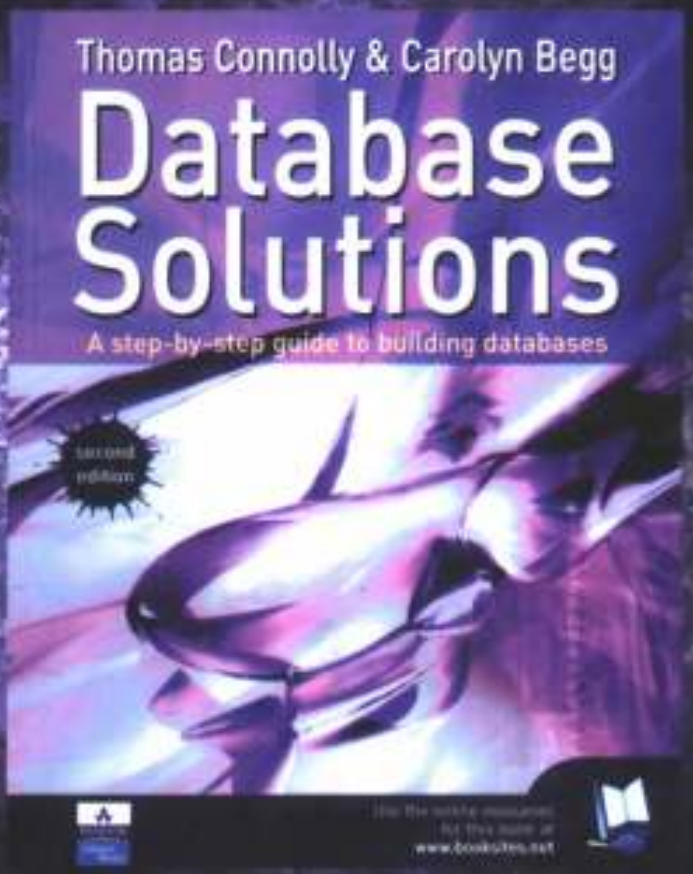
计 算 机 科 学 丛 书

第2版

数据库设计教程

Thomas M. Connolly　Carolyn E. Begg 著　　何玉洁　黄婷儿　等译



Database Solutions

A Step-by-Step Guide to Building Databases, Second Edition

机械工业出版社
China Machine Press

数据库设计是数据库技术的一个重要方面。目前有很多书籍介绍数据库的理论知识，但能与实际应用紧密结合的实用性书籍却相对较少。本书作者具有丰富的数据库设计和教学经验，以通俗易懂的语言描述了分析、设计、实现数据库的整个设计过程，因此本书是一本真正意义上的数据库理论与实践相结合的书籍，可用于数据库基础理论之后的后续学习。

本书主题

- 数据库的基本知识和相关概念
- 全面介绍数据库设计方法学
- 用实例引导读者掌握从需求分析、逻辑建模到物理实现的全过程
- 包含常用应用领域的数据模型，读者可以在此基础上建立自己的数据模型
- 用UML表示法创建数据模型的方法

第2版的新增内容

- 添加了关于SQL和QBE、数据库管理和安全以及数据库领域的现状和趋势的章节，使本书的内容更加全面和先进。
- 对数据库设计方法论的内容进行了更新。
- 本书的配套教学材料包含StayHome数据库的实现、示例数据模型的SQL脚本、本书练习的样例解决方案等，读者可以到华章网站下载。

作者简介

Thomas M. Connolly 是英国佩斯里（Paisley）大学计算技术与信息系统学部主任，该校拥有英国规模最大的IT学科。在产业界任职期间，他设计了世界上第一个商用可移植关系DBMS——RAPPORT，他因设计和开发配置管理工具LIFESPAN而获得英国软件杰出设计大奖（British Design Award for Software Excellence）。

Carolyn E. Begg 毕业于爱丁堡大学，佩斯里大学讲师，研究兴趣包括信息系统、生物学领域数据库系统应用程序等。

www.PearsonEd.com

ISBN 7-111-15471-1

9 787111 154716

封面设计：陈子祥

华章图书

华章网站：http://www.hzbook.com

网上购书：www.china-pub.com

北京市西城区百万庄南街1号 100037

读者服务热线：(010)68995259，68995264

读者服务信箱：hzedu@hzbook.com

ISBN 7-111-15471-1/TP · 4029（课）

定价：35.00元

TP311.13
K263.02

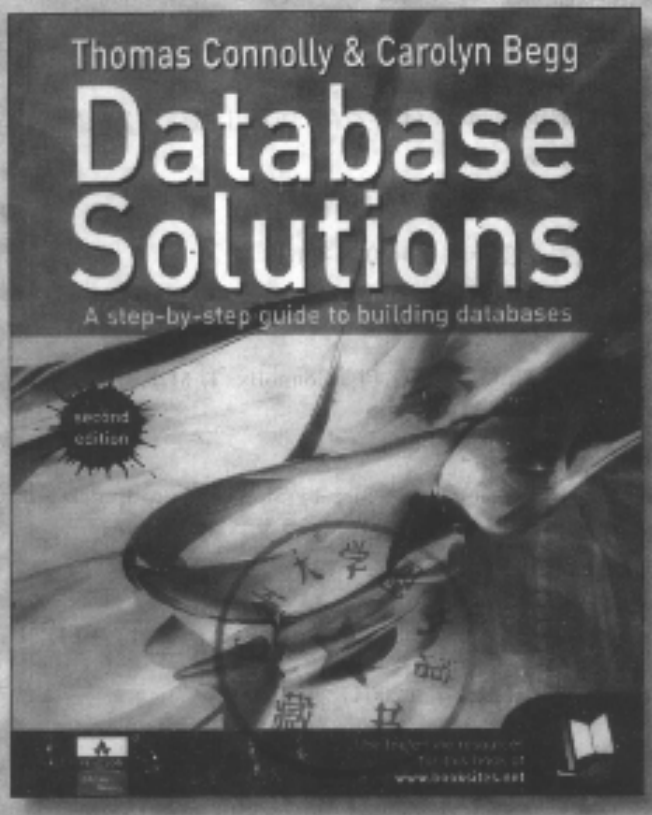
计 算 机 科 学 丛 书

第2版

郑州大学 *04010301511G*

数据库设计教程

Thomas M. Connolly Carolyn E. Begg 著 何玉洁 黄婷儿 等译



Database Solutions

A Step-by-Step Guide to Building Databases, Second Edition

机械工业出版社
China Machine Press

本书详细介绍数据库的分析、设计、实现方法和过程。本书不仅涵盖普通数据库书籍中所没有讲到的数据库分析、设计和执行过程，还通过贯穿全书的实例描述数据库的创建、监控和调整的全过程。主要内容包括数据库的相关知识，数据库分析和设计技术，逻辑数据库设计方法学、物理数据库设计方法学等内容。另外，本书还增加了SQL和QBE、数据库管理和安全性以及数据库的现状和趋势等内容，使内容更为全面。本书既适合作为高等院校数据库课程的教材，也适合相关技术人员作为数据库分析、设计和开发的参考。

Simplified Chinese edition copyright © 2005 by Pearson Education Asia Limited and China Machine Press.

Original English language title: *Database Solutions: A Step-by-Step Guide to Building Databases, Second Edition*(ISBN: 0-321-17350-3) by Thomas M.Connolly and Carolyn E. Begg , copyright © 2004.

All rights reserved.

Published by arrangement with the original publisher, Pearson Education, Inc., publishing as Addison-Wesley.

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。

本书法律顾问　北京市展达律师事务所

本书版权登记号：图字：01-2004-3136

图书在版编目（CIP）数据

数据库设计教程（第2版）/ 康诺利（Connolly, T. M.）等著；何玉洁等译. —北京：机械工业出版社，2005.1

（计算机科学丛书）

书名原文：Database Solutions: A Step-by-Step Guide to Building Databases, Second Edition

ISBN 7-111-15471-1

Ⅰ.数…　Ⅱ.①康…　②何…　Ⅲ.数据库-程序设计-教材　Ⅳ.TP311.13

中国版本图书馆CIP数据核字（2004）第108591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

2005年1月第2版第1次印刷

787mm × 1092mm 1/16 · 21.5印张

印数：0 001 -5 000册

定价：35.00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

出版者的话

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

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

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

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

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

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业

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

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

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

专家指导委员会

（按姓氏笔画顺序）

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

秘　书　组

武卫东　　温莉芳　　刘　江　　杨海玲

译 者 序

随着计算机技术的飞速发展和信息管理领域的日益扩大，数据库技术得到了空前的发展，数据库技术的应用领域也日趋扩大和深入。数据库是用科学的方法管理和组织数据的技术，是计算机科学的一个重要分支。

本书主要讲述数据库设计方法学。全书由六个部分和五个附录组成。

第一部分由第1章~第5章组成，分别介绍关系数据模型、使用SQL和QBE查询数据、数据库系统开发生命周期以及数据库的管理和安全。

第二部分由第6章~第8章组成，介绍设计数据库首先要做的事情：事实发现，其中包括事实发现所使用的技术和方法，整理所发现的事实并建立数据库模型的方法以及如何在数据模型的基础上设计规范化的关系数据库表的方法。

第三部分由第9章~第11章组成，第9章全面概述本书使用的数据库设计方法学技术，并着重介绍方法学中步骤1中的任务——构建ER模型的方法，第10章介绍方法学中步骤2的任务，即如何将ER模型映射为关系数据库的表，第11章介绍高级ER建模技术。

第四部分由第12章~第16章组成，其中第12章介绍方法学中的步骤3，也就是物理数据库设计的第1个步骤，在这个步骤中将逻辑数据库设计转换为目标DBMS支持的物理数据库设计。第13章介绍方法学中的步骤4，也就是物理数据库设计的第2个步骤，在该章中介绍如何分析和实现用户的事务以及如何选择合适的数据库文件的组织方式和索引以提高系统的性能。第14章介绍物理数据库设计的后两个步骤——步骤5和步骤6，内容包括如何设计用户视图以及如何设计数据库安全性机制。

第五部分由第17章和第18章组成，该部分通过第二个示例——宠物诊所的例子，再次演示了数据库方法学中逻辑数据库设计和物理数据库设计的各个步骤的使用。

第六部分由第19章组成，该章介绍一些高级数据库应用，讨论关系模型的局限性，介绍分布式数据库管理系统。面向对象数据库管理系统、对象-关系数据库管理系统、数据仓库、联机分析处理和数据挖掘的主要概念，最后提出将数据库技术与Web技术集成起来的一些方法。

附录A介绍其他一些较常用的表示数据模型的方法。附录B对本书所介绍的数据库设计方法学中的步骤进行归纳和总结，便于读者快速参考和查阅。附录C介绍设计复杂数据库时可以先设计局部逻辑数据模型，然后再将这些局部模型合并为一个全局模型的方法。附录D介绍一些常用的数据库文件和索引的组织方式。附录E介绍15个常用应用领域的数据模型的建立，方便读者在解决类似问题时进行参考。

本书的作者从事过多年的数据库方面的工作，具有丰富的数据库设计和使用经验，因此本书的特点就是数据库技术与实际的应用的结合非常密切，弥补了现在很多数据库图书中理论知识介绍很丰富，但结合实际情况解决实际问题的介绍却不够丰富的缺点。由于本书的作者具有很丰富的数据库设计、开发和使用的经验，因此书中介绍的内容非常有效和实用，非常适合数据库设计人员使用。

本书的另一个特点就是将抽象、枯燥的数据库设计方法用一个很容易理解的例子贯穿起来，通过这个例子逐渐展开，直至使用完数据库设计方法学的全部步骤，因此容易激发读者

的阅读兴趣，也能使读者更好地理解这些方法学的运用。在读者已经基本熟悉了这些方法学的过程之后，再举出第二个例子，第二个例子又一次完全用到了前边介绍的方法学步骤，使读者进一步加深对方法学的理解。此外，本书图文并茂，读者能很直观地理解数据库设计的概念和方法。

本书的第2版在第1版成功的基础之上，又添加了SQL和QBE、数据库管理和安全性以及数据库的现状和未来趋势方面的内容，并将第1版的第10章放在了第2版的附录C中，使全书的结构更加合理、条理更清晰。同时，对第1版大部分章节的内容进行了调整，增加了一些新的知识，去掉了第1版中的一些过时的内容。尤其是在第2版中增加的第5章更强调了数据库安全的重要性，第19章客观地分析了当前数据库技术发展的几个方向。有利于读者对数据库技术有更全面的了解。

本书适合有一定数据库理论基础，又希望能将理论知识应用到实际使用过程中的读者阅读，尤其适合那些在设计数据库时不知从何下手或者经验不足的数据库设计新手阅读。本书可作为数据库理论知识的后续学习内容，也可作为高等院校数据库课程的教材或数据库设计和开发人员的数据库设计参考书。

本书由何玉洁负责组织翻译和审校，主要由何玉洁和黄婷儿完成，梁琦、崔晗、卢立能、何玉书、李迎等参加了本书部分章节的审校工作。

由于译者水平有限，在译校过程中难免出现疏漏和不妥之处，恳请读者批评指正。

何玉洁

2004年9月

前　言

背景

如今，数据库是信息系统的基本框架并且从根本上改变了许多公司和个人的工作方式。数据库技术经过多年的发展已经创造出更强大、更易用的数据库系统，使用户无需具备开发高效系统的知识便可创建数据库并实现其应用。目前的参考文献中，有许多优秀的、阐述数据库的部分发展生命周期的著作。然而，我们发现很少有书籍介绍数据库的分析、设计以及执行，并且很少有书籍采用专业和非专业读者都易懂的方式描述数据库的开发过程。

因此，本书的创作初衷是尽可能清楚地为读者解释如何分析、设计及实现数据库，包括从仅含几张表的简单数据库到包括数十至数百张表的大型数据库。实际上本书对进行理论研究的人士也非常有用，而且简单明了地阐述了数据库设计的方法学。

本书所介绍的用于关系数据库管理系统（当今商业应用中的主要系统）的方法学已经通过业界和学术界多年的检验。这种方法学分为两个阶段：

- 逻辑数据库设计阶段。在这个阶段，我们开发一个模型来描述所需要的内容，并忽略实现的细节。
- 物理数据库设计阶段。在这个阶段，我们确定如何在目标数据库管理系统（DBMS）实现这些需求，如Microsoft Access、Microsoft SQL Server、Oracle、DB2和Informix。

我们为每个阶段都提供了易学的步骤，缺乏经验的设计者可以按照所提供的步骤进行学习。此外，一些指导方针也可以用于完成这个学习过程。对于经验丰富的设计者，这种方法学更多地成为了一种框架或者检查列表，而其说明性则极少。

帮助理解数据库的设计

为了帮助读者应用方法学、理解重要的问题，我们提供了一个综合的工作示例，该示例是基于一家称为StayHome的录像出租公司，这个示例将贯穿本书始终。为更完善地补充方法学，我们在第17章和第18章中选取了另外一个名为Perfect Pets的宠物诊所作为示例。

为了帮助读者理解得更加深入，我们在附录E增加了数据库的解决方案（相应的SQL脚本包含在相应的Web站点上）。每个方案都有一个简短的介绍，读者可以参照这个介绍，在阅读样例之前试着自己进行数据库的设计。

常见的数据模型

在提供数据库设计经验的同时，附录E还提供了许多常见的且有用的数据模型。实际上，据统计约有1/3的数据模型包含了适用于大多数公司的公共结构，而其余的2/3则用于特殊产业或者特殊单位。因此，大多数数据库设计工作包含了重建那些被其他公司创建了多次的结构。这些具有特征性的模型也许并不能完全地满足你的公司的需求，但是，它们可以作为开发的起点，在此基础上，你可以建立更适合本公司特殊需要的模型。我们所提供的模型覆盖了以下的常见业务领域：

- 消费者订购。

- 库存控制。
- 资产管理。
- 项目管理。
- 课程管理。
- 人力资源管理。
- 工资管理。

统一建模语言

越来越多的公司正在采用标准化的方法。使用标准化的方法，他们可以选择一种特定的方法进行数据建模，并将这种方法贯穿于数据库开发过程的始终。本书采用一种常用于逻辑数据库设计的高级数据模型，这种模型是建立在实体－关系（ER）概念上的模型。目前，ER模型没有标准的表示法。大多数书籍在提及关系数据库管理系统（DBMS）的数据库设计时会用到以下两种传统的表示法之一:

- Chen氏表示法，它由代表实体的矩形和代表关系的菱形以及连接矩形和菱形的连线组成。
- Crow的Feet（标注）表示法，它由代表实体的矩形以及代表实体间关系的连线组成。连线一端的标注代表一对多的关系。

当今的CASE工具能很好地支持这两种表示法。但是，它们应用起来比较麻烦而且难以解释。在本书中，我们采用最新的面向对象建模语言中的类图表示法，这种建模语言称为UML（Unified Modeling Language，统一建模语言）。UML表示法结合了三种主要的面向对象设计的要素: Rumbaugh的OMT建模，Booch的面向对象分析和设计以及Jacobson的对象论。可以预计，UML将会成为一种标准，而且对象管理组（OMG）已经采用了UML作为对象方法的标准表示法。

我们相信这种表示法更易于理解和使用。

第2版新增内容

本书第2版对第1版进行了修订，增加了可读性，更新或扩展了现有的一些内容，并引入了一些新的内容。第2版中主要的变化包括:

- 新增一章介绍SQL（Structured Query Language，结构化查询语言）和QBE（Query-by-Example）。SQL和QBE是关系DBMS中最主要的两种语言。
- 新增一章介绍数据库管理和安全性。
- 改进了数据库涉及方法学。特别的，将用户视图从逻辑数据库设计中移到了附录中，以保持基本的方法学的简单性。
- 新增一章介绍数据库发展的趋势，讨论了高级数据库应用的需求以及为什么目前的关系数据库系统不能很好地适应这些需求，然后介绍了分布式数据库管理系统、数据复制、面向对象数据库管理系统、对象－关系数据库管理系统、数据仓库、联机分析处理（OLAP）和数据挖掘的概念，以及将数据库集成到Web环境中的方法。
- 在大多数章的最后通过提出一些问题对本章内容进行回顾，并在本书网站中提供一些回顾问题、练习、测试问题、幻灯片、数据库和附录E中常见数据模型的SQL脚本。

说明怎样执行一个设计

我们认为说明怎样将数据库设计转换为物理实现是非常重要的。在本书中，我们说明了如何用Microsoft Access 2002 DBMS实现第一个示例（名为StayHome的录像租借公司），第二个示例（名为Perfect Pets的宠物诊所）说明了如何在Oracle 9i DBMS中实现数据库设计。

本书的读者对象

哪些人应该阅读本书？我们试图以独立的方式编写这本书，但物理数据库设计是个例外，因为在这一部分，你应该很好地理解目标数据库管理系统是怎样运转的。我们的读者就是那些希望开发数据库的人，包括如下但并非仅仅如下所示的一些人：

- 信息模型设计者和数据库设计者。
- 数据库应用设计者和实现者。
- 数据库制造商。
- 数据和数据库管理者。
- 信息系统，商业IT以及专门从事数据库设计的学者。
- 数据库专业的学生，包括大学生和研究生。
- 希望设计并开发数据库应用的人。

本书的结构

本书共有六个部分和五个附录：

- 第一部分——相关知识介绍。在第1章和第2章，我们介绍了数据库管理系统和关系模型。第3章提供了数据库应用生命周期的概述。在第4章提供了数据库系统开发生命周期的概述，在第5章讨论了数据库管理和安全性。
- 第二部分——数据库分析与设计技术。我们在第6章讨论了数据库分析技术，说明了怎样用其中的一些技术来分析录像租借公司StayHome的需求。在第7章说明了怎样用UML画实体－关系（ER）图。怎样应用规范化规则的内容将在第8章中阐述。ER模型和规范化都是第三部分描述的数据库设计方法学中非常重要的技术。
- 第三部分——逻辑数据库设计。我们用循序渐进的方式描述了逻辑数据库设计的方法。第1步在第9章描述，在这一步，我们为StayHome录像租借公司创建一个ER模型。第2步在第10章介绍，在这一步，我们将ER模型映射为数据库的表集。为支持更复杂的数据库设计，我们在第11章提出了与增强ER建模有关的主要概念，这一章同时描述了如何将这些概念映射到表中。
- 第四部分——物理数据库设计。我们依然用循序渐进的方式描述物理数据库设计的方法。第3步在第12章介绍，这里我们为目标DBMS设计一些基本表。第4步在第13章介绍，这里我们选择文件的组织和索引。在第14章介绍第5步和第6步，我们考虑用户视图的设计以及安全性机制的设计，以防止数据被未授权访问。在第15章介绍步骤7，我们描述了如何引入受控的数据冗余以获得更好的数据库性能。最后，在第16章介绍第8步，我们监视并调整操作系统。正如我们刚刚提到的，我们将告诉你如何在Microsoft Access 2002中完成StayHome数据库系统的设计。
- 第五部分——第二个实例。第二个宠物诊所示例贯穿于第17章和第18章。我们将说明怎

样在Oracle 9i中实现Perfect Pets 数据库的设计。

- 第六部分——数据库的现状和未来趋势。在第19章，我们讨论了高级数据库应用的需求，以及为什么目前的关系数据库系统不能很好地适应这些需求，然后我们介绍了分布式数据库管理系统（DDBMS）、数据复制、面向对象数据库管理系统（OODBMS）、对象-关系数据库管理系统（ORDBMS）、数据仓库、联机分析处理（OLAP）和数据挖掘的概念，以及将数据库集成到Web环境中的方法。
- 附录。附录A说明了两种主要的ER表示法：Chen氏表示法和Crow的Feet表示法。附录B提供了方法学的总结以供参考。附录C提出了对基本逻辑数据库设计方法的扩展，这主要用于使用视图集成方法管理的有多个用户视图的数据库系统。附录D提供了一些文件组织和存储结构的背景知识，这主要用于帮助读者理解在第三部分提出的物理数据库设计方法学的一些概念。附录E提供了15个常见的数据模型。

教学风格

为使本书尽可能易读，我们采用了下述风格和结构：

- 在每章的开始都对该章主题有一个简单而清晰的描述。
- 在每章结束的小结部分，总结该章介绍的主要知识点。
- 在大多数章的最后有复习题。
- 所介绍的每个重要概念都有清晰的定义，并放置在一个定义框中。
- 一系列的注意和提示信息。
- 本书使用了大量的图来阐述和说明概念。
- 实践性很强。每一章都包含许多实用的例子来说明所包含的观点。
- 本书最后的术语表对于快速查找是非常有用的。

相应的教师指南和站点

根据Pearson Education的要求，读者可从网站上获得关于本书的一个全面的补充，其中包含大量的指导性的资源。附加的教师指南包括：

- 讲授建议。其中包括授课建议、授课提示，以及利用每章内容的学生项目思想。
- 解决方案。提供了所有复习题的答案示例。
- 测试题。测试题（类似于每一章最后的复习题）以及解决方案。
- 幻灯片（使用PowerPoint创建）。包含每一章的主要知识点、放大的图解、书中的表，以帮助教师授课并就本书的内容进行课堂讨论。
- StayHome数据库系统在Microsoft Access 2002中的实现。
- 创建Perfect Pets数据库系统的SQL脚本，这个脚本可用于在许多关系DBMS中创建数据库，包括Oracle、Informix和SQL Server。
- 所有在附录E中定义的常用数据模型的SQL脚本，这些脚本可用来创建数据库系统的基本表集合。再强调一次，这些脚本可在许多关系DBMS中使用。

关于教师指南和本书可以在Pearson Education站点上找到，Pearson Education的网址是：http://www.booksites.net/connbegg。

修正与建议

由于此类型的书籍难免存在错误，因此你的指正对本书以后的再版很重要。请把意见、改正和有建设性的建议发邮件至：thomas.connolly@paisley.ac.uk。

感谢

本书是作者在行业、研究、理论领域工作多年的成果，因此很难将那些直接或间接帮助过我们的人全部列出来。很多思想可能是在无意识的时候出现的，但一定是有意识的努力的结果。对于这些我们可能忽略了的人，我们在此表示歉意。但特别的感谢和特别的歉意首先必须送给我们的家人，因为在我们写书期间，他们一直被我们忽视、忽略。

我们首先要感谢Kate Brewin——我们的编辑，以及Mary Lince——我们的文字编辑，我们同时要感谢本书的审校者，他们提出了很多的评论、建议和意见。特别地，我们要提一下Stuart Anderson和Andy Osborn，他们对第1版进行了审校；Aurelie Bechina和Nick Measor，他们对第2版进行了审校；Willie Favero对两版都进行了审校。

我们还应该感谢我们的秘书Lyndonne MacLeod和June Blackburn，感谢他们这几年的帮助和支持。

Thomas M.Connolly
Carolyn E.Begg
Glasgow, May 2003

目　录

出版者的话
专家指导委员会
译者序
前言

第一部分　相关知识介绍

第1章　引言 ……1
1.1　数据库系统使用示例 ……1
1.2　数据库方法 ……3
1.2.1　数据库 ……3
1.2.2　数据库管理系统 ……4
1.2.3　数据库应用程序 ……4
1.2.4　视图 ……5
1.2.5　DBMS环境的组成 ……5
1.2.6　DBMS架构 ……5
1.3　DBMS的功能 ……7
1.4　数据库设计 ……10
1.5　DBMS的优缺点 ……11
1.6　本章小结 ……12
复习题 ……12
第2章　关系模型 ……14
2.1　数据模型 ……14
2.2　术语 ……14
2.2.1　关系数据结构 ……15
2.2.2　关系表的属性 ……17
2.2.3　关系键 ……17
2.2.4　关系数据库的表示 ……19
2.3　关系完整性 ……20
2.3.1　空值 ……21
2.3.2　实体完整性 ……21
2.3.3　参照完整性 ……21
2.3.4　业务规则 ……21
2.4　关系语言 ……22
2.5　本章小结 ……22
复习题 ……22
第3章　SQL和QBE ……24
3.1　结构化查询语言 ……24
3.1.1　SQL的目标 ……25
3.1.2　术语 ……25
3.1.3　书写SQL命令 ……26
3.2　数据操纵 ……26
3.2.1　简单查询 ……27
3.2.2　选择行 ……29
3.2.3　给结果排序 ……32
3.2.4　使用SQL的聚合函数 ……33
3.2.5　对结果分组 ……34
3.2.6　子查询 ……36
3.2.7　多表查询 ……38
3.2.8　INSERT、UPDATE和DELETE语句 ……39
3.3　数据定义 ……41
3.3.1　创建表 ……41
3.3.2　建立视图 ……43
3.4　QBE ……44
3.5　本章小结 ……48
复习题 ……49
练习 ……49
第4章　数据库应用程序生命周期 ……51
4.1　软件危机 ……51
4.2　信息系统生命周期 ……52
4.3　数据库系统开发生命周期 ……52
4.4　数据库规划 ……52
4.5　系统定义 ……52
4.6　需求的收集与分析 ……55
4.7　数据库设计 ……56
4.8　选择DBMS ……57
4.9　应用程序设计 ……58
4.9.1　事务设计 ……58
4.9.2　用户界面设计 ……59

4.10 构建原型 ……59
4.11 实现 ……59
4.12 数据转换与加载 ……60
4.13 测试 ……60
4.14 操作性维护 ……61
4.15 本章小结 ……61
复习题 ……62
第5章 数据库管理和安全性 ……63
5.1 数据管理和数据库管理 ……63
5.1.1 数据管理 ……63
5.1.2 数据库管理 ……64
5.1.3 数据管理与数据库管理的比较 ……64
5.2 数据库安全性 ……65
5.2.1 安全威胁 ……65
5.2.2 对策——基于计算机的控制 ……67
5.3 本章小结 ……71
复习题 ……72

第二部分 数据库分析与设计技术

第6章 事实发现 ……73
6.1 什么时候使用事实发现技术 ……73
6.2 收集哪些事实 ……74
6.3 事实发现技术 ……74
6.3.1 检查文档 ……75
6.3.2 面谈 ……75
6.3.3 观察业务的运转 ……76
6.3.4 研究 ……76
6.3.5 问卷调查 ……76
6.4 StayHome案例研究 ……77
6.4.1 StayHome案例研究——概览 ……77
6.4.2 StayHome案例研究——数据库规划 …80
6.4.3 StayHome案例研究——系统定义 ……84
6.4.4 StayHome案例研究——需求收集和分析 ……86
6.4.5 StayHome案例研究——数据库设计 …91
6.5 本章小结 ……92
复习题 ……92
第7章 实体－关系建模 ……93
7.1 实体 ……93
7.2 关系 ……94
7.2.1 关系的度 ……95
7.2.2 递归关系 ……95
7.3 属性 ……95
7.3.1 简单属性和复合属性 ……96
7.3.2 单值属性和多值属性 ……96
7.3.3 派生属性 ……96
7.3.4 键 ……97
7.4 强实体和弱实体 ……98
7.5 关系的多样性约束 ……98
7.5.1 一对一关系 ……99
7.5.2 一对多关系 ……100
7.5.3 多对多关系 ……101
7.5.4 复杂关系的多样性约束 ……102
7.5.5 基数约束与参与约束 ……103
7.6 关系上的属性 ……103
7.7 ER模型中的设计问题 ……104
7.7.1 扇形陷阱 ……104
7.7.2 深坑陷阱 ……106
7.8 本章小结 ……107
复习题 ……108
练习 ……108
第8章 规范化 ……110
8.1 简介 ……110
8.2 数据冗余和更新异常 ……110
8.2.1 插入异常 ……111
8.2.2 删除异常 ……112
8.2.3 更新异常 ……112
8.3 第一范式 ……112
8.4 第二范式 ……114
8.5 第三范式 ……117
8.6 本章小结 ……119
复习题 ……119
练习 ……120

第三部分 逻辑数据库设计

第9章 逻辑数据库设计——步骤1 ……121
9.1 数据库设计方法学简介 ……121
9.1.1 什么是数据库设计方法学 ……121

9.1.2 数据库设计的各阶段 ……122
9.1.3 数据库设计中的关键成功因素 ……122
9.2 数据库设计方法学概述 ……122
9.3 逻辑数据库设计方法学步骤1简介 ……124
9.4 步骤1：创建并检查ER模型 ……124
9.4.1 步骤1.1：标识实体 ……125
9.4.2 步骤1.2：标识关系 ……127
9.4.3 步骤1.3：标识实体或关系的有关属性 ……130
9.4.4 步骤1.4：确定属性域 ……132
9.4.5 步骤1.5：确定候选键、主键和备用键属性 ……133
9.4.6 步骤1.6：特化/泛化实体（可选步骤）…135
9.4.7 步骤1.7：检查模型的数据冗余 ……135
9.4.8 步骤1.8：检查模型是否支持用户事务 ……136
9.4.9 步骤1.9：与用户一起检查模型 ……138
9.5 本章小结 ……138
复习题 ……138
练习 ……139
第10章 逻辑数据库设计——步骤2 ……140
10.1 步骤2：将ER模型映射为表 ……140
10.1.1 步骤2.1：创建表 ……140
10.1.2 步骤2.2：用规范化方法检查表结构…150
10.1.3 步骤2.3：检查表是否支持用户事务 ……151
10.1.4 步骤2.4：检查业务规则 ……153
10.1.5 步骤2.5：与用户讨论逻辑数据库设计 ……156
10.2 本章小结 ……156
复习题 ……156
练习 ……157
第11章 高级建模技术 ……158
11.1 特化/泛化 ……158
11.1.1 超类和子类 ……158
11.1.2 超类/子类关系 ……158
11.1.3 属性继承 ……159
11.1.4 特化过程 ……160
11.1.5 泛化过程 ……160
11.1.6 超类/子类关系的约束 ……162
11.2 创建表达特化/泛化的表 ……163
11.3 本章小结 ……164
复习题 ……164
练习 ……165

第四部分 物理数据库设计

第12章 物理数据库设计——步骤3 ……167
12.1 逻辑与物理数据库设计的比较 ……168
12.2 物理数据库设计方法学概述 ……168
12.3 步骤3：为目标DBMS转换全局逻辑数据模型 ……169
12.3.1 步骤3.1：设计基本表 ……169
12.3.2 步骤3.2：设计派生数据的表示 …173
12.3.3 步骤3.3：设计其他业务规则 ……174
12.4 本章小结 ……177
复习题 ……177
练习 ……178
第13章 物理数据设计——步骤4 ……179
13.1 步骤4：选择文件组织方式和索引 ……179
13.1.1 步骤4.1：分析事务 ……180
13.1.2 步骤4.2：选择文件组织方式 ……184
13.1.3 步骤4.3：选择索引 ……184
13.2 使用Microsoft Access 2002的StayHome文件的组织与索引 ……187
13.2.1 选择索引指南 ……187
13.2.2 StayHome的索引 ……188
13.3 本章小结 ……189
复习题 ……189
练习 ……189
第14章 物理数据库设计——步骤5和步骤6…190
14.1 步骤5：设计用户视图 ……190
14.2 步骤6：设计安全性机制 ……191
14.3 本章小结 ……195
复习题 ……195
练习 ……195
第15章 物理数据库设计——步骤7 ……196
15.1 步骤7：引入受控冗余的考虑 ……196
15.2 本章小结 ……204
复习题 ……205
练习 ……205

第16章 物理数据库设计——步骤8 ……206
16.1 步骤8: 监视并调整操作系统 ……206
16.1.1 理解系统资源 ……207
16.1.2 小结 ……209
16.1.3 StayHome的新需求 ……209
16.2 本章小结 ……210
复习题 ……211

第五部分 第二个实例

第17章 Perfect Pets——逻辑数据库设计 …213
17.1 Perfect Pets ……213
17.1.1 数据需求 ……213
17.1.2 事务需求 ……215
17.2 使用逻辑数据库设计方法 ……216
17.2.1 步骤1.1: 标识实体 ……216
17.2.2 步骤1.2: 标识关系 ……216
17.2.3 步骤1.3: 标识实体或关系的有关属性 ……219
17.2.4 步骤1.4: 确定属性域 ……220
17.2.5 步骤1.5: 确定候选键、主键和备用键属性 ……220
17.2.6 步骤1.6: 特化和泛化实体 ……220
17.2.7 步骤1.7: 检查模型的数据冗余 ……220
17.2.8 步骤1.8: 检查模型是否支持用户事务 ……222
17.2.9 步骤2.1: 创建表 ……222
17.2.10 步骤2.2: 用规范化方法检查表结构 …222
17.2.11 步骤2.3: 检查模型是否支持用户事务 ……222
17.2.12 步骤2.4: 检查业务规则 ……225
17.2.13 步骤2.5: 与用户一起讨论逻辑数据库设计 ……226
第18章 Perfect Pets——使用物理数据库设计方法学 ……227
18.1 步骤3.1: 设计基本表 ……227
18.2 步骤3.2: 设计派生数据的表示 ……229
18.3 步骤3.3: 设计其他业务规则 ……230
18.4 步骤4.1: 分析事务 ……233
18.5 步骤4.2: 选择文件组织方式 ……233
18.6 步骤4.3: 选择索引 ……237
18.7 步骤5: 设计用户视图 ……238
18.8 步骤6: 设计访问规则 ……239
18.9 步骤7: 考虑引入受控冗余 ……242

第六部分 数据库的现状和未来趋势

第19章 数据库的现状和发展 ……243
19.1 高级数据库应用 ……243
19.2 关系DBMS的缺陷 ……245
19.3 分布式DBMS和复制服务器 ……246
19.3.1 DDBMS的优缺点 ……248
19.3.2 复制服务器 ……250
19.4 面向对象的DBMS和对象-关系DBMS …251
19.4.1 面向对象的DBMS ……251
19.4.2 对象-关系DBMS ……253
19.5 数据仓库 ……254
19.6 联机分析处理 ……256
19.7 数据挖掘 ……258
19.8 网络数据库集成和XML ……259
19.8.1 静态和动态的网页 ……259
19.8.2 Web-DBMS集成需求 ……260
19.8.3 集成Web和DBMS的方法 ……260
19.8.4 XML ……260
19.9 本章小结 ……262
复习题 ……263

附 录

附录A 可选的数据建模表示法 ……265
附录B 数据库设计方法学总结 ……270
附录C 高级数据库逻辑设计 ……275
附录D 文件组织和索引 ……284
附录E 常用数据模型 ……297
术语表 ……319
参考文献 ……326

第一部分 相关知识介绍

第1章 引言
第2章 关系模型
第3章 SQL和QBE
第4章 数据库应用程序生命周期
第5章 数据库管理和安全性

第1章 引 言

本章主题：

- 数据库系统的一般用途。
- 术语“数据库”的含义。
- 术语“数据库管理系统”（DBMS）的含义。
- DBMS环境的主要组成部分。
- DBMS的典型功能与服务。
- DBMS的优缺点。

现在，数据库已经成为我们每天生活中不可缺少的一部分，虽然有人可能并没意识到这一点。在开始讨论数据库之前，我们简要地介绍一些数据库系统的应用。为了方便讨论，我们认为“数据库”是相关数据的集合，而“数据库管理系统”（Database Management System, DBMS）是管理和控制对数据库进行访问的软件。更概括的术语“数据库系统”是指与数据库相关的应用程序的集合。在1.2节中，我们将提供更精确的定义。在本章末尾，我们将介绍现代DBMS的典型功能并简要地介绍DBMS的优缺点。

1.1 数据库系统使用示例

1. 从超市购物

当你从当地超市购买货物时，很可能就是在访问数据库。收银员使用条形码阅读器来扫描你的每种货物。这就链接了一个使用条形码从产品数据库中查询该项货物价格的应用程序，然后该程序产生这些库存项目的数量，在收银机上显示价格。如果记录产品的数量低于指定的最低极限值，数据库系统可能会自动设置一个订单来获得更多的该产品的库存。

2. 使用信用卡购物

当你使用信用卡购物时，服务人员要检查你是否有足够的剩余金额可以购买该商品。这种检查可以用电话来进行，或者也可以用连接到计算机系统的磁卡阅读器自动完成。无论是哪种情况，都在某一个地方有一个数据库，此数据库中包含了你使用信用卡进行购物的信息。

为了检查你的信用卡，存在一个数据库应用程序，此程序使用你的信用卡号码来检查你想购买的商品的价格，以及你这个月已经购买的商品的总额是否在信用限度内。当购买被确认有效后，则这次的购买详细信息又被加到了这个数据库中。在确认此次购买生效之前，这个应用程序也会访问数据库，检查该信用卡是否在被盗或丢失列表中。还有其他数据库应用程序，它们向每个持卡者每月发送通知，并在收到付款后寄去账单。

3. 在旅行社预定假期

当你咨询某次假期的安排时，旅行社可能访问几个包含假期和飞机的详细信息的数据库。当你预定假期时，数据库系统必须进行所有必要的预定安排。在这种情况下，该系统必须要确保不同的代理没有预定相同的假期或飞机上相同的座位。例如，如果在从伦敦到纽约的飞机上只剩下一个座位，两个代理在同一时间预定这最后一个座位，系统不得不处理这种情况，允许一个预定继续，并通知另一个代理没有位置了。通常，旅行代理都有独立的用来记账的数据库。

4. 使用图书馆

图书馆一般会有一个包含所有图书的详细信息的数据库，其中的信息可能还包括读者的信息、预定信息等等，可能会允许读者基于书名、作者或其他数据查找所需书籍。数据库系统可能会允许读者预定书籍，并在书籍可以借阅时发邮件通知读者。该系统也给没有按期还书的借阅者发提醒通知。一般情况下，系统都有一个条形码阅读器，类似于前边超市中描述的那种，用来记录归还和借出图书馆的书籍。

5. 出租录像

当你希望从录像租借公司租一盘录像时，你可能会发现这个公司维护着一个数据库，这个数据库中包括关于每个录像的片名、录像有几份拷贝的详细信息，这个录像目前是可以租借的还是已经被借出了，它的成员（租借者）的详细信息、他们目前正在租借的录像以及归还日期。这个数据库可能还为每个录像存储更详细的信息，比如它的导演和演员。这个公司可以使用这个信息来监视库存的使用，并根据历史的租借数据预测将来的购买倾向。例如，图1-1说明了这个公司的一些示例数据。

Video

catalogNo	title	category	dailyRental	price	directorNo
207132	Die Another Day	Action	5.00	21.99	D1001
902355	Harry Potter	Children	4.50	14.50	D7834
330553	Lord of the Rings	Fantasy	5.00	21.99	D4576
781132	Shrek	Children	4.00	18.50	D0078
449624	Men in Black II	Action	4.00	29.99	D5743
634817	Independence Day	Sci-Fi	4.50	22.99	D3765

Director

directorNo	directorName
D1001	Lee Tamahori
D7834	Chris Columbus
D4576	Peter Jackson
D0078	Andrew Adamson
D5743	Barry Sonnenfeld
D3765	Roland Emmerich

Actor

actorNo	actorName
A1002	Pierce Brosnan
A3006	Elijah Wood
A2019	Will Smith
A7525	Tommy Lee Jones
A4343	Mike Myers
A8401	Daniel Radcliffe

Role

actorNo	catalogNo	character
A1002	207132	James Bond
A3006	330553	Frodo Baggins
A3006	902355	Harry Potter
A2019	330553	Captain Steve Miller
A2019	449624	Agent J
A7525	634817	Agent K
A4343	781132	Shrek

图1-1　录像租借公司的数据示例

6. 使用Internet

Internet上的很多站点是由数据库应用程序驱动的。例如，你可能访问过一个在线书店，这个书店允许借书和买书，比如Amazon.com。这个书店允许你按不同种类借书，比如计算或管理，它也允许你按作者名借书。不管是哪种情况，在这个机构的Web服务器上都有一个数据库，数据库中包含了图书的详细信息、获得方式、邮寄信息、库存级别以及排列信息。图书详细信息包括书名、ISBN、作者、价格、销售历史、出版社、评论以及详细的描述。数据库允许图书被交叉检索，比如，一本图书可以列在几个种类中，比如，计算、编程语言、畅销书以及被推荐图书等。交叉检索也使Amazon可以给你一些其他图书的信息，这些图书一般是与你所感兴趣的图书有关的。

这只是几种数据库系统应用，毫无疑问，你会知道其他许多应用情况。尽管我们熟知并常用这些应用，但在它们的背后却隐藏着复杂的高级技术。这种技术的核心就是数据库本身。对于要尽可能有效地支持最终用户需求的系统来说，需要合适的结构化数据库。构建这种结构就是所说的数据库设计，这也是本书讨论的核心内容。无论要构建的是小型的数据库，还是上面所说的那种大型数据库，数据库设计都是基础，本书中提出的方法学将帮你轻松地构建正确的数据库。拥有一个设计优良的数据库可以使你构建符合用户需求的系统，同时提供可接受的性能。

1.2 数据库方法

在本节中，我们为术语“数据库”和“数据库管理系统”提供更为正式的定义。

1.2.1 数据库

数据库（Database） 逻辑上相关的可共享的数据（以及数据的描述）集合，用于处理公司所需的信息。

让我们详细分析数据库的定义以便全面理解这个概念。数据库是一个巨大的数据存储地，可以同时被许多部门和用户使用。这些用户所需要的所有数据用最小的冗余集成在一起。更重要的是，数据库通常不属于任何一个部门或用户，而是公司内的共享资源。

除了存储公司的数据，数据库也存储数据的描述。因此，数据库也定义为完整记录的自描述集合。数据的描述，也就是元数据——关于数据的数据，被称为系统目录或数据字典。正是数据库的自描述特性提供了被称为数据独立性的特点。这就意味着如果有新的数据结构要加入到数据库中，或者要修改数据库中已经存在的数据结构，那么如果应用程序不直接依赖于被修改的部分，则对使用该数据库的应用程序来说没有影响。例如，如果我们为记录添加了新列或者创建了新表，则对已经存在的应用程序是没有影响的。但是，如果从一个应用程序使用的表中删除了一列，那么这个应用程序就会受到这个变化的影响，因此也必须相应地进行修改。

我们要解释的数据库定义中的最后一个术语就是“逻辑相关”。当我们分析了公司的信息需求后，我们尽力去标识数据库中要描述的重要对象和这些对象之间的逻辑关系。我们提出的数据库设计方法学将引导你标识这些重要的对象和它们之间的逻辑关系。

1.2.2 数据库管理系统

DBMS（数据库管理系统） 一个能够让用户定义、创建和维护数据库以及并控制对数据库的访问的软件系统。

DBMS是与用户、应用程序和数据库进行相互作用的软件。其中，DBMS允许用户从数据库中插入、更新、删除和检索数据。拥有所有数据和数据描述的核心仓库使得DBMS能够提供对数据进行通常查询功能的语言，称为查询语言。这种查询语言的提供（例如SQL），降低了早期系统中所存在的问题，那时，用户不得不使用固定的一组查询，或者用额外的程序来解决主要的软件管理问题。我们将在下一节讨论DBMS的这些典型功能和服务。

结构化查询语言（Structured Query Language, SQL，发音为S-Q-L或者See-Quel） 关系数据库管理系统（例如，Microsoft Access，Microsoft SQL Server和Oracle）的主要查询语言。

1.2.3 数据库应用程序

应用程序（Application Program） 一个通过向DBMS发出合适的请求（一般是一个SQL语句）与数据库交互的计算机应用程序。

用户通过那些用于创建和维护数据库并产生信息的应用程序与数据库进行交互。这些程序可以是传统的批应用程序，或者是目前更常见的联机应用程序。这些应用程序可以用一些程序设计语言编写，也可以在一些更高级的第四代语言中编写。图1-2说明了数据库的使用情形，它显示了Sales（销售）部门和Stock Control（库存控制）部门使用他们的应用程序通过DBMS来访问数据库的情况，每个部门的应用程序处理他们的数据项、维护数据并产生报告。数据的物理结构和存储由DBMS来管理。

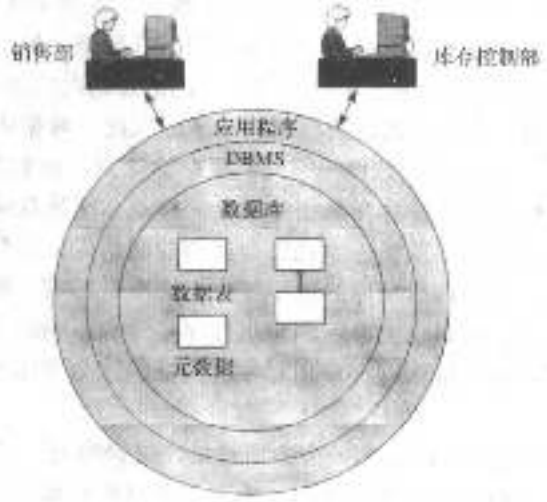


图1-2 销售和库存控制部门通过应用程序和DBMS访问数据库的情形

图1-1说明了数据库的访问形式。它表明销售和库存控制部门使用他们的应用程序，通过DBMS对数据库进行访问。每个部门的应用程序处理数据入口、数据维护和生成报表。数据的物理结构和存储由DBMS管理。

1.2.4 视图

根据前面所描述的功能，DBMS是非常强大的工具。但是，最终用户对系统的任务是复杂还是简单并不感兴趣，他们可能会认为DBMS使事情变得更复杂了，因为他们现在可能看到比他们实际所需要或想做的工作更多的数据。考虑到这个问题，DBMS提供了称为视图的机制，该机制允许每个用户在数据库中有自己定制的数据库视图，这些视图是数据库的子集。

> **视图**（View） 一个“*虚拟的表*”，*它不实际存在在数据库中，但它由DBMS从视图所涉及的基本表中产生。*

视图通常被定义成在基本表上进行操作的查询所产生虚拟表，除了可以让用户以他们希望的方式看到数据来降低复杂度外，视图还有其他几个优点：

- 视图提供了一个安全级别。视图可以不包含那些用户不能看到的数据。例如，我们可以创建一个允许分公司负责人和财务部门看的有关所有职工的数据的视图，这个视图中包括工资的详细信息。而且,我们可以创建另一个视图，在这个视图中不包括工资的详细信息，可由其他职工使用。
- 视图提供了数据库的定制的显示机制。例如，库存控制部门（Stock Control Department）可能希望把录像的日租金率（Daily Rental Rate）称为日租金（Daily Rental）。
- 视图能提供一致的、稳定的数据库结构图，甚至当数据库发生了变化时也是这样（例如，增加或删除列，关系改变了，数据文件分离了，数据库被重构了或者重新命名了）。如果从数据文件中添加或删除了列，而且这些列与视图没有关系，则视图不会受这些变化的影响。因此，视图提供了数据库所提供的另一种数据独立性，我们在1.2.1节中曾描述过数据独立性。

1.2.5 DBMS环境的组成

DBMS环境有五个主要组成部分：硬件、软件、数据、过程和人。

1) **硬件** DBMS和应用程序运行的计算机系统。这可以从单台PC机，到单台大型机，再到计算机网络。

2) **软件** DBMS软件和应用程序以及操作系统，如果将DBMS用在网络上，那么还包括网络软件。

3) **数据** 数据扮演了硬件、软件以及人之间的桥梁。正如我们已经说过的，数据库包含了运行数据和元数据（关于数据的数据）。

4) **过程** 控制数据库设计和使用的指令和规则。这包括如何登录到DBMS的指令集，制作数据库备份以及如何处理软硬件错误。

5) **人** 包括数据库设计者、数据库管理员（DBA）、应用程序员和最终用户。

1.2.6 DBMS架构

在Web出现之前，DBMS通常被分为两部分：

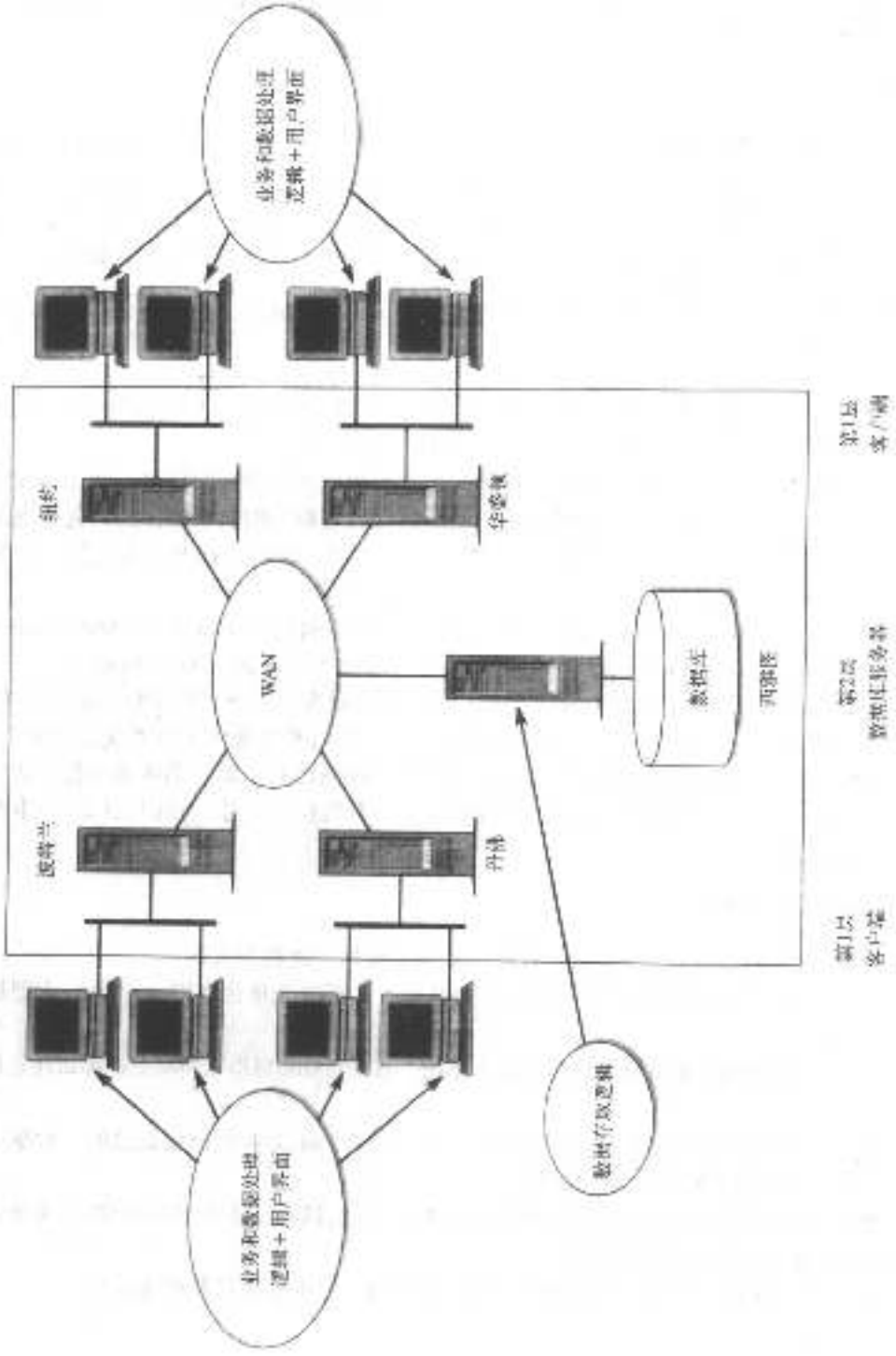


图1-3 StayHome的简化的两层客户-服务器配置

- 用来处理主要业务、数据处理逻辑以及与用户交互的接口的客户端程序。
- 管理和控制对数据库的访问的服务器端程序（有时称为DBMS引擎）。

这就是所谓的两层客户－服务器架构。图1-3所示的为一个办公地点遍布美国的录像出租公司StayHome的简化的客户－服务器架构。它显示了位于西雅图的公司总部的中央数据库和服务器，以及一些位于美国各地的分公司的客户端。

在20世纪90年代中期，应用程序变得更加复杂，并可能会为成百上千的最终用户服务，这种架构的客户端产生了两个问题:

- “胖”客户端，为使运行有效，需要大量客户端计算机上的资源（包括磁盘空间、RAM和CPU的功率）。
- 相当可观的客户端管理的开销。

到1995年，出现了一种新的传统的两层客户－服务器模型的改良版本，它解决了这些问题，这就是所谓的三层客户－服务器架构。这种新的架构提出了三个层次，每层运行于不同的平台上:

1) **用户接口层** 运行在最终用户的计算机（客户端）上。

2) **业务逻辑和数据处理层** 这个中间层运行在服务器上，并且经常被称为应用程序服务器。应用程序服务器是用于为多个客户端提供服务的。

3) **DBMS** 存储中间层所需要的数据。该层可以运行在被称为数据库服务器的独立的服务器上。

这种三层设计与传统的两层设计相比有许多优点，比如:

- “瘦”客户端，需要的硬件代价较小。
- 将许多最终用户的业务逻辑集中在单一的应用程序服务器上，从而简化了应用程序的维护。这消除了传统两层客户－服务器模型软件分布的问题。
- 增加了模块性，更容易修改和替换某一层，而不影响其他层。
- 把核心业务逻辑从数据库功能中分离出来，从而使负载更容易平衡。例如，一个事务处理监视器（Transaction Processing Monitor，TPM）可以用来减少连接数据库服务器的次数（TPM是为了给联机事务处理（OLTP）提供稳定环境而控制客户和服务器间数据传输的程序）。

还有一个优点是三层架构非常自然地适合Web环境，将Web浏览器作为“瘦”客户端，Web服务器作为应用程序服务器。三层客户－服务器架构如图1-4所示。

1.3 DBMS的功能

本节简单看一下当前我们所期望的全面的DBMS提供的功能和服务。

1. 数据存储、检索和更新

这是DBMS的基础功能。从我们前面的讨论中，很清楚要提供这种功能，DBMS要对用户隐藏内部的物理实现细节（例如文件组织方式和存储结构）。

2. 用户可访问的目录

DBMS的一个关键特征是提供了完整的系统目录来保存与数据库结构、用户、应用程序等有关的数据。该目录对用户和DBMS来说都是可以获得的。信息的数量和使用信息的方式随着DBMS的不同而不同。通常情况下，系统目录存储以下内容。

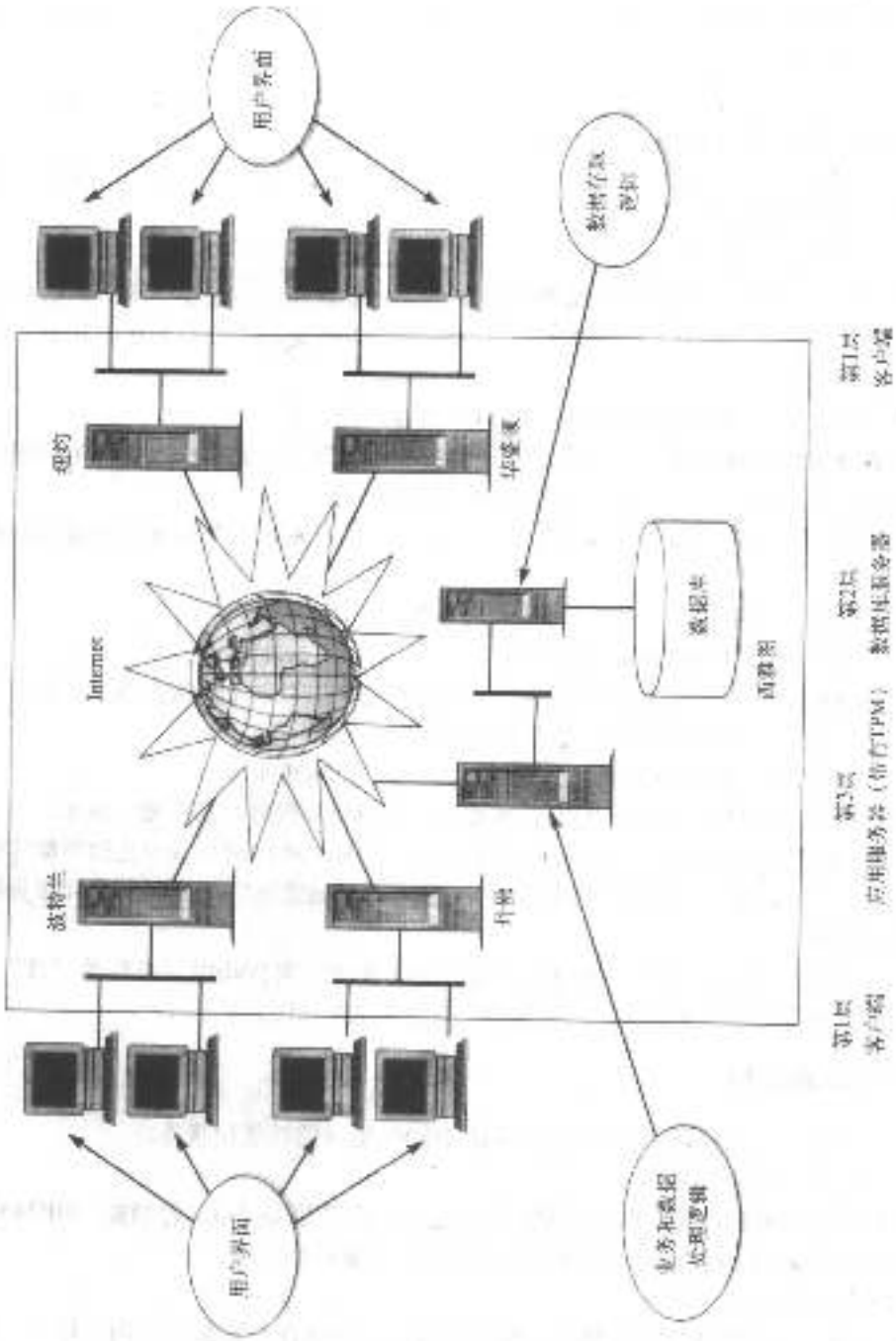


图1-4　SnyHome的简化的三层客户-服务器配置

- 数据项的名称、类型和大小。
- 数据的完整性约束。
- 有权访问该数据的用户名称。

3. 事务支持

> **事务**（Transaction） 由用户或应用程序执行的一个动作或一系列动作，这些动作访问或修改数据库的内容。

例如，StayHome的简单事务可能是在数据库中增加新员工、修改某些员工的工资或着从登记簿中删除一个成员。更复杂一些的例子可能是从数据库中删除负责人，并且重新任命一名成员作为负责人。在这种情况下，对数据库进行的更改就不止一次。如果事务在执行期间失败了，这可能是因为计算机出现故障，这时数据库将会处于一个不一致的状态：已经完成了一些更改，还有一些没有完成。例如，分公司还没有分配一个新的负责人。结果，已经完成的变化不得不撤销，使数据库再次回到一致状态。

为了克服这种问题，DBMS提供了一种机制用来确保给定事务内的所有操作均完成或者均不做。

4. 并发控制服务

使用DBMS的一个主要目的就是使多用户可以同时访问共享数据，这称为并发控制。如果所有用户只是读数据，并发访问相对来说就比较容易，因为他们相互没有影响。但是，当两个或更多的用户同时访问数据库，而且他们之中至少有一个在更新数据，则可能会导致数据不一致。例如，有两个事务T_1和T_2同时执行，如图1-5所示。

时间	T_1	T_2	bal_x
t_1		读 (bal_x)	50
t_2	读 (bal_x)	$bal_x = bal_x - 20$	50
t_3	$bal_x = bal_x + 5$	写 (bal_x)	30
t_4	写 (bal_x)		55
t_5			

图1-5 丢失更新问题

T_2从StayHome成员的金额中减去20美元（当前余额bal_x为50美元），并且T_1为相同的账户添加了5美元。如果事务是顺序执行的，没有交错执行，无论哪个事务先执行，最后的金额都应该是35美元。两个事务几乎是同时开始的，同时读到bal_x为50美元。T_2将bal_x减去20美元得到30美元并且存于数据库中。同时，事务T_1将bal_x增加5美元为55美元，并且将该值也存于数据库中，这样就覆盖了前面的更新，也就“丢失”了20美元。

当多用户访问数据库时，DBMS必须确保不会发生类似的冲突。

5. 恢复服务

当讨论事务支持时，我们注意到如果事务失败，数据库必须要返回到一个一致性状态，这就是所谓的恢复控制。它的产生可能是因为系统崩溃、介质失败和导致DBMS停止的硬软件错误，也可能是因为事务运行期间用户发现了错误，所以在完成前异常终止了该事务。无论是何种情况，DBMS都必须提供将数据库恢复到正确的一致性状态的机制。

6. 授权服务

不难发现这样的情况，我们并不希望数据库中的数据对所有用户都是可见的。例如，我

们可能只希望分公司负责人和财务部门的人员可以看到员工相关的工资信息，而不允许所有其他用户看见这些数据。此外，我们可能希望不允许未授权的用户访问数据库。术语“安全性”（Security）就是防止数据库被非法访问，包括有意的和偶然的。我们希望DBMS提供确保数据安全的机制。

7. 对数据通信的支持

绝大多数用户从终端访问数据库。有时，这些终端直接连接到装有DBMS的主机上。而其他情况下，终端在遥远的地方，与装有DBMS的主机通过网络关系。无论哪种情况，DBMS都必须能结合网络通信软件。甚至面向PC机的DBMS也应该能在局域网（LAN）上运行，这样才能为用户创建一个共享的核心数据库，而不是一系列不同的数据库。

8. 完整性服务

数据库完整性（Database Integrity）指的是存储数据的正确性和一致性。这可以看成是另一种类型的数据库保护。尽管它与安全性相关，但它有更广的内涵。完整性关心的是数据本身的质量。完整性通常用术语“约束”（Constraint）来表达，约束是数据库中不可违反的一致性原则。例如，我们可能希望说明这样的约束，StayHome中的成员都不能同时租借超过10盘的录像。这时，我们希望当租借一盘录像给一个成员时，DBMS会检查是否超过了这个限制，并在达到界限时防止再次借出。

9. 加强数据独立性的服务

正如我们在1.2.4节中讨论的，数据独立性通常是通过视图机制获得的。通常有几种类型的数据库物理特性的变化不会影响视图，例如使用不同的文件组织方式或者修改索引。这称为物理数据独立性。但是，完全的逻辑数据独立性更难实现。增加新表或者新列通常是可以的，但删除它们就不行。在有些系统中，对已经存在的表的任何类型的改变都是禁止的。

10. 实用工具服务

实用工具程序帮助DBA有效地管理数据库。实用工具的例子有：

- 导入工具，将数据从平面文件导入到数据库中；导出工具，将数据库中的数据导出到平面文件中。
- 监视工具，监视数据库的使用和操作。

> 上面的讨论是基于通常情况进行的。DBMS提供的实际功能的等级随产品的不同而不同。例如，面向PC的DBMS可能不支持并发共享访问，它可能只提供了有限的安全性、完整性和恢复控制。但是，现代大型的多用户DBMS产品都提供了以上所有的功能，甚至更多。现代系统是极端复杂的，它由几百万行代码组成的软件代码段以及大量的文档组成。

前面的讨论被有意简化了，但是对DBMS功能的概要介绍已经足够了。有兴趣的读者可以参考Connolly和Begg（2002）。

1.4 数据库设计

到现在为止，我们认为数据库中存在数据的结构是理所当然的。但我们如何获得这种结构？答案很简单：数据库结构是数据库设计期间形成的。但是，进行数据库设计是极为复杂的。构造一个满足公司信息需求的系统需要一个数据驱动的方法，也就是我们首先想到的应

该是数据，然后才是应用程序。系统最终是否能被用户接受，数据库设计是至关重要的。一个设计不良的数据库会产生很多错误，这些错误可能导致错误的决策，这对公司会有非常严重的反映。另一方面，设计良好的数据库可以为决策制定过程的成功提供正确的信息。

我们用了几章的篇幅介绍了完整的数据库设计方法学（参见第9章～第16章）。我们将这个方法学分解为一系列简单的步骤，并提供了全部的概要提示。在这些章中，我们使用了一个叫做StayHome的基于录像出租公司的例子。为了帮助强化方法学，在第17章和第18章中我们用了另一个案例来学习，这是个叫做Perfect Pets的宠物诊所。此外，在附录E中，我们提供了大量的常见的业务数据模型。

遗憾的是，数据库设计方法学并不是非常的流行，可能这是开发数据库系统失败的主要原因。由于数据库设计中缺乏结构化的方法，数据库工程所必需的时间和资源通常被低估了，在面对应用程序的需求时，所开发的数据库并不充分、效率也较低、文档有限、维护也很困难。

我们希望本书中提出的方法学将帮助改变这种状态。

1.5 DBMS的优缺点

你阅读本书可能说明你已经知道了DBMS的许多优点，例如:

- 控制数据冗余。数据库方法尽可能地消除了冗余。但是，并没有完全消除冗余，而是控制了大量数据库固有的冗余。例如，为了表达数据间的关系，键数据项的重复一般是必要的，有时为了提高性能也会重复一些数据项。继续阅读本书，你就会明白受控重复的原因。
- 数据一致性。通过消除或控制冗余，我们降低了不一致性产生的危险。如果数据项在数据库中只存储了一次，则任何对该值的更新均只需进行一次，而且新的值立即就可以被所有用户获得。如果数据项不只存储了一次，而且系统意识到这点，系统将可以确保该项的所有拷贝都保持一致。但是，目前许多DBMS都不能自动确保这种类型的一致性。
- 数据共享。在基于文件的访问方式下，数据通常只被使用部门或人员拥有，而数据库属于整个公司，可以被有权限的用户共享。这种方式使更多的用户共享了更多的数据。此外，新的应用程序可以依赖于数据库中已经存在的数据，并且只增加目前没有存储的数据，而不用重新定义所有的数据需求。新的应用程序也可以依赖于DBMS提供的功能（例如数据定义和操纵，以及并发和恢复控制），而不是自己提供这些功能。
- 增强的数据完整性。正如我们已经说明的，数据库完整性通常是通过术语“约束”来表示的，它是数据库中必须遵循的一致性规则。约束可以应用在一个记录的数据项上，也可以应用在记录之间的关系上。此外，数据集成允许我们定义完整性约束，并由DBMS来保证这些完整性约束的实现。
- 通过数据独立性提高了数据的维护。既然DBMS将数据描述从应用程序中分离出来，这使应用程序不受数据描述变化的影响，这就是数据独立性，它的出现简化了数据库应用程序的维护。

其他的优点包括：可提高安全性、提高数据可访问性和响应性能、提高效率、提高并发性以及提高备份和恢复服务。但是，数据库方法也有一些缺点:

- 复杂性。正如我们前面所提到的，DBMS是极端复杂的软件组合，所有用户（数据库设计者和开发者，DBA以及最终用户）都必须理解这些功能以便能充分利用它。
- DBMS费用。DBMS的费用随所提供的环境和功能的不同而不同。例如，一个PC机的单

用户DBMS可能只需100美元。但是，一个支持成百用户的大型机的多用户DBMS可能是极为昂贵的，可能10万美元到100万美元。而且还有每年维护所用的周期性花费，一般为所列价格的百分之一。

- 转换费用。在某些情况下，DBMS和其他硬件上的费用可能与升级已经存在的应用程序，使它们可以在新的DBMS和硬件上运行一样多。这些费用也包括训练员工使用新的系统和可能雇佣专业人员帮助升级和运行系统的花费。这些费用是一些公司依赖于当前的系统而不能转向更现代数据库系统的一个主要原因。术语“遗留系统”（legacy system）有时就用来指那些比较旧的，而且通常比较差的系统（例如基于文件的、层次或网络系统）。
- 性能。通常，基于文件的系统是为特定应用设计的，例如货物计价。其结果是，性能通常很好。但是，DBMS是为更泛化并迎合许多应用而不是一个而编写的。这可能就会产生某些应用不再那么快了的结果。
- 更高的故障影响。资源高度集中使系统更脆弱。由于所有用户和应用都依赖于DBMS的可用性，所以任何组件的故障都会给操作带来完全的停止直到故障被修复。

1.6 本章小结

- 数据库是逻辑上相关的可共享的数据（以及数据的描述）集合，用于处理公司所需的信息。DBMS是用户可以定义、创建和维护数据库以及提供对该数据库访问进行控制的软件系统。一个应用程序是一个计算机程序，它通过向DBMS发出合适的请求（通常是SQL语句）来与数据库交互、更广泛地说，数据库系统是用于定义与数据库、DBMS和数据库本身交互的应用程序的集合。
- 所有对数据库的访问都要通过DBMS。DBMS提供了允许用户定义数据库，从数据库中插入、更新、删除和检索数据的功能。
- 构成DBMS环境的主要五个方面是：硬件（计算机）、软件（DBMS、操作系统和应用程序）、数据、过程和人。人包括数据库管理员（DBA）、数据库设计者、应用程序员和最终用户。
- 在Web环境中，传统的两层客户-服务器模型已经被三层模型替代，三层模型由用户界面层（客户）、业务逻辑和数据处理层（应用服务器）构成，并且DBMS（数据库服务器）分布在不同的机器上。
- DBMS提供对数据库的受控访问。它提供安全性、完整性、并发和恢复控制以及用户可访问的目录，它也提供视图机制来简化用户要处理的数据。
- 数据库访问方式的优点包括控制数据冗余、数据一致性、数据共享以及增强安全性和完整性。缺点包括复杂性、费用高、降低了性能和更高的故障影响。

复习题

1.1 除了本书1.1节列出的数据库系统外，再列出四个数据库系统的例子。

1.2 讨论下述术语的含义：

(a) 数据

(b) 数据库

(c) 数据库管理系统

(d) 应用程序

(e) 数据独立性

(f) 视图

1.3 描述一下数据库应用的主要特征。

1.4 描述DBMS环境的五个组成部分，并讨论它们彼此之间的关系。

1.5 描述传统的两层客户-服务器架构的问题，并讨论在三层客户-服务器架构中是如何克服这些问题的。

1.6 描述现代的功能全面的多用户DBMS应该提供哪些功能。

1.7 对你在问题6中提出的功能，你认为哪个功能在个人PC DBMS中是不必要的？给出合适的理由。

1.8 讨论DBMS的优点和缺点。

第2章 关 系 模 型

本章主题:

- 什么是数据模型以及它的用途。
- 关系模型的术语。
- 怎样使用表来描述数据。
- 数据库关系的属性。
- 如何标识候选键、主键、备用键和外键。
- 实体完整性和参照完整性的含义。
- 关于SQL和QBE，两种使用最广泛的关系语言。

关系数据库管理系统（Relational Database Management System, RDBMS）已经成为主流的数据处理软件，估计每年出售的总值大约为150亿~200亿美元（还包括500亿的工具出售），并且每年以大约25%的速率在增长。RDBMS代表了第二代DBMS，它基于E.F.Codd博士1970年的“A Relational Model of Data for Large Shared Data Banks”（大型共享数据银行的关系数据模型）论文中所提出的关系数据模型。在这种关系模型中，所有数据都被逻辑地组织到关系（表）中。关系模型的一个强大优势就是这种简单的逻辑结构。然而，在这种简单的逻辑结构背后，是合理的理论基础，而这正是第一代DBMS（网络和层次DBMS）所缺乏的。

本书提出的设计方法学是基于关系数据模型的，因为这种模型是你最可能使用的。在本章中，我们讨论关系数据模型的基本概念。让我们首先看一下什么是数据模型。

2.1 数据模型

数据模型（Data Model） 描述数据、数据间的关系以及公司所使用的数据的约束的概念集合。

模型是“现实世界”对象、事件和他们之间的关系的描述。它主要体现公司本质和内在的东西，而忽视偶然因素。数据模型尽力去描述要建模的公司或者公司的一部分。它应该为数据库设计者和最终用户提供基本概念和符号，使他们明确而准确地交流对公司数据的理解。数据模型可以认为由如下三部分组成:

1）结构部分，由定义如何构造数据库的一组规则组成。

2）操作部分，定义允许在数据上进行的操作类型（包括更新和检索数据的操作和改变数据库结构的操作）。

3）一组可能的完整性规则集合，确保数据是正确的。

数据模型的目的是描述数据并且使数据可以理解。如果能够做到这点，那么设计数据库会变得容易得多。在本章剩下的部分，我们研究这样的数据模型：关系数据模型。

2.2 术语

关系模型基于关系这一数学概念，而关系在物理上对应为表。Codd作为专业数学家，

使用数学中的术语创建了主要的理论和谓词逻辑。在本节中，我们解释关系模型中的术语和结构概念。在2.3节中，我们将讨论模型的完整性规则，在2.4节中，我们解释模型的操作部分。

2.2.1 关系数据结构

关系（Relation） 具有列和行的表。

关系DBMS只需要用户将数据库看成是表。

注意 这种认识只是应用于我们看数据库的方式，并不应用于数据库在磁盘上的物理结构，我们可以应用不同的物理存储结构（例如堆文件和哈希文件）。

属性（Attribute） 关系中被命名的列。

在关系模型中，我们用关系来表示我们要在数据库中表达的对象的信息。我们把关系描述为表，表中的行对应不同的记录，表中的列对应不同的属性。属性可以以任何顺序出现，而关系仍保持不变，因此表达了相同的含义。

例如，在StayHome例子中，分公司的信息由Branch关系描述，有属性列branchNo(分公司号码)、street、city、state、zipCode和mgrStaffNo(对应于分公司负责人的员工号码)。相似的情况是，员工的信息由Staff关系所描述，包括属性列staffNo（员工号码）、name、position、salary和branchNo（员工工作的分公司号码）。图2-1为Branch和Staff关系的例子。从这个例子可以看到，列包含一个属性的值，例如，branchNo列只包含分公司号码。

域（Domain） 一个或多个属性的取值范围。

域是关系模型的一个重要特征。关系数据库中每个属性都与一个域相关。每个属性的域都可能不同，也有可能两个或更多的属性有相同的域。图2-2为Branch和Staff关系中一些属性的域。

注意 在给定的时间，域中可能会有目前并不在属性中出现的值。换句话说，域描述了属性可能的值。

域的概念是很重要的，因为它允许我们定义属性可以具有的值的意义和来源。其结果是，系统可获得更多的信息，并且可以拒绝不合理的操作。例如，比较员工代码和公司代码对我们来说是不明智的，尽管在域定义中，这些属性都是字符串。遗憾的是，现在很多RDBMS并不支持域定义。

元组（Tuple） 关系中的一行记录。

关系的元素是表中的元组或记录。在Staff关系中，每个记录包含五个值，每个值代表一个属性。因为有属性，元组可以以任何顺序出现而关系并没有改变，因此也就表达了相同的意思。

最后，我们给出关系数据库的定义：

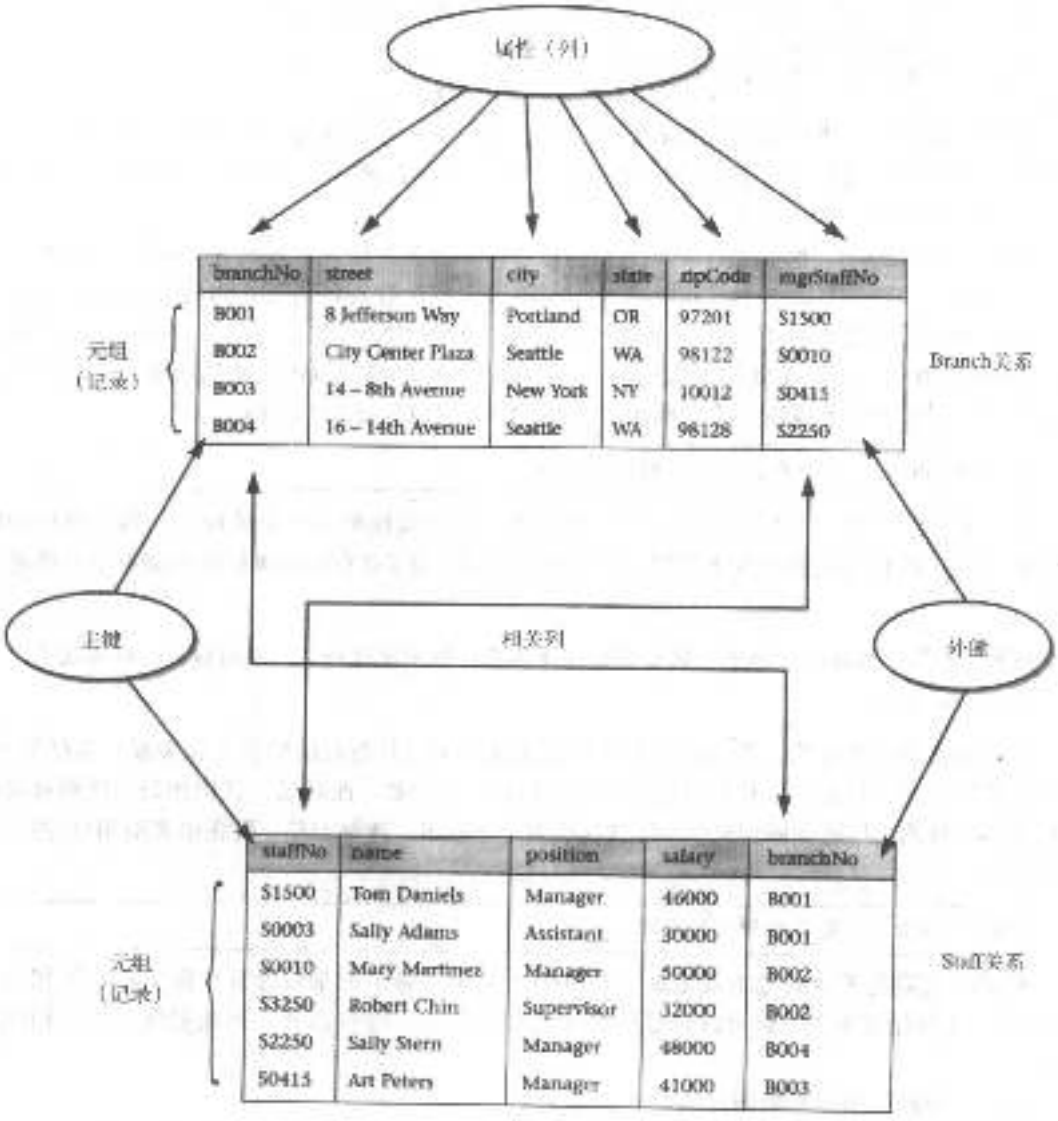
关系数据库（Relational Database） 规范化的表的集合。

关系数据库由结构正确的表组成。我们将这种正确性称为规范化。我们将在第8章讨论规范化问题。

选择术语

关系模型的术语非常令人困惑。本章中，我们介绍两组术语："关系、属性和元组"和"表、列和记录"。其他可能会遇到的术语还有：文件（用于表）、行（用于记录）以及字段（用于列）。你可能也发现了这些术语的不同组合，例如表、行和字段。

> 从现在开始，我们将尽量不用正式的术语，如关系、元组和属性，而是使用更常用的术语，如表、记录和列。



branchNo	street	city	state	zipCode	mgrStaffNo
B001	8 Jefferson Way	Portland	OR	97201	S1500
B002	City Center Plaza	Seattle	WA	98122	S0010
B003	14 – 8th Avenue	New York	NY	10012	S0415
B004	16 – 14th Avenue	Seattle	WA	98128	S2250

staffNo	name	position	salary	branchNo
S1500	Tom Daniels	Manager	46000	B001
S0003	Sally Adams	Assistant	30000	B001
S0010	Mary Martinez	Manager	50000	B002
S3250	Robert Chin	Supervisor	32000	B002
S2250	Sally Stern	Manager	48000	B004
S0415	Art Peters	Manager	41000	B003

图2-1　Branch和Staff关系的例子

属性	域名	含义	域的定义
branchNo	Branch_Numbers	设置所有可能的分公司号	字符：大小为4，范围为B001~B999
street	Street_Names	设置所有可能的街道名	字符：大小为60
staffNo	Staff_Numbers	设置所有可能的员工号	字符：大小为5，范围为S000~S9999
position	Staff_Positions	设置所有可能的员工职位	Director、Manager、Supervisor、Assistant、Buyer中之一
salary	Staff_Salaries	设置所有可能的员工薪水值	货币：8位数，范围为$10000.00~$100000.00

图2-2 Branch和Staff关系的一些属性的域

2.2.2 关系表的属性

关系表有如下属性：

- 数据库中每个表都有区别于其他表的名称。
- 表中的每个单元恰好只包含一个值（例如，在一家分公司的一个单元中存储几个电话号码是不对的。换句话说，表不能包含重复的组。满足此属性的关系表就称为是规范化的或者是第一范式的）。
- 每个列有不同的名字。
- 一个列的值来自相同的域。
- 列的顺序不重要。换句话说，给定列名，我们可以交换整列。
- 每个记录都是不同的，没有重复记录。
- 理论上说，记录的顺序并不重要（但是，实际上，这个顺序可能影响访问记录的效率，参见第13章）。

2.2.3 关系键

正如我们刚刚提到的，表中的每个记录都必须是唯一的。这意味着我们需要能够标识的一个列或列的组合（称为关系键）来提供唯一性。在本节中，我们解释用于关系键的术语。

超键（Superkey） 一个列或者列集，唯一地标识了表中的一个记录。

由于超键可能包含用于唯一标识列所不必要的额外的列，因此，我们对仅包含能够唯一标识记录的最小数量的列更感兴趣。

候选键（Candidate Key） 仅包含唯一标识实体所必需的最小数量的属性/列的超键。

表的候选键有两个属性：

- 唯一性：在每条记录中，候选键的值唯一标识该记录。
- 最小性：没有具有唯一性属性的候选键的子集。

考虑如图2-1所示的Branch表。对于给定的城市的值，我们可以确定几个分公司（例如，西雅图有两个分公司）。因此这个列不能被作为候选键。另一方面，因为StayHome为每个分公司分配了一个唯一的分公司代码，那么对给定的分公司的代码值，branchNo，我们最多可以确定一条记录，因此branchNo是候选键。类似的情况是，由于没有任何两家分公司有相同的邮政编码，所以ZipCode(邮政编码)也可以作为Branch表的候选键。

在一个表中也可能有几个候选键。例如，考虑叫做Role的表，它描述了演员在录像中所扮演的角色。这张表由演员号码（actorNo）、目录号码（catalogNo）和剧中的角色名（character）组成，如图2-3所示。对给定的演员号码actorNo来说，该演员可能在几个不同的录像中参加演出。同样地，对于给定的目录号码catalogNo来说，可能有几个不同的演员参加了该录像的演出。因此，actorNo和catalogNo本身都不能作为候选键。但actorNo和catalogNo的组合，一次最多只能标识一条记录。当键由多于一个的列组成时，我们称之为候选键。

提示　注意不要看到了例子的数据就推演出候选键，除非你确定例子可以代表将要存于表中的数据。通常，表的一个示例不能证明一列或者一些列是候选键。在某一特殊时刻没有出现相同的值的事实并不能保证不存在相同值的可能性。但是，一个示例中相同值的出现可以说明某些列不能联合起来作候选键。标识候选键需要我们知道该列在现实世界中所代表的意义，以此来判断是否可能存在重复值。只有依靠这些语义信息，我们才能确定候选键。

例如，从图2-1的数据中，我们可能认为Staff表合适的候选键是name，雇员的名字。因为在这张表中，只有一个Tom Daniels出现，但是如果有相同名字的成员加入了公司，则将name作为候选键就不对了。

Role

actorNo	catalogNo	character
A1002	207132	James Bond
A3006	330553	Frodo Baggins
A8401	902355	Harry Potter
A2019	634817	Captain Steve Hiller
A2019	445624	Agent J
A7525	445624	Agent K
A4343	781132	Shrek

图2-3　Role表的一个例子

主键（Primary Key）　唯一标识表中记录的候选键。

由于一张表中没有重复记录，因此总是有可能唯一地标识每条记录。这意味着每张表总会有一个主键。最糟的情况是，将表中所有的列组合起来作为主键，但通常更小的子集就完全可以区分记录了。没有选作主键的候选键称为备用键（alternate key）。对于Branch表，如果我们选择branchNo作为主键，则zipCode就是备用键。在Role表中，只有一个候选键，actorNo和catalogNo的组合，所以这两列自然就是该表的主键。

外键（Foreign Key）　一个表中的一个列或多个列的集合，这些列匹配某些其他（也可能是同一个）表中的候选键。

当一列不只出现在一张表中的时候，它的出现通常代表两张表的记录之间的关系。例如，在图2-1中，branchNo在Branch和Staff表中都存在，这是非常有意义的，因为它把分公司和在那里工作的员工联系起来。在Branch表中，branchNo是主键，但在Staff表中，branchNo是外键。我们说Staff表中的branchNo列映射或引用了主表Branch的主键列branchNo。在这种情况下，Staff被称为子表，而Branch表被称为父表。

> 你可以回顾我们在第1章中介绍的DBMS的一个优点是对数据冗余的控制。控制冗余的一个例子就是——在构建关系时，这些普通的列在表达关系时起着非常重要的作用，正如我们将在后续章中看到的。

2.2.4 关系数据库的表示

关系数据库由一个或多个表组成。描述关系数据库通常的惯例是给定每张表的名称，表名后面跟着列名。列名括在圆括号中。通常，主键用下划线表示。StayHome公司的关系数据库的描述是：

Branch	(branchNo,street, city, state, zipCode, mgrStaffNo)
Staff	(staffNo, name, position, salary, branchNo)
Video	(catalogNo, title, category, dailyRental, price, directorNo)
Director	(directorNo, directorName)
Actor	(actorNo, actorName)
Role	(actorNo, catalogNo, character)
Member	(memberNo, fName, lName, address)
Registration	(branchNo, memberNo, staffNo, dateJoined)
RentalAgreement	(rentalNo, dateOut, dateReturn, memberNo, videoNo)
VideoForRent	(videoNo, available, catalogNo, branchNo)

图2-4显示了StayHome数据库的例子。

Branch

branchNo	street	city	state	zipCode	mgrStaffNo
B001	8 Jefferson Way	Portland	OR	97201	S1500
B002	City Center Plaza	Seattle	WA	98122	S0010
B003	16 – 8th Avenue	New York	NY	10012	S0415
B004	16 – 14th Avenue	Seattle	WA	98128	S2250

Staff

staffNo	name	position	salary	branchNo
S1500	Tom Daniels	Manager	46000	B001
S0003	Sally Adams	Assistant	30000	B001
S0010	Mary Martinez	Manager	50000	B002
S3250	Robert Chin	Supervisor	32000	B002
S2250	Sally Stern	Manager	48000	B004
S0415	Art Peters	Manager	41000	B003

Video

catalogNo	title	category	dailyRental	price	directorNo
207132	Die Another Day	Action	3.00	31.99	D1001
002353	Harry Potter	Children	4.50	14.50	D7834
580553	Lord of the Rings	fantasy	5.00	31.99	D4576
781132	Shrek	Children	4.00	18.50	D6078
445624	Men in Black II	Action	4.00	29.99	D5743
634817	Independence Day	Sci-Fi	4.50	32.99	D3745

图2-4 StayHome录像出租数据库的示例

Director

directorNo	directorName
D1001	Lee Tamahori
D7834	Chris Columbus
D4576	Peter Jackson
D9078	Andrew Adamson
D5743	Barry Sonnenfeld
D3761	Roland Emmerich

Actor

actorNo	actorName
A1002	Pierce Brosnan
A1006	Elijah Wood
A2019	Will Smith
A7525	Tommy Lee Jones
A4143	Mike Myers
A8401	Daniel Radcliffe

Role

actorNo	catalogNo	character
A1002	207132	James Bond
A1006	110563	Frodo Baggins
A1006	902563	Harry Potter
A2019	634817	Captain Steve Hiller
A2019	445624	Agent J
A7525	445624	Agent K
A6341	791132	Shrek

Member

memberNo	fName	lName	address
M250178	Bob	Adams	57 - 11th Avenue, Seattle, WA 98103
M166884	Art	Peters	89 Redwood Rd, Portland, OR 97122
M115656	Serena	Parker	22 W. Capital Way, Portland, OR 97201
M284354	Don	Nelson	123 Suffolk Lane, Seattle, WA 98117

Registration

branchNo	memberNo	staffNo	dateJoined
B002	M250178	SA250	1-Jul-01
B001	M166884	S0001	4-Sep-02
B001	M115656	S0003	12-May-00
B002	M284354	S1250	9-Oct-01

RentalAgreement

rentalNo	dateOut	dateReturn	memberNo	videoNo
R753461	4-Feb-03	6-Feb-03	M284354	245456
R753462	4-Feb-03	6-Feb-03	M284354	243431
R868256	5-Feb-03	7-Feb-03	M115656	199004
R868189	5-Feb-03		M115656	178643

VideoForRent

videoNo	available	catalogNo	branchNo
199004	Y	207132	B001
245456	Y	207132	B002
178643	N	634817	B001
243431	Y	634817	B002

图2-4　（续）

2.3　关系完整性

前面，我们讨论了关系数据模型的结构部分。正如我们在2.1节中所提到的，数据模型还有其他两部分：操作部分（定义允许对数据进行的操作）和一组完整性规则（确保数据的正确性）。本节我们讨论关系完整性规则，下一节中我们讨论主要的关系操作语言。

既然每个列都有相关的域，那么就有限制列的取值集合的约束（称为域约束）。此外，还有两个重要的关系完整性规则，即应用于数据库所有实例的约束或限制。关系模型的这两个重要规则就是所说的实体完整性和参照完整性。在我们定义这些术语之前，我们首先需要理解空值的概念。

2.3.1 空值

空值（Null） 表示一个列的值目前还不知道或者对于当前记录来说还不能使用。

空值可以意味着“不知道”，也可以意味着某个记录没有值，或者只是意味着该值还没有提供。空值是处理不完整数据或异常数据的一种方式。但是，空值与数字零或者用空格填充的字符串不同，零和空格是值，但空值代表没有值。因此，空值应该与其他值区别对待。

例如，假定一个分公司暂时没有负责人是可能的，这可能因为负责人最近离开了，而新的负责人还没有上任。这时，相应的mgrStaffNo列的值就是没有定义的。如果没有空值，就必须引入错误的数据来代表这种状态或者增加额外的可能根本对用户没有意义的列。在这个例子中，我们就要用“不存在”这个值来描述没有负责人。或者，我们为Branch表增加新的列“currentManager?”，如果有负责人，值为Y(是)，否则，值为N(否)。这两种方法都会令使用数据库的人困惑。

有了空值的定义，我们现在就开始定义两种关系完整性规则。

2.3.2 实体完整性

第一个完整性规则应用于基本表的主键上。

实体完整性（Entity Integrity） 在一个基本表中，主键列的取值不能为空。

基本表（Base Table） 命名的表，其中的记录物理地存储在数据库中。这与我们在1.2.4节中提到的视图不同。视图是“虚拟的表”，它并不真正存在于数据库中，而是在访问时由DBMS从基本表中产生的。

从前面介绍的定义中，我们知道主键是用于唯一标识记录的最小的标识。这意味着主键的任何子集都不能提供记录的唯一标识。如果我们允许空值作为主键的一部分，则就意味着并不是所有的列都用来区分记录，这与主键的定义矛盾。例如，branchNo是Branch表的主键，我们不应该在插入记录的时候，使branchNo列的值为空。

2.3.3 参照完整性

第二个完整性规则应用于外键上。

参照完整性（Referential Integrity） 如果表中存在外键，则外键值必须与主表中的某些记录的候选键值相同，或者外键的值必须全部为空。

在图2-1中，Staff表中的branchNo是主表Branch的branchNo列的外键。例如，在Staff表中应该不能创建分公司号码为B300的员工号码，除非在Branch表中已经存在号码为B300的分公司。但是，我们应该可以创建分公司号码为空的新的员工记录，即允许发生这样的情况，该新员工已经加入了公司，但还没有分派到某个具体的公司。

2.3.4 业务规则

业务规则（Business Rule） 定义或约束组织的某些方面的规则。

业务规则的例子包括域和关系完整性规则，域用于约束特定列能够取的值，关系完整性规则在前面刚讨论过。另一个例子是多样性，它定义某个实体（比如一个分公司）可能在另一个实体（比如每个雇员）中有若干次出现。用户也可以指定数据必须满足的其他约束。例如，如果StayHome有这样的规则：每个成员一次最多只能借10盘录像，那么用户必须能说明该规则，并希望DBMS实现它。这时，如果某成员当前租的录像的数量是10盘，则该成员就应该不能再租新的了。

遗憾的是，不同的系统对业务规则的支持程度是不同的。我们在第12章和第18章讨论业务规则的实现。

2.4 关系语言

在2.1节中，我们提出了数据模型的一个部分是操作部分，此部分定义了允许对数据进行的操作，包括从数据库中更新或检索数据所用的操作以及改变数据库结构的操作。关系DBMS的这两种主要语言是：

- SQL(Structured Query Language)
- QBE(Query-by-Example)

SQL已经被国际标准化组织（ISO）进行了标准化，使它成为正式的和实际上的定义和操纵关系数据库的标准语言。

QBE是另一种方法，它是基于图形的、“点击”查询数据库的方式，尤其适合于不太复杂并且只用几张表就可以表达的查询。QBE是对于非专业用户从数据库中获得信息的最简单的方式之一。遗憾的是，同SQL不一样，这个语言没有官方标准。但是，开发商所提供的功能通常都很相似，并且这些语言的使用通常比SQL更直观。我们将在下一章对SQL和QBE进行介绍。

2.5 本章小结

- 关系数据库管理系统（RDBMS）目前已经成为主流的DBMS。这个软件代表了第二代DBMS，它基于E.F.Codd博士提出的关系数据模型。
- 关系在物理上描述为表，记录对应每个元组，而列对应属性。
- 关系表的特点是：每个单元只包含一个值、列名唯一、列值来自相同的域、列的顺序不重要、记录的顺序不重要，并且没有重复的记录。
- 超键是唯一标识表中记录的列集合。候选键是最小的超键。主键是选出来唯一标识表中记录的候选键，每个表必须有一个主键。外键就是一个列或多个列，这（些）列是其他表（也可能是本身）中的候选键。
- 空值表示此列的值目前“不知道”，或者对于此记录没有定义。
- 实体完整性是一个约束，表明在基本表中，主键的列不能为空。参照完整性是如果表中存在外键，则外键的值必须与主表中的某些记录的候选键值相同，或者外键的值必须全部为空。
- 访问关系数据库的两种主要语言是SQL和QBE。

复习题

2.1 在关系数据模型语境中讨论下述每个概念。

(a) 关系
(b) 属性
(c) 域
(d) 元组
(e) 关系数据库

2.2 讨论关系表的属性。

2.3 讨论一个表中候选键和主键间的区别，解释外键的含义。表的外键是如何与候选键关联的？用例子来说明你的答案。

2.4 空值的含义是什么？

2.5 为关系模型定义两个主要的完整性规则，讨论为什么强制执行这些规则。

第3章 SQL和QBE

本章主题:

- 关系数据库主要查询语言SQL（Structured Query Language，结构化查询语言）的目的和重要性。
- 如何使用SELECT语句检索数据库中数据。
- 如何使用INSERT语句向数据库中插入数据。
- 如何使用UPDATE语句更新数据库中的数据。
- 如何使用DELETE语句从数据库中删除数据。
- 如何使用CREATE TABLE语句在数据库中创建新表。
- 另一种关系数据库查询语言QBE（Query-by-Example）。

在前面的几章中，我们已经介绍了关系数据模型，并指出在关系数据库管理系统（DBMS）中的两种主要的语言。

- SQL
- QBE

QBE本质上是SQL的图形化的前端，它能提供一种比SQL更简单的查询关系数据库的方法。然而，QBE能够将用图形描述的查询转变成相应的SQL语句，从而在数据库上运行。在本章中，虽然因为SQL的重要地位，我们将主要关注SQL语言，但我们会对这两种语言都进行介绍。如果读者有兴趣查阅有关SQL和QBE更完整的讨论，可以参考2002年Connolly和Begg合作出版的书。

3.1 结构化查询语言

SQL是一种应用最为普遍的商业关系数据库语言，非专业人士和专业人士一样可以很好地使用它。在1974~1977年间，它本是IBM的San Jose研究实验室在SEQUEL和System-R项目开发的一部分。如今，尽管官方规定SQL应读作“S-Q-L”，但仍有很多人将它读作“See Quel”。从20世纪70年代末期Oracle开始，已经出现了很多基于SQL的商业关系数据库管理系统（RDBMS）。在ANSI（美国国家标准化组织）和ISO（国际标准化组织）制定了SQL标准后，SQL已经成为现在定义和操纵关系数据库的正式的事实上的语言了。

SQL的主要特点有:

- 学习起来相对容易。
- 它是一种非过程化语言：你只需指定需要什么数据而不需要指定怎样得到这些数据，也就是说，SQL不需要你指定访问数据的方法。
- 与大多数现代语言一样，SQL的语言格式自由，即语句并不需要在屏幕的特定位置输入。
- 其命令结构由标准英语单词组成，如SELECT、INSERT、UPDATE和DELETE。
- 它可以被很多用户使用，包括数据库管理员（DBA）、管理层职员、应用程序员以及其他最终用户。

SQL语言非常重要，其原因如下：

- SQL是第一种（也是到目前为止唯一的一种）能够被广泛采纳的标准数据库语言。当前，几乎任何一个重要的开发商都提供基于SQL或拥有SQL接口的数据库产品，而且大多数开发商至少代表了一部分制定标准的主体。
- SQL语言为开发商和用户都提供了无限的商机。它成为如IBM SAA（IBM Systems Application Architecture，IBM系统应用体系结构）之类的应用体系结构的一部分，并且成为很多大规模、有影响力的组织（如UNIX标准的X/OPEN联盟）的战略选择。
- SQL也成为Federal Information Processing Standard（FIPS，信息处理标准联盟），使卖给US政府的所有DBMS都保持一致。
- SQL还被应用在其他的标准中，甚至作为一种定义工具影响着其他标准的开发（比如，ISO远程数据访问（RDA）标准。）

在我们深入学习SQL例子之前，先介绍一下SQL的目标。

3.1.1 SQL的目标

理想情况下，一种数据库语言应该允许用户进行以下操作：

- 创建数据库和表结构。
- 实现基本的数据管理工作，比如插入、修改或删除表中数据。
- 能够实现简单和复杂的查询。

另外，一种数据库语言首先必须以用户最小的代价执行上述工作；其次，它的命令结构和语法都必须易于学习；最后，它必须是可移植的：即它必须符合一些已被承认的标准。只有这样，当我们需要把数据从一个DBMS系统转移到另外一个DBMS系统中时，才能延续使用与原来相同的命令结构和语法。SQL是符合以上要求的一种数据库语言。

SQL是一个面向转换（transform-oriented）的语言的例子，或者说是一种用于把数据从输入表转换到要求的输出表而设计的一种语言。ISO SQL标准包含两个主要组成部分：

- 用于定义数据库结构和控制数据存取的数据定义语言（DDL）。
- 用于检索和更新数据的数据操纵语言（DML）。

在1999年ISO SQL标准的最新版本（即常说的SQL3）发布之前，SQL仅包含这些定义命令和操纵命令，不包含流程控制命令，如IF...THEN...ELSE、GO TO 或DO...WHILE。这些都需要通过程序或作业控制语言，或终端用户的交互来实现。由于缺少最基本的计算功能，SQL有两种使用方法。第一种方法是通过终端交互地输入语句来使用SQL。第二种方法是在过程化语言中嵌入SQL语句。在本书中，我们只考虑交互式SQL。若想了解嵌入式SQL的详细情况，可以参考2002年Connolly和Begg合作出版的书。

> **SQL一致性**（SQL Conformance） SQL3有一系列称之为核心SQL的特性，开发商必须遵照SQL3标准实现这些特性。其他的很多特性被划分到包中；比如，对象特征包和OLAP（联机分析处理）包。开发商往往还会实现一些附加的功能，尽管这确实会影响SQL的可移植性。

3.1.2 术语

ISO SQL标准没有使用关系、属性和元组这些形式化的术语，而是采用表、列和行这些

术语。我们对SQL的介绍大多都采用ISO术语。需要特别指出的是，SQL并没有严格遵守第2章描述的关系模式的定义。例如，SQL允许SELECT操作产生的结果表包含重复行；它可以按表中的列进行排序；而且，它允许用户定义表中行的排列顺序。

3.1.3 书写SQL命令

在这一节中，我们将简要描述SQL语句的结构和用来定义各种SQL结构的格式的符号。一个SQL语句由保留字和用户定义的词组成。保留字是SQL语言的固定部分，而且有它固定的语义。它们必须与要求的拼写完全一致，而且不能跨行写。用户定义的词是由用户创造的（符合一定的语法规则），代表各种数据库对象，如表、列、视图、索引等等。在本章中，我们用大写字母代表保留字，小写字母代表用户定义的单词。

SQL语句一般都不区分大小写，也就是说，字母可以用大写也可以用小写。该规则的唯一特例是，字符数据的拼写必须和它在数据库中的拼写完全一致。例如，如果我们在数据库中把一个人的姓存为“SMITH”，然后又用字符串“Smith”进行查找，则该行将无法找到。语句中的词也根据一组语法规则来建立。尽管标准没有要求，但是很多SQL的变种都要求使用语句结束符来标志每个SQL语句的结束（一般都用分号“；”）。

在本章中，我们用下列BNF符号的扩展形式来定义SQL语句:

- 垂直线（|）表示在其中任选一个。例如，a | b | c。
- 大括号表示必需的元素。例如，{a}。
- 方括号代表可选元素。例如，[a]。
- 省略号（…）用来代表某一项可以重复0到多次。

例如，{a|b}（，c…）表示a或b后面跟随0个或多个由逗号隔开的c。

实际上，DDL语句用于创建数据库的结构（即表）和访问机制（即每位用户能够进行的合法访问），而DML语句是用于填充和查询表的。但在本书中，我们主要关注DML语句，以反映它们对于一般用户的相对重要性。

3.2 数据操纵

在这一节中，我们将研究SQL的 DML语句，即:

- SELECT用来查询数据库中的数据。
- INSERT用来向表中加入数据行。
- UPDATE用来更新表中的数据。
- DELETE用来从表中删除数据。

因为SELECT语句比较复杂而其他DML语句相对容易，所以我们将花大部分篇幅来讲述SELECT语句和它的各种形式。我们将从简单的查询开始，然后提高难度，说明如何产生带有排序（sort）、分组（group）、聚合（aggregate）功能的查询，以及涉及多表的查询。然后，我们将考虑INSERT、UPDATE和DELETE语句。

我们用图2-4中所示的StayHome示例数据库来举例说明SQL语句。该数据库主要包括下列表:

```
Staff       (staffNo, name, position, salary, branchNo)
Video       (catalogNo, title, category, dailyRental, price, directorNo)
Director    (directorNo, directorName)
Actor       (actorNo, actorName)
```

```
Role                (actorNo, catalogNo, character)
RentalAgreement     (rentalNo, dateOut, dateReturn, memberNo, videoNo)
VideoForRent        (videoNo, available, catalogNo, branchNo)
```

文字

在我们讨论SQL DML语句之前，我们有必要理解文字的概念。文字是在SQL语句中使用的常量，SQL支持的任何一种数据类型都有不同的文字形式。但是，为简化起见，我们可以将它们分为包含在单引号之中的文字和不包含在单引号中的两类。所有非数字型的数据值必须包含在单引号之中，所有数字型的数据值必须不加单引号。例如，我们用文字向一张表中加入数据:

```
INSERT INTO Video (catalogNo, title, category, dailyRental, price, directorNo)
VALUES ('207132', 'Die Another Day', 'Action', 5.00, 21.99, 'D1001');
```

列dailyRental和price属于数字文字，它们没有单引号；而其他的列都是字符串型的，并且列值都包含在单引号之中。

3.2.1 简单查询

SELECT语句的目的是从一个或多个数据库表中检索和显示数据，这是一个功能非常强大的命令，也是最常使用的SQL命令。SELECT语句的一般格式如下:

```
SELECT          [DISTINCT | ALL] {* | [columnExpression [AS newName]][,...]}
FROM            TableName [alias][,...]
[WHERE          condition]
[GROUP BY       columnList] [HAVING condition]
[ORDER BY       columnList]
```

- columnExpression代表一个列名或一个表达式。
- newName是你给定的用来显示的列标题名称。
- TableName是想访问的已存在的数据库表或视图的名称。
- alias是TableName的可供选择的缩写名称。

组成SELECT语句的子句有:

FROM	指定用到的一个或多个表
WHERE	按照某些条件过滤行数据
GROUP BY	按相同的列值将行分成组
HAVING	按照某些条件过滤组数据
SELECT	指定在输出结果中出现的列
ORDER BY	指定输出的排列顺序

SELECT语句中各子句的顺序不能改变。其中只有前两个子句是必需的：SELECT和FROM；剩下的子句都是可选的。每个SELECT语句将产生一个查询结果表，它由一个或多个列以及0行或多行结果组成。

查询3.1 检索所有的行和列

列出所有录像的全部详细情况

由于在该查询中没有限制条件（即我们将列出Video表中的所有行），所以不需要WHERE子句。我们可以编写以下查询:

```
SELECT catalogNo, title, category, dailyRental, price, directorNo
FROM Video;
```

当希望列出一个表中的所有列时，可以用星号（*）代替列名。因此，上面的查询也可以写为:

```
SELECT *
FROM Video;
```

上述任何一种方法都可以得到表3-1所示的结果表。

表3-1 查询3.1的结果表

catalogNo	title	category	dailyRental	price	directorNo
207132	Die Another Day	Action	5.00	21.99	D1001
902355	Harry Potter	Children	4.50	14.50	D7834
330553	Lord of the Rings	Fantasy	5.00	31.99	D4576
781132	Shrek	Children	4.00	18.50	D0078
445624	Men in Black II	Action	4.00	29.99	D5743
634817	Independence Day	Sci-Fi	4.50	32.99	D3765

查询3.2 查询指定的列、所有行

列出所有录像的种类号、名称和日租金额

在本查询中也没有指定限制条件，所以也不需要WHERE子句。但是，我们只希望列出全部列的一个子集，我们可以编写以下查询:

```
SELECT catalogNo, title, dailyRental
FROM Video;
```

表3-2显示了该查询的结果表。注意，在这个结果中你会发现除非指定排列顺序，否则结果表不会对行进行排序。我们将在下一节介绍怎样对结果表中的行进行排序。

表3-2 查询3.2的结果表

catalogNo	title	dailyRental
207132	Die Another Day	5.00
902355	Harry Potter	4.50
330553	Lord of the Rings	5.00
781132	Shrek	4.00
445624	Men in Black II	4.00
634817	Independence Day	4.50

查询3.3 使用DISTINCT

列出所有录像的种类号

```
SELECT catalogNo
FROM Video;
```

结果表如表3-3a所示。注意，在结果表中有一些重复的值（默认情况下，SELECT语句不会将重复值去掉）。若要去掉重复值，可以用DISTINCT关键字重写上述查询:

```
SELECT DISTINCT catalogNo
FROM Video;
```

这样，我们将得到表3-3b所示的结果。

表 3-3

a) 查询3.3包含重复值的结果	b) 略去重复值的查询3.3的结果
category	category
Action	Action
Children	Children
Fantasy	Fantasy
Children	Sci-Fi
Action	
Sci-Fi	

查询3.4 计算字段

列出租借录像三天的租金

```
SELECT catalogNo, title, dailyRental*3
FROM Video;
```

这个查询和查询3.2很相似，不同的是，我们将计算三天的租金而不只是一天的租金。在这种情况下，我们可以通过将每天的租金乘3得到三天的租金，查询结果见表3-4。

这是使用计算字段（有时也称为派生的字段）的例子。一般来说，要使用计算字段，你可以在SELECT列表中指定一个SQL表达式。一个SQL表达式可以包含加、减、乘、除运算，而且还可以使用圆括号构建复杂的表达式。可以在计算列中使用多个表中的列，但是，在算术表达式中引用的列必须都是数值类型的。

在该结果表中，第三列显示为col3。通常情况下，结果表中的列与检索的数据库表中相应的列有相同的名字。但是，在这种情况下，SQL不知该如何标识该列。有些系统根据该列在表中的位置取名（如col3），也有些系统会将该列名置空格或用SELECT语句中的表达式作列名。SQL标准允许用AS子句来为列命名。对于前面的查询，我们可以写作：

```
SELECT catalogNo, title, dailyRental*3 AS threeDayRate
FROM Video;
```

这样，在结果表中列的标题就是threeDayRate而不是col3。

表3-4 查询3.4的结果表

catalogNo	title	col3
207132	Die Another Day	15.00
902355	Harry Potter	13.50
330553	Lord of the Rings	15.00
781132	Shrek	12.00
445624	Men in Black II	12.00
634817	Independence Day	13.50

3.2.2 选择行

前面的例子说明了如何用SELECT语句从表中检索所有行。但是，我们经常需要限制要检索的行。这可以通过WHERE子句来实现，WHERE子句由关键字WHERE和指定需要检索的行的查询条件组成。下面列出了五种基本的查询条件（在ISO术语中称为“谓词”）：

- 比较：比较一个表达式值与另一个表达式的值。
- 范围：检验一个表达式的值是否在指定的值范围内。

- 集合成员：检查一个表达式的值是否等于集合中的某个值。
- 模式匹配：检查一个字符串是否匹配一个指定的模式。
- 空值（null）：检查列中是否有空值（未知值）。

下面将举例说明这些查询条件的使用。

查询3.5 比较查询条件

列出所有年薪高于$40000的员工的信息

```
SELECT staffNo, ncome, position, salary
FROM Staff
WHERE salary > 40000;
```

在该查询中，我们需要将Staff表中salary列的值大于$40000的行选择出来。为达到该目的，我们指定WHERE子句的条件（谓词）为“salary > 40000”。结果表如表3-5所示。

表3-5 查询3.5的结果表

staffNo	name	position	salary
S1500	Tom Daniels	Manager	46000
S0010	Mary Martinez	Manager	50000
S2250	Sally Stern	Manager	48000
S0415	Art Peters	Manager	41000

在SQL中，下列简单的比较运算符是有效的：

= 等于　　<> 不等于

< 小于　　<= 小于等于

> 大于　　>= 大于等于

利用AND、OR和NOT逻辑运算符可以生成更复杂的谓词，可以用圆括号（如果需要的话）表示运算的先后顺序。条件表达式的运算规则如下：

- 表达式从左往右计算。
- 先计算圆括号中的子表达式。
- NOT的优先级比AND和OR高。
- AND的优先级比OR高。

通常推荐使用圆括号来改变运算顺序，因为它可以除去可能的二义性。

查询3.6 范围查询条件（BETWEEN / NOT BETWEEN）

列出所有年薪在$45000和$50000之间的员工信息

```
SELECT staffNo, ncome, position, salary
FROM Staff
WHERE salary >= 45000 AND salary <= 50000;
```

在这个查询中，我们在WHERE子句中使用逻辑运算符AND来查找Staff表中salary列值在$45000和$50000之间的行。结果如表3-6所示。SQL也提供了范围测试BETWEEN来检测一个数据值是否在指定的一对值之间。我们可以将前面的查询重写为：

```
SELECT staffNo, ncome, position, salary
FROM Staff
WHERE salary BETWEEN 45000 AND 50000;
```

BETWEEN测试包含边界两端的值，所以年薪为$45000或$50000的工作人员也将包含在结果表中。还有一个否定版本的范围测试（NOT BETWEEN）可以用来检测在该范围之外的值。BETWEEN测试并没有增强SQL表达式的表达能力，就像我们看到的，它的表达能力和两个比较测试的表达能力相同。

表3-6 查询3.6的结果表

staffNo	name	position	salary
S1500	Tom Daniels	Manager	46000
S0010	Mary Martinez	Manager	50000
S2250	Sally Stern	Manager	48000

查询3.7 集合成员查询条件（IN / NOT IN）

列出动作类和儿童类的所有录像

```
SELECT catalogNo, title, category
FROM Video
WHERE category = 'Action' OR category = 'Children';
```

如前所示，我们可以在WHERE子句中用一个复合的查询条件来表达该查询。结果如表3-7所示。但是，SQL还提供了一个集合成员关键字IN来测试列值是否匹配值列表中的某个值。我们可以用IN测试重新表述该查询为:

```
SELECT catalogNo, title, category
FROM Video
WHERE category IN ( 'Action' , 'Children');
```

还有一个否定版本的集合成员查询条件（NOT IN）可以用来检测数据值是否在所列列表之外。与BETWEEN相似，IN测试也没有增强SQL表达式的表达能力，但是，IN测试为我们表达查询条件提供了一种更有效的表达方法，尤其是当该集合包含多个值时。

表3-7 查询3.7的结果表

catalogNo	title	category
207132	Die Another Day	Action
902355	Harry Potter	Children
781132	Shrek	Children
445624	Men In Black II	Action

查询3.8 模式匹配查询条件（LIKE / NOT LIKE）

列出所有名为“Sally”的员工信息

SQL有以下两种模式匹配符号:

% 百分号代表0个或多个字符序列（通配符）。

_ 下划线代表一个字符。

在模式中的其他字符都代表其本身，比如:

- name LIKE 'S%' 代表第一个字母必须为S，但是字符串其他部分内容不限。
- name LIKE 'S_ _ _ _' 代表第一个字母必须为S，且字符串必须为5个字母。
- name LIKE '%S' 代表任何长度的字母序列，序列长度至少为1，且最后一个字母必须为S。

- name LIKE '%Sally%' 代表任何内容中包含Sally的字符串。
- name NOT LIKE 'S%' 代表第一个字母不是S的任意字符串。

如果查询字符串中包含模式匹配字符，我们可以用ESCAPE（转义字符）来表示这是模式匹配标识符。比如，检验字符串'15%'，我们可以用谓词：

```
LIKE '15#%' ESCAPE '#'
```

利用SQL的模式匹配查询条件，我们可以用以下查询语句找到所有名为Sally的员工：

```
SELECT staffNo, ncome, position, salary
FROM Staff
WHERE name LIKE 'Sally %';
```

结果如表3-8所示。

表3-8 查询3.8的结果表

staffNo	name	position	salary
S0003	Sally Adams	Assistant	30000
S2250	Sally Stern	Manager	48000

注意 有些RDBMS（如Microsoft Access）用通配符“*”和“?”来替代“%”和“_”。

查询3.9 NULL查询条件（IS NULL / IS NOT NULL）

列出没有按期归还的录像

RentalAgreement表中有一列dateReturn代表归还录像的日期。也许你会认为我们可以用以下查询条件找到这类录像：

```
WHERE (dateReturn = '' OR dateReturn=0)
```

但是，任何一个条件都无法完成以上工作。dateReturn返回null可能代表一个未知值，所以我们无法测试它是否等于或不等于另一个值。若我们试图用该复合条件的任何一个条件执行SELECT语句，我们将得到一张空结果表。我们需要用关键词IS NULL来显式地测试NULL。

```
SELECT dateOut, memberNo, videoNo
FROM RentalAgreement
WHERE dateReturn IS NULL;
```

结果如表3-9所示，否定版本（IS NOT NULL）可以用来测试不是空值的列。

表3-9 查询3.9的结果表

dateOut	memberNo	videoNo
2-Feb-03	M115656	178643

3.2.3 给结果排序

一般来说，SQL查询产生的结果表的行并没有按任何特定的要求排序（尽管有些DBMS可能使用一个默认的顺序，例如，基于主键的排序）。但是，我们可以在SELECT语句中使用ORDER BY子句来保证一个查询的结果被排序。ORDER BY子句由一组用逗号分隔的要在结果表中排序的列名组成。ORDER BY子句允许检索得到的行基于任一列或列的组合以升序

（ASC）或降序（DESC）排列，而不考虑这些列是否出现在结果中。但是，SQL的有些版本坚持ORDER BY元素需要出现在SELECT列表中。但不管是哪种情况，ORDER BY子句必须是SELECT语句的最后一个子句。

查询3.10 给结果排序

列出所有录像，按价格降序排列

```
SELECT *
FROM Video
ORDER BY price DESC;
```

该查询与查询3.1类似，只是增加了将结果表按price列值排序的条件。这通过在SELECT语句的末尾加上ORDER BY子句来实现，指定price列为排序的列，DESC表明以降序排列。在这种情况下，我们得到了表3-10所示的结果。

如果我们得到多个price列值相同的数据，我们可能需要将结果首先以price（主排序键）排序，然后再以title（次排序键）列升序排列。在这种情况下，ORDER BY子句就变成：

```
ORDER BY price DESC, title ASC;
```

表3-10 查询3.10的结果表

catalogNo	title	category	dailyRental	price	directorNo
634817	Independence Day	Sci-Fi	4.50	32.99	D3765
330553	Lord of the Rings	Fantasy	5.00	31.99	D4576
445624	Men In Black II	Action	4.00	29.99	D5743
207132	Die Another Day	Action	5.00	21.99	D1001
781132	Shrek	Children	4.00	18.50	D0078
902355	Harry Potter	Children	4.50	14.50	D7834

3.2.4 使用SQL的聚合函数

ISO标准定义了五个聚合函数：

COUNT	返回指定列的值的个数
SUM	返回指定列的值的总和
AVG	返回指定列的平均值
MIN	返回指定列的最小值
MAX	返回指定列的最大值

这些函数在表中的一个列上操作，并且返回单一值。COUNT、MIN和MAX可以用到数值和非数值型字段上，但SUM和AVG只能用到数值型字段上。除了COUNT(*)以外，每个函数都首先忽略NULL值，然后再对剩下的非空值进行操作。COUNT(*)是COUNT的特殊用法，它统计表中的所有行，不管这些行是否是空值或是出现重复值。

如果希望在应用函数之前先去掉重复列，可以在函数中的列名前加上关键字DISTINCT。DISTINCT对于MIN和MAX函数来说没有作用。但是，对于SUM和AVG的结果会有所影响，所以，是否需要在计算中包括或排除重复值需要仔细考虑。另外，在一个查询中DISTINCT只能指定一次。

要特别注意的是，聚合函数只能在SELECT列表和HAVING子句中使用。在任何别的地方使用都是不正确的。如果SELECT列表中包含了聚合函数并且没有使用GROUP BY子句，那么SELECT列表中的项不能包含对任何列的引用，除非该列是聚合函数的参数。例如，以下查询是不合法的：

```
SELECT staffNo, COUNT(salary)
FROM Staff;
```

因为这个查询没有GROUP BY子句，并且SELECT列表中的staffNo列在聚合函数之外使用。

查询3.11 使用COUNT和SUM

列出年薪高于$40000的员工总数和他们的年薪总和

```
SELECT COUNT(staffNo) AS totalStaff , SUM(salary)AS totalSalary
FROM Staff
WHERE salary>40000;
```

该查询的WHERE子句和查询3.5类似，但是，在这里我们用COUNT函数来计算满足WHERE子句的行数，用SUM函数来计算这些行的年薪总和。结果表如表3-11所示。

表3-11 查询3.11的结果表

totalStaff	totalSalary
4	185000

查询3.12 使用MIN、MAX和AVG

列出员工年薪的最小值、最大值和平均值

```
SELECT MIN(salary) AS minSalary, MAX(salary) AS maxSalary,
       AVG(salary) AS avgsalary
FROM Staff;
```

在该查询中，我们需要考虑所有员工的数据，所以不需要WHERE子句。要求的值可以通过使用MIN、MAX和AVG函数来计算。该查询的结果如表3-12所示。

表3-12 查询3.12的结果表

minSalary	maxSalary	avgSalary
30000	50000	41166.67

3.2.5 对结果分组

前面的查询相当于一篇报告最后的总结，它们将报告中所有的细节数据浓缩成了一行概要数据。但是，在报告中往往还需要小计。我们可以使用SELECT语句中的GROUP BY子句来进行小计。一个包含了GROUP BY子句的查询被称为分组查询，因为它将SELECT表中的数据进行分组并为每个分组产生了一个概要行。在GROUP BY子句中的列被称为分组列。ISO标准要求SELECT子句和GROUP BY子句紧密结合在一起。当使用了GROUP BY子句后，SELECT列表中的每一项在每个分组中必须都是单值的。而且，SELECT子句只能包含：

- 列名。
- 聚合函数。
- 常量。
- 一个包含上述各项组合的表达式。

在SELECT列表中的所有列名都必须出现在GROUP BY子句中，除非该名称只在聚合函数中使用。反之则不然，在GROUP BY子句中出现的列名则不一定出现在SELECT列表中。当WHERE子句

和GROUP BY子句同时使用时，WHERE子句先被应用，然后在剩下的满足条件的行之上进行分组。

> ISO标准在使用GROUP BY时认为两个空值相等。如果两行在相同的分组列上都是空值，而且在其他非空分组列上值都相同，就将它们归入同一分组中。

查询3.13 使用GROUP BY

找出每个分公司的员工总数和他们的年薪总和

```
SELECT branchNo, COUNT(staffNo) AS totalStaff ,
       SUM(salary) AS totalSalary
FROM Stciff
GROUP BY branchNo
ORDER BY branchNo;
```

没有必要在GROUP BY列表中包含staffNo和salary列名，因为它们只在SELECT列表中的聚合函数中出现。另外，branchNo与聚合函数没有关联，所以必须在GROUP BY列表中出现。结果如表3-13所示。

表3-13 查询3.13的结果表

branchNo	totalStaff	totalSalary
B001	2	76000
B002	2	82000
B003	1	41000
B004	1	48000

在概念上，SQL按如下说明完成该查询。

1) SQL根据员工的分公司号对员工进行分组。在每个组中，所有的员工有相同的分公司号。在该例子中，我们将得到四个分组。

branchNo	staffNo	salary	COUNT(staffNo)	SUM(salary)
B001	S1500	46000	2	76000
B001	S0003	30000		
B002	S0010	50000	2	82000
B002	S3250	32000		
B003	S0415	41000	1	41000
B004	S2250	48000	1	48000

2) 针对每个组，SQL计算员工的数目，并计算salary列值的总和来得到年薪总和。SQL在查询结果中为每个组产生一个概要行。

3) 最后，结果将结果按分公司号（branchNo）升序排列。

有限制的分组（HAVING子句）

HAVING子句和GROUP BY子句同时作用来限制在结果表中出现的分组。尽管HAVING和WHERE子句在语法上相似，但是它们的目的不同。WHERE子句过滤进入结果表的单个行，而HAVING子句将过滤进入结果表的整个分组。ISO标准要求在HAVING子句中使用的列名必须在GROUP BY列表中出现，或者被包含在聚合函数中。事实上，HAVING子句的查询条件常常至少包含一个聚合函数，否则查询条件就可以移到WHERE子句并应用到单行中。

(切记聚合函数不可在WHERE子句中使用。)

HAVING子句不是SQL的必要部分——任何用HAVING子句描述的查询都可以用不包含HAVING子句的语句来代替。

查询3.14 使用HAVING

对每个分公司，如果有多名员工，则找出这些分公司的员工数和工资总额

```
SELECT        branchNo, COUNT(staffNo) AS totalStaff,
              SUM(salary) AS totalSalary
FROM          Staff
GROUP BY      branchNo
HAVING COUNT(staffNo) > 1
ORDER BY branchNo;
```

这和前面的例子有相似之处，只是增加了限制条件：我们需要考虑那些员工数多于一个人的分组(分公司)。这个限制应用到分组中，所以用了HAVING子句。结果如表3-14所示。

表3-14 查询3.14结果表

branchNo	totalStaff	totalSalary
B001	2	76000
B002	2	82000

3.2.6 子查询

在这一节中，我们将考虑一个嵌入到另一个SELECT语句中的完整的SELECT语句。内层的SELECT语句（称为子查询）的结果被用在外层的SELECT语句中，帮助决定结果的内容。一个子查询可以用在外层SELECT语句的WHERE和HAVING子句中，被称为子查询或嵌套查询。子查询也可出现在INSERT、UPDATE和DELETE语句中。

查询3.15 使用子查询

找出在位于“8 Jefferson Way”的分公司工作的员工

```
SELECT staffNo, name, position
FROM Staff
WHERE branchNo = (SELECT branchNo
                  FROM Branch
                  WHERE street= '8 Jefferson Way');
```

内层的SELECT语句（SELECT branchNo FROM Branch ……）查找出在街道“8 Jefferson Way”的分公司的分公司号（结果将只有一个分公司号），得到该分公司号后，外层SELECT语句将检索出在该分公司工作的所有员工的详细信息，也就是说，内层的SELECT返回的结果表只包含值“B001”，对应于位于“8 Jefferson Way”的分公司。外层的SELECT语句就变成了：

```
SELECT staffNo, name, position
FROM Staff
WHERE branchNo = 'B001';
```

此查询的结果表如表3-15所示。

表3-15 查询3.15结果表

staffNo	name	position
S1500	Tom Daniels	Manager
S0003	Sally Adams	Assistant

我们可以把子查询看作是生成一个临时表，临时表的结果可以被外层语句存取和使用。子查询可以直接在WHERE或HAVING子句的关系运算符（如，=、<、>、<=、>=、<>）的后面使用。子查询常常被写在圆括号内。

> 注意，如果内层查询的结果有多行，那么我们必须使用集合成员测试IN而不是等于测试（“=”）。比如，如果我们希望找到在华盛顿（WA）的分公司工作的所有员工，WHERE子句就应该变为：
>
> ```
> WHERE branchNo IN ( SELECT branchNo FROM Branch
>                     WHERE state='WA');
> ```

查询3.16 使用包含聚合函数的子查询

列出年薪高于平均年薪的所有员工

```
SELECT staffNo, name, position
FROM Staff
WHERE salary > (SELECT AVG(salary)
                FROM Staff);
```

回顾我们在3.2.4节中讲到的，聚合函数只能用在SELECT列表和HAVING子句中。所以写“WHERE salary > AVG(salary)”这样的语句是不正确的。我们需要使用子查询找出平均年薪，然后使用外层SELECT语句来找出所有年薪高于平均年薪的员工。换句话说，子查询返回平均年薪$41166.67，外层查询就可以写为：

```
SELECT staffNo, name, position
FROM Staff
WHERE salary > 41166.67;
```

结果如表3-16所示。

表3-16 查询3.1结果表

staffNo	name	position
S1500	Tom Daniels	Manager
S0010	Mary Martinez	Manager
S2250	Sally Stern	Manager

> 注意应用在子查询中的下述规则：
>
> 1）ORDER BY子句不能用在子查询中，尽管最外层SELECT语句可能会使用该子句。
>
> 2）子查询SELECT列表必须由一列列名或是一个表达式组成，除非子查询使用关键字EXISTS——见2002年Connolly和Begg合作出版的书。
>
> 3）默认时，子查询中的列名是子查询FROM子句中的表中存在的列名。通过限定列名也可以引用外层查询的FROM子句中的表。
>
> 4）当子查询是比较运算中涉及的两个操作数中的一个时，子查询必须出现在比较操作符的右侧。例如，最后的例子如果写成下面的形式就是不正确的。
>
> ```
> SELECT staffNo, name, position
> FROM Staff
> WHERE (SELECT AVG(salary) FROM Staff) < salary;
> ```
>
> 因为在上述表达中，子查询出现在了比较表达式的左侧。

3.2.7 多表查询

到目前为止，我们所考虑的例子都有一个限制：在结果表中出现的列都来自一个表。但在很多情况下结果表中的列不可能只来自一个表。为了将几个表中的列都加入到结果表中，我们需要使用连接（join）操作。SQL的连接操作通过两表之间的相关行将来自两个表中的信息组合起来。这个联接表中的相关行对是两个表中匹配的列具有相同值的行。

如果需要得到来自多个表的信息，就可以选择使用子查询或是连接操作。如果希望在最终的结果表中得到来自不同表的列，那么必须使用连接操作。要执行连接操作，仅需在FROM子句中加上多个表名，用逗号隔开，并在WHERE子句中指定要连接的列。而且可以在FROM子句中使用表的别名。在这种情况下，别名可以跟在表名后面用一个空格隔开。在列名不明确的情况下，别名可以用来指定列名的出处，它也可以作为表名的一个简称。一旦为表取了别名后，别名可以出现在任何可以出现表名的地方，作为表名的替换。

查询3.17 简单连接

列出所有录像及它们的导演

```
SELECT catalogNo, title, category, v.directorNo, directorName
FROM Video v, Director d
WHERE v.directorNo = d.directorNo;
```

我们需要从Video表和Director表中获得详细信息，所以我们需要使用连接操作。SELECT子句列出了需要显示的列。为了得到需要的行，我们使用查询条件v.directorNo = d.directorNo来得到两个表中directorNo列值相同的行。我们将这两列称为两表的匹配列。结果表如表3-17所示。

> 注意，在SELECT列表中限制导演号directorNo是必要的。因为directorNo可以来自于任何一张表，所以我们需要指定显示的是哪一个表。（我们也可以选择Director表中的directorNo列。）该限制通过在列名前加一个相应表名（或者别名）作为前缀。在这个例子里，我们使用Video表的别名v作为前缀。

表3-17 查询3.17结果表

catalogNo	title	category	v.directorNo	directorName
207132	Die Another Day	Action	D1001	Lee Tamahori
902355	Harry Potter	Children	D7834	Chris Columbus
330553	Lord of the Rings	Fantasy	D4576	Peter Jackson
781132	Shrek	Children	D0078	Andrew Adamson
445624	Men In Black II	Action	D5743	Barry Sonnenfeld
634817	Independence Day	Sci-Fi	D3765	Roland Emmerick

最常见的多表查询是有一对多关系的两个表。前面的涉及录像和导演的查询就是这样的一个例子。任何一个导演可以导演一部或多部电影。在2.2.5节中，我们讲述了在关系数据库中如何描述候选键和外键关系的知识。要在SQL查询中使用关系，我们通过将候选键之一（通常是主键）和相应的外键进行比较来指定查询条件。在查询3.16中，我们将Video表中的外键v. directorNo和Director表中的主键d. directorNo进行比较。

SQL标准是用下面的替代方法来指定连接条件：

```
FROM Video v JOIN Director d ON v. directorNo = d. directorNo;
FROM Video JOIN Director USING directorNo;
FROM Video NATURAL JOIN Director;
```

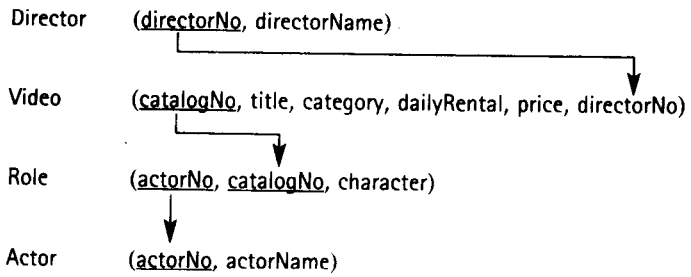
在任何一种情况下，都是用FROM子句替代原先的FROM和WHERE子句。然而，第一种替代方案产生了一张有两个相同directorNo列的表，而剩下的两种方案将产生只包含一个directorNo列的表。

查询3.18 四表连接

列出所有录像以及它们的导演、演员名单和相应的角色

```
SELECT v.catalogNo, title, category, directorName, actorName, character
FROM Video v, Director d, Actor a, Role r
WHERE d.directorNo = v.directorNo AND
      v.catalogNo = r.catalogNo AND
      r.actorNo = a.actorNo;
```

在这个例子中，我们想要显示来自Video、Director、Actor和Role表中的详细信息，所以我们使用连接操作。SELECT子句列出了要显示的列。为了得到需要的行，我们应该基于各种匹配列（即主键/外键）来连接表，如下所示:



结果表如表3-18所示。

表3-18 查询3.18结果表

catalogNo	title	category	directorName	actorName	character
207132	Die Another Day	Action	Lee Tamahori	Pierce Brosnan	James Bond
902355	Harry Potter	Children	Chris Columbus	Daniel Radcliffe	Harry Potter
330553	Lord of the Rings	Fantasy	Peter Jackson	Elijah Wood	Frodo Baggins
781132	Shrek	Children	Andrew Adamson	Mike Myers	Shrek
445624	Men In Black II	Action	Barry Sonnenfeld	Will Smith	AgentJ
445624	Men In Black II	Action	Barry Sonnenfeld	Tommy Lee Jones	Agent K
634817	Independence Day	Sci-Fi	Roland Emmerick	Will Smith	Captain Steve Hiller

3.2.8 INSERT、UPDATE和DELETE语句

SQL是一种完整的数据操纵语言，我们可以用它来查询数据库，也可以用它来修改数据库中的数据。修改数据库的命令不像SELECT语句那样复杂。在这一节中，我们将详细介绍SQL的三种用来修改数据库中表内容的语句。

- INSERT 向表中插入新的数据。
- UPDATE 修改表中的数据。

• DELETE 从表中删除数据行。

1. 插入语句（INSERT语句）

INSERT语句的一般格式为:

```
INSERT INTO TableName [(columnList)]
VALUES (dataValueList)
```

TableName是基本表的表名，columnList代表一组由逗号分隔的一个或多个列名。columnList是可选的，如果省略它，SQL将假定所有列按照它们在建表（CREATE TABLE）定义的顺序排列。如果指定了columnList，那么所有在列表中省略了的列必须在表创建时被指定为可以为空，除非在创建表时指明该列可以使用默认值DEFAULT（见3.3.1节）。dataValueList必须按下列规则与columnList匹配:

• 两个列表中的项数必须一样。
• 在两个列表中，各项的位置必须直接对应，因此dataValueList中的第一项将用在columnList的第一项中，dataValueList中的第二项将用在columnList的第二项中，以此类推。
• dataValueList中每一项的数据类型必须与相应的列的数据类型兼容。

查询3.19 向表中插入一行

向Video表中插入一行

```
INSERT INTO Video
VALUES ('207132', 'Die Another Day', 'Action', 5.00, 21.99, 'D1001');
```

在这个特殊的例子中，我们按照创建表时指定的列的顺序为所有列提供了列值。因此，我们可以省略列名列表。

2. 更新语句（UPDATE语句）

UPDATE语句的一般格式为:

```
UPDATE TableName
SET columnName1 = dataValue1 [, columnName2 = dataValue2...]
[WHERE searchCondition]
```

SET子句指明需要更新的一个列或多个列的列名，WHERE子句是可选的，如果省略WHERE子句，则将修改表中指定列的所有行数据。如果指定了WHERE子句，则只有那些满足searchCondition的行才被更新。

查询3.20 更新表中行

将“Thriller”类的录像的日租金提高10%

```
UPDATE    Video
SET       dailyRental = dailyRental * 1.1
WHERE     action = 'Thriller';
```

3. 删除语句（DELETE语句）

DELETE语句的一般格式为:

```
DELETE FROM TableName
[WHERE searchCondition]
```

同UPDATE语句一样，WHERE子句也是可选的，如果省略WHERE子句，则表中所有的行都将被删除。如果指定了WHERE子句，那么只有符合searchCondition的行才会被删除。

查询3.21 从表中删除行

删除类别号为634817的出租录像

```
DELETE FROM        VideoForRent
WHERE              catalogNo=' 634817';
```

3.3 数据定义

在这一节中，我们主要介绍两种SQL DDL语句，即:

- 创建表语句（CREATE TABLE） 在数据库中创建一张新表。
- 创建视图语句（CREATE VIEW）从基本表创建一个新的视图。

3.3.1 创建表

```
CREATE TABLE TableName
    { ( columnName dataType [NOT NULL] [UNIQUE]
    [DEFAULT defaultOption] [, ... ] }
    [PRIMARY KEY (listOfColumns), ]
    {[UNIQUE(listOfColumns),] [, ... ]}
    {[FOREIGN KEY (listOfForeignKeyColumns)
    REFERENCES ParentTableName [(listOfCandidateKeyColumns)],
        [[ON UPDATE referentialAction]
        [ON DELETE refentialAction]] [, ...] } )
```

CREATE TABLE语句的完整版本非常复杂，在这一节中我们给出该语句的简化的版本，以便说明该语句的主要部分。图3-1显示了创建Branch、Director和Video三张表的CREATE TABLE语句。每个语句首先定义表中的每个列，然后加上一个或两个其他子句：一个子句用于定义主键，另一个子句用于定义外键。

1. 定义列

定义表中一个列的基本格式如下:

```
columnName dataType [NOT NULL] [UNIQUE] [DEFAULT defaultOption]
```

columnName即要定义的列的列名，dataType定义列的类型。ISO标准支持表3-19所示的数据类型。最常用的数据类型如下:

- CHARACTER(L): 通常简写为CHAR，定义一个长度固定为L的字符串。如果输入的字符串中字符个数少于这个长度，则将在该字符串后面补空格使其符合要求的长度。
- CHARACTER VARYING(L): 通常简写为VARCHAR，定义一个可变长度的字符串（最大长度为L）。如果输入字符串的字符个数少于该长度，则只存储输入的那些字符，所以需要较少空间。
- DECIMAL(precision, [scale])或者NUMERIC(precision, [scale]): 定义一个精确表示的数字。precision指定数字位数，scale指定小数点后的数字位数。这两者的差别是，NUMERIC必须提供指定的精度，而DECIMAL可以提供大于等于所要求的精度。例如，DECIMAL(4)可以代表从－9999到+9999的数字，DECIMAL(4,2)可以代表从－99.99到+99.99的数字。
- INTEGER和SMALLINT: 定义不需要分数的数字，一般SMALLINT可以用来存储绝对值最大为32767的数。

```
CREATE TABLE Branch (branchNo      CHAR(4)        NOT NULL,
                     street        VARCHAR(30)    NOT NULL,
                     city          VARCHAR(20)    NOT NULL,
                     state         CHAR(2)        NOT NULL,
                     zipCode       CHAR(5)        NOT NULL UNIQUE,
                     mgrStaffNo    CHAR(5)        NOT NULL,
                     CONSTRAINT pk1 PRIMARY KEY (branchNo),
                     CONSTRAINT fk1 FOREIGN KEY (mgrStaffNo) REFERENCES Staff
                                    ON UPDATE CASCADE ON DELETE NO ACTION);
CREATE TABLE Director (directorNo  CHAR(5)        NOT NULL,
                       directorName VARCHAR(30)   NOT NULL,
                       CONSTRAINT pk2 PRIMARY KEY (directorNo));
CREATE TABLE Video (catalogNo      CHAR(6)        NOT NULL,
                    title          VARCHAR(40)    NOT NULL,
                    category       VARCHAR(10)    NOT NULL,
                    dailyRental    DECIMAL(4, 2)  NOT NULL DEFAULT 5.00,
                    price          DECIMAL(4, 2),
                    directorNo     CHAR(5)        NOT NULL,
                    CONSTRAINT pk3 PRIMARY KEY (catalogNo),
                    CONSTRAINT fk2 FOREIGN KEY (directorNo) REFERENCES Director
                                   ON UPDATE CASCADE ON DELETE NO ACTION);
```

图3-1 创建Branch、Director和Video表的CREATE TABLE语句

- DATE：存储公历格式的日期，是年（4位数）、月（2位数）、日（2位数）的组合。

另外，还可以定义：

- 该列是否不允许为空（NOT NULL）。
- 是否该列的每个值都是唯一的，即该列是否是一个候选键（UNIQUE）。
- 为该列指定一个默认值，当没有为该列指定新值时，将用默认值代替（DEFAULT）。

ISO标准的完全版本也允许指定其他的条件，有兴趣的读者可以参考2002年Connolly和Begg合作出版的书来获得更详细的信息。

表3-19 ISO SQL数据类型

数据类型	声明			
boolean	BOOLEAN			
character	CHAR,	VARCHAR		
bit	BIT,	BIT VARYING		
exact numeric	NUMERIC,	DECIMAL,	INTEGER,	SMALLINT
approximate numeric	FLOAT,	REAL,	DOUBLE PRECISION	
datetime	DATE,	TIME,	TIMESTAMP	
interval	INTERVAL			
large objects	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT	

2. 主键子句和实体完整性

一个表的主键必须包含对每一行都唯一、非空的值。ISO标准用CREATE TABLE语句中的PRIMARY KEY子句来支持实体完整性。比如，我们可以用如下语句定义Video和Role表的主键，Role表的主键是组合键。

```
CONSTRAINT pk PRIMARY KEY (catalogNo)
CONSTRAINT pk1 PRIMARY KEY (catalogNo, actorNo)
```

注意，关键字CONSTRAINT后面跟随的约束名是可选的，而且允许使用SQL语句ALTER TABLE来删除约束。

3. 外键子句和参照完整性

ISO标准用CREATE TABLE语句中的FOREIGN KEY子句来定义外键。ISO标准通过限制子表的INSERT和UPDATE操作实现参照完整性，这些操作试图在子表中创建和父表中的候选键不相匹配的外键值。如果是更新或删除父表中的和子表有匹配行的候选键值的UPDATE和DELETE操作，那么SQL通过在FOREIGN KEY子句中使用ON UPDATE和ON DELETE子操作来实现引用操作。

- CASECADE: 当从父表更新或删除行时，对子表中匹配的行也要作相应的更新或删除。因为这些被更新或删除的行可能还包含候选键（这些候选键又作为另一张表的外键），所以另外一些表的外键规则又将被触发，所以称为级联方式。
- SET NULL: 当从父表更新或删除行时，将子表中的外键值置为NULL。这只有当外键列没有特别指定是非空（NOT NULL）时才有效。
- SET DEFAULT: 当从父表更新或删除行时，将子表中外键的每个成员都置为一个指定的默认值。这只有当外键列具有一个指定的默认值时才有效。
- NO ACTION: 拒绝父表中的更新或删除操作。这是一种默认的方式，如果没有指明ON UPDATE或ON DELETE规则就采用这种方式。

3.3.2 建立视图

建立视图的（简化的）语句格式如下:

```
CREATE VIEW ViewName [(newColumnName [, ... ] ) ]
AS subselect
```

视图通过指定一个SQL SELECT语句（被称为“定义查询”）来定义。可以为视图中的每列指定一个名字。如果指定了列名列表，则它的列名列表包含的列的数目必须同子查询产生的列数相同。若省略了列名列表，则视图中的每一个列名都与子查询语句中的相应列名相同。如果列名有歧义，则必须指定列名列表。当在子查询中包含计算产生的列而没有用AS子句来为该列指定名称时，或者当连接操作后结果中出现了两个同名的列时，都会出现这种情况。

例如，我们需要得到B001分公司的所有员工的不包含年薪信息的信息，则可以建立如下视图:

```
CREATE VIEW StaffBranch1
AS   SELECT staffNo, name, position
     FROM Staff
     WHERE branchNo = 'B001';
```

3.4 QBE

QBE是另一种基于图形的采用点击方法查询数据库的方法。QBE被誉为非专业用户从数据库中得到信息的最便捷的方法之一。QBE使用模板为查询数据提供了一种可视化工具，查询数据库通过解释要回答的查询得到，它通过显示器屏幕而不是键入SQL语句来实现。但是，你必须指明你希望查看的列（在Microsoft Access称为字段），并指定你希望用来限制查询的数据值。QBE这样的语言是一个交互地查询和更新数据库的高效方法。

和SQL类似，QBE也是在IBM中开发出来的（事实上，QBE是IBM的商标），但是一系列包括微软在内的开发商也在销售类似QBE的界面。开发商通常提供SQL和QBE两种工具，使用QBE服务为简单查询提供更直观的交互界面，同时又具备SQL能够处理复杂查询的能力。

读完本节后，你将发现QBE的查询版本常常更加直观。为了说明，我们使用Microsoft Access 2002工具，并且在每个例子中，我们都指出了等价的SQL语句作为比较。

查询3.1　（再次查询）检索所有的行、列

列出所有录像的详细信息

图3-2a显示了该查询的QBE栅格。在栅格的顶部，显示了我们希望查询的表。对于每个显示的表，Microsoft Access将显示那个特定表中的所有字段。然后，我们就可以将我们希望在结果表中看到的字段拖到QBE栅格底部的"字段"行中。在这个具体的例子中，我们希望看到Video表中的所有行，所以我们将栅格上部的"*"拉到Field行中。默认时，Microsoft Access将标记Show行中相应的单元格，来表明在结果表中显示的字段。

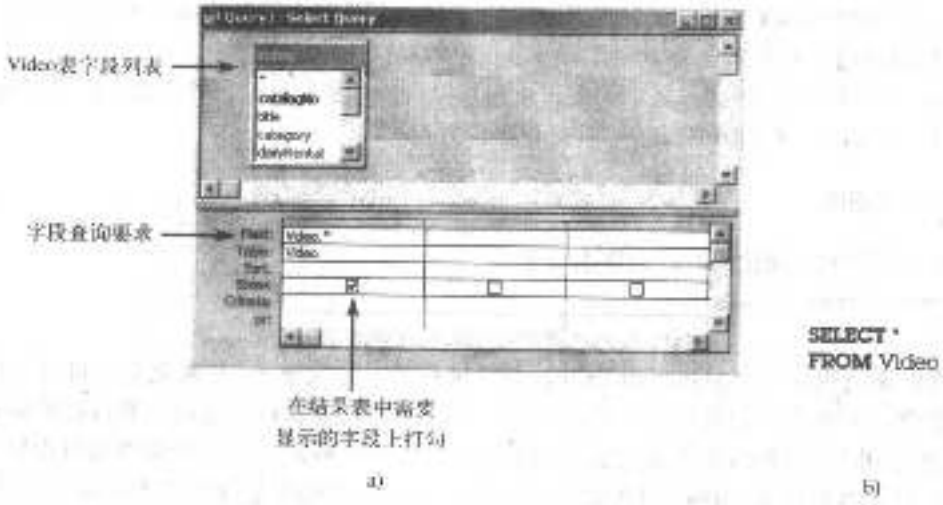


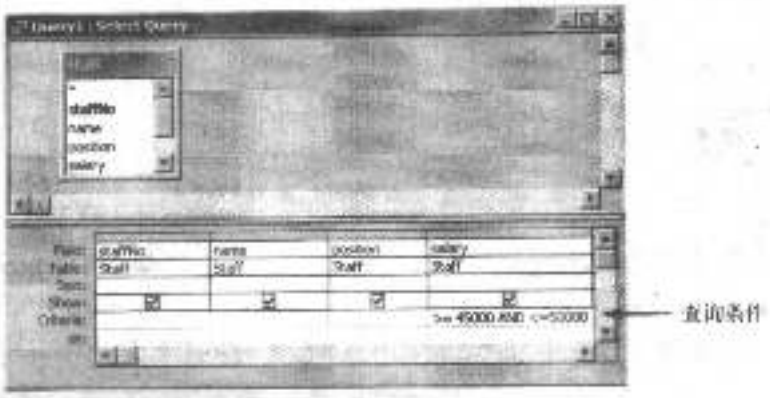
图3-2　a) 查询3.1对应的QBE:列出所有录像的详细信息；b) 与之等价的SQL语句

查询3.6　（再次查询）范围查询条件（BETWEEN/NOT BETWEEN）

列出所有年薪在$45000和$50000之间的员工信息

该查询的QBE栅格如图3-3a所示。在这个例子中，我们在QBE栅格的上部显示了Staff表，然后将相关的字段拖到了栅格底部的Field行中。在这个具体的例子中，我们也指定了限制在结果表中显示的行的条件，就是"salary>=45000 AND salary<=50000"，所以在salary列下的Criteria单元格中我们输入了条件"salary>=45000 AND salary<=50000"。

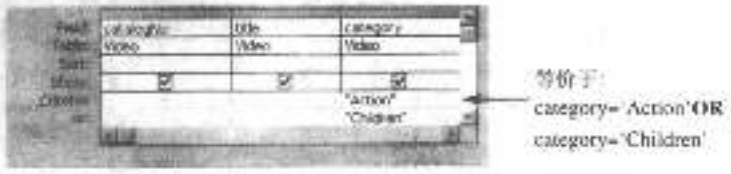
注意，如果条件涉及到了OR运算，则条件中的每个部分需要输入到不同的行中，就像图3-3c所示。此图显示了条件 (category='Action' OR category='Children')。



a)

```
SELECT staffNo, name, position, salary
FROM Staff
WHERE salary >= 45000 AND salary <= 50000;
```

b)



c)

图3-3 a) 与查询3.6相对应的QBE——列出年收入在$45 000和$50 000之间的所有员工;
b) 与之等价的SQL语句; c) 怎样输入涉及到"或"条件的规则的例子

查询3.10 （再次查询）给结果排序

列出所有录像，按价格降序排列

本查询的QBE栅格如图3-4a所示。在这个特殊的例子中，我们希望按价格降序排列结果表。可以在price字段的Sort单元格中的下拉列表框中选择Descending（降序）来实现该功能。注意在本例中，price字段并没有被标记为在结果中显示，因为该字段已经在第一个Field单元格中通过使用"*"包括在结果表中了。

查询3.11 （再次查询）使用COUNT和SUM

列出年薪高于$40000的员工总数和他们的年薪总和

该查询的QBE栅格如图3-5a所示。在本例中，我们希望计算出员工子集（年薪高于$40000的员工）的员工总数和年薪总和。为了实现这个目标，我们使用了聚合函数COUNT和SUM，通过改变查询类型为Totals来得到。这使得在QBE格中显示一个附加的称为Total的行。此行包含GROUP BY 中已选择的所有字段。但是，通过使用下拉列表框，我们可以为staffNo的Total选择COUNT，为salary的Total选择SUM。为了使输出行更加直观，我们可以分别改变totalStaff和totalSalary在输出结果中显示的字段标题。条件">40000"将输入到salary字段中的Criteria单元格中。

a)

```
SELECT *
FROM Video
ORDER BY price DESC;
```

b)

图3-4 a) 与查询3.10相对应的QBE——按价格逆减顺序排列所有录像；b) 等价的SQL语句

a)

```
SELECT COUNT(staffNo) AS totalStaff, SUM(salary) AS totalSalary
FROM Staff
WHERE salary > 40000;
```

b)

图3-5 a) 与查询3.11相对应的QBE——列出年薪高于$40 000的员工人数以及他们的年薪总额；b) 等价的SQL语句

查询3.14 （再次查询）使用HAVING

对于有多于一个员工的分公司办公室，找出这些分公司的员工数和分公司的总年薪

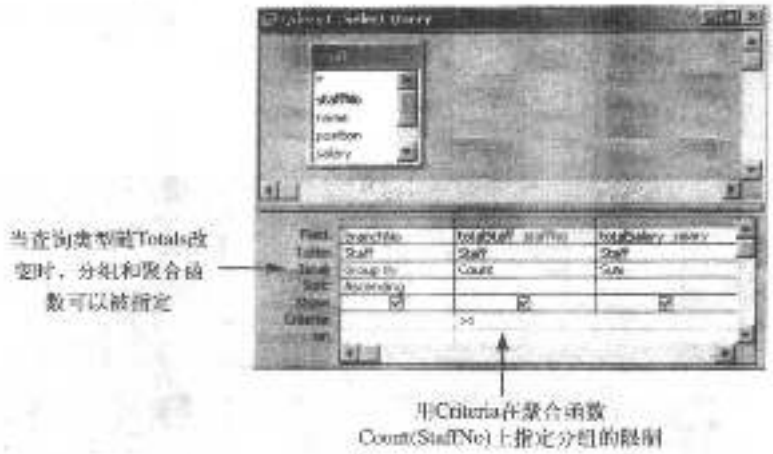
该查询的QBE栅格如图3-6a所示。参照前面的查询，我们将查询类型改为Totals，使用COUNT和SUM功能来计算要求的总额。但是，在这个特定的例子中，我们需要根据分公司号来分组信息（我们需要计算每个分公司的总额），所以branchNo字段的Total单元格需要设为GROUP BY。同样，为了使输出行更加直观，我们可以修改totalStaff和totalSalary的标题名。由于我们只希望显示员工人数多于一人的那些分公司的信息，所以我们在COUNT(staffNo)字段的条件中输入“>1”。

查询3.17 （再次查询）简单连接

列出所有录像及它们的导演

该查询的QBE栅格如图3-7a所示。在栅格的顶部，显示了我们希望查询的表，在本例中就是Video和Director两个表。如前所述，我们将希望在输出结果中包括的列拖到栅格底部。

注意，在SQL查询中我们需要指定Director和Video表的连接，但是，QBE可以自动实现连接。在这些方面，QBE远比SQL容易使用。

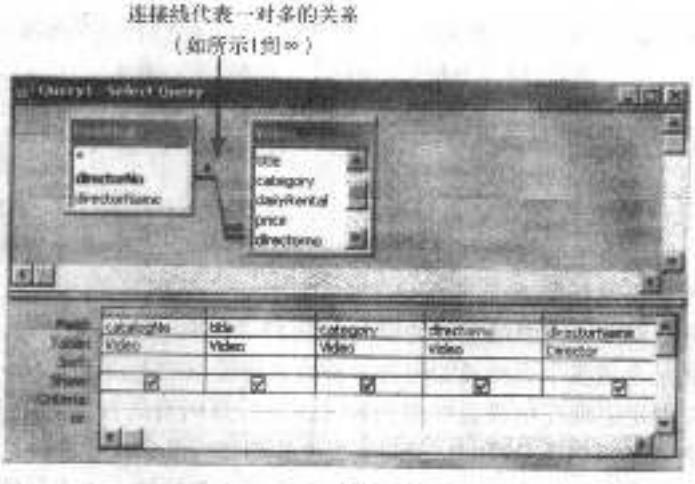


a)

```
SELECT branchNo, COUNT(staffNo) AS totalStaff, SUM(salary) AS totalSalary
FROM Staff
GROUP BY branchNo
HAVING COUNT(staffNo) > 1
ORDER BY branchNo;
```

b)

图3-6 a) 与查询3.14相对应的QBE——对于每个工作人员多于一人的分公司，找出在每个分公司工作的人数和他们的年薪总额；b) 等价的SQL语句



a)

```
SELECT catalogNo, title, category, v.directorNo, directorName
FROM Video v, Director d
WHERE v.directorNo = d.directorNo;
```

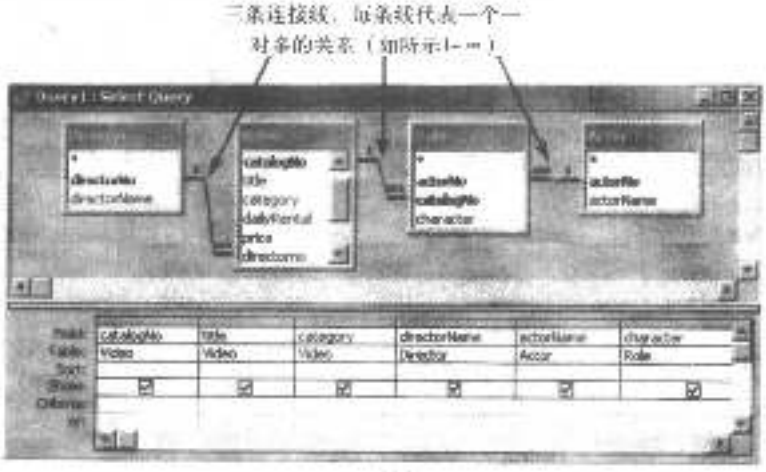
b)

图3-7 a) 与查询3.17相对应的QBE——列出所有录像以及它们相应的导演；b) 等价的SQL语句

查询3.18　（再次查询）四表连接

列出所有录像以及它们的导演、演员名单和相应的角色

该查询的QBE栅格如图3-8a所示，在栅格的顶部，我们显示了希望查询的四张表。如前所述，我们将希望在输出结果中包括的列拖放到栅格底部。如果表间已经建立好了合适的关系，则QBE会根据栅格顶部表示的连接列自动连接四张表。



a)

```
SELECT v.catalogNo, title, category, directorName, actorName, character
FROM Video v, Director d, Actor a, Role r
WHERE d.directorNo = v.directorNo AND
      v.catalogNo = r.catalogNo AND
      r.actorNo = a.actorNo;
```

b)

图3-8　a) 与查询3.18相对应的QBE——列出所有录像以及它们相应的导演的名字，演员的名字和相关的角色；b) 等价的SQL语句

3.5　本章小结

- SQL是一种非过程化语言，它由标准英语单词（如SELECT、INSERT、UPDATE和DELETE等）组成，可同时供专业人员和非专业人员使用。它是定义和操纵关系数据库的正式的和事实上的标准语言。
- SELECT语句是SQL语言中最重要的语句，我们用它来表达一个查询，每个SELECT语句产生一个由一个或多个列组成的包含零行或多行数据的查询结果表。
- SELECT子句指定出现在结果表中的列和（或）计算所得的数据。所有在SELECT子句中出现的列名都必须能够从FROM子句列表中找到相应的表或视图。
- WHERE子句通过为表中行指定查询条件来选择需要在结果表中出现的行。ORDER BY子句允许结果表按一列或多列列值来排序，每一列都可以按升序或降序排列。如果指定了ORDER BY子句，那么该子句就必须作为SELECT语句的最后一个子句。
- SQL支持五个聚合函数（COUNT、SUM、AVG、MIN和MAX），它们将整列作为一个参数并计算出一个值作为结果。在SELECT子句中将聚合函数和列名混合使用是非法的，

除非使用了GROUP BY 子句。

- GROUP BY子句允许在结果表中加入概要信息。一列或多列值相同的行可以归入一个组中，并在使用聚合函数时作为一个整体。这样，聚合函数以每个分组作为一个参数并为每个组计算出一个值作为结果。对于组来说，HAVING子句的作用同WHERE子句一样，用于限制出现在结果表中的组。但是，与WHERE子句不同，HAVING子句可以包含聚合函数。
- 子查询是嵌入到另一个查询中的一个完整的SELECT语句。子查询可以出现在外层SELECT语句中的WHERE或HAVING子句中，通常被称为子查询或嵌套查询。从理论上来说，子查询产生一个包含可以供外层查询访问的数据项的临时表。子查询可以嵌入到另一个子查询中。
- 如果结果表中的列来自多个表，那么就需要使用连接操作。通过在FROM子句中指定多个表，然后在WHERE子句中指定连接列就可以实现表的连接。
- 和SELECT一样，SQL DML还包括INSERT语句用来向表中加入一行数据或使用子查询（subselect）从另外一张表中选择一些行插入表中；UPDATE语句用于更新表中指定列的一个或多个值；DELETE语句用于从表中删除一行或多行数据。
- ISO标准提供了八种基本数据类型：boolean（布尔型）、character（字符型）、bit（位型）、exact numeric（精确数值型）、approximate numeric（近似数）、datetime（日期时间型）、interval（时间间隔型）和character/binary大对象型。
- SQL DDL语句允许定义数据库对象。本章介绍的两种数据库定义语言为CREATE TABLE和CREATE VIEW。
- QBE是另一种基于图形的、采用点击方法查询数据库的方法。QBE被誉为非专业用户从数据库中得到信息的最便捷的方法之一。

复习题

3.1 SQL的两个主要部分是什么？它们各自有什么功能？

3.2 解释SELECT语句中各个子句的功能。这些子句有些什么限制？

3.3 在SELECT语句中使用聚合函数时有什么限制？空值对聚合函数有何影响？

3.4 解释GROUP BY子句是怎样工作的。WHERE子句和HAVING子句之间有什么区别？

3.5 子查询和连接操作有什么不同之处？在什么情况下我们不能使用子查询？

3.6 什么是QBE？QBE和SQL之间的联系是什么？

练习

下面的几张表组成了关系数据库管理系统中某个数据库的一部分：

Hotel	(hotelNo, hotelName, city)
Room	(roomNo, hotelNo, type, price)
Booking	(hotelNo, guestNo, dateFrom, dateTo, roomNo)
Guest	(guestNo, guestName, guestAddress)

这里，Hotel包含了旅馆的详细信息，hotelNo是主键。

Room包含了每个旅馆的房间详细信息，(roomNo, hotelNo)是该表的主键。

Booking包含了订房的详细信息，(guestNo, hotelNo, dateFrom)构成了主键。

Guest包含了顾客的详细信息，guestNo是主键。

1. 创建表

3.7 用SQL创建上述各表（需要时，创建主键和外键）。

2. 填充表

3.8 向这些表中添加一些行。

3.9 更新表，使所有房间的价格上涨5%。

3. 简单查询操作

3.10 列出所有旅馆的详细信息。

3.11 列出位于华盛顿的所有旅馆的详细信息。

3.12 列出住在华盛顿的所有顾客的姓名和地址，按顾客姓名的字母顺序排序。

3.13 列出每晚价格低于$40.00的所有双人房间和家庭套间，按价格升序排列。

3.14 列出没有指定截止日期dateTo的所有订单。

4. 聚合函数

3.15 统计总共有多少个旅馆?

3.16 所有房间的平均价格是多少?

3.17 所有双人房间每晚的费用总和是多少?

3.18 有多少不同的顾客在八月份预订了房间?

5. 子查询和连接操作

3.19 列出Hilton Hotel的所有房间的价格和类型。

3.20 列出现在住在Hilton Hotel的所有顾客。

3.21 列出Hilton Hotel所有房间的详细信息，包括在该房间入住的顾客的名字（如果该房间已经有人住的话）。

3.22 今天Hilton Hotel订房的总收入是多少?

3.23 列出Hilton Hotel中今天没有住客的房间。

3.24 Hilton Hotel中由于这些没有入住的房间，会造成多大损失?

6. 分组

3.25 列出任一旅馆的房间数量。

3.26 列出在华盛顿的任一旅馆的房间数量。

3.27 八月份，每个旅馆的平均订房数为多少?

3.28 华盛顿每个旅馆最常被定的房间的类型是什么?

3.29 今天每家旅馆由于没有入住的房间造成的损失各有多少?

第4章 数据库应用程序生命周期

本章主题:

- 软件开发带来的问题导致了软件危机。
- 软件危机导致了结构化方向到称为信息系统生命周期的软件开发。
- 数据库系统开发生命周期和信息系统生命周期之间的关系。
- 数据库系统开发周期的主要阶段。
- 数据库系统开发生命周期每个阶段的相关活动。

本章以为什么开发软件应用程序需要结构化方法作为开始，介绍了称为信息系统生命周期方法的例子，讨论了信息系统和支持它的数据库之间的关系。然后聚焦于数据库，介绍了称为数据库系统开发生命周期的开发数据库系统的结构化方法的例子。最后，介绍了组成数据库系统开发生命周期（DSDLC）的各阶段。

4.1 软件危机

你可能已经意识到过去的几十年中，软件开发在数量上有着惊人的增长，从小型、由几行代码组成的相对简单的应用程序到大型的、由成百万行代码组成的复杂的应用程序。开发完成之后，许多应用程序表明，这些程序还需要进行不断的维护。这种维护包括修改错误、实现新的用户需求或者修改软件使之运行于新的或升级的平台上。为了支持这些，花费在维护上的精力开始以惊人的速率吸收资源。其结果是，许多主要的软件工程延期、超过预算，并且开发出的软件不可靠，难于维护并且性能差。这导致我们所知道的软件危机。尽管这个术语首先用于20世纪60年代末，但在30多年之后，危机仍然伴随着我们。软件危机的一种迹象就是，OASIG（一个关心IT业的组织方面的特别调查小组）在UK的一份调查，得到如下结论（OASIG，1996，见参考文献）:

- 80% ~ 90%的系统没有达到性能目标。
- 大约80%的交付延期或者超出预算。
- 大约40%开发失败或者放弃。
- 40%以下的完全达到了训练和技术要求。
- 不到25%达到了业务和技术两方面的目标。
- 只有10%~20%达到了所有的成功标准。

软件工程的失败有几个主要原因，包括:

- 缺乏完全明确的需求说明。
- 缺乏合适的开发方法。
- 错误地将设计分解为易管理的组件。

这些问题的一个解决方法就是，提出了软件开发的结构化方法，通常被称为信息系统生命周期（Information Systems Lifecycle）或者是软件开发生命周期（Software Development Lifecycle, SDLC）。

4.2 信息系统生命周期

信息系统（Information System） 收集、管理、控制和分发全公司数据或信息的资源。

信息系统不只是收集、管理、控制数据的使用和产生，而且也将数据转变为信息。信息系统也为那些对公司的成功与否作出重大决定的公司管理者提供有利的依据。信息系统中基本的核心组件是支持它的数据库。

通常，信息系统生命周期的阶段包括：规划、需求收集和分析、设计（包括数据库设计）、构造原型、实现、测试、转换以及操作维护。当然，本书中，我们感兴趣的是信息系统的数据库组件的开发。由于数据库是较大型企业的信息系统的基础组件，因此数据库系统开发生命周期与信息系统生命周期存在内在的必然关系。

4.3 数据库系统开发生命周期

本章中，我们介绍了关系DBMS的数据库系统开发生命周期。数据库系统开发生命周期（DSDLC）各阶段的概要如图4-1所示。每个阶段的名称下面是本章要描述该阶段的节。注意，数据库系统开发生命周期并不是严格顺序进行的，而是通过反复循环包含前面步骤的一些重复。例如，在数据库设计期间遇到了问题，可能就需要其他的需求收集和分析。因为在大多数步骤之间有反复循环，因此，我们只在图4-1中画出了一些明显的反复。

对用户较少的小型数据库应用程序来说，生命周期可能不必很复杂。但是，当要为成千上万用户设计大型的数据库系统时，生命周期就变得极为复杂。

4.4 数据库规划

数据库规划（Database Planning） 尽可能有效地实现数据库应用的各阶段的管理活动。

创建数据库工程的起点就是数据库系统的任务陈述和任务目标。任务陈述定义了数据库系统的主要目标，而每个任务目标标识了数据库必须支持的特定任务。当然，与所有工程一样，数据库规划过程中必然包括对工作量的估计、使用的资源和需要的经费。

正如我们已经注意到的，数据库通常是大型企业信息系统的不可分割的一部分，因此，任何数据库工程都应该完整地包含于公司的全局IS策略中。

数据库规划也可以包括如何收集数据、如何指定格式、需要什么样的文档、如何进行设计和实现的标准的开发。开发和维护标准是很费时的，从开始的建立到后续的维护都需要资源。但是，设计良好的标准为训练职工和衡量质量提供了基础，并且确保了工作遵循的模板，而与职工的技能和经验无关。任何合法的或公司的需求都应该存档，例如某些数据类型的约定必须存档，或者保存一定的时间。

4.5 系统定义

系统定义（System Definition） 定义数据库应用程序的范围和边界，包括主要的用户视图。

在试图设计数据库系统之前，首先标明我们要研究的系统的范围和边界以及它和公司信息系统的其他部分的接口是很有必要的。图4-2说明了如何描述StayHome的数据库应用程序的

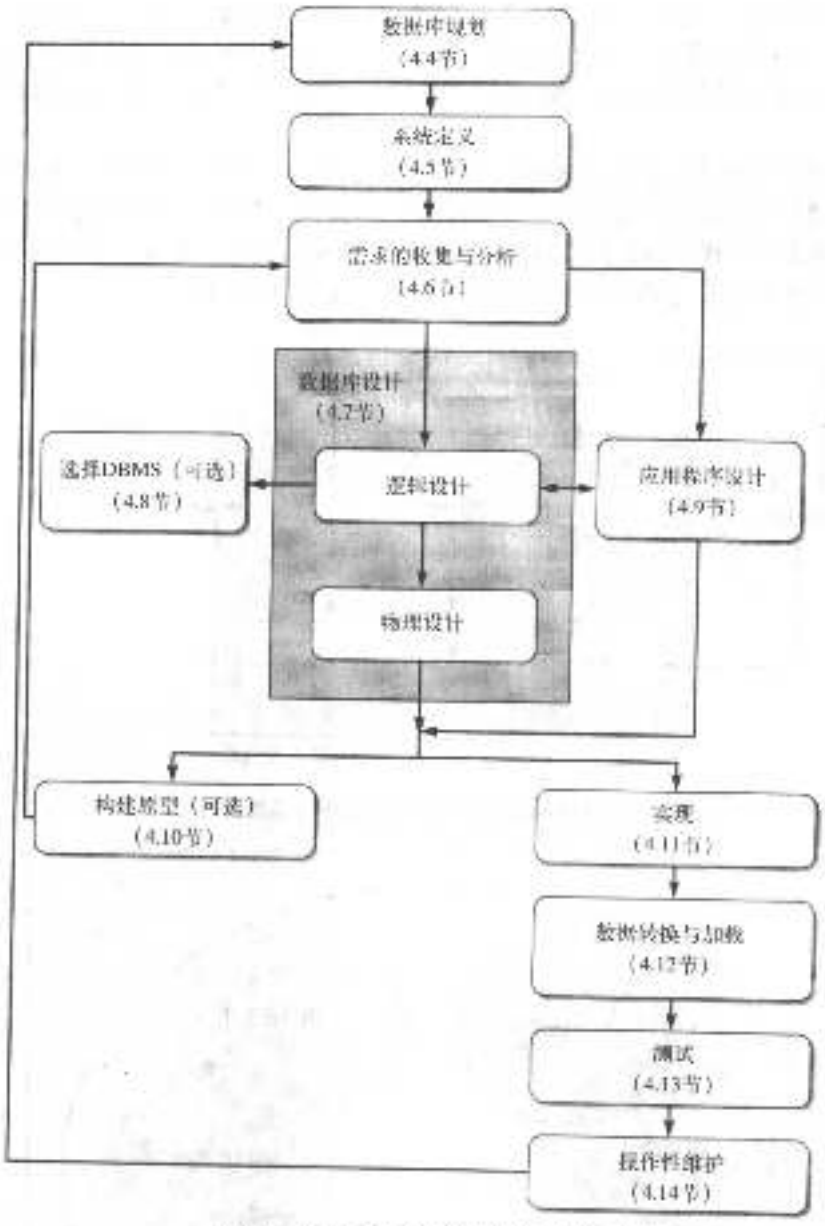


图4-1 数据库应用程序生命周期的主要步骤

系统边界。在为数据库应用程序定义系统边界时，我们不仅要考虑当前的用户视图，也要考虑未来的用户视图。

注意 这种类型的图可以画到任何一个层次。这种图的另一个例子（低一层次）为图6-9。

用户视图

> **用户视图**（User View） 从特定工作（例如经理人或管理者）或者业务应用领域（例如市场、职员或库存控制）的角度，定义的数据库应用的需求。

数据库系统可以有一个或多个用户视图。标明用户视图是开发数据库应用程序的一个重要方面，因为它可以帮助确定当为新的应用程序开发需求时，数据库的主要用户都已经考虑到了。在开发相对复杂的数据库应用程序时，用户视图对允许将需求分解为易处理的小片段也非常有帮助。

用户视图定义了根据要存储的数据和在数据上要执行的事务而得到的所需的数据库应用程序（换句话说，用户要怎样使用数据）。一个用户视图的需求可能不同于其他视图或其他视图的视图。图4-3为多用户视图的数据库系统的图示（用户视图1~6）。注意，用户视图（1、2、3）和（5、6）有重叠需求（用较深颜色标识）。用户视图4有不同的需求。

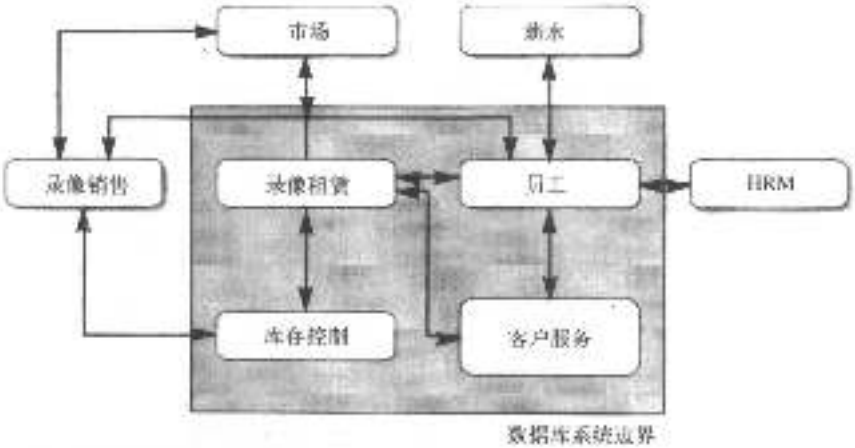


图4-2　StayHome录像出租公司的数据库系统边界

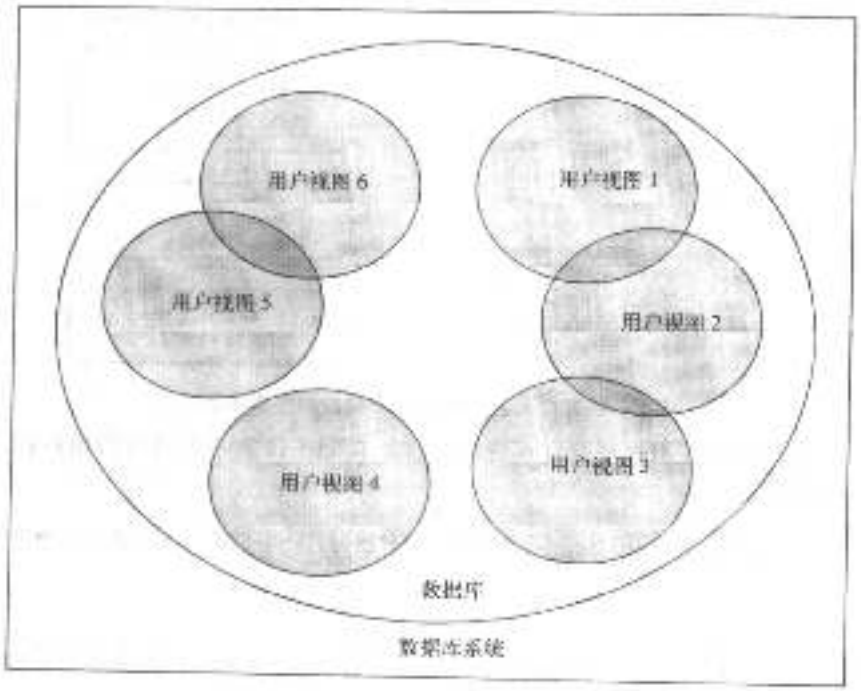


图4-3　有多用户视图的数据库系统：视图4是独立的，其他的视图有部分元素重叠

4.6 需求的收集与分析

需求的收集与分析（Requirements Collection and Analysis） 收集和分析数据库所支持的公司信息，并使用这些信息来标识新的数据库系统需求的过程。

在本阶段中，我们收集和分析公司或公司的部分信息,这些信息将被保存在数据库中。收集信息有许多技术，称为事实发现的技术，我们将在第6章详细讨论。

我们为每个主要的用户视图（也就是，工作角色或业务应用领域）收集信息，包括:

- 对使用或产生的数据的描述。
- 如何使用和产生数据的详细情况。
- 新数据库系统的所有附加需求。

接下来，我们分析这些信息以标识包含在新数据库系统中的需求（特征）。这些信息在文档中被描述为新的数据库系统的需求说明。

提示 需求收集和分析是数据库设计的准备阶段，收集的数据的多少和问题的种类及公司的策略有关。为数据库系统确定需要的功能是一个关键的活动，如果系统功能不够或不完整会让用户很烦恼，并且可能会导致用户拒绝或无法充分利用系统。但是，过多的功能也会有问题，因为它可能会是一个过于复杂的系统，并难于实现、维护、使用和学习。

这个步骤的另一个重要的活动就是确定怎样处理多用户视图的情况。处理多用户视图有三个主要的方法:

- 集中式方法。
- 视图集成方法。
- 以上两种方法的结合。

1. 集中式方法

集中式方法（Centralized Approach） 对于新数据库系统来说，需要将每个用户视图合并到一个需求列表中。在数据库设计阶段创建表示所有用户视图的数据模型。

集中式方法涉及把不同用户视图的需求并合并到一个需求列表中。在数据库设计阶段创建表示所有用户视图的数据模型。图4-4显示的是使用集中式方法对用户视图1到3的管理。通常，当每个用户视图的需求有较大的重叠并且数据库系统不是很复杂时，使用这个方法比较合适。

2. 视图集成方法

视图集成方法（View Integration Approach） 每个用户视图的需求都被用来构建该用户视图的独立的数据模型，最终的数据模型是在数据库设计的后续步骤中通过合并得到的。

视图集成方法将每个用户视图的需求列出独立的需求表。我们创建代表每个用户视图的数据模型，描述单个用户视图的数据模型称为局部逻辑数据模型。然后合并这些局部数据模型创建表达组织中所有用户视图的全局逻辑数据模型。

图4-5显示了使用视图集成方法管理用户视图1~3。通常，当在用户视图间存在相当大的差异并且数据库系统相对比较复杂时，使用这个方法比较合适，它将整个工作划分成更易管理的几个部分。

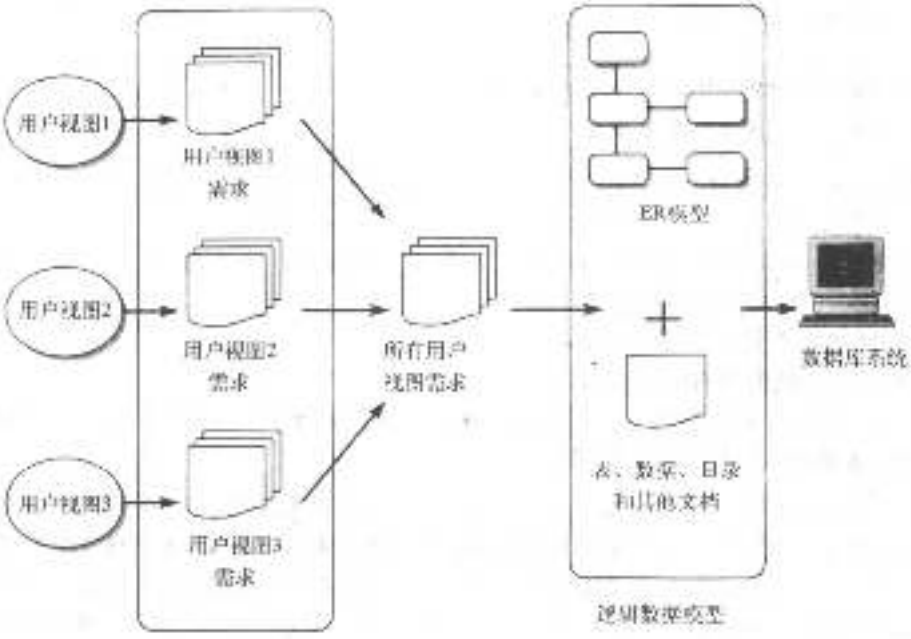


图4-4 处理多用户视图1~3的集中式方法

提示 对于一些复杂的数据库系统，将集中式方法和视图集成方法结合使用来管理多用户视图可能会比较合适。例如，两个或多个用户视图的需求可以首先使用集中式方法进行合并，然后使用它来创建局部逻辑数据模型。（因此在这种情况下，局部数据模型不仅代表一个用户视图，而是代表使用集中化方法合并在一起的一些用户视图。）然后使用视图集成方法将表达一个或多个用户视图的局部数据模型组成表达所有用户视图的全局逻辑数据模型。

我们将在6.4.4节更详细地讨论如何管理多个用户视图，并在本书中演示如何使用集中式方法和视图集成方法为StayHome录像租借公司构建数据库。

4.7 数据库设计

数据库设计（Database Design） 创建支持公司的任务陈述和任务目标的数据库系统的设计的过程。

数据库设计由两个主要阶段组成，分别称为逻辑和物理设计。在逻辑数据库设计阶段，我们要标识数据库中要描述的重要对象以及这些对象之间的关系。在物理数据库设计阶段，我们要确定逻辑设计如何在目标DBMS中物理地实现。在第9章，我们将更详细地讨论数据库设计的这两个阶段，并概要介绍逻辑和物理数据库设计的每个步骤。逻辑数据库设计方法的步骤将在第9章和第10章中详细描述，物理数据库设计将在第12章~第16章介绍。

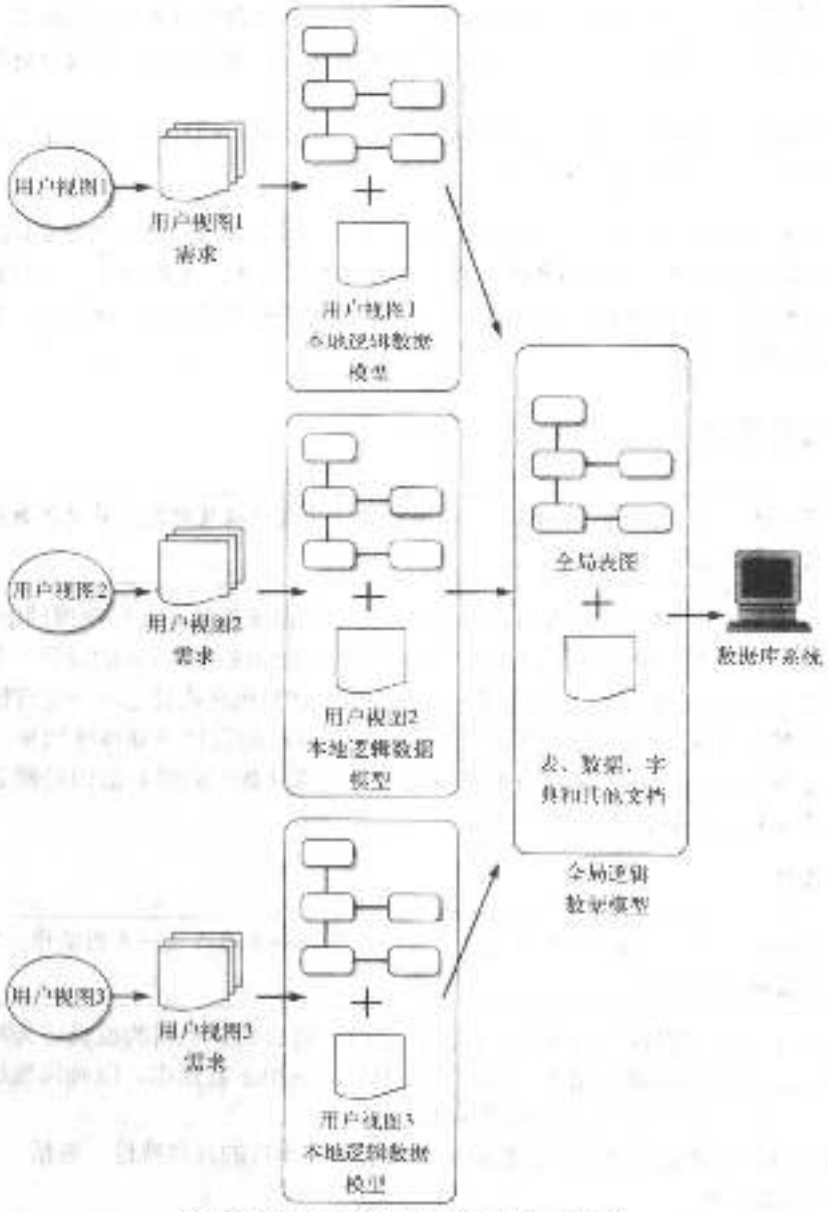


图4-5 处理多用户视图1-3的视图集成方法

4.8 选择DBMS

选择DBMS（DBMS Selection） 指选择合适的DBMS来支持数据库应用程序。

如果目前公司还没有关系DBMS，那么生命周期中合适的选择就是在逻辑和物理数据库设计阶段。但是，如果已经获得足够的关于系统需求的信息，例如网络、性能、重构的难易、安全性和完整性约束等，就可以在先于逻辑设计之前的任何时间做出选择（完整性约束见1.3节）。

尽管选择DBMS的事情不经常发生，但当要扩展业务或者要替换现有系统的时候，评价新的

DBMS产品就变得很有必要。这时，就要选择一个系统来满足公司目前和未来的需求，平衡各方面的费用，包括购买DBMS产品和支持数据库系统的其他软、硬件需求，以及变更与培训员工的费用。

一个简单的方式是根据需求核对DBMS的特征。在选择新的DBMS产品时，应该确保选择过程是计划好的，并且系统确实对公司有益。

提示 现在，世界范围的网络是信息的重要资源，可以用来标识潜在的候选DBMS。供应商的网络站点提供了DBMS产品的有价值的信息。首先，可以浏览一下叫做DBMS ONLINE(可以从www.intelligententerprise.com得到)的DBMS杂志的Web站点，查看一下DBMS产品的全面的索引。

4.9 应用程序设计

应用程序设计（Application Design） 对用户界面的设计以及对使用并处理数据库的应用程序的设计的过程。

在图4-1中，我们了解到数据库和应用程序设计是数据库系统开发生命周期中平行的活动。绝大多数情况下，在产生数据库本身的设计之前我们不能完成应用程序的设计。另一方面，数据库本身就是支持应用程序的，因此，在应用程序设计和数据库设计之间一定有信息流存在。

我们必须确保需求说明所有的功能在数据库应用程序的设计中都体现出来。这包括设计用户和数据之间的交互，我们称之为事务设计。除了设计如何获得必需的功能之外，我们还要设计数据库应用程序的合适的用户界面。

4.9.1 事务设计

事务（Transaction） 由一个用户或应用程序执行的一个动作或一系列动作，它可以访问或改变数据库的内容。

事务代表了“现实世界”的事件，例如在录像出租公司登记新的成员，为租录像的成员创建出租协议，或者增加新的员工。这些事务不得不应用于数据库，以确保数据库与当前的“现实世界”一致，并且支持用户的信息需求。

事务设计的目的就是定义并存档数据库系统需要的事务的高层特性，包括:

- 事务使用的数据。
- 事务的功能特性（事务要做什么）。
- 事务的输出。
- 对用户的重要性。
- 使用的预期率。

有三种主要类型的事务:

- 检索事务。
- 更新事务。
- 混合事务。

检索事务用来检索显示于屏幕（或报告）的数据，或者作为另一个事务的输入的数据。

例如，查询和显示录像（给定录像代码）的详细信息就是检索事务。更新事务就是用来插入新记录、删除旧记录或者修改数据库的已经存在的记录的事务。例如，向数据库中插入新录像的详细信息就是更新事务。混合事务是既检索数据又更新数据的事务。例如，查询并显示录像的详细信息（给定录像编号），然后更新日租金率的值就是混合事务。

4.9.2 用户界面设计

除了设计怎样实现必需的功能之外，我们还必须为数据库应用程序设计合适的用户界面。用户界面应该以用户友好的方式显示需要的信息。用户界面设计的重要性有时会被忽略，或者留到了设计的最后阶段。但是，应该意识到界面可能是系统中最重要的构件之一。如果它易于掌握、操作直观，用户就更易于充分利用提供的信息。另一方面，如果界面没有这些特性，那么系统会产生问题。例如，在实现表格或报告之前，我们首先设计输出界面是很必要的。表4-1列出了设计表格或报告的一些有用的指导（Shneiderman, 1992）。

表4-1 表格/报告设计概要

有意义的标题	便利的游标活动
全面的指导	用于单个字符和整个字段的错误纠正
字段的逻辑组合和序列	用于不可接受值的错误消息
可视化的表单/报表布局	可选字段标记清楚
常用的字段标签	用于字段的解释性消息
一致性的术语和缩略语	完整的信号
一致的颜色	
用于数据项字段的可视化空间和边界	

4.10 构建原型

贯穿设计过程的不同阶段，我们可以选择是完全实现数据库系统的设计还是构建一个原型。

构建原型（Prototyping） *构建数据库系统的工作模型。*

原型通常是不包含全部的需求特征或者没有提供最终系统的全部功能的工作模型。开发原型数据库系统的目的就是允许用户使用原型来标识系统的特征是工作良好，还是不充分，以及如果可能的话提出改善甚至给数据库应用程序添加新特征的建议。按这种方法，我们阐明了需求，并估计了特殊系统设计的可行性。原型应该有相对便宜并容易构造的优点。

现在，通常有两种原型法构造策略：需求原型法和演化原型法。需求原型法使用一个原型来确定提出的数据库系统的需求，并且一旦完成需求，原型就被抛弃了。而演化原型法也是用于相同的目的，重要的不同之处是该原型不会被抛弃，而是得到进一步的开发,并最终成为工作的数据库系统。

4.11 实现

实现（Implementation） *数据库和应用程序设计的物理实现。*

设计阶段完成后（可能包括也可能不包括原型），就该实现数据库和应用程序了。使用所选DBMS的数据定义语言（Data Definition Language, DDL）来完成数据库的实现，或者使用屏蔽了低层的DDL语句且提供了相同功能的图形用户接口(GUI)实现数据库。DDL语句用于创

建数据库结构以及空的数据库文件。这个阶段也实现具体的用户视图。

应用程序可以使用第三或第四代语言（3GL或4GL）实现。这些应用程序的一部分是数据库事务，这个我们使用目标DBMS的数据操作语言（DML）来实现，这些语言可能嵌入在主语言中，例如Visual Basic(VB)、VB.net、Pythen、Delphi、C、C++、C#、Java、COBOL、Fortran、Ada或者Pascal。我们也可以实现应用设计的其他构件，例如菜单屏幕、数据项表格以及报表。另外，目标DBMS也可能有自己的第四代工具，它允许通过非过程化查询语言、报表生成工具、表格生成工具、应用程序生成工具来快速开发应用程序。

应用程序的安全性和完整性控制也被实现了。其中的一些控制是使用DDL实现的，但有一些可能需要使用DDL以外的工具，例如提供的DBMS工具或者操作系统控制。

SQL(结构化查询语言)既是DDL，也是DML。

4.12 数据转换与加载

数据转换与加载（Data Conversion and Loading） *将现有的数据转换到新数据库中，并转换现有的应用程序在新数据库上运行。*

只有在新数据库系统替换旧系统时，该阶段才是必需的。现在，对于DBMS来说，提供将已有文件加载到新的数据库中的工具是很普遍的。该功能通常需要指明源文件的和目标数据库，然后就可以自动地将数据转换为新数据库文件中的格式。如果可以实现，开发者就可以在新系统中转换和使用旧系统中的应用程序。当需要进行转换和加载时，该过程必须正确地实现来确保全部操作的平稳转变。

4.13 测试

测试（Testing） *以寻找错误为目的而执行应用程序的过程。*

在投入使用前，应该对新开发的数据库系统进行全面的测试。测试是通过精心制定的测试计划和真实数据来实现的，因此整个测试过程都要求系统地和严格地执行。注意，在我们的测试定义中，我们没有使用通常的观点，认为测试是为了证明不存在错误。实际上，测试不能证明没有错误，它只能用来说明软件错误的存在。如果测试完成成功，它将会揭示应用程序或者数据库结构方面的错误。作为附加的益处，测试还可以说明数据库和应用程序是按照它们的说明来工作的，并且性能需求得到了满足。此外，从测试阶段收集来的材料还提供了衡量软件可靠性和质量的标准。

在数据库设计过程中，新系统的用户也应该包含在测试过程中。系统测试的理想情景是在一个独立的硬件系统上，有一个测试数据库，但这种情况通常是不能获得的。如果使用真实数据，进行备份是很重要的，以防发生错误。

测试也应该包括数据库系统的可使用性。理想情况下，评估应该按照可用性规格说明来进行。可作为评估标准的例子包括（Sommerville，2000）：

- 可学习性——让一个新用户熟悉系统需要多长时间？
- 性能——系统响应用户工作的情况如何？
- 健壮性——对于用户错误的忍耐性如何？
- 可恢复性——系统从用户错误中恢复的情况如何？

• 适应性——系统与工作的一个模型的绑定程度如何?

这些标准中的某一些可以在生命周期的其他阶段进行评估，当测试完成后，数据库系统就准备停止这些活动并移交给用户了。

4.14 操作性维护

操作性维护（Operational Maintenance） 监视和维护数据库系统安装后的情况。

在这个阶段中，数据库系统进入维护阶段，包括如下的活动:

• 监视数据库系统的性能。如果性能下降到可接受水平之下，则数据库需要调整和改造。
• 维护和升级数据库系统（当需要时）。通过生命周期前面的步骤，新需求要合并到数据库系统中。

我们将在第16章详细介绍该步骤。

4.15 本章小结

• 信息系统是收集、管理、控制和分发全公司数据或信息的资源。
• 数据库是信息系统的基础组件。信息系统的生命周期是内在关系于支持它的数据库的生命周期的。
• 数据库系统开发生命周期主要阶段包括: 数据库规划、系统定义、需求收集与分析、数据库设计、选择DBMS（可选）、应用程序设计、构建原型（可选）、实现、数据转换和加载、测试以及操作性维护。
• 数据库规划是使数据库系统开发生命周期各阶段尽可能有效的管理活动。
• 系统定义就是定义数据库系统的范围和边界，包括主要的用户视图。用户视图可以代表工作角色或业务应用领域。
• 需求的收集与分析是收集和分析数据库所支持的公司信息，并使用这些信息标明新的数据库系统需求的过程。
• 处理多用户视图有三种主要方法称为集中式方法、视图集成方法和两者的结合。集中式方法包括为不同的用户视图收集用户的需求，并集中到一个需求列表中，在数据库设计阶段创建一个代表所有用户视图的数据模型。视图集成方法包括将对每个用户视图的用户需求作为独立的需求列表。创建代表每个用户视图的数据模型，并在数据库设计的后续阶段进行合并。
• 数据库设计就是对支持公司运行和目标的数据库系统进行设计的过程。这个阶段包括逻辑数据库设计和物理数据库设计两部分。
• 选择DBMS的目标就是要选择一个系统，可以满足公司目前和未来的需求，平衡各方面的费用，包括购买DBMS产品和支持数据库系统的其他软/硬件需求，以及变更与培训员工的费用。
• 应用程序设计是对用户界面和使用并处理数据库的应用程序的设计。这个阶段包括两项主要任务: 事务设计和用户界面设计。
• 构建原型就是构建数据库系统的工作模型，使得设计者或用户可以直观地看到系统和评价系统。

- 实现是数据库和应用程序设计的物理实现。
- 数据转换与加载是将已有的数据转移到新数据库中，并且转换现有的应用程序在新数据库上运行。
- 测试就是以寻找错误为目的而运行数据库系统的过程。
- 操作性维护是监视和维护系统安装后的情况的过程。

复习题

4.1 叙述术语“软件危机”的含义。

4.2 讨论信息系统生命周期和数据库系统开发生命周期间的关系。

4.3 简要叙述数据库系统开发生命周期的各个阶段。

4.4 叙述在数据库规划阶段为需求的数据库创建任务陈述和任务目标的目的。

4.5 讨论在设计数据库系统时用户视图代表什么。

4.6 比较并对比集中式方法和视图集成方法在管理有多用户视图的数据库系统的设计上的区别。

4.7 解释为什么在开始物理数据库设计阶段之前选择一个目标DBMS是很必要的。

4.8 讨论应用程序设计中的两个主要活动。

4.9 讨论开发数据库系统原型的好处。

4.10 讨论实现阶段的主要活动。

4.11 叙述数据转换和加载阶段的目的。

4.12 解释测试数据库系统的目的。

4.13 操作性维护阶段的主要活动是什么？

第5章 数据库管理和安全性

本章主题:

- 数据管理和数据库管理的区别。
- 数据管理和数据库管理的目标和主要工作。
- 数据库安全性的范畴。
- 为什么数据库安全是企业主要关心的问题。
- 可能影响数据库系统的威胁类型。
- 如何使用基于计算机的控制方法来保护数据库系统。

在第4章，我们已经学习了数据库系统开发生命周期的主要阶段。在本章中，我们将讨论数据管理员（DA）和数据库管理员（DBA）的作用，以及它们与数据库系统开发生命周期的各阶段的关系。DA和DBA的一个重要工作就是保证数据库的安全。我们还将讨论数据库系统面临的潜在威胁，以及为了将这些威胁所带来的危害最小化而使用的基于计算机的一些对策。

5.1 数据管理和数据库管理

数据管理员（DA）和数据库管理员（DBA）分别负责有关企业数据和企业数据库活动的管理和控制，DA更关心数据库生命周期的早期阶段，从规划到逻辑数据库设计。而DBA则更关心后期的工作，从应用程序和物理数据库设计到数据库的运作维护等。根据企业组织机构和数据库系统的规模和复杂性，DA和DBA可以由一人或多人来担任。我们首先讨论在一个企业中，DA和DBA的目标和具体的工作。

5.1.1 数据管理

数据管理（Data Administration） 对企业数据的管理和控制，包括数据库规划、标准的开发和维护、策略和过程以及逻辑数据库设计。

DA负责管理企业数据，包括非计算机处理的数据，实际上，还需要常常关注那些由企业用户或业务应用领域共享的数据的管理。DA的关键责任包括为高层管理者提供建议和意见，确保数据库技术的应用能够支持公司目标。在有些企业中，数据管理作为一个专门的业务领域而存在，而在另一些企业中，则将它归入到数据库管理中。数据管理的有关工作在表5-1进行了详细描述。

表5-1 数据管理工作

选择合适的生产工具	设定数据采集的标准、建立数据格式
协助公司IT/IS的开发和制定商业策略	估计数据量及可能的增长速度
进行可行性研究并规划数据库的开发	确定数据使用的方式和频率
开发一个企业数据模型	确定数据访问要求，并为合法要求以及企业要求提供保障
确定企业组织机构的数据需要	

（续）

承担逻辑数据库设计 和数据库管理人员、应用程序开发者保持联系，以确保应用程序能满足所有人的要求 为用户提供数据标准和相应责任的培训 保持数据与IT/IS和业务开发一致 确保文档完整性，包括企业的数据模型、标准、策略、程序和最终用户的控制	管理数据字典 保持与最终用户和数据管理人员的联系，以确定新要求并解决数据访问或性能方面的问题 开发一个安全策略

5.1.2 数据库管理

数据库管理（Database Administration） 企业数据库系统的管理和控制，包括数据库物理设计与实现、设置安全性和完整性控制、监控系统性能以及在必要时重组数据库。

DBA比DA更面向技术，要求懂得很多有关DBMS和操作系统环境的知识，他的主要职责是最大程度地使用DBMS软件来开发和维护系统。数据库管理的具体工作在表5-2中进行了详细描述。

表5-2 数据库管理工作

评估和选择DBMS产品 实现物理数据库的设计 用目标DBMS来实现物理数据库的设计 定义安全性和完整性约束 与数据库系统开发者保持联系 确定测试策略 培训用户 负责停止运作已经实现了的数据库系统	监控系统性能、适当地调整数据库 实现数据库的定时备份 确保恢复机制和程序的合适性 确保文档的完整性，包括内部产生的资料 保持与软硬件开发和费用同步发展，在必要时更新设备配置

5.1.3 数据管理与数据库管理的比较

前面几节讨论了数据管理和数据库管理的目标和具体工作，表5-3总结了数据管理和数据库管理的主要工作的不同之处。可能最明显的区别在于工作的本质不同，DA的工作往往具管理性，而DBA的工作则更具技术性。

表5-3 数据管理和数据库管理——主要工作的区别

数据管理	数据库管理
关注战略上的IS规划	评估新的DBMS
确定长期的目标	执行规划来实现当前目标
确定标准、策略和程序	执行标准、策略和程序
确定数据要求	实现数据需求
开发逻辑数据库	开发物理数据库
开发和维护企业数据模型	实现物理数据库的设计
调整数据库的发展	监控、控制数据库的使用
面向管理的	面向技术的
DBMS独立性	依赖于DBMS

5.2 数据库安全性

在这一节，我们将讲述数据库安全性的范畴，并讨论为什么企业要认真对待会给它们的数据库系统带来潜在威胁的事。同时还确定了威胁安全性的范围和它们对数据库系统造成的影响。

数据库安全性（Database Security） 保护数据库系统免受有意的和意外的威胁的机制。

安全性考虑不仅仅针对数据库中保存的数据，安全漏洞还可能影响系统的其他部分，进而可能威胁整个系统。所以，数据库安全性包含硬件、软件、人和数据等。为了有效地实现数据库安全，需要为系统的每个具体任务都定义适当的控制。安全性的必要性在过去往往被忽视或忽略，但现在已经越来越受到各企业的重视。这种改变源于存储在计算机中的重要的企业数据的增长，而且人们认识到这类数据的丢失和不能使用都将引起潜在的灾难。

一个数据库代表一个可以使用合适的控制手段来确保安全的重要的企业资源。我们认为数据库的安全性与如下情况有关:

- 窃取和欺骗
- 机密（秘密）的被窃
- 隐私的被窃
- 完整性的丢失
- 失去有效性

这几点代表了一个组织应该关注的可以减少危险的一些领域，这里的危险是指发生丢失和破坏的可能性。在某些情况下，这些结果联系得极为紧密以至于在一种情况下会引起数据丢失的活动也可能引起另一种情况下数据的丢失。而且，欺骗和隐私的被窃在有意和无意操作中都可能产生，并且不一定需要对数据库或计算机系统进行可觉察的改变。

窃取和欺骗不仅影响数据库环境，也会影响整个企业的运作，因为进行这类犯罪的人，通常关心的是减少这类事情发生的机会。窃取和欺骗不一定需要修改数据，也可能是会引起机密和隐私被窃的活动。

机密指的是需要在数据上维护保密，通常只包含那些对组织来说至关重要的数据；但是，隐私指的是需要保护的关于个人的数据。例如，存在可能导致机密丢失的安全漏洞会使企业丧失竞争力，隐私的丢失可能会导致不利于企业的一些行为的执行。

数据完整性的丢失将导致数据无效和混乱，这可能会严重影响整个组织的运作。很多企业现在都在试图提供永不间断的操作，即称为24 × 7可用性（即，一天24小时，一周7天）。系统不可用意味着数据或系统（或者两者）无法被访问，这会严重影响一个组织的运作。在某些情况下，引起一个系统不可用的事件也可能会引起数据的破坏。

最近，基于计算机的犯罪活动显著增加，而且预计在未来的几年里将会继续增加。数据库安全性的目标就是，在不过度限制用户的前提下，通过性价比较高的方式将有可能导致丢失的事件出现的概率降到最小。

5.2.1 安全威胁

威胁（Threat） 任何可能破坏系统并进而破坏企业的有意或无意的情况或事件。

安全威胁可能是由关系到一个人、一次行动或可能对企业有害的环境的一种情况或一个事件引起的。它可能会引起一个企业有形的损失，比如硬件、软件或数据的丢失，也有可能引起无形的损失，如信誉或客户信任的损失。任何一个企业都面临识别所有可能的安全威胁的问题。所以一个企业至少应该投入一些时间和精力去识别出大多数严重的威胁。

在前几节中，我们已经指出了一些通过有意的和无意的活动引起的安全威胁。有一些威胁可能是有意的，而有一些威胁是无意的，但它们的影响是一样的。有意的威胁通常与人有关，授权用户与非授权用户（有些可能是企业以外的人员）都可能造成这种威胁。

任何一种威胁都必须看成是安全上的漏洞，如果忽视这些漏洞就会给企业带来不利的影响。表5-4列出了各类威胁的例子和可能给企业带来的后果。例如，“利用其他人的方式访问”可能引起企业数据被窃取和欺骗、机密的丢失、隐私的丢失等。

一个企业遭到威胁而引起的后果的程度取决于很多因素，诸如是否有相应的对策和应急计划等。例如，如果硬件出现故障破坏了二级存储，那么在问题被解决之前要终止一切活动。恢复也取决于很多因素，包括最后一次备份是在何时进行的以及恢复系统所需要的时间。

一个企业要能够识别各类可能会遇到的威胁，并启动合适的计划和对策，同时我们也需要考虑实现它们的成本。很明显，把大量的时间、精力和金钱投入在一些发生机率很小的潜在威胁上是不划算的。企业的商业性质也会影响到它需要考虑的威胁的种类，有一些威胁可能是非常罕见的。但是，罕见事件也需要被考虑进来，尤其是当它们的破坏力很大时。图5-1显示了对计算机系统的潜在威胁的总结。

表5-4 威胁安全的例子和可能产生的结果

威 胁	窃取和欺骗	机密性的丢失	秘密的丢失	完整性的丢失	有效性的丢失
利用其他人的方式访问	✓	✓	✓		
未授权的数据修改或备份	✓			✓	
程序的修改	✓			✓	✓
安全策略和程序不合理					
允许机密数据和普通数据混合输出	✓	✓	✓		
在线路上窃听	✓	✓	✓		
黑客的非法入侵	✓	✓	✓		
勒索	✓	✓	✓		
创建系统的漏洞	✓	✓	✓		
窃取数据、程序和设备	✓	✓	✓		✓
安全机制失败，授予过高的访问权	✓	✓	✓		
员工短缺或罢工				✓	✓
员工培训不到位		✓	✓	✓	✓
浏览或透露未授权的数据	✓	✓	✓		
电子干扰和辐射				✓	✓
断电带来的数据损坏				✓	✓
火灾（电器故障、照明故障、纵火）					
洪水、爆炸				✓	✓
对设备的物理破坏				✓	✓
电缆断接				✓	✓
病毒入侵				✓	✓

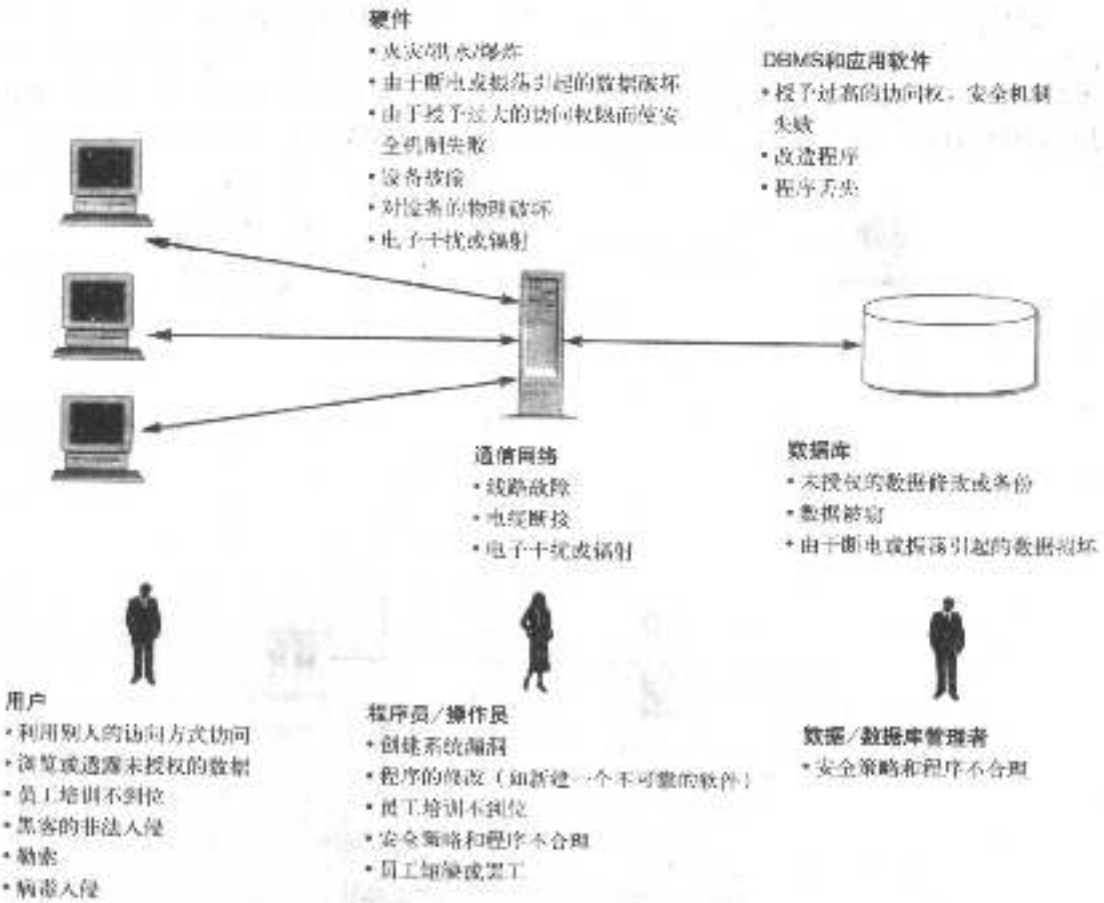


图5-1 计算机系统的潜在威胁

5.2.2 对策——基于计算机的控制

针对数据库系统威胁的对策种类有从物理控制到管理规程的各种手段。尽管基于计算机的控制是可行的，但值得注意的是，一般来说DBMS的安全性和操作系统的安全性相当，因为它们需要紧密的合作。图5-2显示了典型的多用户计算机环境。在这一节，我们集中讨论下列用于多用户环境的基于计算机的安全控制（有些措施在PC机上可能没法实现）。

- 授权
- 视图
- 备份和恢复
- 完整性
- 加密
- 冗余磁盘阵列（RAID）

1. 授权

> **授权**（Authorization） 授予权限和权利使用户只能在合法的权限范围内访问数据库系统和数据库系统对象。

授权控制可以构建在软件中，不仅能够控制某个用户可以访问的数据库系统和系统中的对象，而且也可以控制用户进行的操作。由于这个原因，授权控制有时指的是访问控制。授权的过程包括对要求访问客体的主体的认证，“主体”代表一个用户或程序，而“客体”则代表数据库中的表、视图、过程、触发器或任何可以在数据库系统中被创建的其他对象。

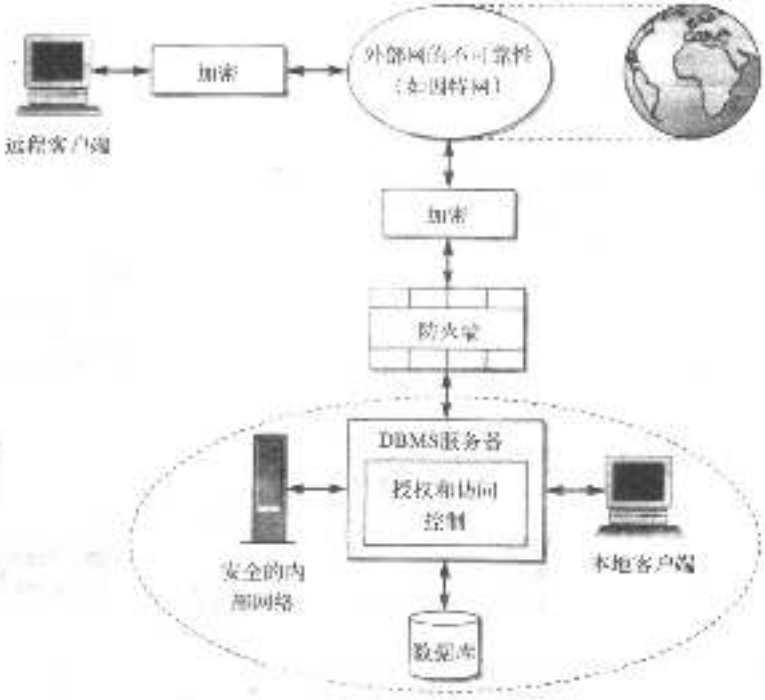


图5-2　典型的多用户计算机环境的描述

2. 认证

认证（Authentication）　确定一个用户是否是如他（她）声称的合法用户的机制。

系统管理员通常需要负责创建每个用户的账号，来管理用户访问计算机系统的权限。每位用户将得到独一无二的身份，操作系统利用它来确定该用户是谁。和每个身份相关联的是一个密码。密码由用户指定并对操作系统来说是可见的，该密码使得操作系统可以认证该用户是谁。

这个过程允许得到认证的用户访问计算机系统，但并不一定拥有访问DBMS和相关应用程序的权利。需要另外一个与之相似的过程来给用户授予使用DBMS的权限。授予使用DBMS的权限的任务通常由DBA实现，它能在DBMS中创建每个用户的账号和密码。

有些DBMS维护一个有效用户身份和相关的密码列表，它与操作系统的有关列表不同。然而，有些DBMS维护一张列表，此列表中的数据项基于当前用户登录账号标识符与操作系统中的用户列表中的数据项对应，当用户已经用一个名字登录操作系统时，这防止了同一个用户用另一个名字登录到DBMS。

(1) 权限

一旦一个用户被允许使用DBMS，可能会自动得到各种其他的权限，例如，拥有访问或创建某些数据库对象（如表、视图和索引）的权限，或者拥有运行各种DBMS工具的权限。

给用户授予权限可以保证他们完成其工作任务。由于过多的授予不必要的权限会危及系统的安全，因此只有当该权限是完成用户工作所必需的时，才授予他（或她）该权限。

(2) 拥有权和权限

DBMS中的某些对象是DBMS本身拥有的，通常以特殊的超级用户的形式出现，比如DBA。相应的，对象的拥有者具有所拥有对象的全部权限。如果其他被授权用户拥有对象，情况也是如此。对象的创建者拥有该对象，而且可以将该对象的合适权限授予其他用户。例如，尽管一个用户拥有一个视图，但是他（或她）可能只拥有查询该视图的权限，这种情况发生在该用户只能查询该视图所基于的基本表的情况下。权限可以传递给其他用户。例如，一些表的拥有者可以授权其他用户查询这些表，但是却不能更新这些表。

当DBMS支持各种不同类型的授权标识符时，那么每种类型具有不同的优先权。例如，一个DBMS可能既允许创建用户标识符也允许创建组用户标识符，单用户标识符的优先级高于组用户标识符。对于这样的DBMS，用户标识符和组标识符可以定义成表5-5a和表5-5b的形式。

表 5-5

a) 用户标识符		b) 组标识符	
用户标识符	类型	组	成员标识符
S0099	User	Sales	S0099
S2345	User	Sales	S2345
S1500	User		
Sales	Group		

在表5-5a中，列名为用户标识符（User Identifier）和类型（Type）的列列出了系统中的每个用户以及他们的用户类型，类型用于区分单个用户和组用户。在表5-5b中，列名为组（Group）和成员标识符（Member Identifier）的列列出了每个组和该组中包含的用户成员。根据具体的标识符授予一定的权限，标识符指明了特定数据库对象所具备的权限种类（如查询、更新、插入、删除或者全部）。

在有些DBMS中，用户需要告诉系统他运行在哪个标识符下，尤其是当该用户属于多个组时。这在熟悉了DBMS提供的认证与其他控制机制的时候就显得特别重要，尤其是在权限被应用到多个授权标识符时，或是权限可以被传递时。这样便可以基于用户的需求以及他们需要使用的应用程序来授予正确的权限。

3. 视图

视图（View） 一个虚拟表，它不一定在数据库中存在，但可以在某个用户提出要求时产生。

视图机制通过对特定用户隐藏部分数据库信息而提供了强大且灵活的安全机制。用户不知道任何在表中存在而在视图中不可见的列和行。视图可以建立在多张表上，用户将被授予使用该视图的适当的权限，但是不能使用基本表。这样，使用视图比简单地给用户授予使用基本表的权限更好。

4. 备份和恢复

备份（Backup） 在脱机存储介质上，周期性地存储一个数据库的副本和日志文件（还有可能包含程序）的过程。

为了保证在系统出现故障时能够恢复数据库，DBMS需要提供备份工具。为了跟踪数据库事务，DBMS还包含了一个特殊文件叫做日志文件（或日记），日志文件主要包括所有更改数据库的信息。建议定期备份数据库副本和日志文件，并且保证备份保存在可靠的地方。当发生故障导致数据库不可用时，备份的副本和日志文件中包括的详细信息可以将数据库恢复到最近的一致性状态。

日记（Journaling） 保存和维护记录了所有对数据库进行的修改的日志文件的过程，当事件执行失败时能够有效地用于恢复数据库。

为对恢复过程提供支持，DBMS需要提供日志工具（有时称为日记），它记录了事务和数据库的当前状态。日志文件的优势在于，当故障发生时，通过使用数据库的备份副本和日志文件中包含的详细信息，数据库可以被恢复到最近的一致性状态。如果在故障发生时没有日志文件，那么唯一可以使用的恢复方式就是使用最新的数据库备份副本来恢复数据库。但是，如果没有日志文件，在数据库备份之后对数据库所作的任何修改都将丢失。

5. 完整性

完整性约束通过限制无效数据的进入以免得出令人误解的或不正确的结果来维护数据库系统的安全。完整性约束在1.3节已经作了介绍，并且我们将在第10章的步骤2.4中进一步讨论。

6. 加密

加密（Encryption） 用一种特殊的算法对数据进行编码，使在没有密钥的情况下数据不会被任何一个程序读取。

如果一个数据库系统存储了非常敏感的数据，那就需要进行预先编码以防可能出现的外部威胁或企图访问该数据的一些操作。有些DBMS提供了一套加密工具。尽管由于解密需要时间会降低系统的性能，但DBMS可以访问这些数据（在解码后）。加密还保护了在通信线路上传输的数据。目前有很多技术可以用于加密数据以隐藏信息，有些是不可逆转的而有一些是可逆转的。不可逆转技术，顾名思义，它不允许原数据可见，但是，可以使用这些数据获得有效的统计信息。可逆转的技术使用的非常广泛。为了在网络中安全的传输数据，要求使用密码系统，密码系统中包括以下内容：

- 用密钥加密数据（明文）。
- 加密算法，用加密密钥将明文转成密文。
- 用解密密钥将密文解码。
- 解密算法，用解密密钥将密文转成明文。

有一种称为“对称加密”的技术，在加密和解密都使用同一个密钥，所以它的安全性依赖于交换密钥时的通信线路的安全程度。但是，大多数用户无法得到可靠的通信线路，而且为了确保可靠性，要求密钥长度同消息一样。但是，大多数系统都是基于短于信息的密钥运作的。一种叫做DES（数据加密标准）的加密规则是IBM开发的标准加密算法。该规则中加密和解密使用一个密钥，该密钥必须是保密的，尽管算法是公开的。该算法将明文的每64位大小的信息块使用56位的密钥进行转换。DES并不被认为是非常可靠的，有些作者还坚持说需要更大的密钥，例如，一种被称为PGP（完好加密）的规则使用128字节的对称算法为其传

递的数据加密。

现在使用特殊的硬件可以破解64位的密钥，当然也要付出一些代价。但是，该技术将在几年之内就会被大家掌握。我们还需要正视一个现实，即在不久的将来80位的密钥也将被破解，只有128位的密钥还不会被破解。“强认证”和“弱认证”有时用来区分以现有的技术和知识都无法被破解的算法和一些会被破解的算法。

另一种密码系统为加密和解密使用不同的密钥，这称为“不对称加密法”。公开密钥系统就是其中的一个实例，它包含两个密钥，一个是公开的，另一个是私有的。加密算法也是公开的。所以如果一个人想给用户发消息，都可以使用对方的公开密钥结合算法来加密。只有私有密钥的拥有者才能解密该消息。公开密钥系统还可以随消息发送一个“数字签名”，证明这个消息是来自于声称发该消息的人。最著名的不对称加密算法是RSA，该名字是由发明该算法的三位发明者的名字的首字母组成。

一般来说，对称算法比非对称算法在计算机中的执行速度更快，但是，在实际应用中，我们常常结合使用这两种方法。即用公开密钥算法加密一个随机产生的密钥，该密钥又用来加密对称算法加密的实际信息。

7. 冗余磁盘阵列（RAID）

运行DBMS的硬件必须是能够容错的，也就是说即使硬件的某个部分出现了故障，DBMS仍能够继续操作。这说明需要有能够无缝地集成到工作系统中的冗余组件，当一个和多个组件出现问题时，可以利用这些冗余组件。能够容错的主要的硬件组成部分包括磁盘驱动器、磁盘控制器、CPU、电源和风扇。磁盘驱动器是最易受攻击的组件，因为它引起故障的时间比其他硬件组件都短。

一种解决方案是采用冗余磁盘阵列（RAID）技术，RAID拥有由一组独立的磁盘阵列组成的大磁盘阵列，使用它可以提高可靠性，有时候还会提高性能。

5.3 本章小结

- 数据管理是对企业数据的管理和控制，包括数据库规划、开发和维护标准、策略和过程以及逻辑数据库的设计。
- 数据库管理是对企业数据库系统的管理、控制，包括物理数据库设计与实现、设置安全性和完整性控制、监控系统性能以及在必要时重组数据库。
- 数据库安全性就是有关保护数据库系统免受有意和无意威胁的机制。
- 安全威胁就是可能破坏系统并进而影响企业的有意的和无意的任何情况和事件。
- 用于多用户环境的基于计算机的安全控制包括：授权、视图、备份和恢复、完整性、加密和冗余磁盘阵列。
- 授权就是授予权利和权限使得用户能够合法地访问数据库系统和访问数据库系统对象。
- 认证就是用来确定一个用户是否是如他（她）声称的合法用户的机制。
- 视图是一个虚拟表，它不一定在数据库中存在，但可以在某个特定用户提出要求时产生。
- 备份即在脱机的存储介质上，周期性地存储一个数据库的副本和日志文件（也可能包含程序）的过程。
- 日记就是保存和维护一个记录了所有对数据库进行的修改的日志文件的过程，当事件执行失败时能够用它恢复数据库。

- 完整性约束通过限制无效数据的进入而得出令人误解的或不正确的结果来维护数据库系统的安全。
- 加密就是用一种特殊的算法将数据编码，这样在没有解码的情况下任何一个程序都不能读取数据。
- 冗余磁盘阵列（RAID）拥有由一组独立的磁盘阵列组成的大磁盘阵列，以用于提高可靠性，同时还能提高性能。

复习题

5.1 定义数据管理和数据库管理的目标和具体工作。

5.2 比较和对比DA和DBA负责的主要工作。

5.3 解释数据库安全性的目标和范畴。

5.4 列出可能影响数据库系统的安全威胁的种类，并讨论每一种威胁可能给企业造成的影响。

5.5 解释下面有关数据库安全的术语:

(a) 授权

(b) 视图

(c) 备份与恢复

(d) 完整性

(e) 加密

(f) RAID

第二部分 数据库分析与设计技术

第6章 事实发现
第7章 实体－关系建模
第8章 规范化

第6章 事 实 发 现

本章主题：

- 在数据库系统开发生命周期中何时要用到事实发现技术。
- 在整个数据库系统生命周期中收集的事实类型。
- 在整个数据库系统生命周期中所产生的文档类型。
- 最泛化的事实发现技术。
- 如何使用每一种事实发现技术，以及每种技术的优点和缺点。
- StayHome录像出租公司案例研究。
- 怎样把事实发现技术运用在数据库系统开发周期的早期阶段。

在第4章，我们学习了数据库系统开发周期的各个阶段。在这些阶段中许多情况下数据库开发者必须捕获必要的事实来构建数据库系统。这些事实覆盖业务和数据库系统用户，包括术语、问题、机会、约束、需求和优先权。这些事实都要通过事实发现技术来捕获。

> **事实发现**（Fact-Finding） 运用面谈和提问等技术来收集有关系统、需求和用户喜好的形式化处理过程。

这一章，我们讨论什么时候数据库开发人员可以使用事实发现技术，以及必须捕获什么类型的事实。我们简要说明了怎样利用这些事实来产生在整个数据库系统开发周期中所要使用的主要的文档类型，并描述了最常使用的事实发现技术，以及它们的优点和缺点。最后我们将以一个叫StayHome的录像出租公司为例，展示如何在数据库系统开发过程的早期阶段使用这些技术。在第9章～第10章和第12章～第16章，我们将用这个实例来展示数据库设计的方法学。

在整个这一章中，我们所说的"数据库开发人员"是指负责一个数据库系统的分析、设计和实现的人或者团体。

6.1 什么时候使用事实发现技术

在数据库应用开发过程中，许多情况下都要用到事实发现技术。然而，事实发现技术在生命周期的早期，包括数据库规划、系统定义、需求收集和分析阶段，都是非常关键的。正是在这些早期阶段，开发人员要了解术语、问题、机会、约束、需求以及业务和系统用户的优先级。事实发现在数据库设计以及以后的阶段中也要用到，但使用的范围要小一些。例如，

在物理数据库设计阶段，当开发人员想要更多地了解所选用的数据库管理系统时，事实发现就是一种技术。即使在最后的操作维护阶段，事实发现技术也用来决定一个系统是否需要提高性能或者进一步开发以满足新的需求。

提示　大概估计一下要在数据库工程的事实发现上花费多少时间和精力是非常重要的。据分析，大量的快速学习将导致瘫痪，而考虑得太少则会由于用错误的方法解决错误的问题而导致不必要的时间和金钱的浪费。

6.2 收集哪些事实

在整个数据库系统周期中，开发人员需要捕获的事实包括系统当前的或者将来的事实。表6-1用例子说明了要捕获的数据种类和在开发周期各个阶段所产生的文档。在第4章中我们提到，数据库系统开发的各个阶段并不是严格按顺序进行的，而是通过反馈循环包括许多前阶段的重复。这也适用于各个阶段的数据采集和文档产生。例如，如果在数据库设计阶段遇到了问题，就有必要对新系统的需求进行进一步的数据采集。

表6-1　数据库应用开发周期各个阶段采集的数据和产生的文档示例

开发阶段	捕获的数据示例	产生的文档示例
数据库规划	数据库工程的目标和目的	任务描述和数据库系统目的
系统定义	主要用户视图描述（包括工作角色/或者业务应用领域）	数据库系统的边界和范围定义，所支持的用户视图的定义
需求收集和分析	用户视图的要求，系统说明，包括性能和安全性要求	用户和系统的需求说明书
数据库设计	用户检查逻辑数据库设计后以及目标DBMS提供的功能的反馈	逻辑数据库设计（包括ER模型、数据字典、表），物理数据库设计
应用程序设计	用户对界面设计的反馈	应用程序设计（包括对程序和界面的描述）
DBMS选择	目标DBMS所提供的功能	DBMS评估和推荐
构建原型	用户对原型的反映	修改了的用户需求和系统说明书
实现	目标DBMS提供的功能	
数据转换和加载	当前数据的格式，目标DBMS的数据导入能力	
测试	测试结果	所使用的测试方法、测试结果的分析
操作维护	性能测试结果，新的或改变用户和系统需求	用户手册、性能结果分析、修改的用户需求与系统说明书

在6.4节中，我们将回过头来检查数据库系统开发周期的前3个阶段，即数据库规划、系统定义、需求收集和分析。对每一个阶段，我们以StayHome录像出租公司为例说明了利用事实发现技术收集数据并产生文档的处理过程。但在本节之前，我们首先回顾一下最常用的事实发现技术。

6.3 事实发现技术

数据库开发人员在一个数据库工程中通常使用几种事实发现技术。常用的技术有五种:

- 检查文档
- 面谈
- 观察操作中的业务
- 研究
- 问卷报告

6.3.1 检查文档

当你想深入了解为什么客户需要数据库应用时，检查文档是非常有用的。检查文档也可以发现文档有助于提供与问题相关的业务信息（或者业务事务的信息）。如果问题与现存系统相关，则一定有与该系统相关的文档。检查与目前系统相关的文档、表格、报告和文件是一种非常好的快速理解系统的方法。表6-2列出了应该检查的文档类型的例子。

表6-2 要检查的文档类型示例

文档的用途	有用资源示例
描述数据库的问题和需求	内部备忘录、电子邮件、会议备忘录 员工客户意见、问题描述文档 效率回顾/报告
描述受问题影响的业务（或部分业务）	组织图表、任务陈述、事务战略计划 正被研究的部分业务的目标、任务/工作描述 手工的表格和报告的例子 计算表格和报告举例 完成的表格/报表
描述当前系统	不同类型的数据流图和图表 数据字典 数据库应用程序设计 程序文档 用户/培训手册

6.3.2 面谈

面谈是最常用的，通常也是最有用的事实发现技术。通过面对面谈话可以获取信息。面谈还有其他目的，如找出事实、确认事实、澄清事实、得到所有最终用户、标识需求、集中意见和观点。然而，使用面谈这种技术需要良好的交流能力，能够有效地和具有不同价值观、不同喜好、观点、动机和个性的人打交道。和其他技术一样，面谈并不是在所有情况下都是最好的。它的优缺点见表6-3。

表6-3 面谈作为事实发现技术的优缺点

优　点	缺　点
谈话对象可以按照谈话人预先确定的感兴趣的内容进行交谈 谈话人可以在谈话过程中改编或重述问题 谈话人可以观察谈话对象的肢体语言 谈话对象可以自由地、开放地回答问题 谈话对象可以了解部分项目	非常浪费时间，代价昂贵，因此不切实际 是否成功依赖于谈话人的交流技巧

有两种类型的面谈：有组织的和没有组织的。没有组织的面谈通常仅由一个通用目标指导，并且有非常少的特定的问题。谈话人依靠谈话对象提供谈话的框架和方向，这种类型的谈话通常不能抓住问题的焦点，因此，你将发现它不是很适用于数据库分析和设计。

在有组织的谈话中，谈话人有特定的问题要问谈话对象。根据谈话对象的回答，谈话者将提出一些附加的问题以获得非常明确的答案并进行一些扩展。没有明确框架限制的问题能够让谈话对象用一种看起来适合的方式回答。例如“为什么你对成员注册报表不满意”。限制

框架问题的答案要么是特定选择，要么是短的直接的回答。例如“你是否按时收到了成员注册报告”或者“成员注册报告所包含的信息是否精确”。这两个问题只需回答“是”或“否”。

提示 为了保证谈话成功，必须选择合适的谈话人选，准备的问题涉及面要广，要引导谈话有效地进行。

6.3.3 观察业务的运转

观察是用来理解一个系统的最有效的事实发现技术之一。使用这项技术可以参与或者观察做事的人以了解系统。当用其他方法收集的数据的有效性值得怀疑或者系统特定方面的复杂性阻碍了最终用户做出清晰的解释时，这种技术尤其有用。

与其他事实发现技术相比，成功的观察要求非常多的准备。为了确保成功，要尽可能多地了解你要观察的人和活动。例如，所观察的活动的低谷、正常以及高峰期分别是什么时候？所观察的人是否会因为有人观察他们并记录他们的活动而心情烦乱？表6-4列出了观察作为一种事实发现技术的优缺点。

表6-4 观察作为事实发现技术的优缺点

优 点	缺 点
可以检查数据和实施的有效性	当有人观察时人们可能自觉或不自觉的行为异常
观察者可以很准确地看到正在做的事情	在那段时间，可能会遗漏一些观察任务，这些任务的难度和量都有所不同
观察者也可以获得描述任务的物理环境的数据	有些任务并不总是以它们被观察时的方式运行
相对低廉	可能不切实际
观察者可以做工作测量	

6.3.4 研究

一个有用的事实发现技术是研究应用和问题。计算机行业的杂志、参考书和因特网是非常好的信息来源。它们可以提供有关其他人如何解决该问题的信息，也可以告诉你要解决此问题的软件包是否存在。表6-5列出了研究作为事实发现技术的优缺点。

表6-5 用研究作为事实发现技术的优点和缺点

优 点	缺 点
如果解决问题的方法已经存在则能够节省时间	可能很浪费时间
研究者可以指导其他人如何解决相似的问题或者怎样满足相似的要求	需要获得合适的信息资源
使研究者能够跟上最新发展	由于问题在其他地方没有写成文档，因此最终可能对解决问题没有什么帮助

6.3.5 问卷调查

另一种事实发现技术是通过问卷来调查。问卷是一种有着特定目的的小册子，这样可以在控制答案的同时，集中一大群人的意见。当和大批听众打交道时，其他的事实发现技术都不能有效地把这些事实列成表格。问卷调查作为事实发现技术的优缺点见表6-6。

问卷有两种格式，自由形式和固定形式。在自由格式问卷上，答卷人提供的答案有更大

的自由。问题提出后，答卷人在题目后的空白地方写答案。自由格式的问题举例如下："你当前收到的是什么报表，它们有什么用"，"这些报告是否存在问题，如果有，请说明"。自由格式问卷存在的问题是答卷人的答案可能难以列成表格，而且，有时答卷人可能答非所问。

表6-6 问卷调查的优缺点

优点	缺点
被调查者可以按方便时间答问卷并交还	交还率可能很低，可能只有5%~10%
相对廉价的从大批人群中收集数据的方式	问卷交还时可能没有回答完整
当调查对象的回答可信度高时，他们提供了真实情况	没有机会修改和重新描述被误解的问题
回答可以列成表格并迅速分析	不能观察和分析答卷人的肢体语言
可以使用各种方式发放问卷，包括人工发放、邮寄、发E-mail	准备问卷非常浪费时间

固定格式问卷包含的问题的答案是特定的，给定一个问题，回答者必须从提供的答案中选择一个。因此，结果一目了然且容易列表。但另一方面，答卷人不能提供一些有用的附加信息。固定格式问题的例子如：现在的录像出租报告的形式非常理想，不必改动。答卷人可以选择的答案有"是"或"否"，或者一组选项包括"非常赞同"、"同意"、"没意见"、"不同意"、"强烈反对"。

6.4 StayHome案例研究

在这一节，我们首先描述StayHome案例研究，然后使用该案例研究说明如何通过数据库规划、系统定义以及需求收集和分析这些阶段来在数据库应用开发周期的早期阶段建立一个数据库工程。

6.4.1 StayHome案例研究——概览

这个案例研究描述了一个叫StayHome的公司，该公司出租录像给其成员。StayHome的首家分公司成立于1982年，地点在西雅图。但该公司现在日益壮大，有许多连锁点遍布全美。该公司的成功秘诀是它给顾客提供了一流的服务，并且收藏的录像门类齐全。

StayHome现在有2000名员工分布在100个部门中。当有新员工加盟公司时，需要使用员工注册表格。来自Mary Martinez的员工注册表格如图6-1所示。

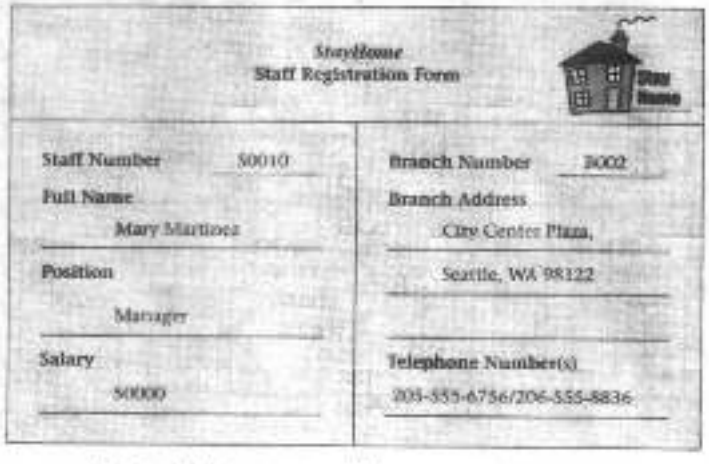
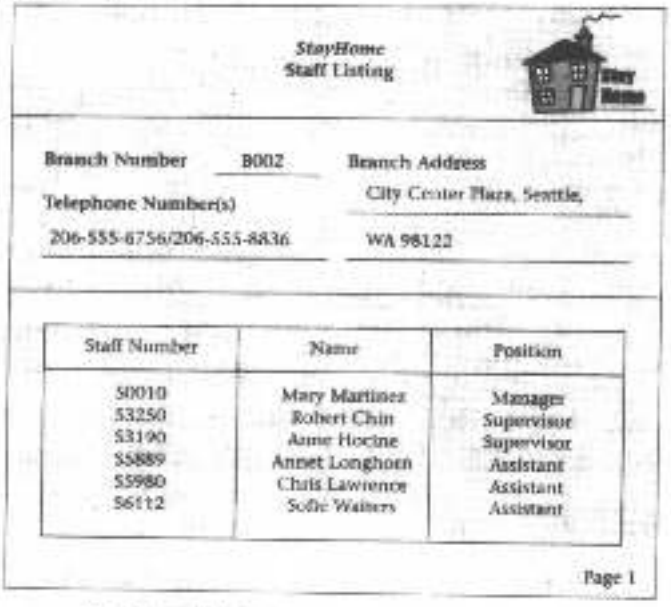


图6-1 来自Mary Martinez的StayHome的员工注册表

每个分公司都有一名经理和数名主管。经理负责日常事务，而主管则监督旗下员工。西雅图分公司员工列表的第一页见图6-2。



StayHome
Staff Listing

Branch Number B002　**Branch Address** City Center Plaza, Seattle, WA 98122

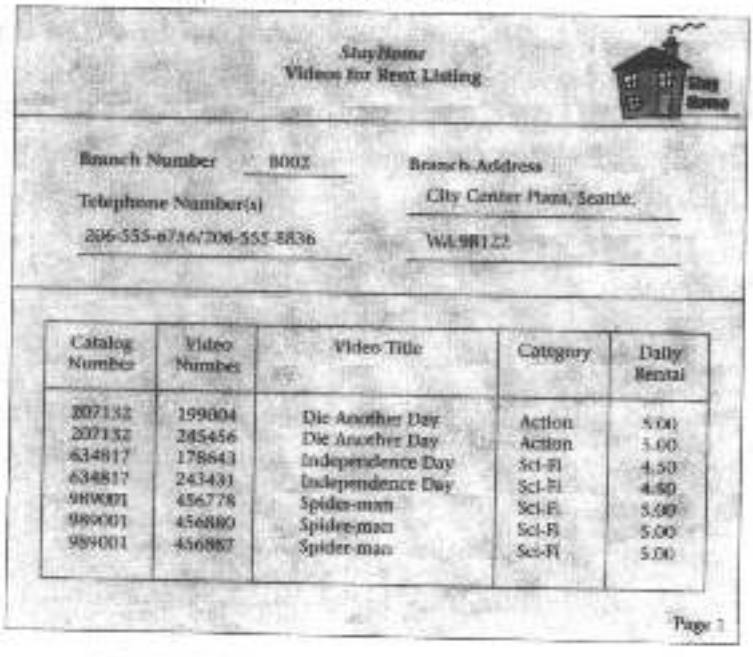
Telephone Number(s) 206-555-6756/206-555-8836

Staff Number	Name	Position
S0010	Mary Martinez	Manager
S3250	Robert Chin	Supervisor
S3190	Anne Hocine	Supervisor
S5889	Annet Longhorn	Assistant
S5980	Chris Lawrence	Assistant
S6112	Sofie Walters	Assistant

Page 1

图6-2　在西雅图分公司工作的员工列表的第一页

StayHome的每个分公司都有库存的录像以备出租。每盘录像用分类号码唯一标识。但是大多数情况下，在一个分公司中，同一盘录像有多份拷贝，因此每份拷贝用录像号码来区分。西雅图分公司可以出租的录像清单的第一页如图6-3所示。



StayHome
Videos for Rent Listing

Branch Number B002　**Branch Address** City Center Plaza, Seattle, WA 98122

Telephone Number(s) 206-555-6756/206-555-8836

Catalog Number	Video Number	Video Title	Category	Daily Rental
207132	199804	Die Another Day	Action	5.00
207132	285456	Die Another Day	Action	5.00
634817	178643	Independence Day	Sci-Fi	4.50
634817	243431	Independence Day	Sci-Fi	4.50
989001	456778	Spider-man	Sci-Fi	5.00
989001	456880	Spider-man	Sci-Fi	5.00
989001	456887	Spider-man	Sci-Fi	5.00

Page 1

图6-3　西雅图分公司可以出租的录像清单的第一页

顾客在租录像之前，必须先成为公司的一名会员。顾客加入时要填写StayHome会员注册表格。顾客Don Nelson的会员注册表格如图6-4所示。StayHome当前大约有10万会员。一个顾客可以在不同分公司分别注册，但每次注册都要填写一张注册表格。西雅图分公司的会员清单列表的第一页如图6-5所示。

StayHome
Member Registration Form

Member Number (Enter if known) M284354	**Branch Number** B002
Full Name Don Nelson	**Branch Address** City Center Plaza, Seattle, WA 98122
Member Address 123 Suffolk Lane, Seattle, WA 98117	**Registered By** Robert Chin
Date Registered 09-Oct-01	

图6-4 顾客Don Nelson的会员注册表格

StayHome
Members Listing

Branch Number B002
Telephone Number(s) 206-555-6756/206-555-8836
Branch Address City Center Plaza, Seattle, WA 98122

Member Number	Name	Address	Date Joined
M129906	Karen Homer	634–12th Avenue, Seattle, WA 98123	10-Jan-94
M189976	John Hood	4/4 Rosie Lane, Seattle	21-May-95
M220045	Jamie Peters	5A- 22nd Street, Seattle, WA 98451	20-May-96
M228877	Claire Sinclair	44B–16th Street, Seattle, WA 98123	28-Aug-96
M265432	Janet McDonald	1 Lincoln Way, Seattle, WA 98234	19-Aug-97
M284354	Don Nelson	123 Suffolk Lane, Seattle, WA 98117	09-Oct-98
M284666	William Carling	1 Sparrowhill Way, Seattle, WA 98111	10-Oct-99

Page 1

图6-5 西雅图分公司的会员清单列表的第一页

会员注册后，可以自由租借录像，一次最多借10盘录像。当会员租借录像时，要填写StayHome录像租借表格。图6-6是Claire Sinclair 租借录像《Harry Potter》和《Shrek》的已完成的租借表格。

StayHome
Video Rental

Member Number M228877　　Branch Number B002

Member Name
Claire Sinclair

Branch Address
City Center Plaza,
Seattle, WA 98122

Video Number	Video Title	Daily Rental	Date Out	Date In	Total Rental
565611	Harry Potter	4.50	12-Dec-03	14-Dec-03	4.50
476667	Shrek	4.00	13-Dec-03		

图6-6　Claire Sinclair 租借StayHome录像的例子

随着公司的壮大，公司使用和产生的数据量日益增大，管理变得非常困难。为确保公司继续发展，StayHome的主管强烈要求建立数据库应用程序来帮助解决日益庞大的数据管理问题。

6.4.2　StayHome案例研究——数据库规划

开发数据库应用的第一个步骤是清楚地定义数据库工程的任务陈述，这个任务陈述定义了数据库应用程序的主要目标。这些推动了业务（如主管或拥有者）中的数据库工程正常地定义任务陈述。任务陈述可以帮助澄清数据库工程的目标，为开发出一个简洁高效的数据库应用程序提供更清楚的途径。

定义好任务陈述之后，下一个活动包括确定任务目标。每个任务目标应该标识一个数据库必须支持的特定任务。前提是数据库支持的任务目标在任务陈述中必须有定义。任务陈述和目标可能伴随着许多额外信息，这些信息通常指定了要完成的工作，完成工作所要使用的资源以及所要支付的金钱等。

1. 创建StayHome数据库系统的任务陈述

为了产生StayHome数据库系统的任务陈述，应该首先和公司主管谈话，并和主管指定的人员会谈。在这个阶段，提出一些谈话对象可以自由回答的问题通常是最有用的。例如，你（数据库开发人员）可以通过询问StayHome主管下述问题来开始谈话：

开发人员：公司的目标是什么？

主管：通过遍布全美国的分公司，我们提供各种各样的录像出租给已注册的会员。

开发人员：你为什么需要使用数据库？

主管：老实说，我们不能应付我们所取得的成功。过去几年，我们新开了几个分公司，每个分公司提供的录像选择范围都很广，注册会员也越来越多。但是这些成功也带来了日益严重的数据管理问题，这导致我们提供的服务水平有所下降。而且，公司各分公司间的合作和信息共享的程度也很低，这是一个令人头疼的问题。

开发人员：你怎么知道使用数据库可以解决你们的问题？

主管：我所知道的全部就是我们被堆积如山的文书所淹没。我们需要一些东西来加速我们的工作，也就是说，它能够自动处理日常事务，这些事情似乎现在每天都要做。我也希望各分公司可以同时开始工作。数据库可以做到这些的，是吗？

这类型问题的回答有助于明确地写出问题陈述。例如，StayHome公司的数据库解决方案的问题陈述如图6-7所示。当你感到已经有了一个清晰的、不含糊的任务陈述，而且StayHome公司的员工都认同时，就可以开始定义任务目标了。

StayHome的数据库系统的目的是收集、存储和控制公司产生的数据，支持面向会员的录像出租业务、方便分公司之间的合作和信息共享。

图6-7 StayHome公司的数据库应用的任务陈述

2. 创建StayHome数据库系统的任务目标

创建任务目标的过程包括与员工中的合适人选进行的引导性谈话。自由提问通常在这个阶段是最有用的。为了获得完整的任务目标，应该与StayHome中不同角色的人员交谈。可以问的典型的问题如下：

- 请描述你的工作。
- 通常在一天里你要做什么工作？
- 你会和什么数据打交道？
- 你需要使用哪些类型的报告？
- 你要明白哪些事情？
- 公司给你的会员提供哪些服务？

这些问题可以问公司的主管或者经理、监理、助理和采购员。当然，随着采访用户的不同，有必要调整问题。

(1) 主管

开发人员：请问你在公司做哪些工作？

主管：我监督公司的运行，保证我们能不断地为会员提供最好的录像出租服务。

开发人员：你每天要处理什么事情？

主管：我视察我的经理们所管理的每个分公司的运行情况。我必须努力保证各个分公司协调工作，共享录像和会员信息。我也检查公司采购员的工作。他负责给公司所有分公司购买录像。一般来说，每一到两个月我会调查各个分公司的情况，以便非常清楚地了解各个经理的能力。

开发人员：你处理哪些数据？

主管：我需要能够亲自处理公司使用和产生的每一件事情，包括员工名册、录像、出租

业务、会员、录像提供商、录像订单。我的意思是每件事情我都要了解。

开发人员：你需要使用哪种类型的报表？

主管：我要知道各个分公司发生了什么情况？我从各种有关员工、库存录像、录像出租、会员和录像提供商和订单的报告中来获取信息。

开发人员：哪些类型的事务你需要很明白？

主管：就像我前面所说，我要知道每一件事情的来龙去脉。

开发人员：公司为会员提供哪些服务？

主管：我要在全美提供服务最好、价格最便宜的录像出租服务。

(2) 经理

开发人员：你能描述一下你的工作吗？

经理：我的职位是经理。我监督我的分公司的日常运营情况，以给会员提供最好的服务。

开发人员：典型地，每天你要处理什么工作？

经理：我要保证每天24小时部门内都有一定数量的合适的人员在值班，同时要保证有合适的录像数量供顾客选择，虽然我不实际参加录像采购——这由公司采购人员负责。我监控新会员的注册和员工的聘用。

开发人员：你处理哪些数据？

经理：我需要处理员工、录像、出租和会员数据。

开发人员：你使用哪些报表？

经理：有关员工、库存录像、录像出租和会员的各种报表。

开发人员：你需要清楚了解哪些事情？

经理：员工、库存录像、录像出租和会员。

开发人员：你的公司提供给会员哪些服务？

经理：我们在本地区提供给会员最好的录像出租。

(3) 监理

开发人员：请介绍一下你的工作。

监理：我的头衔是监理。我监督一个小组的员工，并且在提供录像出租服务时直接与我们的会员打交道。

开发人员：你每天做哪些事情？

监理：我给员工分配具体的任务，例如和会员打交道、整理录像柜和文件。我回答会员对可以出租的录像的咨询，处理录像租借和归还。我随时更新我们的会员信息，当有顾客想作为一名会员加入时，我帮他们注册。

开发人员：你处理哪些数据？

监理：和员工信息、录像、租借事务以及会员信息相关的数据。

开发人员：你要使用哪些报告？

监理：员工情况报告和库存录像报告。

开发人员：你需要明确哪些事情？

监理：我要知道某些录像是否可以出租，会员的详细情况是否是最新的。

(4) 助理

开发人员：请描述一下你的工作。

助理：我是助理。在提供录像出租服务中，我直接与会员打交道。

开发人员：你每天的日常工作是什么？

助理：我负责回答对可出租录像的咨询。你知道我的意思，他们会问："你们有这种录像吗？"，我处理录像的出租和收回，并把它们重新放到录像柜里。工作不忙的时候，我整理文件。

开发人员：你处理哪些类型的数据？

助理：我处理有关录像、租借业务和会员的数据。

开发人员：你使用哪些报告？

助理：没有。

开发人员：你需要明确哪些事情？

助理：我要知道我们是否有某种录像可以出租。

开发人员：你们公司给会员提供哪些服务？

助理：我们要回答有关库存录像的问题，比如"你们有明星Ewan MacGregor主演的片子吗？"，"《2001 A Space Odyssey》是谁主演的，谁导演的？"，你不能相信我们的会员期望我们要知道什么，但是幸运的是大多数人在这里工作是因为我们真的投入到电影中，所以，即使我不知道，我的同伴也会知道。

(5) 采购员

开发人员：请描述一下你的工作。

采购员：我是采购员，负责为公司的所有分公司购买供出租的录像。

开发人员：你的日常工作是什么？

采购员：我直接与分公司经理和录像提供商打交道。我负责回应部门经理的请求，向他们提供录像。我的工作是在向录像供应商购买录像时获得尽可能低的价位。当然，我也依赖经理们处理好他们的事情——我不希望订购各分公司不需要的录像，也不想发现有分公司没有足够份数的某部流行的电影。有空的时候，我去监测各分公司录像出租的情况，来检查他们是否有合适的录像供用户选择。

开发人员：你处理哪些类型的数据？

采购员：我需要知道分公司信息、录像信息、录像出租信息、会员信息、录像订购信息和供应商的情况。

开发人员：你使用哪种形式的报告？

采购员：我需要我购买的录像的订单报告，其他各种报告有：库存录像情况报告、录像租借情况报告，每个分公司和整个公司的会员情况报告。

开发人员：你要明确了解什么事情？

采购员：我需要最新的录像订单信息，只与不会让我们失望的供应商合作，这是非常重要的。还有，我要知道每个分公司的库存录像和录像出租情况。正如我前面所讲，我不想购买各分公司都不需要的录像。

开发人员：你们公司给会员提供什么服务？

采购员：我们要努力给会员提供最好的录像选择，并且租借费用尽可能低。

这种类型问题的答案，有助于开发人员形成明确的任务目标。例如，StayHome数据库的任务目标如图6-8所示。

维护（录入、更新和删除）各个分公司的数据
维护（录入、更新和删除）有关员工的数据
维护（录入、更新和删除）录像数据
维护（录入、更新和删除）会员数据
维护（录入、更新和删除）录像出租业务数据
维护（录入、更新和删除）录像供应数据
维护（录入、更新和删除）提供录像的订单数据
实现对分公司的查询
实现对录像的查询
实现对员工的查询
实现对录像租借的查询
实现对会员的查询
实现对录像供应商的查询
实现对录像订单的查询
跟踪库存录像库存状态信息
跟踪录像租借状态信息
跟踪录像订单状态信息
报告各分公司情况
报告各个分公司员工情况
报告各个分公司录像情况
报告各个分公司会员情况
报告各个录像租借情况
报告供应商的情况
报告录像订单的情况

图6-8 StayHome 数据库的任务目标

6.4.3 StayHome案例研究——系统定义

系统定义的目的是确定数据库应用的范围和边界以及它的主要用户的视图。一个用户视图代表数据库应用必须支持的由一个特殊的工作角色（如经理或助理）或者业务范围（如录像出租或库存控制）所定义的需求。

1. 定义StayHome数据库系统的系统边界

在数据库系统开发生命周期的这个阶段，开发人员应该和用户交流以澄清和扩展前些阶段所获得的数据。但也可以应用其他事实发现技术，包括检查6.4.1节中显示的案例文档。现在应该分析所收集到的数据来定义应用程序边界。StayHome数据库应用的系统边界如图6-9所示。图中包含的是前面会谈所提到的数据以及这些数据之间的相互关系的粗略向导。

2. 标识StayHome数据库系统的主要用户视图

现在应该对收集到的数据进行分析了，以便定义数据库应用的主要用户视图。有关各个用户视图的大部分数据是在和主管、经理、监理、助理和采购员面谈的时候收集到的。StayHome数据库应用的主要用户视图如图6-10所示。

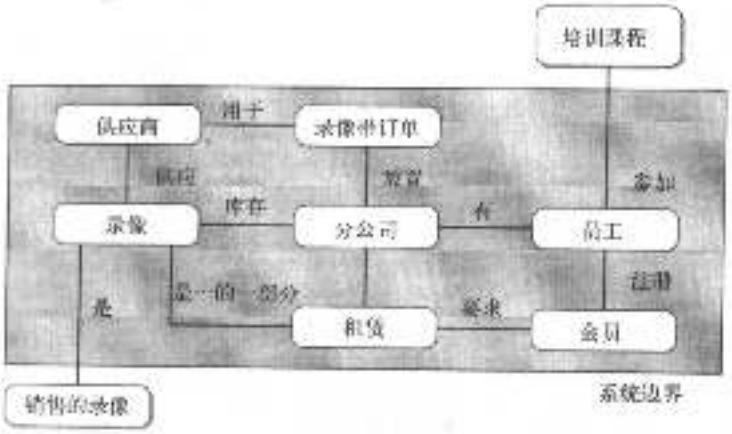


图6-9 StayHome数据库应用的系统边界

用户视图	需 求
Director	报告所有的分公司
	报告在所有分公司的员工
	报告在所有分公司的录像
	报告在所有分公司的会员
	报告在所有分公司的录像租借情况
	报告所提供的录像
	报告录像订单
Manager	维护（录入、更新和删除）给定分公司的数据
	维护（录入、更新和删除）给定分公司的员工的数据
	实现对分公司的检索
	实现对所有分公司的员工的检索
	报告给定分公司的员工
	报告所有分公司的录像
	报告所有分公司的会员
	报告所有分公司的录像租借情况
Supervisor	维护（录入、更新和删除）给定分公司的录像的数据
	维护（录入、更新和删除）给定分公司的会员的数据
	维护（录入、更新和删除）给定分公司的录像租借的数据
	实现对所有分公司的录像的检索
	实现对给定分公司的录像租借的检索
	实现对给定分公司的会员的检索
	跟踪给定分公司的库存中的录像的状态
	跟踪给定分公司的库存中的录像的租借状态
	报告给定分公司的员工
Assistant	维护（录入、更新和删除）给定分公司的录像租借数据
	维护（录入、更新和删除）给定分公司的会员的数据
	实现对所有分公司的录像的检索
	实现对给定分公司的录像租借的检索
	实现对给定分公司的会员的检索
	跟踪给定分公司的库存中的录像的状态
	跟踪给定分公司的库存中的录像的租借状态

图6-10 StayHome数据库应用的主要用户视图

用户视图	需 求
Buyer	维护（录入、更新和删除）录像数据
	维护（录入、更新和删除）录像供应商数据
	维护（录入、更新和删除）录像订单数据
	实现对公司的检索
	实现对所有分公司的录像的检索
	实现对录像供应商的检索
	实现对录像订单的检索
	跟踪录像订单的状态
	报告在所有分公司的录像
	报告在所有分公司的录像租借情况
	报告在所有分公司的会员
	报告录像供应商
	报告录像订单

图6-10 （续）

6.4.4 StayHome案例研究——需求收集和分析

这个阶段，开发人员应该继续收集前面阶段所标识的用户视图的更多细节，产生用户的需求说明。用户的需求说明详细描述了数据库中应该包含的数据以及数据的使用方式。在收集更多的用户视图相关的数据的同时，也应该努力收集系统的一般需求，收集信息的目的是产生系统的需求说明。系统需求说明描述了在新的数据库应用中所要包含的各种特性，如网络需求、共享访问需求、效率需求以及安全级别需求。

在收集用户视图需求的数据和整个系统需求的数据时，开发人员将会了解当前系统的运行方式。当然，我们是正在建立一个新系统，在给新系统引进新的优良特性的同时还应该尽量保留老系统的好的方面。

与此阶段相关的一个非常重要的活动是怎样处理有多个用户视图的情况。我们在4.6节讨论过，有三种主要的方法来处理多用户视图，即集中式方法，视图集成法，以及前面两种方法的综合。这里将简单地告诉大家如何运用这些方法。

1. 收集和StayHome数据库系统的用户视图相关的更多信息

为了找到每个用户视图的更多的需求信息，你可以再次使用一种事实发现技术，如面谈和观察业务操作。例如可以问用户下面这些问题，以了解一个用户视图所需要的数据（用X表示）的情况:

“在X中，你要包含哪些类型的数据？”

“你要对X做那些操作？”

例如，你可以问一个分公司经理如下问题:

开发人员：在员工情况中，你要包含哪些类型的数据？

经理：在员工情况信息中需要包含姓名、职务、薪水。每个员工有员工号，在整个公司是唯一的。

开发人员：你要对员工信息做哪些操作？

经理：我要录入新员工的详细情况，删除离开公司的员工记录。随时更新员工信息，并列表打印我们分公司的员工的姓名、职务、薪水。我需要分配监督人去照看员工们的工作，有时候我想与其他分公司联系，这时候我需要查到其他分公司的经理的名字。

你可以针对数据库中要存储的所有数据问类似的问题，这些问题的回答有助于确定用户需求定义中的必要的细节。

2. 收集StayHome数据库系统的系统需求信息

在和用户商讨用户视图需求的同时，还应该收集关于系统需求的更一般的信息。你可以问如下的问题：

数据库中经常要进行什么操作？

什么事务对这种业务操作是非常关键的？

什么时候运行严格的事务？

它们运行的高峰期、正常期和低谷期各是什么？

数据库系统需要哪种类型的安全机制？

是否存在只能由某些成员使用的敏感数据？

要保存哪些历史数据？

对数据库系统的网络和共享访问有哪些需求？

例如，可以问经理如下问题。

开发人员：数据库中经常要运行什么事务？

经理：我们经常被电话或者会员要求查询某部录像并且看这部录像是否能够租借。当然，录像租出和收回是非常频繁的事情。

开发人员：哪些事务对这些业务操作是关键的？

经理：关键的操作包括：搜索指定录像，录像租出和收回。如果我们不能有效地提供这些最基本的服务，会员就会去其他的地方。

开发人员：什么时候运行这些关键的事务？

经理：每天。

开发人员：对于这种关键的事务，它们运行的高峰期、正常期和低谷期各是什么？

经理：一般来说，早上比较清闲，随着一天时间的推移，业务变得繁忙。每天顾客最多的时候在下午6点~9点。在周五和周六的这段时间内我们甚至不得不增加一倍的值班员工。

你可以问主管如下问题：

开发人员：数据库应用需要哪种类型的安全机制？

主管：我想一个保存录像出租公司的数据库没有什么非常敏感的数据，但是我也不想我们的会员和他们租借的录像被我们的竞争者知道。员工们只需要知道完成他们的工作所需要的数据，并且这些数据的表现形式适合他们的工作。例如，虽然监理和助理有必要知道会员情况，但是会员的记录显示给他们看的时候，只需要一次一条记录，而不能作为一张报表显示。

开发人员：是否存在只能由某些成员使用的敏感数据？

主管：如前面所说，员工们只需要知道完成他们的工作所需要的数据。例如，监督人需要看到员工的详细情况，但是员工的薪水情况应该只有我一个人知道。

开发人员：要保存哪些历史数据？

主管：我希望在会员最后一次租借录像之后，能够保存会员的详细信息两年，这样，我们可以通知他们，告诉他们我们最近的优惠信息来吸引他们回来。我也要保存录像租借信息两年，这样，可以分析找出哪种类型的录像最受欢迎，哪个年龄段的人最经常租录像，等等。

开发人员：对数据库系统的网络和共享访问有哪些需求？

主管：我需要所有分公司和在西雅图的总公司用网络连接，员工可以随时随地访问这个系统。在大部分分公司，我希望大约有2~3名员工可以在任何时候访问这个系统，但是请记住我们有100个分公司，大多数时间，员工们只需访问本地数据。我真的不希望对于本系统被访问的时间和方式有任何限制，除非有什么商业机密。

开发人员：你们需要那种类型的操作失败和数据丢失的保护？

主管：当然是最好的。我们所有的业务都可以用数据库指导完成，所以如果系统不能正常工作，我们就完了。严肃来讲，我认为可以在每晚分公司关门后备份数据，你认为怎样？

你可以针对系统的各个方面问一些类似的问题，这些问题的回答有助于确定系统需求定义中的必要的细节。

3. 管理StayHome数据库系统的用户视图

你怎样决定是用集中式还是视图集成方法来管理多个用户视图？一种帮你做决定的方法是检查在系统定义阶段定义的各个用户视图之间的数据重叠。表6-7交叉引用了主管、经理、监理、助理和采购员的用户视图所使用的主要类型数据（即供应商、录像订单、录像、分公司、员工、租借和会员）。

表6-7　有StayHome数据库系统使用的主要数据类型的用户视图的交叉引用

	Supplier	VideoOrder	Video	Branch	Staff	Rental	Member
Director	×	×	×	×	×	×	×
Manager			×	×	×	×	×
Supervisor			×	×	×	×	×
Assistant			×	×		×	×
Buyer	×	×	×	×		×	×

从这张表可以看出在所有的用户视图之间有数据重叠。然而，主管和采购员视图对附加数据的需求（即供应商和录像订单）和其他视图是不同的。基于这个分析，我们应该使用集中式方法首先把主管视图和采购员视图的需求合并起来（给定集合名字为业务视图），把经理、采购员和助理的视图的需求合并起来（命名为分公司用户视图）。然后开发代表业务视图和分公司用户视图的数据模型，再使用视图集成的方法合并这两个数据模型。当然，像StayHome这样一个简单的示例，可以对所有的用户视图轻松使用集中式方法，但是为了在实践中演示集中式和视图集成两种方法，我们仍决定创建两个视图来标识StayHome的两个集合用户。

对于什么时候使用集中式方法，什么时候使用视图集成方法，很难给出一个准确的区分规则。作为数据库开发人员，你应该根据对数据库系统的复杂性的估计和不同视图的数据重叠程度来作决定。无论使用集中式方法，还是视图集成方法，甚至两种方法的结合来建立支撑数据库，最终都需要为工作数据库系统创建原始的用户视图。我们将在第14章讨论数据库用户视图的建立。在本章的余下部分，我们提供了StayHome的每个分公司用户视图的用户需求定义，以及数据库系统的系统说明。

4. 创建StayHome数据库系统的各分公司用户视图的用户需求定义

各分公司用户视图的用户需求定义作为两个部分列出：第一部分描述分公司用户视图使用的数据。第二部分提供了数据怎样被分公司用户视图使用的例子（即员工在数据上执行的事务）。

(1) 数据需求

StayHome的一个分公司的数据包括分公司地址，地址由街道、城市、州、邮政编码和电话号码组成，电话号码最多3行。每个分公司有一个给定的名称，在全公司是唯一的。

StayHome的每个分公司都有若干员工，包括一个经理，一到多名监理和一些其他员工。经理负责分公司的日常运营，监理负责监督员工，员工数据包括姓名、职务、薪水。每位员工有员工号码，在全公司是唯一的。

每个分公司有录像库存。录像数据包括目录号、录像号、片名、种类、日租费用、购买价格、状态、主要演员名字（以及扮演的角色）和导演。目录号唯一地表示一盘录像。通常一个分公司有一盘录像的多份拷贝，每个拷贝由录像号唯一地表示。录像的种类有动作、成人、儿童、恐怖、科幻。状态指出一盘录像的某份拷贝是否可以出租。

在从公司租借录像之前，客户必须首先注册成为StayHome当地分公司的一名会员。会员信息包括姓名、地址和注册日期。每个会员有会员号，会员号对所有分公司是唯一的，而且可以在多个分公司使用同一会员号注册。负责注册该会员的员工的姓名也要加上。

注册后，会员可以租借录像，一次最多租借10部。租借业务数据包括租借号、会员的全名、会员号、录像号、片名、每日费用、租出的日期和归还的日期。租借号在整个公司是唯一的。

(2) 事务需求

数据录入

a) 录入一个新分公司的详细情况。

b) 录入某分公司的新员工的详细情况（如分公司B001的Tom Daniels）。

c) 录入新发行的录像的详细情况（如《Return of the King》的详细情况）。

d) 录入给定分公司的某部新录像拷贝的详细情况（如分公司B001的《Return of the King》的三份拷贝）。

e) 录入某分公司的新会员的详细情况（如Bob Adams在分公司B002注册）。

f) 录入会员录像的租借协议的详细情况（如会员Don Nelson在2004年5月4日租借《Return of the King》）。

数据更新/删除

g) 更新/删除分公司信息。

h) 更新/删除某分公司的员工信息。

i) 更新/删除给定录像的信息。

j) 更新/删除给定录像的某份拷贝的信息。

k) 更新/删除给定会员的信息。

l) 更新/删除某会员租借某部录像的租借协议的信息。

数据查询

数据库必须支持下列查询:

m) 列出给定城市的分公司情况。

n) 按照员工的名字顺序列出指定分公司的员工名称、职务、薪水。

o) 按照分公司号顺序列出每个分公司的经理名称。

p) 分类列出某分公司的录像名称、种类和可租借情况。

q) 按照片名顺序列出某分公司指定演员的录像名称、种类和是否可以租借。

r) 按照片名顺序列出某分公司指定导演的录像名称、种类和是否可以租借。

s) 列出某个会员租借的全部录像的详细情况。

t) 列出某个分公司指定录像的拷贝情况。

u) 列出指定种类的所有录像名称，按片名排序。

v) 列出每个分公司每种录像的数量，按照分公司号排序。

w) 列出所有分公司的录像的价值。

x) 列出每个演员的录像数量，按照演员名字排序。

y) 列出每个分公司在1999年注册的会员的数量，按照分公司号排序。

z) 列出每个分公司可能的租金收入，按照分公司号排序。

5. 创建StayHome的数据库系统的系统说明

系统说明应该列出该数据库系统的所有的重要特点，应该在系统说明中描述的特点举例如下:

- 初始数据库大小。
- 数据库增长速度。
- 记录查找的类型和平均数量。
- 网络和数据共享需求。
- 性能。
- 安全性。
- 备份和恢复。
- 用户界面。
- 合法问题。

初始数据库大小

a) 大约有20000种录像和400000盘供出租的录像分布在超过100个分公司中。每个分公司平均有4000盘录像，最多有10000盘供出租。

b) 大约有2000名员工工作在各个分公司。每个分公司平均有15名员工，最多有25名。

c) 大约有100000名会员在各个分公司注册。每个分公司平均有1000名会员，最多有1500名会员。

d) 各个分公司大约有400000盘录像供出租。平均每个分公司有4000到10000盘录像。

e) 大约有1000名导演和30000名主要演员出演60000个角色。

f) 大约有50个录像供应商和1000盘录像订单。

数据库增长速度

a) 每月大约有100部新片，每部录像有20份拷贝加到数据库中。

b) 一旦某盘录像的一份拷贝不能再租借出去（如画面质量差、丢失、被偷），则相应的记录从数据库中删除。每月大约有100个这样的记录。

c) 每月会有20名员工加入或者离开。离开公司一年的员工记录从数据库中删除。每月大约删除20条员工记录。

d) 每月大约有1000名新会员注册。如果一个会员两年没有租借任何录像，将删除该会员记录。每月大约有100条会员记录被删除。

e) 每天各分公司总共有5000条新的录像出租记录。录像出租记录在创建两年后被删除。

f) 每个星期大约有50份新的录像购买订单。订单记录在创建两年后被删除。

记录查找的类型和平均数量

a) 查询分公司的详细情况——大约1天10次。

b) 查询一个分公司的员工详细情况——大约每天20次。

c) 查询指定录像的情况——每天约5000次（周日到周四），10000次（周五和周六）。每天下午的6点~9点是高峰时期。

d) 查询某盘录像的某份拷贝的情况——每天约10000次（周日到周四），20000次（周五和周六）。每天的上午6点~9点是高峰时期。

e) 查询指定会员的详细情况——每天大约100次。

f) 查询会员租借录像的详细情况——每天约10000次（周日到周四），20000次（周五和周六）。每天的下午6点~9点是高峰时期。

网络和共享访问需求

a) 所有分公司都必须安全地和位于西雅图公司总部的中央数据库实现网络互连。

b) 系统必须能够支持每个分公司的至少三名成员同时访问。需要考虑这么大数量并发访问的许可需求。

性能

a) 在上班时间但不是高峰期的时候要求单个记录的搜索时间少于1秒，在高峰期各种搜索响应时间少于5秒（每天下午6点~9点）。

b) 在上班时间但不是高峰期的时候要求多条记录的搜索时间少于5秒，在高峰期各种多记录搜索响应时间少于10秒（每天下午6点~9点）。

c) 在上班时间但不是高峰期的时候要求更新/保存记录的时间少于1秒，在高峰期少于5秒（每天下午6点~9点）。

安全性

a) 数据库必须有口令保护。

b) 每个员工应该分配一个到特定用户视图的数据库访问权限，主要是主管、经理、监理、助理和采购员。

c) 员工只能在适合他们完成工作的需要的窗口中看到需要的数据。

备份和恢复

数据库必须在每天半夜12点备份。

用户界面

用户界面必须是菜单驱动的，联机帮助应该易于查找和使用。

法律问题

每个国家都有法律管理个人数据的计算机存储的方式。作为容纳员工和会员信息的StayHome数据库，必须调查并且实现所要遵守的法律。

6.4.5 StayHome案例研究——数据库设计

本章我们说明了分公司用户视图的用户需求说明的创建和StayHome数据库系统的系统说

明的创建。这些文档是下一个阶段——数据库设计的信息源。在第9章、第10章和第12章～第16章，我们将一步一步提供数据库设计的方法学，并且使用本章创建的有关StayHome案例研究的文档来从实践上说明这些方法学。

如果你对开发复杂的多用户视图的数据库系统感兴趣，我们在附录C中演示了使用StayHome的分公司和业务用户视图时视图集成的方法工作情况。

6.5 本章小结

- 事实发现是使用面谈和问卷调查等技术来收集有关系统、需求和偏好的事实的形式化处理过程。
- 事实发现对于数据库系统开发生命周期的早期阶段，包括数据库规划、系统定义、需求收集和分析，都是至关重要的。
- 五个最常用的事实发现技术是检查文档、面谈、观察业务处理、研究和问卷调查。
- 数据库规划阶段的第一步是清楚定义数据库工程的任务陈述和任务目标。任务陈述定义了数据库系统的主要目的。每个任务目标必须确定一个数据库必须支持的特定任务。
- 系统定义阶段的目的是定义数据库系统的边界和用户视图。
- 在需求收集和分析阶段要产生两种主要文档，即用户需求说明和系统说明。
- 用户需求说明详细描述了数据库中的数据和数据的使用。
- 系统说明描述了数据库系统的所有特征，如要达到的性能和安全级别。

复习题

6.1 简要叙述事实发现达到数据库开发人员要求的过程。

6.2 叙述事实发现是如何贯穿在数据库系统开发生命周期的各阶段的。

6.3 对于数据库系统开发生命周期的每个阶段，确定事实捕获的示例和产生的文档。

6.4 在一个数据库项目中，数据库开发人员通常使用几种事实发现技术，五种最常使用的技术是检查文档、面谈、观察操作中的业务、进行研究和使用问卷形式。描述每种事实发现技术并确定每种技术的优缺点。

6.5 描述为数据库系统定义任务说明和任务目标的目的。

6.6 系统定义阶段的目的是什么？

6.7 用户的需求说明和系统说明在内容上有何差别？

6.8 描述在开发多用户视图的数据库系统时，怎样确定使用集中化方法、视图集成方法或两者的结合的方法？

第7章 实体-关系建模

本章主题:

- 在数据库设计中怎样使用ER建模。
- ER模型中的基本概念：实体、关系和属性。
- 显示ER模型的图形技术。
- 怎样辨别和解决在ER模型中可能出现的连接陷阱的问题。

在第6章，我们学习了收集和捕获用户对数据库应用程序的需求信息的技术。一旦完成了数据库系统开发生命周期中的需求收集和分析阶段的工作，并且把数据库需求整理成了文档，就可以开始进行数据库设计了。

数据库设计最困难的一个方面就是设计人员、程序员和最终用户看待和使用数据的方式不同。遗憾的是，除非我们能够获得反映公司运行的共性的理解，否则，我们的设计将达不到用户的要求。为了保证能够准确理解数据的本质，理解公司使用这些数据的方法，我们需要有一个通信模型，这个模型和技术实现无关，且没有二义性。ER（Entity-Relationship, 实体－关系）模型就是一个这样的例子。自从ER模型在1976年问世之后，已经扩展到包含一些高级建模概念。这一章将学习基本的ER模型概念，在第11章介绍一些更流行的概念。

ER建模是一种自上而下的数据库设计方法。我们通过标识模型中必须要表示的重要数据（叫做实体）以及数据间的关系开始ER建模，然后增加细节信息，如实体和关系所要具有的信息（叫做属性），以及实体、关系和属性上的限制。

整个这一章，我们介绍建立一个ER模型的基本概念。虽然，对各个概念的含义有一些泛化协定，但在一张图中，仍然有不同的方式来表示同一个概念。我们选择使用图形化的表示法，它使用更常用的面向对象建模语言，称为UML。其他的ER模型使用的符号在附录A中列出。

> UML（Unified Modeling Language，统一建模语言）是20世纪80年代和90年代产生的许多面向对象分析和设计方法的继承者，它是标准建模语言。

由于ER模型构成了第9章、第10章和第12章~第16章介绍的方法学的基础，因此本章可能将是本书最重要的章节之一。如果你不能马上理解这些概念，不要着急，再阅读一遍，然后看看我们在方法中提供的例子以获得更多的帮助。让我们先开始介绍ER模型的基本概念，即实体、关系和属性。

7.1 实体

> **实体**（Entity） 一组有相同属性的对象，被用户或者团体标识为独立存在的对象的集合。

ER模型的一个基本概念是实体，代表一组现实世界中的对象集合，它们有相同的属性。每个对象必须在集合中被唯一地标识表示，叫做实体事件（entity occurrence）。一个实体是独立存在的，代表一组物理存在（现实）或者概念存在（抽象）的对象的集合，如图7-1所示。

我们用一个唯一的名字和一些特征（叫做属性）来标识每个实体。虽然一个实体有不同

的属性集合，但是每个实体对每个属性都有自己的值。一个数据库通常包含许多不同的实体。

物理存在	概念存在
会员	角色
录像带	租赁
分公司	注册

图7-1　具有物理存在和概念存在的实体的例子

实体的图形化表示

每个实体用矩形来表示，在矩形框中写上实体的名字，名字通常是一个名词。在UML中每个单词的第一个字母大写。图7-2显示了实体Video、Role、Actor的图形化表示。

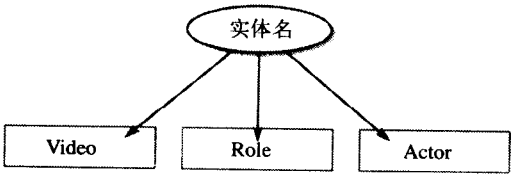


图7-2　实体Video、Role、Actor的图形化表示

7.2 关系

关系（Relationship）　实体之间的具有某种含义的关联。

一个关系是特定实体之间的一组关联。和实体一样，每个关系在集合中也要被唯一地标识。一个可以唯一标示的关联叫做关系事件（relationship occurrence）。

每个关系有一个名字，此名字描述关系的功能。例如，实体Actor与实体Role通过关系Play关联，实体Role与实体Video通过关系Features关联。

关系的图形化表示

每个关系显示为连接关联实体的一条线，并用关系的名字标记。通常，关系用动词（如Plays或Features）或者动词短语（如IsPartOf或WorksAt）命名。关系名短语的第一个字母大写。只要有可能，在给定的ER模型中，关系名应该唯一。

一个关系仅在一个方向标记，代表关系仅在一个方向起作用（例如Actor Plays Role有意义，但是Role Plays Actor没有）。因此，一旦选择了关系名，必须在名字旁边加箭头指示正确的方向，以便给读者解释关系名的意义。图7-3展示了关系Video Features Role 和 Actor Plays Role的图形化表示。

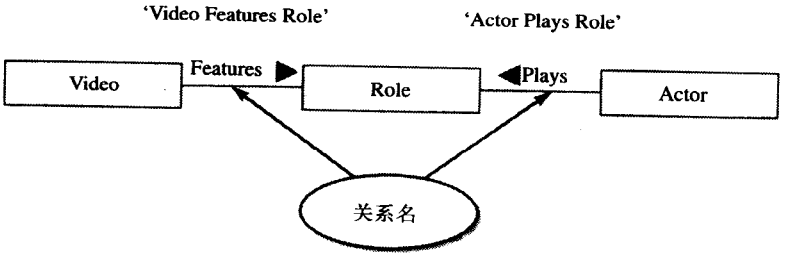


图7-3　Video Features Role和Actor Plays Role关系的图形化表示

7.2.1 关系的度

关系的度（Degree of a Relationship） *在关系中参与的实体的数目。*

包含在特定关系中的实体叫做参与者。在关系中参与者的数目叫做关系的度，它指明了参与在一个关系中的实体的数目。度数为1的关系称为一元关系，它通常被称为递归关系。在后面的几节中我们将详细讨论这种关系。一个度数为2的关系叫做二元关系，在图7-3中显示的两个关系的度数都是2。度数超过2的关系称为复杂关系。

一个度数为3的关系叫做三元关系，例如有三方实体Branch、Staff、Member参与的Registers，如图7-4所示。这个关系的目的是表示一个员工在某个部门注册一个会员，允许会员在多个部门注册，而且会员可以在分公司间移动。

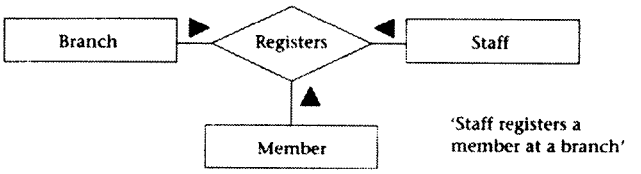


图7-4 名为Registers的三元关系的例子

一个度为4的关系叫四元关系，更高度的关系叫做n元关系。我们遇到的最多的关系类型是二元关系，但是偶尔也会遇到三元和四元关系。

7.2.2 递归关系

递归关系（Recursive Relationship） *在不同的角色中有多次具有相同实体参与的关系。*

让我们考虑一个叫Supervises的递归关系，此关系代表员工和监理之间的关系。在这里，监理也是员工中的一员。换句话说，Staff实体参与Supervises关系两次：第一次作为监理参与，第二次作为一名被管理的员工参与。如图7-5所示。

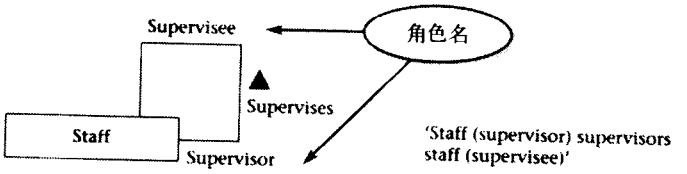


图7-5 名为Supervises 的递归关系的例子

可以给关系一个角色名字以表明每个参与的实体在关系中所起的作用。角色名字对于递归关系决定参与实体的作用是非常重要的。图7-5显示了使用角色名字来描述Supervises递归关系。在Supervises关系中，实体Staff的第一次参与给定的角色名称为Supervisor，第二次参与给定的名称为Supervisee。

7.3 属性

属性（Attribute） *实体或者关系的性质。*

实体的特定性质叫做属性。属性代表我们需要知道的有关实体的内容。例如，实体Video

可以通过catalogNo、title、category、dailyRental和price等属性来描述。这些属性的值描述了每个录像的现状，并代表了存储在数据库中的数据的主要来源。

实体之间的关系也可以有和实体类似的属性，但是我们将推迟到7.6节再讨论关系的属性。

现在我们讨论的属性可以分为：简单属性和复合属性，单值属性和多值属性，还有派生属性。

7.3.1 简单属性和复合属性

简单属性（Simple Attribute） 仅由单个元素组成的属性。

简单属性不能被进一步划分。例如实体录像的category和price属性就是简单属性。简单属性有时候叫做原子属性。

复合属性（Composite Attribute） 由多个元素组成的属性。

复合属性可以被进一步划分为多个独立存在的更小的元素。例如，Member实体的name属性，具有值为Don Nelson的时候，可以被划分为fName（'Don'）和lName（'Nelson'）。

把name属性建模成简单属性还是分解成fName和lName的复合属性，取决于用户访问name是作为一个整体单元还是单个组成元素访问。

7.3.2 单值属性和多值属性

单值属性（Single-Valued Attribute） 对一个实体只有一个值的属性。

对于一个具体的实体来说，大多数属性是单值属性。例如，每个Video实体的出现只有一个catalogNO属性值（例如，207132），因此被看做单值属性。

多值属性（Multi-Valued Attribute） 对于一个实体可以有多个值的属性。

有些属性对某个特定实体具有多个值。例如，对每个Video实体category属性可能有多个值，（例如'Children'和'Comedy'），因此，在这里category就是多值属性。一个多值属性可以有一组具有指定上下限的值。例如，category属性有1到3个值。

简单和复合属性，单值和多值属性的分类不是相互排斥的。换句话说，属性可能有简单的单值属性、复合的单值属性、简单的多值属性和复合的多值属性。

7.3.3 派生属性

派生属性（Derived Attribute） 属性值代表一个从相关属性或者属性集派生的值，在相同实体中不是必须的属性。

有些属性可能和一个特定实体相关。例如，一位员工的年龄（age）是从他的出生日期（DOB）导出的，因此age和DOB属性是相关的。我们把年龄属性看作派生属性，它的值从出生日期得到。

提示 年龄通常不存储在数据库中，因为必须要对这个值进行有规则的更新。另外，由

于生日不会改变，因此年龄可以从生日属性导出，所以可以存储生日，然后，当需要的时候，将年龄从生日属性中导出。

在某些情况下，一个属性的值从一个实体中的值得到，例如年龄。但是某些情况下，属性值可能要从多个实体中才能得到。

7.3.4 键

在2.2.3节，我们介绍了与表相关的键的概念，这些概念同样适用于实体。

超键（Superkey） 可以唯一标识一个实体的属性或者属性组。

候选键（Candidate Key） 可以唯一标识一个实体的最小数目的属性的超键。

主键（Primary Key） 被选中作为标识实体的候选键。

备用键（Alternate Key） 没有被选为主键的候选键。

例如，branchNo（分公司号码）和zipCode（分公司的邮政编码）是实体Branch的候选键，因为对这两个值，每个branch实体的存在都有不同的值。如果我们选择branchNo为主键，那么zipCode就成为备用键。

属性的图形化表示

如果实体和属性一起显示，那么我们就把代表实体的矩形分成两部分。上半部分显示实体名，下半部分列出属性名。例如，图7-6显示了Video、Role和Actor实体以及它们的属性的ER模型。

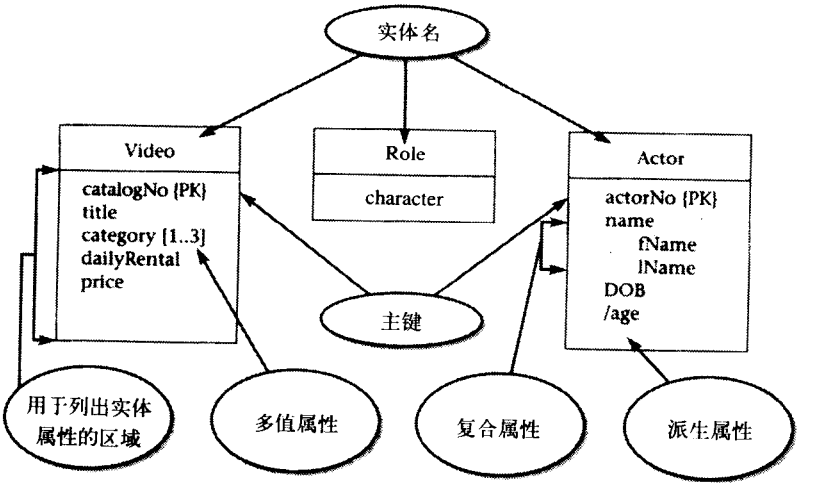


图7-6 Video、Role和Actor实体的属性的图形化表示

如果已经知道了主键，则第一个被列出的属性是实体的主键，主键属性的名字旁可以加标记{PK}。在UML中，属性名的第一个字母是小写，如果名字由几个单词组成，则接下来的每个单词的第一个字母用大写（例如，character、actorNo、catalogNo）。可以使用的附加标记包括部分主键{PPK}，这表明这个属性仅仅是复合主键的一部分，还有备用键标记{AK}。

为简单起见，对单值属性，没有必要使用标记，我们只要简单地在实体名字的下边把属性名显示成一个列表即可。

对于复合属性，可以列出复合属性的名字，后面跟着组成元素的名字，例如，图7-6显示的复合属性name后面紧跟着组成姓名的属性fName和lName。

对于多值属性，我们标记出属性名，并且指出适用于该属性的取值范围。例如，我们用[1..*]标记category属性，其意思是属性category的值有一个到多个。如果我们知道属性值的准确的最大数目，我们可以用一个精确的值来标记属性名。例如，如果属性category可以有一个到最多三个值，我们可以用[1..3]来标记它。

对于派生属性，我们在属性的名字前面加前缀“/”，例如，图7-6中用/age显示派生属性age。

注意 没有为Role实体指定主码。主键的存在或不存在允许我们表示实体是强实体还是弱实体。下面讨论强实体和弱实体。

对于简单的数据库应用，可以为数据模型的每个实体显示属性。但是对于复杂的数据库应用，通常只需显示主键。当在ER模型中只显示主键属性时，可以不使用{PK}标记。

7.4 强实体和弱实体

我们可以把实体分为强实体和弱实体。

强实体（Strong Entity） 不依赖于另一个实体的主键的实体。

弱实体（Weak Entity） 部分或全部依赖于其他一个或多个实体的主键而存在的实体。

例如，不用任何其他实体的存在，我们就可以把一个演员（actor）和其他演员区分开，把一部影片（video）和其他影片区分开，因此Actor实体和Video实体是强实体。换句话说，Actor实体和Video实体是强实体是因为他们有自己的主键，如图7-6所示。

图7-6中也有一个弱实体的例子叫做Role，代表演员在影片中扮演的角色。如果没有Actor实体和Video实体的存在就不能唯一标识一个Role实体的存在，我们把Role实体叫做弱实体。即，Role实体是弱实体是因为它没有自己的主键。

强实体有时候被叫做父实体、拥有者实体或者统治实体，而弱实体则相应被叫做子实体、依赖实体或者从属实体。

7.5 关系的多样性约束

现在我们检查可以放置到参与关系的实体上的约束。这种约束的例子包括：一个分公司必须有多名会员，每个分公司都必须有员工。关系上的主要的约束类型称为多样性。

多样性（Multiplicity） 一个实体中可能和相关实体的一个存在关联的实体事件的数目。

多样性约束了通过特定关系关联其他实体的实体事件的数目。多样性代表了用户或者公司建立的策略，被称为业务规则（business rule）。这些规则确保所有合适的事务规则都被标识并且被表达是为公司建模的一个重要部分。

前面我们提到过，最常用的关系是度为2的二元关系。二元关系上的多样性约束一般被叫做一对一（1:1）、一对多（1:*）或者多对多（*:*）。我们用下面的业务规则来考察这三种类型的关系。

- 一名员工管理一个分公司。

• 一个分公司有许多员工。

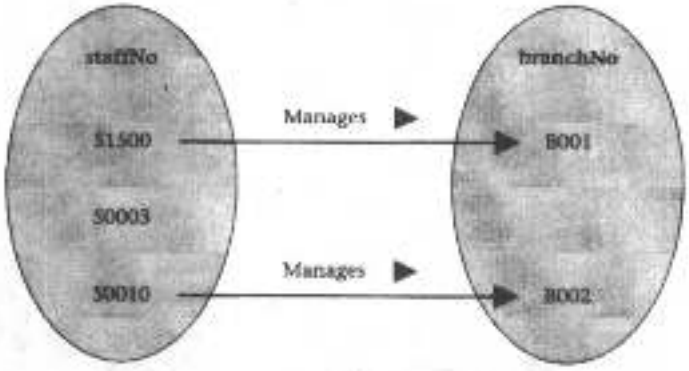
• 演员出演某部影片。

对每个业务规则，我们说明怎样在没有明确指定的情况下找到其中的多样性约束，并说明怎样在ER模型中表示它们。在7.5.4节中，我们将看到度数更高的关系的多样性约束。

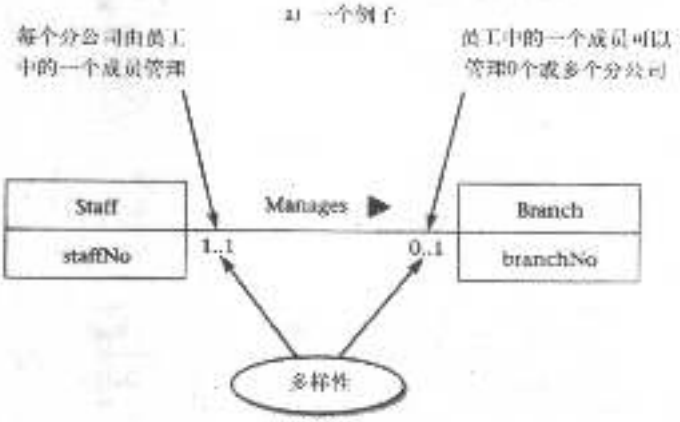
> 一定要注意不是所有的业务规则都可以非常容易的在ER模型中清晰地表示出来的。例如，一名员工每年有一天的额外假期的需求就很难清楚地在ER模型中表示。

7.5.1 一对一关系

让我们考虑关系Manages，它表示实体Staff和Branch之间的关系。图7-7a用Staff和Branch实体的主键属性值显示了Manages关系的一个例子。



a) 一个例子



b) 多样性

图7-7 Staff Manages Branch关系

1. 找出多样性

找出多样性通常需要用样例数据检查特定事务规则的数据之间的关系。我们可以从填好的表格、报表甚至从与用户的讨论中得到样例数据。为了得到正确的结论，强调被检查和讨论的样例数据能够真实地代表所有数据是非常重要的。

在图7-7a中，我们可以看到staffNo为S1500管理branchNo为B001的分公司，并且staffNo为

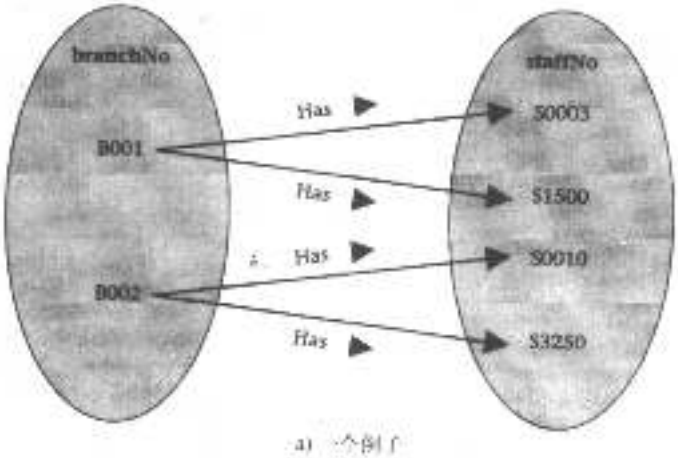
S0010管理branchNo为B002的分公司，但是S0003没有管理任何分公司。换句话说，一名员工可以管理0到1个分公司，一个分公司仅由一名员工管理。由于在这个关系中，对每个员工最多只有一个分公司，而对每个分公司最多只有一名员工，我们把这种类型的关系叫做一对一关系，简写为1：1。

2. 1：1关系的图形化表示

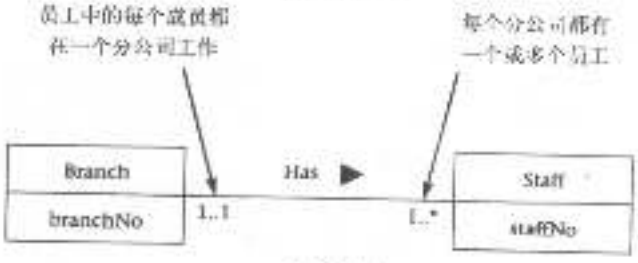
Staff Manages Branch关系的ER模型见图7-7b，为了表示一名员工可以管理0到1个分公司，我们在从Staff实体开始的关系的另一端放置一个"0..1"，为了表示一个分公司只有一名经理，我们在从Branch实体开始的关系的另一端写上"1..1"（注意，对于一个1:1关系，我们也可以选择在两个方向上都有意义的关系名）。

7.5.2　一对多关系

让我们考虑叫做"有"（Has）的关系，同样还使用Branch和Staff实体间的关系。图7-8a使用Staff和Branch实体的主键属性值显示了Branch Has Staff关系的一个例子。



a) 一个例子



b) 多样性

图7-8　Branch Has Staff(1:*)关系

1. 我多样性约束

在图7-8a中，我们看到branchNo为B001的分公司有员工S0003、S1500，而branchNo为B002的分公司有员工S0010、S3250，因此每个分公司都有一到多名员工，每名员工只在一个分公司工作。由于一个分公司可以有多名员工，所以我们把这种类型的关系叫做一对多关系，简写为(1:*)。

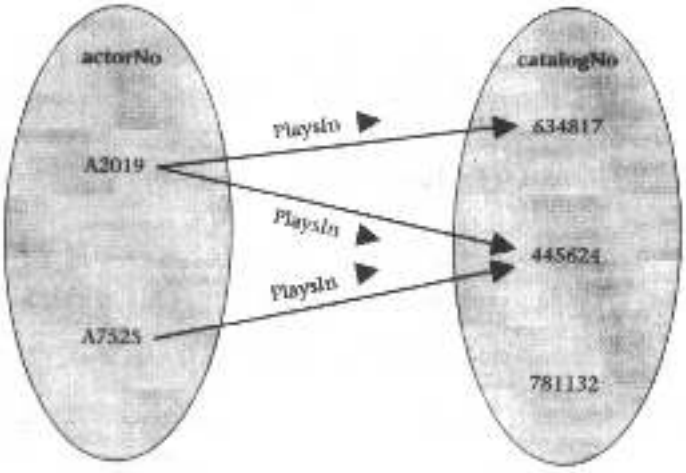
2. 1:*关系的图形化表示

关系Branch Has Staff的ER模型如图7-8b所示。为了表示每个分公司都有一到多个员工，我们在Staff端放一个标记“1..*”，而在Branch端放一个“1..1”表示一名员工只能在一个分公司工作（注意，在1:*关系上，我们选择在1:*方向上有意义的名字）。

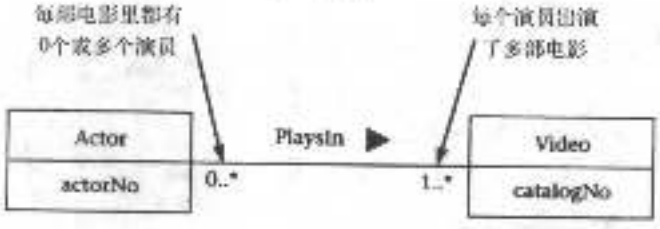
提示 如果你知道多样性约束的最大值和最小值，则可以显示这些信息。例如，假如一个分公司有2~10名员工，则可以用“2..10”代替“1..*”。

7.5.3 多对多关系

现在考虑PlaysIn关系，它关联实体Actor和Video。图7-9用Actor和Video实体的主键属性值表示了Actor PlaysIn Video关系的例子。



a) 一个例子



b) 多样性

图7-9 Actor PlaysIn Video(*:*)关系

1. 找多样性约束

在图7-9a中，我们可以看到actorNo为2019的演员在catalogNo为634817和445624的录像中出演过，ActorNo为7525的演员在catalogNo为445624录像中出演过。换句话说，一名演员可以在一到多部影片中出演角色，我们也可以看到catalogNo为445624的录像有两名演员，但是catalogNo为781132的录像中没有任何演员，我们可以推断一部影片中可以有0到多个演员。

总之，从实体Actor和Video两个实体视角来看，关系PlaysIn都是1:*的。我们用两个方向

上的1:*来代表这种关系，总体上叫做多对多关系，简写为*:*。

2. *:*关系的图形化表示

关系Actor PlaysIn Video的ER模型如图7-9b所示。为了表示每个演员都可以在多部电影中出演，我们在从Actor实体开始的关系的另一端放上标记“1:*”。为了表示每部影片有0到多个演员，我们在从Video开始的关系的另一端放上标记“0:*”。

7.5.4　复杂关系的多样性约束

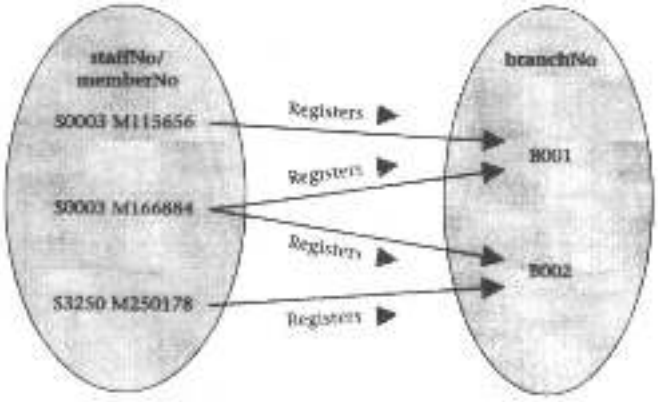
多元关系中的多样性约束稍微复杂一点。例如，三元关系的多样性约束表示在关系中当其他两个实体事件的数目固定的情况下，另一个实体的潜在的数目。让我们再考虑图7-4表示的由实体Branch、Staff和Member参与的三元关系Registers。图7-10a表示了当Staff和Member的值固定的情况下Registers关系的例子。

找出多样性

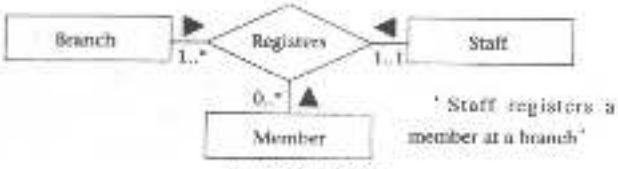
在图7-10a中，我们可以看到对于每个staffNo/memberNo的值对的组合，总是至少有一个对应的branchNo值。当值对staffNo/memberNo为S0003/M166884时对应branchNo B001和B002。这表示会员M166884已经在分公司B001由员工S003注册，接着又在B002分公司由同一名员工注册，说明这名员工已经由B001分公司调到B002分公司。换句话说，从Branch的观点来看多样性约束是1:*。

如果我们从实体Staff的观点重复这个检验，我们发现这个关系的多样性约束是1:1。从Member的观点是0..*。显示Registers关系的多样性约束的ER模型如图7-10b所示。

> 一般来说，n元关系的多样性约束代表关系中其他n-1元固定的情况下某个实体潜在的实体事件的数目。



a) 一个例子



b) 关系的多样性

图7-10　在Staff和Member值固定的情况下来自Branch的三元Registers关系

表示多样性约束的所有可能的方法的总结见表7-1，此表还描述了每个方法的含义。

表7-1 表示多样性约束的方法总结

表示多样性约束的方法	含 义
0..1	0或者1个实体事件
1..1（或仅1）	1个实体事件
0..*（或仅*）	0或者多个实体事件
1..*	1个或者多个实体事件
5..10	最少5个最多10个实体事件
0、3、6~8	0个、3个、6、7、8个实体事件

7.5.5 基数约束与参与约束

实际上，多样性约束由基数约束和参与约束组成。

基数（Cardinality） 描述每个可能参与实体的关系的数目。

参与（Participation） 决定是否所有的或仅有的某些实体在关系中发生参与。

二元关系的基数就是我们所说的一对一、一对多、多对多。参与约束表示是否所有的实体事件都参与关系（强制参与）或者只有部分参与（可选参与）。在图7-11中，我们展示了在图7-7b中的关系Staff Manages Branch的基数和参与约束。在逻辑数据库设计中我们将使用参与约束来决定:

- 怎样为一对一关系创建表（见步骤2.1）。
- 外键是否可以为空（见步骤2.4）。

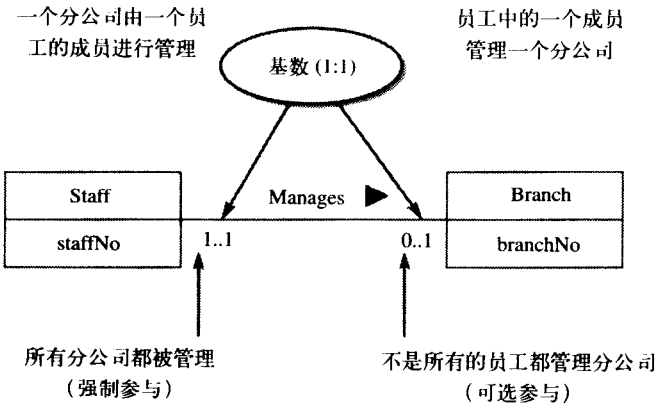


图7-11 以关系Staff Manages Branch（图7-7b中显示的1:1关系）的基数和参与约束的形式显示的多样性

7.6 关系上的属性

在7.3节中我们简单的提到，关系也可以有属性。例如，考虑PlaysIn关系，它关联实体Actor和Video，我们也可能关心演员在影片中所扮演的角色。这个信息是和PlaysIn关系相关

联而不是和实体Actor或者Video关联。我们创建了一个叫做character的属性来存储这个信息，并把它分配到PlaysIn关系中，如图7-12所示。注意，在图中，character属性是用实体的符号表示的。为了区分实体和关系的属性，代表属性的矩形与关系用虚线连接。

与关系相关的一个或者多个属性也许说明关系隐藏了一个没有识别出的实体。例如，在前面的图7-6中显示的实体Role相关联的属性character。

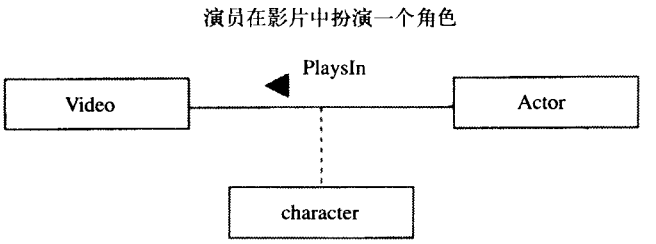


图7-12　具有属性character的PlayIn关系

7.7　ER模型中的设计问题

本节我们说明设计一个ER模型可能会出现的两个问题。这些问题总体上被称为连接陷阱，通常是由于对某些关系的含义的误解而产生的。连接陷阱的主要两种类型是扇形陷阱和深坑陷阱。我们将举例说明怎样识别和解决这类问题。

总之，为了识别连接陷阱，我们必须确信关系的含义(它表示的业务规则)被完全理解并且清楚地定义。如果我们不能理解关系，可能就会建立一个不能代表现实世界的模型。

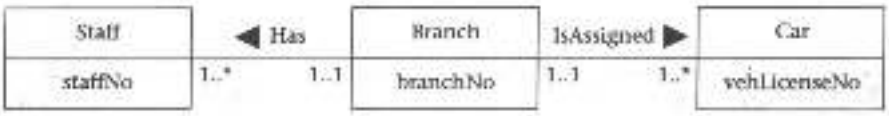
7.7.1　扇形陷阱

扇形陷阱（Fan Trap）　从第三个实体扇出的两个实体有1:*关系，但这两个实体之间应该有直接关系以提供必要的信息。

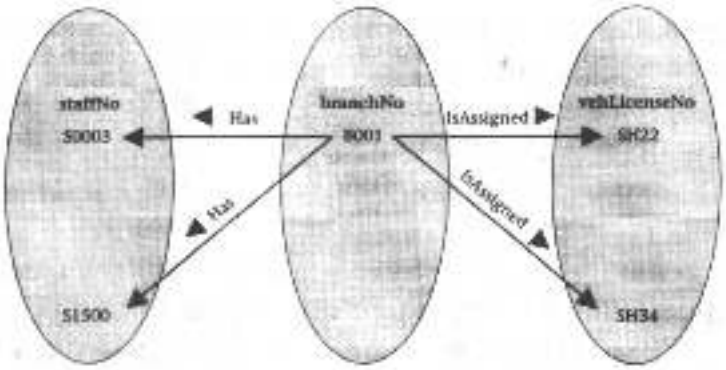
扇形陷阱存在于从同一个实体扇出的两个或者多个一对多关系中。图7-13a说明了一个潜在的扇形陷阱，它显示了两个1:*关系(Has 和IsAssigned)，它们从同一个实体Branch射出。这个模型告诉我们一个分公司有许多员工并且分配了许多车。然而，如果我们想知道哪个成员使用哪辆车，问题就产生了。为了理解这个问题，让我们检查Has和IsAssigned关系的一些例子。这些例子使用的实体分别用Branch、Staff和Car的主键属性值来表示，如图7-13b所示。

假如我们试着回答这个问题：哪名员工使用SH34号汽车，用当前的结构是不可能给出一个明确的答案的。我们只能决定汽车SH34分配给了B001分公司，但是不能判断员工S0003和S1500是否使用这辆汽车。不能明确地回答这个问题是因为扇形陷阱的缘故。

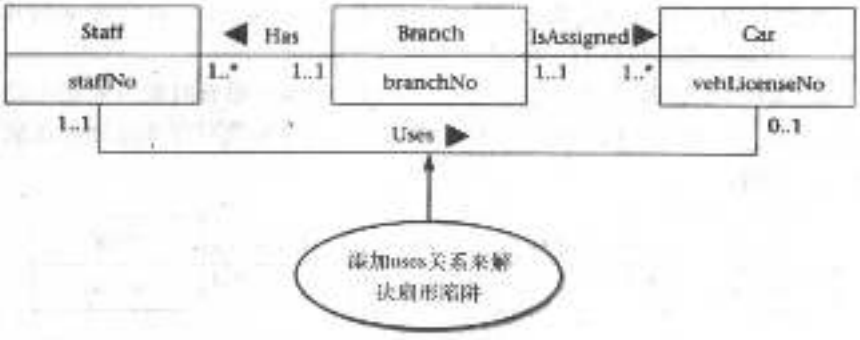
通过给ER模型增加一个Staff Uses Car关系，如图7-13c所示，我们可以解决这个问题，现在检查Has、IsAssigned和Uses关系的例子，我们可以看到S1500使用SH34，如图7-13d所示。



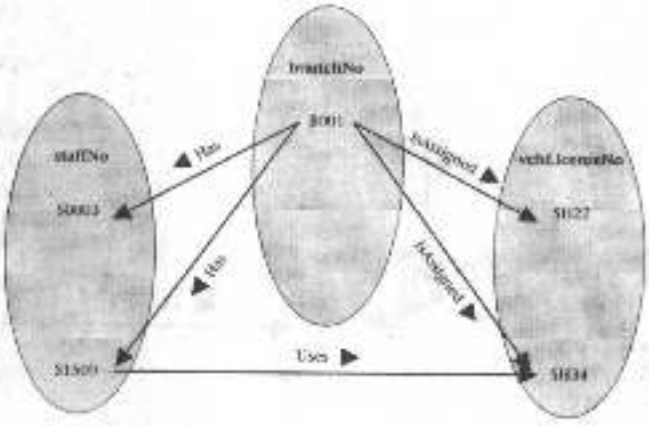
a) 扇形陷阱的例子



b) Branch Has Staff和Branch IsAssigned Car关系的例子，不能说明哪个员工使用SH34



c) 扇形陷阱的解决



d) Branch Has Staff、Branch IsAssigned Car和Staff Uses Car关系的例子。
现在可以说明哪个员工使用此汽车

图7-13 扇形陷阱及其相关问题

7.7.2　深坑陷阱

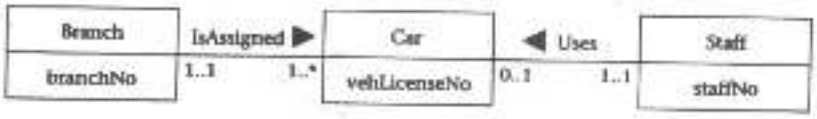
> **深坑陷阱**（Chasm Trap）　一种模型，假设两个实体之间存在关系，但这些实体之间不存在路径。

深坑陷阱可能出现的地方是存在可选参与的关系组成了相关实体之间的路径。图7-14a举例说明了一个潜在的深坑陷阱的情况。图中显示了实体Branch、Car和Staff之间的关系。这个模型告诉我们一个分公司分到了多辆汽车，一名员工可以使用一辆汽车。特别的要注意不是所有的员工都可以使用汽车。当我们想知道一名员工在哪一个分公司工作的时候会产生一个问题。为了理解这个问题，我们检查用Branch、Car和Staff的主键属性的值代表的关系IsAssigned和Uses的例子，如图7-14b所示。

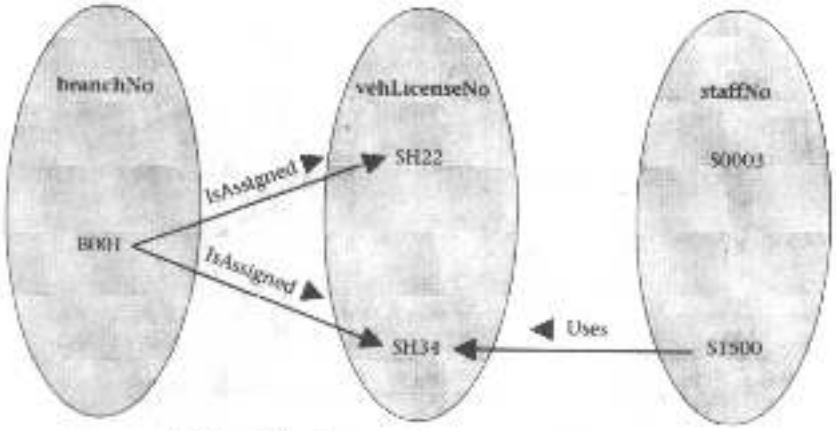
如果我们想回答这个问题：员工S0003在哪个分公司工作？利用当前的结构我们不能知道，因为不是所有的员工都使用汽车，不能回答这个问题是因为缺少信息（我们知道员工必须在某个分公司工作），这是深坑陷阱的结果。Staff Uses Car关系中的staff可选参与意味着有些员工不能通过使用汽车和一个分公司关联起来。

因此，为了解决这个问题，需要消除深坑陷阱。我们在实体Branch和Staff之间增加了一个Has关系，如图7-14c所示。现在我们检查7-14d中所示的关系Has，IsAssigned和Uses的例子，我们知道员工S0003在B001分公司工作。

本章介绍的ER模型的概念对于建立一个复杂的数据库应用有时候可能还不够用。在第11章我们将介绍一些更加普及的和ER建模相关的概念，你将发现对更加复杂的数据建模时，那些概念是非常有用的。

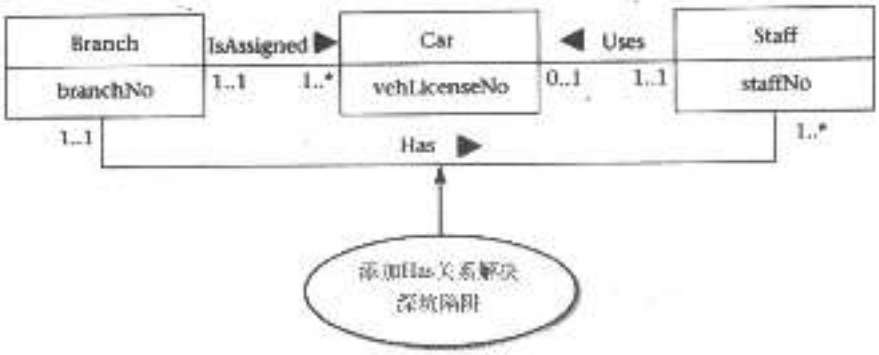


a) 深坑陷阱的例子

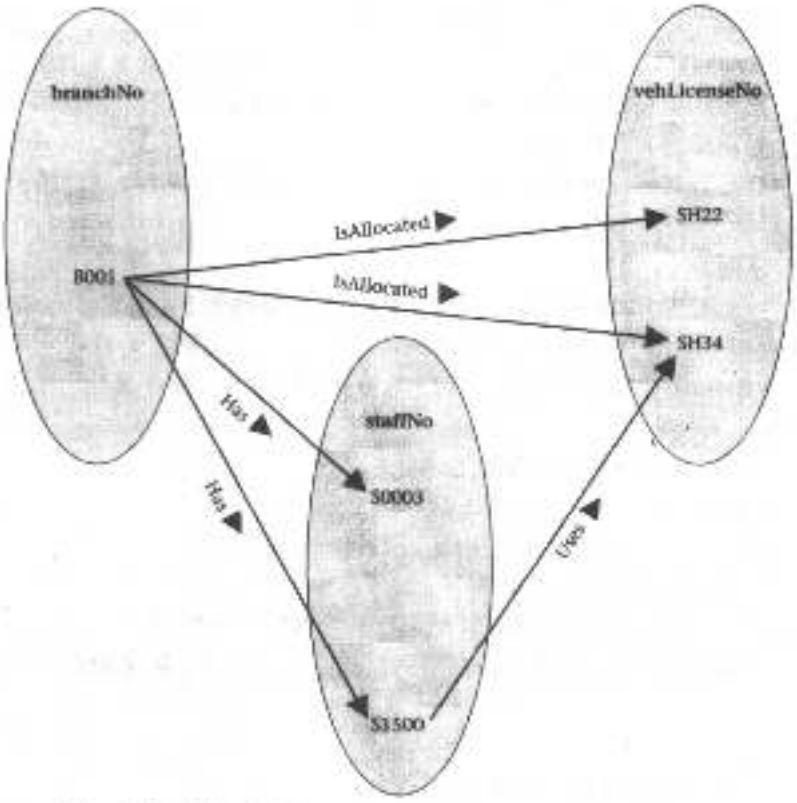


b) Branch IsAssigned Car和Staff Uses Car关系的例子，
不能说明员工S0003在哪个分公司工作

图7-14　深坑陷阱及其相关问题



c) 解决深坑陷阱



d) Branch Has Staff、Branch IsAssigned Car和Staff Uses Car关系的例子。
现在可以说明员工在哪个分公司工作了

图7-14 （续）

7.8 本章小结

- 实体是具有相同性质的一组对象，这些性质是用户或者组织作为独立存在标识。一个唯一可以标识的对象称为实体事件。
- 关系是实体之间的有意义的关联。一个唯一可以标识的关联叫做关系事件。
- 关系的度是关系中参与的实体的数目。

- 递归关系是在一个关系中同一个实体以不同的角色多次参与。
- 属性是实体或者关系的性质。
- 简单属性仅由一个元素组成。
- 复合属性由多个简单元素组成。
- 单值属性对于一个实体事件仅拥有一个值。
- 多值属性对一个实体事件拥有多个值。
- 派生属性代表一个从相关属性或者属性组导出的值，它在实体中不是必须的。
- 强实体不依赖于其他实体的存在来获得主键，弱实体部分或者全部依赖于其他实体来获得主键。
- 多样性约束定义了可能和单个相关实体事件发生关系的实体事件的数目。
- 多样性由两个约束组成，一个是基数约束，它描述了对每个参与实体的可能的关系的数目。另一个是参与约束，它决定了在一个关系中是所有的实体事件还是部分实体事件参与。
- 如果从第三个实体扇出的两个实体有1：*关系时出现扇形陷阱，但这两个实体之间应该有直接关系以提供必要的信息。
- 深坑陷阱提示在实体之间存在一个关系，但是在特定实体事件之间的路径不存在。

复习题

7.1 描述在ER模型中实体的含义，并给出一个物理或概念上存在的实体的例子。

7.2 描述在ER模型中关系的含义，并给出一个一元、二元和三元关系。

7.3 描述在ER模型中属性的含义，并给出一个简单属性、复合属性、单值属性、多值属性和派生属性。

7.4 描述一个关系中的多样性的含义。

7.5 什么是业务规则，多样性如何对这些约束建模？

7.6 在一个关系中，多样性如何表达基数约束和参与约束？

7.7 给出有属性的一个关系的例子。

7.8 描述强实体和弱实体的区别，并给出每种实体的一个例子。

7.9 描述在ER模型中如何出现扇形陷阱和深坑陷阱，并给出解决的办法。

练习

7.10 为下述每个描述创建一个ER图：

(a) 每个公司有四个部门，每个部门只属于一个公司。

(b) 每个部门有一个或多个雇员，并且每个雇员只为一个部门工作。

(c) 每个雇员可以没有或者有一个或多个助手，每个助手属于一个雇员。

(d) 每个雇员没有或者有一个工作经历。

(e) 用一个ER图表达(a)、(b)、(c)和(d)中描述的内容。

7.11 创建一个表达IT培训公司所需数据的ER图。该公司有30名教师，每个培训学期可以培训最多100名学员。公司提供五门高级技术课程，其中每门课程由两个或多个教师组成的培训小组教授。每个教师最多被分配在两个培训小组中，也可以不分配到任何培训小组中而进行研究工作。每个培训学员在每个培训学期参加一门高级技术课

程。

(a) 标识此公司的主要实体。

(b) 标识主要的关系并指明每个关系的多样性。陈述你关于数据的假设。

(c) 使用你对(a)和(b)的答案，画出表达此公司的数据需求的ER图。

7.12 阅读下面的描述EasyDrive School of Motoring的数据需求的案例研究:

EasyDrive School of Motoring 1992年成立于Glasgow（格拉斯哥（英国））。从那之后，学校开始稳步发展，现在已经在Scotland（苏格兰）的主要城市有了几个办事处，每个办事处有一个经理（经理也可以是高级教师（Senior Instructor））、几个高级教师（Senior Instructor）、教师（Instructor）和行政人员。经理负责办事处每日的运作。客户必须首先在办事处注册，他们要先填写一张申请表，申请表中记录了他们的个人信息。一个客户可能需要单独的课程或者预订一些课程。单独的课程需要一个小时，整个过程都在办事处完成。每节课程在给定的时间有一个特定的教师和特定的车辆。课程可以从早8点开始到晚8点结束。每节课程结束后，教师记录客户的学习进展并记录这节课程走的英里数。学校有一个汽车库，可以根据教学目的挑选合适的汽车。给每个教师分配一个特定的车辆。一旦准备好，客户可以申请驾驶考核时间。为了获得驾照，客户必须通过实践的和理论的测试，如果客户没有通过考试，教师必须记录失败的原因。

(a) 标识EasyDrive School of Motoring的主要实体。

(b) 标识(a)中描述的实体间的主要关系，并将每个关系绘制成一个ER图。

(c) 为(b)中描述的每个关系确定多样性约束，并在(b)中创建的ER图中表达每个关系的多样性。

(d) 标识实体和关系的属性，在(c)中创建的ER图中表达这些属性。

(e) 为每个（强）实体确定候选键和主键属性。

(f) 使用你在(a)~(e)的答案，试着在一个ER图中表达EasyDrive School of Motoring的数据需求，说明支持你的设计的任何设想。

第8章 规 范 化

本章主题:

- 存在冗余数据的表在更新异常的时候有什么表现，这可能将不一致性引入数据库。
- 最经常使用的规范化规则，即第一范式（1NF）、第二范式（2NF）和第三范式（3NF）。
- 不符合1NF、2NF或3NF的表怎样含有冗余数据，在更新异常时有什么后果。
- 怎样重建不符合1NF、2NF、3NF的表。

在前面的章节里，我们学习了ER建模方法，一个泛化的自上而下的数据库设计方法。这一章，我们考虑另外一种泛化的数据库设计的方法，叫做规范化。在数据库设计中，规范化可以有两种使用方法。第一种是把规范化用做自下而上的数据库设计方法，第二种是把规范化方法与ER建模结合起来使用。

把规范化作为自下而上的方法包括分析属性之间的关联，基于这个分析，把属性组合在一起形成代表实体和关系的表。然而，当属性很多的时候，这种方法很难使用，因为很难建立属性之间的所有重要的关联。

由于这些原因，这本书提供了一种方法，建议应该先使用自上而下的数据库设计方法，先理解数据。在这个方法中，我们使用ER建模技术创建代表主要实体和关系的数据模型，然后把ER模型翻译成代表数据的一组表。在这一点我们使用规范化方法检查表的设计是否良好。

本章的目的是解释规范化在数据库设计中是一个有用的技术的原因，还要特别介绍如何使用规范化从ER模型来构建表的结构。

8.1 简介

> **规范化**（Normalization） 一种用来产生表的集合的技术，这些表具有符合要求的属性，并能支持用户或公司的需求。

1972年，E.F.Codd博士提出了规范化技术来支持基于关系模型的数据库设计。规范化通常作为对表结构的一系列测试来决定它是否满足或符合给定范式。存在几种范式形式，但最常用的是第一范式（1NF）、第二范式（2NF）和第三范式（3NF）。所有这些范式都是基于在表中的列之间的关系的。

在以下的内容中，我们首先指出引起数据冗余的错误的结构化表是怎样导致问题的出现的，这些问题叫做更新异常。错误的结构化表可能产生于原始ER模型或在将ER模型转化成表时出现错误。然后我们给出第一范式（1NF）、第二范式（2NF）和第三范式（3NF）的定义，并解释每个范式是怎样用来确定和更正表中的不同类型的问题的。

8.2 数据冗余和更新异常

关系数据库设计的一个主要目的是把列组合成表使数据的冗余最小，并减少实现基表（基表的定义参见2.3.2节）所需的文件存储空间。为了说明与数据冗余有关的问题，我们来比

较图8-1的Staff和Branch表与图8-2的StaffBranch表。

Staff

staffNo	name	position	salary	branchNo
S1500	Tom Daniels	Manager	46000	B001
S0003	Sally Adams	Assistant	30000	B001
S0010	Mary Martinez	Manager	50000	B002
S3250	Robert Chin	Supervisor	32000	B002
S2250	Sally Stern	Manager	48000	B004
S0415	Art Peters	Manager	41000	B003

Branch

branchNo	branchAddress	telNo
B001	8 Jefferson Way, Portland, OR 97201	503-555-3618
B002	City Center Plaza, Seattle, WA 98122	206-555-6756
B003	14 – 8th Avenue, New York, NY 10012	212-371-3000
B004	16 – 14th Avenue, Seattle, WA 98128	206-555-3131

图8-1 Staff和Branch表

staffNo	name	position	salary	branchNo	branchAddress	telNo
S1500	Tom Daniels	Manager	46000	B001	8 Jefferson Way, Portland, OR 97201	503-555-3618
S0003	Sally Adams	Assistant	30000	B001	8 Jefferson Way, Portland, OR 97201	503-555-3618
S0010	Mary Martinez	Manager	50000	B002	City Center Plaza, Seattle, WA 98122	206-555-6756
S3250	Robert Chin	Supervisor	32000	B002	City Center Plaza, Seattle, WA 98122	206-555-6756
S2250	Sally Stern	Manager	48000	B004	16 – 14th Avenue, Seattle, WA 98128	206-555-3131
S0415	Art Peters	Manager	41000	B003	14 – 8th Avenue, New York, NY 10012	212-371-3000

图8-2 StaffBranch表

StaffBranch表是Staff表和Branch表一种替换模式。此表有以下结构，每一个表的主键（主键的定义参见2.2.3节）用下划线标识:

Staff (<u>staffNo</u>, name, position, salary, branchNo)

Branch (<u>branchNo</u>, branchAddress, telNo)

StaffBranch (<u>staffNo</u>, name, position, salary, branchNo, branchAddress, telNo)

在StaffBranch表中有冗余数据，因为分公司的细节信息在分公司的每个员工那里被重复了一遍。而在Branch表中，每一个分公司的详细信息仅出现一次，而且在Staff表中只有分公司编号（branchNo）被重复，它用来指出每个员工属于哪个分公司。有冗余数据的表可能有的问题叫做更新异常，它分为插入、删除和更改异常。

8.2.1 插入异常

有两种主要的插入异常，我们用图8-2的StaffBranch表来说明。

- 为了插入一新员工到StaffBranch表中，我们必须包括分公司的详细信息，这个信息决定新员工属于的分公司。例如，要插入在B003分公司工作的新员工，必须输入B003分公司的正确的细节情况使得与StaffBranch表中其他B003分公司的记录信息一致。图8-1所示的表

就不存在这种潜在的不一致性，因为对于每一个员工我们仅需加入正确的分公司号到Staff表中就可以了。另外，B003分公司在Branch表中作为单独的记录在数据库中仅记录一次。

- 如果要在StaffBranch表中插入一个新的、目前还没有员工的分公司的详细信息，需要在与员工有关的列，比如StaffNo，输入空值。但由于StaffNo在StaffBranch表中是主键，输入null给staffNo就违反了实体完整性（实体完整性定义见2.3.2节），因此不允许为空。图8-1所示的表的设计就避免了这个问题的出现，因为在Branch表中插入分公司情况与员工内容是分开的，在有了分公司之后，就可以在以后插入员工到Staff表中。

8.2.2 删除异常

如果我们从StaffBranch表中删除一个记录，而它又是一个分公司的最后一名员工，那么关于本分公司的其他情况也被从数据库中删除了。例如，如果我们从StaffBranch表中删除员工S0415('Art Peters') 的记录，那么与B003分公司相关的情况也从数据库中丢失了。图8-1所示的表的设计就避免了这个问题的出现，因为在分公司表插入的分公司情况与员工内容是分开的，仅仅是用branchNo列把两个表关系起来。如果从Staff表中删除员工S0415的记录，那么B003分公司的情况在Branch表中依旧不受影响。

8.2.3 更新异常

如果我们想在StaffBranch表中改变一个特定分公司的一个列的值，例如分公司B001的电话号码，就必须更改在那个分公司的所有员工的记录。如果此次更改没有在StaffBranch表中的所有记录中正确进行，那么数据库就变得不一致了。在这个例子中，分公司B001对于不同的员工就有不同的电话号码。

以上的例子表明图8-1的Staff表和Branch表比图8-2的StaffBranch有更大的预期实用性。在后面的章节中我们将说明范式是怎样用来规范化表的定义的，使这些表有预期的实用性并且避免潜在的更新异常。

8.3 第一范式

第一范式（First Normal Form, 1NF） 每个列和记录包含一个而且只包含一个值的表。

对于关系数据库来说，只有第一范式（1NF）在创建适当的表中是关键的。所有接下来的范式都是可选的。但是，为了避免在8.2节中所讨论到的更新异常，通常建议考虑第三范式（3NF）。

让我们看看图8-3所示的Branch表，其主键为branchNo。我们可以看到Branch表中除了telNo列之外，其他的列都遵守1NF的定义，因为对于每一个记录的telNo列有多个值。例如分公司号为B001的分公司有三个电话号码，503-555-3618、503-555-2727和503-555-6534。因此此Branch表不属于1NF。

注意 虽然列branchAddress可能会有多个值，但地址的这种表示并不违反1NF。在这个例子中，我们仅仅简单地选择了把一个地址的详细内容作为一个单独的值。

branchNo	branchAddress	telNos
B001	8 Jefferson Way, Portland, OR 97201	503-555-3618, 503-555-2727, 503-555-6534
B002	City Center Plaza, Seattle, WA 98122	206-555-6756, 206-555-8836
B003	14 – 8th Avenue, New York, NY 10012	212-371-3000
B004	16 – 14th Avenue, Seattle, WA 98128	206-555-3131, 206-555-4112

主键

多个值，所以不属于1NF

图8-3 Branch表不是1NF的一个例子

转换为1NF

为了把Branch表转换成1NF，我们创建一个单独的表，叫做BranchTelephone，此表用来保存分公司的电话号码。此表是把telNo列从Branch表中去掉，同时把Branch表的主键（branchNo）复制到新表中。新表BranchTelephone的主键是telNo。图8-4所示的是更改了的Branch表和新的BranchTelephone表的结构。现在Branch表和BranchTelephone表已属于1NF，因为符合每一个表的每一个记录的每一列都有单独的值这一标准。

Branch(不属于1NF)

branchNo	branchAddress	telNos
B001	8 Jefferson Way, Portland, OR 97201	503-555-3618, 503-555-2727, 503-555-6534
B002	City Center Plaza, Seattle, WA 98122	206-555-6756, 206-555-8836
B003	14 – 8th Avenue, New York, NY 10012	212-371-3000
B004	16 – 14th Avenue, Seattle, WA 98128	206-555-3131, 206-555-4112

给branchNo进行备份

删除telNo列并创建新列telNo

Branch(1NF)

branchNo	branchAddress
B001	8 Jefferson Way, Portland, OR 97201
B002	City Center Plaza, Seattle, WA 98122
B003	14 – 8th Avenue, New York, NY 10012
B004	16 – 14th Avenue, Seattle, WA 98128

主键

BranchTelephone(1NF)

branchNo	telNo
B001	503-555-3618
B001	503-555-2727
B001	503-555-6534
B002	206-555-6756
B002	206-555-8836
B003	212-371-3000
B004	206-555-3131
B004	206-555-4112

变为外键　　变为主键

图8-4 由于将Branch表中的telNo列移出，并创建一个叫做BranchTelephone的表，使Branch成为1NF的

8.4 第二范式

第二范式仅仅应用于具有复合主键的表，也就是主键是由两个或多个列复合而成的表。具有单列主键的表自动就是2NF的。不属于2NF的表可能遇到8.2节中讨论的更新异常问题。

> **第二范式**（Second Normal Form, 2NF） 一个第一范式的表并且每个非主键列都可以从构成主键的全部的列得到。

让我们查看图8-5所示的TempStaffAllocation表。这个表指出了在每个分公司的临时工每周的工作时间。TempStaffAllocation表的主键由StaffNo和branchNo列组合而成。注意，我们用“非主键”列来表明这些列并不是主键的一部分。例如，TempStaffAllocation表中的非主键列有branchAddress（分公司地址）、name（名字）、position（职位）和hoursPerWeek（每周工作小时）。TempStaffAllocation表下面所示的箭头指出了主键列和非主键列的特定关系。

> 在图8-5中所示的TempStaffAllocation表的列之间的特定关系更加形式化地指出了功能依赖。功能依赖是表中列的含义的性质，表明了各列是如何相互依赖。
>
> 例如，假设一个表含有列A和列B，列B功能上依赖于列A（用A→B表示）。如果我们知道了A的值，则在任何时候，我们在所有记录中只能找到对应这个A值的唯一B值。因此，当两个记录含有相同的A值的时候，它们都含有相同的B值。但对于给定的B值就可能有几个不同的A值。

我们看到TempStaffAllocation表含有冗余数据，可能会遇到8.2节中描述的更新异常。例如，要更改Ellen Layman的名字，我们不得不更改TempStaffAllocation表中的两条记录。如果仅更改了一条，数据库将产生不一致性。TempStaffAllocation表包含冗余数据的原因是这些表不符合2NF的定义。

考虑TempStaffAllocation表中的非主键列branchAddress，branchAddress列的值可以由branchNo列（主键的一部分）决定。换句话说，branchNo列中的每个唯一的值在branchAddress列中都有相同的值。例如，在branchNo列中每次出现B002值时，在branchAddress列中都出现相同的地址“City Center Plaza，Seattle，WA 98122”。

现在让我们来看看非主键列name和position。name和position的值能由staffNo列（主键的一部分）的值决定。例如，在staffNo列中每次出现S4555值时，name和position列就分别出现名字Ellen Layman和地址Assistant。

最后，让我们查看非主键列hoursPerWeek。hoursPerWeek列的值是由staffNo列和branchNo列（主键的全部）一起决定的。例如，在每次staffNo列和branchNo列中分别出现S4555和B002值时，那么hoursPerWeek列的值就是16。

> 第二范式的形式化定义是一个属于第一范式的表，并且它的每一个非主键列在功能上都完全依赖于主键。完全功能依赖是指，如果B不依赖于A的任何子集，则B完全功能依赖于A，如果B依赖于A的子集，则就称其为部分依赖。如果在主键中存在部分依靠，则这个表就不属于2NF。这个部分依赖必须在表中去掉才能成为2NF。

转换到2NF

为了把图8-5所示的TempStaffAllocation表转化成2NF，我们需要删除仅由主键的一部分决定的非主键列。换句话说，我们需要删除可由staffNo或者branchNo决定但不需要两者共同决定的列。对于TempStaffAllocation表，这就意味着我们必须删除branchAddress、name和

staffNo	branchNo	branchAddress	name	position	hoursPerWeek
S4555	B002	City Center Plaza, Seattle, WA 98122	Ellen Layman	Assistant	16
S4555	B004	16 – 14th Avenue, Seattle, WA 98128	Ellen Layman	Assistant	9
S4612	B002	City Center Plaza, Seattle, WA 98122	Dave Sinclair	Assistant	14
S4612	B004	16 – 14th Avenue, Seattle, WA 98128	Dave Sinclair	Assistant	10

复合主键

branchAddress列中的值仅能从branchNo中计算出来，所以此表不是第2范式的

branchNo列的值可以从branchAddress中计算出来

name和position列的值仅能从staffNo中计算出来，所以此表不是第2范式的

hoursPerWeek列中的值仅能从staffNo和branchNo中计算出来

图8-5 TempStaffAllocation表不是2NF的

position列，并将它们放置到一个新表中。

为此，我们创建两个新的表分别叫Branch和TempStaff。Branch表含有描述分公司情况的列，TempStaff表包含描述临时员工情况的列。

- 通过从TempStaffAllocation表中删除branchAddress列，并将与此列相关的主键列（在这里为branchNo）复制到新创建的Branch表中。
- 同样的方法，通过从TempStaffAllocation表中删除name和position列，同时复制与这些列相关的主键列（在这里为staffNo）来创建TempStaff表。

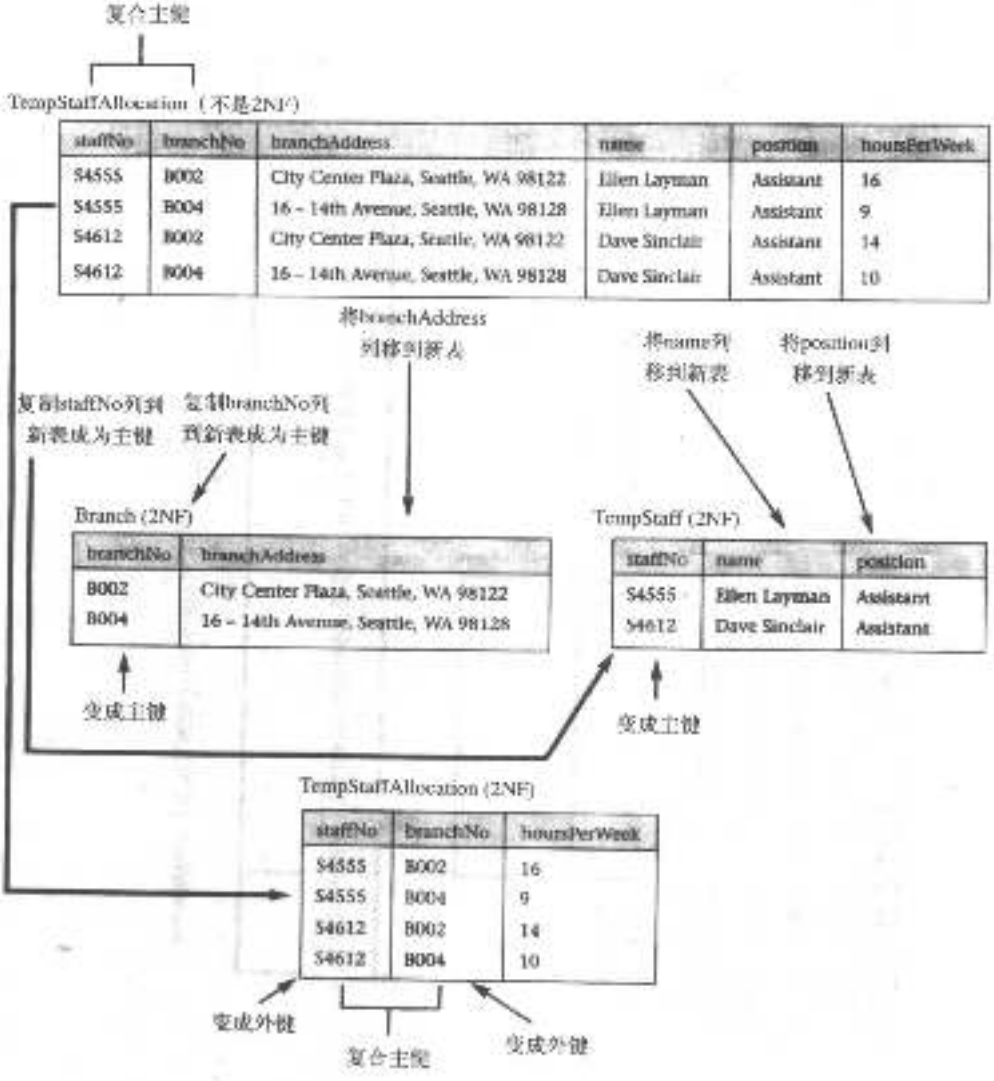


图8-6　由于删除了TempStaffAllocation表的branchAddress、name和position列，并创建新表Branch和TempStaff表，使TempStaffAllocation成为2NF的

由于在TempStaffAllocation表中hoursPerWeek列的存在并不破坏2NF规则，因此不必要删除此列。为确保维护临时工和分公司间的关系以记录临时工工作的小时总数，我们将staffNo

和branchNo列作为TempStaffAllocation表的外键。

为确保维护临时工和他所工作的分公司之间的关系，我们将staffNo和branchNo列作为TempStaff Allocation表的外键。

图8-6显示了修改后的TempStaffAllocation表的结构和新的Branch和TempStaff 表的结构，新的Branch表的主键是branchNo，新的TempStaff 表的主键是staffNo。

由于TempStaff表和Branch表都只有单个的主键列，所以他们肯定属于2NF，改变了的TempStaffAllocation表也属于2NF，这是因为非主键列hoursPerWeek既依赖于staffNo，又依赖于branchNo。

8.5 第三范式

虽然第二范式比第一范式减少了很多数据冗余，但它们可能仍然遇到更改异常。

> **第三范式**（Third Normal Form, 3NF） 一个已经是第一和第二范式的表，并且所有的非主键列的值都只能可以从主码列得到，而不能从其他的列得到。

让我们看一下图8-2所示的StaffBranch表，它的主键是staffNo，在图8-7中，我们指明了这个表中的列间的特定关系。我们可以看到，StaffBranch表包含冗余数据，并且可能造成8.2节所描述的更新异常。例如，要改变B001分公司的电话号码，我们必须修改StaffBranch表中的两条记录，如果只修改了一条记录，则数据库将会不一致。StaffBranch表包含冗余数据的原因是这个表没有遵守我们定义的3NF标准。

StaffBranch (Not 3NF)

staffNo	name	position	salary	branchNo	branchAddress	telNo
S1500	Tom Daniels	Manager	46000	B001	8 Jefferson Way, Portland, OR 97201	503-555-3618
S0003	Sally Adams	Assistant	30000	B001	8 Jefferson Way, Portland, OR 97201	503-555-3618
S0010	Mary Martinez	Manager	50000	B002	City Center Plaza, Seattle, WA 98122	206-555-6756
S3250	Robert Chin	Supervisor	32000	B002	City Center Plaza, Seattle, WA 98122	206-555-6756
S2250	Sally Stern	Manager	48000	B004	16 – 14th Avenue, Seattle, WA 98128	206-555-3131
S0415	Art Peters	Manager	41000	B003	14 – 8th Avenue, New York, NY 10012	212-371-3000

主键

所有非主键列中的值可以从主键staffNo中得出

branchAddress列和telNo列的值可以从branchNo中得出，所以该表不符合3NF

branchNo和telNo列的值可以从branchAddress获得，所以该表不符合3NF

branchNo和branchAddress列可以从telNo得到，所以该表不符合3NF

图8-7 BranchManager表不是3NF的

StaffBranch表不是3NF的，因为有branchNo、branchAddresss和telNo列。尽管我们可以从主键staffNo得到员工的分公司号、分公司地址和电话号码，但如果我们知道分公司号、分公司地址或分公司电话号码，我们也可以得到一个给定分公司的详细情况。换句话说，我们可以使用非主键列（也就是branchNo、branchAddress或telNo）的值得到信息。例如，对于staffNo列中值为S1500的行，可以得到branchAddress列中的值为“8 Jefferson Way, Portland, OR 97201”。但对于branchNo列中值为B001的行，也有branchAddress列中的值为“8 Jefferson Way, Portland, OR 97201”。也就是说，根据已知的branchNo的值可以得到员工所工作的地址。这在3NF中是不允许的，3NF要求所有的非主键列的值只能从主键列的值得到。

> 第三范式的形式化定义是一个即属于第一范式又属于第二范式的表，并且它的非主键列都不传递依赖于主键。传递依赖是一种类型的功能依赖，它是指表中的列之间存在特定类型的依赖关系。
>
> 例如，考虑一个含有列A，B，C的表。如果B功能依赖于A（A→B），并且C功能依赖于B（B→C），那么C就通过B（即使A不功能依赖于B或C）传递依赖于A。如果传递依赖存在于主键中，这个表就不属于3NF。这个传递依赖必须在表中删除才能转换成3NF。

转换到3NF

要将图8-7所示的StaffBranch表转换为3NF，就要删除可以使用其他非主键列得到的非主键列。换句话说，需要删除描述员工工作地址的列，我们删除branchAddress和telNo列并复制branchNo列，创建一个名字为Branch的新表，此表中包含这些删除的列，并且用branchNo列作为此表的主键。branchAddress和telNo列在Branch表中是候选键，因为这两个列可用于唯一地标识一个给定的分公司。要维护员工和员工所工作的分公司间的关系，因为StaffBranch表中的branchNo列是此表的一个外键。

修改后的StaffBranch表和新的Branch表的结构如图8-8所示。修改后的StaffBranch表是3NF的，因为每个非主键列都仅能从主键staffNo得到。

新的Branch表也是3NF的，因为每个非主键列都可以从主键列branchNo得到。尽管这个表中的其他两个非主键列branchAddress和telNo也可以用于得到给定分公司的细节信息，但这并不违反3NF，因为这些列是Branch表的候选键。这个例子说明，3NF的定义可以推广到表中的列全部是候选键的情况。

因此，对于包含多个候选键的表，你可以使用3NF的推广定义，即如果这个表已经是1NF和2NF的，并且所有的非主键列都只可以从候选键的列得到。而且，这个推广对2NF的定义也适用，即如果一个表是1NF的，并且每个非主键列的值只能由所有的候选键得到，则它也是2NF的。注意，这个推广不能改变1NF的定义，因为这个范式与键和表中列间的特定关系无关。

通过检查非主键列和构成主键的列间的关系来进行规范化（这可以鉴别大多数的问题并发现表中的冗余），还是使用推广的定义来增加识别遗漏的冗余的可能，使用哪个会更好些？事实上，不管你使用基于主键的2NF和3NF的基本的定义还是使用推广的定义，表的分解是一样的。

在第三范式之后还有其他的范式，例如Boyce-Codd范式（BCNF）、第四范式（4NF）和第五范式（5NF）。但是，这些范式并不常用，因为它们试图定义和解决在表中不经常出现的问题。如果你想更多地了解BCNF，4NF和5NF，可以参考Connolly和Begg写的《Database Systems》（2002）。

staffBranch（不是3NF）

staffNo	name	position	salary	branchNo	branchAddress	telNo
S1500	Tom Daniels	Manager	46000	B001	8 Jefferson Way, Portland, OR 97201	503-555-3618
S0003	Sally Adams	Assistant	30000	B001	8 Jefferson Way, Portland, OR 97201	503-555-3618
S0010	Mary Martinez	Manager	50000	B002	City Center Plaza, Seattle, WA 98122	206-555-6756
S3250	Robert Chin	Supervisor	32000	B002	City Center Plaza, Seattle, WA 98122	206-555-6756
S2250	Sally Stern	Manager	48000	B004	16 – 14th Avenue, Seattle, WA 98128	206-555-3131
S0415	Art Peters	Manager	41000	B003	14 – 8th Avenue, New York, NY 10012	212-371-3000

复制branchNo列到新表成为主键

将branchAddress列移到新表

将telNo列移到新表

Branch（3NF）

branchNo	branchAddress	telNo
B001	8 Jefferson Way, Portland, OR 97201	503-555-3618
B002	City Center Plaza, Seattle, WA 98122	206-555-6756
B003	14 – 8th Avenue, New York, NY 10012	212-371-3000
B004	16 – 14th Avenue, Seattle, WA 98128	206-555-3131

成为主键　　成为候选键　　成为候选键

staff（3NF）

staffNo	name	position	salary	branchNo
S1500	Tom Daniels	Manager	46000	B001
S0003	Sally Adams	Assistant	30000	B001
S0010	Mary Martinez	Manager	50000	B002
S3250	Robert Chin	Supervisor	32000	B002
S2250	Sally Stern	Manager	48000	B004
S0415	Art Peters	Manager	41000	B003

主键　　成为外键

图8-8 由于删除了name列，使Branch表是3NF的

8.6 本章小结

- 规范化是指创建的一些具有预期实用性的表的一项技术，这些表能支持用户或公司的要求。
- 含有冗余数据的表可能会导致的问题叫更新异常，这可分为插入异常、删除异常和更改异常。
- 第一范式（1NF）的定义是一个表中的每一个列和记录有且仅有一个值。
- 第二范式（2NF）的定义是一个属于第一范式的表，它的每一个非主键列必须完全由全部的主键列决定。
- 第三范式（3NF）的定义是一个即属于第一范式又属于第二范式的表，它的所有非主键列的值都只能从列值得到，而不能从其他列得到。

复习题

8.1 讨论在数据库设计中如何使用规范化。

8.2 描述在有冗余数据的表上可能出现的更新异常的种类。

8.3　描述违反第一范式（1NF）的表的特点，并描述如何将这样的表转换为1NF。

8.4　表必须满足的最小范式是什么？给出这个范式的定义。

8.5　描述一个将第一范式(1NF)的表转换为第二范式（2NF）表的方法。

8.6　描述符合第二范式（2NF）的表的特点。

8.7　描述完全函数依赖的含义，并描述这种类型的依赖与2NF的关系，给出一个能够说明你的答案的例子。

8.8　描述符合第三范式（3NF）的表的特点。

8.9　描述传递依赖的含义，描述这种类型的依赖与3NF的关系，给出一个能够说明你的答案的例子。

练习

8.10　图8-9所示的表列出了牙科医生/病人的预约数据。一个病人在指定的日期和时间预约一个特定治疗室的牙科医生。

(a) 图8-9所示的表存在更新异常，给出插入、删除和修改异常的例子。

(b) 描述并说明对图8-9所示的表进行规范化，使之符合3NF的过程。说明你对表中所示数据所作的假设。

staffNo	dentistName	patientNo	patientName	appointment date	appointment time	surgeryNo*
S1011	Tony Smith	P100	Gillian White	12-Aug-03	10.00	S10
S1011	Tony Smith	P105	Jill Bell	13-Aug-03	12.00	S15
S1024	Helen Pearson	P108	Ian MacKay	12-Sept-03	10.00	S10
S1024	Helen Pearson	P108	Ian MacKay	14-Sept-03	10.00	S10
S1032	Robin Plevin	P105	Jill Bell	14-Oct-03	16.30	S15
S1032	Robin Plevin	P110	John Walker	15-Oct-03	18.00	S13

图8-9　牙科医生的预约登记表

8.11　一个名为InstantCover的代理给整个Scotland的旅馆提供兼职/临时工，图8-10所示的表列出了在两个旅馆工作的代理员工的工作时间。National Insurance Number（NIN，国家保险号码）对每个雇员都是唯一的。

(a) 图8-10所示的表中存在更新异常，给出插入、删除和修改异常的例子。

(b) 描述并说明对图8-10所示的表进行规范化，使之符合3NF的过程。说明你对表中所示数据所作的假设。

NIN	contractNo	hoursPerWeek	eName	hotelNo	hotelLocation
113567WD	C1024	16	John Smith	H25	Edinburgh
234111XA	C1024	24	Diane Hocine	H25	Edinburgh
712670YD	C1025	28	Sarah White	H4	Glasgow
113567WD	C1025	16	John Smith	H4	Glasgow

图8-10　InstantCover的员工及其工作的旅馆

第三部分 逻辑数据库设计

第9章 逻辑数据库设计——步骤1
第10章 逻辑数据库设计——步骤2
第11章 高级建模技术

第9章 逻辑数据库设计——步骤1

本章主题:

- 什么是设计方法学。
- 数据库设计的两个主要阶段：逻辑设计和物理设计。
- 数据库设计中的关键成功因素。
- 逻辑和物理数据库设计中的方法学。
- 数据库设计方法学中步骤1的任务：构建ER模型。
- 数据库设计步骤1产生的文档，包括ER图和数据字典。

在第4章中，我们描述了数据库系统开发生命周期的主要阶段，其中之一就是数据库设计。正如我们在第6章介绍的那样，只有在完全完成了需求分析之后，才可以开始数据库设计阶段的工作。数据库设计由两个主要阶段组成：逻辑数据库设计和物理数据库设计。在本章和第10章中，我们将介绍逻辑数据库设计方法学，在第12章~第16章中介绍物理数据库设计方法学。我们从数据库设计方法学的概要开始介绍，然后详细描述与逻辑数据库设计步骤1有关的任务。

9.1 数据库设计方法学简介

如果所需要的数据库变得相当复杂，就需要有一种系统化的方法去设计和构建数据库，使数据库既满足用户需求又能获得性能需求（例如响应时间）。这种系统化方法就是数据库设计方法学。在概要介绍方法学之前，首先讨论什么是数据库设计方法学，然后说明数据库设计中的关键成功因素。

9.1.1 什么是数据库设计方法学

> **设计方法学**（Design Methodology） 一种使用过程、技巧、工具和文档来支持和简化设计过程的结构化方法。

数据库设计方法学由一系列步骤组成，这些步骤在工程的每个阶段引导设计者使用合适的技术，这些阶段还帮助设计者规划、管理、控制和评价数据库开发过程。此外，这个方法是一个结构化的方法，用于以标准化的和有组织的方式分析和建立数据库需求模型。

9.1.2 数据库设计的各阶段

在本书中，我们提出了一种方法，这种方法将数据库设计分成两个主要的阶段：逻辑数据库设计和物理数据库设计。

> **逻辑数据库设计**(Logical Database Design) 按照特定的数据模型，构建企业所使用的数据的模型的过程，但独立于特定的DBMS和其他的物理考虑事项。

在逻辑数据库设计阶段，我们构建数据库的逻辑表示，其中包括标识重要的实体和关系，然后将这些表示转换成表的集合。逻辑数据库设计是物理数据库设计的信息来源，为物理数据库设计人员设计有效数据库提供非常重要的信息。

> **物理数据库设计**(Physical Database Design) 在二级存储上的数据库的实现的描述，它描述基本表、文件组织、用户高效访问数据的索引和相关的完整性约束及安全性限制。

在物理数据库设计阶段，我们确定如何在目标关系DBMS中物理地实现逻辑设计。这个阶段允许设计者决定如何实现数据库，因此，物理设计与特定的DBMS有关。

> 强调一下，在逻辑数据库设计之前有一个阶段叫做概念数据库设计，这个阶段从创建企业使用的数据的概念数据模型开始，它完全独立于所有的实现细节，比如使用的数据模型（例如，关系数据模型）和其他的物理考虑。但是，我们是在设计关系数据库，因此，我们将概念数据库设计和逻辑设计阶段结合，并使用更通用的术语——逻辑数据库设计。

9.1.3 数据库设计中的关键成功因素

下述方针对于成功进行数据库设计是很重要的：

- 尽可能多地与用户进行交流。
- 在整个数据建模过程中使用一种结构化方法学。
- 使用数据驱动方法。
- 在数据模型中加入结构化和完整性考虑。
- 将规范化和事务有效性技术结合进方法学中。
- 尽可能多地使用图去表示数据模型。
- 使用数据库设计语言（DBDL）。
- 构建数据字典补充数据模型图。
- 乐于重复以上步骤。

所有这些方针都被融入方法学中，并在接下来的章节中详细描述。

9.2 数据库设计方法学概述

本节我们将给出数据库设计方法学的概述。方法学中的步骤如图9-1所示，哪个步骤在哪章讨论被列在了相毗的列中。

逻辑数据库设计主要分为以下两个主要步骤：

- 在步骤1中，我们创建一个ER模型并检查这个模型是否有最小的冗余，是否可以支持用

户事务。这个步骤的输出是创建一个ER模型，这个模型完全并准确地表达企业（或企业的一部分）对数据的需求。

- 在步骤2中我们将ER模型映射为表的集合。对每个表的结构都使用规范化来检查。规范化能够确保表在结构上是一致的、逻辑的，并且有最小的冗余。对表也进行检查以确保它们能支持所需的事务，同时也定义数据库要求的完整性约束。

> 对有很多不同用户视图的数据库系统来说，可能需要创建一个或多个逻辑数据模型设计，并在数据库设计过程的以后阶段对这些模型进行合并。我们将在附录C中描述与合并数据模型有关的典型任务。

物理数据库设计包括六个主要步骤：

- 步骤3包括使用目标DBMS的功能设计基本表和完整性约束。
- 步骤4包括为基本表选择文件组织方式以及索引。通常，DBMS提供一定数量的可供选择的关于数据的文件组织方式，这不包括PC DBMS，这些DBMS一般有固定的存储结构。
- 步骤5包括在数据库系统开发生命周期的需求分析和收集阶段确定的用户视图的设计。
- 步骤6包括设计安全性措施以避免未授权用户对数据的访问。
- 步骤7考虑放宽加在表上的规范化约束，从而改善整个系统的性能。这个步骤只在需要时才做，因为在引入冗余时会同时产生一些问题，仍需要维护其一致性。
- 步骤8不断地通过监视和调整操作系统来标识和解决由设计引起的性能问题，并实现新的或者改变的需求。

逻辑数据库设计	章
步骤1：创建并检查ER模型	9
步骤1.1：标识实体	
步骤1.2：标识关系	
步骤1.3：标识实体或关系的有关属性	
步骤1.4：确定属性域	
步骤1.5：确定候选键、主键和备用键属性	
步骤1.6：特化/泛化实体（可选步骤）	
步骤1.7：检查模型的数据冗余	
步骤1.8：检查模型是否支持用户事务	
步骤1.9：与用户一起检查模型	
步骤2：将ER模型映射为表	10
步骤2.1：创建表	
步骤2.2：用规范化方法检查表结构	
步骤2.3：检查表是否支持用户事务	
步骤2.4：检查业务规则	
步骤2.5：与用户讨论逻辑数据库设计	
物理数据库设计	
步骤3：为目标DBMS转换全局逻辑数据模型	12
步骤3.1：设计基本表	
步骤3.2：设计派生数据的表示	
步骤3.3：设计其他业务规则	

图9-1 逻辑和物理数据库设计方法学中的各阶段

步骤4：选择文件组织方式和索引　13
　步骤4.1：分析事务
　步骤4.2：选择文件组织方式
　步骤4.3：选择索引
步骤5：设计用户视图　14
步骤6：设计安全性机制　14
步骤7：引入受控冗余的考虑　15
步骤8：监视并调整操作系统　16

图 9-1（续）

附录B给出了这些方法学的总结，这有助于那些已经熟悉数据库设计并只需要了解主要步骤的用户。

在整个方法学中，用户在评审和检查数据模型以及支持文档时起着非常关键的作用。有些步骤可能与你所分析的企业的复杂性以及性能和安全性的要求无关。

提示　数据库设计是一个迭代过程，开始以后就要不断进行精化。尽管我们介绍的数据库设计方法学是一个过程化的过程，但这并不意味着要以过程化的方式执行。你在某一个阶段得到的结果可能会改变你在上一个阶段作出的决定。同样，你会发现在后一个阶段中查看一下前面的结果也是有所帮助的。这种方法可以作为框架指导你高效地进行数据库设计。

9.3　逻辑数据库设计方法学步骤1简介

本节概述逻辑数据库设计方法学的第一个步骤——为在分析阶段标识的用户视图建立一个ER模型。

在分析阶段，你已经标识了若干用户视图，视图的多少依赖于这些视图的交叠数量和数据库系统的复杂度，你可能已经对有些用户视图进行了合并。在6.4.4节讨论的需求收集和分析阶段，我们使用集中式方法为StayHome创建了两个用户视图的集合，它们表达了下述用户视图的合并需求：

- Branch用户视图表示Manager（经理）、Supervisor（监理）和Assistant（助理）用户视图。
- Business用户视图表示Director（导演）和Buyer（购买者）用户视图。

在后面的内容中，我们将为StayHome的Branch用户视图构建一个逻辑数据模型。在第4章中，我们引入了术语“局部逻辑数据模型”来描述表达一个或多个（但不是数据库中全部）用户视图的模型，但本章描述的是数据库设计方法学，所以我们只使用更通用的术语“逻辑数据模型”。

如果你有兴趣构造更复杂的数据库，那么首先需要创建独立的局部逻辑数据模型来代表数据库中的不同的用户视图，我们在附录C中描述并演示了使用Branch和Business用户视图合并StayHome的数据模型的方法。

9.4　步骤1：创建并检查ER模型

目标　构建将由数据库支持的企业（或企业的一部分）的数据需求的ER模型。

每个ER模型组成如下:

- 实体。
- 关系。
- 属性和属性域。
- 主键和备用键。
- 完整性约束。

ER模型是由文档支持的，包括数据字典和ER图，它们是在模型的开发过程中产生的。我们将详细介绍在每个步骤中所产生的支持文档的类型。步骤1包括的主要任务有:

- 步骤1.1: 标识实体。
- 步骤1.2: 标识关系。
- 步骤1.3: 标识实体或关系的有关属性。
- 步骤1.4: 确定属性域。
- 步骤1.5: 确定候选键、主键和备用键属性。
- 步骤1.6: 特化/泛化实体（可选步骤）。
- 步骤1.7: 检查模型的数据冗余。
- 步骤1.8: 检查模型是否支持用户事务。
- 步骤1.9: 与用户一起检查模型。

现在，让我们开始为StayHome的Branch用户视图构建ER模型。

9.4.1 步骤1.1: 标识实体

目标 标识需要的实体。

构建ER数据模型的第一步就是定义用户感兴趣的主要对象。这些对象就是模型中的实体。标识实体的一种方法就是仔细研究用户的需求说明。从说明中，可以定义提到的名词或名词短语（例如，staff number（员工号）、staff name（员工名）、catalog number（目录号）、title（标题）、daily rental rate（日租金）、purchase price（购买价格）），还应该查找主要对象，如people（人物）、place（地点）或concepts of interest（利润），排除那些只表示其他对象特征的名词。

例如，可以把staff number和staff name 组成实体Staff。catalog number、title、daily rental rate和purchase price组成实体Video（录像）。

标识实体的另一种方法是查找那些有存在必要的对象。例如，Staff（员工）是一个实体，因为无论是否知道他们的名字、地址和工资，员工都是存在的。如果可能，应该依靠用户辅助完成这项活动。

有时由于实体在用户需求说明中提出的方式不同，因此标识实体是很困难的。用户通常习惯于用“例子”或“类推”来谈论。用户可能用人员的名字（people's names），而不是通常所说的staff（员工）。在某些例子中，尤其是在既有公司又有人员的例子中，用户使用工作角色术语。这些角色可能是工作名称或职责，如Manager（经理）、Deputy Manager（副经理）、Supervisor（监理）或Assistant（助理）。更麻烦的是，用户经常使用同义词和多义词。

> 当两个词意义相同时是同义词，例如，branch和outlet。由于上下文关系，一个词有不同的意义就是多义词，例如词program有几个不同的意义，如一系列事件、工作计划、软件程序和研究课程。

某一个对象是否是实体、关系或属性，并不总是很明显的。例如，对“婚姻”怎样建模？实际上，对于不同的实际需求，可以把“婚姻”作为实体、关系或属性中的任意一个或者全部。你会发现分析是主观的，不同的设计者可能会产生不同的解释，但这些解释却是等效的。在某种程度上来说，这种工作依靠的是判断力和经验。数据库设计者必须根据现实世界进行选择，并根据实际情况对事务进行分类。因此，从用户提供的需求说明得到的一组实体可能不是唯一的。然而，分析过程的不断重复必定会引导你选择对完成系统需求来说已经足够用的实体。

提示　数据库设计的主观性开始可能令人烦恼。但是，根据本书提供的方法学，你会发现设计是可以完成的，并且在有了一些练习和经验后，设计变得容易多了。为了进一步学习，在第17章和第18章中，我们将完成另一个例子的学习。在附录E中我们提供了你可能会遇到的许多常见的业务数据模型。

1. StayHome实体

对于StayHome的Branch用户视图，可以定义下列实体：

Branch(分公司)	Staff（员工）
Video（录像）	VideoForRent（出租录像）
Member（成员）	RentalAgreement（出租协议）
Actor(演员)	Director（导演）

2. 实体文档

当标识实体时，应使实体的名字对用户来说意义直观。在数据字典中记录实体的名字和描述。如果有可能，记录希望每个实体出现的序号。如果一个实体有不同的名字，这些名字被认为是同义词或别名，而这些也应记录在数据字典中。图9-2说明了从StayHome实体的Branch用户视图的文档的数据字典中抽取出的内容。

Entity name	Description	Aliases	Occurrence
Branch	Place of work	Outlet and Branch Outlet	One or more *StayHome* branches are located in main cities throughout the US.
Staff	General term describing all staff employed by *StayHome*	Employee	Each member of staff works at a particular branch.

图9-2　从数据字典中抽取出来的StayHome实体的描述

9.4.2 步骤1.2：标识关系

目标 标识实体间存在的重要关系。

标识完实体之后，下一步就是标识这些实体间的所有关系。标识实体时，一种方法是在用户的需求说明中寻找名词。标识关系时，也可以利用需求说明的语法。一般地，关系由动词或动词短语表示。例如：

- Branch *Has* Staff（分公司有员工）。
- Branch *IsAllocated* VideoForRent（分公司被分派了出租录像）。
- VideoForRent *IsPartOf* RentalAgreement（出租录像是出租协议的一部分）。

事实上，用户的需求说明中记录了这些关系，说明它们对用户很重要，因此在模型中必然应该包含这些关系。

提示 我们只对实体间必需的关系感兴趣。在前面的例子中，标识了Branch IsAllocated VideoForRent 和VideoForRent IsPartOf RentalAgreement关系。我们也可以认为在Branch和RentalAgreement（出租协议）之间包含一个关系（例如，Branch Handles RentalAgreement（分公司处理出租协议））。尽管这是一个可能的关系，但根据需求，它并不是我们在模型中感兴趣的关系。我们将要在步骤1.7中进一步讨论。

要确保在用户需求说明中的显式或隐式的关系均被标识出来，原则上说，检查每一对具有潜在关系的实体应该是可能的，但对包含成百个实体的大系统来说这是一项令人望而生畏的任务。另一方面，不进行这样的检查也是不明智的。不过，当检查模型是否支持用户需要的事务时，是否丢失了关系会很明显地显示出来。

在大多数情况下，关系都是二元的，换句话说，关系只存在于两个实体之间。但是，你应该注意那些多于两个实体间的复杂关系和那些只涉及一个实体的递归关系。就StayHome的Branch用户视图而言，应该标识如下的非二元关系：

Registers（登记） Branch、Member和Staff之间的三元关系。

Supervises（监督） 基于Staff的递归关系。

1. StayHome的关系

对于StayHome的Branch用户视图而言，可以标识图9-3中所示的关系。

实　体	关　系	实　体
Branch	Has	Staff
	IsAllocated	VideoForRent
Branch, Staff*	Registers	Member
Staff	Manages	Branch
	Supervises	Staff
Video	Is	VideoForRent
VideoForRent	IsPartOf	RentalAgreement
Member	Requests	RentalAgreement
Actor	PlaysIn	Video
Director	Directs	Video

注：*表示一个三元关系。

图9-3 StayHome中Branch用户视图的关系的初稿

(1) 使用实体－关系概念（ER）建模

直观地显现一个复杂系统，要比把该系统解释成很长的文字描述容易得多。实体-关系图（ER图）可用于更好地描述实体和它们之间的关系。图9-4所示的ER模型是描述上述实体和关系的初稿。

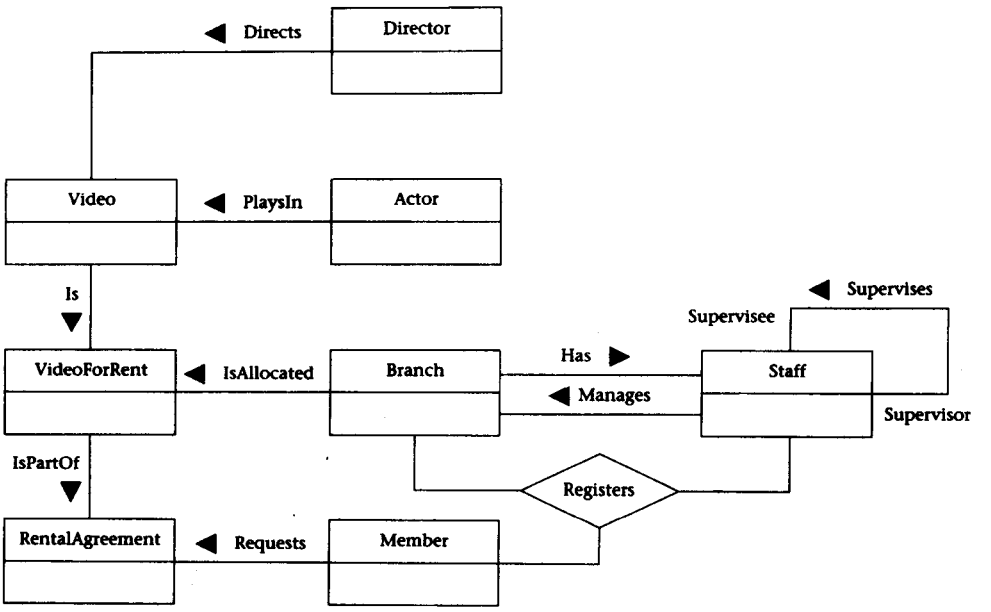


图9-4 StayHome中Branch用户视图中实体和关系的ER模型初稿

提示 在整个数据库设计过程中，我们建议在任何需要的时候都使用ER建模，这样可以帮助构建想要的模型的图形描述。不同的人可以使用不同的ER模型符号。本书中，我们使用最先进的面向对象标记法——UML(Unified Modeling Language，统一建模语言)，但其他标记法也实现相似的功能。

(2) 确定关系的多样性约束

标识完要建模的关系后，接下来要确定关系的种类。如果知道了多样性的特定值，或者甚至知道了上限和下限限制，则应把这些值也记录在文档中。

实 体	多 样 性	关 系	实 体	多 样 性
Branch	1..*	Has	1..1	Staff
	1..*	IsAllocated	1..1	VideoForRent
Branch, Staff *	1..*, 1..1	Registers	0..*	Member
Staff	0..1	Manages	1..1	Branch
	0..*	Supervises	0..1	Staff
Video	1..*	Is	1..1	VideoForRent
VideoForRent	0..*	IsPartOf	1..1	RentalAgreement
Member	0..*	Requests	1..1	RentalAgreement
Actor	1..*	PlaysIn	0..*	Video
Director	1..*	Directs	1..1	Video

注：* 表示一个三元关系。

图9-5 StayHome中关系的多样性约束

一个包括多样性约束的模型更明确地描述了关系的语义，可以在模型中产生更好的表达结果。多样性约束被用来检查和维护数据特征。当数据库更新时，要判断是否违反了规则，这些约束就会起作用了。

2. StayHome的多样性约束

对于StayHome示例，应该标识图9-5中所示的多样性约束。图9-6显示了用这些新添加的信息更改后的ER模型。

(1) 检查扇形陷阱和深坑陷阱

标识了关系后，应该检查每个关系所描述的是不是确实是所需要的，并且应该检查是否在不经意间产生了陷阱（扇形陷阱和深坑陷阱参见7.7节）。

(2) 将关系存档

在标识关系时，为关系指定一个有意义并且对用户来说直观的名字。在数据字典中，要记录关系的描述以及关系的多样性约束。图9-7显示了从StayHome的Business用户视图的数据字典中选出的有关关系的文档。

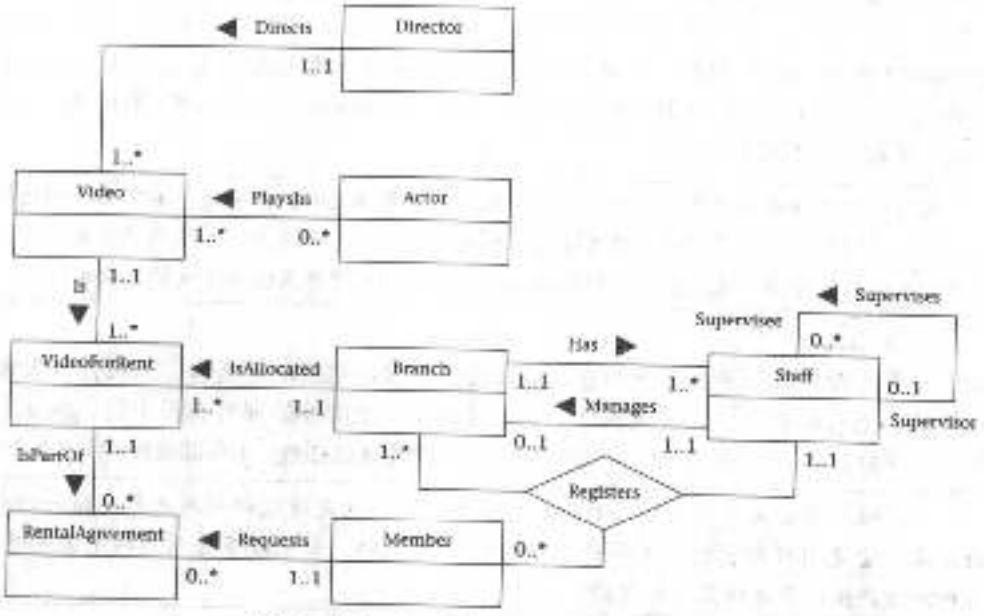


图9-6 为StayHome的ER模型添加多样性约束

Entity	Multiplicity	Relationship	Multiplicity	Entity
Branch	1..*	Has	1..1	Staff
Branch	1..*	IsAllocated	1..1	VideoForRent
Staff	0..1	Manages	1..1	Branch
Staff	0..*	Supervises	0..1	Staff

图9-7 从StayHome的Branch用户视图的数据字典中抽取的关系的描述

9.4.3 步骤1.3：标识实体或关系的有关属性

目标 为实体和关系标识相关属性。

方法学中的下一步骤就是为已经确定的数据库中的实体和关系标识属性类型。与标识实体很相似，首先要在用户需求说明中寻找名词或名词短语。当这个名词或名词短语是特性、标志或前面定义的实体或关系的特征时就可被标识成属性（属性的详细内容参见7.3节）。

提示 当我们在用户需求说明中标识完实体或关系之后，最简单的事情就是考虑“哪些信息是我们需要保存的”，在用户的需求说明中应该有这个问题的答案。但是，有时可能需要询问用户来明确需求。但是，他们所给的答案可能又包含了其他概念，因此对于用户的回答必须仔细考虑。

1. 简单/复合属性

标识属性是简单的还是复合的是很重要的。复合属性由简单属性组成。例如，一个地址属性可以是简单的，把所有关于地址的细节当作一个值，例如“8 Jefferson Way,Portland,OR,97201”。但是，地址属性也可以是由简单属性组成的复合属性，把描述地址细节的各元素分开，如街道（8 Jefferson Way）、城市（Portland）、州（OR）和邮编（97201）。（简单/复合属性定义参见7.3.1节。）

选择地址属性是简单的还是复合的，是由用户需求决定的。如果用户不需要访问地址中各个独立的部分，就可以把地址属性描述为简单属性。另一方面，如果用户需要访问地址中独立的各个部分，就应该把地址属性描述为由必需的简单属性所组成的复杂属性。

2. 单值/多值属性

除了可将属性分为简单和复合的之外，属性还可以是单值的或是多值的属性。大多数属性是单值的，但有时也会碰到多值属性，也就是说某一个实体的属性有多个值。例如，可把实体Branch（分公司）的属性telNo（电话号码）标识为多值属性。（多值属性参见7.3.2节。）

也可以把分公司电话号码作为一个独立的实体，这是对这种问题进行建模一种替换方法，它是同样有效的。正如将在步骤2.1中看到的，如果把多值属性映射为实体，两种方法产生的最终结果是相同的。

3. 派生的属性

如果属性值可以通过检查其他属性的值得到，则此属性就称为派生属性。通常，在逻辑数据模型中见不到这些属性。然而，有时该派生属性值或派生属性所依赖的源属性值会被删除或修改。这时，为了避免潜在地丢失信息，派生属性在数据模型中必须可见。如果模型中出现派生属性，则必须在它的名字前加上前缀“/”来说明它是派生而来的。（派生属性定义参见7.3.3节。）

在物理数据库设计期间，我们才考虑派生属性的表达。由于使用属性的方式不同，在每次访问派生属性或当它所派生的值发生变化时，都要计算派生属性的值。这并不是逻辑数据库设计要研究的内容，我们将在第12章步骤3.2中讨论如何更好地从物理上描述派生属性。

4. 潜在问题

当标识属性时，从原始的选择中删除一个或更多实体是很有可能的。这种情况下，应返回前面的步骤，将新的实体存档并重新检查相关的关系。

提示 生成一张包含用户需求说明中的所有属性的列表是很有用的。当把属性与具体的实体或关系关联起来时，就把该项属性从列表中删除。这样一来，就可以保证一个属性只与一个实体或关系相关。当表为空时，所有的属性就都与某实体或关系相关了。

也必须注意那些可能不只与一个实体相关的属性，这表明可能会有如下情况:

- 可能已经标识了几个可以合在一起的实体。例如，可能标识了实体Manager和Supervisor，它们都有属性StaffNo（员工号）、name（名字）和salary（工资），而这些可以用一个叫做Staff（员工）的实体来表达，此实体包含staffNo、name、salary和position属性。

> 另一方面，有些实体可能包含很多相同的属性，但其中的每个实体又有自己各自不同的属性。在第11章中，我们将看到一些高级的ER建模概念，如特化和泛化，并提供了使用它们的方法。这些高级概念可以更精确地表达如上这种情况。在这儿，我们省略了这些高级概念，让它们成为可选步骤（步骤1.6）从而使基本的方法学尽可能简单。

- 已经标识完了实体间的关系。这时，必须把属性与一个实体（也就是父实体）相联，并要确保步骤1.2中标识的关系。如果情况不是这样，则必须根据新标识的关系更新文档。例如，假设已经用如下属性标识了实体Branch和Staff:

Branch	branchNo、street、city、state、zipCode、managerName
（分公司）	（分公司号、街道、城市、州、邮编、经理名字）
Staff	staffNo、name、position、salary
（员工）	（员工号、名字、职位、工资）

由于Branch中managerName属性的出现是要表达Staff Manages Branch（员工管理分公司）关系。这种情况下，属性managerName应该从实体Branch中删除，并且应该在模型中添加Manages关系。

5. StayHome中实体的属性

对于StayHome这个例子中的Branch用户视图，应该对实体进行标识和添加有关的属性，如下述所示:

Branch	branchNo、address (复合:street、city、state、zipCode)、telNo (多值)
（分公司）	（分公司号、地址（复合: 街道、城市、州、邮编）、电话号码）
Staff	staffNo、name、position、salary
（员工）	（员工号、名字、职位、工资）
Video	catalogNo、title、category、dailyRental、price
（录像）	（类别号、标题、类别、日租、价格）
Director	directorName
（导演）	（导演姓名）
Actor	actorName
（演员）	（演员姓名）
Member	memberNo、name (复合: fName、lName)、address
（成员）	（成员号、名字（复合: 姓、名）、地址）
RentalAgreement	rentalNo、dateOut、dateReturn
（出租协议）	（出租号、出租日期、归还日期）
VideoForRent	videoNo、available
（出租录像）	（录像号、可否出租）

注意　实体Branch中的address属性和实体Member中的name属性是复合型的，而实体Member中的address属性和实体Staff、Director和Actor中的name属性是简单型的。这就反映了这些属性的不同的用户访问需求。

6. StayHome中关系的属性

很难把属性dateJoined（描述会员在分公司登记的日期）与某一个特定的实体相联。不能和实体Member相联，因为一个Member能在不只一个分公司登记。事实上，这个属性不能与以上的任何一个实体相联，但应该与实体Member、Branch和Staff之间的三元关系Registers（登记）相联。类似的情况下，属性character（描述演员在录像中的角色名称）应与实体Actor和Video之间的多对多关系PlaysIn（出演）相联。

7. 将属性存档

在标识属性时，应该为他们取有意义且直观的名字，然后为每个属性记录如下信息：

- 属性名称和描述。
- 数据类型和长度。
- 属性的别名或同义词。
- 属性可不可以为空（空定义参见2.3.1节）。
- 属性是否为多值的。
- 属性是否是复合的，如果是，组成复合属性的简单属性是什么。
- 属性是否为派生的，如果是，应该怎样计算。
- 属性的默认值（如果指定的话）。

图9-8显示了StayHome的Branch用户视图数据字典中的部分属性的文档。

Entity	Attributes	Description	Data type and length	Nulls	Multi-valued	...
Branch	branchNo	Uniquely identifies a branch	4 fixed characters	No	No	
	address: street	Street of branch address	30 variable characters	No	No	
	city	City of branch address	20 variable characters	No	No	
	state	State of branch address	2 fixed characters	No	No	
	zipCode	Zip code of branch address	5 variable characters	No	No	
	telNo	Telephone numbers of branch	10 variable characters	No	Yes	
Staff	staffNo	Uniquely identifies a member of staff	5 fixed characters	No	No	
	name	Name of staff member	30 variable characters	No	No	

图9-8　显示了StayHome的Branch用户视图的数据字典中的部分属性描述

9.4.4　步骤1.4：确定属性域

目标　确定ER模型中的属性域。

这步的目标就是确定ER模型中的属性域。域是一组值的集合，一个或多个属性可以从中

选择它们的值。例如StayHome的属性域包括:

- 合法的分公司号的属性域是一个四位字符的定长字符串，并且第一个字符必须是字母，后三个字符是从000到999的数字。
- 合法的电话号码的属性域是一个10位的数字串。
- VideoForRent实体的available属性的可能值是Y或N。这个属性的域是包含值Y和N的单个字符。

一个开发完备的数据模型为模型中的每个属性指定域，其中包括:

- 该属性的允许值集合。
- 该属性大小和格式。

将属性域存档

当标识属性域时，在数据字典中记录它们的名字和特征。更改属性的数据字典项，用域来代替数据类型和长度信息。

9.4.5 步骤1.5：确定候选键、主键和备用键属性

> **目标** 为每个实体标识候选键，如果有多个候选键，则选择其中之一作为主键，并标识其他的备用键。

这步就是要为实体确定候选键，然后从中选出一个主键。一定要确保候选键永不为空(如果候选键不只包含一个属性，那么每个属性都不能为空)。如果标识了多个候选键，则必须选择其中之一作为主键，其他的候选键被称为备用键。(键的定义参见2.2.3节和7.3.4节。)

> **提示** 正如我们在2.2.3节中指出的，人名通常不适合作候选键。例如，你可能觉得对Staff（员工）实体来说，name（名字）是比较合适的候选键。但是，可能有两个同名的人都加入了StayHome，这样就可看出，name不可作为候选键。我们可以对StayHome的成员的名字进行类似的讨论。这时，定义一个具有唯一性属性，要比把不同的属性组合起来来提供唯一性要好得多。例如，Staff实体的staffNo属性和Member实体的memberNo属性，就是这样的新属性。

当从候选键中选择主键时，应遵循如下指南来进行选择:

- 具有最少的一组属性的候选键。
- 值很少变化的候选键。
- 在未来不会失去唯一性的候选键。
- 字符最少的候选键（对文本属性来说)。
- 最大值最小的候选键（对数值型属性来说)。
- 从用户的观点来看最易于使用的候选键。

在标识主键的过程中，要注意实体是强实体还是弱实体。如果可以为一个实体赋予一个主键，则这个实体将被看成是强实体。反之，如果不能为一个实体标识主键，则这个实体就是弱实体（弱实体定义参见7.4节)。但是，与弱实体有关的一个或多个属性也可能成为最终的主键的一部分，但它们并不提供唯一性。

> 只有在将弱实体映射为表时才能标识弱实体的主键，我们将在第10章的步骤2.1中描述。

1. StayHome 主键

对于StayHome的Branch用户视图，应该像图9-9所示的那样标识主键。

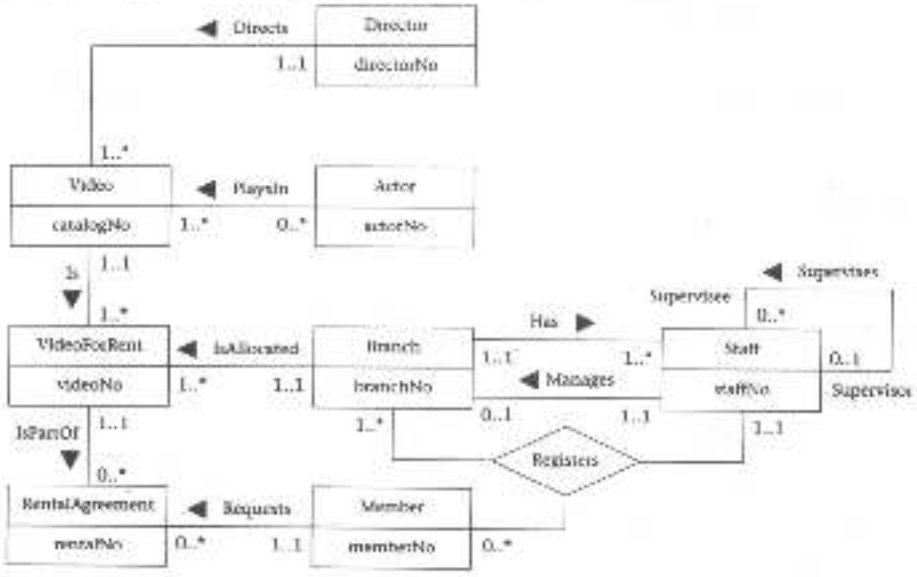


图9-9　显示主键的StayHome的Branch用户视图的ER模型

> 从StayHome所给的用户需求来说，对Director（导演）和Actor（演员）实体来说没有明显的键。实际上，Director实体中唯一用于标识的属性就是导演的名字，而Actor实体也是用演员名来标识实体的。正如我们刚才所说的，这些属性并不适合作主键，所以我们为这两个实体构造主键，分别叫做directorNo(导演号)和actorNo（演员号）。

2. 候选键、主键和备用键存档

在数据字典中记录候选键、主键和备用键(如果可获得)的定义。图9-10显示了从数据字典中选出的带有键的StayHome属性的文档。

Entity	Attributes	Description	Key	Nulls	...
Branch	branchNo	Uniquely identifies a branch	Primary key	No	
	address				
	street	Street of branch address		No	
	city	City of branch address		No	
	state	State of branch address		No	
	zipCode	Zip code of branch address	Alternate key	No	
	telNo	Telephone numbers of branch		No	
Staff	staffNo	Uniquely identifies a member of staff	Primary key	No	
	name	Name of staff member		No	

图9-10 数据字典中有主键和备用键定义的StayHome属性

9.4.6 步骤1.6：特化/泛化实体（可选步骤）

目标 如果合适的话，标识超类和子类实体。

在这个步骤中，可以选择使用特化或泛化过程来继续逻辑数据模型的开发。超类和子类的建模为数据模型添加了更多的信息，但是也使模型更加复杂。结果，由于这步是可选的，因此我们现在将略过特化和泛化的细节，有兴趣的读者可以参见第11章。

9.4.7 步骤1.7：检查模型的数据冗余

目标 在ER模型中检查冗余的情况。

在这个步骤中，用指定的目标检查ER模型是否有冗余存在，并删除这些冗余。在这个步骤中有三个活动：

1）重新检查一对一（1:1）关系。

2）删除冗余关系。

3）当访问冗余时考虑时间尺度。

1. 重新检查一对一（1:1）关系

在标识实体时，可能标识了两个实体，但它们表示同一个对象。例如，标识了实体Branch和Outlet，实际上它们是一样的。换句话说，Branch是Outlet的同义词。这时，这两个实体应该合并。如果主键不同，那么选择其中之一作为主键，另一个作为候选键。

2. 删除冗余关系

如果通过其他关系可以获得相同的信息，则这个关系就是冗余的。因为冗余关系是不必要的，而我们又试图建立最小化的数据模型，所以应该删除冗余的关系。标识在两个实体之间是否有多条路径相对比较容易，但这并不意味着这些关系之中有冗余的，因为它们可能代表着用户的不同的关系。

例如，考虑图9-11中VideoForRent、RentalAgreement和Member实体间的关系，有两种方法可以查看哪个会员租借了哪些录像。使用Member和VideoForRent实体间的Rents关系可以有一个直接的路径，而且通过RentalAgreement实体使用Requests和IsPartOf关系则有一个间接的通路。在访问这两个通路之前，我们需要建立每个关系的目的。Rents关系表明哪个会员租借了哪些录像。另一方面，Requests关系表明哪个会员拥有哪些租借协议，而IsPartOf关系表明哪些录像与哪些租借协议有关。尽管在会员和他们租借的录像之间确实存在关系，但这个关系不是直接的，通过租借协议所表达的关系会更准确。因此Rents关系是冗余的，这个关系并没有为VideoForRent和Member直接的关系带来任何附加的信息，VideoForRent和Member间的关系可以更简单地通过RentalAgreement实体得到。为确保我们创建一个最小的模型，必须将冗余的Rents关系删掉。

3. 当访问冗余时考虑时间尺度

当访问冗余时，关系的时间尺度也是很重要的。例如，考虑在实体Man、Woman和Child之间建立关系的情况，如图9-12所示。很明显，在Man和Child之间有两条路径：一条是通过直接关系FatherOf（是某人的父亲），另一条是通过关系MarriedTo（与某人结婚）和MotherOf

（是某人的母亲）。这样，你可能觉得FatherOf关系是不必要的。但这可能是不正确的，原因有两个：

- 父亲可能在前一次结婚中有孩子，而我们通过1:1关系所建的模型只是父亲当前婚姻的情况。
- 孩子的父亲和母亲可能没有结婚，或者孩子的父亲和孩子的母亲以外的人结了婚（或者孩子的母亲嫁给了并不是孩子的父亲的人）。

以上任何一种情况，都必须建立FatherOf关系。

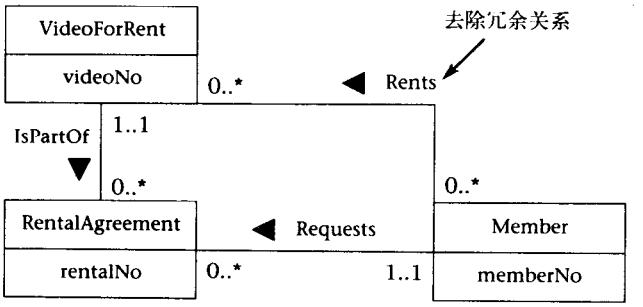


图9-11　去除冗余关系

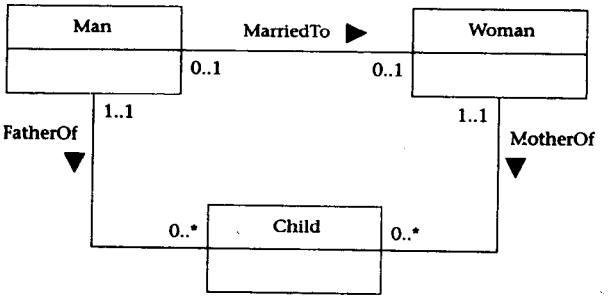


图9-12　无冗余的关系

提示　这里要说明的意思是，当访问冗余时，检查实体间每个关系的意义是很重要的。

9.4.8　步骤1.8：检查模型是否支持用户事务

目标　确保ER模型支持视图所需的事务。

现在已经得到了一个描述公司数据需求的ER模型。这个步骤的目标就是检查ER模型，确保模型支持所需要的事务。在我们的例子中，StayHome的Branch用户视图的事务需求列在6.4.4节。

使用ER模型和数据字典，尝试手工地完成操作。如果可以以这种方式解决所有事务，就已经完成了ER模型是否支持用户事务的检查。但是，如果不能手工地操作一个事务，则数据模型一定会有问题存在，而且这个问题是必须要解决的。在这种情况下，很可能是数据模型中丢失了一个实体、关系或者属性。

我们检查两个可能的方法来确保ER模型支持所需的事务。

1. 描述事务

使用第一种方法，通过将每个事务的需求描述存档，检查模型中是否提供了事务所需的

所有信息（实体、关系和其他属性）。下面，让我们来检查StayHome的一个示例事务的需求:

事务（o）根据分公司号，列出了每个分公司的每个主管的名字。

每个主管的名字均包含在实体Staff中，而分公司的具体情况包含在实体Branch中。这里可以使用Staff Manages Branch关系，找出每个分公司主管的名字。

2. 使用事务路径

第二个方法，根据必需的事务使数据模型有效化，可以在ER模型中以图形方式直接描述每个事务的路径。此方法的一个例子是使用6.4.4节中列出的数据查询，如图9-13所示。很明显，事务越多，图越复杂。为了使可读性良好，可能需要几张这样的图来描述所有事务。

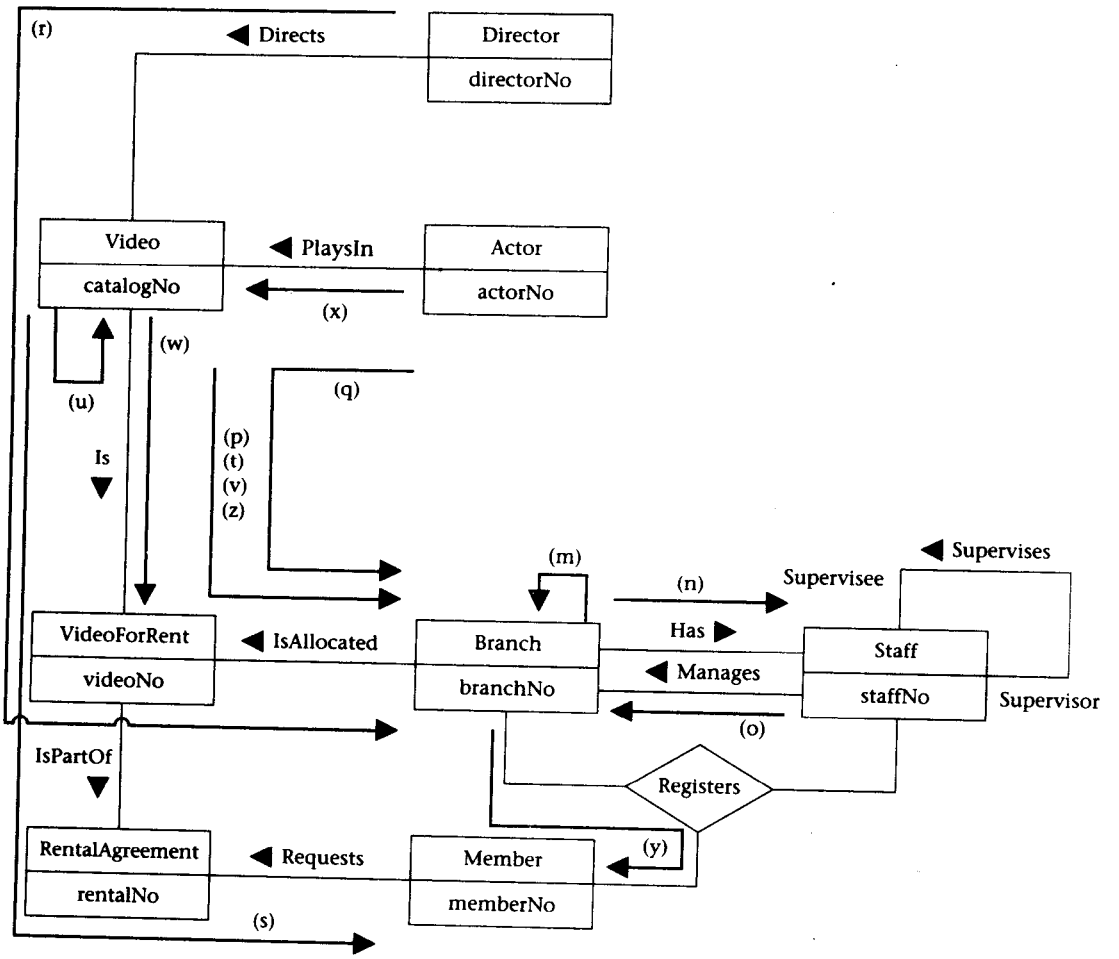


图9-13 使用路径来检查ER模型是否支持用户事务

提示 这些看起来很难，但确实是这样。因此，你可能会尽量略去该步骤。但是现在进行检查比将来做要好得多，因为你会发现在后面做检查要复杂得多，而且比在数据模型中解决问题要付出更多的代价。

9.4.9 步骤1.9：与用户一起检查模型

目标 与用户一起检查ER模型可以确保此模型是“真实”的表达了企业（和企业的一部分）的需求。

在完成步骤1之前，应该和用户一起检查ER模型。ER模型包括数据字典、ER图和任何附加的描述数据模型的文档。如果在数据模型中有任何异常，必须进行相应的修改，这可能需要重复以前的一些步骤。你应该重复这个过程直到用户肯定这个模型能够真实地表达企业（或企业的一部分）的需求为止。

在下面的各章中，我们继续后边的主要步骤，这些步骤将数据模型映射为一系列表的集合，并检查这些表是否满足用户的需求。

9.5 本章小结

- 设计方法学就是一种使用过程、技术、工具和文档来支持和简化设计过程的结构化的方法。
- 本书中使用的数据库设计方法学有两个主要的阶段：逻辑数据库设计和物理数据库设计。
- 逻辑数据库设计是基于一个特定数据模型来构建企业中使用的数据模型的过程，它独立于特定的DBMS和其他的物理因素。在我们的例子中，逻辑设计是基于关系模型的。
- 物理数据库设计是描述在二级存储中的数据库如何实现的过程，它描述了文件的组织方式和高效访问数据的索引，以及有关的完整性约束和安全性限制。物理设计与具体的DBMS有关。
- 在数据库设计中，有一些关键的因素决定数据库设计是否成功，例如，与用户进行交互以及愿意重复一些步骤。
- 方法学中步骤1的主要目的是构建表达企业（或企业的一部分）的数据需求的ER模型。
- ER模型包括实体、关系、属性、属性域、候选键、主键和备用键。
- ER模型由文档描述，文档包括ER图和一个数据字典。
- 应该检查ER模型以确保没有任何冗余并支持用户需要的事务。

复习题

9.1 描述设计方法学的目的。
9.2 描述数据库设计包含的主要阶段。
9.3 标识成功的数据库设计的重要因素。
9.4 讨论在数据库设计过程中，用户所起的重要作用。
9.5 讨论与逻辑数据库设计方法学有关的每个步骤的主要活动。
9.6 讨论与物理数据库设计方法学有关的每个步骤的主要活动。
9.7 讨论逻辑数据库设计的步骤1的目的。
9.8 标识与逻辑数据库设计步骤1有关的主要任务。
9.9 讨论从用户的需求文档中标识实体和关系的方法。

9.10 讨论从用户的需求文档中标识属性的方法以及属性与实体和关系的结合。

9.11 讨论为ER模型检查冗余的方法，给出一个例子来说明你的答案。

9.12 讨论检查一个ER模型是否支持用户需要的事务的两个方法。

9.13 标识并描述在逻辑数据库设计步骤1中产生文档的目的。

练习

9.14 标识附录E中给出的每个案例研究中的实体、关系和有关的属性，并创建ER图（不要看本书给出的每个案例研究的答案）。比较你的ER图和答案的ER图，并说明不同之处。

第10章 逻辑数据库设计——步骤2

本章主题:

- 如何将ER模型映射为表集合。
- 如何使用规范化方法检查表结构。
- 如何检查所建的表是否支持用户需要的事务。
- 如何定义和存档完整性约束。

本章讲述了逻辑数据库设计方法学（方法学的介绍见附录B）中的第二个步骤。在这个步骤中，为在步骤1中创建的ER模型创建表集合。然后通过规范化方法检查表的结构和它们是否支持用户的事务。最后，在最终的逻辑数据模型中检查所有的业务规则。

10.1 步骤2：将ER模型映射为表

目标 从ER模型中创建表，并检查这些表的结构。

这个步骤的主要目标是为步骤1中建立的ER模型产生表的描述。这组表应该代表逻辑数据模型中实体、关系、属性和约束。然后检查每个表的结构，确保建表过程中没有产生错误。如果表中有错误，则表明在建表的过程中或在ER模型中仍有没发现的错误。步骤2中的任务包括:

- 步骤2.1: 创建表。
- 步骤2.2: 用规范化方法检查表结构。
- 步骤2.3: 检查表是否支持用户事务。
- 步骤2.4: 检查业务规则。
- 步骤2.5: 与用户讨论逻辑数据库设计。

我们使用上一章步骤1中为StayHome创建的Branch视图的ER模型来介绍步骤2。这个视图代表Manager（管理者）、Supervisor（监理）和Assistant（助理）用户视图。

10.1.1 步骤2.1：创建表

目标 从ER模型映射表集合。

在这步中，为ER模型创建表来表达实体、关系、属性和约束。每个表的结构来源于ER所描述的信息，这些信息包括ER图、数据字典和任何其他相关的文档。我们使用关系数据库定义语言（Database Definition Language，DBDL）来描述每个表的组成。使用DBDL，首先指定表的名字，在后面跟着该表的简单属性的名字，并将这些属性的名字括在括号里。然后标识该表的主键以及任何备用键和外键。对每个外键，还要给出包含被引用主键的表。（关系键定义参见2.2.3节。）

我们将使用StayHome中的Branch视图的ER模型来说明这个过程，Branch视图的ER模型如图9-9所示。但在某些情况下，有必要添加一些本模型中没有的例子来说明特殊的要点。

(1) 如何表达实体

对ER模型中的每个实体，创建一个包含实体的所有简单属性的表。对复合属性，仅包含表中组成复合属性的简单属性。例如，对复合属性address，应该包含它的简单属性street、city、state和zipCode。如果可能，标识每个表中组成主键的列。对图9-9中所示的实体，应该创建如图10-1所示的初始表结构。

Actor (actorNo, actorName) **Primary Key** actorNo	**Branch** (branchNo, street, city, state, zipCode) **Primary Key** branchNo **Alternate Key** zipCode
Director (directorNo, directorName) **Primary Key** directorNo	**Member** (memberNo, fName, lName, address) **Primary Key** memberNo
RentalAgreement (rentalNo, dateOut, dateReturn) **Primary Key** rentalNo	**Staff** (staffNo, name, position, salary) **Primary Key** staffNo
Video (catalogNo, title, category, dailyRental, price) **Primary Key** catalogNo	**VideoForRent** (videoNo, available) **Primary Key** videoNo

图10-1 描述图8-16所示实体的初始表结构

在有些情况下，我们还不能标识出组成表的所有列。原因是我们还必须要描述实体间的关系，这意味着在ER模型中没有表达出关系之前我们不能标识出组成弱实体的主键的列。由于在图9-9中没有弱实体，因此我们使用图C-1中的一个弱实体VideoOrderLine的例子，在本步骤的最后讨论这个弱实体的主键列的标识。

(2) 如何表达关系

一个实体与另一个实体间的关系（关系在7.2节讨论）由主键/外键机制表达。为了决定将外键属性放在哪里，首先必须标识关系中包含的“父”实体和“子”实体。父实体指的是把自己的主键拷贝到代表子实体的表中作为外键的实体。

我们考虑不同类型的关系和多值属性的父/子实体的标识：

- 一对多（1:*）二元关系
- 一对多（1:*）递归关系
- 一对一（1:1）二元关系
- 一对一（1:1）递归关系
- 多对多（*:*）二元关系
- 复杂关系
- 多值属性

1. 一对多（1:*）二元关系

对每个1:* 二元关系（1:*关系在7.5.2节讨论），关系“一”端的实体被设计为父实体，“多”端的实体被设计为子实体。为了描述这种关系，父实体主键的拷贝，被放置在子实体的表中，作为外键。

现在，由图9-9所示的Branch Has Staff关系来看怎样将1:*关系描述为表。在这个例子中，Branch（分公司）在“一”边，代表了父实体；Staff在“多”边，代表了子实体。这两个实体间的关系是通过将Branch（父）实体的主键（叫做branchNo）拷贝到Staff（子）表中建立的。图10-2a说明了Branch Has Staff ER图，图10-2b显示了相应的表。

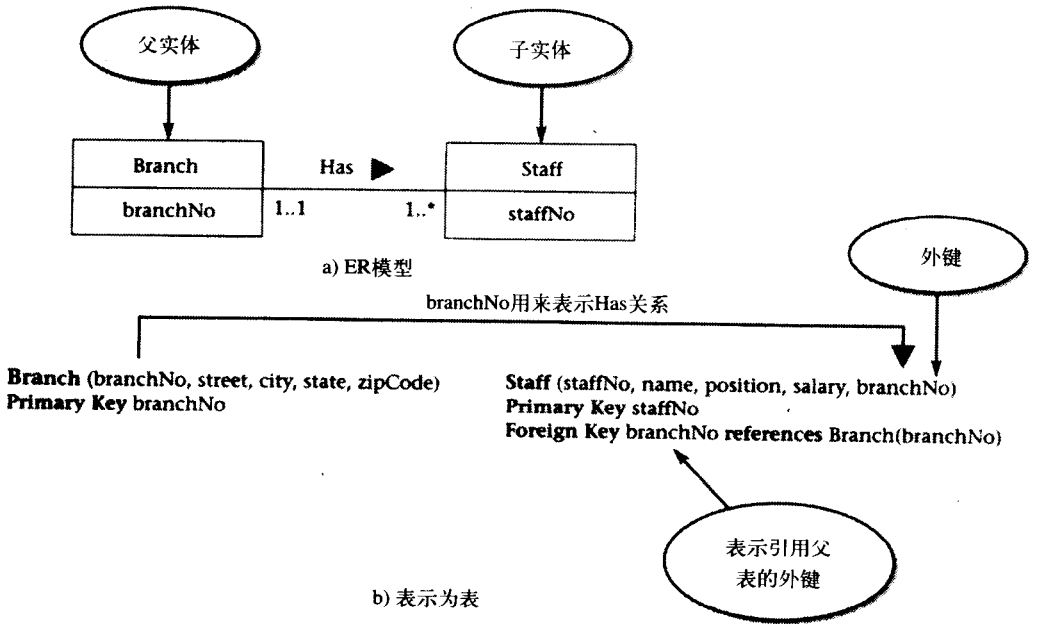


图10-2　1:* 关系Branch Has Staff

图9-9中还有几个其他的1:* 关系的例子，如Director Directs Video和Member Requests RentalAgreement。对局部逻辑数据模型中的每个1:* 关系都应该重复上述给出的过程。

> 在1:* 关系有一个或多个属性的情况下，这些属性也应随着主键加到子表中。例如，如果Branch Has Staff关系有一个叫做dateStart(开始日期，代表一个成员加入分公司的开始时间)的属性，则这个属性也应随着Branch表主键的拷贝加入到Staff表中。

2. 一对多（1:*）递归关系

1:* 递归关系的表示与上面所介绍的很类似。在图9-9中，有一个1:* 递归关系（递归关系在7.2.2节中定义）：Staff Supervises Staff。这时，父实体和子实体都是Staff。根据上面给出的规则，应把Staff(父)实体的主键拷贝到Staff(子)表中来表达Supervises关系，这时就应该建立这个列的第二个拷贝来作为外键。这个列的拷贝被重新命名为supervisorStaffNo，由此来更好地表示它的意思。图10-3a说明了Staff Supervises Staff ER模型，图10-3b显示了相应的表（包含完整的Branch Has Staff关系）。

3. 一对一（1:1）二元关系

创建代表1:1关系（1:1关系在7.5.1节定义）的表稍微有些复杂，因为不能使用元组的数目来标识一个关系中的父实体和子实体，而是需要使用参与（参与的定义见7.5.5节）过程来决定是把实体结合为一个表来表示关系好，还是建两个表由外键来表示关系好。我们考虑如何建表来表示如下的参与约束：

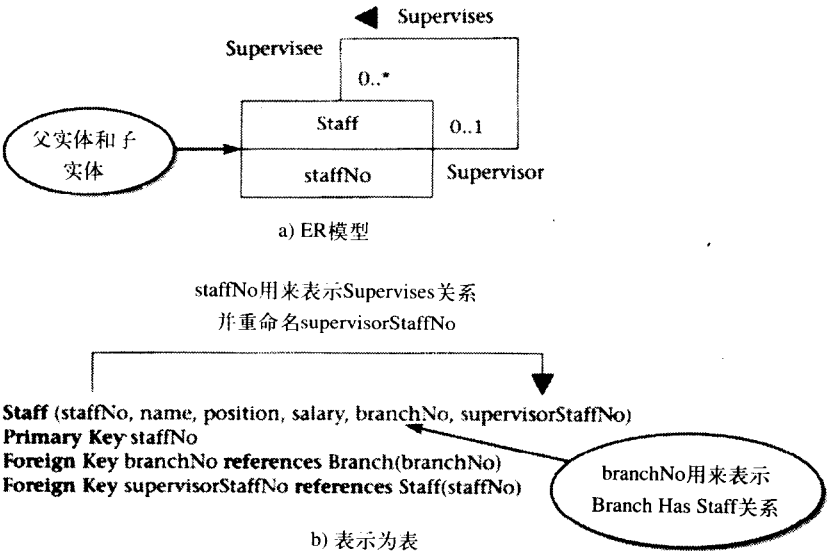


图10-3 1:* 递归关系Staff Supervises Staff

- 1:1关系的两边都是强制参与。
- 1:1关系的一边是强制参与。
- 1:1关系的两边均为可选参与。

(1) 1:1关系的两边都是强制参与

在这种情况下，应该将实体组合为一个表，并选择初始实体中的一个主键作为新表的主键，其他的主键用作备用键。

在图9-9所示的关系中没有这样的例子。然而，让我们看一下如何表达称为Staff Uses Car 的1:1关系，其所涉及的两个实体均是强制参与，如图10-4a所示。Car实体的主键是车牌号（vehLicenseNo），另两个属性为make和model。在这个例子中，把实体Staff和Car的所有属性放入一个表中。选择一个主键作为新表的主键，如staffNo，另一个就成为了备用键，如图10-4b所示。

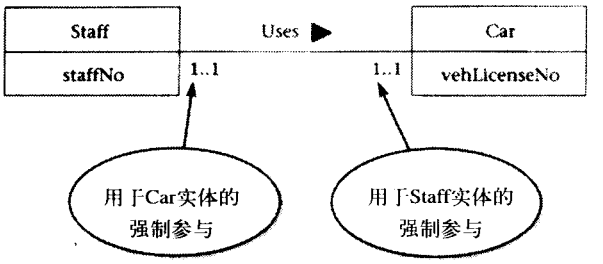
> 在这个例子中，有强制参与的1:1关系的两边都有一个或多个属性，这些属性也应该被包括在描述实体和关系的表中。例如，如果Staff Uses Car关系有一个叫做dateStart的属性，则这个属性也应该是StaffCar表中的一列。

注意，仅当在两个实体之间没有其他关系的时候，才有可能把两个实体合并到一张表中。如果还存在其他关系，则应该用主键/外键机制来描述Staff Uses Car关系。我们在第三部分只简要地讨论如何设计这种情况下的父实体和子实体。

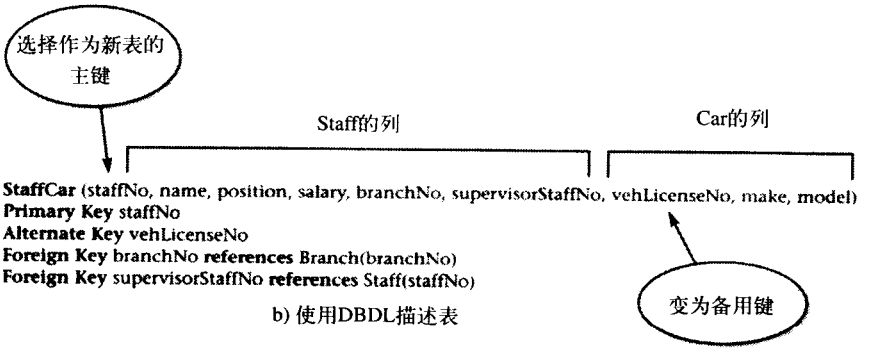
(2) 1:1关系的一边是强制参与

在这种情况下，可以使用参与约束来标识1:1关系的父实体和子实体。关系中的可选参与的实体被设计为父实体，关系中的强制参与的实体被设计为子实体。正如前面所说的，父实体主键的拷贝，被放置在描述子实体的表中。

将有可选参与的实体（父实体）的主键拷贝到有强制参与的实体（子实体）中的原因是，因为这个主键的拷贝（外键）将总是持有一个值，从而避免了在结果表中这个列出现空值。如果我们不遵守这个规则，并选择将外键放置在表示有可选参与的实体的表中，则这个列将会包含空值。

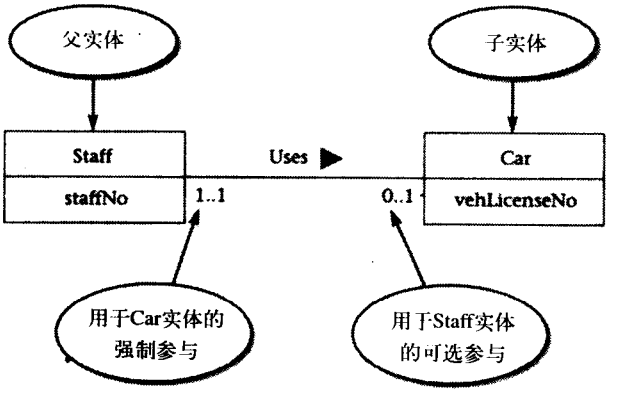


a) ER模型

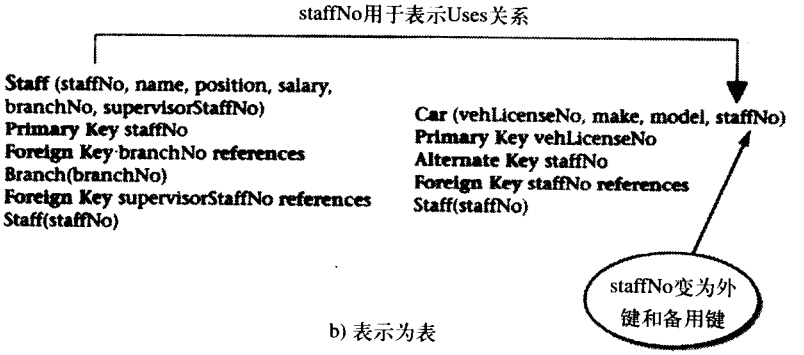


b) 使用DBDL描述表

图10-4 两个实体均有强制参与的1:1Staff Uses Car关系

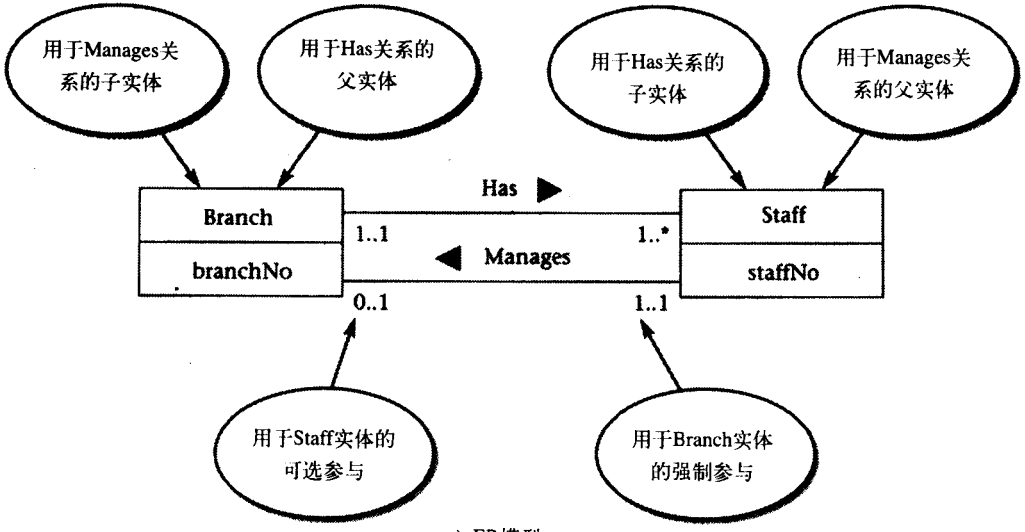


a) ER模型



b) 表示为表

图10-5 有强制参与的Car实体和可选参与的Staff实体的1:1 Staff Uses Car关系



a) ER模型

branchNo用于表示Has关系

staffNo用于表示Manages关系，
并重命名为mgrStaffNo

Branch (branchNo, street, city, state, zipCode, mgrStaffNo)
Primary Key branchNo
Foreign Key mgrStaffNo **references** Staff(staffNo)

Staff (staffNo, name, position, salary, branchNo, supervisorStaffNo)
Primary Key staffNo
Foreign Key branchNo **references** Branch(branchNo)
Foreign Key supervisorStaffNo **references** Staff(staffNo)

b) 使用DBDL描述表

图10-6 对Branch实体有强制参与，而对Staff实体有可选参与的Staff Manages Branch关系

现在我们只对Car实体考虑如何描述具有强制参与的Staff Uses Car 1:1关系，如图10-5a所示。在关系中具有可选参与的Staff实体被设计为父实体，而在关系中有强制参与的Car实体被设计为子实体。因此， Staff实体(父实体)主键的拷贝（staffNo）放置在了Car实体（子实体）的表中，如图10-5b所示。在这种情况下，staffNo也成为了Car表的备用键。

图9-9中也有单边强制参与的1:1关系的一个例子，它就是Staff Manages Branch关系，它只有Branch实体这一边为强制参与。按照上面的规则，Staff实体设计为父实体， Branch实体设计为子实体。因此，Staff(父)实体的主键的拷贝——staffNo，应复制到Branch（子）表中，并重新命名为mgrStaffNo，以便更清晰地表达Branch表中的外键含义。图10-6a为Staff Manages Branch 的ER 模型，图10-6b中为相对应的表。

> 在只有一边实体强制参与的1:1关系中有一个或多个属性的情形中，这些属性也应该随其主键加入到子表中。例如，如果Staff Manages Branch关系有一个叫做dateStart的属性，则这个属性将随着staffNo(重命名为mgrStaffNo)的拷贝加入到Branch表中，成为该表中的一列。

(3) 1:1关系的两边均为可选参与

在这种情况下，父实体和子实体之间的设计是任意的，除非你可以得到关于关系的更多信息来帮助你判断使用哪个设计。

让我们考虑如何描述两边都是可选参与的Staff Uses Car关系，这个关系是1:1的，如图10-7a所示（注意，下面的讨论与具有两边强制参与的1:1关系也是有关的，但这时不能把所有的选项都放在一个表中）。如果没有额外的信息帮助你选择父、子实体，则你的选择就是任意的。换句话说，你既可以把Staff实体的主键拷贝到Car实体中，也可以反过来做。

然而，让我们假设你发现大部分汽车（car），但不是所有的都被员工使用，而只有一小部分员工使用汽车。现在，可以说Car实体（尽管是可选的）比Staff实体更接近于强制类型。因此，可以令Staff实体为父实体，令Car实体为子实体，并且将Staff实体主键（staffNo）的拷贝放置在Car表中，如图10-7b所示（Staff表和Car表的组合，与前面讨论的在一边具有强制参与的1:1关系的例子一样）。

4. 一对一（1:1）递归关系

对于1:1递归关系，应该遵循上面所描述的对1:1关系的“参与”规则。但是，在这种特殊的1:1关系情景中，关系两边的实体是相同的，对于在两边有强制参与的1:1递归关系，应该用主键的两个拷贝来把这个递归关系描述为一个表。同前面一样，主键的一个拷贝代表外键，并且应该将它重新命名来表示它代表的关系。

对于一边是强制参与的1:1递归关系，你既可以向前面描述的那样，用主键的两个拷贝创建一个新表，也可以创建一个新表来代表关系。这个新表只有两个列，都是主键的拷贝。同前面一样，主键的拷贝作为外键，并且必须重新命名来表示在表中的意思。

对于两边是可选参与的1:1递归关系，应该像前面所描述的那样创建一个新表。

5. 多对多（*:*）二元联系

对于每个*:*二元联系，创建一个表达关系的表，这个表包含关系的任何属性。我们将参与关系的实体的主键属性拷贝到新表中，使之作为外键。一个外键或全部外键将组成新表的主键，可能要结合此关系的一些属性。

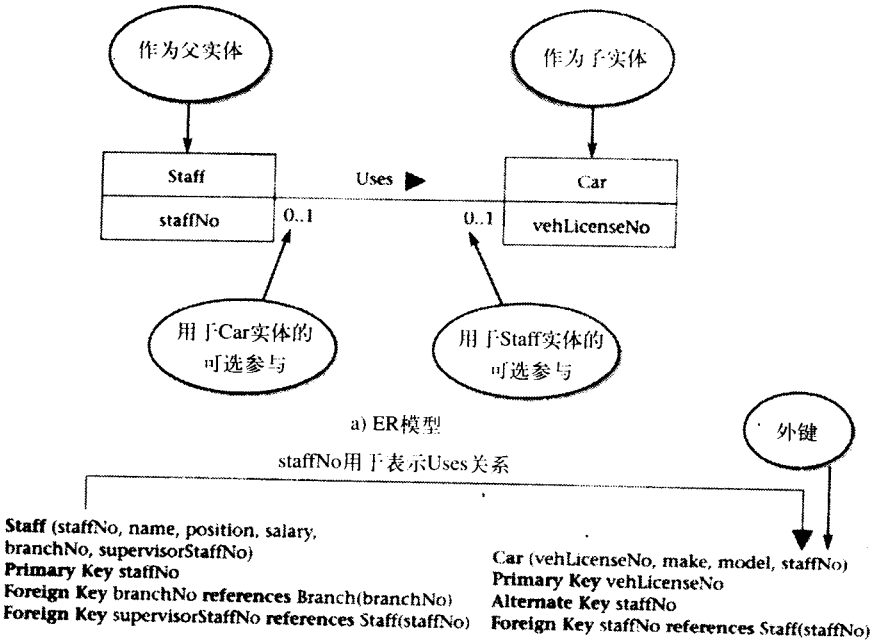


图10-7 两个实体都是可选参与的 1:1 Staff Uses Car关系

例如，考虑图9-9所示的Actor PlaysIn Video这个*:*关系。这个*:*关系两边的实体分别为Actor和Video，作为父实体，将它们的主键（actorNo和catalogNo）复制到叫做Role的新表中来表达这个关系。注意，PlaysIn关系有一个叫做character的属性，这个属性也被包含在Role表中。图10-8a所示的为Actor PlaysIn Video ER图，图10-8b所示的是相应的表。

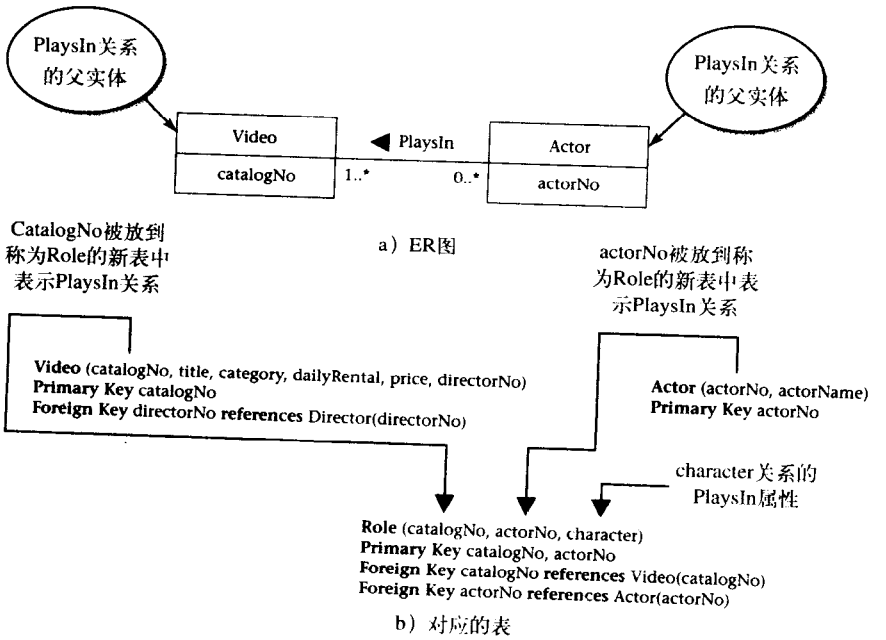


图10-8 *:* Actor PlaysIn关系

注意，叫做Role的新表有一个复合主键，这个主键由两个外键(catalogNo和actorNo)组成。

6. 复杂关系类型

有多于两个参与实体的关系是复杂关系，为每个复杂关系创建一个表达关系的表，将参与复杂关系的这些实体的主键复制到新表中，并作为外键，此表还包含与关系相关的全部属性。一个或多个外键将组成新表的主键，还可以加上关系中的一些其他属性。

例如，复杂（三元）关系Registers表达了图9-9中所示的作为分公司的一个新会员注册的会员、职工和分公司间的关系。Registers周围有Staff、Member和Branch三个实体，它们作为父实体，我们将这些实体的主键（staffNo、MemberNo和branchNo）复制到一个叫做Registration的新表中，这个新表表达了这个关系。注意，Registers关系有一个叫做dateJoined的属性，这个属性也包含在Registration表中。图10-9a显示了Registers复杂（三元）关系的ER图，图10-9b显示了相应的表。

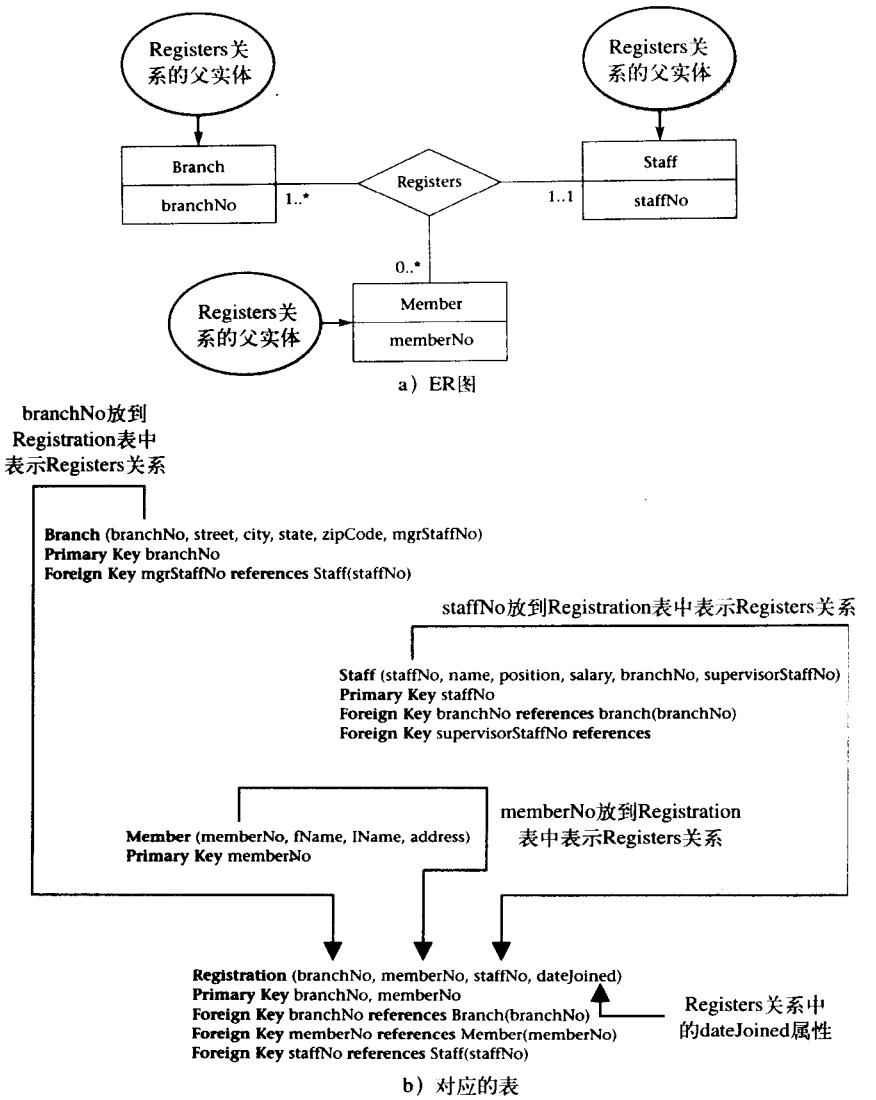
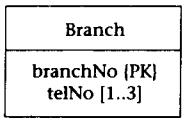


图10-9　复杂的Registers关系

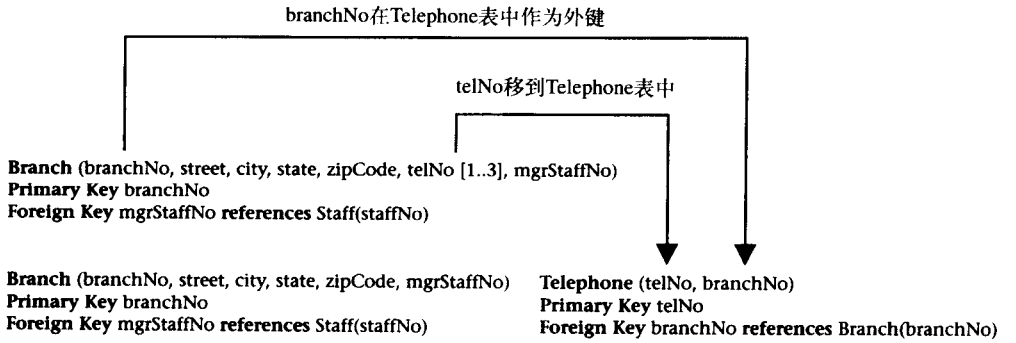
7. 多值属性

对于每个与实体有关的多值属性，应该遵守上述1:*关系中所描述的规则。在一端的实体被指定为父实体，在多端的多值属性被指定为子实体。创建一个新的表包含这些多值属性，并将父实体的主键拷贝过来作为外键。除非多值属性自己本身是父实体的备用键，否则新表的主键由多值属性和父实体的原始主键组成。

例如，要表达有三部电话的分公司的情况，Branch实体定义telNo属性多值属性。要表达这个关系，我们创建一个新的叫做Telephone的表来表达多值属性telNo。图10-10a说明了Branch实体的ER图，这个图中只显示了主键和telNo多值属性，图10-10b显示了对应的表。



a）ER图



b）对应的表

图10-10 Branch实体的多值telNo属性

在表10-1中，我们总结了如何用表表达实体、关系和多值属性。

在数据库设计方法学的步骤1.6中，我们给出了使用特化/泛化的增强的ER概念来表达实体，第11章会详细描述这部分内容，我们也将在第11章介绍如何将增强的概念映射到表中。

表10-1 如何将实体、关系和多值属性表达为表的总结

实体/关系/属性	表达为表
强实体或弱实体	创建包含所有简单属性的表
1:*二元关系	将“一”端实体的主键复制到表达“多”端实体的表中，关系中的任何属性也复制到“多”端的表中
1:*递归关系	“一”端的实体和“多”端的实体是一样的，代表实体的表有主键的另一个拷贝，这个拷贝是被重命名的，并且有关系的其他属性
1:1二元关系:	
两端都是强制参与	将实体组合成一张表
一端是强制参与	将有可选参与的实体的主键复制到表达有强制参与的实体的表中，关系的属性也被复制到表达有强制参与的实体的表中
两端都是可选参与	没有更多的信息，将一个实体的主键拷贝到另一个实体中。但如果信息是可获得的，则将更具有强制参与的实体作为子实体

（续）

实体/关系/属性	表达为表
*:*二元关系/复杂关系	创建表达关系的表，此表中包含任何与关系有关的属性，将每个父实体中的主键复制到新表中作为外键
多值属性	创建一个表达多值属性的表，并将父实体的主键复制到新表中作为外键

将表和外键属性存档

在步骤2.1的结尾，将从逻辑数据模型得来的表的全部组成部分进行存档。StayHome数据库应用程序的Branch视图的所有表显示在图10-11中。

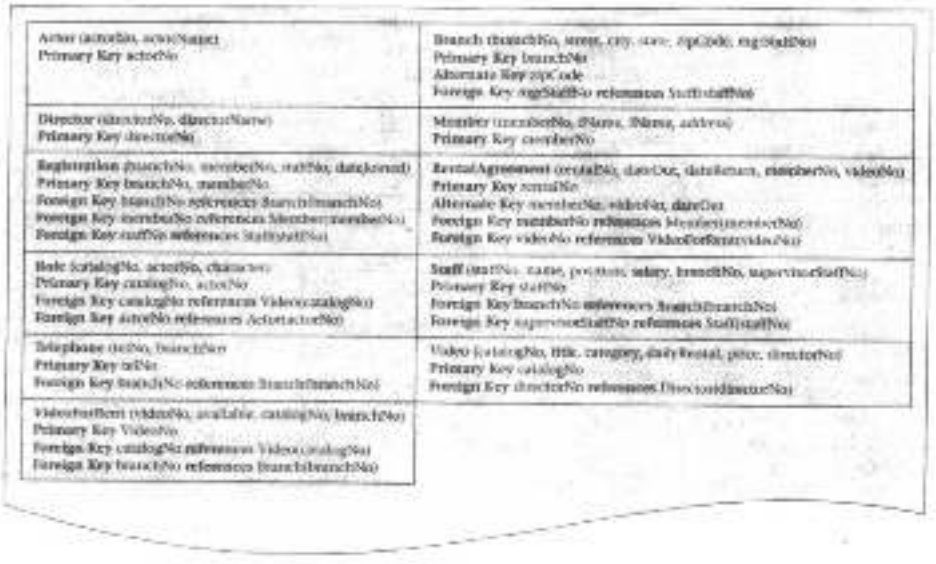


图10-11　StayHome数据库应用的Branch视图中的表

既然每个表都有了自己的全部列的集合，现在就可以标识任何新的主键或备用键了。这对于依赖于从父实体那里拷贝主键来组成自己的主键的弱实体来说尤其重要。例如，附录C的图C-1中显示的VideoOrderLine弱实体有一个复合主键，此主键由实体Video的主键catalogNo的拷贝和实体VideoOrder的主键OrderNo的拷贝共同组成。如图C-2所述。

可以扩展DBDL的语法来表示外键的完整性约束，你将在步骤2.4看到这些。同时也应该更新数据字典来表明这个步骤中新出现的主键和备用键的存在。例如，加入主键之后，表RentalAgreement获得了一个新的备用键，该备用键由memberNo、videoNo和dateOut共同组成。

10.1.2　步骤2.2：用规范化方法检查表结构

目标　检查和使用规范化标准，使每个表的结构正确。

这个步骤的目的是检查步骤2.1中创建的每个表中的列的组成，用规范化方法检查每个表的组成来避免不必要的数据重复。

应该确保步骤2.1中所建的每个表至少是第三范式（3NF）的。如果所标识的表不是第三范式的，可能表明ER模型的某部分是错误的、或者由模型创建表时产生了错误。如果必要的话，可能需要重新构造数据模型或者表。

10.1.3 步骤2.3：检查表是否支持用户事务

目标 确保所建表支持用户所需的事务。

这个步骤的目标是检查步骤2.1中所建的表是否如用户需求说明中所要求的那样，支持用户所需的事务。这种类型的检查在步骤1.8中已经进行过，它是确保局部逻辑数据模型支持所需求的事务。在这个步骤中，是检查前面步骤所建的表是否也支持这些事务，并由此确保在建表的时候，没有错误发生。

检查表是否支持事务的一种方法是检查是否支持事务的数据需求，以确保数据在一个或多个表中存在。同时，如果事务所需求的数据在多个表中，则应该检查这些表是否能够通过主键/外键机制连接起来。我们通过检查6.4.4节中所给的事务的例子来说明这种方法。表10-2a给出了数据项和更新/删除事务，表10-2b给出了StayHome的Branch视图的查询事务，这两个图同时给出了每个事务所需要的表。在这里，我们加重显示了事务所需要的列，必要时还可包括在连接表中的列。

从这些分析得出，图9-8所示的表支持StayHome的Branch视图的所有事务。在第17章中，我们将使用Microsoft Access DBMS来说明对表10-2b所示的一些查询事务的实现。

表10-2a) StayHome的Branch视图的数据项和更新/删除事务所需的表

事　务	需求的表
(a) 输入一个新的分公司的详细信息	**Branch (branchNo, street, city, state, zipCode, mgrStaffNo)**
(g) 更新/删除分公司的详细信息	**Telephone (telNo, branchNo)** 外键branchNo引用Branch (branchNo)
(b) 输入分公司中一个新成员的详细信息Staff (h) 更新/删除分公司成员的详细信息	**Staff (staffNo , name, position, salary, branchNo, supervisorStaffNo)**
(c) 输入新出版的录像的详细信息 (i) 更新/删除已有的录像的详细信息	**Video (catalogNo, title, category, dailyRental, price, directorNo)** 外键directorNo引用Director (directorNo)
	Director (directorNo , directorName)
	Role (catalogNo , actorNo , character) 外键CatalogNo引用Video(catalogNo) 外键actorNo引用Actor(actorNo)
	Actor (actorNo, actorName)
(d) 输入已有分公司的复制新录像的详细信息 (j) 更新/删除录像拷贝的详细信息	**VideoForRent (videoNo , available, catalogNo, branchNo)**
(e) 输入在已有分公司中登记的新成员的详细信息 (k) 更新/删除已有成员的详细信息	**Member (memberNo, fName, lName, address)** **Registration (branchNo , memberNo , staffNo, dateJoined)** 外键memberNo引用Member (memberNo)
(f) 输入某成员租用录像的出租协议的详细信息 (l) 更新/删除租用录像的成员的给定出租协议的详细信息	**RentalAgreement (rentalNo, dateOut, dateReturn, memberNo, videoNo)**

从这个分析可以看出，图10-11显示的这些表支持所有的StayHome的Branch用户视图的事务。

表10-2b) StayHome的Branch视图的查询事务所需的表

事 务	需求的表
(m) 列出在给定城市的分公司的详细信息	**Branch (branchNo, street, city, state, zipCode, mgrStaffNo)** **Telephone (telNo, branchNo)** 外键branchNo引用Branch (branchNo)
(n) 列出给定分公司中的员工的名字、职务和工资，按员工名字排序	**Staff (staffNo, name, position, salary, branchNo,** supervisorStaffNo)
(o) 列出每个分公司经理的名字，按分公司号排序	**Branch (branchNo**, street, city state, zipCode, **mgrStaffNo)** 外键mgrStaffNo引用Staff (StaffNo) **Staff (staffNo, name**, position, salary, branchNo, supervisorStaffNo)
(p) 列出某具体分公司中所有录像的题目、类别和是否可获得，按种类排序	**Video (catalogNo, title, category**, dailyRental, price, directorNo) **VideoForRent (videoNo, available, catalogNo, branchNo)** 外键catalogNo引用Video (catalogNo)
(q) 列出某一具体分公司中对所给定的演员所参加演出的所有录像的题目、类别和是否可获得，按题目排序	**Actor (actorNo, actorName)** **Role (catalogNo, actorNo**, character) 外键catalogNo引用Video (catalogNo) 外键actorNo引用Actor (actorNo) **Video (catalogNo, title, category**, dailyRental, price, directorNo) **VideoForRent (videoNo, available, catalogNo, branchNo)** 外键catalogNo引用Video (catalogNo)
(r) 列出某一具体分公司中对于所给出的导演所执导的所有录像的题目，类别和是否可获得	**Director (directorNo, directorName)** **Video (catalogNo, title, category**, dailyRental, price, **directorNo)** 外键directorNo引用Director (directorNo) **VideoForRent (videoNo, available, catalogNo, branchNo)** 外键catalogNo引用Video (catalogNo)
(s) 列出具体成员当前正在租用的所有录像的详细信息，按题目排序	**Video (catalogNo, title, category, dailyRental, price, directorNo)** **VideoForRent (videoNo, available, catalogNo,** branchNo) 外键catalogNo引用Video (catalogNo) **RentalAgreement (rentalNo, dateOut, dateReturn, memberNo, videoNo)** 外键videoNo引用VideoForRent(VideoNo) 外键memberNo引用Member (memberNo) **Member (memberNo , fName, lName**, address)
(t) 列出具体分公司给定的录像拷贝的详细信息	**Video (catalogNo , title, category, dailyRental, price, directorNo)** **VideoForRent (VideoNo, available, catalogNo, branchNo)** 外键videoNo引用VideoForRent (videoNo)

（续）

事　务	需求的表
(u) 列出某一类别中所有录像的标题，按标题排序	**Video** (catalogNo, **title, category**, dailyRental, price, directorNo)
(v) 列出每个分公司中每个类别中录像的总数，按分公司号排序	**Video (catalogNo, title, category**, dailyRental, price, directorNo) **VideoForRent** (videoNo, available, **catalogNo, branchNo)** 外键catalogNo引用Video (catalogNo)
(w) 列出所有分公司录像的总价值	**Video (catalogNo**, title, category, dailyRental, **price**, directorNo) **VideoForRent** (videoNo, available, **catalogNo**, branchNo) 外键catalogNo引用Video (catalogNo)
(x) 列出每个演员的录像总数，按演员的名称排序	**Video (catalogNo**, title, category, dailyRental, price, directorNo) **Role (catalogNo, actorNo**, character) 外键catalogNo引用Video (catalogNo) 外键actorNo引用Actor (actorNo) **Actor (actorNo, actorName)**
(y) 列出2002年加入每个分公司的成员的总数，按分公司号排序	**Registration (branchNo , memberNo,** staffNo, **dateJoined)**
(z) 列出每个分公司录像所有可能的日租金，按分公司号排序	**Video (catalogNo**, title, category, **dailyRental**, price, directorNo) **VideoForRent** (videoNo, available, **catalogNo, branchNo)** 外键catalogNo引用Video (catalogNo)

提示　与上一章中介绍的步骤1.8合在一起，这看起来似乎是一件很困难的工作，也确实是这样。因此，你可能想省略这一步。然而，现在做这些检查要比将来在数据模型中发现和解决错误好得多。

10.1.4 步骤2.4：检查业务规则

目标　检查逻辑数据库设计中表达的业务规则。

业务规则是用于防止数据库不完整、不准确或不一致的约束。尽管DBMS在完整性方面的控制可能存在也可能不存在，但这不是问题所在。在这个阶段，你只关心高级设计，即确定需要什么样的数据完整性约束，而不管怎样去实现。标识完完整性约束之后，就可以得到一个完整而准确地描述视图的局部逻辑数据模型。如果必要的话，可以从逻辑数据模型产生物理数据库设计，例如为用户构建系统的原型。

我们考虑下面五种类型的完整性约束：

- 需要的数据
- 列的值域约束
- 实体完整性
- 多样性
- 参照完整性
- 其他业务规则

1. 需要的数据

某些列必须要包含值，换句话说，它们不允许有空值（空值的定义参见2.3.1节）。例如，

每个成员必须有一个工作职位（例如经理和监理）。当在步骤1.3中将列（属性）存档到数据字典中时，就应该已经标识了这些约束。

2. 列的值域约束

每个列都有一个值域（一组对该列合法的值）。例如，员工中一个成员的职位是导演、经理、监理、助理或采购员，所以职位（position）这一列的值域就由并且仅由这些值组成。当在步骤1.4中为数据选择列（属性）域的时候就应该标识约束（域的定义见2.2.1节）。

3. 实体完整性

实体的主键不能为空。例如，Staff表中的每个记录的主键列staffNo必须有值。当在步骤1.5为每个实体标识主键时，这些约束就应该被考虑到（实体完整性定义见2.3.2节）。

4. 多样性

多样性表达了数据库中数据间的关系的约束。例如，分公司必须有会员而且每个分公司必需要有员工。确保标识并表达了所有合适的业务规则是对企业数据需求进行建模的重要方面。在步骤1.2中，我们定义了实体间的关系以及所有的业务规则，可以按这种方式表达的业务规则都定义在这个步骤的文档中了。

5. 参照完整性

外键包含与父表相匹配的主键值，使子表中的每个记录与父表中的记录关系起来。参照完整性（参照完整性定义见2.3.3节）意味着，如果外键有值，则这个值必须引用父表中存在的记录。例如，Staff表中branchNo列将员工中的成员与该成员所工作的Branch表的记录相关联。如果branchNo列非空，则它必然包含Branch表中branchNo列的一个已存在的值，否则该成员将被分配到一个不存在的分公司。

关于外键，有两点必须要强调。

• 外键允许为空吗？

例如，对员工来说没有分公司号，那么还能够保存该成员的详细信息吗？这个问题并不是说分公司号是否存在，而是说分公司号是否必须要被指定。通常，如果关系中的子表是强制参与的，那么就不允许为空。另一方面，如果子表是可选参与的，那么就允许为空。

• 如何保证参照完整性？

为保证参照完整性，应该指定存在约束（existence constraint），该约束定义了主键和外键在什么条件下能够被插入、更新或删除。考虑1:* 关系Branch Has Staff。Branch表的主键branchNo在Staff表是外键。让我们考虑如下六种情况。

(1) 情况1：向子表中插入记录（Staff）

为了保证参照完整性，检查新的Staff记录的外键branchNo是为空还是为一个Branch表中已存在的记录的值。

(2) 情况2：从子表中删除记录（Staff）

如果子表的一个记录被删除，参照完整性不受影响。

(3) 情况3：更新子表记录中的外键（Staff）

这与情况1相似。为了保证参照完整性，检查被更新的Staff记录的外键列（branchNo）是置为空值还是置为Branch记录中已存在的值。

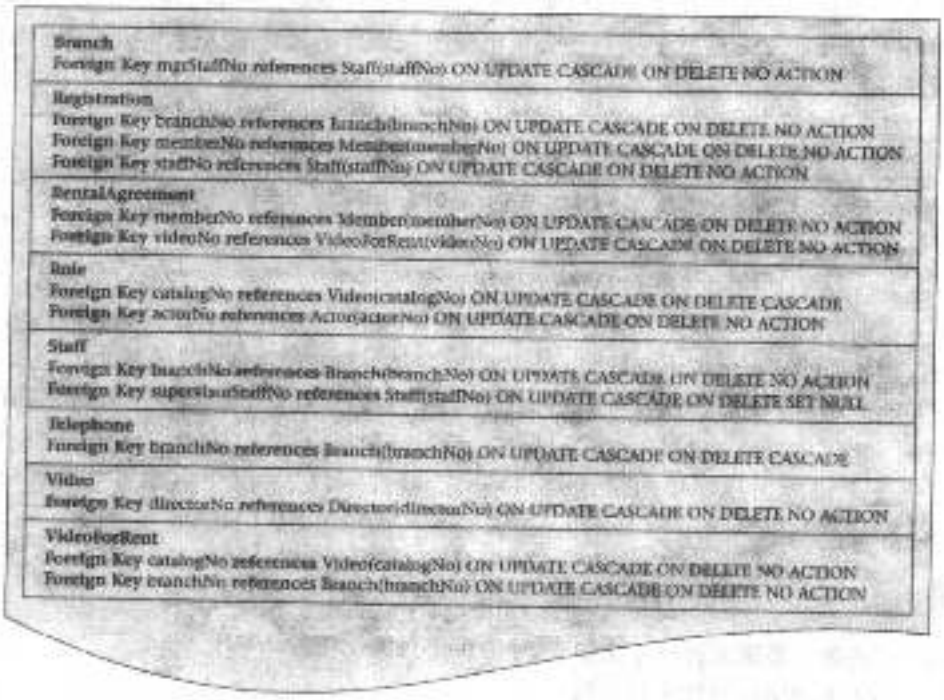
(4) 情况4：向父表中插入记录（Branch）

向父表中插入记录并不影响参照完整性，只不过是父亲没有孩子——换句话说，就是一个分公司没有成员。

(5) 情况5：从父表中删除记录（Branch）

若父表中的一个记录被删除了，如果有一个子记录引用这个被删除的父记录，则参照完整性就丢失了。换句话说，如果被删除的分公司中还有员工在其中工作，则就失去了参照完整性。在这种情况下，可以考虑如下几种操作：

- NO ACTION（不操作）：如果有任何相关的子记录，就不从父表中将该记录删除。在我们的例子中，"如果当前还有成员在工作，则不能删除分公司"。
- CASCADE（级联）：当父记录被删除后，自动删除相关的子记录。如果被删除的子记录也在其他关系中充当父记录，那么也应该删除那些子表中的相关记录。换句话说，删除从父表级联到子表。在我们的例子中，"删除一个分公司的同时，自动删除所有在那个公司工作的成员。"很明显，在这种情况下，这种策略并不明智。
- SET NULL（置空）：当父记录被删除的时候，相关子记录中外键的值自动被置为空值。在我们的例子中，"如果分公司被删除，表明原来在那个公司工作的员工现在所在的分公司是未知的。"正如步骤1.3中定义的，仅当组成外键的列允许为空时，才可以考虑这个策略。
- SET DEFAULT（置预定值）：当父记录被删除时，所有相关子记录中的外码的值自动置为他们的预置值。在我们的例子中，"如果分公司被删除，表明从前在该分公司工作的员工归属为另一个（预定的）公司。"正如步骤1.3所定义的，仅当组成外键的列有预置值时才可以考虑这种策略。



Branch
Foreign Key mgrStaffNo references Staff(staffNo) ON UPDATE CASCADE ON DELETE NO ACTION

Registration
Foreign Key branchNo references Branch(branchNo) ON UPDATE CASCADE ON DELETE NO ACTION
Foreign Key memberNo references Member(memberNo) ON UPDATE CASCADE ON DELETE NO ACTION
Foreign Key staffNo references Staff(staffNo) ON UPDATE CASCADE ON DELETE NO ACTION

RentalAgreement
Foreign Key memberNo references Member(memberNo) ON UPDATE CASCADE ON DELETE NO ACTION
Foreign Key videoNo references VideoForRent(videoNo) ON UPDATE CASCADE ON DELETE NO ACTION

Role
Foreign Key catalogNo references Video(catalogNo) ON UPDATE CASCADE ON DELETE CASCADE
Foreign Key actorNo references Actor(actorNo) ON UPDATE CASCADE ON DELETE NO ACTION

Staff
Foreign Key branchNo references Branch(branchNo) ON UPDATE CASCADE ON DELETE NO ACTION
Foreign Key supervisorStaffNo references Staff(staffNo) ON UPDATE CASCADE ON DELETE SET NULL

Telephone
Foreign Key branchNo references Branch(branchNo) ON UPDATE CASCADE ON DELETE CASCADE

Video
Foreign Key directorNo references Director(directorNo) ON UPDATE CASCADE ON DELETE NO ACTION

VideoforRent
Foreign Key catalogNo references Video(catalogNo) ON UPDATE CASCADE ON DELETE NO ACTION
Foreign Key branchNo references Branch(branchNo) ON UPDATE CASCADE ON DELETE NO ACTION

图10-12 StayHome的Branch视图的表的参照完整性约束

- NO CHECK（不检查）：当父记录被删除时，不做任何保证参照完整性的操作。这种策略只有极端的情况下才被考虑。

(6) 情况6: 更新父记录的主键（Branch）

如果父记录的主键值被更新了，若有子记录引用此主键值，则失去了参照完整性。也就是说，如果当前被更新的分公司有员工工作，更新后，员工工作的分公司与实际的分公司将不一致。为了确保参照完整性，可使用上面已介绍的策略。在CASCADE情况中，更新父记录的主键的同时更新子记录，也就是处于级联的状态下。

图10-12为StayHome的Branch视图创建的表的参照完整性约束。

6. 其他业务规则

最后，考虑被称为业务规则的约束。更新实体的时候，可能会被现实世界的事务要求所约束。例如，StayHome有这样的业务规则，防止每个成员一次借超过10盘的录像。

7. 将所有完整性约束存档

在数据字典中将所有的完整性约束存档，以作为物理数据库设计阶段的参考。

10.1.5 步骤2.5: 与用户讨论逻辑数据库设计

目标 确保局部逻辑数据模型与描述模型的文档确实表达了用户视图。

此视图的逻辑数据库设计应该已经设计完全，并且全部存档。但在完成这个步骤之前，应该与用户一起研究这个设计。

如果设计只有一个用户视图的数据库，或使用集中化方法并合并了多个用户视图，那么就可以开始物理数据库的设计了。物理数据库设计在第12章到第16章描述。但是，如果是设计有多个用户视图的复杂的数据库，并且使用视图集成方法来管理这些用户视图，则在进行物理数据库设计之前应先阅读附录C。

10.2 本章小结

- 逻辑数据库设计步骤2的主要目的是为ER图创建表并检查表的结构。
- 使用规范化方法检查表的结构。
- 检查表结构以确保它们支持用户所要求的事务。
- 业务规则可以防止数据库不完整、不准确和不一致。这些规则包括：完整性约束、需要的数据、列值约束、实体完整性、多样性、参照完整性和其他的业务规则。
- 已有约束通过定义条件确保了参照完整性，可以根据这些条件插入、更新和删除主键或外键。
- 当一个子记录引用了要删除/更新的父记录时，可以考虑使用如下几种策略：NO ACTION、CASCADE、SET NULL、SET DEFAULT和NO CHECK。

复习题

10.1 描述逻辑数据库设计方法学步骤2的主要目的和任务。

10.2 描述下述情况创建表的规则:

(a) 强实体和弱实体

(b) 一对多（1:*）二元关系

(c) 一对多（1:*）递归关系

(d) 一对一（1:1）二元关系
(e) 一对一（1:1）递归关系
(f) 多对多（*:*）二元关系
(g) 复杂关系
(h) 多值属性

给出能够说明你的答案的例子。

10.3 讨论如何应用规范化技术检查从ER图创建的表的结构以及支持的文档。

10.4 讨论可以用于检查表是否支持用户需要的事务的一个方法。

10.5 讨论业务规则的含义，给出能够说明你的答案的例子。

10.6 如果有一个子记录引用了一个我们将要删除的父记录，描述一个可以应用的备用策略。

练习

10.7 为附录E中给出的每个ER图创建表的描述，在创建时首先不要看答案中的表描述。比较你的表和答案中的表，并说明有何不同。

第11章 高级建模技术

本章主题:

- 基本ER建模概念的限制和使用增强的数据建模概念创建更复杂应用的需求。
- 与增强的实体－关系模型（EER）相关的主要概念叫做特化/泛化。
- 在EER模型中显示特化/泛化的图表技术。
- 在EER模型中怎样创建代表特化/泛化的表。

在第7章中，我们介绍了与实体－关系（ER）建模相关的基本概念，并在第9章和第10章介绍的逻辑数据库设计方法学中使用了这些概念来构建ER模型。这些基本概念对于表达传统的基于管理的数据库应用的数据模型通常是够用的。然而，对于更复杂的数据库应用，这些基本的ER概念就有限了。这就需要开发具有附加“语义”的建模概念。具有附加语义概念的ER模型就是增强的实体-关系模型（Enhanced Entity-Relationship，EER）。在本章中，我们介绍了与EER模型有关的最有用的概念之一，叫做特化/泛化（Specialization/Generalization），并且说明了如何使用它（方法学概要见附录B）。

本书中所介绍的数据库设计方法学在步骤1.6中提供了使用EER模型的附加概念的选择。是否使用这个步骤，依赖于用户需求（或部分用户需求）的复杂性，同时，使用附加的建模概念有帮于数据库设计过程。

11.1 特化/泛化

特化/泛化的概念是与称为超类和子类的特定类型的实体以及属性继承（Attribute Inheritance）的过程有关的。本节我们以定义超类和子类并检查超类/子类的关系开始。我们描述了属性继承过程并比较特化和泛化过程。我们也介绍怎样使用UML(统一建模语言)符号表达特化/泛化。

11.1.1 超类和子类

超类（Superclass） 是一个实体，包含所有在实体中出现的公共属性和关系。

子类（Subclass） 是一个实体，有一个区分的角色，并且包含在（超类）实体中出现的部分具体属性和关系。

通常一个称为超类的实体包括很多更具体的称为子类的实体。例如，Staff就是一个具有许多不同子类的实体。Staff实体的成员的实体可以被划分为Manager（经理）、Secretary（秘书）和SalesPersonnel（销售人员）。换句话说，Staff实体是子类Manager、Secretary和SalesPersonnel的超类。

11.1.2 超类/子类关系

在一个超类和它的任一个子类之间的关系均是1:1关系（1:1关系在5.5.1节定义），被称为超类/子类关系。例如，Staff/Manager构成了一个超类/子类关系。子类的每个成员也是超类的成员，但是有一个唯一的角色。

我们可以使用超类和子类来避免在一个实体中用不同的属性来描述实体。例如，SalesPersonnel可能有特殊属性，如salesArea和carAllowance，等等。如果所有的Staff的公用属性和那些某个特殊工作要求的具体属性都由实体Staff代表，那么对于某些特殊工作的属性就会产生许多空值。很明显，销售人员有和其他员工共同的属性，包括staffNo，name，position和salary，但也有不共享的属性。如果我们要在一个实体中描述员工的所有成员，这些不公用的属性就会产生问题。定义超类/子类能让我们描述只和Staff具体子类相关，但并不与Staff全部相关的关系。例如，销售人员有特殊的关系，这个关系并不是对所有员工都是适用的，如SalesPersonnel Requires Car(销售人员需要汽车)。

为了说明上面所提到的问题，让我们考虑图11-1中叫做AllStaff的表。这张表包含职员中所有成员的详细信息，不管他们是什么职位。将所有员工的详细信息放在一张表中的结果时，当填入适合于所有员工的列（即staffNo、name、position、salary和branchNo）时，那些只对某些工作角色可用的列只被填入了部分值。例如，与SalesPersonnel子类相关的列（即salesArea、vehLicenseNo和carAllowance）对不在该子类中的员工的其他成员都没有值。

对所有职员均有效的列　　对销售人员均有效的列

AllStaff

staffNo	name	position	salary	branchNo	salesArea	vehLicenseNo	carAllowance
S1500	Tom Daniels	Manager	46000	B001			
S0003	Sally Adams	Assistant	30000	B001			
S0010	Mary Martinez	Manager	50000	B002			
S0099	Joe Hope	Sales Personnel	35000	B002	WA 1A	SH22	5000
S3250	Robert Chin	Supervisor	32000	B002			
S2250	Sally Stern	Manager	48000	B004			
S2345	Linda Haven	Sales Personnel	37500	B002	WA 2B	SH34	5000
S0415	Art Peters	Manager	41000	B003			

图11-1 包含员工中所有成员的详细信息的AllStaff表

> 将超类和子类引入ER模型有两个重要的原因。第一个原因是，避免多次描述相近的概念，因此可以节省时间，并使ER模型的可读性更强。第二个原因是，在设计中增加了更多人们较熟悉的语义信息。例如，断言Manager IS-A member of staff（经理是员工中的一个成员）和van ISA type of vehicle（运货车是车辆的一种。）将重要的语义内容以一种简单易懂的形式表达出来。

11.1.3 属性继承

正如上面所提到的，子类中的实体代表超类中的相同的“现实世界”对象。因此，子类的一个成员继承了与超类有关的属性，但也可以有此子类的特定属性。例如，SalesPersonnel子类的成员有子类特定的属性，即salesArea、vehLicenseNo和carAllowance，以及Staff超类的所有属性，即staffNo、name、position、salary和branchNo。

子类是有自己的权限的实体，它可以有一个或多个子类。有多个超类的子类称为共享子类（shared subclass）。换句话说，共享子类的成员一定是相关超类的成员。其结果是，超类

的属性由共享的子类继承，而共享子类也可能有自己的附加属性。这个过程也就是多重继承。

> 实体以及它的子类和它的子类的子类等等，被称为类型层次（type hierarchy）。类型层次有不同的名称，包括特化层次(specialization hierarchy)、泛化层次（generalization hierarchy）和IS-A层次（例如Manager IS-A（member of）Staff）。例如，Manager是Staff的特化，Staff是Manager的泛化。在接下来的内容中，我们描述特化和泛化的过程。

11.1.4 特化过程

> **特化**（Specialization）通过标识用来区分实体间成员的特征来最大化实体间成员的差别的过程。

特化是一个自顶向下的方法，它定义超类集合以及有关的子类。这些子类集合的定义基于超类中实体的一些特别特征。当我们标识实体的一个子类时，我们把特殊的属性与子类相关联（在必要时），并且标识出子类和其他实体或子类（必要时）之间的关系。

11.1.5 泛化过程

> **泛化**（Generalization）通过标识实体间的公共特征来最小化实体间差别的过程。

泛化过程是一个自底向上的方法，它从初始的子类中产生泛化的超类。泛化过程可以被看成是特化过程的逆过程。例如，考虑下面的模型，其中Manager、Secretary和SalesPersonnel代表不同的实体。如果在这些实体上应用泛化过程，我们要标识出它们之间的相似之处，如公共的属性和关系。正如前面所说的，这些实体分享对所有员工来说公共的属性，因此我们将Manager、Secretary和SalesPersonnel标识为泛化超类Staff的子类。

图表描述

UML对子类和超类有特定的符号。例如，将Staff实体的特化/泛化看成代表工作角色的子类。Staff超类和Manager、Secretary和SalesPersonnel子类可以用图11-2所示的EER模型来图解的描述。注意Staff超类和子类，作为实体，它们以长方形表示。特化/泛化子类由带箭头的线关系，箭头指向超类。箭头下面的标签{Optional, And}（{可选，与}），描述了特化/泛化关系的约束。这些约束在接下来的内容中将更详细地介绍。

给定子类的特定属性列在代表子类的方框的下部。例如，salesArea、vehLicenseNo和carAllowance属性只与子类SalesPersonnel相关，并且对于子类Manager或Secretary是无效的。类似的情况下，对子类Manager，属性bonus（红利）是特定的；而对子类Secretary，属性typingSpeed（打字速度）是特定的。

图11-2也说明了适用于特定子类或仅适用于超类的关系。例如，子类Manager通过Manages关系与Branch实体关联，而Staff实体通过Has关系与Branch实体关联。

> **注意** Manages关系中Manager的多样性是1:1，而以前Manages关系中的Staff的多样性是0:1（换句话说，Manager具有强制参与，而Staff为可选参与）。

在图11-3中，Staff特化/泛化被扩展为一个共享的叫做SalesManager的子类和一个叫做Secretary的超类，而Secretary又有自己的叫做AssistantSecretary的子类。换句话说，共享子类

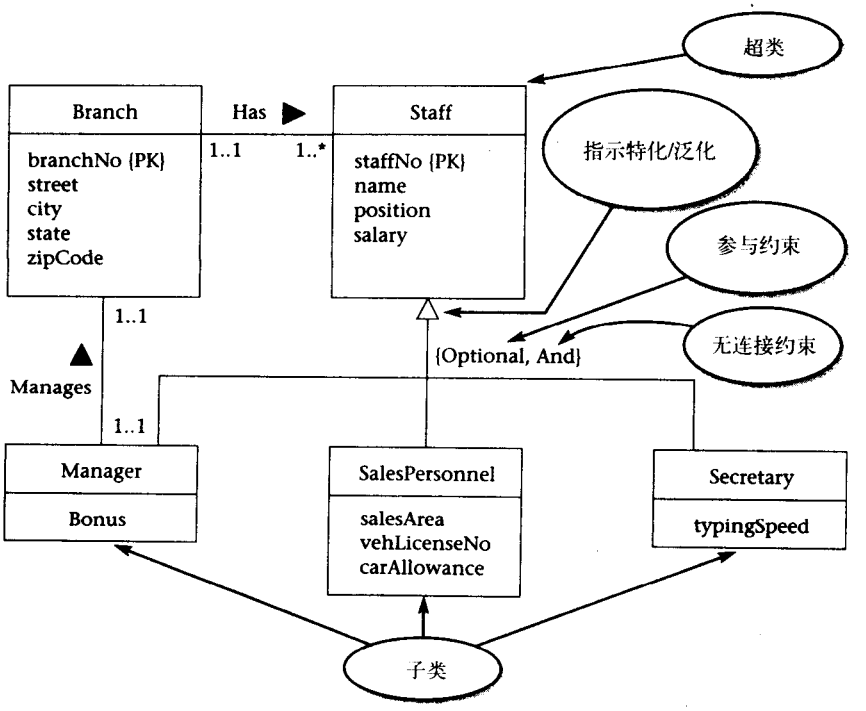


图11-2 通过特化/泛化将Staff实体转换为代表工作角色的子类

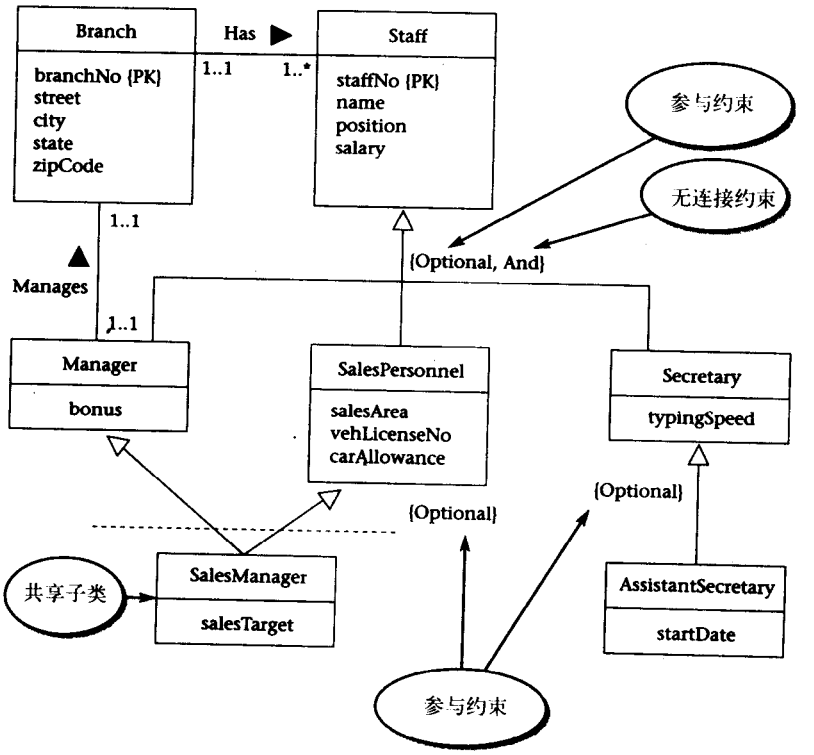


图11-3 Staff实体的特化/泛化，它包括SalesManager共享子类和AssistantSecretary子类，此子类有自己的Secretary子类

SalesManager的成员必须是SalesPersonnel和Manager子类以及Staff超类的成员。结果是，Staff超类的属性（staffNo、name、position、salary）、SalesPersonnel子类的属性（salesArea、vehLicenseNo、carAllowance）和Manager(bonus)被子类SalesManager继承，而SalesManager也有其自身的属性，称为salesTarget(销售目标)。

AssistantSecretary是Secretary的子类，而Secretary又是Staff的子类。这意味着AssistantSecretary的成员必须是Secretary子类和Staff超类的成员。结果，AssistantSecretary子类继承了Staff超类（staffNo, name, position, salary）的属性和Secretary子类（typingSpeed）的属性，同时它还可以有自己的属性startDate。

11.1.6 超类/子类关系的约束

在超类/子类关系可以使用两类约束，分别为参与（participation）约束和无连接（disjoint）约束。

1. 参与约束

参与约束（Participation Constraint） 决定在超类中的每次出现是否必须作为子类的一个成员。

参与约束可以是强制的也可以是可选的。有强制参与的超类/子类关系指明超类中出现的每个实体必须也是子类的成员。为了表达强制参与，Mandatory（强制）放置在指向超类的三角下的花括号中。例如，图11-4中的Vehicle（车辆）特化/泛化（Van、Bus和Car）(货运车、公共汽车和小轿车)有强制参与，这意味着每辆车都必须是货运车、公共汽车或小轿车。

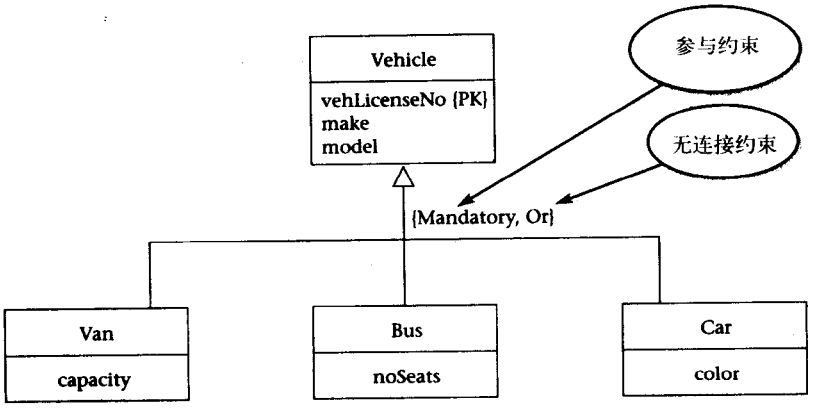


图11-4 将实体Vehicle特化/泛化为vehicle（车辆）类型

可选参与的超类/子类关系明确指明超类的成员不需要属于任何子类。为了表达可选参与，Optional（可选）放置在指向超类的三角下的花括号中。例如，在图11-2中，工作角色特化/泛化为可选参与，这意味着员工的成员不需有附加的工作角色，如Manager、Secretary或者SalesPersonnel（经理、秘书或者销售人员）。

2. 无连接约束

无连接约束（Disjoint Constraint） 描述子类成员之间的关系，并且表明超类的成员是否可能是一个或多个子类的成员。

只有当超类有多个子类的时候，才应用无连接约束。如果子类是无连接的，那么实体只

能是一个子类的成员。为了表示无连接超类/子类关系，Or(或)放置在花括号中参与约束的旁边。例如，图11-4中，Vehicle的特化/泛化（Van、Bus和Car）是无连接的，这意味着车辆是货运车、公共汽车或小轿车。

如果一个特化/泛化的子类不是无连接的（称为非无连接），那么，实体的出现可能是多个子类的成员。为了表达非无连接超类/子类关系，And(与)放置在花括号中参与约束的旁边。例如，图11-2（和图11-3中），工作角色的特化/泛化的子类（Manager、Secretary、SalesPersonnel）是非无连接的，这意味着实体的出现既可以是Manager子类的成员，又可以是SalesPersonnel子类的成员。共享子类SalesManager的出现也证实了这一点。

特化/泛化的参与和无连接约束分为四类：强制的和非无连接的、可选的和非无连接的、强制的和无连接的、可选的和无连接的。

11.2 创建表达特化/泛化的表

在第10章，我们描述了如何使用ER模型基本概念从数据模型创建表。在这一节中，我们说明如何为特化/泛化集合创建表。我们在图11-2和11-4中说明这个EER模型的过程。同从前一样，我们用关系数据库的数据库定义语言（DBDL）来描述每个表。

对EER模型中的每个超类/子类关系，标识超类为父实体，子类为子实体。关于如何将关系最好地描述为一个或多个表，有不同的选择。最正确的选择依赖于表11-1所示的超类/子类关系的参与和无连接约束。

表11-1 表达基于参与和无连接约束的超类/子类关系的选择

部分约束	无连接约束	必需的表
强制	非无连接{与}	单表
可选	非无连接{与}	两张表：一张表用于超类 一张表用于所有子类
强制	无连接{或}	许多表：一张表用于每个连接的超类/子类
可选	无连接{或}	许多表：一张表用于超类 其他表对应每个子类

我们用图11-2的Staff特化/泛化作为我们的第一个例子。Staff超类和它的子类（Manager、SalesPersonnel或Secretary）的关系是可选的，作为Staff的成员可以不属于任何一个子类（是非无连接的），也可以属于多个子类。基于表11-1中所给出的选择，应该为超类建一张表，为所有的子类建一张表，来代表Staff超类/子类关系，如图11-5所示。为了清楚起见，我们也采用一张表来代表Branch实体和它与Staff的关系。

我们使用图11-4中的Vehicle特化/泛化作为我们的第二个例子。Vehicle超类和它的子类（Van、Bus或Car）的关系是强制的，Vehicle超类的所有成员必须属于子类中的一个（是无连接的），而作为Vehicle超类的一个成员则可以只属于一个子类。基于表11-1中所给出的选项，应该为每个连接的超类/子类建一张表来代表Vehicle超类/子类关系，如图11-6中所示。

尽管表11-1中的选项对如何表达超类/子类关系提供了一些较好的指导，但还有其他的因素可能影响最终的选择：

- 子类是否包含在不同的关系中。
- 每个子类不同属性的数目。
- 由超类和每个子类代表的实体出现的相对数目。

Staff 超类
staff (staffNo, name, position, salary, branchNo)
Primary Key staffNo
Foreign Key branchNo **references** Branch(branchNo)

Staff 子类
AllStaffSubclasses (subclassStaffNo, bonus, salesArea, vehLicenseNo, carAllowance, typingSpeed)
Primary Key subclassStaffNo
Foreign Key subclassStaffNo **references** Staff(staffNo)

Branch
Branch (branchNo, street, city, state, zipCode, mgrStaffNo)
Primary Key branchNo
Foreign Key mgrStaffNo **references** AllStaffSubclasses(subclassStaffNo)

图11-5　代表 Staff 特化/泛化和图11-2所示的Branch实体的表

Van子类
Van (vehLicenseNo, make, model, capacity)
Primary Key vehLicenseNo

Bus子类
Bus (vehLicenseNo, make, model, noSeats)
Primary Key vehLicenseNo

Car子类
Car (vehLicenseNo, make, model, color)
Primary Key vehLicenseNo

图11-6　代表图11-4所示的 Vehicle 特化/泛化的表

11.3　本章小结

- 超类是一个实体，包含在实体中出现的所有公共的属性和关系。
- 子类是一个实体，有唯一的角色，并且包含在（子类）实体中出现的某些属性和关系。
- 属性继承是这样一个过程，在这种情形中，子类的成员可以处理子类特有的属性，并且继承这些与超类有关的属性。
- 特化是一个过程，它通过标识实体成员间的不同特征来最大化它们的差别。
- 泛化是一个过程，它通过标识实体的共有特征来最小化它们的差别。
- 在超类/子类关系上可以使用两类约束，分别为参与约束和无连接约束。
- 参与约束决定在超类中的每个出现是否必须作为子类的一个成员。
- 无连接约束描述子类成员之间的关系，并且表明超类的成员是否可以是一个或多个子类的成员。

复习题

11.1　描述超类和子类代表什么？

11.2　描述超类和子类间的关系。

11.3　描述并用一个例子说明属性的继承过程。

11.4　将超类和子类的概念引入EER模型的主要原因是什么？

11.5　描述共享的子类代表什么？

11.6 描述并对比特化和泛化的过程。

11.7 描述应用在特化和泛化关系上的两个主要约束。

练习

11.8 检查特化/泛化是如何在附录E中描述的某些案例研究中应用的。

11.9 考虑对于练习7.12中描述的案例研究，在ER模型中引入增强的特化/泛化概念是否合适。如果合适，用增强的概念重画这个ER图为ERR图。

第四部分 物理数据库设计

第12章 物理数据库设计——步骤3
第13章 物理数据库设计——步骤4
第14章 物理数据库设计——步骤5和步骤6
第15章 物理数据库设计——步骤7
第16章 物理数据库设计——步骤8

第12章 物理数据库设计——步骤3

本章主题：

- 物理数据库设计的目的。
- 如何把逻辑数据库设计映射为物理数据库设计。
- 如何为目标DBMS设计基本表。
- 如何设计派生数据的表示。
- 如何为目标DBMS设计业务规则。

在本章和接下来的几章中，我们通过一个例子描述并说明关系数据库的物理数据库设计方法。本章从逻辑数据模型以及描述方法学中步骤1~3（步骤1~3在第9章~第10章介绍）所建模型的文档开始。方法学开始于步骤1所创建的逻辑数据模型，接下来在步骤2中使用这些逻辑模型派生一组表。检查逻辑模型和派生的表以确保它们是使用规范化技术正确创建的，并保证它们支持用户所需的事务。

在数据库设计方法学的第三个阶段，也就是物理数据库设计阶段，必须确定如何将逻辑数据库结构（也就是实体、属性、关系和约束）转换为目标DBMS可以实现的物理数据库设计。因为物理数据库设计的许多方面都高度地依赖于目标DBMS，你可能会发现不只有一种方法可以实现数据库中指定的部分。因此为了正确地完成这项工作，需要具有对目标DBMS功能的全面的认识，并且需要理解特定实现细节的每个可供选择方案的优点和缺点。对某些系统来说，考虑到数据库的使用目的，可能还需要选择合适的存储策略。PC RDBMS，例如Microsoft Access，通常有固定的存储结构，因此，如果使用这样的系统，就不必考虑这一步。

在本章中，我们介绍如何将从逻辑数据模型派生来的表转化成具体的数据库实现。在第13章中，我们将介绍为基本表选择文件组织方式以及确定何时创建索引的方针。在第14章，我们将讨论在创建用户视图过程中确保数据库安全的方法以及合适的数据库安全机制。在第15章中，我们给出当反规范化物理数据模型并引入冗余以提高性能的一些指导。最后在第16章中，我们将讨论监视并调整操作系统的过程。

尽管在后续的章节中，我们将介绍物理实现细节来使讨论更清楚，为了说明各DBMS之间的差别，我们将使用Microsoft Access来演示StayHome示例的实现细节。为了对照，在第18章、第19章用到的第二个例子将使用Oracle DBMS。

在介绍物理数据库设计方法学之前，我们先简要地介绍一下设计过程。

12.1 逻辑与物理数据库设计的比较

逻辑数据库设计极大地独立于实现细节，例如目标DBMS的具体功能、应用程序、编程语言或者任何其他的物理考虑。逻辑数据库设计的输出是一个逻辑数据模型，此模型包括描述此模型的文档和一组关系表，例如数据字典。同时，这些也代表着物理设计过程的使用的信息源，并且它们提供了要进行有效的数据库设计非常重要的依据。

逻辑数据库设计关心的是“什么”，而物理数据库设计关心的是“怎么”。特别是作为物理数据库设计者，必须知道支持DBMS的计算机系统怎样运行，并且必须要有对目标DBMS功能的整体认识。因为当前系统所提供的功能变化范围很大，因此物理设计必须依赖于具体的DBMS系统。但是，物理数据库设计并不是独立的行为——在物理、逻辑和应用设计之间经常是有反复的。例如，在物理设计期间为了改善系统性能而进行的决策，如合并表，可能影响逻辑数据模型（应用设计的讨论见4.9节）。

12.2 物理数据库设计方法学概述

物理数据库设计（Physical Database Design） *产生辅存上的数据库实现的描述的过程，它描述了基本表、文件组织方式和用于实现数据有效访问的索引以及任何相关的完整性约束和安全限制。*

物理数据库设计的步骤如图12-1所示。我们将物理数据库设计方法分为6个主要步骤，接着逻辑数据库设计方法的两个步骤，从步骤3开始，右边的列列出了讨论这个步骤的章。

	章
步骤3：为目标DBMS转换全局逻辑数据模型	12
步骤3.1：设计基本表	
步骤3.2：设计派生数据的表示	
步骤3.3：设计其他业务规则	
步骤4：选择文件组织方式和索引	13
步骤4.1：分析事务	
步骤4.2：选择文件组织方式	
步骤4.3：选择索引	
步骤5：设计用户视图	14
步骤6：设计安全性机制	14
步骤7：引入受控冗余的考虑	15
步骤8：监视并调整操作系统	16

图12-1 物理数据库设计方法学中的步骤

物理数据库设计的步骤3包括使用从目标DBMS可获得的功能设计基本表和完整性约束。

这步也考虑如何表达模型中的派生数据和有关的业务规则。

步骤4包括分析必须支持的事务，并基于这个分析为基本表选择文件组织方式和索引。典型情况下，PC DBMS有固定的存储结构，但其他的DBMS会为数据提供一些可选择的文件组织方式。从用户的观点来看，对表的内部存储的描述应该是不可见的——用户不用指明记录存储在哪里、怎么存储，就可以访问表和记录。作为物理数据库设计者，必须提供DBMS和操作系统的物理设计细节。对于DBMS，必须指明用于代表每个表的文件的组织方式；对于操作系统，你必须指明像每个文件的位置和安全措施这样的细节。

步骤5包含确定应该如何实现每个用户视图。步骤6包括设计安全性措施以保证数据不会被未授权的用户访问，包括在基本表上需要的访问控制。

步骤7考虑放宽逻辑数据模型提出的规范化约束以改善整个系统的性能。这个步骤应该仅在需要的时候考虑，因为在维护数据一致性时还要考虑引入冗余所带来的问题。步骤8是一个监视和调整正操作的系统的持续的过程，用于标识和解决由数据库设计产生的所有的性能问题，并实现新的需求或改变需求。

附录B为那些已经熟悉数据库设计方法学的用户总结了方法学的步骤，并且简单地对主要步骤进行了回顾。在本章余下的部分中，我们说明了数据库设计方法学的步骤3。在本章和接下来的几章中，我们通过描述如何实现可选设计来说明物理数据库设计和实现之间的紧密关系。

12.3 步骤3：为目标DBMS转换全局逻辑数据模型

目标 从逻辑数据模型产生基本的工作关系数据库。

物理数据库设计第一步包括将从逻辑数据模型产生的表转换为在目标关系DBMS中可以实现的形式。这步的第一部分要比较在逻辑数据库设计阶段收集的信息和在数据字典中存档的信息。这步的第二部分要使用这些信息进行基本表的设计。这个过程需要具有非常了解目标DBMS所提供的有关功能的知识。例如，应该知道：

- 怎样创建基本表。
- 系统是否支持主键、外键和备用键的定义。
- 系统是否支持所需数据的定义，即系统是否允许某列被定义为NOT NULL。
- 系统是否支持域定义。
- 系统是否支持关系完整性规则。
- 系统是否支持业务规则定义。

步骤3中的三个任务是：

- 步骤3.1：设计基本表。
- 步骤3.2：设计派生数据的表示。
- 步骤3.3：设计其他业务规则。

12.3.1 步骤3.1：设计基本表

目的 确定如何在目标DBMS中描述在逻辑数据模型标识的基本表。

要开始物理设计过程，首先要比较和吸收在逻辑数据库设计阶段创建的表的信息。这些

必要的信息可以从数据字典中获得，并且使用数据库设计语言（DBDL）来定义表。对每个在逻辑数据模型中标识的表，应该有如下的定义：

- 表名。
- 括在括号内的简单列名表。
- 主键以及在适当的地方的备用键（AK）和外键（FK）。
- 任何标识出的外键的参照完整性约束。

应该对每个列有如下定义：

- 它的域，包括数据类型、长度和域上的约束。
- 每个列设置可选的默认值。
- 该列是否可以为空。
- 该列是否是派生列，如果是，怎样计算。

为了描述基本表设计，我们使用DBDL的扩展形式来定义域、默认值和空指示。例如，对于图10-8所定义的StayHome数据库应用的Branch表，可以创建如图12-2所示的设计。

```
Branch_Numbers域                    长度为4的固定长度的字符串
Street_Names域                      最大长度为30的可变长度的字符串
City_Names域                        最大长度为20的可变长度的字符串
State_Codes域                       长度为2的固定长度的字符串
Zip_Codes域                         长度为5的固定长度的字符串
Staff_Numbers域                     长度为5的固定长度的字符串
branch(     branchNo      Branch_Numbers      NOT NULL,
            street        Street_Names        NOT NULL,
            city          City_Names          NOT NULL,
            state         State_Names         NOT NULL,
            zipCode       Zip_Codes           NOT NULL,
            mgrStaffNo    Staff_Numbers       NOT NULL)
Primary Key branchNo
Alternate Key zipCode
Foreign Key mgrStaffNo References Staff (staffNo) ON UPDATE CASCADE ON DELETE NO ACTION
```

图12-2　使用扩展DBDL的Branch表的物理设计

1. 在Access 2002中实现基本表

接下来就是要确定如何实现基本表。正如我们已经说过的，这个决定依赖于目标DBMS，有些系统比其他系统提供了更多的定义基本表和完整性约束的功能。为了演示这个过程，我们说明了如何在Microsoft Access 2002中创建基本表以及完整性约束。在第18章，我们将看到如何在Oracle 9i中创建表和一致性约束。

> 当讨论Microsoft Access时，我们使用开发商的术语，即用术语“字段”代替“列”。

Microsoft Access提供了五种创建空表的方法：

- 使用Database Wizard（数据库向导）在一个操作中创建整个数据库需要的全部表、窗体和报表。尽管Database Wizard不能用于在已经存在的数据库中添加新的表、窗体和报表，但它可以创建一个新的数据库。
- 使用Table Wizard（表向导）为不同的预定义的表选择字段，这些表包括商业合同、家庭财产清单或医疗记录。

- 直接将数据输入空表（称为数据表单）。当保存新的数据表单时，Access将分析你的数据，并自动为每个字段赋予正确的数据类型和格式。
- 使用在SQL View中的CREATE TABLE语句。
- 使用Design View（设计视图），用图解来指明表的所有细节。

2. 在Microsoft Access中使用SQL创建空表

在3.3.1节中，我们说明了SQL CREATE TABLE语句，用这个语句可以创建基本表。Microsoft Access 2002并没有完全遵从SQL3标准（例如，Access CREATE TABLE语句不支持DEFAULT子句），但正如我们很快将要看到的，默认值和某些业务规则仍可以在SQL之外指定。另外，数据类型也和SQL标准有一点不同，如表12-1所示。图12-3显示了在SQL View中用SQL语句创建Branch表的情况（将这个语句与3.3.1节的等价的SQL语句比较）。

```
CREATE TABLE Branch (branchNo   CHAR(4)       NOT NULL,
                     street     VARCHAR(30)    NOT NULL,
                     city       VARCHAR(20)    NOT NULL,
                     state      CHAR(2)        NOT NULL,
                     zipCode    CHAR(5)        NOT NULL UNIQUE,
                     mgrStaffNo CHAR(5)        NOT NULL,
                     CONSTRAINT pk2 PRIMARY KEY (branchNo),
                     CONSTRAINT fk1 FOREIGN KEY (mgrStaffNo) REFERENCES Staff)
```

图12-3 在SQL View中创建Branch表

表12-1 Microsoft Access数据类型

数据类型	用 法	大 小
Text	文本或文本/数字，不需要计算的数字，如电话号码	最多255个字符
Memo	较长文本和数字，如备注或描述	最多64000个字符
Number	用于进行算术运算的数字数据，不包括货币的计算（用Currency类型）	1，2，4或8字节
Date/Time	日期和时间	8字节
Currency	货币值，使用Currency数据类型防止计算时的舍入	8字节
Autonumber	唯一顺序号（每次加1）或者是随机数字，添加记录时自动插入	4字节
Yes/No	仅包含两个值中的一个的字段，例如Yes/No、True/False、On/Off	1位
OLE Object	在其他程序中使用OLE协议创建的对象（例如Microsoft Word文档、Microsoft Excel电子表格、图片、声音或其他二进制数据），可以链接或嵌入到Microsoft Access表中	最多1GB
Hyperlink	存储超链接的字段	最多64000个字符
Lookup Wizard	创建允许从其他的表或从列表框的列表值中选择值的字段。在数据类型列表中选择这个选项，会启动一个向导来指导你进行定义	典型为4字节

图12-4为创建Branch表的设计视图（Design View）。不论你用什么方法创建表，都可以使用表设计视图来进一步定制你的表，例如添加新的域，设置默认值或者创建输入掩码。

3. 在Access中创建两个表之间的关系

关系是在Relationships窗口中创建的。要创建一个关系，首先显示要创建关系的表，然后将父表的主键属性拖到子表的外键属性处。这时，Access将显示一个窗口允许指定参照完整性约束。

图12-5a显示了当创建1对1（1:1）关系：Staff Manages Branch时所显示的参照完整性对话框，图12-5b为创建完关系后的relationships窗口。

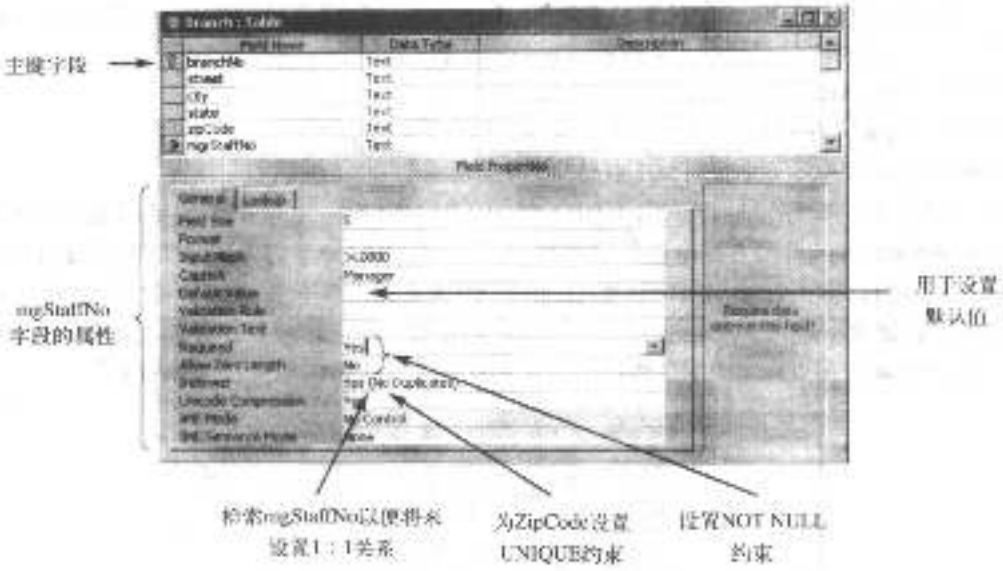
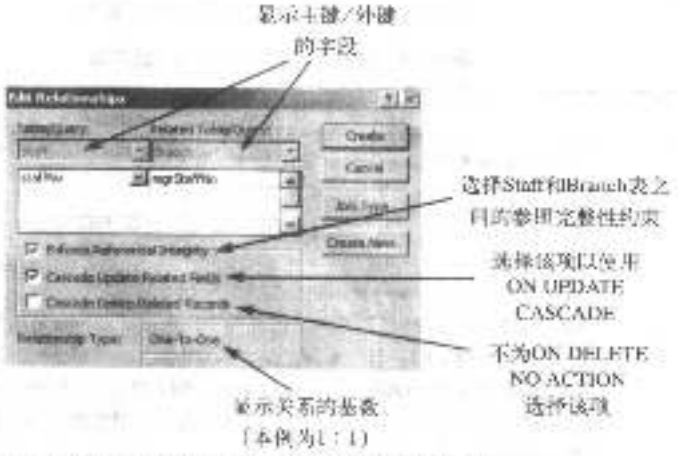


图12-4　创建Branch表的设计视图



a) 为1：1 Staff Manages Branch关系设置参照完整性约束

b) 显示1：1 Staff Manages Branch关系的relationships窗口

图12-5　创建1：1 Staff Manages Branch关系

在Microsoft Access中，设置参照完整性要注意如下两点：

1) 只要相关字段中的一个是主键或者有唯一值索引，那么就创建一对多（1:*）关系；如

果两个相关的字段都是主键或者有唯一值索引，那么就创建1：1关系。因此，为了确保Manages关系是1：1的，不仅要保证Staff表中的staffNo字段被设为主键，而且必须保证Branch表中的mgrStaffNo字段的Indexed属性设置为Yes（没有重复），如图12-4所示。

2) 就更新和删除来说，只有两个参照完整性动作：NO ACTION和CASCADE。因此，如果在步骤2.4定义完整性约束中标识了其他的动作，则必须考虑是否要修改这些约束使它们符合Access中可获得的约束，或者要研究如何用程序代码实现这些约束。你将在第18章看到一个如何实现并不是直接由目标DBMS支持的参照完整性的例子。

4. 基本表的存档设计

基本表的设计应该被完全记录在文档中，这样可以选择设计目标，特别是当有很多选择时，记录下选择其中的一种方法的原因。

12.3.2 步骤3.2：设计派生数据的表示

目标 设计派生数据在数据库中的表示。

如果一个列的值可以从其他列得到，则此列就称为派生列或计算列。例如，下述列都是派生列：

- 在某个分公司工作的员工人数。
- 某个分公司的全部员工的月工资总和。
- 某个会员已经租借的录像的总数。

正如我们在第9章的步骤1.3所说的，派生列经常不出现在ER模型中，但作为文档出现在数据字典中。如果某个派生列显示在ER模型中，则在这个名字的前边用一个“/”来表明这个列是派生的。第一个步骤是检测逻辑数据模型并产生一个包含所有派生列的列表。

从物理数据库设计的角度看，将派生列存储在数据库中还是每当需要时再进行计算，需要进行一下权衡。为了确定哪个更好，应该考虑：

- 存储派生数据的代价以及维护它与派生它数据的一致性的代价。
- 每次计算它需要的代价。

根据性能约束来选择一种代价低的方式。对于上面给出的最后的例子，应该在Member表中存储附加的列来表达每个会员目前租借的录像数目。RentalAgreement表和Member表有新的派生列，如图12-6所示。

这个新派生列的附加的存储并不是特别有意义的，但noOfRentals列需要在每次会员租借和归还录像时都进行修改，应该确保这个修改是一致的以维护数据的正确，并因此保证数据库的一致性。通过按这种方式存储数据，当查询这个信息时，可以立刻获得这个值而且不再需要进行计算。

另一方面，如果noOfRentals列不直接存储在Member表中，那么在每次需要这个值时都必须要计算，这个计算包括连接Member和RentalAgreement表。例如，要计算会员“Don Nelson”目前租借的录像的数量，则应该使用下述SQL查询：

```
SELECT COUNT(*) AS noOfRentals
FROM Member m, RentalAgreement ra
WHERE m.memberNo=ra.memberNo AND m.fName = 'Don' AND
        m.lName = 'Nelson' ;
```

RentalAgreement

rentalNo	dateOut	dateReturn	memberNo	videoNo
R753461	4-Feb-03	6-Feb-03	M284354	245456
R753462	4-Feb-03	6-Feb-03	M284354	243431
R668256	5-Feb-03	7-Feb-03	M115656	199004
R668189	2-Feb-03		M115656	178643

Member

memberNo	fName	lName	address	noOfRentals
M250178	Bob	Adams	57 – 11th Avenue, Seattle, WA 98105	0
M166884	Art	Peters	89 Richmond Rd, Portland, OR 97117	0
M115656	Serena	Parker	22 W. Capital Way, Portland, OR 97201	2
M284354	Don	Nelson	123 Suffolk Lane, Seattle, WA 98117	2

图12-6　有派生列noOfRentals的RentalAgreement表和Member表

如果经常进行这种类型的查询或者从性能的角度看，这是一个关键条件，则将这个派生列存储起来比在每次需要时计算它更合适。在我们的这个例子中，StayHome在会员租借新录像时都要运行这个查询。通过与StayHome员工进行讨论，估计RentalAgreement表大约有400000条记录，因此，由于RentalAgreement表可能比较大而且经常需要这个查询，因此可以决定将这个派生列加到Member表中会效率更高。这个查询现在可以写成：

```
SELECT noOfRentals
FROM Member
WHERE fName = 'Don' AND lName = 'Nelson';
```

提示　当系统的查询语言不能很方便地使用算术运算来计算派生列时，存储派生列也是个很好的方法。例如，SQL有一个有限的聚合函数集，而且不能很容易地处理递归查询。

为派生数据设计文档

表达派生数据的设计应该完全记录在文档中，并且要记下选择某个设计的原因。特别是，当有很多方法可供选择时，尤其要记下选择某个方法的原因。

12.3.3　步骤3.3：设计其他业务规则

目标　为目标DBMS设计其他业务规则。

对表的更改可能会受更改所表达的控制现实世界的事务业务规则的约束。因此，你必须设计域约束以及关系完整性约束。这个步骤的目标是设计应用到数据上的其他的业务规则。这些规则的设计同样要依赖于所选的DBMS。有些系统比其他系统提供了更多的定义业务规则的功能。在上一步骤中，如果系统遵循SQL2标准，某些规则的实现就很容易。例如，StayHome中有这样一条规则，要防止一个会员同时租借10盘以上的录像带，可以把这个规则设计到创建RentalAgreement表的SQL2 Create Table语句中，使用如下的子句：

```
CONSTRAINT member_not_renting_too_many
     CHECK (NOT EXISTS (SELECT memberno
                        FROM rentalagreement
                        GROUP BY memberno
                        HAVING COUNT(*)>=10))
```

而在一些系统中，可以使用触发器来加强约束。就前面的例子来说，在一些系统中，我们可以创建如图12-7所示的触发器来增强完整性约束。当向RentalAgreement表中插入一条记录或者更新已有记录时，激活该触发器。如果某会员目前租借了10盘录像带，则系统显示消息并且终止事务。

注意 不要过于考虑触发器的细节问题。在第18章步骤3.3我们将更详细地讨论触发器。

1. 在Microsoft Access 2002中创建业务规则

在Microsoft Access 2002中创建业务规则有几种方法，例如:

a）确认字段规则。

b）确认记录规则。

c）使用VBA (Visual Basic for Application)确认窗体。

我们用一些简单的例子说明上面的每一点。

```
CREATE TRIGGER member_not_renting_too_many
BEFORE INSERT OR UPDATE ON rentalagreement
FOR EACH ROW
DECLARE
      x  NUMBER;
BEGIN
      SELECT COUNT(*) INTO x
            FROM rentalagreement r
            WHERE r.memberno = :new.memberno;
      IF x  >= 10 THEN
         raise_application_error(-20000,('Member' ||:new.memberno||
                                 'already renting 10 videos');
      END IF;
END;
```

图12-7 每个会员不能一次租借超过10盘的录像带的触发器

(1) 确认字段规则

通过定义字段确认规则，可以确保输入到字段中的数据是正确的。字段确认规则用来检查用户放置到该字段中的数据是否正确。如果输入的值违反了确认规则，将会显示所定义的提示消息。

例如，StayHome有一个简单的约束，也就是出租录像的返回日期不能早于当前日期，尽管日期在开始时可能未被指定。可以在RentalAgreement表的字段级使用函数Date()来实现此约束，此函数返回当前日期如图12-8所示。

(2) 确认记录规则

当保存一条完整的记录的时候，记录确认规则将起作用。与字段确认规则不同，记录确认规则可以引用其他字段。当你想比较表中不同字段的值时，这是很有用的。例如，

StayHome有这样一个约束，录像带的最长租期是5天，尽管日期在开始时可能并未指定。可以使用确认规则在RentalAgreement表的记录级实现该约束：

```
[dateReturn] Is Null OR [dateReturn]<=[dateOut]+5
```

图12-9为具有这个规则集的表的Validation Rule属性对话框。

图12-8　Microsoft Access中的字段确认的例子

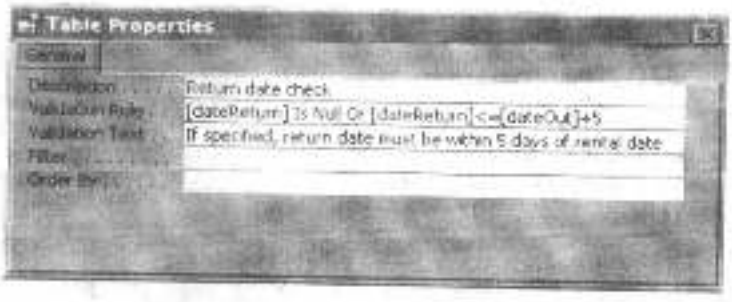


图12-9　Microsoft Access中的记录确认的例子

(3) 使用VBA (Visual Basic for Application)确认窗体

正如我们刚刚提到的，StayHome有这样一条约束，会员不能同时租借超过10盘录像带。这个约束更复杂一些，它要求检查该会员当前租了多少盘录像带。在Access中实现该约束的一种方法是使用事件过程（BeforeUpdate），如图12-10所示。BeforeUpdate事件在更新记录前触发，可以将这个事件与相应的代码相关联。

在一些系统中，可能不支持某些或全部的业务规则，这时在应用程序中设计规则就很必要，正如我们在上一个例子中所看到的，该例子已经用VBA代码实现了约束。当然，在应用代码中实现业务规则存在着潜在的危险，可能会导致加倍的努力，甚至更糟，如果规则不是在应该实现的地方都实现，那么将会导致不一致。

2. 业务规则的存档设计

业务规则的设计应该全部存档，尤其是，当有多种选择存在的时候，将选择其中一种方法的原因存档。

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
Dim MyDB As Database
Dim MySet As Recordset
Dim MyQuery As String

'Set up query to select all records for specified member'
MyQuery = "SELECT rentalno FROM rentalagreement WHERE memberno ='"+ memberNoField +"'"

'Open the database and run the query'
Set MyDB = DBEngine.Workspaces(0).Databases(0)
Set MySet = MyDB.OpenRecordset(MyQuery)

'Check if any records have been returned, then move to the end of the file to allow RecordCount'
'property to be correctly set'
If (NOT MySet.EOF) Then
      MySet.MoveLast
      If (MySet.RecordCount >= 10) Then 'If currently 10 - cannot rent any more'
            MsgBox "Member currently has 10 videos out"
            Me.Undo
      End If
End If

MySet.Close
MyDB.Close
End Sub
```

表单上的字段名

图12-10 检查会员当前没有租借超过10盘录像带的VBA代码

12.4 本章小结

- 物理数据库设计是产生在二级存储中的数据库的实现的描述的过程，它描述了基本表、文件组织方式和用于实现数据有效访问的索引以及任何相关的完整性约束和安全限制。只有当对目标DBMS提供的功能有全面的认识时，才可以着手基本表的设计。
- 在物理数据库设计的初始步骤（步骤3）中，将逻辑模型转换为目标关系DBMS中可以实现的形式，包括设计基本表、表达派生的数据以及业务规则。
- 在接下来的步骤（步骤4）中，分析事务并基于这个分析，设计文件的组织方式和用于存储基本表的索引。
- 数据库代表本质上共同的资源，所以这个资源的安全性是极为重要的。步骤5和步骤6的目标就是设计如何实现在逻辑数据库设计阶段标识的安全措施。这可能包括创建用户视图和访问控制机制的使用，例如SQL所提供的。
- 在步骤7中，考虑引入受控的冗余来提高系统性能。步骤8包括监视和调整操作系统来获得最高的系统性能。

复习题

12.1 解释逻辑数据库设计和物理数据库设计之间的差别，为什么这些任务要由不同的人

完成?

12.2 描述物理数据库设计的输入和输出。

12.3 描述本章介绍的物理设计方法学中的主要步骤。

12.4 描述设计基本表时需要的信息的类型。

12.5 描述如何在数据库中处理派生数据的表达。给出能够说明你的答案的例子。

练习

12.6 按照你的方法选择附录E中给出的一些案例研究，在某个你访问的目标DBMS中，完成本章所讨论的物理数据库设计方法学中的步骤。

第13章 物理数据设计——步骤4

本章主题:

- 如何分析用户事务来确定可能会影响性能的特性。
- 基于对事务的分析，如何选择合适的文件组织方式。
- 何时选择索引来改善系统性能。

本章介绍物理数据库设计方法学的步骤4。在上一步骤中，我们说明了如何将逻辑设计转换为一组基本表，如果需要的话还要转换为业务规则。但是，即使是最简单的数据库，为了获得合适的性能也必须要进行额外的考虑。在本章中，我们考虑物理数据库设计的下一个步骤，这一步骤要考虑与文件组织方式和索引的选择是否正确有关的那些系统性能（方法学总结见附录B）。

提示 如果不熟悉文件组织方式和索引的概念，我们强烈建议在阅读本章之前，先阅读附录D。

与逻辑设计相同，物理设计也必须遵循数据的特性以及用途。尤其是，必须理解数据库要支持的典型工作量。在分析阶段，你也可能发现某些用户会有这样的需求，要求某些事务必须要运行多快，或者每秒必须要处理多少个事务。这种信息构成了在本步骤期间所做的大量决定的基础。

正如我们前面所提到的，要着手物理数据库设计，就必须理解目标DBMS的工作，尤其是文件组织方式、索引和它所支持的查询处理技术。例如，可能有这样的情况，DBMS不使用二级索引，即使允许使用。因此，添加二级索引将不能提高系统的查询性能，而且最终还可能是不正确的。

> 你可能回忆起来在第3章中，SQL和QBE是非过程化数据操纵语言（DML）。这样的语言隐藏了如何在辅助存储（二级）中访问数据的低层细节。这是DBMS的工作，或者更准确些，是DBMS的查询优化器的工作。典型情况下，查询优化器为执行用户请求分析大量不同的策略，并选择其中认为可以提供最佳性能的策略。这种分析是基于数据库统计所估计的数据库操作的消耗的，例如表中记录的数量、每条记录的长度和索引的可用性。
>
> DBMS最终所选择的策略对用户没有影响。事实上，你将发现你可以定义多种存储结构，查询优化器可以利用这些结构选择最优的策略。

13.1 步骤4：选择文件组织方式和索引

目标 确定最佳文件组织方式来存储基本表以及实现所要求性能的索引。

在附录D中，我们为对这些术语不熟悉的读者提供了对文件组织方式和索引的简要介绍。这里复述一下，文件组织方式是指当文件存储在磁盘上时，记录在文件中的排列的方式。索引是一种数据结构，它允许DBMS在文件中更快地定位某些记录，并且因此提高对用户查询

的响应。

可以获得什么样的文件组织方式依赖于目标DBMS，有些系统比其他系统提供了更多的文件组织方式。完全了解可用的结构，并且知道目标系统怎样使用这些结构是很重要的。

只有了解了必须要支持的事务的细节，才能做出有意义的物理设计抉择。在分析事务时，要标识出性能标准，例如:

- 经常运行的事务和对性能产生重大影响的事务。
- 业务操作的关键事务。
- 当对数据库有很高要求时，每日/每周内访问数据库的次数（最大负荷）。

你将使用这些信息来标识可能会引起性能问题的数据库部分。同时，需要标识事务的高层功能，例如在更新事务中更新的列或者查询中检索的列。你将使用这些信息来选择正确的文件组织方式和索引。

作为结果，我们将步骤4分为如下几步:

- 步骤4.1: 分析事务。
- 步骤4.2: 选择文件组织方式。
- 步骤4.3: 选择索引。

13.1.1 步骤4.1: 分析事务

目标 理解运行在数据库上的事务的功能并分析重要的事务。

要进行有效的物理数据库设计，就必须要很好地理解运行在数据库中的事务。

提示 在许多情况中，分析所有预期的事务是极为费时的，因此应该至少研究最重要的那些事务。建议用户查询的最活跃的20%占总的数据访问的80%。当进行分析时，你会发现这个80/20规则是很有用的方针。

为了帮助标识要研究的那些事务，可以使用事务应用图（Transaction Usage Map），它用图形的方式表明了哪些表潜在地可能被多次使用，或者使用事务/表交叉引用矩阵，它显示了每个事务访问的表。为了将目光主要集中在有问题的地方，一种处理方法是:

1）将所有事务路径映射到表中。

2）确定哪些表最常被事务访问。

3）分析选出的包含了这些表的事务。

1. 将所有事务路径映射到表中

在逻辑数据库设计方法学的步骤1.8、步骤2.3和步骤3.2中，将事务路径映射到实体/表，检查模型是否支持用户所需的事务。如果使用了类似图9-17的事务路径图表，将能使用该图表来确定最常被访问的表。另一方面，如果以另一种方式检查事务，将会发现创建一张事务/表的交叉引用矩阵是很有用的。该矩阵显示了所需事务，以及这些事务所访问的表。例如表13-1就是为StayHome的输入、更新/删除和检索（也称为查询）事务的事务/表交叉引用矩阵(StayHome事务列表参见6.4.4节):

（e）输入在一指定分公司登记的新成员的详细信息。

（k）更新/删除指定成员的详细信息。

（p）根据种类排序，列出指定分公司中的所有录像的标题、种类和可用性。

（q）根据标题排序，列出指定分公司中的给定演员姓名的所有录像的标题、种类和可用性。

（r）根据标题排序，列出指定分公司中的给定导演姓名的所有录像的标题、种类和可用性。

（s）列出指定会员当前出租的所有录像的详细信息。

这个矩阵以可视化的方式总结了在数据库中运行的事务的访问模式。例如，矩阵表明了事务（e）读Staff表并且也向Member和Registration表中插入记录。更有用的是，你可以表明每个小格中，在一定时间间隔（例如，每小时、每天、每周）内访问的数量。但是，为了保证矩阵的简单性，我们并不显示出这些信息。该矩阵显示出Video和VideoForRent表被四个查询事务（p、q、r和s）访问。

表13-1 交叉引用的事务和表

事务/表	(e)				(k)				(p)				(q)				(r)				(s)			
	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D
Branch																								
Staff		×																						
Video										×				×				×				×		
VideoForRent										×				×				×				×		
RentalAgreement																						×		
Member	×					×	×	×														×		
Registration	×																							
Actor														×										
Role														×										
Director																		×						

注：I表示插入；R表示读取；U表示更新；D表示删除。

2. 确定频率信息

在对StayHome分公司经理进行讨论的时候，估计大概有20000个录像带标题和40000盘可出租的录像带分布于100多个分公司办公室中，每个分公司平均有4000盘录像带，最多有10000盘录像带。另外，StayHome保存了大约10000个导演和60000个角色中的30000主要演员的数据。图13-1为添加了这些数据的简化的逻辑数据模型。

图13-2为事务（p）、(q)和(r)的事务使用映射图，这些事务都访问了VideoForRent和Video表。由于VideoForRent表较大，因此尽可能有效地访问该表是很重要的。现在可以决定对包括这些表的事务进行更进一步的分析。

在考虑每个事务时，不仅要知道每小时运行的平均次数和最大次数，而且还应该知道事务运行的日期和时间，包括最大负荷可能发生的时间。例如，有些事务可能在绝大多数时间都以平均速率运行，但在星期五早晨的会议前的星期四的14:00到16:00之间，产生最大负荷。其他事务可能只在特定的时间运行，例如星期五/星期六的18:00到21:00，也是它们的峰值。

若事务需要经常访问某些表，那么它们的操作模式就是非常重要的。如果这些事务以相互独立的方式进行操作，那么可能的系统性能问题就会减少。但是，如果操作模式有冲突，则应更仔细地检查事务来确定是否可以有可能通过改变表的结构来改善性能，从而减少潜在的问题，正如我们要在第15章步骤7中所讨论的。

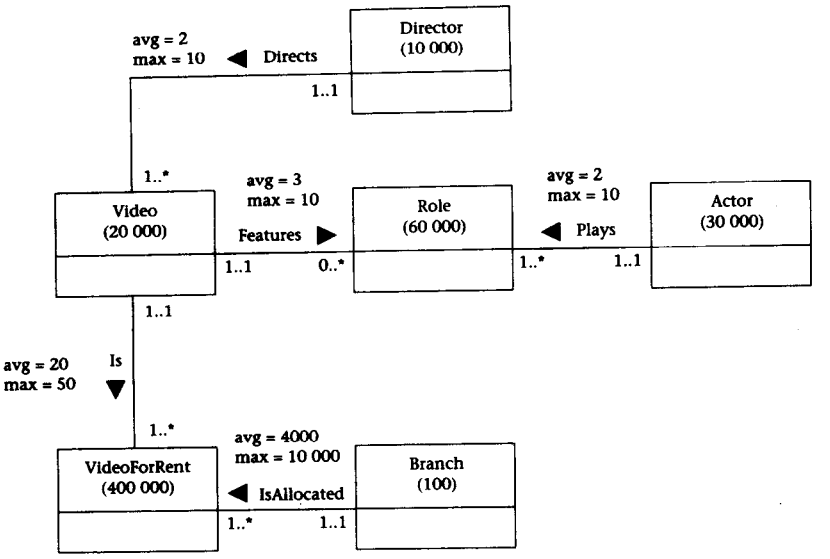


图13-1 显示预期事件的简化逻辑数据模型

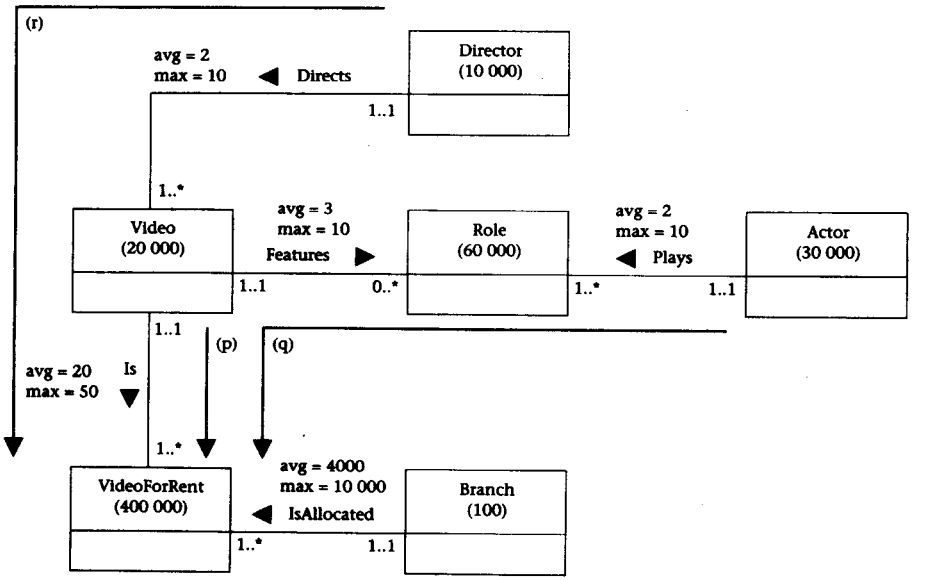


图13-2 示例事务的事务使用映射图

3. 数据应用分析

在标识了重要的事务之后，现在应该开始更详细的分析每个事务。对每个事务而言，应该确定:

(a) 该事务访问的表和列以及访问的类型。也就是，是否插入、更新、删除和检索（也称为查询）事务。

对于更新事务，要注意被更新的列，因为这些列可能是避免访问结构（例如二级索引）的候选。

(b) 在查询条件（在SQL中，这些是WHERE子句中的条件）中使用的列。检查这些条件

是否包括:

- 模式匹配，例如（name LIKE '%Smith%'）。
- 范围查找，例如（salary BETWEEN 30000 AND 40000）。
- 准确匹配的键值检索，例如（salary=30000）。

这不只应用于查询而且也应用于更新和删除事务，这样可以限定表中被更新/删除的记录。

- 这些列可能是访问结构的候选。

（c）对于查询，包含在两个或更多的表的连接中的列。

- 这些列也可能是访问结构的候选。

（d）事务运行的预期频率，例如，某事务每天大约运行50次。

（e）事务的性能目标，例如，事务必须在1秒钟之内完成。

- 对常用和关键事务中的查询条件使用的列，需要比访问结构有更高的优先级。

图13-3显示了事务（p）的一个事务分析模式的例子。这个模式显示了该事务平均的频率是每小时50次，峰值出现在18:00到21:00之间，每小时100次。换句话说，一般情况下，大约一半的分公司每小时运行该事务一次，在顶峰时，所有分公司每小时运行该事务一次。

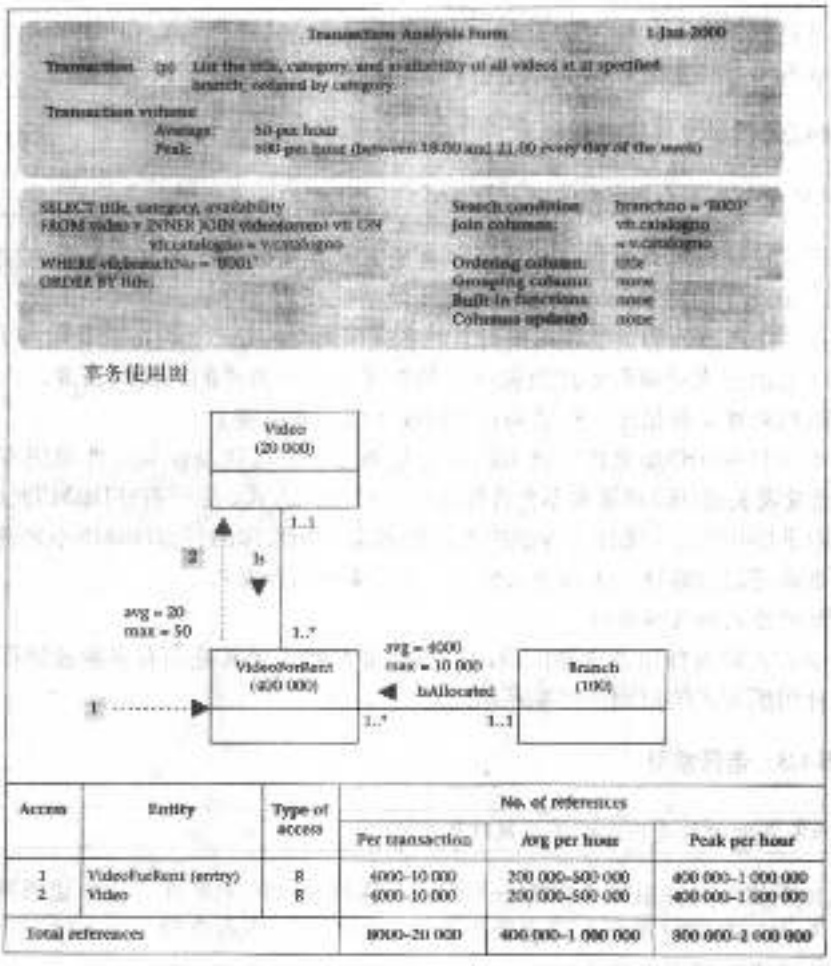


图13-3 样例事务分析模式

该模式也显示了必需的SQL语句和事务使用映射图。在这个阶段，写出全部的SQL语句就过于详细了，但至少应该标识出与SQL语句相连的细节类型，也就是:

- 将要使用的所有查询条件。
- 连接表所需的列（对查询事务来说）。
- 用于排序的列（对查询事务来说）。
- 用于分组的列（对查询事务来说）。
- 可能使用的内置函数（例如AVG、SUM）。
- 被该事务更新的列。

我们将使用这些信息来确定所需的索引，这些很快就要讨论。在事务使用映射图下面，有详细的细目文档:

- 每个表如何被访问（在这里是读）。
- 事务运行时，每次访问多少记录。
- 每小时平均和峰值时访问多少记录。

注意　*对于更新事务而言，对表进行两次访问：一个是读数据，一个是更新数据。*

频率信息将标识需要仔细考虑的表，以确保正确使用访问结构。正如前面所提到的，有时间约束的事务使用的查询条件要比访问结构更优先。

13.1.2　步骤4.2：选择文件组织方式

目标　*确定每个基本表的有效文件组织方式。*

物理数据库设计的主要目标之一就是以有效方式存储数据。例如，如果想按姓名以字母顺序检索员工记录，则用员工姓名对文件排序就是很好的文件组织方式。但是，如果想要检索所有工资在某个范围内的员工，则按员工姓名排序的文件就不是好的文件组织方式。

一些文件组织方式对成批处理数据进入数据库是很有效率的。换句话说，你可能想使用有效的存储结构来建立数据库，然后为通常的操作使用来改变它。

因此，如果目标DBMS允许，这步的目标是为每个表选择最佳的文件组织方式。在许多方面，你可能发现关系DBMS基本不允许你选择文件组织方式，尽管有些DBMS允许创建索引。

我们在附录D中给出了选择文件组织方式的指南。如果你的目标DBMS不允许选择文件的组织方式，则你可以省略这个步骤进入到下一个步骤——步骤4.3。

将文件组织方式的选择存档

文件组织方式的选择以及选择的原因应该全部存档。尤其是在有多种选择存在时，选择其中一种文件组织方式的原因一定要存档。

13.1.3　步骤4.3：选择索引

目标　*确定添加索引是否会改善系统性能。*

为表选择正确的文件组织方式的一种方法是保持记录的无序性并且创建所需数目的二级索引。另一种方法是通过指定主键或聚簇索引使表中记录为有序的。这种情况下，应该选择如下列来排序或者聚簇索引记录。

• 经常用于连接操作的列，因为这样使连接更有效率。

• 在表中经常按某列的顺序访问记录的列。

如果选择排序的列是表的键，那么该索引就是主索引；如果排序的列不是键，那么该索引就是聚簇索引。记住每个文件只能有一个主索引或者是一个聚簇索引，而不能两者兼有。

1. 指定索引

> SQL标准的初始版本有创建和删除索引的语句。但是，这些语句在1992年该标准的第2版中被删除了，因为它们主要被认为是物理概念，而不是逻辑概念。不过，绝大多数主流关系DBMS以这样或那样的形式支持这些语句。我们下面所使用的这些SQL语句就是目前产品所支持的典型语句。

用SQL创建索引，通常使用CREATE INDEX语句。例如，为Video表创建基于catalogNo列的主索引，可以使用如下SQL语句:

```
CREATE UNIQUE INDEX catalogno_index
    ON video(catalogno);
```

如果想为VideoForRent表创建基于catalogNo列的聚簇索引，可以使用如下的SQL语句:

```
CREATE INDEX catalogno_index
    ON videoforrent(catalogno) CLUSTER;
```

正如我们已经提到的，在某些系统中文件的组织是固定的。例如，不久前，Oracle还只支持B^+树，但现在增加了对哈希聚簇的支持。另一方面，INGRES提供了范围更广的不同索引结构，在CREATE INDEX语句中可以使用选项子句进行选择:

```
[STRUCTURE = BTREE | ISAM | HASH | HEAP ];
```

用SQL删除索引，一般可以使用DROP INDEX语句。例如，删除聚簇索引catalogNo_Index，可以使用如下SQL语句:

```
DROP INDEX catalogNoPrimaryIndex;
```

> 注意，Microsoft Access 不支持CREATE INDEX语句，但可以通过字段属性对话框来创建索引。我们在图12-4中可以看到这样一个例子，并详细地讨论了这个问题。

2. 选择二级索引

二级索引提供了为基本表指定其他键的机制，可以用于更有效地检索数据。例如，Member表在会员号memberNo上进行Hash映射，也就是主索引。但是，可能会有经常使用基于lName（名字）列来访问数据，这时，就可以决定增加lName作为二级索引。

但是，在保持和使用二级索引时还要考虑到当检索数据时，必须要平衡改善性能。考虑如下:

• 每当在表中插入一条记录时，都要给每个二级索引增加一个索引记录。

• 当表中相应记录被更新时，也要更新二级索引。

• 需要使用额外的磁盘空间来存储二级索引。

• 在查询优化期间，性能可能退化，因为在选择最佳执行策略之前，查询优化器可能考虑所有的二级索引。

3. 选择索引“意愿表”的方针

确定需要什么样的二级索引的一种方法就是创建你认为是索引的候选列的“意愿表”

(wish-list)，然后考虑维护每个这样的索引的影响。我们提供如下方法帮助你创建“意愿表”:

1) 不必为小表创建索引。在内存中查询该表会比存储额外的索引结构更加有效。

2) 一般情况下，如果没有文件组织方式的键，则索引表的主键。尽管SQL标准提供了指明主键的子句（在上一章的步骤3.1中讨论），但这并不保证在某些RDBMS中主键将被索引。

3) 为检索数据时大量使用的列增加二级索引（例如，为Member表增加基于lName列的二级索引，如前面所述）。

4) 如果经常基于外键访问数据，则为该外键增加二级索引。例如，可能要经常使用branchNo列（分公司号）来连接VideoForRent表和Branch表。则为VideoForRent表创建基于branchNo的二级索引会更有效率。

5) 为经常有如下情况的列添加二级索引:

- 查询或连接条件
- ORDER BY
- GROUP BY
- 其他操作包括排序（例如UNION或DISTINCT）

6) 为在内置函数包含的列增加二级索引，包括内置函数中用于聚簇的列。例如，要知道每个分公司中员工的平均工资，可以使用如下的SQL查询:

```
SELECT branchNo,AVG(salary)
FROM Staff
GROUP BY branchNo;
```

由前面的方针可知，由于有GROUP BY子句，因此可以考虑为branchNo列添加索引。但是，为branchNo列和salary列共同创建索引会更加有效，因为这使得DBMS只根据索引中的数据就可以完成整个查询，而不需要访问数据文件。有时，这被称为仅索引计划(index-only plan)，因为只使用索引中的数据就可以产生所需的响应。

7) 作为比上一种情况更一般的情况，为可以导致仅索引计划的列添加二级索引。

8) 避免为经常被更新的列或表设置索引。

9) 如果查询将检索表中记录的大部分（例如25%），即使表很大，也不创建索引。这时，查询整表要比用索引查询更有效。

10) 避免为由长字符串组成的列创建索引。

提示 如果查询条件包含多个条件，并且条件中包括一个OR子句，而该条件没有索引/排序，那么为其他列添加索引将不会改善查询速度，因为仍然需要表的线性查找。例如，假定只有Video表的category和dailyRental列有索引，并且要使用如下查询:

```
SELECT *
FROM video
WHERE (category = 'Action' OR dailyrental > 3 OR price > 15);
```

尽管这两个索引可以用来查找满足category = 'Action' OR dailyRental > 3 OR price > 15条件的记录，但price列没有索引的事实意味着这些索引并不能用于整个WHERE子句。因此，除非有其他查询可以从category和dailyRental列的索引得到好处，对本查询是不会有索引方面的益处的。

但另一方面，如果WHERE子句中的查询条件是AND，那么category和dailyRental列的两个索引就可以优化查询。

从“意愿表”中删除索引

已经画出了潜在索引的意愿表，现在该考虑在更新事务上每个可能的索引的影响。如果维护索引可能会降低重要的更新事务，那么就考虑从表中删除该索引。但是，要注意，特殊的索引可能也会令更新操作更有效。例如，如果想更新给定员工号码（staffNo）的员工的工资，并且在staffNo列上有索引，那么就可以更快地找到要被更新的记录。

提示 当确定索引是否能改善性能、改善的性能很小或者是相反方向的影响时，最好的方法就是进行试验。在最后一种情况中，很明显应该把该索引从“意愿表”中删除。如果使用索引对性能有微小的改善，那么有必要进一步检查在什么样的情况下索引是有用的，是否这些情况对实现该索引是十分重要的。

对于执行一个特殊的查询或者更新，有时称为查询执行计划（QEP），有些系统允许检查优化器的策略。例如，Oracle有EXPLAIN PLAN诊断实用工具，Microsoft Access有一个性能分析器，DB2有一个EXPLAIN实用工具，INGRES有一个在线QEP-视图工具。当查询比预期的慢时，更应该使用这些工具来确定变慢的原因，从而找到可以改善查询性能的可选策略。

提示 如果大量的记录被插入到有索引的表中，可以先删除索引，再执行插入，然后再重新创建索引会更有效。作为一个不成文的规则，如果插入动作将增加表的10%的大小，那么暂时将索引删除。

4. 更新数据库策略

我们在前面提到过，查询优化器依赖于存储在系统目录中的数据库统计来选择最佳的策略。每当创建索引时，DBMS自动地将此索引增加到系统目录中。但是，你会发现DBMS要求使用一个工具来更新系统目录中与表和索引相关的统计信息（系统目录定义参见1.2.1节）。

5. 将索引的选择存档

索引的选择以及选择的原因应该全部存档。特别是那些出于性能的原因不能创建索引的列，也应该存档。

13.2 使用Microsoft Access 2002的StayHome文件的组织与索引

即使不是全部，也同大部分的PC DBMS一样，Microsoft Access使用了固定的文件组织方式，如果使用Access，则步骤4.2就可以省略。

13.2.1 选择索引指南

Microsoft Access确实支持我们现在正简要讨论的索引。在Access中，表的主键是自动被索引的，但是数据类型为Memo、Hyperlink 或OLE对象的字段是不能被索引的。对其他字段来说，Microsoft 建议满足如下条件时，为该字段创建索引：

- 字段的数据类型是Text、Number、Currency或Date/Time。
- 想要查询存储在该字段中的值。
- 想要对该字段中的值排序。
- 想要存储该字段中的许多不同的值。如果该字段中许多值是相同的，索引可能不会显著提高查询速度。

另外，Microsoft 建议：

- 应该从连接的两边或者是关系两边的字段考虑索引字段，这时，如果还不存在索引的话，Access将会自动创建外键字段上的索引。
- 当用连接字段的值对记录分组的时候，应该为该表中与计算列相同的列指定GROUP BY子句。

Microsoft Access可以优化简单和复杂的查询条件（Access中称为表达式）。就某些特殊类型的复杂表达式来说，Microsoft Access使用称作Rushmore的数据访问技术来实现更高级的优化。复杂表达式是由用AND或OR连接的两个简单表达式组成的，例如:

```
branchno = 'B001' AND available = Yes
category = 'Action' OR dailyrental > 3
```

在Access中，复杂表达式是否全部或部分可优化，依赖于一个或两个简单表达式是否是可优化的，以及是用什么操作符连接他们的。如果下面三个条件全都满足，那么复杂表达式就是可优化的:

- 该表达式使用AND或OR来连接两个条件。
- 两个条件都是由简单的可优化表达式组成的。
- 两个表达式都包含索引字段。这些字段可独立地进行索引或者它们是复合字段索引的一部分。

在Access中创建索引

在Access中，可以在表的Design View（设计视图）中的Field Properties（字段属性）中的Indexed属性中创建索引。索引属性有如下值:

No	无索引（默认）
Yes（可重复）	索引允许重复
Yes（不可重复）	索引不允许重复

我们可以在上一章的图12-4中看到为mgrStaffNo列设置索引的示例。

13.2.2 StayHome的索引

基于上面所提到的方针，你应该确保为每个表创建了主键，这样Access会自动为该列添加索引。另外，应该确保在Relationships窗口中正确地创建了关系，这样Access会自动为外键列添加索引。

表13-2　StayHome的Branch视图的附加索引

表	列	事　务	原　因
Branch	city	(m)	搜索条件
Staff	name	(n)	排序
Video	category	(p)	排序
		(u)	搜索条件
		(v)	分组
	title	(q)、(r)、(u)	排序
		(t)	搜索条件
Actor	actorName	(q)	搜索条件
		(x)	分组、排序
Director	directorName	(r)	搜索条件
Member	fName/lname	(s)	搜索条件
RentalAgreement	dateReturn	(s)	搜索条件
Registration	dateJoined	(y)	搜索条件

从6.4.4节中所列的StayHome的事务中，可以确定创建如图13-2所示的索引。该图显示了每个表中应该被索引的列、使用该列的事务、以及添加该索引（因为该列在查询条件中被作为排序列中的搜索条件，或者被作为分组列）的原因。作为练习，为附录C中给出的StayHome的Business视图中的事务建立索引文档。

注意 在VideoForRent表中，available列被事务用作查询条件。但是，这个列只有两个值（Y或N），因此，由前面所提到的指导方针可知，并不值得为该列建立索引。

13.3 本章小结

- 在步骤4中，你选择最佳文件组织方式来存储基本表和获得所需性能所必需的索引。这包括分析数据库上要运行的事务、选择合适的基于这些分析的文件组织方式和添加索引。
- 在理解要支持的事务的细节之前，是不能做出有意义的物理设计决定的。这包括分析最重要的事务，也就是，最常运行或者业务操作的关键事务。
- 二级索引提供了为基本表说明其他键的机制，这可用于更有效地检索表。但是，在维护和使用二级索引时还要考虑到当检索数据时，必须要平衡改善性能。
- 为表选择合适的文件组织方式的一种方法是保持记录的无序，并且创建所需的二级索引。另一种方法是将表中的记录由指定的主键或者聚簇索引来排序。
- 确定所需要的二级索引的一种方法是创建索引候选列的“意愿表”，然后考虑维护这些索引的影响。

复习题

13.1 描述数据库设计方法学中步骤4的目的。

13.2 讨论分析必须支持的事务的目的并描述应该收集和分析的信息类型。

13.3 什么时候不在表中添加任何索引？

13.4 讨论选择一个列作为候选索引的主要原因。给出能够说明你的答案的例子。

13.5 如果已经标识了某个列作为候选索引，在什么环境下决定对它进行索引？

练习

13.6 按你的方式利用附录E中给出的一些案例研究进行练习，对目标DBMS应用在本章中讨论的物理设计方法学中的步骤。

第14章 物理数据库设计——步骤5和步骤6

本章主题:

- 数据库代表了共同的资源，必须要保证其安全性。
- 如何设计用户视图。
- 怎样设计安全机制来满足用户需求。

本章介绍数据库设计方法学的步骤5和步骤6。在前面的两章中，我们将逻辑设计转换为一组基本表和业务规则，并选择了合适的文件组织方式和基于最重要事务分析的索引。在本章中，我们研究如何设计在数据库应用生命周期的需求分析和收集阶段标识的用户视图以及安全性机制。与物理数据库设计的其他步骤相同，实现用户视图和安全机制也依赖于目标DBMS。

14.1 步骤5：设计用户视图

目标 设计关系数据库应用生命周期分析阶段所标识的用户视图。

数据库设计方法学的第一个步骤包括为数据库分析阶段标识的每个视图或合并后的用户视图产生逻辑数据模型。在6.4.4节，我们为StayHome标识了五个视图，即Manager、Supervisor、Assistant、Director和Buyer。按照这些用户视图的数据需求分析，我们使用集中式方法将用户视图的需求合并为如下两个:

- Branch，包括Manager、Supervisor和Assistant用户视图。
- Business，包括Director和Buyer用户视图。

本步骤的目的是设计前面所标识的所有用户视图。对个人计算机上的独立的DBMS，使用视图通常是很方便的，它用来简化查询。但是，在多用户的DBMS中，视图在定义数据库结构和加强安全性上扮演了重要的角色。在第12章所讨论的基本表设计中，为了说明此过程，我们介绍了创建视图所使用的两种方式:

- 1999 ISO SQL标准（SQL3）
- Microsoft Access 2002

1. 1999 ISO SQL标准（SQL3）

通常，视图使用SQL或类似QBE的工具来创建。例如，对分公司B001的监理和助理，你可能希望创建基本表Staff的视图，这个视图不包含工资信息。创建此视图的SQL语句为:

```
CREATE VIEW Staff1_View
AS SELECT staffNo,name,position
   FROM Staff
   WHERE branchNo = 'B001';
```

这条语句创建了与Staff表具有相同列，但没有salary和branchNo列的视图，此视图的名字为Staff1_View。如果查询此视图，将得到图14-1所示的数据。

Staff1_View

staffNo	name	position
S1500	Tom Daniels	Manager
S0003	Sally Adams	Assistant

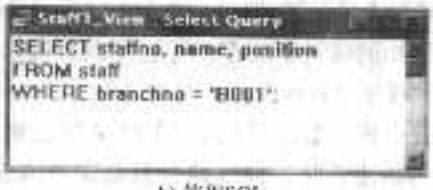
图14-1 Staff1_View视图的列表

为了确保只有分公司的经理可以看到salary列，监理和助理应该不能访问Staff基本表。即，应该给他们授予对Staff1_View视图的访问权限，由此来拒绝他们对敏感的工资数据的访问权限。我们将在步骤7.3进一步讨论访问权限。

2. 在Microsoft Access 2002中创建视图

Microsoft Access不支持SQL CREATE VIEW语句。取而代之，你可以使用QBE或SQL来创建（存储）查询。例如，可以使用QBE查询创建Staff1_View视图，如图14-2a所示。或者使用如图14-2b所示的SQL语句。这个查询现在可以用于创建其他的查询，或者插入/更新/删除基本表Staff中的记录，并且可以作为创建窗体和报表的基础。

a) 使用QBE



b) 使用SQL

图14-2 在Microsoft Access中创建（存储）查询

14.2 步骤6：设计安全性机制

> **目标** 为数据库设计在数据库应用生命周期的需求收集和分析阶段用户指定的安全性机制。

一个数据库代表了重要的公司资源，因此其安全性是非常重要的。在数据库应用生命周期的需求收集和分析阶段可能已经记录了安全需求。本步骤的目的是决定如何实现这些安全性需求。不同的DBMS提供的安全性功能略有差别，因此你必须知道目标DBMS提供的安全机制。正如我们在第5章讨论的，关系DBMS通常提供两种类型的数据库安全：

- 系统安全
- 数据安全

系统安全包括系统级的数据库访问和使用，例如，用户名和口令。数据安全包括数据库对象的访问和使用（例如表和视图）以及用户在这些对象上的可执行操作。

为了说明设计访问规则的过程，我们介绍了两种设计安全性机制使用的两个特定方法:

- 1999 ISO SQL标准（SQL3）
- Microsoft Access 2002

在第18章中，我们将使用一个不同的工作例子，来说明如何在Oracle 9i中设计安全性机制。

1. 1999 ISO SQL标准（SQL3）

提供数据安全性的一种方法是使用SQL的访问控制功能。正如我们刚刚提到的，通常情况下，用户不应该获得直接访问基本表的权限，而是要通过步骤5所设计的用户视图来访问基本表。这提供了很大程度上的数据独立性并避免了用户受数据库结构变化的影响。我们简单地回顾SQL的访问控制机制。对于更详细的信息，有兴趣的读者可参阅Connolly和Begg（2002）的著作。

每个数据库用户都由数据库管理员（DBA）分配一个认证标识，出于明确的安全原因，通常情况下，此标识都有一个相关的口令。DBMS执行的每个SQL语句都代表了一个具体的用户操作。认证标识被用来确定用户可以引用的对象和可以对这些对象执行的操作。SQL中创建的每个对象都有一个拥有者，它是由认证标识标志的。默认情况下，这个对象拥有者是唯一知道该对象的存在并且可以对该对象执行全部操作的人。

权限（Privilege）是允许用户对一给定的基本表或视图可执行的操作。例如，SELECT是从表中检索数据的权限，UPDATE是更新表中记录的权限。当用户使用SQL CREATE TABLE语句创建表时，他（她）自动成为该表的拥有者，并拥有该表的所有权限。其他用户在最初并不拥有这个新创建的表的任何权限。表拥有者必须明确使用SQL GRANT语句授予他们必须的权限，使他们有权访问此表。WITH GRANT OPTION子句说明允许获得权限的用户将该权限授予别人。可以用SQL REVOKE语句收回权限。

当用户用CREATE VIEW语句创建一个视图后，他（她）自动成为此视图的拥有者，但并没有获得此视图的全部权限。要创建视图，用户必须有对组成视图的表的SELECT权限。但拥有者将仅获得他所拥有的视图中的表的其他权限。

例如，允许用户MANAGER检索Staff表中的记录，并且可以对Staff表进行插入、更新和删除的操作，可以使用如下SQL语句:

```
GRANT ALL PRIVILEGES
ON Staff
TO Manager WITH GRANT OPTION;
```

这时，MANAGER将也可以引用此表和它随后所创建的表中的全部列。子句WITH GRANT OPTION说明MANAGER能将这些权限授予其他它认为合适的用户。另一个例子是给认证标识为ADMIN的用户用如下SQL语句授予Staff表的SELECT权限:

```
GRANT SELECT
ON Staff
TO Admin;
```

这条语句省去了子句WITH GRANT OPTION，所以ADMIN不能将该权限再授给其他用户。

2. Microsoft Access 2002中的安全性

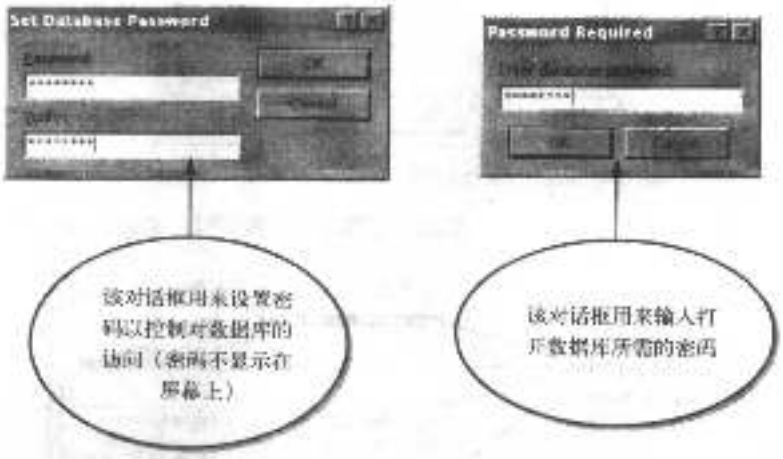
Microsoft Access 2002不支持SQL GRANT和REVOKE语句。作为替换，Access为数据库

安全提供如下两种方法：

- 为打开数据库设置口令（系统安全）。
- 用户级安全，用来限制用户可以读和更新的数据库部分（数据安全）。

(1) 设置口令

更简单的打开数据库的方法是设置一个口令。一旦设置了口令（从Tools中的Security菜单），当打开数据库时，都会弹出要求口令的对话框。此对话框用来设置口令和在打开数据库时要求用户输入口令，如图14-3所示。



a) Set Database Password对话框　　b) 在启动时显示的Password对话框

图14-3 用口令使StayHome数据库安全

只有输入了正确口令的用户，才允许打开数据库。由于Microsoft Access将口令进行了加密，因此该方法是很安全的，所以不能通过直接读取数据库文件的方法来访问数据库。但是，一旦打开数据库，数据库所包含的所有对象对用户就都是可用的。

(2) 用户级安全

Microsoft Access中的用户级安全与大多数网络系统中使用的方法很相似。当开始使用Microsoft Access时，要求用户标识自身并输入口令。在工作组信息文件中，用户被标识为组（group）的成员。访问提供两个默认组：管理员（Admins组）和用户（Users组），但是你也可以定义其他的组。图14-4显示了为用户和组账号定义安全等级的对话框，它显示了非默认的组，称为Assistants，以及一个叫做Assistant的用户，他是Users和Assistants组的成员。

许可（Permission）被授给组和用户来规定用户对数据库中的对象可以进行的操作，许可可以使用User and Group Permissions对话框实现。表14-1说明了在Microsoft Access中可以设置的许可。例如，图14-5说明了一个叫做Assistant的用户在StayHome中只有读StaffT_View的权限。类似的情况是，所有对基本表Staff的访问权限均被删除了，因此，Assistant用户只可以使用这个视图来浏览Staff表中的数据。

3. Microsoft Access的其他安全特性

除了前面介绍的两个Microsoft Access数据库安全方法之外，其他的安全性特性包括：

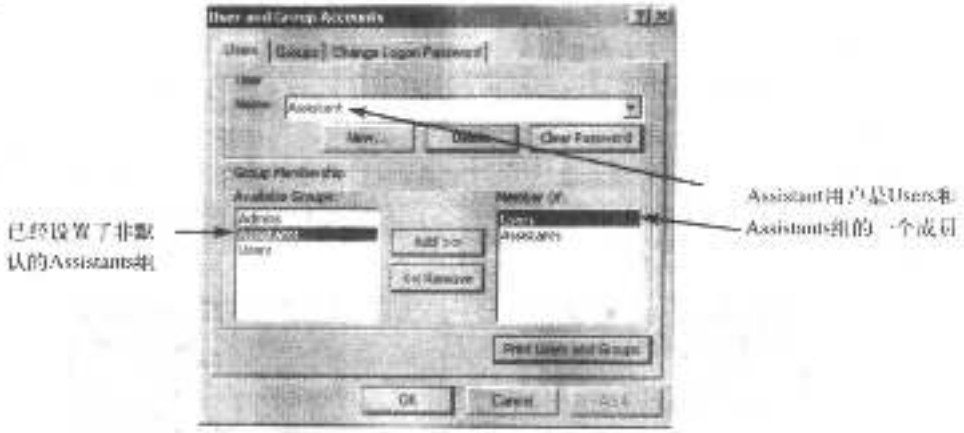


图14-4　StayHome数据库的User and Group Accounts对话框

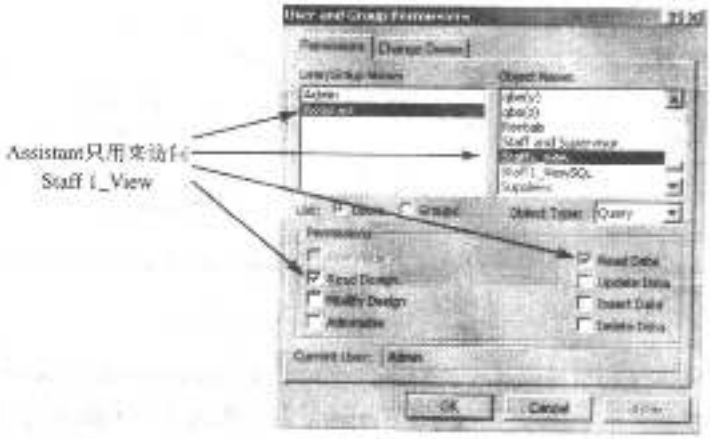


图14-5　User and Group Permissions对话框表明Assistant用户只有读Staff1_View的权限

• 加密/解密：对数据库加密以压缩一个数据库文件，并使它不会被某个实用工具或文字处理软件破译出来。如果你希望传递一个数据库或希望将数据库存储在一个软盘或压缩磁盘上时，这个技术非常有用。解密数据库是加密的逆过程。

表14-1　Microsoft Access许可

许　可	说　明
Open/Run	打开数据库、窗体、报表或运行一个宏
Open Exclusive	用排他访问方法打开一个数据库
Read Design	在Design视图中浏览对象
Modify Design	浏览和更改数据库对象，并删除它们
Administer	对于数据库，设置数据库口令，复制数据库以及更改启动属性。完全访问数据库对象包括分配许可的能力
Read Data	浏览数据
Update Data	浏览和更改数据（但不能插入或删除数据）
Insert Data	浏览和插入数据（但不能更改或删除数据）
Delete Data	浏览和删除数据（但不能插入或更改数据）

- 防止用户复制数据库、设置密码或设置启动选项。
- 安全性VBA代码：可通过每当进入一个会话时设置一个密码来获得此功能，或者通过将数据库文件保存为MDE文件来实现安全性VBA代码，这种形式的文件在从数据库中删除它之前编译这个VBA源代码。将数据库保存为一个MDE文件也防止了用户修改窗体和报表，而不用指定登录密码和设置用户级安全。

将用户视图和安全措施存档

每个用户视图和所关联的安全机制的设计都应该全部存档。如果物理设计影响了逻辑数据模型，那么应该更新这些模型。

14.3 本章小结

- 数据库实际上代表了公共的资源，所以该资源的安全机制是极为重要的。
- 在步骤5中，确定如何在目标DBMS中实现每个用户视图。
- 在步骤6中，确定在需求收集和分析阶段如何标识将在目标DBMS中实现的安全性措施，包括对基本表的访问控制。
- 关系DBMS通常提供两种类型的数据库安全：系统安全和数据安全。系统安全包括系统级的数据库访问和使用，例如，用户名和口令。数据安全包括数据库对象的访问和使用（例如表和视图）以及用户在这些对象上拥有的可执行操作。

复习题

14.1 描述本章介绍的物理数据库设计方法学的主要步骤的目的。

14.2 讨论系统安全和数据安全的区别。

14.3 描述SQL的访问控制功能。

14.4 描述Microsoft Access 2002的安全特性。

练习

14.5 按你的方式用附录E中给出的一些案例研究进行练习，对你访问的DBMS应用在本章中讨论的物理设计方法学中的步骤。创建你认为合适的用户视图以及安全性机制，证明你选择的用户视图和安全性机制是正确的。

第15章 物理数据库设计——步骤7

本章主题:

- 反规范化的含义。
- 何时利用反规范化来改善系统性能。

本章介绍数据库设计方法学中的步骤7。在前边的三章中，我们将逻辑设计转换为一组表和业务规则，并基于对数据库必须支持的事务的分析，为表选择合适的文件组织方式和索引。然后我们决定如何实现用户视图以及如何实现数据库安全。在某些情况下，通过放宽规范化的要求可以改善性能，这就是本章要讨论的内容。

15.1 步骤7：引入受控冗余的考虑

目标 确定是否放松规范化规则引入受控冗余数据来改善系统性能。

规范化是确定哪些列属于同一张表的技术。关系数据库设计的基本目的之一就是对列进行分组，并组织到表中，因为它们之间有直接关系（称为函数依赖，见8.4节）。在数据上实现规范化的结果就是逻辑数据库设计，它是结构上一致的和最小冗余的。

但是，规范化的数据库设计可能不能提供最大的处理效率。这时，你可能愿意接收规范化设计方面的一些损失而实现更好的性能。当估计系统不能满足所需要的性能时，应该只考虑这点。

> 我们不提倡将规范化从逻辑数据库设计中省略：规范化迫使你完整地理解数据库中每个表的每个列。着手于这一步，可能是整个系统成功的最重要的因素。如果考虑反规范化，则下述因素必须要考虑：
>
> - 反规范化使实现更加复杂。
> - 反规范化经常会牺牲灵活性。
> - 反规范化可能加快检索速度，但会降低更新速度。

通常，术语“反规范化”是指对基本表结构的修改，使得新表比原始表的范式低。但是，我们也更宽泛地使用这个术语来指我们将两个表合成一个新表的情况，该新表与原表满足相同范式但比原始表包含更多的空值。

提示 通常，如果性能达不到要求，并且表的更新率较低，查询率较高，则反规范化就是可行的。

在步骤4.1描述的事务/表的交叉引用矩阵为这步提供了有用的信息。该矩阵以可视化方式总结出数据库中运行的事务的访问模式。可以使用它来强调可能的反规范化的候选并评价这种方法对模型其他部分的影响。在处理地址时，你已经遇到了反规范化的一个隐式的例子。例如，考虑Branch表的定义：

```
Branch  (branchNo, street, city, state, zipCode, mgrStaffNo)
```

严格地说，该表并不满足第三范式：zipCode函数决定了city和state，换句话说，如果知道了邮政编码，那么也就可以知道城市和州。因此，规范化该表，把该表一分为二是必要的，如下所示：

```
Branch (branchNo, street, zipCode, mgrStaffNo)
ZipCode(zipCode, city, state)
```

但是，你很少会希望访问分公司地址的时候，没有城市和州这两列，这就意味着当想得到完全的地址时，将不得不执行连接。所以，我们通常实现原始的Branch表并且满足第二范式（2NF）。

但是，没有固定的规则来决定何时对表反规范化，让我们来看看反规范化的更通常的情况，以便加速进行常用或关键的事务：

- 步骤7.1：合并一对一（1：1）关系。
- 步骤7.2：复制一对多（1：*）关系中的非键列来减少连接。
- 步骤7.3：复制一对多（1：*）关系中的外键列来减少连接。
- 步骤7.4：复制多对多（*：*）关系中的列来减少连接。
- 步骤7.5：引入重复组。
- 步骤7.6：创建提取表。
- 步骤7.7：分区表。

1. 步骤7.1：合并一对一（1：1）关系

重新检查一对一（1：1）关系来确定将表合并成一个表的效果。应该只考虑将经常引用的和不经常引用的表分开。让我们考虑Staff和NOK之间的潜在1：1关系（1：1关系定义参见7.5.1节），如图15-1a所示，Staff实体包含关于员工的信息，NOK实体包含员工的近亲的信息。

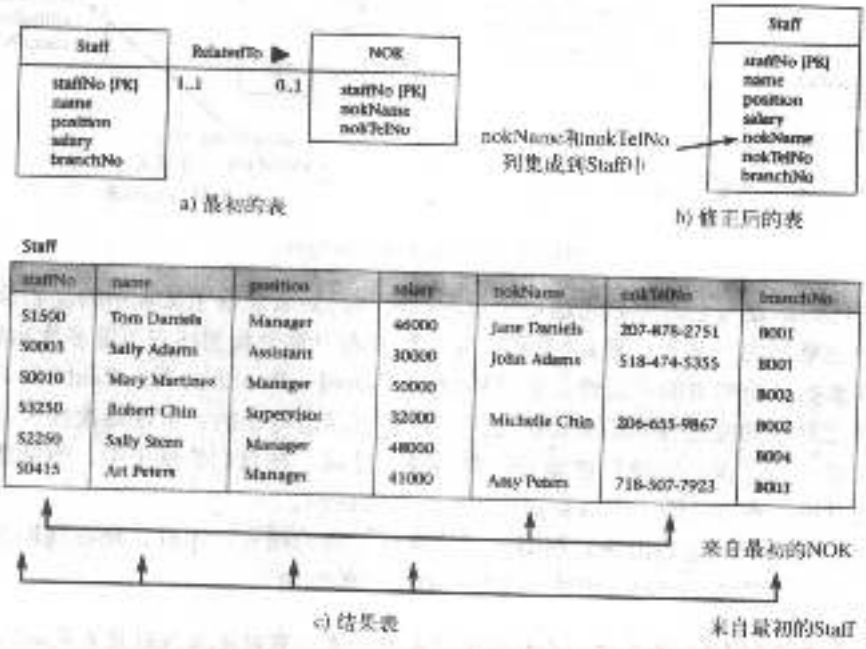


图15-1 Staff和NOK

我们可将这两个表合并在一起，如图15-1b所示。Staff和NOK之间的关系是1:1的，并且参与是可选的。由于参与是可选的，所以当将两个表合并在一起的时候，一些列的一些记录出现了空值，如图15-1c所示。如果Staff表很大，包含在该参与中的该部分的记录的比例很小，那么会有很大数量的空间浪费。因此空间的浪费就不得不与合并表所带来的性能的提高进行权衡了。

2. 步骤7.2：复制一对多（1:*）关系中的非键列来减少连接

为了减少或删除经常性或关键性查询的连接，在一对多关系中，应该考虑在子表中复制一个或更多的父表中的非键列的益处。例如，当访问表VideoForRent时，通常会同时访问录像带的日租金率。典型的SQL查询语句是：

```
SELECT vfr.*, v.dailyRental
FROM videoForRent vfr ,Video v
WHERE vfr.catalogNo = v.catalogNo
      AND branchNo = 'B001';
```

初始的表如图15-2a所示。

如果复制VideoForRent表中的dailyRental列，那么就可以在查询中不涉及Video表，SQL语句如下：

```
SELECT vfr.*
FROM VideoForRent vfr
WHERE branchNo = 'B001';
```

修正后的VideoForRent表如图15-2所示。

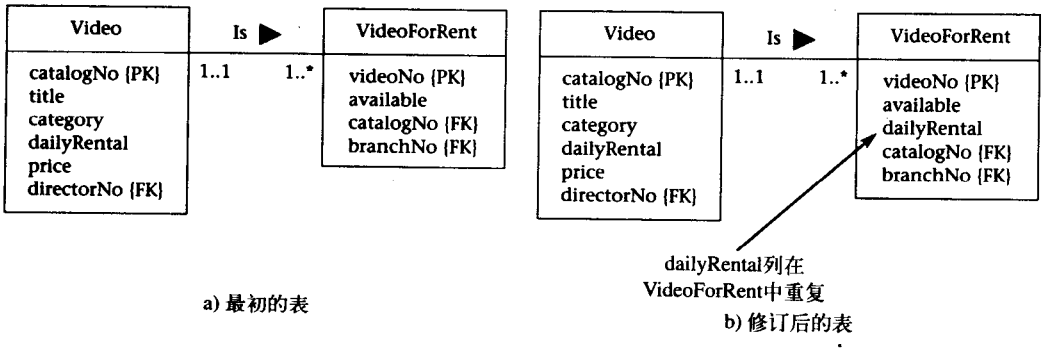


图15-2 Video和VideoForRent

这种变化的益处与它产生的问题是相对的。例如，如果修改了父表中的复制数据，则必须在子表中也更新它。另外，对1:*关系来说，在子表中每个数据项有可能多次出现。因此，你也必须要维护多份拷贝的一致性。如果Video和VideoForRent表的dailyRental列不能自动更新，则要考虑潜在的完整性的丢失。即使这个过程可以是自动的，但在每次插入、更新或删除记录的时候，为了保持完整性也需要花费额外的时间。在我们的例子中，当录像带变旧的时候，可能日租金率会降低，所以复制可能是没有保证的。

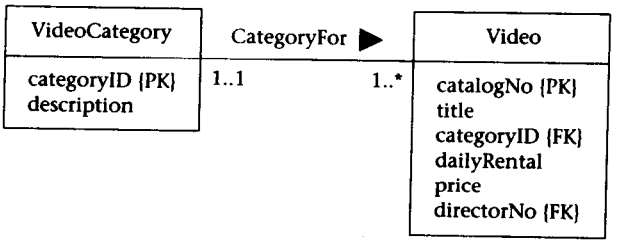
要考虑的另一个问题是由于复制而占用的存储空间的增长。不过，现在辅助存储器相对价格较低，这可能是个小问题。但这并不是任意复制的理由。

提示 一对多（1:*）关系的一个特殊例子是查找表，有时也称为引用表或挑选列表。

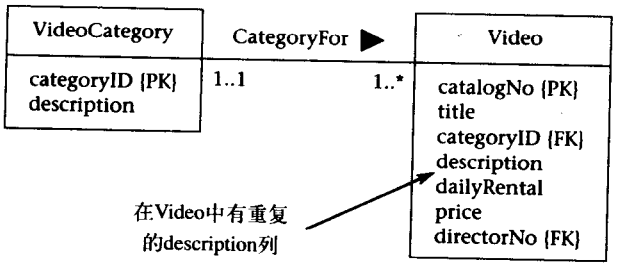
通常，查找表包含一段代码和一个描述。例如，可以为录像类别定义一个查找表并修改它，该表如图15-3a所示。如果这个查找表经常使用，或者是关键的查询，而且描述可能不变化，则考虑在子表中复制该description列，如图15-3b所示。原始的查找表不是多余的，它仍可用于验证用户的输入。但是，通过将description列复制到子表中，就不必将查找表与子表进行连接。

使用查找表的优点包括：

- 降低了子表（在这里，是Video表）的大小，分类代码占1字节，而分类描述占8字节。
- 如果描述可以修改（这不是这个特例的情形），在查找表（VideoCategory）中只需修改一次，而在子表（Video）中就要修改多次。
- 查找表可用于验证用户的输入。



a) 原始表



b) 修正后的表

图15-3 录像分类的查找表

3. 步骤7.3：复制一对多（1：*）关系中的外键列来减少连接

同样，为了减少或删除常用或关键查询的连接，应该考虑复制关系中一个或更多个外键列的好处。例如，StayHome的一个常用查询是列出每个分公司的所有租借协议，使用的SQL语句如下：

```
SELECT ra.*
FROM RentalAgreement ra ,VideoForRent vfr
WHERE ra.videoNo=vfr.videoNo AND vfr.branchNo='Bool';
```

初始表如图15-4a所示。

正如从这个查询中所看到的，为了得到租借协议的列表，必须使用VideoForRent表来获得所需的分公司号——branchNo。可以在RentalAgreement表中复制外键列branchNo来消除连接。也就是，在Branch和RentalAgreement表之间引入直接的关系。这时，可以将SQL查询简化为：

```
SELECT *
```

```
FROM RentalAgreement
WHERE branchNo = 'B001';
```

修正的表如图15-4b所示。如果确实有变化，引入额外的外键约束是很必要的，如步骤2.3中所讨论的。

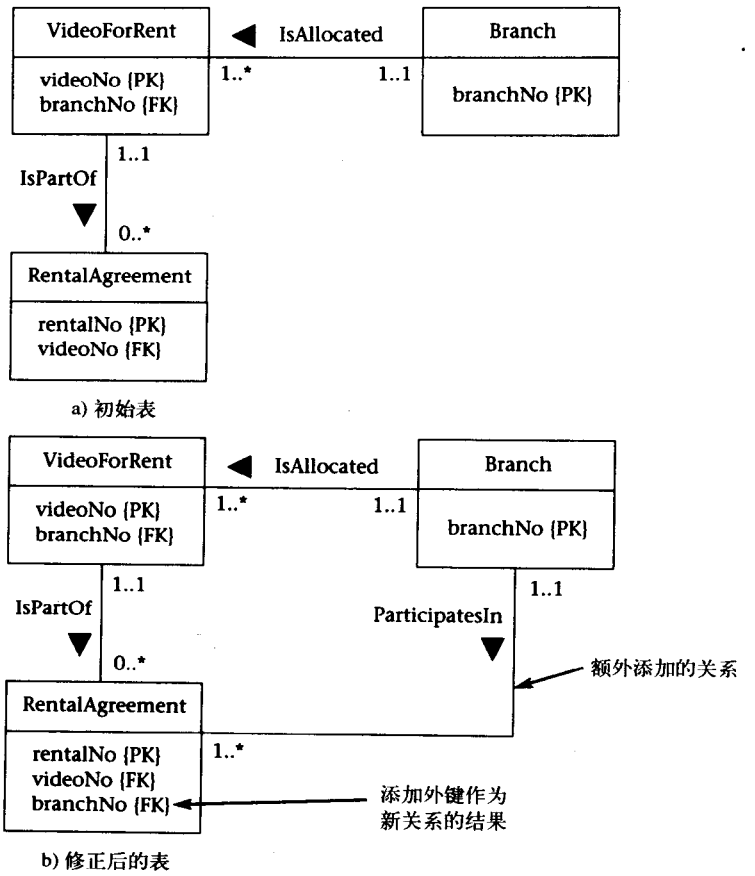


图15-4 RentalAgreement和VideoForRent

注意 Branch和RentalAgreement之间的新关系是1：*的。换句话说，对任何租借协议来说，有并且只有一个与之相联的分公司。如果关系是多对多的，那么上面的变化将不能正常工作。例如，另一个常用查询可能是列出一个分公司中存储的录像带的标题，使用如下SQL查询：

```
SELECT v.title
FROM Video v VideoForRent vfr
WHERE v.catalogNo=vfr.catalogNo AND vfr.branchNo = 'B001';
```

该查询不能简单地为Video表增加branchNo列，因为Branch和Video之间的关系是*：*的，也就是说，一个录像带标题被许多分公司所存储，一个分公司也有许多录像带标题。但是，这时你可能会考虑在VideoForRent表中，复制Video表的title列，尽管这时会增加存储，但在这里可能是很有效的。

4. 步骤7.4：复制多对多（*：*）关系中的列来减少连接

在步骤2.1中，将每个*：*关系映射为三张表：两个从原始实体派生的表和一个表达两个实体间关系的新表。现在，如果你希望从*：*关系中产生信息，则必须要连接这三个表。有时，在中间表中，可以复制初始实体中的列来减少要被连接的表的数量。

例如，Video和Actor之间存在*：*关系，Role作为中间实体。考虑列出录像带标题和每个演员扮演的角色的查询：

```
SELECT v.title,a.*,r.*
FROM  Video v , Role r, Actor a
WHERE  v.catalogNo=r.catalogNo AND r. actorNo=a, actorNo;
```

基于表的图如图15-5a所示。

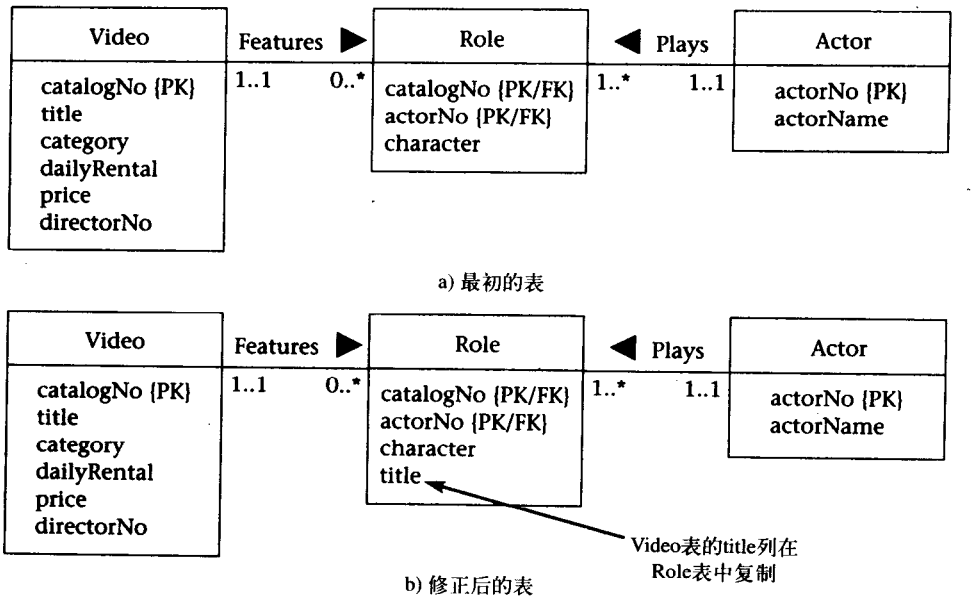


图15-5 Video、Actor和Role

如果在Role表中复制title列，则可以在查询中删除Video表，修正后的SQL查询如下：

```
SELECT a.*,r.*
FROM Role r , Actor a
WHERE r.actorNo = a.actorNo;
```

修正后的表如图15-5b所示。

5. 步骤7.5：引入重复组

为了达到所有实体都满足第一范式（1NF）的要求，应该从逻辑数据模型中删除重复组。重复组被分离到新表中，与初始表（父表）一起形成1:*关系。有时，再次引入重复组是提高系统性能的很有效的方法。

例如，每个StayHome分公司的办公室最少有一个最多有三个电话号码。在逻辑数据模型中，创建了与Branch实体有三对一（3:1）关系的Telephone实体。如图15-6a所示。

如果对这些信息的访问是重要的和经常的，那么合并表并在初始的Branch表中存储电话的详细信息可能会更有效。在这里，每个电话号码用一个列存储，如图15-6b所示。

图15-6 Branch和Telephone

通常，在如下情况下，应该只考虑这种类型的反规范化：

- 重复组中项的绝对数量是已知的（在本例中，是三个电话号码的最大值）。
- 该数量是静态的，并且随着时间的流逝，不会发生改变（StayHome已经固定了分公司中电话线的最大数量，并且不想发生改变）。
- 该数量并不是很大，通常不大于10，尽管这不像前两个条件那么重要。

有时，在重复组中只存储最近的和当前的值，或者只是有重复组，因为它是最经常使用的。在前面的例子中，你可能选择在Branch表中存储一个电话号码，而让其他的电话号码存储在Telephone表中，这将删除Branch表中的空值，因为每个分公司必须至少有一个电话号码。

6. 步骤7.6：创建提取表

可能有这样的情况，你不得不在每天峰值的时候运行某种报表。这些报表要访问派生数据并且基于相同的一组基本表执行多表连接。但是，报表所基于的数据可能是静态的，或者有时并不需要当前的数据（也就是，如果数据已经存在几个小时了，产生的报表可能会更好）。这时，可能需要创建一张基于报表所需要的表的反规范化的提取表，并且允许用户直接访问提取表代替访问基本表。生成提取表的最常用的技术是在系统使用最少的时候，在前一天晚上批运行时创建和生成这些表。

7. 步骤7.7：分区表

除了将表合并在一起之外，还有另外一个方法用于解决非常大的表（和索引）的关键问题，这个方法就是将表分解成一些较小的并且更易于维护的片断，称为分区（partition）。如图15-7所示，有两种主要类型的分区。

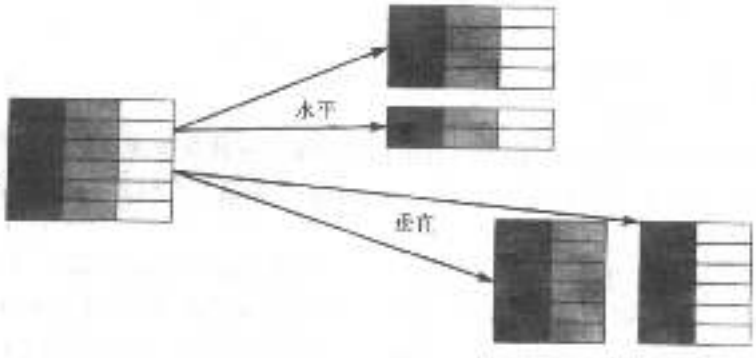


图15-7 水平分区和垂直分区

水平分区（Horizontal Partitioning） 将表中的记录分布在几个较小的表中。

垂直分区（Vertical Partitioning） 将表中的列分布在一些较小的表中（主键是被复制的，以便重构原始表）。

分区在存储和分析大数量的数据的应用中非常有用。例如，假设VideoForRent表中存储几十万条记录，以便进行各种分析。在一个分公司中查找某个特定记录可能是相当耗时间的；但我们可以通过水平分区这个表来减少查找时间，使每个分公司占用一个分区。在Oracle中可以使用如图15-8所示的SQL语句为这个场景创建一个（哈希）分区。

```
CREATE TABLE VideoForRent_Partition(
        videoNo CHAR(6) NOT NULL,
        available CHAR NOT NULL,
        catalogNo CHAR(6) NOT NULL,
        branchNo CHAR(4) NOT NULL,
        PRIMARY KEY videoNo,
        FOREIGN KEY catalogNo REFERENCES
        Video(videoNo),
        FOREIGN KEY branchNo REFERENCES
        Branch(branchNo))
PARTITION BY HASH (branchNo)
(PARTITION b1 TABLESPACE TB01,
PARTITION b2 TABLESPACE TB02,
PARTITION b3 TABLESPACE TB03,
PARTITION b4 TABLESPACE TB04);
```

图15-8 创建（哈希）分区的Oracle SQL语句

和哈希分区一样，常见的分区还有范围分区（每个分区中包含一个或多个列的范围值）和列表分区（每个分区包含一列的值列表）。还有复合分区，如范围－哈希分区和列表－哈希分区（每个分区包含值的范围或值的列表，然后再根据哈希函数进一步分解每个分区）。

也有这种情况，我们经常需要在一个非常大的表中检查某些特定的列，这时对表进行垂直分区就比较合适，将经常访问的列放置到一个表中，将其他的列放置在其他表中（在每个分区中都要复制主键以便重构原表）。

分区表有许多优点：

- 改善负载平衡：分解后的表可以放置在二级存储的不同的地方，因此允许并发访问，而如果表不被分解，则要在相同的时间争用相同的存储区域。
- 改善性能：通过限制将被检测或处理的数据的数量，并且通过建立并行执行机制，可以改善性能。
- 增强可用性：如果分区被存放在不同的存储区域中，并且当一个存储区域变成不可用时，其他分区的内容仍然是可用的。
- 改善可恢复性：分区越小，恢复起来就更快（而且，DBA备份小分区比备份很大的表要更容易）。
- 安全性：一个分区中的数据可以限制只有某些用户可以访问，不同分区中的数据可以有不同的访问限制。

分解表也有一些缺点：

- 复杂性：分区表通常对最终用户是不透明的，而且使用多个分区表数据的查询写起来将比较复杂。
- 降低性能：当查询用到多个分区中的数据时，执行速度比未分区之前慢。
- 重复：垂直分解涉及主键的复制，这导致了存储需求的增加，也导致了潜在的不一致性的增加。

8. 考虑反规范化的影响

应该考虑反规范化对方法学前边的步骤的影响。例如，可能需要重新考虑已进行反规范化的表中索引的选择，以检查现存的索引是否应该被删除，或者添加额外的索引。另外，需要考虑如何维护数据完整性，通常的解决方法是：

- 触发器：触发器可用于自动更新派生的或复制的数据。
- 事务：在每个应用中构建事务使对反规范化数据的更新作为一个事务的操作。
- 批调和：在合适的时间运行批程序以保持反规范化数据的一致。

就维护完整性而言，触发器是最好的解决办法，尽管它们可能会产生性能问题。表15-1总结了反规范化的优点和缺点。

表15-1　反规范化的优点和缺点

优　点	缺　点
可以通过下述方法改善性能： • 预计算派生数据 • 最小化需要的连接 • 减少表中外键的数量 • 减少索引的数量（因此节省了存储空间） • 减少了表的数量	可能会加快检索速度，但会降低更新速度
	总是与应用有关并且如果应用被改变，则需要进行重新评估 可能增加表的大小 在某些情况下可能简化实现，但在其他情况下可能会更复杂 牺牲了灵活性

9. 将引入的冗余存档

引入的冗余以及引入的原因应该全部存档。尤其是，将存在许多可选方案时选择其中一种的原因存档。更新逻辑数据模型和支持文档来反映反规范化所带来的变化作为文档的结果存档。

15.2　本章小结

- 在步骤7中，考虑引入受控冗余来改善性能。
- 有时可能发生这样的情况，接收失去完全规范化设计所带来的好处而改善性能是很必要的。仅当估计系统不能满足所需要的性能时，才考虑这点。
- 作为一个规则，如果性能不能达到要求而且表的更新率比较低而查询率非常高，则非规范化可能是个可行的选择。
- 在如下情况下考虑反规范化，特别对于加速常用或关键事务：合并一对一（1:1）关系；复制一对多（1:*）关系中的非键列来减少连接；复制一对多（1:*）关系中的外键列来减少连接；复制多对多（*:*）关系中的列来减少连接；引入重复组；创建提取表；划分非常大的表。

复习题

15.1 描述数据库设计方法学中步骤7的目的。

15.2 解释反规范化的含义。

15.3 讨论什么情况下反规范化一个表比较合适，给出能够说明你的答案的例子。

15.4 描述分区表的两个主要方法，并讨论每种方法的适用情况以提高性能。给出能够说明你的答案的例子。

练习

15.5 对附录E中的每个案例研究，讨论什么时候进行反规范化是合适的。

第16章 物理数据库设计——步骤8

本章主题:

- 监视并调整操作系统的重要性。
- 如何确保有效性。
- 系统资源怎样影响性能。

本章讲述了我们的物理数据库设计方法学的最后一步。在前面的四章中，我们介绍了将逻辑设计转换为一组基本表、选择合适的文件组织方式和基于对绝大多数重要事务分析而得来的索引，考虑利用受控冗余来获得额外的系统性能，并且说明了如何实现数据库安全。

随着用户需求的深化，调整或调节数据库以继续获得进一步的功能通常是很有必要的。此外，你可能发现需求变化了，既是系统成功、用户要求更多功能的结果，也是业务深化和调整的结果。在本章中，我们考虑影响这些方面的物理数据库设计的其余步骤。

16.1 步骤8：监视并调整操作系统

目标 *监视操作系统并改善系统的性能以改正不正确的设计决策，或者反映变化的需求。*

物理数据库设计的一个主要目标是以有效的方式存储数据，衡量有效性时有很多因素可以使用:

- 事务吞吐量：这是在一个给定时间间隔内处理事务的数量。在有些系统中，比如机票预订系统，高的事务吞吐量是整个系统成功的关键。
- 响应时间：这是完成单个事务所花费的时间。从用户的角度看，你希望尽可能地最小化响应时间。但有一些因素影响了响应时间，而这些因素可能是你无法控制的，比如系统加载时间或通信时间。可以通过下述方法缩短响应时间:
  - 降低争用和等待时间，特别是磁盘I/O等待时间。
  - 降低所需时间资源的数量。
  - 使用较快的组件。
  - 磁盘存储：这是存储数据库文件所需的磁盘空间，你可能希望最小化使用的磁盘存储的数量。

但没有一个因素总是正确的。通常，你在几个因素之间进行权衡，来获得合理的平衡。例如，增加数据存储的数量可能会降低响应时间或事务的吞吐量。不应该将原始的物理数据库看成是静止不变的，而是要估计操作系统会怎样实现。一旦实现了初始设计，就应该监视并调整系统，以作为观察性能和改变需求的结果。许多DBMS提供了让数据库管理员（DBA）监视系统的操作并调整系统的功能。

调整数据库可以得到许多好处:

- 可以避免使用多余的硬件。
- 可能会减少对硬件配置的要求。这导致更小和更便宜的硬件和潜在地更昂贵的维护。
- 调整较好的系统有更快的响应速度和更高的吞吐量，从而使用户和公司更有效益。

• 改善的响应速度能提高员工的士气。

• 改善的响应速度能增强顾客的满意度。

后两条好处比其他的更抽象。但是，我们当然可以肯定地说，很慢的响应速度会使员工失去信心并且会潜在地失去客户。

要调整数据库系统，需要理解不同的系统组件间的相互作用以及对数据库性能的影响。

16.1.1 理解系统资源

为了改善性能，你必须清楚四个基本的硬件组件是如何相互影响并影响系统性能的:

• 主存

• CPU

• 磁盘I/O

• 网络

每个资源都可能会影响系统的其他资源，同样，改进一个资源可能也会改进其他的系统资源。例如:

• 添加更多的主存一定会减少页面调度，这有助于避免CPU瓶颈。

• 更有效地使用主存可能会减少磁盘I/O。

1. 主存

访问主存要比访问辅存快得多，有时要快几万或几十万倍。通常，对DBMS和数据库应用来说，可用的主存越多，应用程序运行得越快。但是，至少总是保留5%的主存可用是较明智的。同样地，主存也不应该超过10%是可用的，否则主存并不是被最佳使用。当主存不能容纳所有的进程时，操作系统把进程页转移到磁盘上，来释放主存。当这些页中的某页又被需要时，操作系统必须把它从磁盘再调回到主存。有时，需要将整个程序从主存调到磁盘，然后再交换回主存，以释放主存。当换页（也叫做交换）次数过多时，就会发生问题。

为确保有效地利用主存，你需要知道目标DBMS是如何使用主存的，它在主存中保存的缓冲区是什么，允许你调整这些缓冲区的参数是什么等等。例如，Oracle在主存中保留一个数据字典高速缓存，理想的情况是这个高速缓存应该足够大以处理90%的数据字典访问，而不必从磁盘检索信息。你也需要理解用户的访问模式: 访问数据库的并发用户的增加将导致使用的内存数量增加。

2. CPU

CPU控制系统其他资源的任务并执行用户进程。该部件的主要目标是防止CPU争用，争用时进程均在等待CPU。当操作系统或应用程序对CPU提出过多的要求时，CPU就会产生瓶颈。这经常是过度换页的结果。

你需要知道24小时内一般的工作量，并确保不仅在正常的工作量下而且在峰值的工作量下也有足够的资源（例如，如果你发现在正常工作量期间有90%的CPU在使用，并有10%的CPU空闲，那么在峰值工作量期间CPU可能就不够了）。一个选择是确保在峰值期间不运行不必要的工作，让这些工作在峰值之外的时间运行。另一个选择可以是考虑多CPU，这允许分布式处理和以平行的方式处理操作。

CUP MIPS（Millions of Instructions Per Second，每秒指令数）可用于作为比较平台的参数，并以此决定它们处理企业吞吐量需求能力。

3. 磁盘I/O

对任何大型的DBMS来说，在存储和检索数据时都有相当数量的磁盘I/O。在最近几年，CPU时钟速度得到了飞速的发展，而I/O速度并没有得到成比例的增长。数据在磁盘上的组织方式对整个磁盘的性能有很大的影响。可能引起的一个问题是磁盘争用，当多个处理器都试图同时访问同一个磁盘时就会发生这种情况。大多数磁盘对访问的数量和每秒可传输的数据量都有限制，当达到这个限制时，CPU就不得不等待磁盘。为避免这种情况，建议存储时将数据分布在多个可用的磁盘上，以减少发生性能问题的可能性。图16-1说明了数据在磁盘中分布数据的基本原理：

- 操作系统文件应该与数据库文件分离。
- 主数据库文件应该与索引文件分离。
- 恢复日志文件（如果可能并且可用），应该与数据库其他的部分分离。

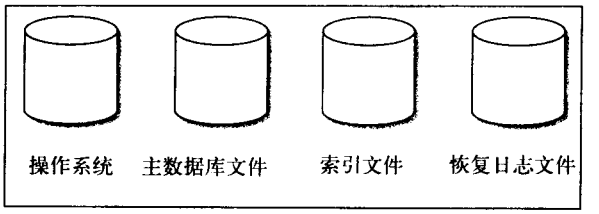


图16-1 典型的磁盘配置

如果磁盘仍然出现过载，可以将一个和多个经常访问的文件移动到较少被访问的磁盘上（这称为分布式I/O），通过对每个磁盘使用这些原则，让每个磁盘基本具有相同数量的I/O，就可以获得负载平衡。再强调一次，你必须理解DBMS是如何操作的，了解硬件的特性以及用户的访问模式。

RAID

磁盘I/O已经因RAID技术的出现而得到了很大的改进。RAID最初表示Redundant Arrays of Inexpensive Disks（冗余磁盘阵列），但最近，RAID中的“I”已经替换为Independent（独立性）。RAID工作模式是有一个大的磁盘阵列，它由几个独立的且有组织的磁盘排列而成，用于增强性能以及提高可靠性。

性能是通过数据条（striping）增加的，数据条是指将数据划分成相同大小的分区（分条单位），这些分区透明地分布在多个磁盘上。这会让用户觉得是一个大的、非常快的磁盘，而实际上这些数据是被分布在几个较小的磁盘上的。通过允许多个I/O并行进行，分条技术提高了整个I/O的性能，同时数据分条也平衡了磁盘间的负载。通过使用奇偶检验机制或错误校验机制在多个磁盘上存储冗余信息也提高了可靠性。在磁盘失败时，冗余信息可用于重构失败磁盘上的内容。

有几种形式的磁盘的配置，即RAID级别，每个级别提供的性能和可靠性略有不同。这些RAID级别是：

- RAID 0——无冗余：这个级别维护无冗余数据，因此有最好的写性能，因为不需要复制更新。数据条在块一级完成。
- RAID 1——镜像：这个级别维护（镜像）不同磁盘上的数据的两个相同的拷贝。在磁盘失败时要维护一致性，写可能不是同步完成的。这是最昂贵的存储方式。

- RAID 0+1——非冗余和镜像的：这个级别将分条和镜像结合起来。
- RAID 2——错误校正代码：在这个级别，分条单位是一个位，并且错误校正代码被作为冗余机制使用。
- RAID 3——位交叉奇偶校验：这个级别通过在阵列中的一个磁盘上存储奇偶信息来提供冗余。奇偶信息可用于恢复在其他磁盘上的数据。这个级别比RAID 1使用的存储空间少，但奇偶磁盘可能会成为瓶颈。
- RAID 4——块交叉奇偶校验：在这个级别，分条单位是一个磁盘块——在一个单独的磁盘上维护一个奇偶块，用于与其他磁盘上的块相对应。如果某个磁盘失败了，则此奇偶块可与其他磁盘上对应的块一起恢复失败磁盘上的块。
- RAID 5——块交叉分布的奇偶校验：这个级别使用奇偶数据来提供冗余，这同RAID 3的方式一样，但是在所有的磁盘上将奇偶数据分条，这同分条源数据的方式很类似，这减轻了在奇偶磁盘上的瓶颈。
- RAID 6——P+Q冗余：这个级别类似于RAID 5，但维护附加的冗余数据以防止多个磁盘失败。它使用错误校验代码而不是奇偶校验。

对于大多数数据库应用程序，通常选择RAID 1、RAID 0+1和RAID 5。例如，Oracle推荐对于重做日志文件使用RAID 1，对于数据库文件，Oracle推荐使用RAID 5，只要写负载是可接受的，否则，Oracle推荐RAID 1和RAID 0+1。再强调一次，你应该了解哪个RAID选项适合于你的硬件配置，并且要知道不同的DBMS组件如何使用磁盘I/O，以便你可以选择合适的方案。

4. 网络

当大量数据在网络间传送时，就会发生网络瓶颈问题。

16.1.2 小结

调整是永远都不会终结的活动。在系统的整个生命周期中，需要监视系统性能，尤其是解决环境和用户需求的变化。但是，在操作系统某一方面上进行提高性能的改变，可能会对另一方面起反作用。例如，为表增加索引，会改善一种应用的性能，但可能会影响另一个可能更重要的应用。因此，当对操作系统进行修改时，一定要小心。如果可能，在实验数据库上测试这些更改，或者在系统没有被完全使用时（例如，工作时间以外），对系统进行测试。

提示 正如我们在方法学的步骤4中所讨论的，大多数的性能增益来自好的数据库设计，正确事务分析和使用合适的索引。尽管跳过某些步骤或仓促完成某些步骤会节省时间，但我们强烈提倡不要这样做，我们相信在数据库设计中花费足够的时间在以后的过程中将受益无穷。

将调整活动记入文档

调整系统所使用的机制应该被完全地记录到文档中，包括使用这种调整方案的原因。特别的，当有多个可选方案时，要记录下选择其中某个方案的原因。

16.1.3 StayHome的新需求

在调整系统以维护最佳性能的同时，也要考虑变化的需求。我们已经确定了Video表应该

存储录像的封面和简要的故事情节，以准备在网上制定录像目录。可以使用Microsoft Access中的OLE（对象链接与嵌入）数据类型，该数据类型用来存储Microsoft Word或Excel的文档、图片、声音以及在其他程序中创建的其他类型的二进制数据。OLE对象可以链接或嵌入到Microsoft Access表的字段中，并且能够在窗体或报表中显示出来。

为了满足这种新需求，可以重构Video表，增加如下的列：

- videoCover列，声明为OLE Object数据类型，这个列字段保存录像封面的图像，这个图像是通过扫描封面并将这个图像保存为BMP图形文件创建的。
- storyLine列，声明为Memo数据类型，可存储较长的文本。

使用这些新列的表格如图16-2所示。这两个额外加入的列的主要问题是潜在地扩大了使用的磁盘空间，这些空间能存储大量的图形文件，并为故事情节存储大量的文本。因此，需要继续监视StayHome数据库的性能，以确保满足新需求的同时没有危害系统的性能。

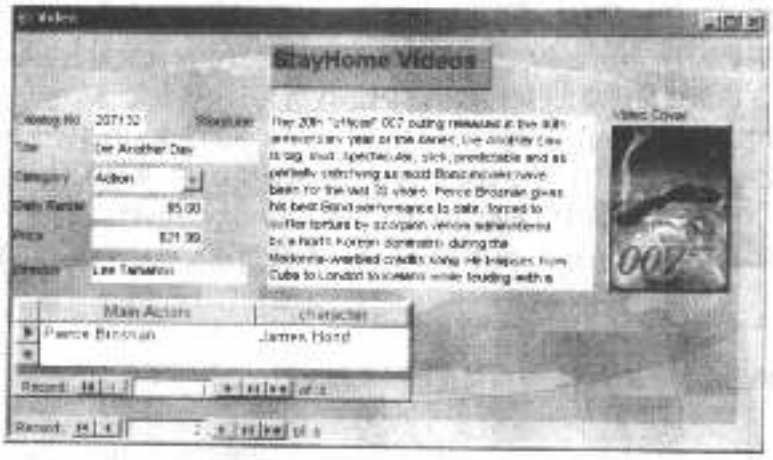


图16-2　基于修改后增加了videoCover和storyLine两列的Video表的窗体

> 既然已经学习了方法学中的所有步骤，你可能想回顾9.1.3节，重新认识我们所说的成功的数据库设计的关键。当你首次读这一节时，可能很难看出某些因素的正确，但希望你现在会发现所有因素的重要性。

16.2　本章小结

- 步骤8是物理数据库设计的最后一步，包括监视并调整操作系统，以获得最大性能或者反映变化的需求。
- 物理数据库设计的一个主要目标就是以有效的方式存储数据。有许多因素可用于衡量有效性，包括吞吐量、响应时间和磁盘存储。
- 为改善性能，你必须知道下述四个硬件组件是如何相互作用的以及如何影响系统性能的。这个四个硬件是：主存、CPU、磁盘I/O和网络。
- 磁盘I/O由于引入RAID（Redundant Array of Independent Disks）技术而发生了革命性的变化。RAID 工作模式是有一个大的磁盘阵列，它由几个独立的且有组织的磁盘排列而成，用于增强性能以及提高可靠性。

复习题

16.1 描述本章给出的物理数据库设计方法学步骤的主要目的。

16.2 可以使用哪些因素衡量有效性?

16.3 讨论四个基本的硬件组件是如何相互作用并影响系统性能的。

16.4 应该如何在磁盘中分布数据?

16.5 什么是RAID技术? 这个技术是如何提高性能和可靠性的?

第五部分　第二个实例

第17章　Perfect Pets——逻辑数据库设计
第18章　Perfect Pets——使用物理数据库设计方法学

第17章　Perfect Pets——逻辑数据库设计

在本章中，我们又提供了一个例子，以帮助加强理解第9章－第16章中所提出的方法学。我们在这一章使用逻辑数据库设计方法学的步骤，然后在下一章使用物理数据库设计方法学的步骤。为了说明物理实现的不同方面，我们也使用了一个不同的关系数据库管理系统，即Oracle 9i。我们建议首先阅读下面所介绍的例子，然后读者最好自己按照方法学的步骤进行设计，你可以将自己的方法和书中的示例进行比较。附录A对方法学进行了总结。

17.1　Perfect Pets

叫做Perfect Pets的组织为全美的宠物提供了个体健康管理。位于美国主要城市的不同诊所共同提供了该服务。Perfect Pets的执行官注意到在该业务领域缺乏交流，尤其是在不同的诊所之间共享信息和资源很少。为了解决这个问题，执行官要求建立一个中央数据库，来帮助该业务更有效地运行。执行官为当前系统提供了如下的描述。

17.1.1　数据需求

1. 宠物诊所

Perfect Pets在全美的大城市有许多宠物诊所。每个诊所的详细信息包括诊所号、诊所地址（由街区、城市、州和邮政编码组成）、电话和传真号码，每个诊所有一个经理和一些员工（例如，兽医、护士、秘书、清洁人员等）。在整个组织中，诊所号是唯一的。

2. 员工

员工中每个成员的详细信息包括员工号、姓名（姓和名）、地址（街区、城市、州和邮编）、电话号码、出生日期、性别、社会保障号码（SSN）、职位和当前的全年工资。在整个业务中，员工号是唯一的。

3. 宠物主人

当宠物主人第一次到Perfect Pets诊所时，宠物主人的详细信息就被记录下来。其中包括主人号码、主人姓名（姓和名）、地址（街区、城市、州和邮编）、家庭电话号码。对每个诊所来说，主人的号码是唯一的。

4. 宠物

需要诊治的宠物的详细信息包括宠物号码、宠物名、宠物类型、宠物描述、出生日期

（如果不知道，记录一个大致的日期）、到诊所的登记日期、当前状态（生/死）以及宠物主人的详细信息。对每个诊所来说，宠物号码是唯一的。

5. 检查

当一只生病的宠物到达诊所后，值班兽医开始为它进行检查。记录每次检查的详细信息，其中包括检查号码、检查的日期和时间、兽医的姓名、宠物号码、宠物名字、宠物类型，以及检查结果的详细描述。对每个诊所来说，检查号码是唯一的。作为检查的结果，兽医应该为宠物提出诊治方案。

6. 诊治方案

Perfect Pets为所有类型的宠物提供不同的诊疗方式。这些诊治方式在所有诊所内遵循相同的标准。每次诊治的详细信息，包括诊治方案号、诊治方案的具体内容、宠物主人的花费。例如，诊治方案包括:

T123	盘尼西林抗生素系列	$50.00
T155	猫科绝育手术	$200.00
T112	预防猫科流行性感冒的接种	$70.00
T56	小型犬科——每天的照顾（包括饲养）	$20.00

每次检查都要付20.00美元，检查时记录该检查的诊治种类。诊治号码唯一地标识每种诊治种类，并且在所有的Perfect Pets诊所是通用的。

7. 宠物诊治

基于对生病的宠物的检查结果，兽医提出一种或几种类型的诊治方案。这个诊治方案要记录的信息包括检查号码和日期、宠物号码、名字和种类、诊治方案号、处方描述、每种类型的诊治数量以及诊治的开始和结束日期。每种类型的诊治方案的额外事项也要记录。

8. 围栏

有时，必须把生病的宠物留在诊所。每个诊所有20~30个动物围栏，每个围栏可容纳1~4个宠物。每个围栏有唯一的围栏号、容量和状态（表明是否可用）。生病的宠物被安置在围栏中，同时记录下宠物的详细信息、该宠物需要的诊治方案以及所有额外的照管该宠物的信息。同时也要记录宠物待在围栏中的详细信息，包括围栏号、宠物进入和离开围栏的日期。根据宠物的生病情况，在一个围栏中可能会同时容纳多只宠物。对每个诊所来说，围栏号是唯一的。

9. 账单

宠物的主人要负责宠物诊治的费用。检查诊治宠物后，要给该宠物的主人开账单。记录账单的详细情况包括账单号码、账单日期、宠物主人号码、宠物主人姓名和完整地址、宠物号码、宠物名字和诊治方案的详细信息。账单提供了每种诊治所需的费用和对一个宠物所进行的所有诊治的总的费用。

在支付账单的时候，还要记录一些数据，包括支付账单的日期和支付的方法（例如，支票、现金、信用卡）。在整个组织中，账单号码是唯一的。

10. 外用品、非外用品及药品的供应

每个诊所都有一些外用品（例如，注射器、无菌胶带、绷带）和非外用品（例如，塑料袋、围裙、小盘子、宠物名称的标签、宠物食品）的储备。外用和非外用品供应的详细信息包括项目号和名称、项目描述、储存数量（在每个月的最后一天确定）、追加订货程度、追加订货数量和费用。在诊所中，项目号对每种外用和非外用品是唯一的。

每个诊所也存储一些药品（例如，抗生素、镇痛剂）。药品供应的详细信息包括药品号和名称、药品介绍、计量、配方、储存数量（每个月的最后一天确定）、记录级别、记录订货数量和费用。药品号唯一地标识每种供应药品，并在整个组织内通用。

11. 预约

如果宠物要稍后再看医生，主人和宠物就要进行预约。记录预约的详细信息，包括预约号码、主人号码、主人姓名（姓和名）、家庭电话号码、宠物号码、宠物名称、宠物种类和预约日期和时间。对每个诊所来说，预约号码是唯一的。

17.1.2 事务需求

下面列出的是Perfect Pets数据库应用应该支持的事务。

1. 数据应该能够支持下述维护事务

(a) 创建和维护记录Perfect Pets的各诊所的详细信息和每个诊所的员工的记录。

(b) 创建和维护记录宠物主人的详细信息的记录。

(c) 创建和维护宠物的详细信息。

(d) 创建和维护记录可实现的诊治宠物方案的详细信息的记录。

(e) 创建和维护记录对宠物所进行的检查和诊治的详细信息的记录。

(f) 创建和维护记录给宠物主人的治疗其宠物的花费的账单的详细信息的记录。

(g) 创建和维护记录每个诊所中的外科用品、非外科用品和药品的供应的记录。

(h) 创建和维护记录每个诊所中可用的围栏，以及每个围栏宠物的分配的记录。

(i) 创建和维护每个诊所中，宠物主人/宠物的预约。

2. 数据应该能够支持下述查询事务

(a) 以报表形式列出经理的名字、诊所地址、每个诊所的电话号码，按诊所号码排序。

(b) 以报表形式列出宠物主人的姓名、号码，以及他们宠物的详细信息。

(c) 列出对某一给定的宠物进行检查的历史信息。

(d) 列出基于给定的检查结果提供的诊治方案的详细信息。

(e) 为某一给定的宠物主人列出没有支付的账单的详细信息。

(f) 以报表形式列出到某一给定日期还没有支付的账单，按账单号排序。

(g) 列出某一给定日期纽约地区诊所可使用的围栏的详细信息，按诊所号排序。

(h) 以报表形式提供每个诊所全体员工月工资的总和，按诊所号排序。

(i) 列出诊治费用的最大值、最小值和平均值。

(j) 列出每种类型的宠物的总数，按宠物类型排序。

(k) 以报表形式列出所有超过50岁的兽医和护士的姓名和员工号，按员工姓名排序。

(l) 列出某一诊所，在某一给定日期的预约情况。

(m) 列出每个诊所的宠物的总数，按诊所号排序。

(n) 以报表形式列出1997年～1999年之间的宠物主人的账单的详细信息，按账单号排序。

(o) 列出特定宠物主人的宠物的号码、名字和详细描述。

(p) 以报表形式列出每个诊所应再订购的药物，按诊所号排序。

(q) 列出每个诊所当前所储存的非外科用品和外科用品的总价值，按诊所号排序。

17.2 使用逻辑数据库设计方法

在这一节中，我们继续使用逻辑数据库设计方法学中的各个步骤，提出一个可满足以上需求的逻辑数据模型。我们假定需求收集和分析阶段只定义了一个用户视图。

17.2.1 步骤1.1：标识实体

逻辑数据库设计的第一个步骤是标识在数据库中必须描述的主实体。由上面的描述，可以标识如下实体：

Clinic（诊所）	Staff（员工）
PetOwner（宠物主人）	Pet（宠物）
Examination（检查）	Treatment（诊治方案）
Pen（围栏）	PetTreatment（宠物诊治）
Invoice（账单）	Appointment（预约）

Stock(特化Surgical（外用）、NonSurgical（非外用）和Pharmaceuticals（药品））

将实体存档

为实体命名时，应该有意义的并且对用户直观的名字。在数据字典中记录他们的详细信息。图17-1为数据字典中记录Perfect Pets实体的部分。

Entity name	Description	Aliases	Occurrence
Clinic	Veterinary clinics.	Surgery	One or more *Perfect Pet* clinics located in main cities throughout the US
Staff	General term describing all staff employed by *Perfect Pets*.	Vet, Nurse, Secretary	Each member of staff works at a particular clinic
PetOwner	Owners of pets taken to *Perfect Pets*.		Owner takes his/her pet to a particular clinic.

图17-1　数据字典中记录Perfect Pets实体的部分

17.2.2 步骤1.2：标识关系

标识完实体后，下一步就是标识存在于这些实体之间的所有关系（见5.2节）。对于Perfect Pets而言，应该标识出如图17-2所示的关系。

1. 确定关系的多样性约束

标识完要创建的关系后，现在应该确定每个关系的多样性（见5.5节）约束。对于Perfect Pets而言，应该标识出如图17-3所示的多样性约束。

2. 使用实体－关系（ER）建模

在数据库设计阶段，将创建几个代表Perfect Pets 的ER模型。图17-4显示了Perfect Pets的ER模型初稿。

实体	关系	实体
Clinic	Has	Staff
	Holds	Stock
	Registers	Pet
	Provides	Pen
	Schedules	Appointment
	IsContactedBy	PetOwner
Staff	Manages	Clinic
	Performs	Examination
PetOwner	Owns	Pet
	Pays	Invoice
	Attends	Appointment
Pet	Undergoes	Examination
	IsAllocatedTo	Pen
	Attends	Appointment
Examination	ResultsIn	PetTreatment
Treatment	UsedIn	PetTreatment
Invoice	ResultsFrom	Examination

图17-2 Perfect Pets中关系的初稿

实体	多样性	关系	实体	多样性
Clinic	1..1	Has	1..*	Staff
	1..*	Holds	1..*	Stock
	1..1	Registers	1..*	Pet
	1..1	Provides	20..30	Pen
	1..1	Schedules	1..*	Appointment
	1..1	IsContactedBy	1..*	PetOwner
Staff	1..1	Manages	0..1	Clinic
	1..1	Performs	0..*	Examination
PetOwner	1..1	Owns	1..*	Pet
	1..1	Pays	1..*	Invoice
	1..1	Attends	1..*	Appointment
Pet	1..1	Undergoes	1..*	Examination
	1..*	IsAllocatedTo	0..*	Pen
	1..1	Attends	1..*	Appointment
Examination	1..1	ResultsIn	1..*	PetTreatment
Treatment	1..1	UsedIn	1..*	PetTreatment
Invoice	1..1	ResultsFrom	1..1	Examination

图17-3 上面所定义的关系的多样性约束

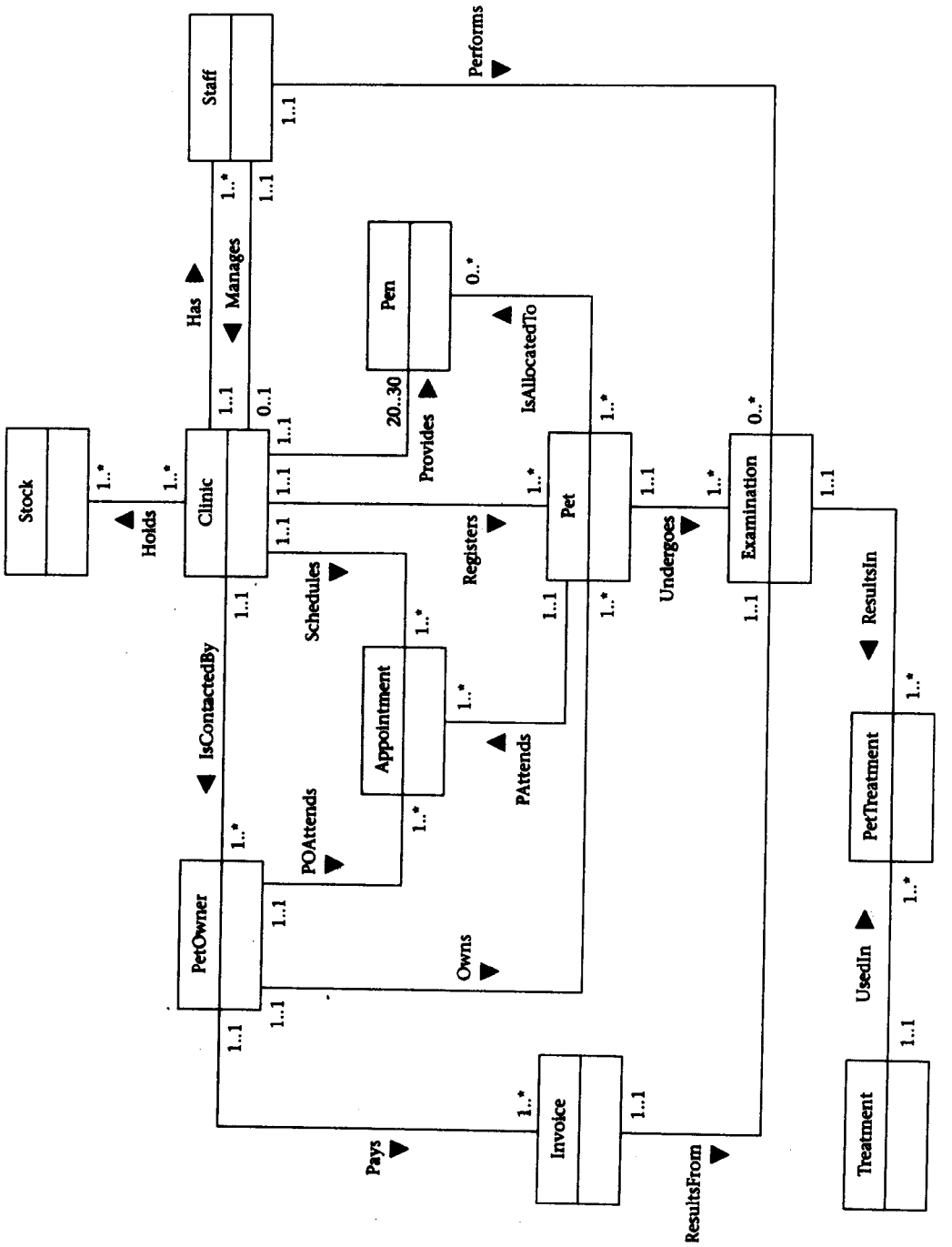


图17-4 Perfect Pets的ER模型初稿

17.2.3 步骤1.3：标识实体或关系的有关属性

下一个步骤是标识与已经标识的实体或关系有关的属性。对于Perfect Pets而言，应该标识如图17-5a所示的与实体相关的属性。

实　体	属　性
Clinic	clinicNO、address (street、city、state、zipCode)、telNo、faxNo
Staff	staffNo、sName (sFName、sLName)、sAddress (sStreet、sCity、sState、sZipCode)、sTelNo、DOB、sex、SSN、position、salary
PetOwner	ownerNo、oName (oFName、oLName)、oAddress (oStreet、oCity、oState、oZipCode)、oTelNo
Pet	petNo、petName、petType、petDescription、pDOB、dateRegistered、petStatus
Examination	examNo、examDate、examTime、examResults
Treatment	treatNo、description、cost
Pen	penNo、penCapacity、penStatus
Invoice	invoiceNo、invoiceDate、datePaid、paymentMethod
Stock: Item	itemNo、itemName、itemDescription、itemCost
Stock: Pharmacy	drugNo、drugName、drugDescription、dosage、methodAdmin、drugCost
Appointment	appNo、aDate、aTime
PetTreatment	startDate、endDate、quantity、ptComments

a) 与实体有关的属性

关　系	属　性
IsAllocatedTo	dateIn、dateOut、comments
Holds	inStock、reorderLevel、reorderQty

b) 与关系有关的属性

图17-5 Perfect Pets中的属性

但是，当检查关于围栏的信息时，描述宠物进/出围栏日期的dateIn/dateOut属性以及comment（注释）属性，很难只与Pen实体或Pet实体相关。同样，描述不同物品存储数量的inStock属性以及reorderLevel/reorderQty(追加订货级别/追加订货数量)属性，很难只与Clinic或Stock实体相关。在这两个例子中，应该确保没有漏掉任何实体或者将这些属性与相应的关系相联，如图17-5b所示。

应该注意，在两个实体中不应有相同的属性，属性的出现实际上是代表了实体之间的关系。例如，在17.1.1节中所给出的需求说明中，在“检查”项目中说明每次检查的详细信息包括“兽医姓名”。在这里你可能会被误导，认为同时在Staff和Examination实体中包含兽医姓名。但是，这是不正确的：在这种情况下，兽医姓名的出现代表了一个关系，不应该把它作为Examination实体的一个属性。如果你确实把它作为Examination的一个属性，将会导致Examination表不符合第三范式要求。

提示 没有经验的设计者经常犯这样的错误，一定要注意。

将属性存档

为属性命名时，应该为它们取有意义而且对用户直观的名字，在数据字典中记录它们的

详细信息，正如第9章步骤1.3中所讨论的那样。

17.2.4 步骤1.4：确定属性域

现在要为上一步在数据字典中所标识的属性添加必要的属性域。

17.2.5 步骤1.5：确定候选键、主键和备用键属性

这个步骤主要是为实体标识候选键，然后选择其中之一作为主键。在标识主键的过程中，要特别注意实体是强实体还是弱实体。

在标识候选键时，应该注意到Clinic实体的诊所号、Staff实体的员工号、Treatment实体的诊治方案号码、Invoice实体的账单号、Stock实体的项目/药品号，这些号码在整个组织中都是唯一的。另一方面，PetOwner实体的宠物主人号码、Pet实体的宠物号码、Pen实体的围栏号码，只是在某一个具体的诊所内是唯一的。一个组织给不同的办事处一定程度的地方自治权是很普遍的。但是，在中央数据库系统中，在全局范围内是唯一的就比较合适。在讨论Perfect Pets的管理需求时，每个诊所中的号码，都应该在全局范围内分配，而不是在每个具体的诊所中分配。如果不这样的话，也应该在那些号码中添加诊所号码，同时保证那些号码在每个诊所中是唯一的，如此一来，就具有全局范围内的唯一性了。

请记住，现在应该标识主键，如图17-6所示（其他备用键如图17-9所示）。尤其要注意的是，应该标识PetTreatment为弱实体。

17.2.6 步骤1.6：特化和泛化实体

当前的Stock实体的定义已经为继续进行逻辑数据库设计方法学的其他步骤作好了准备。但是，为了更准确地建模，你可能想要增加额外的信息。需求说明表明外用品和非外用品的供应有唯一的项目号来区分它们，也有唯一的药品号来区分药品的供应。此外，这两种类型供应的属性方面稍微有些不同。因此，我们可以考虑把Surgical/NonSurgical Stock和Pharmaceuticals作为Stock实体的特殊类型。此特化/泛化如图17-6所示（可选步骤）。为了简便，我们把Surgical/Non-Surgical Stock和Pharmaceuticals分别重新命名为Item和Pharmacy（特化和泛化的定义见第11章）。

你也可以标识Vet(兽医)、Nurse(护士)、Secretary(秘书)和Cleaner(清洁人员)为Staff的特殊类型。尽管这些职位都有相同的属性，但只有Vet实体参与了Performs（执行）与Examination的关系。这将是模型化Staff的一个非常有效的方法，但为了尽量保持模型简单，我们将省略掉这个特化/泛化步骤。

17.2.7 步骤1.7：检查模型的数据冗余

现在，得到了一个Perfect Pets的逻辑数据模型。但是，这个数据模型包含一些应该被删除的冗余。特别要注意的是，必须要：

- 重新检查一对一（1:1）关系。
- 删除冗余关系。

1. 一对一（1:1）关系

在图17-6中有两个1:1关系：Staff Manages Clinic和Invoice ResultsFrom Examination。但是，在这两个例子中，两个实体显然不同，因此不应该合并在一起。

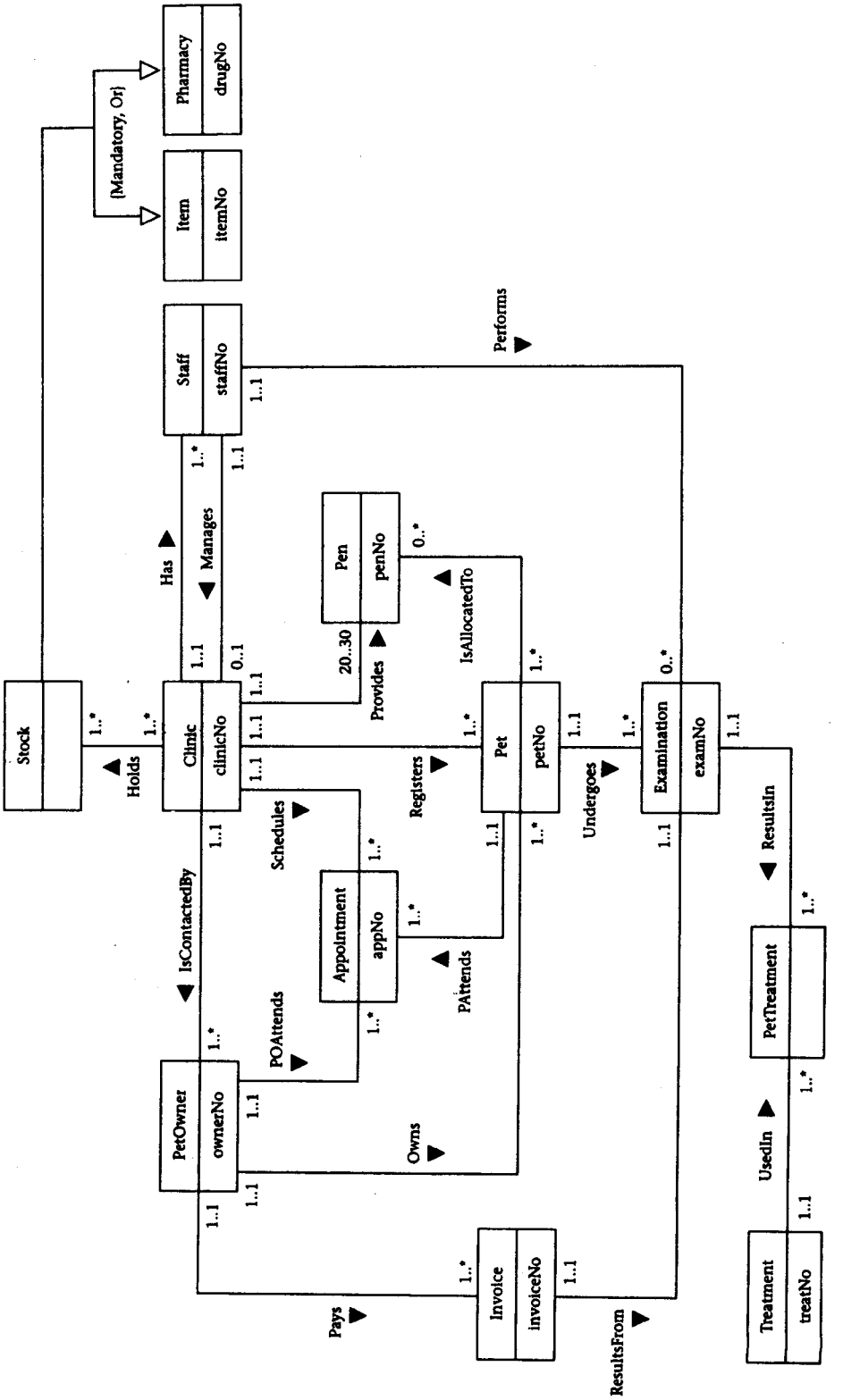


图17-6 具有主键的Perfect Pets ER模型和特化/泛化Stock

2. 冗余关系

在图17-6中，在PetOwner、Pet、Clinic和Appointment之间有一些关系，同时，最近的一次检查对于标识出任何冗余关系是很有用的。首先，注意到PetOwner/Pet实体在POAttends/PAttends /Owns关系中为强制参与，同时，由于一个PetOwner可能拥有许多宠物(Pet)。因此，对任何一个给定的Appointment，我们都通过POAttends关系来标识Owner，但不能通过Owns关系来标识Pet。然而，对任何给定的Appointment，可以通过PAttends关系来标识Pet；对任何给定的Pet，可以通过Owns关系来标识PetOwner。这就说明POAttends关系是冗余的。同样，通过PAttends关系，可以标识Pet；通过Registers关系，可以标识包含在Appointment中的Clinic。这表明Schedules关系也是冗余的。

注意到Clinic和PetOwner之间的IsContactedBy关系似乎也是冗余的。但是，当宠物主人第一次与诊所关系时，Perfect pets记录了宠物主人的详细信息，但只有当第一次预约的时候才获得宠物的详细信息，所以要留下IsContactedBy关系。修正后的逻辑数据模型如图17-7所示。

17.2.8 步骤1.8：检查模型是否支持用户事务

在这个步骤中，检查已经创建的局部逻辑数据模型是否支持用户所需的事务。检查包括如下方面：

- 数据模型中是否存在必需的属性。
- 如果属性要从多个实体中得到，则两个实体之间是否有通路；换句话说，在两个实体之间要有已经标识了的关系，不论是直接的还是间接的。

17.1.2节中所标识的查询事务的路径图如图17-8所示，你可以很容易地通过一个或多个关系检查是否可以从一个实体或从多个实体得到所必需的属性。

17.2.9 步骤2.1：创建表

在这个步骤中，从逻辑数据模型创建表达用户视图中所描述的实体和关系的表，这时，要为关系数据库使用数据库设计语言（DBDL）。

将表和外键属性存档

在步骤2.1的最后，将从逻辑数据模型创建的表的全部结构都存档。每个表的DBDL描述如图17-9所示。

17.2.10 步骤2.2：用规范化方法检查表结构

在这个步骤中，要确保上一步所建的表至少要满足第三范式（3NF）。如果发现了不满足第三范式的表，则可能表明逻辑数据模型中的某些部分是错误的，或者是从模型产生表的时候产生了错误。然而，图17-9中所标识的表确实是满足3NF的。

17.2.11 步骤2.3：检查模型是否支持用户事务

这个步骤与步骤1.8类似，在这个步骤中，除了要检查从实体到表的映射关系并且要确定外键之外，在这个情景下，还可以再次检查从实体到表的映射是否正确地完成，以及所创建的表是否支持17.1.2节中标识的用户事务。

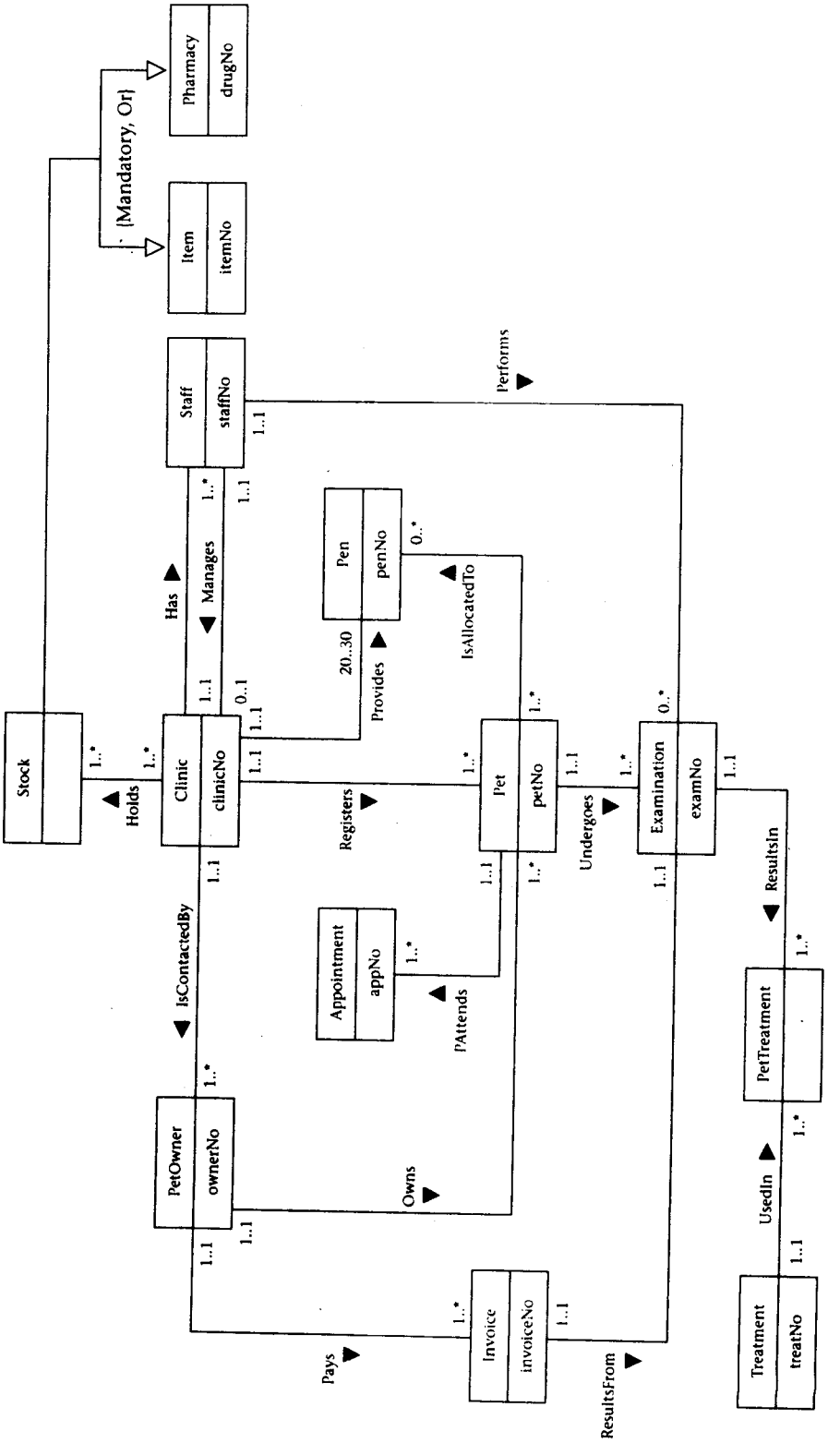


图17-7 Perfect Pets删除与关系模型不相容的特性后的局部逻辑数据模型

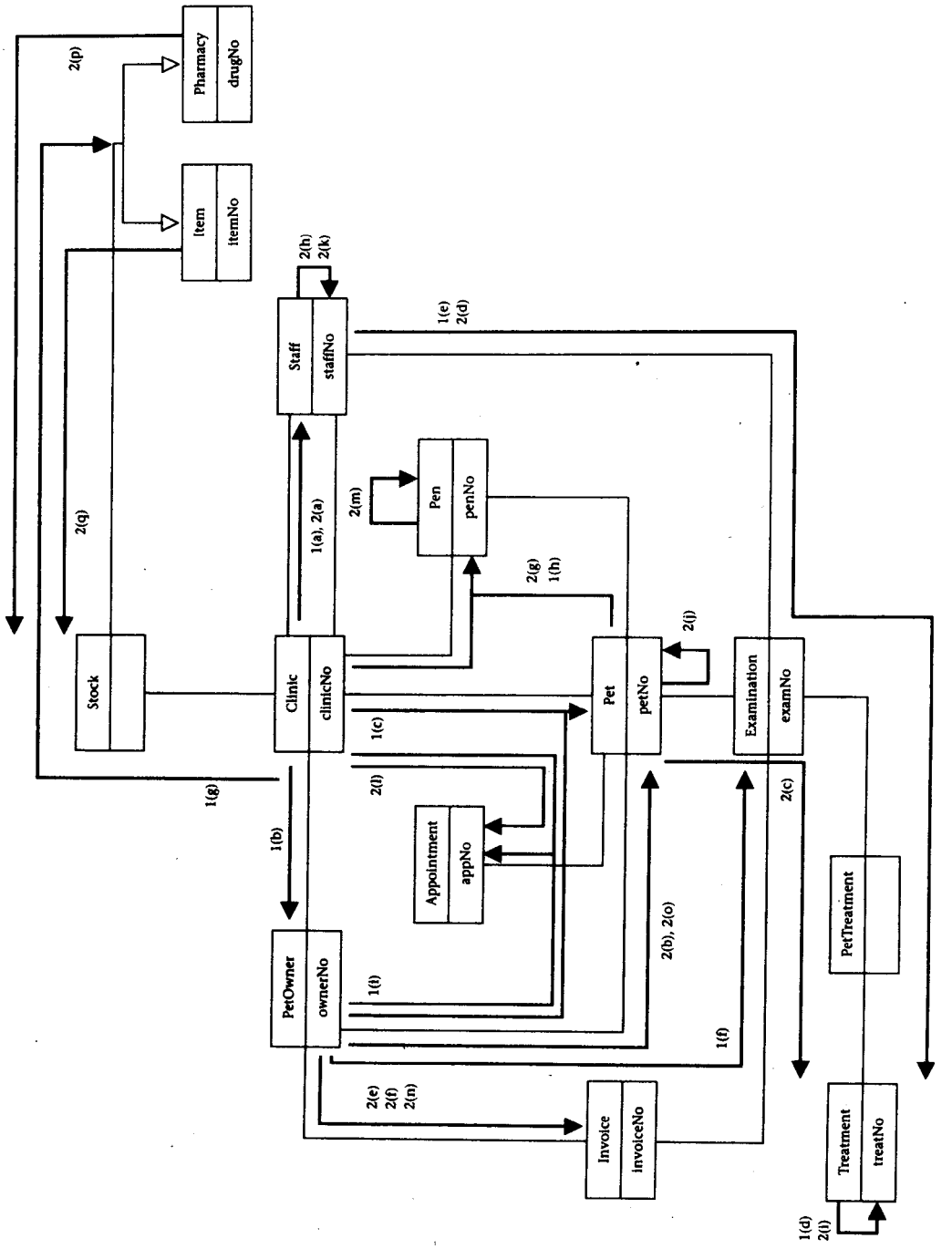


图17-8 Perfect Pets的事务路径表

Clinic (clinicNo, street, city, state, zipcode, telNo, faxNo, mgrStaffNo) **Primary Key** clinicNo **Alternate Key** zipCode **Alternate Key** telNo **Alternate Key** faxNo **Foreign Key** mgrStaffNo **references** Staff(staffNo)	**Staff** (staffNo, sFName, sLName, sStreet, sCity, sState, sZipCode, sTelNo, DOB, sex, SSN, position, salary, clinicNo) **Primary Key** staffNo **Alternate Key** SSN **Foreign Key** clinicNo **references** Clinic(clinicNo)
PetOwner (ownerNo, oFName, oLName, oState, oZipCode, oTelNo, clinicNo) **Primary Key** ownerNo **Foreign Key** clinicNo **references** Clinic(clinicNo)	**Pet** (petNo, petName, petType, petDescription, pDOB, dateRegistered, petStatus, ownerNo, clinicNo) **Primary Key** petNo **Foreign Key** ownerNo **references** Owner(ownerNo) **Foreign Key** clinicNo **references** Clinic(clinicNo)
Examination (examNo, examDate, examTime, examResults, petNo, staffNo) **Primary Key** examNo **Alternate Key** staffNo, examDate, examTime **Foreign Key** petNo **references** Pet(petNo) **Foreign Key** staffNo **references** Staff(staffNo)	**Treatment** (treatNo, description, cost) **Primary Key** treatNo
Pen (penNo, penCapacity, penStatus, clinicNo) **Primary Key** penNo **Foreign Key** clinicNo **references** Clinic(clinicNo)	**PetPen** (penNo, petNo, dateIn, dataOut, comments) **Primary Key** penNo, petNo, dateIn **Alternate Key** penNo, petNo, dateOut **Foreign Key** penNo **references** Pen(penNo) **Foreign Key** petNo **references** Pet(petNo)
PetTreatment (examNo, treatNo, startDate, endDate, quantity, ptComments) **Primary Key** examNo, treatNo **Foreign Key** examNo **references** Examination(examNo) **Foreign Key** treatNo **references** Treatment(treatNo)	**Item** (itemNo, itemName, itemDescription, itemCost) **Primary Key** itemNo
Pharmacy (drugNo, drugName, drugDescription, dosage, methodAdmin, drugCost) **Primary Key** drugNo	**ItemClinicStock** (itemNo, clinicNo, inStock, reorderLevel, reorderQty) **Primary Key** itemNo, clinicNo **Foreign Key** itemNo **references** Item(itemNo) **Foreign Key** clinicNo **references** Clinic(clinicNo)
PharmClinicStock (drugNo, clinicNo, inStock, reorderLevel, reorderQty) **Primary Key** drugNo, clinicNo **Foreign Key** drugNo **references** Pharmacy(drugNo) **Foreign Key** clinicNo **references** Clinic(clinicNo)	**Invoice** (invoiceNo, invoiceDate, datePaid, paymentMethod, ownerNo, examNo) **Primary Key** invoiceNo **Foreign Key** ownerNo **references** Owner(ownerNo) **Foreign Key** examNo **references** Examination(examNo)
Appointment (appNo, aDate, aTime, petNo) **Primary Key** appNo **Foreign Key** petNo **references** Pet(petNo)	

图17-9 从Perfect Pets逻辑数据模型创建的表

17.2.12 步骤2.4：检查业务规则

业务规则是为了防止数据库不一致而强加的约束。六种完整性约束中，有四种在上一步已经标识了，并且存档在数据字典中。这四种是：需要的数据、属性域约束、实体完整性和多样性。剩下的两种为：参照完整性和其他业务规则。

1. 参照完整性

在这里要考虑两点：

- 标识外键是否可以为空（NULL）。通常，如果关系中子表部分是强制的，那么就不允许为空。如果子表部分是可选的，那么就允许为空。
- 标识现有约束条件，表明外键的插入、更新或删除情况。通常，要为每个外键明确说明两个动作：一个是ON UPDATE动作，另一个是ON DELETE动作，用于表明当在父表中更新

或删除一条记录时，如何保证参照完整性。图17-10说明了图17-9中标识的外键所需的必要动作。

2. 其他业务规则

最后，考虑是否还有在Perfect Pets定义了但还没有在数据模型中描述的其他类型的约束。这样的约束一般被称为业务规则。

3. 将所有的业务规则存档

为了考虑物理数据库设计，应该在数据字典中记录所有的业务规则。

Clinic Foreign Key mgrStaffNo references Staff(staffNo) ON UPDATE CASCADE ON DELETE NO ACTION
Staff Foreign Key clinicNo references Clinic(clinicNo) ON UPDATE CASCADE ON DELETE NO ACTION
PetOwner Foreign Key clinicNo references Clinic(clinicNo) ON UPDATE CASCADE ON DELETE NO ACTION
Pet Foreign Key ownerNo references Owner(ownerNo) ON UPDATE CASCADE ON DELETE CASCADE Foreign Key clinicNo references Clinic(clinicNo) ON UPDATE CASCADE ON DELETE NO ACTION
Examination Foreign Key petNo references Pet(petNo) ON UPDATE CASCADE ON DELETE CASCADE Foreign Key staffNo references Staff(staffNo) ON UPDATE CASCADE ON DELETE NO ACTION
Pen Foreign Key clinicNo references Clinic(clinicNo) ON UPDATE CASCADE ON DELETE CASCADE
PetPen Foreign Key penNo references Pen(penNo) ON UPDATE CASCADE ON DELETE CASCADE Foreign Key petNo references Pet(petNo) ON UPDATE CASCADE ON DELETE CASCADE
PetTreatment Foreign Key treatNo references Treatment(treatNo) ON UPDATE CASCADE ON DELETE NO ACTION
ItemClinicStock Foreign Key itemNo references Item(itemNo) ON UPDATE CASCADE ON DELETE NO ACTION Foreign Key clinicNo references Clinic(clinicNo) ON UPDATE CASCADE ON DELETE NO ACTION
PharmClinicStock Foreign Key drugNo references Pharmacy(drugNo) ON UPDATE CASCADE ON DELETE NO ACTION Foreign Key clinicNo references Clinic(clinicNo) ON UPDATE CASCADE ON DELETE NO ACTION
Invoice Foreign Key ownerNo references Owner(ownerNo) ON UPDATE CASCADE ON DELETE NO ACTION Foreign Key examNo references Examination(examNo) ON UPDATE CASCADE ON DELETE NO ACTION
Appointment Foreign Key petNo references Pet(petNo) ON UPDATE CASCADE ON DELETE CASCADE

图17-10　Perfect Pets表的参照完整性约束

17.2.13　步骤2.5：与用户一起讨论逻辑数据库设计

现在，逻辑数据库设计已经完成并存档，这时，应该与用户一同检查模型和支持文档。我们假定在设计中并没有发现什么大的问题。到目前为止我们已完成了Perfect Pets的逻辑数据库设计。在下一章，我们将继续进行物理数据库设计阶段。

第18章 Perfect Pets——使用物理数据库设计方法学

在这一章中，我们为第17章的Perfect Pets示例继续完成物理数据库设计。为了说明某些物理的实现，我们使用Oracle 9i。正如我们在第17章的开始所提到的那样，我们建议在阅读本章前，自己按步骤做一下。然后可以对照我们例子的解决方法来检查你自己的方法。如果你不熟悉文件组织和检索，可以先阅读附录D。你会发现附录B的方法学总结也是很有帮助的。

在本章，我们要完成物理数据库设计方法学中的步骤，为Perfect Pets数据库提出合适的物理设计。

18.1 步骤3.1：设计基本表

在逻辑数据库设计阶段，创建了一些描述逻辑数据模型中的实体和关系的基本表的设计包括：

- 对于每个表，包括它的属性、主键、备用键、外键和完整性约束。
- 对每个属性，包括它的域、可选的默认值、是否可以为空和是否是派生的。

基本表的设计还包括，使用这些信息去定义域、默认值和空指示符。例如，对Perfect Pets的Pen表，可以使用扩展的数据库设计语言（DBDL）产生如图18-1所示的设计。使用这个信息可以确定在目标DBMS中如何实现基本表，在这个例子中是Oracle 9i。

```
domain Pen_Numbers          固定长度字符串，长度为4
domain Pen_Capacity         1~4之间的整数
domain Pen_Status           一个字符，表明围栏是否可获得，可获得为A，不可获得为N
domain Clinic_Numbers       固定长度字符串，长度为5
Pen(    penNo          Pen_Numbers        NOT NULL,
        penCapacity    Pen_Capacity       NOT NULL DEFAULT 2,
        penStatus      Pen_Status         NOT NULL DEFAULT 'A',
        clinicNo       Clinic_Numbers     NOT NULL)
        Primary Key penNo
        Foreign Key clinicNo References Clinic (clinicNo) ON UPDATE CASCADE ON DELETE NO ACTION
```

图18-1 Pen表的DBDL

1. 在Oracle 9i中创建基本表

在一些系统中，不能完全遵循1999 SQL标准（SQL3），不支持PRIMARY KEY、FOREIGN KEY和DEFAULT中一条或更多条子句。同样，许多系统不支持域。但是，Oracle 9i 支持SQL3中许多CREATE TABLE语句，因此可以这样定义：

- 主键，使用PRIMARY KEY子句。
- 备用键，使用UNIQUE关键字。
- 默认值，使用DEFAULT子句。
- 非空列，使用NOT NULL关键字。
- 外键，使用FOREIGN KEY子句。

• 其他列或表约束，使用CHECK和CONSTRAINT子句。

但是，尽管Oracle 9i确实允许创建用户自定义类型，但数据类型与SQL标准稍有不同，如表18-1所示。

表18-1 部分Oracle数据类型

数据类型	用 法	长 度
char(size)	存储定长字符数据（默认长度是1）	最多2000字节
nchar(size)	除最大长度由数据库字符集（例如，美式英语、东欧语言或韩语）确定之外，其他与char型相同	
varchar2(size)	存储变长字符数据	最多4000字节
nvarchar2(size)	与varchar2型相同，最大长度与nchar相同	
varchar	目前与char型相同，但建议使用varchar2，因为varchar可能变成独立的数据类型，在以后的发布中有不同的比较语义	最多2000字节
number(l,d)	存储定点或浮点数字，l代表长度，d代表小数位的长度。例如，number(5,2)型数据最大不超过999.99	1.0E－130
decimal(l,d)、dec(l,d)或numeric(l,d)	与number相同，与SQL标准兼容	
integer、int、smallint	与SQL标准兼容，转化为number(38)	
date	存储的日期从公元前4712年1月1日到4712年12月31日	
blob	一个二进制大对象	最多4GB
clob	一个字符大对象	最多4GB
raw(size)	原始二进制数据，例如图形字符或数字化图片	最多2000字节

在第12章，我们看到Microsoft Access有一个Autonumber(自动序列号)数据类型，它表示每当插入一条记录时，都为某列生成一个新的序列号。Oracle没有这样的数据类型，但它的（非标准）SQL CREATE SEQUENCE语句具有相似的功能。例如，语句:

```
CREATE SEQUENCE appNoSeq
        START WITH 1 INCREMENT BY 1 CACHE 30;
```

创建一个叫做appNoSeq的序列，初始值为1，每次加1。CACHE 30子句说明Oracle应该预先分配30个序列号，并为了提高访问速度，将他们保存在内存中。

一旦创建了序列号，就可以在SQL语句中用下述的伪列来访问它的值:

• CURRVAL 返回序列中的当前值。

• NEXTVAL 增加序列中的值，并返回新值。

例如，SQL语句:

```
INSERT INTO Appointment(appNo,adate,aTime,petNo)
VALUES(appNoSeq.NEXTVAL,SYSDATE,'12.00', '010090');
```

这条语句向Appointment表中插入一条新记录，并置appNo（预约号）列的值为序列中的下一个可用值。

2. 使用SQL*PLUS在Oracle 9i中建立空表

为了说明在Oracle中建立空表的过程，我们首先使用SQL*PLUS，它是交互的、由命令行驱动的到Oracle数据库的SQL接口。图18-2说明了如何使用Oracle SQL CREATE TABLE语句创建Pen表。

Oracle允许指定的约束有效（默认设置）和无效。在某些情况下，由于功能原因，可能要暂时使约束无效。例如:

```
SQL*Plus: Release 9.0.1.3.0 - Production on Mon Nov 18 11:12:03 2002

(c) Copyright 2001 Oracle Corporation.  All rights reserved.


Connected to:
Personal Oracle9i Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production

SQL> CREATE TABLE Pen(
  2  penNo CHAR(4) NOT NULL,
  3  penCapacity NUMBER DEFAULT 2 NOT NULL
  4     CHECK (penCapacity BETWEEN 1 AND 4),
  5  penStatus CHAR DEFAULT 'A' NOT NULL
  6     CHECK (penStatus = 'A' OR penStatus = 'M'),
  7  clinicNo CHAR(5) NOT NULL,
  8  PRIMARY KEY (penNo),
  9  CONSTRAINT clinicnumber3 FOREIGN KEY (clinicNo)
 10     REFERENCES Clinic(clinicNo));

Table created.

SQL>
```

图18-2 使用Oracle SQL CREATE TABLE语句创建Pen表

- 使用SQL*Loader向表中装载大量数据。
- 执行批处理，对表进行很大量的更改。
- 同时对一张表进行导入和导出。

默认时，Oracle强制在定义的外键上保证参照完整性，因此，它强制执行参照动作ON DELETE NO ACTION和ON UPDATE NO ACTION。同时，它也允许指定附加的ON DELETE CASCADE子句（允许从父表中删除数据时级联删除子表中相应数据）和ON DELETE SET NULL子句（允许从父表删除数据，并将子表中相应的外键值置为NULL）。但它不支持ON UPDATE CASCADE动作和SET DEFAULT动作，如果需要这些动作，则必须考虑用触发器实现或者用应用程序代码实现。我们将在步骤3.3对此做简单考虑。

3. 使用表向导来建表

在Oracle 9i中可以使用“表向导”（Table Wizard）来建表，它是企业管理控制台（Enterprise Manager Console）的一部分。使用交互界面，“表向导”引导你一步步完成对每个列的定义，包括数据类型、定义列上的约束和表上的约束以及关键字域。图18-3显示了“表向导”在创建Treatment表时的最终形式。

4. 将基本表设计存档

基本表设计也应该将表以及选择某设计的原因存档，尤其是当存在多个选择时，将选择其中一种方法的原因存档。

18.2 步骤3.2：设计派生数据的表示

17.1.1节给定的需求表明只有一个派生项，也就是账单上要记录的给宠物进行所有诊治的总费用。获得该信息的算法可以写成如下SQL语句。

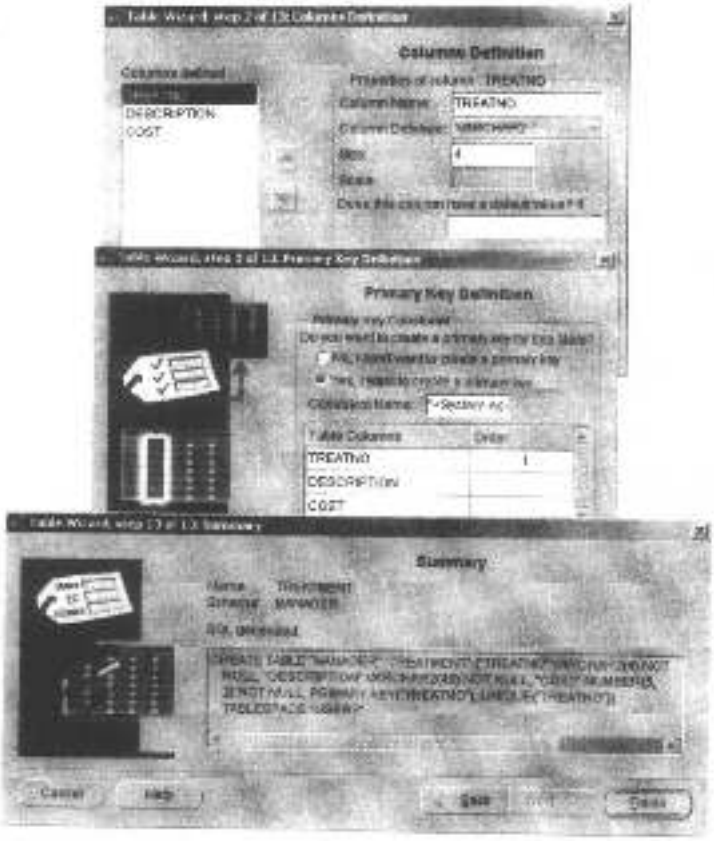


图18-3　使用Oracle创建表向导创建Treatment表

```
SELECT SUM(pt.quantity*t.cost)
FROM Invoice i , Examination e, PerTreatment pt, Treatment t
WHERE i.examNo=e.examNo AND e.examNo=pt.examNo AND
      pt.treatNo=t.treatNo;
```

如果经常访问Invoice表，那么将总费用存在Invoice表中可以提高系统性能。但是，从预期的频度来看(参见本章后边的表18-2)，并不是经常要访问Invoice表，在这种情况下，只要在需要的时候计算总费用就可以了。

18.3　步骤3.3：设计其他业务规则

有时，更改表可能会受业务规则的限制。这些规则的设计也依赖于目标DBMS，有些系统在定义规则方面比另一些系统提供了更多的功能。在第12章中，我们看到如果系统遵循1999 SQL标准，则某些规则可能更容易实现。正如我们在前面所看见的，Oracle 9i允许将定义约束的CHECK和CONSTRAINT子句作为SQL CREATE TABLE语句中的一部分，也允许使用before triggers（前触发器）和after triggers（后触发器）定义其他约束。为了更灵活，Oracle 9i允许创建存储过程，并可以在SQL中调用它。

例如，图18-1中的Pen表的外码clinicNo，就应该有ON UPDATE CASCADE动作。但是，正像我们已经注意到的，Oracle 的CREATE TABLE语句不支持该动作。但是，可以使用触发器来实现该动作，如图18-4所示。

```
--Before the clinicNo column is updated in the Pen table, fire this trigger
--to verify that the new foreign key value is present in the Clinic table.
CREATE TRIGGER Pen_Clinic_Check_Before
   BEFORE UPDATE OF clinicno ON pen
   FOR EACH ROW WHEN (new.clinicno IS NOT NULL)
DECLARE
   dummy CHAR(5);
   invalid_clinic EXCEPTION;
   valid_clinic EXCEPTION;
   mutating_table EXCEPTION;
   PRAGMA EXCEPTION_INIT (mutating_table,-4091);
--Use cursor to verify parent key value exists.
--Use FOR UPDATE OF to lock parent key's record so that it cannot be deleted
--by another transaction before this transaction completes.
CURSOR update_cursor (sn CHAR(5)) IS
   SELECT clinicno FROM clinic
   WHERE clinicno = sn
   FOR UPDATE OF clinicno;
   BEGIN
         OPEN update_cursor (:new.clinicno);
         FETCH update_cursor INTO dummy;
--Verify parent key. Raise exceptions as appropriate.
         IF update_cursor%NOTFOUND THEN
               RAISE invalid_clinic;
         ELSE
               RAISE valid_clinic;
         END IF;
         CLOSE update_cursor;
   EXCEPTION
   WHEN invalid_clinic THEN
         CLOSE update_cursor;
         raise_application_error(-20000, 'Invalid Clinic Number'||:new.clinicno);
   WHEN valid_clinic THEN
         CLOSE update_cursor;
--A mutating table is a table that is currently being modified by an INSERT, UPDATE,
--or DELETE statement, or one that might need to be updated by the effects of a declarative
--DELETE CASCADE referential integrity constraint.
--This error would raise an exception, but in this case the exception is OK, so trap it,
--but don't do anything.
   WHEN mutating_table THEN
         NULL;
   END;
```

Before触发器

触发触发器的条件

行级触发器

指定的clinicNo无效

a) Pen表的触发器

```
--Create a sequence number and a public variable UPDATESEQ.
CREATE SEQUENCE updatesequence INCREMENT BY 1 MAXVALUE 500 CYCLE;
CREATE PACKAGE seqpackage AS
   updateseq NUMBER;
END seqpackage;
CREATE or REPLACE PACKAGE BODY seqpackage AS END seqpackage;

--Add a new column to the Pen table to flag changed records.
ALTER TABLE pen ADD updateid NUMBER;

--Before updating the Clinic table using this statement trigger, generate a new
--sequence number and assign it to the public variable UPDATESEQ.
CREATE TRIGGER Cascade_ClinicNo_Update1
   BEFORE UPDATE OF clinicno ON clinic
   DECLARE
         dummy NUMBER;
```

存储序列号的包

给Pen表添加额外的列

语句级的Before触发器

图18-4 当Clinic表中的主键更新时，Oracle触发器强制Pen表中的外键clinicNo执行ON UPDATE CASCADE动作

```
        BEGIN
              SELECT updatesequence.NEXTVAL
              INTO dummy FROM dual;
              seqpackage.updateseq :=dummy;
        END;

--Create a row after-trigger that cascades the update to the Pen table.
--Only cascade the update if the child row has not already been updated by the trigger.
CREATE TRIGGER Cascade_ClinicNo_Update2
   AFTER UPDATE OF clinicno ON clinic
   FOR EACH ROW
   BEGIN
         UPDATE pen SET clinicno = :new.clinicno, updateid = seqpackage.updateseq
         WHERE pen.clinicno = :old.clinicno AND updateid IS NULL;
   END;

--Create a final statement after-trigger to reset the updateid flags
CREATE TRIGGER Cascade_ClinicNo_Update3
   AFTER UPDATE OF clinicno ON clinic
   BEGIN
         UPDATE pen SET updateid = NULL
         WHERE updateid = seqpackage.updateseq;
   END;
```

为更新设置新的序列号

行级的After触发器

为这些记录更新Pen表并设置更新标志

语句级的After触发器。重新设置更新后记录的标志

b) Clinic表的触发器

图18-4 （续）

1. 触发器1（Pen_Clinic_Check_Before）

每当更新Pen表中的clinicNo列时，都会激活图18-4a中的触发器。在更新发生之前，触发器检查指定的新值是否存在于Clinic表中。如果引发了Invalid_Clinic异常，触发器就发出一个错误信息，并防止更新的发生。应该注意下面的几点：

- BEFORE关键字表明触发器应该在更新Pen表中的clinicNo列之前执行。
- FOR EACH ROW关键字表明这是一个行级触发器，表明在一个事务中更新Pen表的每一行时，都要执行该触发器。与此对应的触发器的类型还有语句级触发器，它对每个事务执行一次。我们将很快看到语句级触发器的例子。
- WHEN子句指明激活触发器的条件。
- new关键字用于指向该列的新值，old关键字用于指向该列的旧值。

2. 修改Clinic表以支持触发器

每当更新Clinic表中的clinicNo列时，都会激活图18-4b所示的三个触发器。在定义触发器之前，创建了一个序列号——updateSequence（更新序列）以及一个公共变量updateSeq（通过seqPackage包，三个触发器可以访问该变量）。另外，修改Pen表增加一个叫做updated的列，用该列作为是否已更新记录的标志，以防止在级联操作中多次更新记录。

3. 触发器2（Cascade_ClinicNo_Update1）

在更新Clinic表的ClinicNo列之前，激活语句级触发器Cascade_ClinicNo_Update1，设置新的用于更新的序列号。

4. 触发器3（Cascade_ClinicNo_Update2）

激活行级触发器Cascade_ClinicNo_Update2，用于将Pen表中所有含有旧值的记录中旧的

clinicNo值（:old.clinicno）更新为新值(:new.clinicno)，并且标志已经更新该记录。

5. 触发器4（Cascade_ClinicNo_Update3）

更新后，激活最后一个语句级触发器Cascade_ClinicNo_Update3，将更新记录的标志恢复为没有更新。

> 不用太关心触发器的工作细节。最应该注意的是，实现这些动作所必需的大量的编程工作。换句话说，想一想，如果DBMS提供了这些功能，可以节省多少工作。

18.4 步骤4.1：分析事务

创建了基本表、完整性约束和业务规则后，下一步就是分析事务为每个基本表确定合适的文件组织方式和索引。假定17.1.2节所标识的事务Perfect Pets是最重要的事务。为了集中在可能会出现问题的区域，我们在第13章中建议的继续进行的一个方法是:

1）将所有的事务路径映射到表。

2）确定最常被事务访问的表。

3）分析涉及到这些表的事务。

在步骤1.8和2.3中，已经完成了第一步（见图17-9）。为了实现第二步，需要估计各表的访问频率。如果可能，应该把频率信息加入到事务路径图中。但这有时会使该图混乱而难于解释，所以你可能更想单独保存这些频率信息。在讨论了Perfect Pets中的员工后，得到的频率信息如图18-5所示。

此外，必须估计每个事务可能的频率。分析所有这些信息以标识需要特殊考虑的地方。

18.5 步骤4.2：选择文件组织方式

如果目标DBMS允许，这步的目标就是为每张表选择最佳的文件组织方式。为了完成此步骤，需要理解目标DBMS在逻辑和物理两层上是如何操作的。在这个步骤中，我们查看Oracle是如何存储数据的。这实际上是技术上的讨论，但是可以帮你了解例子中这步所需的知识的类型。

1. Oracle的逻辑数据库结构

在逻辑层，Oracle维护表空间、模式、数据块和区（extent）/段（segment），下面我们来解释它们。

(1) 表空间

Oracle数据库被划分成逻辑存储单元，称为表空间。表空间用于把相关的数据结构组织在一起。例如，表空间通常把应用中的所有对象组织在一起，以简化一些管理操作。

每个Oracle数据库都包含一个叫做SYSTEM的表空间，它是数据库创建时自动生成的，SYSTEM表空间总是包含整个数据库的系统目录（在1.2.1节定义）表。一个小型数据库可能只需要SYSTEM表空间，但是，建议你至少要创建一个表空间来单独存储用户的数据，以减少对于相同数据文件（见图16-1）中字典对象和模式对象之间的争用。图18-6说明了一个包含SYSTEM表空间和USER_DATA表空间的Oracle数据库。

可用CREATE TABLESPACE命令创建一个新的表空间。例如:

```
CREATE TABLESPACE USERS
DATAFILE 'DATA3.ORA' SIZE 100K;
```

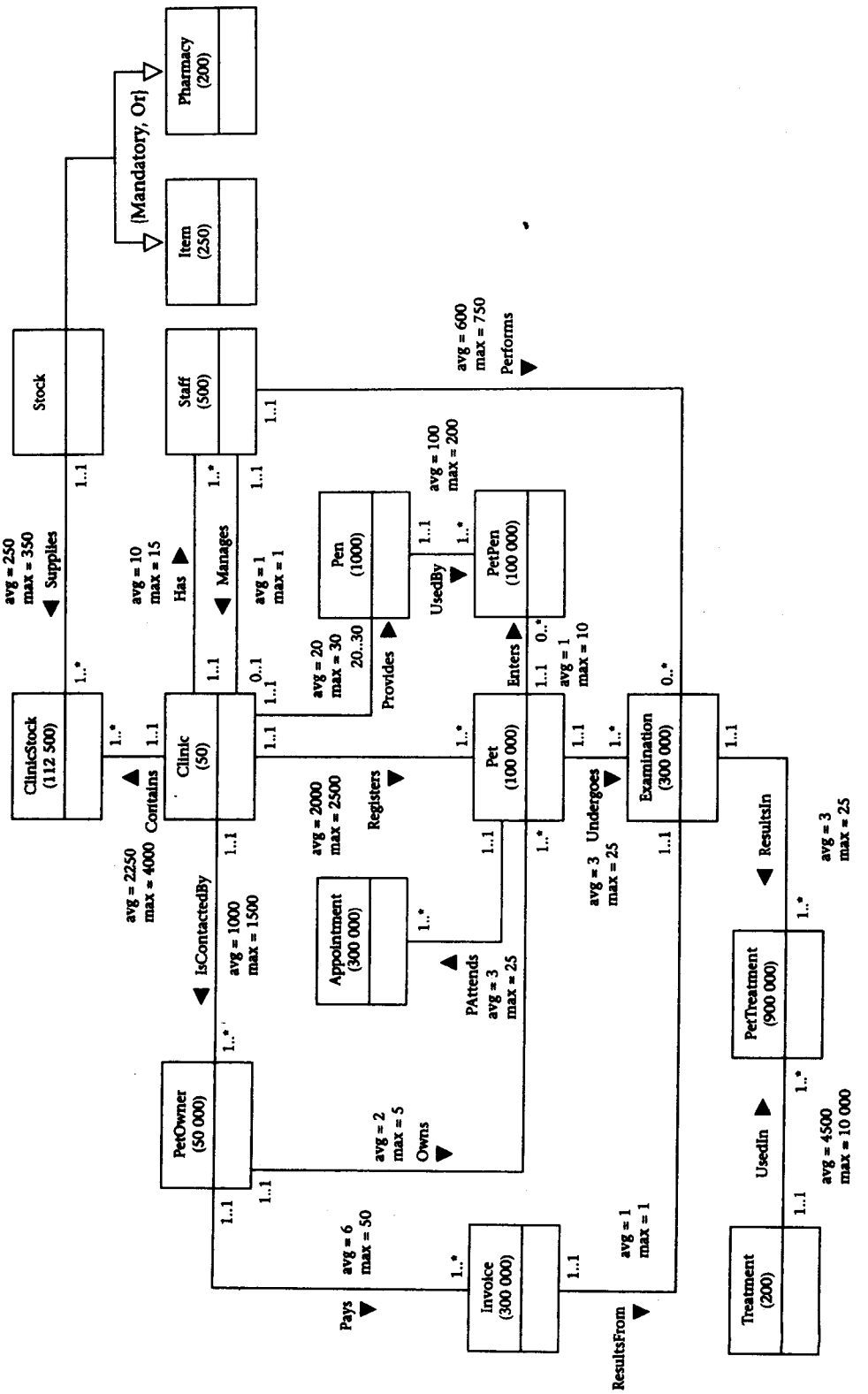


图18-5 说明预期事件的Perfect Pets的逻辑数据模型

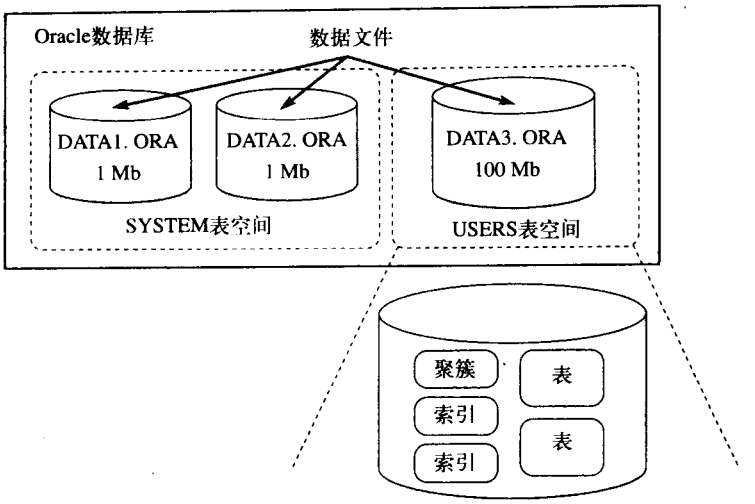


图18-6 一个Oracle数据库、表空间和数据文件之间的关系

用CREATE TABLE或ALTER TABLE语句可使一张表与某一表空间联系起来。例如:

```
CREATE TABLE Pen(penNo CHAR(4) NOT NULL,...)
   TABLESPACE USERS;
```

如果创建新表时没有指明表空间，就使用用户账号创建时与用户相关联的默认表空间。我们将在步骤6中看到如何指定用户账号的表空间。

(2) 用户、模式和模式对象

用户（user）（有时称为用户名username）是在数据库中定义的一个名字，该名字可以连接和访问对象。模式（schema）是对象的一个命名的集合，例如与某特定用户相关联的表、视图、聚簇和过程等。模式和用户帮助DBA(数据库管理员)管理数据库安全。

为了访问数据库，用户必须运行一个数据库应用（比如Oracle窗体或SQL*PLUS），并且用数据库中已定义的用户名与数据库进行连接。当创建了一个数据库用户时，就为用户创建了一个同名的相应模式。默认时，一旦用户连接到数据库，这个用户就可以访问相应模式中所包含的全部对象。由于用户只与同名的模式相联，因此术语用户和模式经常是可以替换的。

注意 *在表空间和模式之间没有必然的关系，同一模式中的对象可以位于不同的表空间，表空间也可包含来自不同模式的对象。*

(3) 数据块、区和段

数据块是Oracle使用和分配的最小存储单位。一个数据块对应物理磁盘空间上的若干字节。创建数据库时，为每个Oracle数据库设置数据块的大小。这个数据块大小应该是操作系统块大小（不超过最大的系统操作限制）的整数倍，以避免多余的I/O操作。

逻辑数据库空间的上一层称为区（extent）。区是为了存储特定类型的信息而分配的若干连续数据块。区的上一层称为段（segment）。段是为某一逻辑结构分配的一组区。例如，每个表的数据存储于自己的数据段中，而每个索引的数据存储在它们自己的索引段中。图18-7显示了数据块、区和段之间的关系。

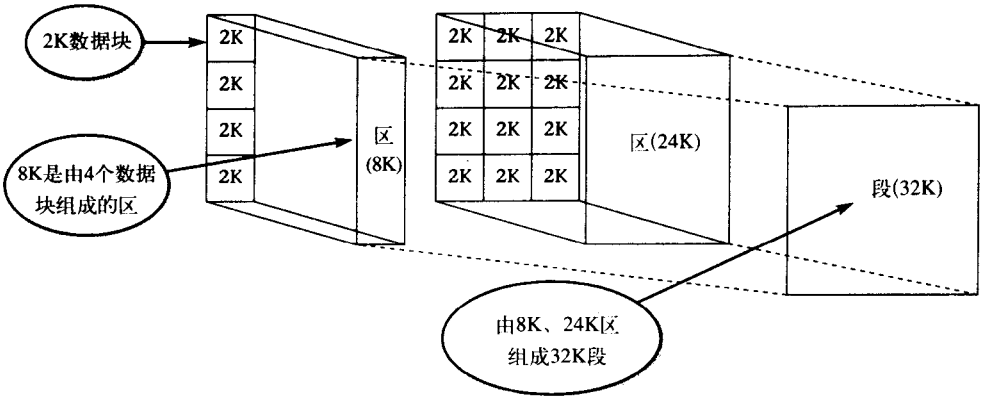


图18-7 数据块、区和段之间的关系

当一个段的已有区存储满时，Oracle会动态地再分配空间。因为区是按需要进行分配的，因此一个段的区在磁盘中有可能连续，也有可能不连续。

2. Oracle的物理数据库结构

Oracle中主要的物理数据库结构就是数据文件（datafile）、重做日志文件（redo log file）和控制文件(control file)。

(1) 数据文件

每个Oracle数据库都有一个或多个物理数据文件。逻辑数据库结构（如表和索引）的数据物理上存储在这些数据文件中。一个或多个数据文件构成了表空间。最简单的Oracle数据库有一个表空间和一个数据文件。比较复杂的数据库可能有4个表空间，每个表空间由两个数据文件组成，也就是共有8个数据文件。数据文件和表空间的结构如图18-6所示。

(2) 重做日志文件

每个Oracle数据库都有两个或多个重做日志文件，这些文件记录了对数据进行的所有更改，从而便于恢复数据。如果发生错误，则修改的数据没有永久地写入数据文件，这时可以从重做日志文件得到要进行的修改，从而防止操作丢失。

(3) 控制文件

每个Oracle数据库都有一个控制文件，该文件包含指定数据库物理结构的入口，例如:

- 数据库名称。
- 数据库的数据文件和重做日志文件的名字和位置。
- 数据库创建的时间戳。

3. PCTFREE和PCTUSED

这两个空间段参数——PCTFREE和PCTUSED，对性能也有很大的影响。当创建或修改表或群集（有自己的数据段）时，可以指定这些参数。当创建或修改索引（有自己的索引段）时，也可以指定存储参数PCTFREE。这两个参数的使用如下:

- PCTFREE: 设置数据块中预留的空白空间的最小百分比，此空白空间是用于更新已在数据块中的记录的（默认值为10）。
- PCTUSED 在新记录被插入块中之前，设置用于记录用户数据以及所有Oracle需要的费用的块空间的最小百分比（默认为40）。当数据块中存储的数据达到PCTFREE所限定的界限后，Oracle认为该数据块不能再插入新的记录，直到该数据块存有数据的部分所占

的百分比下降到低于参数PCTUSED所规定的百分比，才可以再插入新的记录。

PCTFREE的值越低，为更新现有记录而预留的空间就越少，同时也就允许插入更多新的记录。这样，虽然节省了空间，但却增加了处理上的耗费。因为，当块中的自由空间被新插入的或更新后的记录填满后，就要重新组织该块。PCTUSED的值越低，数据库中空白的空间就越大，但是减少了插入/更新操作时的处理代价。

很明显，PCTFREE和PCTUSED的和不能超过100。如果它们的和小于100，那么平衡空间和I/O使用的最佳配置是这两个参数的和，它不同于被记录占用的百分百空间百分比。例如，如果块的大小是2048字节，其中有100个字节作为块的头，记录的大小是390个字节，则记录的大小是可用块大小的20%。那么，为了充分利用空间，PCTFREE和PCTUSED的和最好为80%。另一方面，如果它们的和等于100，则Oracle将尽力保持PCTFREE大小的自由空间，这将会导致最高的处理代价。PCTFREE和PCTUSED的组合使用如图18-8所示。

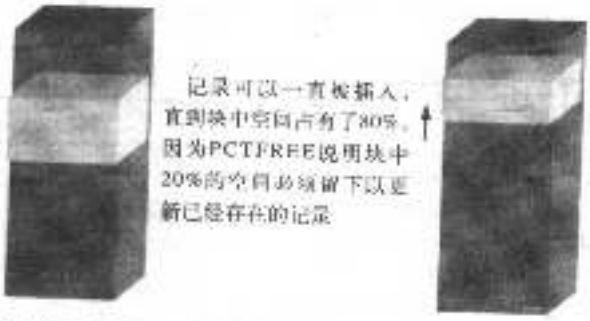


图18-8 PCTFREE和PCTUSED的组合使用，其中PCTFREE=20%、PCTUSED=40%

以上关于Oracle的逻辑和物理数据库结构的描述并不是很详细，然而，对它们的介绍是为了让你对目标DBMS方面的知识有一个了解，这些知识是成功地进行物理数据库的某些方面的设计所需要的。我们同时也强调了不同系统之间的区别，例如，一边看这一步，一边翻回到第13章步骤4.2，那里是使用Microsoft Access来处理的。你将看到，在那一步中，其实什么也没做——Microsoft Access 97中的文件组织方式是固定的，对于PC RDBMS来说，这是很正常的。如果现在来研究其他的多用户RDBMS，如Ingres或SQL Server，你会发现这些系统的逻辑和物理结构又有不同。

文件组织方式

Oracle 9i支持聚簇和非聚簇的表。选择使用聚簇的还是非聚簇的表依赖于前面对事务的分析，但这个选择会对性能有影响。在考虑合适的聚簇之前，忽略掉一些小表是个很好的办法，因为小表通常可以全部调入内存进行处理。从图18-5中，我们可以看到Perfect Pets数据库中的小表是Clinic、Staff、Pen、Treatment、Item和Pharmacy，因此我们不对这些表进行进一步考虑。如果我们现在考虑剩下的表，可以产生17.1.2节列出的查询事务与基本表之间的相互作用的总结，如表18-2所示。基于附录D.7给出的指南，我们决定为PetOwner表和Pet表创建一个基于连接列ownerNo的聚簇索引。

18.6 步骤4.3：选择索引

Oracle自动为每个主键添加索引。另外，Oracle推荐不要显式地在表上定义UNIQUE索引，

而是在需要的列上定义UNIQUE完整性约束。Oracle是通过在唯一键上自动定义唯一性索引来保证UNIQUE完整性约束的。这个推荐的例外通常与性能有关。例如，用了UNIQUE约束的CREATE TABLE... AS SELECT要比创建表时没有约束，然后手工创建UNIQUE索引慢。

让我们假定使用标识的主键、备用键以及指定的外键来创建表。现在我们必须要做的是确定是否需要任何附加的索引。在第13章，我们建议创建一个愿望列表（wish-list），然后在愿望列表中考虑每个潜在的索引，以确定查询性能的提高是否超过了更新发生时性能的下降。在建立愿望列表之前，忽略掉一些小表（Clinic、Staff、Pen、Treatment、Item和Pharmacy）是个很好的办法，因为小表通常可以全部调入内存进行处理，而不需要额外的索引。现在我们考虑剩下的表和它们的相互作用，如表18-2所示。表18-3显示了添加索引可能带来的好处。

注意　事务2(e)(datepaid IS NULL)使用的查询条件表明要在datePaid列上创建索引。但是，当查询条件包含IS NULL或IS NOT NULL条件时，Oracle并不使用索引。还有，由于事务2(j)每天只运行一次，并且并不是有很多宠物类型，所以Pen表中的petType列上的索引并不恰当。

表18-2　表和查询事务键的相互作用

表	事　务	访　问	频度（每天）
Appointment	2(1)	join: Pet on petNo search condition: aDate	250
Examination	2(c), 2(d)	join: Pet on petNo	100
	2(d)	join: Staff on staffNo	
Invoice	2(e), 2(f)	join: PetOwner on ownerNo search condition: datePaid IS NULL	10
	2(n)	join: PetOwner on ownerNo search condition: invoiceDate	1(每月)
ItemClinicStock	2(q)	search condition: InStock < reorderLeve	50(每月)
Pet	2(b)	join: PetOwner on ownerNo	1(每月)
	2(j)	group: petType order by : petType aggregate: count on petType	1
	2(l)	join: Clinic on clinicNo	250
	2(o)	join: PetOwner on ownerNo	1500
PharmClinicStock	2(p)	search condition: inStock < reorderLevel	50(每月)

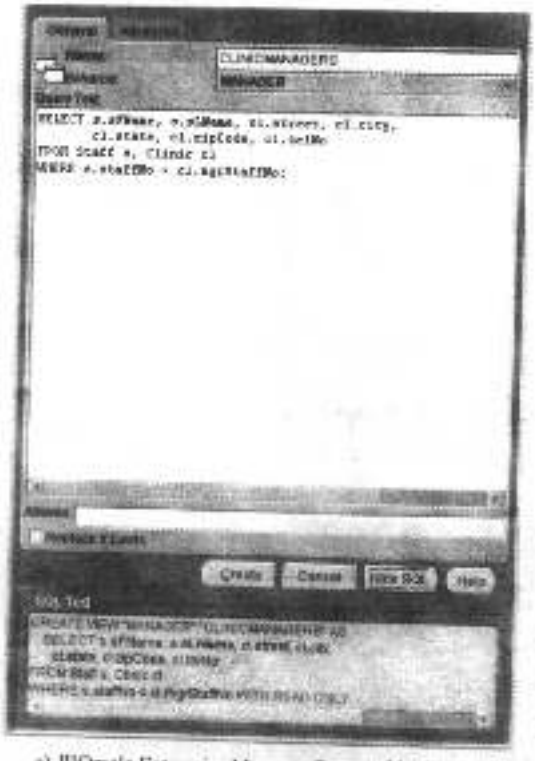
表18-3　Perfect Pets数据库的其他索引

表	索引
Pet	clinicNo ownerNo
Appointment	aDate petNo
Invoice	ownerNo invoiceDate

18.7　步骤5：设计用户视图

Oracle 9i支持SQL CREATE VIEW语句，所以为每个用户创建视图是很容易的。另外，使

用Oracle Forms Builder(Oracle窗体构造器）可以创建基于一个或多个表或基于视图的窗体。例如，可以为诊所经理的详细信息创建一个视图。图18-9a说明了使用Schema Manager创建一个叫做Clinic_Managers的视图，而图18-9b显示了从这个视图构建的窗体。



a) 用Oracle Enterprise Manager Console创建视图

b) 从这个视图构建的表单

图18-9 创建用户视图

18.8 步骤6：设计访问规则

作为数据库分析阶段的一部分，需要确定用户的类型，也就是谁将使用系统，以及为完成他们指定的任务所必须赋给他们的访问级别。正如我们在第14章步骤6中所提到的，数据库安全性通常既包括系统安全又包括数据安全。Oracle使用的一种系统安全形式是标准的用户名和密码机制，尽管认证用户的职责可以交给操作系统，但在这种形式中，在获得访问数据库的权力之前，用户必须提供合法的用户名和密码。图18-10说明了使用密码认证机制创建一个叫做ADAMS的新用户。当用户ADAMS想要连接数据库的时候，将会弹出一个类似图18-11的连接(Connect)或登录（Log On）对话框，用户填入用户名和密码以访问指定的数据库。

权限

权限是执行特定类型的SQL语句或访问其他用户的对象的权利。一些权限的例子包括：

- 连接到数据库（创建一个会话）。
- 创建表。
- 从其他用户的表中查询行。

将权限授权给用户后，这些用户就可以完成他们需要的任务。过度授权会降低安全性，因此应该只授权给绝对需要该权限完成工作的用户。在Oracle中，有两种不同的权限：

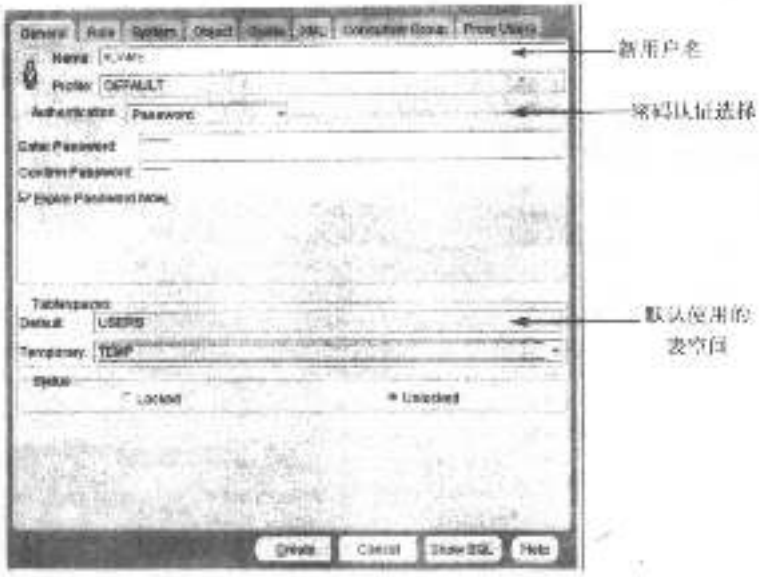


图18-10　用密码认证机制创建的新用户ADAMS

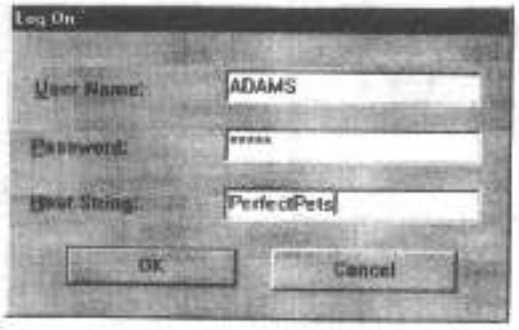


图18-11　要求用户名、密码和要连接的数据库的连接对话框

• 系统权限

• 对象权限

(1) 系统权限

系统权限就是执行特定的动作或实现在特定类型的任何模式对象上的动作的权利。例如，创建表空间和在数据库中创建数据库就是系统权限。总共有80多个系统权限。要授权给用户和角色（角色的定义稍后讨论）或从用户和角色收回系统权限，使用如下方法之一：

• Oracle安全管理的授权系统权限/角色对话框和收回系统权限/角色对话框。

• SQL GRANT和REVOKE语句。

但只有用ADMIN OPTION被授予一定权限的用户或具有GRANT ANY PRIVILEGE系统权限的用户才能授予或撤销系统权限。

(2) 对象权限

对象权限是在具体表、视图、序列、进程、函数或包上完成特定动作的权限或权利。不同的对象具有不同的对象权限。例如，从Pen表中删除行的权限就是对象权限。

一些模式对象（例如，聚簇、索引和触发器）并没有相关的对象权限，它们的使用是由

系统权限控制的。例如，如果要更改聚簇，那么用户必须自己拥有该聚簇，或者拥有ALTER ANY CLUSTER系统权限。

用户自动拥有包含在自己模式中的模式对象的所有对象权限。用户可以将自己拥有的任何模式对象的任何对象权限授权给其他用户或角色。如果授权时GRANT语句包含WITH GRANT OPTION，那么被授权者以后也可以将该对象权限授权给其他用户，否则被授权者可以使用该权限但却不能再将权限授权给其他用户。

表和视图的对象权限如表18-4所示。

(3) 角色

一个用户可以以两种不同的方式获得权限。

• 显式授权给用户。例如，可以明确地将向Clinic表中插入记录的权限授予用户ADAMS。例如：

```
GRANT INSERT ON clinic TO ADAMS;
```

表18-4 每种对象权限允许权限拥有者对表和视图所做的操作

对象/权限	表	视图
ALTER	用ALTER TABLE语句改变表定义	N/A
DELETE	用DELETE语句从表中删除行。注意：授予表DELETE权限时必须同时授予该表的SELECT权限	用DELETE语句从视图中删除行
INDEX	用CREATE INDEX语句创建表的索引	N/A
INSERT	用INSERT语句为表插入新行	用INSERT语句为视图插入新行
REFERENCES	创建引用表表的约束。不能将该权限授予角色	N/A
SELECT	用SELECT语句查询表	用SELECT语句查询视图
UPDATE	用UPDATE更改表中的数据。注意，授予表UPDATE权限时必须同时授予该表的SELECT权限	用UPDATE改变视图中的数据

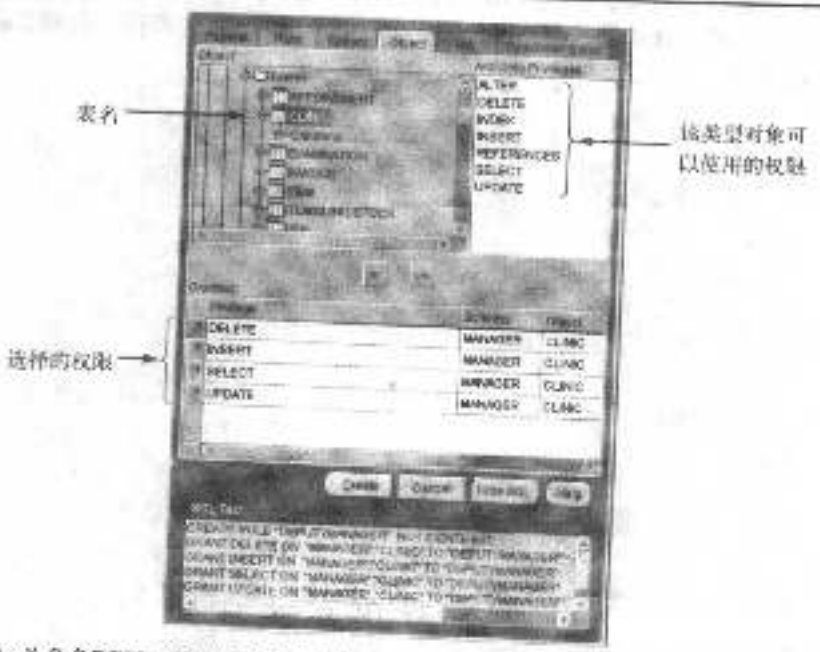


图18-12 为角色DEPUTYMANAGER设置Clinic表的Select、Insert、Update和Delete权限

- 也可以将权限授予一个角色（一个命名的权限组），然后将该角色授予一个或多个用户。例如，将对Clinic表的查询、插入、更新和删除权限授予角色DEPUTYMANAGER,然后就可以将用户ADAMS和GLENN添加到角色中。一个用户可以属于几个角色，几名用户也可属于同一个角色。图18-12说明了使用Oracle Security Manager（安全管理），将权限授予角色DEPUTYMANAGER的例子。

提示 因为角色可以更简便有效的管理权限，所以应该尽量把权限授予角色，而不是某个具体的用户。

18.9 步骤7：考虑引入受控冗余

在步骤1.7中，关系Clinic Schedules Appointment被认为是冗余的，但这会引起潜在的性能问题。例如，为了确定在某个诊所可获得的预约时间，就必须要访问Pet表。在这种情况下，恢复Schedules关系并将Clinic表的主键（clinicNo）加到Appointment表中作为外键可能是更可取的。

考虑隐式地反规范化

在Appointment表中拥有附加的clinicNo列，可能对在这个列上创建索引以提高事务2（1）的性能是值得的。

(4) 实现

现在，已经完成了基本表、文件组织方式、索引、视图和安全机制的构建与制定，可以开始操作数据库了。但是，正如第16章所讨论的，这并不是数据库设计的结束——为保证系统运行正常，不断地监视与调整操作系统是很重要的活动，这可以获得系统的持久的成功。另外，一旦系统在运行，作为用户反馈和更改需求的结果，有些更改是需要的。有时，这些变化只不过是表面的，只是要求改变用户界面，而不影响数据库本身。但有时，也需要修改数据库的结构。这时，就必须重新进行一些逻辑和物理设计中的步骤，以确保正确地设计和实现这些更改。

第六部分 数据库的现状和未来趋势

第19章 数据库的现状和发展

本章主题：

- 高级数据库应用的需求。
- 为什么关系DBMS目前不能很好地支持高级数据库应用。
- 分布式DBMS（DDBMS）的主要概念。
- 与数据库复制有关的主要概念。
- 面向对象的DBMS（OODBMS）和对象关系DBMS（ORDBMS）的主要概念。
- 数据仓库的主要概念。
- 联机分析处理（OLAP）和数据挖掘的主要概念。
- 将数据库集成到网络环境中的方法。

为了使本书更完整，我们编写了本章来讨论数据库系统的现状和未来的趋势。我们讨论的很多主题本身就是一个重要的研究领域，我们仅仅提供了它们的一个简要说明。如果读者对这些领域感兴趣，建议参考2002年Connolly和Begg合作出版的书。本章将涉及到以下内容：

- 高级数据库应用。
- 目前的关系DBMS的缺陷。
- 分布式DBMS和复制服务器。
- 面向对象的DBMS和对象关系DBMS。
- 数据仓库。
- OLAP和数据挖掘。
- 网络数据库集成和XML。

19.1 高级数据库应用

在过去的十年里，计算机行业发生了翻天覆地的变化。在数据库系统方面，关系DBMS已经在传统的商业应用中得到了广泛的应用，这些应用包括订单处理、库存控制、银行业和航空机票预订等。但是，对于那些应用需求与传统商业数据库应用有很大不同的应用领域，现有的RDBMS的功能是不够的。这些应用包括：

- 计算机辅助设计。
- 计算机辅助制造。
- 办公室信息系统和多媒体系统。
- 地理信息系统。
- 交互式、动态的Web站点。

1. 计算机辅助设计（CAD）

CAD数据库存储了与诸如建筑、航空、集成电路芯片这样的机械电气设计数据。这类设计有一些共同特征:

- 设计数据总是被分为很多类，每类有少量的实例。传统的数据库与之正好相反。例如，StayHome数据库仅仅由几十个表组成，尽管VideoForRent、Member、RentalAgreement这样的表中包含了几千条数据。
- 设计工作量非常大，可能有几百万个部分组成，常常要设计多个独立的子系统。
- 设计不是静态的，而是随着时间推移而发展的。当设计发生改变时，与它有关的内容也将随之改变。设计的动态性意味着在一开始无法预见到某些操作。
- 由于拓扑或是功能的关联、容错性等原因，更新操作的影响范围较大。一个更改很可能影响大量设计对象。
- 很多设计方案常常是为了设计出一个组件，而且必须为每个部分维护正确的版本，这涉及到一些版本控制和配置管理。
- 参与设计的员工可能会有几百人，而且他们可能在一个大型设计的多个版本中并行工作。尽管如此，最终的产品必须是一致的、协调的。这有时候被称为“协作工程”。

2. 计算机辅助制造（CAM）

CAM数据库存储了和CAD系统相似的数据，只是增加了有关离散产品（如在生产线上的汽车）和连续产品（如化学合成）的数据。例如，在化学制造系统中，有监视系统状态信息的应用程序，如反应容器的温度、流速、生产速度等。还包含控制各种物理过程的应用程序，如开阀、给反应堆容器提供更多的热量、提高冷却系统的流速等。这些应用程序必须能够实时地响应，而且必须能够为了保持优越的性能，以很小的代价调整过程。在这个例子中，系统需要维护好具有一定层次结构的大量数据，并处理好数据之间复杂的关系。它也必须能够快速操纵数据以进行浏览或对更改做出响应。

3. 办公信息系统（OIS）和多媒体系统

OIS系统存储了在业务中涉及到计算机信息控制的所有数据，包括电子邮件、文档、货物清单等等。为了更好地支持这个领域，除了名字、地址、日期和货币信息外，我们需要处理更多种类型的信息。现代系统可以处理自由格式的文本、图片、图表、音频、视频等。例如，一个多媒体文档可能包含文本、图片、动画、电子数据表和声音解说等。文档可能用特定的结构来安排这些内容，用一种标记语言来描述这种结构，如SGML（标准通用标记语言）、HTML（超文本标记语言）或XML（扩展性标记语言），就像我们在19.8中将要详细讨论的。

文档可能由多个系统用户共享，比如通过基于Internet技术的电子邮件和布告栏来共享[⊖]。而且，这样的应用程序需要存储的数据的结构比由数字和文本字符串组成的记录的结构更复杂。而且，现在对于使用电子设备代替手写笔记的需求也越来越多。尽管很多笔记可以通过手写分析技术转换成ASCII文本，但是大多数这样的数据还不能做到这一点。除了单词外，手写数据还包括草图、图表等等。

4. 地理信息系统（GIS）

GIS数据库存储了各种空间和时间信息，如土地管理和地下水开采中的应用。这些系统中

⊖ 世界上最大的数据库World Wide Web是一个关键的数据库系统，受到很多人的关注，但它却几乎没有应用任何数据库技术。这部分内容将在19.8节Web和DBMS的集成中介绍。

的大多数数据由调查和卫星图产生，而且往往数量很多。研究工作可能涉及到一些鉴别特征的工作，例如，使用先进的“模式识别”技术基于形状、颜色或是纹理来鉴别。

例如，EOS（Earth Observing System，地球观察系统）就是一个在过去十年中由NASA发射的用来收集信息的卫星群，收集到的信息将为研究地球的长期发展趋势的科学家研究地球的空气、海洋和陆地提供依据。据预测，这些卫星每年可以返回的数据量可达 $\frac{1}{3}\times10^{15}$ 字节。这些数据将和别的数据合成到一起，存储在EOSDIS（EOS Data and Information System，EOS数据和信息系统）中。EOSDIS不仅为科学家还为非科学家提供了他们所需的信息。例如，学生可以访问EOSDIS来查看世界天气图的模型。该数据库的巨大容量和支持成千上万用户提出的大量的信息要求，将为DBMS带来很多挑战。

5. 交互式、灵活多变的Web站点

设想一个Web站点拥有在线的目录来销售衣服，该站点维护早期的访问者的偏好，并允许访问者进行以下操作:

- 浏览目录中的所有物品，选择一件，便可以获得实物的图片和相关信息。
- 选择与用户定义的条件集合相匹配的物品。
- 根据客户指定的服装细节（如颜色、尺寸、造型），得到服装的3D效果图。
- 根据给定的物品的附加细节，选择一段画外音说明。
- 浏览有合适折扣的账单总额。
- 通过一个可靠的在线事务交易完成购买。

这类应用的要求与前面提到的那些高级应用相比没有多少区别：它们需要处理多媒体内容（文本、音频、图像、视频数据和动画），需要根据用户的喜好、选择来交互地改进显示方式，而且还要能够处理复杂数据，站点支持的3D效果图功能给系统附加了复杂性。

就像我们将在19.8节中谈到的，Web现在提供了一种相对新的数据管理模式，XML这样的语言在电子商务领域得到了广泛应用。Forrester Research Group预言，到2006年美国的B2B交易额在欧洲将达到$2.1万亿，在美国将达到$7万亿。总之，到2006年，全球公司的电子商务税收总额将达到$12.8万亿，将达到全球18%的销售量。随着因特网应用的普及及其有关技术的逐渐成熟，我们将看到网络站点和B2B交易将处理越来越多复杂的彼此相关的数据。

19.2 关系DBMS的缺陷

在第2章，我们提到关系数据模型拥有强大的理论基础，基于一阶的谓词逻辑。该理论支持了SQL语言的发展，SQL现在已经成为定义和操纵关系数据库的标准语言。关系数据模型的另一个好处在于它的简单性、适合联机事务处理（OLTP）、支持数据独立性。但是，关系数据模型特别是RDBMS并不是没有缺点的。在这一节中，我们将简要的讨论一下一些常见的不足。

1. 对“现实世界”实体的表达能力弱

规范化通常导致表与“现实世界”中的实体不对应，它将“现实世界”中的实体分割成几张表来显示，以物理表示法来反映实体结构，这样效率会比较差，常常要在查询处理中进行很多连接操作。

2. 语义过载

关系模型表达数据和数据间关系的构造只有一种——表。例如，为了表达实体A和B之间

的多对多（*:*）关系，我们需要创建三张表，两个分别用于表达实体A和B，第三张表用于表达实体间的关系。它没有一种机制来区分实体和关系，也无法区分在实体间存在的不同种类的关系。例如，一个1:*关系可能是Has、Supervises、Manages等等。如果可以进行区分，也许我们就可以将语义构建到操作中。所以，我们说关系模型语义过载了。

3. 不能很好的支持业务规则

在2.3节中，我们已经介绍了实体和参照完整性的概念，在2.2.1中我们也介绍了域的概念，它也是业务规则。但是，很多商业化系统不能完全支持这些规则，所以需要将它们内置到应用程序中。这样当然是危险的，而且容易导致做重复的工作。更糟糕的是，可能还会引起不一致现象。而且，在关系模型中不支持其他类型的业务规则，这又意味着它们需要被构建到DBMS或应用程序中。

4. 有限的操作

关系模型只有一些固定的操作集，例如面向集合和记录的操作，操作是在SQL规格说明中提供的。但是，SQL目前不允许指定新的操作。因此，在给许多“现实世界”对象的行为建模就有了太多的限制。例如，一个GIS应用程序典型的使用点、线、线组、多边形和一些处理距离、交叉点和包含关系的操作。

5. 处理递归查询困难

数据的原子性意味着在关系模型中不允许出现重复的数据组，这样就导致了处理递归查询极为困难。递归查询就是那些有关表和自身直接或间接的关系的查询。为了解决这个问题，SQL可以嵌入在一个高级程序设计语言中，由高级程序设计语言来提供支持反复操作的功能。而且，很多RDBMS提供了具有类似结构的报表书写程序。不管是哪种情况，都是应用程序而不是系统的内在功能提供了所需要的功能。

6. 阻抗失配

在3.1.1节中，我们注意到，直到最新版本的SQL标准，都缺少完全的计算功能。为了解决这个问题并且提供更多的灵活性，SQL标准提供嵌入式SQL来帮助开发更加复杂的数据库应用程序。但是，这引起了阻抗不匹配（impedance mismatch）的问题，因为我们将两种不同的程序设计模式混合在了一起。

1) SQL是一种处理行数据的声明性语言，而诸如C语言这样的高级语言则是过程化的语言，一次只能处理一行数据。

2) SQL和3GL使用不同的模型来表达数据。例如，SQL提供内置的数据类型Date（日期型）和Interval（时间间隔型），而在传统的编程语言中却没有这样的类型。因此，就需要应用程序在两种表示法之间进行转换。而这样做无论从程序设计的工作量还是运行时资源的使用来看都是低效的。而且，由于我们使用两种不同的系统，因此，不可能将类型检测作为一个整体自动进行。

最新颁布的SQL标准——SQL3，通过引入很多新的特征弥补了上述的一些不足，例如作为数据定义语言一部分的定义新的数据类型和操作的能力以及增加一些新的结构以使语言具备计算完整性。

19.3 分布式DBMS和复制服务器

开发一个数据库系统的主要动机是，集成一个企业的操作型数据并提供对数据的访问控制。尽管我们可能认为集成和访问控制意味着集中化，但这并不是目的。事实上，计算机网络的发展

促进了非集中式的工作模式。这种非集中式方法镜像很多公司的组织架构，它们被逻辑地分成分公司、部门、项等等，并且在物理上划分成办公室、分支机构、工厂，每个单元维护自己的操作型数据。分布式DBMS的发展反映了这种组织结构，使得任何一个单元中的数据都可访问，并且将最经常使用的数据存储在与本地最近的地方，这改进了共享数据的能力和访问数据的效率。

分布式数据库（Distributed Database） 一个逻辑上相关的共享数据（以及数据的描述）的集合，它们物理上分布在一个计算机网络上。

分布式DBMS（Distributed DBMS） 允许管理分布式数据库并且使得这种分布对用户来说是透明的软件系统。

一个分布式数据库管理系统（DDBMS）由一个被分成多个片断的逻辑数据库组成。每个片断在一个独立的DBMS的控制下，存储在一个或多个计算机（副本）上，这些计算机又通过一个通信网络连接在一起。每个站点都能够自主地处理用户提出的访问本地数据的要求（即每个站点都有一定的本地自治能力），也能够处理那些存储在网络中其他计算机上的数据。

用户通过应用程序访问分布式数据库。应用程序被划分为不需要别的站点数据的程序（本地应用程序）和需要其他站点数据的程序（全局应用程序）。我们需要一个至少有一个全局应用程序的DDBMS，因此，一个DDBMS拥有下列特征：

- 逻辑上相互关联的共享数据的集合。
- 数据被分割成多个片断（片断可以是水平片断也可以是垂直片断，这与我们在第15章中谈到的水平分区和垂直分区类似）。
- 片断可以被复制。
- 片断/副本分布在不同的站点上。
- 这些站点通过一个通信网络连接在一起。
- 每个站点上的数据都由一个DBMS控制。
- 每个站点上的DBMS能够自主地处理本地的应用程序。
- 每个DBMS至少参与一个全局应用程序。

为系统中的每个站点都配置自己的本地数据库是没有必要的，就像图19-1中显示的DDBMS的拓扑所示。

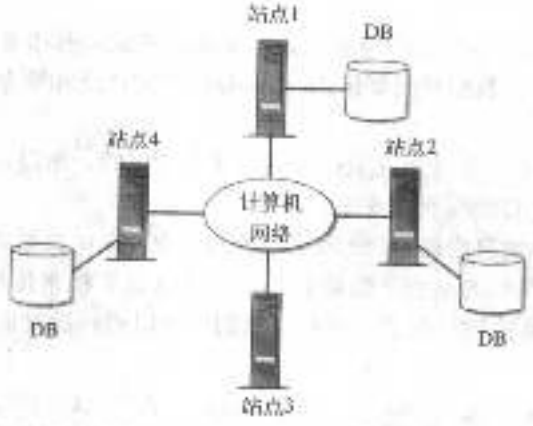


图19-1 分布式数据库管理系统

从DDBMS的定义中我们知道，系统需要使“分布”的特点在用户看来是透明的（不可见的）。所以，分布式数据库被分割成多个片断存储在不同的计算机中，而且片断可能是复制的，这些事实都需要对用户隐藏。透明性的目标是使得分布式系统看起来跟集中式系统一样。有时，这被称为分布式DBMS的基本原则，该需求为最终用户提供了重要的功能，但是，这产生了许多需要DDBMS处理的额外的问题。

分布式处理

将分布式DBMS和分布式处理进行比较是很重要的。

分布式处理（Distributed Processing）　一个能够通过计算机网络来访问的集中式数据库。

DDBMS的定义的关键点是系统由在物理上分布于网络中的很多站点的数据组成。如果数据是集中的，尽管其他用户可能是通过网络来访问数据，但我们也不认为它是DDBMS，而只能称之为分布式处理。图19-2说明了分布式处理的拓扑。比较本图和图19-1，图19-2在站点2上有一个集中数据库，而图19-1显示多个站点都有各自的数据库。

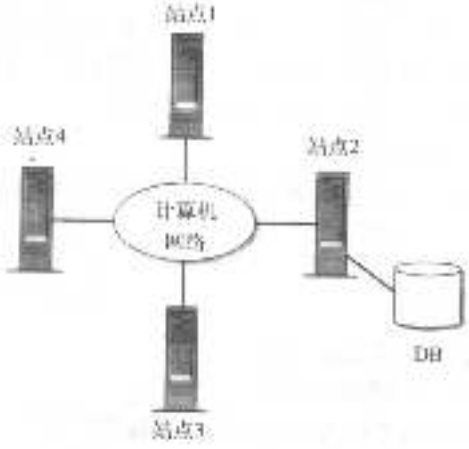


图19-2　分布式处理

19.3.1　DDBMS的优缺点

数据和程序的分布相对于传统的集中式数据库系统来说，有很多潜在的优点。但是，它也有很多缺点。在本节中，我们将简要讨论一下DDBMS的优点和缺点。

1. 优点

反映了企业的结构　很多企业本来就是分布在多个地区的，所以，将应用在这些企业上的数据库分布在不同地区是自然而然的事情。

提高了共享能力和本地自治能力　地理上分散的企业可以由数据的分布性来体现，某个站点的用户可以访问存储在别的站点上的数据。数据可以放置经常使用该数据的用户的附近，这样，用户就可以在本地控制数据了，而且，我们还可以根据数据的使用情况来相应地建立和加强本地策略。

提高了可用性　在集中式DBMS中，计算机的故障将中止DBMS的操作。但是，DDBMS中一个站点的故障，或者是一些通信线路的故障，只会使得有些站点不能被访问，但不会使

整个系统不可操作。

提高了可靠性 由于数据可以被复制，所以它将在多个站点存在。一个结点或是一条通信线路的故障不一定使该数据不可访问。

提高了性能 由于数据位于“最需要使用”的站点附近，加上DDBMS本身固有的并行性，所以它比远程集中式数据库更容易改进数据库的访问速度。而且，由于每个站点都只处理整个数据库的一部分数据，可能就没有集中式DBMS中对CPU和I/O服务的争用。

经济 众所周知，建立一个由多个小计算机组成的系统比建立具有相同性能的单一的大计算机的花费要少。让分公司和部门都拥有自己单独的计算机是很经济的。所以，通过在网络中添加工作站比更新主机系统要更经济。

模型增长 在一个分布式环境中，处理扩展更加简单。新的站点可以加入到网络中而不影响其他站点的数据操作。这个灵活性使得企业的扩展相对容易。

2. 缺点

复杂性 一个DDBMS需要对用户隐藏分布的本质，而且还需要提供了用户满意的性能、可靠性和可用性水平，这使得DDBMS比集中式DBMS更复杂。复制也增加了系统的复杂性，因为如果处理不完全，与集中式系统相比，将导致可用性、可靠性和性能降低，那么我们前面列举的一系列优点也将变成缺点。

价格 提高复杂性就意味着获得和维护DDBMS的代价要比一个集中式DBMS高。另外，DDBMS需要额外的硬件来建立站点之间的网络。而且还有使用网络而产生的通信费用。另外，还有为了管理和维护各个本地DBMS和网络的人力资源消耗。

安全性 在集中式系统中，可以很容易控制对数据的访问。但在DDBMS中，不仅需要在多个站点控制对复制数据的访问，而且网络本身也需要保证安全性。在过去，网络被认为是一个不安全的通信媒体，尽管这个问题现在仍然存在，但最近已经在网络安全性方面有了重大的改进。

完整性控制更加困难 加强完整性约束通常需要访问定义了约束的大量数据，但不包括实际的更新操作本身在内。在一个DDBMS中，用来加强完整性约束的通信和处理代价可能都是被禁止的。

缺乏标准 尽管DDBMS依赖于有效的通信，但是我们到现在才开始看到通信标准和数据访问协议的出现。标准的缺乏严重限制了DDBMS的潜力。而且也没有帮助用户将集中式DBMS转换成分布式DBMS的工具和技术。

缺乏经验 尽管很多协议和问题已经被理解了，但DDBMS的一般目的并没有得到广泛认同。结果，我们没有达到像集中式DBMS一样的经验水平。展望该技术的未来，这可能会是最严重的威胁。

数据库设计更加复杂 除了需要考虑设计集中式数据库的一些问题外，设计分布式数据库还需要考虑数据的分段、将分段数据分布到具体的站点以及数据复制问题。

到目前为止，前面我们提到的DDBMS的一般用途没有被广泛接受，尽管很多协议和问题已经被很好地理解。但是，数据复制（在多个服务器上存储和维护数据）似乎是更好的解决方法。每个主要的数据库开发商都拥有相应的复制解决方案，而很多非数据库开发商也有复制数据的一些方法。复制服务器就是一种方案，是一种潜在的更为简单的数据分布方法，我们现在就对其进行详细讨论。

19.3.2 复制服务器

复制（Replication） 在一个或多个站点上生成或产生数据的多个副本的过程。

复制是一种重要的机制，因为它可以使企业中的用户在任何时候任何地点访问最近的数据。复制提供了很多好处，包括当集中式资源超载时改进系统性能，增加可靠性和数据可用性，支持可移动计算和数据仓库。在本节中，我们将讨论几种有关数据复制的背景知识，包括期望的功能和数据所有权。我们从讨论什么时候更新复制的数据开始。

1. 同步复制和异步复制

DDBMS中的更新复制数据的协议建立在一旦源数据被更新则复制的数据也被立刻更新（即，作为封闭事务的一部分）的基础上。这种复制方案称之为同步复制。这种机制适合那种必须保持所有的副本都同步的环境下（例如，金融事务），这有很多缺点。例如，如果一个或多个站点中的副本不可用，那么整个事务就会因此无法完全执行成功。进一步说，同步复制数据需要用大量的消息来进行协调，这将给网络增加巨大的负担。

很多商业性DBMS提供了一种同步复制的替代机制，称为异步复制。使用这种机制，目标数据库将在源数据库已经完成修改后再进行更新。重新获得一致性的延时从几秒到几个小时甚至几天不等。但是，数据最终将在所有复制站点达到一致。尽管这违背了分布式数据独立性的原则，但它被认为是数据完整性和可用性之间的一个实用的折中，这可能更适合那些需要复制但是不一定要求同步或并发的企业。

2. 功能

作为最基本的标准，我们希望一个分布式数据复制服务能够将数据从一个数据库拷贝到另一个数据库中，同步复制或者异步复制都行。但我们还需要其他的功能，例如:

- **复制机制的规格说明** 系统应该提供一种允许授权的用户指定要复制的数据和对象的机制。
- **订阅机制** 系统应该提供一种允许授权的用户订阅复制的数据和对象的机制。
- **初始化机制** 系统应该提供一种允许初始化目的副本的机制。
- **伸缩性** 该服务应该能够处理小数据量和大数据量的复制。
- **映射和转换** 该服务应该能够处理来自不同DBMS和平台的复制。这可能涉及到把数据从一种数据模型映射或转换到另一种数据模型，或者是将一种数据类型的数据转换为另一个DBMS中的相应的数据类型。
- **对象复制** 它应该能够复制除数据之外的其他对象，例如，有些系统允许复制索引和存储过程（或是触发器）。
- **易于管理** 它应该能够让DBA易于管理系统、检查状态以及监控复制系统组成部分的性能。

3. 数据所有权

所有权就是指哪个站点拥有更新数据的权限。主要的所有权种类有主/从、工作流和任意更新权（有时称之为对等或是对称复制）。

(1) 主/从所有权

利用主/从所有权，异步复制数据将被一个站点拥有，该站点被称为主站点，而且只能由该站点进行更新。用“发布-订阅”来比喻，主站点（发布者）使数据可被访问。其他的站点将“订阅”主站点拥有的数据，这就意味着它们只能在本地系统中接受只读的副本。潜在的，

每个站点都可以成为无重叠数据集的主站点。但是，只能有一个站点可以修改特定数据集的主副本，因此站点之间不会发生更新冲突。

一个主站点可能拥有整张表的数据，在这种情况下，其他站点只能订阅该表的只读副本。另外，多个站点可能分别拥有一张表的不同的片断，其他站点只能订阅片断的只读副本。这种复制类型也被称为异步复制。

(2) 工作流所有权

和主/从所有权一样，该模型也避免了更新冲突，同时提供了更灵活的所有权模式。工作流所有权允许更新复制数据的权限从一个站点传到另一个站点。但是，在任何时刻，只能有一个站点能够更新特定的数据集合。工作流所有权的一个典型的例子是一个订单处理系统，订单处理有一系列的步骤，例如订单输入、订单批准、发货、运输等等。在一个集中式DBMS中，这类应用程序都在一个集成的数据库中访问和更新数据，即每个应用程序当且仅当前一个步骤已经完成时，才按顺序更新数据。

(3) 在任何地方更新数据的所有权（对称复制）

前面两种模型有一个共同的特性：在任何给定的时刻，只有一个站点可以更新数据，其他的站点只有对该副本的只读权限。在有些环境中，这就显得太局限了。这种在任何地方更新的模型在多个站点拥有更新复制数据的权力，这创建了一个对等环境。这允许本地站点能够自主的处理，即使这些更新对其他站点是不可用的。

共享拥有权可能导致冲突情况，而且复制架构要能够开发一种发现冲突和解决冲突的方法。一个简单的在单表中发现冲突的机制是，源站点发送在最后一次刷新操作之后，所有更新了的记录的旧值和新值（分别标记为前像、后像）。在目标站点，复制服务器可以在已经更新了的目标数据库中检查每个值。但是，我们必须要考虑其他类型的冲突的监测，例如违反了两个表之间的参照完整性。目前已经有很多机制来解决冲突，但最常用的是：最早/最晚的时间戳、站点优先权和保持手工解决方法。

19.4 面向对象的DBMS和对象-关系DBMS

在19.2节中，我们回顾了关系模型相对新兴的高级数据库应用的要求的不足之处，在本节，我们将简要介绍试图解决这些已经认识到的不足之处的两个有竞争力的方法。这两种方法都基于面向对象的概念，解决了软件开发中的传统的问题。

19.4.1 面向对象的DBMS

一种方法是将面向对象概念和数据库系统结合起来，即面向对象数据库管理系统(OODBMS)。OODBMS出现在工程和设计领域，并且已经成为金融业和电信业应用中受欢迎的系统。有关面向对象数据模型已经提出了很多种不同的定义。例如，Kim（1991年）定义了面向对象数据模型（OODM）、面向对象数据库（OODB）和面向对象DBMS（OODBMS）如下：

OODM 一个捕获面向对象编程中支持的对象的语义的（逻辑）数据模型。
OODB 由OODM定义的持久的、共享的对象集合。
OODBMS OODB的管理者。

这些定义是非常不具体的，而且反映了一个事实，即没有一个面向对象数据模型能够和关系系统的基本数据模型等价。每个系统提供了自身的基本功能的描述。基于一些现在的商业OODBMS，我们可以发现面向对象数据模型概念是从不同的领域中提取出来的，如图19-3所示。

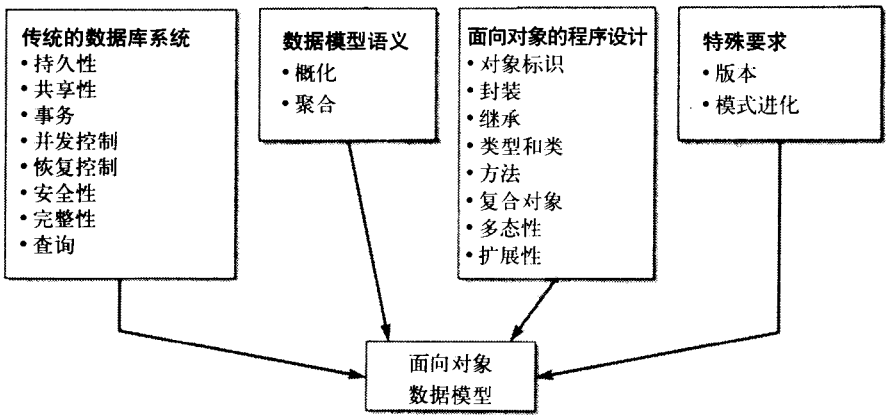


图19-3　面向对象数据模型的起源

OODBMS最早被提出来的一个缺点是它们缺乏形式化的数据模型。但是，在过去的十年里，几家重要的厂商（包括Sun Microsystems、eXcelon Corporation、Objectivity Inc.、POET Software、Computer Associates和Versant Corporation）组成了对象数据管理组（ODMG）定义了OODBMS标准。ODMG已经提出了一个对象模型用来指定数据库对象语义的标准模型。这个模型很重要，因为它决定了能被OODBMS理解并执行的内置语义。ODMG也提供了对象定义语言和对象查询语言，它们形成了SQL的一个超集。

1. OODBMS的优点

OODBMS的很多优点是“面向对象”和系统相结合的结果，例如:

- 丰富了建模功能。
- 扩展性。
- 消除了“阻抗不匹配”。
- 能够支持高级数据库应用。

另外一些优点则是利用更多合适的协议的结果，例如:

- 支持模式进化。
- 支持长时间的事务。

另外还要提到的两个优点是:

- 更具表达力的查询语言　OODBMS一般使用“导航访问”从一个对象移到下一个对象，这和SQL的关联访问不同（即，查询语句的声明是基于一个或多个谓词）。“导航访问”更适合于处理部分爆发数据、递归查询等等。
- 提高了性能　有很多基准已经指出OODBMS比RDBMS提供了更显著的性能改善。例如，在1989年和1990年，OO1基准连续研究了OODBMS GemStone、Ontos、ObjectStore、Objectivity/DB、Versant和RDBMS INGERS及Sybase，结果显示OODBMS的性能平均比RDBMS提高30%。

2. OODBMS的缺点

OODBMS也有一些缺点:

- 缺少可借鉴的经验。
- 缺少标准。
- 和RDBMS的竞争。
- 复杂性。
- 缺少视图的支持。
- 缺少安全性支持。

19.4.2 对象-关系DBMS

从传统的关系数据模型脱离出来并将面向对象概念和数据库系统集成在一起的方法，有时也被称为是革命性的方法。与此相对应，对象-关系DBMS（ORDBMS）更是一个将面向对象概念和扩展关系模型的数据库系统集成起来的革命性的方法。

直到最近，DBMS的选择似乎都在RDBMS和OODBMS之间。但是，很多RDBMS产品的厂商已意识到OODBMS的威胁和承诺。他们承认传统的RDBMS不能适用于19.1节中提到的高级应用，而且需要附加的功能。但是，他们否认扩展的RDBMS不能提供有效的功能或者处理速度太慢而不能解决新的复杂性问题。

如果我们注意观察出现的高级数据库应用，我们会发现它们使用了很多面向对象的特征，例如用户扩展类型系统、封装性、继承性、多态性、方法的动态绑定，包括非第一范式对象的复杂对象以及对象标识（如图19-3所示）。弥补关系模型的缺点的最明显的方法是扩展该模型使其具备这些类型特征。尽管每个RDBMS实现了不同的特征组合，但这个方法已经被很多扩展型RDBMS所采用。因此，现在出现了一系列的这种模型，每个模型的特点都依赖于它朝哪个方向以何种程度来扩展。但是，所有的模型确实都共享相同的基本关系表和查询语言，都合并了一些“对象”的概念，而且有些模型还能在数据库中像存储数据一样存储方法（或者是存储过程、触发器）。

RDBMS的三个最重要的厂商——Oracle、Informix和IBM都已经将它们的系统扩展成为了ORDBMS，尽管每家提供的功能有些细微的差别。ORDBMS的概念（作为RDBMS和OODBMS的混合）保留了从RDBMS那里得到的知识和经验的财富。尽管这样，有些专家预测ORDBMS将比RDBMS多占50%的市场份额。

也许已经设想到，这个领域的操作标准是基于对SQL标准的扩展。国际化标准组织早在1991年就已经开始研究SQL的对象扩展问题了，该扩展成为了SQL 1999标准（通常被称为SQL3）的一部分。SQL3标准是对关系模型和查询语言的扩展标准的一种尝试。

1. ORDBMS的优点

除了弥补在19.2节中提到的很多弱点外，关系数据模型的扩展的主要优点来自于它的重用性和共享性。重用性源于扩展DBMS服务器以实现标准功能，而不必在每个应用程序中都编程。例如，应用程序可能需要空间数据类型来表示点、线和多边形，并有计算两点间距离、点到线距离、点是否包含在多边形内、两个多边形是否有重叠等其他一些内容的相关功能集合。如果我们能够将该功能植入服务器中，那么就不必在每个需要用到这些功能的应用程序中都定义它们，而是允许该功能集被所有应用程序共享使用。这些优点不仅提高了开发者的

效率，也提高了最终用户的效率。

另外一个显著的优点是扩展关系方法保留了开发关系型应用程序的一些重要的知识和经验。这是很重要的一个优点，因为很多组织都发现进行改变并不是很贵。如果设计了新的功能，该方法能够允许企业在不失去现有数据库特征和功能的优点的情况下，以革命性的方式利用新的扩展。所以，ORDBMS可以作为一个综合的方法、一个概念实验项目来介绍。最近的SQL3标准是与SQL2标准完全兼容的标准，因此任何遵从SQL3的ORDBMS都能够提供这些兼容性。

2. ORDBMS的缺点

ORDBMS方法有一个明显的缺点，就是复杂性和相关费用增加，而且，有些关系型方法的支持者还认为这种扩展失去了关系模型最重要的简单性和纯正性。也有一些人认为RDBMS扩展后，只能适应小部分的应用，而不能成功得到目前关系型技术的理想性能。

另外，面向对象支持者也没有被这些扩展所吸引。他们认为ORDBMS术语是有启迪作用的，ORDBMS不讨论对象模型，而是使用像用户定义数据类型之类的术语。面向对象的术语是围绕抽象类型、类层次结构和对象模型等。但是，通过增加一些复杂性，ORDBMS厂商正试图描绘对象模型以作为关系模型的扩展。这潜在地避开了面向对象的观点，强调了两种技术之间语义上的重大差别。对象应用不是简单的基于关系型的数据中心。面向对象模型和程序将关系和封装的对象更紧密地结合在了一起，更贴近地反映了“现实世界”。事实上，对象本质上不是数据的扩展，而是一个对“现实世界”关系和行为有更强表达能力的完全不同的一个概念。

19.5 数据仓库

19世纪70年代以来，企业越来越多的关注他们在能够自动处理业务的新的计算机系统上的投资（被称为联机事务处理（OLTP）系统）。通过这种方法，企业能够通过该系统获得更大的竞争力，即为顾客提供更有效更经济的服务。在这段时期，企业在他们的数据库中积累了日益增长的大量的数据。但是，现在这种系统如此普遍，企业开始关注使用这些操作数据来做出决策的方法，以获得更强的竞争力。

运作的系统从来就不是为了支持商业决策而设计的，所以使用这样的系统可能永远也不能得到简单的解决方案。问题是一个企业可能会有几个系统同时存在，所以有时候会出现重复和冲突的定义，例如数据类型。企业的困难是如何将数据源转换为知识源，从而为用户提供一个集成/合成的企业数据视图。于是，数据仓库的概念应运而生，它满足了系统的要求，能够支持决策制定、从多个操作型数据源中获得数据。

数据仓库（Data Warehouse） 从一些不同的操作型数据源整理出企业数据的合成的/集成的视图，并有一系列最终用可使用的工具来支持从简单到复杂的查询以制定决策的工具。

在数据仓库中的数据是面向主题的、集成的、随时间变化的、稳定的。

- **面向主题** 指数据仓库是围绕企业的主要的主题（比如顾客、产品、销售）而不是根据主要的应用领域（如用户发货、库存控制和产品销售）来组织系统的。这反映了存储支持决策数据的需要，而不是存储面向应用的数据的需要。
- **集成的** 因为源数据来自企业范围内不同的应用系统，所以源数据常常是不一致的，例

如，常常使用不同的数据类型或格式。集成后的数据源必须一致以便给用户提供一个统一的数据视图。

- **随时间变化** 数据仓库中的数据只在某个时间点或在一定的时间间隔中才是准确的、有效的。数据仓库随时间变化的特征在数据运行的时候也会出现，在所有数据都隐式或显式的与时间有关，事实上数据代表了一系列的“快照”时都会出现。
- **稳定的** 数据并不是实时更新的，而是定期从操作型系统中刷新。新数据常常作为数据库的补充增加进来，而不是替代原先的数据。数据库不停的吸收新的数据，并逐渐将它和原先的数据集成到一起。

数据仓库的典型架构如图19-4所示。

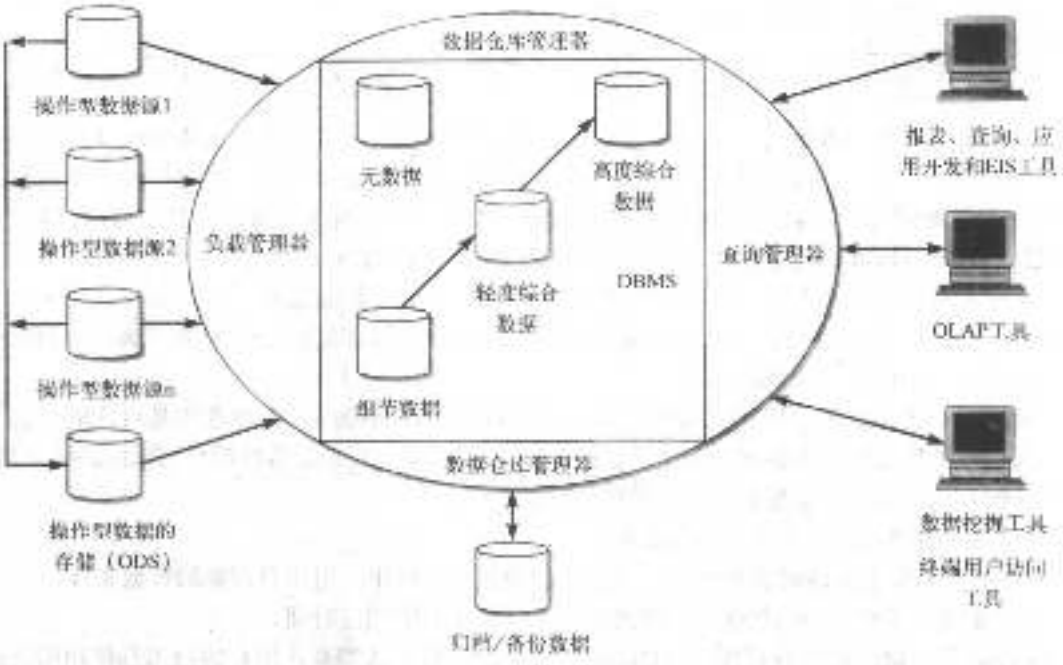


图19-4 典型的数据仓库架构

数据仓库的操作型数据源是由大型主机、私有的文件系统、私有的工作站和服务器以及诸如因特网这样的外部系统提供的。一个操作型数据存储（ODS）是用于分析的当前的和集成的数据的仓库，它常常和数据仓库有同样的组织方式，但是实际上仅仅是作为把数据移入数据仓库的集结区。负载管理器完成所有与数据抽取和加载数据到数据仓库有关的操作。数据仓库管理器运行所有和数据管理有关的操作，例如源数据的转移和合并、索引的创建和在基本表上建立视图、产生聚合、备份并归档数据等。查询管理器完成所有与管理用户查询有关的操作。细节数据不是在线存储的数据，而是要经过综合后用于下一个级别的数据。但是，细节数据定期作为综合数据的补充加入到数据仓库之中。数据仓库存储所有由数据仓库管理器事先定义的轻度综合和高度综合的数据。综合信息的目的是为了加快查询速度。尽管初级的综合数据增加了操作代价，但是在响应用户查询时降低了执行综合操作（例如排序和分组）的需求，从而使代价得到弥补。综合数据在新数据载入数据仓库时进行更新。为了归档和备份的目的，细节数据和综合数据都被脱机存储。元数据（关于

数据的数据）的定义被数据仓库中的所有处理使用，包括提取和加载处理、数据仓库的管理处理以及部分查询管理处理。

数据仓库的首要目的是为制定决策的商业用户提供信息。这些用户通过最终用户访问工具（end-user access tool）来和数据仓库进行交互。数据仓库必须有效地支持特殊查询、日常程序分析以及更复杂的数据分析。典型的最终用户访问工具类型包括报表和查询工具、应用程序开发工具、可执行信息系统（EIS）工具、联机分析处理（OLAP）工具以及数据挖掘工具。我们将在下面章节中讨论OLAP和数据挖掘工具。

数据集市

和数据仓库同时出现的一个相关的概念是数据集市。

数据集市（Data Mart） 数据仓库的一个子集，支持一个部门或商业领域的决策制定要求。

数据集市中存有数据仓库中的部分数据，常常是以和特定部门或是商业领域（如市场营销、客户服务）有关的综合数据的形式存放。数据集市可以独立存在，也可以和企业数据仓库相连。当数据仓库逐渐增大时，就可能会危及到为企业机构的各种要求提供服务的能力。数据集市的普及取决于企业数据仓库是否难以创建和使用的事实。

有好几种创建数据集市的方法，一种方法是先构建多个数据集市，然后最终合成为一个数据仓库。另一种方法是，为一个企业数据仓库创建一个基础结构，同时为了满足企业的中间需求创建一个或多个数据集市。

数据集市体系结构可以创建成两层或三层的数据库应用程序。数据仓库是可选的，属于第一层（如果数据仓库为数据集市提供数据），数据集市是第二层，最终用户工作站是第三层，数据分散在三层之中。创建数据集市的理由包括:

- 为用户提供最经常分析的访问数据。
- 提供一种符合要求的数据形式，符合部门或数据领域中一组用户浏览的数据集合。
- 由于减少了被访问的数据量，因此改进了最终用户的响应时间。
- 提供了由诸如联机分析处理（OLAP）和数据挖掘工具等最终用户访问工具使用的合适的结构化数据，这些工具可能要求它们有自己内部的数据库结构。事实上，为了支持它们特定的功能，这些工具常常创建它们自己的数据集市。
- 数据集市常常使用相对较少的数据，所以数据清理、数据加载、数据转移和数据集成工作都变得非常容易，因此实现和创建一个数据集市与建立一个企业数据仓库比起来要简单很多。
- 实现一个数据集市的成本常常比建立一个数据仓库的成本要少。
- 数据集市的潜在用户能够更清楚地被定义，并且与企业数据仓库项目比起来，他们能够更容易被发展成为支持数据集市项目的目标群。

支持数据仓库和数据集市应用程序的数据库，与支持传统的OLTP应用程序的数据库有很大不同。读者如果对如何设计一个支持决策应用的数据库感兴趣，可以参考2002年Connolly和Begg出版的书。

19.6 联机分析处理

在过去几十年里，我们已经看到RDBMS广泛使用，并且受欢迎的程度越来越高，所以我

们现在发现企业的大部分数据都存储在这种系统中的。关系数据库主要用于支持传统的OLTP系统。为了给OLTP系统提供合适的支持，RDBMS已经发展成为能够高效执行相对简单的事务的系统。

在过去几年里，RDBMS厂商将目标投向了数据仓库市场，并已经改进了他们的系统作为创建数据仓库的工具。一个数据仓库存储了操作型数据，并能够支持从相对简单到非常复杂的查询。但是，响应特定查询的能力依赖于数据仓库中能够使用的最终用户访问工具的种类。一般的工具（如报表和查询工具）很容易支持对过去事件的“谁”和“什么”的查询。一个直接由数据仓库执行的典型的查询是：“Seattle 2003年第三季度的总税收有多少？”但是，使用称为联机分析处理（OLAP）的工具来访问数据仓库，就能够支持更高级的查询并进行数据分析。

联机分析处理（OLAP） 对大量多维数据集进行动态综合、分析和合并。

OLAP描述了一种技术，这种技术使用综合数据的多维数据视图来快速访问策略信息，以便进行高级分析。OLAP使得用户能够通过快速、一致和交互的方式访问大量可能的数据视图，对企业数据有更深入的了解和认识。OLAP允许用户以如下方式使用企业数据：它是企业真正的维度模型。在OLAP能够轻而易举地回答“谁”和“什么”这样的问题时，它们还能回答“如果……怎样……”和“为什么”这样的问题，这就是它们和一般目的的查询工具的不同。OLAP能够支持对未来动作做出决策。一个典型的OLAP计算比简单地综合数据更复杂，例如，“如果法律费用上涨3.5%，政府金额超过$100000的财物的税收下降1.5%，对美国不同地区的销售业会有什么影响？”所以，OLAP可以进行不同的分析，从基础导航到浏览（即“切片和切块”）、计算以及更复杂的分析（如时间序列和复杂建模）。

在各种商业领域中有很多OLAP的应用例子，如表19-1所示。

表19-1 在不同商业领域的OLAP应用实例

商业领域	OLAP应用实例
金融业	预算、基于活动的成本、金融性能分析、财务模型
销售业	销售分析和销售预测
市场营销	市场研究分析、销售预测、促销分析、顾客分析、市场/顾客切分
制造业	产品计划和次品分析

所有OLAP应用的首要要求是为用户提供及时（Just-in-Time，JIT）信息的能力，这对为企业战略方向作出有效的决策是必需的。JIT信息是一种经过计算的数据，常常反映了复杂的关系，通常是在不运行的时候计算的。只有响应时间一直很短，分析建模复杂的关系才有实际意义。另外，由于数据关系的特性不能预知，所以数据模型必须是灵活的。一个真正灵活的数据模型保证OLAP系统能够响应为了有效地出决策而不断变化的业务要求。尽管OLAP应用的商业领域有很大不同，但是它们都需要有企业数据的多维视图、支持复杂的计算（如预测）和时间智能。时间智能是任何一个分析型应用的关键特征，因为性能常常由时间来衡量。例如，这个月和上个月比较，或者是这个月和去年这个月的比较。

成功实现OLAP应用程序的好处有：

- 对战略信息的访问更可控、更及时，可以作出更有效的决策。

- 通过给最终用户更多的自主权来改变数据库，建立他们自己的模型，缩短了IT工作人员进行应用程序开发所做的工作。
- OLAP应用程序依赖于数据仓库和OLTP系统来刷新它们的数据源，所以保留了企业数据的完整性控制。
- 减少了OLTP系统和数据仓库的查询量和网络流量。
- 使企业能更快响应市场需求，减少了潜在的风险，提高了收益。

19.7 数据挖掘

仅仅将信息存储在数据仓库中并不能给组织提供所需要的好处。为了实现数据仓库的价值，我们需要将数据仓库中隐藏的知识挖掘出来。但是，随着数据仓库中数据的数量和复杂性的增长，业务分析人员只使用简单查询和报表工具确定趋势和数据之间的关系变得越来越困难了。数据挖掘是从海量数据中抽取出趋势和模式的最好的方法。数据挖掘能够在数据仓库中发现一些查询和报表不能有效发现的信息。

数据挖掘（Data Mining） 从大型数据库提取出有效的、先前不知道的、可理解的、可控制的信息的程序，并用它作出重大的商业决策的过程。

数据挖掘与数据分析和为寻找隐藏在数据集中的未知的模式和关系而使用的软件技术有关，数据挖掘关注的是发现隐藏的未知的信息，因为发现很明显的模式和关系几乎没有任何意义。检查数据中隐藏的规则和特征能够确定模式和关系。

数据挖掘分析人员往往基于数据工作，而得出最精确结果的技术通常要求利用大量的数据来得到可靠的结论。在分析过程中，首先为样本数据结构开发一个最优表示，并在这个过程中获得知识。这个知识接着被扩展到大量的数据集中，假定那个稍大一些的数据集拥有一个和样本数据相似的结构。

数据挖掘可以为已经在数据仓库方面大量投资的公司带来丰厚的回报。尽管数据挖掘仍然是一个相对新的技术，但它已经在很多领域中得到了应用。表19-2列出了数据挖掘在零售业/市场营销、银行业、保险业、药业方面的数据挖掘应用的例子。

表19-2 数据挖掘应用举例

零售业/市场营销	**保险业**
确定顾客的购买模式	提出分析
发现顾客的统计特征之间的关联	预测会购买新保险方案的顾客
预测通过邮件促销的反应情况	**药业**
市场篮子分析	通过刻画病人行为特征来预测外科的访问率
银行业	针对不同的病，确定成功的药物治疗方案
检查欺诈信用卡的使用模式	
确定贵宾级顾客	
预测可能会改变他们的信用卡关系的顾客	
决定由顾客群共同使用的信用卡	

数据挖掘技术有四个主要的操作，包括预测建模、数据库分割、链接分析和违反检测。尽管四个主要的操作中的任何一个都可以用来实现表19-2中列出的商业应用，但在应用和相应的操作中有一定的关系。例如，直接市场营销策略通常使用数据库分割操作来实现，而欺诈检测

可以使用四种方法中的任一种来实现。而且，很多应用在多种操作同时使用时工作的特别好。例如，一个常见的寻找顾客方法就是先分割数据库，然后对结果数据片断进行预测建模。

实现数据挖掘操作的技术随操作的不同而不同，但是，每种操作都有它自身的长处和弱点。记住，有时候数据挖掘工具提供了操作方法的选择来实现某个技术。选择常常基于一定的输入数据类型、挖掘输出的透明性、对缺失变量值的容忍程度、可能的准确水平、增长的处理海量数据的能力等。

19.8 网络数据库集成和XML

自从1989年出现万维网，仅仅十几年的时间它就成为到目前为止最受欢迎、功能最强大的网络信息系统。在过去几年里，它几乎呈指数级发展，它引发了信息革命并且将在今后十年内继续飞速发展。现在，将网络和数据库结合给创造高级数据库应用提供了很多新的机会（我们在19.1节中给出了一个例子）。

网络是传递和分发数据中心的最引人注目的平台，企业现在正迅速地利用网络开发新的数据库应用程序或重建现有系统，将它作为实现新颖商业解决方案的战略平台，并有效成为以网络为中心的组织。

因特网最初和ARPANET网中一系列结点相连，在1997年1月因特网大约拥有了超过一亿的用户[⊖]。一年之后，估计在超过100个国家拥有了2.7亿用户，到2001年初，估计用户人数达到了3.9亿，而到了2003年则增长到了6亿多。有一个组织预言到2004年，用户将增长到9.4亿。而且，有些人估计现在因特网上有将近25亿个文档，每天增加750万个。如果我们将企业内部网和外部网上的文档都加在一起，文档数量将达8000亿，这简直令人难以相信。

19.8.1 静态和动态的网页

存储在文件中的HTML/XML文档是静态网页的一个例子，静态文档的内容不会改变，除非文件自身被修改了。而一个动态网页的内容将在每次访问它时生成。结果，动态网页拥有静态网页没有的一些特征，例如:

- 它可以响应来自浏览器的用户输入。例如，通过表单的形式返回请求的结果或数据库的查询结果。
- 它可以为每个用户定制。例如，如果一个用户在访问某个网站或网页时指定了特别的爱好（如兴趣领域、专业层次），则该信息可以被收集起来，并返回合乎他们喜好的信息。

如果文档作为动态文档发布，比如那些由数据库查询产生的文档，则超文本需要由服务器生成。为了实现该功能，我们可以编写脚本来实现不同数据格式向HTML的转化。这些脚本语言也应该能够解释HTML形式的客户端查询请求，能够理解由拥有该数据的应用程序（如DBMS）生成的结果。由于数据库是动态的，随着用户的创建、插入、更新、删除数据会发生改变，所以生成动态网页比创建静态网页更适合于数据库应用。我们将简要讲述创建动态网页的方法。

现在很多网站都是基于文件系统的，所以每个网络文档都是独立存储的文件。对于小网站，该方法的实现很简单。但是，对于大型网站来说，这会导致严重的管理问题。例如，维护成千上万个不同的文件中的文档的当前副本是很困难的，而维护这些文件之间的链接关系

⊖ 在这里，因特网包括Web、E-mail、FTP、Gopher以及Telnet服务。

就更加困难了，尤其是当这些文档是由不同的作者创建和维护时。

在实际实现时产生了另一个问题，就是现在很多网站包含了更多的具有动态特性的信息，如产品和价格信息。在数据库和独立的HTML/XML文件中维护这些信息是一项非常庞大的工作，而且很难保持同步。由于这样或那样的原因，允许直接从Web中访问数据库是目前日益广泛采用的方法，这也是管理动态Web内容采用的方法。在数据库中存储Web信息可以替代文件存储或作为文件存储的补充。

19.8.2 Web-DBMS集成需求

当很多DBMS厂商在设法为Web提供合适的数据库连接解决方案时，大多数企业要求有一种通用性解决方案，使得它们可以不必完全依赖于某种技术。在本节中，我们简要列举出将数据库应用和Web集成在一起的一些主要要求。这些要求是理想的，而且在目前来说是不能完全实现的，而且有些要求可能要和其他要求进行一些折中。这些要求如下:

- 以安全方式访问有价值的企业数据的能力。
- 数据和厂商的独立连接，允许自由地选择现在或将来的DBMS。
- 数据库界面的独立性，它不依赖于任何Web浏览器或Web服务器。
- 能够利用企业的DBMS的所有特征的连接解决方案。
- 一个开放性架构方案，允许各种系统和技术的互操作。
- 一个具有扩展性、增长性、能根据企业战略进行改变的高性价比解决方案，有助于降低开发成本和维护应用程序的成本。
- 支持跨多个HTTP请求的事务。
- 支持基于会话和应用的认证。
- 可接受的性能。
- 最小的管理负荷。
- 高生产率的工具集，允许以相对容易、快速的方式开发和维护应用程序。

19.8.3 集成Web和DBMS的方法

将数据库和Web环境集成起来有很多种方法，而且在如今瞬息万变的时代，新的方法正在不断涌现。下面是一些常用的例子:

- 诸如 JavaScript 和 VBScript 之类的脚本语言。
- 通用网关接口（CGI），一个早期的广泛使用的技术。
- HTTP cookies。
- 扩展Web服务器，例如 Netscape API（NSAPI）和Microsoft 的 Internet Information Server API（ISAPI）。
- Java和JDBC、SQLJ、Servlets和JavaServer Page（JSP）。
- 特定厂商的解决方案，如Microsoft的带有Active Server Pages（ASP）和ActiveX Data Object (ADO) 的Web Solution Platform，以及 Oracle 的 Internet Platform，它带有Oracle Portal和Oracle PL/SQL Server Pages （PSP）。

19.8.4 XML

Web中的大多数文档是即时存储并转变成HTML的。HTML的一个优点就是它的简单性，

使得它被很多用户使用。但是，它的简单性又是它的缺点，随着用户对标记的需求的增长，他们希望标记可以简化他们的工作并使HTML文档更引人注目、更动态。为了满足该需求，厂商提出了一些特定浏览器的HTML标记，但它难以开发复杂的、可以广为浏览的Web文档。为了弥补这个不足，World Wide Web Consortium（W3C）提出了一个新的标准，称为XML（eXtensible Markup Language，可扩展标记语言），这个语言可以保持一般的应用独立性，从而使得HTML可移植、功能更强大。XML是SGML（Standard Generalized Markup Language）的一个受限制的版本，它是专门为Web文档设计的。例如，XML支持指向多文档的链接，而HTML链接只能指向一个目的文档。

> **XML** 一种元语言（描述其他语言的语言），允许设计者创建自己定制的标记来提供在HTML中得不到的功能。

XML影响了程序设计中的每个方面，包括图形接口、嵌入式系统、分布式系统，而且根据我们的推测，数据库管理也可以。例如，自从XML描述了数据的结构后，它就成为定义异构数据库的结构和数据源的一个有效机制。由于XML能够定义数据库的整体模式，因此可以用它来改变一个Oracle模式的内容，例如，将它转变成Informix或Sybase的模式。它实际上已经成为了软件行业中数据通信的标准，而且它迅速的替代了EDI（Electronic Data Interchange，电子数据互换）系统，EDI系统曾经是企业间互换数据的主要媒体。有些分析家相信XML技术将成为创建和存储大多数文档的语言，包括Internet和非Internet领域。

1. XML和数据库

随着XML格式的数据量的不断增加，存储、检索、查询这类数据的需求也在增加。据估计，未来将存在两种主要的模式：以数据为中心和以文档为中心。在以数据为中心模式中，XML将作为结构化数据的存储和交换格式，而且按照一般的规律，很有可能由机器来代替人工的处理和阅读。在以数据为中心的模式中，事实上以XML格式存储和转换数据的可能性很小，可能常常使用其他的格式。在这种情况下，数据可以存储在关系、对象-关系或面向对象DBMS中。例如，Oracle已经完全将XML集成到Oracle 9i系统中。XML可以使用XMLType或CLOB/BLOB（Character/Binary Large Object，字符/二进制大对象）数据类型作为一个完整的文档存储在数据库中，或者可以被分解成一些主要元素并以上述方式存储起来。Oracle查询语言已经扩展成为允许基于XML的内容的查找。

在以文档为中心的模式中，文档是为人类消费（如书籍、报纸、电子邮件等）而设计的。由于信息本身的特征，很多数据可能是没有规律或是不完整的，而且它的结构可能会快速或不可预料的改变。但是，关系DBMS、对象-关系DBMS和面向对象DBMS不能很好地处理这类数据。内容管理系统是处理这类文档的最重要的工具。使用这样的系统，你可能会发现一个本地XML数据库。

> **本地XML数据库**（Native XML Database） 为XML文档（与文档中的数据不同）定义一个（逻辑）数据模型，并根据这个模型来存储和检索文档。至少，该模型必须包括元素、属性、PCDATA和文档顺序。XML文档必须是（逻辑）存储单位，尽管它不受潜在的物理存储模型的限制（因此没有排除传统的DBMS）。

2. 查询语言

正如前面提及的，DBMS厂商已经扩展了SQL来处理基于XML内容的查询。许多公司已

经联合起来制定将XML扩展到SQL中的标准，这个工作被称为SQL/XML，最初的工作将提交至ISO和ANSI。另外，W3C组成了一个XML Query Working Group（查询工作组）来为XML文档开发数据模型、在该模型之上的查询操作集以及基于这些查询操作（称为XQuery）的查询语言。查询运行在单个文档或是固定的文档集上，它们可以选择整个文档或是符合基于文档内容和结构的条件的文档子集。查询还能基于所选内容构造新文档。最后，能像访问数据库一样访问XML文档集。

与Web相关的技术在迅速变化，很可能我们在未来的几年中会看到这个领域重大的发展。

19.9 本章小结

- 高级的数据库应用包括计算机辅助设计（CAD）、计算机辅助制造业（CAM）、办公室信息系统（OIS）和多媒体系统、地理信息系统（GIS）以及灵活多变的交互式动态Web站点。
- 关系模型，尤其是关系系统有很多缺陷，如对现实世界实体的表达能力弱、语义过载、不能很好地支持业务规则、操作能力有限、处理递归查询困难以及阻抗不匹配等。RDBMS的有限的建模能力使得它不适合高级数据库应用。
- 一个分布式数据库是一个逻辑上相关的、共享的数据（以及数据描述）的一个集合，它们物理上分布在一个计算机网络上。DDBMS是一个能够透明地管理分布式数据库的软件。DDBMS和分布式处理不同，在分布式处理中整个网络只可以访问一个集中式DBMS。
- DDBMS的优点有，它反映了企业的结构，能够使远程数据更好地共享，提高了可靠性、可用性和性能，它更经济，而且提供了模块的增长。主要的缺点是成本高、复杂性高、缺乏标准和经验。
- 复制是在一个或多个站点上产生或生成数据的多个拷贝的过程。复制有很多好处，包括当集中的数据源超载时改进系统性能，增加可靠性和数据可用性，支持可移动计算和有助于决策支持的数据仓库。
- 一个面向对象DBMS（OODBMS）是OODB的管理器。OODB是在OODM中定义的对象持久的、共享的信息库。OODM是一个捕获面向对象编程支持的对象语义的数据模型。OODBMS的优点包括丰富的建模能力、可扩展性、消除阻抗失配、查询语言表达能力更强。
- 几家重要的厂商已经组成Object Data Management Group（对象数据管理组，ODMG）来定义OODBMS的标准。ODMG已经提出了一个对象模型用来指定数据库对象的语义的标准模型。这个模型很重要，因为它决定了能被OODBMS理解并执行的内置语义。使用这些语义来设计的类库和应用程序应该能够在支持对象模型的各种OODBMS中移植。
- 扩展关系数据模型不是单一的，而是出现了一系列的这种模型，每个模型的特点都依赖于它朝哪个方向以何种程度来扩展。但是，所有的模型确实都共享相同的基本关系表和查询语言，都集成了一些“对象”概念，而且有些模型还能在数据库中存储方法或存储过程/触发器。这些系统现在通常被称为对象-关系DBMS（ORDBMS）。
- 数据仓库是一个从一些不同的操作数据源中整理出的固定的/集成的企业数据的视图，它提供一系列最终用户能够使用的、支持从简单到复杂的查询的访问工具，以制定决策。数据仓库中的数据是面向主题的、集成的、随时间变化的、稳定的。数据集市是数据仓库的一个子集，它支持一个特定的部门或业务领域制定决策的需求。

- OLAP对大量多维数据集进行动态综合、分析和合并。OLAP描述了一种技术，这种技术使用综合数据的多维数据视图来为用于高级分析目的的决策信息提供快速访问。
- 数据挖掘是从大型数据库提取出有效的、以前不知道的、可理解的、可用信息的过程，可以用它来作出重大的商业决策。
- Web是到目前为止最受欢迎、功能强大的网络信息系统。为了避免在静态网页之外冗余地存储数据库中的数据，将操作数据库和Web环境集成起来是很重要的。集成Web和数据库有很多种方法，而且在技术成熟之后，该领域很可能在今后几年中会有很大的改变。
- XML（可扩展标记语言）是一种元语言（描述其他语言的语言），允许设计者创建自己定制的标记来提供在HTML中得不到的功能。XML可以影响程序设计中的每一部分，包括数据库管理。

复习题

19.1 讨论高级数据库应用的一般特征。

19.2 讨论为什么关系数据模型和关系DBMS的缺陷使得它们不适合于高级数据库应用。

19.3 解释DDBMS的含义，并讨论提供这样一个系统的动机。

19.4 比较并对比DDBMS和分布式处理。在哪种环境下，你会选择DDBMS而不是分布式处理？

19.5 讨论DDBMS的优点和缺点。

19.6 描述希望复制服务器提供的功能。

19.7 比较并对比复制的不同所有权模型。为你的答案举几个例证。

19.8 给出OODBMS的定义，OODBMS的优点和缺点是什么？

19.9 给出ORDBMS的定义，ORDBMS的优点和缺点是什么？

19.10 给出数据仓库的定义，讨论建立一个数据仓库的好处。

19.11 描述数据仓库中存储的数据的特征。

19.12 讨论数据集市和数据仓库的区别，并指出建立数据集市的主要理由。

19.13 讨论什么是联机分析处理（OLAP），以及OLAP和数据仓库的不同之处。

19.14 描述OLAP应用程序，并指出这些应用程序的特征。

19.15 讨论数据挖掘是怎样实现数据仓库的价值的。

19.16 为什么我们希望从操作性数据库存储的数据中动态地生成一个网页？列出Web与数据库集成的一些要求。

19.17 什么是XML？讨论管理基于XML的数据的方法。

附　录

附录A　可选的数据建模表示法
附录B　数据库设计方法学总结
附录C　高级数据库逻辑设计
附录D　文件组织和索引
附录E　常用数据模型

附录A　可选的数据建模表示法

在这个附录中你将学到：

- 可选的数据建模表示法。

在第7章，我们学习了怎样使用一种正在发展的流行的表示符号，称为UML（统一建模语言），来建立实体-关系模型(ER模型)。在这个附录中，你将看到两种其他的经常用于建立ER模型的表示法。第一种叫做Chen氏表示法，第二种叫做Crow的 Feet 表示法。我们通过在ER模型的主要概念中应用这些符号来说明这两种表示法，然后用图9-9所示的ER模型的例子来说明这些表示法。

A.1　使用Chen氏表示法的ER模型

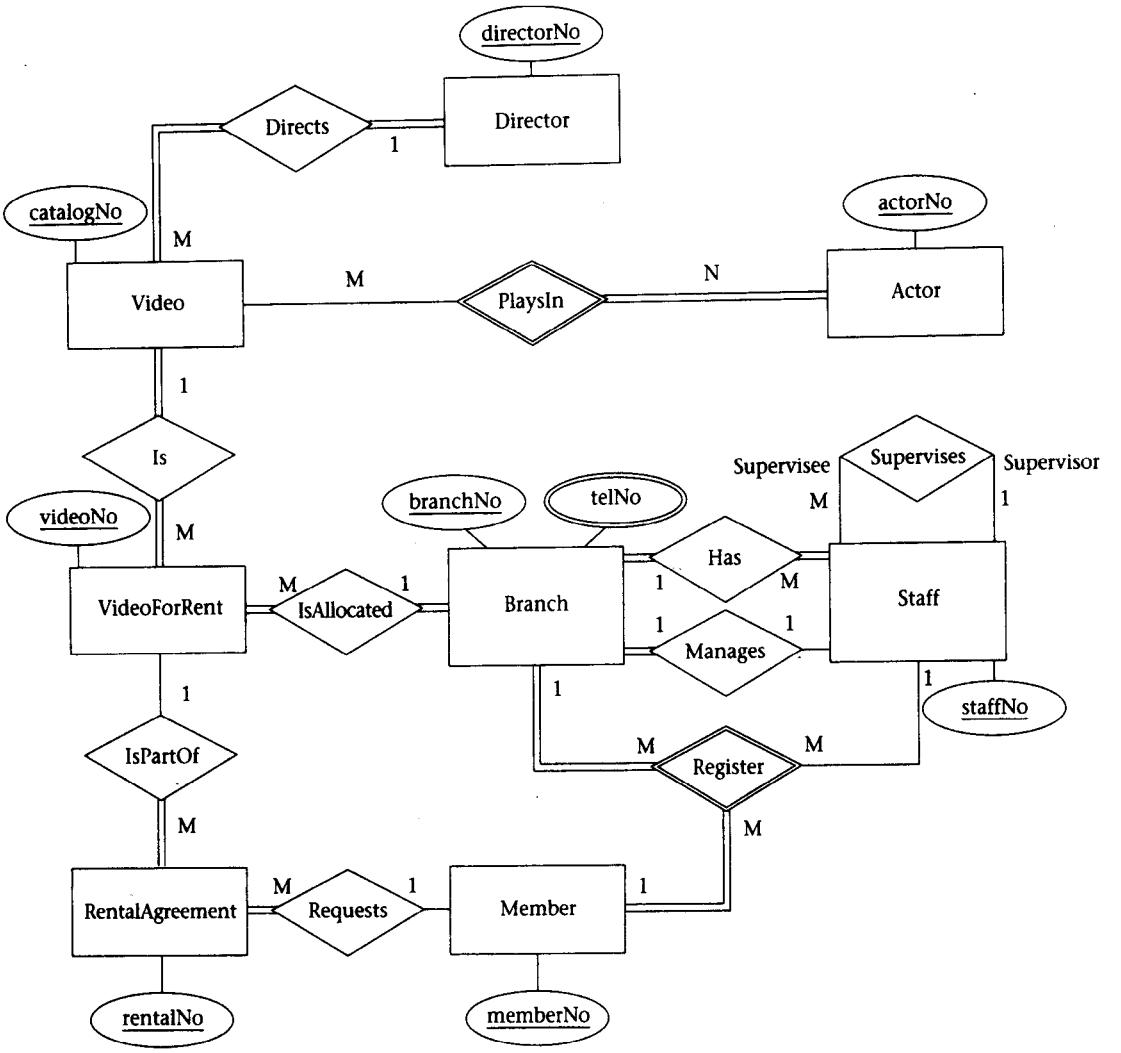
表A-1列出了在ER模型中的Chen氏表示法。图A-1显示了用Chen氏表示法来重新画的图9-9所示的模型。

表A-1　ER模型的Chen氏表示法

符　号	含　义
实体名	弱实体
实体名	强实体
关系名	关系
关系名	与弱实体关联的关系

（续）

符 号	含 义
关系名 角色名 实体名 角色名	带有角色名的递归关系，用来标识实体中的关系
属性名	属性
属性名	主键属性
属性名	多值属性
属性名	派生属性
1 1	一对一关系(1 : 1)
1 M	一对多关系(1 : M)
M N	多对多关系(M : N)
A 1 M B	用于实体A和B的带有强制参与的1:M关系
A 1 M B	实体A为可选参与，实体B为强制参与的1:M关系
A 1 M B	用于A、B的带有可选参与的1:M关系
A B C	实体A、B、C之间的三元关系
超类 子类 子类	泛化/特化。如果圆中包含“d”关系则是无连接。如果圆中包含“0”则是非连接，从超类引出的双线表示强制参与，单线表示可选参与



图A-1 用Chen氏表示法的ER模型

A.2 使用Crow的Feet表示法的ER模型

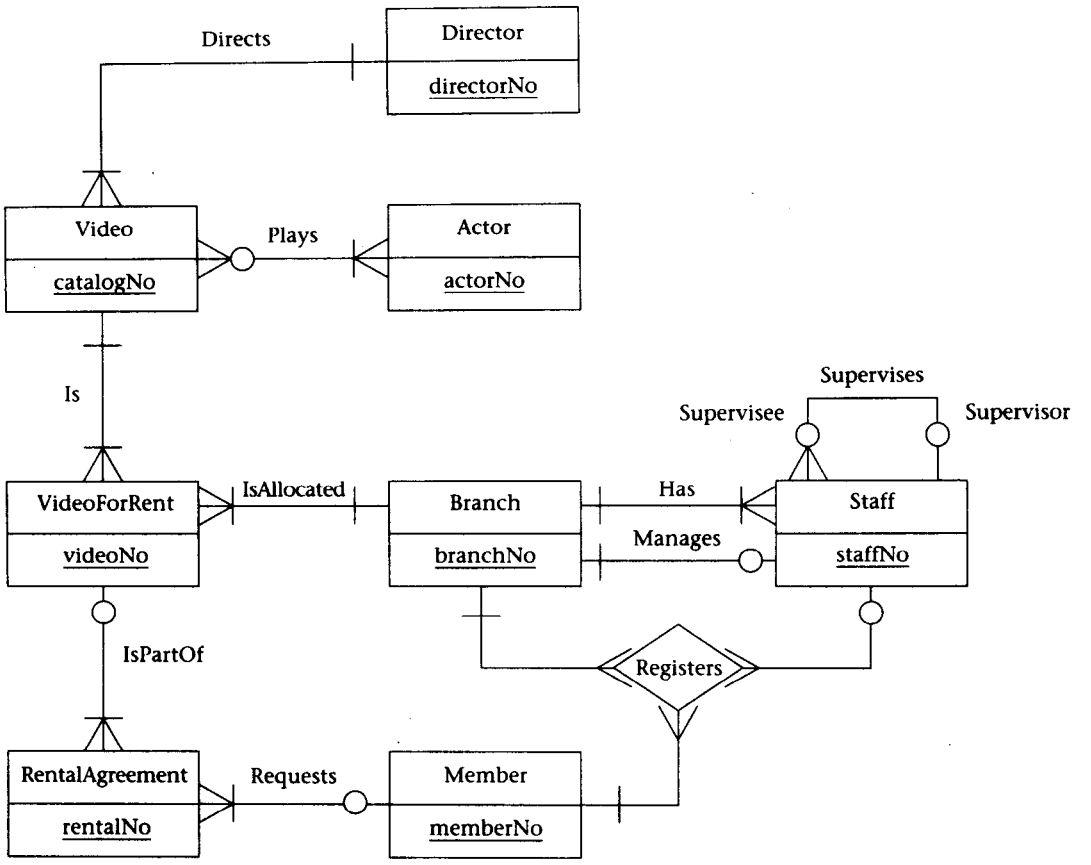
表A-2列出了在ER模型中的主要概念的Crow的 Feet表示法。图A-2显示了用Crow的Feet表示法来重新画的图9-9所示的模型。

表A-2 ER模型的Crow的 Feet表示法

符　　号	含　　义
实体名	实体
关系名	关系

（续）

符 号	含 义
关系名 角色名 实体名 角色名	带有角色名的递归关系，用来标识关系中实体的角色
实体名 属性名 属性1 属性2 属性n	属性列在实体符号下面 主键属性带有下划线 多值属性放在括号{}中
关系名	一对一关系
关系名	一对多关系
关系名	多对多关系
关系名 A B	用于实体A和B的带强制参与的一对多关系
关系名 A B	实体A带可选参与，B为强制参与的一对多关系
关系名 A B	实体A和B带可选参与的一对多关系
A B C	实体A、B、C之间的三元关系
超类 子类 子类	“框中框”约定广泛用于表示泛化/特化，并受多种CASE工具的支持，包括Oracle CASE Designer



图A-2 使用Crow的 Feet表示法重画的图9-9所示的ER模型

附录B 数据库设计方法学总结

在这个附录中你将学到:

- 数据库设计由两个阶段组成: 逻辑和物理数据库设计。
- 数据库设计方法学中包含的主要阶段的步骤。

在这本书中，我们介绍了一种关系数据库的数据库设计方法学。这种方法学有两个主要阶段，逻辑数据库设计和物理数据库设计，这些在第9章、第10章和第12章~第16章中有详细介绍。在这个附录中，我们为那些已经熟悉了数据库设计的读者总结了这两个阶段的设计步骤。

B.1 步骤1: 创建并检查ER模型

在分析阶段，将确定一系列的用户视图。根据交迭的数量，为了便于管理可能需要合并一些视图。这个步骤的目的是为每个这样的视图（可能是合并的）构建一个公司（或者是公司的一部分）的逻辑数据模型。

B.1.1 步骤1.1: 标识实体

标识和文档化公司视图中的主要的实体。

B.1.2 步骤1.2: 标识关系

标识已经确定的实体间存在的主要关系，确定关系的多样性约束。文档化关系，必要时使用ER模型。

B.1.3 步骤1.3: 标识实体或关系的有关属性

将属性与合适的实体和关系关联起来，表示简单/复合的属性、单值/多值属性和派生的属性，文档化属性。

B.1.4 步骤1.4: 确定属性域

确定ER模型中的属性的域，文档化属性域。

B.1.5 步骤1.5: 确定候选键、主键和备用键属性

为每个实体确定候选键，如果有多于一个的候选键，则选择其中的一个作为主键，其他的作为备用键。对每个实体的候选键、主键和备用键进行存档。

B.1.6 步骤1.6: 特化/泛化实体（可选步骤）

如果合适的话，标识超类和子类实体。

B.1.7 步骤1.7: 检查模型的数据冗余

检查ER模型确保没有冗余，特别要反复检查1:1关系并删除冗余关系。

B.1.8 步骤1.8：检查模型是否支持用户事务

确保ER模型支持用户所需的视图事务。

B.1.9 步骤1.9：与用户一起检查模型

B.2 步骤2：将ER模型映射为表

将ER模型映射为一组表，并检查表的结构。

B.2.1 步骤2.1：映射表

在这一步骤中，我们为步骤1创建的ER模型建立基本表来描绘实体、关系、属性和约束。表结构是从描述ER模型描述的信息中派生出来的。这些信息包括数据字典和其他描述模型的文档。同样，要为在给ER模型创建表的过程中产生的新的主键和候选键建立文档。

建立表的基本规则如下：

- 为每个实体建立一个包括这个实体的所有简单属性的表。
- 可用主键/外键机制来描述每个关系。为了确定外键，必须在关系中标识父实体和子实体。父实体把它的一个主键放置到子实体中，作为外键。表B-1给出了从ER模型创建表的规则。

表B-1 如何将实体、关系和多值属性表达为表的总结

实体/关系/属性	表达为表
强实体或弱实体	创建包含所有简单属性的表
1:*二元关系	将“一”端实体的主键复制到表达“多”端实体的表中，关系中的任何属性也复制到“多”端的表中
1:*递归关系	“一”端的实体和“多”端的实体是一样的，代表实体的表有主键的另一个拷贝，这个拷贝是被重命名的，并且有关系的其他属性
1:1二元关系：	
两端都是强制参与	将实体组合成一张表
一端是强制参与	将有可选参与的实体的主键复制到表达有强制参与的实体的表中，关系的任何属性也被复制到表达有强制参与的实体的表中
两端都是可选参与	没有更多的信息，将一个实体的主键拷贝到另一个实体中。但如果信息是可获得的，则将更具有强制参与的实体作为子实体
*:*二元关系/复杂关系	创建表达关系的表，此表中包含任何与关系有关的属性，将每个父实体中的主键复制到新表中作为外键
多值属性	创建一个表达多值属性的表，并将父实体的主键复制到新表中作为外键

- 对每个超类/子类关系，可以定义超类作为父实体而子类为子实体。关于如何最好地描述这样的关系为一个表或多个表，有各种不同的方法。对最合适方法的选择基于在超类/子类关系中的参与约束和无连接的约束，表B-2给出了如何从EER模型映射表的总结。

表B-2 表示基于参与和无连接约束的超类/子类关系的可选的选项

参与约束	无连接约束	需要的表
强制	非无连接约束{And}	一张表
可选	非无连接约束{And}	两张表：一个表用于超类，另一张表用于所有的子类
强制	非无连接约束{Or}	多张表：每个表用于超类/子类的组合
可选	非无连接约束{Or}	多张表：一张表用于超类、其他的用于每个子类

B.2.2 步骤2.2：用规范化方法检查表结构

这个步骤的目的是检查在步骤2.1中建立的每个表的列的分组。可以用规范化的规则检查每个表的组成。每个表最少应该是第三范式（3NF）的。

B.2.3 步骤 2.3：检查表是否支持用户事务

在这个步骤中，我们要确定表是否支持视图所需要的事务。视图需要的事务可以从用户需求说明书确定。

B.2.4 步骤2.4：检查业务规则

检查逻辑数据库设计中表达的所有业务规则。这些约束包括：指明需要的数据、属性域约束、实体完整性、多样性、参照完整性和其他业务规则。对所有的整体性约束进行存档。

B.2.5 步骤2.5：与用户讨论逻辑数据模型

确保逻辑数据模型是公司（或者是公司一部分）所需数据的真实的模型化描述。

B.2.6 步骤2.6：构建并检查全局逻辑数据模型[⊖]

合并单个的局部逻辑数据模型为一个完整的全局逻辑数据模型，以描述公司(或是公司的一部分)的数据要求。

1. 步骤2.6.1：合并局部逻辑数据模型为全局模型

合并单个的局部逻辑数据模型为一个完整的全局逻辑数据模型。这个步骤的一些基本的任务如下：

- 回顾实体/表的名字和内容以及它们的主键。
- 回顾关系/外键的名字和内容。
- 从局部数据模型合并实体/表。
- 包含（不是合并）对每个局部数据模型唯一的实体/表。
- 从局部逻辑数据模型中合并关系。
- 包含（不是合并）在每个局部逻辑数据模型中唯一的关系。
- 从局部数据模型合并关系/外键。
- 包含（不是合并）对每个局部数据模型唯一关系/外键。

⊖ 当使用视图集成方法创建多用户视图数据库时，只需步骤2.6。

- 检查漏掉的实体/表和关系/外键。
- 检查外键。
- 检查业务规则。
- 画出全局ER/表图。
- 更新文档。

2. 步骤2.6.2：检查全局逻辑数据模型

这个步骤的作用与步骤2.3和2.4相同，也是用规范化检查为全局数据模型建立的表的结构，然后检查这些表能否支持所有的用户事务。

3. 步骤3.3：检查未来的可变性

确定在可预见的将来是否存在有意义的变化并估计全局数据模型是否能适应这些变化。

4. 步骤2.6.4：与用户讨论全局数据模型

确保我们建立的全局数据模型是公司（或者是公司的一部分）需要的数据的真实描述。

B.3 步骤3：为目标数据库管理系统转换全局逻辑数据模型

从逻辑数据模型中产生一个基本表的工作集。

B.3.1 步骤3.1：设计基本表

决定如何在目标数据库管理系统中表述在逻辑数据模型中建立的基本表。为表的设计建立文档。

B.3.2 步骤3.2：设计派生数据的表示

考虑如何表达派生数据。需做出的选择是在每次需要时计算派生列还是引入冗余列来在表中存储派生数据。将派生数据的设计存档。

B.3.3 步骤3.3：设计其他业务规则

为目标DBMS设计其他的业务规则，将对其他业务规则的设计存档。

B.4 步骤4：选择文件组织方式和索引

决定要用来保存基本表的文件组织方式，也就是说，表和记录在辅存中的存取方式。考虑增加索引来提高性能。

B.4.1 步骤4.1：分析事务

理解在数据库中运行的事务的功能并分析重要的事务。

B.4.2 步骤4.2：选择文件组织方式

为每个基本表确定一个有效的文件组织方式。

B.4.3 步骤4.3：选择索引

确定增加索引是否会提高系统的整体性能。

B.5 步骤5：设计用户视图

设计在需求收集和分析阶段标识的用户视图。

B.6 步骤6：设计安全性机制

为数据库实现设计安全性机制，这些安全性机制是在需求收集和分析阶段由用户指定的。将对安全性机制的设计存档。

B.7 步骤7：引入受控冗余的考虑

确定引入受控的冗余以降低规范化规则是否会提高系统的整体功能。考虑重复的列或者连接一些表可以获得提高性能的目的。尤其是考虑合并一对一（1：1）关系，在一对多（1：*）的关系中重复非键列来减少连接，在一对多（1：*）的关系中重复外键列来减少连接，在多对多（*：*）关系中用重复列来减少连接，可以引入重复的组，建立摘要表和分区表。

B.8 步骤8：监视和调整运行的系统

监控运行的系统并提高系统性能来改进不合适的设计决定或者反映变化的需求。

附录C 高级数据库逻辑设计

在这个附录中你将学到:

- 怎样根据企业对数据的要求将局部逻辑数据模型合并到全局逻辑数据模型中。
- 怎样确保最后的全局模型能够准确表示被建模型企业需要的数据。

本附录描述了当创建一个有多个用户视图并且还要对这些用户视图使用视图集成方法进行管理的复杂数据库系统时，应该如何做。本附录假定你已经用第9章和第10章中介绍的数据库设计方法学的步骤1和步骤2建立好了表达一个和多个用户视图的局部数据模型。

在第6章中，我们确定了StayHome数据库系统的几个用户视图，即Director、Manager、Supervisor、Assistant和Buyer。在对每个用户视图的要求进行分析后，我们决定使用集中式和视图集成方法来管理这些用户视图。我们决定使用集中式方法来合并Manager、Supervisor、Assistant用户视图为一个用户视图集合，这个用户视图集合被称为Branch，并将Director和Buyer视图合并到称为Business的用户视图集合中。在第9章和第10章中，我们使用Branch用户视图来说明应用方法学中的步骤1和步骤2来构建一个局部逻辑数据模型的过程。E-R图如图9-9所示，并且在图10-11中显示了对各表的详细描述。

在本附录中，我们首先给出了有关StayHome中的Business用户视图的用户需求规格说明书。我们不演示为这些用户视图的集合创建局部逻辑数据模型的步骤，但我们提出了逻辑模型的重要组件，即E-R图和基于该模型的表的描述。然后，我们用Branch和Business用户视图的局部逻辑数据模型来阐明怎样合并数据模型。

C.1 StayHome的Business用户视图

在这一节中，我们提出了有关StayHome的Business用户视图的用户需求规格说明书和相应的局部逻辑数据模型。

提示 *你会发现阅读下一节中的需求是非常有用的，然后可以自己试着用方法学中的步骤1和步骤2，然后可以将自己的解决方案和我们例子中的解决方法比较一下。*

C.1.1 用户的需求规格说明书

Business用户视图的需求规格说明书列出了两部分内容：一部分描述了在Business用户视图中要用到的数据信息，另一部分提供了怎样使用数据的例子（即在数据上执行的事务）。

1. 数据要求

在StayHome的分公司中包含的详细信息包括分公司的地址和电话号码。每个分公司都有一个全公司唯一的分公司号。

StayHome的每个分公司都有包含经理（Manager）在内的员工，在员工的详细信息有姓名、职位和年薪。每个员工都有一个在全公司唯一的员工号。

StayHome的每个分公司保存有一些录像。录像的详细信息包括目录号、录像号、标题、种类、日租金和购买价格。目录号唯一地标识每个录像。但大多数情况下，在一个分公司中每个录像都会有几盘相同的拷贝，这些拷贝就使用录像号来标识。

StayHome的每个分公司从录像供应商那里得到录像。录像供应商的详细信息包括供应商号、名字、地址、电话号码和状态。公司从这些供应商那里订购录像，订单的详细信息包括订单号、供应商号、供应商地址、录像目录号、录像标题、录像购买价格、数量、订购日期、发货日期和接收该批录像的分公司地址。

StayHome的顾客必须首先注册成为StayHome的某个分公司的会员。会员的详细信息包括名字、地址和该会员注册成的日期。每个会员有一个给定的会员号，用于唯一地标识该会员，甚至当该会员还在其他分公司注册的时候，也使用这个会员号。

每个出租录像的详细信息包括出租号、全名、成员号、录像号、标题、日租金以及出租和归还的日期。其中出租号作为唯一标识。

2. 事务需求

(1) 数据录入

1) 输入新出版的录像的详细信息（例如Return of the King（王者归来）这部录像的详细信息）。

2) 输入录像供应商的详细信息（例如WorldView Videos供应商的信息）。

3) 输入录像订单的详细信息（例如B002分公司定购了10盘Return of the King）。

(2) 数据更新/删除

4) 更新/删除已有录像的详细信息。

5) 更新/删除已有的录像供应商的详细信息。

6) 更新/删除已有的录像订单的详细信息。

(3) 数据查询

7) 列出所有分公司中职员的名字、职位和工资，按分公司号排序。

8) 列出给定分公司的经理的名字和电话号码。

9) 列出给定分公司中所有的录像的目录号和标题，按标题排序。

10) 列出给定分公司号的给定录像的拷贝的数量。

11) 列出每个分公司会员的数目，按分公司号排序。

12) 列出每个分公司中今年加入的会员的数目，按分公司号排序。

13) 列出某两个日期之间每个分公司出租的录像的数目，按分公司号排序。

14) 列出给定分公司中，每种类别中录像的数目，按类别排序。

15) 列出所有录像供应商的名字、地址和电话号码，按供应商号码排序。

16) 列出录像供应商的名字和电话号码。

17) 列出给定的录像供应商的所有订单的详细信息，按照订单的日期排序。

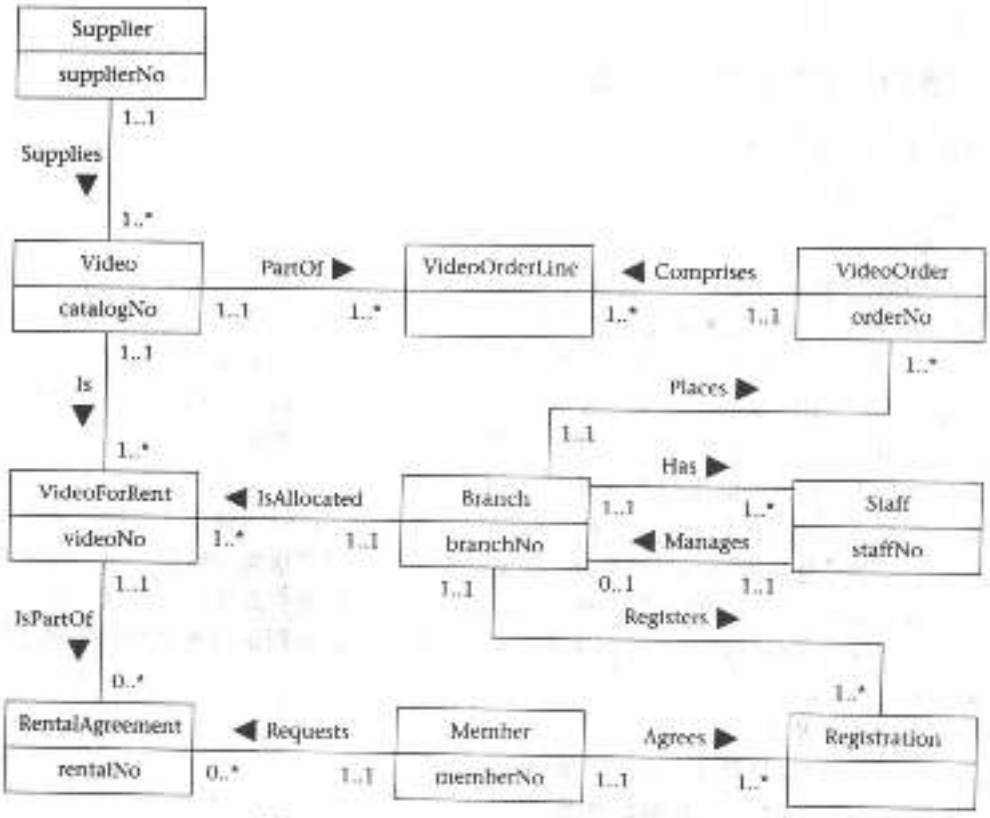
18) 列出某个日期所签订的所有订单的详细信息。

19) 列出在给定日期之间，每个分公司的全部日租金，按照分公司号排序。

C.1.2 局部逻辑数据模型

正如我们刚刚提到的，不需要再为StayHome的Business视图构建局部逻辑数据模型，我们假定在方法学的步骤1和步骤2中已经构造了该模型，并且给出了逻辑模型的重要部分，即

- ER图，参见图C-1。
- 表，参见图C-2。



图C-1 StayHome 的Business视图的ER模型

Branch (branchNo, address, telNo, mgrStaffNo) **Primary Key** branchNo **Alternate Key** telNo **Foreign Key** mgrStaffNo **references** Staff(staffNo)	**Member** (memberNo, name, address) **Primary Key** memberNo
Registration (branchNo, memberNo, dateJoined) **Primary Key** branchNo, memberNo **Foreign Key** branchNo **references** Branch(branchNo) **Foreign Key** memberNo **references** Member(memberNo)	**RentalAgreement** (rentalNo, dateOut, dateReturn, memberNo, videoNo) **Primary Key** rentalNo **Alternate Key** (memberNo, videoNo, dateOut) **Foreign Key** memberNo **references** Member(memberNo) **Foreign Key** videoNo **references** Video(videoNo)
Staff (staffNo, name, position, salary, branchNo) **Primary Key** staffNo **Foreign Key** branchNo **references** Branch(branchNo)	**Supplier** (supplierNo, sName, sAddress, sTelNo, status) **Primary Key** supplierNo **Alternate Key** sTelNo
Video (catalogNo, title, category, dailyRental, price, supplierNo) **Primary Key** catalogNo **Foreign Key** supplierNo **references** Supplier(supplierNo)	**VideoForRent** (videoNo, available, catalogNo, branchNo) **Primary Key** videoNo **Foreign Key** catalogNo **references** Video(catalogNo) **Foreign Key** branchNo **references** Branch(branchNo)
VideoOrder (orderNo, dateOrdered, dateRequired, branchNo) **Primary Key** orderNo **Foreign Key** branchNo **references** Branch(branchNo)	**VideoOrderLine** (orderNo, catalogNo, quantity) **Primary Key** (orderNo, catalogNo) **Foreign Key** orderNo **references** VideoOrder(orderNo) **Foreign Key** catalogNo **references** Video(catalogNo)

图C-2 StayHome的Business视图的表

现在让我们使用Branch和Business局部逻辑数据模型来构建StayHome的全局逻辑数据模型。我们下面将描述合并两个数据模型的过程，包括给第10章中描述的逻辑数据库设计方法

学的步骤2增加一个附加步骤。

C.2 步骤2.6：构建并检查全局逻辑数据模型

目标 将每个局部逻辑数据模型合并成一个全局逻辑数据模型，该模型代表了需要建立模型的组织（或组织的一部分）的数据要求。

在步骤2的这个子步骤中，将构建一个全局逻辑数据模型，它通过合并数据库中每个用户视图产生的局部逻辑数据模型，来表示所有用户视图。但是，如果你使用集中式方法来管理多个用户视图，那么一个局部逻辑数据模型就代表了两个或更多用户视图的合并后的要求。将模型合成在一起，检查在全局模型上使用的所有表都是规范化的，并且仍然能够支持需要的事务是很必要的，就像第10章步骤2.2和步骤2.3中做的那样。但是，你只需检查那些在合并过程中发生了变化的部分。在一个大系统中，这将显著减少进行重新检查的工作量。

尽管每个局部逻辑数据模型都应该是正确的、全面的、内容明确的，但是毕竟每个模型仅仅是一个或多个（而不是全部）数据库用户视图的表达。换句话说，这个模型从严格意义上来说并不是一个企业组织功能的模型，而是一个或多个用户视图的模型，所以它可能是不完整的。这就意味着，当你查看用户视图的全部集合时，可能会发现不一致现象或是重叠现象。因此，当将局部逻辑数据模型合并成一个全局模型时，必须解决各个用户视图之间存在的冲突和重叠现象。

在步骤2.6中涉及到的工作有：

- 步骤2.6.1：将局部逻辑数据模型合并成全局模型
- 步骤2.6.2：检查全局逻辑数据模型
- 步骤2.6.3：考虑未来的增长情况
- 步骤2.6.4：和用户共同评审全局逻辑数据模型

C.2.1 步骤2.6.1：将局部逻辑数据模型合并成全局模型

目标 将每个局部逻辑数据模型合并成一个全局逻辑数据模型。

现在，已经为每个局部逻辑数据模型建立了ER图、一组表、一个数据字典和一些描述了数据约束的支持文档。在这个步骤中，为了合并模型，可以使用这些已有的内容来标识模型间的相似和不同之处。

对于一个拥有较少的实体/表的简单数据库系统来说，比较局部模型、把它们合并到一起并解决存在的分歧是一件容易的事。但是，在一个大型系统中，必须采用一个更加系统化的实现过程。下面提出了一种方案，可以用于将局部模型合并在一起并解决所有可以发现的不一致性问题。这个方案的关键工作包括：

1) 检查实体/表的名字和内容，以及它们的主键。
2) 检查关系/外键的名字和内容。
3) 合并局部数据模型中的实体/表。
4) 包含（不合并）对每个局部数据模型来说唯一的实体/表。
5) 合并来自局部数据模型的关系/外键。
6) 包含（不合并）对每个局部数据模型来说唯一的关系/外键。

7) 检查是否有丢失的实体/表和关系/外键。

8) 检查外键。

9) 检查完整性约束。

10) 画出全局ER图和表。

11) 更新文档。

在上述工作中，我们使用了术语“实体/表”和“关系/外键”，因为你可能希望:

- 检查ER图和它们支持的文档。
- 检查根据ER图和支持文档创建出来的表。
- 使用上述两种信息源的综合。

提示 也许合并几个局部数据模型的最简单的方法是，首先合并其中的两个模型成为一个新的模型，然后相继合并其余的局部数据模型，直到将所有的局部模型合并到最终的全局数据模型中为止。这可能要比同时合并所有的局部数据模型更简单。

为了确保可比较性，创建每个局部模型时都应遵循方法学中的步骤1和步骤2是很重要的。

- 检查实体/表的名字和内容，以及它们的主键

通过检查数据字典，检查出现在局部数据模型中的实体/表的名字。当有两个或多个实体/表时，可能会出现如下问题:

1) 名字相同，但实际上并不是同一个实体/表（异物同名）。

2) 实体/表相同，但却有不同的名字（同物异名）。

为了解决这些问题，比较每个实体/表中的数据内容是必要的。特别地，可以使用主键（或候选键）去标识视图间命名不同的等价的实体/表。StayHome中Branch和Business视图的实体/表和主键的比较见表C-1。每个视图的共同的实体/表用黑体显示。

表C-1 StayHome中Branch和Business视图的实体/表及主键的比较

Branch用户视图		Business用户视图	
表	主键	表	主键
Branch	**branchNo**	**Branch**	**branchNo**
Staff	**staffNo**	**Staff**	**staffNo**
Telephone	telNo		
Video	**catalogNo**	**Video**	**catalogNo**
VideoForRent	**videoNo**	**VideoForRent**	**videoNo**
		Supplier	supplierNo
		VideoOrder	orderNo
		VideoOrderLine	orderNo, catalogNo
RentalAgreement	**rentalNo**	**RentalAgreement**	**rentalNo**
Member	**memberNo**	**Member**	**memberNo**
Registration	**branchNo, memberNo**	**Registration**	**branchNo, memberNo**
Actor	actorNo		
Role	catalogNo, actorNo		
Director	directorNo		

- 检查关系/外键的名字和内容

这和描述实体/表的工作相同，StayHome中Branch和Business用户视图中的关系/外键的比较如表C-2所示。在两个视图中公共的关系/外键用粗体显示。

表C-2 StayHome中Branch和Business视图中关系/外键的比较

Branch用户视图			Business用户视图		
子表	外键	父表	子表	外键	父表
Branch	**MgrStaffNo→**	**Staff(staffNo)**	**Branch**	**MgrStaffNo →**	**Staff(staffNo)**
Telephone	branchNo→	Branch (branchNo)			
Registration	**branchNo→**	**Branch(branchNo)**	**Registration**	**branchNo →**	**Branch (branchNo)**
	memberNo→	**Member(memberNo)**		**memberNo→**	**Member(memberNo)**
	staffNo→	Staff(staffNo)			
Staff	**branchNo→**	**Branch (branchNo)**	**Staff**	**branchNo→**	**Branch (branchNo)**
	supervisor StaffNo→	Staff(staffNo)			
Video	directorNo→	Director(directorNo)	Video	supplierNo→	Supplier(supplierNo)
VideoForRent	**catalogNo→**	**Video (catalogNo)**	**VideoForRent**	**catalogNo →**	**Video (catalogNo)**
	branchNo→	**Branch(branchNo)**		**branchNo→**	**Branch (branchNo)**
RentalAgreement	**memberNo→**	**Member(memberNo)**	**RentalAgreement**	**memberNo→**	**Member(memberNo)**
	videoNo→	**VideoForRent (videoNo)**		**videoNo→**	**VideoForRent(videoNo)**
			VideoOrder	branchNo→	Branch (branchNo)
			VideoOrderLine	orderNo→	Video0rder(orderNo)
				catalogNo→	Video (catalogNo)
Role	actorNo →	Actor(actorNo)			
	catalogNo→	Video (catalogNo)			

• 从局部数据模型中合并实体/表

应该检查模型中要被合并的每个实体/表的名称和内容，以决定是否这些实体/表代表同样的事物，因而能被合并。这项任务一般包括如下步骤:

1) 合并名称相同、主键相同的实体/表。

2) 合并名称相同、主键不同的实体/表。

3) 合并名称不同、主键相同或不同的实体/表。

4) 包含（不合并）对每个局部数据模型而言是唯一的实体/表

前面的任务应该已标识出所有相同的实体/表。而所有剩余的实体/表不用进行任何变化就可以加入到全局模型中。

• 从局部数据模型中合并关系/外键

在这个步骤中，检查数据模型中的每一个关系的名称和目标。在合并关系/外键之前，消除关系中的冲突是很重要的，比如多样性约束不同。这个步骤主要包括合并名称相同、目标相同的关系/外键，然后合并名称不同、目标相同的关系/外键。

• 包含（不合并）对每个局部数据模型来说是唯一的关系/外键

前面的工作应该标识了相同的关系/外键（通过定义，它们必须是相同实体/表间的关系，这些实体/表在前边已经被合并了）。所有剩下的关系不用做任何变化就可以加入全局模型中。

• 检查是否有遗漏的实体/表和关系/外键

也许创建全局模型时最复杂的任务之一就是标识不同局部数据模型间缺失的实体/表和关系/外键。如果存在一个关于公司的企业数据模型，模型中确定了组织使用的所的重要数据，这样可以发现在任何局部数据模型中都没有出现的实体/表和关系。防止出现该情况的方法为，

当与用户讨论具体视图时，让他们特别注意是否有存在于其他视图的实体/表和关系/外键。否则，就要检查每个实体/表的属性/列，并在其他局部数据模型中寻找对实体/表的引用。你可能会发现在一个局部数据模型中的实体/表的某一个属性/列对应另一个局部数据模型中的实体/表的主键、备用键、甚至是非键属性/列。

• 检查外键

在这个步骤中，已经合并了实体/表和关系，改变了主键，并且已经标识了新的关系。检查子表中的外键是否仍然是正确的，并且当需要时作必要的修改。StayHome数据库的代表全局逻辑数据模型的所有表如图C-3所示。

Actor (actorNo, actorName) **Primary Key** actorNo	**Branch** (branchNo, street, city, state, zipCode, mgrStaffNo) **Primary Key** branchNo **Alternate Key** zipCode **Foreign Key** mgrStaffNo **references** Staff(staffNo)
Director (directorNo, directorName) **Primary Key** directorNo	**Member** (memberNo, fName, lName, address) **Primary Key** memberNo
Registration (branchNo, memberNo, staffNo, dateJoined) **Primary Key** branchNo, memberNo **Foreign Key** branchNo **references** Branch(branchNo) **Foreign Key** memberNo **references** Member(memberNo) **Foreign Key** staffNo **references** Staff(staffNo)	**RentalAgreement** (rentalNo, dateOut, dateReturn, memberNo, videoNo) **Primary Key** rentalNo **Alternate Key** memberNo, videoNo, dateOut **Foreign Key** memberNo **references** Member(memberNo) **Foreign Key** videoNo **references** Video(videoNo)
Role (catalogNo, actorNo, character) **Primary Key** catalogNo, actorNo **Foreign Key** catalogNo **references** Video(catalogNo) **Foreign Key** actorNo **references** Actor(actorNo)	**Staff** (staffNo, name, position, salary, branchNo, supervisorStaffNo) **Primary Key** staffNo **Foreign Key** branchNo **references** Branch(branchNo) **Foreign Key** supervisorStaffNo **references** Staff(staffNo)
Supplier (supplierNo, name, address, telNo, status) **Primary Key** supplierNo **Alternate Key** telNo	**Telephone** (telNo, branchNo) **Primary Key** telNo **Foreign Key** branchNo **references** Branch(branchNo)
Video (catalogNo, title, category, dailyRental, price, directorNo, supplierNo) **Primary Key** catalogNo **Foreign Key** directorNo **references** Director(directorNo) **Foreign Key** supplierNo **references** Supplier(supplierNo)	**VideoForRent**(videoNo, available, catalogNo, branchNo) **Primary Key** videoNo **Foreign Key** catalogNo **references** Video(catalogNo) **Foreign Key** branchNo **references** Branch(branchNo)
VideoOrder (orderNo, dateOrdered, dateReceived, branchNo) **Primary Key** orderNo **Foreign Key** branchNo **references** Branch(branchNo)	**VideoOrderLine** (orderNo, catalogNo, quantity) **Primary Key** orderNo, catalogNo **Foreign Key** orderNo **references** VideoOrder(orderNo) **Foreign Key** catalogNo **references** Video(catalogNo)

图C-3 StayHome的全局逻辑数据模型的表结构

• 检查完整性约束和业务规则

检查全局逻辑数据模型中的完整性约束和业务规则与最初在用户视图中指定的约束是否有冲突，如果标识了任何新的关系或者创建了新的外键，则要确保指定了适当的参照完整性约束。任何冲突都必须和用户一起协商解决。

• 画出全局ER图和表

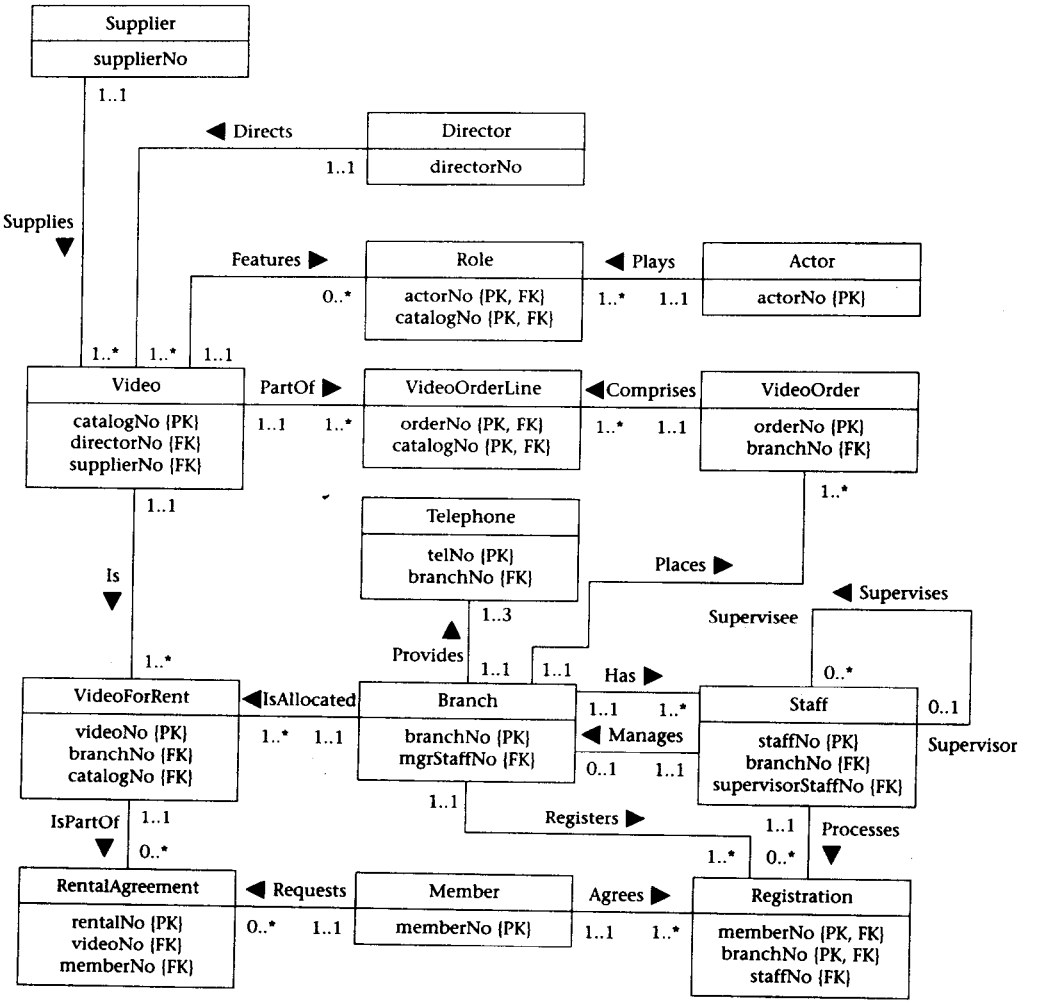
现在可以绘制一张最终的代表所有合并的局部逻辑数据模型的图了。如果是基于表所作的合并，那么我们称最终的结果图为“全局表图”，它列出了主键和外键。如果是使用局部ER图进行合并，则结果图就简单地称为“全局ER图”。StayHome数据库的全局表图如图C-4所示。

• 更新文档

更新文档以反映全局数据模型开发阶段发生的变化。使文档保持最新状态，反映出当前的数据模型是很重要的。如果后来模型中发生了变化，包括在数据库实现阶段或是在数据库维护阶段发生变化，文档都应该同时被更新。过期的信息在以后会带来相当大的困扰。

C.2.2 步骤2.6.2：检查全局逻辑数据模型

目标 检查从全局逻辑数据模型创建的表的结构是否进行适当的规范化，并支持必需的事务。



图C-4 StayHome数据库系统的全局表图

在这个步骤中，用规范化方法检查为全局数据模型创建的表的结构，并检查这些表是否能够支持所有的用户事务，这些操作同第10章中步骤2.2和2.3中的操作一样。但是，你只需要检查在合并阶段模型发生变化的部分就可以了。在一个大型系统中，这对于减少需要重新检查的工作量是很重要的。

C.2.3 步骤2.6.3：考虑未来的增长情况

> **目标** 确定在可预见的未来是否会有较大的变化，并且估计全局逻辑数据模型是否可以容纳这些变化。

全局逻辑数据模型容易扩展是很重要的。如果模型只能支持当前的需求，那么模型的生命期可能就会很短，并且为了容纳新的需求，可能需要做较多的工作。创建一个可扩展的模型是很重要的，同时这个模型应该可以继续发展以支持新的需求，并且对已经存在的用户的影响比较小。当然，这很难实现，因为企业并不知道将来想要做什么。即使企业知道要做什

么，为了容纳未来可能的增长，在时间和金钱上都要付出很昂贵的代价。因此，应该有所选择地在某些方面进行扩充。

C.2.4 步骤2.6.4：和用户共同评审全局逻辑数据模型

目标 确保全局逻辑数据模型确实是企业数据需求的表达。

全局逻辑数据模型现在应该已经是完整的和准确的。应该与用户评审模型和描述模型的文档，以确保它确实反映了企业的数据需求。

现在你已经准备从逻辑设计转换为物理设计了，这些内容包含在方法学的步骤3到步骤8中，我们在第12章~第16章中进行了介绍。

C.3 附录总结

逻辑数据库设计方法学中的步骤2中的2.6步是可选的，而且只有当创建一个相当复杂的数据库系统，这个复杂的数据库系统可能包含多个多变的用户视图，并采用视图集成方法来管理时，才使用这个步骤。

附录D 文件组织和索引

在这个附录中你将学到:

- 主存储和辅助存储的区别。
- 文件组织的含义和访问方法。
- 堆文件是如何组织的。
- 有序文件是如何组织的。
- 哈希文件是如何组织的。
- 索引是什么以及它是如何用于提高数据库检索速度的。
- 主索引、二级索引和聚簇索引间的区别。
- 多级索引是如何组织的。
- B^+树是如何组织的。
- 位图索引是如何组织的。
- 连接索引是如何组织的。
- 如何选择合适的文件组织。
- 索引聚簇和哈希聚簇是如何组织的。

第13章中介绍的物理数据库设计方法学中的步骤4.2和4.3关心的就是对合适的文件组织以及对已经创建好的表达公司（或公司的一部分）的数据需求的基本表的索引的选择。在这部分附录中，我们介绍在辅助存储设备中的数据库的物理存储的概念。计算机的主存储器，即主存，不适合于存储数据库。尽管主存储器的访问速度比辅存要快得多，但是即使用主存来存储一个基本的数据库所需的数据量，其空间都是不够大而且是不可靠的。由于主存的数据会随着电源的断开而丢失，所以我们认为主存是不稳定的存储器。相反，在辅存中的数据即使在电源断开的情况下也可以保留，因此，被认为是稳定的存储器。此外，主存的每个存储单元的价格要比磁盘贵得多。

在下面的几节中，我们介绍物理存储的基本概念以及文件组织的主要类型，即堆（无序的）、顺序的（有序）和哈希文件。在D.5节，我们讨论了索引是怎样被用来提高数据库检索性能的，特别是考虑了多级索引、B^+树、位图索引和连接索引。这个附录中的例子均来自6.4.4节介绍的StayHome案例研究。

D.1 基本概念

辅存中的数据库被组织成一个或多个文件，每个文件由一条或多条记录组成，而每条记录又由一个或多个字段组成。典型的情况是，一条记录对应一个实体的出现，而一个字段对应一个属性或列。考虑从StayHome例子中得到的Staff表，如图D-1所示。

我们可能希望表中的任何一条记录都映射到保存Staff表的操作系统文件中的一条记录，记录中的每个字段被存储成Staff表中的一列。当需要从DBMS中获取一条记录时，例如员工号为S0003的记录，DBMS将此条逻辑记录映射到一个物理记录，并利用操作系统的文件访问例程检索物理记录到主存中的DBMS的缓冲区中。

staffNo	name	position	salary	branchNo
S1500	Tom Daniels	Manager	46000	B001
S0003	Sally Adams	Assistant	30000	B001
S0010	Mary Martinez	Manager	50000	B002
S3250	Robert Chin	Supervisor	32000	B002
S2250	Sally Stern	Manager	48000	B004
S0415	Art Peters	Manager	41000	B003

图D-1 StayHome示例中的Staff表

物理记录是磁盘和主存之间的交换单元。一般来说，一条物理记录由多条逻辑记录组成，依赖于逻辑记录的大小，一条逻辑记录可能只对应一条物理记录，但对于大的逻辑记录可以跨越多条物理记录。经常使用术语“块”和“页”来代替物理记录，在本附录的其余部分我们使用术词“页”。例如，图D-1中的Staff记录可以存储在两个页上，如图D-2所示。

staffNo	name	position	salary	branchNo	页
S1500	Tom Daniels	Manager	46000	B001	
S0003	Sally Adams	Assistant	30000	B001	1
S0010	Mary Martinez	Manager	50000	B002	
S3250	Robert Chin	Supervisor	32000	B002	
S2250	Sally Stern	Manager	48000	B004	2
S0415	Art Peters	Manager	41000	B003	

图D-2 Staff表在页中的存储

数据记录在文件中存储和存取的顺序依赖于文件的组织。

文件组织（File Organization） *当文件存储在磁盘上时在文件中安排记录的顺序。*

文件组织的主要类型有:

- 堆（无序）文件: 记录在磁盘中的存储没有特定顺序。
- 有序（排序的）文件: 记录是按照指定字段的值排序的。
- 哈希文件: 记录是根据哈希函数的值在磁盘中存储的。

对应每一种文件组织方式，都有一组存取方法。

存取方法（Access Method） *该步骤涉及从文件中存储和检索记录。*

由于某些存取方法只能用在某一种文件组织中（例如，我们不能对一个没有索引的文件使用索引存取方法），所以文件组织和存取方法这两个术语是可以交替使用的。在这个附录的后续部分，我们将讨论文件组织的主要类型，并提供了选择合适的索引和文件组织的方针。

D.2 堆文件

一种无序的文件，有时称为堆文件，是最简单的文件组织方式。记录按照它们插入的顺序放置在文件中。一条新记录被插入到文件的最后一页，如果最后一页没有多余的空间，那么就在文件中添加一个新页。这对插入操作是非常有效的。然而，由于堆文件中没有根据某

一字段的值排出特定顺序，因此存取一条记录必须要使用线性查找。线性查找包括从文件中读出页一直到找到所需的记录为止。这就使得从包含许多页的堆文件中进行检索的速度相对来说比较慢，除非这个检索包括文件中所有记录的大部分。

要删除一条记录，首先需要检索出所需的页，然后将这条记录标记上已删除标记，并把页写回到磁盘。被删除的记录的空间是不可再用的。因此，随着删除动作的发生，系统的性能也日益恶化了。这就意味着，堆文件需要数据库管理员（DBA）定期地对文件进行重组织，收回那些被删除记录的未使用空间。

对于向表中插入大批量数据的操作，堆文件是最好的文件组织方式之一，因为记录被顺序地插入到文件的末尾。那样不会在计算记录要插入哪一页时产生额外消耗。

D.3 有序文件

文件中的记录可以根据一个或多个字段的值来排序，组成一个按键值排序的数据集，被用来对文件进行排序的字段称为排序字段（ordering field）。如果排序字段也正好是文件的主键，那么也就保证了在每条记录中都有唯一的值，这个字段也被称为文件的排序字段。例如，考虑下面的SQL查询语句:

```
SELECT *
FROM Staff
ORDER BY staffNo;
```

如果Staff表中的记录已经根据有序列staffNo的值排序了，那么就有可能减少查询的执行时间，因为没有必要再排序了。

注意 尽管在2.2节中我们说明了关系模型的记录是无序的，但这个是指由外部的（逻辑的）性质，而不是可执行的（物理的）的性质。物理上总是第一条记录、第二条记录、一直到第n条记录。

如果记录是按照staffNo来排序的，在某些情况下，我们可以使用二分查找法来执行基于staffNo查询条件的查询，例如，考虑下面的SQL查询语句:

```
SELECT *
FROM Staff
WHERE staffNo = 'S1500';
```

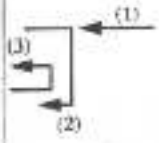
如果我们使用图D-1中显示的数据并且假定每一页上只有一条记录，就会得到如图D-3所示的排好序的结果。二分查找的过程如下:

1) 检索文件的中间页。检查所需的记录是否在这一页的第一条记录和最后一条记录之间。如果是，则不需要再检索其他的数据页了。

2) 如果这一页的第一条记录的主键值比所需的记录的主键值大，则所需记录（如果存在的话）必然在前面的数据页中。因此，我们重复上面的步骤，从文件前面的中间一页开始检索。

3) 如果这一页的最后一条记录的主键值比所需的记录的主键值小，则所需记录必然在后面的数据页中。因此，我们重复上面的步骤，从文件后面的中间一页开始检索。这样，检索的数据页中的一半的查询空间都被消除了。

页	staffNo	name	position	salary	branchNo
1	S0003	Sally Adams	Assistant	30000	B001
2	S0010	Mary Martinez	Manager	50000	B002
3	S0415	Art Peters	Manager	41000	B003
4	S1500	Tom Daniels	Manager	46000	B001
5	S2250	Sally Stern	Manager	48000	B004
6	S3250	Robert Chin	Supervisor	32000	B002



图D-3 在有序的Staff文件中的二分检索

在我们的例子中，中间页是第3页，检索页上的记录的主键值（S0415）不等于所需的记录的主键值（S1500）。第3页的主键值比所需的记录的主键值要小，因此，我们就可以从检索中省略掉文件的前一半页面。现在我们检索后面的页面的中间页，也就是第5页，这一次主键的值（S2250）比S1500大，这样我们又可以省掉了这次查找空间的后一半。我们现在检索剩余空间的中间页（第4页），就是我们所需要的记录。

一般来说，二分查找比线性查找更有效率。但是，二分查找多用于主存中数据的检索，而对辅存中的数据使用的就比较少了。

在一个有序的文件中插入和删除记录是一件很麻烦的事情，因为要保持记录的有序性，要插入一条新记录，我们必须按照顺序找到合适的插入位置，并且要找到插入的空间。如果在找到的页面中有足够的空间可以插入新记录，那么这一页就可以重新排序并且写回到磁盘中。如果不是，那么就有必要移动一条或多条记录到下一个页面中。同样，如果下一个页面也没有多余的空间，那么就需要将记录再移动到下一个页面中，依次类推。

在一个大的文件的接近开始的位置插入一条记录可能是一项非常费时的工作。一种解决办法是建立一个临时的无序文件，叫做溢出文件。将插入的记录加到溢出文件中，而且，定期将溢出文件写到主文件中去。这虽然使得插入工作非常有效率，但是在查询上却产生了不好的影响。如果用二分查找没有找到所需的记录，就必须用线性查找在溢出文件中进行查找了。相反的，要删除一条记录，必须确认记录被移动到有效的位置。

除非文件中有一个主索引，否则有序文件很少用在数据库存储中（参考D.5.1节）。

D.4 哈希文件

在一个哈希文件中，记录不必按照顺序写到文件中。取而代之的是，哈希函数根据一个或多个字段的值计算出记录要存储在页面中的位置。所依据的字段叫做哈希字段，或者，如果这个字段恰好是文件的主键，则它就被称为哈希主键。哈希文件中的记录是随机地分布在文件的有效空间中的。因此，哈希文件有时被称为随机文件或直接文件。

哈希函数的使用使得记录尽可能分散在文件中的各部分。其中有一种哈希函数，叫做除留余数法，此种方法使用MOD（取模）函数，取出要使用的字段的值除以预先确定好的一个整型除数，用余数作为磁盘的地址。

大多数哈希函数存在的问题是它们都不能保证一个唯一的地址，因为哈希字段可能取到的值的数量基本上比记录的有效地址的数量大得多。哈希函数产生的每个地址对应于一个页，或者一个桶（Bucket），桶用于存放多条记录。在一个桶中，记录按照到达的顺序存放。当两

条或多条记录的地址相同时，就发生了冲突。在这种情况下，由于它的哈希地址被占用了，因此我们必须把新的记录插入到其他的位置。解决冲突问题使得哈希文件的管理被复杂化了，而且也降低了整体的性能。

D.5 索引

在这部分中，我们讨论利用索引来提高数据查询效率的技术。

索引（Index） 一种数据结构。使用DBMS快速地在文件中查找记录，并能快速响应用户的查询。

数据库中的索引类似于书的目录。它是与要查找信息的文件相关联的一个辅助结构，就像查找书的目录一样，先找到一个关键词然后找到这个关键词所在的一个或多个页的列表。索引的存在使得我们不必在每次寻找一项的时候都要浏览整个文件。在存在数据库索引的情况下，所需要的项将是文件中的一条或多条记录。与书的目录类似，索引也是有序的，并且每个索引项都包含所需要的项和一个或多个这些项能够被找到的位置（记录标识）。

虽然索引在数据库管理系统中不是必需的，但是它们对性能有很重要的影响。就像使用书的目录一样，虽然我们能够通过浏览整本书找到需要的关键词，但是这样做将是单调而费时的。在书后有一个按字母顺序排列的索引能够使我们直接找到我们想要的页。

一个索引结构是与一个特定的查询关键字相关的，并且包含由关键字值和包含这个值的逻辑记录组成的记录。包含逻辑记录的文件称为数据文件，而含有索引记录的文件称为索引文件。索引文件中的值是根据索引字段的值来排序的，这个索引字段通常是基于表中的某个列的。

D.5.1 索引的类型

有不同种类型的索引，主要包括：

- 主索引 数据文件根据一个有序的键字段顺序排序（参见D.3节），并且索引字段是建立在有序的键字段上的。键字段是保证每条记录都有唯一的值。
- 聚簇索引 数据文件根据非键字段的值来顺序排序，并且索引字段是建立在这些非键字段上的，因此对应索引字段的一个值可能会有多条记录。非键字段被称为聚簇字段。
- 二级索引 定义在一个数据文件中的非排序字段上的索引。

一个文件最多只能有一个主索引或一个聚簇索引，但可以有多个二级索引。另外，一个索引可能是稀疏的或密集的：稀疏索引只为文件中的某一些查询键值建立了索引记录；密集索引为文件中的每一个查询键值建立了索引记录。

一个索引的查询关键字可以包含多个字段。图D-4考虑了Staff表中的四个密集索引：一个基于salary列，一个基于branchNo列，一个基于复合索引（salary、branchNo），最后一个基于复合索引（branchNo、salary）。

D.5.2 二级索引

二级索引也是像主索引一样的有序文件。但是，与主索引相关的数据文件是根据索引关键字的值排序的，而与二级索引相关的数据文件也许不是按照索引关键字的值进行排序的。

a) (salary，branchNo)和Salary

b) (branchNo，salary)和branchNo

图D-4 Staff表上的索引

此外，同主索引不同，二级索引字段不需要唯一的值。例如，我们可能希望建立一个基于staff表的branchNo字段的二级索引。从图D-1我们可以看到字段branchNo列的值并不是唯一的。

二级索引提高了使用非主索引字段的字段进行查询的性能。然而，查询的性能的改进也要平衡数据更新时对索引进行维护的费用。这是第13章讨论的物理数据库设计的内容。

D.5.3 多级索引

再次考虑Staff表，这一次用salary列来排序，要查询“工资在$32000到$45000之间的所有员工的信息”。我们已经注意到如果文件是有序的，我们就可以执行二分查找找到第一条记录，然后从这点开始顺序查找其余的满足条件的记录。但是，如果Staff文件很大，则初始的二分查找就会相当慢。

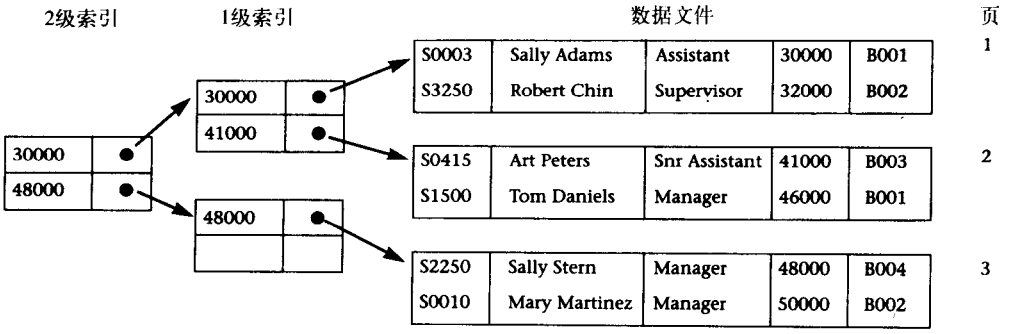
克服这个问题的一种方法是创建一个基于salary列的索引。如果此索引包含数据文件中每个页的salary列的第一个值的入口，则我们就可以在索引文件上实现二分查找，找到包含第一个工资值大于$32000的数据页。

然而，当索引文件逐渐扩大到多个页的时候，对所需索引的查找时间就增加了。多级索引试图通过减少查找范围来克服这个问题。也就是像对待其他任何文件一样看待索引，将索引分成小索引并且再为这些小索引建立索引。图D-5显示了图D-1所示的Staff表的二级索引（稀疏的）的例子。数据文件的每一页能够存储两条记录，作为示例，尽管在实际情况当中每一个数据页中可以有多条索引记录，但是我们这里只显示了每一页中只有两条索引记录。每个索引记录保存一个访问关键字（salary）和一个页地址。存储的访问关键字的值对应的是存储地址的页上的第一条记录。

要查找我们所需要记录的地址，从第二级索引开始查找其最后的访问关键字的值小于或等于32000的页，在我们例子中是30000。这一条索引记录包括一个指向第一级索引页的地址，用这个地址继续查找。重复上面的过程将到达数据文件中存储有第一条记录的第一页。我们现在连续地读取数据文件就能找到其余的符合条件的记录。

IBM的索引顺序存取方法（Indexed Sequential Access Method，ISAM）就是基于两级索引的结构。其插入操作是用溢出文件来处理的，如D.3节所介绍。通常来说，尽管三级索引在

实际当中是很普遍的，但也可以建立n级索引。但当多于三级索引时，文件将会很大。在接下来的几节中，我们将讨论一种特殊类型的多级密集索引，称为B⁺树。



图D-5 多级索引示例

D.5.4 B⁺树

ISAM主要的缺点是当数据库增大时，由于溢出文件而使得数据库性能恶化了，因此DBA需要定期对索引进行重新组织。重新组织索引不但很费时，而且在重新组织过程中文件是不可用的。B⁺树结构通过溢出发生时的分裂节点解决了这个问题。

B⁺树是一种特殊类型的多级索引，在B⁺树中任何一条树的顶层（称为根节点）到底层（称为叶节点）的层数都是相同的，也就是说，这个树是平衡的。叶节点包含有指向表中记录的指针，而不是记录本身。

通过确保查找相同数目的节点，在B⁺树中查找任何数据记录总是需要大致相同的时间，因为通过的节点个数相同，也就是说，要保证树的深度是不变的。由于B⁺树是一个密集索引，所以每条记录都有索引来指向，这样就没有必要为数据文件进行排序了。但是，当更新树的内容时，维护索引的代价是很昂贵的。

实际上，树中的每个节点都是一页，因此我们通常在同一页上存储几个关键字的值。例如，如果我们假设一页有4096字节，而关键字字段和它相应的指针需要4个存储字节存储，而且每个页需要4字节来指向同层中的下一节点，那么，每一页能存储（4096-4）/（4+4）=511个索引记录。根节点能存储511条记录也就有512个分支。每个子节点也可以存储511条记录，这样总共可以有261632条记录。每个子节点也能再有512个子节点，因此，在树的第二级就共有262144个子节点。每个这样的子节点可以有511个记录，总共133955584个。这就给出了理论上的索引记录的最大数：

根	511
一级	261632
二级	133955584
总数	134217727

这样，我们就可以随机地向一个包含有134217727条记录，有4个磁盘存取入口（实际上，根节点一般存放在主存中，因此也就少了一个磁盘存取入口）的文件中存取一条记录。实际上，每一页中存储的记录数并没有理论数目那么多。

D.5.5 位图索引

另一种类型的索引也变得日益普及，特别是在数据仓库中，它就是位图索引。位图索引通常用在有稀疏域（即，这个域只有有限数量的值）的列上。与为列存储实际的值不同，位图索引只为每个列存储一个位向量（bit vector），用于表明哪个记录包含这个特定的域值。位图中与一个行标识符对应的每一位被置为1。如果域值的范围比较小，则利用位图索引是非常节省空间的。

例如，考虑图D-6a所示的Video表。假设category列只有如下几个值：Action、Children、Fantasy和Sci-Fi，同时假设dailyRental列只取下述值中的一个：$4.00、$4.50和$5.00，我们可以构造位图索引来表达这两个列，如图D-6b所示。

catalogNo	title	category	dailyRental	price	directorNo
207132	Die Another Day	Action	5.00	21.99	D1001
902355	Harry Potter	Children	4.50	14.50	D7834
330553	Lord of the Rings	Fantasy	5.00	31.99	D4576
781132	Shrek	Children	4.00	18.50	D0078
445624	Men in Black II	Action	4.00	29.99	D5743
634817	Independence Day	Sci-Fi	4.50	32.99	D3765

a)

Action	Children	Fantasy	Sci-Fi
1	0	0	0
0	1	0	0
0	0	1	0
0	1	0	0
1	0	0	0
0	0	0	1

4.00	4.50	5.00
0	0	1
0	1	0
0	0	1
1	0	0
1	0	0
0	1	0

b)

图D-6 a）Video表；b）category和dailyRental列上的位图索引

位图索引提供了两个优于B*树索引的优点。首先，位图索引比B*树索引更紧凑，因此比B*树索引需要的空间要少，位图索引本身也成为一种压缩技术；第二，当查询包括多个谓词并且每个都有子句的位图索引时，位图索引可以极大地提高性能。例如，考虑如下查询：

```
SELECT catalogNo, title
FROM Video
WHERE category = 'Fantasy' AND dailyRental = 5.00;
```

在这个例子中，我们可以为category使用第三个位向量并用dailyRental的第三个位向量进行AND逻辑运算，来获得属于Fantasy类录像且日租金为$5.00的位均为1的位向量。

D.5.6 连接索引

另一种类型的索引也在日益普及，它同样也特别适用于数据仓库领域，它就是连接索引（join index）。连接索引是在来自两个或多个表（这些表具有相同域）的列上建立索引。例如，

考虑图D-7a所示的扩展的Branch表和Member表，我们可以在非键列city上创建一个连接索引，产生的索引表如图D-7b所示。我们选择在BranchRowID上排序连接索引，但实际上可以在这三个列中的任何一个列上进行排序。有时也可以创建两个连接索引，一个如图所示，另一个用两个rowID列。

Branch

rowID	branchNo	street	city	state	zipCode	mgrStaffNo
20001	B001	8 Jefferson Way	Portland	OR	97201	S1500
20002	B002	City Center Plaza	Seattle	WA	98122	S0010
20003	B003	14 – 8th Avenue	New York	NY	10012	S0415
20004	B004	16 – 14th Avenue	Seattle	WA	98128	S2250
20005	...					

Member

rowID	memberNo	fName	lName	street	city	state	zipCode
30001	M250178	Bob	Adams	57 – 11th Avenue	Seattle	WA	98105
30002	M166884	Art	Peters	89 Redmond Rd	Portland	OR	97117
30003	M115656	Serena	Parker	22 W. Capital Way	Portland	OR	97201
30004	M284354	Don	Nelson	123 Suffolk Lane	Seattle	WA	98117
30005	...						

a)

连接索引

branchRowID	memberRowID	city
20001	30002	Portland
20001	30003	Portland
20002	30001	Seattle
20002	30004	Seattle
20004	30001	Seattle
20004	30004	Seattle
20005	...	

b)

图D-7 a）Branch表和Member表；b）在非键列city上的连接索引

D.6 选择文件组织方式指南

为了更彻底地理解文件组织方式，在本节中我们提出了基于下述文件类型选择文件组织方式的指南。

- 堆
- 哈希
- 索引顺序访问方法（ISAM）
- B*树

1. 堆（无序的）

堆在下述情况下是一种很好的存储结构:

1) 当数据被批量加载到表中时。例如，你可能希望在创建表之后在表中插入一批记录。如果选择堆作为初始的文件组织方式，那么在完成了插入操作后对文件重组可能会更有效。

2) 表只有几个数据页大小。在这种情况下，找到任何记录的时间都是很短的，即使需要对整个表进行顺序查找。

3) 每当访问表时都需要检索表中的每条记录（以任何顺序）。例如，检索StayHome中的所有会员的地址。

4) 当表有附加的访问结构时，例如索引键，则可以使用堆存储来节省空间。

当只访问一个表中被选定的记录时，堆文件是不合适的。

2. 哈希

当检索是基于与哈希字段值精确匹配的记录时，哈希是一个很好的存储结构，特别是当访问顺序是随机的时候。例如，如果在Member表的memberNo列上进行哈希，则检索memberNo等于M250178的记录就非常有效。但哈希在下述情况下就不是一个好的存储结构:

1) 当对记录的检索是基于哈希字段值的模式匹配时。例如，检索会员号（memberNo）以“M2”开始的所有会员。

2) 当对记录的检索是基于哈希字段值的一个范围时。例如，检索会员号在“M200000”和“M200100”之间的所有会员。

3) 当对记录的检索是基于列而不是基于哈希列时。例如，如果Member表在memberNo列上进行哈希，则当基于IName列检索记录时这个哈希就不能使用。在这种情况下，就需要执行一个线性查找来检索记录，或者将IName作为一个二级索引。

4) 当记录的检索是基于哈希的部分字段时。例如，如果Role表在catalogNo和catorNo列上进行哈希，则当只基于catalogNo列查找记录时这个哈希就不能使用。同样，它也需要执行一个线性查找来检索记录。

5) 当哈希列经常被更新时。当更新一个哈希列时，DBMS必须删除整个记录并将它重新定位到新的地址（如果哈希函数产生了一个新的地址）。因此，经常更新哈希列会影响性能。

3. 索引顺序访问方法（ISAM）

ISAM是一个比哈希更灵活的存储结构，它支持基于准确键匹配的检索、基于模式匹配的检索、基于范围值和部分键规格说明的检索。但是，ISAM索引是静态的，它在文件被创建时创建，因此，你会发现随着表的不断修改，ISAM文件的性能会恶化。更新也引起ISAM文件丢失访问键的顺序，因此按访问键顺序进行检索将变得很慢。这两个问题已经被B^+树文件组织方式克服了。

4. B^+树

同样，B^+树也是一个比哈希更灵活的存储结构。它支持基于精确键匹配、模式匹配、范围值和部分键匹配的检索。B^+树索引是动态的，随着表的增长而增长。因此，与ISAM不同，B^+树文件的性能不会随着文件的修改而恶化。即使在文件被修改时，B^+树也维护访问键的顺序，因此按访问键的顺序检索记录比ISAM更有效。但如果表不经常被修改，则ISAM结构可能会更有效，因为它比B^+树的索引级别低，B^+树的叶结点包含指向表中的实际记录的指针，而不是实际的记录本身。

D.7 聚簇和非聚簇的表

有些DBMS（比如Oracle）支持聚簇和非聚簇的表。使用聚簇的表还是使用非聚簇的表依赖于前边对事务的分析，但这个选择会对性能有影响。在这一节中，我们简单介绍这两种类型的结构并给出使用聚簇表的指南。

聚簇是物理地存储在一起的一个或多个表的组，它们共享相同的列并且经常被同时使用。利用物理地存储在一起的相关记录，就可以减少磁盘访问时间。在聚簇中的表的相关列被称为聚簇键（cluster key）。聚簇键仅被存储一次，因此表的聚簇存储比分别存储（不聚簇）要更有效。

图D-8说明了当我们基于branchNo列聚簇表时，Branch和Staff表的存储情况。当这两个表被聚簇时，每个不同的branchNo列的值只被存储一次。每个branchNo值被附加在这两个表的列上。

street	city	state	zipCode	mgrStaffNo	branchNo	staffNo	name	position	salary
8 Jefferson Way	Portland	OR	97201	S1500	B001	S1500	Tom Daniels	Manager	46000
						S0003	Sally Adams	Assistant	30000
City Center Plaza	Seattle	WA	98122	S0010	B002	S0010	Mary Martinez	Manager	50000
						S3250	Robert Chin	Supervisor	32000
...									

Staff表

Branch表

聚簇键

图D-8 Branch 和 Staff 表如何在 branchNo上聚簇存储

正如我们现在讨论的，Oracle支持两种类型的聚簇：索引聚簇和哈希聚簇。

D.7.1 索引聚簇

在索引聚簇中，有相同聚簇键的记录存储在一起。Oracle建议在下述情况下使用索引聚簇：

- 检索聚簇键值在一个范围内的记录。
- 聚簇的表可能会不可预料地增长。

聚簇可以提高数据检索的性能，这依赖于数据的分布和在数据上经常使用的SQL操作。特别地，在一个查询中连接的表适合使用聚簇，因为对连接表中的相同记录的检索有相同的I/O操作。

要在Oracle中创建一个聚簇键列为branchNo、名字为BranchIndexedCluster的索引聚簇，可以使用如下SQL语句：

```
CREATE CLUSTER BranchIndexedCluster
   (branchNo CHAR(4))
SIZE 512
STORAGE(INITIAL 100K NEXT 50K PCTINCREASE 10);
```

SIZE参数指定存储所有具有相同聚簇键值的记录的空间（单位为字节）的大小，这个大小是可选的，如果省略，Oracle为每个聚簇键值预留一个数据块。INITIAL参数指定聚簇的第一

个区的大小（单位为字节），NEXT参数指定将要分配的下一个区的大小（单位为字节）。PCTINCREASE参数指定一个百分比，这个百分比用于指定第三个和后续的区比前一个区增长的大小（默认值为50）。在我们的例子中，我们指定每个后续的区应该比前一个区大10%。

一旦创建好了哈希聚簇，就可以创建作为这个结构的一部分的表了。例如:

```
CREATE TABLE Branch
( branchNo CHAR(4) PRIMARY KEY,
... )
CLUSTER BranchIndexedCluster(branchNo);
```

使用索引聚簇指南

当决定是否聚簇表时，你会发现下述指南会很有帮助:

- 考虑对经常在连接语句中访问的表建立聚簇。
- 如果表只是偶尔被连接或者它们的公共列值经常被修改，则不要聚簇表。（修改记录的聚簇键值比在非聚簇的表中修改此值要花费更多的时间，因为Oracle必须将修改的记录移植到其他的块中以维护聚簇。）
- 如果经常需要在一个表上进行完全搜索，则不要聚簇这个表（对一个聚簇表进行完全搜索比在非聚簇表上进行完全搜索用的时间长，Oracle可能要读更多的块，因为表是被一起存储的）。
- 如果经常从一个父表和相应的子表中查询记录，则考虑给1对多（1:*）关系创建聚簇表。（子表记录存储在与父表记录相同的数据块中，因此当检索它们时可以同时在内存中，因此需要Oracle完成较少的I/O）。
- 如果经常查询同一个父表中的多个子记录，则考虑单独将子表聚簇。（这样提高了从相同的父表查询子表记录的性能，而且也没有降低对父表进行完全搜索的性能）。
- 如果从所有有相同聚簇键值的表查询的数据超过了一个或两个Oracle块，则不要聚簇表。（要访问在一个聚簇表中的记录，Oracle读取所有包含那个记录值的全部数据块，如果记录占据了多个数据块，则访问一个记录需要读的次数比在一个非聚簇的表中访问相同记录读的次数要多）。

D.7.2 哈希聚簇

哈希聚簇以与索引聚簇相似的方式对表数据进行聚簇。但以哈希聚簇存储的记录基于对记录的聚簇键值应用哈希函数的结果。有相同哈希键值的所有记录被存储在一起。Oracle建议在下述情况下使用哈希聚簇:

- 基于涉及所有聚簇键列的相等条件的查询（例如，返回分公司B001的所有记录）。
- 聚簇表是静态的或者当创建聚簇时我们可以确定记录的最大数量和所需的最大空间。

在Oracle中，要在branchNo列上创建一个叫做BranchHashCluster的哈希聚簇，可以使用如下的SQL语句:

```
CREATE CLUSTER BranchHashCluster
( branchNo CHAR(4))
HASH IS branchNo HASHKEYS 5000;
```

使用哈希聚簇的指南

你会发现当决定是否使用哈希聚簇时下述指南很有帮助:

- 当经常使用有相同列的包含相等条件的查询子句访问表时，考虑使用哈希聚簇来存储表。用这些列作为聚簇键。
- 如果可以确定存放具有给定聚簇键值的所有记录所需的空间（包括现在的和将来的），则将此表以哈希聚簇存储。
- 如果空间不够，并且不能为将要插入的新记录分配额外的空间，那么不要使用哈希聚簇。
- 如果偶尔创建一个新的、很大的哈希聚簇来保存这样的表是不切实际的，那么不要用哈希聚簇存储经常增长的表。
- 如果经常需要进行全表搜索，并且必须要为表的预期增长中的哈希聚簇分配足够的空间，则不要将此表以哈希聚簇存储。（这样的完全检索必须要读分配给哈希聚簇的全部的块，即使有些块可能只包含很少的记录。单独地存储表将减少由完全的表检索读取的块的数量。）
- 如果你的应用程序经常修改聚簇键的值，则不要存储将表以哈希聚簇方式存储。
- 不管这个表是否经常与其他表连接，只要进行哈希对于基于以前的指南的表是合适的，那么在哈希聚簇中存储一个表可能是有用的。

附录小结

- 堆文件适合在文件中插入大量记录的情况。但对仅查询已选择的记录的情况就是不合适的。
- 当检索是基于精确的键匹配时哈希文件是合适的。但当检索是基于模式匹配、值的范围、部分键或当检索是基于其他的非哈希列时，哈希文件就不合适了。
- ISAM比哈希更灵活，它支持基于精确键匹配、模式匹配、值的范围和部分键规格说明的检索。但是，ISAM索引是静态的，因此当表被修改时性能会恶化。更新也引起了ISAM文件丢失访问键顺序，因此按访问键顺序的访问将会变慢。
- 这两个问题被B^+树文件组织方式克服了，B^+树索引是动态的索引。如果一个表不是经常被修改或不是很大，则ISAM结构可能会更有效，因为它的索引级别比B^+树低，B^+树的叶结点包含记录指针。
- 二级索引提供了为基本表指定一个附加键的机制，这个附加键可以用于更有效地检索数据。但在这个机制中有附加的开销，而且使用二级索引必须要与检索数据获得的性能进行权衡。
- 聚簇是一个或多个表作为一个组物理地存储在一起，因为它们共享相同的列并且经常一起使用。有了物理地存储在一起的相关的记录，可以减少磁盘存取时间。聚簇中的表的相关列被称为聚簇键。聚簇键仅存储一次，因此聚簇存储一组表的效率比单独存储（没有聚簇）每个表要高。Oracle支持两种类型的聚簇：索引聚簇和哈希聚簇。

附录E 常用数据模型

在这个附录中你将学到:

- 关于构建逻辑数据模型的更多的知识。
- 关于常用的逻辑数据模型。

在这个附录中，我们介绍了一些也许你会觉得有用的常用的数据模型。实际上，据估计1/3的数据模型是由适用于大多数公司的普通结构组成的，剩下的2/3是由工业或者公司的特定情况组成的。这样，大多数的数据模型就是在重复你以前在其他公司已经做过很多次的工作。

因此，这个附录的两个主要目的就是向我们提供:

- 建立数据模型的附加知识。
- 在你的业务中有用的数据模型模板。虽然这里提到的模型也许并不能确切地描述你的公司的要求，但是它可能会给你提供一个起点，使你能更好地建立符合你公司特殊要求的数据模型。

我们提供了下面的经常用到的商业领域的模型:

- 客户订购登记（customer order entry）
- 库存控制（inventory control）
- 资产管理（asset management）
- 项目管理（project management）
- 课程管理（course management）
- 人力资源管理（human resource management）
- 工资管理（payroll management）

我们也提供了下面这些用的也许不是很常用的数据模型，但是这些模型对于商业目标和我们的学习目的来说还是有用的:

- 车辆租赁（vehicle rentals）
- 学生住宿（student accommodation）
- 客户运送（client transportation）
- 出版商印刷（publisher printing）
- 国家图书馆（county library）
- 房地产租赁（real estate rentals）
- 旅行代理（travel agent）
- 学生管理（student results）

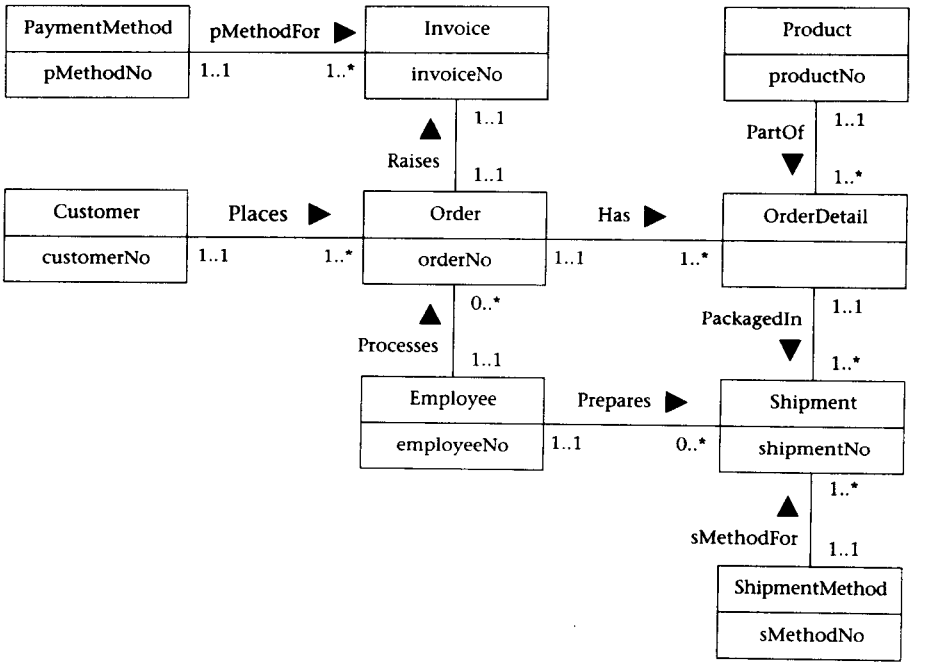
在每个示例中，我们都提供了对需求的简单描述，并且提供了一个典型逻辑数据模型的例子，并将这些模型映射为一系列的表。我们假设你熟悉本书使用的数据模型的符号，如果你不熟悉的话，可以参看第7章的ER模型，那里介绍了主要的概念和在附录中要用到的符号。你也可以在附录B中找到方法学的概要。

E.1 客户订购登记

一个公司希望为其客户订购行为建立一个数据库。一个消费者可以有一个或多个订单，

每个订单可以有一种或多种商品。每个订单有一个发票，可以通过多种方式来支付，例如支票、信用卡或者现金。开始运行这个客户订购登记的员工的名字要被记下来。

有一个相应的部门工作人员来负责整理订单并把这些订单发给顾客。如果订单上的货物在库中没有，就需写明库中有什么，这样在订单中可能就会用其他货物来填充。它的逻辑数据模型如图E-1所示，相关的表如图E-2所示。



图E-1 客户订单项的逻辑数据模型

Customer	(customerNo, customerName, customerStreet, customerCity, customerState, customerZipCode, custTelNo, custFaxNo, DOB, maritalStatus, creditRating) Primary Key customerNo Alternate Key custTelNo Alternate Key custFaxNo
Employee	(employeeNo, title, firstName, middleName, lastName, address, workTelExt, homeTelNo, empEmailAddress, socialSecurityNumber, DOB, position, sex, salary, dateStarted) Primary Key employeeNo Alternate Key socialSecurityNumber
Invoice	(invoiceNo, dateRaised, datePaid, creditCardNo, holdersName, expiryDate, orderNo, pMethodNo) Primary Key invoiceNo Foreign Key orderNo references Order(orderNo) Foreign Key pMethodNo references PaymentMethod(pMethodNo)
Order	(orderNo, orderDate, billingStreet, billingCity, billingState, billingZipCode, promisedDate, status, customerNo, employeeNo) Primary Key orderNo Foreign Key customerNo references Customer(customerNo) Foreign Key employeeNo references Employee(employeeNo)
OrderDetail	(orderNo, productNo, quantityOrdered) Primary Key orderNo, productNo Foreign Key orderNo references Order(orderNo) Foreign Key productNo references Product(ProductNo)

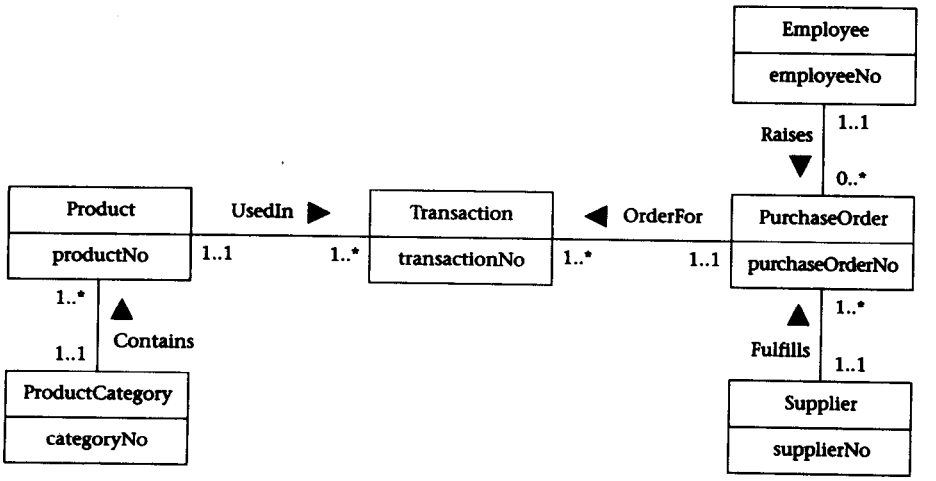
图E-2 客户订单项的表

PaymentMethod	(pMethodNo, paymentMethod) Primary Key pMethodNo
Product	(productNo, productName, serialNo, unitPrice, quantityOnHand, reorderLevel, reorderQuantity, reorderLeadTime) Primary Key productNo Alternate Key serialNo
Shipment	(shipmentNo, quantity, shipmentDate, completeStatus, orderNo, productNo, employeeNo, sMethodNo) Primary Key shipmentNo Foreign Key orderNo, productNo references OrderDetail(orderNo, productNo) Foreign Key employeeNo references Employee(employeeNo) Foreign Key sMethodNo references ShipmentMethod(sMethodNo)
ShipmentMethod	(sMethodNo, shipmentMethod) Primary Key sMethodNo

图E-2 （续）

E.2 库存控制

一个公司希望为控制它的库存建立一个数据库，库存中的产品被分为几类，如服装、食品和文具。当产品需要从供应商那里重新订购时，工作人员需要提出一个购买订单。跟踪记录要提供买进、卖出的货物以及其他的费用。逻辑数据模型如图E-3所示，相关的表如图E-4所示。



图E-3 库存控制的逻辑数据模型

Employee	在D.1.2节定义
Product	(productNo, productName, serialNo, unitPrice, quantityOnHand, reorderLevel, reorderQuantity, reorderLeadTime, categoryNo) Primary Key productNo Alternate Key serialNo Foreign Key categoryNo references ProductCategory(categoryNo)
ProductCategory	(categoryNo, categoryDescription) Primary Key categoryNo
PurchaseOrder	(purchaseOrderNo, purchaseOrderDescription, orderDate, dateRequired, shippedDate, freightCharge, supplierNo, employeeNo) Primary Key purchaseOrderNo Foreign Key supplierNo references Supplier(supplierNo) Foreign Key employeeNo references Employee(employeeNo)

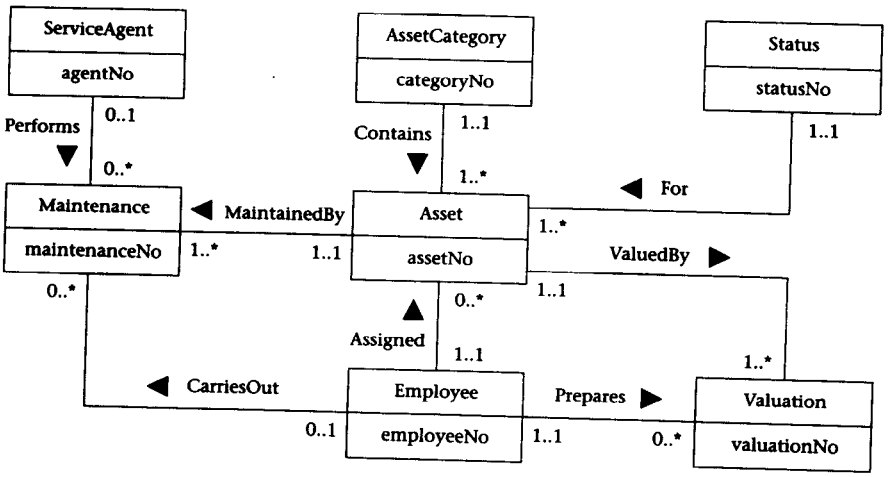
图E-4 库存控制的表

Supplier	(supplierNo, supplierName, supplierStreet, supplierCity, supplierState, supplierZipCode, suppTelNo, suppFaxNo, suppEmailAddress, suppWebAddress, contactName, contactTelNo, contactFaxNo, contactEmailAddress, paymentTerms) Primary Key supplierNo Alternate Key supplierName Alternate Key suppTelNo Alternate Key suppFaxNo
Transaction	(transactionNo, transactionDate, transactionDescription, unitPrice, unitsOrdered, unitsReceived, unitsSold, unitsWastage, productNo, purchaseOrderNo) Primary Key transactionNo Foreign Key productNo references Product(productNo) Foreign Key purchaseOrderNo references PurchaseOrder(purchaseOrderNo)

图E-4 （续）

E.3 资产管理

一个公司希望为管理它的每批资产（如PC机、打印机、汽车、桌子、椅子等）建立一个数据库。资产被分为几类，如计算机和设备。一个资产被分配给一个员工。通常情况下，财务部门的工作人员要检查每批资产的现有市场价值，并记录下日期和现有价值。作为估价的结果，公司也许会决定卖掉现有资产。同样，每批资产都要进行维护。在某些情况下，维护是需要工作人员来进行的。另外，也可能是拿到外面的公司去进行维护。它的逻辑数据模型如图E-5所示，相关表如图E-6所示。



图E-5 资产管理的逻辑数据模型

Employee	在D.1.2节定义
Asset	(assetNo, assetDescription, serialNo, dateAcquired, purchasePrice, currentValue, dateSold, nextMaintenanceDate, employeeNo, assetCategoryNo, statusNo) Primary Key assetNo Alternate Key serialNo Foreign Key employeeNo references Employee(employeeNo) Foreign Key assetCategoryNo references AssetCategory(assetCategoryNo) Foreign Key statusNo references Status(statusNo)
AssetCategory	(assetCategoryNo, assetCategoryDescription) Primary Key assetCategoryNo
Maintenance	(maintenanceNo, maintenanceDate, maintenanceDescription, maintenanceCost,

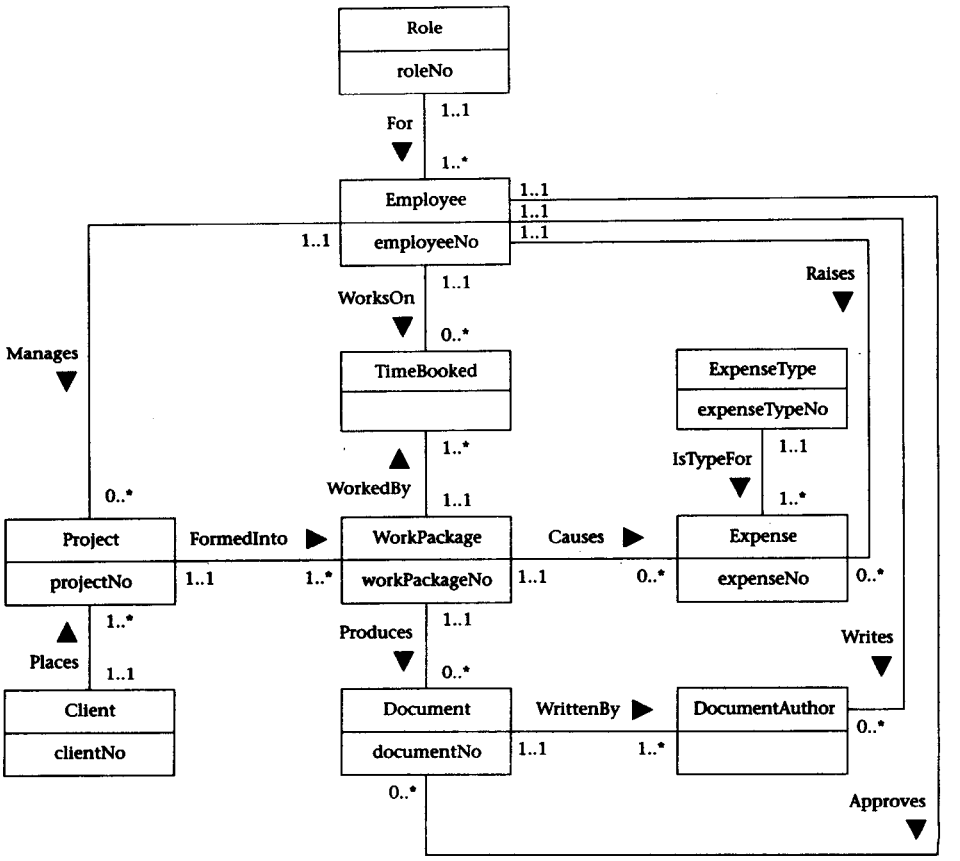
图E-6 资产管理的表

	assetNo, employeeNo, agentNo) Primary Key maintenanceNo Foreign Key assetNo references Asset(assetNo) Foreign Key employeeNo references Employee(employeeNo) Foreign Key agentNo references ServiceAgent(agentNo)
ServiceAgent	(agentNo, agentName, agentStreet, agentCity, agentState, agentZipCode, agentTelNo, agentFaxNo, agentEmailAddress, agentWebAddress, contactName, contactTelNo, contactFaxNo, contactEmailAddress) Primary Key agentNo Alternate Key agentName Alternate Key agentTelNo Alternate Key agentFaxNo
Status	(statusNo, statusDescription) Primary Key statusNo
Valuation	(valuationNo, valuationDate, valuationPrice, assetNo, employeeNo) Primary Key valuationNo Foreign Key assetNo references Asset(assetNo) Foreign Key employeeNo references Employee(employeeNo)

图E-6 （续）

E.4 项目管理

一个顾问公司希望建立一个数据库来帮助管理它的项目。每个项目都对应一个用户而且



图E-7 项目管理的逻辑数据模型

有一个任命的项目经理。一个项目被分为几个工作包，工作人员用工作包来核算自己的时间和花费。每个员工都有一个特殊的角色，这个角色定义了用户的费用比例。随着时间的推移，一个工作人员可能在同一个项目的几个工作包中工作。另外，大多数（但不是全部）工作包都有一定数量的与之相关的可交付的文档，每个文档也许是由多个工作人员写的。其逻辑数据模型如图E-7所示，相关的表如图E-8所示。

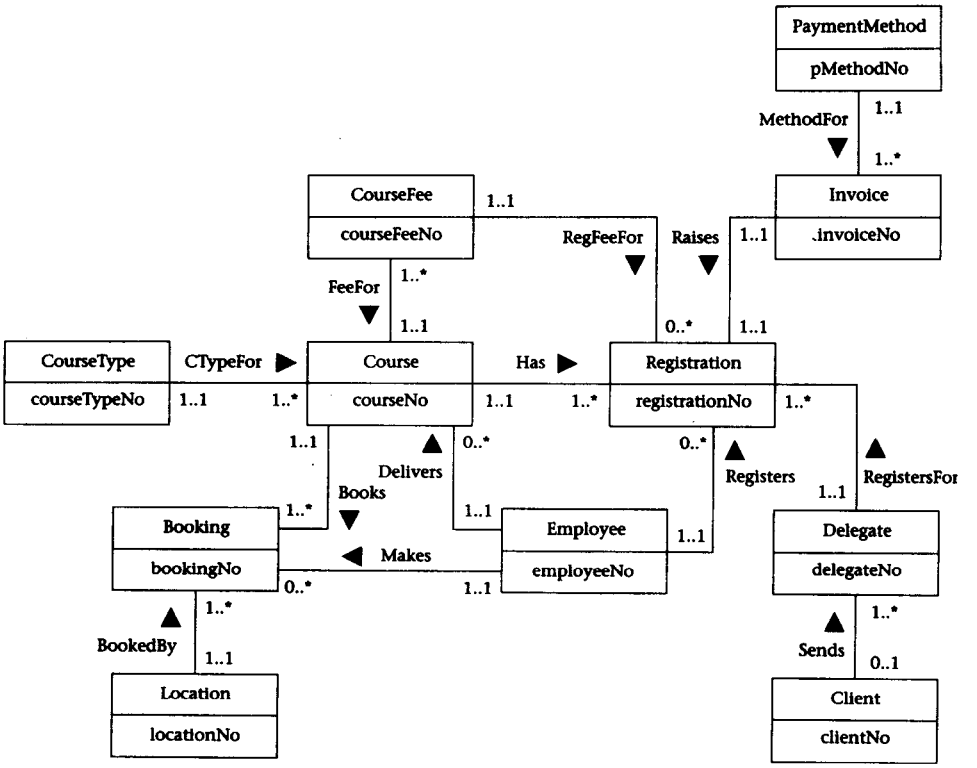
Client	(clientNo, clientName, clientStreet, clientCity, clientState, clientZipCode, clientTelNo, clientFaxNo, clientWebAddress, contactName, contactTelNo, contactFaxNo, contactEmailAddress) Primary Key clientNo Alternate Key clientName Alternate Key clientTelNo Alternate Key clientFaxNo
Document	(documentNo, documentTitle, documentDate, versionNo, workPackageNo, approvedByEmployeeNo) Primary Key documentNo Foreign Key workPackageNo references WorkPackage(workPackageNo) Foreign Key approvedByEmployeeNo references Employee(employeeNo)
DocumentAuthor	(documentNo, employeeNo) Primary Key documentNo, employeeNo Foreign Key documentNo references Document(documentNo) Foreign Key employeeNo references Employee(employeeNo)
Employee	(employeeNo, dateStartRole, firstName, middleName, lastName, address, workTelExt, homeTelNo, empEmailAddress, socialSecurityNumber, DOB, position, sex, salary, dateStarted, roleNo) Primary Key employeeNo Alternate Key socialSecurityNumber Foreign Key roleNo references Role(roleNo)
Expense	(expenseNo, expenseDate, expenseDescription, expenseAmount, workPackageNo, employeeNo, expenseTypeNo) Primary Key expenseNo Alternate Key workPackageNo, employeeNo, expenseDate Foreign Key workPackageNo references WorkPackage(workPackageNo) Foreign Key employeeNo references Employee(employeeNo) Foreign Key expenseTypeNo reference ExpenseType(expenseTypeNo)
ExpenseType	(expenseTypeNo, expenseTypeDescription) Primary Key expenseTypeNo
Project	(projectNo, projectName, plannedStartDate, plannedEndDate, actualStartDate, actualEndDate, projectedCost, actualCost, clientNo, managerEmployeeNo) Primary Key projectNo Foreign Key clientNo references Client(clientNo) Foreign Key managerEmployeeNo references Employee(employeeNo)
Role	(roleNo, roleDescription, billingRate) Primary Key roleNo
TimeBooked	(workPackageNo, employeeNo, dateStartWork, dateStopWork, timeWorked) Primary Key workPackageNo, employeeNo Foreign Key workPackageNo references WorkPackage(workPackageNo) Foreign Key employeeNo references Employee(employeeNo)
WorkPackage	(workPackageNo, plannedStartDate, plannedEndDate, actualStartDate, actualEndDate, projectedCost, actualCost, projectNo) Primary Key workPackageNo Foreign Key projectNo references Project(projectNo)

图E-8 项目管理的表

E.5 课程管理

一个培训公司希望建立一个关于课程信息的数据库。公司发布几个讨论会和一些培训课程。每个课程由一些员工在某个地点完成（如内部研讨会，Hilton大厦S10房间）。费用随着课

程不同或者公司委托的代表的数量不同而不同。例如，如果公司派一个人，收费可能是$1000。如果派两个人，则第一个人收费是$1000，第二个人的费用也许就是$750。某项课程可能由几个代表参加，这依赖于一些课程的整体规定。每个代表可以以个人或公司的名义来注册。注册了代表的员工的名字将被记录下来。发票被交给本人或者是公司。其逻辑数据模型如图E-9所示，相关的表如图E-10所示。



图E-9 课程管理的逻辑数据模型

Client	在D.4.2节定义
Employee	在D.1.2节定义
PaymentMethod	在D.1.2节定义
Delegate	(delegateNo, delegateTitle, delegateFName, delegateLName, delegateStreet, delegateCity, delegateState, delegateZipCode, attTelNo, attFaxNo, attEmailAddress, clientNo) Primary Key delegateNo Foreign Key clientNo references Client(clientNo)
Booking	(bookingNo, bookingDate, locationNo, courseNo, bookingEmployeeNo) Primary Key bookingNo Foreign Key locationNo references Location(locationNo) Foreign Key courseNo references Course(courseNo) Foreign Key bookingEmployeeNo references Employee(employeeNo)
Course	(courseNo, courseName, courseDescription, startDate, startTime, endDate, endTime, maxDelegates, confirmed, delivererEmployeeNo, courseTypeNo) Primary Key courseNo Foreign Key delivererEmployeeNo references Employee(employeeNo) Foreign Key courseTypeNo references CourseType(courseTypeNo)
CourseFee	(courseFeeNo, feeDescription, fee, courseNo)

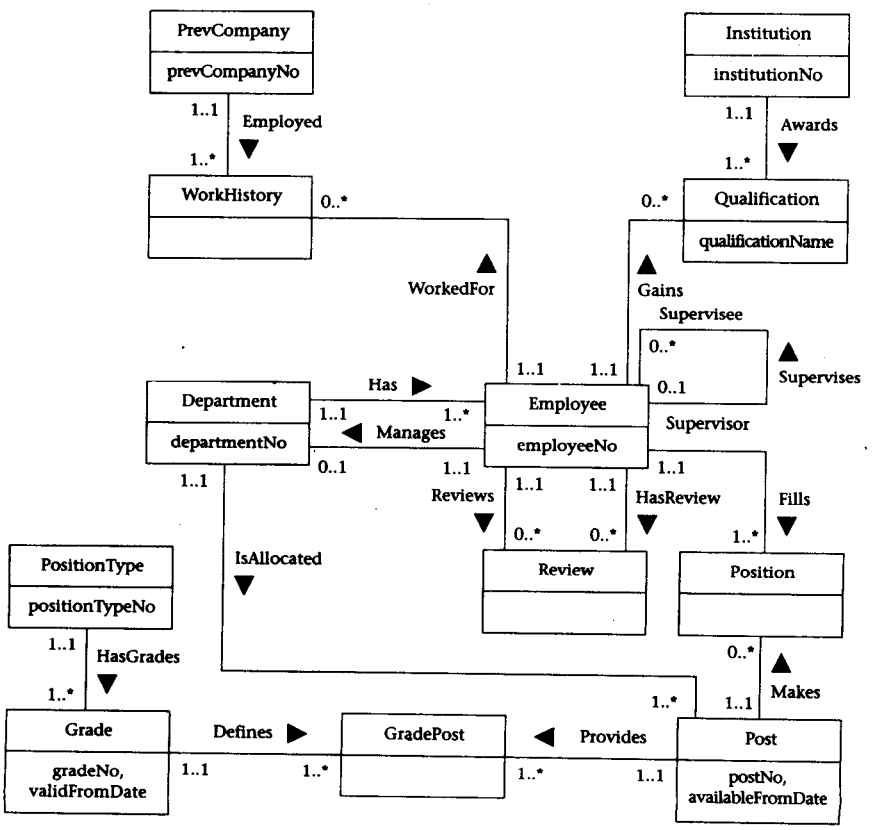
图E-10 课程管理的表

	Primary Key courseFeeNo Foreign Key courseNo references Course(courseNo)
CourseType	(courseTypeNo, courseTypeDescription) Primary Key courseTypeNo
Invoice	(invoiceNo, dateRaised, datePaid, creditCardNo, holdersName, expiryDate, registrationNo, pMethodNo) Primary Key invoiceNo Foreign Key registrationNo references Registration(registrationNo) Foreign Key pMethodNo references PaymentMethod(pMethodNo)
Location	(locationNo, locationName, maxSize) Primary Key locationNo
Registration	(registrationNo, registrationDate, delegateNo, courseFeeNo, registerEmployeeNo, courseNo) Primary Key registrationNo Foreign Key delegateNo references Delegate(delegateNo) Foreign Key courseFeeNo references CourseFee(courseFeeNo) Foreign Key registerEmployeeNo references Employee(employeeNo) Foreign Key courseNo references Course(courseNo)

图E-10 （续）

E.6 人力资源管理

人力资源部门希望建立一个数据库来管理它的员工。一个公司有几个部门，而一个员工属于一个部门。这个部门指派一个经理来全面负责部门事务和部门员工。但为了有助于管理



图E-11 人力资源管理的数据模型

好部门工作，某些工作人员被任命来管理一组人员。当有一个新的员工进入公司时，需要他以前的工作经历和成绩。通常来说，每个员工都要经历一次面试，这通常是由经理来进行的，但有些时候也指派给一个代表来完成。

公司定义了一系列的职位类型，例如经理、业务分析员、销售人员和秘书，而且每个类型都有相关的等级，员工所处的位置决定了员工的工资。在高层，工资是可以通过谈判来决定的。职位依据其工作量来分配给一个部门。例如，一个部门可能分配给两个业务分析员的职位。每个岗位会分配给一个员工，随着时间的过去，各个职位都会被分配给工作人员。其逻辑数据模型如图E-11所示，相关的表如图E-12所示。

Department	(departmentNo, departmentName, deptLocation, managerEmployeeNo) Primary Key departmentNo Foreign Key managerEmployeeNo references Employee(employeeNo)
Employee	(employeeNo, title, firstName, middleName, lastName, address, workTelExt, homeTelNo, empEmailAddress, socialSecurityNumber, DOB, position, sex, salary, dateStarted, dateLeft, departmentNo, supervisorEmployeeNo) Primary Key employeeNo Alternate Key socialSecurityNumber Foreign Key departmentNo references Department(departmentNo) Foreign Key supervisorEmployeeNo references Employee(employeeNo)
Grade	(gradeNo, validFromDate, validToDate, gradeDescription, gradeSalary, noDaysLeaveEntitlement, positionTypeNo) Primary Key gradeNo, validFromDate Foreign Key positionTypeNo references PositionType(positionTypeNo)
GradePost	(gradeNo, validFromDate, postNo, availableFromDate) Primary Key gradeNo, validFromDate, postNo, availableFromDate Foreign Key gradeNo, validFromDate references Grade(gradeNo, validFromDate) Foreign Key postNo, availableFromDate references Post(postNo, availableFromDate)
Institution	(institutionNo, institutionName, instAddress, instTelNo, instFaxNo, instWebAddress, contactName, contactTelNo, contactFaxNo, contactEmailAddress) Primary Key institutionNo Alternate Key institutionName Alternate Key instTelNo Alternate Key instFaxNo
Position	(employeeNo, postNo, startDate, endDate) Primary Key employeeNo, postNo, startDate Foreign Key employeeNo references Employee(employeeNo) Foreign Key postNo, startDate references Post(postNo, availableFromDate)
PositionType	(positionTypeNo, positionTypeDescription) Primary Key positionTypeNo
Post	(postNo, availableFromDate, availableToDate, postDescription, salariedHourly, fullPartTime, temporaryPermanent, freeLaborStandardsActExempt, departmentNo) Primary Key postNo, availableFromDate Foreign Key departmentNo references Department(departmentNo)
PrevCompany	(prevCompanyNo, pCompanyName, pCompanyStreet, pCompanyCity, pCompanyState, pCompanyZipCode, pCompanyTelNo, pCompanyFaxNo, pCompanyWebAddress, contactName, contactTelNo, contactFaxNo, contactEmailAddress) Primary Key prevCompanyNo Alternate Key pCompanyName Alternate Key pCompanyTelNo Alternate Key pCompanyFaxNo
Qualification	(qualificationName, employeeNo, gradeObtained, startQualDate, endQualDate, gpa, institutionNo) Primary Key qualificationName, employeeNo Foreign Key employeeNo references Employee(employeeNo) Foreign Key institutionNo references Institution(institutionNo)

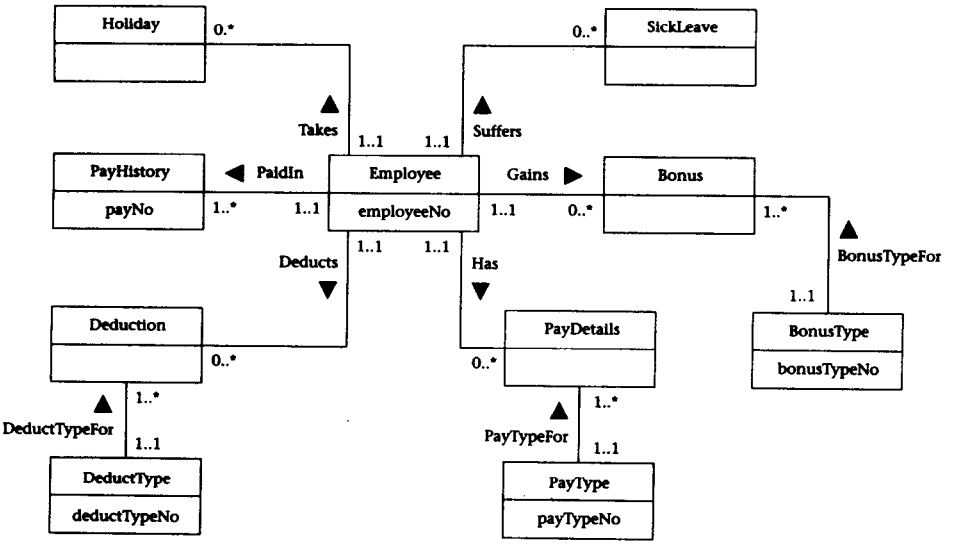
图E-12 人力资源管理的表

Review	(revieweeEmployeeNo, reviewerEmployeeNo, reviewDate, comments) Primary Key revieweeEmployeeNo, reviewerEmployeeNo, reviewDate Foreign Key revieweeEmployeeNo references Employee(employeeNo) Foreign Key reviewerEmployeeNo references Employee(employeeNo)
WorkHistory	(prevCompanyNo, employeeNo, prevPosition, prevGrade, prevSalary, prevLocation, prevResponsibilities) Primary Key prevCompanyNo, employeeNo Foreign Key prevCompanyNo references PrevCompany(prevCompanyNo) Foreign Key employeeNo references Employee(employeeNo)

图E-12 （续）

E.7 工资管理

工资管理部门希望建立一个数据库来管理员工的工资。要计算员工的工资，就需要考虑不在休假日期以内的假期、工作期间的病假时间、奖金和扣除的部分。必须指明给每个员工发薪水的方式，随着时间的推移，方式可能会有些改变。大多数的员工是通过银行卡来结算工资的，但是也有一部分人使用现金或支票。如果是通过银行卡，就需要知道账号和卡的类型。付款方式只可能是一种方式。有几种原因可以扣除工资：例如，个人所得税、国家税、医疗保险、退休保险或者预付款。其逻辑数据模型如图E-13所示，相关的表如图E-14所示。



图E-13 工资管理的逻辑数据模型

Employee	在D.1.2节定义
Bonus	(employeeNo, bonusDate, bonusAmount, bonusTypeNo) Primary Key employeeNo, bonusDate Foreign Key employeeNo references Employee(employeeNo) Foreign Key bonusTypeNo references BonusType(bonusTypeNo)
BonusType	(bonusTypeNo, bonusDescription) Primary Key bonusTypeNo
Deduction	(employeeNo, deductDate, deductAmount, deductTypeNo) Primary Key employeeNo, deductDate

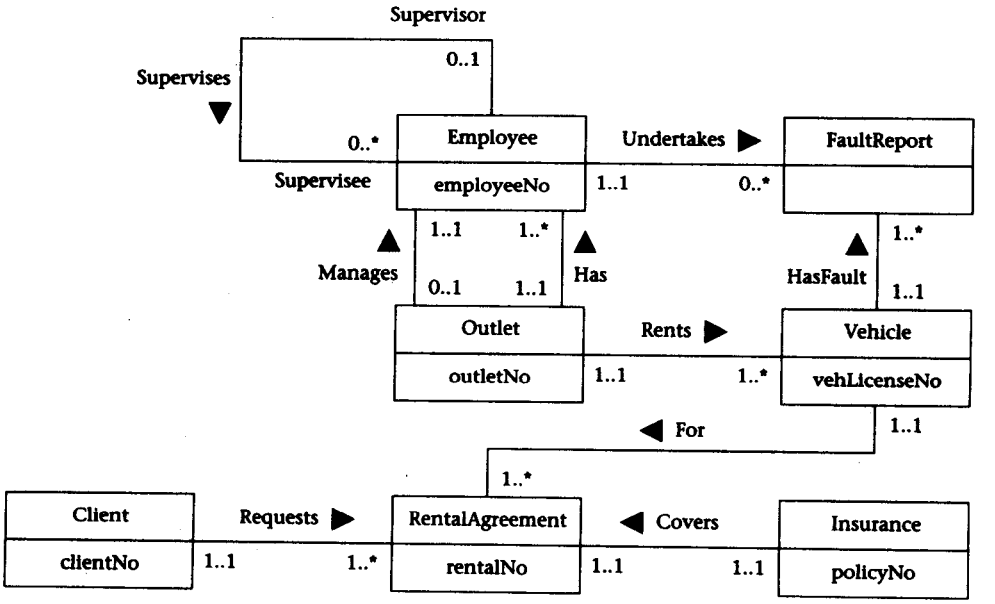
图E-14 工资管理的表

	Foreign Key employeeNo references Employee(employeeNo) Foreign Key deductTypeNo references DeductType(deductTypeNo)
DeductType	(deductTypeNo, deductDescription) Primary Key deductTypeNo
Holiday	(employeeNo, startDate, endDate) Primary Key employeeNo, startDate Foreign Key employeeNo references Employee(employeeNo)
PayDetails	(employeeNo, startDate, routingNumber, accountType, bankName, bankAddress, payTypeNo) Primary Key employeeNo, startDate Foreign Key employeeNo references Employee(employeeNo) Foreign Key payTypeNo references PayType(payTypeNo)
PayHistory	(payNo, employeeNo, payDate, checkNumber, payAmount) Primary Key payNo Foreign Key employeeNo references Employee(employeeNo)
PayType	(payTypeNo, payTypeDescription) Primary Key payTypeNo
SickLeave	(employeeNo, startDate, endDate, reason) Primary Key employeeNo, startDate Foreign Key employeeNo references Employee(employeeNo)

图E-14 （续）

E.8 车辆租赁

一个租赁公司希望建立一个数据库来管理对用户的车辆租赁。公司有不同的部门，每个部门都有一定的员工包括一个经理和几个高级技师，高级技师负责把工作分配给下面的一组普通技工。每个部门都有库存的车辆，以便租给用户最少4个小时到最多6个月。每个用户和公司之间的租赁合同都有唯一的租赁号。用户必须拿出在租用期间的保险金。每次租用过后要对车辆进行检查，以验证它的损坏程度。其逻辑数据模型如图E-15所示，相关的表如图E-16所示。



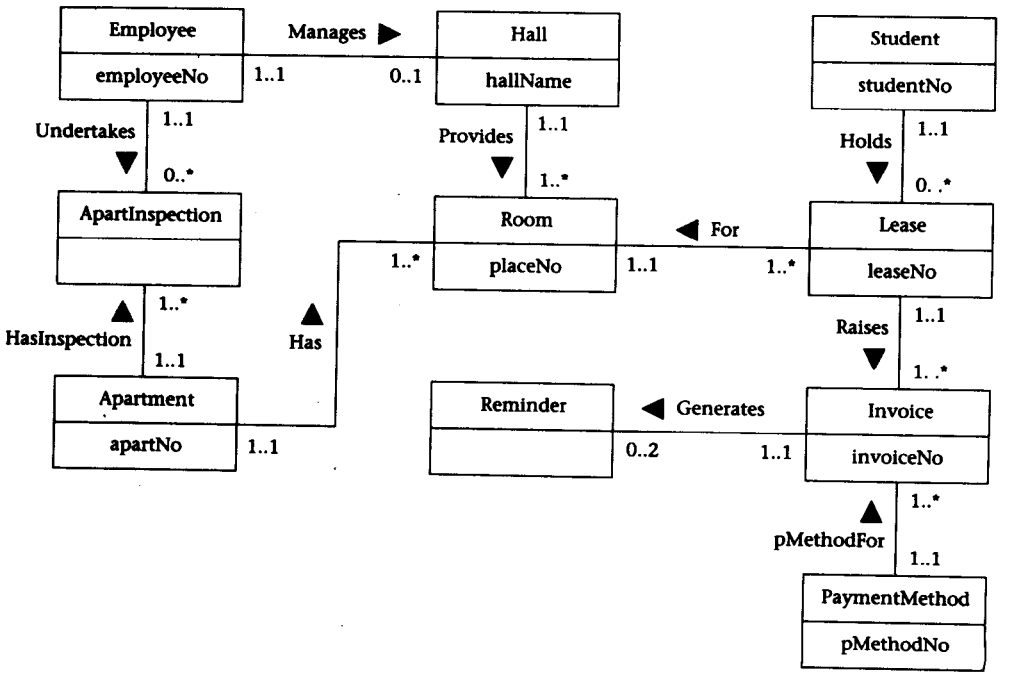
图E-15 车辆租赁的逻辑数据模型

Client	在D.4.2节定义
Employee	(employeeNo, title, firstName, middleName, lastName, address, workTelExt, homeTelNo, empEmailAddress, socialSecurityNumber, DOB, position, sex, salary, dateStarted, outletNo) Primary Key employeeNo Alternate Key socialSecurityNumber Foreign Key outletNo references Outlet(outletNo)
FaultReport	(vehLicenseNo, dateChecked, timeChecked, comments, employeeNo) Primary Key vehLicenseNo, dateChecked Foreign Key vehLicenseNo references Vehicle(vehLicenseNo) Foreign Key employeeNo references Employee(employeeNo)
Outlet	(outletNo, outletStreet, outletCity, outletState, outletZipCode, outletTelNo, outletFaxNo, managerEmployeeNo) Primary Key outletNo Alternate Key outletTelNo Alternate Key outletFaxNo Foreign Key managerEmployeeNo references Employee(employeeNo)
RentalAgreement	(rentalNo, dateStart, timeStart, dateReturn, timeReturn, mileageBefore, mileageAfter, policyNo, insuranceCoverType, insurancePremium, clientNo, vehLicenseNo) Primary Key rentalNo Alternate Key policyNo Foreign Key clientNo references Client(clientNo) Foreign Key vehLicenseNo references Vehicle(vehLicenseNo)
Vehicle	(vehLicenseNo, vehicleMake, vehicleModel, color, noDoors, capacity, hireRate, outletNo) Primary Key vehLicenseNo Foreign Key outletNo references Outlet(outletNo)

图E-16 车辆租赁的表

E.9 学生住宿

一个大学的后勤部门希望建立一个数据库来管理学生宿舍的分配。需要住宿的学生都要



图E-17 学生住宿的逻辑数据模型

填写一个申请表，表中有学生的详细情况和要申请宿舍的类型的介绍以及时间。学生也许租用一个厅室的一个房间或者是学生公寓。厅室只能提供单独的房间，房间有房间号、住宿号以及月租金。住宿号唯一地决定了被后勤部门控制的厅室中的每个房间，以便租给学生时使用。每个厅室由后勤部门的一个成员管理。

后勤部门也提供公寓给学生，每个房间有一个唯一的公寓号。这些公寓房间是已经装修好的而且提供单个房间给3个、4个或者是5个学生一起住。公寓中的每个床位都有月租金、房间号和住宿号。住宿号唯一地确定了所有学生公寓中的可用房间，在房间租给学生时使用。公寓被几个成员共同监督以保证住宿能够维护。

在每个新的学年开始，签订新的租用合同，最少的租用时间为一个学期，最多的是一年。学生要交一个学年的住宿费用，然后每个学期都有一个发票。如果学生在一个规定的日期之前没有交费，则会收到两封提示交费的信。其逻辑数据模型如图E-17所示，相关的表如图E-18所示。

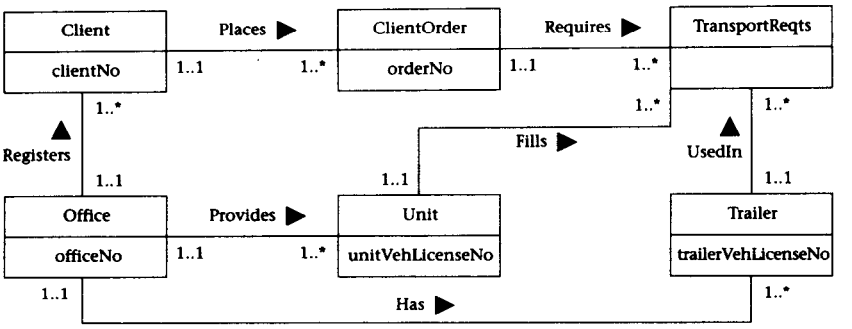
Employee	在D.1.2节定义
PaymentMethod	在D.1.2节定义
Apartment	(apartNo, apartAddress, noOfRoomsInApart) Primary Key apartNo
ApartInspection	(apartNo, dateOfInspection, comments, status, employeeNo) Primary Key apartNo, dateOfInspection Foreign Key apartNo references Apartment(apartNo) Foreign Key employeeNo references Employee(employeeNo)
Hall	(hallName, hallAddress, hallTelNo, hallFaxNo, noOfRoomsInHall, managerEmployeeNo) Primary Key hallName Alternate Key hallTelNo Alternate Key hallFaxNo Foreign Key managerEmployeeNo references Employee(employeeNo)
Invoice	(invoiceNo, semester, dateDue, datePaid, leaseNo, pMethodNo) Primary Key invoiceNo Foreign Key leaseNo references Lease(leaseNo) Foreign Key pMethodNo references PaymentMethod(pMethodNo)
Reminder	(invoiceNo, dateReminder1sent, dateReminder2sent, dateInterview, comments) Primary Key invoiceNo Foreign Key invoiceNo references Invoice(invoiceNo)
Lease	(leaseNo, duration, dateStart, dateLeave, studentNo, placeNo) Primary Key leaseNo Alternate Key placeNo, dateStart Alternate Key studentNo, dateStart Foreign Key studentNo references Student(studentNo) Foreign Key placeNo references Room(placeNo)
Room	(placeNo, roomNo, rentPerSemester, hallName, apartNo) Primary Key placeNo Alternate Key roomNo, hallName Alternate Key roomNo, apartNo Foreign Key hallName references Hall(hallName) Foreign Key apartNo references Apartment(apartNo)
Student	(studentNo, studentFirstName, studentMiddleIntial, studentLastName, studentHomeStreet, studentHomeCity, studentHomeState, studentHomeZipCode, studentHomeTelNo, studentSex, studentDOB, studentType, studentStatus, accommodationTypeRequired, accommodationDuration) Primary Key studentNo

图E-18 学生住宿的表

E.10 客户运送

一个专门在全美国进行运输业务的运送公司希望建立一个数据库以控制用户的运输订单。

客户向一个办事处进行注册可以有一个或多个订单。每个订单都描述了运输的路线，包括各个地点和目的地。这样，每个订单的运输所需就可以计算出来了。运输所需包括这次运输需要的班数和拖车数等。每个办事处分配给几个班和拖车。一个班可以有一辆或两辆拖车。其逻辑数据模型如图E-19所示，相关的表如图E-20所示。



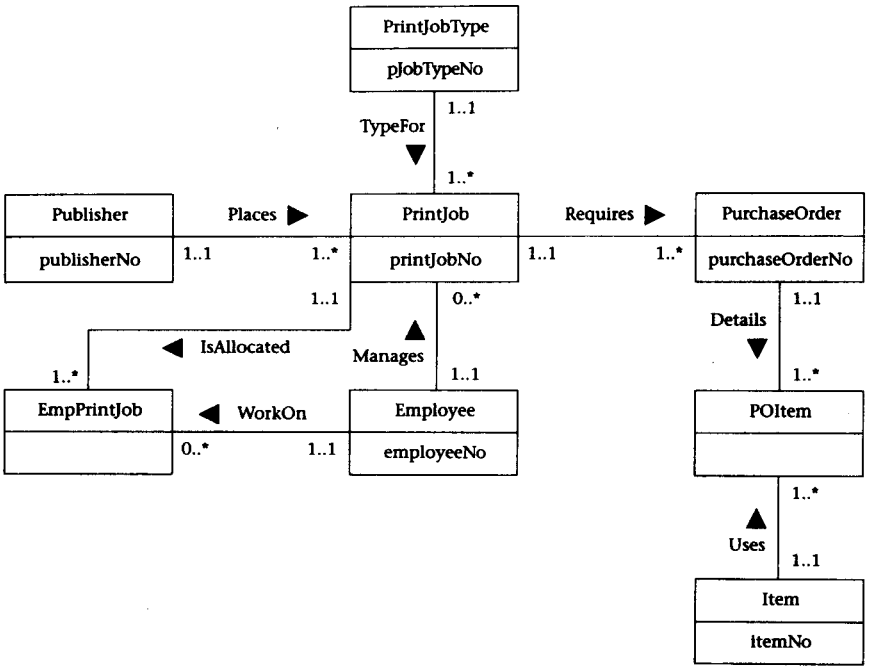
图E-19 客户运送的逻辑数据模型

Client	(clientNo, clientName, clientStreet, clientCity, clientState, clientZipCode, clientTelNo, clientFaxNo, clientWebAddress, contactName, contactTelNo, contactFaxNo, contactEmailAddress, officeNo) Primary Key clientNo Alternate Key clientTelNo Alternate Key clientFaxNo Foreign Key officeNo references Office(officeNo)
Office	(officeNo, officeAddress, officeTelNo, officeFaxNo) Primary Key officeNo Alternate Key officeTelNo Alternate Key officeFaxNo
ClientOrder	(orderNo, dateOrder, collectionDate, collectionAddress, deliveryDate, deliveryAddress, loadWeight, loadDescription, clientNo) Primary Key orderNo Foreign Key clientNo references Client(clientNo)
Trailer	(trailerVehLicenseNo, trailerDescription, trailerLength, maxCarryingWeight, officeNo) Primary Key trailerVehLicenseNo Foreign Key officeNo references Office(officeNo)
TransportReqts	(orderNo, transportReqPartNo, unitVehLicenseNo, trailerVehLicenseNo1, trailerVehLicenseNo2) Primary Key orderNo, transportReqPartNo Foreign Key unitVehLicenseNo references Unit(unitVehLicenseNo) Foreign Key trailerVehLicenseNo1 references Trailer(trailerVehLicenseNo) Foreign Key trailerVehLicenseNo2 references Trailer(trailerVehLicenseNo)
Unit	(unitVehLicenseNo, unitDescription, maxPayLoad, officeNo) Primary Key unitVehLicenseNo Foreign Key officeNo references Office(officeNo)

图E-20 客户运送的表

E.11 出版商印刷

一个印刷书的印刷公司希望建立数据库来处理用户的印刷需要。一本书的出版发行人提交了一份描述印刷工作的单据。印刷工作需要使用材料，如纸和墨水，这些要通过一个或多个购买清单来指定。每个印刷任务都要指派一个管理人来负责保证这次印刷任务的正确执行。对于大的印刷任务，一般还要分配其他的工作人员。其逻辑数据模型如图E-21所示，相关的表如图E-22所示。



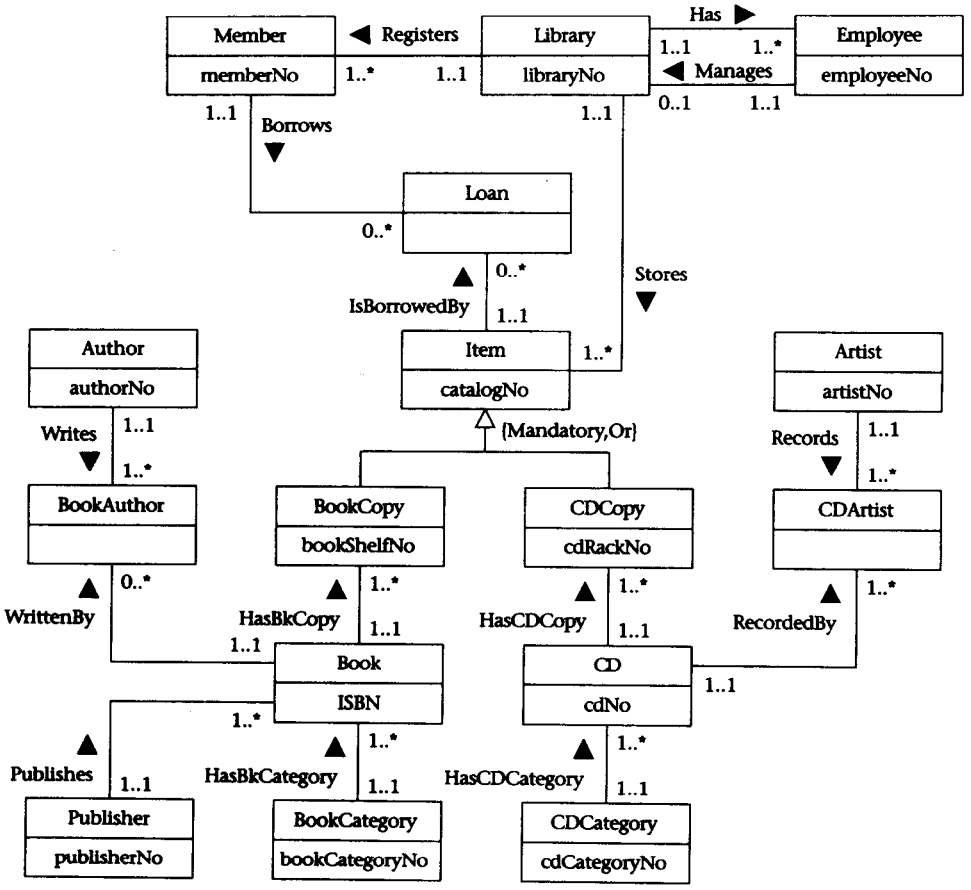
图E-21 出版商印刷的逻辑数据模型

Employee	在D.1.2节定义
EmpPrintJob	(employeeNo, printJobNo, jobDate) Primary Key employeeNo, printJobNo Foreign Key employeeNo references Employee(employeeNo) Foreign Key printJobNo references PrintJob(printJobNo)
Item	(itemNo, itemDescription, itemPrice, itemQuantityInStock, itemReorderLevel, itemReorderQuantity, itemReorderLeadTime) Primary Key itemNo
PrintJob	(printJobNo, printJobDescription, printJobDateReceived, printJobDateCompleted, managerEmployeeNo, publisherNo, printJobTypeNo) Primary Key printJobNo Foreign Key managerEmployeeNo references Employee(employeeNo) Foreign Key publisherNo references Publisher(publisherNo) Foreign Key printJobTypeNo references PrintJobType(printJobTypeNo)
Publisher	(publisherNo, publisherName, publisherStreet, publisherCity, publisherState, publisherZipCode, pubTelNo, pubFaxNo, pubWebAddress, contactName, contactTelNo, contactFaxNo, contactEmailAddress, creditRating) Primary Key publisherNo Alternate Key publisherName Alternate Key pubTelNo Alternate Key pubFaxNo
POItem	(purchaseOrderNo, itemNo, quantity) Primary Key purchaseOrderNo, itemNo Foreign Key purchaseOrderNo references PurchaseOrder (purchaseOrderNo) Foreign Key itemNo references Item(itemNo)
PrintJobType	(printJobTypeNo, printJobTypeDescription) Primary key printJobTypeNo
PurchaseOrder	(purchaseOrderNo, printJobNo, purchaseOrderDate) Primary Key purchaseOrderNo Foreign Key printJobNo references PrintJob(printJobNo)

图E-22 出版商印刷的表

E.12 国家图书馆

一个国家希望建立一个数据库来管理它的当地图书馆。每个图书馆都有一定数量的工作人员，其中的一个被指派为图书馆的经理来管理这里的员工和图书馆每天的工作。每个图书馆都储存有大量的书和CE。每个市民在注册成为图书馆的一名有效的借书人之前不能借任何书，但是在注册之后就可以从国家图书馆里借阅所有的书了。书存储在书架上，而CE存储在图书馆中间的书架上。一般来说，一个图书馆都存有每本书和CE的一定数量的拷贝。书的出版商的详细信息被保存，但不保存CE出版商的详细信息。要寻找一个项目，可以根据书或者CE名、作者/艺术家的名字、书或CE的种类、出版商的名字来查找。其逻辑数据模型如图E-23所示，相关的表如图E-24所示。



图E-23 国家图书馆的逻辑数据模型

Publisher	在D.11.2节定义
Artist	(artistNo, name) Primary Key artistNo
Author	(authorNo, name) Primary Key authorNo
Book	(ISBN, title, year, publisherNo, bookCategoryNo) Primary Key ISBN

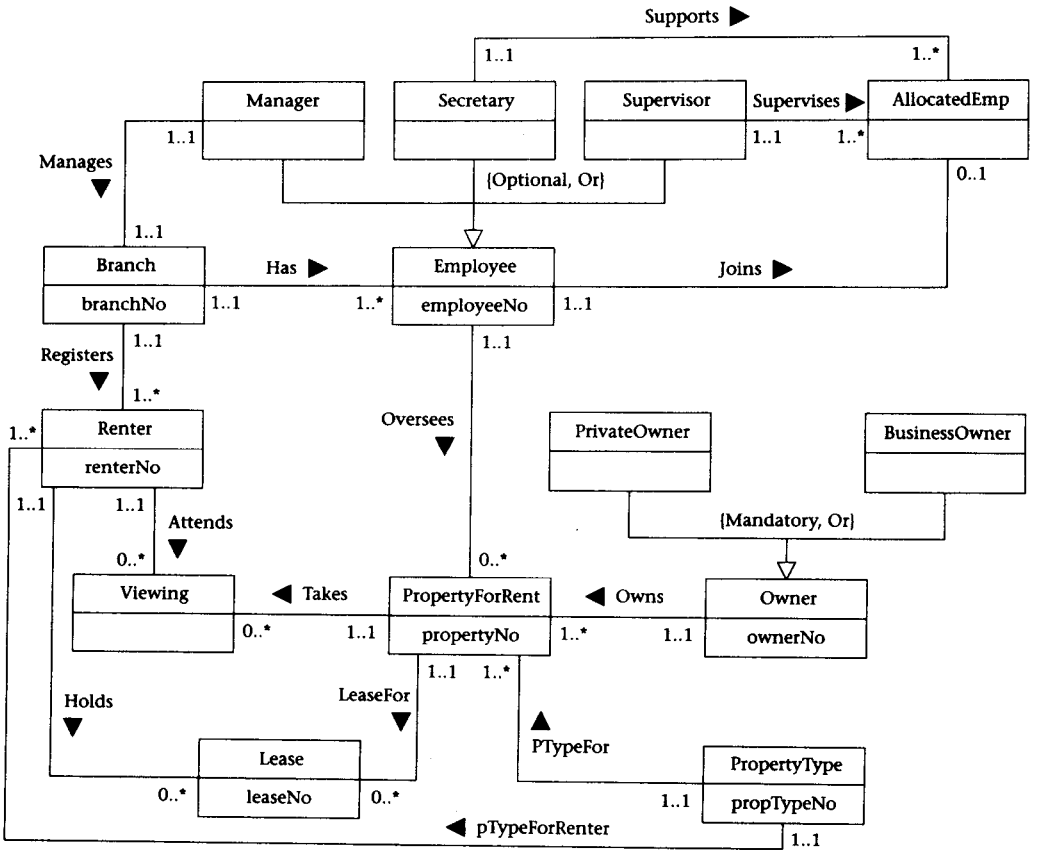
图E-24 国家图书馆的表

	Foreign Key publisherNo references Publisher(publisherNo) Foreign Key bookCategoryNo references BookCategory(bookCategoryNo)
BookAuthor	(ISBN, authorNo) Primary Key ISBN, authorNo Foreign Key ISBN references Book(ISBN) Foreign Key authorNo references Author(authorNo)
BookCategory	(bookCategoryNo, bookCatDescription) Primary Key bookCategoryNo
BookCopy	(catalogNo, bookShelfNo, ISBN, dateInStock, libraryNo) Primary Key catalogNo Alternate Key bookShelfNo Foreign Key ISBN references Book(ISBN) Foreign Key libraryNo references Library(libraryNo)
CD	(cdNo, title, releaseDate, cdCategoryNo) Primary Key cdNo Foreign Key cdCategoryNo references CDCategory(cdCategoryNo)
CDArtist	(cdNo, artistNo) Primary Key cdNo, artistNo Foreign Key cdNo references CD(cdNo) Foreign Key artistNo references Artist(artistNo)
CDCategory	(cdCategoryNo, cdCatDescription) Primary Key cdCategoryNo
CDCopy	(catalogNo, cdRackNo, cdNo, dateInStock, libraryNo) Primary Key catalogNo Alternate Key cdRackNo Foreign Key cdNo references CD(cdNo) Foreign Key libraryNo references Library(libraryNo)
Employee	(employeeNo, title, firstName, middleName, lastName, address, workTelExt, homeTelNo, empEmailAddress, socialSecurityNumber, DOB, position, sex, salary, dateStarted, libraryNo) Primary Key employeeNo Alternate Key socialSecurityNumber Foreign Key libraryNo references Library(libraryNo)
Library	(libraryNo, libStreet, libCity, libState, libZipCode, libTelNo, libFaxNo, libWebAddress, managerEmployeeNo) Primary Key libraryNo Alternate Key libTelNo Alternate Key libFaxNo Foreign Key managerEmployeeNo references Employee(employeeNo)
Loan	(catalogNo, memberNo, dateOut, dateReturn) Primary Key catalogNo, memberNo Foreign Key catalogNo references BookCopy(catalogNo) and CDCopy(catalogNo) Foreign Key memberNo references Member(memberNo)
Member	(memberNo, memTitle, memFirstName, memMiddleName, memLastName, memAddress, memWorkTelExt, memHomeTelNo, memDOB, memSex, dateJoined, libraryNo) Primary Key memberNo Foreign Key libraryNo references Library(libraryNo)

图E-24 （续）

E.13 房地产租赁

一个在全美都有分支机构的房地产租赁机构希望建立一个数据库来管理代表房主出租房屋，房主被划分为Business拥有者和Private拥有者。在每个分支机构中，全体员工检查房屋租赁情况，而且负责房屋检查和租用合同。一些员工也被指定为Supervisor角色，负责监督一组员工并保证分支机构的有效管理。每组员工的管理工作由一个Secretary来承担。其逻辑数据模型如图E-25所示，相关的表如图E-26所示。



图E-25 房地产租赁的逻辑数据模型

AllocatedEmp	(superviseeEmployeeNo, supervisorEmployeeNo, secretaryEmployeeNo) Primary Key superviseeEmployeeNo Foreign Key superviseeEmployeeNo references Employee(employeeNo) Foreign Key supervisorEmployeeNo references Employee(employeeNo) Foreign Key secretaryEmployeeNo references Employee(employeeNo)
Branch	(branchNo, branchStreet, branchCity, branchState, branchZipCode, branchTelNo, branchFaxNo, managerEmployeeNo) Primary Key branchNo Alternate Key branchTelNo Alternate Key branchFaxNo Foreign Key managerEmployeeNo references Employee(employeeNo)
BusinessOwner	(ownerNo, businessName, businessAddress, businessTelNo, businessFaxNo, contactName, contactTelNo, contactFaxNo, contactEmailAddress) Primary Key ownerNo Alternate Key businessName Alternate Key businessTelNo Alternate Key businessFaxNo
Employee	(employeeNo, title, firstName, middleName, lastName, address, workTelExt, homeTelNo, empEmailAddress, socialSecurityNumber, DOB, position, sex, salary, typingSpeed, dateStarted, branchNo) Primary Key employeeNo Alternate Key socialSecurityNumber Foreign Key branchNo references Branch(branchNo)
Lease	(leaseNo, rentStart, rentFinish, depositPaid, renterNo, propertyNo) Primary Key leaseNo

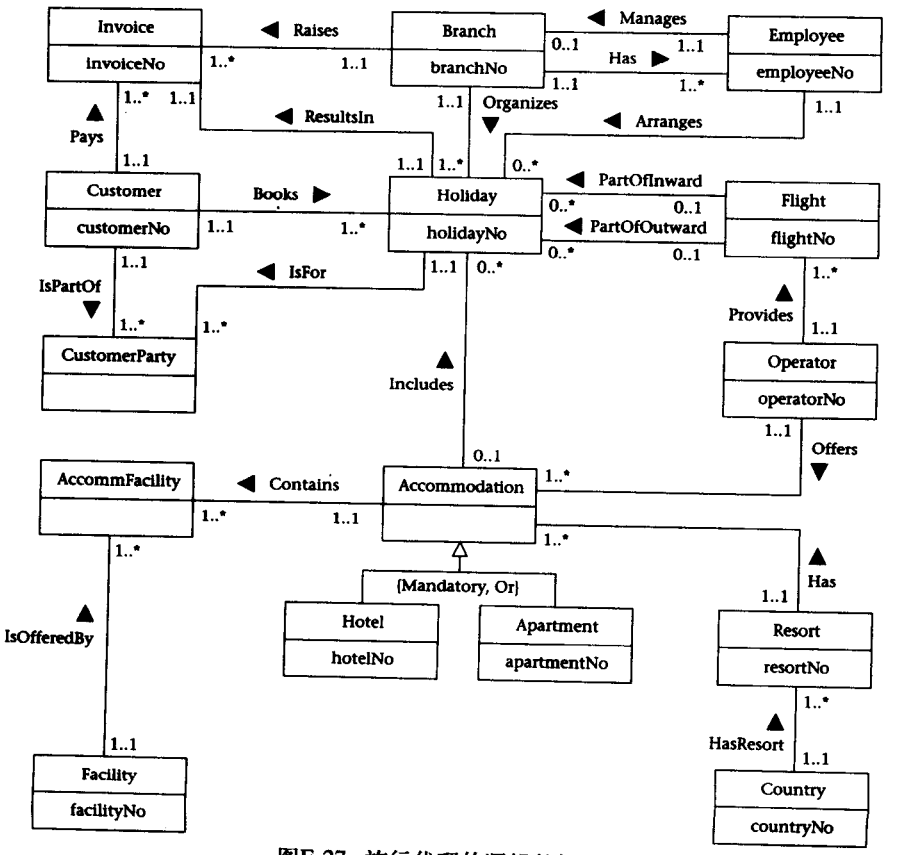
图E-26 房地产租赁的表

	Foreign Key renterNo references Renter(renterNo) Foreign Key propertyNo references PropertyForRent(propertyNo)
PrivateOwner	(ownerNo, ownerName, ownerAddress, ownerTelNo) Primary Key ownerNo
PropertyForRent	(propertyNo, propStreet, propCity, propState, propZipCode, noRooms, rent, propTypeNo, ownerNo, employeeNo, branchNo) Primary Key propertyNo Foreign Key propTypeNo references PropertyType(propTypeNo) Foreign Key ownerNo references PrivateOwner(ownerNo) and BusinessOwner(ownerNo) Foreign Key employeeNo references Employee(employeeNo) Foreign Key branchNo references Branch(branchNo)
PropertyType	(propTypeNo, propTypeDescription) Primary Key propTypeNo
Renter	(renterNo, rFName, rLName, rAddress, rTelNo, maxRent, prefTypeNo) Primary Key renterNo Foreign Key prefTypeNo references PropertyType(propTypeNo)
Viewing	(propertyNo, renterNo, dateView, comments) Primary Key propertyNo, renterNo, dateView Foreign Key propertyNo references PropertyForRent(propertyNo) Foreign Key renterNo references Renter(renterNo)

图E-26 （续）

E.14 旅行代理

一个旅行代理机构希望为他的消费者假日登记行为建立一个数据库。旅行代理在美国的



图E-27 旅行代理的逻辑数据模型

主要城市都分有分公司。消费者可以通过电话在任何分公司登记假期旅行。尽管有时用户只需要飞机航班或者是食宿，但是每个假期活动中一般都有飞机航班和食宿。一旦旅行机构找到了适合消费者的活动，就会提供航班和食宿。但是，预约只能保留24个小时，在这期间消费者必须决定接受或拒绝这次登记。一旦接受这次登记，假日旅游的账单就会发给用户，而用户也必须在出发前最少四个星期一次性付清。开始为用户登记的员工的名字也要记录下来。其逻辑数据模型如图E-27所示，相关的表如图E-28所示。

Branch	在D.13.2节定义
ApartFacility	(apartmentNo, facilityNo, comments) Primary Key apartmentNo, facilityNo Foreign Key apartmentNo references Apartment(apartmentNo) Foreign Key facilityNo references Facility(facilityNo)
Apartment	(apartmentNo, apartmentName, apartmentType, apartmentDescription, apartmentRating, apartmentStreet, apartmentCity, apartmentState, apartmentCountry, apartmentZipCode, noOfRooms, operatorNo, resortNo) Primary Key apartmentNo Foreign Key operatorNo references Operator(operatorNo) Foreign Key resortNo references Resort(resortNo)
Country	(countryNo, countryName) Primary Key countryNo Alternate Key countryName
Customer	(customerNo, customerName, customerStreet, customerCity, customerState, customerZipCode, custTelNo, custFaxNo, nationality, sex, DOB, passportNo) Primary Key customerNo Alternate Key custTelNo Alternate Key custFaxNo Alternate Key passportNo
CustomerParty	(customerNo, holidayNo) Primary Key customerNo, holidayNo Foreign Key customerNo references Customer(customerNo) Foreign Key holidayNo references Holiday(holidayNo)
Employee	(employeeNo, title, firstName, middleName, lastName, address, workTelExt, homeTelNo, empEmailAddress, socialSecurityNumber, DOB, position, sex, salary, dateStarted, branchNo) Primary Key employeeNo Alternate Key socialSecurityNumber Foreign Key branchNo references Branch(branchNo)
Facility	(facilityNo, description, additionalCharge) Primary Key facilityNo
Flight	(flightNo, planeType, seatCapacity, airportDepart, departTime, airportArrive, arriveTime, operatorNo) Primary Key flightNo Foreign Key operatorNo references Operator(operatorNo)
Hotel	(hotelNo, hotelName, hotelStreet, hotelCity, hotelState, hotelCountry, hotelZipCode, hotelTelNo, hotelFaxNo, hotelType, hotelDescription, hotelRating, hotelManagerName, operatorNo, resortNo) Primary Key hotelNo Foreign Key operatorNo references Operator(operatorNo) Foreign Key resortNo references Resort(resortNo)
HotelFacility	(hotelNo, facilityNo, comments) Primary Key hotelNo, facilityNo Foreign Key hotelNo references Hotel(hotelNo) Foreign Key facilityNo references Facility(facilityNo)
Holiday	(holidayNo, status, dateBooked, cateringType, startDate, finishDate, invoiceNo, totalCost, dateSent, datePaid, bookCustomerNo, hotelNo, apartmentNo, inwardFlightNo, inwardNoOfSeats, outwardFlightNo, outwardNoOfSeats, employeeNo, branchNo) Primary Key holidayNo Foreign Key bookCustomerNo references customer(customerNo)

图E-28 旅行代理的表

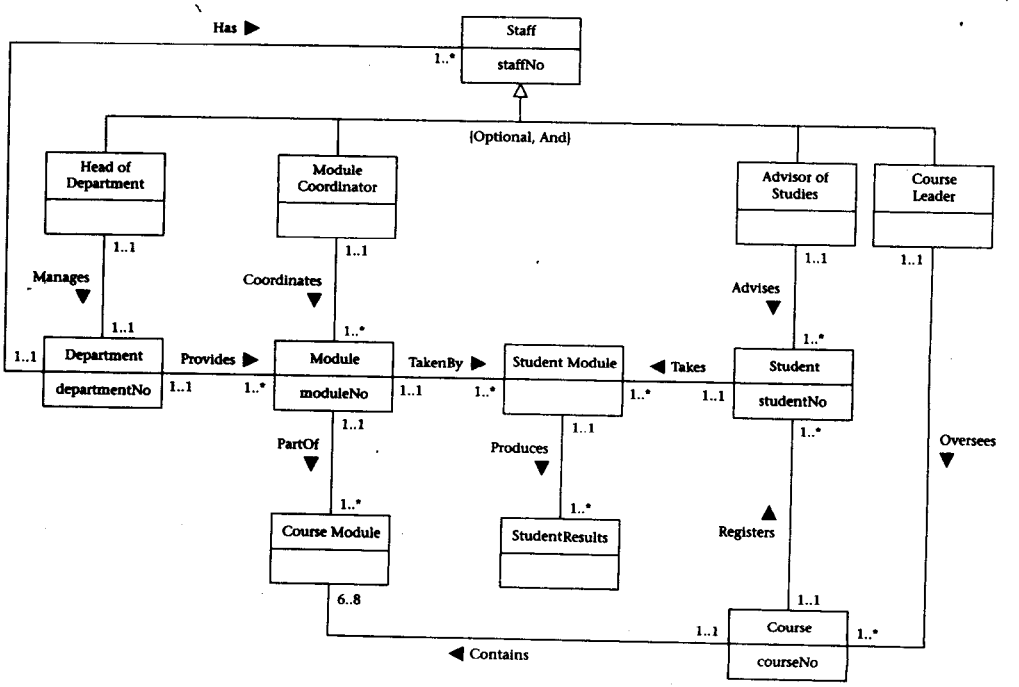
	Foreign Key hotelNo references Hotel(hotelNo) Foreign Key apartmentNo references Apartment(apartmentNo) Foreign Key inwardFlightNo references Flight(flightNo) Foreign Key outwardFlightNo references Flight(flightNo) Foreign Key employeeNo references Employee(employeeNo) ForeignKey branchNo references Branch(branchNo)
Operator	(operatorNo, operatorName, operatorType, operatorStreet, operatorCity, operatorState, operatorZipCode, operTelNo, operFaxNo, contactName, contactTelNo, contactFaxNo, contactEmailAddress) Primary Key operatorNo Alternate Key operTelNo Alternate Key operFaxNo
Resort	(resortNo, resortName, distanceFromAirport, timeFromAirport, countryNo) Primary Key resortNo Foreign Key countryNo references Country(countryNo)

图E-28 （续）

E.15 学生管理

一所大学希望建立一个数据库以便对学生进行管理。当一个学生进入学校时，他就会选择一个专业。每个学生也会指定一个指导老师。每个专业每年由一些课程组成。组成专业的课程的最小和最大数目分别是6和8。每个学生都要选修这些课程并且通过，否则就不允许进入下一年的下一个专业学习或者是毕业。一般情况下给一个学生三次机会来通过这门课程的考试。但是，也可以在任意的其他大学完成课程。有些特殊的课程可以在一个或多个专业中存在。

一个大学可以有几个部门，每个部门可以提供一个课程的部长职务。每个部门都有一个部门负责人，每门课程也都有一个课程领导。每个课程都会分配一定数量的员工（称为课程合作人）来负责课程的估价和教学。其逻辑数据模型如图E-29所示，相关的表如图E-30所示。



图E-29 学生管理的逻辑模型

Course	(courseNo, courseName, level, entranceRequirements, maxNumber, departmentNo, courseLeaderNo) Primary Key courseNo Alternate Key courseName Foreign Key departmentNo references Department(departmentNo) Foreign Key courseLeaderNo references Department(courseLeaderNo)
CourseModule	(courseNo, moduleNo) Primary Key courseNo, moduleNo Foreign Key courseNo references Course(courseNo) Foreign Key moduleNo references Module(moduleNo)
Department	(departmentNo, departmentName, location, HODstaffNo) Primary Key departmentNo Alternate Key departmentName Foreign Key HODstaffNo references Staff(staffNo)
Module	(moduleNo, moduleName, semesterDelivered, moduleAims, moduleObjectives, moduleSyllabus, moduleResources, moduleModeOfAssessment, moduleCoordinatorStaffNo, departmentNo) Primary Key moduleNo Alternate Key moduleName Foreign Key moduleCoordinatorStaffNo references Staff(staffNo) Foreign Key departmentNo references Department(departmentNo)
Staff	(staffNo, title, firstName, lastName, address, homeTelNo, workTelExt, empEmailAddress, socialSecurityNumber, DOB, position, sex, salary, dateStarted, departmentNo) Primary Key staffNo Alternate Key socialSecurityNumber Foreign Key departmentNo references Department(departmentNo)
Student	(studentNo, studentFirstName, studentMiddleName, studentLastName, studentHomeStreet, studentHomeCity, studentHomeState, studentHomeZipCode, studentHomeTelNo, familyHomeStreet, familyHomeCity, familyHomeState, familyHomeZipCode, familyHomeTelNo, studentDOB, studentSex, nationality, courseNo, advisorStaffNo) Primary Key studentNo Foreign Key courseNo references Course(courseNo) Foreign Key advisorStaffNo references Staff(staffNo)
StudentModule	(studentNo, moduleNo) Primary key studentNo, moduleNo Foreign Key studentNo references Student(studentNo) Foreign Key moduleNo references Module(moduleNo)
StudentResult	(studentNo, moduleNo, attempt, attemptDate, mark, proposal, additionalComments) Primary key studentNo, moduleNo, attempt Foreign Key studentNo, moduleNo references StudentModule(studentNo, moduleNo)

图E-30 学生管理的表

术 语 表

Access method（访问方法）：此步骤包括从文件中存储和检索记录。

Alias（别名）：某属性的另一个名字。在SQL中，可以用别名替换表名。

Alternate key（备用键，ER/关系模型）：在实体/表中没有被选为主键的候选键。

Anomalies（异常）：参见update anomalies。

Application design（应用程序设计）：数据库应用程序生命周期的一个阶段，包括设计用户界面以及使用和处理数据库的应用程序。

Application server（应用服务器）：处理业务逻辑以及三层结构中的数据处理层。

Attribute（属性）（关系模型）：属性是关系中命名的列。

Attribute（属性）（ER模型）：实体或关系中的一个性质。

Attribute inheritance（属性继承）：子类成员可以拥有其特有的属性，并且继承那些与超类有关的属性的过程。

Authentication（认证）：确定用户是否就是他所声明的那个人。

Authorization（授权）：使主体可以合法地访问数据库系统和数据库系统对象的权限授予过程。

Backup（备份）：定期将数据库和日志文件（也可能还有程序）拷贝到离线存储介质的过程。

Base table（基本表）：一个命名的表，其记录物理地存储在数据库中。

Binary relationship（二元关系）：一个ER术语，用于描述两个实体间的关系。例如，Branch Has Staff。

Bottom-up approach（自底向上方法）（用于数据库设计）：一种设计方法学，它从标识每个设计组件开始，然后将这些组件聚合成一个大的单元。在数据库设计中，可以从标识属性开始底层设计，然后将这些属性组合在一起构成代表实体和关系的表。

Business rules（业务规则）：定义或约束组织的某些方面的规则。

Candidate key（候选键，ER/关系模型）：仅包含唯一标识实体所必需的最小数量的属性/列的超键。

Cardinality（基数）：描述每个参与实体的可能的关系数目。

Centralized approach（集中式方法，用于数据库设计）：将每个用户视图的需求合并成新数据库应用程序的一个需求集合。

Chasm trap（深坑陷阱）：假设实体间存在一个关系，但某些实体间不存在通路。

Client（客户端）：向一个或多个服务器请求服务的软件应用程序。也可参见两层/三层客户服务器体系结构（two-tier/three-tier client-server architecture）。

Clustering field（聚簇字段）：记录中的任何用于聚簇（集合）行记录的非键字段，这些行在这个字段上有相同的值。

Clustering index（聚簇索引）：在文件的聚簇字段上定义的索引。一个文件最多有一个主

索引或一个聚簇索引。

Column（列）：参见attribute。

Complex relationship（复杂关系）：度数大于2的关系。

Composite attribute（复合属性）：由多个简单组件组成的属性。

Composite key（复合键）：包含多个列的主键。

Concurrency control（并发控制）：在多用户环境下同时执行多个事务并保证数据完整性的一个DBMS服务。

Constraint（约束）：数据库不允许包含错误数据的一致性规则。

Data administration（数据管理）：管理和控制公司数据，包括数据库规划、开发和维护标准、策略和过程，以及概念和逻辑数据库设计。

Data conversion and loading（数据转换与加载）：数据库系统开发生命周期中的一个阶段，包括转换现有数据到新数据库中以及将现有应用程序转换到新的数据库上运行。

Data dictionary（数据字典）：参见system catalog。

Data independence（数据独立性）：使用数据的应用程序的数据描述部分。这意味着，如果将新的数据结构添加到数据库中，或者数据库中现有的结构被修改了，那么使用此数据库的应用程序就会受到影响，除非应用程序不直接依赖于被修改的部分。

Data mart（数据集市）：数据仓库的子集，支持特定部门和业务领域的需求。

Data mining（数据挖掘）：从大的数据库中抽取有效的、以前不知道的、可理解的和可操作的信息的过程，并使用这些信息制定重要的商业决策。

Data model（数据模型）：描述数据、数据间的关系以及公司所使用的数据的约束的概念集合。

Data redundancy（数据冗余）：参见redundant data。

Data security（数据安全）：包括对数据库对象（如表和视图）的访问和使用以及用户可以在这些对象上实施的操作。

Data warehouse（数据仓库）：从不同的操作数据源抽取出来的企业数据的固定的或集成的视图，有很多最终用户工具可用于支持从简单到很复杂的查询来支持决策制定。

Database（数据库）：逻辑上相关可共享的数据（以及这些数据的描述）集合，用于处理公司所需的信息。

Database adminstration（数据库管理）：对数据库应用的物理实现进行管理和控制，包括物理数据库设计和实现、建立安全性和完整性控制、监视系统性能以及必要时重新组织数据库。

(database) Application program（(数据库）应用程序）：通过给DBMS发出合适的查询（通常是一个SQL语句）与数据库交互作用的计算机程序。

Database design（数据库设计）：数据库应用生命周期中的一个阶段，包括创建一个支持公司的操作和目标的数据库的设计。

Database integrity（数据库完整性）：指存储数据的正确性和一致性。完整性通常用约束来表达。

Database Management Systems（数据库管理系统，DBMS）：一个能够让用户定义、创建和维护数据库并控制对数据库的访问的软件系统。

Database planning（数据库规划）：尽可能有效地实现数据库应用的各阶段的管理活动。

Database security（数据库安全性）：防止数据库被有意或无意破坏的机制。RDBMS通常提供两种类型的安全：数据安全和系统安全。

Database server（数据库服务器）：同服务器（参见两层/三层客户端-服务器体系结构）。

DBMS engine（DBMS引擎）：同服务器（参见两层客户端-服务器体系结构）。

DBMS selection（选择DBMS）：数据库应用生命周期中的一个阶段，包括选择一个合适的DBMS来支持数据库应用。

Degree of a relationship（关系的度）：一个关系中参与的实体的个数。

Denormalization（反规范化）：形式上，这个术语指的是对基本表结构的修改，这样新的表比原始的表的规范化程度要低。但也可以用此术语更宽泛地形容将两个表合并成一个新表的情景，而这个新表与原来的表具有相同的范式，但比原始表包含更多的空值。

Derived attribute（派生属性）：表示其值可以从一个相关属性或属性集的值派生得到的属性，这个属性在实体中不是必需的。

Design methodology（设计方法学）：一种结构化的方法，它使用过程、工具和文档来支持和简化设计过程。

Disjoint constraint（无连接约束）：描述子类的成员间的关系，并指明超类某个成员是否有可能成为一个或多个子类的成员。

Distributed database（分布式数据库）：多个逻辑上相互关联的、共享数据的（并且共享数据的描述）、物理上分布在计算机网络上的数据库的集合。

Distributed DBMS（分布式DBMS，DDBMS）：透明地管理分布式数据库的软件。

Domain（域）：一个或多个属性的取值范围。

Encryption（加密）：用一个特定的算法对数据进行编码，使数据不能被没有密钥的程序读取。

Entity（实体）：具有相同性质的对象的集合，它是由用户或公司标识并可以独立存在的。

Entity integrity（实体完整性）：在一个基本表中，主键列的取值不能为空。

Entity occurrence（实体出现）：实体中的一个唯一可标识的对象。

Entity-Relationship model（实体-关系模型）：公司的实体、属性和关系的详细逻辑表示。

Fact-finding（事实发现）：使用诸如面谈和提问等技术收集关于系统的事实、需求和性能的形式化过程。

Fan trap（扇形陷阱）：当从第三个实体扇出的两个实体有1:*关系时出现扇形陷阱，但这两个实体在它们之间应该有直接关系以提供必要的信息。

Field（字段）：同tuple。

File（文件）：存储在辅助存储器中的相关记录的一个命名集合。

File-based system（基于文件的系统）：一个文件集合，用来管理（创建、插入、删除、更新和检索）一个或多个文件中的数据，并产生基于这些文件中的数据的应用（通常是报表）。

File organization（文件组织）：当文件存储在磁盘上时，对文件中的记录的安排方式。

First normal form（第一范式，1NF）：表中每个列的交叉处以及记录包含且仅包含一个值的表。

Foreign key（外键）：一个表中的一个列或者多个列的集合，这些列匹配某些其他（也可

能是同一个）表中的候选键。

Fourth-Generation Language（第四代语言，4GL）：一种非过程化语言，比如SQL，它只需要用户定义必须完成什么操作，4GL负责将所进行的操作翻译成如何实现这些操作。

Full functional dependency（完全函数依赖）：一个列在功能上依赖于复合主键，但不依赖于主键的任何一个子集的条件。

Functional dependency（函数依赖）：描述表中列之间的关系。例如，如果A和B是某个表中的列，若A的每个值只与B中的一个确定值对应，则B在功能上依赖于A（表示为A→B）。（A和B每个都可以包含多个列。）

Generalization（泛化）：通过标识实体间的公共特征使实体间差别最小化的过程。

Generalization hierarchy（泛化层次结构）：同type hierarchy。

Global data model（全局数据模型）：代表整个公司（或被模型化的公司的一部分）的数据模型。

Implementation（实现）：数据库应用生命周期中的一个阶段，包括数据库和应用程序设计的物理实现。

Index（索引）：一种允许DBMS将特定的记录更快地放置到文件中，从而加快对用户查询的响应的数据结构。

Information system（信息系统）：能够在整个公司范围内收集、管理、控制和分发数据/信息的资源。

Inheritance（继承）：参见attribute inheritance。

Integrity constraints（完整性约束）：防止出现数据库中的数据不一致的约束。

IS-A hierarchy（IS-A层次结构）：同type hierarchy。

Journaling（日记）：将所有对数据库的修改保存和维护在日志文件中的过程，使在失败时可用它有效地恢复数据库。

Local logical data model（局部逻辑数据模型）：代表特定用户视图或用户视图的组合的数据模型。

Logical database design（逻辑数据库设计）：基于特定的数据模型构建公司的数据的模型的过程，但不依赖于特定的DBMS以及其他的物理条件。

Meta-data（元数据）：关于数据的数据，参见system catalog。

Mission objective（使命目标）：标识数据库必须支持的特定任务。

Mission statement（使命语句）：定义数据库应用程序的主要目标。

Multiplicity（多样性）：定义与某个相关实体的一次出现有关的实体的出现数目。

Multi-valued attribute（多值属性）：为一个实体的出现保存多个值的属性。

Nonkey attribute/column（非键属性/列）：不是键的一部分的属性/列。

Normal forms（范式）：规范化过程的一个阶段。前三个范式分别为第一范式（1NF）、第二范式（2NF）和第三范式（3NF）。

Normalization（规范化）：一种产生带有需要的特性的技术，这种特性能支持用户和公司的需求。

Null（空值）：表示一个列的值目前还不知道或对于当前记录来说还不能使用。

Object-oriented Data Model（面向对象数据模型，OODM）：在面向对象程序设计中捕获

对象语义的数据模型。

Object-oriented Database（面向对象数据库，OODB）：在面向对象数据模型中定义的持久的和共享的对象的信息库。

Object-oriented DBMS（面向对象DBMS，OODBMS）：面向对象数据库的管理程序。

Object-relational DBMS（面向关系DBMS，ORDBMS）：扩展关系DBMS结合“对象”的一些概念。没有单独的ORDBMS，它是这样的一些系统，其特性依赖于扩展的方式和程度。

OnLine Analytical Processing（联机分析处理，OLAP）：对大量的多维数据进行动态的综合、分析和合并。OLAP描述了使用统计数据的多维视图以便快速访问进行高级数据分析所需的重要信息的技术。

Operational maintenance（操作性维护）：数据库应用生命周期的一个阶段，包括监视和维护系统安装后的运行。

Participation constraint（参与约束，EER模型）：确定超类中的每个出现是否必须作为子类的一个成员进行参与。

Participation constraint（参与约束，ER模型）：确定是否所有或者仅仅是某些实体出现参与到关系中。

Physical database design（物理数据库设计）：在二级存储上产生数据库实现的描述的过程，它描述基本表、文件的组织、用于获得有效访问的索引以及所有与完整性约束和安全性限制有关的说明。

Primary index（主索引）：在文件的有序键字段上构建的索引。一个文件最多可以有一个主索引或一个群集索引。有序键用于保证在每个记录中有一个唯一的值。

Primary key（主键，ER模型）：用来标识每个实体的出现的候选键。

Primary key（主键，关系模型）：唯一标识表中记录的候选键。

privileges（权限）：允许用户在给定基本表或视图上执行的操作。

Prototyping（构建原型）：数据库应用程序生命周期的一个阶段，包括构建数据库应用程序的工作模型。

QBE（Query-by-Example）：一种用于关系DBMS的非过程化的数据库语言。QBE是一个图形化的“点-按”查询数据库的方法。

RDBMS：关系型DBMS。

Record（记录）：同tuple。

Recovery control（恢复控制）：当失败时，将数据库还原到正确状态的过程。

Recursive relationship（递归关系）：一种关系，当同一个实体在不同的角色中参与多次时，就会出现递归关系。例如，Staff Supervises Staff。

Redundant data（冗余数据）：在多个表中存储的重复数据。

Referential integrity（参照完整性）：如果一个表中存在外键，则外键值必须与主表中的某些记录的候选键键值相同，或者外键值必须全部为空。

Relation（关系）：具有有列和行的表。

Relational model（关系模型）：以表（或关系）的形式表示数据的数据模型。

Relational database（关系数据库）：规范化的表的集合。

Relationship（关系）：实体间有意义的关系。

Relationship occurrence（关系出现）：两个实体出现之间的唯一可标识的联系。

Replication（复制）：在一个或多个站点上产生数据的多个拷贝的过程。

Requirements collection and analysis（需求收集与分析）：数据库应用程序生命周期的一个阶段，包括收集和分析数据库应用程序所要支持的关于公司的信息，并使用这些信息来标识新的数据库应用的需求。

Row（行）：同元组（tuple）。

Second normal form（第二范式，2NF）：一个已经是第一范式的表，同时满足所有的非主键列只能从构成主键的全部列中获得。

Secondary index（二级索引）：在数据文件的非有序字段上定义的索引。

Security（安全）：指防止数据库被非授权的用户访问，包括有意的或无意的。RDBMS通常提供两种类型的安全：数据安全和系统安全。

Server（服务器）：为发出请求的客户提供服务的软件应用程序。参见两层/三层客户端-服务器体系结构。

Simple attribute（简单属性）：只有一个组件的属性。

Single-valued attribute（单值属性）：对于一个实体出现只有一个值的属性。

Specialization（特化）：通过标识用来区分实体间成员的特征来最大化实体间成员的差别的过程。

Specialization hierarchy（特化层次结构）：同type hierarchy。

SQL（Structured Query Language，结构化查询语言）：一种用于RDBMS的非过程化数据库语言。换言之，你只需指定你需要哪些信息，而不需要指定如何得到这些信息。SQL已经被国际标准化组织（ISO）标准化了，因此SQL是定义和操纵RDBMS的正式的和实际上的标准语言。

Strong entity（强实体）：一个不依赖于其他实体的主键的存在而存在的实体。

Subclass（子类）：为（超类）实体中的某些出现并保持特定属性和关系并有不同角色的实体。

Superclass（超类）：为实体中的所有出现保存公共属性和关系的实体。可参见specialization和generalization。

Superkey（超键，ER模型）：一个属性或属性集，唯一地标识了每个实体的出现。

Superkey（超键，关系模型）：一个列或者列集，唯一地标识了表中的一个记录。

System catalog（系统目录）：保存关于数据库的结构、用户、应用程序等信息的数据。

System definition（系统定义）：数据库应用生命周期中的一个阶段，包括定义数据库应用程序以及它的主要用户视图的范围和边界。

System security（系统安全）：在系统级保护数据库的访问和使用，比如用户名和密码。

Table（表）：同relation。

Table diagram（表图）：在数据库（包括主键和外键）中对表的图形化的表示。

Ternary relationship（三元关系）：三个实体间的关系。例如，Branch、Staff和Member间的Registers关系。

Testing（测试）：数据库应用生命周期的一个阶段，包括执行应用程序并有意地发现错误。

Third normal form（第三范式，3NF）：一个已经是1NF和2NF的表，同时满足所有的非主键列的值仅能从主键列得到，而不能从其他列得到。

Third-Generation Language（第三代语言，3GL）：一种过程化的语言，比如COBOL、C、C++，它需要用户（通常是程序员）指定必须要干什么事情以及如何干这些事情。

Threat（威胁）：有意的或无意的可能会影响系统并进而影响企业的情况或事件。

Three-tier client-server architecture（三层客户端-服务器体系结构）：由处理用户界面的客户和处理业务逻辑的应用程序服务器以及数据处理层组成，而数据库服务器是用来运行DBMS的。

Top-down approach（自顶向下方法，用于数据库设计）：一种设计方法，此种方法从定义系统的主要结构开始，然后将这些结构逐步细分成更小的单元。在数据库设计中，通过标识实体和数据间的关系开始这个顶层的步骤，然后逐步添加细节，比如你希望保存的关于实体和关系的信息（称为属性）以及在实体、关系和属性上的所有约束。

Transaction（事务）：由用户或应用程序执行的一个动作或一系列动作，这些动作访问或修改数据库的内容。

Transaction Processing Monitor（事务处理监视器，TPM）：控制数据在客户端和服务器间转换的程序，以便为联机事务处理（OLTP）提供一个一致的环境。

Transitive dependency（传递依赖）：假设A、B、C是表中的列，如果B依赖于A（A→B），并且C依赖于B（B→C），则C通过B传递而依赖于A（假设A不依赖于B或C）。如果在主键上存在一个传递依赖，则此表就不是3NF的。必须从表中去掉传递依赖以达到3NF的要求。

Tuple（元组）：关系中的一行记录。

Two-tier client-server architecture（两层客户端-服务器体系结构）：由处理主要业务和数据处理逻辑以及与用户的接口的客户端应用程序和管理和控制数据库访问的服务器程序组成。

Type hierarchy（类型层次结构）：一个实体以及它的子类和它们的超类，等等。

UML（Unified Modeling Language，统一建模语言）：在20世纪80年代和90年代引入的诸多面向对象分析与设计方法中的一种较新的方法。

Update anomalies（更新异常）：当用户试图更新一个包含冗余数据的表时可能引起的不一致。有三种类型的异常：插入、删除和更新。

User view（用户视图）：从特定的工作（比如经理或管理者）或业务应用领域（比如市场、职员或库存控制）角度定义的数据库应用的需求。

View（视图）：一个“虚拟的表”，它不实际存在在数据库中，但它由DBMS从视图所涉及的基本表中产生。

View integration approach（视图集成方法，用于数据库设计）：每个用户视图的需求，用来构建代表用户视图的独立数据模型。在数据库设计阶段，结果数据模型被合并成一个更大的模型。

Weak entity（弱实体）：一个其主键部分或完全依赖于某些其他实体的存在而存在的实体。

XML（eXtensible Markup Language，可扩展标记语言）：一个能够使设计者创建自己定制的标记的元语言（描述其他语言的语言），提供了HTML不具有的功能。

参 考 文 献

Chen P.P. (1976). The Entity-Relationship model – Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1), 9–36

Codd E.F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), 377–387

Connolly T.M. and Begg C.E. (1999) *Database Systems: A Practical Approach to Design, Implementation, and Management*, 2nd edn. Harlow, England: Addison- Wesley

OASIG (1996). Research report. Available at *http://www.comlab.ox.ac.uk/oucl/users/john.nicholls/oas-sum.html*

Shneiderman D. (1992). *Design the User Interface: Strategies for Effective Human-Computer Interaction*, 2nd edn. Reading, MA: Addison-Wesley