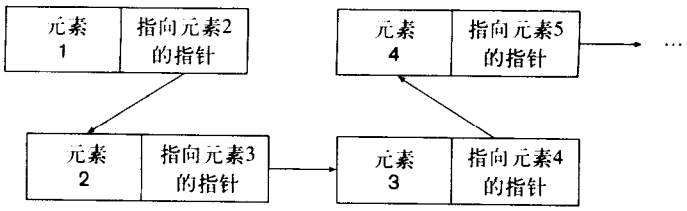
2. 数据间接放置

这种连接机制将指针链接到数据，但不是真正的数据（如图C-2c和C-2d所示）。间接放置的优点是扫描具有指定特征数据的数据结构时更加有效，这是因为通过关键特征的紧凑入口和指向关联数据的指针就能完成扫描。还有，连接和数据的放置是分离的，因此数据记录的物理组织可以采用最合适的方式（例如，对于排序顺序可采用物理顺序）。其缺点是在检索引用的数据和检索数据时都需额外的访问时间，同时指针也需要额外的空间。

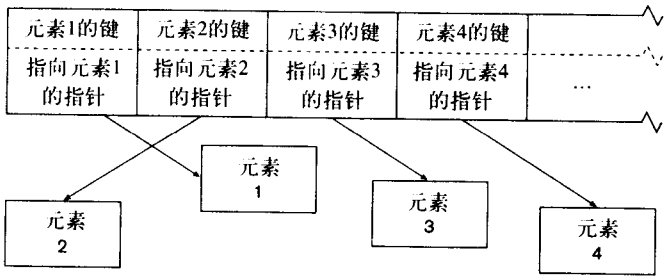
任何数据结构、文件组织或数据库体系结构都能使用这四种基本方法的组合方式。



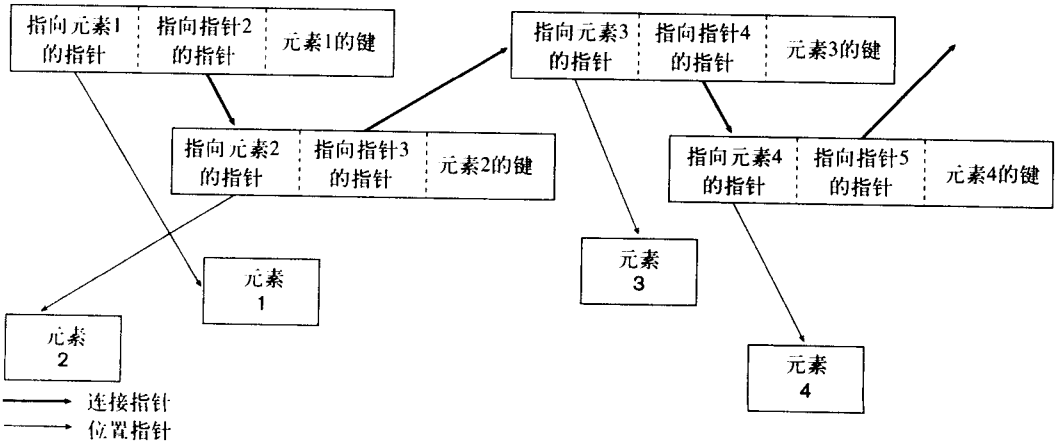
a) 地址顺序连接（顺序）



b) 指针顺序连接（简单的链或列表）



c) 地址顺序、数据间接连接（键索引）



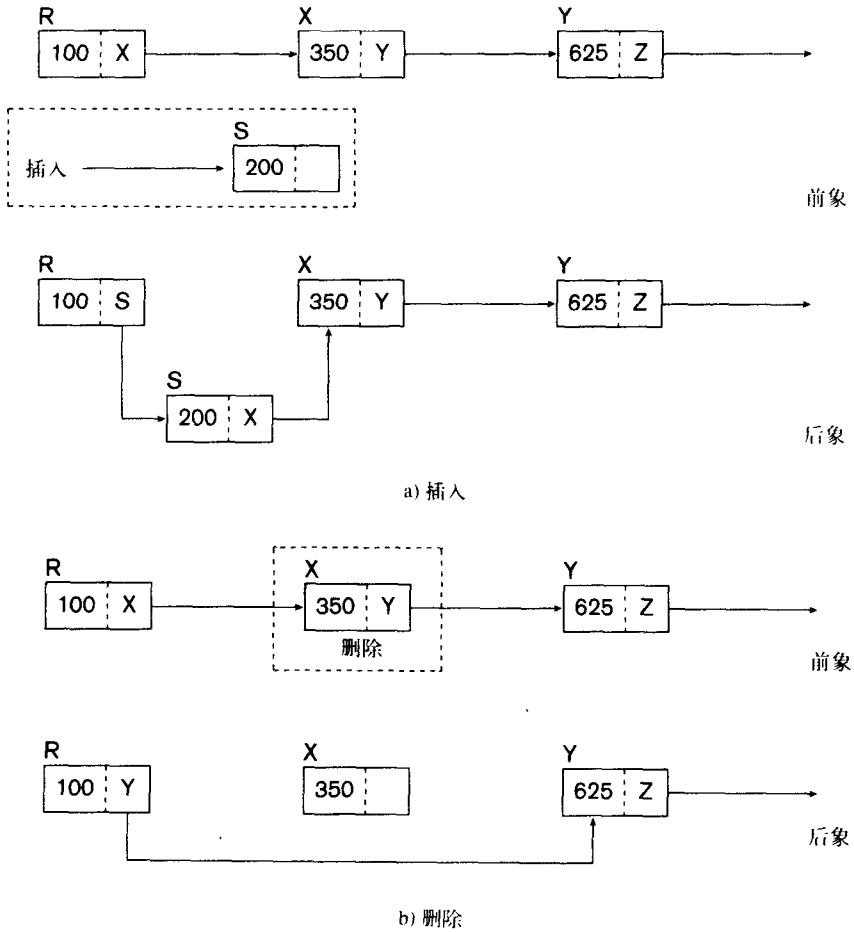
d) 指针顺序、数据间接连接（链键索引）

图C-2 基本定位方法

C.3 线性数据结构

指针顺序数据结构适合于存储易变的数据，这些数据可在运作数据库中找到。事务数据（例如顾客订单和个人修改请求）和历史数据（例如产品价格和学生班级注册）构成了运作数据库的主体部分。还有，由于运作数据库的用户以多种不同顺序查看数据（例如以订单日期、产品编号或顾客编号为序查看顾客订单），在同样数据上保持不同的指针链的这种能力可以支持这个数据集上的一大批用户需求。

使用线性数据结构处理数据更新的功能如图C-3所示。图C-3a显示了怎样轻松地把一个新记录插入到线性（或链式）结构中。这张图表示一个产品记录文件。为简单起见，每个产品记录只用一个产品编号和按产品编号顺序指向下一个产品记录的指针来表示。假设有一个存储在有效位置（S）的新记录，通过改变与位置R和S的记录有关的指针可以把这个新记录插入到链中。在图C-3b中，删除一个记录也一样容易，只要改变位置R的记录的指针即可。虽然存储指针需要额外的空间，但与存储产品的所有数据（产品编号、描述、现有数量、标准价格等等）的空间相比，这个空间还是最小的。按这个结构给出的产品编号来查找记录是很容易的，但是如果逻辑顺序记录在磁盘上分隔存储，那么顺序检索记录的实际时间是很长的。



图C-3　维护指针顺序结构

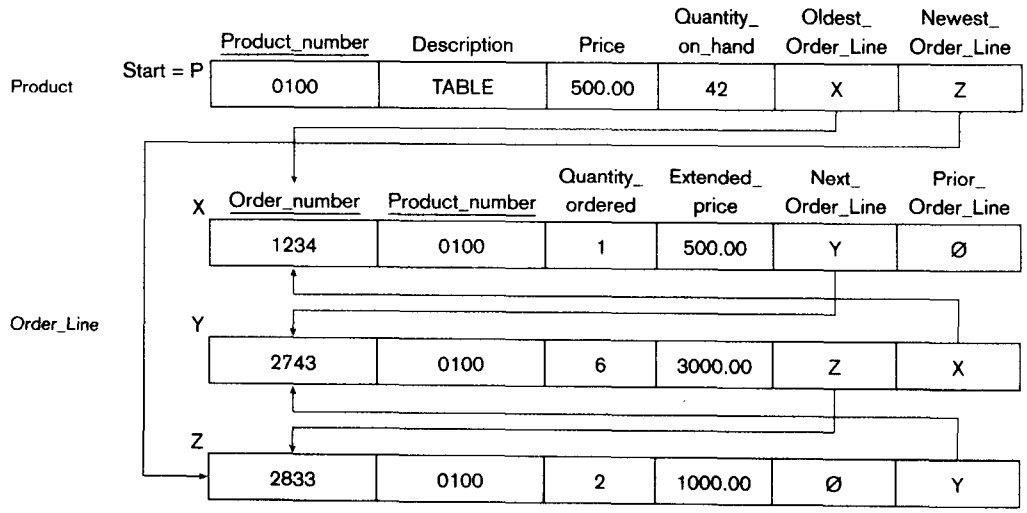
这个例子简要地说明了线性数据结构，下面考虑这种结构的四种情况：栈、队、排序表和多表。在本节结尾，我们将给出有关线性、链式数据结构给出几点忠告。

C.3.1 栈

栈具有所有记录的插入和删除操作均在数据结构的同一端执行的性质。栈具有后进先出（LIFO）性质。栈的一个就像是食堂里的一叠盘子。在业务信息系统中，栈用于保留无优先权和不排序的记录（例如与同一个顾客订单有关的行项）。

C.3.2 队列

队列具有所有插入均发生一端，而所有删除均发生在另一端的性质。队列具有先进先出（FIFO）性质。队列就像是超市里一支结账队伍。在业务信息系统中，队列用于以插入的顺序来维护记录的列表。例如，图C-4显示了松谷家具公司中以订购单行（Order Line）记录来遍历公共的产品（Product）记录的链队列。



图C-4 带双向指针的队列的例子

在这个例子中，Product记录在数据结构中是链首结点。Oldest_order_line字段值是指向产品0100中最早的（第一个输入的）Order Line记录的指针。Order_Line记录中的Next_order_line字段包含一个以时序顺序指向下一个记录的指针。指针中的ϕ称为空指针，表示链尾。

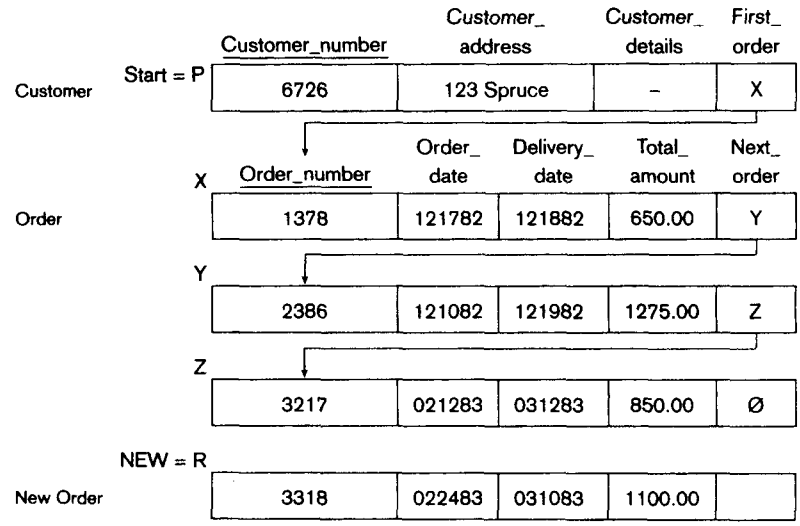
这个例子还介绍了双向链的概念，即有向前指针和后向指针。next和prior指针的好处是记录中的数据可以在前向或后向上进行检索和引用，并且维护链的代码实现起来要比单向链容易。

C.3.3 排序表

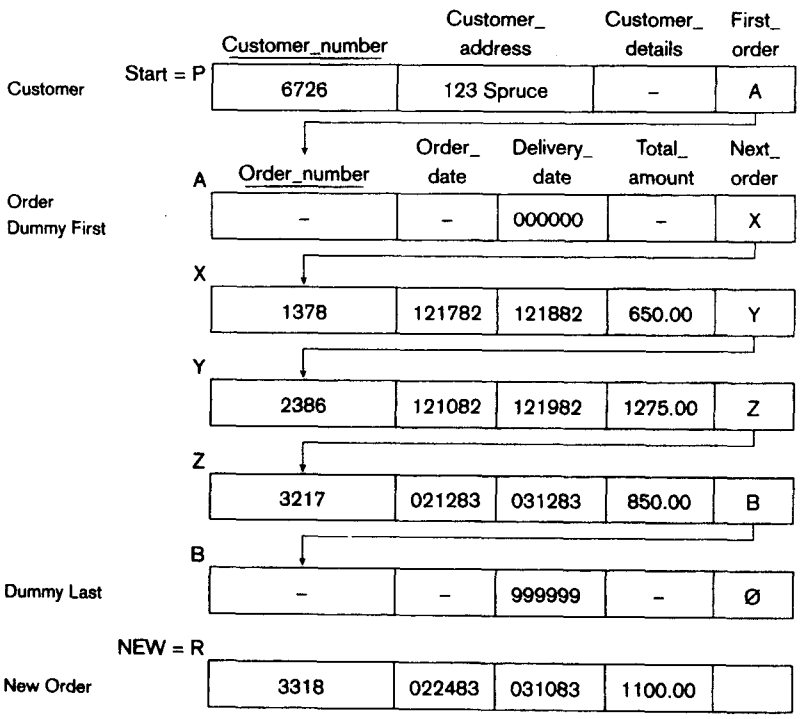
排序表具有插入和删除可发生在表中的任何地方、且记录必须以键字段值的逻辑顺序维护的性质。排序表就像是电话目录。在业务信息系统中，排序表经常出现。图C-5a显示了一个与顾客（Customer）记录有关的订单（Order）记录的单向、指针顺序排序的表，其中的记录是据Delivery_date排序的。

维护一个排序表比维护一个栈或队列更复杂，这是因为插入和删除可以发生在链的任何地方，并且这条链上可以有零个或多个已存在的记录。如果要保证插入和删除总是发生在链的中间，那么通常就要把链的第一个记录和最后一个记录置为哑（dummy）记录（参见图C-5b）。图C-5c显示了向图C-5b的排序表中插入一个新Order后的结果。为了执行插入，必须从指针First_order的地址处开始扫描列表。一旦找到链中的真正位置后，还必须有一条规则，以决定

如果表中允许有重复的键值，那么这个重复键值的记录应该存储在什么位置。通常，带重复键值的记录将成为这些重复键值记录中的第一个，这样只需最少的扫描。

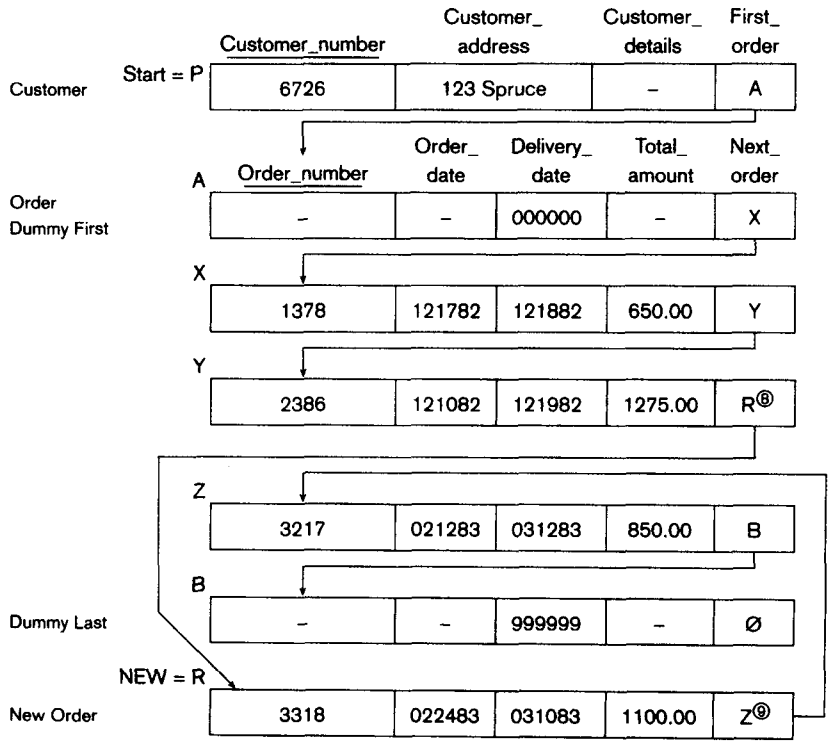


a) 在新的Order插入前，并且第一个和最后一个Order记录都不是哑记录



b) 在新的Order插入前，并且第一个和最后一个Order记录都是哑记录

图C-5　排序表的例子



c) 在新的Order插入后（紧靠指针小圆圈中的号码指出在改变指针值的相关维护过程中步骤的编号）

图C-5 （续）

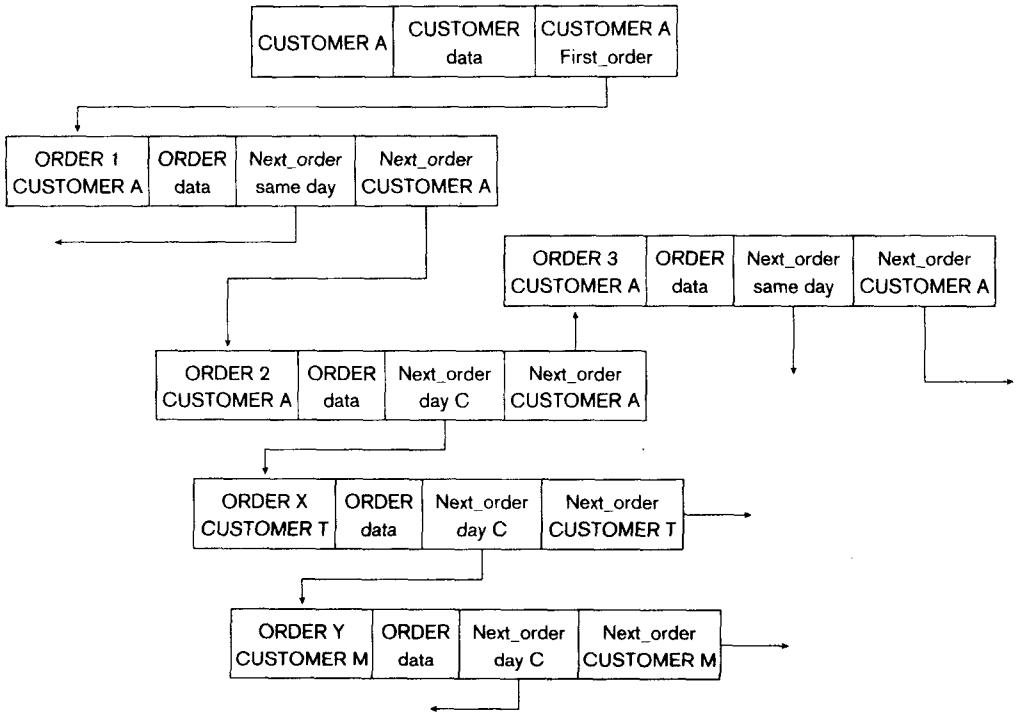
如果使用支持链，特别是支持排序表的文件组织或DBMS时，那么不一定需要编写维护列表的代码。相反，这个代码将在你使用的技术中存在。在你的程序中，只要发出插入、删除或更新命令，那么支持的软件将会完成链的维护工作。图C-6包含了在图C-5b的排序表中插入一个新记录时所需要的代码的概要。在这个概要中，位置变量PRE和AFT分别用于保存新的Order记录的前驱和后继的值。第7步写在方括号内，表示如果需要时重复键值将会在什么地方出现。符号“←”表示左边变量中的值将用右边变量中的值替换。图C-5中改变指针值的第8步和第9步显示了该图的例子中哪一个指针将被改变。但你很可能缺乏桌面检查这个子程序来通过人工执行去看这些变量如何被置值和改变。

```
/*  Establish position variables beginning values */
1   PRE ← First_order(START)
2   AFT ← Next_order(PRE)
/*  Skip/scan through chain until proper position is found */
3   DO WHILE Delivery_date(AFT) < Delivery_date(NEW)
        4   PRE ← AFT
        5   AFT ← Next_order(AFT)
6   ENDO
7   [If Delivery_date(AFT) = Delivery_date(NEW) then indicate a Duplicate Error and
    terminate procedure]
/*  Weld in new chain element */
8   Next_order(PRE) ← NEW
9   Next_order(NEW) ← AFT
```

图C-6 记录插入代码概要

C.3.4 多表

多表数据结构是在同样的记录之间保持多种顺序的数据结构。这样，同样的记录中存在多条链，并且不需重复存储就能以某种顺序来扫描这些记录。要在灵活的访问代价和额外的存储空间及每条链的维护工作之间作出权衡。使用多表，有可能遍历一个关联时转到另一个关联。例如，在根据给定的顾客（Customer）（一个表）访问订单（Order）记录时，我们可以找与给定的订单记录同一天交付的所有订单记录。这样的多表如图C-7所示。



图C-7　多表结构的例子

多表提供了和多索引相同的一些好处（见第6章有关主键索引和辅键索引的讨论）。多表的主要缺点以及它没有在关系DBMS中使用的主要原因是扫描表的代价要比访问索引的代价高，并且尚无快速方式响应多表的多键限定（例如要找Northwest区域的所有顾客记录以及Paper产品系列中的所有产品）。由于这一点和其他理由，在现代数据库技术中，索引一般替代了线性数据结构。但是，一些遗留应用可能仍然在使用单表或多表结构的技术。

C.4 链结构的危险性

除了禁止链在快速响应多键限定中使用这个局限性外，链还有下面的危险和限制：

1) 由于顺序存储的记录未必在物理上相邻存储，因此扫描一条链将花费很长的时间。

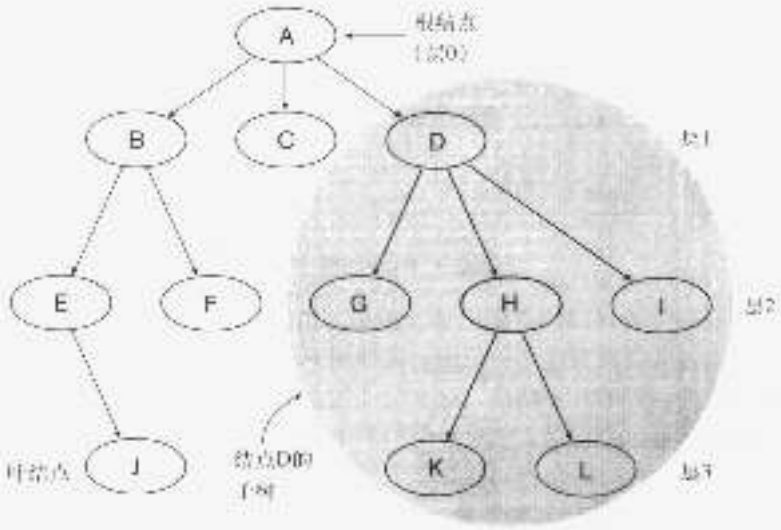
2) 链很容易被破坏。如果在链维护例程中间发生了异常事件，那么链可能被部分更新而链将变得不完整或不准确。某些安全性机制能处理这样的错误，但这些机制需要额外存储空间或处理开销。

C.5 树

线性数据结构可能会变得很长，因此扫描很费时，这是任何线性结构中的固有问题。还好，

利用分而治之策略的非线性结构已被开发出来。树就是一种常见的非线性结构。树（参见图C-8）是一种由一组结点组成的数据结构，其分支都起源于树的顶端结点。（因此树是倒置的！）树的顶端结点称为根结点。除了根结点外，树中每个结点都有一个父结点，并且可以有零个、一个或多个子结点。结点以层次方式定义：根的层次为0，其子结点的层次为1，以此类推。

树中没有子结点的结点称为叶结点（例如图C-8中的结点J、F、C、G、K、L和I）。一个结点的子树由这个结点及其所有子孙结点组合而成。



图C-8 树数据结构的例子

平衡树

在数据库管理系统中，树经常用于把数据组织到键索引中。和线性数据结构一样，由于这个结构是DBMS软件维护的，因此数据库程序员不必维护树结构。但是，数据库设计者可以控制索引树的结构，以便调整索引处理的性能。

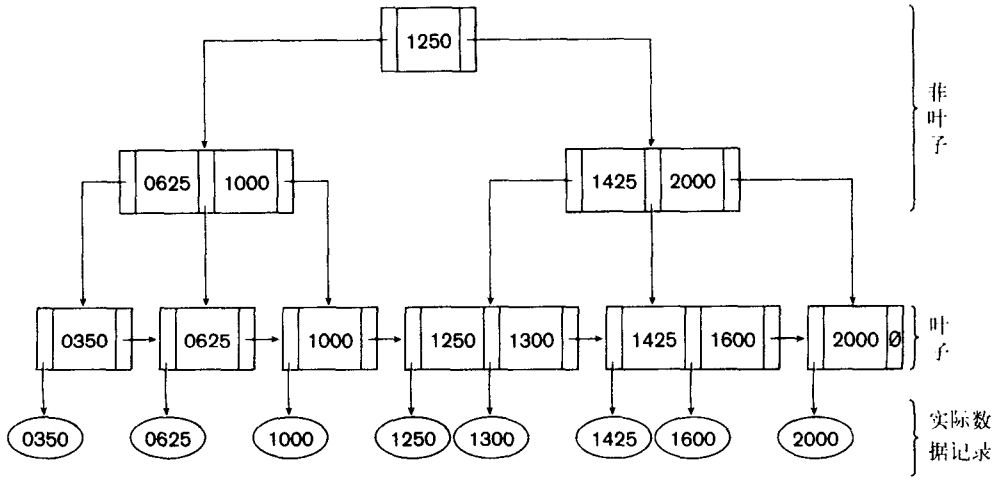
用于建立键索引的树的最常用的形式是平衡树，即B树。在B树中，所有叶子与根的距离相同。因此，B树是真正有效的。B树既支持记录的随机检索，也支持顺序检索。B树最常用的形式是B+树。度数为m的B+树有如下特殊的平衡树性质：

- 每个结点的子结点的数目在m/2和m之间（m是一个大于等于3的整数，通常是奇数）；而根结点不受这个下界的约束。

这个性质将导致结点的动态组织。这个性质在本节后面将加以说明。

许多操作系统支持的数据访问方法VSAM（虚拟顺序访问方法）是基于B树数据结构的。VSAM是ISAM（索引顺序存取方法）的比较现代的版本。ISAM和VSAM之间有两个主要的区别：1）ISAM中索引项的位置受磁盘驱动器的物理边界的限制。而VSAM中的索引项可以跨越物理边界。2）ISAM文件在插入和删除许多键后其文件结构会变得低效。因而需要重建；而VSAM中的索引在索引段变得不易控制时能以增量方式动态重组织。

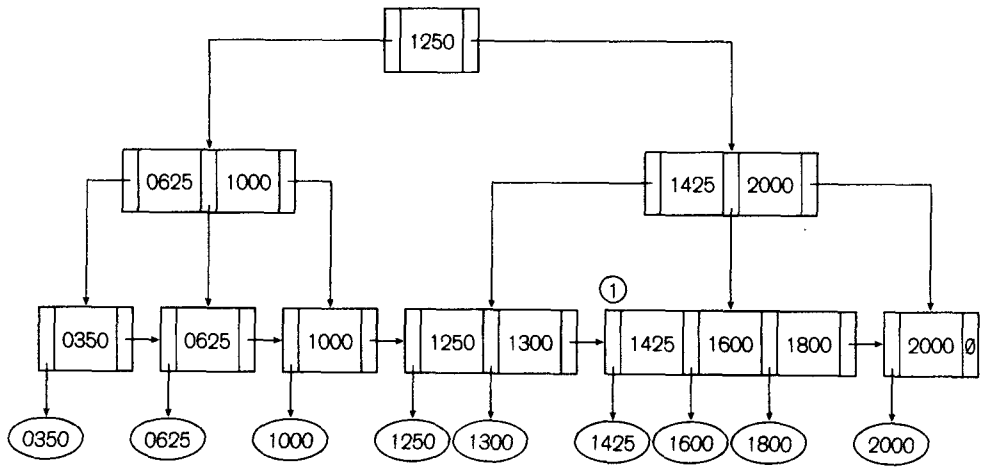
B+树（度数为3）的例子出现在图C-9松谷家具公司的Product文件中。在这个图中，每个垂直箭头表示值等于箭头左边的数值但小于箭头右边的数值这样一条路径。例如，在包含值625和1000的非叶结点中，从这个结点底部的中间引出的箭头是值等于625但小于1000的一条路径。水平箭头用于连接叶结点，以便不必在树的层次上移动就能进行顺序处理。



图C-9　B+树的例子

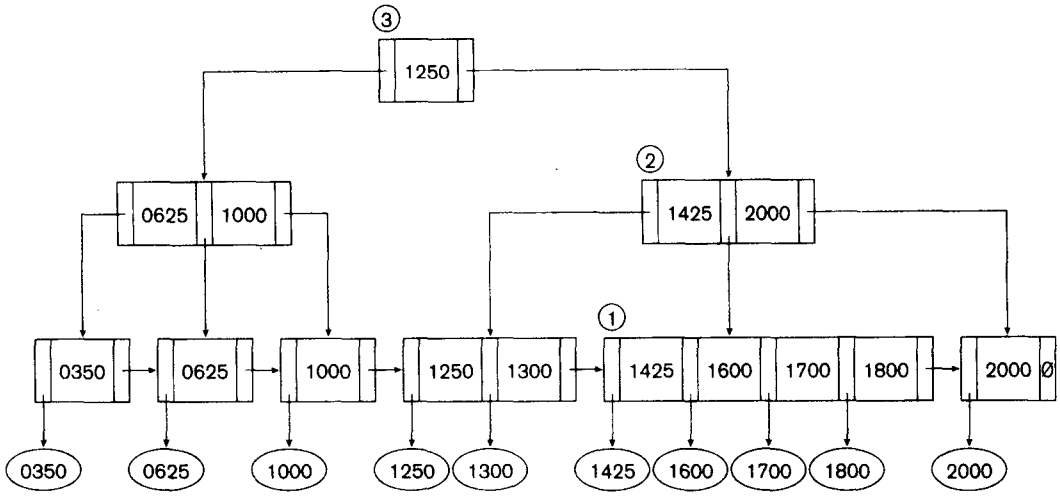
假设需检索产品编号为1425的数据记录。注意，根结点的值为1250。由于1425大于1250，因此沿着这个结点右边的箭头到达下一层次。在找到的目标值（1425）的结点中，再沿着中间箭头往下到达包含值为1425的叶子结点。这个结点包含一个指针，指向产品编号为1425的数据记录，因此现在能检索到这个记录。你也可以沿着类似的路径找到产品编号为1000的记录。由于数据记录存储在索引之外，因此在同样数据上可以维护多个B+树索引。

利用B+树还能支持记录的插入和删除。对B+树结构的改变都是动态的，并仍保留B+树的性质。考虑插入键值为1800的记录到图C-9的B+树中。插入的结果如图C-10a所示。由于结点1只有三个子结点（水平指针不作为子指针使用），图C-10a中的B+树仍然满足所有B+树性质。现在考虑插入另一个记录的效果。这次将1700插入到图C-10a的B+树中。这个插入的初始结果显示在图C-10b中。此时，结点1违反了度数的限制，因此这个结点必须分裂成两个结点。分裂结点1将会在结点2产生新的项，这样这个结点会有四个子结点，多了一个。因此，结点2必须也分裂，并加一个新的项到结点3。最后的结果如图C-10c所示。

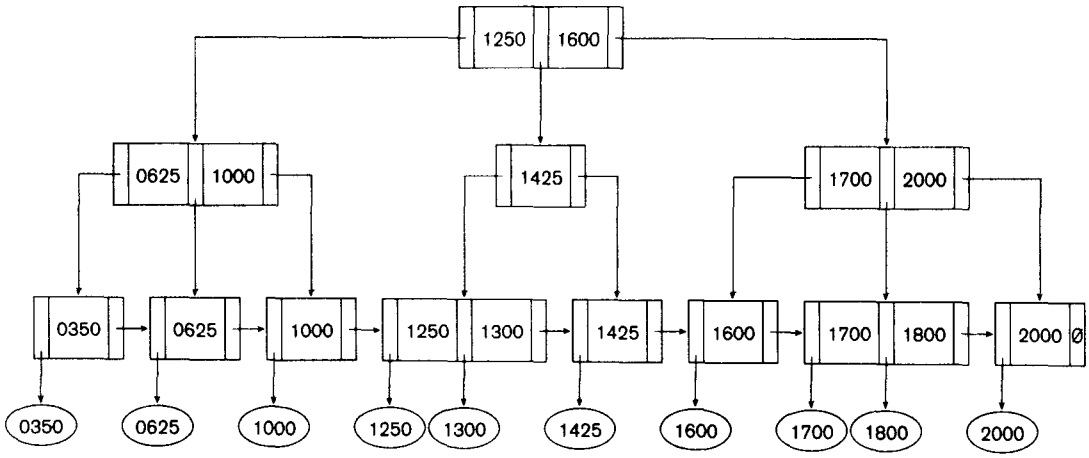


a) 插入记录1800

图C-10　在B+树中插入记录



b) 开始试图插入记录1700



c) 插入记录1700后的最终的B+树

图C-10 （续）

在根变得太大（大于m个子女）时，也会发生有趣的事情。在这种情况下，根要分裂，树要增加一个层次。删除记录会删除叶结点中的一个项。如果这个删除使叶结点的子女少于m/2个，那么叶结点就要与邻近的叶结点合并；如果合并后的叶结点太大（多于m个子女），那么合并的叶结点就要分裂，直接导致键在结点之间作一些倾斜的重新分布，结果使B+树重新动态地组织，从而使树保持平衡（从根引出的任何路径深度相等），并且每个结点的项都在限定范围内（这样控制了树的茂盛度，即宽度）。

如果要学习关于B树更多的内容，请阅读Comer（1979）有关B树性质和设计的一篇经典文章。

C.6 参考文献

Comer, D.1979, "The Ubiquitous B-tree." *ACM Computing Surveys* 11 (June) : 121-37.

附录D 对象-关系数据库

本书描述了设计和实现数据库时使用最广泛的两种模型：关系型和面向对象型。关系模型（与关系数据库管理系统）是目前最常使用的实现主流业务应用的模型。这个模型的优势包括：

- 数据独立性，已在第1章介绍过。
- 功能强大的查询语言（SQL），已在第7章介绍过。
- 成熟的关系数据库管理系统技术，包含在线备份和恢复、灵活的并发控制等特性（已在第12章介绍过）。

尽管有这些优势，但我们在第14章（以及其他章中）已提到，关系数据模型不适合处理复杂数据类型，例如图像、音频、视频和空间（或地理）数据。因此，引入了面向对象数据模型来管理这些复杂的数据类型（这种模型已在第14、15章介绍过）。面向对象数据库管理系统的市场增长较慢，但是这种技术在集成了多媒体数据类型的Web服务器这样的应用方面有着非常好的前景。面向对象数据库系统的弱点（至少到目前）是不具备关系系统所具有的查询功能、在线备份和恢复、灵活的并发控制等重要特性。

总之，关系和面向对象数据库系统各有特色和弱点。一般来说，一种系统的弱点通常是另一种系统的特色（反之亦然）。这就导致业界开发新一代混合数据库系统——对象-关系系统，企图把这两种模型的优点结合起来（Stonebraker等，1990）。大多数主流的DBMS供应商都开发了对象-关系数据库系统。这些系统能处理对象和规则、封装、多态和继承，它们与RDBMS是相容的。本附录将对这种系统作简短的介绍，包括它的优点和缺点。

D.1 基本概念和定义

对象-关系数据库管理系统（Object-Relational Database Management System，ORDBMS）是以集成的方式支持关系型和面向对象特色的数据库引擎（Frank，1995）。这样，用户在使用公共接口（例如SQL）时能定义、操纵、查询关系数据和对象。DBMS的基本数据模型是关系型。这样，尽管数据库以面向对象形式出现在程序员面前，但存储和处理仍是关系型的；因此在关系和对象型之间映射需要花额外的开销。

在对象-关系数据库中还使用两个术语：扩展关系（extended relational）和通用服务器（universal server）。术语*扩展关系*（比较旧的术语）起源于在已有的关系DBMS中加入面向对象特性而开发的第一个版本。术语*通用服务器*（比较现代的术语）是基于用相同服务器技术管理任何类型数据（包含用户定义的数据类型）的已确定目标。

D.1.1 ORDBMS的特性

正如前面的内容所述，ORDBMS的作用是把关系数据库模型和对象数据库模型的最好特性结合起来。其主要特性通常包含下面两点：

1) SQL的增强版本（类似于SQL3）可用来创建和操纵关系表和对象类型或类。

2) 支持包含继承、多态、用户定义（或抽象）数据类型和导航式访问的传统的面向对象函数。

D.1.2 复杂数据类型

在SQL中有三种主要的数据类型：NUMERIC、CHARACTER和TEMPORAL。NUMERIC

类型包含DECIMAL和INTEGER两种，而TEMPORAL类型包含DATE和TIME两种（Celko，1995）。

现代业务应用通常需要存储和操纵关系系统不易处理的复杂数据类型。这些复杂数据类型中比较常见的几种在表D-1中列出。对于每种数据类型，这个表提供简短的描述和若干典型的应用。例如，Graphic数据类型由点、线和圆等几何对象组成；其典型的应用有计算机辅助设计（CAD）、计算机辅助制造（CAM）、计算机辅助软件工程（CASE）工具和演示图（例如流程图和曲线图）。

表D-1 复杂数据类型

数据类型	描 述	应 用 实 例
Image	位映射表示	文档、照片、医疗图像、指纹
Graphic	几何对象	CAD、CAM、CASE、演示图
Spatial	地理对象	地图
Recursive	嵌套对象	材料单
Audio	声音剪辑	音乐、动画、游戏
Video	视频段、电影	教学片、产品演示、电影
Array	下标变量	时序、多维数组、多值属性
Computational-intensive	需要大量计算的数据	数据挖掘、金融工具（例如派生）

D.2 增强的SQL

ORDBMS最强大的特性是能使用关系查询语言（如SQL3）的扩充版本来定义、检索和操纵数据（Cattell，1994；Chaudhri和Zicari，2001）。

D.2.1 简单的例子

假设一个组织想要创建一个名为EMPLOYEE的关系来记录员工数据（参见图D-1）。表中传统的关系属性是Emp_ID、Name、Address和Date_of_Birth。此外，组织想要存储每个员工的照片以便确定员工身份。这个属性（其类型由一个对象定义）在图D-1中命名为Emp_Photo。在检索一个员工的数据时，员工照片将与其他数据一起显示。

Emp_ID	Name	Address	Date_of_Birth	Emp_Photo
12345	Charles	112 Main	04/27/1973	
34567	Angela	840 Oak	08/12/1967	
56789	Thomas	520 Elm	10/13/1975	

图D-1 带复杂对象的关系

SQL3提供的语句既能创建表也能创建数据类型（即对象类）。例如，为EMPLOYEE所用的SQL CREATE TABLE语句如下所示:

```
CREATE TABLE EMPLOYEE
  Emp_ID INTEGER NOT NULL,
  Name CHAR(20) NOT NULL,
  Address CHAR(20),
  Date_of_Birth DATE NOT NULL,
  Emp_Photo IMAGE);
```

例中的IMAGE是数据类型（即类），Emp_Photo是类中的一个对象。IMAGE可以是预定义的类，也可以是用户定义的类。如果是用户定义的类，那么就要用SQL CREATE CLASS语句来定义这个类。用户还可以定义在类中定义的数据上对数据操作的方法。例如，IMAGE上的一个方法是Scale（），用来放大或缩小照片的尺寸。方法是用Java、C++或Smalltalk来编写的，并与IMAGE CALSS封装在一起。方法还可以像属性一样，在查询的SELECT表中使用。

D.2.2 内容寻址

关系SQL语言功能最强大的一个特性是能在表中选择满足限定条件的记录子集。例如，在EMPPLOYEE关系中一个简单的SQL语句用来选择所有生日在12/31/1940或之前的员工的记录（见第7章）。相比之下，“纯”对象数据库系统不具有这种查询能力。相反，访问对象是导航式的，即每个对象都有一个惟一的对象标识符（参见第14章）。

ORDBMS产品扩展关系内容寻址能力来查找复杂对象。（Norman和Bloor，1996）。**内容寻址**（content addressing）是一种允许用户查询数据库以选择满足一个限定条件的所有记录或对象的机制。

SQL3包含对复杂数据类型的内容寻址的扩展能力。先来看在EMPLOYEE表中怎样应用内容寻址。应记住，每个员工记录都包含（或至少链接到）该员工的一张照片。用户可能要提出下列查询：给定一个人的照片，通过扫描EMPLOYEE表来决定是否存在与这张照片高度匹配的员工，然后显示选中的员工或若干员工的数据（包含照片）。假定照片的电子图像存储在名为My_Photo的位置。那么这种情况的查询可写成下列形式:

```
SELECT *
   FROM EMPLOYEE
   WHERE My_Photo LIKE Emp_Photo;
```

ORDBMS的内容寻址是允许用户搜索与图像、音频或声频节段、文档等多媒体对象匹配的数据的一种功能强大的特性。这个特性的一个常见的应用就是搜索数据库，查找与指纹或声音匹配的项目。

D.3 对象-关系方法的优点

对于需要在大数据集上进行复杂处理或重要的查询活动的应用而言，与关系数据库系统相比，对象-关系方法具有下列优点（Moxon，1997）:

1）减少了网络传输。使用方法来扫描数据的查询可以完全在服务器上执行，不必发送大量数据返回到客户机。

2）应用和查询性能。处理大型数据集的方法能充分利用大型并行服务器来很好地改善性能。

3）软件维护。数据与方法一起存储在服务器上极大地简化了软件维护的工作。

4) 集中式数据和事务管理。所有事务的完整性、并发性、备份和恢复都在数据库引擎中处理。

D.4 ORDBMS供应商和产品

主要的供应商和他们提供的ORDBMS产品见表D-2。表中还显示了每一个供应商允许用户定义自己的数据类型以及操纵数据的方法的扩展技术（或框架）。

表D-2 ORDBMS供应商和产品

供应商	产 品	Extender技术	站 点
IBM	DB2 Universal Database	DB2 Extenders	www_4.ibm.com/software/data/db2/udb/
Informix	Dynamic Server	Data Blades	www.informix.com/informix/products/ integration/ datablade/ datablade_ds.htm
Oracle	Oracle 8	Data Cartridges	www.oracle.com/oramag/webcolums/ora8804.html
Computer Associates	ODB-II (Jasmine)	none	www.cai.com/products/jasmine.htm

D.5 参考文献

Cattell, R. 1994. *Object Data Management: Object-Oriented and Extended Relational Database Systems*. Reading, MA: Addison-Wesley, pp. 303-304.

Celko, J. 1995. *Instant SQL Programming*. Chicago: Wrox Press, pp. 54-55.

Chaudhri, A.B., and R. Zicari. 2001. *Succeeding with Object Databases*. New York: Wiley.

Frank, M. 1995. "Object-Relational Hybrids." *DBMS* 8 (July): 46-56.

Melton, J. 1995. "A Case for SQL Conformance Testing." *Database Programming & Design* 10 (7): 66-69.

Moxon, B. 1997. "Database out of This Universe." *DB2 Magazine* 2 (Fall): 9-16.

Norman, M., and R. Bloor. 1996. "To Universally Serve." *Database Programming & Design* 9 (July): 26-35.

Stonebraker, M., et al. 1990. "Third-Generation Database System Manifesto—The Committee for Advanced DBMS Function." *SIGMOD Record* 19, 3: 31-44.

D.6 Web资源

参见表D-2。

- www.object-relational.com Barry&Associates公司的网站，包括许多对象-关系数据库管理系统特性的比较。
- http://technet.oracle.com/products/oracle8/info/objwp3/x003twp.htm. 该文章是对Oracle8的对象-关系数据库特性的综述。

术 语 缩 写

ACM	Association for Computing Machinery（美国计算机协会）
AITP	Association of Information Technology Professionals（信息技术专业学会）
ANSI	American National Standards Institute（美国国家标准学会）
API	Application Program Interface（应用程序接口）
ASCII	American Standards Code for Information Interchange（美国信息交换标准码）
ASP	Active Server Pages（活动服务器页面）
BCNF	Boyce-Codd Normal Form（Boyce-Codd范式）
BOM	Bill of Materials（材料单）
B2B	Business-to-Business（企业对企业）
B2C	Business-to-Consumer（企业对消费者）
CAD/CAM	Computer-Aided Design/Computer-Aided Manufacturing（计算机辅助设计/计算机辅助制造）
CASE	Computer-Aided Software Engineering（计算机辅助软件工程）
CD-ROM	Compact Disk-Read-Only Memory（光盘只读存储器）
CFML	ColdFusion Markup Language（ColdFusion标记语言）
CGI	Common Gateway Interface（公共网关接口）
COM	Component Object Model（组件对象模型）
CPU	Central Processor Unit（中央处理单元）
CRM	Customer Relationship Management（客户关系管理）
CRT	Cathode Ray Tube（阴极射线管）
C/S	Client/Server（客户/服务器）
CSF	Critical Success Factor（关键成功因素）
CSS	Cascading Style Sheets（层叠样式表）
DA	Data Administrator (或Data Administration)（数据管理员或数据管理）
DBA	Database Administrator (或Database Administration)（数据库管理员或数据库管理）
DBD	Database Description（数据库描述）
DBMS	Database Management System（数据管理系统）
DB2	Data Base2 (一种IBM关系型 DBMS)
DCL	Data Control Language（数据控制语言）
DDL	Data Definition Language（数据定义语言）
DFD	Data Flow Diagram（数据流图）
DK/NF	Domain-Key Normal Form（域-键范式）
DML	Data Manipulation Language（数据操纵语言）
DNS	Domain Name Server（域名服务器）
DSS	Decision Support System（决策支持系统）
EDI	Electronic Data Interchange（电子数据交换）

EDW	Enterprise Data Warehouse（企业数据仓库）
EER	Extended Entity-Relationship（扩展实体-联系）
E-R	Entity-Relationship（实体-联系）
ERD	Entity-Relationship Diagram（实体-联系图）
ERP	Enterprise Resource Planning（企业资源规划）
ETL	Extract-Transform-Load（抽取-转换-装载）
FK	Foreign Key（外键）
GUI	Graphical User Interface（图形用户界面）
HTML	Hypertext Markup Language（超文本标记语言）
HTTP	Hypertext Transfer Protocol（超文本传输协议）
IBM	International Business Machines（国际商用机器公司，IBM公司）
I-CASE	Integrated Computer-Aided Software Engineering（集成计算机辅助软件工程）
ID	Identifier（标识符）
IDS	Intrusion Detection System（非法入侵检测系统）
I/O	Input/Output（输入/输出）
IRDS	Information Repository Dictionary System（信息库词典系统）
IRM	Information Resource Management（信息资源管理）
IS	Information System（信息系统）
ISA	Information Systems Architecture（信息系统体系结构）
ISAM	Indexed Sequential Access Method（索引顺序存取方法）
ISO	International Standards Organization（国际标准化组织）
IT	Information Technology（信息技术）
ITAA	Information Technology Association of America（美国信息技术协会）
JDBC	Java Database Connectivity（Java数据库连接）
JSP	Java Server Pages（Java服务器页面）
LAN	Local Area Network（局域网）
LDB	Logical Database（逻辑数据库）
LDBR	Logical Database Record（逻辑数据库记录）
MB	Million Bytes（兆字节）
MIS	Management Information System（管理信息系统）
M:N	Many-to-Many（多对多）
M:1	Many-to-One（多对一）
MOLAP	Multidimensional On-line Analytical Processing（多维联机分析处理）
MPP	Massively Parallel Processing（大规模并行处理）
MVCH	Mountain View Community Hospital（山景社区医院）
NUPI	Nonunique Primary Index（非惟一主索引）
NUSI	Nonunique Secondary Index（非惟一辅索引）
ODBC	Open Database Connectivity（开放数据库连接）
ODBMS	Object Database Management System（对象数据库管理系统）
ODL	Object Definition Language（对象定义语言）
ODS	Operational Data Store（运作型数据存储）

OLAP	On-line Analytical Processing（联机分析处理）
OLE	Object Linking and Embedding（对象链接和嵌入）
OLTP	On-line Transaction Processing（联机事务处理）
OO	Object-Oriented（面向对象）
OODM	Object-Oriented Data Model（面向对象数据模型）
OQL	Object Query Language（对象查询语言）
O/R	Object/Relational（对象/关系）
ORDBMS	Object-Relational Database Management System（对象-关系数据库管理系统）
OSAM	Overflow Sequential Access Method（溢出顺序存取法）
PC	Personal Computer（个人计算机）
PK	Primary Key（主键）
PVF	Pine Valley Furniture（松谷家具公司）
QBE	Query-by-Example（按例查询）
RAID	Redundant Array of Inexpensive Disks（廉价冗余磁盘阵列）
RDBMS	Relational Database Management System（关系数据库管理系统）
ROLAP	Relational On-line Analytical Processing（关系联机分析处理）
SDLC	Systems Development Life Cycle（系统开发生命周期）
SGML	Standard Generalized Markup Language（标准通用标记语言）
SMP	Symmetric Multiprocessing（对称多处理）
SQL	Structured Query Language（结构化查询语言）
SQL/DS	Structured Query Language/Data System (an IBM relational DBMS)（结构化查询语言/数据系统，一种IBM关系数据库）
SSL	Secure Sockets Layer（安全套接层）
TCP/IP	Transmission Control Protocol/Internet Protocol（传输控制协议/网间协议）
TQM	Total Quality Management（全面质量管理）
UDF	User-Defined Function（用户定义函数）
UDT	User-Defined Datatype（用户定义数据类型）
UML	Unified Modeling Language（统一建模语言）
UPI	Unique Primary Index（惟一主键索引）
URL	Uniform Resource Locator（统一资源定位器）
USI	Unique Secondary Index（惟一辅索引）
VBA	Visual Basic for Applications（编写应用程序用的VB语言）
VLDB	Very Large Database（大规模数据库）
W3C	World Wide Web Consortium（万维网联盟）
WWW	World Wide Web（万维网）
XHTML	Extensible Hypertext Markup Language（可扩展超文本标记语言）
XML	Extensible Markup Language（可扩展标记语言）
XSL	Extensible Style Language（可扩展样式语言）
1:1	One-to-One（一对一）
1:M	One-to-Many（一对多）
1NF	First Normal Form（第一范式）

2NF	Second Normal Form（第二范式）
3NF	Third Normal Form（第三范式）
4NF	Fourth Normal Form（第四范式）
5NF	Fifth Normal Form（第五范式）

术 语 表

注意：在后面的括号中列出的是定义术语的章或附录。

中止事务（Aborted transaction）：在进行过程中异常中断的事务。（12）

抽象类（Abstract class）：没有直接实例的类，但它的子孙可以有直接实例。（14）

抽象操作（Abstract operation）：定义操作的形式或协议，但没有定义实现的一种操作。（14）

动作（Action）：可以在数据对象上执行的操作，如创建、删除、更新、或读取。（4）

动作断言（Action assertion）：对组织动作进行约束或控制的语句。（4）

主动数据仓库（Active data warehouse）：一种企业数据仓库，可以接近实时地从记录系统中接受事务数据的输入，并立即将合适的数据转换并载入数据仓库，为事务处理系统提供对企业数据仓库接近实时的访问。（11）

ActiveX：由微软开发的一组松散定义的技术，它扩展了浏览器功能并允许操纵浏览器的内部数据。（10）

后象（After-Image）：记录(或者内存页)在修改后得到的一个拷贝。（12）

聚合（Aggregation）：把数据从细节级转换到汇总级的过程（11）。组件对象和聚合对象之间的PART-OF联系。（14）

别名（Alias）：属性的一个可选的名字。（5）

锚点对象（Anchor object）：关于限制某些动作的业务规则（事实）。（4）

异常（Anomaly）：当用户试图更新含有冗余数据的表时，可能发生的错误和不一致性。三种异常类型是插入、删除和修改。（5）

应用分割（Application partitioning）：在编写完毕后，将各部分的应用程序代码分派给不同的客户端或服务器分区的过程，以获得更好的性能和互操作性（即一个组件在不同平台上执行的能力）。（9）

应用程序接口（Application Program Interface，API）：计算机操作系统向应用程序提供的例程集合。（9）

数组（Array）：能用位置定位的元素的有序集合，其大小可以动态变化。（15）

关联（Association）：两个或多个对象类之间已命名的联系。（14）

关联类（Association class）：自身具有属性或操作的关联，或者参与其他类的关系的关联。（14）

关联角色（Association role）：关联中与类连接的那一端。（14）

关联实体（Associative entity）：一种实体类型，关联一个或多个实体类型实例，并拥有专属于那些实体类型之间的联系自身的属性（3）

异步分布式数据库（Asynchronous distributed database）：分布式数据库技术的一种形式，其中复制数据的拷贝保存在另外的结点上，这样，本地服务器不通过网络就能够访问数据。（13）

原子文字（Atomic literal）：不能再进一步分解为其他组件的常数。（15）

属性（Attribute）：组织所关心的实体类型的性质或特征。（3）

属性继承（Attribute inheritance）：子类型实体继承超类型所有属性值的一种性质。（4）

授权规则（Authorization rule）：集成在数据库管理系统中，限制对数据的访问，并限制人们在访问数据时所能采取的动作的各种控制法则。（12）

备份工具（Backup facility）：产生整个数据库备份拷贝（保留本）的一种自动转储工具。（12）

后向恢复（回滚）（Backward recovery（rollback））：退出或者撤销不想要的数据库变更，即把被变更的记录的前象应用于数据库，并使数据库回归到原先的状态。常用于逆转由中止或异常中断的事务所作出的变更。（12）

包（Bag）：包含重复元素的无序集合。（15）

基表（Base table）：关系数据模型中的一种表，包含有插入的原始数据，基表对应于数据库概念模式中确定的关系。（7）

前象（Before-image）：记录(或者内存页)在修改前的一个拷贝。（12）

行为（Behavior）：表示对象如何动作和反应的。（14）

二元联系（Binary relationship）：两个实体类型的实例之间的联系。（3）

生物设备（Biometric device）：度量或者检测个人特征（如指纹、声纹、眼球图或者签名动力学等）的技术。（12）

位图索引（Bitmap index）：一个位表，它的每一行代表键的一个不同取值，而每一列是一位。当该位为1时，则表明这一列位置上的记录包含有相应的字段值。（6）

块因子（Blocking factor）：一页中物理记录的数目。（6）

Boyce-Codd范式（Boyce-Codd Normal Form, BCNF）：关系中每个决定因子都是一个候选键。（B）

浏览器（Browser）：一种软件，它能显示HTML文档并且允许用户访问与HTML文档相关的文件和软件。浏览器基于超链接的使用，通过点击某个对象，超链接允许用户从一个文档跳到另一个文档。大多数浏览器支持用户下载和传输文件、访问新闻组、播放音频和视频文件、执行由代码编写的小模块（如Java applets或ActiveX控件）。（10）

业务功能（Business function）：支持企业某方面任务的一组相关业务流程。（2）

业务规则（Business rule）：定义或约束业务中有关方面的语句，目的在于声明业务的结构以及控制或影响业务的行为。（3）

企业对企业（B2B，Business to Business）：用来描述与包括供应商和厂商在内的其他企业之间进行电子业务的术语。（10）

企业对客户（B2C，Business to Consumer）：用来描述对消费者进行零售业务的电子业务术语。（10）

候选键（Candidate key）：惟一标识关系中某行的属性或属性集。（5）

基数约束（Cardinality constraint）：与某实体的每一个实例相联系的另一个实体实例的个数。（3）

层叠样式表（Cascading Style Sheet，CSS）：由W3C开发的样式表，定义不同元素的外观并可应用到任何Web页面上。之所以称之为层叠样式表，是因为可以将多个样式表应用到同一Web页面上。（10）

目录（Catalog）：一系列放到一起可以构成对数据库的描述的模式。（7）

检查点工具（Checkpoint facility）：DBMS用来周期性地停止接收任何新事务的一种工具，此时，系统应处于静止状态，数据库和事务日志应是同步的。（12）

类图（Class diagram）：类图显示面向对象模型的静态结构：对象类、对象类的内部结构和对象类参与的联系。（14）

类范围属性（Class-scope attribute）：类范围属性是类的一个属性，其值在整个类中都是一样的，而不是每个实例有一个特定的值。（14）

客户/服务器体系结构（Client/server architecture）：参见Client/server system（客户/服务器系统）。

客户/服务器系统（Client/server system）：提供请求服务的客户和服务器之间分布式处理的网络计算模型。在数据库系统中，数据库通常位于处理DBMS的服务器上。客户通常处理应用系统或从其他包含应用程序的服务器中请求服务。（9）

ColdFusion**标记语言**（ColdFusion Markup Language，CFML）：用于创建ColdFusion应用页面脚本的语言。该语言以HTML为模板构成，它包含执行诸如读和更新数据库表等操作的标记。（10）

集合文字（Collection literal）：文字或对象类型的集合。（15）

提交协议（Commit protocol）：确保事务或者成功地完成或者被中止的一种算法。（13）

公共网关接口（Common Gateway Interface，CGI）：一种Web服务器接口，用于指定Web服务器与CGI程序之间信息的传递。（10）

完备性约束（Completeness constraint）：一种约束类型，用于解决如下问题：超类型的实例是否必须是至少一个子类型的成员。（4）

复合属性（Composite attribute）：可分解为更小组成部分的属性。（3）

复合标识符（Composite identifier）：由复合属性组成的标识符。（3）

复合键（Composite key）：由多个属性构成的主键。（5）

复合（Composition）：属于一个整体对象的某个部分对象，并与整体对象共存亡。（14）

计算机辅助软件工程（Computer-Aided Software Engineering，CASE）：为系统开发过程的某些部分提供自动支持的软件工具。（2）

概念模式（Conceptual schema）：数据库整体结构的详细的、技术性的独立性规格说明。（2）

具体类（Concrete class）：有直接实例的类。（14）

并发性控制（Concurrency control）：在多用户环境中，管理对数据库同时发生的多个操作的进程，用以维护数据完整性，并使这些操作彼此互不干扰。（12）

并发透明性（Concurrency transparency）：分布式数据库的一种设计目标，即虽然一个分布式系统运行许多事务，但它仍然表现出仿佛任一给定的事务只是该系统中的惟一的活动。因此，当并发地处理多个事务时，其结果必须与顺序处理每个事务的情况完全相同。（13）

一致性维度（Conformed dimension）：一个或多个维度的表与两个或更多的事实表相关，并且对于每个事实表，这些维表具有相同的业务含意和主键。（11）

约束（Constraint）：数据库用户不能违背的规则。（1）

构造器操作（Constructor operation）：创建一个类的新实例的操作。（14）

内容寻址（Content addressing）：一种工具，允许用户查询数据库，以选择满足限定条件的所有记录以及/或对象。（D）

Cookie：通过Web服务器在客户端存储的数据块。当用户后来又访问该站点时，cookie的内容将送回Web服务器，用来验证用户身份，并且返回定制的Web页面。（10）

相关子查询（Correlated subquery）：在SQL中，内层查询的处理依赖于外层查询数据的

子查询。(8)

对应对象 (Corresponding object)：影响在另一业务规则之上执行动作的能力的业务规则（事实）。(4)

数据 (Data)：在用户环境中有意义的事实、文本、图形、图像、音频和视频段。(1)

数据管理 (Data administration)：负责组织中数据资源整体管理的一种高级功能，包括维护公司范围内的定义与标准。(12)

数据控制语言 (Data Control Language，DCL)：用于控制数据库的命令，包括权限和数据提交（保存）的管理。(7)

数据定义语言 (Data Definition Language，DDL)：用于定义数据库的命令，包括创建、修改和删除表以及建立约束。(7)

数据词典 (Data dictionary)：关于数据库的一个信息库，它记录了数据库中的数据元素。(12)

数据独立性 (Data independence)：数据描述和使用该数据的应用程序分离。(1)

数据操纵语言 (Data Manipulation Language，DML)：用于维护和查询数据库的命令，包括更新、插入、修改和查询数据。(7)

数据集市 (Data mart)：限定在一定范围内的数据仓库，它的数据或者从数据仓库中选择和汇总而来，或者从源数据系统中提取、转换和装载而来。(11)

数据挖掘 (Data mining)：一种混合了多种技术的知识发现，这些技术包括传统的统计学、人工智能和计算机图形学。(11)

数据清洗 (Data scrubbing)：在数据转换和载入数据仓库之前，使用模式识别和其他人工智能技术改善原始数据质量的一种技术。(11)

数据管家 (Data steward)：负责确保组织内的应用软件真正支持该组织企业目标的人员。(12)

数据转换 (Data transformation)：数据调和的组成部分，它把源运作系统的数据格式转换为EDW的数据格式。(11)

数据类型 (Data type)：系统软件（如DBMS）可识别的用于表示组织数据的具体编码模式。(6)

数据可视化 (Data visualization)：为了便于人们分析而用图形和多媒体的格式表示数据。(11)

数据仓库 (Data warehouse)：一种集成的决策支持数据库，其内容从不同的操作数据库中派生而来。(1) 一种面向主题的、集成的、随时间变化的、不可更新的数据集合，用于支持决策过程的管理。(11)

数据库 (Database)：逻辑相关数据的有组织的集合。(1)

数据库管理 (Database administration)：负责物理数据库设计和处理诸如安全性强化、数据库性能、备份与恢复等技术问题的一种技术功能。(12)

数据库应用 (Database application)：一个应用程序（或一组相关程序），它用于代表数据库用户执行一系列的数据库活动（创建、读取、更新和删除）。(1)

数据库变更日志 (Database change log)：被事务修改的记录的前象和后象。(12)

数据库崩溃 (Database destruction)：指数据库本身受损、被毁坏，或是无法读取。(12)

数据库管理系统 (Database Management System，DBMS)：用于创建、维护并提供对用户数据库有控制访问的软件应用。(1)

数据库恢复（Database recovery）：在数据库丢失或者损害后，迅速并准确地复原数据库的一种机制。（12）

数据库安全性（Database security）：保护数据库免受偶然或故意的丢失、破坏与误用。（12）

数据库服务器（Database server）：在客户/服务器环境中负责数据库存储、访问以及处理的计算机。有人也用这个术语来描述两层的客户/服务器环境。（9）

DCS-1000（Carnivore）：FBI开发的设备，可通过跟踪电子邮件头来监控电子邮件的传输量。（10）

死锁（Deadlock）：当两个或更多事务都锁定了共同的资源，且彼此都在等待对方释放该资源而形成的一种僵局。（12）

死锁预防（Deadlock prevention）：指用户应用程序必须在一个事务的开始点便锁定该程序所需要的所有记录，而不是每次仅锁定一个记录。（12）

死锁化解（Deadlock resolution）：指允许死锁出现，但在DBMS里建立某些机制，以便检测出死锁并打破死锁的一种方法。（12）

分散型数据库（Decentralized database）：既没有通过网络和使数据如同处于一个逻辑数据库的数据库软件彼此互连，而是分散地存放在多个地点的计算机内的数据库。（13）

度（Degree）：参与联系的实体类型的个数。（3）

非规范化（Denormalization）：将规范化的关系转化为非规范化的物理记录规格说明的过程。（6）

依赖数据集市（Dependent data mart）：从企业数据仓库和它的调和数据中获取全部数据的数据集市。（11）

导出（Derivation）：从其他业务知识中导出的语句。（4）

导出属性（Derived attribute）：其值可以从其他相关属性值计算出的属性。（3）

导出数据（Derived data）：为终端用户的决策支持应用而选择、格式化和聚合的数据。（11）

导出事实（Derived fact）：使用算法和推理从其他业务规则中导出的事实。（4）

决定因子（Determinant）：函数依赖中箭头左边的属性。（5）

词典（Dictionary）：由键-值对组成的不带重复的无序序列。（15）

不相交规则（Disjoint rule）：指定如果一个实体实例（超类型的实体实例）是一个子类型的成员，则不能同时是另一个子类型的成员。（4）

不相交约束（Disjointness constraint）：该约束解决这样的问题，是否一个超类型的实例同时是两个或多个子类型的成员。（4）

分布式数据库（Distributed database）：物理上虽然存放在多个地点的计算机内，但通过一条数据通信链路连接起来的逻辑上是单一的数据库。（13）

域名服务器平衡（DNS(domain name server)balancing）：一种负载平衡方法，站点主机名的DNS服务器返回站点的多个IP地址。（10）

动态SQL（Dynamic SQL）：使应用程序在执行时才生成明确的SQL代码的方法。（8）

动态视图（Dynamic view）：由于用户的请求而动态创建的一种虚拟表。动态视图不是一张临时表，相反，它的定义存储在系统目录中，视图的内容被物化为使用视图的一条SQL查询的结果。它与物化视图的区别视RDBMS不同而不同，物化视图可以存储在磁盘上，在间隔内或使用时刷新。（7）

电子业务（Electronic business，e-business）：采用因特网相关技术，使顾客与供应商的关系更加紧密的一种电子业务活动。（10）

电子商务（Electronic commerce，e-commerce）：基于因特网的业务事务，包括订单处理和执行、交互客户关系管理、电子数据交换（EDI）和账单支付等活动。(10)

嵌入式SQL（Embedded SQL）：在一个用其他语言（如C或Java）编写的程序中包含硬编码的SQL语句的过程。(8)

封装（Encapsulation）：一种对外隐藏对象的内部实现细节的技术。(14)

加密（Encryption）：对数据进行编码或者搅乱，使别人不能读出这些数据。(12)

增强型实体-联系模型（Enhanced entity-relationship(ERR)model）：使用新的建模概念对原始的E-R模型进行扩充所得到的模型。(4)

企业数据模型（Enterprise data model）：显示组织的高层实体及实体间联系的图形化模型。(1)

企业数据建模（Enterprise data modeling）：数据库开发的第一步，它指定组织数据库的范围和一般内容。(2)

企业数据仓库（Enterprise Data Warehouse，EDW）：一个集中的、集成的数据仓库，它是决策支持应用的终端用户可得到的所有数据的控制点和惟一来源。(11)

企业键（Enterprise key）：值在所有关系里都惟一的主键。(5)

企业资源规划系统（Enterprise resource planning (ERP) system）：一个集成了企业的所有功能（例如，制造、销售、财务、营销、库存、会计和人力资源）的企业管理系统。ERP系统是为企业考察和管理其活动提供必要数据的应用软件。(1)

实体（Entity）：在用户环境中组织所希望维护的数据，如人、地方、对象、事件、概念。

实体聚簇（Entity cluster）：一个或多个实体类型及其相关联系的集合组成的一个抽象的实体类型。(4)

实体实例（Entity instance）：实体类型中的一个特定记录。(3)

实体完整性规则（Entity integrity rule）：主键属性（或组成主键的部分属性）不可以为空。(5)

实体类型（Entity type）：具有共同性质和特征的实体集合。(3)

实体-联系图（Entity-relationship (E-R) diagram）：实体-联系模型的图形表示。(3)

实体-联系模型（Entity-relationship (E-R) model）：组织或业务领域中数据的逻辑表示。(3)

等值联结（Equi-join）：其联结条件为相同列（字段）的值相等。这些相同的列将（冗余地）出现在结果表中。(8)

事件（Event）：由事务产生的数据库动作（创建、更新或删除）。(11)

事件驱动（Event-driven）：非过程程序设计方法，在事件发生时检测该事件并做出相应反应。(9)

排他锁（也称为X锁或写锁，Exclusive lock (X lock or write lock)）：在该锁解开之前防止其他事务读与更新记录的一种技术。(12)

可扩展标记语言（Extensible Markup Language，XML）：一种基于SGML的脚本语言，它允许创建定制标记，从而使跨组织的数据传输和共享更为容易。(10)

区（Extent）：磁盘存储空间的连续扇区。(6)

数据库中类的所有实例的集合，此时译为外延。(15)

外部网（Extranet）：使用因特网协议使公司的顾客和供应商对公司数据和信息进行有限的访问。(1)

事实（Fact）：两个或多个术语之间的关联。(3)

故障透明性（Failure transparency）：指分布式数据库的一个设计目标，它可以保证每个事务的所有操作都被提交或者都没有被提交。(13)

胖客户机（Fat client）：负责处理表示逻辑、广泛的应用和业务规则逻辑以及许多数据库管理系统功能的客户端个人电脑。(9)

字段（Field）：可以被系统软件识别的最小的数据单元。(6)

文件组织（File organization）：在二级存储设备上物理安排文件中记录的技术。(6)

文件服务器（File server）：一种管理文件操作的设备，并由每个连接到局域网的客户端PC所共享。(9)

防火墙（Firewall）：一种限制从外部访问公司数据的硬件/软件安全组件。(10)

第一范式（First normal form）：不含多值属性的关系。(5)

外键（Foreign key）：数据库关系中的一个属性，它在同一个数据库中的另一个关系中是主键。(5)

前向恢复/前滚（Forward recovery (rollforward)）：从当前数据库的前一份拷贝重新开始运行的一种技术。就是把此前已获得正常结果的事务的后象直接应用于数据库，使得数据库能够立即迅速转向该事务处理后的状态。(12)

第四范式（Fourth normal form）：满足BCNF，即不含多值依赖的关系。(B)

函数（Function）：一个已存储的子例程，只有一个返回值和输入参数。(8)

功能分解（Functional decomposition）：将系统描述逐步细分为更详细描述的一个迭代过程，其中，一个功能由支持该功能的更详细的功能来描述。(2)

函数依赖（Functional dependency）：两个属性或两个属性集之间的约束。(5)

概化（Generalization）：从一组更特殊的实体定义更一般实体类型的过程。(4)

全局事务（Global transaction）：分布式数据库中的一种事务，它需要引用一个或者多个非本地站点上的数据，以满足其请求。(13)

粒度（Grain）：事实表的详细级别，由组成主键的所有组件的交集决定，包括所有外键和除此之外的主键元素。(11)

散列索引表（Hash index table）：文件组织的一种形式，使用哈希方法将键映射到索引表中的一个位置，在该位置存放指向对应于哈希键的实际数据记录的指针。(6)

散列文件组织（Hashed file organization）：一种存储系统，其中每个记录的地址是使用一个哈希算法来决定的。(6)

散列算法（Hashing algorithm）：一个将主键值转换到相对记录号（或相对文件地址）的程序。(6)

异义（Homonym）：可能有多种含义的属性。(5)

水平分割（Horizontal partitioning）：将表的行分布到几个单独的文件。(6)

超文本标记语言（Hypertext Markup Language，HTML）：用于在Web浏览器上显示文档的脚本语言，类似于SGML（一种更全面的信息管理标准）。(10)

标识符（Identifier）：可以惟一标识一个实体类型实例的属性或属性集。(3)

标识属主（Identifying owner）：弱实体类型所依赖的实体类型。(3)

标识联系（Identifying relationship）：弱实体类型与其属主之间的联系。(3)

不一致读问题（Inconsistent read problem）：当一个用户读到已被其他用户部分更新的数据时，所出现的不可重复的读取现象。(12)

增量提交（Incremental commitment）：系统开发项目中的一种策略，它在每个阶段后进行检查，在每次检查后重新调整项目的后续部分。(2)

增量抽取（Incremental extract）：一种数据获取方法，只获取自上次数据提取后源数据发生变化的部分。(11)

独立数据集市（Independent data mart）：一种数据集市，它的数据是从运作环境中提取的，并且不具有数据仓库的优点。(11)

索引（Index）：一个表或其他数据结构，用于决定文件中满足某些条件的行的位置。(6)

索引文件组织（Indexed file organization）：记录在文件里顺序或是非顺序存放，使用索引来定位每条记录。(6)

信息（Information）：以增加使用数据者知识的某种方式进行处理的数据。(1)

信息工程（Information engineering）：一种自顶向下的面向数据的形式化方法，用于信息系统的创建和维护。(2)

信息库（Information repository）：一种用来存储元数据的组件。它描述组织的数据与数据处理的资源，管理总体的信息处理环境，并把该组织的业务信息及其应用软件包组合在一起。(12)

信息库词典系统（Information Repository Dictionary System，IRDS）：用于管理和控制对信息库访问的一种计算机软件工具。(12)

信息系统体系结构（Information Systems Architecture，ISA）：表示组织中信息系统的未来结构的概念性蓝图或规划。(2)

信息系统（Informational system）：建立在历史数据和预测数据基础上的系统，用于支持决策制定、复杂查询和数据挖掘应用。(11)

企业内部网（Intranet）：使用因特网协议建立对公司内数据和信息的访问。(1)

非法入侵检测系统（Intrusion detection system，IDS）：一种试图识别非法进入计算机系统或误用计算机系统企图的系统。IDS可以监控网络上传输的包，监控系统文件、日志文件，或设置尝试诱捕黑客。(10)

Java：一种通用的、面向对象的编程语言，特别适用于Web。称为Java applets的小Java程序从Web服务器下载到客户端，并且在与Java兼容的浏览器上运行。(10)

Java servlet：一种不是在操作系统而是在其他应用程序中执行的小程序，并且它是存储在服务器上而不是和应用程序一起存储在客户端。(10)

JavaScript：一种基于Java但比较容易学习的脚本语言，用于获得Web页面的交互性。(10)

联结（Join）：将两张具有公共域的表合并成为一张表或视图的关系操作。(8)

联结索引（Join index）：创建在来自于多个表的列上的一个索引，这些列有相同的值域。(6)

联结操作（Joining）：把不同来源的数据合并为一张表或视图的过程。(11)

日志记录工具（Journalizing facility）：事务和数据库变化的一种审计追踪工具。(12)

遗留数据（Legacy data）：新系统安装前使用的系统所包含的数据。遗留数据常常驻留在主机系统中，该系统已经被客户机/服务器系统或Web系统所替代。(1)

列表（List）：同类型元素的有序集合。(15)

本地自治（Local autonomy）：分布式数据库的一种设计目标，也就是说，站点在连接其

他结点失败时可以独立管理和操作其数据库。(13)

本地事务（Local transaction）：分布式数据库中的一种事务，它仅要求引用存放在发生该事务的站点的数据。(13)

位置透明性（Location transparency）：分布式数据库的一个设计目标，也就是说，使用数据的用户（或用户程序）无须知道该数据所在的位置。(13)

加锁（Locking）：对用户为了更新而检索出来的任何数据必须加以锁定，以便拒绝其他用户使用，一直到更新完成或者中止为止。(12)

加锁层次/粒度（Locking level (granularity)）：每次锁定所包括的数据库资源的范围。(12)

逻辑数据集市（Logical data mart）：由数据仓库的关系视图所创建的数据集市。(11)

购物篮分析（Market basket analysis）：对某个顾客的购买行为进行研究。(11)

大规模并行处理/无共享体系结构（Massively parallel processing (MPP)/shared nothing architecture）：每个CPU拥有自己专用的存储器的大规模并行处理系统。(9)

物化视图（Materialized view）：基于SQL查询，以与动态视图相同的方式创建的数据拷贝或数据副本。然而，物化视图以表的形式存在，因此必须注意使它与相关的基表保持同步。(7)

最大基数（Maximum cardinality）：与某实体的每一个实例相关联的另一个实体实例的最大个数。(3)

元数据（Metadata）：描述其他数据的性质或特征的数据。(1)

方法（Method）：操作的实现。(14)

中间件（Middleware）：允许某个应用程序与其他软件实现互操作而不要求用户了解和编码实现这种互操作性的低层操作的软件。(9)

最小基数（Minimum cardinality）：与某实体的每一个实例相关联的另一个实体实例的最小个数。(3)

多维OLAP（Multidimensional OLAP，MOLAP），一种OLAP工具，把数据装入一个中间结构，通常是三维或更高维的数组。(11)

多重分类（Multiple classification）：一个对象是多于一个类的实例。(14)

多重性（Multiplicity）：说明在一个给定的联系中有多少个对象参与的规格说明。(14)

多值属性（Multivalued attribute）：在给定实体实例中可以带有多个值的属性。(3)

多值依赖（Multivalued dependency）：一种存在于至少含有3个属性（如A、B和C）的关系里的函数依赖。对于每个A的值，B和C都有定义明确的值集；而B和C的值彼此无关。(B)

自然联结（Natural join）：与等值连结相同，只是在结果表中不包括重复列。(8)

范式（Normal form）：通过对关系应用某些关于函数依赖（或是属性间的关联）的简单准则后所得到的关系的状态。(5)

规范化（Normalization）：将含有异常的关系分解为更小的，结构良好的关系的过程。(5)

空值（Null）：当没有其他值适用，或是适用值未知时，可以赋给属性的值。(5)

对象（Object）：对象是一个实体，在应用域内扮演着定义明确的角色，有状态、行为和标识。(14)

对象类（Object class）：共享公共结构和公共行为的对象集。(14)

对象图（Object diagram）：与给定类图兼容的实例图。(14)

对象-关系数据库管理系统（Object-relational database management system）：以集成

的方式支持关系和面向对象特性的数据库引擎。(C)

联机分析处理 (On-Line Analytical Processing, OLAP): 使用一组图形工具向用户提供数据的多维视图，并允许用户利用简单的窗口技术对数据进行分析。(11)

开放数据库互连标准 (Open database connectivity (ODBC) standard): 一个为应用程序提供一种公用语言以访问和处理SQL数据库而与所访问的特定关系数据库管理系统无关的应用程序接口。(9)

操作 (Operation): 由类的所有实例提供的函数或服务。(14)

运作数据存储 (Operational Data Store, ODS): 一种集成的、面向主题的、可更新的、具有当前值的详细数据库，它可以向作决策支持处理的终端用户提供服务。(11)

运作系统 (Operational system): 基于当前数据实时执行某种业务的系统，也称为记录系统。(11)

外连结 (Outer join): 若某一行在其他表的相同列中没有匹配的值，该行仍然包含在结果表中。(8)

交叠规则 (Overlap rule): 指定一个实体实例可以同时是两个（或多个）子类型的成员。(4)

重载 (Overriding): 用子类中更特殊的方法实现来替换从超类中继承的方法的过程。(14)

页 (Page): 操作系统在二级存储（磁盘）一次输入或输出操作中所读写的数据量。对于具有磁类型的I/O来说，其等价的术语是记录块。(6)

部分函数依赖 (Partial functional dependency): 非键属性函数依赖于部分（不是全部）主键的一种函数依赖。(5)

部分特化规则 (Partial specialization rule): 指定一个超类型的实例可以不属于任何子类型。(4)

周期数据 (Periodic data): 一旦添加到存储器就不会被物理更改或删除的数据。(11)

持久化存储模块 (Persistent Stored Module, SQL/PSM): 在SQL-99中定义的SQL扩展，包括通过用户会话创建和撤销代码模块的功能，这些代码模块存储在数据库模式中。(8)

物理文件 (Physical file): 用来存储物理记录的二级存储（如磁带和硬盘）的已命名的一部分。(6)

物理记录 (Physical record): 一组在内存里邻接存储的字段，可以由DBMS作为一个单元进行读写。(6)

物理模式 (Physical schema): 关于概念模式的数据如何存储在计算机二级存储器中的规格说明。(2)

插件 (Plug-ins): 通过增加某些特性（如加密、动画或无线访问）来扩展浏览器功能的软件或硬件模块。(10)

指针 (Pointer): 一个数据字段，它用来定位一个相关的数据字段或记录。(6)

多态 (Polymorphism): 同样的操作可以以不同方式使用到两个或多个类上的现象。(14)

主键 (Primary key): 惟一标识关系中每一行的的属性（或属性集）。(5)

过程 (Procedure): 一个过程性的SQL语句集合，在数据库模式中被分配了一个惟一的名字，并存储在数据库中。(8)

项目 (Project): 为了达到一个目标而进行的具有开始和结束的相关活动的计划。(2)

原型法 (Prototyping): 系统开发的迭代过程，在分析员和用户的紧密配合下，通过不断的修正将需求最终转换成一个工作系统。(2)

代理服务器（Proxy server）：管理来往于局域网的因特网通信量的防火墙组件。它还能处理访问控制和文档高速缓存。(10)

查询操作（Query operation）：能访问对象的状态、但不能改变其状态的操作。(14)

按例查询（Query-by-Example）：一种使用图形化方法构建查询的直接操纵数据库语言。(9)

调和数据（Reconciled data）：详细的当前数据，并且是所有决策支持应用系统的一个权威的来源。(11)

恢复管理器（Recovery manager）：DBMS的一个模块，在出现故障时它将数据库还原成正确的条件，并继续处理用户的请求。(12)

递归外键（Recursive foreign key）：引用自己所在关系主键值的外键。(5)

廉价冗余磁盘阵列（Redundant Array of Inexpensive Disk，RAID）：一组物理磁盘驱动器，对于数据库用户（以及程序）而言，它们就好像是一个大的逻辑存储单元。(6)

参照完整性（Referential integrity）：完整性约束的一种，它指定关系中属性的值（或存在）依赖于同一个关系或其他关系中主键的值（或存在）。(7)

参照完整性约束（Referential integrity constraint）：一个关系中的外键值，或者与另一个关系的主键值相匹配，或者必须为空。(5)

刷新模式（Refresh mode）：一种定期重写目标数据的填充数据仓库的方法。(11)

关系（Relation）：一个命名的两维数据表。(5)

关系数据库管理系统（Relational DBMS，RDBMS）：一个数据库管理系统，它把数据作为表的集合来管理；其中所有的数据联系用相关表中的公共数值来表达。(7)

关系OLAP（Relational OLAP，ROLAP）：一种OLAP工具，把数据库看作是传统的关系数据库（可以是星型模式或其他规范化/非规范化的表集合）。(11)

联系实例（Relationship instance）：两个（或多个）实体实例之间的关联，其中每个联系实例恰好包括参与联系的各实体类型中的一个实例。(3)

联系类型（Relationship type）：两个（或多个）实体类型之间有意义的关联。(3)

复制透明性（Replication transparency）：指分布式数据库的一种设计目标，也就是说，虽然一个给定的数据项可以在网络的多个结点上复制，但程序员或者用户仍然可以把该数据项当成是在单一结点上的单一数据项那样进行处理。也可以称之为段落透明性。(13)

信息库（Repository）：所有数据定义、数据联系、屏幕和报表格式及其他系统组成部分的中央知识库。(1,2)

复原/再运行（Restore/rerun）：利用数据库的备用拷贝，重新处理当天事务（直至故障点）的一种技术。(12)

逆向代理（Reverse proxy）：一种负载平衡方法，截取来自客户端的请求并且在Web服务器中高速缓存返回客户端的响应。(10)

路由器（Router）：通信网络的中间设备，用来传输信息包，并以最有效的路径将信息包发送到正确的目的地。(10)

标量聚合（Scalar aggregate）：一个包含聚合函数的SQL查询返回单个的值。(7)

模式（Schema）：数据库的一部分，包含用户创建的对象（如基表、视图和约束）的描述的结构。(7)

范围操作（Scope operation）：应用于类而不是应用于对象实例的一种操作。(14)

第二范式（Second normal form）：一个关系满足第一范式，且每一个非键属性都完全函

数依赖于主键。(5)

辅键（Secondary key）：字段或字段的组合，多个记录可以在这些字段上有相同的值。也称为非惟一键。(6)

选择（Selection）：根据预先定义的准则划分数据的过程。(11)

半联结（Semijoin）：分布式数据库所采用的一种联结操作，其中仅有联结属性才从一个站点传输到另一个站点，而不是从每个符合条件的行选出的全部属性。(13)

顺序文件组织（Sequential file organization）：文件中的记录按照主键值顺序存储。(6)

服务器端扩展（Server-side extension）：一种软件程序，直接与Web服务器交互，以处理各种请求。(10)

集（Set）：没有重复元素的无序集合。(15)

共享锁/S锁/读锁（Shared lock (S lock or read lock)）：允许别的事务读但不能更新记录或其他资源的一种技术。(12)

简单属性（Simple attribute）：不能分解为更小组件的属性。(3)

雪花模式（Snowflake schema）：星型模式的扩展版本，在这种模式里，维表规范化为若干相关表。(11)

软硬件负载平衡（Software and hardware load balancing）：一种负载平衡方法，对一个IP地址的请求在TCP/IP路由层被分配到驻留在站点的多台服务器上。(10)

特化（Specialization）：定义超类型的一个或多个子类型并形成超类型/子类型联系的过程。(4)

标准通用标记语言（Standard Generalized Markup Language，SGML）：1986年由国际标准化组织采纳的脚本文档的信息管理标准，其中定义了跨平台和跨应用的脚本文档格式化、索引和链接信息。(4)

星型模式（Star schema）：一种简单的数据库设计，在这种模式中，维数据与事实或事件数据相分离。多维模型是星型模式的另一个名字。(11)

状态（State）：包含对象的性质（属性和联系）和这些性质具有的值。(14)

静态抽取（Static extract）：一种获取所需要的源数据在某个时间点的快照的方法。(11)

存储过程（Stored procedure）：一个代码模块，通常使用诸如Oracle的PL/SQL或者Sybase的Transact-SQL这样的专用语言编写，能够实现应用逻辑或某项业务规则，并存储在服务器上，在调用时运行。(9)

条（Stripe）：RAID中所有磁盘上的页集，它们与磁盘开始位置相对距离相等。(6)

强实体类型（Strong entity type）：不依赖于其他实体类型而独立存在的实体类型。(3)

结构断言（Structural assertion）：表示组织静态结构某些方面的语句。(4)

结构化文字（Structured literal）：由固定数目的已命名元素组成，每一个元素可以是文字或对象类型。(15)

子类型（Subtype）：对组织有意义的实体类型中实体的分组，它们共享与其他分组不同的公共的属性和关系。(4)

子类型鉴别符（Subtype discriminator）：超类型的一个属性，其值决定了目标子类型或子类型。(4)

超类型（Supertype）：与一个或多个子类型有联系的通用实体类型。(4)

超类型/子类型层次（Supertype/subtype hierachy）：超类型和子类型的层次结构安排。其中，每一个子类型只有一个超类型。(4)

对称多处理（Symmetric Multiprocessing，SMP）：一种多个处理器共享一块内存的并行处理体系结构。(9)

同步分布式数据库（Synchronous distributed database）：分布式数据库技术的一种形式。其中，网络上的所有数据一直保持最新的状态，使得任何站点上的用户随时都能够访问网络上任何地方的数据，并能得出相同的结果。(13)

同义词（Synonyms）：两个（或多个）属性名字不同，但是意义却相同，因为它们描述的是一个实体的相同特征。(5)

系统目录（System catalog）：一种系统所创建的数据库，用于描述全部数据库对象，包括数据词典信息，也包括用户访问信息。(12)

系统开发生命周期（Systems Development Life Cycle，SDLC）：用于开发、维护和替换信息系统的传统方法学。(2)

表空间（Tablespace）：命名的磁盘存储空间，用来存储数据库表的物理文件。(6)

术语（Term）：在业务中有特定含义的词或短语。(3)

三元联系（Ternary relationship）：同时存在于三个实体类型的实例之间的联系(3)

瘦客户机（Thin client）：一台用来处理用户界面和若干应用的PC，通常没有或只有很小的本地数据存储器。(9)

第三范式（Third normal form）：满足第二范式，且不含传递依赖关系。(5)

三层体系结构（Three-tier architecture）：一种包含三层的客户/服务器配置，即包含一个客户机层和两个服务器层。尽管两个服务器层的本质有所不同，但它们共同的配置是都拥有一个应用程序服务器。(9)

时间戳（Time stamp）：与数据值相联系的时间值。(3)

盖时间戳（Timestamping）：分布式数据库中的一种并发性控制机制，它为每个事务分配一个全局惟一的时间戳。盖时间戳是在分布式数据库中使用锁的另一种方式。(13)

自顶向下规划（Top-down planning）：一种试图获取对整个组织信息系统需求的广泛理解的通用的信息系统规划方法学。(2)

完全特化规则（Total specialization rule）：指定超类型的每一个实体实例必须是联系中一些子类型的一个成员。(4)

事务（Transaction）：必须完整地加以处理或者不限于在单一计算机系统内完成的离散工作(作业)单元。输入一张顾客订单就是事务的一个例子。(12)

事务边界（Transaction boundary）：即事务的逻辑起点与终点。(12)

事务日志（Transaction log）：在数据库中处理的每一项事务的基本数据之记录。(12)

事务管理器（Transaction manager）：分布式数据库中的一种软件模块，用来维护所有事务以及合适的并发性控制模式的日志。(13)

临时数据（Transient data）：对现存数据的修改直接在前面数据的基础上进行，因此前面的数据内容遭到了破坏。(11)

传递依赖（Transitive dependency）：两个（或多个）非键属性间的函数依赖。(5)

触发器（Trigger）：一个命名的SQL语句集合，当数据修改（INSERT、UPDATE和DELETE）发生时考虑它（被触发）。如果触发器中的条件满足，则执行预先设定的动作。(8)

两阶段提交（Two-phase commit）：用于协调分布式数据库中的更新操作的一种算法。(13)

两阶段加锁协议（Two-phase locking protocol）：为了得到某个事务所必需的锁的一种算

法过程，该事务要求在释放任何锁之前，先得到所有其必需之锁；即在得到锁时归入一个增长阶段，而在它们释放时形成一个收缩阶段。(12)

一元关联（Unary relationship）：同一实体类型的不同实例之间的联系。(3)

更新模式（Update mode）：一种只向数据仓库写入源数据变化的方法。(11)

更新操作（Update operation）：改变对象状态的操作。(14)

用户视图（User view）：用户实现一些任务所需要的数据库一些部分的逻辑描述。(1)

用户自定义数据类型（User-defined Datatype，UDT）：SQL-99允许用户建立标准类型的子类或创建一个具有对象行为的类型来定义自己的数据类型。UDT也可以定义函数和方法。(8)

用户自定义过程（User-defined procedure）：一种允许系统设计者除了授权规则之外定义自己的安全性过程的用户出口（或界面）。(12)

VBScript：一种基于Microsoft Visual Basic的脚本语言，与JavaScript类似。(10)

向量聚合（Vector aggregate）：一个包含聚合函数的SQL查询返回的多个值。(7)

版本设置（Versioning）：每个事务限制于一个该事务开始时刻的数据库视图，而每当事务修改了某个记录时，DBMS便会创建一个新的记录版本而不是覆盖老的记录。这样，也就无须任何形式的加锁了。(12)

垂直分割（Vertical partitioning）：将表的列分布到多个独立的物理记录。(6)

Visual Basic应用程序（Visual Basic for Application，VBA）：伴随Access 2000的编程语言。(9)

弱实体类型（Weak entity type）：依赖于其他实体类型而存在的实体类型。(3)

良构关系（Well-structured relation）：含有最小冗余的关系，用户可以插入、修改和删除表中行，而且不会出现错误和不一致的情况。(5)

万维网（World Wide Web，WWW）：驻留在特定服务器（称为Web服务器或HTTP服务器）上并且内部相互链接的超文本文档的全体集合。配置Web服务器，使每个人能够很容易地访问它们所拥有的信息，同时允许文件被访问、传输和下载。也称为W3或者Web。(10)

万维网联盟（World Wide Web Consortium，W3C）：一个国际性的公司联盟，致力于开发开放式标准，这些标准有利于开发Web规范，使Web文档能够跨平台地一致显示。(10)

XHTML：一种混合脚本语言，它扩展HTML代码使之与XML兼容。(10)