计 算 机 科 学 丛 书

PEARSON

数据库系统

数据库与数据仓库导论

内纳德·尤基克（Nenad Jukić）

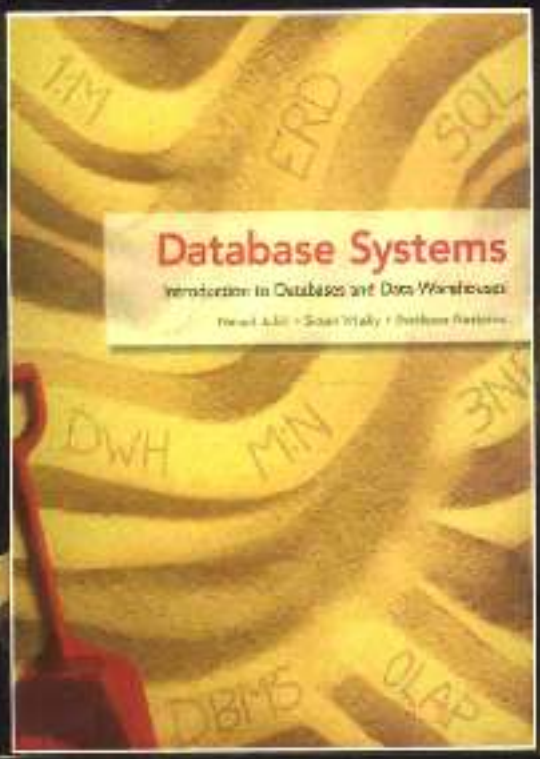
[美] 苏珊·维布斯基（Susan Vrbsky） 著

斯维特洛扎·奈斯特罗夫（Sveflozar Nestorov）

李川 刘 静 等译

Database Systems

Introduction to Databases and Data Warehouses



机械工业出版社
China Machine Press

数据库系统 数据库与数据仓库导论

Database Systems Introduction to Databases and Data Warehouses

本书是一本面向应用的入门级综合性数据库教材，通过直观易懂的方式讲解操作型数据库和分析型数据库。教材内容经过多年的课堂教学和就业实践，不断修订和改进，学生获益良多，用人单位好评如潮。

本书特点

- 内容全面，理论清晰。详细解析操作型数据库和分析型数据库的基本概念、设计方法和使用技巧，简要介绍数据挖掘、NoSQL数据库等高级主题，结合丰富的习题和案例帮助学生掌握基础知识。
- 面向实战，重视应用。针对当今快速变化的市场需求，培养学生设计和使用数据库的能力，重点是学会如何将理论知识成功地应用到信息系统、商业数据分析和决策支持等应用环境中，真正实现学以致用。
- 资源丰富，免费实用。访问 dbtextbook.com 获取免费资源：专门为本书开发的基于Web的数据建模套件ERDPlus，可创建ER图、关系模式和维度模型等；SQL脚本和数据集；DBMS软件使用指导；作者电子邮箱。

作者简介

内纳德·尤基克（Nenad Jukić）芝加哥洛约拉大学昆兰商学院教授，商务智能和数据仓库研究生课程中心负责人。主要从事信息技术领域的研究，为多家财富500强公司、美国政府和军事机构提供数据库方面的专业技术服务。

苏珊·维布斯基（Susan Vrbsky）阿拉巴马大学计算机科学研究生课程中心负责人。主要研究领域是数据库和云计算，包括数据密集型计算、实时数据库和绿色计算等，发表学术论文百余篇，自然科学基金获得者。

斯维特洛扎·奈斯特罗夫（Svetlozar Nestorov）现为芝加哥大学计算研究所高级研究助理，之前是芝加哥大学计算机科学系助理教授，为本科生和研究生讲授数据库和计算机系统课程。他参与创立的旅游搜索引擎Mobissimo被《时代》杂志评为50个最能网站之一。他拥有斯坦福大学本科、硕士和博士学位，博士论文师从Jeffrey Ullmam教授。

PEARSON

上架指导：计算机/数据库

ISBN 978-7-111-48698-5

9 787111 486985 >

定价：79.00元

PEARSON

www.pearson.com

投稿热线：(010) 88379604
客服热线：(010) 88378991 88361066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn

计 算 机 科 学 丛

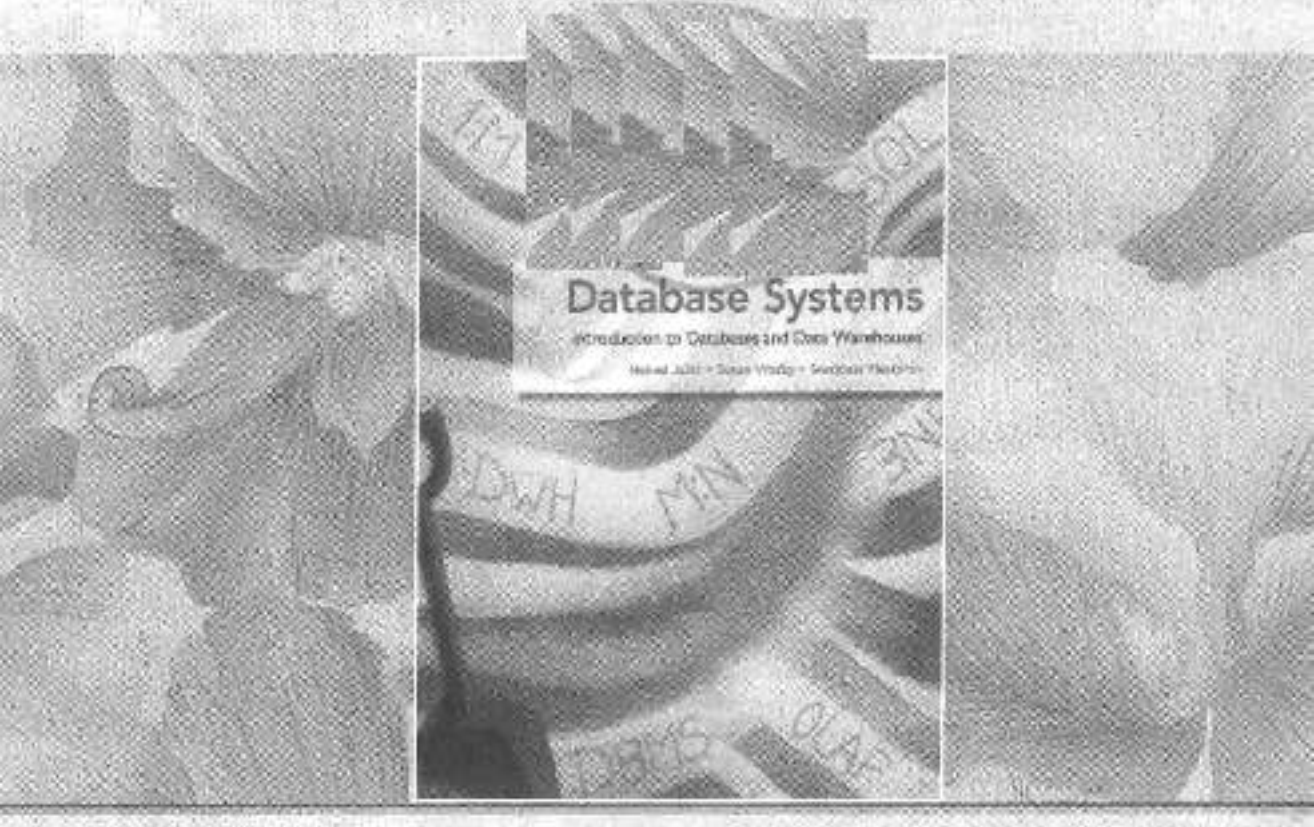
数据库系统

数据库与数据仓库导论

内纳德·尤基克（Nenad Jukić）
[美] 苏姗·维布斯基（Susan Vrbsky） 著
斯维特洛扎·奈斯特罗夫（Svetlozar Nestorov）
李川 刘 静 等译

Database Systems

Introduction to Databases and Data Warehouses



机械工业出版社

图书在版编目（CIP）数据

数据库系统：数据库、数据仓库导论 /（美）尤基奥（Jukic, N.），（美）维布斯基（Vrbsky, S.），（美）奈斯特罗夫（Nestorov, S.）著；李川等译. —北京：机械工业出版社，2015.3
（计算机科学丛书）
书名原文：Database Systems - Introduction to Databases and Data Warehouses

ISBN 978-7-111-48698-5

1. 数… Ⅱ. ①尤… ②维… ③奈… ④李… Ⅲ. 数据库系统 高等学校 教材 Ⅳ. TP311.13

中国版本图书馆 CIP 数据核字（2014）第 281468 号

本书版权登记号：图字：01-2013-3388

Authorized translation from the English language edition, entitled *Database Systems - Introduction to Databases and Data Warehouses*, 9780132575676 by Nenad Jukic, Susan Vrbsky, Svetlozar Nestorov, published by Pearson Education, Inc., Copyright © 2014.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Chinese simplified language edition published by Pearson Education Asia Ltd., and China Machine Press Copyright © 2015.

本书中文简体字版由 Pearson Education（培生教育出版集团）授权机械工业出版社在中华人民共和国境内（不包括中国台湾地区和中国香港、澳门特别行政区）独家出版发行。未经出版者书面许可，不得以任何方式抄袭、复制或节录本书中的任何部分。

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

本书是一部反映现代商务需要的IT环境，介绍操作型数据库和分析型数据库的基本概念、设计方法和使用技巧，兼具时效性、理论性和实用性。主要内容包括：数据库基础知识、数据库需求与ER建模、关系数据库模型、SQL、数据库的实现与使用、数据完整性概念、数据仓库与数据库设计比较、数据仓库的实现与使用、DBMS功能与数据库管理。书中包含丰富的实例、练习、作业和案例，配有数学图示和课程资源，帮助读者举一反三、学以致用。

本书适合作为高等院校计算机相关专业数据库原理与设计课程的教材，也可作为数据库技术人员的参考书。

出版发行：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：曲 熙　　责任校对：殷 虹
印　　刷：三河市宏图印务有限公司　　版　　次：2015年4月第1版第1次印刷
开　　本：185mm×260mm 1/16　　印　　张：23.75
书　　号：ISBN 978-7-111-48698-5　　定　　价：79.00元

凡购本书，如有缺页、倒页、脱页，由本社发行中心调换
客服热线：(010) 88379426 88361066　　购书热线：(010) 68326294 88379649 68995259　　投稿热线：(010) 88379604　　华章网站：www.hzbook.com

版权所有·侵权必究
未经许可不得翻印
本书法律顾问：北京大成律师事务所 韩光 / 邹晓东

文艺复兴以来，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举世性的优势；也正是这样的优势，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，还揭示了学术的演变，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和发展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到积极的推动作用，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公司较早意识到"出版要为教育服务"。自1998年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson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、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

"计算机科学丛书"的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专门为其书的中译本作序。迄今，"计算机科学丛书"已经出版了近两百个品种，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。其影印版"经典原版书库"作为姊妹篇也被越来越多实施双语教学的学校所采用。

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。随着计算机科学与技术专业学科建设的不断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将步入一个新的阶段，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。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：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：（010）88379604
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

华章教育
华章科技图书出版中心

译者序

Database Systems: Introduction to Databases and Data Warehouses

数据库技术诞生于20世纪60年代，现已发展成为一门内容丰富的学科，形成总产值达数百亿美元的产业。随着大数据时代的来临，数据库技术正以惊人的速度将现实世界中的信息转化为数据存储到各类计算机系统中，而且这一过程的发展态势可能超出人类的有限预想。其中蕴含的不仅是自然和生命，还有人类的行为、情感和历史。同我们生存其中的真实自然界一样，新兴的“数据自然界”中也潜藏着无尽的奥秘和巨大的财富，因此吸引着大批科学界、人文界及商界的学者和技术人员投身其中。正确解读和有效利用这些数据是新技术革命时代指引人类前行的灯塔。

本书的基本内容包括10个章节及10个附录。第1～6章主要介绍操作型数据库，第7～9章介绍分析型数据库、数据仓库与数据集市的相关内容，第10章从一个更高的视角给出数据库管理相关概述。本书由浅入深，由数据库的基本概念论及数据库的现状，最后对数据库技术的发展趋势进行展望，讲解过程中注重理论联系实际，令读者的学习和理解更为深刻和透彻。本书既可作为初学者的入门教程，也可供较有经验的用户借鉴，特别适合作为大学本科生数据库课程的参考读物。

本书的翻译工作是在极其紧张的条件下，经过所有团队成员的艰辛拼搏最终完成的，其中凝聚着每位参与者的真诚与责任。翻译工作由四川大学李川副教授统一负责，刘一静老师协助进行最终统稿，参与人员还有吴诗极、聂章艳、张永辉、李艳梅、谢世娜等。大家在节假日加班工作，对译文字雕句琢，最终有了本书的诞生。机械工业出版社的编辑在本书的出版过程中给予我们大力支持和理解，没有他们的奉献，也不可能有本书的面世。

尽管译者心正意诚，然则受限于自身水平，本书一定存在不少问题，还期望各位读者给予批评指正，各位的反馈将使本书更趋完善。最后，真诚期望本书对大家有益，这是对我们翻译工作的最大认可！

译者

2014年12月9日夜

于四川大学家中

本书是一本入门级的综合性数据库教材，适用于研究生或本科生的信息系统数据库课程。虽然选择本书作为教材的学生大多已经掌握了一定的信息系统基础知识，然而从本书的内容安排来看，它依然适用于那些没有预修相关基础课程的学生。每章中都包含必要的引言，之后再详细介绍数据库各方面的知识，旨在站在数据库专家的高度，为读者和用户讲解数据库知识。

操作型数据库和分析型数据库

当今的实用领域，如信息系统、商业数据分析及决策支持查询等，都同时需要操作型和分析型两种数据库系统功能。本书在介绍这两种类型的数据库时具时效性、理论性和实用性的特点。采用本书作为教材的学生，将学会如何设计和使用操作型和分析型数据库，并将所学内容运用到现今的商业环境。

操作型数据库和分析型数据库都是目前信息系统研究的主流，因此应采取的教学方法是：即使应用环境仅涉及其中一种类型，我们也将以一种有意义的方式同时讲授两种类型的数据库知识。根据我们的想法和经验，采用本书作为教材可以实现简单明了的教学。我们已基于这本教材进行了多年的数据库课程教学（包括本科生教学和研究生教学，涵盖学期授课和季度授课），每次都能实现预定的教学目标，并且好评如潮。此外，我们还收到大批已毕业或在读的本科生、研究生以及用人单位（包括实习和全职）的书面反馈，证明学习过这门课程的学生在面临有数据库相关技能需求的公司招聘时都有成竹。

本书特点

本书将所有关于设计和使用操作型、分析型数据库的基本内容分为 10 章及 10 个附录。

第 1 ~ 6 章主要介绍操作型数据库以及一些数据库基本问题，包括：数据库需求、ER 建模、关系建模、数据库约束、更新异常、规范化、SQL 语言、数据库应用、数据质量问题。

第 7 ~ 9 章主要介绍分析型数据库、数据仓库与数据集市的相关内容，包括：数据仓库概念、维度建模（星形模式）、数据仓库 / 数据集市建模方法、提取 / 转换 / 加载（ETL）处理、在线分析处理（OLAP）/ 商务智能（BI）功能、数据仓库 / 数据集市前端。

第 10 章从一个更高的视角（忽略细节）给出数据库管理相关概述。

附录（A、B、C、D、E、F、G、H、I、J）中是一些其他数据库相关主题的概述，包括：扩展 ER 建模（EER）、更高范式（高于第三范式）、企业资源计划（ERP）、数据管理和主数据管理、面向对象数据库、分布式数据库、并行数据库、云计算、数据挖掘、XML、NoSQL 数据库、大数据。

在本书的网站中，可以得到一个免费的基于 Web 的数据建模套件 ERDPlus（erdplus.com），该套件是结合本书内容设计和开发的。学生和教师可以利用该套件，特别是套件里针对学术研究的设置来创建 ER 图、关系模式及维度模型（星形模式）。我们鼓励教师和学生尝试该数据建模套件，体验其简洁性、易用性、轻便性及学术研究适用性。当然，也欢迎教师和学生使用其他工具来创建 ER 图、关系模型及维度模型。本书中所包含的练习可以用

ERDPlus 完成，也可以通过其他建模工具和方法（如 Visio、ERwin、ER/Studio、MS Word、MS Excel、MS PowerPoint、free drawing 等）完成。

配套资源

本书还有一些专为教师和学生准备的配套资源，内容如下。

配套网站（dbtexthook.com），包括：

- ERDPlus 的链接，它是一个数据建模软件（结合本书内容开发）。
- SQL 脚本及数据集。
- 关于免费得到 DBMS 和 OLAP/BI 软件的教程。
- 本书作者的邮箱。

教师资源网站（www.pearsonhighered.com/jukic）[1]，包括：

- PPT（快速版、经典版及完全版）。
- 教师手册，包括：
  - 各章最后的复习题、练习及小案例的答案。
  - 额外的练习题（配有答案）。
  - 教学大纲及教学计划。
  - 教学方案。
- 测试题文件，内容为涵盖各章节内容的测试问题集，问题形式包括多项选择题、判断题、简答题，每一道题目都有相应章节的索引及难度等级。

智能课堂电子教材

智能课堂电子教材（CourseSmart eTextbook）可以节省学生购买教材或商家推荐教材的开销。学生可以使用主流信用卡获得所需课程的全部讲授内容，课程教材可以通过书名、作者来查找。使用智能课堂电子教材，学生能够搜索特定的关键词或页码，在线作标记，打印带有课堂记录的读书作业，并能标记出重要章节以方便复习。想获取更多信息或购买智能课堂电子教材，请访问 www.coursesmart.com。

教学方法

本书集合了描述清晰的理论概念、简单易懂的实例，广泛且具有普适性的实用构件。对于学生需要掌握的每一项技能，每一章最后都提供了大量相应的练习和小案例。

在大多数章末都有标题为“问题说明”的部分，这部分内容可作为额外主题进行选择阅读或是作为相同主题的内容延伸（取决于课程难度及课程时间）。

下面给出各章教学安排的概述。

第 1 章　引言

主要内容：本章给出各章内容的快速概览；数据库相关的基本术语、概念及组件，如数据和信息、数据库管理系统（DBMS）、数据库系统开发步骤，以及操作型数据库与分析型数据库的对比。

---

[1] 关于本书教辅资源，用书教师可向培生教育出版集团北京代表处申请，电话：010-57355169/57355171，电子邮件：service.cn@pearson.com。——编辑注

教学方法：这一章采用了简短的描述和简明的实例，目的是简要地给出本书的总框架，为后面的章节做好铺垫。

第 2 章　数据库需求与 ER 建模

主要内容：实体－联系（ER）建模是将用户数据库需求形式化的一种概念化方法。本章将对其进行全面介绍。本章应用了陈氏 ER 标记的一种变形版本，当然，其他标记方法和概念数据建模方法也有涉及。本章强调 ER 建模的目的是收集并可视化用户需求。本章将介绍 ER 模型的各种组件：实体（包括弱实体）、属性（常规属性、单一属性、复合属性、多值属性、派生属性），以及一对一、一对多、多对多联系（二元或一元）。

附加内容：章末探讨了 ER 建模方面的一些更深入的问题（在相同实体之间的多实例多对多联系、联合实体，以及三元或更高元的联系）。

教学方法：本章内容基于实例，针对需求收集和 ER 模型创建提供了相应的综合案例。通过本章的练习、小案例以及免费软件（ERDPlus–ER 图功能）强化所介绍的概念。这一章的目的是让学生对 ER 建模相关的描述性概念以及需求可视化有深入了解，并通过大量实用练习来强化这些内容。

第 3 章　关系数据库建模

主要内容：本章全面介绍了关系数据库模型，包括关系概念、关系模式、完整性约束及用户自定义约束。此外，还讲解了将 ER 图（实体、属性以及一元和二元的一对一、一对多、多对多联系）映射为关系模式的过程。

附加内容：章末探讨了关于关系数据库建模的一些更深入的问题（映射关联实体、映射三元联系、设计者创建的主码和自动编号选项、ER 建模和关系建模的必要性）。

教学方法：这一章的教学基于实例说明 ER 图概念的映射过程及关系模式的创建过程。本章的练习、小案例及免费软件（ERDPlus–关系模式功能）强化了本章所介绍的内容。这一章的目的是让学生对关系数据库建模概念有深入了解，并且通过大量练习来强化这些内容。

第 4 章　更新操作、更新异常与规范化

主要内容：本章介绍了更新操作（插入、删除和修改）、规范化和更新异常（以此说明规范化的必要性），介绍和探讨了函数依赖的概念，同时还介绍了第一范式（1NF）、第二范式（2NF）及第三范式（3NF）（其他范式的介绍可以在附录 B 中找到）。

附加内容：章末探讨了有关规范化的更深入的问题（规范化例外、逆规范化、规范化与 ER 建模，为流数据库内容增添新表）。

教学方法：本章内容基于实例，说明更新操作、更新异常及规范化过程，并通过练习强化所介绍的概念。这一章的目的是让学生对更新和规范化概念有深入了解，并且通过大量练习来强化这些内容。

第 5 章　SQL

主要内容：本章全面介绍了 SQL（结构化查询语言），包括 SQL 创建、更新语句及关系型数据库查询，以及检索数据命令，如 SELECT 语句（伴随多条件，采用 AND、OR 和 NOT 类作符）、聚集函数（SUM、COUNT、AVG、MIN、MAX）、GROUP BY、ORDER

BY、HAVING、嵌套查询、UNION 和 INTERSECT 操作符、IN、EXISTS、各种连接、其他 SQL 语句和函数。

附加内容：章末探讨了有关 SQL 的更深入的问题（SQL 中观测值的不合理使用、SQL 标准，以及常见 RDBMS 工具包中的不同 SQL 语法）。

教学方法：本章内容基于实例，讲解如何用 SQL 语句实现关系数据库的建立、插入及查询，包含用 SQL 命令实现上述操作的具体实例。本书的 Web 主页（dbtextbook.com）提供了六种常见 DBMS 包（Oracle、MySQL、Microsoft SQL Server、PostgreSQL、Teradata、IBM DB2），其中包含了本章所有相关 SQL 语句的脚本。教师可以在本章授课中直接利用这些脚本在 RDBMS（根据教师选择）中复制、粘贴、执行 SQL 语句。通过这样的做法，教师可以向学生介绍 SQL 命令，同时还可以展示数据库的建立、插入和查询。数据集、练习、小案例可以强化本章内容。此外，本书 Web 主页介绍了怎样获得免费的、无访问限制的最新关系型 DBMS 软件。这一章的目的是让学生对 SQL 概念有深入了解，并且通过大量练习来强化这些内容。

第 6 章　数据库的实现与使用

主要内容：这一章讲述关于数据质量的问题——数据的准确性、完整性、一致性、唯一性、实时性及统一性。这些问题所涉及的数据都存储在数据库系统中。这一章还包括数据库前端接口（数据库形式、报表、应用），参照完整性选项（删除和更新选项：级联、限制、设置为空、设置为默认值）、索引，以及用户自定义约束的实现。

附加内容：章末探讨了断言和触发器。

教学方法：本章在教学实用上将介绍一些精简但有意义的关于数据库实现和使用方面的最基本问题，这些问题在第 5 章中并未提及。本章将通过大量的例子来讲解所提到的概念，并且通过大量实用练习来强化这些概念。

第 7 章　数据仓库概念

主要内容：本章给出了数据仓库和数据集市这两个术语的定义，并且介绍了数据仓库的基本构件和基本概念（资源系统、ETL（提取、转换、加载）、集成的分析数据仓库、面向主题的数据库、OLAP/BI 前端）。同时，本章还给出了关于数据仓库开发步骤的概述。

教学方法：本章采用一些简短的描述和简要的实例对数据仓库进行入门级讲解，并为后面两章的内容做好简洁扼要的铺垫。

第 8 章　数据仓库与数据集市建模

主要内容：本章介绍了维度建模——一种用于分析型数据库的概念和逻辑数据设计技术（如数据仓库和数据集市）。本章介绍的概念包括：维度表和事实表、星形模式、雪花模型、星座模型、缓慢变化维度。本章介绍了用于分析型数据库建模的 ER 建模技术（相对于第 2 章中提到的用于操作型数据库的 ER 建模技术）。本章还概述了不同数据仓库项目的开发方法：数据仓库总线结构（Kimball 方法）、一致维度、规范化数据仓库（Inmon 方法）、以及独立数据集市。

附加内容：章末将维度建模和 ER 建模这两种数据仓库 / 数据集市设计方法进行了比较。

教学方法：这一章的教学通过实例说明基于单数据源或多数据源的维度模型（星形模

式）、精细的和聚集的事实表、缓慢变化维度，以及其他维度建模内容。本章还列举了ER建模或规范化的数据仓库实例。练习、小案例、免费软件（ERDPlus—星形模式功能）强化了本章所介绍的概念。这一章的目的是让学生对数据仓库和数据集市建模概念有深入了解，并且通过大量实用练习来强化这些内容。

第9章　数据仓库的实现与使用

主要内容：本章给出ETL过程的概述，包括用于从操作型数据库中提取有用数据（以供分析使用）的相关基础设施和过程的建立，将这些数据进行格式变换以适应目标数据仓库模型的结构，通过数据清理和清洗来确保已变换数据的质量，以及将已变换和具有质量保证的数据加载到目标数据仓库的方法。本章定义了“在线分析处理”（OLAP）和“商务智能”（BI）这两个术语，它们通常与分析型数据库的前端应用有关。本章还介绍了OLAP/BI工具中的常用功能。

附加内容：章末讨论了用于OLAP/BI工具的不同数据库及不同的OLAP/BI架构。

教学方法：本章的教学基于实例说明所介绍的概念。配套的网站（dbtextbook.com）提供了如何免费且无限制地获取最新OLAP/BI软件的教程、数据集、练习题。本章简明扼要地讲解了有关数据仓库实现与使用方面的最基本问题。

第10章　DBMS功能与数据库管理概述

主要内容：本章给出了DBMS功能和组件的全面概述，以及数据库管理的相关问题，如数据安全、备份、恢复、性能及优化。

教学方法：本章对所介绍的内容进行了快速概述，让学生粗略地了解DBMS的功能和数据库管理的相关内容。

附录

主要内容：附录部分给出附加的数据库相关内容概述，包括扩展的ER建模（EER）、更高范式（高于第三范式）、企业资源计划（ERP）、数据管理与主数据管理、面向对象数据库、分布式数据库、并行数据库、云计算、数据挖掘、XML、NoSQL数据库及大数据。

教学方法：附录部分以简短的说明和实例来陈述内容，以此帮助学生粗略地了解一些数据库相关的附加内容。

我们要感谢各位审稿人在本书形成过程中提出的真知灼见，他们是：
Gary Baram，天普大学
Jeff Hassett，犹他大学
Emily Kelly，芝加哥洛约拉大学
Barry King，巴特勒大学
Mark Llewellyn，中佛罗里达大学
Brian Mennecke，艾奥瓦州立大学
Sree Nilakanta，艾奥瓦州立大学
Janet Renwick，阿肯色大学
John Russo，温特沃斯理工学院
Julian Scher，新泽西理工学院
Linda Zhou，马里兰大学

真诚地感谢他们在时间、精力、建议、专业知识方面给予的慷慨帮助。感谢行业专家 Benjamin Korallus（普华永道）、Robin Edison 和 Douglas Marcis（OptumInsight）、Stan Ozenbaugh 和 Kevin Little（天睿公司）、Mark Watson 和 Abhishek Sharma（堪萨斯城联邦储备银行）、Ryane Bohm（通用电气公司）、Alexander Rukhostkiy（波音公司）、Creighton Lang（NTT 数据公司）、Zachary Relli（博思艾伦咨询公司），以及 Gregory Roszczybiuk（李奥贝纳广告公司）。还要感谢我们的同事 Stefan Lessmann（汉堡大学）、Mary Malliaris 与 Faruk Guder（芝加哥洛约拉大学）、Michael Goul（亚利桑那州立大学）、Hugh Waston（佐治亚大学）、Boris Jukić（克拉克森大学），以及 Miguel Velasco（明尼苏达大学）。同样感谢芝加哥洛约拉大学的学生：Melanie Ruiz、Jordan Braun、Subash Pant、Sarah Shea、Adam Chorazy、Anton Dokov 和 Karan Desai。

特别感谢世界级的数据库管理员 / 数据库设计者 / 程序设计者 Zoran Avramović，他对本书的仔细审阅和真率见解促使我们坚持不懈地工作。同时特别感谢 Ivan Bare，他在本书的准备阶段提供了大量的后勤保障及技术支持。

在此，Nenad 还想对他有幸遇到的信息技术领域的优秀教师表达谢意，他们是：Zoran Vlašić 教授、Damir Kalpić 教授、Allen Parrish 教授及 Susan Vrbsky 教授。

在撰写本书的过程中，我们得到了 Paul Gray 博士友好、中肯的建议及鼓励，他已于 2012 年去世。Paul 是信息系统教育与研究领域的创始先驱之一，在此对他表示深切的怀念。

Nenad Jukić
Susan Vrbsky
Svetlozar Nestorov

Nenad Jukić：信息系统领域教授，芝加哥洛约拉大学昆兰商学院商务智能与数据仓库研究生课程中心负责人。

Jukić 博士从 1999 年起在昆兰商学院信息系统与运营管理系从事本科生、研究生和行政教育教学工作。2005 ~ 2007 年，Jukić 博士还是北京国际 MBA 项目位于北京大学的中国经济研究中心信息系统领域的访问教授。1997 ~ 1999 年，他在位于密歇根州艾伦代尔的伯格州立大学计算与信息系统学院任教。

Jukić 博士在克罗地亚萨格勒布大学计算与电气工程学院获得计算科学与电气工程学士学位，在位于美国阿拉巴马州塔斯卡卢萨的阿拉巴马大学获得硕士和博士学位。

Jukić 博士积极开展各种信息技术研究工作，涉及的领域包括：数据库建模与管理，数据仓库、商务智能、数据挖掘、电子商务和 IT 战略。他的研究成果公开发表于众多管理信息系统及计算机科学领域的学术期刊、会议出版刊物和书籍上。除学术工作之外，他还为多家公司和机构的数据库、数据仓库、商务智能项目担任技术顾问，涵盖创业公司、财富 500 强公司、美国政府部门及军事机构。

Suan V. Vrbsky：副教授，阿拉巴马大学计算机科学研究生课程中心负责人。

Vrbsky 博士从 1992 年起在阿拉巴马大学计算机科学系从事本科生及研究生课程教学工作，现任阿拉巴马大学云计算与集群计算实验室主任。1983 ~ 1986 年，她同时在南伊利诺伊大学计算机科学系任教。

Vrbsky 博士在美国西北大学获得学士学位，在南伊利诺伊大学卡本代尔分校获得计算机科学硕士学位，在伊利诺伊大学香槟分校获得计算机科学博士学位。

Vrbsky 博士的研究领域是数据库及云计算，涵盖数据密集型计算、实时数据库、数据库安全、移动数据库及绿色计算。她在计算机科学学术期刊、会议出版刊物和书籍上合作发表了 100 余篇文章，并且是自然科学基金获得者。

Svetlozar Evtimov Nestorov：芝加哥大学计算研究所高级研究助理。

Nestorov 博士曾任芝加哥大学计算机科学系助理教授，在此期间，他从事本科生和研究生数据库与计算机系统课程的教学工作。之后，他参与创立了 Mobissimo，这是一个得到风险投资的旅游搜索引擎，在 2004 年被《时代》杂志评选为 50 个最酷网站之一。

Nestorov 博士在斯坦福大学获得计算机科学与数学学士学位及计算机科学硕士学位，凭借“结构化和半结构化数据的数据挖掘技术”这篇论文获得该校的计算机科学博士学位。他的导师是 Jeffrey Ullman 教授。

Svetlozar 负责尼尔森公司数据中心数据仓库项目的设计和开发，这也是芝加哥大学布斯商学院的一部分。他感兴趣的研究领域还包括数据挖掘、高性能计算和 Web 技术。

推荐阅读

数据挖掘：概念与技术（第3版）

数据挖掘与R语言

R语言与数据挖掘最佳实践和经典案例

数据仓库与数据挖掘

推荐阅读

数据集成原理

作者：Anhai Doan 等 译者：孟小峰 等 中文版：978-7-111-47166-0 定价：85.00元

数据集成的第一部综合指南，从理论原则到实现细节，
再到语义网和云计算目前所面临的新挑战

这是一本数据集成技术的权威之作。书中的大部分技术都是作者提出来的。本书内容全面，很多技术细节都介绍得非常清楚，是数据集成相关工作人员的必读书籍。

——Philip A. Bernstein，微软杰出科学家

本书的三位作者对数据集成领域都有重要贡献，既有学术背景，又有工业界的经历。书中包含很多例子和相关信息，以便于读者理解理论知识。本书包含了现代数据集成技术的很多方面，包括不同的集成方式、数据和模式匹配、资源处理和包装器，还包括Web以及多种数据类型和数据格式带来的挑战。本书非常适合作为研究生数据集成课程教材。

——Michael Carey，加州大学欧文分校信息与计算机科学Prin教授

数据库系统概念（第6版）

作者：Abraham Silberschatz 等 译者：杨冬青 等 中文版：978-7-111-37529-6 定价：99.00元
中文精编版：978-7-111-40085-1 定价：59.00元
英文精编版：978-7-111-40086-8，定价：69.00元

数据库领域的殿堂级作品
夯实数据库理论基础，增强数据库技术内功的必备之选
对深入理解数据库，深入研究数据库，深入操作数据库都具有极强的指导作用！

本书是数据库系统方面的经典教材之一。其内容由浅入深，既包含数据库系统基本概念，又反映数据库技术的进展。已被国际上许多著名大学所采用，包括斯坦福大学、耶鲁大学、马克萨斯大学、康奈尔大学、伊利诺伊大学等。数据也有多所大学采用本书作为本科生和研究生数据库课程的教材和主要教学参考书，收到了良好的效果。

目　录

Database Systems: Introduction to Databases and Data Warehouses

出版者的话

译者序

前言

致谢

作者简介

第1章　引言……1

1.1　基本术语……1

1.2　数据库系统开发步骤……4

1.2.1　数据库需求收集、定义和可视化……4

1.2.2　数据库建模……5

1.2.3　数据库实现……6

1.2.4　开发前端应用程序……6

1.2.5　数据库部署……6

1.2.6　数据库使用……6

1.2.7　数据库管理和维护……7

1.3　数据库新版本……7

1.4　数据库使用范围……7

1.5　参与数据库系统的人员……7

1.5.1　数据库分析者、设计者和开发者……8

1.5.2　前端应用程序分析者和开发者……8

1.5.3　数据库管理员……8

1.5.4　数据库终端用户……8

1.6　操作型数据库与分析型数据库……9

1.7　关系数据库管理系统……9

1.8　本书内容安排……9

关键术语……10

复习题……10

第一部分　操作型数据库

第2章　数据库需求与ER建模……14

2.1　引言……14

2.2　ER建模基本构件……14

2.3　实体……14

2.4　属性（唯一和非唯一）……15

2.5　联系……15

2.6　联系类型（最大基数和）……17

2.7　联系和联系实例……18

2.8　联系属性……19

2.9　实例：数据库需求集及其ER图……20

2.10　复合属性……22

2.11　复合的唯一属性……23

2.12　多个唯一属性（候选码）……23

2.13　多值属性……24

2.14　派生属性……24

2.15　可选属性……25

2.16　实例：包含多种类型属性的实体……26

2.17　联系中最小基数和最大基数精切伯……26

2.18　一元联系和联系的角色……27

2.19　相同实体间的多种联系……28

2.20　弱实体……29

2.21　实体、属性和联系的命名约定……31

2.22　多个ER图……32

2.23　实例：另一组数据库需求及其ER图……33

2.24　数据库需求和ER模型的使用……34

2.25　各种ER符号体系……36

2.26　扩展的ER模型……37

2.27　问题说明：相同实体之间具有多个实例的M：N联系……38

2.28 问题说明：关联实体……41
2.29 问题说明：三元（及更高阶）联系……42
总结……46
关键术语……48
复习题……49
练习……49
小案例……50
第3章 关系数据库建模……54
3.1 引言……54
3.2 关系数据库模型基本概念……54
3.3 主码……56
3.4 将实体映射为关系……56
3.5 将具有复合属性的实体映射为关系……57
3.6 将具有唯一复合属性的实体映射为关系……58
3.7 将具有可选属性的实体映射为关系……59
3.8 实体完整性约束……59
3.9 外码……60
3.10 将联系映射为关系数据库组件……60
3.10.1 1：M联系的映射……61
3.10.2 M：N联系的映射……63
3.10.3 1：1联系的映射……65
3.11 参照完整性约束……66
3.12 实例：将ER图映射为关系模式……67
3.13 将拥有若干候选码（多个唯一属性）的实体映射为关系……69
3.14 将具有多值属性的实体映射为关系数据库组件……70
3.15 将具有派生属性的实体映射为关系……71
3.16 实例：将具有多种类型属性的实体映射为关系模式……71
3.17 一元联系的映射……72
3.17.1 1：M一元联系的映射……72
3.17.2 M：N一元联系的映射……73
3.17.3 1：1一元联系的映射……74
3.18 相同实体间的多个联系的映射……74
3.19 弱实体的映射……75
3.20 实例：将另一个ER图映射为关系模式……77
3.21 关系数据库约束……79
3.21.1 隐含约束……80
3.21.2 用户自定义约束……80
3.22 问题说明：关联实体映射……83
3.23 问题说明：三元联系映射……83
3.24 问题说明：设计者创建的主码和自动编号选项……85
3.25 问题说明：ER建模和关系建模……86
总结……87
关键术语……87
复习题……88
练习……88
小案例……89
第4章 更新操作、更新异常与规范化……91
4.1 引言……91
4.2 更新操作……91
4.2.1 插入操作实例……91
4.2.2 删除操作实例……91
4.2.3 修改操作实例……92
4.2.4 关于更新操作术语的说明……92
4.3 更新异常……92
4.3.1 实例场景……92
4.3.2 实例关系（包含冗余数据）……93
4.3.3 插入异常……95
4.3.4 删除异常……95
4.3.5 修改异常……95
4.4 函数依赖……95
4.5 函数依赖实例……96
4.6 简化函数依赖……98
4.6.1 增广函数依赖……98
4.6.2 等价函数依赖……99
4.7 函数依赖类型……100
4.7.1 部分函数依赖……100
4.7.2 完全函数依赖……100
4.7.3 传递函数依赖……101
4.8 另一个函数依赖实例……101

4.9 规范化……103
4.9.1 第一范式……103
4.9.2 第二范式……105
4.9.3 第三范式……106
4.9.4 其他范式……108
4.9.5 消除冗余和更新异常……108
4.10 另一个规范化实例……110
4.11 问题说明：规范化例外情况……112
4.12 问题说明：逆规范化的规范化与性能……113
4.13 问题说明：ER 建模和规范化……114
4.14 问题说明：用于简化数据库内容的设计者添加的实体（表）和码……115
关键术语……117
复习题……118
练习……118
第 5 章 SQL……123
5.1 引言……123
5.2 SQL 命令综述……123
5.2.1 数据定义语言……123
5.2.2 数据操纵语言……124
5.2.3 数据控制语言和事务控制语言……124
5.3 SQL 数据类型……124
5.4 SQL 语法简要说明……124
5.5 CREATE TABLE……125
5.6 DROP TABLE……127
5.7 INSERT INTO……128
5.8 SELECT……130
5.9 WHERE……132
5.10 DISTINCT……133
5.11 ORDER BY……134
5.12 LIKE……135
5.13 聚集函数……135
5.14 GROUP BY……136
5.15 HAVING……139
5.16 嵌套查询……141
5.17 IN……142
5.18 JOIN……143
5.19 别名……145
5.20 多关系连接……146
5.21 ALTER TABLE……147
5.22 UPDATE……148
5.23 DELETE……148
5.24 CREATE VIEW 和 DROP VIEW……149
5.25 集合运算：UNION、INTERSECT、EXCEPT(MINUS)……150
5.26 使用其他 SQL 命令的实例……151
5.27 CREATE TABLE（附加实例）……152
5.28 INSERT INTO（附加实例）……154
5.29 约束管理……157
5.30 SELECT（附加实例）……158
5.31 关系与自身的连接（自连接）……158
5.32 OUTER JOIN……158
5.33 无主码 / 外码组合的连接……160
5.34 IS NULL……160
5.35 EXISTS……160
5.36 NOT……161
5.37 从查询中插入关系……161
5.38 其他 SQL 功能……162
5.39 问题说明：SQL 中观测值使用不当……162
5.40 问题说明：SQL 标准和 SQL 语法差异……163
5.40.1 SQL 语法差异 1：DATE 和 TIME 数据类型……163
5.40.2 SQL 语法差异 2：FOREIGN KEY……163
5.40.3 SQL 语法差异 3：别名关键词 AS 的使用……164
5.40.4 SQL 语法差异 4：ALTER TABLE……164
5.40.5 SQL 语法差异 5：集合运算……165
5.40.6 SQL 语法差异 6：FULL OUTER JOIN……166
5.40.7 SQL 语法差异 7：约束管理……166

5.40.8 SQL 语法差异8：GROUP BY ……167
关键术语 ……167
复习题 ……168
练习 ……169
小案例 ……170
第6章 数据库的实现与使用 ……172
6.1 引言 ……172
6.2 参照完整性约束：实现删除和更新操作 ……172
6.2.1 删除选项 ……173
6.2.2 更新选项 ……176
6.2.3 实现删除和更新选项 ……178
6.3 实现用户自定义约束 ……179
6.3.1 CHECK子句 ……180
6.3.2 实现用户自定义约束的其他机制 ……181
6.4 索引 ……181
6.5 数据库前端 ……186
6.6 数据质量问题 ……190
6.7 问题说明：断言和触发器 ……194
关键术语 ……196
复习题 ……196
练习 ……197

第二部分 分析型数据库

第7章 数据仓库概念 ……202
7.1 引言 ……202
7.2 操作型信息与分析型信息 ……202
7.2.1 数据组成差别 ……203
7.2.2 技术差别 ……204
7.2.3 功能差别 ……204
7.3 数据仓库定义 ……207
7.3.1 结构化数据存储 ……207
7.3.2 集成性 ……207
7.3.3 面向主题 ……207
7.3.4 企业范围 ……207
7.3.5 历史性 ……207
7.3.6 时变性 ……208
7.3.7 分析型信息的检索 ……208
7.3.8 细节数据和汇总数据 ……208
7.4 数据仓库组件 ……208
7.4.1 源系统 ……209
7.4.2 数据仓库 ……209
7.4.3 ETL ……209
7.4.4 数据仓库前端（BI）应用 ……210
7.5 数据集市 ……210
7.6 数据仓库开发步骤 ……210
7.6.1 需求收集、定义与可视化 ……211
7.6.2 数据仓库建模 ……212
7.6.3 创建数据仓库 ……213
7.6.4 创建ETL架构 ……213
7.6.5 开发前端（BI）应用 ……213
7.6.6 数据仓库部署 ……213
7.6.7 数据仓库使用 ……214
7.6.8 数据仓库管理与维护 ……214
7.7 数据仓库的新版本 ……214
关键术语 ……215
复习题 ……215
第8章 数据仓库与数据集市建模 ……216
8.1 引言 ……216
8.2 维度建模基本概念 ……216
8.3 初始实例：基于单个数据源的维度模型 ……217
8.4 维度特性、事实特性及初始实例分析 ……220
8.5 扩展实例：基于多个数据源的维度模型 ……222
8.6 其他可能的事实属性 ……225
8.7 事实表中的事务标识码 ……225
8.8 事实表中的事务时间 ……227
8.9 一个维度模型中的多个事实表 ……230
8.10 细节事实表与聚集事实表 ……232
8.10.1 细节事实表 ……233
8.10.2 聚集事实表 ……233
8.10.3 细节事实表与聚集事实表的其他实例 ……237
8.11 事实表的粒度 ……238
8.12 缓慢变化维度与时间戳 ……239

8.12.1 Type1 方法……240
8.12.2 Type2 方法……240
8.12.3 Type3 方法……241
8.13 其他维度建模问题……242
8.13.1 雪花模型……242
8.13.2 立方体……243
8.14 数据仓库（数据集市）建模方法……243
8.15 规范化数据仓库……243
8.16 规范化数据仓库实例……244
8.17 维度建模数据仓库……247
8.18 维度建模数据仓库实例……248
8.19 独立数据集市……250
8.20 问题说明：维度建模与 ER 建模作为数据仓库 / 数据集市设计技术的比较……251
关键术语……253
复习题……253
练习……254
小案例……260
第 9 章 数据仓库的实现与使用……261
9.1 引言……261
9.2 创建数据仓库……261
9.3 ETL：提取，转换，加载……263
9.4 在线分析处理……268
9.5 OLAP/BI 工具……269
9.6 OLAP/BI 工具功能……269
9.6.1 切片和切块……271
9.6.2 旋转……272
9.6.3 下钻和上卷……273
9.6.4 OLAP/BI 工具附加功能概述……274
9.7 OLAP/BI 工具用途……275
9.8 数据仓库 / 数据集市前端（BI）应用……275
9.9 管理仪表板……278
9.10 数据仓库部署……278
9.11 问题说明：OLAP/BI 工具数据库模型……279
9.12 问题说明：OLAP/BI 工具数据架构方法……280
9.12.1 MOLAP……280
9.12.2 ROLAP……281
9.12.3 HOLAP……282
关键术语……282
复习题……282
练习……283

第三部分 其他主题

第 10 章 DBMS 功能与数据库管理概述……288
10.1 引言……288
10.2 DBMS 组件……288
10.3 数据库管理概述……289
10.4 数据库系统监测与维护……289
10.5 数据库安全：防范非法存取……291
10.6 数据库备份与恢复……292
10.7 数据完整性保护……293
10.8 数据库性能优化……294
10.9 数据库政策与标准的开发与实施……294
关键术语……295
复习题……295

附 录

附录 A 扩展的 ER……298
附录 B 关于规范化及更高范式……304
附录 C 企业资源计划……309
附录 D 数据管理与主数据管理……311
附录 E 面向对象数据库……314
附录 F 分布式数据库、并行数据库与云计算……319
附录 G 数据挖掘……325
附录 H XML……328
附录 I NoSQL 数据库……335
附录 J 大数据……338
术语表……342
索引……354

引　言

1.1　基本术语

现今人们每天都在同数据库打交道，比如购物、浏览网页或者银行交易。数据库对于现代企业和机构的正常运转至关重要。本书将讨论数据库与数据库系统的发展和使用中一些最根本的问题。下面将首先介绍与数据库相关的基本术语。

数据（data）是指已记录或者可获取的事实。除了文本和数字，数据也可以以其他形式存在，比如图标、图形、图像、声音和视频等。无论形式如何，数据都是由于能被用户利用才被记录和保存起来的。

信息（information）是指用户以某种目的获取的数据。通常来说，是从进行某项活动所需要的数据集合中获得有用的信息，活动可以是以某种形式进行的数据搜索、数据处理、数据操纵等。

为了阐明术语“数据”和“信息”，让我们来看一个电话本的实例。一个电话本是一组收集到的数据——如个人、餐馆、银行等的电话号码。在特定情况下，这些电话号码就会成为信息。例如，一个拥有电话本的人饿了，决定给离家最近的披萨店（Pizza Adria）打电话叫外卖。此时，Pizza Adria 的电话对他来说就变成了信息，而从电话本中搜寻 Pizza Adria 店的电话号码则是从数据中获取信息的必要步骤。

来看看另一个例子：我们假设一个大型零售店的经理想要了解当前一个季度的服装销售情况。首先，假设这个零售店里的每一笔交易都记录在计算机文件里，经理可以直接获取。然而，要在如此巨大的文件里获取有关服装销售业绩的信息，并检查每个客户记录，这对于经理来说不是一件容易的事情。因而，数据必须按照一定的方式来处理才能较直观地呈现给经理。比如，当前季度的服装销售数据可以累加起来。这个累加结果可以跟前一季度的同种商品销售总量进行比较，或者跟去年同季度不同商品的销售总量进行比较。这些操作都能实现从收集的数据中获取信息。

“数据”和“信息”这两个术语经常被彼此替换并作为同义词使用。这种现象普遍存在。正如我们前面所说，信息是我们需要的数据。如果组织和保存数据的目的是为了满足用户的需要，那么这样的数据就是信息[1]。本书频繁地将“数据”和“信息”作为两个可以互换的同义词使用。

0001	B	2	11:01
0001	F	3	11:01
0002	S	2	11:02
0002	B	1	11:02
0003	F	2	11:03

图 1-1　没有元数据的数据

元数据（metadata）是描述数据结构和数据属性的数据。元数据是正确理解和使用数据的关键。如图 1-1 所示，如果不清楚

[1] 另一方面存在这样的情形，存储的数据中无用数据比例过大，导致可以忽略有实际用途的数据。这也就是人们常说的“数据量过多而信息量过少”。

元数据，就很难理解图中的内容，数字列和数据文本也将显得没有意义。

图 1-2 再次给出图 1-1 中的数据并附上了元数据描述，现在图中数据所表示的意义就清楚了。根据元数据可知，图中的每行代表 2013 年 9 月 1 日餐饮连锁店 Burger Prince 旗下 101 号商店的一条商品交易记录，其中，第一列为交易标识符，第二列为交易中所购商品，第三列为商品购买数量，最后一列则为交易时间。

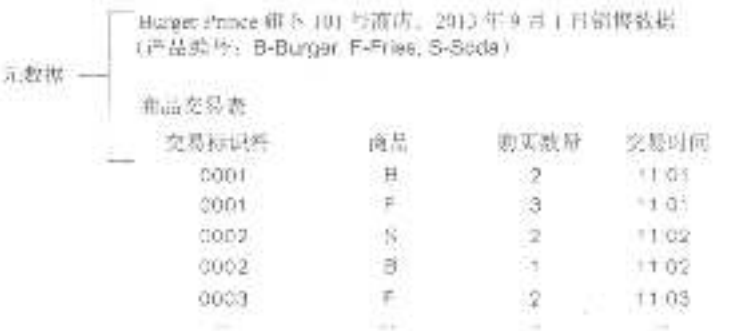


图 1-2 有元数据的数据

数据库（database）是指在计算媒介上按结构存储的相关数据。数据库的目的是将数据组织起来，以便用户直接从数据中获得信息。数据库的结构由数据库的元数据解释。**数据库元数据**（database metadata）常常被定义成关于数据的数据或者数据库的内容而不是数据本身。数据库元数据包含以下信息：

- 数据结构名称（如表名、列名）。
- 数据类型（如 Product- 字符型，ItemsSold- 整数型）。
- 数据描述（如 ItemsSold 代表商品销售数量）。
- 其他描述数据库所存数据特征的信息。

数据库管理系统（DBMS）是用来完成以下功能的软件：

- 数据库创建。
- 插入、存储、检索、更新以及删除数据库中的数据。
- 数据库维护。

打个比方，DBMS 和数据库之间的关系就像演示软件（比如微软的 PowerPoint）和演示之间的关系。演示软件是用来创建演示、插入演示内容、执行演示以及改变或者删除演示内容的。

相似地，DBMS 是用来创建数据库、在数据库中插入数据、从数据库中检索数据以及在数据库中修改数据的。

数据库系统（database system）基于计算机系统，其目的是在用户和数据库信息之间保证高效的交互，典型框架如图 1-3 所示。

数据库系统[⊖]的三个主要成分是数据库、DBMS 和**前端应用程序**（front-end application）。数据库是数据库系统的核心，而数据库的所有交互操作，则是通过 DBMS 来完成的。前端应用程序是为用户和 DBMS 之间更易于交互而提供的一种机制。下面请看一个用户通过前

⊖ 描述图 1-3 所示系统结构的另一个常用术语是“信息系统”，“信息系统”和“数据库系统”两个术语通常有相同的含义，但常用场合不同。由于本书的重点是数据库，因而本书使用“数据库系统”。

端应用程序与 DBMS 进行交互的例子。

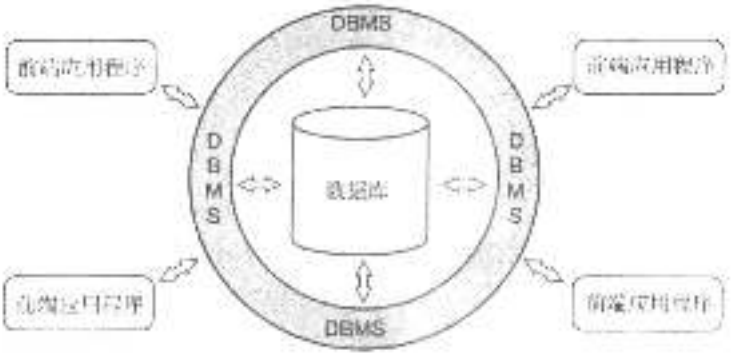


图 1-3 典型的数据库系统结构

ATM 屏幕显示了用户的操作界面，如"从支票账户取款"或"从储蓄账户取款"，这就是一个前端应用程序的例子。屏幕上的操作一旦被选择，就会触发前端应用程序与 DBMS 的通信，紧接着 DBMS 与数据库的通信也被触发。例如，通过 ATM 屏幕上的操作选择，用户可以请求从支票账户取款 20 美元。此时前端应用程序将发送几条命令给 DBMS 来响应用户的请求。一条命令告诉 DBMS 将查数据库中用户的支票账户是否有足够资金供用户取走，如果资金充足，ATM 将为用户出钞。同时，另一条命令告诉 DBMS 将用户支票账户的余额减少 20 美元。

人们可使用数据库系统来辅助处理与工作或者生活相关的事务，这样的人常常称为**终端用户 / 业务用户** (end user/business user) 术语。"终端用户"将系统中数据的实际使用人员与在数据库实现和维护阶段进行测试的技术人员区别开来。

终端用户与数据库之间通过前端应用程序进行的交互操作类型称为**间接交互**（indirect interaction）。另一种终端用户与数据库之间的交互操作类型称为**直接交互**（direct interaction），即终端用户直接与 DBMS 进行通信。图 1-4 展示了直接交互与间接交互。

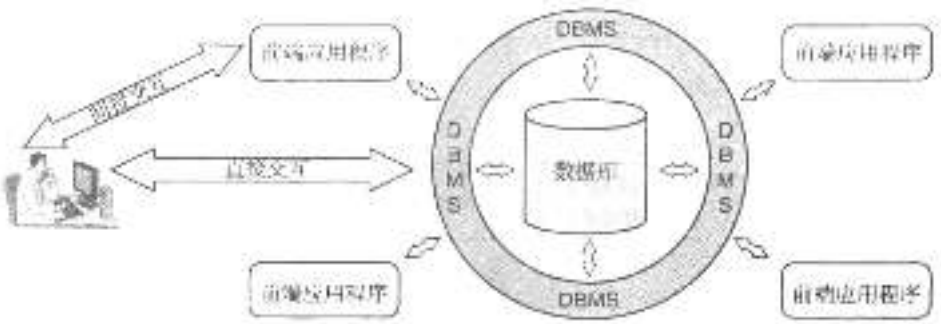


图 1-4 终端用户与数据库系统之间的直接交互和间接交互

间接交互操作仅需要终端用户拥有很少、甚至毫无数据库相关技能，而进行直接交互操作的终端用户则需要先接受数据库相关的知识训练，并达到一定的实际操作水平和知识层次。直接交互操作要求终端用户懂得如何使用特定的 DBMS 命令，这就要求终端用户会使用 DBMS 语言。

数据库和 DBMS 是数据库系统不可缺少的组成成分。比如，在人们有权限获取数据库

中的数据，且有能力，有时间进行直接交互操作的情况下，前端应用程序将失去必要性。此时，没有前端应用程序，数据库系统仍然可以使用。然而，在大多数情况下，日常生活和工作中的数据库系统往往拥有前端应用程序，而终端用户与数据库之间的交互大多也是通过前端应用程序来实现间接交互操作。

1.2 数据库系统开发步骤

一旦数据库项目启动，那么，目标，预算等前期开发活动就已经开始实施。这些活动在数据库系统的实际开发过程之前进行。下面将给出开发数据库系统的实际活动流程，如图 1-5 所示。

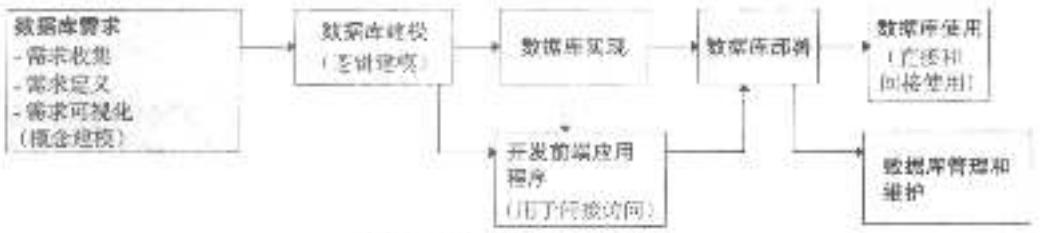


图 1-5 数据库系统开发步骤

接下来，我们将对图 1-5 中的每一个步骤做出简要的介绍和讨论。

1.2.1 数据库需求收集、定义和可视化

数据库系统开发中，第一步也是最为关键的步骤，就是数据库需求收集、定义和可视化（requirements collection ,definition, visualization）。如果这个步骤是成功的，接下来的步骤在很大程度上也会成功。而一旦该步骤产生了错误，剩下的所有步骤乃至整个项目都将功亏一篑。图 1-5 和图 1-6 对该步骤进行了突出显示以强调其重要性。

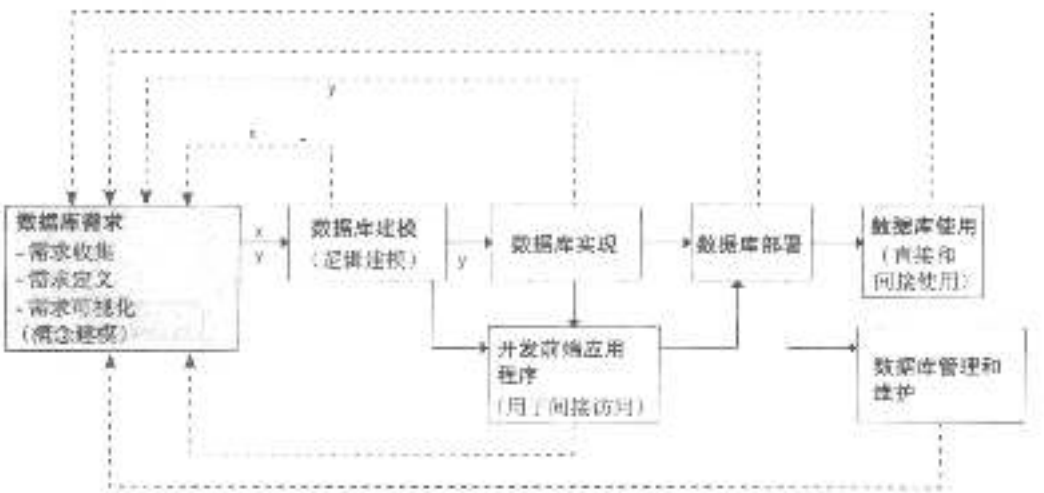


图 1-6 数据库需求收集、定义和可视化过程迭代的特点

此步骤得到终端用户的具体需求，包括数据库系统将要保存的数据类型、保存方式、数据库系统的容量与功能。这些需求将用于数据库建模、实现以及为数据库创建前端应用程序

的过程中。

收集好的需求应该在文档里进行清晰的定义和表述，然后利用概念数据建模技术将其可视化成**概念数据库模型**（conceptual data base model），即实体－联系（ER）建模[⊖]。"概念数据建模"是需求可视化的另一个术语。

数据库需求阶段的步骤和方法应该是个循环迭代的过程。刚开始收集、定义及可视化得到一个比较小的需求集合，进而数据库开发人员与系统的潜在终端用户针对这个集合进行讨论。这些讨论可以形成另一个需求的收集、定义及可视化的循环过程，进而逐渐扩大最初的需求集合。

如图1-6所示，即使需求集合在数据库需求的收集、定义和可视化步骤之中已经确定，它依然可能被数据库开发过程中的其他步骤改变。

将收集、定义以及可视化数据库中所有的需求作为一个独立步骤，然后再去完成数据库系统开发过程中的其他所有步骤，这种方法是不可取的。常见的做法是允许在其后数据库系统开发过程的每个步骤中完善和添加需求。如图1-6中的虚线所示。

例如，数据库工程中一个通常的经验做法是将收集、定义以及可视化作为起始需求的一部分，基于这些需求创建和实现数据库模型的初期部分。随后是一系列类似的迭代过程（如图1-6中字母x标记的部分），添加额外需求（收集、定义和可视化）并用其进行数据库模型的扩展。

其他步骤同样可以通过迭代完善需求集合，比如在创建前端应用程序或者使用数据库的实际过程中，可以根据需要修改、增加或减少初始的需求集合。

每次初始的需求集合被改变，概念模型就要做相应的改变，需求的改变会一直扩展到全部后续步骤中：数据库的建模与创建、前端应用程序的创建、数据库的开发、使用、管理和维护。

数据库中任何开发步骤都不允许隐式地改变需求。即在数据库实现过程中（图1-6的中间部分），不允许开发人员临时创建新的不被需求包含的数据库构件（例如，数据库的表或表中的列）。如果在数据库实现过程中发现需要一个有用的新构件时，正确的做法是（图1-6中字母y标识的部分）返回到需求部分，同时扩展需求文档和概念模型，将新构件添加到需求中。该新需求还应该进一步在扩展后的数据库模型中得到反映。这样一个新的数据库构件才能被添加。

数据库需求的可视化被公认为是数据库系统开发过程中最关键的一步。这一步骤的好坏将决定整个数据库项目的成败。如果这一步骤有误，需求将会偏离目标，数据库的最终结果也就不能满足终端用户的需求。

1.2.2 数据库建模

在收集、定义和可视化需求之后的第一个步骤是**数据库建模/逻辑数据库建模**（database modeling /logical database modeling）。在本书中，我们使用术语"数据库建模"来指代可由DBMS软件执行的数据库模型创建过程。数据库模型也指常说的**逻辑数据库模型/实施型数据库模型**（logical database model /implementational database model），而不是概念数据库模型。概念数据库模型仅仅是需求的可视化，并且与特定的DBMS无关。因此，一个数据库

⊖ 实体－联系（ER）建模在下一章中讲解。

有两个模型：一个是在需求收集、定义和可视化步骤中为需求可视化创建的概念数据模型；另一个是实施数据库模型，也称逻辑模型，它在数据库建模步骤中创建，以便在利用DBMS进行数据库实施的后续步骤中使用。概念数据模型是实施（逻辑）数据库模型的蓝本。

大多数的现代数据库中，数据库建模（如逻辑数据建模）涉及所谓的关系数据库模型的创建[1]。ER模型使用概念模型技术将需求可视化，而关系数据库建模则是将ER模型直接映射到关系模型的过程。

1.2.3 数据库实现

数据库模型创建完成后，接下来的步骤便是**数据库实现**（database implementation）。这个步骤包括使用DBMS在初始为空的数据库中实现数据库模型。数据库实现是数据库开发人员使用DBMS创建数据库模型以实现实际功能的简单过程，这一过程与使用构建工具为一个实际的工程创建蓝本大致相同。如前所述，大多数现代数据库都是基于关系数据库建模的。正因如此，它们都是由关系DBMS软件来实现的。SQL[2]（Structured Query Language，结构化查询语言）是大多数关系DBMS软件包所使用的语言。SQL包括创建、修改和删除等命令，这些命令将在数据库实现过程中使用。

1.2.4 开发前端应用程序

开发前端应用程序（developing front-end application）是指通过设计、创建应用程序以方便终端用户间接使用数据库的过程。大多数数据库系统都包含这样的应用程序。前端应用程序不仅基于数据库模型，还要基于需求中明确提出的用户所需要的前端应用。前端应用程序通常包含窗体和报表等形式的接口，可以方便地通过导航机制（如菜单）找到。图1-5说明前端应用程序的设计和创建可以与数据库实现同时进行。例如，前端应用程序的外观和风格、特定组件（比如窗体和报表）的数量，及其各自的功能都可以在数据库实现之前或数据库实现之中被确定。当然，前端应用程序的实际创建也将涉及到与数据库之间的连接，这一连接的最终实施则只能在数据库实现完成以后进行。

1.2.5 数据库部署

当数据库及其关联的前端应用程序实施完成后，接下来的步骤便是**数据库部署**（database deployment）。这一步骤涉及数据库系统的发布，即数据库及其前端应用程序供终端用户的使用。通常来说，这一步也包括将初始的数据填入部署好的数据库中。

1.2.6 数据库使用

一旦数据库系统部署完成，终端用户便可以进行**数据库使用**（database use）。数据库使用涉及对数据库系统中数据的**插入**、**修改**、**删除**以及**查询**（insertion, modification, deletion, and retrieval）。用户可以通过前端应用程序间接使用数据库，也可以通过DBMS直接使用数据库。正如我们所提到的那样，SQL是大多现代关系数据库包所使用的语言。SQL中包含了插入、修改、删除以及查询数据的命令。前端应用程序可以使用这些命令（间接操作），或者终端用户也可以直接使用这些命令（直接操作）。

---

[1] 关系数据模型在第3、4章给出。

[2] SQL有时也读"Sequel"。SQL在第5章讲解。

1.2.7 数据库管理和维护

数据库管理和维护（database administration and maintenance）活动用来支持终端用户对数据库的使用。数据库管理和维护活动包括处理某些技术问题，比如为数据库中的信息提供安全支持，确保足够的硬盘空间来存储数据库的内容，实现数据备份和恢复等过程。

1.3 数据库新版本

在大多数情况下，经过一段时间的使用后，修改和扩展已经存在的数据库系统就变得很有必要，这便需要对已有的数据库系统开发新的版本。数据库的新版本应该按照旧版本的开发步骤来执行，如图1-7所示。

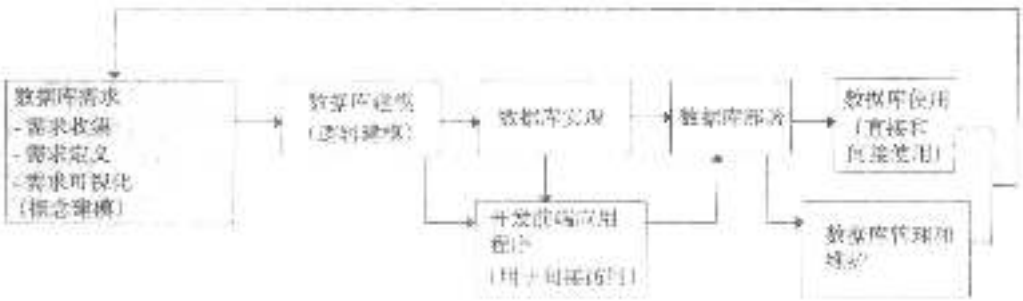


图1-7 数据库新版本的开发

与数据库系统的初始版本相同，后续版本的开发也要从需求的收集、定义和可视化步骤开始。但与初始版本不同的是，在后续版本中，不是所有的需求都要从头开始收集。初期的需求提供了增加和修改的插入点。大多数的增加和修改来自于终端用户在使用前一版本时的观察和反馈，指示了数据库系统可以被改善和扩展的方向。另一个新的需求则来自由数据库系统支持的市场需求的改变，或者是底层技术的改变。不论是什么样的原因，改变是数据库系统固有的特点，应该周期性地推出新版本并做出相应的处理。

1.4 数据库使用范围

数据库有不同的使用范围，从供少量单独用户（个人）使用的数据库到供数以千计的终端用户使用的大型企业数据库。但无论使用范围如何，所有的数据库都有相同的基本开发步骤，即需求收集、建模、实现、部署和使用。不同的数据库使用范围反映在数据库的大小、复杂性、耗时以及每个步骤所需要的资源上。

1.5 参与数据库系统的人员

创建和使用数据库系统涉及不同角色的人员。有关数据库创建、维护和使用的人员，可以根据不同项目和不同公司来设定相应的角色和头衔。人员角色的分类方法有很多种。这里，我们将数据库项目和系统参与人员划分为以下四类：

- 数据库分析者、设计者和开发者。
- 前端应用程序分析者和开发者。
- 数据库管理员。

- 数据库终端用户。

这些分类与不同人员参与的数据库生命周期相关，同时也与分类本身的职能相关。

1.5.1 数据库分析者、设计者和开发者

数据库分析者（database analyst）参与需求的收集、定义和可视化阶段。**数据库设计者/数据库建模者或构建者**（database designer/dataset modeler or architect）参与数据库建模阶段。**数据库开发者**（database developer）负责使用DBMS软件完成数据库模型中的数据库使用功能。同时扮演不同的角色[①]是常有的情况。这些角色负责将市场需求转换到数据库系统中核心的数据库结构。理想的人员要求是精通技术和市场，或者能明确地掌握数据库建模方法以及市场组织过程和原则。

1.5.2 前端应用程序分析者和开发者

前端应用程序分析者（front-end applications analyst）负责为前端应用程序收集和定义需求，**前端应用程序开发者**（front-end applications developer）负责创建前端应用程序。前端应用程序分析者需要确定对于间接使用数据库的终端用户来说，怎样的前端程序才最有用且最适合。这个角色同时还需要决定每个前端应用程序的特征、外观和风格。前端应用程序开发者的职责是基于前端应用程序分析者定义的需求来创建前端应用程序。前端应用程序开发者需要一定的编程知识。

1.5.3 数据库管理员

一旦数据库及其相关的前端应用程序被设计、实现并开发为可运行的数据库系统，就必然要面临管理问题。数据库管理人员围绕数据库系统展开维护，管理的相关工作。**数据库管理员**（database administrator，DBA）的工作涉及与安全相关的技术问题（如创建、分配用户名和密码、数据库的权限管理等）、数据库系统的备份和恢复、监督空间使用和根据需要增加内存空间、以及其他数据库系统相关技术。数据库管理员应该拥有处理这些问题的技能，并掌握数据库设计的细节技术。

1.5.4 数据库终端用户

显然，**数据库终端用户**（database end user）在数据库系统的人员分类中是最重要的一类，他们是数据库系统存在的原因。数据库系统的质量取决于能在多大程度上更快、更容易地支持终端用户获取准确的、完整的信息。终端用户在技术复杂程度方面，需要的数据量、使用数据库系统的频率以及其生产效益方面都存在差异。正如前面我们提到的那样，大多数终端用户通过前端应用程序以间接方式来使用数据库。某些终端用户非常熟悉DBMS的工作机制并且有权限访问DBMS，这些用户则可以通过直接方式使用数据库系统。

我们在这里介绍了组织或公司中直接参与数据库系统的人员的两大类角色。除了这两种类型，还有一些角色，如DBMS软件开发人员以及负责数据库系统所依附的计算机系统的管理员等，他们都是数据库系统运行过程中需要的。

---

① 实际上（特别是在小公司和小组织中），人们同时负责数据库系统的所有部分，包括设计、实施、管理和维护。

1.6 操作型数据库与分析型数据库

数据库系统中收集的信息概括起来有两个使用目的：操作目的和分析目的。

术语**操作信息/事务信息**（operational information/transactional information）是指支撑市场和其他组织日常操作需要所收集和使用的信息。每条信息都来自具体的事务，如从ATM中取款或购买机票等都是操作信息。这也说明了为什么操作信息也称为“事务信息”。

操作型数据库（operational database）收集并呈现日常处理中的操作信息，比如在ATM取款操作中扣除用户支票中的余额或者为顾客处理购买机票的账单等。

术语**分析信息**（analytical information）是指为支持分析任务而收集和使用的信息。ATM的使用方式就是一个具体的分析信息的实例：比如，分析一天中的哪个时间段内发生了最多的取款操作，或者一天中的那个时间段内发生了最少的取款操作。这些信息可以用于为ATM设置放钞计划表。另一个分析信息的例子是显示航空公司机票的销售趋势，比如，美国的哪条航线销售量最多以及那条航线销售量最少。这些信息有助于制定航班计划。

从上面的介绍中不难发现，分析信息需要在操作信息的基础之上得到。例如，为了创建展示ATM一天中不同时间段使用情况的分析信息，必须先合并大量单个ATM取款操作的事务信息实例。同样，为了创建显示不同航线的销售趋势的分析信息，必须先合并大量单个购买机票的事务信息实例。

过去，大多数组织或公司都仅仅维护和使用一些操作型数据库。而现在，创建和使用**分析型数据库**（analytical database）的公司数量不断增加。为反映该事实，本书内容涵盖了关于操作型和分析型数据库的一些内容。关于开发和使用操作型数据库的内容在第2～6章给出，关于分析型数据库的开发和使用在第7～9章给出。

1.7 关系数据库管理系统

关系数据库模型是DBMS软件组成部分的基础。DBMS软件包用于实现目前大多数的操作型和分析型数据库。关系DBMS软件包括Oracle、MySQL、Microsoft SQL Server、PostgreSQL、IBM DB2以及Teradata。在这些DBMS工具中，Teradata适用于大型分析型数据库，其他的DBMS可同时用于操作型数据库和分析型数据库。在第10章，我们将介绍一些DBMS包的基本功能，并展示这些功能如何用于管理和维护操作型数据库和分析型数据库。

1.8 本书内容安排

本书着重介绍与设计、开发、使用操作型和分析型数据库相关的最基本的内容。内容组织如下：

第1章介绍基本术语，列出并简要概述了数据库系统开发的步骤，概括数据库范围，确定参与数据库系统的人员角色分配，明确操作型数据库和分析型数据库的定义。

第2章包括操作型数据库的需求收集和可视化、ER模型、概念数据模型。

第3章介绍操作型数据库建模，包括关系建模和逻辑数据建模。

第4章进一步深入讲解操作型数据库建模并讨论与关系数据库模型相关的其他内容。

第5章详细介绍SQL相关内容，以及如何用其创建和使用关系数据库。

第6章主要介绍操作型数据库系统的实现和使用。

第7章介绍分析型数据库相关的基本术语和概念，比如众所周知的数据仓库和数据集市。
第8章进一步介绍分析型数据库，包括数据仓库和数据集市的建模技术。
第9章介绍数据仓库/数据集市的实现和使用。
第10章介绍操作型数据库和分析型数据库中常用的DBMS功能。
附录包括其他类型数据库的介绍，并简单介绍数据库相关的其他主题。

关键术语[⊖]

analytical database（分析型数据库），10
analytical information（分析型信息），10
conceptual database model（概念数据库模型），5
data（数据），1
database（数据库），2
database administration and maintenance（数据库管理与维护），7
database administrator（DBA，数据库管理员），9
database analyst（数据库分析员），8
data deployment（数据库部署），7
database designer/dataset modeler or architect（数据库设计者/数据库建模者或构建者），8
database developer（数据库开发者），9
database implementation（数据库实现），6
database management system（DBMS，数据库管理系统），2
database metadata（数据库元数据），2
database modeling/logical database modeling（数据库建模/逻辑数据库建模），6
database system（数据库系统），3
database use（数据库使用），7
developing front-end application（前端应用程序开发），7
direct interaction（直接交互），3
end users/business user（终端用户/业务用户），3
front-end application（前端应用程序），3
front-end applications analyst（前端应用程序分析者），9
front-end applications developer（前端应用程序开发者），9
indirect interaction（间接交互），3
information（信息），1
insertion ,modification, deletion ,and retrieval（插入、修改、删除以及查询），7
logical database model/implementational database model（逻辑数据库模型/实施型数据库模型），6
metadata（元数据），2
operational database（操作型数据库），10
operational information/transactional information（操作型信息/事务信息），10
requirements collection, definition, and visualization（需求收集、定义以及可视化），5

复习题

Q1.1 给出几个数据的实例。
Q1.2 给出几个数据向信息转化的实例。
Q1.3 给出你自己的展示数据收集的实例，先给出一个没有元数据的，再给出有元数据的。
Q1.4 描述数据库和DBMS的关系。
Q1.5 数据库系统的主要组成部分是什么？
Q1.6 给出一个间接使用数据库的例子。
Q1.7 数据库系统的开发步骤是什么？
Q1.8 解释数据库需求收集、定义以及可视化过程中的迭代性质。
Q1.9 概念数据建模的目标是什么？

⊖ 本书各章的"关键术语"中，页码为英文原书页码，与书中页边标注的页码一致。

Q1.10 逻辑数据库建模的目标是什么？

Q1.11 简要描述数据库实现的过程。

Q1.12 简要描述前端应用程序开发的过程。

Q1.13 数据库开发的最终产出是什么？

Q1.14 构成数据库使用的四种操作是什么？

Q1.15 给出数据库管理和维护活动的例子。

Q1.16 数据库原始版本和后续版本的开发有哪些相似的地方和不同的地方？

Q1.17 数据库范围是怎样反映在数据库系统的开发中的？

Q1.18 参与数据库项目的人员有哪四大类？

Q1.19 数据库分析者的职责是什么？

Q1.20 数据库设计者的职责是什么？

Q1.21 数据库开发者的职责是什么？

Q1.22 前端应用程序分析者的职责是什么？

Q1.23 前端应用程序开发者的职责是什么？

Q1.24 "数据库系统的用户"与终端用户是如何关联起来的？

Q1.25 给出操作（事务）信息的例子。

Q1.26 给出分析信息的例子。

Q1.27 列出关系 DBMS 软件工具。

第一部分

Database Systems: Introduction to Databases and Data Warehouses

操作型数据库

数据库需求与 ER 建模

2.1 引言

确定数据库需求（database requirement）并创建能将这些需求可视化的概念数据库模型，是开发数据库过程的第一步，也是最关键的步骤。

数据库需求是一系列表述，这些表述指明了该数据库数据及元数据的细节和约束。数据库需求可从业务人员、业务文件、业务政策或者从这三者与其他资源的结合中得到。

正确收集的需求应该确切地描述数据库将记录那些信息，以及以什么方式记录这些信息。实体－联系建模（entity-relationship（ER）modeling）是一种广泛使用的概念数据库建模方法，这种方法可以对收集到的需求进行构建和组织，同时以图形的方式将需求展示出来。

在本章中，我们将介绍如何适当地收集数据库需求，并使用 ER 建模技术将其形象地表示出来。

2.2 ER 建模基本构件

ER 建模后得到的 ER 图（ER diagram，ERD）是整个数据库的蓝图。**实体**（entity）和**联系**（relationship）是 ER 图的两个基本构件。目前还没有一个所有数据库都遵循的通用 ER 符号体系。相反，当前采用的 ER 符号体系种类繁多。虽然采用来自不同符号体系的 ER 符号来表示实体和联系时差异巨大，但其含义通常是相同的。本书将采用陈氏 ER 符号体系的修改版本来表示所有构件，主要原因如下：

- 教学价值：易于初学者学习和使用。
- 完整性：所有基本 ER 概念都有相应的表示。
- 描述可辨别：所有概念都有图形化的表示，概念间可辨别性强。
- 与软件兼容：本书提供的数据库建模软件 ERDPlus（erdplus.com 可下载）也采用了该符号体系。

一旦掌握了一种 ER 符号体系，开发者就可以快速、直接地适应任意其他符号。本章的末尾将给出几种其他符号体系的描述，同时也将说明：熟悉某种 ER 符号的开发者可以非常轻易、自然地理解和使用其他 ER 符号系统。

2.3 实体

实体是 ER 图的基本组件，用于描述数据库所记录的内容。实体[1]可以表示现实世界中的众多概念，如人、地点、对象、事件、项目等。例如，一个零售公司的 ER 图可能包含顾客（CUSTOMER）、商店（STORE）、产品（PRODUCT）和交易额（SALES TRANSACTION）

[1] 实体有时也称作"实体类型"，本书将简单地使用术语"实体"。

两个实体。

ER图用矩形代表实体，实体名写在矩形里面，同一个ER图中的不同实体应该有不同名字。图2-1展示了CUSTOMER及STORE两个实体的例子。

图2-1中每个实体包含多个**实体实例/实体成员**（entity instance/entity member），如实体CUSTOMER可能包含Joe、Sue、Pat等实例。实体本身需要画在ER图中，实体实例虽然不需要表示在ER图中，但会被记录到根据该ER图所创建的数据库之中。

图2-1 两个实体

2.4 属性（唯一和非唯一）

ER图中每个实体都有属性，实体的一个属性描述该实体的一种特征。由于实体表示的是数据库所记录内容的组件，因而实体属性表示每个实例需要记录的细节。例如，对实体CUSTOMER可以记录以下属性：编号（CustID），姓名（CustName），生日（CustBDate），性别（CustGender）。图2-2描述了如何将属性放入ER图中：每个属性由一个写有属性名字的椭圆表示。同一个实体的不同属性名字不同。每个椭圆用一条线连接到相应实体。**唯一属性**（unique attribute）是指可以唯一标识实体实例的属性。通常来说一个实体至少要有一个唯一属性。如图2-2所示，ER图中实体的唯一属性都带有下划线。该图所示的数据库需求表明，每个顾客的编号必须唯一，但两个或多个顾客的生日、姓名或性别则可以相同。

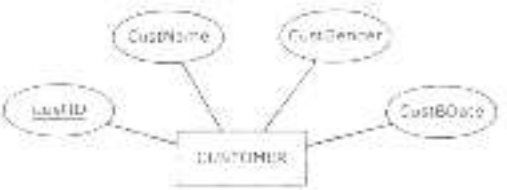


图2-2 一个带属性的实体

2.5 联系

ER图中每个实体必须通过联系至少与一个其他实体相关联。在ER图中联系[⊖]表示为一个菱形，菱形中间写有代表联系名字的词或短语。菱形与所有参与该联系的实体进行连线。

基数约束

ER图中，在实体与联系的连线上往往写有一些符号，这些符号就是**基数约束**（cardinality constraint）。基数约束用于表示该实体可以有多少实例与另一实体的实例存在联系。考虑图2-3中的ER图，菱形"ReportsTo"表示实体EMPLOYEE和实体DEPARTMENT之间的联系。而表示基数约束的符号则常常写于实体和联系之间靠近实体一端的连线上。

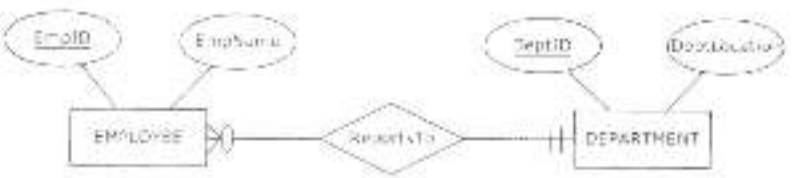


图2-3 两个实体间的联系

每个基数约束包含以下两个部分：

⊖ 联系有时被称作"联系类型"，本书简单地使用术语"联系"。

- **最大基数**（maximum cardinality）——靠近实体一端的基数约束部分；
- **最小基数 / 参与**（minimum cardinality/participation）——远离实体一端的基数约束部分。

最大基数可以是一个（表示为"|"）或多个（表示为">"）。

最小基数可以是可选的（表示为"0"）或者是强制的（表示为"|"）。

图 2-4 列出了实体 A 与联系 B 之间所有四种可能的基数约束。

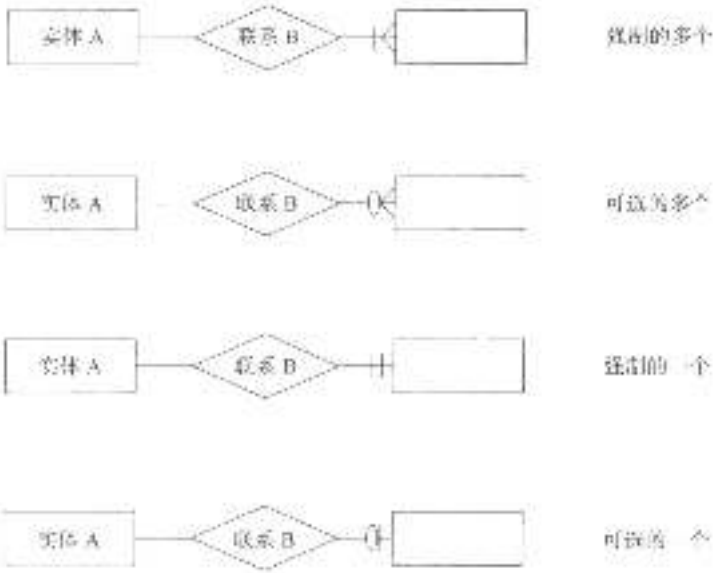


图 2-4　四种可能的基数约束

接下来将以图 2-3 中包含关系 ReprotsTo 的 ER 图为例说明基数约束的概念。以下是图 2-3 中 ER 图的基本需求：

- 数据库将记录职员（employee）和部门（department）信息。
- 对每个职员记录其姓名及唯一职员编号（employeeID）。
- 对每个部门记录其唯一部门编号（departmentID）和位置。
- 每个职员必须且只能为一个部门作报告。一个部门可以有许多职员为其报告，但也可以没有任何职员。

首先考虑图 2-3 中 ReprotsTo 菱形右侧的基数约束符号"||"。强制参与符号（本例中的左侧竖线）表示每个职员必须至少为一个部门作报告。换言之，关系 ReortsTo 的 EMPLOYEE 实体的最大基数约束是 1。最大基数约束符号一个（本例中的右侧竖线）表示每个职员至多可以为一个部门作报告。因此，这两个符号放在一起，表示每个职员必须且只能为一个部门作报告。

15

接下来考虑 ReportsTo 菱形左侧的基数约束符号">0"。可选参与符号（0）表示部门可以没有职员来作报告，换言之，ReportsTo 关系中的 DEPARTMENT 实体的最小基数约束是 0。最大基数约束多个（>）表示一个部门可能有多个职员来作报告。因此，这两个符号放在一起表示每个部门可以有多个职员来作报告，但也可以没有。换言之，一个部门最少有 0 个职员，最多有多个职员。

注意，解释 ER 图中联系的合理方法，是利用矩形－菱形－基数约束—矩形规则分别从

相反方向分两次来考虑这个关系。例如，联系 ReportsTo 就可以解释成：

- 一个方向：矩形（职员）－菱形（作报告）－基数约束（有且仅有一个）－矩形（部门）；
- 另一个方向：矩形（部门）－菱形（接受报告）－基数约束（从 0 到多个）－矩形（职员）。

为进一步重申关系和基数约束的概念，本书接下来将在图 2-5 中给出 ReportsTo 联系的几种可能版本（为简洁起见，省略了实体属性）。

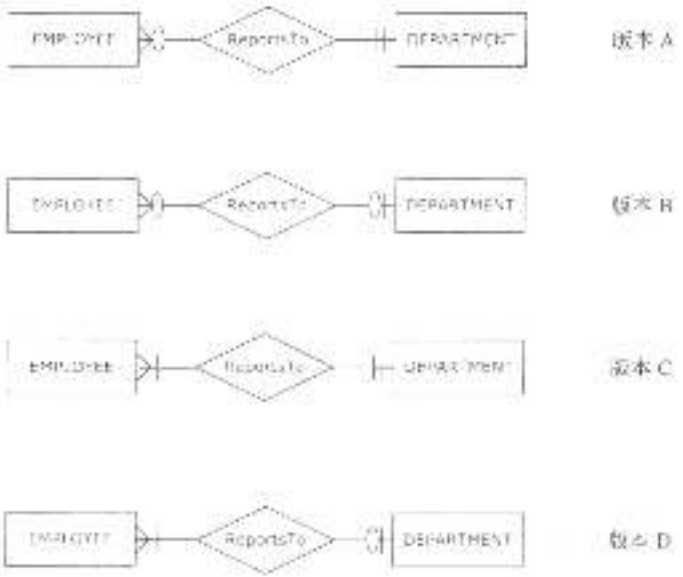


图 2-5 ReportsTo 联系的几种可能版本

以下是每个版本的需求

版本 A：

- 每个职员必须且只能为一个部门作报告。一个部门可能同时有多个职员作报告，也可能没有。

版本 B：

- 一个职员可以为一个或者不为任何部门作报告。一个部门可以同时有多个职员作报告，也可能没有任何职员。

版本 C：

- 每个职员必须且只能为一个部门作报告。一个部门必须有至少一个职员作报告，也可以同时有多个职员。

版本 D：

- 一个职员可以为一个或者不为任何部门作报告。一个部门必须有至少一个职员作报告，也可以同时有多个职员。

2.6 联系类型（最大基数侧）

联系两侧的最大基数约束可为一个或者多个。因此，若不考虑最小基数而仅考虑最大基数，则联系共有以下几种情况：

- **一对一联系**（1 ∶ 1）。

- **一对多联系**（1 ∶ M）。
- **多对多联系**（M ∶ N）。

图 2-6 给出了这三种类型的联系及其最大基数列（由于最小基数不影响最大基数，为简明起见，将其省略）。

[17]

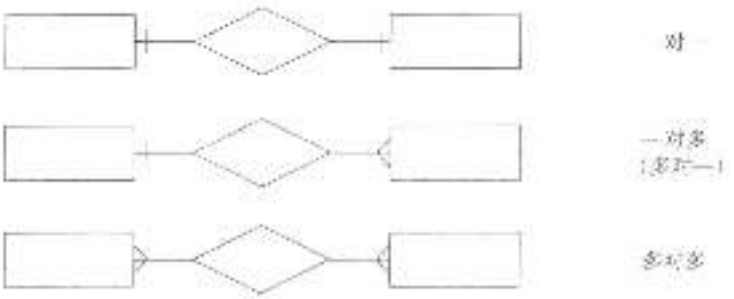


图 2-6　联系的三种类型（最大基数列）

图 2-3 和 2-5 给出了 1 ∶ M 联系的例子。图 2-7 的例子同样也是 1 ∶ M 联系，该联系反映了如下需求：

- 每个商店（store）必须且只能位于一个区域（region）。每个区域内至少要有一个商店，当然也可以有多个商店。

图 2-7　1 ∶ M 联系

图 2-8 给出了一个 M ∶ N 联系的例子。反映的需求如下：

- 一个职员（employee）可能分配了多个项目（project），但也可能一个项目也没有。一个项目至少应分配给一个职员，当然也可以分给多个职员共同完成。

图 2-8　M ∶ N 联系

图 2-9 给出了一个 1 ∶ 1 联系，反映的需求如下：

- 每个职员（employee）要么分有一辆车（vehicle），要么一辆也没有。每辆车必须且只能分给一个职员。

图 2-9　1 ∶ 1 联系

2.7　联系和联系实例

前面曾经讨论过，每个实体都有自己的实例，如实体 EMPLOYEE 可能有实例 Bob、Lisa、Maria 等。实体本身需要画在 ER 图中，实体实例虽然不需要表示在 ER 图中，但会被记录

到根据该ER图所创建的数据库之中。与此类似，联系也有自己的实例。图2-10给出了一个联系及其实例的例子

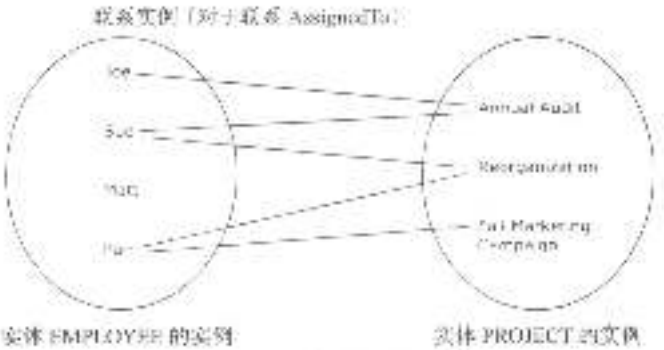


图2-10 一个联系及其实例

如图2-10所示，当一个实体的实例通过联系与另一个实体的实例相关联时，一个联系实例就产生了。就像实体和实体实例的情况一样，联系本身会被画进ER图，联系实例则不需要表示在ER图中，但会被记录到根据该ER图所创建的数据库中。

注意，图2-10中的实体EMPLOYEE在联系AssignedTo中为可选参与，所以才可能存在职员（如Matt）与右边所有项目都不相连的情况。而由于PROJECE实体在联系AssignedTo中为强制参与，因此每一个PROJECT的实例都必须至少与一个EMPLOYEE实体的实例存在连线。

2.8 联系属性

在许多情况下，多对多联系有自己的属性，这些属性就是**联系属性**（relationship attributes）。图2-11给出了一个例子，该ER图的需求如下：

- 数据库将记录学生（students）和校园组织（campus organizations）信息。
- 对每个学生记录其唯一学号（student ID）、姓名（name）及性别（gender）。
- 对每个校园组织记录其唯一组织编号（organization ID）和位置（location）。
- 数据库中每个学生必须至少属于一个校园组织，也可以同时属于多个组织。
- 数据库中每个校园组织至少有一个学生加入其中，也可以同时拥有多个学生。
- 对属于某个校园组织的每个具体的学生实例，记录该学生在该校园组织中的职能（function）（如主席、副主席、会计、成员等）。

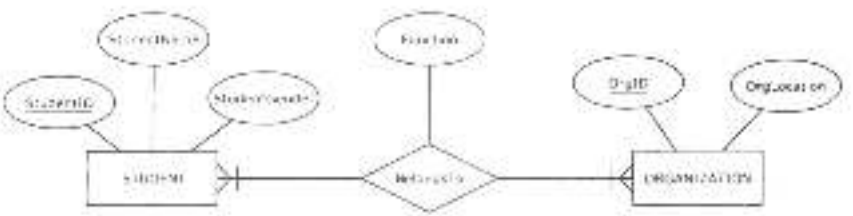
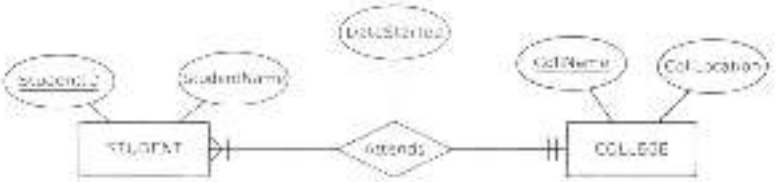


图2-11 带有一个属性的M：N联系

上面的最后一条需求表明，一个学生在不同的校园组织中可以有多个不同的职能。若将职能作为实体 STUDENT 的一个或多个属性，则无法知道某个职能是该学生在哪个社会组织中的职能。若将职能作为 ORGANIZATION 的一个或多个属性，又无法知道该社会组织的某职能具体由哪个学生来担任。因此，如图 2-11 所示，职能属性唯一恰当的位置是作为 BelongsTo 联系的属性。

19



对比

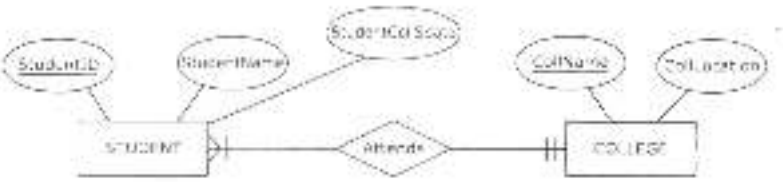


图 2-12　带有和不带有一个属性的 1 : M 联系

接下来讨论 1 : 1 联系与 1 : M 联系是否也可以拥有属性。为了回答这个问题，图 2-12 给出了两个有细微差别的 ER 图，这两个图的需求如下：

- 数据库记录学生（student）和学院（college）信息。
- 对每个学生记录其姓名和唯一学号（student ID）。
- 对每个学院记录其唯一名称（name）和位置（location）。
- 每个学生必须且只能加入一个学院。
- 每个学院有多个学生。
- 对每个学生记录其注册进入学院的日期。

图 2-12 的上半部分给出了 1 : M 联系的一个属性。下半部分基于同样的需求，但将学生的注册日期作为实体 STUDENT 的属性 StudentCollSdate，而不是作为联系 Attends 的属性 DateStarted。由于一个学生只能加入一个学院，因此该学生加入该学院的日期可以作为 STUDENT 实体本身的一个属性。如本例所述，1 : M 联系的一个属性可转化为在该联系中最大基数为 1 的实体属性（本例中的 STUDENT）。更一般地讲，联系属性可以转化为在该联系中最大基数为 1 的实体属性。因此，1 : M 联系或 1 : 1 联系的属性都不是必要的。相反，如图 2-11 所示，对一些 M : N 联系来说，属性则是必要的。

20

2.9　实例：数据库需求集及其 ER 图

下面这些与零售相关的实例都是数据库需求及其对应 ER 图的例子。

ZAGI 零售公司的销售部门准备创建一个数据库来记录销售过程的细节。在访问公司并学习了公司文件以后，数据库团队抽取出了如下需求。

ZAGI 零售公司销售部门数据库将收集如下数据：

- 对每个在售产品（product）：产品号（product ID）（唯一）、产品名称（product name）、价格（price）；
- 对每个种类（category）：种类号（category ID）（唯一）、种类名称（category name）；
- 对每个自动售货机（vendor）：售货机号（vendor ID）（唯一）、售货机名称（vendor name）；
- 对每位顾客（customer）：顾客号（customer ID）（唯一）、姓名（name）、邮政编码（zip code）；
- 对每个商店（store）：商店号（store ID）（唯一）、邮政编码（zip code）；
- 对每个区域（region）：区域号（region ID）（唯一）、区域名称（region name）；
- 对每项销售交易（sales transaction）：交易号（transaction ID）（唯一）、交易时间（date of transaction）。
- 每个产品必须且只能由一个售货机供应；
- 每个售货机可以包含一个或多个产品；
- 每个产品必须且只能属于一个种类；
- 每个种类可以包含一个或多个产品；
- 每个商店必须且只能位于一个区域；
- 每个区域可以包含一个或多个商店；
- 每项销售交易只能出现在一个商店中；
- 每个商店可以有一项或多项销售交易发生；
- 每项销售交易必须且只能跟一位顾客相关；
- 每位顾客可以与一项或多项销售交易相关；
- 每个产品可以通过一项或多项销售交易售出；
- 每项销售交易可以包含一个或多个产品；
- 对于每个通过销售交易售出的产品实例，记录其售出的数量。

21

图 2-13 给出了基于以上需求得到的 ER 图。注意，ER 符号体系的知识，能够为数据库需求收集人员获得结构化的、有用的需求提供帮助。换言之，ER 图的构建必须以完成数据库需求搜集过程为基础，认清这一点可以帮助数据库需求收集人员提出恰当的问题。下面列出了几个相应问题的例子（来自图 2-13 中 ER 图的需求搜集过程）：

- 对一位顾客，你最想记录什么信息？（顾客号、姓名及邮政编码）
- 每位顾客的顾客号是否唯一？（是的）
- 每位顾客的姓名是否唯一？（不是）
- 一个产品属于一个种类还是多个？（一个）
- 一个产品是否只能来自一个售货机？（是的）
- 是否曾经有过某售货机不提供任何产品的情况？（没有）
- 是否需要记录那些到目前为止没有买过任何商品的顾客（那些没有参与任何销售交易的顾客）？（不需要）

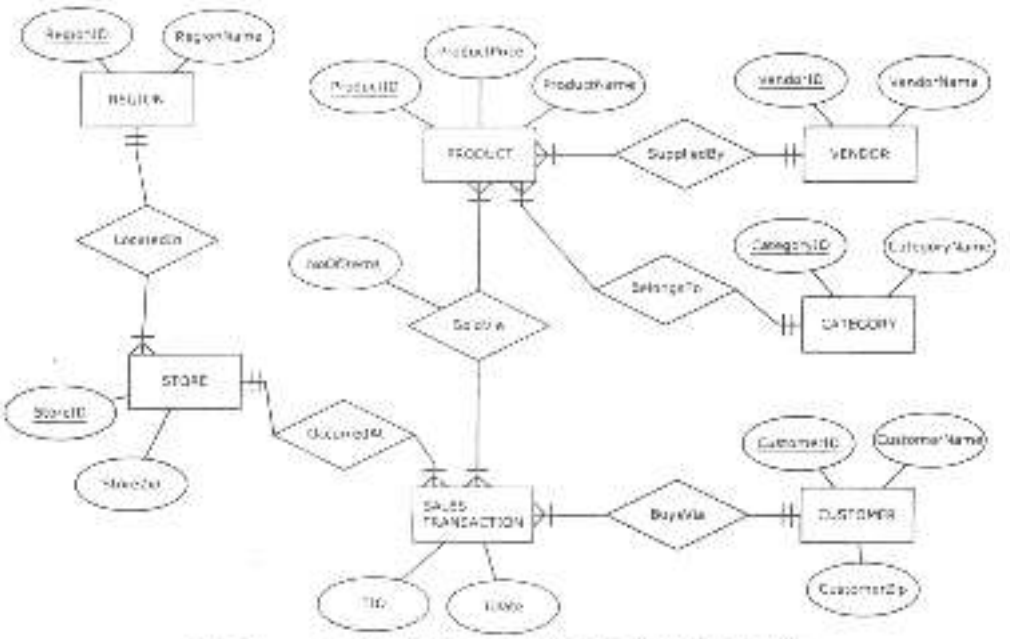


图 2-13　一个 ER 图实例：ZAGI 零售公司销售部门数据库

2.10　复合属性

除前文中提到的常规属性外，ER 图还可以描述几种其他属性。**复合属性**（composite attribute）就是其中的一种。复合属性是若干属性的组合。图 2-14 就给出了一个复合属性的例子。

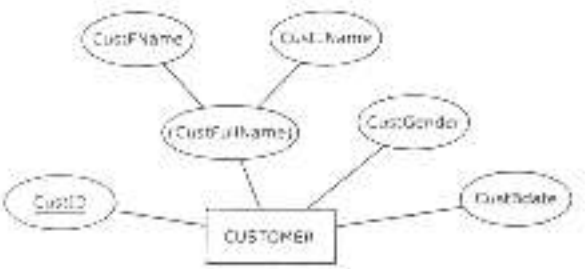


图 2-14　拥有复合属性的实体

在图 2-14 中，属性 CustFullName（顾客全名）由两个部分组成：属性 CustFName（顾客名字）和属性 CustLName(顾客姓氏)。复合属性用于表示由若干单个属性组成的属性集合拥有新含义的情况。在图 2-14 中，实体 CUSTOMER 有 5 个属性：编号（CustID）（描述顾客信息的唯一标识号）、性别（CustGender）、生日（CustBdate）、名字（CustFName）、姓氏（CustLName）。全名并不是实体 CUSTOMER 的额外属性，相反，它仅仅表示将名字和姓氏结合起来以后所得到的完整的顾客姓名。

图 2-15 给出了另一个复合属性的例子。实体 STORE 共有 6 个属性，每个属性都有自己的意义。若将属性街道（Street）、编号（StreetNumber）、城市（City）、州（State）、邮编（Zip）放在一起考虑，则可得到一个新的含义：商店地址（Store Address）。

图 2-16 给出了包含 7 个属性的 BOUTIQUECLIENT 实体的例子。这家独具一格的裁缝店记录了每位顾客的尺寸信息及顾客的名字（firstname）。同时，该裁缝店还记录了顾客的 5 种穿衣指数：腿长（inseam）、腰围（waist）、袖长（sleeves）、肩宽（shoulders）以及领围（collar）。这个例子说明一个简单属性可能是多个复合属性的组成部分。腿长及腰围可组成属性裤子尺寸（pantsize），而腰围、肩宽、领围及袖长可组成属性衬衣尺寸（shirtsize）。

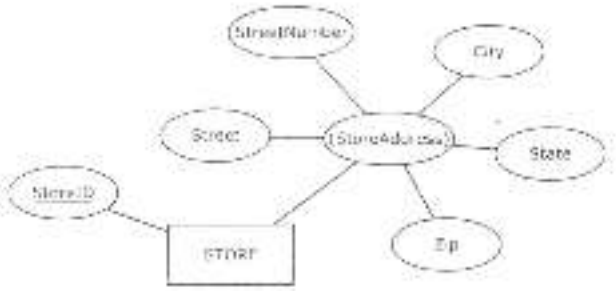


图 2-15　另一个拥有复合属性的实体

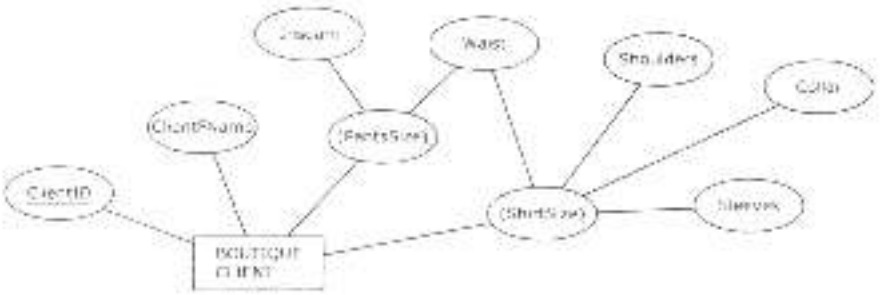


图 2-16　复合属性共享成分

2.11　复合的唯一属性

图 2-17 给出了拥有复合的唯一属性（composite unique attribute）的实体例子。在图 2-17 中，教学楼（Building）、房间号（Room Number）、座位数（Number of seats）是实体 CLASSROOM 的三个属性，但它们都不唯一。CLASSROOM 实体的需求表示如下：

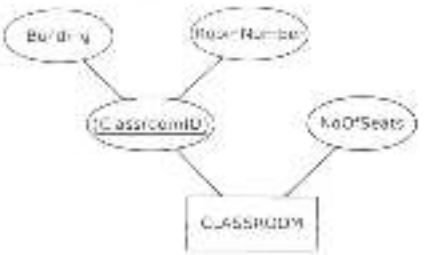


图 2-17　拥有复合的唯一属性的实体

- 同一栋教学楼中可以有多个教室（如 A 教学楼中有若干个教室）。
- 可以有多个教室拥有相同的房间号（如 A 教学楼的 111 房间与 B 教学楼的 111 房间）。
- 可以有多个教室拥有相同的座位数（如若干个教室都有 40 个座位）。

由于三个属性都不唯一，所以 CLASSROOM 没有一个可以唯一标识其本身的简单属性（single-component）。然而教学楼和房间号两个属性的结合却是唯一的（因为在整个数据库中，给定教学楼和房间号可以确定唯一教室），因此，这两个属性组成的复合属性可以作为唯一属性。

复合属性 ClassroomID 使得实体 CLASSROOM 符合每个实体至少要有一个唯一属性的规则。

2.12　多个唯一属性（候选码）

图 2-18 给出的实体同时拥有多个唯一属性，这里的每个唯一属性就叫做一个候选码（candidate key）。"候选"的意义在于：这些属性都可作为构建整个数据库时主要区别属性（或

标主码）的备选。换言之，这些候选码中的一个会成为表中该实体的主码。主码将在第3章进行讨论。

在图2-18中，属性EmpID和SSN都可以区分一个EMPLOYEE实体，因此两者都是EMPLOYEE的候选码。所有候选码都有下划线标识。属性薪水（Salary）并不唯一，因此不是候选码。

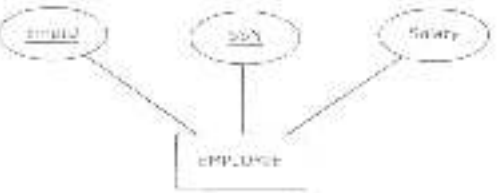


图2-18　拥有多个单一属性（候选码）的实体

一个实体可以同时将常规属性（单个复合属性）或复合属性作为主码。图2-19中，实体VEHICLE的属性VIN（vehicle identification number）是唯一的，同样，State和LPNumber组合起来的复合属性LPlate也是唯一的。

24

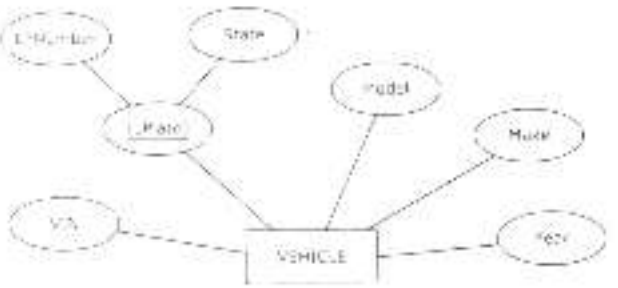


图2-19　拥有一个单一的和复合的候选码的实体

2.13　多值属性

多值属性（multivalued attribute）用于实体实例的同一属性可以有多个不同取值的情况。图2-20给出了一个有多值属性的实体例子，标识多值属性的椭圆外画有双线。图2-20中的ER图需要记录用户的多个电话号码。

多值属性用于那些属性取值多于一个的实体。图2-20就是这样的情景：对于每个职员，需要记录若干个联系电话。例如，某职员可能有两个联系电话，而有些职员可能有多于两个的联系电话、或是少于两个的联系电话。

如图2-21所示，若对每个职员都记录两个联系电话（办公室电话和手机），则不需要使用多值属性，只要分别使用两个单值属性就可以了。

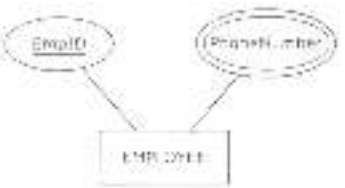


图2-20　一个多值属性

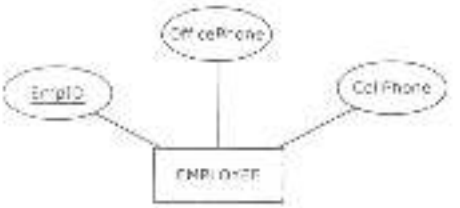


图2-21　不使用多值属性的场景

2.14　派生属性

派生属性（derives attribute）是非永久性存于数据库的属性。派生属性的值可以从别的

属性值或其他数据（如当前日期）派生出来。图 2-22 给出了一个从 ER 模型中得到派生属性的例子。

派生属性用虚线的椭圆标识。属性 OpeningDate 作为一个常规属性会存于最终的数据库中，YearsInBusiness 作为一个派生属性则不会存于数据库中，而是从商店的 OpeningDate 及当前日期中派生出来。如果 YearsInBusiness 是一个常规属性，其值将存于数据库中且需要人工更新（一般每年一次），否则数据库中将出现错误信息。而将 YearsInBusiness 作为派生属性后，就可以确保数据库以后将该属性作为一个公式，用于得到正确的当前 YearsInBusiness 的属性值。

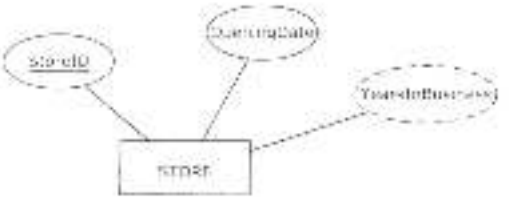
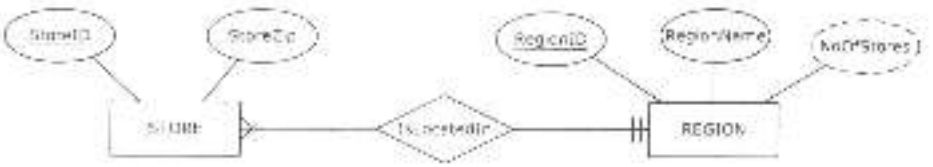


图 2-22 一个派生属性的例子

图 2-23 给出了另一个派生属性的例子。在本例中，对与 REGION 的实例有 IsLocatedIn 联系的 STORE 实例进行计数，就可以得出派生属性 NoOfStores 的值。



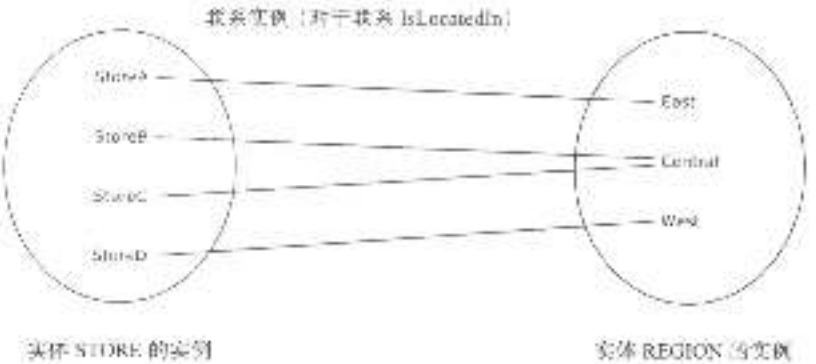


图 2-23 另一个派生属性的例子

在本例中，属性 NoOfStores 在东区（East）、中心区（Central）、西区（West）的值分别为 1、2、1。如果实体 STORE 有了一个新的实例 StoreE，且该实例通过新的 IsLocatedIn 联系实例与 REGION 实例 West 相关，则西区（West）的派生属性 NoOfStores 会自动地从 1 增加为 2。

2.15 可选属性

对每个实例，实体的大部分属性都有相应的取值，但有的属性也可能没有取值，这些属性就是**可选属性**（optional attribute）。图 2-24 给出了一个可选属性的例子，表示了如下需求：

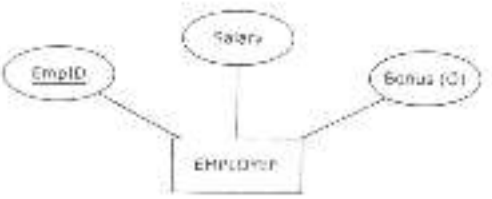


图 2-24 一个可选属性

- 对每个职员记录其唯一的编号（employee ID）及其薪水（salary）和年终奖（annual bonus）。但并非所有职员都有年终奖（annual bonus）。

图中的实体只有一个可选属性，其他属性均为必需属性。我们在属性名后加一个带括号的大写字母 O 来表示可选属性。

[26]

2.16　实例：包含多种类型属性的实体

简单回顾一下各种类型的属性，图 2-25 展示了一个包含多种类型属性的实体。该实体反映了以下需求：

- 数据库将记录雇员信息。
- 对于每一名雇员，将记录以下属性：雇员的唯一 ID、唯一的 e-mail、姓、名（姓和名可以组合成完整的姓名）、多个电话号码、出生日期、年龄（通过出生日期和当前日期计算）、工资、奖金（可选）、多种技能、唯一的社会保险号码、雇员首次被雇佣的日期以及雇员在该公司的工作年限（通过雇员首次被雇佣的日期和当前日期计算）。

图 2-25　拥有多种类型属性的实体

2.17　联系中最小基数和最大基数确切值

在某些情况下，联系中的**最小基数和最大基数确切值**（exact minimum and/or maximum cardinality）是事先知道的。在本书中，我们用圆括号括起来的一对数表示最小基数和最大基数，括号内的第一个数表示最小基数，第二个数表示最大基数。根据事先知道的最小基数和最大基数的信息，这些数对可以出现在联系的任意一边或者两边。

如图 2-26 所示，对于联系 EnrollsIN，需求如下：

- 每名学生必须选择最少 2 门最多 6 门课程，且一门课程要有最少 5 名最多 40 名学生。

图 2-26　具有确定最小基数和最大基数的联系

具体的最大基数和最小基数是用菱形符号和基数约束符号之间的数对表示的。

本书中，当需要用圆括号括起来的数对表示最小和最大基数时，即使它们中的一个是不确定的值，我们仍需要将两个值（最小值和最大值）都表示出来。如图 2-27 所示，对于联系 EnrollsIN，需求如下：

[27]

- 每名学生最多选择 6 门课程，也可以不选任何课程。一门课程必须要有至少 5 名学生，选课人数没有上限。

图 2-27 一个既有确定的又有不确定的基数的联系

在这个例子中，通过两种方式表明实体 STUDENT 可选地参与到联系 EnrollsIn 中：一种是使用最小基数的可选参与符号；另一种是通过联系符号与基数约束符号之间的数对说明最小约束为 0。同样，实体 CLASS 的最大基数（多个但没有确定值）也是通过两种方式来说明的：一种是通过鸭掌符号（ε）说明最大基数；另一种是通过数对中的字母 M 表明最大基数。

2.18 一元联系和联系的角色

联系的度（degree of a relationship）表示有多少个实体参与到该联系中。目前为止，本书所涉及的联系都只涉及两个实体。两个实体之间的联系叫做**二元联系**（binary relationship）或者度为 2 的联系（因为该联系涉及两个实体）。尽管绝大多数业务 ER 图中的联系都是二元联系，但是也会出现度不为 2 的联系。度为 1 的联系也称为**一元联系**（unary relationship）或者**递归联系**（recursive relationship），出现在一个实体与它自己相联系的情况中。图 2-28 展示了三个一元联系的例子，分别是 1 : N、M : N 和 1 : 1 三种情况。

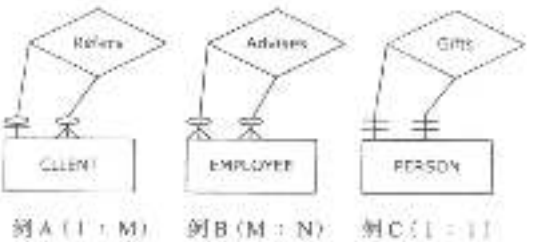


图 2-28 一元联系的例子

图 2-28 中，例 A 的需求如下：

- 一个客户可以推荐多个客户，也可以不推荐任何客户。每个客户可以由另一个客户推荐或者没有被任何客户推荐。

图 2-28 中，例 B 的需求如下：

- 一个雇员可以指导许多雇员，也可以不指导任何雇员。一个雇员可以被多个雇员指导，也可以不被任何雇员指导。

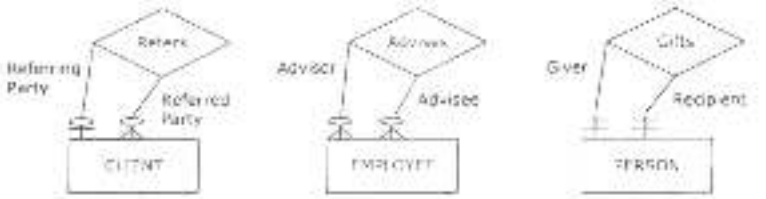
图 2-28 中，例 C 的需求如下：

- 在一个赠送礼物活动数据库中（神秘圣诞老人[⊖]），每个人只能向另一个人赠送礼物，每个人也只能收到另一个人的礼物。

在 ER 图中，**联系角色**（relationship role）可以表达额外的语义信息。数据建模者可以使用联系角色进一步说明每个实体在联系中的角色。联系角色可以用在任何度的联系中，但联系角色的作用通常体现在一元联系中。我们用写在联系连线旁边的文本来表示联系角色。图 2-29 展示了一些带有特定联系角色的一元联系的例子。

图 2-29 中的各图可以做如下额外解释。

⊖ 西方过圣诞节的一个传统活动。通常是一个组织内部或一个大家庭中，每个人被随机地分配以匿名的方式向另外一个人赠送礼物。



例A　　例B　　例C

图2-29　具有角色名的一元联系的例子

图2-29中，例A的解释如下：

- 一个客户可以是推荐者，推荐多个客户。该客户也可以不是推荐者。
- 一个客户可以是被推荐者，且只能被一个客户推荐。该客户也可以不是被推荐者。

图2-29中，例B的解释如下：

- 一个雇员可以是指导者，指导多个雇员，也可以不是指导者。
- 一个雇员可以是被指导者，被多个雇员指导，也可以不是被指导者。

图2-29中，例C的解释如下：

- 一个人必须是礼物赠送者，且只能向一个人赠送礼物。
- 一个人必须是礼物接受者，且只能接受一个人的礼物。

图2-30展示了一个二元联系中联系角色的例子。

图2-30　具有角色名的二元联系的例子

正如前文中提到的，ER图建模者可以任意使用联系角色。在某些情况下，联系角色能够使问题变得清晰，然而过度使用联系角色会产生重复问题。例如，在图2-30中，有一定经验的读者不需要显式地通过联系角色来理解联系"Ships"的含义。因此，不标明联系角色的ER图将提供同样的信息，并且更加简明。

29

2.19　相同实体间的多种联系

ER图中，相同实体之间通过多种联系连接起来的情况是很常见的。图2-31展示了这种情况。该例子基于如下需求：

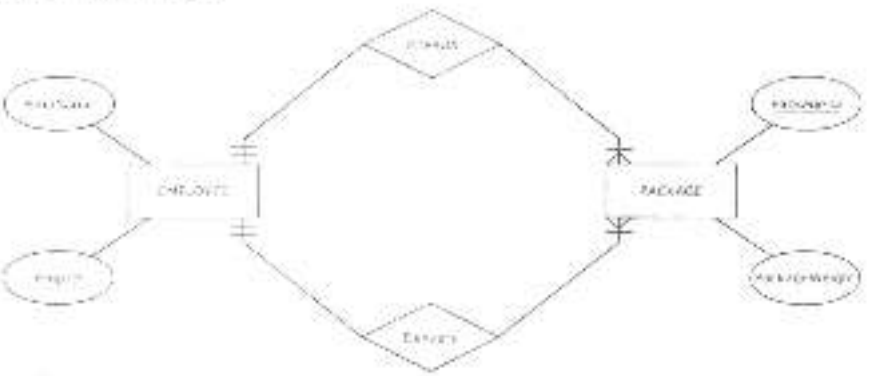


图2-31　相同实体间的多种联系

- 一个招聘公司要创建一个数据库以记录它的雇员信息和包裹信息。
- 每个包裹只能由一个雇员分拣。
- 每个雇员可以分拣多个包裹。
- 每个包裹只能由一个雇员递送。
- 每个雇员可以递送多个包裹。

2.20　弱实体

通常情况下，实体至少要有一个唯一属性。在 ER 图中，**弱实体**（week entity）用来表示没有唯一属性的实体。弱实体是用双框的矩形表示的。在 ER 图中，弱实体必须和它的**属主实体**（owner identity）通过**标识性联系**（identifying relationship）连接起来。该联系是用双框的菱形表示的。

图 2-32 展示了一个弱实体的例子。该例子基于如下需求：

- 一个公寓出租公司要创建一个数据库以记录它的建筑物信息和公寓房间信息。
- 对于每个建筑物，将记录唯一的建筑物 ID 和该建筑物的层数。
- 对于每套公寓，将记录公寓编号和公寓内的房间数。
- 每个建筑物内有多套公寓，每套公寓只能位于一个建筑物内。
- 在我们的数据库中，多套公寓可以有相同的公寓编号，但在一个建筑物内每套公寓只能有一个唯一的公寓编号。

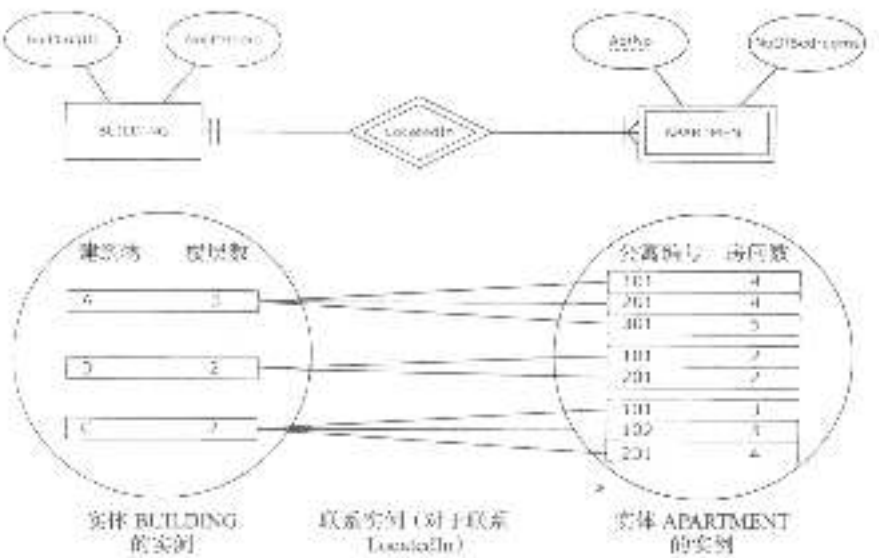


图 2-32　一个弱实体的例子和该实体的实例

正如需求中描述的，实体 APARTMENT 没有唯一属性。然而，在每个建筑物内部，公寓编号是唯一的。这种属性叫做**部分码**（partial key），在 ER 图中用下划虚线表示。部分码和属主实体的唯一属性的组合可以唯一标识弱实体的每个实例。例如，弱实体 APARTMENT 的实例可以由它的部分码 AptNo 和属主实体 BUILDING 的主码 BuildingID 唯一标识，如下所示：

A 101，A 201，A 301，B 101，B 201，C 101，C 102，C 201

[31]　事实上，弱实体的概念和多值复合属性的概念很相似。图 2-33 说明了这一点。

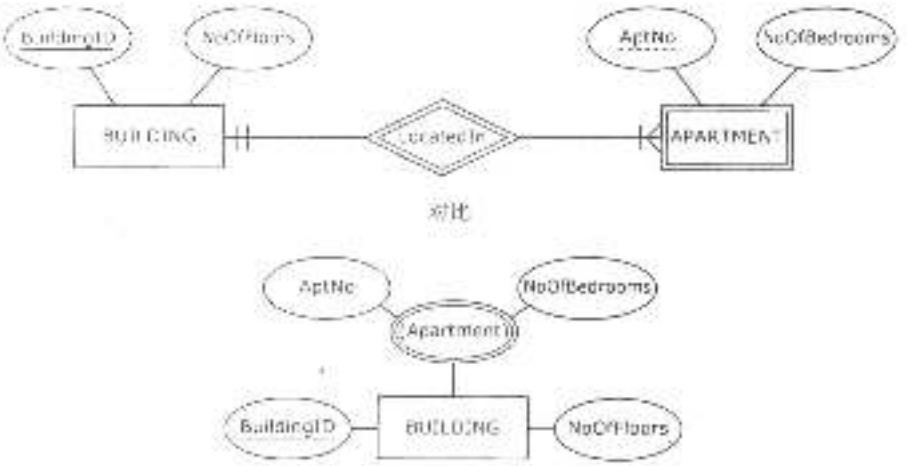


图 2-33　弱实体与多值复合属性

图 2-33 中的两个 ER 图都满足同样的数据库需求。然而弱实体符号能描述部分码，而多值复合属性则不能。例如，图 2-33 上半部分的 ER 图中，我们能显式地说明一个建筑物内的公寓编号是唯一的，而图 2-33 下半部分的 ER 图则不能。

一个弱实体和其他实体之间可以有普通（非标识性的）联系，而多值复合属性则不能表现这种联系。图 2-34 展示了一个弱实体参与非标识性联系的例子。

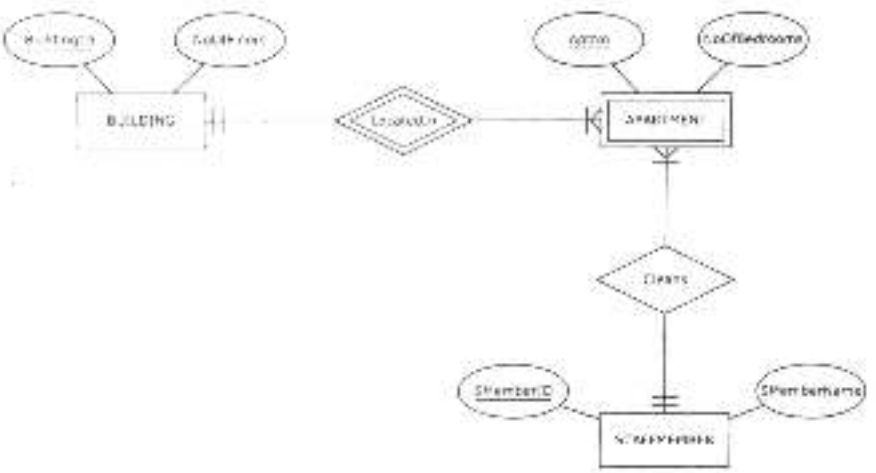


图 2-34　一个有标识性联系和普通联系的弱实体

尽管弱实体和多值复合属性有很多相似之处，然而，正如前文所述，这两个概念之间还是有显著区别的。

每个弱实体在和其主实体之间的标识性联系中总存在强制的一基数约束，用以确保每个弱实体的实例只和属主实体的一个实例发生关联。另一方面，属主实体可以强制性或选择性地参与到标识性联系中。也就是说，属主实体中的实例可以不与任何弱实体中的实例发生关联。

大多数情况下，标识性联系都是1：M的联系。然而，也会出现1：1联系的情况，在这种情况下，弱实体中的部分码就不必作为标识属性了。图2-35说明了这种情况。该例子基于如下需求：

- 数据库将记录雇员以及他们的配偶信息。
- 对于每个雇员，将记录他的唯一ID和姓名。
- 对于每个配偶，将记录他的姓名和出生日期。
- 每个雇员可以有一个配偶或者没有配偶。
- 每个配偶只能和一个雇员结婚。

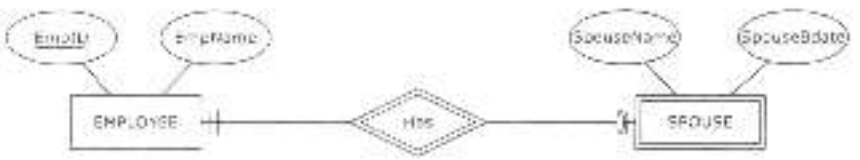


图2-35 一个有1：1标识性联系的弱实体

需要注意的是，弱实体SPOUSE不需要部分标识，因为每个雇员只能和一个配偶相关联，因此不需要部分标识来区别不同的配偶。

2.21 实体、属性和联系的命名约定

在ER建模的过程中，采取特定的准则来为实体、联系和属性命名是一个很好的做法。在本书中，我们约定：使用大写字母来命名实体，使用大写字母和小写字母的组合来命名属性和联系。

对于命名实体和属性，一个常见的准则是使用单数（不使用复数）名词以使ER图尽量清晰易读。例如，实体名STUDENT、STORE和PROJECT要比STUDENTS、STORES和PROJECTS更好。同样，属性名Phone要比属性名Phones好，尽管它是一个多值属性。实体和多值属性都有多个实例，这点从概念本身就能理解，并不需要使用一个复数形式的词来说明。

对于命名联系，通常是使用动词或动词短语而不是名词。例如，用Inspects、Manages和BelongsTo来命名联系是比Inspection、Management和Belonging更好的选择。

当命名实体、属性和联系时，建议尽量简洁但又不要太过简略而使概念模糊不清。例如，在一个大学的ER图中，实体名STUDENT要比实体名UNIVERSITY STUDENT更好。在上下文环境中，很显然STUDENT是指大学学生，因此UNIVERSITY STUDENT就有些冗长了。同样，使用US作为"University Student"的缩写对于将来使用该数据库的一般用户来说又太过隐晦了。这并不是说使用多个单词或者缩写总是不好的选择，例如，SSN就是一个很好的属性名，因为大家都知道它是社会安全号码的缩写。而使用多个单词的短语NoOfFloors命名属性则是比单个单词Floor或者缩写NoF更好的选择，因为使用Floor会产生歧义，大多数用户也都不会理解NoF的含义。

正如本章开始所提到的，ER建模的一条基本规则是同一个实体内的属性必须有不同的名称。而ER建模中一个良好的规则是使整个ER图中的属性名均不相同。例如，不要使用单词Name表示两个不同实体EMPLOYEE和CUSTOMER的两个不同属性，而应该使用不同的单词，如EmpName和CustName。

这里提到的准则都不是强制性的，并且会有例外的情况。然而，如果始终遵守这些准则，这样得到的 ER 图通常会比不使用这些准则得到的 ER 图更加清晰易读。

2.22　多个 ER 图

在一个 ER 图中，每个实体总能通过直接或间接的联系连接到其他所有实体。换句话说，在每个 ER 图中，每两个实体之间总会有一条路径。

如果一个 ER 模式包含不能互相连接的实体，实际上，这个 ER 模式表示了多个单独数据库的 ER 图。例如，在 2-36 中，实体 A、B、C 与实体 D、E、F、G 不相连。因此，实体 A、B、C 属于一个 ER 图的组成部分，而实体 D、E、F、G 属于另一个 ER 图的组成部分。

33

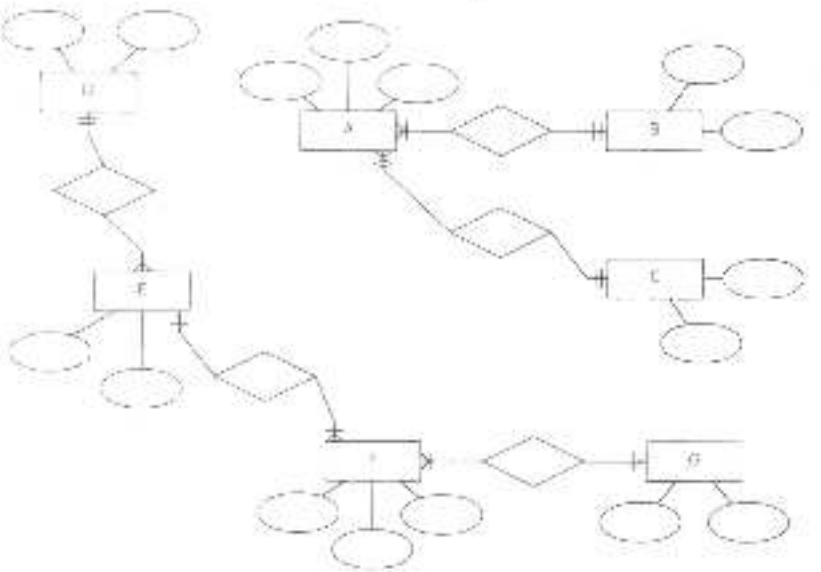


图 2-36　一个模式中包含两个独立的 ER 图（可能会产生误导）

总的来说，当描述多个 ER 图时，每个图应该分开表示。不要采用图 2-36 的方式，一个模式中包含两个 ER 图。更好的选择是如图 2-37 所示，单独呈现每个 ER 图。这可消除由于没有注意到模式包含多个（不相连的）ER 图而造成的混淆。

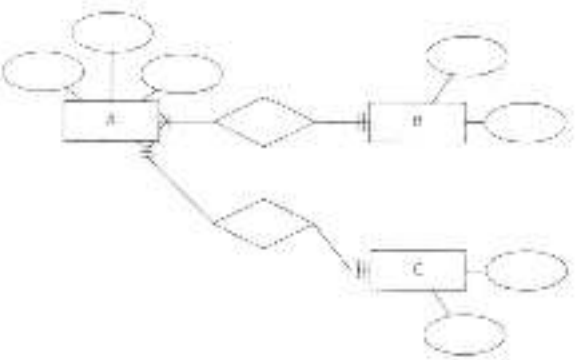


图 2-37　单个模式中包含单个的 ER 图

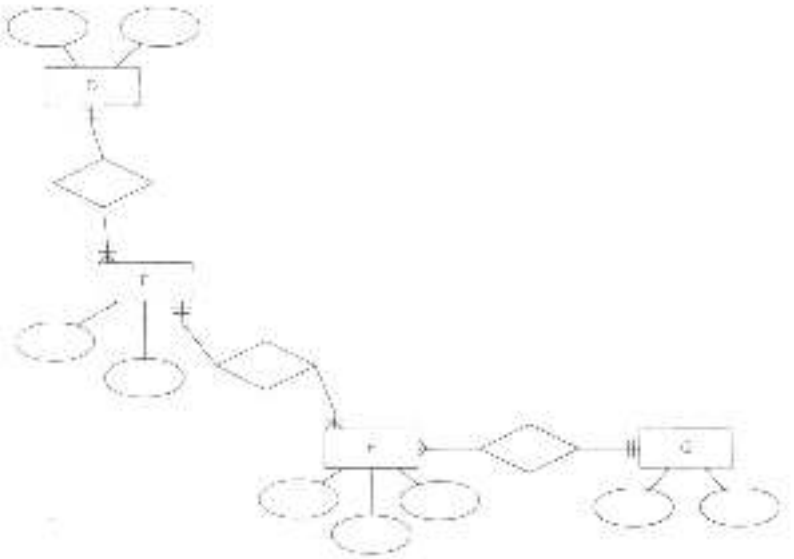


图 2-37（续）

2.23　实例：另一组数据库需求及其 ER 图

复习一下已经介绍过的 ER 模型的概念，请看图 2-38 中另一个 ER 图的例子，该例子基于以下需求：

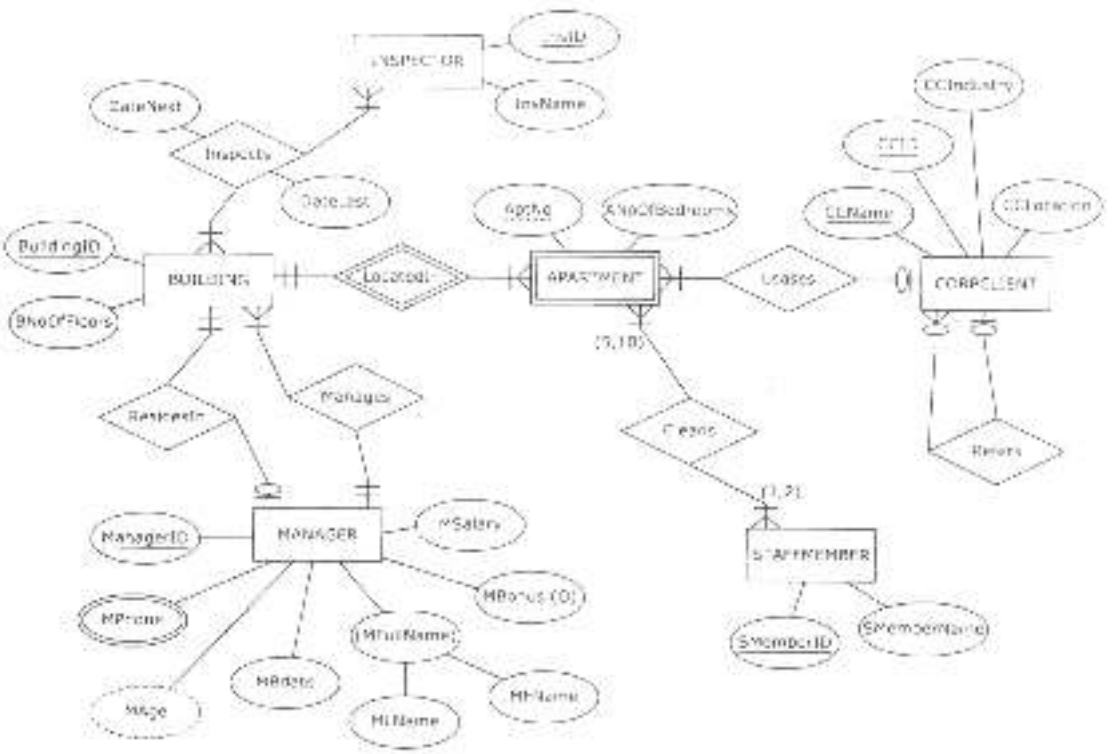


图 2-38　另一个 ER 图的例子：房地产公司 HAFH 的地产数据库

HAFH（Home Away from Home）房地产公司向企业客户提供公寓出租业务。HAFH 房地产公司生产数据库记录 HAFH 的建筑物、公寓、企业客户、建筑物管理员、清洁工成员和建筑物检查员的信息。

房地产公司 HAFH 的生产数据库将记录以下数据：

- 对于每个建筑物：BuildingID（唯一）和 BNoOfFloors（建筑物内的楼层数）。
- 对于每套公寓：AptNo（部分唯一，即在一个建筑物内唯一）和 ANoOfBedrooms（公寓内的房间数）。
- 对于每个企业客户：CCID（唯一）、CCName（唯一）、CCLocation 和 CCIndustry。
- 对于每个建筑物管理员：ManagerID（唯一）、MFullName（由 MFName 和 MLName 组合而成）、多个 MPhones、MBDate、MAge（由 MBDate 和当前日期生成）、MSalary 和 MBonus（不是每个管理员都有奖金）。
- 对于每个清洁工成员：SMemberID（唯一）和 SMemberName。
- 对于每个检查员：InsID（唯一）和 InsName。
- 每个建筑物内有一套或多套公寓，每套公寓只位于一个建筑物内。
- 每套公寓要么只出租给一个企业客户，要么没有出租。每个企业客户可以租用一套或多套公寓。
- 每个企业客户可以推荐许多企业客户也可以不推荐任何企业客户。每个企业客户只能被一个企业客户推荐或者不被任何企业客户推荐。
- 每套公寓由一个或所有清洁工负责管理，每个清洁工负责管理 5～10 套公寓。
- 每个建筑物管理员管理一个或多个建筑物，每个建筑物只由一个管理员管理。
- 每个管理员只能位于一个建筑物内，每个建筑物内要么有一个管理员居住，要么没有管理员居住。
- 每个检查员检查一个或多个建筑物，每个建筑物可以有一个或多个检查员。
- 对于某个检查员检查的建筑物，记录该检查员上次检查该建筑物的日期以及下次将要检查的日期。

2.24　数据库需求和 ER 模型的使用

ER 建模为收集、构建和可视化数据库需求提供了一种简明易懂的技术。理解 ER 建模不仅对基于需求构建 ER 模型是很重要的，并且对需求收集的过程本身也很重要。它有助于我们为建立实体、属性和联系的相关事实而提出正确的问题或找到相应的答案。

初学者第一次使用 ER 建模时常犯的错误是不能区分实体和 ER 图本身。例如，考虑下面一组简单的需求：

- 公司 X 记录其部门以及从属于部门的雇员信息。
- 每个雇员只能属于一个部门。
- 每个部门可以有多个雇员，也可以没有雇员。

图 2-39 的上半部分展示了一个基于该需求的错误 ER 图的例子。该图表明，将该公司描述成一个实体是错误的。这样该实体将只有一个实例（公司 X），这是不必要的，它会使 ER 图本身变得复杂。相反，基于以上一组需求的 ER 图只需分为两个简单的实体：雇员和部门，见图 2-39 的下半部分。不是用一个实体表示公司 X，而是用整个 ER 图（两个实体以及它们之间的联系）表示公司 X。

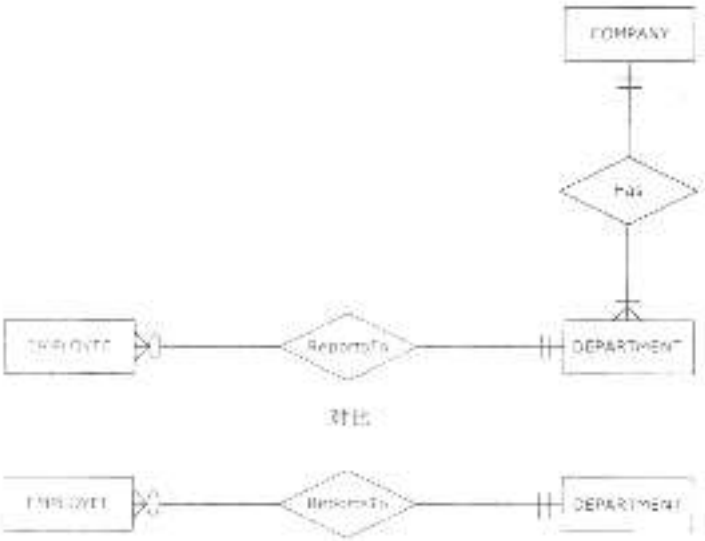


图 2-39　错误解释和正确解释需求的 ER 图

注意，如果上述需求的第一条变为一个行业协会想要记录其成员公司、部门以及雇员，这时图 2-39 上半部分的图就是正确的。然而，需求中明确说明了这是一个公司记录其雇员和部门的数据库，因此，图 2-39 下半部分的图对于该需求是正确的。

为了进一步说明这一点，考虑图 2-40 中的 ER 图。

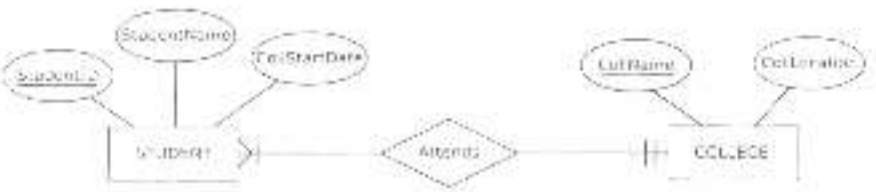


图 2-40　包含学生和大学实体的 ER 图

图 2-40 对于以下需求是正确的：

- 一个非盈利的基金会记录它所资助的奖学金获得者。
- 对于每名学生（奖学金获得者），记录他的唯一学生 ID、姓名以及他就读大学的入学日期。
- 对于每一所拥有该基金会资助学生的大学，基金会记录该大学唯一的大学名称和大学的地理位置。
- 每一名奖学金获得者只能就读于一所大学。数据库中的每所大学至少有一名奖学金获得者。

注意，图 2-40 中的 ER 图对于只记录一所大学的数据库是不正确的。在这种情况下，实体 COLLEGE 是不必要的，就像图 2-39 中的实体 COMPANY 一样。

另一个数据库需求收集和 ER 建模中常见的新手错误是不区分以下两者：

- 对需要被记录的数据建模。
- 对发生在一个组织内的所有事情建模。

例如，下面的一组需求就没有区分这两者：

36

- 航空公司 X 想记录航班、航班空乘人员和乘客信息。
- 每个航班配备多名空乘人员。每名空乘人员可以配备到多个航班。
- 每位乘客可以乘坐多个航班。每个航班载有多位乘客。
- 每名空乘人员可以为多位乘客服务。每位乘客可以由多名空乘人员服务。

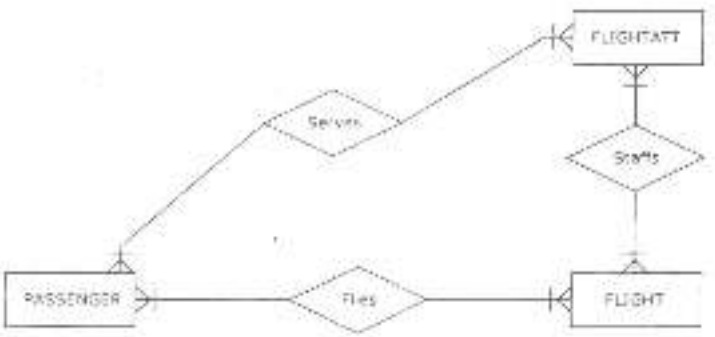
37

图 2-41 上半部分展示了一个基于以上需求的 ER 图。这些需求都准确地表示了航空公司 X，但问题在于这些需求没有考虑到什么是可能和/或有必要记录的数据。这组需求的前三条是合理的，而最后一条有问题。的确，乘客可以由多名空乘人员服务、每名空乘人员也可为多位乘客提供服务。然而，我们很可能不会真的想要记录所有空乘人员为乘客提供服务的组合。即使我们真的想要记录这些信息，也需要考虑这样做的代价和实用性。在这个例子中，如果我们想要实现一种机制以记录每名空乘人员第一次为某旅客服务的信息（如拿饮料、回答问题等），很可能这种机制将会代价过高且没有意义。因此，需求的最后一条应该删除，删除后的数据库需求如下：

- 航空公司 X 想记录航班、航班空乘人员和乘客信息。
- 每个航班配备多名空乘人员。每名空乘人员可以配备到多个航班。
- 每位乘客可以乘坐多个航班。每个航班载有多名乘客。

38

因此，应该如图 2-41 的下半部分所示来构建 ER 图。



对比

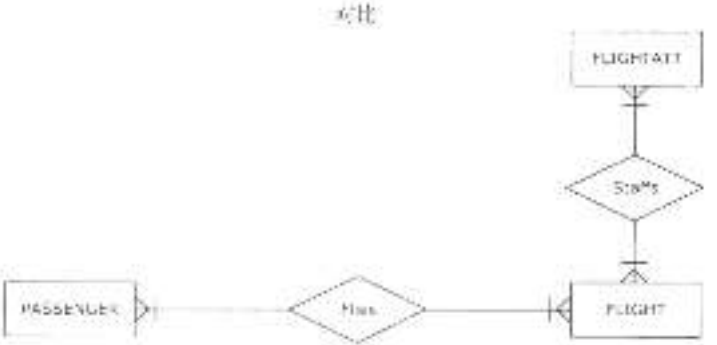


图 2-41 基于不可行的需求与适当需求的 ER 图

2.25 各种 ER 符号体系

正如前文所提到的，现在没有一个所有数据库工程都遵守的且被普遍接受的 ER 符号体

系。相反，目前常用的 ER 符号体系有许多种。然而，如果设计者熟悉某一种 ER 符号体系，那么掌握其他的 ER 符号体系则会很容易。

图 2-42 用三种不同的符号体系说明了同样的 ER 图。图 2-42 顶部的图使用本书的符号体系来表示 ER 图。

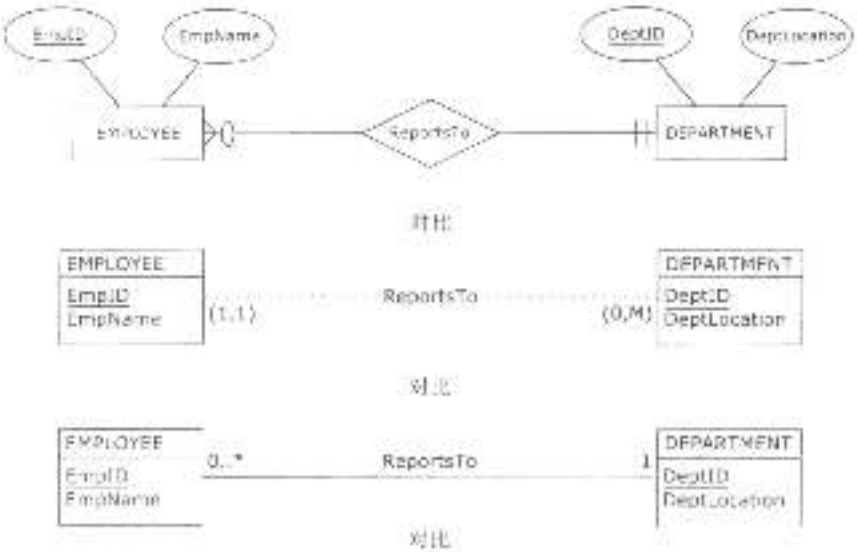


图 2-42　各种 ER 符号体系的例子

图 2-42 中间的图使用了另一种符号体系。在该符号体系中，属性名和实体名放在同一个矩形内，联系用连接实体的虚线表示，联系名嵌入该虚线中。基数约束用括号内的一对数表示，但基数约束是反过来的（与本书所用的符号体系相比）。解释联系的规则是实体－基数约束－联系－实体。例如，实体 EMPLOYEE 的每个实例只和实体 DEPARTMENT 的实例参与到联系 ReportsTo 中一次，而实体 DEPARTMENT 的每个实例可以和实体 EMPLOYEE 的实例 0 次或多次参与到联系 ReportsTo 中。

图 2-42 底部的图使用 UML（Unified Modeling Language）符号体系来表示同样的 ER 图。该符号体系中实体和联系的表示方法和中间图的表示方法很相似，但基数约束的解释方法和顶部的图是一样的。如果最小基数是可选的，则用两个点分开的数字表示；如果是强制的，则只用一个数字表示。

除了这里已经列出来的 ER 符号体系，还有许多其他的 ER 符号体系。幸运的是，它们都描述了同样的概念：实体、联系和属性。熟悉这些概念的开发者能很快适应任何 ER 符号体系。

39

2.26　扩展的 ER 模型

扩展的 ER 模型（enhanced ER，EER）是对 ER 符号体系的扩充，可以描述标准 ER 模型之外的一些概念。附录 A 给出了一个 EER 的简要总结。尽管 EER 对传统 ER 概念的扩展有时会很有用，但大多数业务数据库都可以使用本章介绍的 ER 符号来建模。如果有需要，一个有经验的 ER 建模者能够很容易地学习和应用 EER 扩展。

本章涵盖了数据库需求和 ER 建模的大部分基本问题。下面几节是与 ER 建模有关的几个额外问题。

2.27　问题说明：相同实体之间具有多个实例的 M ：N 联系

在某些情况下，M ：N 联系可以在相关实体的相同实例之间多次出现。下面的例子说明了这种情况。观察以下需求：

- 数据库将记录学生和课程信息。
- 对于每名学生，将记录他的唯一学生 ID 和姓名。
- 对于每门课程，将记录唯一的课程 ID 和课程等级。
- 数据库中的每名学生可以选修多门课程，也可以不选修任何课程。
- 数据库中的每门课程至少要被一名学生选修。
- 对于每个选修某一门课程的学生实例，将记录该学生本门课程的成绩和学期。

图 2-43 展示了一个基于这些需求的 ER 图。

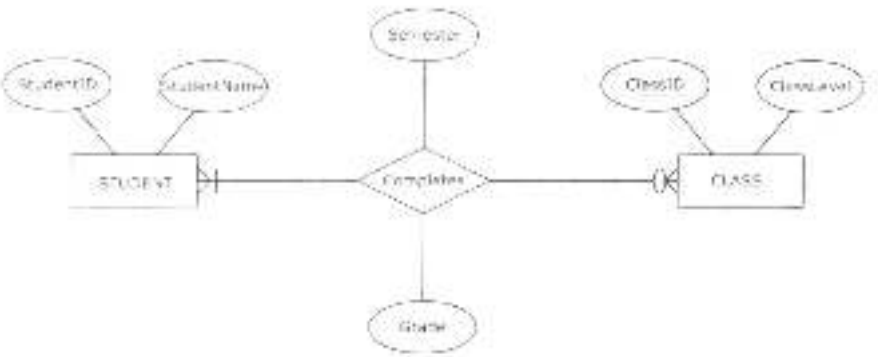


图 2-43　一个描述学生完成课程的 M ：N 联系的 ER 图

图 2-44 展示了图 2-43 中联系的几个可能的实例。

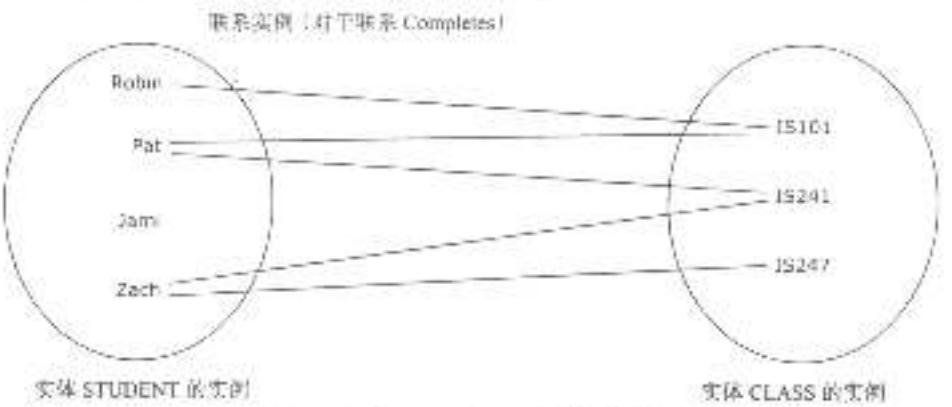


图 2-44　图 2-43 中 M ：N 联系的实例

如果我们在上面的需求中增加一条很小但很重要的需求，这将会彻底改变 M ：N 联系的性质：

- 一名学生可以多次选修同一门课程（例如，如果一名学生该课程的成绩低于最低成绩

要求，他准备重修该课程，直到达到最低成绩要求）。

这条增加的需求，可允许实体的相同实例之间的联系具有多个联系实例，如图2-45所示。Pat选修了课程IS101一次。

40

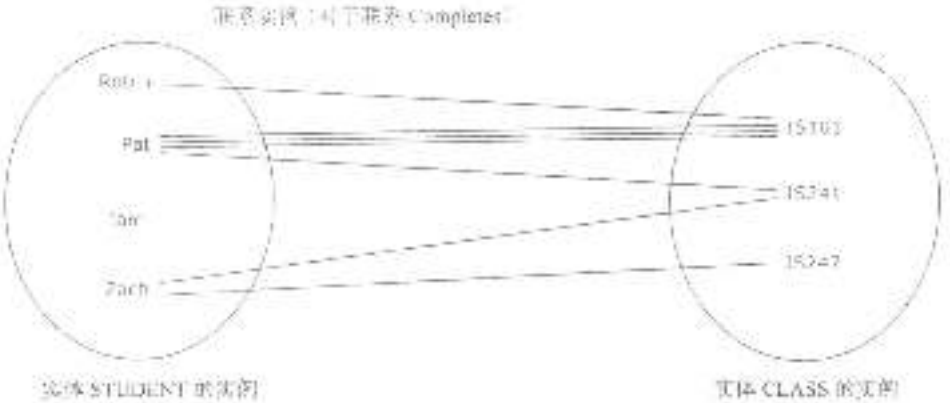


图2-45 针对新增需求的M：N联系Completes的实例

这种扩展的需求不能用一个M：N联系描述。如图2-46所示，我们将使用有两个主实体的弱实体。

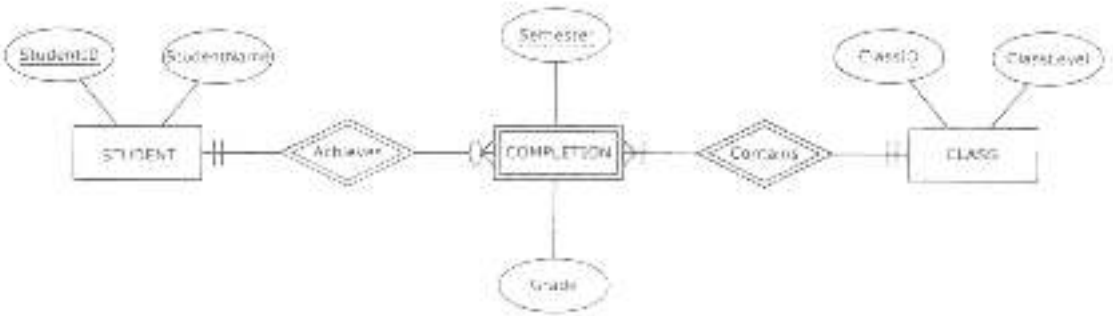


图2-46 一个M：N联系表示为一个弱实体的ER图

我们使用弱实体的原因是使属性Semester成为一个部分标识符。这样，即使某学生多次选修一门课程的成绩相同，也能通过属性Semester区分同一学生多次选修的同一门课程。例如，假设一个学生多次选择一门课程，每次的成绩都是D。最终，他重选该课程并且以成绩C通过了该课程。

41

当使用弱实体表示一个M：N联系时，弱实体的基数约束总是强制为1，M：N联系的基数约束被转移到相应的非弱实体上。考虑图2-43和图2-46。在图2-43中，联系STUDENT Completes CLASS与图2-46中的联系STUDENT Achieves COMPLETION相对应，它们都是可选的最大基数为多个的联系。同样，图2-43中的联系CLASS Completed by STUDENT与图2-46中的联系CLASS Contains COMPLETION相对应，它们都是强制性的最大基数为多个的联系。

图2-47展示了另外一个相同实体的多个实例之间具有M：N联系的例子，使用弱实体表示该联系。这个例子的需求如下：

- 一个汽车出租公司想要记录它的车辆和租用车辆的顾客信息。

- 对于每位顾客，将记录他的唯一 ID 和姓名。
- 对于每辆汽车，将记录它的唯一的 VIN 和品牌。
- 一位顾客可以租用多辆汽车，但至少要租用一辆。一辆汽车可以被许多顾客租用，但是也会有车辆从未被租用过。
- 每次一位顾客租用一辆汽车，将记录租车日期、租用时间和租用一天的价格。

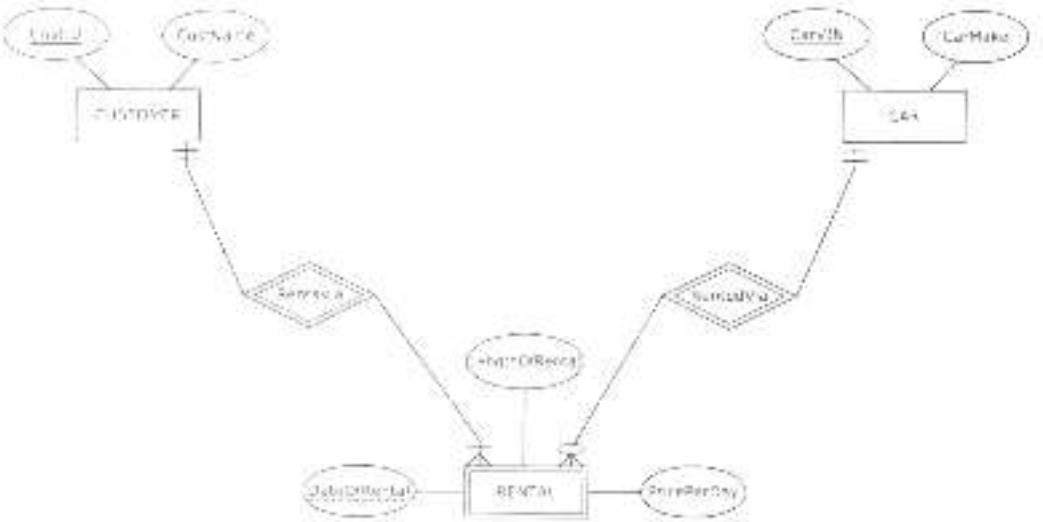


图 2-47　另一个 M：N 联系表示为弱实体的例子

注意，如果我们增加一条很小但很重要的需求：

- 每当一位顾客租用一辆汽车时，这次出租都会有一个唯一的出租 ID。

那么得到的 ER 图应该如图 2-48 所示。

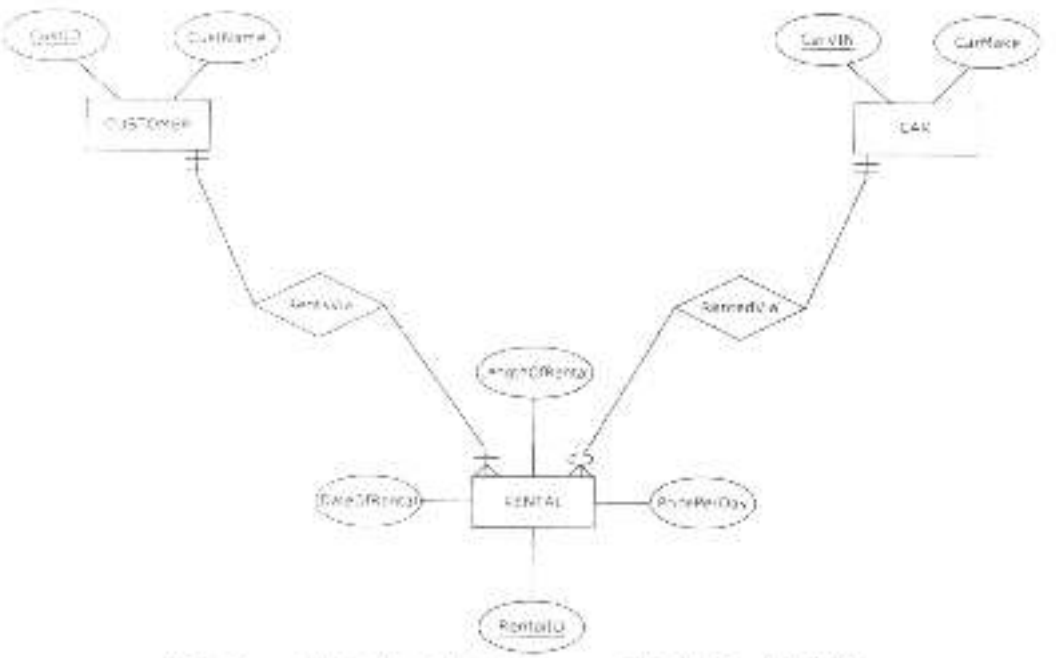


图 2-48　一个普通实体，而不是将 M：N 联系表示成一个弱实体

既然每次出租都有一个唯一属性，那么我们就不必用弱实体表示出租，而是用一个普通实体来表示它。

正如图 2-48 的例子所展示的，在需求中为一个 M ： N 联系（尤其是具有多个属性的 M ： N 联系）增加一个唯一标识属性，将会把这个 M ： N 联系转换成一个普通实体。采用这种方式来简化 ER 图以及后续的结果数据库是一种很常用的技术。 42

2.28　问题说明：关联实体

关联实体（associated entity）是一种概念，适用于描述 M ： N 联系的替代方式之一。关联实体用一个内部有菱形的矩形来表示。关联实体没有唯一或部分唯一的属性，且通常没有任何属性。

图 2-49 展示了一个 M ： N 联系以及其用关联实体替代的形式。在图 2-49 中，上、下两图是相互等价的，且是根据完全相同的需求来设计的。只要联系 AssignedTo 没有属性，则这个相应的关联实体 ASSIGNMENT 也没有属性。 43

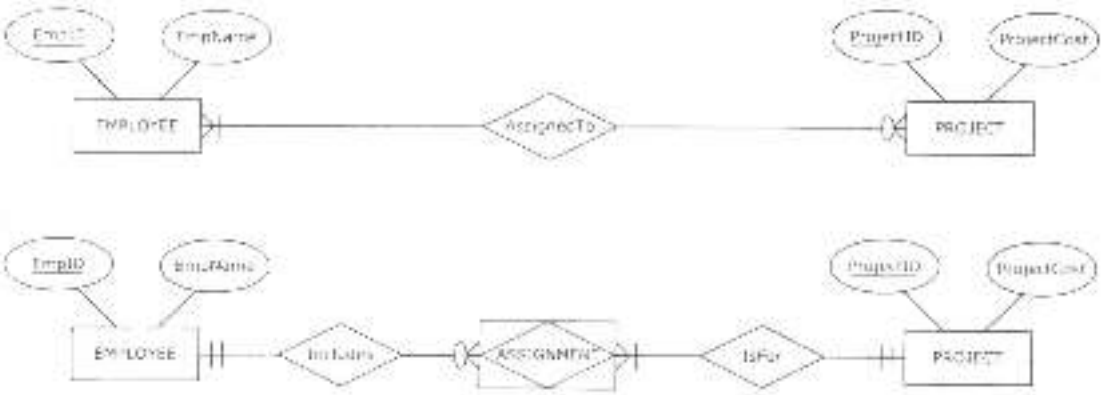


图 2-49　两个等价的联系，既可以表示成 M ： N 联系也可以表示成关联实体

在图 2-49 的上图中，实体 EMPLOYEE 是可选择性参与到联系 AssignedTo 中的。相应地，在图 2-49 下图所示与其等价的关联实体 ASSIGNMENT 中，实体 EMPLOYEE 也是可选择性参与到联系 Includes 中的。另一方面，在图 2-49 上图中，实体 PROJECT 是强制参与到联系 AssignedTo 中的。相应地，在图 2-49 下图所示与其等价的关联实体 ASSIGNMENT 中，实体 PROJECT 也是强制参与到联系 IsFor 中的。同时需要注意的是，关联实体自身的基数约束在两种联系中都是强制唯一的（一般来说这是关联实体的问题）。

图 2-50 展示了一个一元 M ： N 联系以及其用关联实体表示的替代形式。在图 2-50 中，上、下两图是相互等价的，且是根据完全相同的需求设计的。

图 2-51 展示的是带一个属性的 M ： N 联系以及其用关联实体表示的替代形式。在图 2-51 中，上、下两图是相互等价的，且是根据完全相同的需求设计的。在上图中，只要联系 SoldVia 有一个属性 NoOfItems，则在相应的下图中，其等价的关联实体 LINEITEM 同样有一个属性 NoOfItems。每一次销售事务可以包含多个不同数量的产品，如上图所示。因此，在下图中，每一次销售事务可以有多个项目行。每一个项目行代表在一次销售事务中销售了某种产品及其销售数量。 44

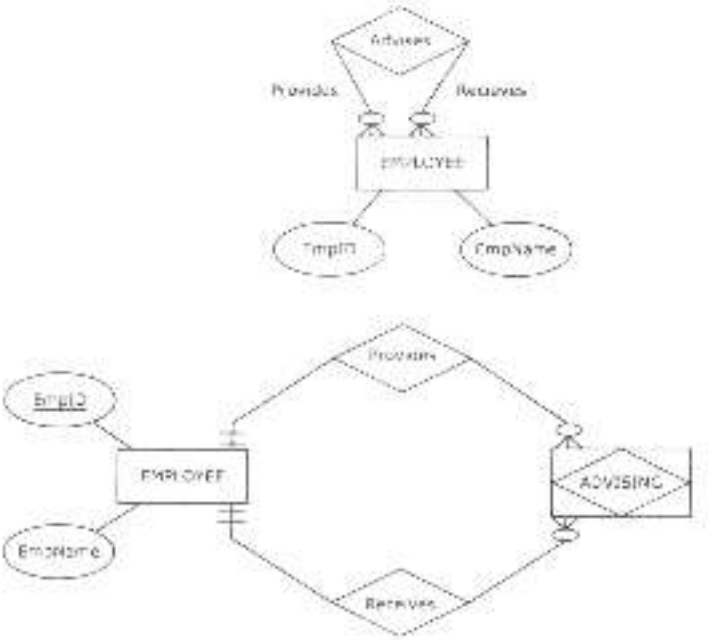


图 2-50　两个等价的联系，既可以表示成一元 M ：N 联系也可以表示成关联实体

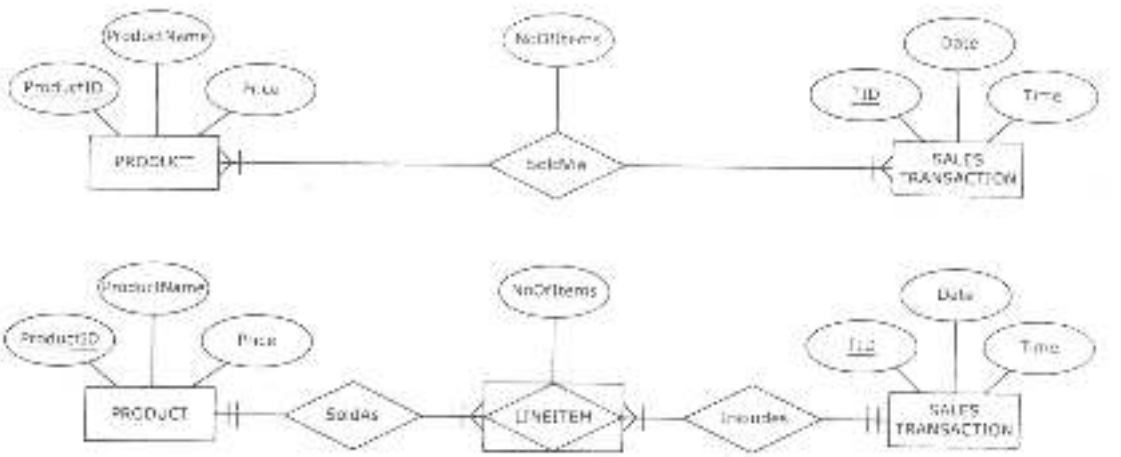


图 2-51　两个等价的联系，既可以表示成有一个属性的 M ：N 联系也可以表示成关联实体

关联实体并不是描述二元或一元联系的必需结构。正如上面的例子所展示的那样，对于二元或一元联系而言，关联实体仅仅是另一种描述联系的方式。若不用关联实体，同样可以很容易地描述这种联系。但正如下一小节所述，对那些度大于 2 的联系（如三元联系）而言，关联实体则提供了一种消除 ER 图中潜在歧义的方法。

2.29　问题说明：三元（及更高阶）联系

一个度为 3 的联系，包含 3 个实体，也称为**三元联系**（ternary relationship）。本书将利用下面的例子来展示三元联系。

MG (Manufacturing Guru) 公司想要记录它的供应商、零部件和产品。

特别地，MG 公司想要记录哪家供应商提供了哪些零部件给哪类产品。在需求收集的过程中，MG 提供了下列具体需求：

- 我公司有多类产品。
- 我公司有多家供应商。
- 我公司有多类零部件。
- 我公司想要记录哪家供应商提供了哪些零部件给哪类产品。
- 每类产品包含一个或多个零部件，每一零部件有一家或多家供应商供应。
- 每一家供应商可以提供多种零部件给多类产品，但是也可不提供任何零部件给任何产品。
- 每一类零部件可以被一家或多家供应商提供给一类或多类产品。

简单地在这三个实体之间创建三个二元联系并不能完整地描述以上需求。图 2-52 展示了三个联系，分别表示：哪类产品与哪些供应商有关联、哪家供应商与哪些零部件有关联、哪种零部件与哪些产品有关联（为简单起见，这部分中所有实体都没有带属性）。但是，基于图 2-52 创建的数据库并不能记录哪家供应商提供了哪些零部件给哪类产品。

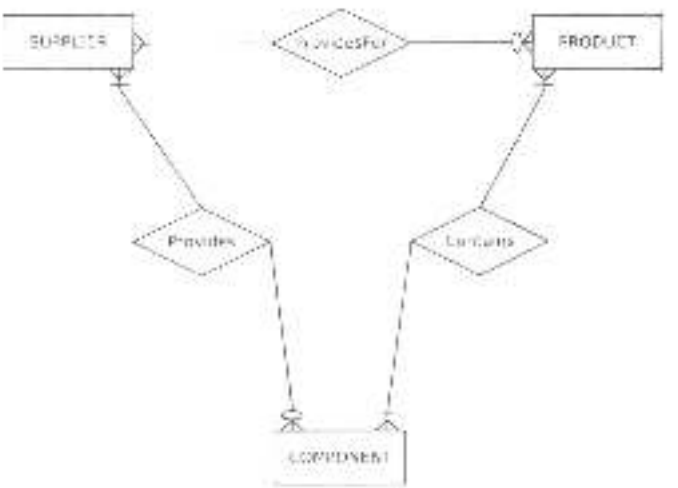


图 2-52　三个二元联系不足以描述 MG 公司的需求

比如，基于图 2-52 创建的数据库，我们可以描述供应商 A1 和 A2 是零部件 B 的供应方，同时，零部件 B 是产品 C 的零部件之一。但是，假设供应商 A1 为产品 C 供应零部件 B，供应商 A2 没有为产品 C 供应零部件 B（即供应商 A2 供应零部件 B，但是不为产品 C 提供零部件 B），则以图 2-52 为基础建立的数据库就不能描述这样一个场景。

为了涵盖所有可能的合理应用场景，所有三个实体都必须统一集中于一种联系，这样就得到了一个三元联系。

三元联系的一个棘手问题是在某一种三元联系中，不能明确地表示基数限制。图 2-53 中展示了一种包含实体 SUPPLIER、PRODUCT 和

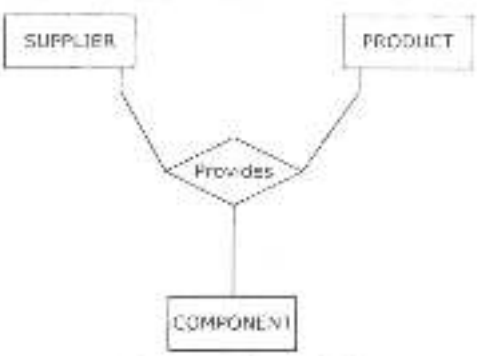


图 2-53　一个三元联系

COMPONENT 的三元联系 Provides，图中没有给出基数限制的原因是无法毫无歧义地标明基数限制。

在图 2-53 中，对那些没有为任何产品提供任何零部件的供应商，图中并没有清楚表示我们可以在什么地方放置记录这些供应商的符号。若只是在联系的零部件一侧设置一个可选符号，这样虽然简洁但依然存在歧义：究竟是希望记录那些不为任何产品提供任何零部件的供应商，还是希望记录那些没有任何供应商为其提供任何零部件的产品。

换句话说，这种表示方式根本不能清楚地描述一个联系的基数。但是，如图 2-54 所示，如果用关联实体来替代的话，就可以清楚地描述三元联系的所有可能的基数限制了。

46

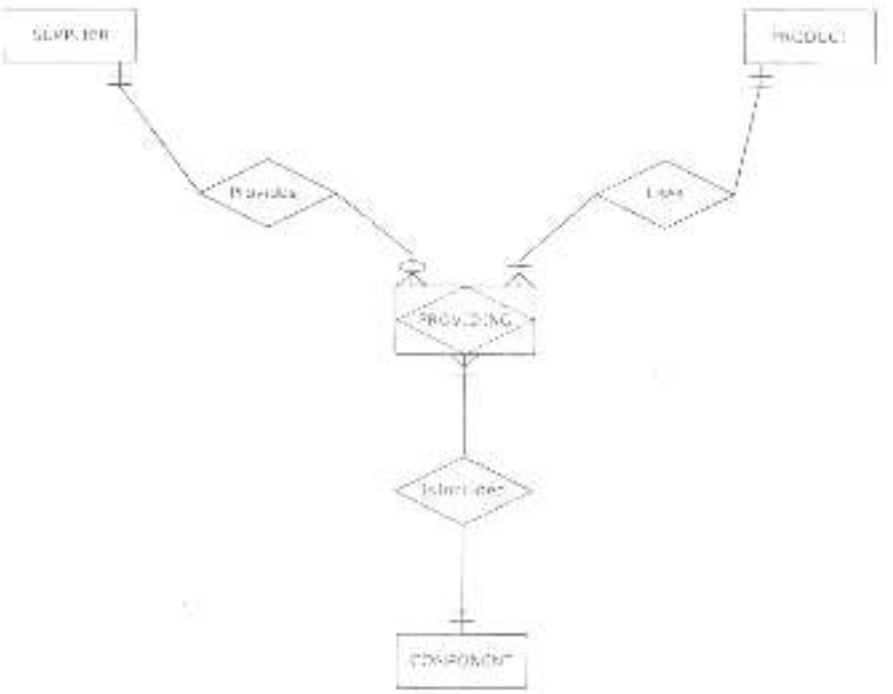


图 2-54　通过关联实体建立的三元联系

注意，图 2-54 描述了哪家供应商为哪类产品提供了哪些零部件。如果我们要为供应商记录下产品零部件的实际交付量，则需要增加额外的属性，如数量。但是，在图 2-54 所示的关联实体中仅简单地为供应商增加数量（quantity）这一属性也无法解决上述问题，因为同一家供应商可以不止一次地为同一产品提供同等数量的同一零部件。如果只增加数量这个属性，就没有办法唯一区分同一家供应商为同一产品提供等数量的同一零部件的不同实例。对这一情况的解决办法之一是创建另一属性作为一个唯一交付标识。如果给这些需求增加一项规定，对于每一供应商为某一特定产品供应特定零部件，都认为是一个单独的个体交付并且有自己独特的交付标识，这样一来，就不需要用关联实体来表示了，而是可以用图 2-55 所示的常规实体来表示。

47

二元联系中还需要注意的是，除了少数例外情况，三元联系适用于绝大多数像多对多对多（many-to-many-to-many）这样的联系。图 2-56 展示了一个多对多对一（many-to-many-one）的三元联系。

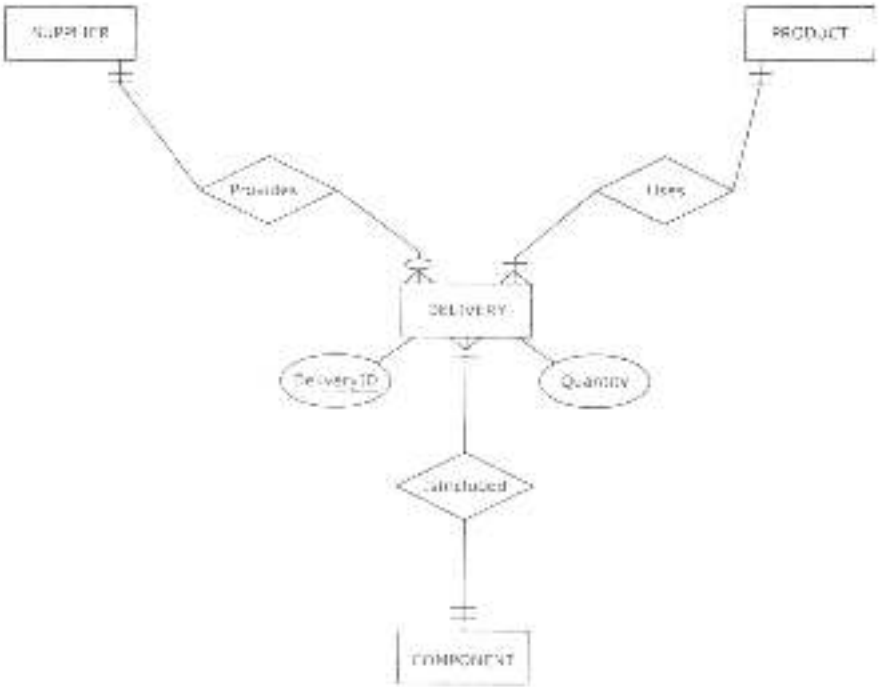


图 2-55　普通实体代替三元联系

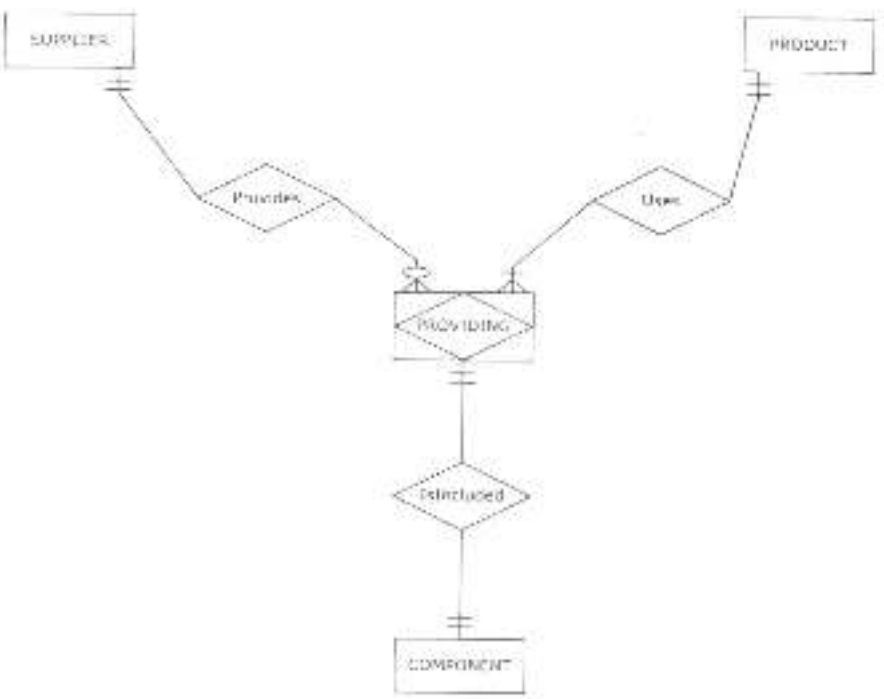


图 2-56　一个多对多对一的三元联系

图 2-56 表示每一零部件都是由某一家供应商专门为某一产品供应的。因此，此图可以简化成如图 2-57，从而消除了转化成三元关系的必要性。

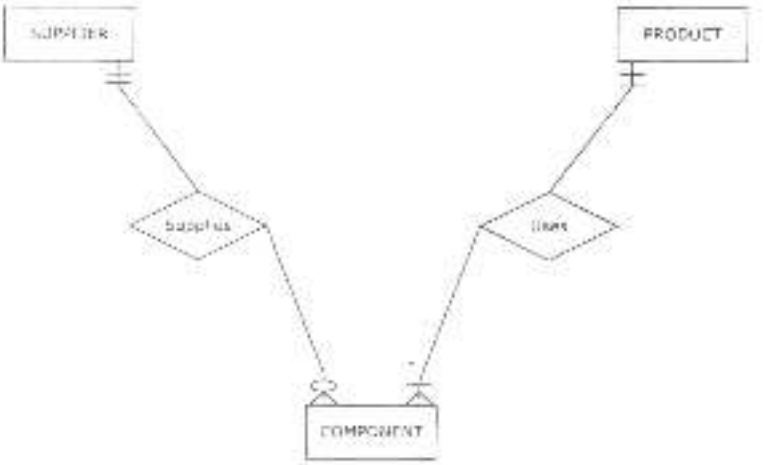


图 2-57　一个被消除了的多对多对一的三元联系

但是，值得注意的是，若想记录实际的各种交易及其数量，还是不得不使用图 2-55，除非联系 IsIncluded 是 1 : 1 的。

在实际中，三元联系比较少见，度大于 3 的联系更为少见。在多数情况下，当设计者试图创建度大于 2 的联系时，往往会尝试创建额外的实体而非联系。

48

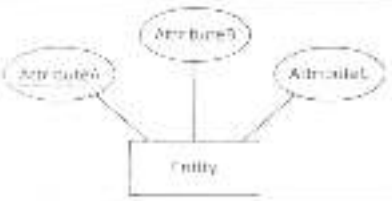
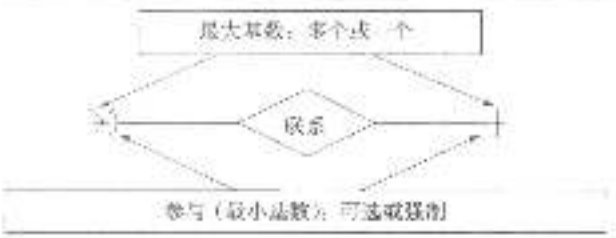
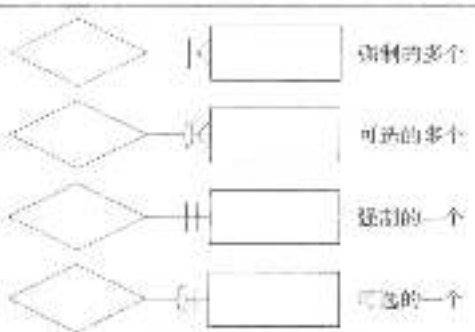
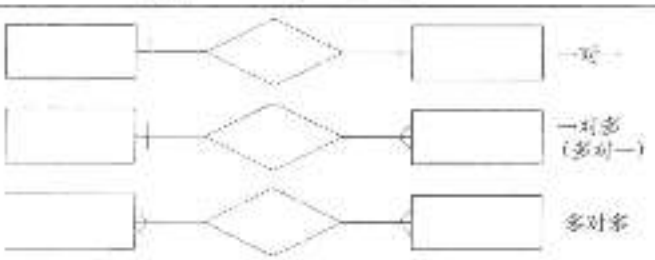
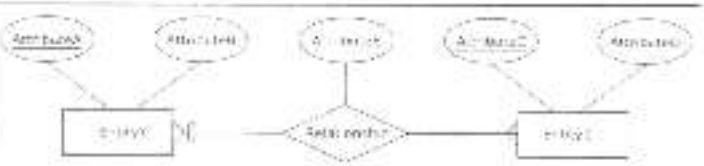
总结

表 2-1 总结了本章介绍的基本 ER 模型概念。

表 2-1　基本 ER 模型概念总结

概念	图形表示
一般属性	Attribute
唯一属性	Attribute
复合属性	AttributeA　AttributeB　AttributeC　Attribute1
复合的唯一属性	AttributeA　AttributeB　AttributeC　(Attribute1)
多值属性	Attribute
派生属性	Attribute

（续）

概念	图形表示
可选属性	
带有属性的实体 注：每个实体必须至少有一个唯一属性 在一个实体内，每个属性名不同 在一个 ER 图内，每个实体名不同	
联系	
基数约束	
四种可能的基数约束	
三种类型的联系（最大基数侧）	
带有一个属性的联系 注：可适用于多对多联系	

49
-
50

（续）

概念	图形表示
具有确定最小基数和最大基数的联系	
一元联系 联系的度：1（涉及一个实体）	
二元联系 联系的度：2（涉及两个实体）	
三元联系 联系的度：3（涉及三个实体） 注：多用作多对多对多联系 三元联系较少见（度高于3的联系更少见）	
弱实体 注：涉及与其主实体的三标识性联系（1：M或1：1）关联起来 如果标识性联系是1：M的，那么弱实体必须有一个部分性一的属性	
关联实体 注：描述M：N联系的另一种方法，特别适用于描述一元联系	

31

关键术语

associative entity（关联实体），43
attribute (of an entity)（（实体的）属性），14
binary relationship（二元联系），28
candidate key（候选码），24
cardinality constraint（基数约束），15
composite attribute（复合属性），22
composite unique attribute（复合的唯一属性），23
database requirement（数据库需求），13
degree of a relationship（联系的度），28
derived attribute（派生属性），25
enhanced ER（EER，扩展的ER），40
entity（实体），13
entity instance/ entity member（实体实例/实体成员），14
entity-relationship/ER modeling（实体-联系建模/ER建模），13
ER diagram（ERD，ER图），13
exact minimum cardinality and/or maximum cardinality（最小基数和/或最大基数确切值），27
identifying relationship（标识性联系），30
many-to-many relationship/M：N relationship（多

对多联系），17
maximum cardinality（最大基数），15
minimum cardinality/participation（最小基数/参与），15
multivalued attribute（多值属性），25
one-to-many relationship/1 : M relationship（一对多联系），17
one to one relationship/1 : 1 relationship（一对一联系），17
optional attribute（可选属性），26
owner entity（主实体），30
partial key（部分码），31
relationship（联系），15
relationship attribute（联系属性），19
relationship instance（联系实例），18
relationship role（联系的角色），29
ternary relationship（三元联系），45
unary relationship/recursive relationship（一元联系/递归联系），28
unique attribute（唯一属性），14
weak entity（弱实体），30

复习题

Q2.1 ER建模的目的是什么？
Q2.2 ER模型的基础结构是什么？
Q2.3 唯一标识是什么？
Q2.4 基数限制的定义是什么？
Q2.5 四种可能的基数限制是什么？
Q2.6 三种基数的类型是什么（最大基数除外）？
Q2.7 复合属性是什么？
Q2.8 派生码是什么？
Q2.9 多值属性是什么？
Q2.10 派生属性是什么？
Q2.11 可选属性是什么？
Q2.12 最小基数和最大基数这两个限制在联系里是如何定义的？
Q2.13 二元联系是什么？
Q2.14 一元联系是什么？
Q2.15 弱实体是什么？
Q2.16 关联实体是什么？
Q2.17 三元联系是什么？

52

练习

E2.1 建立一个有多个属性的实体实例。
E2.2 为一个有两个实体（都包含多个属性）的应用场景创建需求和ER图，需要包含以下联系：
E2.2a 1 : M联系，参与要求是在1这方是强制性的而在M这方是可选择的参与。
E2.2b 1 : M联系，参与要求是在1这方是可选择的而在M这方是强制性的参与。
E2.2c 1 : M联系，参与要求是在两方都是强制性的参与。
E2.2d 1 : M联系，参与要求是在两方都是可选择的参与。
E2.2e M : N联系，参与要求是在一方是强制性的而在另一方是可选择的参与。
E2.2f M : N联系，参与要求是在两方都是强制性的参与。
E2.2g M : N联系，参与要求是在两方都是可选择的参与。
E2.2h 1 : 1联系，参与要求是在一方是强制性的而在另一方是可选择的参与。
E2.2i 1 : 1联系，参与要求是在两方都是强制性的参与。

E2.2　下一个练习，学习如何在两方都是可选的练习。

E2.3　为一个由两个实体（都包含多个属性）的应用场景创建概念和 ER 图，并为这些属性与多对多关系。

E2.4　创建一个有复合属性的实体实例。

E2.5　创建一个不具有任何属性的实体实例。

E2.6　创建一个在模式时，表示唯一属性的实体实例。

E2.7　创建一个具有多值属性的实体实例。

E2.8　创建一个具有派生属性的实体实例。

E2.9　创建一个有可选属性的实体实例。

E2.10　为一个有两个实体（都包含多个属性）的应用场景创建概念和 ER 图，其中两个实体相关联，一个有四个属性的实体与另一个实体之间的一对多关系。

E2.11　为一个包含一个弱实体的实体的应用场景创建概念和物理 ER 图。

E2.12　为一个包含四个实体（均包含多个属性）的应用场景创建概念 ER 图，两个实体彼此间有多对多关系。

E2.13　为一个有两个实体（都包含多个属性）的应用场景创建概念和 ER 图，要求在第一个实体中有一个是弱实体，另一个是共享主实体。

小案例

小案例 1　Investco Scout

Investco Scout 是一家投资研究公司，正为其客户数据库创建 ER 图，需求如下：

- Investco Scout 跟踪各种不同的证券投资公司，并管理所其可供交易证券在其可供交易中的证券。
- 对于每一家投资公司，Investco Scout 系统会记录该公司的唯一标识符和唯一的投资公司名称，以及记录该投资公司在不同地区下的名称。
- 对于每一种共同基金，Investco Scout 系统会记录共同基金的唯一标识符，并会记录共同基金的名称及其成立日期。
- 对于每一种证券，Investco Scout 系统会记录一个唯一的证券标识符，还会记录证券的名称和它的类型。
- 投资公司可以管理多种共同基金，Investco Scout 系统会记录当前管理过任何共同基金的投资公司列表，一个特定共同基金只由一家投资公司管理。
- 一项共同基金可以包含一种或多种证券，一种证券可以被多个共同基金包含。Investco Scout 会记录没有被任何共同基金包含的证券。
- 对于每一个包含在共同基金里的证券实例，Investco Scout 会记录正被包含的数量。

小案例 2　Funky Bizz

Funky Bizz 提供娱乐服务，主要是租赁乐器给乐队。试为 Funky Bizz 的运营数据库创建一个 ER 图，满足以下需求：

- Funky Bizz 的交易记录需要记录乐器、乐队、连接技术人员以及演出。
- 对于每一件乐器，Funky Bizz 需要记录唯一的乐器序列号、乐器型号、品牌、乐器制造年份以及乐器目前的每小时租赁价格。
- Funky Bizz 的客户是乐队，对于每支乐队，Funky Bizz 系统需要记录乐队的名称和唯一的乐队标识符、乐队记录的乐队地址、乐队联系人及各种电话号码。
- 连接技术人员是通过外部机构来雇佣，对于每一位技师，需要记录一个唯一的 SSN 以及姓名，此

社、电话号码。

- Funky Bizz 也要记录客户乐队的演出信息。对于每一场演出，需要记录由演出场地和日期组成的唯一标识符，还要记录演出的类型和名字（一场演出可以有名字也可以没有）。
- 一支乐队可以不仅使任何乐器，也可能拥有20个乐器。一件乐器可以被一支乐队拥有或者不被拥有。
- 一名维修技术人员可以维护许多乐器或者不维护任何乐器；同样，一件乐器可以被多名技师维护或者没有被任何技师维护。
- 一支乐队可以在很多场演出中表演，但不必在每一场演出中都有表演。每一场演出至少有一支乐队表演，但确实会有多支乐队。
- 对于每一支乐队，Funky Bizz 记录每一支乐队的表演次数。

小案例 3 Snooty Fashions

Snooty Fashions 提供独家定制的时装设计服务。试为 Snooty Fashions 的运营数据库创建 ER 图，满足以下需求：

- 对于每一位设计师：有唯一的设计师标识符和 SSN，还要包含其姓名（全名）。
- 对于每一位顾客：有唯一的顾客标识符、姓名和各种电话号。
- 对于每一位裁缝技师：有唯一的 SSN、姓名（全名）。
- 对于每一套搭配好的服饰：有唯一的标识符、完成日期和价格。
- 对于每一场时装表演：有唯一的标识符、日期和地点。
- 一位设计师可以设计很多套服饰，但每套服饰只有一位设计师。
- 一套服饰售（预售）给一位顾客。顾客可以买一套或多套服饰。（Snooty Fashions 不会记录没有购买任何服饰的顾客）。
- 每位裁缝技师需要至少给一套服饰做剪裁，也可以为多套服饰做剪裁。每一套服饰有至少一位裁缝技师为其剪裁，也可以有多位。
- Snooty Fashions 会记录每位裁缝技师为某一套服饰剪裁的开始时间。
- 每位设计师可以参加很多场的时装表演，也可以不参加任何一场时装表演。每场时装表演可以以一位或两位 Snooty Fashions 设计师为主角（Snooty Fashions 不会记录没有以 Snooty Fashions 设计师为主角的时装表演）。

小案例 4 Signum Libri

Signum Libri（SL）是一家出版公司。试为 SL 运营数据库创建 ER 图，满足下列需求：

- 对于每一本由 SL 出版的书：有书名、流派类型、出版日期、页数。
- 对于每一位作者：有唯一的作者标识符及作者姓名。
- 对于每一家代理商：有唯一的代理商标识符及其姓名。
- 对于每一位编辑：有唯一的编辑标识符及其姓名。
- 每本书由一位作者撰写，一位作者可以撰写多本书。SL 不会记录没有为 SL 写书的作者。同一作者写的不同的书有不同的书名。但是，不同的作者可以写书名相同的两本不同的书。
- 每位作者由一家代理商代理。每家代理商至少代理一位作者，也可以代理多位作者。
- 一本书有一位编辑。一位编辑至少负责编辑一本书，也可以是多本。
- 每位编辑可以指导一位或多位编辑，但是不可以指导任意一位编辑。每位编辑可以至多有一位导师编辑，但也可以没有。

小案例 5 ExoProtect

ExoProtect 是一家保险公司。试写出如图 2-58 所示 ExoProtect 的工电脑数据库 ER 图的所有需求：

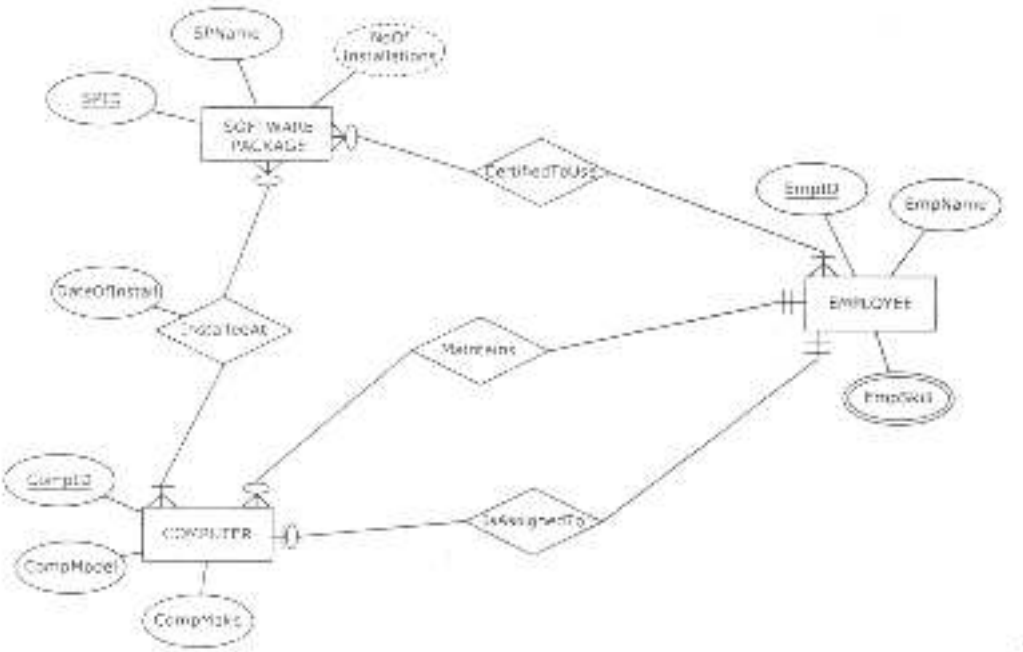


图 2-58　ExcProtect 的工电脑数据库 ER 图

小案例 6　Jone Dozers

Jone Dozers 是一家建筑设备公司。试写出如图 2-59 所示 Jone Dozers 的销售和租赁数据库 ER 图的所有需求。

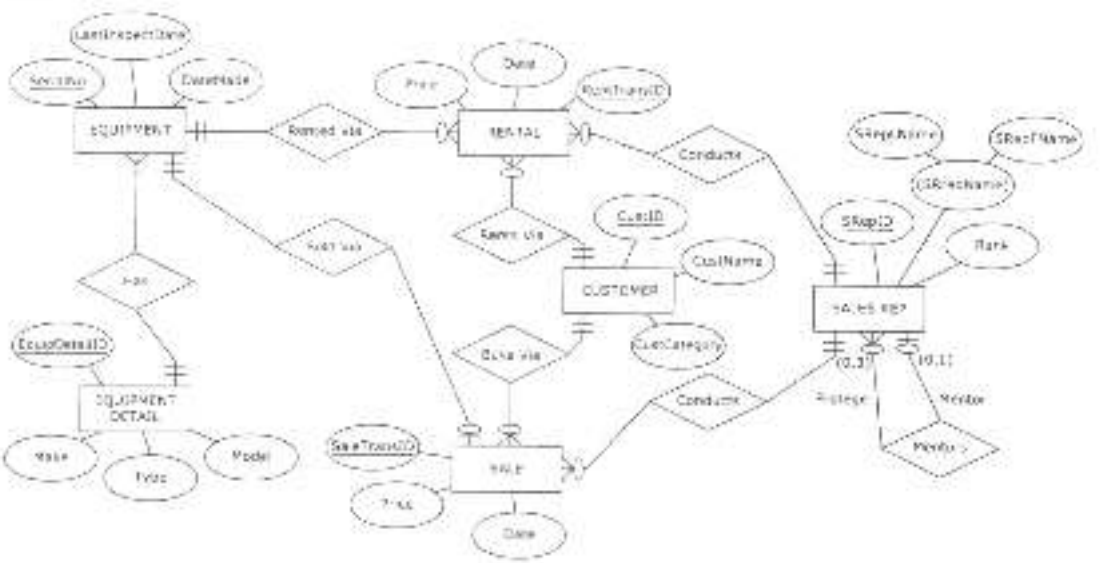


图 2-59　Jone Dozers 的销售和租赁数据库 ER 图

小案例 7　Midtown Memorial

Midtown Memorial 是一家医院。试写出如图 2-60 所示 Midtown Memorial 医院的病人药品分发数据库 ER 图的所有需求。

54 / 55

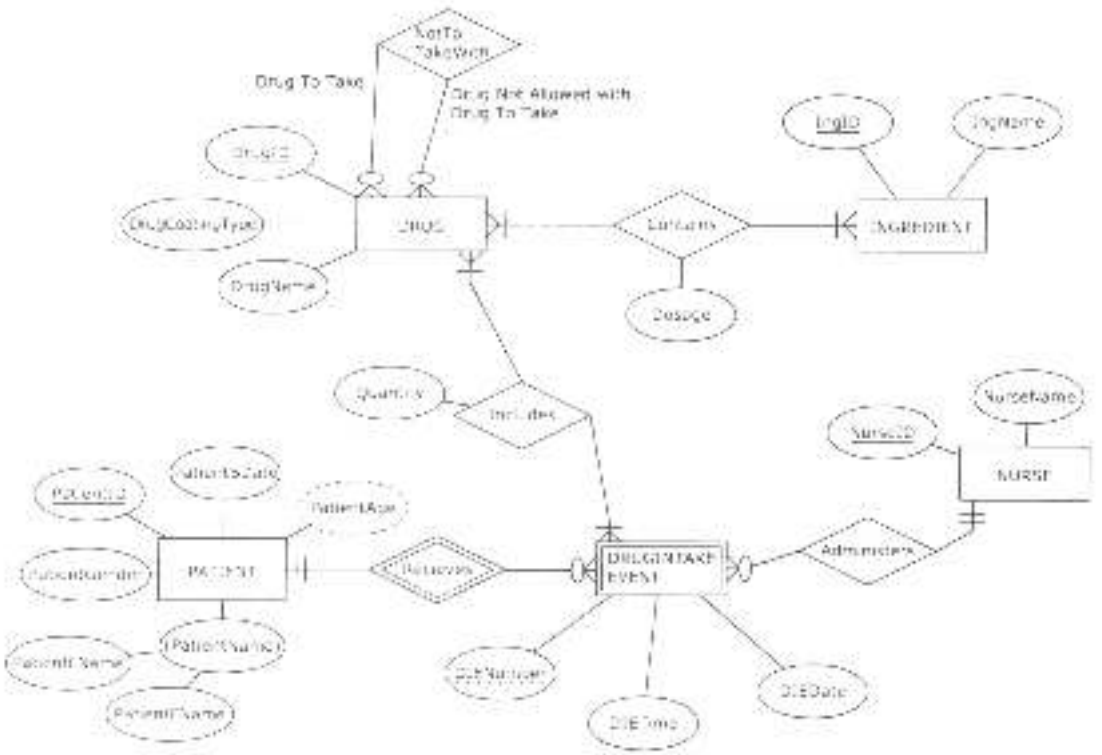


图 2-60 Midtown Memorial 医院的病人药品分发数据库 ER 图

第 3 章

Database Systems: Introduction to Databases and Data Warehouses

关系数据库建模

3.1 引言

在第 1 章中我们曾用“逻辑数据库模型”这一术语来特指 DBMS 软件所采用的数据库模型。其中最常用的逻辑数据库模型是**关系数据库模型**。采用关系数据库模型建模的数据库称作**关系数据库**。

在创建关系数据库的过程中，一旦数据库的需求以 ER 图的形式被收集并可视化，接下来的工作就是将 ER 图转化为一个逻辑数据库模型，即**关系模式**（relational schema）。关系模式是对关系数据库模型进行可视化描述的关系图表。

目前大多数商业 DBMS 软件包，比如 Oracle、MySQL、Microsoft SQL server、PostgreSQL、Teradata、IBM DB2 以及 Microsoft Access，都是**关系数据库管理系统**（relational DBMS，RDBMS）软件包，它们都采用关系数据库模型并用于实现关系型数据库。

在本章中，我们将讲述关系数据库模型的基本概念，以及如何将 ER 图转化为关系模式。

3.2 关系数据库模型基本概念

关系数据库模型中，一个重要的概念是**关系**（relation）。一个关系在关系数据库中的存在形式是包含行与列的一张表。而一个关系数据库就是一系列具有不同名称的相关关系的集合。

一个关系有时也称作一张**关系表**（relation table），或简称为**表**（table）。表中的一列（column）有时被称为一个**域**（field）或一种**属性**（attribute）。表中的一行（row）有时被称为一个**元组**（tuple）或**记录**（record）。关系数据库中各术语的同义词在表 3-1 中给出。

如表 3-1 所示，我们常将一种关系称作一张表，但我们要牢记：尽管所有的关系都是表，但并非所有的表都是关系。一张表如果能够称作一个关系，它需要满足如下条件：

表 3-1 关系数据库模型所使用的同义词

关系	=	关系表	=	表
列	=	属性	=	域
行	=	元组	=	记录

1. 每一列必须有一个名称，在同一张表中，不能有相同的列名。
2. 在同一张表中，不能有完全相同的行。
3. 在每一行中，属于任何一列的值必须为单值，表中任一行中不允许出现多值。
4. 每一列中的值必须属于同一个（预定义的）域。

我们将在图 3-1 的例子中说明这些规则。在该图中，我们采用两张表来记录雇员信息，并假设在记录雇员信息的表中有如下一些预定义的域：

雇员 ID：4 位数字。

雇员姓名：20 个字符以内。

雇员性别：字符 M 或 F。

雇员电话号码：字符串（供区号用）形式，二位数字。
雇员出生日期：日期，日、月、年。

关系型（关系）

EmpID	EmpName	EmpGender	EmpPhone	EmpBdate
0001	Joe	M	x234	1/11/1985
0002	Sue	F	x345	2/7/1983
0003	Amy	F	x456	4/4/1980
0004	Pat	F	x567	3/8/1971
0005	Mike	M	x678	5/5/1965

非关系表

EmpID	EmpInfo	EmpInfo	EmpPhone	EmpBdate
0001	Joe	M	x234	1/11/1985
0002	Sue	F	x345	2/7/1983
0001	Joe	M	x234	1/11/1985
0004	Pat	F	x567, x789	2/8/1971
0005	Mike	M	x678	a long time ago

图3-1 关系以及非关系表的实例

图3-1中位于上部的表就是一张满足以上条件的合格关系表：没有同名的列，没有相同的行。每一行中的属性值都是单值，每一列中的属性值都属于相同的域。

而图3-1中位于下部的表则是一张非关系表。该表违背了前述提及的所有条件：有两列同名，有两行完全相同，有一个记录中雇员电话号码这一属性的取值是多值，且有一行中雇员出生日期的取值不满足预定义的域。

注意：图3-1中的两张表均可由MS Excel等电子制表工具来实现，但仅上面的那张表可以由RDBMS来实现。RDBMS不允许实现下面的那张表，当试图实现下面那张表时，RDBMS将会显示违规的错误提示。

除了前述提及的四项关系表条件之外，关系表还具有两个额外的属性：

5. 列顺序无关。

6. 行顺序无关。

这两个属性告诉我们，无论关系表中的行与列如何排序，表中都将包含同样的信息，即都会被视作同一张表。当某张表中的所有行与列都被重新排序后，每一行仍然包含着同样的信息（只是顺序不同而已），且表中仍然包含着同样的行（只是顺序不同而已）。图3-2中的例子说明了这个问题，它显示了两张行列顺序不同的表，这两张表其实是相同的表。

一个关系

EmpID	EmpName	EmpGender	EmpPhone	EmpBdate
0001	Joe	M	x234	1/11/1985
0002	Sue	F	x345	2/7/1983
0003	Amy	F	x456	4/4/1980
0004	Pat	F	x567	3/8/1971
0005	Mike	M	x678	5/5/1965

图3-2 行和列出现顺序不同的关系表实例

司一个关系（与行和列的顺序无关）

EmpName	EmpID	EmpGender	EmpBdate	EmpPhone
Joe	0001	M	1/11/1985	x234
Amy	0003	F	8/4/1990	x456
Sue	0002	F	2/7/1983	x345
Pat	0004	F	3/8/1971	x567
Mike	0005	M	5/5/1965	x678

图 3-2（续）

3.3　主码

每一个关系都有一列（在一些情况下允许多列）作为**主码**（primary key）。其目的是在关系中区分不同的行。以下是主码的定义。

主码

每一个关系必须有一个主码，对于每一行来说，主码是取值不同的一列（或几列的集合）。

在图 3-3 所示的雇员关系中，EmpID 列可作为主码。注意，主码名称下面有一条下划线，以将主码和其他列区分开来。

EMPLOYEE

EmpID	EmpName	EmpGender	EmpPhone	EmpBdate
0001	Joe	M	x234	1/11/1985
0002	Sue	F	x345	2/7/1983
0003	Amy	F	x456	8/4/1990
0004	Pat	F	x567	3/8/1971
0005	Mike	M	x678	5/5/1965
0012	Mike	M	x666	8/1/1974
0007	Barbara	F	x777	4/5/1980
0011	Von	M	x777	3/4/1981
0009	Amy	F	x777	1/11/1985

图 3-3　带下划线主码的关系表

在图 3-3 的例子中，每一个雇员具有独一无二的 EmpID 取值。而这个关系中的其他列不能作为主码，这是因为，不同的雇员可能会拥有相同的姓名、相同的性别、相同的出生日期以及相同的电话号码。

3.4　将实体映射为关系

如前所述，一旦 ER 图建立，接下来的工作就是将其映射为一系列的关系。这一过程始于将实体映射为关系，即每一个常规实体成为一个关系，每一个常规实体中的常规属性成为新创建的关系中的一列。如果一个实体中有一个单值的且唯一标识的属性，那么这个属性在最终映射得到的关系中就可作为主码[5]。

---

[5] 本章的后面部分将介绍如何将有多个唯一属性的实体映射为关系。

图3-4中给出了一个实例以说明一个实体如何映射为一个关系。用一个长方形来表示关系，长方形中包含了关系中所有列的列名。注意，其中CustID这一列名下面有下划线，表明CustID是关系CUSTOMER中的主码。

59

一旦映射关系作为关系数据库的一部分被创建，它就可以装载数据。图3-5展示了CUSTOMER关系的样例数据。

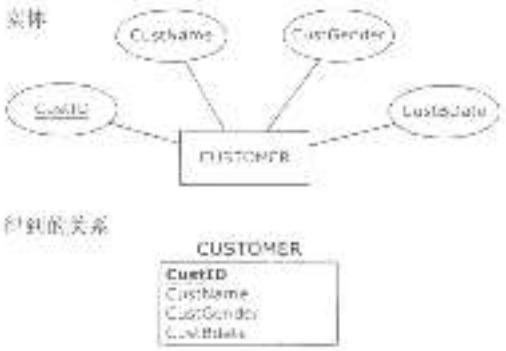


图3-4 将实体映射成关系

CUSTOMER

CustID	CustName	CustGender	CustBdate
1111	Tom	M	1/1/1965
2222	Jenny	F	2/2/1968
3333	Greg	M	1/2/1962
4444	Sophia	F	2/2/1983

图3-5 图3-4所示关系表的样本数据记录

3.5 将具有复合属性的实体映射为关系

如果一个实体包含复合属性，则复合属性中的每一个组成部分都将映射为关系中的一列，而复合属性自身并未显式地出现在映射后的关系中。

图3-6给出了一个例子以说明如何将一个具有复合属性的实体映射为关系。注意，复合属性的名称并未出现在映射得到的关系中。

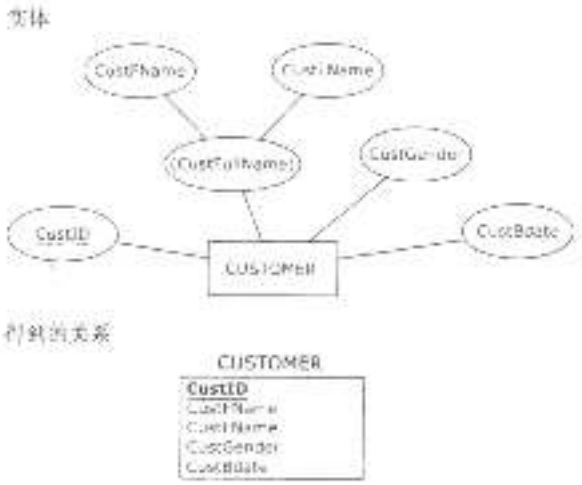


图3-6 将具有复合属性的实体映射为关系

一旦映射关系作为关系数据库的一部分被创建，它就可以装载数据。如图3-7所示。

60

尽管复合属性的名称并没有作为关系模式的一部分，但它仍然可能作为基于关系数据库的前端应用的一部分。举个例子来说，如图3-7所示，包含CUSTOMER关系的数据

库前端应用也许会存在一个复合属性 CustFullName，这个复合属性包含了 CustFName 和 CustLName。图 3-8 说明了如何通过相应数据库的前端应用[⊖]向用户展示关系信息。

CUSTOMER

CustID	CustFName	CustLName	CustGender	CustBdate
1111	Tom	Lendrum	M	1/1/1965
2222	Jenny	Jones	F	2/2/1968
3333	Greg	Newton	M	1/2/1962
4444	Sophia	Danks	F	2/2/1983

图 3-7　图 3-6 所示关系的样本数据记录

CUSTOMER

	CustFullName			
CustID	CustFName	CustLName	CustGender	CustBdate
1111	Tom	Lendrum	M	1/1/1965
2222	Jenny	Jones	F	2/2/1968
3333	Greg	Newton	M	1/2/1962
4444	Sophia	Danks	F	2/2/1983

图 3-8　将图 3-7 所示的关系在终端应用程序中展示给用户

3.6　将具有唯一复合属性的实体映射为关系

第 2 章中曾提到一个实体可能拥有一个具有唯一标识的复合属性。图 3-9 中进一步说明了如何将一个具有唯一标识复合属性的实体映射为一个关系。

图 3-9　将具有唯一复合属性的实体映射为关系

注意，在得到的关系中，Building 列和 RoomNumber 列的名称下面都有下划线，这是因为这两列组合起来形成 CLASSROOM 关系的主码。由多个属性组合起来形成的主码称为复

⊖ 前端应用将在第 6 章进行讨论。

合主码（composite primary key）。图3-10展示了由CLASSROOM关系的样例数据所描述的四个不同教室及其座位容量。注意，对于每一行而言，没有任一个单独的列具有独一无二的取值，但由Building和RoomNumber组成的复合属性却具有唯一的取值。

61

CLASSROOM

Building	RoomNumber	NoOfSeats
Maguire	110	100
Maguire	210	50
Houser	110	50
Houser	210	50

图3-10 图3-9所示关系的样本数据记录

3.7 将具有可选属性的实体映射为关系

第2章中曾提到一个实体可以包含可选属性，可选属性的取值允许为空。当可选属性被映射为关系时，所得结果中的可选列被标记为(O)，如图3-11所示。

一旦图3-11中的关系作为关系数据库的一部分得以实现，它就可以装载数据，如图3-12所示。属性Bonus是可选属性，所以在Bonus这一列中允许属性取值为空。

注意，图3-12雇员关系表的第一行和最后一行记录的Bonus列中都没有属性取值。在数据库术语中，一个未被赋值的记录称为“空”，意为“没有取值”。所以，我们可以将第一行和最后一行的Bonus属性取值称为空值。

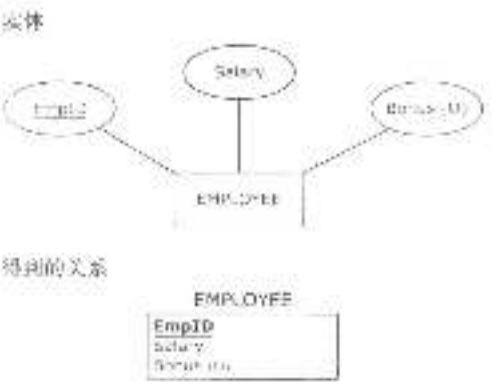


图3-11 将具有可选属性的实体映射为关系

EMPLOYEE

EmpID	Salary	Bonus
1234	$75,000	
2345	$45,000	$10,000
3456	$55,000	$4,000
1324	$70,000	

图3-12 图3-11所示关系的样本数据记录

3.8 实体完整性约束

术语“约束”是指为确保关系数据库的有效性而需满足的各种规则。**实体完整性约束**（entity integrity constraint）则是指“所有的主码属性必须有非空取值”这一关系数据库规则。

实体完整性约束

在一张关系表中，不存在包含空值的主码列。

换言之，实体完整性约束是一种要求没有可选属性存在的主码列规则。每一种RDBMS都需执行这个规则。

举例来说，在图3-13雇员关系中的每一条记录，在EmpID这项属性上都必须有一个非空取值，因为EmpID是该关系中的主码。如果EmpID属性有空值，则将违反完整性约束。

因此，RDBMS 不允许插入的雇员关系记录中 EmpID 属性值为空。

EMPLOYEE

EmpID	Salary	Bonus
1234	$75,000	
2345	$50,000	$10,000
3456	$55,000	$4,000
1324	$70,000	

VALID

EMPLOYEE

EmpID	Salary	Bonus
1234	$75,000	
2345	$50,000	$10,000
*	$55,000	$4,000
1324	$70,000	

INVALID

违反实体完整性约束

图 3-13　实体完整性约束：遵守与违反的例子

同样，在图 3-14 中，每一个在 CLASSROOM 关系中记录的 Building 和 RoomNumber 属性也不能为空，因为这两项属性组合起来构成一个复合主码。

[62]

CLASSROOM

Building	RoomNumber	NoOfSeats
Magnin	110	100
Magnin	210	50
Houser	110	50
Houser	210	50

VALID

CLASSROOM

Building	RoomNumber	NoOfSeats
Magnin	110	100
Magnin	210	50
Houser	*	50
Houser	210	50

INVALID

违反实体完整性约束

图 3-14　实体完整性约束：对一个遵守与违反的例子

3.9　外码

在将 ER 图映射为关系模式的过程中，除了映射实体之外，实体间的联系也是需要映射的。外码（foreign key）正是用来刻画关系数据库模型中实体间联系的一种机制。外码的定义如下：

外码

外码是关系中的一列，这一列又恰好是另外一个关系中的主码。

每当有外码出现在关系模式中时，都会包含一条从外码指向相应主码的连线。

本章中的例子通过关系模式说明了外码的概念，并描述了外码如何应用在关系数据库模型的一对一（1 : 1）、一对多（1 : M）以及多对多（M : N）的关系中。

3.10　将联系映射为关系数据库组件

前面已经介绍了如何将 ER 图映射为关系。接下来将介绍如何将实体间的联系映射为关系模式。

[63]

3.10.1 1：M联系的映射

首先看看一对多联系的映射规则：

由一对多联系中属于M侧的实体所映射得到的关系有一个外码，这个外码对应于由1侧的实体映射得到的关系中的主码。

图3-15中的例子展示了这一规则的应用。在一对多关系ReportsTo中，雇员实体属于M侧而部门实体属于1侧。在将雇员实体映射得到的关系中，有一个外码列DeptID，这一列对应于部门这一关系中的主码。图3-15所示的关系模式中，从雇员关系的外码到部门关系的主码间有一条连线。

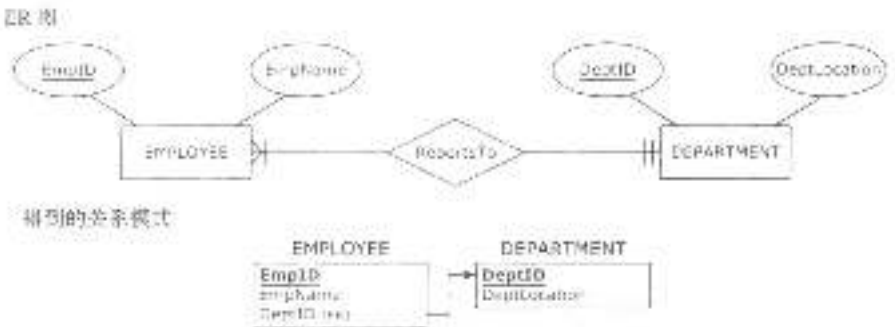


图3-15 一对多联系的映射

图3-15所示的关系模式一旦作为关系数据库得以实现，它就可以装载数据。如图3-16所示。

注意，外码值（雇员关系中DeptID这列的取值）将雇员与他们隶属的部门关联了起来。

我们接下来将讨论可选参与和强制参与对关系数据库中的1侧和M侧的影响。

1侧的强制参与 首先来看看强制参与在1侧的影响。注意，图3-15所示的雇员实体在隶属关系上是强制参与的，则雇员关系中的DeptID外码列要求有非空取值。因此，在图3-16中可以看到，雇员关系中的每一条记录在DeptID列都有一个非空值。

M侧的强制参与 我们再来看看强制参与在M侧的影响。图3-15所示的部门实体在隶属关系中要求强制参与，所以在图3-16中所有的部门都至少有一个雇员与之相对应。换句话说，在部门关系中，不存在没有与之对应的雇员关系的记录。部门（1，SuiteA）对应于雇员（1234，Becky）和（3456，Rob），而部门（2，SuiteB）对应于雇员（2345，Molly）和（1324，Ted）。

EMPLOYEE

EmpID	EmpName	DeptID
1234	Becky	1
2345	Molly	2
3456	Rob	1
1324	Ted	2

DEPARTMENT

DeptID	DeptLocation
1	Suite A
2	Suite B

图3-16 图3-15所示关系数据库的样本数据记录

1侧的可选参与 图3-17反映了可选参与在1侧的影响。在这个例子中，雇员实体在其隶属关系上是可选参与的，因此，雇员关系中的DeptID这一外码是一个可选列。

一旦图3-17所示的关系模式作为关系数据库实现，它就可以加载数据，如图3-18所示。

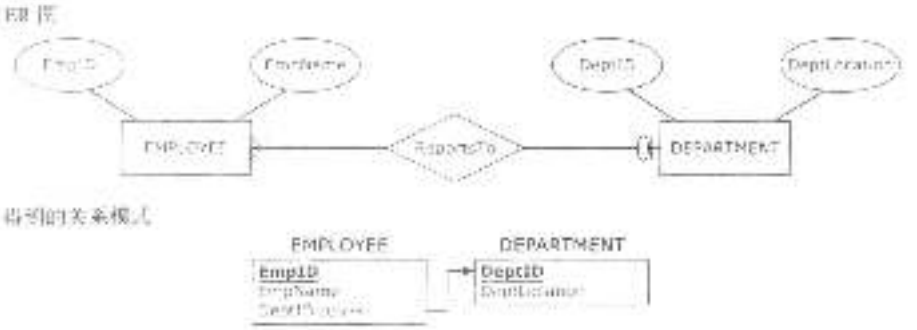


图 3-17　在 1 侧可选参与的一对多映射

EMPLOYEE

EmpID	EmpName	DeptID
1234	Becky	1
2345	Molly	2
3456	Rob	
1324	Tag	2

DEPARTMENT

DeptID	DeptLocation
1	Suite A
2	Suite B

图 3-18　图 3-17 所示关系数据库的样本数据记录

注意，图 3-18 中的雇员表与图 3-16 中的不同。在图 3-18 中，并不是所有的 DeptID 列都有取值，如雇员（3456，Rob）就没有 DeptID 属性值。

M 侧的可选参与　让我们再来看看在 M 侧的可选参与的例子。如图 3-19 和图 3-20 所示。

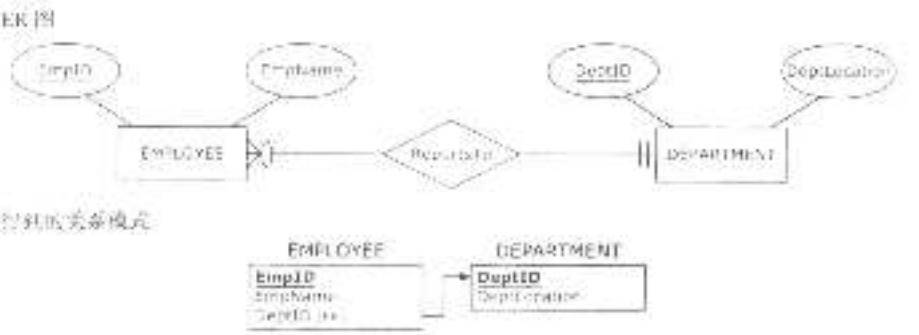


图 3-19　在 M 侧可选参与的一对多映射

上图 3-19 的关系模式作为关系数据库实现，它就可以装载数据，如图 3-20 所示。

EMPLOYEE

EmpID	EmpName	DeptID
1234	Becky	1
2345	Molly	2
3456	Rob	1
1324	Tag	2

DEPARTMENT

DeptID	DeptLocation
1	Suite A
2	Suite B
3	Suite C

图 3-20　图 3-19 所示关系数据库的样本数据记录

注意，图 3-20 中存在一个部门（3，SuiteC），没有与之相对应的雇员记录。如图 3-19

中间新的实体，其主属关系在M处的可选参与将导致这一现象的出现。 65

外键的重命名 图3-21展示了另外一个将一对多关系映射为关系模式的例子。在这个例子中，教授关系中的主码ProfID在其对应的学生表中更名为AdvisorID。将外码重命名是完全合法的。在本例中，我们将教授ID重命名为导师ID，会帮助我们更好地理解学生关系中教授所扮演的角色（即指导角色）。

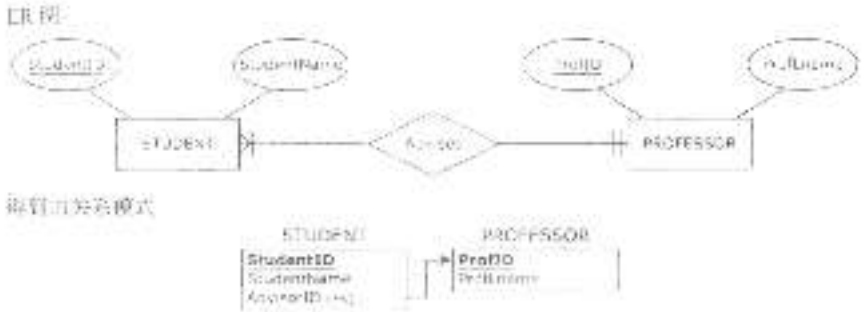


图3-21 为一个一对多联系的例子

如图3-22所示，一旦图3-21中所示的关系模式实现为一个关系数据库，我们就可向其中添加数据。

STUDENT

StudentID	StudentName	AdvisorID
1111	Robin	P11
2222	Pat	P22
3333	Jan	P11

PROFESSOR

ProfID	ProfLname
P11	Zydak
P22	Lash

图3-22 图3-21所示关系数据库的样本数据记录

3.10.2 M：N联系的映射

首先将多对多联系映射的规则：

除了多对多联系的两个实体需映射为关系之外，**多对多联系本身也需要映射为关系**。这种新关系有**两个外码**，对应多对多联系中两个实体的主码。**这两个外码就构成了这个新关系的复合主码**。

图3-23的例子中展示了学生（STUDENT）与组织（ORGANIZATION）这两个实体间的多对多联系——从属（BelongsTo）关系，并说明了多对多联系映射规则的具体应用。当图3-23中的ER图映射为关系模式时，除了学生和组织这两个关系之外，还要建立一个代表“从属”这个多对多联系的关系。诸如“从属”这样的多对多联系，有时候也被称作桥关系（bridge relation）。从属关系有两个外码，每一个外码都用一条连线与它们的对应源实体相连。其中一条连线由从属关系中的StudentID连接到学生关系中的主码StudentID，另一条连线则由从属关系的OrgID连接到组织关系中的主码OrgID。在从属关系的两个外码StudentID和OrgID下面都有一条下划线，表示这两个码组合起来形成从属关系的复合主码。（一个桥关系可以取与原ER图实体多对多联系不一样的名字，比如，数据库设计者也许会决定使用“参与”（PARTICIPATION）这个名字来代替“从属”，因为前者比后者更适合。尽管如此，我们在这个例子中仍然采用从属这个名称。）

一旦图 3-23 中的关系模式作为关系数据库实现，它就可以装载数据，如图 3-24 所示。注意从属关系中的取值是如何将学生与他们从属的组织联系起来的。

66

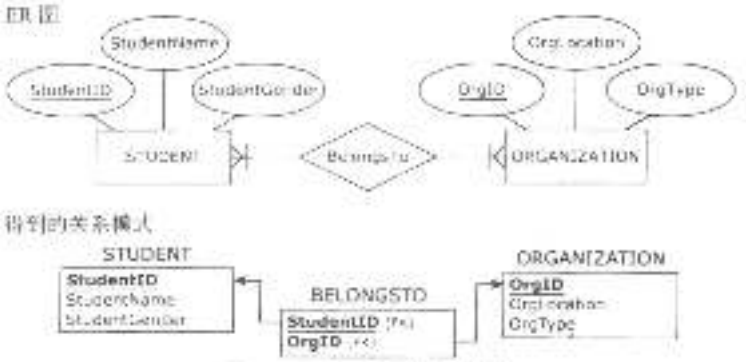


图 3-23 多对多联系的映射

STUDENT

StudentID	StudentName	StudentGender
1111	Robin	Male
2222	Pat	Male
3333	Jami	Female

ORGANIZATION

OrgID	OrgLocation	OrgType
O11	Student Hall	Charity
O41	Damen Hall	Sport
O47	Student Hall	Charity

BELONGSTO

StudentID	OrgID
1111	O11
1111	O41
2222	O11
2222	O41
2222	O47
3333	O11

图 3-24 图 3-23 所示关系数据库的样本数据记录

如果学生实体在从属关系中是可选参与的，则可能会存在额外的属于学生关系中的学生，但这些学生的 ID 没有出现在从属关系表的 StudentID 中。同样，如果组织实体在从属关系中是可选参与的，则可能会存在额外的属于组织关系中的组织，这些组织的 ID 也没有出现在从属关系表的 OrgID 中。这样的情况在图 3-25 和图 3-26 中有所体现。图 3-25 展示了两侧均为可选参与的从属关系。一旦图 3-25 所示的关系模式作为关系数据库实现，它就可以装载数据，如图 3-26 所示。注意，学生（4444，Abby）并不属于任何组织；而组织（O50，DamenHall，Politics）也同样没有任何学生。这可能是图 3-25 中的从属关系两边的可选参与导致的。

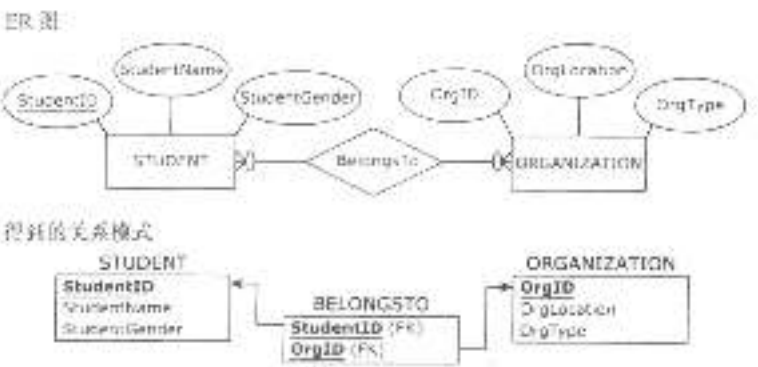


图 3-25 （两侧同时具有可选参与的）多对多联系

STUDENT

StudentID	StudentName	StudentGender
1111	Robin	Male
2222	Pat	Male
3333	Jami	Female
4444	Abby	Female

ORGANIZATION

OrgID	OrgLocation	OrgType
O11	Student Hall	Charity
O41	Damen Hall	Sport
O47	Student Hall	Charity
O50	Damen Hall	Politics

BELONGSTO

StudentID	OrgID
1111	O11
1111	O41
2222	O11
2222	O41
2222	O47
3333	O11

图3-26 图3-25所示关系数据库的样本数据记录

我们曾在第2章中提到，一个多对多联系可以有自己的属性。当一个具有自身属性的多对多联系被映射为关系模型时，多对多联系中的每一个属性都将映射为关系中的一列。图3-27展示了这种情况。 67

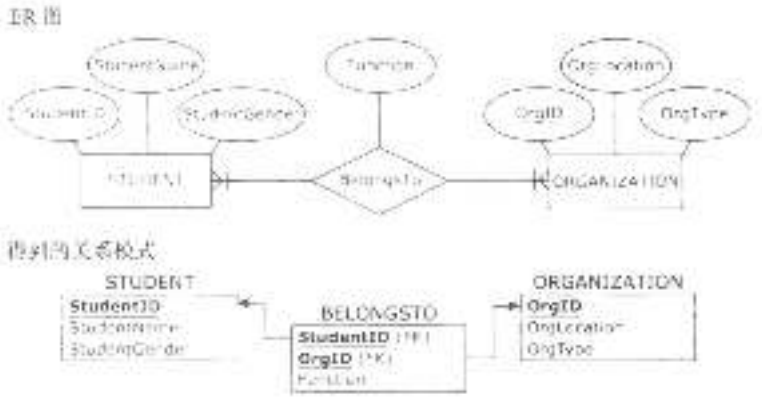


图3-27 具有属性的多对多映射

一旦如图3-27所示的关系模式被作为关系数据库实现，它就可以加载数据，如图3-28所示。

STUDENT

StudentID	StudentName	StudentGender
1111	Robin	Male
2222	Pat	Male
3333	Jami	Female

ORGANIZATION

OrgID	OrgLocation	OrgType
O11	Student Hall	Charity
O41	Damen Hall	Sport
O47	Student Hall	Charity

BELONGSTO

StudentID	OrgID	Function
1111	O11	President
1111	O41	Member
2222	O11	VP
2222	O41	Member
2222	O47	Treasurer
3333	O11	Member

图3-28 图3-27所示关系数据库的样本数据记录

3.10.3 1：1联系的映射

一对一联系的映射与一对多联系的映射很相似。映射得到的某个关系中将会有一个外码，这个外码指向另一个关系的主码。在一对多联系映射中我们需要遵从一条规则，这个规则要求由1侧映射得到的关系的主码，同时是由M侧得到的关系的外码。在一对一联系中，两个实体的最大基数值为1。因此，我们可简单地挑选任意一个映射关系，并将其外码对应

另一个映射关系的主码即可。

若选取某一个关系包含外码并没有特别明显的优势，那么此时的选择可以是任意的。然而，很有可能选择其中某一个关系来包含外码，比选择另一关系包含外码更有效率。举个例子，如果待选择的关系中，其中一个的外码是可选性的，而另一个是强制性的，那么选择强制性的外码将更可取。考虑图 3-29 和图 3-30 所示的情况，这是一个一对一的联系，我们将选择 VEHICLE 这个关系来包含外码 EmpID，这样可以保证外码的取值没有空值。而只我们采用另一种选择，即用 EMPLOYEE 关系来包含外码 VehicleID，这个外码就是可选性的，这将使得外码中存在空值。可选性的外码虽是合法的（如图 3-17 和图 3-18 所示），但是当可选性和强制性外码同时可供选择时，我们最好选择强制性外码。

68

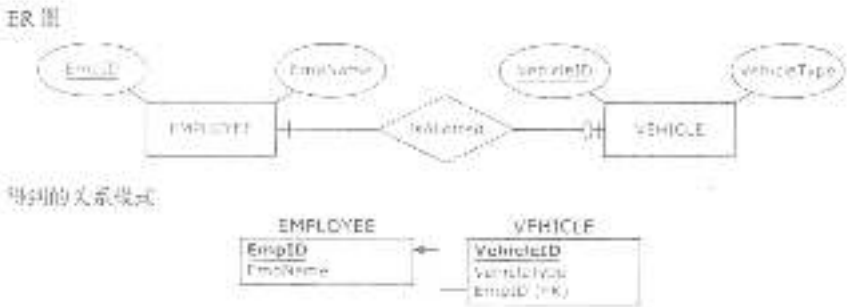


图 3-29　1：1 关系的映射

EMPLOYEE

EmpID	EmpName
1234	Becky
2345	Molly
3456	Rob
1324	Ted

VEHICLE

VehicleID	Vehicletype	EmpID
111	Sedan	1234
222	Van	2345
333	Van	3456

图 3-30　图 3-29 所示关系数据库的样本数据记录

3.11　参照完整性约束

参照完整性约束（referential integrity constraint）是在关系数据库中对外码有效取值的定义规则。

参照完整性约束

在包含外码的关系中，每一行的外码取值要么对应其参照关系中的主码取值，要么为空。

以上定义允许外码取值为空。当带外码的联系具有可选参与的实体，并且该实体在映射为关系后同样具有外码时，将允许那些映射后具有外码的实体具有可选参与性。我们采用图 3-17 所示的例子来说明参照完整性约束的定义。

在图 3-17 所示的 ER 图中，实体 EMPLOYEE 在与实体 DEPARTMENT 的关系中具有可选参与性。在相应的关系模式中，EMPLOYEE 关系有一个外码 DeptID。图 3-31 中给出了对应于图 3-17 的几个符合和违反参照完整性约束的例子。

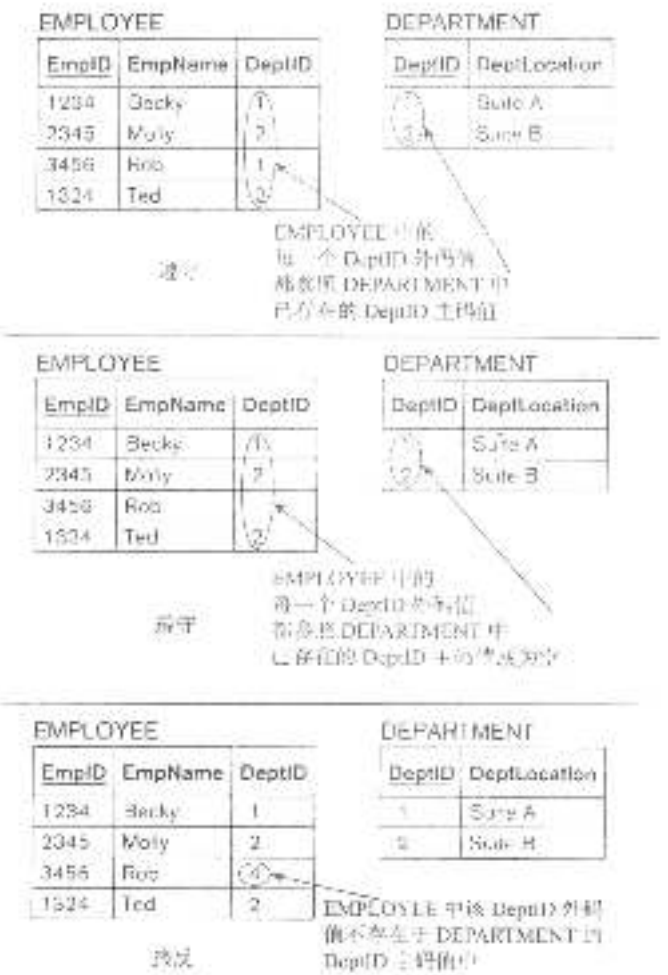


图3-31 参照完整性约束：遵守和违反的例子

在图3-31上面的例子中，所有在EMPLOYEE关系中的外码都参照了已有的DEPARTMENT关系中的主码，因此没有违反参照完整性约束。

在图3-31中间的例子中，所有在EMPLOYEE关系中的外码都参照了已有的DEPARTMENT关系中的主码。其中的一个外码取值为空，这也符合参照完整性约束（并且符合图3-17中EMPLOYEE实体在关系中的可选参与）。

在图3-31下面的例子中，EMPLOYEE关系中的一个外码取值（DeptID=4）并没有参照已有的DEPARTMENT关系中的主码，因此这是一个违反参照完整性约束的例子。

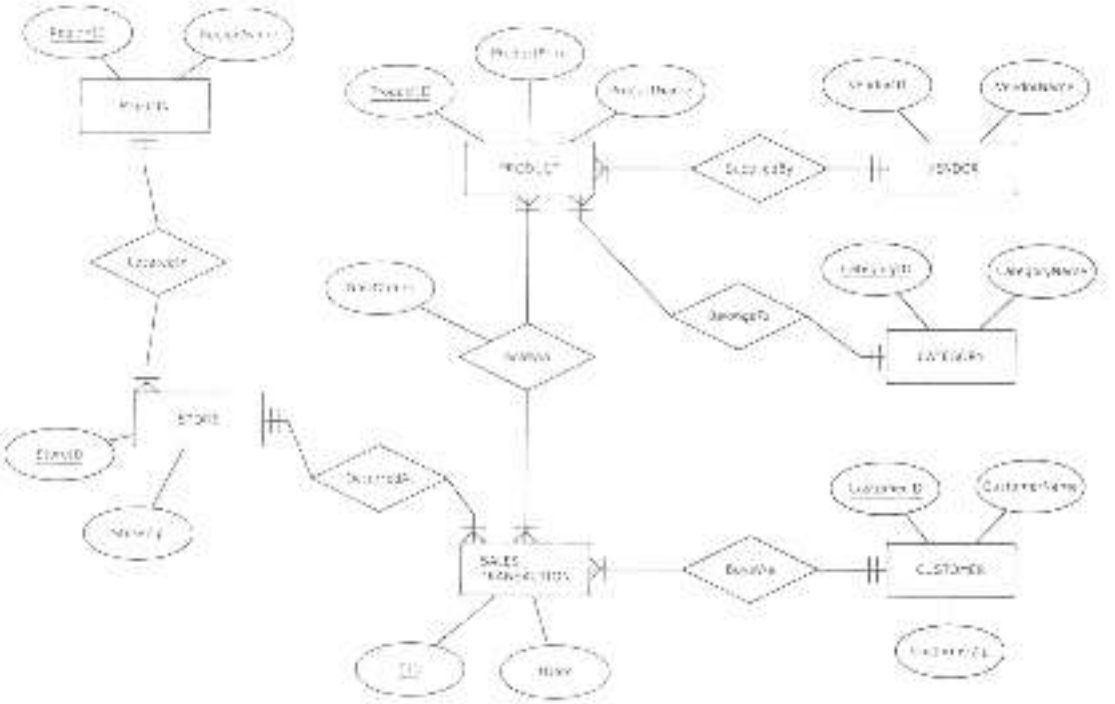
如前所述，关系模式是由关系和关系之间的连线构成的，其中连线将由外码连接到相应的主码。由于每一条从外码连接到相应主码的连线都要服从完整性约束，所以我们把这样的连线称作**参照完整性约束连线**（referential integrity constraint line）。

69

3.12 实例：将ER图映射为关系模式

我们现在给出一个具体的实例来总结一下前面提到的那些将ER图映射为关系数据库的

规则。图 3-32 展示了由 ZAGI 零售公司销售部门数据库 ER 图映射而成的关系模式（这个实例曾在前一章的图 2-13 中展示过，为方便后续讲解，我们将其重新绘出）。



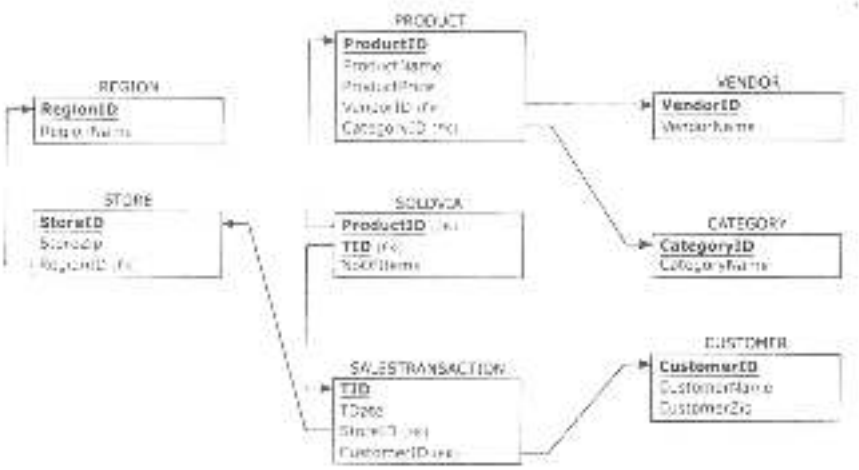


图 3-32　将 ER 图映射为关系模式的例子：ZAGI 零售公司

这张 ER 图中有 7 个实体和 1 个多对多关系。因此，其关系模式中就有 8 个关系表，对应为 7 个实体与 1 个实体间的多对多关系。

一旦图 3-32 所示的关系模式作为关系数据库实现，它就可以装载数据，如图 3-33 所示。

REGION

RegionID	RegionName
C	Chicagoland
T	Toledo

PRODUCT

ProductID	ProductName	ProductPrice	VendorID	CategoryID
1X1	Zip Disp	$130	PG	CP
2X2	Cozy Pillow	$20	MK	FW
3X3	Cozy Sheet	$15	MK	FW
4X4	Jiffy Sheet	$30	PG	FW
5X5	Tiny Tent	$150	MK	CP
6X6	Piggy Tent	$250	MK	CP

VENDOR

VendorID	VendorName
PG	Pocket Gear
MK	Mountain King

STORE

StoreID	StoreZip	RegionID
S1	60000	C
S2	60005	C
S3	35400	T

CATEGORY

CategoryID	CategoryName
CP	Camping
FW	Footwear

SALES TRANSACTION

TID	CustomerID	StoreID	TDate
T111	1-2-333	S1	1-Jan-2013
T222	2-3-444	S2	1-Jan-2013
T333	1-2-333	S3	2-Jan-2013
T444	3-4-555	S3	2-Jan-2013
T555	2-3-444	S3	2-Jan-2013

SOLDVIA

ProductID	TID	NoOfItems
1X1	T111	1
2X2	T222	1
3X3	T333	5
1X1	T333	1
4X4	T444	1
2X2	T444	2
4X4	T555	4
5X5	T555	2
6X6	T555	1

CUSTOMER

CustomerID	CustomerName	CustomerZip
1-2-333	Tim	60137
2-3-444	Tony	60611
3-4-555	Pam	35201

图 3-33　图 3-32 所示 ZAGI 零售公司销售部门数据库的样本数据记录

3.13　将拥有若干候选码（多个唯一属性）的实体映射为关系

第 2 章中曾提到一个实体可以具有多个唯一属性。在这种情况下，每个唯一属性称为一个“候选码”。术语“候选码”得名于这样一个事实：候选码中的一个必将由数据库设计者选中作为实体—关系映射过程中的主码，而其他没有被选中的候选码则被映射为非主码列。图 3-34 展示了一个拥有若干候选码的实体映射为关系的例子。

在关系模式中，只有主码有下划线。在图 3-34 所示的关系中，只有 EmpID 有下划线，因为数据库设计者选择它作为主码。SSN 没有下划线，因为在这里它并没有被选择作为主码。尽管如此，我们还是可以在非主码的唯一属性后面用带括号的大写字母 U 来表示其可被唯一标识。

一旦图 3-35 所示的关系模式作为数据库实现，它便可以装载数据，如图 3-35 所示。

当候选码中同时存在复合的和规则的（非复合的）候选码时，则以非复合候选码作为主码将会是更好的选择。如图 3-36 所示：

LPNumber（牌照号码）列和 State（所属州）列组合起来虽可构成 VEHICLE 关系的复合唯一属性，但 VIN（车牌号）因为是非复合唯一属性而最终被选作主码。

一旦图 3-36 所示的关系作为关系数据库的一部分得以实现，它就可以加载数据，如图 3-37 所示。

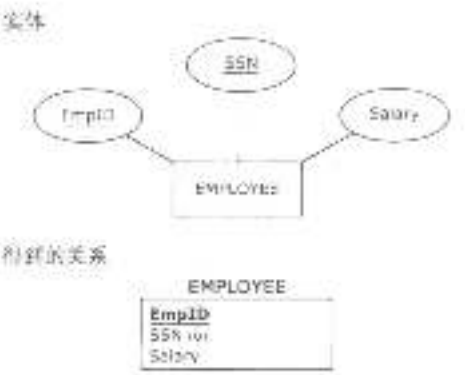


图 3-34　将具有候选码的实体映射为关系

EMPLOYEE

EmpID	SSN	Salary
1234	111-11-1111	$95,000
2345	222-22-2222	$50,000
3456	333-33-3333	$55,000
1324	444-44-4444	$70,000

图 3-35　图 3-34 所示关系的样本数据记录

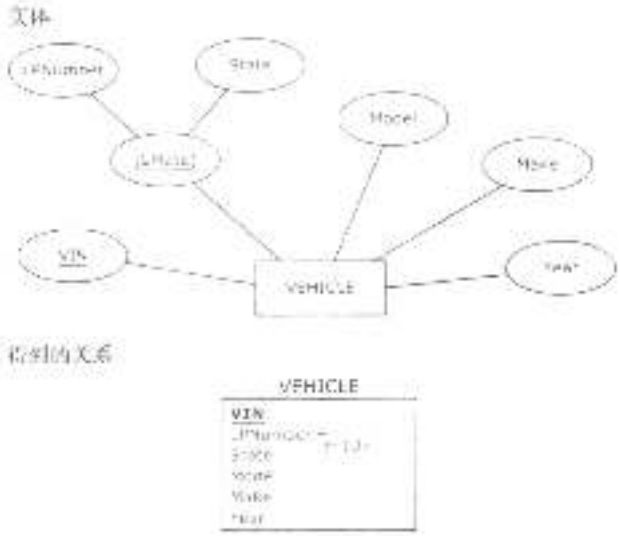


图 3-36　将具有非复合候选码的实体映射为关系

VEHICLE

VIN	LPNumber	State	Make	Model	Year
11111	X123	IL	Ford	Fiesta	2012
22222	X456	IL	Ford	Escape	2009
33333	X123	MI	Chevrolet	Volt	2012

图 3-37　图 3-36 所示关系的样本数据记录

3.14　将具有多值属性的实体映射为关系数据库组件

我们在第 2 章中曾提到，多值属性适用于同一个实体的某一个属性具有多个取值的情况。一个包含多值属性的实体将被映射为不含多值属性的关系。多值属性将被映射为一个单独的关系，这个关系中包含一个代表多值属性的列和一个连接相应主码的外码列，这两列组成这个单独关系的一个复合主码。图 3-38 中展示了如何将一个具有多值属性的实体映射为关系。

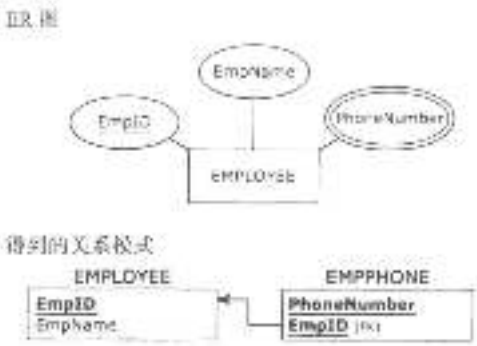


图 3-38　将具有多值属性的实体映射为关系

一旦如图 3-38 所示的关系模式作为关系数据库实现，它就可以装载数据，如图 3-39 所示。

由于 EMPLOYEE 实体中有多值属性 PhoneNumber，所以在 EMPPHONE 关系中，将允许在多行中出现不同的电话号码属于相同的雇员。

EMPPHONE 关系中的两列都不是唯一标识的（例如雇员 1234、3456 及 1324 共享同一个电话号码），但是 EmpID 和 PhoneNumber 这两列的组合却是可唯一标识的。因此，在 EMPPHONE 关系中，EmpID 和 PhoneNumber 将组合起来形成复合主码。

EMPLOYEE

EmpID	EName
1234	Becky
2345	Molly
3456	Rod
1324	Ted

EMPPHONE

EmpID	PhoneNumber
1234	630-111-4567
1234	630-222-4567
2345	630-333-4567
3456	630-111-4567
3456	630-444-4567
1324	630-111-4567
1324	630-555-4567
1324	630-666-4567

图3-39 图3-38所示关系数据库的样本数据记录

3.15 将具有派生属性的实体映射为关系

在第2章中我们曾提到，一个派生属性是那些属性值并非永久存储在数据库中的属性。派生属性的属性值是通过计算其他存储在数据库中的属性值而得到的。派生属性并不会被映射为关系模式的一部分，如图3-40所示。

一旦图3-40所示的关系作为关系数据库实现，它就可以加载数据，如图3-41所示。

图3-40 将具有派生属性的实体映射到关系

尽管派生属性不是关系模式的一部分，但它仍可以作为所创建数据库的前端应用的一部分得以实现。举例来说，一个基于数据库的前端应用包含一个STORE关系，如图3-41所示，这个关系中包含了一个新增的计算得到的列YearsInBusiness。这个列的值通过表达式（当前日期－开始日期）得到。图3-42展示了一个用户在2013年夏季可以在前端应用中看到的关系。

STORE (RELATION)

StoreID	OpeningDate
1111	1.1.2000
2222	2.2.2001
3333	3.3.2002
4444	2.2.2001

图3-41 图3-40所示关系的样本数据记录

STORE

Sid	OpeningDate	YearsInBusiness
1111	1.1.2000	13
2222	2.2.2001	12
3333	3.3.2002	11
4444	2.2.2001	12

图3-42 将图3-41所示的关系在终端应用程序中展示给用户

3.16 实例：将具有多种类型属性的实体映射为关系模式

为了对各种类型的属性的映射规则做一个总结，我们采用图3-43来展示一个从EMPLOYEE实体映射得到的关系模式（图中内容曾在前一章的图2-25中展示过，为方便讲解，在此我们将其重新给出）。

一旦图3-43所示的关系模式作为关系数据库实现，它就可以装载数据，如图3-44所示。

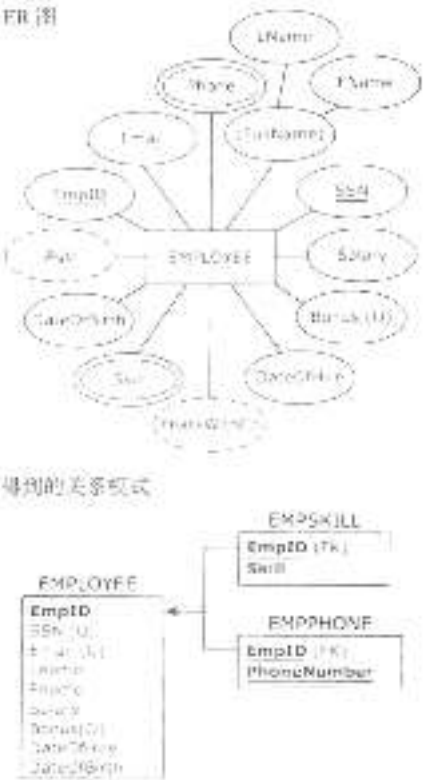


图 3-43　将具有多种类型属性的实体映射到关系

EMPLOYEE

EmpID	SSN	Email	FName	LName	Salary	Bonus	DateOfHire	DateOfBirth
1234	111-11-1111	bk@compx.com	Becky	Kaiser	$75,000		1.1.2000	11.12.1970
2345	222-22-2222	mn@compx.com	Molly	Nepe	$50,000	$10,000	2.2.2002	9.8.1973
3456	333-33-3333	rd@compx.com	Rob	Suzy	$55,000	$4,000	3.4.2003	11.11.1976
1324	444-44-4444	t@compx.com	Ted	Lovan	$70,000		9.8.2004	5.6.1971

EMPPHONE

EmpID	PhoneNumber
1234	630-111-4567
1234	630-222-4567
2345	630-333-4567
3456	630-111-4567
3456	630-444-4567
1324	630-111-4567
1324	630-555-4567
1324	630-666-4567

EMPSKILL

EmpID	Skill
1234	CPA
1234	CFP
2345	CPA
3456	CPA
3456	CFP
3456	CPP
1324	CFP

图 3-44　图 3-43 所示关系数据库的样本数据记录

3.17　一元联系的映射

ER 图中的一元联系可以用与二元联系同样的方式映射为关系模式，即使用一对多和一对一关系中的外码，以及在多对多关系中由两个外码构成的新关系来实现这样的映射。这里我们将对一对多、多对多以及一对一的一元关系映射进行描述。

3.17.1　1 : M 一元联系的映射

下面是一对多一元联系的映射规则：

一个由一对多一元联系实体映射得到的关系中包含一个外码，并且这个外码对应于关系自身的主码。

图3-45展示了一对多一元联系的映射。

注意，外码列ReferredBy有一个区别于其相应主码列ClientID的名称，这是合法的，这一点在本章中已有提及。实际上，在这个例子中，重命名是必须的，因为不能有名称完全相同的两列。重命名也说明了关系表中外码的角色。

同时应该注意，因为在参照关系的一侧有选择性参与，所以外码ReferredBy也是选择性的。

一旦图3-45所示的关系作为关系数据库实现，它就可以装载数据，如图3-46所示。

注意，图3-46中有一个顾客（C111）并没有被任何其他顾客所介绍，并且其中有两个顾客（C333和C444）也没有介绍任何其他顾客。这样的情况是允许的，因为图3-45中的关系Refers对于两侧来说都是可选择的。

图3-45 1：M一元联系映射

CLIENT

ClientID	ClientName	ReferredBy
C111	Mark	
C222	Mike	C111
C333	Lily	C111
C444	Jane	C222

图3-46 图3-45所示关系的样本数据记录

3.17.2 M：N一元联系的映射

下面是多对多一元联系映射的规则：

除了多对多一元联系中的实体需要映射为关系之外，多对多联系本身也需要映射为另一个关系。这个新关系有两个外码，它们由多对多联系中的两个实体的主码映射得到。其中每一个外码都将作为新关系中复合主码的一部分。

图3-47说明了多对多一元联系的映射。

图3-47 M：N一元联系映射

两个外码都被重命名以表达它们在关系ADVISING中的角色。一旦图3-47所示的关系模式作为关系数据库实现，它就可以装载数据，如图3-48所示。

因为图3-47中的ADVISING关系在其两侧都是可选性的，所以在EMPLOYEE关系中，允许出现没有作为指导者的雇员（1324，Ted），以及没有被指导的雇员（1234，Becky）。

EMPLOYEE

EmpID	EmpName
1234	Becky
2345	Molly
3456	Rob
1324	Tao

ADVISING

Advisor	Advisee
1234	2345
1234	3456
2345	1324
3456	1324
1234	1324

图 3-48　图 3-47 所示关系模式的样本数据记录

3.17.3　1：1 一元联系的映射

一对一一元联系的映射同一对多一元联系的映射很相似，如图 3-49 中的例子所示。这个例子描述了（在 Secret Santa 节日里的）礼物赠送事件，在这个事件中，每一个人将赠送另一个人一份礼物，并且每一个人会从确定的人那里得到礼物。

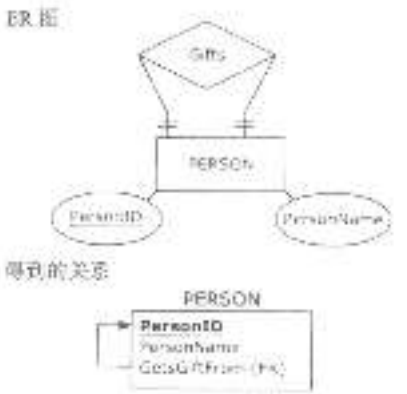


图 3-49　1：1 一元联系映射

一旦图 3-49 所示的关系作为关系数据库的一部分实现，它就可以装载数据，如图 3-50 所示。

PERSON

PersonID	PersonName	GetsGiftFrom
P111	Rose	P333
P222	Violet	P111
P333	Jamee	P444
P444	Lena	P222

图 3-50　图 3-49 所示关系的样本数据记录

注意，送礼关系在其两侧都是强制参与的，每一个人都必须送出一份礼物，并且每一个人也必须得到一份礼物，如图 3-50 中 PERSON 关系表中的记录所示。

3.18　相同实体间的多个联系的映射

如第 2 章所述，在 ER 图中相同的实体之间有多于一个联系存在的情况并不常见。图 3-51 所示的例子展示了在相同实体间的多个联系。

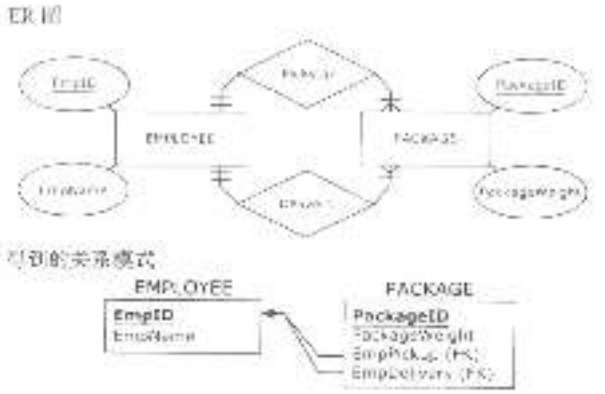


图 3-51 在相同实体之间映射多个关系

注意，PACKAGE 关系有两个外码，它们都参照于 EMPLOYEE 关系中的主码。两个外码都被重命名以更好地诠释 EMPLOYEE 关系的角色。

一旦图 3-51 所示的关系模式作为关系数据库实现，它就可以装载数据，如图 3-52 所示。 76

EMPLOYEE

EmpID	EmpName
1234	Bessy
2345	Molly
3456	Roy
1234	Ted

PACKAGE

PackageID	PackageWeight	EmpPickup	EmpDelivery
P111	5	1234	2345
P222	12	1234	1234
P333	8	2345	1234
P444	10	3456	1234
P555	7	1234	3456

图 3-52 图 3-51 所示关系模式的样本数据记录

3.19 弱实体的映射

弱实体的映射与常规实体的映射很相似。映射得到的关系有一个复合主码，这个复合主码由部分标识符、与属主实体主码相连的外码共同来构成。图 3-53 展示了弱实体的映射。

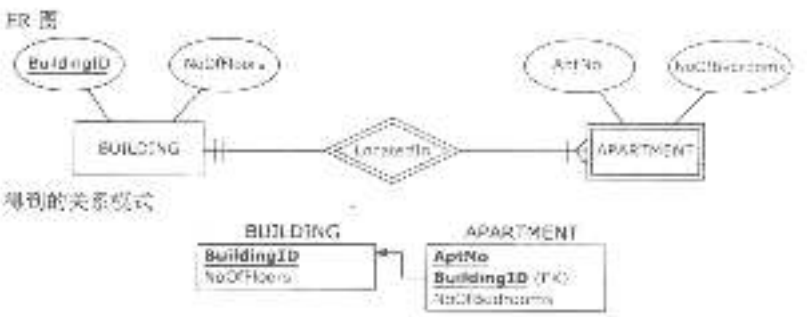


图 3-53 弱实体映射

一旦图 3-53 所示的关系模式作为关系数据库实现，它就可以装载数据，如图 3-54 所示。

在图 3-54 中，一个雇主实体实例可以由多个弱实体实例联合构成，这些弱实体都有彼此不同的部分码值。举例来说，在图 3-54 的 APARTMENT 关系中，建筑 A 有三个部门，且这三个部门有不同的部门编号，则部分码 AptNo 和 BuildingID 共同组成了关系 APARTMENT 的主码。

BUILDING

BuildingID	NoOfFloors
A	3
B	2
C	2

APARTMENT

BuildingID	AptNo	NoOfBedrooms
A	101	4
A	201	4
A	301	5
B	101	2
B	201	2
C	101	3
C	102	3
C	201	4

图 3-54　图 3-53 所示关系模式的样本数据记录

弱实体可以有多个雇主实体。在那种情况下，由弱实体映射得到的关系有一个复合主码，这个复合主码由部分标识符及所有雇主主码相对应的外码构成。图 3-55 展示了有两个雇主实体的一个弱实体的映射。

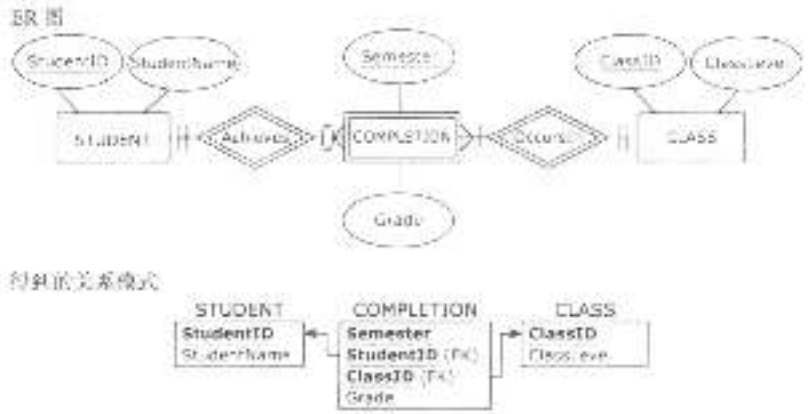


图 3-55　有两个雇主实体的一个弱实体的映射

注意，关系 COMPLETION 有两个外码，这两个外码来自于弱实体 COMPLETION 的两个雇主实体映射而成的关系。这两个外码与部分标识符 Semester 一起构成了关系 COMPLETION 的复合主码。

77　　把图 3-55 所示的关系模式作为关系数据库实现，它就可以装载数据，如图 3-56 所示。

STUDENT

StudentID	StudentName
1111	Robin
2222	Pat
3333	Jamie

CLASS

ClassID	ClassLevel
IS101	Freshman
IS241	Sophomore
IS247	Sophomore

COMPLETION

StudentID	ClassID	Semester	Grade
1111	IS101	Spring10	D
1111	IS101	Spring11	D
1111	IS101	Spring12	A
1111	IS241	Fall12	B
2222	IS101	Fall12	A
2222	IS241	Fall12	C
2222	IS247	Spring13	B
3333	IS101	Fall12	A

图 3-56　图 3-55 所示关系模式的样本数据记录

在第2章中我们曾讲解过这个实例，学生可能会多次学习同一门课程直到他们通过了最低通过分数C。所以，举个例子，如果学生1111学习了课程IS101三次，前两次他得到分数D，第三次他得到分数A。

当一个联系是一对一的联系时，其中的一个弱实体就没有部分标识符。这时，由属主实体映射得到的关系的主码就成为了这个弱实体映射得到关系的主码。图3-57展示了将一对一联系中的弱实体映射为关系。

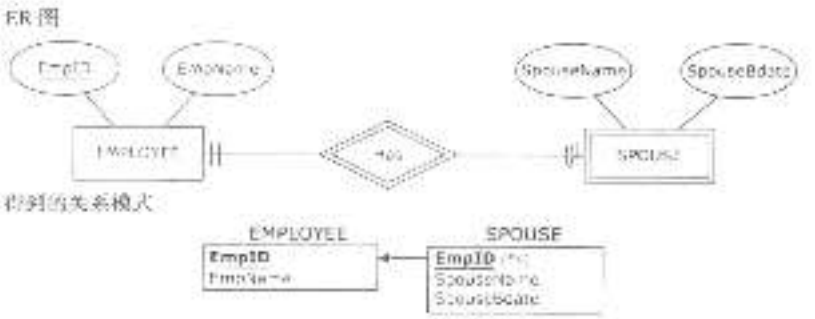


图3-57 没有部分标识符的弱实体映射

因为实体SPOUSE没有部分标识符，所以关系SPOUSE中的外码同时也是其主码，而并非由部分码组成的复合主码。

一旦图3-57所示的关系模式作为关系数据库实现，它就可以装载数据，如图3-58所示。

EMPLOYEE

EmpID	EmpName
1234	Becky
2345	Molly
3456	Rob
1324	Ted

SPOUSE

EmpID	SpouseName	SpouseBdate
1234	Steve	Jan 18
3456	Lucky	Jun 21
1324	Tina	Feb 11

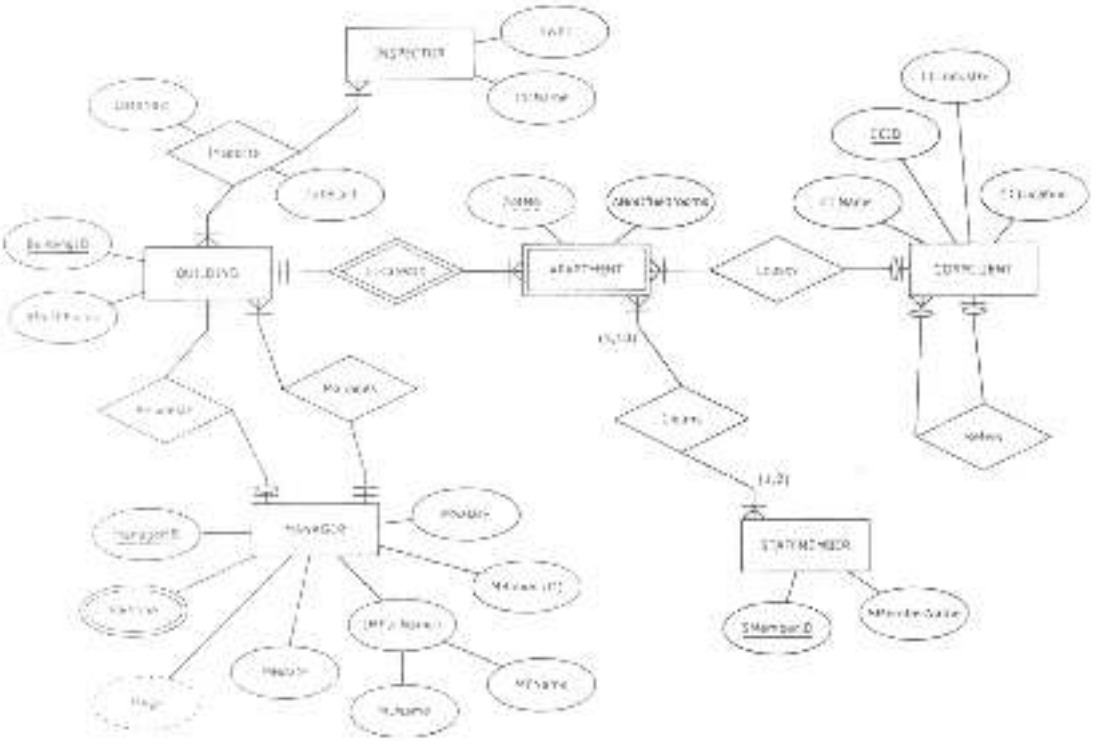
图3-58 图3-57所示关系模式的样本数据记录

3.20 实例：将另一个ER图映射为关系模式

为了再次说明本章中介绍的ER图－关系模式映射规则，图3-59展示了由HAFH Realty Company Property管理数据库的ER图映射得到的关系模式（这个例子曾在前一章的图2-38中列举过，重放在此以方便讲解）。

这张ER图有6个实体以及2个多对多联系。其中一个实体有多值属性，因此，映射得到的关系模式有9个关系，每个实体对应为一个联系，每个多对多联系也对应为一个关系，另外还有一个多值属性对应为一个关系。

一旦图3-59所示的关系模式作为关系数据库实现，它就可以装载数据，如图3-60所示。



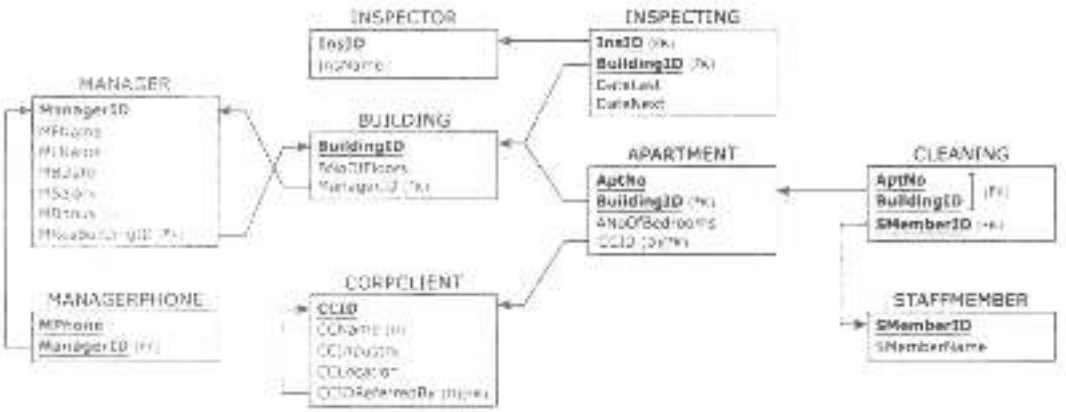


图 3-59　另一个将 ER 图映射为关系模式的例子：HAFH Realty

INSPECTOR

InsID	InsName
I11	John
I22	Nina
I33	Mick

BUILDING

BuildingID	BNoOfFloors	BManagerID
B1	5	M12
B2	6	M23
B3	4	M23
B4	4	M34

图 3-60　图 3-59 所示 HAFH Realty Company Property 管理数据库的样本数据记录

APARTMENT

BuildingID	AptNo	ANoOfBedrooms	CCID
B1	41	1	
B1	21	1	C111
B2	11	2	C222
B2	31	2	
B3	11	2	C777
B4	11	2	C777

INSPECTING

InsID	BuildingID	DateLast	DateNext
I11	B1	15-MAY-2012	14-MAY-2013
I11	B2	17-FEB-2013	17-MAY-2013
I22	B2	17-FEB-2013	17-MAY-2013
I22	B3	11-JAN-2013	11-JAN-2014
I33	B3	12-JAN-2013	12-JAN-2014
I33	B4	11-JAN-2013	11-JAN-2014

MANAGER

ManagerID	MFName	MLName	MBDate	MSalary	MBonus	MResBuildingID
M12	Boris	Grant	23-JUN-1980	50000		B1
M23	Alistair	Lee	30-OCT-1975	50000	5000	B2
M34	George	Sherman	11-JAN-1976	52000	2000	B4

CLEANING

BuildingID	AptNo	SMemberID
B1	21	5432
B1	41	9876
B2	11	9876
B2	31	5432
B3	11	5432
B4	11	7552

MANAGERPHONE

ManagerID	MPhone
M12	555-2222
M12	555-3232
M23	555-9986
M34	555-9996

STAFFMEMBER

SMemberID	SMemberName
5432	Brian
9876	Boris
7552	Caroline

CORPCLIENT

CCID	CCName	CCIndustry	CCLocation	CCIDReferredBy
C111	BlingNotes	Music	Chicago	
C222	SkyJet	Airline	Oak Park	C111
C777	WindyCT	Music	Chicago	C222
C888	SouthAlps	Sports	Rosemont	C777

图3-60（续）

3.21 关系数据库约束

如本章前述，一系列约束旨在形成令关系数据库有效的规则。关系数据库规则符合下面两种约束中的一种：**隐含约束**（implicit constraint）和**用户自定义约束**（user defined constraint）。

3.21.1　隐含约束

- 关系模式的每一个关系必须有独一无二的关系名。
- 每一个关系必须符合以下条件：
  - 每一列必须有不同的名字。
  - 不能有完全相同的行。
  - 在每一行的每一列中的取值应该是单值。
  - 每一列中的取值应该服从预定义的域（这一限制也称为**域约束**）。
  - 列顺序无关。
  - 行顺序无关。
- 每一个关系必须有一个主码，主码必须能唯一标识一行记录（这一限制也称为**主码约束**）。
- 主码值不能为空（**实体完整性约束**）。
- 在一个包含外码的关系的每一行中，外码的取值要么匹配其参照关系的主码，要么为空（**参照完整性约束**）。

78

79
80

3.21.2　用户自定义约束

除了隐含约束外，关系数据库的设计者也可为数据库设定特定的其他约束，这样的约束称为"用户自定义约束"。这里我们列举一些用户自定义约束的例子（在第6章中我们将介绍在已执行的数据库中采用用户自定约束的机制）。

一个用户自定义约束的例子是指定ER图中的可选属性并将其映射为关系模式。例如，设计者可以分别指定实体EMPLOYEE以及图3-11和图3-12所示的EMPLOYEE关系中的Bonus为可选性。

另一个用户自定义约束的例子是指定一个强制性外码，如图3-15和图3-16中所示。在这个例子中，关系EMPLOYEE中的DeptID列是强制性的，这是因为ReportsTo关系的1侧是强制参与的。

图3-15和图3-16中还同时展示了另一个用户自定义约束，即关系DEPARTMENT上的限制。该关系中的每一行必须被EMPLOYEE关系中的一个外码所参照，这是因为在关系ReportsTo中的M侧是强制参与的。注意，图3-19和图3-20中，部门实体的隶属关系在M侧的可选参与，将使得同一约束不会同时存在于图3-19和图3-20中。

其他用户自定义约束的例子还包括图3-61和图3-62所示的指定最小和最大基数。

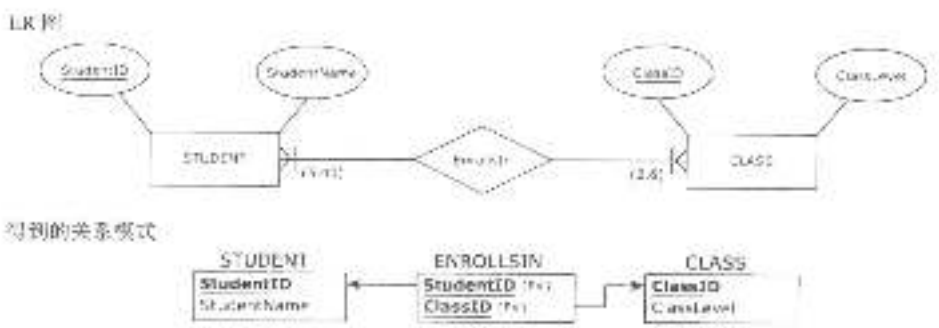


图3-61　指定最小和最大基数

STUDENT

StudentID	SName
1111	Robin
2222	Pat
3333	Jami
4444	Zach
5555	Louie

CLASS

ClassID	ClassLevel
IS346	Junior
IS401	Senior

ENROLLSIN

StudentID	ClassID
1111	IS346
2222	IS346
3333	IS346
4444	IS346
5555	IS346
1111	IS401
2222	IS401
3333	IS401
4444	IS401
2222	IS401

图 3-62 图 3-61 所示关系数据库的样本数据记录

一旦图 3-61 所示的关系模式作为关系数据库实现，它就可以装载数据，如图 3-62 所示。可以看到，在关系的两侧，所有的记录都服从指定的最大和最小基数限制，即每门课程有 5 名学生，每名学生可以选 2 门课程。如前所述，我们将在第 6 章介绍执行约束的机制。

到目前为止，我们所提到的用户自定义约束都将被指定作为 ER 图的一部分。另外一种用户自定义约束称为业务规则（business rule），是在最终生成的数据库中来指定约束，该约束并没有作为创建 ER 图的一部分。业务规则可以以注释的方式添加（如脚注，评论，特殊符号或者其他种类的记号）。

举个例子来说，一个业务规则中指定任意雇员的薪水将不能低于 5 万美元或者高于 20 万美元。这个规则可以以脚注的形式放在一个 ER 图中，如图 3-63 所示。

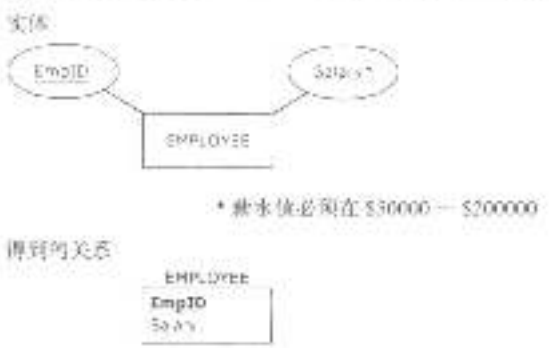


图 3-63 薪水数额的业务规则

一旦图 3-63 所示的关系模式作为关系数据库实现，它就可以装载数据，如图 3-64 所示。所输入的记录同时也会服从雇员薪水限制的业务规则。

EMPLOYEE

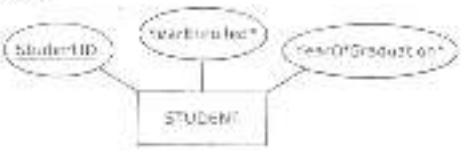
EmpID	Salary
1234	$75,000
2345	$50,000
3456	$55,000
1324	$70,000

图 3-64 图 3-63 所示关系的样本数据记录

另外一个例子如图 3-65 所示。一个业务规则规定了学生的毕业年份不得早于其注册年份。

一旦图 3-65 中的关系模式作为关系数据库实现，它就可以装载数据，如图 3-66 所示。所输入的记录同时也会服从学生毕业年份不得早于其注册年份的业务规则。

实体



* 毕业年份不得早于注册年份

得到的关系

图 3-65　注册年份和毕业年份的业务规则

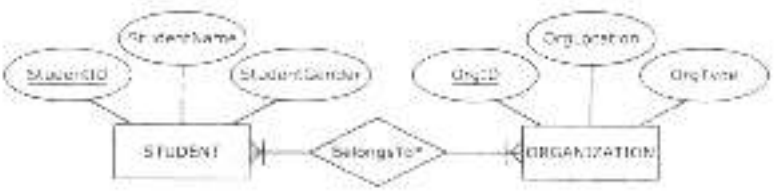
STUDENT

StudentID	YearEnrolled	YearOfGraduation
1111	2012	2016
2222	2013	2017
3333	2013	2017

图 3-66　图 3-65 所示关系的样本数据记录

又如图 3-67 中的业务规则，规定了每一个学生组织必须同时有男性和女性会员。

ER 图



* 每一个学生组织必须同时有男性和女性学生

得到的关系模式

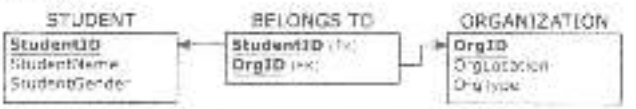


图 3-67　学生组织中学生性别的业务规则

一旦图 3-67 所示的关系模式作为关系数据库实现，它就可以装载数据，如图 3-68 所示。输入数据库的数据将服从学生组织必须有男性和女性会员的约束。代表组织会员的初始数据一旦向二张表格中加载，便会有一个机制去验证数据是否服从数据库的业务规则。

在第 6 章中，我们将会进一步讨论在数据库中设定业务规则（如同以上列举的实例中所展示的那样）的意义和方法。

这一章介绍了有关关系数据库建模的最基础的问题。下面的内容将涉及一些数据库建模的其他问题。

STUDENT

StudentID	StudentName	StudentGender
1111	Robin	M
2222	Pat	M
3333	Lam	F

BELONGSTO

StudentID	OrgID
1111	O11
3333	O11
2222	O11
3333	O41
2222	O41
3333	O43
1111	O43

ORGANIZATION

OrgID	OrgLocation	OrgType
O11	Student Hall	Charity
O41	Dormer Hall	Social
O43	Student Hall	Charity

图 3-68　图 3-67 所示关系数据库的样本数据记录

3.22　问题说明：关联实体映射

在第 2 章中我们曾提到关联实体这个 ER 建模概念，可以将其视作另一种处理多对多关系的方法。事实，关联实体将采用与多对多关系一样的方法映射为关系数据库构件。在这两种方法中，都会创建一个新的关系，这个关系中有两个外码，分别指向参与多对多联系的两个实体所得到的关系。

图 3-69 展示了一个多对多联系及其关联实体形式是如何以相同的方式映射为关系模式的。

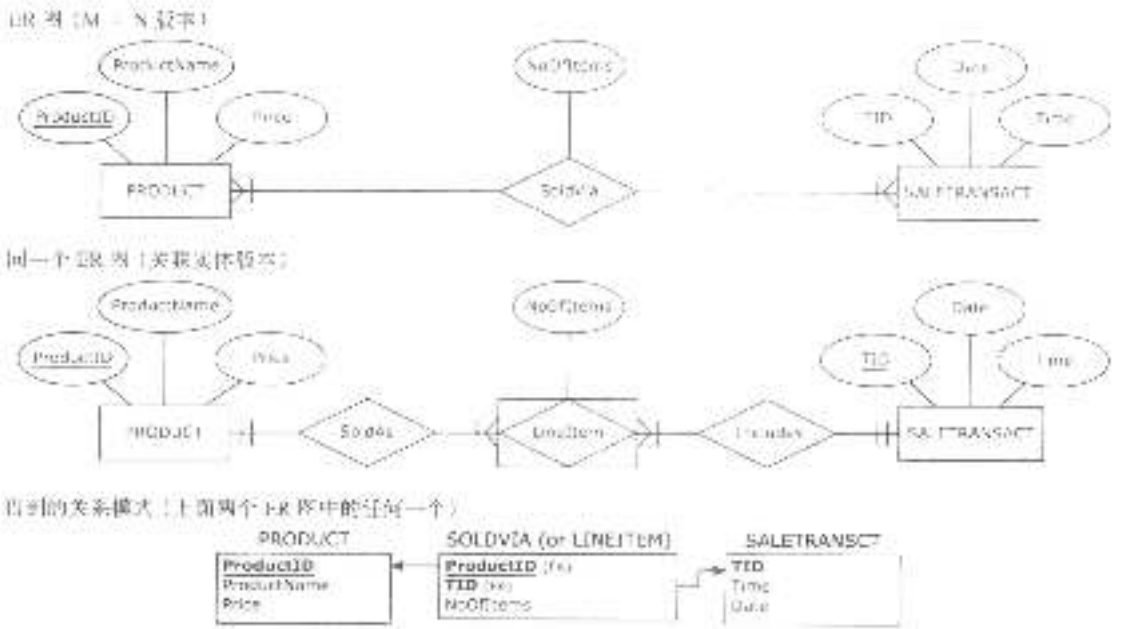


图 3-69　多对多联系及其关联实体形式以相同的方式映射为关系模式

3.23　问题说明：三元联系映射

在第 2 章中曾提到三元联系是多对多对多的联系。三元联系的映射与多对多联系的映射

非常相似。一个带有若干外码的新关系将被创建，这些外码都来自参与的关系，它们共同组成这个新关系的主码。图 3-70 展示了三元联系的映射。

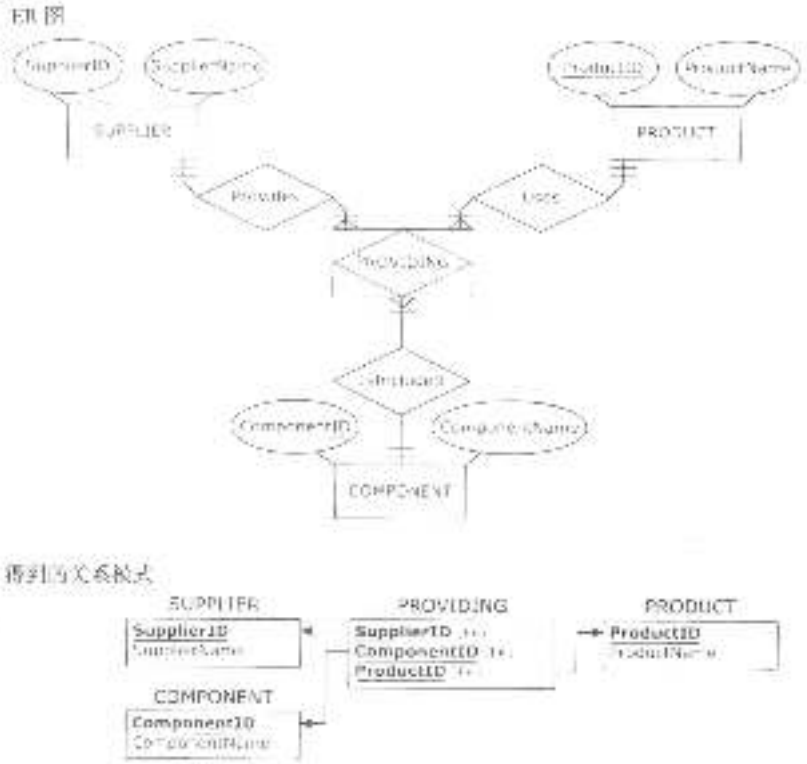


图 3-70　三元联系的映射

一旦图 3-70 所示的关系模式作为关系数据库实现，它就可以加载数据，如图 3-71 所示。

SUPPLIER

SupplierID	SupplierName
S1	Acme
S2	Xparts
S3	Compy

PRODUCT

ProductID	ProductName
P1	Bicycle
P2	Tricycle
P3	Scooter

COMPONENT

ComponentID	ComponentName
C1	Wheel
C2	Handle
C3	Seat

PROVIDING

SupplierID	ProductID	ComponentID
S1	P1	C1
S2	P1	C1
S3	P1	C1
S1	P1	C2
S2	P1	C2
S3	P1	C2
S1	P1	C3
S2	P1	C3
S3	P1	C3
S1	P2	C1
S1	P2	C2
S1	P2	C3
S1	P3	C1
S1	P3	C2

图 3-71　图 3-70 所示关系数据库的样本数据记录

在本例中，图3-71中的记录表明三个供应商提供了自行车的三个部件，并且供应商A1还专门为三轮车和轻便摩托车提供部件。

3.24 问题说明：设计者创建的主码和自动编号选项

很多现代数据库设计和DBMS工具为数据库设计者提供了一个自动编号数据类型选项，这个选项可以自动地在一列中生成连续数值型数据值。这个选项最常见的用途是创建设计者设定的主码列。举例来说，考虑以下需求：

- 医院数据库将保持对病患的跟踪。
- 对于每一个病患，医院将保持对其唯一的SSN（和出生日期和姓名）的掌握。

84

基于这种关系的实体及其映射得到的关系如图3-72所示。

PatientID列并没有在需求中写明。但是，在与终端用户讨论后，设计者决定不使用病患的SSN作为主码，而是创建另外一列PatientID作为关系PATIENT的主码，并且用自动增长的整数数据来装载关系表。首先，需求被更改如下：

- 医院数据库将保持对病患的跟踪。
- 对于每一个病患，医院将保持对其唯一的SSN以及唯一的PatientID的掌握（PatientID是一个简单的整数，每一个新病患都会被赋予紧随当前已分配整数之后的一个整数）。

如图3-73所示，根据修改过的需求，ER图也需做相应的修改。结果映射得到的关系中新增了PatientID这一列。

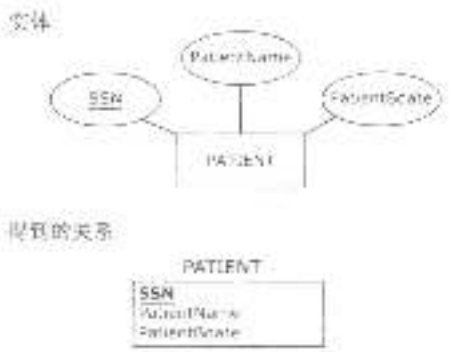


图3-72 实体及其得到的关系PATIENT

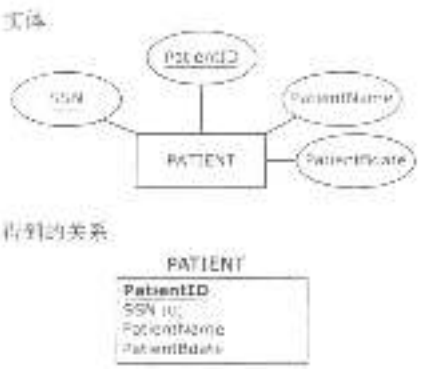


图3-73 实体以及添加了设计者创建的主键列以后的关系PATIENT

在关系PATIENT的实现过程中，自动编号数据类型被选择作为PatientID列的值。关系PATIENT中的数据如图3-74所示。SSN、PatientBdate以及PatientName这三列的数据需要输入。PatientID列的数值则由数据库系统来自动生成。

PATIENT

PatientID	SSN	PatientName	PatientBdate
1	123-44-4444	Ernest	1/1/1929
2	567-88-8888	Hans	2/2/1931
3	912-33-3333	Sally	4/3/1951

图3-74 图3-73所示关系的样本数据记录

3.25 问题说明：ER 建模和关系建模

一般教材中的数据库建模方法是在收集需求的同时进行 ER 建模，然后将 ER 模型映射为关系模式。但是，很多从业人员更倾向于直接从需求中创建关系模式。采用这种方式时，ER 建模的阶段就被省略了。

85

有经验的数据库建模者常感到他们的专业程度已经处于相当高的水平，不再需要针对每一个数据库都创建两种类型的图表。他们觉得看着需求立刻创建出可以被 DBMS 软件在所执行的关系模式是更可取的做法，这种做法直接跳过了 ER 图建模这一阶段。

这种简化的数据库建模方法乍一看非常有吸引力，因为它可以使目的变得简单并且节省时间。尽管如此，我们还是给出一些反对这种做法的建议，原因如下：

- ER 模型更有利于将需求可视化。
- 一些确定的概念只有在 ER 图中才能得到可视化的图形描述。
- 在 ER 图中每一个属性只被提到一次。
- ER 模型是更好的交流和文档化工具。

下面将就上述各原因进行讨论。

1. ER 模型更有利于将需求可视化

所有的需求都能以直接明了的形式在 ER 图中得到显式的可视化表达。换句话说，有一些需求在关系模式中的表达并没有那么直接明了。

让我们看看图 3-32。举例来说，“每一个产品都通过一笔或多笔交易销售，而每一笔销售交易都包括一个或多个产品”这个需求可由 ER 图得到可视化描述。同样的需求也可由关系 SOLDVIA 来描述，包括两个连接到关系 SALESTRANSACTION 和 PRODUCT 的外码。对于一个有经验的关系数据库建模者来说，这个关系模式已经可以很好地可视化上述需求了。但是，对于大多数需要可视化描述需求的商业委托人来说，ER 模型更简单也更清晰。

2. 一些确定的概念只有在 ER 图中才能得到可视化的图形描述

一些需求中的确定概念，只能通过 ER 图而不能通过关系模式进行图形化的可视描述。

让我们看看图 3-15。ER 图将“每一个部门必须至少有一个隶属雇员”这一需求捕捉并可视化了，而同样的需求将不能在关系模式中可视化。

另外一个能通过 ER 图而不能通过关系模式可视化的例子是复合属性，如图 3-6 所示。

3. 在 ER 图中每一个属性只被提到一次

看看那些对应于结果数据库的属性，由于 ER 图中每个属性只被提到一次，所以 ER 图是更简单的方法。

让我们看看图 3-59。BuildingID 是实体 BUILDING 的一个属性，它在 ER 图中只被提到过一次，同 ER 图中出现的其他属性一样。而在关系模式中，BuildingID 属性在关系 BUILDING 中提到一次，并且作为外码在其他三个不同的关系中出现，共被提到 4 次。一个有经验的数据库建模者也许可以将原始属性和其外码区分开来。但是，一个普通的商业用户在面临满是外码以及同一属性的多个实例的关系模式时可能会崩溃。

4. ER 模型是更好的交流和文档化工具

由于上述各种原因，在数据库设计的所有环节中，采用 ER 图都比采用关系模式更加便于交流。ER 图更有利于解释、讨论以及验证需求和最终得到的数据库，并更有利于数据库开发过程以及后续的数据库执行过程。

86

总结

除了3.2节的表3-1外，表3-2和表3-3总结了本章介绍的基本的关系概念。

表3-2 关系数据库的基本约束总结

约束	描述
命名关系	关系模式中的每个关系都必须有不同的名称
命名列	在一个关系中，每一列都必须有不同的名称
行唯一	在一个关系中，每一行必须唯一
列值为单值	在一个关系的每一行中，每一列的值必须为单值
域约束	在一个关系中，每一列的所有值必须来自预先定义的域
列的顺序	在一个关系中，列的顺序是无关的
行的顺序	在一个关系中，行的顺序是无关的
主码约束	每一个关系必须有一个主码，主码是一列（或一组列），其值对于每一行是唯一的
实体完整性约束	主码列不能为空值
外码	外码是关系中的一列，它参照另一个（或参照的）关系的主码
参照完整性约束	包含外码的关系中，每一行的外码取值要么对应其参照关系中的主码取值，要么为空

表3-3 将关系模式映射为规则的基本ER总结

ER构件	映射规则
一般实体	映射为关系
一般属性	映射为关系的列
唯一属性	实体的一个唯一属性被选为主码。如果还有其他的唯一属性，则被标记为唯一（但不作为主码）
复合属性	只有复合属性的各个部分映射为关系的列（复合属性本身不作映射）
复合的唯一属性	只有它的各个部分被映射。如果实体中没有单值唯一属性，则复合属性的各部分变为复合主码。否则，将各部分标记为唯一（不作为主码）
多值属性	被映射为将为复合主码的单独的关系。复合主码是由表示多值属性的列和外码组成的，该外码参照了表示包含多值属性实体的关系的主码
派生属性	不作映射
可选属性	映射为一列，标记为可选
1：M二元联系	位于M侧的实体被映射得到的关系有一个外码，这个外码对应于由1侧的实体映射得到的关系中的主码
M：N二元联系	映射为带有两个外码的新关系，这两个外码对应于多对多关系中两个实体的主码。这两个外码构成这个表示多对多联系的新关系的复合主码。如果存在属性，则联系的属性映射为新关系的列
1：M一元联系	一对多一元联系实体映射得到的关系中包含一个外码，并且这个外码对应于关系自身的主码
M：N一元联系	映射为带有两个外码的新关系，这两个外码都对应于多对多关系中实体的主码。每一个外码都将作为新关系中复合主码的一部分
关联实体	与映射M：N联系的规则相同，创建一个带有外码的新关系，外码对应于表示参与关联实体的关系的主码
弱实体	映射为带有一个外码的新关系，外码对应于表示主实体的关系的主码。由部分码映射得到的列和主实体的外码一起作为复合主码（如果没有部分码，外码单独作为主码）
三元联系	与关联实体的映射规则相同

87

关键术语

attribute（of a relation）（关系的）属性，57
autonumber data type（自动编号数据类型），84
bridge relation（桥关系），66
business rule（业务规则），81
column（列），57
composite primary key（复合主码），61

designer-created primary key（设计者创建的主码），84
domain constraint（域约束），79
entity integrity constraint（实体完整性约束），62
foreign key（外码），63
implicit constraint（隐含约束），78
primary key（主码），59
primary key constraint（主码约束），79
record（记录），57
referential integrity constraint（参照完整性约束），69
referential integrity constraint line（参照完整性约束连线），69
relation（关系），57
relation database（关系数据库），57
relation database model（关系数据库模型），57
relational DBMS，RDBMS（关系数据库管理系统），57
relation schema（关系模式），57
relation table（关系表），57
row（行），57
table（表），57
tuple（元组），57
user-defined constrain（用户自定义约束），78

复习题

Q3.1　一张表成为一个关系必须满足哪些条件？
Q3.2　什么是主码？
Q3.3　如何将一个具有简单属性的普通实体映射为一个关系？
Q3.4　如何将一个复合属性映射为一个关系？
Q3.5　如何将一个嵌套复合属性映射为一个关系？
Q3.6　如何将一个可选多属性映射为一个关系？
Q3.7　给出实体完整性约束的定义。
Q3.8　什么是外码？
Q3.9　如何将两个实体间的一对多联系映射为关系模式？
Q3.10　如何将两个实体间的多对多联系映射为关系模式？
Q3.11　如何将两个实体间的一对一联系映射为关系模式？
Q3.12　给出参照完整性约束的定义。
Q3.13　如何将依赖码映射为关系？
Q3.14　如何将多值属性映射为关系模式？
Q3.15　如何将派生属性映射为关系模式？
Q3.16　如何将一对多一元联系映射为关系模式？
Q3.17　如何将多对多一元联系映射为关系模式？
Q3.18　如何将一对一一元联系映射为关系模式？
88 | Q3.19　如何将弱实体映射为关系模式？
Q3.20　如何将三元联系映射为关系模式？
Q3.21　列出关系数据库模型的隐含约束。
Q3.22　什么是用户自定义约束？
Q3.23　什么是业务规则？
Q3.24　什么是设计者所创建的主码？
Q3.25　反对不创建 ER 模型而直接创建关系数据库模型的主要原因是什么？

练习

E3.1　给出两个实例，一个是属于关系的表，另一个是不属于关系的表。
E3.2　给出两个实例，一个是遵从实体完整性约束的关系，另一个是违反实体完整性约束的关系。

E3.3 给出两个实例，一个是服从参照完整性约束的关系，另一个是违反参照完整性约束的关系。

E3.4 将 E2.1 中的实体映射为关系。

E3.5 将以下练习题中的 ER 图映射为关系模式，并满足以下要求：

- E3.5a 对于 E2.2a，在每一个关系中给出几个记录，并且为这些取值标明是在 1 侧的可选参与还是在 M 侧的可选参与。
- E3.5b 对于 E2.2b，在每一个关系中给出几个记录，并且为这些取值标明是在 1 侧的可选参与还是在 M 侧的强制参与。
- E3.5c 对于 E2.2c，在每一个关系中给出几个记录，并且为这些取值标明是在 1 侧的强制参与还是在 M 侧的强制参与。
- E3.5d 对于 E2.2d，在每一个关系中给出几个记录，并且为这些取值标明是在 1 侧的可选参与还是在 M 侧的可选参与。
- E3.5e 对于 E2.2e，在每一个关系中给出几个记录，并且为这些取值标明是在 1 侧的强制参与还是在另一侧的可选参与。
- E3.5f 对于 E2.2f，在每一个关系中给出几个记录，并且为这些取值标明是在 1 侧的强制参与还是在其他边的强制参与。
- E3.5g 对于 E2.2g，在每一个关系中给出几个记录，并且为这些取值标明是在 1 侧的可选参与还是在其他边的强制参与。
- E3.5h 对于 E2.2h，在每一个关系中给出几个记录，并且为这些取值标明是在 1 侧的强制参与还是在其他边的可选参与。
- E3.5i 对于 E2.2i，在每一个关系中给出几个记录，并且为这些取值标明是在 1 侧的强制参与还是在其他边的强制参与。
- E3.5j 对于 E2.2j，在每一个关系中给出几个记录，并且为这些取值标明是在 1 侧的可选参与还是在其他边的可选参与。

E3.6 将 E2.3 中的 ER 图映射为关系，在得到的关系中给出几个记录。

E3.7 将 E2.4 中的实体映射为关系，在得到的关系中给出几个记录。

E3.8 将 E2.5 中的实体映射为关系，在得到的关系中给出几个记录。

E3.9 将 E2.6 中的实体映射为关系，在得到的关系中给出几个记录。

E3.10 将 E2.7 中的实体映射为关系，在得到的关系中给出几个记录。

E3.11 将 E2.8 中的实体映射为关系，在得到的关系中给出几个记录。

E3.12 将 E2.9 中的实体映射为关系，在得到的关系中给出几个记录。

E3.13 将 E2.10 中的 ER 图映射为关系，在得到的关系中给出几个记录。

E3.14 将基于 E2.11 中的需求得到的 ER 图映射为关系，在得到的关系中给出几个记录。

E3.15 将基于 E2.12 中的需求得到的 ER 图映射为关系，在得到的关系中给出几个记录。

E3.16 将基于 E2.13 中的需求得到的 ER 图映射为关系，在得到的关系中给出几个记录。

E3.17 构造一个服从某业务规则的 ER 图实例，将这个 ER 图映射为关系并给出几个记录，以说明它们服从业务规则。

小案例

小案例 1 Investco Scout

将 Investco Scout Funds 数据库（由第 2 章的小案例 1 创建）的 ER 图映射为关系模式。

小案例 2 Funky Bizz

将 Funky Bizz Operations 数据库（由第 2 章的小案例 2 创建）的 ER 图映射为关系模式。

小案例 3 Snooty Fashions

将 Snooty Fashions Operations 数据库（由第 2 章的小案例 3 创建）的 ER 图映射为关系模式。

小案例 4 Signum Libri

将 Signum Libri Operations 数据库（由第 2 章的小案例 4 创建）的 ER 图映射为关系模式。

小案例 5 ExoProtect

将 ExoProtect Employees' Computers 数据库（由第 2 章的小案例 5 创建）的 ER 图映射为关系模式。

小案例 6 Jones Dozers

将 Jones Dozers Sales and Rentals 数据库（由第 2 章的小案例 6 创建）的 ER 图映射为关系模式。

小案例 7 Midtown Memorial

将 Midtown Memorial Patients Drug Dispense 数据库（由第 2 章的小案例 7 创建）的 ER 图映射为关系模式。

更新操作、更新异常与规范化

4.1 引言

在前面的章节中，我们讲述了关系数据库的基本概念以及如何将 ER 图映射成关系模式。本章我们将会集中关注以下操作：向关系中输入数据、修改关系中已有的数据、从关系中删除数据。这三个操作合起来统称为**更新操作**（update operation）。

如果一个关系中存储了多个重复实例（例如，一个购物表包含多行同一顾客地址），这些重复的实例便是**冗余数据**（redundant data）。不良的数据库设计会导致关系中包含不必要的冗余数据，并致使更新操作出现某些异常。本章将会详细地说明这些**更新异常**（update anomaly）。

关系数据库可能包含冗余数据并致使更新异常，而**规范化**（normalization）过程可用于改进关系数据库的设计。规范化过程将从关系数据库中删除不必要的冗余数据，以此来消除更新异常的可能性。本章将会详细地描述规范化过程。

4.2 更新操作

用户可以从关系中检索数据，或者更新关系中的数据。数据检索操作通常也称为**读操作**（read operation），用于从关系中读取数据。更新操作也叫**写操作**（write operation），用于更新关系中的数据内容。

关系中的数据内容有三种更新方式：向关系中输入新数据、从关系中移除数据以及改变关系中已有的数据。因此，对应有三种更新操作：

- **插入操作**（insert operation）：用于向关系中输入新数据。
- **删除操作**（delete operation）：用于从关系中移除数据。
- **修改操作**（modify operation）：用于改变关系中已有的数据。

下面将分别给出三种更新操作的例子。

4.2.1 插入操作实例

开始时，图 4-1 中的关系 EMPLOYEE 有三条记录。插入一条新记录以后，关系 EMPLOYEE 中有四条记录。

插入前

EMPLOYEE

EmpID	EmpName	Salary
1234	Becky	$75,000
2345	Molly	$50,000
3456	Rob	$55,000

插入后

插入一条记录

EMPLOYEE

EmpID	EmpName	Salary
1234	Becky	$75,000
2345	Molly	$50,000
3456	Rob	$55,000
1324	**Ted**	**$70,000**

图 4-1 插入操作

4.2.2 删除操作实例

开始时，图 4-2 中的关系 EMPLOYEE 有三条

记录，删除一条记录以后，关系 EMPLOYEE 中有两条记录。

删除前

EMPLOYEE

EmpID	EmpName	Salary
1234	Becky	$75,000
2345	**Molly**	**$50,000**
3456	Rob	$55,000

删除一条记录

删除后

EMPLOYEE

EmpID	EmpName	Salary
1234	Becky	$75,000
3456	Rob	$55,000

图 4-2　删除操作

4.2.3　修改操作实例

图 4-3 中的关系 EMPLOYEE 有三条记录。雇员 Molly 这条记录的 Salary 的值从 $50 000 变成了 $60 000。执行完这个操作以后，关系 EMPLOYEE 仍然有三条记录，但是其中一条记录被修改了。

[92]

4.2.4　关于更新操作术语的说明

在实际工程中以及相关文献中，术语“更新操作”通常有两种用法。一种用法是将“更新操作”视为插入、删除和修改操作的集合。另一种用法是将“更新操作”仅视为修改操作。这两种用法都很常见。分清这两种用法很重要。通常根据术语所处的上下文，可以很清楚地确定到底是那种用法。在本章中，我们将术语“更新操作”视为插入、删除和修改操作的集合。[1]

修改前

EMPLOYEE

EmpID	EmpName	Salary
1234	Becky	$75,000
2345	Molly	**$50,000**
3456	Rob	$55,000

修改一条记录

修改后

EMPLOYEE

EmpID	EmpName	Salary
1234	Becky	$75,000
2345	Molly	**$60,000**
3456	Rob	$55,000

图 4-3　修改操作

4.3　更新异常

在包含冗余（不必要的重复）数据的关系中，三种更新操作均可能引起更新异常。因此，共有三种类型的更新异常：插入异常、删除异常和修改异常。我们将使用关系 AD CAMPAIGN MIX 作为例子来说明三种类型的更新异常。该关系来自下面将要介绍的 Pressly 广告代理的实际场景。在本章的后续部分，这个例子还将用来说明函数依赖和规范化过程。

4.3.1　实例场景

Pressly 广告代理通过多种宣传模式来管理广告宣传活动。每种宣传模式都有一个模式

[1] 在第 5 章将会看到，SQL 命令 UPDATE 是修改操作的命令，这使得术语“更新操作”被视为修改操作。

标识符 ModeID（如1、2、3），它表示某种媒体（如电视、广播、平面媒体）的特定覆盖范围（如本地、全国）。例如，使用电视媒体覆盖本地区，该宣传模式的 ModeID 值为 1；使用电视媒体覆盖全国，该宣传模式的 ModeID 值为 2。

Pressly 广告代理管理的每个广告宣传活动都有唯一的标识符、唯一的名字、开始日期、持续时间和一个宣传经理（包括经理的名字和唯一的标识符）。每个广告可以有多种不同的宣传模式。当使用多种不同的宣传模式进行广告宣传时，广告宣传的预算会按百分比分配到每种模式上。当只使用一种宣传模式进行广告宣传时，广告宣传的预算 100% 分配到这种模式上。 93

4.3.2 实例关系（包含冗余数据）

Pressly 广告代理例子中使用单个关系 AD CAMPAIGN MIX 来存储数据。该例子中的关系是一个包含冗余数据的关系，因此容易产生更新异常。

关系 AD CAMPAIGN MIX 由以下的列组成：

AdCampaignID	广告宣传活动标识符，每个广告宣传活动有唯一的值
AdCampaignName	广告宣传活动名称，每个广告宣传活动有唯一的名称
StartDate	广告宣传活动的开始日期（可以有多个广告宣传活动在同一日期开始）
Duration	广告宣传活动的持续天数（可以有多个广告宣传活动持续天数相同）
CampaignMgrID	宣传经理的标识符，每个宣传经理有唯一的值（每个广告宣传活动只有一个宣传经理；一个宣传经理可以管理多个宣传活动）
CampaignMgrName	宣传经理的名字（多个宣传经理可能有相同的名字）
ModeID	宣传模式标识符，每个宣传模式有唯一的值
Media	宣传模式的媒体类型（每个宣传模式只有一种媒体，但同一种媒体可以在多个宣传模式中使用）
Range	宣传模式的覆盖范围（每个宣传模式只有一种覆盖范围，但同一种覆盖范围可以在多个宣传模式中使用）
BudgetPctg	广告宣传活动为某种宣传模式分配的预算百分比

图 4-4 展示了关系 AD CAMPAIGN MIX 中的记录。一个广告宣传活动可以有多种模式，这个关系中的每条记录描述了一次宣传活动的一种模式。因此，能够唯一区分关系中的每一行的属性，是由宣传活动的唯一属性和模式的唯一属性构成的组合属性。模式本身只有一个唯一属性：唯一的 ModeID。然而，每个广告宣传活动有两个唯一属性：唯一的 AdCampaignID 和唯一的 AdCampaignName。组合码 AdCampaignID、ModeID 被选作这个关系的主码。因此，组合码 AdCampaignName、ModeID 是这个关系的一个候选码，未被选作主码。 94

AD CAMPAIGN MIX

AdCampaignID	AdCampaignName	StartDate	Duration	CampaignMgrID	CampaignMgrName	ModeID	Media	Range	BudgetPctg
111	SummerFun13	6.6.2013	12 days	CM100	Roberto	1	TV	Local	50%
111	SummerFun13	6.6.2013	12 days	CM100	Roberto	2	TV	National	50%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	1	TV	Local	80%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	3	Radio	Local	30%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	5	Print	Local	10%
333	FallBall13	6.8.2013	12 days	CM102	John	3	Radio	Local	80%
333	FallBall13	6.8.2013	12 days	CM102	John	4	Radio	National	20%
444	AutumnStyle13	6.9.2013	5 days	CM103	Nancy	5	Print	National	100%
555	AutumnColors13	6.9.2013	3 days	CM100	Roberto	3	Radio	Local	100%

图 4-4 关系 AD CAMPAIGN MIX

非码列（nonkey column）是指关系中既不是主码也不是候选码的列。在关系 AD CAMPAIGN MIX 中，StartDate、Duration、CampaignMgrID、CampaignMgrName、Media、Range 和 BudgetPctg 都是非码列。

关系 AD CAMPAIGN MIX 的这种设计会使它包含冗余数据。例如，宣传活动 222 的名称、开始日期和持续时间重复了三次，宣传经理 CM100 的名字重复了三次，宣传模式 1 的媒体和覆盖范围重复了两次。关系 AD CAMPAIGN MIX 中这些冗余数据的例子见图 4-5。

AD CAMPAIGN MIX

AdCampaignID	AdCampaignName	StartDate	Duration	CampaignMgrID	CampaignMgrName	ModeID	Media	Range	BudgetPctg
111	SummerFun13	6.6.2013	12 days	CM100	Roberta	1	TV	Local	50%
111	SummerFun13	6.6.2013	12 days	CM100	Roberta	2	TV	National	50%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	1	TV	Local	80%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	3	Radio	Local	30%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	5	Print	Local	10%
333	FallBash13	6.9.2013	12 days	CM102	John	3	Radio	Local	80%
333	FallBash13	6.9.2013	12 days	CM102	John	4	Radio	National	20%
444	AutumnStyle13	6.9.2013	5 days	CM103	Nancy	6	Print	National	100%
555	AutumnCelebr13	6.9.2013	3 days	CM100	Roberta	3	Radio	Local	100%

宣传模式 1 的媒体和覆盖范围重复了两次

宣传经理 CM100 的名字重复了三次

宣传活动 222 的名称、开始日期和持续时间重复了三次

图 4-5　关系 AD CAMPAIGN MIX 中冗余数据的例子

由于这种设计允许冗余数据出现，因此关系 AD CAMPAIGN MIX 会产生下面几个例子所示的更新异常，见图 4-6。

AD CAMPAIGN MIX

AdCampaignID	AdCampaignName	StartDate	Duration	CampaignMgrID	CampaignMgrName	ModeID	Media	Range	BudgetPctg
111	SummerFun13	6.6.2013	12 days	CM100	Roberta	1	TV	Local	50%
111	SummerFun13	6.6.2013	12 days	CM100	Roberta	2	TV	National	50%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	1	TV	Local	80%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	3	Radio	Local	30%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	5	Print	Local	10%
333	FallBash13	6.9.2013	12 days	CM102	John	3	Radio	Local	80%
333	FallBash13	6.9.2013	12 days	CM102	John	4	Radio	National	20%
444	AutumnStyle13	6.9.2013	5 days	CM103	Nancy	6	Print	National	100%
555	AutumnCelebr13	6.9.2013	3 days	CM100	Roberta	3	Radio	Local	100%
????	????	????	????	????	????	7	Internet	National	????

修改异常的例子：
为了将宣传活动 222 的持续时间从 30 天改为 45 天，必须要修改一条记录

删除异常的例子：
不得不在不删除宣传经理 CM103 和宣传模式 6 的情况下，删除宣传活动 444

插入异常的例子：
不能在不插入一个使用模式 7 的实际宣传活动的情况下，插入新的宣传模式 7

图 4-6　关系 AD CAMPAIGN MIX 中的更新异常

4.3.3 插入异常

插入异常（insertion anomaly）是指当用户想要插入关一真实世界中实体的数据时，还必须输入另一个真实世界中的实体的数据。

例如，假设这个广告代理决定在以后的宣传活动中使用一种新的宣传模式：（7，Internet，National）。但这种新的模式并不能直接输入到关系 AD CAMPAIGN MIX 中，用户还必须输入一个使用该模式的宣传活动。然而，此时并没有使用模式 7 的宣传活动。由于 AD CAMPAIGN MIX 是这个数据库中唯一的关系，所以关于宣传模式 7 的数据将不会存在于数据库中，直到它被插入到关系 AD CAMPAIGN MIX 中。而将宣传模式 7 的信息作为一条记录插入到关系 AD CAMPAIGN MIX 中，让其余的列为空，这种做法是不可行的。因为这种插入操作将会导致主码 AdCampaignID 的值为 NULL，而这违反了实体完整性约束。

4.3.4 删除异常

删除异常（deletion anomaly）是指当用户想要删除某一真实世界中实体的数据时，还必须删除另一个真实世界中实体的数据。

例如，假设该广告代理决定取消广告宣传活动 444，并在关系 AD CAMPAIGN MIX 中删除宣传活动 444 的相关记录。这一删除行为同时也会在关系 AD CAMPAIGN MIX 中删除宣传模式（6，Print，National），因为此时没有其他宣传活动使用该模式。又由于宣传模式的信息只能在关系 AD CAMPAIGN MIX 中找到（因为数据库中没有其他的关系），所以，该删除操作最终将会导致宣传模式 6 的信息从数据库中消失。然而，该广告代理在将来的宣传活动中可能还会使用宣传模式 6，因此在数据库中保留宣传模式 6 的信息将是必要的。 95

另外，在关系 AD CAMPAIGN MIX 中删除宣传活动 444 的相关记录，同时也会删除宣传经理（103，Nancy），因为没有其他宣传活动由该宣传经理管理。由于宣传经理的信息只能在关系 AD CAMPAIGN MIX 中找到，该删除操作最终将会导致宣传经理 103 的信息从数据库中消失。然而，该广告代理将来可能还会使用宣传经理 103 来管理广告宣传活动，因此在数据库中保留宣传经理 103 的信息是必要的。

4.3.5 修改异常

修改异常（modification anomaly）是指当用户要修改某一值时，同样的修改操作需要重复多次。

例如，假设该广告代理决定将宣传活动 222 的持续时间从 30 天延长到 45 天。该操作需要在 3 条不同的记录上进行。[⊖]

可能存在更新异常的关系（例如关系 AD CAMPAIGN MIX）可以通过规范化过程来改进。大多数的规范化过程是基于对函数依赖概念的理解进行的。因此，在介绍规范化过程之前，我们先来讨论函数依赖。 96

4.4 函数依赖

函数依赖（functional dependency）是指关系的每条记录中一列（或几列）的值唯一决定

⊖ 这些修改操作只进行了一部分的情况下（如 3 次修改操作只进行了 1 次修改操作）会发生错误，关系中将会有不正确的数据，该宣传活动的持续时间将会有多个版本，这会导致数据一致性问题，见第 6 章的讨论。

这条记录的另一列的值。例如，图 4-7 的客户信息表包括列 ClientID 和 Client Name。

列 ClientID 函数确定列 ClientName，因为某一 ClientID 的值只能和一个 ClientName 的值相关联。另一方面，列 ClientName 不能函数确定列 ClientID，因为某一 ClientName 的值可以与多个 ClientID 的值相关联。例如，在关系 CLIENT 中，有两个不同的客户有同样的名字 William，而每个客户有不同的 ClientID。

CLIENT

ClientID	ClientName	...	...
1001	William	...	...
2001	Matthew	...	...
3001	Lee	...	...
4001	Linda	...	...
5001	William		

图 4-7　关系 CLIENT

函数依赖的表示方法

有几种表示函数依赖的方式。一种方式是用以下符号表示函数依赖：

A → B

箭头左边的列（或几列）确定右边的列（或几列）。例如，图 4-7 中的函数依赖可以表示为：

ClientID → ClientName

另一种表示函数依赖的方式是图形化的，如图 4-8 所示，在表的列上用一个箭头表示同样的函数依赖。

图 4-8　函数依赖的图形化描述

| 97

4.5　函数依赖实例

最开始，在图 4-4 的关系 AD CAMPAIGN MIX 中，我们可以识别出以下函数依赖的集合：

在关系 AD CAMPAIGN MIX 中最开始识别出的函数依赖的集合

（集合 1）CampaignMgrID → CampaignMgrName

（集合 2）ModelID → Media, Range

（集合 3）AdCampaignID → AdCampaignName, StartDate, Duration, CampaignMgrID, CampaignMgrName

（集合 4）AdCampaignName → AdCampaignID, StartDate, Duration, CampaignMgrID, CampaignMgrName

（集合 5）AdCampaignID, ModelID → AdCampaignName, StartDate, Duration, CampaignMgrID, CampaignMgrName, Media, Range, BudgetPctg

（集合 6）AdCampaignName, ModelID → AdCampaignID, StartDate, Duration, CampaignMgrID, CampaignMgrName, Media, Range, BudgetPctg

下面将详细讨论关系 AD CAMPAIGN MIX 中这些函数依赖的集合。

（集合 1）CampaignMgrID → CampaignMgrName

CampaignMgrID 函数确定 CampaignMgrName。在关系 AD CAMPAIGN MIX 的记录中，CampaignMgrID 的每个特定的值总是和相同的 CampaignMgrName 的值一起出现。例如，在关系 AD CAMPAIGN MIX 中，CampaignMgrID 的值 CM100 出现在多条记录中，但它总是和 CampaignMgrName 的值 Roberta 一起出现。这对于任意 CampaignMgrName 都是正确的，每个 CampaignMgrID 的值总是只与一个 CampaignMgrName 的值同时出现。反过来则不成立，因为

我们可以雇佣名字相同的两个宣传经理（例如，我们可以雇佣另一个叫 Roberta 的宣传经理），当然，他们将会有两个不同的 CampaignMgrID 的值。一个 CampaignMgrName 的值不一定只和一个 CampaignMgrID 的值相关联，因此 CampaignMgrName 不能函数确定 CampaignMgrID。

（集合 2）ModeID → Media, Range

每个 ModeID 的值总是只与一个 Media 的值以及一个 Range 的值相关联。例如，在关系 AD CAMPAIGN MIX 中，ModeID 的值 3 出现了多次，但它总是与同一 Media 的值 Radio 以及同一 Range 的值 Local 相关联。另一方面，单独的 Media 或者 Range 的值能与多个不同的 ModeID 的值相关联，因此列 Media 和列 Range 都不能确定列 ModeID。

（集合 3）AdCampaignID → AdCampaignName, StartDate, Duration,
CampaignMgrID, CampaignMgrName

AdCampaignID 的每个值总是只与 AdCampaignName、StartDate、Duration、CampaignMgrID 和 CampaignMgrName 的一个值相关联。例如，AdCampaignID 的值 333 在关系 AD CAMPAIGN MIX 中出现多次，但它总是与相同的 StartDate 的值 6.9.2013 相关联。每个单独的 AdCampaignID 的值总是只与一个 StartDate 的值相关联。反过来则不成立，因为两个或多个不同的宣传活动可能在同一日期开始（例如，宣传活动 333、444 和 555 在同一日期开始）。一个 StartDate 的值不一定只与一个 AdCampaignID 的值相关联，因此 StartDate 不能函数确定 AdCampaignID。列 Duration、CampaignMgrID 和 CampaignMgrName 情况相似，它们也不能函数确定 AdCampaignID。另一方面，列 AdCampaignName 可以确定列 AdCampaignID，见集合 4。

（集合 4）AdCampaignName → AdCampaignID, StartDate, Duration,
CampaignMgrID, CampaignMgrName

AdCampaignName 的每个值总是只与 AdCampaignID、StartDate、Duration、CampaignMgrID 和 CampaignMgrName 的一个值相关联。例如，AdCampaignName 的值 FallBall13 在关系 AD CAMPAIGN MIX 中出现多次，但它总是与相同的 StartDate 的值 6.9.2013 相关联。每个单独的 AdCampaignName 的值总是只与一个 StartDate 的值相关联。反过来则不成立，两个或多个不同的宣传活动可能在同一日期开始（例如，宣传活动 FallBall13、AutumnStyle13 和 AutumnColors13 在同一日期开始）。一个 StartDate 的值不一定只与一个 AdCampaignName 的值相关联，因此 StartDate 不能函数确定 AdCampaignName。列 Duration、CampaignMgrID 和 CampaignMgrName 情况相似，它们不能函数确定 AdCampaignID。另一方面，列 AdCampaignID 可以确定列 AdCampaignName，见集合 3。

（集合 5）AdCampaignID, ModeID → AdCampaignName, StartDate, Duration,
CampaignMgrID, CampaignMgrName,
Media, Range, BudgetPctg

在任何关系中，主码总是能函数确定关系中的其他列。因此，在关系 AD CAMPAIGN MIX 中，组合主码 AdCampaignID 和 ModeID 能函数确定关系中其他所有的列。每个 AdCampaignID 和 ModeID 的组合的值都只与 AdCampaignName、StartDate、Duration、CampaignMgrID、CampaignMgrName、Media、Range 和 BudgetPctg 的一个值相关联。

(集合 8) AdCampaignName, ModeID → AdCampaignID, StartDate, Duration, CampaignMgrID, CampaignMgrName, Media, Range, BudgetPctg

在任何关系中，候选码总是能函数确定关系中的其他列。因此，在关系 AD CAMPAIGN MIX 中，组合候选码 AdCampaignName 和 ModeID 能函数确定关系中其他所有的列。每个 AdCampaignID 和 ModeID 的组合的值都只与 AdCampaignName、StartDate、Duration、CampaignMgrID、CampaignMgrName、Media、Range 和 BudgetPctg 的一个值相关联。

4.6　简化函数依赖

数据库的规范化过程是基于分析数据库中每个关系的函数依赖而进行的。有些函数依赖与数据库的规范化过程不相关，因此我们可以只选择对于进行规范化过程必需的函数依赖进行描述分析。这会减少需要考虑的函数依赖的数目，以此来简化规范化过程。

例如，**平凡函数依赖**（trivial functional dependency）是指一个属性（或属性集）函数确定它自己（例如，A → A 或 A，B → A，B）或它的子集（例如，A，B → A）。平凡函数依赖没有出现在识别出的函数依赖集合中。描述平凡函数依赖是不必要的，并且会导致需要考虑的函数依赖的集合增大且变得复杂。例如，在上面例子的函数依赖的集合中加入平凡函数依赖

99

CampaignMgrID, CampaignMgrName → CampaignMgrName

这意味着增加不必要的混乱。

除了平凡函数依赖外，在不丢失规范化过程所需信息的情况下，还有其他类型的函数依赖可以从图形化的描述中删去。特别是下面将要说明的**增广函数依赖**（augmented functional dependency）和**等价函数依赖**（equivalent functional dependency），可以将其从已有的函数依赖的集合中去掉，以简化规范化过程。

4.6.1　增广函数依赖

假设 Joe 比 Sue 更重是一个真语句。在这种情况下，拿着一袋土豆的 Joe 比拿着同样一袋土豆的 Sue 更重也是一个真语句，但是这在原语句的基础上增加了不必要的语句。增广函数依赖的概念与之类似。

假设关系中存在以下函数依赖：

A → B

这种情况下，包含一个已有函数依赖的函数依赖，如下所示：

A, C → d

这就是一个增广函数依赖。

为了使规范化过程清晰和简单，增广函数依赖通常不出现在函数依赖集合中。因此，集合 5 应该写成下面的形式：

(集合 5) AdCampaignID, ModeID → BudgetPctg

因为在这个集合中所有其他的函数依赖都是不必要的函数依赖。

这是因为函数依赖（来自集合 3）

AdCampaignID → AdCampaignName, StartDate, Duration, CampaignMgrID, CampaignMgrName

存在，函数依赖

AdCampaignID, ModelID → AdCampaignName, StartDate, Duration, CampaignMgrID, CampaignMgrName

是在此基础上增补的，因此应该从集合5中删除。

同样，由于函数依赖（来自集合2）

ModelID → Media, Range

存在，函数依赖

AdCampaignID, ModelID → Media, Range

是在此基础上增补的，因此应该从集合5中删除。

4.6.2 等价函数依赖

考虑下面这种简单的场景，英国情报局虚拟人物James Bond也被叫做特工007是一个真语句；这种情况下，下面两个语句，James Bond是英国情报局最著名的虚拟人物和特工007是英国情报局最著名的虚拟人物，是两个等价的语句。等价函数依赖的概念与之类似。 100

假设关系中存在以下函数依赖：

A → B
B → A

这种情况下，A能确定的B也能确定（反过来也一样）。因此，A → B和B → A是等价函数依赖。此外

A → B, X
B → A, X

是等价函数依赖，并且

Y, A → B, X
Y, B → A, X

也是等价函数依赖。

为了使规范化过程清晰和简单，当存在多个等价函数依赖时，我们可以只选其中一个。因为函数依赖

AdCampaignID → AdCampaignName

和函数依赖

AdCampaignName → AdCampaignID

是等价的，所以集合3与集合4是等价的集合，集合5与集合6也是等价的集合。因此，可以从集合中删除集合4和集合6。

以下是对于关系AD CAMPAIGN MIX最终描述的简化后的函数依赖的集合列表。

在关系 AD CAMPAIGN MIX 中简化后的函数依赖的集合

(集合 1) CampaignMgrID → CampaignMgrName

(集合 2) ModelID → Media, Range

(集合 3) AdCampaignID → AdCampaignName, StartDate, Duration, CampaignMgrID, CampaignMgrName

(集合 5) AdCampaignID, ModelID → BudgetPctg

集合 5：通过删除已存在集合 1 和集合 3 中的增广函数依赖来进行简化
集合 4：与集合 3 等价，删除
集合 6：与集合 5 等价，删除

图 4-9 也被示了简化后的函数依赖的集合。

图 4-9　关系 AD CAMPAIGN MIX 中的函数依赖

4.7　函数依赖类型

作为典型规范化过程的函数依赖可以分成三类：**部分函数依赖**（partial functional dependency）、**完全函数依赖**（full key functional dependency）和**传递函数依赖**（transitive functional dependency）。下面将分别举例说明这三种函数依赖。[⊖]

[101]

4.7.1　部分函数依赖

部分函数依赖（partial functional dependency）是指关系的一列函数依赖于组合主码的一部分。

只有组合主码才有不同的组成部分，而单列的主码则没有。因此，部分函数依赖只出现在关系中有组合主码的情况下。

图 4-9 中，函数依赖

AdCampaignID → AdCampaignName, StartDate, Duration, CampaignMgrID, CampaignMgrName
ModelID → Media, Range

是部分函数依赖。AdCampaignID 是主码 AdCampaignID 和 ModelID 的一部分，因此 AdCampaignID 函数确定关系中，其他列的函数依赖是部分函数依赖。同样，ModelID 也是主码 AdCampaignID 和 ModelID 的一部分，因此 ModelID 函数确定关系中，其他列的函数依赖是部分函数依赖。

4.7.2　完全函数依赖

完全函数依赖（full key functional dependency）是指主码函数确定关系中的其他列，并

⊖ 在本章中，我们给出了部分函数依赖和完全函数依赖的简单定义，该定义足够使人清晰地理解典型的规范化过程。附录 B 给出了更加精确的定义，这些定义更加复杂，但是对于大多数真实世界的应用场景来说是不必要的。

且主码的任意一部分都不能单独确定同样的列。

如果关系中有一个单独（非组合）的主码，那么该主码完全函数确定关系中其他的所有列。然而，如果关系中有一个组合码，该组合码的一部分可以部分确定关系中的列，那么这个主码就不是完全函数确定这些部分列了。在图 4-9 中，函数依赖

AdCampaignID, ModelID → BudgetPctg

是一个完全函数依赖。主码的列 AdCampaignID 和 ModelID 一起函数确定列 BudgetPctg。主码的任何一部分都不能部分确定列 BudgetPctg。

4.7.3 传递函数依赖

传递函数依赖（transitive functional dependency）是指非码列函数确定关系中的其他非码列。图 4-9 中，函数依赖

CampaignMgrID → CampaignMgrName

是一个传递函数依赖。CampaignMgrID 是一个非码列，CampaignMgrName 也是一个非码列。因此，CampaignMgrID 函数确定 CampaignMgrName 是一个传递函数依赖。

图 4-10 指出了关系 AD CAMPAIGN MIX 中的完全函数依赖、部分函数依赖和传递函数依赖。

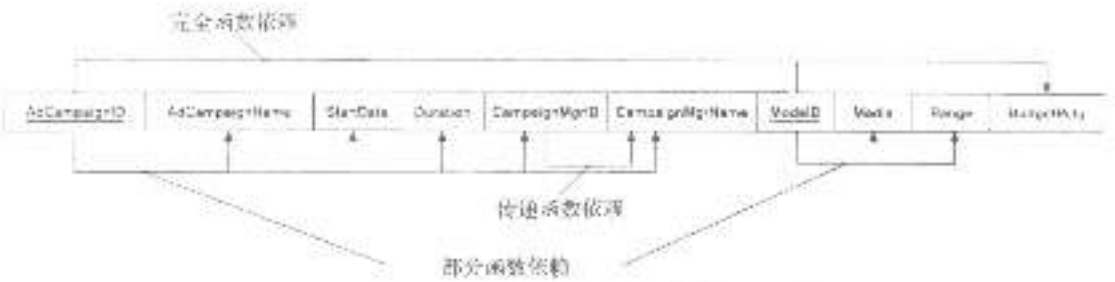


图 4-10 关系 AD CAMPAIGN MIX 中的函数依赖（按照类型的）

4.8 另一个函数依赖实例

识别部分函数依赖、完全函数依赖和传递函数依赖对于进行典型的规范化过程是十分重要的。为了加强对这些概念的理解，我们看一下另一个关系以及它的函数依赖。 [102]

Central Plane 大学使用一个单独的关系 RECRUITING 来存储数据。对于 Central Plane 大学的每个招生人员，关系 RECRUITING 列出了他从某一城市招收的学生数目，以及 Central Plane 大学招收学生所在城市和州的人口数。关系 RECRUITING 由以下列组成。

RecruiterID	招生人员标识符，每个招生人员有一个唯一的值
RecruiterName	招生人员名字
StatusID	招生人员状态标识符（每个招生人员只有一种状态）
StatusName	招生人员状态描述
City	招生人员所招收学生的城市名称（一个招生人员可以在多个城市招收学生，同一个城市也可以有多个招生人员招收学生。多个城市可以有相同的名称，但是一个州内的城市必须有不同的名称。）
State	城市所在州

（续）

StatePopulation	州人口数
CityPopulation	城市人口数
NoOfRecruits	某一招生人员在某一城市招生数目

图 4-11 展示了关系 RECRUITING 和关系中的记录。该关系的主码是组合码 RecruiterID、City 和 State。关系中的每条记录描述了一个招生人员在一个城市的招生记录。因此，由招生人员的唯一属性以及城市和该城市所在州的唯一属性构成的组合属性，能唯一确定关系中的一行。RecruiterID 可以区分出一个招生人员，City 和 State 合起来可以区分一个城市（单独的 City 列是不够的，因为不同的州内可以有相同的城市名称，如图 4-11 所示）。

103

RECRUITING

RecruiterID	RecruiterName	StatusID	Status	City	State	StatePopulation	CityPopulation	NoOfRecruits
R1	Katy	IF	Internal Full Time	Portland	ME	1,350,000	70,000	11
R1	Katy	IF	Internal Full Time	Grand Rapids	MI	9,900,000	190,000	20
R2	Abe	IP	Internal Part Time	Rockford	IL	12,900,000	340,000	17
R3	Jane	CN	Contractor	Spokane	WA	6,800,000	210,000	8
R3	Jane	CN	Contractor	Portland	OR	3,900,000	600,000	30
R3	Jane	CN	Contractor	Eugene	OR	3,900,000	380,000	20
R4	Mana	IF	Internal Full Time	Rockford	IL	12,900,000	340,000	14
R4	Mana	IF	Internal Full Time	Grand Rapids	MI	9,900,000	190,000	9
R5	Don	CN	Contractor	Grand Rapids	MI	9,900,000	190,000	33

图 4-11 关系 RECRUITING

关系 RECRUITING 有以下函数依赖：

RecruiterID → RecruiterName, StatusID, Status
StatusID → Status
State → StatePopulation
City, State → CityPopulation
RecruiterID, City, State → NoOfRecruits

图 4-12 也展示了关系 RECRUITING 中的函数依赖。

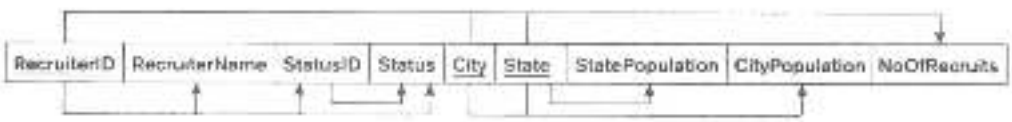


图 4-12 关系 RECRUITING 中的函数依赖

RecruiterID → RecruiterName, StatusID, Status

是一个部分函数依赖，因为 RecruiterID 是主码 RecruiterID、City、State 的一部分，并且函数确定列 RecruiterName、StatusID 和 Status。

StatusID → Status

是一个传递函数依赖，因为非码列 StatusID 可以确定非码列 Status。

City, State → CityPopulation

是一个部分函数依赖，因为 City、State 是主码 RecruiterID、City、State 的一部分，并且函数确定列 CityPopulation。

State → StatePopulation

是一个部分函数依赖，因为 State 是主码 RecruiterID、City、State 的一部分，并且函数确定列 StatePopulation。

RecruiterID, City, State → NoOfRecruits

是一个完全函数依赖，因为主码 RecruiterID、City、State 可以确定列 NoOfRecruits。

4.9 规范化

正如引言中提到的，规范化可用来改进数据库表的设计过程。该过程基于范式（normal form）来进行，每个范式包含一系列表需要满足的条件。范式有很多种，其中最基本的是第一范式（first normal form，1NF）、第二范式（second normal form，2NF）和第三范式（third normal form，3NF）。从较低的范式到较高的范式，范式的条件越来越严格。

规范化过程需要检查每个表以确认它是否满足某一范式。如果满足某一范式，则继续检查它是否满足下一个更高的范式。如果不满足某一范式，则应该采取相应的方法将该表转换成几个表，以满足这个范式。

规范化到第一范式就是把非关系表转换成关系表。规范化到更高的范式（如第二范式、第三范式）是为了改进包含冗余信息的关系表的设计，以减少更新异常的问题。 104

我们使用下面将介绍的非关系表 VET CLINIC CLIENT 来说明第一范式。用本章前面介绍的关系表 AD CAMPAIGN MIX 来说明第二范式和第三范式。

4.9.1 第一范式

第一范式（1NF）要求表中的每一列的每个值都只能是该列范围内的单个值。

1NF

一个表满足 1NF，如果表的每一行都是唯一的并且任何行都没有包含多个值的列。

复习一下第 3 章的内容，每个关系表必须满足以下两个条件：

- 一个表的每一行必须唯一。
- 在每一行中，每一列的值必须是单值。表中的任意一行均不能出现多值的列。

因此，根据定义，每个关系表都是满足 1NF 的。所以，规范化到 1NF 不是关系数据库规范化过程的一部分。规范化到 1NF 只发生在将一个非关系表转换成一个关系表的情况。

图 4-13 展示了一个非关系表 VET CLINIC CLIENT。每个客户可以有多个宠物，在列 PetNo、PetName 和 PetType 上每条记录可以有多个取值。列 PetNo、PetName 和 PetType 是相关的，因为它们都对应真实世界中的宠物的概念。因此，列 PetNo、PetName 和 PetType 组成了一组**相关多值列**（related multivalued column）。[⊖]

VET CLINIC CLIENT

ClientID	ClientName	PetNo	PetName	PetType
111	Lisa	1	Tolu	Dog
222	Lydia	1	Fluffy	Dog
		2	JoJo	Bird
		3	Ziggy	Snake
333	Jane	1	Fluffy	Cat
		2	Coco	Cat

图 4-13 非关系表（非 1NF）

注意在图 4-13 中，对于顾客 Lisa 的记录，

⊖ 一组相关多值列也叫做“重复组”。

列 PetNo，PetName 和 PetType 都只有一个值，但是对于其他顾客来说，这几列仍然有多个取值。在顾客 Lydia 的记录中这几列有三个值，在顾客 Jane 的记录中这几列有两个值。

规范化到 1NF 就是通过消除一条记录中同一列有多个取值的可能性，把一个非关系表变成一个关系表（或一组关系表）。

一种将非关系表转换成 1NF，即转换成关系表的方法，就是对于相关多值列的每个值创建一条单独的记录，如图 4-14 所示。

VET CLINIC CLIENT

ClientID	ClientName	PetNo	PetName	PetType
111	Lisa	1	Tofu	Dog
222	Lydia	1	Fluffy	Dog
222	Lydia	2	JoJo	Bird
222	Lydia	3	Ziggy	Snake
333	Jane	1	Fluffy	Cat
333	Jane	2	Cleo	Cat

图 4-14　通过增加记录将图 4-13 所示的关系表规范化到 1NF

原来的非关系表的每条记录中列 ClientID 和 ClientName 的值都只出现了一次，而在图 4-14 所示转换后的表中，每条记录中列 ClientID 和 ClientName 的值重复出现，且重复次数与该条记录中宠物多值相关列的取值重复的次数相同。例如，在图 4-14 中，值 222，Lydia 重复了三次，因为客户 Lydia 有三个宠物，而值 333，Jane 重复了两次，因为客户 Jane 有两个宠物。通过这种方式，规范化到 1NF 后的关系的主码是由原来的非关系表的主码和一个（或多个）非码列构成的组合码，这里的非码列在相关多值列中有唯一的值。在图 4-14 中，主码是由原来的主码列 ClientID 和非码列 PetNo 构成的。ClientID 和 PetNo 的组合唯一标识了图 4-14 中表的每条记录。

另一种将非关系表规范化到 1NF 的方法是为每一组相关多值列新创建一个单独的表。如图 4-15 所示，新的表 PET 保存了与宠物相关的一组列 PetNo，PetName 和 PetType，这些列在原始表中每条记录有多个取值。新表还包含原始表的主码（ClientID），它既是原始表的外码也是主码。主码的另一部分（PetNo）与外码结合在一起，对于新表的每条记录都有唯一的值。图 4-15 展示了图 4-13 中的表进行上述过程后的结果：

VET CLINIC CLIENT

ClientID	ClientName
111	Lisa
222	Lydia
333	Jane

PET

ClientID	PetNo	PetName	PetType
111	1	Tofu	Dog
222	1	Fluffy	Dog
222	2	JoJo	Bird
222	3	Ziggy	Snake
333	1	Fluffy	Cat
333	2	Cleo	Cat

图 4-15　通过新增关系表将图 4-13 所示的关系表规范化到 1NF

注意，新创建的表 PET 对于相关多值列的每个取值都有一条单独的记录，这与图 4-14 的情况相同。为了将一个表规范化到 1NF，必须为相关多值列的每个取值创建一条记录。

下面我们将考虑一个扩展的例子，它有多组相关多值列。图 4-16 中的表有两组相关多

105

值列。一组相关多值列包括与宠物相关的列 PetNo、PetName 和 PetType，另一组相关多值列包括与家庭成员相关的列 HHMember、Name 和 Relation。

VET CLINIC CLIENT

ClientID	ClientName	PetNo	PetName	PetType	HHMember	Name	Relation
111	Lisa	1	Tofu	Dog	1 2 3	Joe Sally Clyde	Husband Daughter Son
222	Lydia	1 2 3	Fluffy JoJo Ziggy	Dog Bird Snake	1 2	Bill Lilly	Husband Daughter
333	Jane	1 2	Holly Cleo	Cat Cat	1	Jill	Sister

图4-16 有两组相关多值列的非关系表（非1NF）

在一个表有多组相关多值列的情况下，需要为每组相关多值列创建一个单独的表来规范化到1NF。图4-17说明了这点。请注意，在图4-17中，每个新创建的表的组合主码都包含了原始表的主码。表 HOUSEHOLD MEMBER 的主码包括 ClientID（原始表的主码）以及 HHMember，表 PET 的主码包括 ClientID（原始表的主码）以及 PetNo。 106

VET CLINIC CLIENT

ClientID	ClientName
111	Lisa
222	Lydia
333	Jane

PET

ClientID	PetNo	PetName	PetType
111	1	Tofu	Dog
222	1	Fluffy	Dog
222	2	JoJo	Bird
222	3	Ziggy	Snake
333	1	Fluffy	Cat
333	2	Cleo	Cat

HOUSEHOLD MEMBER

ClientID	HHMember	Name	Relation
111	1	Joe	Husband
111	2	Sally	Daughter
111	3	Clyde	Son
222	1	Bill	Husband
222	2	Lilly	Daughter
333	1	Jill	Sister

图4-17 将图4-16所示的关系表规范化到1NF

需要记住规范化到1NF并不是关系数据库中关系规范化过程的一部分，正如我们前面提到的，关系数据库中的每个关系都已经满足1NF。规范化到1NF只发生在将非关系表转换成关系表的情况，关系数据库表的规范化过程是从第二范式（2NF）开始的。

4.9.2 第二范式

对于一个关系表，规范化过程是从检查它是否满足第二范式（2NF）开始的。

2NF

如果一个表满足1NF且**不包含部分函数依赖**，则这个表满足2NF。

部分依赖是指组合主码的一部分可以函数确定关系的一列。如果一个关系有单列的主码，那么这个关系中就不可能存在部分函数依赖。这种关系本身就满足2NF，因此不需要把它规范化到2NF。然而，有组合主码的关系则可能存在部分函数依赖。如果一个关系中存在部分函数依赖，那么它就不满足2NF，需要被规范化到2NF。

将一个关系规范化到2NF会为关系中的每个部分依赖集合创建一个额外的关系。这些额外创建的关系的主码是原关系的主码中可以函数确定其他列的部分。原关系中被部分确定 107

的列也是新创建的表的一部分。在规范化到 2NF 过程之后，原始表仍然保留，但它不再包含部分函数依赖的列。

我们将使用关系 AD CAMPAIGN MIX 来说明规范化到 2NF 的过程。考虑图 4-10 所示的函数依赖，关系 AD CAMPAIGN MIX 中的部分函数依赖如下：

AdCampaignID → AdCampaignName, StartDate, Duration, CampaignMgrID, CampaignMgrName

ModelID → Media, Range

图 4-18 展示了将关系 AD CAMPAIGN MIX 规范化到 2NF 后的结果。关系 AD CAMPAIGN MIX 中的部分依赖被消除了。修改后的关系 AD CAMPAIGN MIX 包含了完全依赖：

AdCampaignID, ModelID → BudgetPctg

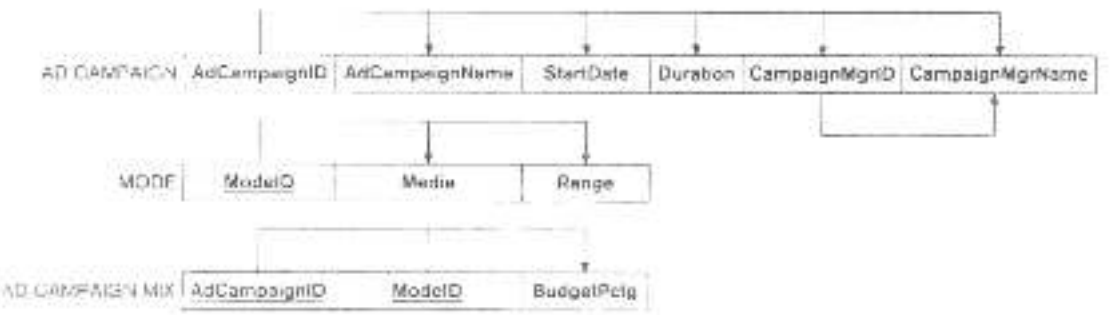


图 4-18　Pressly AdAgency：规范化到 2NF

对于每个部分依赖集合，都创建了一个单独的表。

新的关系 AD CAMPAIGN 包含了前面的部分依赖：

AdCampaignID → AdCampaignName, StartDate, Duration, CampaignMgrID, CampaignMgrName

新的关系 MODE 包含了前面的部分依赖：

ModelID → Media, Range

修改后的关系 AD CAMPAIGN MIX 不再包含函数依赖部分主码的列（即部分依赖的列）。

4.9.3　第三范式

对于一个满足 2NF 的关系表来说，接下来的规范化过程是检查它是否满足第三范式（3NF）。

3NF

如果一个表满足 2NF 且不包含传递函数依赖，则这个表满足 3NF。

传递依赖是指一个关系的非码列确定另一个非码列。如果一个关系有传递依赖，那么它不满足 3NF，需要被规范化到 3NF。

将一个关系规范化到 3NF 会为关系中的每个传递依赖集合创建一个额外的关系。这些额外创建的关系的主码是原始表中可以函数确定其他非码列的一个（或多个）非码列。原关系中被传递确定的非码列也是新创建的表的一部分。在规范化到 3NF 过程之后，原始表仍然保留，但它不再包含传递函数依赖的列。

108

我们将使用关系 AD CAMPAIGN MIX 来说明规范化到 3NF 的过程。考虑图 4-18 所示的函数依赖。关系 AD CAMPAIGN MIX 和 Mode 不包含传递依赖，它们已经满足 3NF。而关系 AD CAMPAIGN 包含以下传递依赖：

CampaignMgrID → CampaignMgrName

因此，它不满足 3NF。将关系 AD CAMPAIGN 规范化到 3NF 后的结果如图 4-19 所示。关系 AD CAMPAIGN 中的传递依赖被消除了，并且创建了一个单独的关系 CAMPAIGN MANAGER。该关系包含以前的传递依赖。

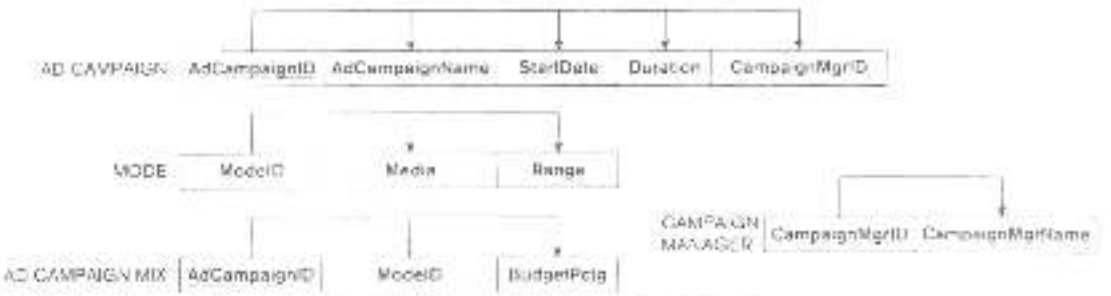


图 4-19 Pressly AdAgency：规范化到 3NF

修改后的关系 AD CAMPAIGN 将不再包含传递依赖的列。

图 4-20 展示了图 4-19 中满足 3NF 关系的数据记录。可将图 4-20 中的记录与图 4-4 中的记录相比较，这两个图都展示了同样的数据。在图 4-4 中，数据存储在容易发生更新异常的非规范化关系中。而在图 4-20 中，同样的数据包含在 4 个规范化的表中，不容易发生更新异常。

AD CAMPAIGN

AdCampaignID	AdCampaignName	StartDate	Duration	CampaignMgrID
111	SummerFun13	6.6.2013	12 days	CM100
222	SummerZing13	6.8.2013	30 days	CM101
333	FallBall13	6.9.2013	12 days	CM102
444	AutumnStyle13	6.9.2013	5 days	CM103
555	AutumnColors13	6.9.2013	3 days	CM100

CAMPAIGN MANAGER

CampaignMgrID	CampaignMgrName
CM100	Roberta
CM101	Sue
CM102	John
CM103	Nancy

MODE

ModeID	Media	Range
1	TV	Local
2	TV	National
3	Radio	Local
4	Radio	National
5	Print	Local
6	Print	National

图 4-20 Pressly AdAgency：规范化后的关系数据

AD CAMPAIGN MIX

AdCampaignID	ModeID	BudgetPctg
111	1	50%
111	2	50%
222	1	60%
222	3	30%
222	5	10%
333	3	80%
333	4	20%
444	6	100%
555	3	100%

图 4-20（续）

注意，将一个关系从 2NF 规范化到 3NF 的过程中，开始只有一个关系，最后会得到多个关系，这些关系是通过它们之间的参照完整性约束联系起来的。图 4-19 和图 4-20 展示的四个关系是将关系 AD CAMPAIGN MIX（来自图 4-4）规范化后的结果。图 4-21 展示了规范化后关系之间的参照完整性约束的关系模式。

图 4-21　Pressly AdAgency：3NF 关系的关系模式

4.9.4　其他范式

在大多数真实的业务问题中，按照本章所述的方法，通过去除部分和传递依赖规范化到 3NF，这对于消除不必要的冗余以及更新异常的威胁来说已经足够了。3NF 之上更高的范式是基于其他类型的函数依赖（完全、部分和传递函数依赖之外）进行的，通常只具有理论价值。详细信息见附录 B。

4.9.5　消除冗余和更新异常

109 ~ 110

正如本章前面提到的，在包含冗余数据的关系中，更新操作可能会引起更新异常。我们将使用 Pressly 广告代理的例子来说明规范化如何消除冗余和更新异常。

观察图 4-20 所示的记录。每个宣传活动的名称、开始日期和持续时间只列出一次，每个宣传经理的名字只列出了一次，每个宣传模式的媒体和覆盖范围的值也只列出一次。只有外码的值对于同一事件重复出现了多次，例如在关系 AD CAMPAIGN 的列 CampaignMgrID

中，值CM100出现了两次。在关系AD CAMPAIGN MIX的列AdCampaignID中，值222出现了三次。在满足3NF的关系表中多次出现的外码值代表必要的冗余（necessary redundancy），这对于连接各个表来说是必要的。

下面这个例子（见图4-22）说明了规范化过程如何解决更新异常问题。

AD CAMPAIGN

AdCampaignID	AdCampaignName	StartDate	Duration	CampaignMgrID
111	SummerFun13	6.6.2013	12 days	CM102
222	SummerZing13	6.8.2013	45 days	CM101
333	FallBal13	8.9.2013	12 days	CM102
555	AutumnColors13	8.9.2013	3 days	CM100

修改异常已解决：
只修改了一条记录

CAMPAIGN MANAGER

CampaignMgrID	CampaignMgrName
CM100	Roberta
CM101	Sue
CM102	John
CM103	Nancy

删除异常已解决：
删除宣传444，但宣传经理CM103和宣传模式6信息仍保留在数据库中

MODE

ModeID	Media	Range
1	TV	Local
2	TV	National
3	Radio	Local
4	Radio	National
5	Print	Local
6	Print	National
7	Internet	National

插入异常已解决：
新的宣传模式7被录入，尚没有插入使用新模式7的实际活动

AD CAMPAIGN MIX

AdCampaignID	ModeID	BudgetPctg
111	1	50%
111	2	50%
222	1	60%
222	3	30%
222	5	10%
333	3	80%
333	4	20%
555	3	100%

图4-22 Pressly AdAgency：解决更新异常后的规范化关系数据

为了说明规范化如何解决插入异常问题，考虑下面的例子。假设该广告代理决定为以后的宣传活动增加一种宣传模式，ModeID：7，Media：Internet，Range：National。这个新的模式不能直接输入到图4-4的关系AD CAMPAIGN MIX中，还必须同时输入一个实际使用该模式的宣传活动。而在图4-22中，新的（宣传模式7，Internet，National）很容易输入到关系MODE中，不需要输入其他信息。 [111]

为了说明规范化如何解决删除异常问题，考虑下面的例子。如果该广告代理决定取消广

者宣传活动444，那么在图4-4的关系AD CAMPAIGN MIX中删除关于宣传活动444的记录。同时还将删除（宣传模式6，Print，National）（因为此时没有其他宣传活动使用该模式）以及（宣传经理103，Nancy）（因为此时没有其他宣传活动是由她管理的）。另一方面，对于图4-22中同样的场景，从关系AD CAMPAIGN中删除关于宣传活动444的记录（也从关系AD CAMPAIGN MIX中删除该宣传活动的记录），但（宣传模式6，Print，National）的记录仍保留在关系MODE中，（宣传经理103，Nancy）的记录仍保留在关系CAMPAIGN MANAGER中。

为了说明规范化如何解决修改异常问题，考虑下面的例子。如果该广告代理决定将宣传活动222的持续时间从30天延长到45天，在图4-4的关系AD CAMPAIGN MIX中，需要对三条记录进行修改。另一方面，对于图4-22中同样的场景，只需要在关系AD CAMPAIGN中修改宣传活动222所应的一条记录。

4.10　另一个规范化实例

为了说明前面介绍的概念，我们将说明图4-11所示的关系RECRUITING从2NF规范化到3NF的过程。

考虑图4-12中的函数依赖，关系RECRUITING中的部分依赖如下：

RecruiterID → RecruiterName, StatusID, Status
City, State → CityPopulation
State → StatePopulation

将关系RECRUITING规范化到2NF的结果如图4-23所示。关系中的部分依赖被消除，对于每个部分依赖集合都创建了一个单独的表。

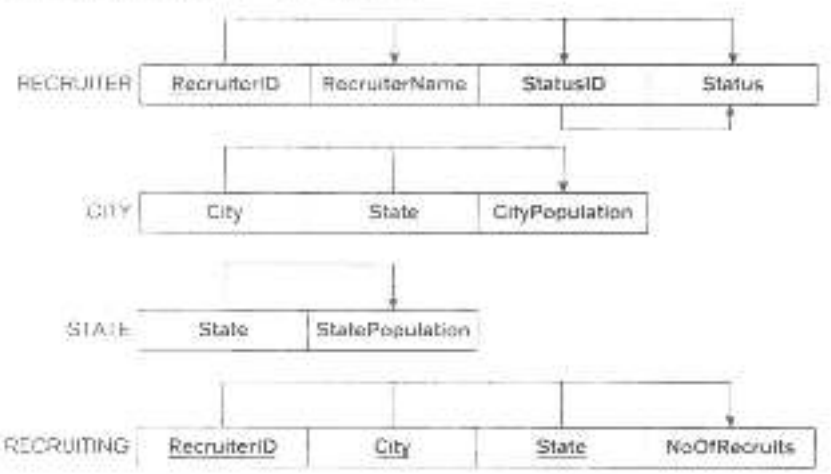


图4-23　Central Plane University：规范化到2NF

考虑图4-23所示的函数依赖，关系CITY、STATE和RECRUITING都没有传递依赖，已经满足3NF。而关系RECRUITER有下面的传递依赖：

112

StatusID → Status

因此必须将它规范化到3NF。将关系RECRUITER规范化到3NF的结果如图4-24所示。关系RECRUITER中的传递依赖被消除了，并且创建了一个单独的表STATUS，该表包含了之

前的传递依赖。

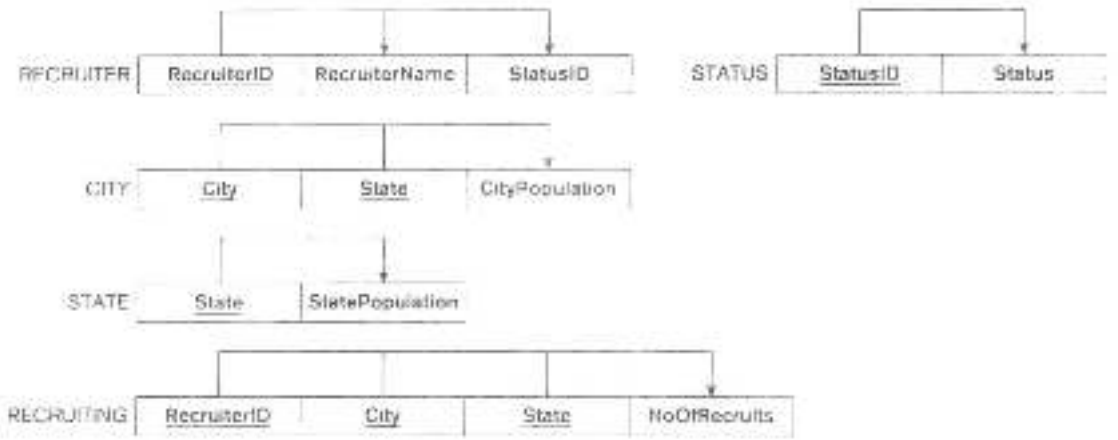


图 4-24　Central Plains University：规范化到 3NF

图 4-25 展示了图 4-24 所示的满足 3NF 的关系记录。

113

RECRUITER

RecruiterID	RecruiterName	StatusID
R1	Katy	I
R2	Ana	IP
R3	Jane	CN
R4	Maria	IF
R5	Dan	CN

STATE

State	StatePopulation
ME	1,350,000
MI	9,900,000
IL	12,900,000
WA	6,800,000
OR	3,900,000
MN	5,400,000

STATUS

StatusID	Status
CN	Contractor
IF	Internal Full Time
IP	Internal Part Time

RECRUITING

RecruiterID	City	State	NoOfRecruits
R1	Portland	ME	11
R1	Grand Rapids	MI	20
R2	Rockford	IL	17
R3	Spokane	WA	8
R3	Portland	OR	30
R3	Eugene	OR	20
R4	Rockford	IL	14
R4	Grand Rapids	MN	9
R5	Grand Rapids	MI	33

CITY

City	State	CityPopulation
Portland	ME	70,000
Grand Rapids	MI	190,000
Rockford	IL	340,000
Spokane	WA	210,000
Portland	OR	630,000
Eugene	OR	380,000
Grand Rapids	MN	11,000

图 4-25　RECRUITING：规范化后的关系数据

比较图 4-25 和图 4-11，这两个图展示了同样的数据。在图 4-11 中，数据存储在包含冗余数据且容易发生更新异常的非规范化关系中。而在图 4-20 中，这些数据包含在 5 个规范化表中，不包含冗余数据且不容易发生更新异常。

本章介绍了关于更新操作和规范化的大部分基本问题。下面几节是关于规范化的几个额外的问题。

4.11　问题说明：规范化例外情况

通常情况下，我们期望将数据库关系规范化到 3NF 以消除数据冗余和避免更新异常。然而，我们需要根据实际情况明智而审慎地决定是否将关系规范化到 3NF。在某些情况下，我们不必把某些关系规范化到 3NF。

考虑图 4-26 所示的关系 SALES AGENT。

SALES AGENT

SAgentID	SAgentName	City	State	ZipCode
SA1	Rose	Glen Ellyn	IL	60137
SA2	Sidney	Chicago	IL	60611
SA3	James	Chicago	IL	60610
SA4	Violet	Wheaton	IL	60187
SA5	Nicole	Kenosha	WI	53140
SA6	Justin	Milwaukee	WI	53201

图 4-26　SALES AGENT

关系 SALES AGENT 中的函数依赖如图 4-27 所示。由于传递依赖

ZipCode → City, State

存在，关系 SALES AGENT 满足 2NF 但不满足 3NF。

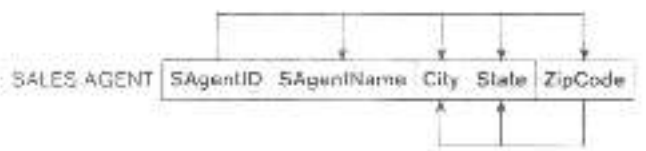


图 4-27　SALES AGENT 中的函数依赖

我们当然可以把该关系规范化到 3NF，如图 4-28 所示。然而，我们也可以将关系 SALES AGENT 保持原样。要做这个决定，则需要评估列 City 和 State 不出现在表 SALES AGENT 中所带来的好处是否值得创建一个单独的关系 ZIPCODE-CITY，因为增加另一个表并在这个表上增加参照完整性约束会增加整个关系模式的复杂性。例如，如果相同邮政编码的销售代理的数目很少，那么规范化到 3NF 所带来的好处将会很小。在这种情况下，我们可以认为在原关系中保留列 State 和 City 而引起的冗余是可以接受的，因此保持原关系不变。另一个例子，考虑一个公司将每个州的销售代理人数限制为最多 2 名的情况。在这种情况下，原关系中冗余的可能性更小，因此更不需要规范化。

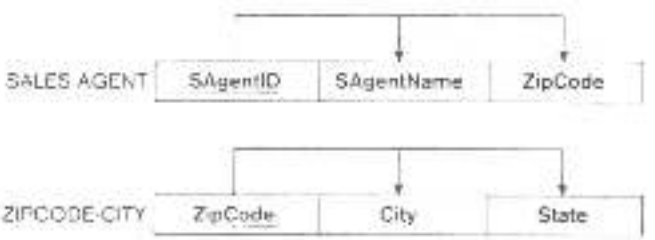


图 4-28　SALES AGENT 中的 3NF

当然，这些例子并不意味着规范化过程总是可选的。准则是：在通常情况下将关系数据库规范化到 3NF，但同时如果有清晰合理的原因，允许存在某些例外。

4.12　问题说明：逆规范化的规范化与性能

在规范化过程中，包含冗余数据列的较大关系被分解成较小的关系。这个过程的后果之一是规范化之前属于少数几个关系的数据，在规范化之后会分布到更多的关系中。这会影响数据检索的性能。我们将用下面的例子来说明这种影响。

比较图 4-4 和图 4-20。图 4-20 中的关系是图 4-4 中关系的一个规范化的版本。假设 Pressly 广告代理最常用的检索之一如下所示：

对于某一广告活动的每种宣传模式，返回以下的列：AdCampaignID，AdCampaignName，CampaignMgrID，CampaignMgrName，ModeID，Media，Range 和 BudgetPctg。

该检索的结果如图 4-29 所示。

RETRIEVED DATA

AdCampaignID	AdCampaignName	CampaignMgrID	CampaignMgrName	ModeID	Media	Range	BudgetPctg
111	SummerFun13	CM100	Roberta	1	TV	Local	50%
111	SummerFun13	CM100	Roberta	2	TV	National	50%
222	SummerZing13	CM101	Sue	1	TV	Local	60%
222	SummerZing13	CM101	Sue	3	Radio	Local	30%
222	SummerZing13	CM101	Sue	5	Print	Local	10%
333	FallBall13	CM102	John	3	Radio	Local	80%
333	FallBall13	CM102	John	4	Radio	National	20%
444	AutumnStyle13	CM103	Nancy	6	Print	National	100%
555	AutumnColors13	CM100	Roberta	3	Radio	Local	100%

图 4-29　Pressly Ad Agency：检索的数据

从图 4-4 的表中检索这些信息只需简单地从表 AD CAMPAIGN MIX 中选择这 8 列即可。然而，从图 4-20 所示的表中检索同样的信息，则需将关系 AD CAMPAIGN、CAMPAIGN MANAGER、MODE 和 AD CAMPAIGN MIX 连接[⊖]，然后再从连接的结果中选择所需的 8 列。连接关系从连接结果中检索数据的过程，要比从一个非规范化的表中检索数据花费更多的时间。因此，一些检索操作（如图 4-29 所示的例子）在非规范化的数据库中比在规范化的数据库中要快。在这种情况下，就需要在减少冗余和更快的检索（性能）之间进行权衡。 115

逆规范化（denormalization）是指规范化过程的逆过程，它把多个规范化的表连接成一个非规范化的表。逆规范化可以用来处理规范化与性能的权衡问题。例如，可以在关系数据库中使用一个规范化的主要版本，所有的数据插入、修改和删除操作都在这个版本中进行以避免更新异常。同时，数据库中还有一个物理的逆规范化的副本，可以从主版本定期生成和存储，用于更快的检索。所有的更新操作都在主版本上进行，不易发生更新异常。数据检索操作可以在逆规范化的副本上进行，以获得更好的性能。在更新数据很少但读数据很多的情况下，这种设置会很方便。逆规范化的副本可以根据用户对于最近数据的需求来重新创建[⊜]。

⊖ 连接操作将在第 5 章中详细介绍。

⊜ 逆规范化方法可以用于数据仓库（见第 7、8、9 章）和物化视图（见第 10 章）。

逆规范化不是在所有情况下都可以采取的默认过程。相反，需要在分析利弊之后审慎而审慎地使用逆规范化。

4.13　问题说明：ER 建模和规范化

正如本书第 2 章和第 3 章所提到的，ER 建模并将 ER 模型映射成关系模式是最常用的数据库设计方法之一。面对如图 4-4 所示的非规范化的表时，除了识别函数依赖，然后进行规范化过程 2NF 和 3NF，设计者还可以分析该表并且根据它创建一个 ER 图，然后将该 ER 图映射成关系模式。

再看一下图 4-4 中的关系表。数据库设计者分析该表（查看列，检查表中的数据记录以及与该表的用户交谈）将会创建以下需求。

Pressly 广告代理的数据库需要记录以下数据：

- 对于每个广告宣传活动：AdCampaignID（唯一）、AdCampaignName（唯一）、开始日期和宣传活动的持续时间。
- 对于每个广告宣传经理：CampaignMgrID（唯一）和 CampaignMgrName（唯一）。
- 对于每种广告宣传模式：ModeID（唯一）、Media 和 Range。
- 每个广告宣传活动只由一个宣传经理管理。
  每个宣传经理管理至少管理一个广告宣传活动，但也可以管理多个广告宣传活动。
- 每个广告宣传活动至少使用一种宣传模式，但也可以使用多种宣传模式。
  一种广告宣传模式可以被多个广告宣传活动使用，也可以不被任何广告宣传活动使用。
- 一个广告宣传活动每使用一种宣传模式，将记录该宣传活动为这种宣传模式分配的预算的百分比。

基于以上的数据库需求，将创建如图 4-30 所示的 ER 图。将图 4-30 所示的 ER 图映射成关系模式的结果如图 4-31 所示。

116

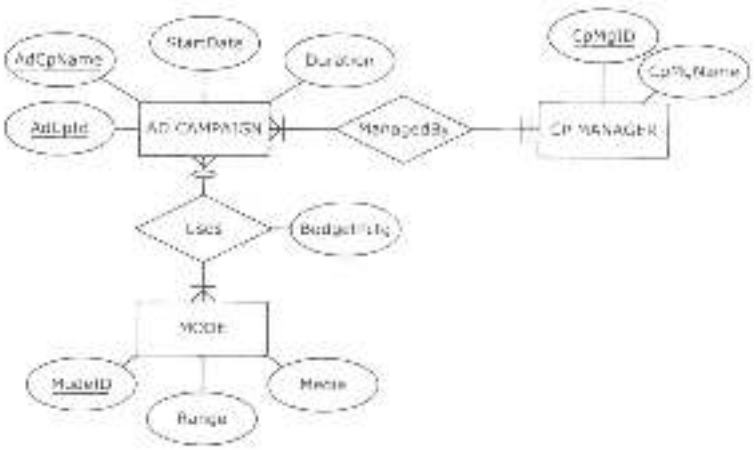


图 4-30　Pressly Ad Agency 的 ER 图

注意，图 4-31 中的关系模式与图 4-21 中通过将关系 AD CAMPAIGN MIX 规范化到 3NF 而得到的关系模式完全相同。

正确使用 ER 建模技术时，将会得到包含设计良好且足够规范化到 3NF 的关系表的关系模式。例如，如果检查第 3 章的图 3-32 或图 3-59 所示关系模式中的关系，可以发现这些模

式中的所有关系都是满足 3NF 的。

图 4-31 将 Pressly Ad Agency 的 ER 图映射为关系模式

规范化作为一个明确的过程，对于一个正确设计的数据库项目来说，并不是一个必需的部分。只有在面对设计不正确且需要改进的关系时，才需要将数据库规范化到 3NF。这种情况并不少见，因此熟悉规范化过程是很重要的。

4.14 问题说明：用于简化数据库内容的设计者添加的实体（表）和码

在某些情况下，即使一个关系已经满足 3NF，数据库内容仍然可以简化。以下是使用**设计者添加的实体 / 表**（designer-added entity/table）和**设计者添加的码**（designer-added key），来达到这一目的的简要讨论。

117

考虑图 4-20 中的关系 MODE。该关系满足 3NF，每条记录代表一个不同的宣传模式。正如前面提到的，关系 MODE 可以通过将图 4-30 中的 ER 图映射成相应的关系来得到。数据库设计者在与最终用户讨论后，可以将关系 MODE 中每个宣传模式的描述媒体和覆盖范围中重复的文本型的值替换成更短的数字型的值。这种情况下，数据库设计者将会为图 4-30 中的 ER 图增加以下需求（新的需求用下划线标识）。

Pressly 广告代理的数据库需要记录以下数据：

- 对于每个**广告宣传活动**：AdCampaignID（唯一）、AdCampaignName（唯一）、开始日期和宣传活动的持续时间。
- 对于每个**广告宣传经理**：CampaignMgrID（唯一）和 CampaignMgrName（唯一）。
- 对于每种**广告宣传模式**：ModeID（唯一）、Media 和 Range。
- 对于每个**广告宣传媒体**：MediaID（唯一）和 Media。
- 对于每个**广告宣传的覆盖范围**：RangeID（唯一）和 Range。
- 每个广告宣传活动只由一个宣传经理管理。
  每个宣传经理至少管理一个广告宣传活动，但也可以管理多个广告宣传活动。
- 每个广告宣传活动至少使用一种宣传模式，但也可以使用多种宣传模式。
  一种广告宣传模式可以被多个广告宣传活动使用，也可以不被任何广告宣传活动使用。
- 一个广告宣传活动每使用一种宣传模式，将记录该宣传活动为这种宣传模式分配的预算的百分比。
- 每种广告宣传模式包含一种媒体，每种广告媒体可以被包含在一种或多种宣传模式中。

- 每种广告宣传模式包含一个覆盖范围，每个广告覆盖范围可以被包含在一种或多种宣传模式中。

基于新增加的需求，如图 4-32 所示，将会通过增加两个设计者添加的实体来增加原 ER 图，因此会在广告代理数据库中创建两个设计者添加的表，如图 4-33 所示。

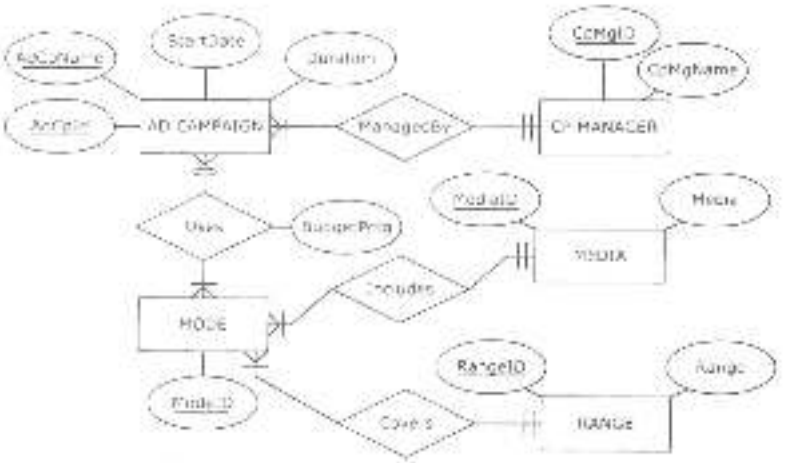


图 4-32 将 Pressly Ad Agency 的例子扩展的 ER 图

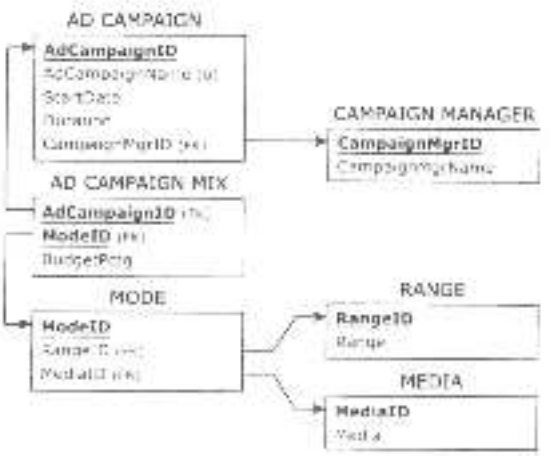


图 4-33 将 Pressly Ad Agency 扩展的 ER 图映射为关系模式

设计者添加的码 MediaID 和 RangeID 将会分别作为设计者添加的表 MEDIA 和 RANGE 的主码。

图 4-34 展示了增加后的 AD CAMPAIGN MIX 数据库中的记录。值 TV、Radio 和 Print 在表 MEDIA 中只被列出一次，而这些值对应的外码在表 MODE 中重复出现了多次。同样，文本型的值 Local 和 National 在表 RANGE 中只被列出一次，而这些值对应的外码在表 MODE 中重复出现了多次。

118

119

与逆规范化的情况一样，不能草率地使用设计者添加的码和表来增加数据库，对于每一次增加，都应该在分析利弊之后谨慎进行。

AD CAMPAIGN

AdCampaignID	AdCampaignName	StartDate	Duration	CampaignMgrID
111	SummerFun13	6.8.2013	12 days	CM100
222	SummerFing13	6.8.2013	30 days	CM101
333	FallBal 3	6.8.2013	12 days	CM102
444	AutumnStyle13	6.8.2013	5 days	CM103
555	AutumnColors13	6.8.2013	3 days	CM100

CAMPAIGN MANAGER

CampaignMgr D	CampaignMgrName
CM100	Roberta
CM101	Sue
CM102	Ann
CM103	Nancy

RANGE

RangeID	Range
L	Local
N	National

MEDIA

MediaID	Media
T	TV
R	Radio
P	Print

AD CAMPAIGN MIX

AdCampaignID	ModelID	BudgetPctg
111	1	50%
111	2	50%
222	1	60%
222	2	30%
222	5	10%
333	5	80%
333	4	20%
444	6	100%
555	3	100%

MODE

ModelID	Media	Range
1	T	L
2	T	N
3	R	L
4	R	N
5	P	L
6	P	N

图4-24 Peasly Ad Agency：图4-33中关系的数据记录

关键术语

augmented functional dependency（增广函数依赖），100
delete operation（删除操作），91
deletion anomaly（删除异常），95
denormalization（逆规范化），116
designer-added entity/table（设计者添加的实体/表），117
designer-added key（设计者添加的码），117
equivalent functional dependency（等价函数依赖），100
first normal form，1NF（第一范式），104
full key functional dependency（完全函数依赖），101
functional dependency（函数依赖），97
insert operation（插入操作），91
insertion anomaly（插入异常），95
modification anomaly（修改异常），96
modify operation（修改操作），91
necessary redundancy（必要的冗余），111
nonkey column（非码列），95
normal form（范式），104
normalization（规范化），91
partial functional dependency（部分函数依赖），101
read operation（读操作），91
redundant data（冗余数据），91
related multivalued column（相关多值列），105
second normal form，2NF（第二范式），104
third normal form，3NF（第三范式），104
transitive functional dependency（传递函数依赖），101
trivial functional dependency（平凡函数依赖），99
update operation（更新操作），91
update anomaly（更新异常），91
write operation（写操作），91

复习题

Q4.1　三个更新操作分别是什么？
Q4.2　什么是冗余数据？
Q4.3　何时会产生插入异常？
Q4.4　何时会产生删除异常？
Q4.5　何时会产生修改异常？
Q4.6　什么是函数依赖？
Q4.7　什么是部分函数依赖？
Q4.8　什么是完全函数依赖？
Q4.9　什么是传递函数依赖？
Q4.10　给出 1NF 的定义。
Q4.11　给出 2NF 的定义。
Q4.12　给出 3NF 的定义。
Q4.13　什么是逆规范化？它的目的是什么？
Q4.14　什么是设计者添加的实体、表和列？

练习

E4.1　写出一个容易产生更新异常的关系（包含多条记录）的例子。
E4.2　使用 E4.1 中的关系，说明一个插入异常的例子。
E4.3　使用 E4.1 中的关系，说明一个删除异常的例子。
120　E4.4　使用 E4.1 中的关系，说明一个修改异常的例子。
E4.5　考虑下面的关系和数据。

AIRPORT KLX TABLE

Date	AirlineID	AirlineName	TerminalID	NumberOfGates	NumberOfDepartingFlights
11-Dec	UA	United	A	20	34
11-Dec	NW	Northwest	A	20	17
11-Dec	AA	American	A	20	11
11-Dec	DL	Delta	B	15	20
11-Dec	JB	Jet Blue	B	15	6
12-Dec	UA	United	A	20	29
12-Dec	DL	Delta	B	15	20
12-Dec	SWA	Southwest	C	15	17

- AIRPORT KLX 表记录了 KLX 机场每天起飞航班的数据。
- KLX 机场的每家航空公司都有一个唯一的航空公司 ID 和航空公司名称。
- KLX 机场的航站楼都有一个唯一的航站楼 ID 和固定数目的登机口。
- 每家航空公司被永久分配使用 KLX 机场的一个（且仅有一个）航站楼。
- 可以将多家航空公司分配到 KLX 机场的同一个航站楼。
- 该表记录了每天每家航空公司在 KLX 机场起飞的航班数。

E4.5a　使用表 AIRPORT KLX，描述一个插入异常的例子。
E4.5b　使用表 AIRPORT KLX，描述一个删除异常的例子。
E4.5c　使用表 AIRPORT KLX，描述一个修改异常的例子。
E4.5d　列出表 AIRPORT KLX 中的完全函数依赖、部分函数依赖（如果存在）和传递数依赖（如果

存在）。

E4.5e 写出将表 AIRPORT KLX 规范化到 2NF 的结果。

E4.5f 写出将表 AIRPORT KLX 规范化到 3NF 的结果。

E4.5g 使用 E4.5f 得到的表，说明 E4.5a、E4.5b 和 E4.5c 中的异常是如何被消除的。

E4.6 考虑下面的关系和数据：

DEPARTMENT OF TRANSPORTATION (DOT) PROJECT TABLE

ProjectID	ProjectName	CountyID	County name	ProjectManagerID	ProjectManagerName	ProjectMilesWithinCounty
1	Road X	1	Wilson	M1	Bob	10.00
1	Road X	2	Ottawa	M1	Bob	17.00
1	Road X	3	Davis	M1	Bob	12.00
2	Road Y	3	Davis	M2	Sue	23.00
3	Bridge A	1	Wilson	M3	Lee	0.50
3	Bridge A	2	Ottawa	M3	Lee	0.30
4	Tunnel Q	2	Ottawa	M1	Bob	2.00
5	Road W	4	Perry	M4	Bob	23.00

- DEPARTMENT OF TRANSPORTATION(DOT) 项目表记录了项目和项目长度（英里）的数据。
- 每个项目都有唯一的项目 ID 和项目名称。
- 每个县都有唯一的县 ID 和县名称。
- 每个项目经理都有唯一的项目经理 ID 和名字。
- 每个项目只有一个项目经理。
- 一个项目经理可以管理多个项目。
- 一个项目可以横跨多个县。
- 该表的列 ProjectMilesWithinCounty 记录了一个项目在某一县内的长度。

121

E4.6a 使用表 DOT PROJECT，描述一个插入异常的例子。

E4.6b 使用表 DOT PROJECT，描述一个删除异常的例子。

E4.6c 使用表 DOT PROJECT，描述一个修改异常的例子。

E4.6d 列出表 DOT PROJECT 中的完全函数依赖、部分函数依赖（如果存在）和传递函数依赖（如果存在）。

E4.6e 写出将表 DOT PROJECT 规范化到 2NF 的结果。

E4.6f 写出将表 DOT PROJECT 规范化到 3NF 的结果。

E4.6g 使用 E4.6f 得到的表，说明 E4.6a、E4.6b 和 E4.6c 中的异常是如何被消除的。

E4.7 考虑下面的关系和数据：

SHIPMENTS TABLE

ShipmentID	ShipmentDate	TruckID	TruckType	ProductID	ProductType	Quantity
111	1-Jan	T-111	Semi-Trailer	A	Tire	160
111	1-Jan	T-111	Semi-Trailer	B	Battery	120
222	2-Jan	T-222	Van Truck	C	SparkPlug	10,000
333	3-Jan	T-333	Semi-Trailer	D	Wipers	5,000
333	3-Jan	T-333	Semi-Trailer	A	Tire	200
444	3-Jan	T-222	Van Truck	C	SparkPlug	25,000
555	3-Jan	T-111	Semi-Trailer	B	Battery	180

- 一家汽车配件公司的表 SHIPMENTS 记录了出货数据。
- 每次出货都有唯一的出货 ID 和出货日期。
- 每次出货都有一辆卡车运输。

- 每辆卡车都有唯一的卡车 ID 和卡车类型。
- 每次出货可以运输多种产品。
- 每种产品都有唯一的产品 ID 和产品类型。
- 该表的列 Quantity 记录了一次出货运输某种产品的数量。

E4.7a　列出表 SHIPMENTS 中的完全函数依赖、部分函数依赖（如果存在）和传递函数依赖（如果存在）。

E4.7b　写出将表 SHIPMENTS 规范化到 2NF 的结果。

E4.7c　写出将表 SHIPMENTS 规范化到 3NF 的结果。

E4.8　考虑下面的关系和数据。

LANGUAGE SCHOOL TABLE

CourseID	CourseLanguage	CourseLevel	ClientID	ClientName	Attendance	FinalScore
10	German	Basic	C111	Mr. Smith	100%	80%
11	German	Intermediate	C222	Ms. Jones	90%	90%
12	German	Advanced	C333	Mr. Vance	95%	100%
10	German	Basic	C444	Ms. Clark	100%	100%
11	German	Intermediate	C555	Ms. Wong	90%	95%
12	German	Advanced	C666	Ms. Hess	95%	90%
20	Japanese	Basic	C111	Mr. Smith	100%	100%
21	Japanese	Intermediate	C222	Ms. Jones	95%	100%

- 表 LANGUAGE SCHOOL 记录了学员参加语言班的数据。
- 每个课程都有唯一的课程 ID、课程语言和课程等级。
- 每名学员都有唯一的学员 ID 和名字。
- 每个课程可以有多名学员参加。
- 每名学员可以参加多个课程。
- 当一名学员完成一个课程，将记录他的出勤情况和在该班级的最终成绩。

122

E4.8a　列出表 LANGUAGE SCHOOL 中的完全函数依赖、部分函数依赖（如果存在）和传递函数依赖（如果存在）。

E4.8b　写出将表 LANGUAGE SCHOOL 规范化到 2NF 的结果。

E4.8c　写出将表 LANGUAGE SCHOOL 规范化到 3NF 的结果。

E4.9　考虑下面的关系和数据。

HEALTH CENTER TABLE

DocID	DocName	PatientID	PatientName	InsuranceCoID	InsuranceCoName	NextAppointmentDate
1	Dr. Joe	111	Max	11	ACME Insurance	1-Dec
1	Dr. Joe	222	Mia	11	ACME Insurance	2-Dec
1	Dr. Joe	333	Pam	12	Purple Star	3-Dec
2	Dr. Sue	333	Pam	12	Purple Star	7-Dec
3	Dr. Kim	555	Lee	13	Tower Block	2-Dec
3	Dr. Kim	111	Max	11	ACME Insurance	3-Dec
4	Dr. Joe	666	Alex	14	All Protect	9-Dec

- 表 HEALTH CENTER 记录了病人和医生预约的数据。
- 每位病人都有唯一的病人 ID 和名字。
- 每名医生都有唯一的医生 ID 和名字。
- 每位病人在一个保险公司投保。
- 每个保险公司都有唯一的保险公司 ID 和保险公司名称。

- 每次预约都是在一位病人和一名医生之间进行的。
- 每位病人可以有多个预约安排（但与一名医生之间只能有一次预约安排）。

E4.9a 列出表 HEALTH CENTER 中的完全函数依赖、部分函数依赖（如果存在）和传递函数依赖（如果存在）。

E4.9b 写出将表 HEALTH CENTER 规范化到 2NF 的结果。

E4.9c 写出将表 HEALTH CENTER 规范化到 3NF 的结果。

E4.10 考虑下面的关系和数据。

SURGERY SCHEDULE TABLE

PatientID	PatientName	SurgeonID	SurgeonName	NurseID	NurseName	SurgeryDate	SurgeryType	NurseRole
111	Joe	AAA	Dr. Adams	N1	Mike	1-Jan	Knee Surgery	1st Assistant
111	Joe	AAA	Dr. Adams	N2	Sue	1-Jan	Knee Surgery	2nd Assistant
111	Joe	AAA	Dr. Adams	N2	Sue	2-Jan	Ankle Surgery	1st Assistant
111	Joe	BBB	Dr. Brown	N2	Sue	3-Jan	Elbow Surgery	1st Assistant
222	Pat	CCC	Dr. Crown	N3	Tina	1-Jan	Knee Surgery	1st Assistant
333	Bob	AAA	Dr. Adams	N4	Lee	1-Jan	Hip Surgery	1st Assistant
333	Bob	AAA	Dr. Adams	N3	Tina	1-Jan	Hip Surgery	2nd Assistant
333	Bob	DDD	Dr. Adams	N5	Sue	2-Jan	Shoulder Surgery	1st Assistant
444	Pat	BBB	Dr. Brown	N6	Pam	1-Jan	Hip Surgery	1st Assistant

- 表 SURGERY SCHEDULE 记录了手术计划的数据。
- 每位病人都有唯一的病人 ID 和名字。
- 每名外科医生都有唯一的外科医生 ID 和名字。
- 每名护士都有唯一的护士 ID 和名字。
- 每个手术计划只有一种类型。
- 每位病人可以有多个手术计划，但是在一天之内只能做一次手术。
- 在一次手术期间，一名外科医生为一位病人手术，同时分配一名或多名护士。
- 在每次手术期间，一名护士只能有一种角色。
- 一名护士可能在不同的手术中有不同的角色。
- 外科医生和护士每天可以参与多台手术。

[123]

E4.10a 列出表 SURGERY SCHEDULE 中的完全函数依赖、部分函数依赖（如果存在）和传递函数依赖（如果存在）。

E4.10b 写出将表 SURGERY SCHEDULE 规范化到 2NF 的结果。

E4.10c 写出将表 SURGERY SCHEDULE 规范化到 3NF 的结果。

E4.11 考虑下面的关系和数据。

STUDENT INTERNSHIP TABLE

StudentID	StudentName	EmployerID	EmployerName	PositionID	PositionDescription
111	Joe	A	Axel Alarm	1	Sales Intern
222	Mac	A	Axel Alarm	2	IT Help Desk Intern
333	Sue	B	Barex Inc	2	Sales Intern
444	Pat	B	Barex Inc	1	Front Desk Intern
555	Bob	B	Barex Inc	1	Front Desk Intern
666	Joe	C	Calypso Corp	1	Marketing Specialist Intern

- 表 STUDENT INTERNSHIP 记录了学生实习的数据。
- 每名学生都有唯一的学生 ID 和名字。
- 每个雇主都有唯一的雇主 ID 和名字。

- 对于每个雇主，都有若干个职位。
- 一个雇主的每个职位有唯一的职位 ID（两个不同雇主的两个职位可能有相同的职位 ID）和职位描述。
- 每名学生只能为一个雇主工作，并且在该雇主处只能有一个职位。
- 可以有多名学生在同一雇主的同一职位工作。

E4.11a 列出表 STUDENT INTERNSHIP 中的完全函数依赖、部分函数依赖（如果存在）和传递函数依赖（如果存在）。

E4.11b 写出将表 STUDENT INTERNSHIP 规范化到 2NF 的结果。

E4.11c 写出将表 STUDENT INTERNSHIP 规范化到 3NF 的结果。

E4.12 考虑下面的关系和数据。

MOVIE ACTORS TABLE

MovieID	MovieName	ActorID	ActorName	ActorAssistantID	ActorAssistantName	ActorMovieSalary
M100	Silent Code	A11	Paloma Luna	AA01	Julie J	$1,200,000
M100	Silent Code	A22	Logan Jones	AA01	Julie J	$1,300,000
M100	Silent Code	A33	Esmeralda Po	AA02	Bob B	$1,200,001
M100	Silent Code	A44	C.C. Rooney	AA02	Bob B	$500,000
M200	Winter Promises	A11	Paloma Luna	AA03	Lisa L	$1,000,000
M200	Winter Promises	A33	C.C. Rooney	AA01	Julie J	$900,000
M200	Winter Promises	A55	Max Smith	AA02	Bob B	$300,000

- 表 MOVIE ACTORS 记录了演员出演电影的数据。
- 每部电影都有唯一的电影 ID 和电影名。
- 每个演员都有唯一的演员 ID 和名字。
- 每个演员助理都有唯一的演员助理 ID 和名字。
- 每部电影有多个演员参演。每个演员可以出演多部电影。
- 演员每参演一部电影，根据电影合约，演员会获得相应的片酬。
- 每部电影会雇佣多个演员助理。
- 在一部电影中，一个演员分配一个演员助理。一个演员助理可以被分配给多个演员。
- 一个演员助理可以在多部电影中被分配给同一个（或几个）演员。

124

E4.12a 列出表 MOVIE ACTORS 中的完全函数依赖、部分函数依赖（如果存在）和传递函数依赖（如果存在）。

E4.12b 写出将表 MOVIE ACTORS 规范化到 2NF 的结果。

E4.12c 写出将表 MOVIE ACTORS 规范化到 3NF 的结果。

E4.13 考虑下面的关系和数据。

BANK ACCOUNTS TABLE

AccountID	AccountType	CurrentBalance	AccountHolderID	AccountHolderName
A111	Checking	$1,000.00	C111 C222	Joe Smith Sue Smith
A222	Checking	$2,000.00	C333	Mary Moore
A333	Money Market	$15,200.00	C444 C555 C666	Pat Clark Lisa Clark Timmy Clark
A444	Savings	$3,000.00	C111 C222	Joe Smith Sue Smith

125 ~ 126

E4.13a 通过在已有的表中增加记录将表 BANK ACCOUNTS 规范化到 1NF。

E4.13b 通过在已有表的基础上创建另外的表将表 BANK ACCOUNTS 规范化到 1NF。

SQL

5.1 引言

本章将介绍结构化查询语言（Structured Query Language）SQL 及其功能。SQL 不仅可用于数据库的查询，还可以用于数据库的创建，实现数据库结构的添加、修改和删除，以及数据库记录的插入、删除及修改操作。本书的后续部分将采用实例介绍在数据库的创建和使用中所涉及的 SQL 命令及函数。

事实上，每一个现代关系型 DBMS 都使用了 SQL。关系数据库获得成功与普及的原因之一就在于，作为标准查询语言的 SQL 函数仅需极少的变化就能适用于大部分关系型 DBMS。例如，Oracle、MySQL、Microsoft SQL Server、PostgreSQL、Teradata、IBM DB2 等许多数据库都使用了 SQL。

SQL 是一种标准的通用语言，因此将其从一种 DBMS 转向另一种 DBMS 的过程往往较为容易。事实上，编写数据检索查询语句及其他 SQL 语句的能力，并不依赖于某个具体的数据库软件，而是依赖于用户对该通用语言的掌握程度。

即使存在一个 SQL 标准（如本章末所讨论的），不同的 RDBMS 仍然可以使用 SQL 语言略有差异的不同版本。本章将给出一些基本的 SQL 命令，这些命令通用于不同时期的 RDBMS 包。

5.2 SQL 命令综述

SQL 是一种综合的数据库语言，涵盖了多种功能命令。SQL 命令可以根据其功能分为以下几类：

数据定义语言（Data Definition Language，DDL）

数据操纵语言（Data Manipulation Language，DML）

数据控制语言（Date Control Language，DCL）

事务控制语言（Transaction Control Language，TCL）

本书将首先给出几种 SQL 命令用途的简要概述，然后再针对每种命令分别给出实例介绍。

5.2.1 数据定义语言

DDL 用于创建和修改数据库结构。注意，关系数据库的结构采用关系模式来表示，关系模式则由数据关系及其参照完整性来描述。DDL 的目的在于实现关系模式（以及一些额外结构，如索引等约束），并得到一个实际的关系数据库。DDL SQL 命令的例子有：

CREAT

ALTER

DROP

本书将采用实例对创建、修改数据库结构的上述 SQL 命令进行逐一说明。

5.2.2 数据操纵语言

DML 用于操作数据库中的数据，包括对数据的插入、修改、删除及检索等操作。

一旦关系模式实现为实际的关系数据库，数据就要插入其中。在数据库的生命周期中，允许有新数据的插入，并允许已有数据的修改和删除。DML 包括以下几种插入、修改及删除数据记录的命令：

INSERT INTO

UPDATE

DELETE

这些插入、修改及删除命令同样将采用相应的实例来阐释说明。

DML 语句还包括如下的数据检索命令：

SELECT

数据检索也称为查询（query），是数据库中最常见的数据操作。因此，SELECT 命令也是 SQL 中使用最频繁的命令。SELECT 语句之后可以有多个其他的 SQL 关键字。本书后面将用实例对 SELECT 命令及其后的 SQL 关键字进行阐释说明。

5.2.3 数据控制语言和事务控制语言

DCL 和 TCL 语句可用于与数据库维护及管理相关的多种过程。DCL 命令帮助实现数据的存取控制，而 TCL 则用于数据库中的事务管理。（本书第 10 章给出了数据库管理的概述，我们将在第 10 章讨论 DCL 和 TCL 命令。）

5.3 SQL 数据类型

用 SQL 命令创建好关系以后，关系的每一列都有一个特定的数据类型。下表给出了一些最常见的 SQL 数据类型（SQL data type）：

CHAR（n）	n 个字符的固定长度字符串
VARCHAR（n）	最大长度为 n 个字符的可变长度字符串
INT	整数
NUMERIC（x，y）	x 位数字，小数点后的位数为 y
DATE	日期值（年，月，日）

5.4 SQL 语法简要说明

在用实例一一阐释每个 SQL 命令之前，我们先来看一些 SQL 语法的简短说明：

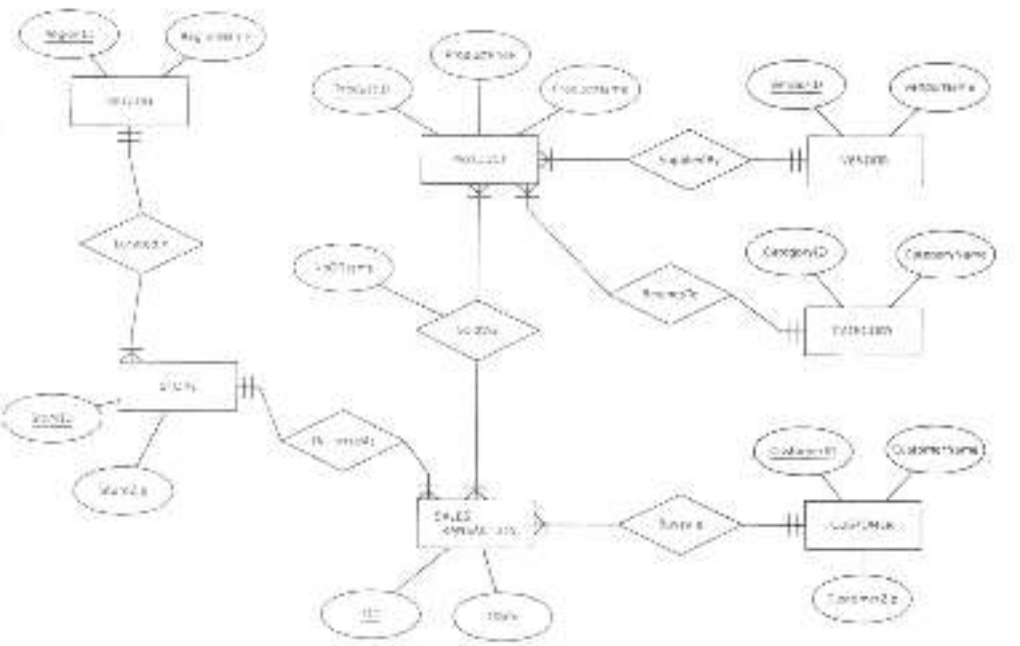
- 分号。紧跟在每一条 SQL 语句后面，表示一个 SQL 命令的结束。在一系列 SQL 语句里，分号表示每条 SQL 语句何时结束。
- SQL 关键字。即 SQL 命令中所使用的表和列的名字，不区分大小写。语句可以是大写字母也可以是小写字母（如，“SELECT”与“select”或“SeLeCt”相同）。考虑到可读性，本书的 SQL 关键字使用大写字母，表名和列名都使用小写字母（这是若干常见的书写方法之一）。
- 一条 SQL 语句可以写成一个长句子，作为文本中的一行。但为了增强可读性，往往

将 SQL 语句拆分成多行来书写。

本章中的 SQL 例子可以在大多数现代 RDBMS 软件[⊖]中执行。本章的命令列表中，存在多种语法命令，这些命令将在本章末进行说明。

5.5 CREATE TABLE

SQL 命令 CREATE TABLE 用于关系表的创建和连接。我们将使用第 3 章图 3-32 中的 ZAGI 零售公司销售部门数据库来阐释 CREATE TABLE 的用途。为方便起见，图 5-1a 重复展示了图 3-32。



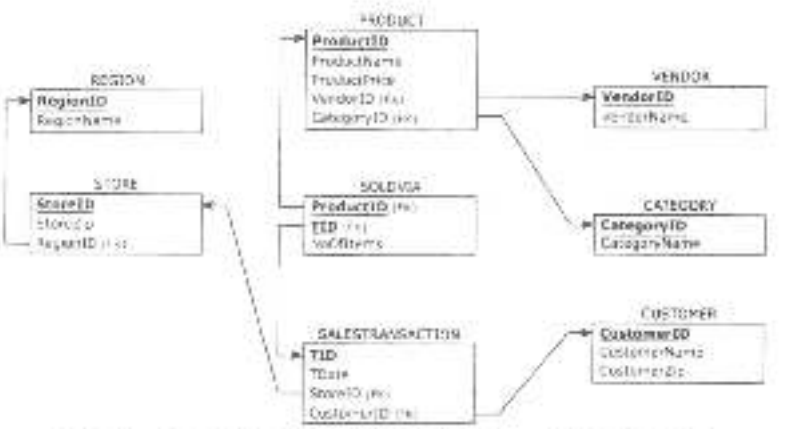


图 5-1a ZAGI 零售公司销售部门数据库：ER 图与关系模式

⊖ 本章中的 SQL 语句都可以按其出现的顺序得到执行。执行结果是一个已创建、已填充、可查询的数据库。本章中 6 种流行的 DBMS 软件（Oracle、MySQL、Microsoft SQL Server、PostgreSQL、Teradata 以及 IBM DB2）中的 SQL 语句脚本均能在 dbtexbook.com 找到。

表5-1a展示了ZAGI零售公司销售部门数据库的关系模式。图5-1b则给出了使用CREATE TABLE语句创建相应关系表的SQL代码。

```
CREATE TABLE vendor
(       vendorid            CHAR(2)                 NOT NULL,
        vendorname          VARCHAR(25)             NOT NULL,
        PRIMARY KEY (vendorid) );

CREATE TABLE category
(       categoryid          CHAR(2)                 NOT NULL,
        categoryname        VARCHAR(25)             NOT NULL,
        PRIMARY KEY (categoryid) );

CREATE TABLE product
(       productid           CHAR(3)                 NOT NULL,
        productname         VARCHAR(25)             NOT NULL,
        productprice        NUMERIC(7,2)            NOT NULL,
        vendorid            CHAR(2)                 NOT NULL,
        categoryid          CHAR(2)                 NOT NULL,
        PRIMARY KEY (productid),
        FOREIGN KEY (vendorid) REFERENCES vendor(vendorid),
        FOREIGN KEY (categoryid) REFERENCES category(categoryid) );

CREATE TABLE region
(       regionid            CHAR(1)                 NOT NULL,
        regionname          VARCHAR(25)             NOT NULL,
        PRIMARY KEY (regionid) );

CREATE TABLE store
(       storeid             VARCHAR(3)              NOT NULL,
        storezip            CHAR(5)                 NOT NULL,
        regionid            CHAR(1)                 NOT NULL,
        PRIMARY KEY (storeid),
        FOREIGN KEY (regionid) REFERENCES region(regionid) );

CREATE TABLE customer
(       customerid          CHAR(7)                 NOT NULL,
        customername        VARCHAR(15)             NOT NULL,
        customerzip         CHAR(5)                 NOT NULL,
        PRIMARY KEY (customerid) );

CREATE TABLE salestransaction
(       tid                 VARCHAR(8)              NOT NULL,
        customerid          CHAR(7)                 NOT NULL,
        storeid             VARCHAR(3)              NOT NULL,
        tdate               DATE                    NOT NULL,
        PRIMARY KEY (tid),
        FOREIGN KEY (customerid) REFERENCES customer(customerid),
        FOREIGN KEY (storeid) REFERENCES store(storeid) );

CREATE TABLE soldvia
(       productid           CHAR(3)                 NOT NULL,
        tid                 VARCHAR(8)              NOT NULL,
        noofitems           INT                     NOT NULL,
        PRIMARY KEY (productid, tid),
        FOREIGN KEY (productid) REFERENCES product(productid),
        FOREIGN KEY (tid) REFERENCES salestransaction(tid) );
```

图5-1b　为ZAGI零售公司销售部门数据库创建关系表的SQL代码

在CREATE TABLE命令的语义中，CREATE TABLE<tablename>之后是圆括号。括号中首先是列的描述，如列名、该列数据的类型及可能的列约束。NOT NULL约束就是这种约束的例子，它表示该列不是可选的。列的描述之间用逗号隔开。列的描述之后是关系表的描述。表的描述指明了表的约束，包括表的主码和外码等。

观察图 5-1b 中的 CREATE TABLE 语句。第一个 CREATE TABLE 语句创建了表 VENDOR，该表有两个列：VendorID 和 VendorName。对每个列，语句都定义了其数据类型。由于这两个列都不是可选列，因此都有 NOT NULL 标识（可选列没有 NOT NULL 标识）。列的描述后面是表的描述，即描述哪个或哪些列是表的主码。对 VENDOR 表而言，VendorID 列是主码。再看一个 CREATE TABLE 语句中描述复合主码的例子：关系 SOLDVIA 的 CREATE TABLE 命令。该 CREATE TABLE 命令描述了 SOLDVIA 关系的复合主码，该复合属性包含 ProductID 和 TID 两个列。

对于有外码的关系表，CREATE TABLE 语句则通过列出外码所参照的列和表来描述参照完整性约束。例如，PRODUCT 关系的 CREATE TABLE 语句便对两个外码 VendorID 和 CategoryID 分别给出了其参照完整性约束的描述。其具体表述如下：

```
FOREIGN KEY (vendorid) REFERENCES vendor(vendorid),
```

该描述指明在 PRODUCT 关系中的 VendorID 参考了 VENDOR 关系中的主码 VendorID。在一些 RDBMS 软件中，该表述还可以写为：

```
FOREIGN KEY (vendorid) REFERENCES vendor,
```

该精简版后没有指出 VENDOR 关系中的主码列名，但也同样表示了 PRODUCT 关系中的外码 VendorID 参照了 VENDOR 关系中的主码。

要创建包含外码的关系表，必须先创建被外码参照的主码关系表。图 5-1b 中 CREATE TABLE 语句的排列顺序保证了这一点。例如，在创建 PRODUCT 关系表之前，我们需先创建 VENDOR 和 CATEGORY 关系，这是由于 PRODUCT 关系的外码参照了 VENDOR 和 CATEGORY 两个关系的主码。 129

实际上，CREATE TABLE 语句常常并不直接由数据库开发人员来编写。在很多情况下，开发人员往往会使用 CASE（computer-aided software engineering）工具来自动生成 CREATE 语句，而不是人工编写 CREATE TABLE 语句。例如，开发人员可以用 CASE 工具来创建图 5-1a 中所示的关系模式，并且为关系中的列定义数据类型（或类似 NOT NULL 的其他约束）。一旦关系中列的细节确定好以后，开发人员就可以激活相应的函数（如，点击 CASE 工具的相应按钮）来自动地创建 CREATE TABLE 语句。这样就能得到如图 5-1b 所示的 CREATE TABLE 语句。 130

5.6 DROP TABLE

命令 DROP TABLE 用于从数据库中移除关系表。例如，若希望移除表 SOLDVIA，我们可以给出如下语句：

```
DROP TABLE soldvia;
```

注意，若希望将 ZAGI 零售公司销售部门数据库的所有关系表一一移除，我们需要考虑 DROP TABLE 语句的顺序问题。若一个关系的主码被其他关系的外码所参照，则参照完整性约束会阻止对该主码所在关系的删除操作。因此我们需要先删除外码所在关系，然后才能删除被外码参照的主码关系。 131

例如，下面的 DROP TABLE 命令是无效的：

DROP TABLE sequence ZAGI database—INVALID:

```
DROP TABLE region;
DROP TABLE store;
DROP TABLE salestransaction;
DROP TABLE product;
DROP TABLE vendor;
DROP TABLE category;
DROP TABLE customer;
DROP TABLE soldvia;
```

上面的这个 DROP TABLE 序列是无效的（部分语句不会执行），原因在于该序列试图删除那些被参照的关系。例如，由于关系 STORE 参照了 REGION，因此我们不能首先删除 REGOIN 关系。我们可以先删除 STORE 关系，然后再删除 REGOIN 关系。

下面的 DROP TABLE 序列是有效的：

DROP TABLE sequence ZAGI database—VALID:

```
DROP TABLE soldvia;
DROP TABLE salestransaction;
DROP TABLE store;
DROP TABLE product;
DROP TABLE vendor;
DROP TABLE region;
DROP TABLE category;
DROP TABLE customer;
```

在该 DROP TABLE 语句序列中，没有哪个关系是在外码所在关系删除之前被删除的。若我们执行该 DORP TABLE 语句序列，则 ZAGI 零售公司销售部门数据库的所有表都会被逐一删除。这种情况下，可以通过执行图 5-1b 所列出的语句来重新创建 ZAGI 零售公司销售部门的数据。

5.7　INSERT INTO

INSERT INTO 语句用于向已创建好的关系表中填入数据。为了阐释 INSERT INTO 命令的用处，我们将用第 3 章中图 3-33 的记录来填充 ZAGI 零售公司销售部门数据库。为方便起见，图 5-1c 重复给出了图 3-33。

REGION

RegionID	RegionName
C	Chesapeake
T	Tristate

STORE

StoreID	StoreZip	RegionID
S1	60600	C
S2	60605	C
S3	35400	T

PRODUCT

ProductID	ProductName	ProductPrice	VendorID	CategoryID
1X	Air Bag	$100	PG	CP
2X2	Easy Boot	$20	MK	FW
3X3	Cozy Sock	$15	MK	FW
4X4	Duck Boot	$60	PG	FW
5X5	Tent Tent	$150	MK	CP
6X6	Bigger Tent	$200	MK	CP

VENDOR

VendorID	VendorName
PG	Patrick Gear
MK	Mountain Kings

CATEGORY

CategoryID	CategoryName
CP	Camping
FW	Footwear

SALESTRANSACTION

TID	CustomerID	StoreID	TDate
T111	1-2-333	S1	1-Jan-2013
T222	2-3-444	S2	1-Jan-2013
T333	1-2-333	S3	2-Jan-2013
T444	2-4-555	S3	2-Jan-2013
T555	2-3-444	S3	2-Jan-2013

SOLDVIA

ProductID	TID	NoOfItems
1X1	T111	1
2X2	T222	1
3X3	T333	2
1X1	T333	1
4X4	T444	1
2X2	T444	2
4X4	T555	4
5X5	T555	2
6X6	T555	1

CUSTOMER

CustomerID	CustomerName	CustomerZip
1-2-333	Tina	60137
2-3-444	Tony	60611
2-4-555	Pam	35401

图 5-1c　ZAGI 零售公司销售部门数据库的记录

图 5-1d 所示是 ZAGI 零售公司销售部门数据库的数据填充语句 INSERT INTO。

```
INSERT INTO vendor   VALUES ('PG','Pacifica Gear');
INSERT INTO vendor   VALUES ('MK','Mountain King');

INSERT INTO category VALUES ('CP','Camping');
INSERT INTO category VALUES ('FW', Footwear');

INSERT INTO product  VALUES ('1X1','Zzz Bag',100,'PG','CP');
INSERT INTO product  VALUES ('2X2','Easy Boot',70,'MK','FW');
INSERT INTO product  VALUES ('3X3','Cozy Sock',15,'MK','FW');
INSERT INTO product  VALUES ('4X4','Dura Boot',90,'PG','FW');
INSERT INTO product  VALUES ('5X5','Tiny Tent',110,'MK','CP');
INSERT INTO product  VALUES ('6X6','Biggy Tent',250,'MK','CP');

INSERT INTO region   VALUES ('C','Chicagoland');
INSERT INTO region   VALUES ('T','Triprate');

INSERT INTO store    VALUES ('S1','60600','C');
INSERT INTO store    VALUES ('S2','60605','C');
INSERT INTO store    VALUES ('S3','35400','T');

INSERT INTO customer VALUES ('1-2-333','Tina','60137');
INSERT INTO customer VALUES ('2-3-444','Tony','60611');
INSERT INTO customer VALUES ('3-4-555','Pat','35401');

INSERT INTO salestransaction VALUES ('T111','1-2-333','S1','01/Jan/2013');
INSERT INTO salestransaction VALUES ('T222','2-3-444','S2','01/Jan/2013');
INSERT INTO salestransaction VALUES ('T333','1-2-333','S3','02/Jan/2013');
INSERT INTO salestransaction VALUES ('T444','3-4-555','S3','02/Jan/2013');
INSERT INTO salestransaction VALUES ('T555','2-3-444','S3','02/Jan/2013');

INSERT INTO soldvia  VALUES ('1X1','T111',1);
INSERT INTO soldvia  VALUES ('2X2','T222',1);
INSERT INTO soldvia  VALUES ('3X3','T333',5);
INSERT INTO soldvia  VALUES ('1X1','T333',1);
INSERT INTO soldvia  VALUES ('4X4','T444',1);
INSERT INTO soldvia  VALUES ('2X2','T444',2);
INSERT INTO soldvia  VALUES ('4X4','T555',4);
INSERT INTO soldvia  VALUES ('5X5','T555',2);
INSERT INTO soldvia  VALUES ('6X6','T555',1);
```

图 5-1d ZAGI 零售公司销售部门数据库的 INSERT INTO 语句

语句中 INSERT INTO <tablename> 后紧跟的是关键字 VALUES 以及圆括号，圆括号中列出了需要插入的数据值。图 5-1d 中的第一个 INSERT INTO 语句实现了向 VENDOR 关系的 VendorID 列插入值 PG，向 VendorName 列插入值 Pacifica Gear。各个列在 CREATE TABLE 语句中的顺序决定了待插入值在 INSERT INTO 语句中的顺序。关系 VENDOR 的 CREATE TABLE 语句首先创建 VendorID 列，然后创建 VendorName 列，因此 VENDOR 关系的每一个 INSERT INTO 语句中，都应该先列出待插入的 VendorID 值，再列出待插入的 VendorName 值。

此外，INSERT INTO 语句还可以有如下形式：INSERT INTO <tablename> (<columnname>, <columnname>, …)VALUES(value, value, …)；此处待插入值的顺序则由语句中表名后面的列名顺序决定。 [32]

例如，图 5-1d 的最后一个 INSERT INTO 命令可以写成如下形式：

```
INSERT INTO soldvia(noofitems, tid, productid) VALUES (1, 'T555', '6x6');
```

这种 INSERT INTO 语句的书写方式允许 INSERT INTO 语句中值的顺序与 CREATE TABLE 语句中的顺序不同。另一方面，这种方式明确给定每个列的列名，因而增加了 INSERT

INTO 命令的语义。在列名明确给出的形式中，并不是所有的列都需要指明。我们可以只向已准备好数据的列输入数据。然而不管使用那种 INSERT INTO 语义，强制性的列都需要有数据输入。

在 SQL 语句中，数据类型为字符型（如 CHAR 或 VARCHAR）的列值需要用引号分隔，而数值型（如 INT 或 NUMERIC）则不需要。例如，图 5-1d 中关系 VENDOR 的 INSERT INTO 语句中，所有值都有引号进行分隔，其原因在于 VENDOR 关系的两个列都是字符类型。再者，由于 PRODUCT 关系的 CREATE TABLE 语句中，第三列 ProductPrice 是数值型，因此在 PRODUCT 关系的 INSERT INTO 语句中，第三个值没有使用引号。

事实上，如图 5-1d 中的 INSERT INTO 语句往往并不由负责数据实体的终端用户直接编写。通常的做法是借助第 6 章所述的表单（form）等前端软件（front-end application）来确保终端用户对数据实体的相应操作，即前端软件将代替终端用户产生相应语句。此外，当从外部向关系表中导入大量数据时，前端软件也会产生相应的 INSERT INTO 语句。

观察图 5-1d 可以看出，没有外码的关系表会优先执行插入操作。关系表中对外码的插入操作必须在其所参照的主码完成插入之后进行。参照完整性要求必须在输入了被参照的主码值之后，才能输入外码的值。

[133]

5.8　SELECT

SELECT 语句用于从数据库关系中检索数据，是最常用的 SQL 语句。SELECT 语句的结果是一张关系表，该关系表包含所有满足 SELECT 语句条件的记录。在其简单形式中，SELECT 语句的构成如下：

```
SELECT      <columns>
FROM        <tables>
```

在 SELECT 子句中，关键字 SELECT 之后紧跟的是待检索的列名，列名之间用逗号隔开。SELECT 子句后面往往跟有一个 FROM 子句。在 FROM 子句中，关键字 FROM 之后紧跟待检索关系表的表名。Query1 就是一个最简单的 SELECT 语句实例：

Query1 的文本描述：检索关系 PRODUCT 中的所有内容。

```
Query1:  SELECT    productid, productname, productprice, vendorid,
                   categoryid
         FROM      product;
```

[134]

Query1 的结果在图 5-2 中给出。

ProductID	ProductName	ProductPrice	VendorID	CategoryID
1X1	Zzz Bag	100	PG	CP
2X2	Easy Boot	70	MK	FW
3X3	Cosy Sock	15	MK	FW
4X4	Dura Boot	90	PG	FW
5X5	Tiny Tent	150	MK	CP
6X6	Biggy Tent	250	MK	CP

图 5-2　Quer1 和 Query1a 的结果

Query1 检索了 PRODUCT 关系中的所有内容。

Query1a 也可以得到同样的结果：

```
Query1a: SELECT    *
         FROM      product;
```

SELECT 关键字之后的星号（*）称为"通配符"。使用星号和列出所有列名得到的结果相同。因此，若我们希望显示关系中所有的列，使用星号更简洁。但若希望检索结果中列的显示顺序与 CREATE TABLE 语句中列的顺序不同，则不能使用星号。这是唯一例外的情况。Query2 给出了一个这样的例子。

Query2 的文本描述：检索关系 PRODUCT 中的所有内容，以如下顺序来显示所有的列：ProductName，ProductID，VendorID，CategoryID，ProductPrice。

```
Query2:  SELECT    productname, productid, vendorid, categoryid,
                   productprice
         FROM      product;
```

Query2 的结果在图 5-3a 中给出。

ProductName	ProductID	VendorID	CategoryID	ProductPrice
Zzz Bag	1X1	PG	CP	100
Easy Boot	2X2	MK	FW	70
Cozy Sock	3X3	MK	FW	15
Dura Boot	4X4	PG	FW	90
Tiny Tent	5X5	MK	CP	150
Bigsy Tent	6X6	MK	CP	250

图 5-3a Query2 的结果

Query3 的例子中则给出了使用 SELECT 子句仅查询部分列的用法。

Query3 的文本描述：显示 PRODUCT 关系中的 ProductID 和 ProductPrice 两列。

```
Query3:  SELECT    productid, productprice
         FROM      product;
```

图 5-3 给出了 Query3 的结果。 135

ProductID	ProductPrice
1X1	100
2X2	70
3X3	15
4X4	90
5X5	150
6X6	250

图 5-3 Query3 的结果

除了用于列举关系中的列，SELECT 子句还能用于显示以表达式形式给出的派生属性（计算列的值得到）。此时，SELECT 语句可组织成如下形式：

```
SELECT    <columns, expressions>
FROM      <tables>
```

Query3a 给出了一个这样的例子。

Query3a 的文本描述：对关系 PRODUCT，显示 ProductID 和 ProductPrice 两个列，并显示一个新的列表示为增长 10% 以后的 ProductPrice 值。

```
Query3a: SELECT    productid, productprice, productprice * 1.1
         FROM      product;
```

图 5-3a 给出了 Query3a 的结果。

ProductID	ProductPrice	ProductPrice*1.1
1X1	100	110
2X2	70	77
3X3	15	16.5
4X4	90	99
5X5	150	165
6X6	250	275

图 5-3a Query3a 的结果

SELECT FROM 语句还包含了其他可选关键字，如 WHERE、GROUP BY、HAVING、ORDER BY，其顺序如下所示：

```
SELECT      <columns, expressions>
FROM        <tables>
WHERE       <row selection condition>
GROUP BY    <grouping columns>
HAVING      <group selection condition>
ORDER BY    <sorting column, expressions>
```

我们将在下面的例子中给出这些关键字的解释说明。

5.9 WHERE

SELECT 语句可以包含 WHERE 条件，WHERE 条件决定了应该检索哪些列，不检索哪些列。Query4 给出了一个带有 WHERE 条件的 SELECT 查询的简单例子。

Query4 的文本描述：对所有价格高于 $100 的产品，检索其产品编号、产品名称、供应商编号及产品价格。

```
Query4: SELECT    productid, productname, vendorid, productprice
        FROM      product
        WHERE     productprice > 100;
```

图 5-4 给出了 Query4 的结果。

ProductID	ProductName	VendorID	ProductPrice
5X5	Tiny Tent	MK	150
6X6	Biggy Tent	MK	250

图 5-4 Query4 的结果

所有 ProductPrice 值大于 100 的记录行都将被检索出来，其余的则不会。

逻辑条件决定哪些记录将被检索出来。下列逻辑比较操作都可以作为逻辑条件：

= 等于

< 小于

> 大于

<= 小于等于

>= 大于等于

!= 不等于。

<> 不等于（可选择）

136

一个 WHERE 子句中可以使用多个比较表达式，用布尔逻辑操作 AND 或 OR 进行连接。

Query5 给出了 WHERE 子句中包含多个比较表达式的例子。

Query5 的文本描述： 检索产品种类为“FW”且价格小于等于 $110 的所有产品的产品编号、产品名称、供应商编号、产品价格。

```
Query5: SELECT      productid, productname, vendorid, productprice
        FROM        product
        WHERE       productprice <= 110 AND
                    categoryid = 'FW';
```

表 5-5 给出了 Query5 的结果。

ProductID	ProductName	VendorID	ProductPrice
2X2	Easy Boot	MK	70
3X3	Cosy Sock	MK	15
4X4	Dura Boot	PG	90

表 5-5 Query5 的结果

注意，就像 INSERT INTO 语句中给出的情况一样，查询中的字符型值需要用引号分隔（如 'FW'）。

5.10 DISTINCT

DISTINCT 关键字可以结合 SELECT 语句一起使用。为了阐述其用途，首先请看 Query6。

Query6 的文本描述： 检索 PRODUCT 关系中所有记录的 VendorID。

```
Query6: SELECT      vendorid
        FROM        product;
```

图 5-6 给出了 Query6 的结果。

VendorID
PG
MK
MK
PG
MK
MK

图 5-6 Query6 的结果

PRODUCT 关系共有 6 条记录，图 5-6 列出了每条记录的 VendorID 值。注意，虽然查询的结果以关系表的形式展示，但由于表中的行值并不唯一，因此该查询结果不符合关系表的定义。

若需要不重复地展示关系 PRODUCT 中所有存在的 VendorID 值，那么请看 Query7 的例子。

Query7 的文本描述： 不重复地显示关系 PRODUCT 中所有的 VendorID。

```
Query7: SELECT      DISTINCT vendorid
        FROM        product;
```

图 5-7 给出了 Query7 的结果。

VendorID
PG
MK

图 5-7 Query7 的结果

137

在 SELECT 关键字后使用 DISTINCT 可以去掉查询结果中重复出现的值。

5.11 ORDER BY

若希望对查询结果的某一列或者多个列进行排序，则需要在 SELECT 查询中使用 ORDER BY 关键字。Query8 的查询结果需要按照商品价格进行排序。

Query8 的文本描述： 检索 FW 类所有产品的产品编号、产品名称、种类编号以及产品价格，查询结果按商品价格排序。

Query8：
```
SELECT      productid, productname, categoryid, productprice
FROM        product
WHERE       categoryid = 'FW'
ORDER BY    productprice;
```

图 5-8 给出了 Query8 的结果。

ProductID	ProductName	CategoryID	ProductPrice
3X3	Cosy Sock	FW	15
2X2	Easy Boot	FW	70
4X4	Dura Boot	FW	90

图 5-8 Query8 的结果

默认情况下，ORDER BY 对结果进行升序排列。若希望对结果进行降序排列，则需要使用 Query9 中所示的关键字 DESC。

Query9 的文本描述： 检索 FW 类所有产品的产品编号、产品名称、种类编号及产品价格，结果按商品价格降序排列。

Query9：
```
SELECT      productid, productname, categoryid, productprice
FROM        product
WHERE       categoryid = 'FW'
ORDER BY    productprice DESC;
```

图 5-9 给出了 Query9 的结果。

ProductID	ProductName	CategoryID	ProductPrice
4X4	Dura Boot	FW	90
2X2	Easy Boot	FW	70
3X3	Cosy Sock	FW	15

图 5-9 Query9 的结果

查询的结果还可以按照多个列的值进行排序，如 Query10 中所示。

Query10 的文本描述： 检索所有产品的产品编号、产品名称、种类编号、产品价格，结果按种类编号排序，相同种类编号的产品则根据产品价格排序。

Query10：
```
SELECT      productid, productname, categoryid, productprice
FROM        product
ORDER BY    categoryid, productprice;
```

图 5-10 给出了 Query10 的结果。Query10 先根据 CategoryID 进行排序，然后根据 ProductPrice 进行排序。

ProductID	ProductName	CategoryID	ProductPrice
1X1	Zzz Bag	CP	100
5X5	Tiny Tent	CP	150
8X6	Biggy Tent	CP	250
3X3	Cosy Sock	FW	15
2X2	Easy Boot	FW	70
4X4	Dura Boot	FW	90

图 5-10 Query10 的结果

5.12 LIKE

若希望检索记录中那些特定列的值与给定条件部分匹配，就需要在 SELECT 命令的 WHERE 子句中使用 LIKE 关键字。请看 Query11。

Query11 的文本描述：检索所有产品名字里面包含“Boot”的所有产品记录。

Query11：
```
SELECT      *
FROM        product
WHERE       productname LIKE '%Boot%';
```

图 5-11 给出了 Query11 的结果。

ProductID	ProductName	ProductPrice	VendorID	CategoryID
2X2	Easy Boot	70	MK	FW
4X4	Dura Boot	90	PG	FW

图 5-11 Query11 的结果

“%”是一个通配符，其含义是“空串或任意字符串”，因此该查询语句会检索出所有商品名称 ProductName 中包含“Boot”（“Boot”前或后可出现任意字符串，包括空串）的所有商品记录。另一个通配符“_”则表示“任意的一个字符”。

5.13 聚集函数

为了计算和统计查询结果，SQL 提供了如下聚集函数（aggregate functions）：COUNT、SUM、AVG、MIN 以及 MAX。COUNT 函数用于实现查询结果中记录的计数，而 SUM、AVG、MIN 以及 MAX 函数则分别用于计算查询结果集合里值的总和、平均值、最小值及最大值。SUM 和 AVG 函数只能对数值型数据进行运算，MIN 和 MAX 函数不仅可以对数值型数据进行运算，还可以对日期和字符型数据进行运算。

下面请看 Query12。

Query12 的文本描述：检索所有产品的平均价格

Query12：
```
SELECT      AVG(productprice)
FROM        product;
```

这个查询给出了所有商品价格的平均值。表 5-12 是 Query12 的查询结果。

AVG(ProductPrice)
112.5

图 5-12 Query12 的结果

继续看看 Query13。

Query13 的文本描述：给出产品总个数。

Query13：
```
SELECT    COUNT(*)
FROM      product;
```

139

注意，AVG、SUM、MIN、MAX 这几个函数需要指定一个列名作为其计算依据（如 Query12 中的 AVG(ProductPrice) 指定了列 ProductPrice）。而 COUNT 函数还可以使用星号（*）作为计算依据。例如 Query13 计算了 PRODUCT 关系中共有多少条记录。其中 COUNT(*) 中的星号表示所有记录。Query13 计算了 PRODUCT 表中记录的条数。图 5-13 给出了该查询的结果。

在 Query13 中，若使用任意一个列名（如 COUNT(ProductID) 或 COUNT(ProductName)）代替星号作为函数的计算依据，将得到完全相同的结果。这是由于 PRODUCT 表没有可选的列，并且每个列都有与每条记录相对应的值。如果一个表包含了值为空的可选列，则 COUNT 函数在计算时会跳过这些空值。因此，当列中存在空值时，使用列名进行计数就会得到与 COUNT(*) 不同的结果。

COUNT(*)
6

图 5-13 Query13 的结果

Query14 计算了表 PRODUCT 中共有多少个不同的代理商。

Query14 的文本描述：检索出售产品的代理商总数。

Query14：
```
SELECT    COUNT(DISTINCT VendorID)
FROM      product;
```

图 5-14 给出了 Query14 的结果。

COUNT(DISTINCT VendorID)
3

图 5-14 Query14 的结果

如 Query15 所示，同一个 SELECT 语句中同样可以使用多个聚集函数。

Query15 的文本描述：检索供应编号为“CP”的商品个数、商品平均价格以及商品的最低和最高价格。

Query15：
```
SELECT    COUNT(*), AVG(productprice), MIN(productprice),
          MAX(productprice)
FROM      product
WHERE     vendorid = 'CP';
```

图 5-15 给出了 Query15 的结果。

COUNT(*)	AVG(ProductPrice)	MIN(ProductPrice)	MAX(ProductPrice)
3	169.66667	102	250

图 5-15 Query15 的结果

5.14 GROUP BY

在 SQL 查询中，聚集函数常常结合 GROUP BY 关键字一起使用，这样就可以实现对组的聚集操作。组是一系列相关联的数据记录构成的集合。请看 Query16。

Query16 的文本描述：对每个代理商，检索其代理商编号、所供应的产品总数及所供应产品的平均价格。

140

Query16：
```
SELECT    vendorid, COUNT(*), AVG(productprice)
FROM      product
GROUP BY  vendorid;
```

图 5-16 给出了 Query16 的结果。

VendorID	COUNT(*)	AVG(ProductPrice)
PG	2	95
MK	4	121.25

图 5-16 Query16 的结果

如图 5-16a 中所示，Query16 将 PRODUCT 关系中 VendorID 值相同的记录归为一组，然后计算该组中记录的个数及所售商品的平均价格。

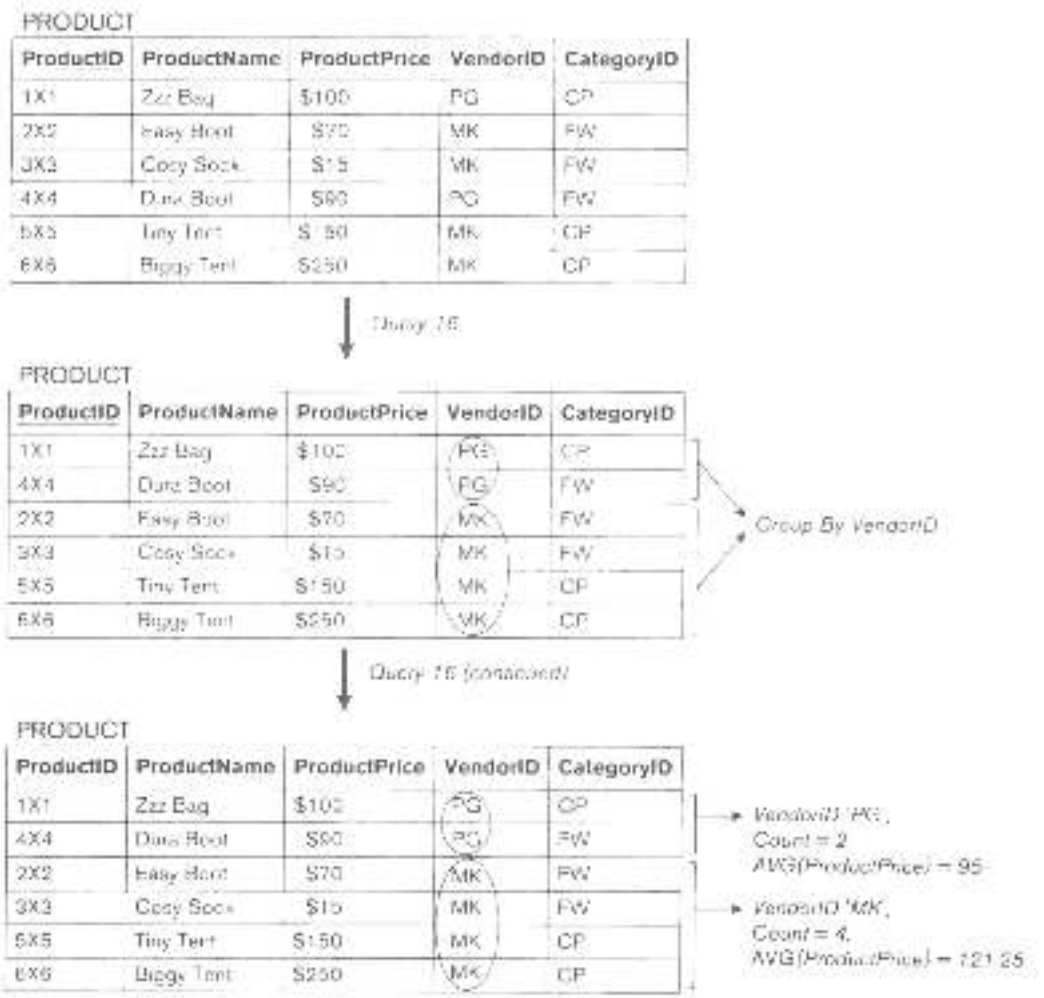


图 5-16a Query16 剖析

若 SELECT 语句使用了聚集函数，则单个列名除非出现在 GROUP BY 子句中，否则不能成为 SELECT 语句的一部分。 [141]

下面的列子是一个无效的查询，该查询试图实现 Query16 描述的问题。

```
Query16: SELECT        vendorid, COUNT(*), AVG(productprice)
无效: FROM        product;  ERROR MESSAGE RETURNED
```

为实现组内数据的聚集操作，初学者在使用 SQL 查询时常常容易出错。如在 Query16 INVALID 中，一个典型的错误就是在 SELECT 语句中列出单个列名及相应的聚集函数，且没有使用包含该列的 GROUP BY 子句。该语句会报错且不会得到执行。查询无效的原因在于，聚集函数（如 COUNT(*)、AVG(ProductPrice)）需要针对多条记录进行计算，计算结果不能与单个记录特定列的值混合在一起。若没有 GROUP BY 子句，COUNT(*)、AVG(ProductPrice) 函数就只能应用到整个关系表，其结果就是一个计数值 COUNT 和一个平均值 AVG。查询无法将针对整个表的一个 COUNT 值和一个 AVG 值与表中每条记录的 VendorID 值结合起来。因此为了使查询得到执行，需要在查询的最后加上 GROUP BY 子句，如 Query16 所示。在 Query16 中，GROUP BY 子句指明了表中 COUNT(*) 以及 AVG(ProductPrice) 聚集对象的子集范围。子集的个数就是表中不同 VendorID 的个数，因而每个 VendorID 都有一个 COUNT 值和一个 AVG 值与之对应（如图 5-16 所示）。

GROUP BY 子句的使用规则是，出现在 GROUP BY 子句后的列也应该在 SELECT 子句之后出现。若一个查询的 SELECT 子句后面并没有列出其 GROUP BY 子句后出现的列，则该查询虽然仍旧会被执行并得到结果，但结果往往益处不大。请看 Query17。

Query17 的文本描述： 对每个代理商，检索其供应的产品数量及平均价格。

Query17：
```
SELECT      COUNT(*), AVG(productprice)
FROM        product
GROUP BY    vendorid;
```

COUNT(*)	AVG(ProductPrice)
2	95
4	121.25

图 5-17　Query17 的结果

图 5-17 给出了 Query17 的结果。

查询显示了正确的结果，但终端用户可能会不明白结果中的值所指代的含义。特别地，终端用户更加无法意识到结果的第一行指代代理商 PG，第二行指代代理商 MK。在查询的 SELECT 子句后面加上 VendorID 后（如 Query16 所示），查询结果将会变得更易理解。

查询可以把 WHERE 与 GROUP BY 结合起来使用。请看 Query18。

Query18 的文本描述： 对每个代理商，检索其代理商编号、所售产品中价格大于等于 $100 的产品数量。

Query18：
```
SELECT      vendorid, COUNT(*)
FROM        product
WHERE       productprice >= 100
GROUP BY    vendorid;
```

VendorID	COUNT(*)
PG	1
MK	2

图 5-18　Query18 的结果

142 图 5-18 给出了 Query18 的结果。

Query18 将 PRODUCT 关系中产品价格 ProductPrice 大于等于 $100 且具有相同 VendorID 值的记录进行分组。查询给出了每个组的 VendorID 值以及该组中记录的个数。

GROUP BY 子句还可以对多个列同时进行分组。请看 Query19。

Query19 的文本描述： 考虑所有产品的组，每个组内的产品都是由同一个代理商供应且属于同一个种类。检索每个组的代理商编号、产品种类编号、该组中产品的数量及平均价格。

Query19：
```
SELECT      vendorid, categoryid, COUNT(*), AVG(productprice)
FROM        product
GROUP BY    vendorid, categoryid;
```

图 5-19 给出了 Query19 的结果。

VendorID	CategoryID	COUNT(*)	AVG(ProductPrice)
MK	CP	2	200
MK	FW	2	425
PG	CP	1	100
PG	FW	1	90

图 5-19 Query19 的结果

Query19 将 PRODUCT 关系中 VendorID 值与 CategorID 值都相同的记录进行分组，对每个组给出其 VendorID 值、CategoryID 值、组内记录的条数以及组内产品的平均价格。

为了有效阐述 GROUP BY 的概念，让我们再来看看两个 SOLDVIA 关系的查询。SOLDVIA 关系包含每个销售交易中售出的产品信息。

首先请看 Query20。

Query20 的文本描述： 对每个产品，给出产品编号以及在所有销售交易里售出的项目个数。

```
Query20: SELECT     productid, SUM(noofitems)
         FROM       soldvia
         GROUP BY   productid
```

图 5-20 给出了 Query20 的结果。

ProductID	SUM(NoOfItems)
1X1	2
2X2	3
3X3	5
4X4	5
5X5	2
6X6	1

图 5-20 Query20 的结果

接着请看 Query21。

Query21 的文本描述： 对每个产品，给出其产品编号以及包含售出该产品的销售交易个数。

```
Query21: SELECT     productid, COUNT(tid)
         FROM       soldvia
         GROUP BY   productid
```

图 5-21 给出了 Query21 的结果。

ProductID	COUNT(TID)
1X1	2
2X2	2
3X3	1
4X4	2
5X5	1
6X6	1

图 5-21 Query21 的结果

[145]

5.15 HAVING

GROUP BY 子句后面可以跟有 HAVING 关键字。HAVING 子句决定了查询中哪些组将在结果中得到展示，当然也决定了哪些组不在结果中展示。有 HAVING 子句的查询必须同时包含 GROUP BY 子句。请看 Query22。

Query22 的文本描述： 考虑产品的所有分组，每个组的产品拥有相同的种类且由同一个代理商供应。对那些包含产品个数大于 1 的组，给出其代理商编号，产品种类编号，组内产品数量以及组内产品平均价格。

```
Query22: SELECT     vendorid, categoryid, COUNT(*), AVG(productprice)
         FROM       product
         GROUP BY   vendorid, categoryid
         HAVING     COUNT(*) > 1;
```

Query22 对 PRODUCT 关系中有相同的 VendorID 和 CategoryID 的记录进行分组，并对产品个数大于 1 的组，输出其代理商编号 VendorID、种类编号 CategoryID、组内产品数量以及组内产品平均价格。图 5-22 给出了 Query22 的结果。

正如 Query22 中阐释的一样，HAVING 对分组的记录与 WHERE 子句有相同的效果。

WHERE 条件决定哪些记录会出现在结果中，哪些记录不会。HAVING 条件则决定哪些组会出现在结果中，哪些组不会。

VendorID	CategoryID	COUNT(*)	AVG(ProductPrice)
MK	CP	2	200
MK	FW	2	42.5

图 5-22　Query22 的结果

WHERE 和 HAVING 可以出现在同一个查询中，如 Query23 所示。

Query23 的文本描述： 考虑产品的所有分组，同一个组内的产品有相同的种类和相同的代理商，且产品价格大于等于 $50。对每个产品数量大于 1 的组，检索其代理商编号、产品种类编号、组内产品数量以及产品的平均价格。

144

Query23：

```
SELECT      vendorid, categoryid, COUNT(*), AVG(productprice)
FROM        product
WHERE       productprice >= 50
GROUP BY    vendorid, categoryid
HAVING      COUNT(*) > 1;
```

图 5-23 给出了 Query23 的结果。

VendorID	CategoryID	COUNT(*)	AVG(ProductPrice)
VK	CP	2	200

图 5-23　Query23 的结果

Query23 将找出 PRODUCT 关系中 ProductPrice 大于等于 $50 的所有记录。然后把具有相同 VendorID 和 CategoryID 的记录进行分组。对每个记录个数大于 1 的分组，在结果中列出其 VendorID、CategoryID、组内记录数量以及组内产品平均价格。

为了着重阐述 HAVING 的概念，下我们再来看几个 SOLDVIA 关系中带有 HAVING 子句的 GROUP BY 查询的例子。

首先请看 Query24。

Query24 的文本描述： 对那些在所有的销售交易中被卖出超过 3 个的产品，检索其产品编号以及被卖出的总个数。

Query24：

```
SELECT      productid, SUM(noofitems)
FROM        soldvia
GROUP BY    productid
HAVING      SUM(noofitems) > 3;
```

ProductID	SUM(NoOfItems)
3X3	5
4X4	5

图 5-24　Query24 的结果

图 5-24 给出了 Query24 的结果。

接下来请看 Query25。

Query25 的文本描述： 对那些被两个或两个以上的销售交易卖出的产品，检索其产品编号以及包含该产品的销售交易的个数。

Query25：

```
SELECT      productid, COUNT(tid)
FROM        soldvia
GROUP BY    productid
HAVING      COUNT(tid) > 1;
```

ProductID	COUNT(TID)
1X1	2
2X2	2
4X4	2

图 5-25　Query25 的结果

145　图 5-25 给出了 Query25 的结果。

下面的两个查询是上面两个查询稍作改变后的版本，用以说明HAVING子句中的分组选择条件可以包含聚集函数，且该聚集函数并没有出现在该查询的SELECT子句中。

Query26的文本描述： 对那些在所有的销售交易中被售出超过3个的产品，检索其产品编号。

Query26：
```
SELECT      productid
FROM        orderitem
GROUP BY    productid
HAVING      SUM(noofitems) > 3;
```

ProductID
3X3
4X4

图5-26 Query26的结果

图5-26给出了Query26的结果。

在Query26中，HAVING部分的SUM(noofitems)并没有出现在SELECT子句中，Query27也同样说明了这一点，该查询的SELECT子句并没有用到HAVING子句的聚集函数COUNT(lid)。

Query27的文本描述： 对那些在两个或两个以上销售交易中出现的产品，检索其产品编号。

Query27：
```
SELECT      productid
FROM        orderitem
GROUP BY    productid
HAVING      COUNT(lid) > 1;
```

ProductID
1X
2X2
4X4

图5-27 Query27的结果

图5-27给出了Query27的结果。

5.16 嵌套查询

有时一个查询中可能包含其他一个或多个查询。包含在其他查询里的查询称为**嵌套查询**（nested query）。嵌套查询也称为**内部查询**（inner query）。包含了嵌套查询的查询则称为**外部查询**（outer query）。

例如，现有一个查询需要检索价格低于平均价格产品的产品编号、产品名称及产品价格。如Query28中所示，WHERE子句的条件是平均价格，而平均价格是另一个查询的输出结果。在这种情况下，Query28中第一行的SELECT查询是从外部查询开始的，圆括号里面的查询就是嵌套查询（或内部查询）。

Query28的文本描述： 对所有价格低于平均价格的产品，检索其产品编号、产品名称以及产品价格。

Query28：
```
SELECT      productid, productname, productprice
FROM        product
WHERE       productprice < (SELECT AVG(productprice)
                            FROM product);
```

图5-28给出了Query28的结果。

ProductID	ProductName	ProductPrice
1X1	Zzz Bag	100
2X2	Easy Boot	70
3X3	Cosy Sock	15
4X4	Dura Boot	90

图5-28 Query28的结果

初学者的另一个常见错误，就是试图把Query28写成如下形式：

Query28：无效

```
SELECT      productid, productname, productprice
FROM        product
WHERE       productprice = AVG (productprice),
                                  ERROR MESSAGE RETURNED
```

[146]

依照句法，聚集函数只能出现在 SELECT 子句或者 HAVING 部分。若一个聚集函数是在 SELECT 关键字与 FROM 关键字之间给出，那么该聚集函数就会作用到关系中的所有记录（若命令中没有 GROUP BY 子句）或者是作用到组内的所有记录（若命令中含有 GROUP BY 子句）。若命令中使用了 GROUP BY，则聚集函数还可以用于 HAVING 分组选择条件。SQL 查询中的聚集函数除了用于语句中的 SELECT 和 HAVING 部分以外，没有其他有效使用形式。因此，上面的 Query28 INVALID 将不会得到执行。AVG(ProductPrice) 函数必须是对某个表而言的，但在 Query28 INVALID 中却没有指明 AVG(ProductPrice) 所对应的表。

5.17　IN

在 SQL 中，IN 关键字用于单个值与多个值组合而成的集合之间的比较。请看 Query29。

Query29 的文本描述： 对那些在所有的销售交易中被售出超过 3 个的产品，检索其产品编号以及产品价格。

```
Query29:  SELECT      productid, productname, productprice
          FROM        product
          WHERE       productid IN
                      (SELECT       productid
                      FROM          soldvia
                      GROUP BY      productid
                      HAVING        SUM(noofitems) > 3);
```

图 5-29 给出了 Query29 的结果：

ProductID	ProductName	ProductPrice
3X3	Cosy Sock	15
4X4	Ours Boot	90

图 5-29　Query29 的结果

这个查询阐释了如何用关键字 IN 实现与数值集合的比较，该数值集合即嵌套查询的输出结果。Query29 中的嵌套查询（内部查询）与 Query26 相同，该嵌套查询的输出结果是那些被售出数量总和超过 3 个的所有产品所对应的 ProductID 构成的集合。换言之，就是检查那个产品的编号 ProductID 与该集合中任意一个值相匹配。对于那些已经包含在集合中的产品，本次查询将输出其产品名 ProductName 以及产品价格 ProductPrice。

Query30 是另一个使用 IN 语句的嵌套查询例子。

Query30 的文本描述： 对那些被两个或多个销售交易售出的产品，检索其产品名称及产品价格。

[147]

```
Query30:  SELECT      productid, productname, productprice
          FROM        product
          WHERE       productid IN
                      (SELECT       productid
                      FROM          soldvia
                      GROUP BY      productid
                      HAVING        COUNT(*) > 1);
```

图 5-30 给出了 Query30 的结果。

ProductID	ProductName	ProductPrice
[illegible]	Zip Bag	100
4X4	Dura Boot	90
2X2	Easy Boot	70

图 5-30 Query30 的结果

Query30 中的嵌套查询（内部查询）与 Query27 相同，该嵌套查询的输出结果是那些出现在两个或两个以上销售交易中的产品所对应的 ProductID 构成的集合。外部查询则使用这个集合来判断那个产品的编号与集合中任意一个能相匹配。对这些匹配成功的产品，查询输出其产品名 ProductName 以及产品价格 ProductPrice。

5.18 JOIN

目前为止我们讨论的查询都是针对单个表的查询。接下来我们将描述如何同时实现多个表的查询。SQL 中的 JOIN（连接）操作可以实现这一功能。Query31 给出了 JOIN 用途的实例。

Query31 的文本描述：对每个产品，检索其产品编号、产品名称、产品代理商名称及产品价格。

```
Query31: SELECT    product.productID, productName, vendorname, productprice
         FROM      product, vendor
         WHERE     product.vendorID = vendor.vendorID;
```

图 5-31 给出了 Query31 的结果。

ProductID	ProductName	VendorName	ProductPrice
1X1	Zip Bag	Pacifica Gear	100
4X4	Dura Boot	Pacifica Gear	90
2X2	Easy Boot	Mountain King	70
9X9	Cozy Sock	Mountain King	15
5X5	Tiny Tent	Mountain King	150
6X6	Digga Tent	Mountain King	250

图 5-31 Query31 的结果

对每个产品，Query31 检索了产品标识、产品名称、代理商名称、产品价格。其中 ProductID、ProductName 以及 ProductPrice 是关系 PRODUCT 中的列，而 VendorName 是另一个关系 VENDOR 中的列。因而为了检索到需要的信息，我们需要同时查询两张表。WHERE 关键字后面的表达式就是 JOIN 条件，该条件表明了两张表是如何进行连接的。注意，来自 PRODUCT 关系中的列 VendorID 以及来自 VENDOR 关系中的列 VendorID 前面都有关系名和一个句点限定。这种限定是十分必要的，因为这两列有相同的名字，必须通过关系名进行区分。

注意理解 Query31 语句中 JOIN 条件的重要性：

```
WHERE     product.vendorID = vendor.vendorID;
```

为理解查询中该部分的意义和重要性，下面请看 Query32。Query32 同样从两个表中检索了数据，但是却没有使用 JOIN 条件。

```
Query32: SELECT    productID, productname, vendorname, productprice
         FROM      product, vendor
```

图 5-32 给出了 Query32 的结果。

ProductID	ProductName	VendorName	ProductPrice
1X1	Zzz Bag	Pacifica Gear	100
2X2	Easy Boot	Pacifica Gear	70
3X3	Cosy Sock	Pacifica Gear	15
4X4	Dura Boot	Pacifica Gear	90
5X5	Tiny Tent	Pacifica Gear	150
6X6	Biggy Tent	Pacifica Gear	250
1X1	Zzz Bag	Mountain King	100
2X2	Easy Boot	Mountain King	70
3X3	Cosy Sock	Mountain King	15
4X4	Dura Boot	Mountain King	90
5X5	Tiny Tent	Mountain King	150
6X6	Biggy Tent	Mountain King	250

图 5-32 Query32 的结果

由于没有使用 JOIN 条件，Query32 并不是为每个产品分别给出一个单独的记录，而是给出了一个笛卡尔积，将前一个关系中的每条记录分别与后一个关系中的每条记录进行连接，不管两条记录是不是有相同的 VendorID 值，PRODUCT 关系中的每条记录与 VENDOR 关系中的每条记录都进行了连接。因此，Query32 的结果一共有 12 行，这是因为 PRODUCT 关系有 6 条记录，VENDOR 关系有 2 条记录（6 x 2 = 12）。

下面的两个例子说明了 JOIN 操作是如何工作的。首先请看 Query33，该查询是 Query32 的扩展版本，查询输出了 PRODUCT 和 VENDOR 两个关系进行笛卡尔积以后得到的所有列。

```
Query33: SELECT    *
         FROM      product, vendor
```

图 5-33 给出了 Query33 的结果。

来自关系 PRODUCT　　来自关系 VENDOR

ProductID	ProductName	ProductPrice	VendorID	CategoryID	VendorID	VendorName
1X1	Zzz Bag	100	PG	CP	PG	Pacifica Gear
2X2	Easy Boot	70	MK	FW	PG	Pacifica Gear
3X3	Cosy Sock	15	MK	FW	PG	Pacifica Gear
4X4	Dura Boot	90	PG	FW	PG	Pacifica Gear
5X5	Tiny Tent	150	MK	CP	PG	Pacifica Gear
6X6	Biggy Tent	250	MK	CP	PG	Pacifica Gear
1X1	Zzz Bag	100	PG	CP	MK	Mountain King
2X2	Easy Boot	70	MK	FW	MK	Mountain King
3X3	Cosy Sock	15	MK	FW	MK	Mountain King
4X4	Dura Boot	90	PG	FW	MK	Mountain King
5X5	Tiny Tent	150	MK	CP	MK	Mountain King
6X6	Biggy Tent	250	MK	CP	MK	Mountain King

图 5-33 Query33 的结果

接下来请看 Query34，该查询在 Query33 的基础上添加了 JOIN 条件。

```
Query34: SELECT    *
         FROM      product, vendor
         WHERE     product.vendorid = vendor.vendorid;
```

图 5-34a 给出了 Query34 产生的结果。

来自关系 PRODUCT　　来自关系 VENDOR

ProductID	ProductName	ProductPrice	VendorID	CategoryID	VendorID	VendorName
1X1	Zzz Bag	100	PG	CP	PG	Pacifica Gear
2X2	Easy Boot	70	MK	FW	PG	Pacifica Gear
3X3	Cosy Sock	15	MK	FW	PG	Pacifica Gear
4X4	Dura Boot	90	PG	FW	PG	Pacifica Gear
5X5	Tiny Tent	150	MK	CP	PG	Pacifica Gear
6X6	Biggy Tent	250	MK	CP	PG	Pacifica Gear
1X1	Zzz Bag	100	PG	CP	MK	Mountain King
2X2	Easy Boot	70	MK	FW	MK	Mountain King
3X3	Cosy Sock	15	MK	FW	MK	Mountain King
4X4	Dura Boot	90	PG	FW	MK	Mountain King
5X5	Tiny Tent	150	MK	CP	MK	Mountain King
6X6	Biggy Tent	250	MK	CP	MK	Mountain King

product.vendorid = vendor.vendorid

图 5-34a　Query34 结果的产生

查询语句的 SELECT 和 FROM 子句创建了一个笛卡尔积（PRODUCTVENDOR）。继而如图 5-34a 中所示，查询语句中的 JOIN 条件（即 WHERE 子句）将识别出那些来自 PRODUCT 关系的 VendorID 值与来自 VENDOR 关系的 VendorID 值相匹配的所有记录。一旦有这样的记录行，则显示出来。图 5-34b 给出了 Query34 的结果。 149

ProductID	ProductName	ProductPrice	VendorID	CategoryID	VendorID	VendorName
1X1	Zzz Bag	100	PG	CP	PG	Pacifica Gear
4X4	Dura Boot	90	PG	FW	PG	Pacifica Gear
2X2	Easy Boot	70	MK	FW	MK	Mountain King
3X3	Cosy Sock	15	MK	FW	MK	Mountain King
5X5	Tiny Tent	150	MK	CP	MK	Mountain King
6X6	Biggy Tent	250	MK	CP	MK	Mountain King

图 5-34b　Query34 的结果

要把 Query34 变成 Query31，只需要将 SELECT 关键字后面的星号改为需要陈列的列名：ProductID、ProductName、VendorName、ProductPrice。 150

5.19　别名

在查询语句的 FROM 子句中，每个关系都可以设定一个**别名**（alias）。别名可以是任意一个其他的名字（通常较短）。查询中的任何地方都可以使用别名来代替长的关系名。

别名可以是一个或多个字符，首字符必须与原名首字符一致，其余字符可以是任意字母或数字。

别名本身对查询没有影响，但却可以提高查询的可读性，以方便需要阅读这些查询的用户或开发人员。若我们在每个列名之前加上关系的别名，这样就可以明显看出这个列出自哪个关系。设定了别名以后，则不需要在每个列前面加上实际的表名，而是可以使用更简的别名来精简查询语句的长度。

例如，Query31 中使用了别名以后的不同形式见 Query31a。

```
Query31a: SELECT    p.productid, p.productname, v.vendorname, p.productprice
          FROM      product p, vendor v
          WHERE     p.vendorid = v.vendorid;
```

Query31a 和 Query31 的执行过程完全相同，输出的记录行也完全相同。该查询所做的形式改变仅仅是为了提高可读性。在语句的 FROM 部分，PRODUCT 关系后面紧跟的是其别名 p；类似地，VENDOR 关系的别名是 v。查询中的其他地方使用了这些别名，以指明某个列出自的关系。

别名还可以用来重命名查询得到的列，如 Query31b 所示。

```
Query31b: SELECT    p.productid pid, p.productname pname,
                    v.vendorname vname, p.productprice pprice
          FROM      product p, vendor v
          WHERE     p.vendorid = v.vendorid;
```

Query31b 和 Query31 的执行过程与输出记录行也是相同的，唯一的区别在于该查询结果中列名将使用别名而不是使用原来的列名，如图 5-31b 所示。

PID	PName	VName	PPrice
1X1	Zzz Bag	Pacifica Gear	100
2X2	Easy Boot	Mountain King	70
3X3	Cosy Sock	Mountain King	15
4X4	Dura Boot	Pacifica Gear	90
5X5	Tiny Tent	Mountain King	150
6X6	Biggy Tent	Mountain King	250

图 5.31b　Query31b 的结果

WHERE 子句中的列名不能使用别名，必须使用其本身的列名，在 GROUP BY 和 HAVING 子句中同样如此。关系表的别名没有类似限制，可以用于 SELECT 语句的任何地方。

使用别名的语句中还可以使用 SQL 关键字 AS，如 Query31c 中所示。

```
Query31c: SELECT    p.productid AS pid, p.productname AS pname,
                    v.vendorname AS vname, p.productprice AS pprice
          FROM      product p, vendor v
          WHERE     p.vendorid = v.vendorid;
```

Quer31c 与 Query301b 的执行过程以及执行结果完全相同。

5.20　多关系连接

一个查询可以包含多个 JOIN 来连接多个关系。请看 Query35。

Query35 的文本描述：对销售交易的每一行，给出交易标识、交易日期、售出的产品

名称、售出数量以及产品交易金额。

Query35：

```
SELECT      t.tid, t.tdate, p.productname,
            sv.noofitems AS quantity,
            (sv.noofitems * p.productprice) AS amount
FROM        product p, salestransaction t, soldvia sv
WHERE       sv.productid = p.productid AND
            sv.tid = t.tid
ORDER BY    t.tid;
```

对每一行销售交易记录，Query35 检索其交易标识、交易日期、售出的产品名、售出数量以及交易金额。TID 和 TDate 两个列来自于 SALESTRANSACTION 关系。ProductName 是 PRODUCT 关系中的列。NoOfItems 列来自于 SOLDVIA 关系，并且在结果中被重命名为数量 Quantity。金额 Amount 来自 SOLDVIA 关系中的 NoOfItems 列以及 PRODUCT 关系中的 ProductPrice 列。

Query35 的 WHERE 子句中的 AND 操作连接了两个 JOIN 条件：一个条件连接 SOLDVIA 关系与 PRODUCT 关系，另一个则连接 SOLDVIA 关系和 SALESTRANSACTION 关系。Query35 的查询结果如图 5-35 所示。 152

TID	TDate	ProductName	Quantity	Amount
T111	01-JAN-13	Zzz Bag	1	100
T222	01-JAN-13	Easy Boot	1	70
T333	02-JAN-13	Zzz Bag	1	100
T333	02-JAN-13	Cosy Sock	5	75
T444	02-JAN-13	Dura Boot	1	90
T444	02-JAN-13	Easy Boot	2	140
T555	02-JAN-13	Biggy Tent	1	250
T555	02-JAN-13	Dura Boot	4	360
T555	02-JAN-13	Tiny Tent	2	300

图 5-35 Query35 的结果

到目前为止，本书介绍了用于创建和移除关系以及在关系中插入数据、检索数据的相关 SQL 语句。接下来，本书将会演示如何利用 SQL 改变关系的结构（用 ALTER TABLE 命令）、修改关系中的数据（用 UPDATE 命令）以及删除关系中的数据（用 DELETE 命令）。

5.21 ALTER TABLE

ALTER TABLE 命令用于修改已创建的关系结构。例如，若用户想要向关系 VENDOR 中增加一个可选列 VenderPhoneNumber，就需要使用以下 ALTER TABLE 语句：

Alter 语句 1：

```
ALTER TABLE vendor ADD
        vendorphonenumber      CHAR(12);
```

这条语句[⊖]将会在关系 VERDOR 中增加一个新的列。VERDOR 中每一条记录对应该列的值都被初始化为 NULL。

ALTER TABLE 语句也可以用来删除列。例如，如果用户想要删除关系 VENDOR 中的列 VenderPhoneNumber，则可以使用以下语句：

⊖ 请参看本节 5.40.4 节 SQL 语法差异说明中有关 ALTER TABLE 在不同 RDBMS 软件中的不同语义。

Alter 语句 2：
```
ALTER TABLE vendor DROP
        vendorphonenumber;
```

除了增加和删除列之外，ALTER TABLE 命令还可用于其他关系结构的改变操作，本章后面部分将会给出介绍。

5.22　UPDATE

SQL 中 UPDATE 命令用于修改存于数据库关系中的数据。我们可以使用以下 INSERT INTO 语句向关系 PRODUCT 中插入一件新的产品：

Insert 语句 1：
```
INSERT INTO product VALUES ('YX7', 'Alloy Brick', 1000, 'MK', '19');
```

新插入产品的实际价格本是 $10，但由于使用 INSERT INTO 语句时输入错误，误将关系 PRODUCT 中此产品的价格设定为了 $1000。为了修改此产品的价格，需要用到如下 UPDATE 语句：

Update 语句 1：
```
UPDATE      product
SET         productprice = 10
WHERE       productid = 'YX7';
```

在以上 UPDATE 语句中，关键词 UPDATE 后面是待更新记录所在表的表名。SET 子句指定哪列（或多列）将被更新及更新为何值。WHERE 子句选择哪些记录将被更新。

我们可以使用如下 ALTER TABLE 语句实现向关系 PRODUCT 中再增加一列：

Alter 语句 3：
```
ALTER TABLE product ADD
        Discount  NUMBER(1,2);
```

初始时关系 PRODUCT 中新增加的 Discount 列都被赋值为 NULL。假设现需要对所有产品设定 20% 的折扣，则需要使用如下 UPDATE 语句：

Update 语句 2：
```
UPDATE      product
SET         discount = 0.2;
```

153

由于该 UPDATE 语句中没有 WHERE 子句，因而所有记录的 Discount 列都将被设为 0.2。

假设现在我们需要将供应商 MK 所提供产品的折扣增加到 30%，那么就需要使用如下 UPDATE 语句：

Update 语句 3：
```
UPDATE      product
SET         discount = 0.3
WHERE       vendorid = 'MK';
```

现在所有供应商为 MK 的记录，其 Discount 列的值均更改为 0.3。

如果我们不再需要列 Discount，则可用如下 ALTER TABLE 语句删除该列：

Alter 语句 4：
```
ALTER TABLE product DROP
        discount;
```

5.23　DELETE

SQL 中 DELETE 命令用于删除数据库关系中的数据。如果要从关系 PRODUCT 中删除 Product ID 为 7×7 的产品，就可以使用如下 DELETE 语句：

Delete 语句 1：
```
DELETE FROM     product
WHERE           productid = '7x7';
```

在上面的 DELETE 语句中，关键词 DELETE FROM 后面是待删除记录所在表的表名。WHERE 子句确定那条记录（哪些记录）将被删除。注意，如果 DELETE 子句没有 WHERE 条件，则整个表的所有记录都将被删除。这种情况下，虽然表中所有记录都被删除了，但是表结构仍然存在。这和使用 DROP TABLE 命令不同，DROP TABLE 命令会同时删除表中所有记录以及表结构本身。

事实上，INSERT INTO、DELETE、UPDATE 语句通常不是由负责数据修改和删除的终端用户直接写出的。通常的做法是用户通过第 6 章所述表单（form）等前端软件（front-end application）来完成这些操作。前端软件将代替终端用户生成 UPDATE 和 DELETE 等语句，从而完成数据修改和数据删除等操作。

5.24 CREATE VIEW 和 DROP VIEW

视图（view）是 SQL 中的一种机制，该机制允许查询的结构保存在 RDBMS 中。视图也称为虚表，也就是说它不是一个实表，也没有任何物理存储的数据。视图被调用时，执行的是从实表中检索数据的操作。视图可以像其他存储在数据库中的表一样用在 SELECT 语句中。比如，视图中的列可以被选择，也可以和其他表或视图建立连接。视图中的数据还可以被分组，等等。

利用如下 CREATE VIEW 语句创建视图 PRODUCTS_MORE_THAN_3_SOLD。

Create View 语句 1：

```
CREATE VIEW      products_more_than_3_sold AS
SELECT           productid, productname, productprice
FROM             product
WHERE            productid IN
                 (SELECT        productid
                 FROM           soldvia
                 GROUP BY       productid
                 HAVING         SUM (soldbase) > 3);
```

154

以上语句将 Query29 保存为一个视图，这样它就可以用在任意的 SELECT 查询的 FROM 部分。为此，Query29 就可以执行如下。

Query29a：

```
SELECT        *
FROM          products_more_than_3_sold;
```

执行 Query29a 和 Query29 将会产生同样的结果，如图 5-29 所示。

以下语句将会创建视图 PRODUCTS_IN_MULTIPLE_TRNSC。

Create View 语句 2：

```
CREATE VIEW      products_in_multiple_trnsc AS
SELECT           productid, productname, productprice
FROM             product
WHERE            productid IN
                 (SELECT        productid
                 FROM           soldvia
                 GROUP BY       productid
                 HAVING         COUNT(tid) > 1);
```

此视图同样可以像其他常规表一样被查询。如 Query30a 所示。

Query30a：

```
SELECT        *
FROM          products_in_multiple_trnsc;
```

执行 Query30a 和 Query30 将会产生同样的结果，如图 5-30 所示。

若需要移除一个视图，则可用 DROP VIEW 语句。以下两条语句将移除之前创建的两个视图。

Drop View 语句 1：`DROP VIEW products_more_than_3_sold;`

Drop View 语句 2：`DROP VIEW products_in_multiple_trnsc;`

当然，如果需要再次创建以上视图，我们可以简单地再次执行 Create View 语句 1 和 Create View 语句 2。

5.25　集合运算：UNION、INTERSECT、EXCEPT(MINUS)

SQL 同样支持标准的**集合运算**（set operator）：并运算、交运算和差运算。SQL 中的集合运算用于合并两条或者多条能相容的 SELECT 语句所产生的结果。

定义两个集合相容，要求这两个集合列的数目相同，且每个列的数据类型相匹配。换句话说，一个集合的第一列的数据类型能兼容另一个集合第一列的数据类型，第二列的数据类型兼容另一个集合第二列的数据类型，以此类推。

集合运算可以合并 SELECT 语句查询关系表、视图或者其他 SELECT 查询所产生的结果。

为了演示 SQL 中的集合运算，我们将会利用 SELECT 语句查询视图 PRODUCTS_MORE_THAN_3_SOLD 和 PRODUCTS_IN_MULTIPLE_TRNSC（假设上一节的 CREATE VIEW 语句被再次执行）。注意，这两个视图是相容的，因为它们的列的数目相同且有相匹配的数据类型。

155

Query36 将演示 UNION（并）运算的用法。

Query36 的文本描述： 检索所有满足条件的产品的 product id, product name 和 product price，条件为该产品在所有的销售事务中售出超过三件，或者该产品在超过一条销售事务中被售出。

Query36：

```
SELECT    *
FROM      products_more_than_3_sold
UNION
SELECT    *
FROM      products_in_multiple_trnsc;
```

Query36 的结果如图 5-36 所示。

ProductID	ProductName	ProductPrice
1X1	Zzz Bag	100
2X2	Easy Boot	70
3X3	Cosy Sock	15
4X4	Dura Boot	90

图 5-36　Query36 的结果

在 Query36 中，第一条 SELECT 子句找到在所有的销售事务中售出超过三件的产品集合（见图 5-29），第二条 SELECT 子句找到在超过一条销售事务中被售出的产品集合（见图 5-30）。并运算合并这两个集合并且消除重复。如果某一产品售出超过三件并且在超过一条销售事务中出现，则这一产品只会被列出一次。

Query37 演示的是 INTERSECT（交）运算的用法。

Query37 的文本描述：检索所有满足条件的产品的 product id, product name 和 product price，条件为该产品在所有的销售事务中售出超过三件，同时该产品在超过一条销售事务中被售出。

Query37：
```
SELECT      *
FROM        products_more_than_3_sold
INTERSECT
SELECT      *
FROM        products_in_multiple_trans;
```

Query37 的结果如图 5-37 所示。

ProductID	ProductName	ProductPrice
2X4	Dura Boot	60

图 5-37 Query37 的结果

在 Query36 和 Query37 中，第一条 SELECT 子句找出在所有的销售事务中售出超过三件的产品集合，第二条 SELECT 子句找到在超过一条销售事务中被售出的产品集合，交运算找到的是同时出现在这两个集合中的产品。

Query38 演示的是 MINUS（差）运算的用法，在某一确定的 RDBMS SQL 实现中也称为 EXCEPT[⊖]。

Query38 的文本描述：检索所有满足条件的产品的 product id，product name 和 product price，条件为该产品在所有销售事务中售出超过三件，但是该产品没有在超过一条销售事务中被售出。

Query38：
```
SELECT      *
FROM        products_more_than_3_sold
MINUS
SELECT      *
FROM        products_in_multiple_trans;
```

Query38 的结果如图 5-38 所示。

ProductID	ProductName	ProductPrice
8X3	Crazy Sock	15

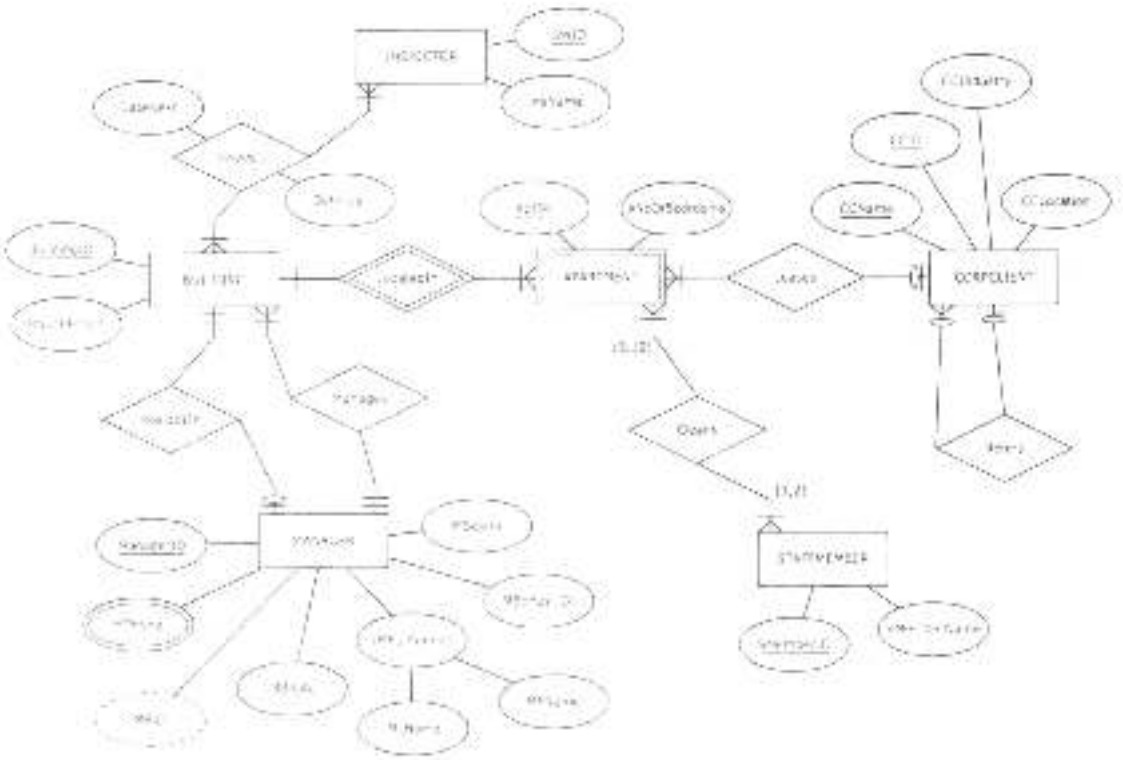
图 5-38 Query38 的结果

在以上的两个查询中，对于 Query38，第一条 SELECT 子句找到在所有的销售事务中售出超过三件的产品集合，第二条 SELECT 子句找到在超过一条销售事务中被售出的产品集合。差运算得到出现在第一个集合但没有出现在第二个集合里的所有产品。

5.26 使用其他 SQL 命令的实例

本章到目前为止，我们用 ZAGI Retail Company Sales Department Database 作为例子概述和演示了最常用的 SQL 命令和语句。接下来，我们将用另一个示例数据库 HAFH Realty Company Property Management Database 补充一些 SQL 命令和语句，同时介绍一些附加的 SQL 命令。该数据库的 ER 图和关系模式已在第 3 章的图 3-59 给出，为方便起见，这里重新给出，如图 5-39a 所示。

⊖ 请查看本章 5.40.5 节 SQL 语法差异说明中有关交和差运算在不同 RDBMS 软件中的其他语义。



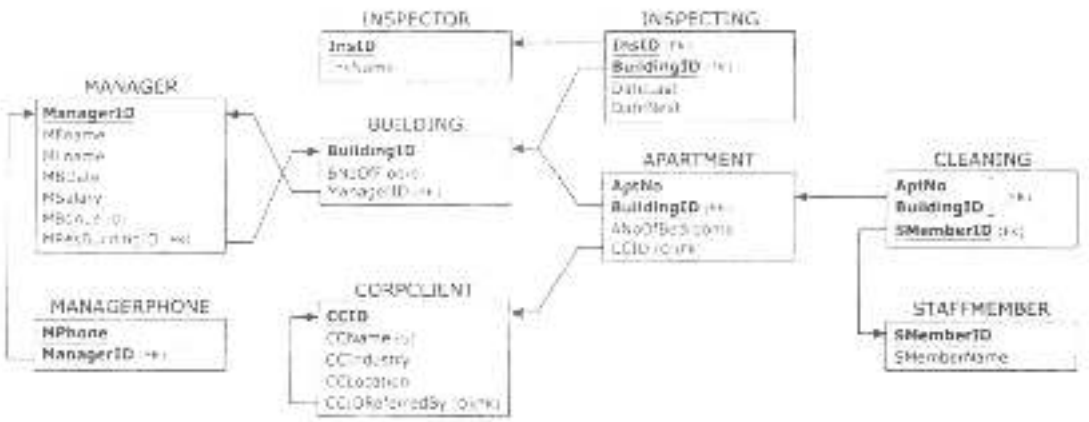


图 5-39a HAFH 房地产公司物业管理部门数据库：ER 图及关系模式

5.27 CREATE TABLE（附加实例）

首先请看图 5-39b 所示的 CREATE TABLE 语句集合，这些语句用来为图 5-39a 所示的 HAFH Realty Company Property Management Database 创建关系模式。

关于图 5-39b 所示的 CREATE TABLE 语句，考虑如下注释信息。

如图 5-39a 所示，实体 CORPCLIENT 的 CCID 和 CCName 属性都是唯一的。在创建关系 CORPLIENT 的 CREATE TABLE 语句中，我们将 CCID 指定为主码。为了标识候选码，

CCName 也是唯一的。在 CREATE TABLE 语句中，我们将 CCName 设计为独立的一列。

```
CREATE TABLE manager
(       managerid               CHAR(4)                         NOT NULL,
        mfname                  VARCHAR(15)                     NOT NULL,
        mlname                  VARCHAR(15)                     NOT NULL,
        mbdate                  DATE                            NOT NULL,
        msalary                 NUMERIC(9,2)                    NOT NULL,
        mbonus                  NUMERIC(9,2),
        mresbuildingid          CHAR(3),
        PRIMARY KEY (managerid) );

CREATE TABLE managerphone
(       managerid               CHAR(4)                         NOT NULL,
        mphone                  CHAR(11)                        NOT NULL,
        PRIMARY KEY (managerid, mphone),
        FOREIGN KEY (managerid) REFERENCES manager(managerid) );

CREATE TABLE building
(       buildingid              CHAR(3)                         NOT NULL,
        bnoofflcors             INT                             NOT NULL,
        bmanagerid              CHAR(4)                         NOT NULL,
        PRIMARY KEY (buildingid),
        FOREIGN KEY (bmanagerid) REFERENCES manager(managerid) );

CREATE TABLE inspector
(       insid                   CHAR(3)                         NOT NULL,
        insname                 VARCHAR(15)                     NOT NULL,
        PRIMARY KEY (insid) );

CREATE TABLE inspecting
(       insid                   CHAR(3)                         NOT NULL,
        buildingid              CHAR(3)                         NOT NULL,
        datelast                DATE                            NOT NULL,
        datenext                DATE                            NOT NULL,
        PRIMARY KEY (insid, buildingid),
        FOREIGN KEY (insid) REFERENCES inspector(insid),
        FOREIGN KEY (buildingid) REFERENCES building(buildingid) );

CREATE TABLE corpclient
(       ccid                    CHAR(4)                         NOT NULL,
        ccname                  VARCHAR(25)                     NOT NULL,
        ccindustry              VARCHAR(25)                     NOT NULL,
        cclocation              VARCHAR(25)                     NOT NULL,
        ccidreferredby          CHAR(4),
        PRIMARY KEY (ccid),
        UNIQUE (ccname),
        FOREIGN KEY (ccidreferredby) REFERENCES corpclient(ccid) );

CREATE TABLE apartment
(       buildingid              CHAR(3)                         NOT NULL,
        aptno                   CHAR(5)                         NOT NULL,
        anoofbedrooms           INT                             NOT NULL,
        ccid                    CHAR(4),
        PRIMARY KEY (buildingid, aptno),
        FOREIGN KEY (buildingid) REFERENCES building(buildingid),
        FOREIGN KEY (ccid) REFERENCES corpclient(ccid) );
```

图 5-39b HAFH 房地产公司物业管理部门数据库的 CREATE TABLE 语句

```
CREATE TABLE staffmember
(       smemberid               CHAR(4)                         NOT NULL,
        smembername             VARCHAR(15)                     NOT NULL,
        PRIMARY KEY (smemberid) );

CREATE TABLE cleaning
(       buildingid              CHAR(3)                         NOT NULL,
        aptno                   CHAR(5)                         NOT NULL,
        smemberid               CHAR(4)                         NOT NULL,
        CONSTRAINT cleaningpk PRIMARY KEY (buildingid, aptno, smemberid),
        CONSTRAINT cleaningfk FORRIGN KEY (buildingid, aptno)
                                REFERENCES apartment(buildingid, aptno) );
        CONSTRAINT cleaningfk2 FORRIGN KEY (smemberid)
                        REFERENCES staffmember(smemberid) );
```

图 5-39b（续）

实体 MANAGER 的 Bonus 属性和实体 APARTMENT 的 CCID 属性是可选的。因此，关系 MANAGER 中的 Bonus 列和关系 APARTMENT 中的 CCID 列都是可选的，同时没有 NOT NULL 限制。

一个客户可以但不是必须被其他客户参照。因此，关系 CORPCLIENT 中的 CCIDReferredBy 外码是可选列。同时，关系 CORPCLIENT 中的 CCIDReferredBy 列没有 NOT NULL 限制。

实体 MANAGER 和 BUILDING 之间存在 ResidesIn 和 Manages 两个联系。关系 MANAGER 中的 MResBuildingID 列作为外码参照到关系 BUILDING，以实现联系 ResidesIn。关系 BUILDING 中的 BManagerID 列作为外码参照到关系 MANAGER，以实现联系 Manages。由于在用于创建关系 MANAGER 的 CREATE TABLE 语句里，并没有将 MResBuildingID 这一列初始声明为外码，因此我们需要在最后引用关系 BUILDING 时将 MResBuildingID 声明为外码。然而，只要关系 MANAGER 比关系 BUILDING 先创建，我们就不能在创建关系 MANAGER 的 CREATE TBALE 语句里声明参照到关系 BUILDING 的外码。因为，关系 BUILDING 不存在[⊖]。为解决这个问题，在最初创建表的时候，不是将 MResBuildingID 作为外码而是作为非外码对待。稍后，我们将会演示如何在已有关系中添加参照完整性约束。

157

158
-
159

联系 Refer 是一元联系。创建 CORPCLIENT 关系的 CREATE TABLE 语句中将会演示一元联系如何实现。外码 CCIDReferredBy 参照关系 CORPCLIENT 自身的主码。

为了说明这些约束（比如主码和外码的声明）可以有名称，我们在创建关系 CLEANING 的 CREATE TABLE 语句中命名了它的约束。为约束命名并不是必需的，但有时会很实用。在本章接下来的约束管理实例中，将会演示一个为约束命名的实用性例子。

5.28　INSERT INTO（附加实例）

数据库 HAFH Realty Company Property Management Database 的数据记录在第 3 章的图 3-60 中，为方便使用，这里将重复给出，如图 5-39c 所示。

⊖ 在 MANAGER 关系之前创建 BUILDING 关系，这样可以让 MResBuildingID 列成为 MANAGER 关系的外码，但同时又阻止了关系 BUILDING 中的 BManagerID 列成为外码。换言之，一个问题会被另一个问题取代。

INSPECTOR

InsID	InsName
I11	Jane
I22	Nikki
I33	Mick

BUILDING

BuildingID	BNoOfFloors	BManagerID
B1	5	M12
B2	6	M23
B3	4	M23
B4	4	M34

APARTMENT

BuildingID	AptNo	ANoOfBedrooms	CCID
B1	41	1	
B1	21	1	C111
B2	11	2	C222
B2	31	2	
B3	11	2	C777
B4	11	2	C777

INSPECTING

InsID	BuildingID	DateLast	DateNext
I11	B1	15-MAY-2012	14-MAY-2013
I11	B2	17-FEB-2013	17-MAY-2013
I22	B2	17-FEB-2013	17-MAY-2013
I22	B3	11-JAN-2013	11-JAN-2014
I33	B3	12-JAN-2013	12-JAN-2014
I33	B4	11-JAN-2013	11-JAN-2014

MANAGER

ManagerID	MFName	MLName	MBDate	MSalary	MBonus	MResBuildingID
M12	Boris	Grant	22-JUN-1980	60000		B1
M23	Austin	Lee	30-OCT-1975	50000	5000	B2
M34	George	Sherman	11-JAN-1978	52000	2000	B4

CLEANING

BuildingID	AptNo	SMemberID
B1	21	5432
B1	41	9876
B2	11	9876
B2	31	5432
B3	11	5432
B4	11	7652

MANAGERPHONE

ManagerID	MPhone
M12	555-2222
M12	555-3232
M23	555-9988
M34	555-9899

STAFFMEMBER

SMemberID	SMemberName
5432	Brian
9876	Boris
7652	Caroline

CORPCLIENT

CCID	CCName	CCIndustry	CCLocation	CCIDReferredBy
C111	BingNotes	Music	Chicago	
C222	Skyjet	Airline	Oak Park	C111
C777	WindyCT	Music	Chicago	C222
C888	SouthAlps	Sports	Rosemont	C777

图 5-39c HAFH 房地产公司物业管理部门数据库的样本数据记录

考察图 5-39d 所示的 INSERT INTO 语句，它是用于为数据库 HAFH Realty Company Property Management Database 插入数据记录的。

```
INSERT INTO manager VALUES ('M12', 'Boris', 'Grant', '20/Jun/1980', 20000, null, null);
INSERT INTO manager VALUES ('M23', 'Austin', 'Lee', '30/Oct/1975', 50000, 5000, null);
INSERT INTO manager VALUES ('M34', 'George', 'Sherman', '12/Jan/1976', 52000, 2000, null);

INSERT INTO managerphone VALUES ('M12','555-2222');
INSERT INTO managerphone VALUES ('M23','555-3232');
INSERT INTO managerphone VALUES ('M23','555-9984');
INSERT INTO managerphone VALUES ('M34','555-3959');

INSERT INTO building VALUES ('B1', '5', 'M12');
INSERT INTO building VALUES ('B2', '6', 'M23');
INSERT INTO building VALUES ('B3', '4', 'M23');
INSERT INTO building VALUES ('B4', '4', 'M34');

INSERT INTO inspector VALUES ('I11', 'Gene');
INSERT INTO inspector VALUES ('I22', 'Niko');
INSERT INTO inspector VALUES ('I33', 'Mink');

INSERT INTO inspecting VALUES ('I11','B1','15/May/2012','14/May/2013');
INSERT INTO inspecting VALUES ('I11','B2','17/Feb/2013','17/May/2013');
INSERT INTO inspecting VALUES ('I33','B3','17/Feb/2013','17/May/2013');
INSERT INTO inspecting VALUES ('I22','B1','11/Jan/2013','11/Jan/2014');
INSERT INTO inspecting VALUES ('I33','B1','12/Jan/2013','12/Jan/2014');
INSERT INTO inspecting VALUES ('I33','B4','11/Jan/2013','11/Jan/2014');

INSERT INTO corpclient VALUES ('C111', 'WingNotes', 'Music', 'Chicago', null);
INSERT INTO corpclient VALUES ('C222', 'SkyCat', 'Airline', 'Oak Park', 'C111');
INSERT INTO corpclient VALUES ('C777', 'WindyCT', 'Music', 'Chicago', 'C222');
INSERT INTO corpclient VALUES ('C888', 'SouthAlps', 'Sports', 'Rosemont', 'C777');

INSERT INTO apartment VALUES ('B1', '31', 1, 'C111');
INSERT INTO apartment VALUES ('B1', '41', 1, null);
INSERT INTO apartment VALUES ('B2', '11', 2, 'C222');
INSERT INTO apartment VALUES ('B2', '31', 2, null);
INSERT INTO apartment VALUES ('B3', '11', 2, 'C777');
INSERT INTO apartment VALUES ('B4', '11', 2, 'C777');

INSERT INTO staffmember VALUES ('5432', 'Brian');
INSERT INTO staffmember VALUES ('9876', 'Boris');
INSERT INTO staffmember VALUES ('7652', 'Caroline');

INSERT INTO cleaning VALUES ('B1', '31', '5432');
INSERT INTO cleaning VALUES ('B1', '41', '9876');
INSERT INTO cleaning VALUES ('B2', '11', '9876');
INSERT INTO cleaning VALUES ('B2', '31', '5432');
INSERT INTO cleaning VALUES ('B3', '11', '5432');
INSERT INTO cleaning VALUES ('B4', '11', '7652');
```

图 5-39d　HAFH 房地产公司物业管理部门数据库：INSERT INTO 插入记录

正如本章之前所提到的那样，记录添加的顺序是很重要的。因为，参照完整性约束要求被参照关系的主码值的录入要先于外码值的录入。例如，在关系 CORPCLIENT 中，记录录入时，要求被参照客户数据的录入要先于参照客户数据的录入。

注意，我们最开始没有为关系 MANAGER 的 MRcsBuildingID 插入数据，而是让这列的值暂时空缺（如图 5-39a 所示）。其原因是实体 MANAGER 和实体 BUILDING 之间存在两个联系。一个联系中 MANAGER 列参照 BUILDING，在另一个联系中 BUILDING 列参照 MANAGER。在一个关系模式中实现以上两个联系会导致**循环外码依赖**（circular foreign key dependency）。

表中不能出现对尚未创建的表的约束。因此，我们选择开始只在关系 BUIILDING 中建立参照完整性约束，在该约束中关系 BUILDING 的外码列参照 MANAGER 关系的主码列。其他参照完整性约束（关系 MANAGER 的外码列参照 BUILDING 的主码列）将会在后面部分进行补充说明。

5.29 约束管理

当原始数据都已经录入以后，我们就可以添加其缺失的参照完整性约束，即关系 MANAGER 的外码 ReBuildingID 参照了关系 BUILDING 的主码，其过程可用如下语句实现：

Alter 语句 5:
```
ALTER TABLE manager
ADD CONSTRAINT fkresidesin
FOREIGN KEY (rebuildingid) REFERENCES building
        (buildingid);
```

只要参照完整性 FKResidesIn 添加到位，我们就可以通过执行如下 UPDATE 语句为关系 MANAGER 的 ReBuildingID 列赋值。

Update 语句 4:
```
UPDATE      manager
SET         rebuildingid = 'B1'
WHERE       managerid = 'M1';
```

Update 语句 5:
```
UPDATE      manager
SET         rebuildingid = 'B2'
WHERE       managerid = 'M2';
```

Update 语句 6:
```
UPDATE      manager
SET         mresbuildingid = 'B4'
WHERE       managerid = 'M34';
```

160 - 161

执行了以上 UPDATE 语句之后，关系 MANAGER 中记录的 MResBuildingID 就已被赋值。所有图 5-39c 所示的原始数据就已都被录入到 HAFH 数据库中。

现在我们就可以实现 MResBuildingID 列的强制性约束要求，相关的 ALTER TABLE 语句如下。

Alter 语句 6:
```
ALTER TABLE     manager
MODIFY          (mresbuildingid NOT NULL);
```

以后所有录入到关系 MANAGER 的记录都必须给 MReaBuildingID 列赋值，同时会匹配关系 BUILDING 中 BuildingID 的值。

162

因为 FKResidesIn 有名称，因此在需要时就可以选择移除表 MANAGER 和表 BUILDING。首先考察如下移除表的语句。

DROP TABLE sequence HAFH database—First seven tables:

```
DROP TABLE cleaning;
DROP TABLE staffmember;
DROP TABLE apartment;
DROP TABLE tenpatient;
DROP TABLE inspecting;
DROP TABLE inspector;
DROP TABLE managerphone;
```

该语句序列将会移除所有关系而不只是关系 BUILDING 和 MANAGER。这是因为关系 BUILDING 和 MANAGER 相互参照。下面的语句序列

DROP TABLE sequence HAFH database—Last two tables (a):

```
DROP TABLE building;
DROP TABLE manager;
```

以及语句序列

DROP TABLE sequence HAFH database—Last two tables (b):

```
DROP TABLE manager;
DROP TABLE building;
```

都不能实现以上要求。然而由于FKResidesIn约束有名称，那么就可以用如下语句删除该约束。

Alter 语句 7：
```
ALTER TABLE manager
DROP CONSTRAINT fkresidesin;
```

反之，这就允许我们执行语句序列DROP TABLE sequence HAFH database—Last two tables(a)。

移除表中参照完整性约束的命令可以完全规避之前执行 DROP TABLE 语句的顺序问题。例如，如果我们要移除 HAFH 数据库中所有的参照约束（假设它们都有名称），那么就可以以任意的顺序执行 DROP TABLE 语句。

5.30　SELECT（附加实例）

接下来介绍附加的SQL查询实例，演示SQL附加的命令和功能。这些查询实例将从HAFH数据库中查询记录。

5.31　关系与自身的连接（自连接）

若一个关系有一个外码参照其本身的主码，则在查询中这个关系就可以和它自己建立连接。这样的连接也称为自连接(self-JOIN)。请看Query39。

Query39 的文本描述：检索参照和被参照的公司客户的名称。

Query39：
```
SELECT    c.cname AS client, r.cname AS recommender
FROM      corpclient c, corpclient r
WHERE     r.ccid = c.ccidreferredby;
```

Query39 的结果如图 5-39 所示。

Client	Recommender
Skylet	BingNotes
WindyCT	Skylet
SouthAlps	WindyCT

图 5-39　Query39 的结果

在这个查询中考察了关系 CORPCLIENT 两次，使用了两个不同的别名 c 和 r。别名 c 代表关系 CORPCLIENT 的客户角色，而别名 r 代表关系 CORPCLIENT 的推荐人角色。它们之间建立连接后使得客户的 CCIDreferredby 的值匹配到推荐人的 CCIDvalue。

注意，由于在这种情况下我们两次参照了同一关系，因而必须使用别名。如果没有使用别名，查询语句就会产生歧义且不会执行（将会返回错误报告）。

5.32　OUTER JOIN

当一个关系外码的值和另一个关系主码的值相匹配时，JOIN 运算能合并这两个关系的记录。Query40 展示了关系 APARTMENT 和关系 CORPCLIENT 之间 JOIN 运算的例子。

Query40：
```
SELECT    a.buildnum, a.apnum, c.cname
FROM      apartment a, corpclient c
WHERE     a.ccid = c.ccid;
```

Query40 的结果如图 5-40 所示。

注意，Building31 中的 Apartment41 和 Building32 中的 Apartment31 没有出现在查询结果里，这是因为它们的 CCID 不能匹配任何公司客户的 CCID。类似地，公司客户 South

Alps 也没有出现在查询结果里。原因也是其 CCID 不能和任一 apartment 相匹配。

BuildingID	AptNo	CCName
B1	21	BlingNotes
B2	11	SkyJet
B3	11	WindyCT
B4	11	WindyCT

图 5-40 Query40 的结果

OUTER JOIN（外连接）语句是 JOIN（连接）[⊖]的变体操作，它将保留那些在连接中不能匹配的记录。有三种形式的外连接：LEFT OUTER JOIN（左外连接）、RIGHT OUTER JOIN（右外连接）和 FULL OUTER JOIN（全外连接）。

Query41 是左外连接的一个例子。

```
Query41: SELECT    a.buildingid, a.aptno, c.ccname
         FROM      apartment a LEFT OUTER JOIN corpclient c
         ON        a.ccid = c.ccid;
```

Query41 的结果如图 5-41 所示。

对左外连接来说，所有出现在左外连接运算之前（左边）关系中的记录（此例中的 APARTMENT）都会被保留在查询结果里，包括那些没能满足匹配条件的记录。若关系中的记录以这种方式与其他关系的记录建立了连接，那些不能匹配的记录将会被设为空值。由图 5-41 可以看出，在使用了左外连接运算后，尽管它们都没有相应的 CCName 值，BuildingB1 中的 Apartment41 仍出现在查询结果里，BuildingB1 中 Apartment31 也出现在查询结果里，其 CCName 值均为 NULL。

BuildingID	AptNo	CCName
B1	21	BlingNotes
B1	41	
B2	11	SkyJet
B2	31	
B3	11	WindyCT
B4	11	WindyCT

图 5-41 Query41 的结果

164

Query42 是右外连接的一个例子。

```
Query42: SELECT    a.buildingid, a.aptno, c.ccname
         FROM      apartment a RIGHT OUTER JOIN corpclient c
         ON        a.ccid = c.ccid;
```

Query42 的结果如图 5-42 所示。

对于右外连接来说，所有出现在右外连接运算之后（右边）关系中的记录（此例中的 CORPCLIENT）都会保留在查询结果里，包括那些没能满足匹配左边关系条件的记录。由图 5-42 可以看出，在使用了右外连接运算后，尽管 South Alps 这条记录没有相应的 BuildingID 和 AptNo 值，客户名 South Alps 仍出现在查询结果里，其 BuildingID 和 AptNo 值为 NULL。

BuildingID	AptNo	CCName
B1	21	BlingNotes
B2	11	SkyJet
B3	11	WindyCT
B4	11	WindyCT
		SouthAlps

图 5-42 Query42 的结果

Query43 是全外连接的一个例子。

```
Query43: SELECT    a.buildingid, a.aptno, c.ccname
         FROM      apartment a FULL OUTER JOIN corpclient c
         ON        a.ccid = c.ccid;
```

⊖ 为与外连接（OUTER JOIN）区分，规定常规连接（JOIN）为内连接（INNER JOIN）。

165 Query43 的结果如图 5-43 所示。

对于个外连接来说[⊖]，即使两个关系没有相匹配的记录，其所有记录也都会保留在查询结果里。

BuildingID	AptNo	CCName
B1	31	BlingNotes
B1	41	
B2	11	Skyjet
B2	31	
B3	11	WindyCT
B4	11	WindyCT
		SouthAtlas

图 5-43　Query43 的结果

5.33　无主码 / 外码组合的连接

两张表之间建立连接也可以不通过一张表外码连接另一张表主码的形式。两张表之间的连接条件是它们相对应的某列具有相同的值。Query44 给出的例子连接了既不是主码也不是外码的两个列。

Query44 的文本描述： 对于每一个 manager，如果有职员和他的 first name 相同，则检索 manager 的 ID、first name、last name 以及相应职员的 ID。

Query44：
```
SELECT    *.managerid, *.mfname, *.mlname, *.smemberid
FROM      manager *, employee *
WHERE     *.mfname = *.sfemployeename
```

Query44 的结果如图 5-44 所示。

MgrID	MgrFname	MgrLname	SmemberID
M12	Boris	Grant	9876

图 5-44　Query44 的结果

5.34　IS NULL

在某一查询中，包含某一列的值与空值相比较时，就需要使用 IS NULL。请看 Query45。

Query45 的文本描述： 检索那些没有奖金的 manager 的记录。

Query45：
```
SELECT    *
FROM      manager
WHERE     (Bonus IS NULL)
```

Query45 的结果如图 5-45 所示。

ManagerID	MFname	MLname	MDate	MSalary	MBonus	MresBuildingID
M12	Boris	Grant	20-JUN-1980	60000		B1

图 5-45　Query45 的结果

5.35　EXISTS

在查询的内查询（嵌套查询）中，若使用了外查询 SELECT 部分列出的某关系的某列或某些列，则将这个内查询称为**相关子查询** (correlated subquery)。在这种情况下，EXISTS 操作就可以用来测试内查询的结果是否为空。请看 Query46。

166

Query46 的文本描述： 检索所有 manager 居住的 building 记录。

⊖ 请查看本章 5.40.6 节语法声音说明，了解不同 RDBMS 软件中定义全外连接（FULL OUTER JOIN）的其他语义。

Query46：
```
SELECT    *
FROM      building b
WHERE     EXISTS
          (SELECT    *
          FROM      manager m
          WHERE     b.buildingid = m.mbuildingid);
```

Query46 的结果如图 5-46 所示。

在 Query46 中，内查询使用了外查询中使用过的关系 BUILDING 的 BuildingID 列。因此，内查询是外查询相关的。对于每一个在关系 BUILDING 中的建筑 X，只要其满足在关系 MANAGER 里有一条 manager 住在 X 中的记录，EXISTS 操作将会返回一个布尔值 TRUE；若没有记录，就返回布尔值 FALSE。

BuildingID	BNOofFloors	BManagerID
B1	5	M12
B2	6	M23
B4	4	M34

图 5-46 Query46 的结果

以上查询是一个相关嵌套查询，因为内查询参照了 BuildingID 列，而它又是外查询中 BUILDING 表的一列。

5.36 NOT

NOT 操作配合条件比较使用，返回布尔值 TRUE 或 FALSE。请看 Query47。

Query47 的文本描述：检索所有没有 manager 居住的 building 记录

Query47：
```
SELECT    *
FROM      building b
WHERE     NOT EXISTS
          (SELECT    *
          FROM      manager m
          WHERE     b.buildingid = m.mbuildingid);
```

Query47 结果如图 5-47 所示。

BuildingID	BNOofFloors	BManagerID
B3	4	M23

图 5-47 Query47 的结果

SQL 中的关键词 NOT 也可以用于其他类型中使用布尔逻辑的查询。例如，NOT 可以和 IN 组合成 NOT IN 条件，或者结合 IS NULL 组成 IS NOT NULL 条件。 167

5.37 从查询中插入关系

查询中检索出的数据可以填入到其他关系中。考察如下示例场景：如果我们想要创建一个关系 CLEANING 的非规范化拷贝，要求除了包含 SMemberID 列还要包含 SMemberName 列。首先，我们需要创建一个空的非规范化关系 CLEANINGDENORMALIZED，该过程可以使用如下 CREATE TABLE 语句：

Create Table 语句 1：

```
CREATE TABLE cleaningdenormalized
        buildingid      CHAR(3)      NOT NULL,
        aptno           CHAR(3)      NOT NULL,
        smemberid       CHAR(4)      NOT NULL,
        smembername     VARCHAR(15)  NOT NULL,
        PRIMARY KEY (buildingid, aptno, smemberid));
```

这个新建立的关系可用如下语句来填充赋值：

Insert 语句 2：

```
INSERT INTO    cleaningdenormalized
SELECT         c.buildingid, c.area, c.memberid, m.membername
FROM           cleaning c, staffmember m
WHERE          c.memberid = m.memberid;
```

通过结合 INSERT INTO 语句和 SELECT 查询，就实现了自动将数据录入到关系 CLEANINGDENORMALIZED 中。

5.38　其他 SQL 功能

本章所涵盖的 SQL 功能为理解典型的公司所使用的 SQL 提供了基础。除了本章所提到的功能之外，还存在其他 SQL 功能。一旦熟练掌握本章所介绍的 SQL 功能，以后学习和使用其他 SQL 就简单了。

本章所给出的问题是最基本的 SQL 问题，下面两节将会介绍一些附加的 SQL 问题。

5.39　问题说明：SQL 中观测值使用不当

初学者中最常见的一个 SQL 错误就是创建过于简单的查询，并通过不当地使用观测值而得到正确的查询结果。考察如下从 ZAGI Company Sales Department Database 中检索数据的要求。

Request A：对于在所有的销售事务中售出超过三件的产品，检索其 product id、product name 和 product price。

考察如下两个 SQL 查询：

SQL Query A

```
SELECT      productid, productname, productprice
FROM        product
WHERE       productid IN
            (SELECT     productid
            FROM        salesitem
            GROUP BY    productid
            HAVING      SUM(productquantity) > 3);
```

SQL Query B

```
SELECT      productid, productname, productprice
FROM        product
WHERE       productid IN ('3X3', '4X4');
```

158

因为 ProductID 的值为 3×3 和 4×4 的产品销量超过三件，所以这两个查询有相同的查询结果。

ProductID	ProductName	ProductPrice
3X3	Cosy Sock	15
4X4	Dura Boot	90

然而 Query A 才是满足 Request A 的正确查询，Query B 则不是。

Query A 使用 SQL 命令，确定哪些产品在销售事务中售出超过三件。即使数据库中表的数据更新时，它也可以产生正确的查询结果。

另一方面，Query B 是基于观测值的。在目前看来，ProductID 的值为 3×3 和 4×4 的

产品销量超过三件，所以会产生正确的结果。当这种情况不再适用时，数据库有更新（如添加了新的销售量超过三件的产品），则 Query B 就会产生不正确的结果。

正如此例所演示的那样，在 SQL 查询中，当 SQL 命令可以用来代替从数据库中检索所需的值的条件时，不要在查询条件里直接使用属性列的值。注意，对于 Request B，Query B 就可以产生正确的结果。

Request B：对满足条件 ProductID 的值为 3×3 和 4×4 的产品，检索其 product id、product name 和 product price。

5.40 问题说明：SQL 标准和 SQL 语法差异

20 世纪 70 年代 IBM 开发了 SQL，它成为了从关系型数据库中查询数据的标准语言。到 20 世纪 80 年代，它被美国国家标准协会 (ANSI) 和国际标准化组织 (ISO) 所采用。这个标准分为几部分，除了本章所讨论的语言逻辑框架和核心要素之外，还存在很多扩展，用以进一步加强 SQL 的功能。这些扩展包括如何将 SQL 与其他程序语言、外部数据、多媒体等联合使用，以及其他日益增加的功能。

这个标准被初次采用之后，ANSI 和 ISO 又颁布了多个版本的 SQL 标准。与之前的版本一样，最新的版本也以年份命名，即 SQL:2011。通常情况下，RDBMS 的供应商们都会努力遵守 SQL 标准。但是，供应商实现 SQL 时存在差异，包括标准里规定的 SQL 命令和功能的实现数量以及标准里未规定的扩展和修正，都可能存在差异。这就导致 SQL 脚本（SQL 语句集合）很难从一个 DBMS 移植到其他 DBMS。

现在流行的不同 RDBMS 包[⊖]实现 SQL 时会存在细微的 SQL 语法差异。为说明这个问题，本书将就多个主流 RDBMS 包选择性地阐述和讨论一些 SQL 语法差异。这些讨论是从后面列出的 SQL 语法差异中截取的一部分。正如后文所示，这些差异是很细微的，SQL 的使用者通过一点额外的学习就可以很容易地从一个 RDBMS 切换到另一个 RDBMS。 169

5.40.1 SQL 语法差异 1：DATE 和 TIME 数据类型

在不同的 RDBMS 中，一些 SQL 数据类型在语法上略有差异。例如，在很多 RDBMS 包（如 MySQL、Microsoft SQL Server、PostgreSQL、Teradata、IBM DB2）中，除了 DATE 型数据可记录日期之外，TIME 型数据也可记录一天中的时刻。

然而，Oracle 没有特定的数据类型 TIME。其数据类型 DATE 同时包含日期和时刻。这与 MySQL 和 Microsoft SQL Server 中结合日期和时刻类型实现的 DATETIME 数据类型相同。

除了 DATE 和 TIME 这两种数据类型，现在的 RDBMS 包中，还存在其他与日期和时间相关的数据类型，比如 TIMESTAMP 数据类型可用于记录高精度（微秒甚至更高精度）的日期和时间的值，并同时兼具时区声明的功能；而 INTERVAL 数据类型则可用于记录时间间隔。

5.40.2 SQL 语法差异 2：FOREIGN KEY

在很多的 RDBMS 包中，指定外码时可以不必显式指出此外码所参照的关系的主码名称。例如，以下用于声明外码的 CREATE TABLE 语句在多数的 RDBMS 包（如 Oracle、

⊖ 本书的完整资源（可从 dbtexbook.com 处得到）包含了本章提及的 6 种主流 RDBMS 包（Oracle、MySQL、Microsoft SQL Server、PostgreSQL、Teradata 和 IBM DB2）所有 SQL 语句代码的完整版本。

PostgreSQL、Teradata、IBM DB2、Microsoft SQL Server）中是有效的：

```
FOREIGN KEY (vendorid) REFERENCES vendor
```

但是，在有些 RDBMS 包（如 MySQL）中必须显式指出此外码所参照关系的主码名称，如下所示：

```
FOREIGN KEY (vendorid) REFERENCES vendor(vendor.id);
```

这种声明外码的方式在不需要显式指出主码名称的 RDBMS 包（如 Oracle 和 IBM DB2）中同样适用。

5.40.3　SQL 语法差异 3：别名关键词 AS 的使用

在多数 RDBMS 包中，表和列都可以使用别名关键词 AS。例如，以下查询语句可用于 PostgreSQL、Teradata、MySQL、Microsoft SQL Server：

```
SELECT      p.productid AS pid, p.productname AS pname,
            v.vendorname AS vname, p.productprice AS pprice
FROM        product AS p, vendor AS v
WHERE       p.vendorid = v.vendorid;
```

但是，这些语句不能在 Oracle 中适用，因为 Oracle 不允许用 AS 关键词指定关系的别名。在 Oracle 中同样的声明如下：

```
SELECT      p.productid AS pid, p.productname AS pname,
            v.vendorname AS vname, p.productprice AS pprice
FROM        product p, vendor v
WHERE       p.vendorid = v.vendorid;
```

或者

```
SELECT      p.productid pid, p.productname pname,
            v.vendorname vname, p.productprice pprice
FROM        product p, vendor v
WHERE       p.vendorid = v.vendorid;
```

注意，以上三种声明语句同时适用于 PostgreSQL、Teradata、MySQL、Microsoft SQL Server。

[170]

5.40.4　SQL 语法差异 4：ALTER TABLE

在一些 RDBMS 包（如 MySQL、Teradata、MySQL）中，ALTER TABLE 语句里如需使用关键词 ADD，要求使用括号，如下述例子所示：

```
ALTER TABLE vendor ADD
            vendorphonenumber       CHAR(11) );
```

然而，在另一些 RDBMS 包（如 Microsoft SQL Server、PostgreSQL、IBM DB2）中，ALTER TABLE 语句里如需使用关键词 ADD，则不要求使用括号，以上语句改写如下：

```
ALTER TABLE vendor ADD
            vendorphonenumber       CHAR(11) ;
```

除了这些差异之外，在大多数 RDBMS 包中，ALTER TABLE 语句里如需使用关键词 DROP，不要求使用括号。但是 Oracle 除外，它要求在使用关键词 DROP 时使用括号。下面是在不同 RDBMS 包中同一语句的三个不同声明。

第一个在 Oracle 中不被允许，但是在 PostgreSQL、Teradata、MySQL、IBM DB2 中适用。

```
ALTER TABLE vendor DROP vendorphonenumber;
```

第二个在 Oracle 中有要求，但是在其他前述提到的包中不被允许。

```
ALTER TABLE vendor DROP (vendorphone,vendorcity);
```

第二个是另一个 Microsoft SQL Server 要求的需指定关键词 COLUMN 的差异。

```
ALTER TABLE vendor DROP COLUMN vendorphone,vendor);
```

当在 IBM DB2 中声明一条 ALTER TABLE 语句删除一张表中的结构（如一列或一个约束）时，需要添加一条额外的命令来确保查询继续。这条命令应被加在 ALTER TABLE 命令之后，如下所示：

```
ALTER TABLE vendor DROP vendorphoneNumber;
RUNSTATS TABLE cust(all);
```

还有一些关于 ALTER TABLE 命令用法的语法差异。比如，可以在 Oracle 中写出下述的语句：

```
ALTER TABLE manager
      MODIFY (weekbuildingid NOT NULL);
```

但是，在其他 RDBMS 包（如 PostgreSQL、IBM DB2）中，语法差异如下：

```
ALTER TABLE manager
      ALTER weekbuildingid SET NOT NULL;
```

在其他 RDBMS 包（如 MySQL、Microsoft SQL Server）中，不仅存在语法差异，每一列的所有属性（如数据类型、NULL/NOT NULL 等）都要包含在 ALTER 语句里。因此，以上语句应改写如下：

MySQL：

```
ALTER TABLE manager
      MODIFY weekbuildingid CHAR(11) NOT NULL;
```

Microsoft SQL Server：

```
ALTER TABLE manager
      ALTER COLUMN weekbuildingid CHAR(11) NOT NULL;
```

171

5.40.5 SQL 语法差异 5：集合运算

尽管大多数 RDBMS 包在实现并集运算时都遵循统一风格，但是，在实现交运算和差运算时仍然存在差异。

比如，在 Oracle、PostgreSQL、Teradata、IBM DB2、Microsoft SQL Server 中，以下查询执行的是交运算：

```
SELECT      *
FROM        products_more_than_3_sold
INTERSECT
SELECT      *
FROM        products in multiple_times;
```

MySQL 中没有明确的交运算，因此，要实现与此相同的查询，使用如下语句：

```
SELECT DISTINCT      *
FROM                 products_more_than_3_sold
WHERE                (productid, productname, productprice) IN
                     (SELECT * FROM products in multiple_times);
```

考察如下查询，这是一个在 Oracle 中执行差运算的例子：

```
SELECT      *
FROM        products_more_than_3_sold
MINUS
SELECT      *
FROM        products in multiple_times;
```

同样的查询在 PostgreSQL 和 Microsoft SQL Server 中可以表示如下：

```
SELECT      *
FROM        products_more_than_3_sold
EXCEPT
SELECT      *
FROM        products_in_multiple_trnsc;
```

以上两种情况使用了不同的操作来实现同一个查询，这在 Teradata 及 IBM DB2 中都是允许的。

而 MySQL 则不会执行一个完全不同的操作，例如，要实现与上例相同的查询，MySQL 将只能使用如下语句：

```
SELECT      *
FROM        products_more_than_3_sold
WHERE       (productid, productname, productprice) NOT IN
            (SELECT * FROM products_in_multiple_trnsc);
```

5.40.6 SQL 语法差异 6：FULL OUTER JOIN

在不同的 RDBMS 中实现全外连接时存在差异。比如，在 Oracle、PostgreSQL、Teradata、IBM DB2、Microsoft SQL Server 中，以下查询执行的是全外连接：

```
SELECT      a.buildingid, a.aptno, c.cname
FROM        apartment a FULL OUTER JOIN corpclient c
ON          a.ccid = c.ccid;
```

一些 RDBMS 包没有明确的 FULL OUTER JOIN 操作。相应地，用户需要通过联合左外连接和右外连接来实现全外连接。同样的查询在 MySQL 中可以表示如下：

```
SELECT      a.buildingid, a.aptno, c.cname
FROM        apartment a LEFT OUTER JOIN corpclient c
ON          a.ccid = c.ccid;
UNION
SELECT      a.buildingid, a.aptno, c.cname
FROM        apartment a RIGHT OUTER JOIN corpclient c
ON          a.ccid = c.ccid;
```

172

5.40.7 SQL 语法差异 7：约束管理

在本章前几节的叙述中，介绍了如何通过添加和删除完整性约束实现完整性的启用和停用。另外一种在已创建数据库的生命周期里处理启用和停用完整性问题的方式是，使用由某些 RDBMS 包（如 Oracle、Microsoft SQL Server）提供的 ENABLE 和 DISABLE 约束选项。

例如，假设图 5-39b 所示的 CREATE TABLE 语句的前三条已经执行，且关系 MANAGER 已被添加到其中的 FKResidesIn 约束修改。这些约束可以在 HAFH 数据库生命周期内的任意时刻被停用，在 Oracle 和 Microsoft SQL Server 中可以分别通过使用如下语句来实现：

```
ALTER TABLE manager
DISABLE CONSTRAINT fkresidesin;

ALTER TABLE manager
NOCHECK CONSTRAINT fkresidesin;
```

如果需要，在 Oracle 和 Microsoft SQL Server 中可以分别通过使用如下语句重新启用约束：

```
ALTER TABLE manager
ENABLE CONSTRAINT fkresidesin;

ALTER TABLE manager
CHECK CONSTRAINT fkresidesin;
```

与此例中的命令用于移除表中参照完整性约束的情况类似，移除参照完整性的命令也可以用于规避之前删除表的 DROP TABLE 语句的执行顺序问题。比如，我们删除了所有 HAFH 数据库中的参照约束（假设它们都被命名），就能以任意顺序执行 DROP TABLE 命令。

5.40.8 SQL 语法差异 8：GROUP BY

在标准的 SQL 中，当 SELECT 命令使用 GROUP BY 子句时，则出现在关键词 SELECT（不同于聚集函数处所讨论的）之后列的名称必须同时出现在关键词 GROUP BY 之后。例如，在大多数 RDBMS 包（包括 Oracle、PostgreSQL、Teradata、IBM DB2、Microsoft SQL Server）中，下述对 ZAGI Retail Company Database 的查询将是无效的，且会返回一个错误报告：

```
SELECT    vendor_id, product_name, COUNT(*)
FROM      products
GROUP BY  vendor_id; ERROR MESSAGE RETURNED
```

此查询之所以无效，是因为 ProductName 没有在关键词 GROUP BY 之后列出，却在关键词 SELECT 之后列出。这违反了出现在关键词 SELECT 之后列的名称必须同时出现在关键词 GROUP BY 之后这一规则。

然而，MySQL 允许执行这类查询，且会返回如下结果。但它在本质上却是错误/不完全的结果：

VendorID	ProductName	COUNT(*)
MK	Easy Boot	4
PG	Zzz Bag	2

此查询正确地列出了 VendorID 为 MK 的有 4 件产品以及 VendorID 为 PG 的有 2 件产品。但是它允许列出 ProductName（标准的 SQL 不允许），并且只列出每组中一件产品的名称而忽略其他产品的名称。MySQL 允许这些违背 SQL 标准的查询存在，是为了简化某些合法的查询。

作为示例，考察如下查询：

```
SELECT    smemberid, smembername, COUNT(*)
FROM      cleaningdenormalized
GROUP BY  smemberid, smembername;
```

产生如下查询结果：

SMemberID	SMemberName	COUNT(*)
7552	Carolyn	1
5432	Brian	5
9876	Boris	2

此查询在 MySQL 中可以简化如下：

```
SELECT    smemberid, smembername, COUNT(*)
FROM      cleaningdenormalized
GROUP BY  smemberid;
```

这个查询合法，因为 SMembeName 和 SMemberID 在每一组中都是唯一的（因为每一个 SMemberID 对应一个 SMembeName）。

正如上例所演示的那样，在 MySQL 中 GROUP BY 的不标准实现使得某些合法的查询得以缩减，但同时会导致某些查询返回有歧义的结果。

关键术语

aggregate function（聚集函数），139　　alias（别名），151

ALTER TABLE（修改表），153
AVG（求均值），139
circular foreign key dependence（循环外码依赖），160
correlated subquery（相关子查询），166
COUNT（计数），139
CREATE TABLE（创建表），129
CREATE VIEW（创建视图），154
data control language，DCL（数据控制语言），127
data definition language，DDL（数据定义语言），127
data manipulation language，DML（数据操纵语言），127
DELETE（删除），154
DISTINCT（去重），137
DROP TABLE（删除表），131
DROP VIEW（删除视图），155
EXCEPT（差），156
EXISTS（存在），166
FULL OUTER JOIN（全外连接），164
GROUP BY（分组），140
HAVING，144
IN，147
inner query（内查询），146
INSERT INTO，132
INTERSECT（交运算）
IS NULL，166
JOIN（连接），148
LEFT OUTER JOIN（左外连接），164
LIKE，139
MAX，139
MIN，139
MINUS，156
nested query（嵌套查询），146
NOT，167
ORDER BY，138
OUTER JOIN（外连接），164
outer query（外查询），146
query（查询），128
RIGHT OUTER JOIN（右外连接），164
SELECT，134
self-JOIN（自连接），163
set operator（集合运算），155
SQL data type（SQL 数据类型），128
SQL standard（SQL 标准），127
structured query language，SQL（结构化查询语言），127
SUM（求和），139
transaction control language，TCL（事务控制语言），127
UNION（并运算），155
UPDATE（更新），153
view（视图），154
WHERE，136

复习题

Q5.1 DDL SQL 语句的目的是什么？
Q5.2 DML SQL 语句的目的是什么？
Q5.3 CREATE TABLE 命令的目的是什么？
Q5.4 DROP TABLE 命令的目的是什么？
Q5.5 INSERT INTO 命令的目的是什么？
Q5.6 SELECT 命令的目的是什么？
Q5.7 WHERE 条件的目的是什么？
Q5.8 DISTINCT 关键词的目的是什么？
Q5.9 ORDERDED BY 子句的目的是什么？
Q5.10 LIKE 关键词的目的是什么？
Q5.11 SQL 提供了哪些聚集函数？
174 Q5.12 GROUP BY 子句的目的是什么？
Q5.13 HAVING 子句的目的是什么？
Q5.14 什么是嵌套查询？

Q5.15 IN 关键字的目的是什么？
Q5.16 JOIN 条件的目的是什么？
Q5.17 别名的目的是什么？
Q5.18 ALTER TABLE 命令的目的是什么？
Q5.19 UPDATE 命令的目的是什么？
Q5.20 DELETE 命令的目的是什么？
Q5.21 CREATE VIEW 和 DROP VIEW 命令的目的是什么？
Q5.22 SQL 中有哪些聚集操作是可用的？
Q5.23 OUTER JOIN 的目的是什么？
Q5.24 IS NULL 关键字的目的是什么？
Q5.25 EXSITS 关键字的目的是什么？
Q5.26 NOT 关键字的目的是什么？

练习

E5.1 在 ZAGI 零售公司的销售部门数据库中写出能实现下面任务的 SQL 查询：

E5.1a 输出所有代理商的编号 VendorID 以及名称 VendorName。
E5.1b 输出所有顾客的姓名 CustomerName 以及邮政编码 CustomerZip。
E5.1c 输出所有价格大于等于 $100 的产品对应的产品编号 ProductID，产品名称 ProductName、产品价格 ProductPrice。
E5.1d 输出所有产品的产品编号 ProductID、产品名称 ProductName、产品价格 ProductPrice、代理商名称 VendorName，结果根据产品编号 ProductID 排序。
E5.1e 输出所有产品的产品编号 ProductID，产品名称 ProductName、产品价格 ProductPrice、代理商名称 VendorName，种类名称 CategoryName，结果根据产品编号 ProductID 排序。
E5.1f 输出 Camping 种类所有产品的产品编号 ProductID，产品名称 ProductName、产品价格 ProductPrice，结果根据产品编号 ProductID 排序。
E5.1g 输出所有包括 ProductName 为 Dura Boot 的销售交易中的交易编号 TID、顾客姓名 CustomerName、销售日期 TDate。
E5.1h 输出所有区域的区域编号 RegionID，区域名称 RegionName 以及该区域中的商店数量。
E5.1i 输出所有产品种类的种类编号 CategoryID、种类名称 CategoryName 以及该种类中所有产品的平均价格。
E5.1j 输出所有产品种类的种类编号 CategoryID 以及该种类中产品的个数。
E5.1k 对那些销售的所有产品总数大于 5 的交易，输出其 TID 以及所售出的产品总数。
E5.1l 输出最便宜的产品对应的产品编号 ProductID 以及产品名称 ProductName。
E5.1m 输出所有价格低于平均价格的产品的产品编号 ProdeorID、产品名称 ProductName 以及代理商名称 VendorName。
E5.1n 输出销量最多产品的产品编号 ProductID。
E5.1o 使用连接语句编写 Query29（不使用嵌套查询）。
E5.1p 使用连接语句编写 Query30（不使用嵌套查询）。

E5.2 在 HAFH 房地产产权管理数据库中写出能实现下面功能的 SQL 查询：

E5.2a 输出每个联系的编号 SMemberID 以及姓名 SMemberName。
E5.2b 输出每个企业的 CCID、CCName 和 CCIndustry。
E5.2c 输出每个建筑的 BuildingID，BNoOfFloors 以及经理的 MFName，MLName。
E5.2d 对那些薪水低于 $55000 的经理，输出其 MFName、MSalary、MBdate 以及所管理的建筑数量。

E5.2e　输出所有由 WindyCT 公司租赁的房间的 BuildingID 和 AptNo。

E5.2f　输出安排表中排在 1-JAN-2014 以后所有检查员的 InsID 和 InsName，注意结果的去重。

E5.2g　对那些来自 Chicago 地区的公司所租赁的公寓，输出这些公寓的清洁人员的 SMemeberID 和 SMemberName。

E5.2h　对所有 Music industry 中提及了其他顾客的顾客，输出该顾客及提及顾客的 CCName。

E5.2i　输出所有尚未被出租公寓的 BuildingID、AptNo 以及 ANoOfBedrooms。

其他（更大型）的数据集在 dbtextbook.com 处下载。

小案例

小案例 1　Investco Scout

- 为第 3 章小案例 1 中所创建的关系模式描述的 Investco Scout Funds 数据库编写 CREATE TABLE 语句创建关系表。
- 假设下面是要添加到第 2 章小案例 1 所述 Investco Scout Funds 数据库需求中的额外内容：Investco Scout 将记录每个投资公司的 CEOFName 以及 CEOLName（除了对每个投资公司记录的唯一公司标识、唯一公司名称以及公司的多个地址以外）。
- 修改第 2 章小案例 1 的 ER 图以反映需求的相应变化。
- 修改第 3 章小案例 1 的关系模式以反映需求的相应变化。
- 编写反映关系模式变化的 ALTER TABLE 命令。
- 观察下面关于投资公司以及共同基金的信息，创建 INSERT INTO 语句向 Investco Scout Funds 数据库中已创建好的关系表中插入以下信息。

175

投资公司

Company: ACF, Acme Finance	CEO: Mick Dempsey	Locations: Chicago, Denver
Company: TCA, Tara Capital	CEO: Ava Newton	Locations: Houston, New York City
Company: ALB, Albritton	CEO: Lena Dollar	Locations: Atlanta, New York City

证券 (ID, Name, Type)

AE	Abhi Engineering	Stock
BH	Blues Health	Stock
CM	County Municipality	Bond
DU	Downtown Utility	Bond
EM	Emmitt Machines	Stock

共同基金 (Inception Date, ID, Name, Mix)

ACF:			
1/1/2005	BG	Big Growth	(500 AE Stocks, 300 EM Stocks)
1/1/2006	SG	Steady Growth	(300 AE Stocks, 300 DU Bonds)
TCA:			
1/1/2005	LF	Tiger Fund	(1000 EM Stocks, 1000 BH Stocks)
1/1/2006	OF	Owl Fund	(1000 CU Bonds, 1000 DU Bonds)
ALB:			
1/1/2005	JU	Jupiter	(2000 EM Stock, 1000 DU Bonds)
1/1/2006	SA	Saturn	(1000 EM Stock, 2000 DU Bonds)

小案例2 Funky Bizz

- 为第3章小案例2创建的关系模式所描述的Funky Bizz Operations数据库编写CREATE TABLE语句创建关系表。
- 使用INSERT INTO语句向Funky Bizz Operations数据库的每张表中插入至少2条至多10条的记录。

小案例3 Snooty Fashions

为第3章小案例3创建的关系模式所描述的Snooty Fashions Operations数据库编写CREATE TABLE语句创建关系表。

使用INSERT INTO语句向Snooty Fashions Operations数据库的每张表中插入至少2条至多10条的记录。

小案例4 Signum Libri

- 为第3章小案例4创建的关系模式所描述的Signum Libri Operations数据库编写CREATE TABLE语句创建关系表。
- 使用INSERT INTO语句向Signum Libri Operations数据库的每张表中插入至少2条至多10条的记录。

小案例5 ExoProtect

- 为第3章小案例5创建的关系模式所描述的ExoProtect Employees' Computers数据库编写CREATE TABLE语句创建关系表。
- 使用INSERT INTO语句向ExoProtect Employees' Computers数据库的每张表中插入至少2条至多10条的记录。

小案例6 Jones Dozers

- 为第3章小案例6创建的关系模式所描述的Jones Dozers Sales and Rentals数据库编写CREATE TABLE语句创建关系表。
- 使用INSERT INTO语句向Jones Dozers Sales and Rentals数据库的每张表中插入至少2条至多10条的记录。

小案例7 Midtown Memorial

- 为第3章小案例7创建的关系模式所描述的Midtown Memorial Patients Drug Dispensal数据库编写CREATE TABLE语句创建关系表。
- 使用INSERT INTO语句向Midtown Memorial Patients Drug Dispensal数据库的每张表中插入至少2条至多10条的记录。

第 6 章

Database Systems: Introduction to Databases and Data Warehouses

数据库的实现与使用

6.1 引言

前面的章节介绍了如何使用 SQL 语言来创建数据库并与之进行交互，本章将会讨论有关数据库实现与使用的附加问题，尤其关注与业务设计者和使用者最为相关的问题。（而物理方面的设计问题涉及硬件和软件的配置，比如数据库在硬件驱动上的物理配置，这些则超出了本书的范围。）本章覆盖的主题包括：

- 实现删除、更新[⊖]操作参照完整性约束的选项。
- 实现用户自定义约束。
- 索引。
- 数据库前端。
- 数据质量问题。

6.2 参照完整性约束：实现删除和更新操作

参照完整性约束规定了带外码的表和被外码参照的带主码的表之间的关系。在第 3 章中定义术语如下：

参照完整性约束

在包含外码的关系中，每一行的外码取值要么对应其参照关系中的主码取值，要么为空。

考察图 6-1 所示的一个简单例子。

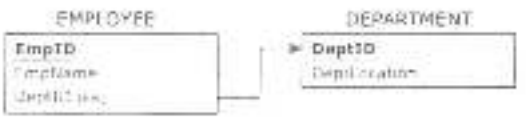


图 6-1 两个关系和一个参照完整性约束

在这个例子中，关系 EMPLOYEE 中外码 DeptID 参照关系 DEPARTMENT 中主码 DeptID。

图 6-1a 展示的场景中，关系 EMPLOYEE 和 DEPARTMENT 之中的值都遵循参照完整性约束，关系 EMPLOYEE 的外码 DeptID 中每一个值都匹配被参照关系 DEPARTMENT 中主码的值。关系 EMPLOYEE 中的第一条和第三条记录的外码 DeptID 值为 1，匹配关系 DEPARTMENT 中的第一条记录的主码值。关系 EMPLOYEE 中的第二条和第四条记录的外

[177]

⊖ 与第 4 章所提到的一样，术语“更新”在实际中通常作为修改的同义词使用（比操作改变了关系中记录的值），而不是包含插入、删除、修改的更一般性的术语。在本章中，“更新”仅作为修改操作的同义词。

码 DeptID 值为 2，匹配关系 DEPARTMENT 中的第二条记录的主码值。

EMPLOYEE

EmpID	EmpName	DeptID
1234	Becky	1
2345	Molly	2
3456	Rob	1
1324	Ted	2

DEPARTMENT

DeptID	DeptLocation
1	Suite A
2	Suite B
3	Suite C

图 6-1a 参照完整性约束：一个例子

参照完整性约束将会影响到通过联系关联到一起的所有关系对于记录的删除与更新（修改）操作。主码侧对记录的删除和更新会影响外码侧的记录，而外码侧对记录的删除和更新不会影响主码侧的记录。例如，在本章中将会看到，关系 DEPARTMENT 中对记录的删除和更新将会影响关系 EMPLOYEE 中的记录，而关系 EMPLOYEE 中对记录的删除和更新将不会影响关系 DEPARTMENT 中的记录。

6.2.1 删除选项

目前的 DBMS 涵盖了多种选项以实现关于删除和更新操作的参照完整性约束。本节将会通过图 6-1a 所示的例子来演示这些选项。

首先讨论实现删除操作参照完整性的多种选项。当试图删除被参照关系中的主码值时，这些选项将决定包含外码的关系中的记录将会发生什么变化。

1. 限制删除

如果记录的主码值被一个外码值所参照，**限制删除**（delete restrict）选项将不允许该记录被删除。

例如，在图 6-1a 中，参照完整性约束连接关系 DEPARTMENT 中主码 DeptID 列和关系 EMPLOYEE 中外码 DeptID 列，考虑对其实施限制删除选项的情况。在这种情况下，关系 DEPARTMENT 中显示的前两条记录（1，Suite A）和（2，Suite B）不能删除，但是第三条记录（3，Suite C）可以删除。任何试图删除关系 DEPARTMNET 中前两条记录的操作都会被 DBMS 阻止，同时会提供错误信息报告，提示用户此删除操作不被允许（非法）。然而，删除关系 DEPARTMENT 中的第三条记录（3，Suite C）是合法的，因为在表 EMPLOYEE 中没有记录的外码 DeptID 列值为 3。参见图 6-2。 [178]

EMPLOYEE

EmpID	EmpName	DeptID
1234	Becky	1
2345	Molly	2
3456	Rob	1
1324	Ted	2

DEPARTMENT

DeptID	DeptLocation
1	Suite A
2	Suite B
3	Suite C

不能删除

可以删除

图 6-2 参照完整性约束：一个限制删除选项的例子

2. 级联删除

如果记录的主码值被一个外码值所参照，**级联删除**（delete cascade）选项允许该记录被删除。但是，那些外码值参照了被删除记录主码值的记录也会被删除。换句话说，删除操作

从包含了被参照主码的表级联到外码所在的表。

例如，在图6-1a中，参照完整性约束连接关系DEPARTMENT中主码DeptID列和关系EMPLOYEE中外码DeptID列，考虑对其实施级联删除选项的情况。这里假设删除关系DEPARTMENT中显示的第一条记录（1，Suite A），此删除操作将会自动导致关系EMPLOYEE中显示的第一条和第三条记录，即（1234，Becky，1）和（3456，Rob，1）也被删除。图6-3显示的是此删除操作的结果。

[179]

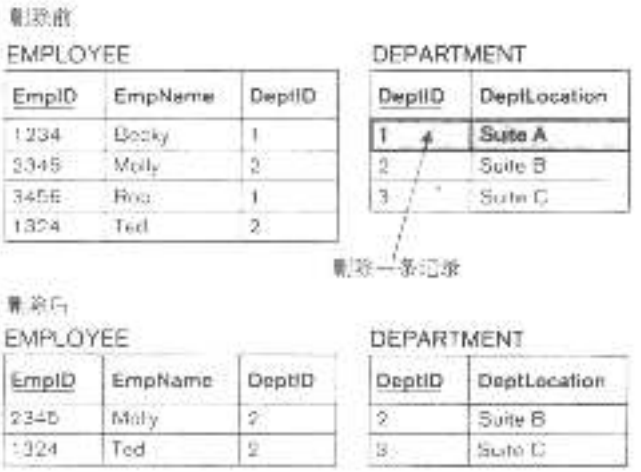


图6-3　参照完整性约束：一个级联删除选项的例子

注意，如果有一个额外的关系表，其外码值参照了EMPLOYEE表的EmpID列，则级联删除也会涉及这张表。例如，这个额外关系表中外码参照了EmpID为1234或3456的所有列都会被删除。

同时需要注意，删除DEPARTMENT表中的记录（3，Suite C）将不会级联到EMPLOYEE表，因为表EMPLOYEE中没有外码DeptID值为3。

3. 删除设置为空

如果记录的主码值被另一关系中记录的外码值所参照，**删除设置为空**（delete set-to-null）选项允许该记录被删除。同时，那些外码值参照了被删除记录主码值的记录，其外码值将会被设置为空。

例如，在图6-1a中，参照完整性约束连接关系DEPARTMENT中主码DeptID列和关系EMPLOYEE中外码DeptID列，考虑对其实施删除设置为空选项的情况。这里假设删除关系DEPARTMENT中的第一条记录（1，Suite A），此删除操作会自动导致关系EMPLOYEE中的第一条和第三条记录，即（1234，Becky，1）和（3456，Rob，1）的外码DeptID值被删除。图6-4显示的是此删除操作的结果。

4. 删除设置为默认值

如果记录的主码值被另一关系中记录的外码值所参照，**删除设置为默认值**（delete set-to-default）选项允许该记录被删除。同时，那些参照了被删除记录主码值的记录，其外码值将会被设置为预先定义的默认值。

例如，在图6-1a中，参照完整性约束连接关系DEPARTMENT中主码DeptID列和关系EMPLOYEE中外码DeptID列，考虑对其实施删除设置为默认值选项的情况。这里假设一

个部门即宣解散，因此，将会删除关系 DEPARTMENT 中该部门的记录，被删除部门的所有雇员将会被移到部门3。反之，关系 EMPLOYEE 中外码 DeptID 列的值预先设置的默认值为3。删除关系 DEPARTMENT 中显示的第一条记录（1，Suite A），此删除操作会自动导致关系 EMPLOYEE 中显示的第一条和第三条记录，即（1234，Becky，1）和（3456，Rob，1）的外码 DeptID 列值被设置为3。图6-5显示的是此删除操作的结果。 180

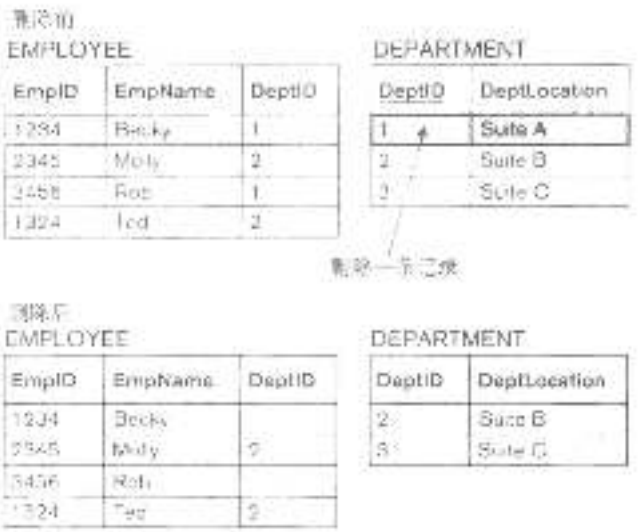


图6-4 参照完整性约束：一个删除设置为空选项的例子

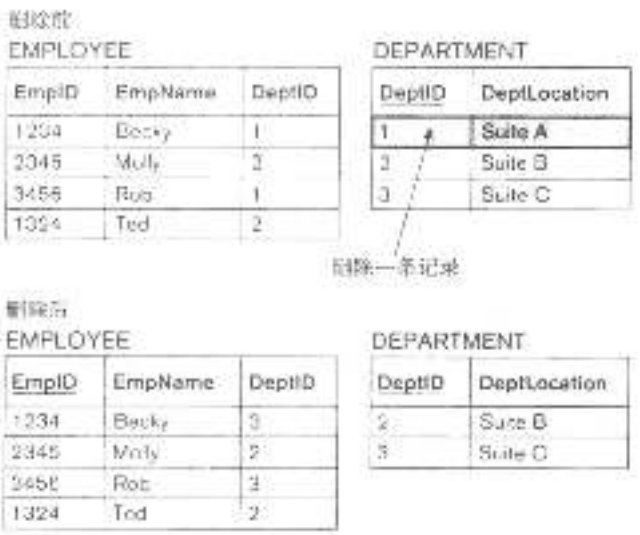


图6-5 参照完整性约束：一个删除设置为默认值选项的例子

注意，无论将何种删除选项（限制、级联、设置为空、设置为默认值）运用于参照完整性约束，如图6-1a所示，表 EMPLOYEE 中的任意记录都可以自由删除，同时不会对表 DEPARTMENT 造成任何影响。表 EMPLOYE 中的记录可以自由删除的原因是没有其他关系参照它（将其主码作为外码）。

6.2.2　更新选项

接下来讨论实现更新操作参照完整性的多种选项。当试图更新（改变）被参照关系中主码的值时，这些选项决定了包含外码的关系中的记录将会发生什么变化。更新选项和之前讨论的删除选项相类似。和删除选项的情况一样，更新选项讨论的是参照完整性关系中主码侧的记录更新操作带来的影响，而不是外码侧的记录更新操作带来的影响。

1. 限制更新

如果记录的主码值被一个外码值参照，**限制更新**（update restrict）选项不允许该记录主码值被修改。

例如，在图 6-1a 中，参照完整性约束连接关系 DEPARTMENT 中主码 DeptID 列和关系 EMPLOYEE 中外码 DeptID 列，考虑对其实施限制更新选项的情况。在这种情况下，关系 DEPARTMENT 中显示的前两条记录（1，Suite A）和（2，Suite B）的主码值不能被修改。任何试图修改关系 DEPARTMNET 中前两条记录的操作都会被 DBMS 阻止，同时会提供错误信息报告，提示用户此更新操作不被允许（非法）。然而，更新关系 DEPARTMENT 中的第三条记录（3，Suite C）是合法的，因为在表 EMPLOYEE 中没有记录的外码 DeptID 列值为 3。具体操作参见图 6-6。

181

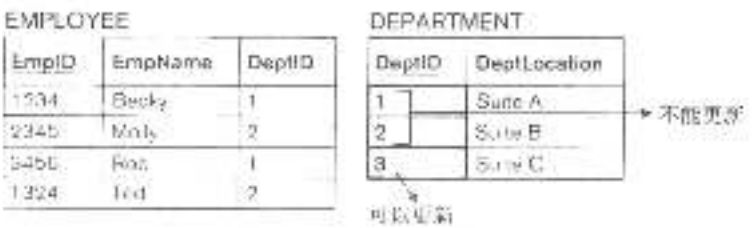


图 6-6　参照完整性约束：一个限制更新选项的例子

2. 级联更新

如果记录的主码值被一个外码值参照，**级联更新**（update cascade）选项允许该记录的主码值被修改。同时，那些参照被修改记录主码值的记录的外码值也会被修改为新的值。换句话说，更新操作从包含被参照主码的表级联到外码所在的表。

例如，在图 6-1a 中，参照完整性约束连接关系 DEPARTMENT 中主码 DeptID 列和关系 EMPLOYEE 中外码 DeptID 列，考虑对其实施级联更新选项的情况。这里假设关系 DEPARTMENT 中显示的第一条记录（1，Suite A）的主码值由 1 更改为 8。此更新操作将会自动导致关系 EMPLOYEE 中显示的第一条和第三条记录，即（1234，Becky，1）和（3456，Rob，1）的外码 DeptID 列值也由 1 更改为 8。图 6-7 显示的是此更新操作的结果。

182

3. 更新设置为空

如果记录的主码值被另一关系的外码值参照，**更新设置为空**（update set-to-null）选项允许该记录的主码值被修改。因此，那些外码值参照被更新记录的主码值的记录，其外码值将会被设置为空。

例如，在图 6-1a 中，参照完整性约束连接关系 DEPARTMENT 中主码 DeptID 列和关系 EMPLOYEE 中外码 DeptID 列，考虑对其实施更新设置为空选项的情况。这里假设关系 DEPARTMENT 中显示的第一条记录（1，Suite A）的主码值由 1 更改为 8。此更新操作将会自动导致关系 EMPLOYEE 中显示的第一条和第三条记录，即（1234，Becky，1）和（3456，

Rob，1）的外码 DeptID 列的值被删除。图 6-8 显示的是此更新操作的结果。

更新前

EMPLOYEE

EmpID	EmpName	DeptID
1234	Becky	1
2345	Molly	2
3456	Rob	1
1324	Ted	2

DEPARTMENT

DeptID	DeptLocation
1	Suite A
2	Suite B
3	Suite C

将记录的键主1更新为8

更新后

EMPLOYEE

EmpID	EmpName	DeptID
1234	Becky	6
2345	Molly	2
3456	Rob	6
1324	Ted	2

DEPARTMENT

DeptID	DeptLocation
6	Suite A
2	Suite B
3	Suite C

图 6-7 参照完整性约束：一个级联更新选项的例子

更新前

EMPLOYEE

EmpID	EmpName	DeptID
1234	Becky	1
2345	Molly	2
3456	Rob	1
1324	Ted	2

DEPARTMENT

DeptID	DeptLocation
1	Suite A
2	Suite B
3	Suite C

将记录的键1更新为8

更新后

EMPLOYEE

EmpID	EmpName	DeptID
1234	Becky	
2345	Molly	2
3456	Rob	
1324	Ted	2

DEPARTMENT

DeptID	DeptLocation
8	Suite A
2	Suite B
3	Suite C

图 6-8 参照完整性约束：一个更新设置为空选项的例子

4. 更新设置为默认值

如果记录的主码值被另一关系的外码值参照，**更新设置为默认值**（update set-to-default）选项允许该记录的主码值被修改。同时，那些参照被更新记录的主码值的记录，其外码值将会被设置为预先定义的默认值。

例如，在图 6-1a 中，参照完整性约束连接关系 DEPARTMENT 中主码 DeptID 列和关系 EMPLOYEE 中外码 DeptID 列，考虑对其实施更新设置为预先设置的默认值的情况。这里假设一个部门的 DeptID 值发生修改，这个部门的所有雇员自动移到部门 DeptID 值为 3 的部门。因此，关系 EMPLOYEE 中外码 DeptID 列的值预先设置的默认值为 3。此更新操作

将会自动导致关系 EMPLOYEE 中显示的第一条和第三条记录，即（1234，Becky，1）和（3456，Rob，1）的外码 DeptID 列的值被更改为 3。图 6-9 显示的是此更新操作的结果。

更新前

EMPLOYEE

EmpID	EmpName	DeptID
1234	Becky	1
2345	Molly	2
3456	Rob	1
1324	Ted	2

DEPARTMENT

DeptID	DeptLocation
1　4	Suite A
2	Suite B
3	Suite C

将记录的值由 1 更新为 3

更新后

EMPLOYEE

EmpID	EmpName	DeptID
1234	Becky	3
2345	Molly	2
3456	Rob	3
1324	Ted	2

DEPARTMENT

DeptID	DeptLocation
1	Suite A
2	Suite B
3	Suite C

图 6-9　参照完整性约束：一个更新设置为默认值选项的例子

注意，无论将何种更新选项（限制、级联、设置为空、设置为默认值）运用于参照完整性约束，表 EMPLOYEE 中的任意记录都可以自由更新为不违反参照完整性（或者其他关系数据库约束）的任何值，同时不会对表 DEPARTMENT 造成任何影响。在本例中，表 EMPLOYEE 中 DeptID 列的值只能更改为 1、2 或 3，任何其他值均会违反参照完整性约束。

6.2.3　实现删除和更新选项

下面的 SQL 代码包含了实现表 EMPLOYEE 中外码 DeptID 和表 DEPARTMENT 中主码 DeptID 之间参照完整性约束的声明。

```
CREATE TABLE employee
(       empid               CHAR(4),
        empname             CHAR(20),
        deptid              CHAR(2),
        PRIMARY KEY         (empid),
        FOREIGN KEY         (deptid) REFERENCES department)
```

对删除和更新操作而言，默认的 SQL 选项是有限制的。换句话说，对于表 EMPLOYEE 中外码 DeptID 和表 DEPARTMENT 中主码 DeptID 之间的参照完整性约束，以上代码实现的是限制删除和限制更新选项。

当然，SQL 允许指定其他删除和更新选项[⊖]。下面给出指定一个非限制的删除选项的实例。比如，对于表 EMPLOYEE 中外码 DeptID 和表 DEPARTMENT 中主码 DeptID 之间的参照完整性约束来说，以下 SQL 代码实现的是级联删除选项。

```
CREATE TABLE employee
(       empid               CHAR(4),
        empname             CHAR(20),
```

⊖ 并非每一个 DBMS 都涵盖了本章中提到的所有删除和更新选项，甚至不同的 DBMS 未实现不同的删除和更新选项。实现删除和更新选项的语义和方法在不同 DBMS 中各不相同，这里给出的例子说明了这一情况。

```
        deptid              CHAR(2),
        PRIMARY KEY         (empid),
        FOREIGN KEY         (deptid) REFERENCES department
                                     ON DELETE CASCADE);
```

接下来给出的例子是指定一个非限制的更新选项。对于表 EMPLOYEE 中外码 DeptID 和表 DEPARTMENT 中主码 DeptID 之间的参照完整性约束来说，以下 SQL 代码实现更新设置为空选项。

```
CREATE TABLE employee
(       empid               CHAR(4),
        ergname             CHAR(20),
        deptid              CHAR(2),
        PRIMARY KEY         (empid),
        FOREIGN KEY         (deptid) REFERENCES department
                                     ON UPDATE SET NULL);
```

[184]

最后，给出同时指定非限制的删除和更新选项的例子。对于表 EMPLOYEE 中外码 DeptID 和表 DEPARTMENT 中主码 DeptID 之间的参照完整性约束来说，以下 SQL 代码实现的是级联删除和更新设置为空选项。

```
CREATE TABLE employee
(       empid               CHAR(4),
        ergname             CHAR(20),
        deptid              CHAR(2),
        PRIMARY KEY         (empid),
        FOREIGN KEY         (deptid) REFERENCES department
                                     ON DELETE CASCADE
                                     ON UPDATE SET NULL);
```

作为以上 SQL 例子的补充，这里用一幅图展示 MS Access 是如何实现删除和更新选项的。图 6-10 展示的即是 MS Access 允许设计者设置删除和更新选项的特性。对于基于 SQL 的系统来说，默认的删除和更新都是设置为限制的。但是，正如图 6-10 所示，用户可以选择级联的删除或更新选项，还可同时选择级联的删除和更新选项。

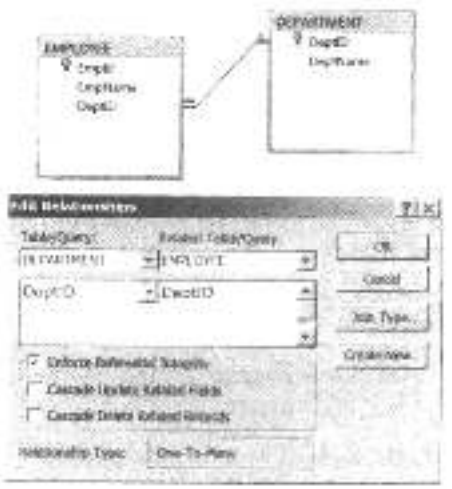


图 6-10　MS Access 中更新和删除选项的例子

6.3　实现用户自定义约束

在第 3 章定义了术语“用户自定义约束”，同时给出了一些示例。与删除和更新选项一

样，不同的 DBMS 包实现用户自定义约束也是不同的。本节将通过一个示例展示用户自定义约束的 SQL 实现机制。

6.3.1　CHECK 子句

在 SQL 中实现用户自定义约束的一种方法是 CHECK 子句。作为示例，考察图 6-11 中所示的例子。

图 6-11　一个带有用户自定义约束的关系

在这个例子中，用户自定义约束为指定关系 EMPLOYEE 中 Salary 列的值在 50 000 ~ 200 000 之间。

以下代码为本例中关系 EMPLOYEE 的 SQL CREATE TABLE 声明语句：

```
CREATE TABLE employee
(empid        CHAR(4),
salary        NUMBER(6) CHECK (salary >= 50000 AND salary <= 200000)
PRIMARY KEY   (empid(1))
```

185

在这种情况下，CHECK 子句执行的是用户自定义约束，不允许加入薪水不在指定区间的雇员。例如，考察以下八条声明语句：

```
INSERT INTO employee VALUES ('1234', 75000);
INSERT INTO employee VALUES ('2345', 50000);
INSERT INTO employee VALUES ('3456', 55000);
INSERT INTO employee VALUES ('1324', 70000);
INSERT INTO employee VALUES ('3876', 270000);
INSERT INTO employee VALUES ('1010', 30000);
```

前四条语句可以顺利执行，而后两条语句则被拒绝执行，因为其违反了用户自定义约束。执行以上语句之后关系 EMPLOYEE 的结果如图 6-12 所示。

EMPLOYEE

EmpID	Salary
1234	75000
2345	50000
3456	55000
1324	70000

图 6-12　四个插入操作被接受，两个被拒绝

正如上例所示，CHECK 子句可以用来指定一个关系表中某一特定列的约束。以下例子则演示了如何利用 CHECK 子句来指定多列的约束。考察图 6-13 所示的例子。

在此例中，一条业务规则为学生的毕业年份不能早于入学年份。

以下代码为创建关系 STUDENT 的 SQL CREATE TABLE 声明语句：

图 6-13 另一个带有用户自定义约束的关系

```
CREATE TABLE student
(studentid        CHAR(4),
yearenrolled      INT,
yearofgraduation  INT,
PRIMARY KEY (studentid),
CHECK (yearenrolled <= yearofgraduation));
```

在这种情况下，CHECK 子句执行的是用户自定义约束，不允许插入毕业年份早于入学年份的学生。例如，考察以下四条声明语句：

```
INSERT INTO student VALUES ('1111', 2013, 2016);
INSERT INTO student VALUES ('2222', 2013, 2016);
INSERT INTO student VALUES ('3333', 2013, 2017);
INSERT INTO student VALUES ('4444', 2013, 2012);
```

前三条语句可以顺利执行，而最后一条语句则被拒绝执行，因为后者违反了用户自定义约束。执行以上语句之后关系 STUDENT 的结果如图 6-14 所示。

STUDENT

StudentId	YearEnrolled	YearOfGraduation
1111	2013	2016
2222	2013	2016
3333	2013	2017

图 6-14 三个插入操作被接受，一个被拒绝

6.3.2 实现用户自定义约束的其他机制

作为 CHECK 机制的补充，还有许多其他实现用户自定义约束的更为复杂的方法。这些方法包括断言和触发器（在 6.7 节有演示）、使用特定的数据库编程语言编程、结合 SQL 以及非 SQL 语句处理数据库中的数据（如 PL/SQL），在常规的编程语言代码内嵌入 SQL 语句（如 C++ 或 Java）。在多数情况下，用户自定义逻辑不是作为数据库的一部分来实现，而是作为数据库前端应用的一部分。

为了更好地使用数据库，完整地实现用户自定义约束显得十分重要。正如上文提到的，有多种方法可实现用户自定义约束。从商业角度考虑，用户自定义约束的执行力比选择何种方法来实现该约束更重要。通常，当存在多种可选择的方法时，会从技术角度做出选择，比如选择执行速度最快的方法。

6.4 索引

在对一个有大量记录的关系进行数据查询和检索时，索引（index）是一种加速机制。多数关系型 DBMS 软件工具都允许定义索引。以下例子是对索引原则在概念上的简单说明。

图 6-15 展示的是关系 CUSTOMER。在实际情况下，这样一张表中会有大量的记录，为便于观察，这里只显示了 13 条记录。这张关系表是基于 CustID 排序之后的结果，因此，剩

下的 CustName 列以及 Zip 列是乱序的。

基于未排序列的值查找某一特定记录的过程称为**线性/顺序查找**（linear/sequential search）。该搜索顺序地逐一匹配每一个元素的值，直到找到所要查找的内容。比如，假设需要通过姓名来查找名为 Steve 的客户。因为 CustName 列未排序，所以必须使用线性查找，从第一条记录开始逐一检查 CustName 列的值，直到找到 Steve。在这个例子中，前 8 条记录的 CustName 列的值都会被检查，直到检查第 9 条记录（1008，Steve，60222）为止，如图 6-16 所示。

187

CUSTOMER

CustID	CustName	Zip
1000	Zach	60111
1001	Ana	60333
1002	Matt	60222
1003	Lara	60555
1004	Pam	60444
1005	Sally	60555
1006	Bob	60333
1007	Adam	60555
1008	Steve	60222
1009	Pam	60333
1010	Ema	60111
1011	Peter	60666
1012	Fiona	60444

图 6-15　CUSTOMER 关系

基于已排序的值查找某一特定记录时，利用非线性查找可以很快得到结果。**二分查找**（binary search）即为非线性查找方法明显快于线性查找方法的一个例子。二分查找利用排序之后的列表，通过查找列表中间的值，将其分为规模相同的两部分（强调二分）。如果查找内容的值大于列表中间的值，则列表的前半部分将会从查找空间中消去。相应地，如果查找内容的值小于列表中间的值，则列表的后半部分将会从查找空间中消去。不管是哪种方式，经过这样简单的一步，查找空间就减少了一半。在余下的列表空间内重复执行以上步骤，直到找到所要查找的内容。

188

CUSTOMER

CustID	CustName	Zip	
1000	Zach	60111	→ 步骤 1（不是 Steve）
1001	Ana	60333	→ 步骤 2（不是 Steve）
1002	Matt	60222	→ 步骤 3（不是 Steve）
1003	Lara	60555	→ 步骤 4（不是 Steve）
1004	Pam	60444	→ 步骤 5（不是 Steve）
1005	Sally	60555	→ 步骤 6（不是 Steve）
1006	Bob	60333	→ 步骤 7（不是 Steve）
1007	Adam	60555	→ 步骤 8（不是 Steve）
1008	Steve	60222	→ 步骤 9　找到客户 Steve
1009	Pam	60333	
1010	Ema	60111	
1011	Peter	60666	
1012	Fiona	60444	

图 6-16　一个线性查找的例子

例如，假设需要查找 CustID 号为 1008 的客户记录。如图 6-17 所示，二分查找首先查找表中间位置的记录，其 CustID 号为 1006。因为 1006 小于 1008，接下来的查找将会忽略表的前半部分记录，同时考察表余下部分中间位置的记录的 CustID 值。其值为 1010，大于 1008，因此接下来的查找将会忽略余下部分的后半部分。目前为止，查找空间已经进一步减少。接下来的查找将会继续考察余下部分中间位置记录的 CustID 值。其值即为 1008，查找结束，找到记录（1008，Steve，60222）。

注意，二分查找法比线性查找法需要的步骤少得多。再如，考察从按姓氏字母排序的电话簿中查找给定姓氏的电话号码。因其已经排好序，故可以采用二分查找法。从这本电话簿

的中间页码开始，忽略不包含待查找姓氏的部分。多次重复以上缩减查找空间的步骤之后，就可找到待查找的电话号码。现在假设有一个特定电话号码，需要查找该电话号码对应的用户名。因为电话号码并未排序，所以必须使用线性查找法。可以想象，在找到这个特定号码之前需要查找大量的电话号码是一件多么漫长而痛苦的事。

CUSTOMER

CustID	CustName	Zip
1000	Zach	60111
1001	Ana	60333
1002	Matt	60222
1003	Lara	60555
1004	Pam	60444
1005	Sally	60555
1006	Bob	60333
1007	Adam	60555
1008	Steve	60222
1009	Pam	60333
1010	Ema	60111
1011	Peter	60666
1012	Fiona	60444

步骤 1　消去在这之上的记录（由于 CustID 值小于 1008）

步骤 3　找到客户 1008

步骤 2　消去在这之下的记录（由于 CustID 值大于 1008）

图 6-17　一个二分查找的例子

当关系表基于某一列的值排序之后，其他列必然为乱序。正如上文提到的，查找未排序列的值比已排序列的值要慢得多。基于索引的查找可以加快对未排序列的查找。图 6-18 展示的是为关系 CUSTOMER 中的 CustName 建立索引的例子。在数据库系统中，索引可以通过只包含两列的附加表来实现。其中一列包含索引列排好序的值，另一列包含指向原始表中对应记录的指针。在本例中，索引表 CUSTNAME_INDEX 中一列为已排好序的 CustName 值，另一列为指向 CUSTOMER 表对应记录的指针。

189

CUSTNAME_INDEX

CustName	CustID
Adam	1007
Ana	1001
Bob	1006
Ema	1010
Fiona	1012
Lara	1003
Matt	1002
Pam	1004
Pam	1009
Peter	1011
Sally	1005
Steve	1008
Zach	1000

CUSTOMER

CustID	CustName	Zip
1000	Zach	60111
1001	Ana	60333
1002	Matt	60222
1003	Lara	60555
1004	Pam	60444
1005	Sally	60555
1006	Bob	60333
1007	Adam	60555
1008	Steve	60222
1009	Pam	60333
1010	Ema	60111
1011	Peter	60666
1012	Fiona	60444

图 6-18　一个索引的例子

现在来看看索引是如何加快查找速度的。图 6-16 所示的例子展示了如何查找 CustName

为 Steve 的记录，可见使用线性查找需要 9 个步骤。而图 6-19 展示的是对同一查找在表 CUSTNAME_INDEX 中使用二分法的版本。因为 CustName 在索引表中是排好序的，故可以使用二分查找法。在这个例子中，在索引表中查找到 CustName 为 Steve 需要 3 步。一旦在索引表中找到 CustName 为 Steve 的记录，其指针的值就对应指向 CUSTOMER 表中的相应记录，从而完成查找。

图 6-19　一个使用索引加快查找速度的例子

现在来考察另一个例子，同样用于演示如何使用索引提高查找效率。假设用户需要从 CUSTOMER 表中找到有多少条记录的 CustName 为 Pam（再次提醒，关系 CUSTOMER 中 CustName 列未排序）。

首先查看图 6-20 所示的情况，它展示的是不使用索引的情况下，在关系 CUSTOMER（见图 6-15）中查找是如何执行的。

190

图 6-20　另一个二分查找的例子

因为 CustName 列并未排序，所以不得不线性地查找关系 CUSTOMER 中的每一条

记录。用计数器 counter 统计客户名字 Pam 出现的次数，初始值设为 0。核查每一记录时，若其 CustName 不为 Pam，则计数器值不变；若 CustName 为 Pam，则计数器值加 1。CustName 为 Pam 可以出现在任何一条记录中，因此必须核查关系中的每一条记录，故这种查找方法的步骤数为关系表中记录的总数。在本例中，需要 13 步来找到结果，因为有 13 条记录。

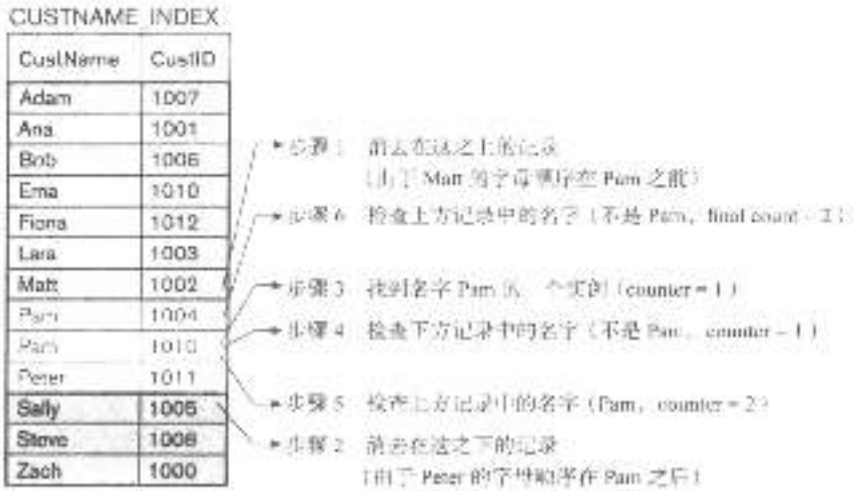
现在来看看如何利用索引加快查找速度。图 6-21 展示了使用图 6-19 所示 CUSTNAME_INDEX 索引表来实现本次查找的版本。因为索引表中 CustName 已排好序，所以可以使用二分查找法来找到特定的客户姓名。在这种情况下，从索引表中找到 CustName 为 Pam 需要 3 步。虽然需要额外的步骤来统计 CustName 为 Pam 的所有记录并核实是否所有 CustName 为 Pam 的记录都被统计到，但总计所需的步骤数仍然少于未使用索引的步骤数。待查找文件越庞大，查找步骤数上的优势会越显著，因为线性查找方法需要遍历表中的每一条记录，而使用索引的查找仅需要搜索部分记录。

191

CUSTNAME_INDEX

CustName	CustID
Adam	1007
Ana	1001
Bob	1006
Ema	1010
Fiona	1012
Lara	1003
Matt	1002
Pam	1004
Pam	1010
Peter	1011
Sally	1005
Steve	1008
Zach	1000

步骤 1　消去在这之上的记录
（由于 Matt 的字母顺序在 Pam 之前）



步骤 6　检查上方记录中的名字（不是 Pam，final count = 2）

步骤 3　找到名字 Pam 的一个实例（counter = 1）

步骤 4　检查下方记录中的名字（不是 Pam，counter = 1）

步骤 5　检查上方记录中的名字（Pam，counter = 2）

步骤 2　消去在这之下的记录
（由于 Peter 的字母顺序在 Pam 之后）

图 6-21　另一个使用索引加快查找速度的例子

以上所有例子都是为了说明一种简单的机制可以用来加快查找和检索的速度。但需注意，这些例子只是对索引原则的一种简单说明。为便于演示，所有例子中都是将关系的主码作为索引的指针。在实际情况下，索引中指针的值可以是记录在硬盘上的物理地址或者其他记录物理地址的相关机制。同样，除了示例中的索引列简单排序以及二分查找法之外，还可使用其他不同逻辑与技术方案的 DBMS 工具来实现索引，如集群索引、散列索引、B+ 树等。这些方案都超出了本书的范畴，对本书读者来说，重要的是知道所有方案都有一个共同目标：通过建立索引来加快查找和检索的速度。需要特别注意的是，无论 DBMS 系统使用哪种逻辑和技术方案来实现索引，当索引被创建之后，对于终端用户来说，它都能以无缝的形式提高查找和检索数据的效率。

为关系表中的某一列创建索引是很简单的，涉及书写 CREATE INDEX 的 SQL 语句如下：

```
CREATE INDEX custname_index ON customer(custname);
```

一旦这条语句被执行，就意味着在关系 CUSTOMER 中涉及 CustName 列的查找和检索将会更快。

值得注意的是，索引在加快查找和检索的同时也带来了更多的成本。一旦使用索引，数据库就需要添加额外的空间，因为需要为每一索引创建额外的表。同时，插入、删除、修改原始表中的记录将会消耗更多的时间，因为这同时伴随着对索引表的插入、修改、删除以及重排序。因此，对数据库添加索引需要同时考虑其优势和成本。典型的情况是，对大型表中常被许多用户查找或检索的列建立索引。如果已有的索引无用，可以使用 DROP INDEX 语句来删除索引，如下所示：

```
DROP INDEX custname_index
```

这条语句删除了索引，该索引将不再可用。

6.5　数据库前端

在多数情况下，一些打算使用数据库的用户（通常是大多数用户）都缺乏时间或经验来直接使用数据库中的数据。因此，期望每一个用户都能自己编写查询或其他语句来实现数据库中的数据操作是不现实的。大多数的数据库终端用户访问数据库是通过前端应用来实现的。**数据库前端**（database front-end）应用可以包含多种不同的组件，本节将介绍几种常见的数据库前端组件。

表格（form）是一种数据库前端组件，它使得终端用户无需经过训练便可直接输入、检索数据。表格提供了进入数据库表或查询的接口。图 6-22 所示为向图 6-15 所示关系 CUSTOMER 输入数据的表格形式接口。

图 6-22　一个用于向关系 CUSTOMER 输入数据的表格的例子

每当用户向表格中输入一个值时，SQL 语句

```
INSERT INTO customer VALUES (...)
```

192 会在用户行为之后被执行。这当然比每次插入一条记录时都人工书写 INSERT INTO 语句方便得多。表格也可以用来删除和修改记录。当终端用户使用表格删除或修改记录时（如图 6-23 所示），相应的 SQL 语句（DELETE 或 UPDATE）就会在用户行为之后执行。

比如，用户选中图 6-23 所示表格中的最后一条记录，并在键盘上按下 Delete 键，如下语句就会被生成和执行：

```
DELETE FROM customer
WHERE custid = '1012';
```

如果用户将图 6-23 所示的第一条记录的客户名从 Zach 修改为 Zachary，并在键盘上按下回车键，如下语句就会被生成和执行：

```
UPDATE customer
SET custname = 'Zachary'
WHERE custid = '1006';
```

客户

删除记录：选中记录，按下键盘的 Delete 键

更改记录：更改任意一行的任意值，按下键盘的回车键

Customer ID	Customer Name	Zip
1000	Zach	60111
1001	Ana	60333
1002	Matt	60222
1003	Lara	60555
1004	Pam	60444
1005	Sally	60555
1006	Bob	60333
1007	Adam	60555
1008	Steve	60222
1009	Pam	60333
1010	Emma	60111
1011	Peter	60666
1012	Fiona	60444

图 6-23　一个用于更新关系 CUSTOMER 中数据的表格的例子

表格还可以用来实现用户与数据库中其他类型数据的交互。图 6-24 所示即为表格用于查找和检索的例子。

客户搜索表

通过 Customer ID、Customer Name 和 Zip Code 搜索客户

输入搜索值，然后单击 SEARCH 按钮

Customer ID:

Customer Name:

Zip Code:

SEARCH

图 6-24　一个用于在关系 CUSTOMER 中搜索数据的表格的例子

当向 Customer Name 编辑框中输入数据并单击 SEARCH 按钮时，就会为用户生成一条 SQL 语句。假设用户向 Zip Code 编辑框中输入 60555，并单击 SEARCH 按钮，以下查询将被生成和执行：

193

```
SELECT    *
FROM      CUSTOMER
WHERE     zip = '60555';
```

查询结果将会以表格形式返回，如图 6-25 所示。

搜索结果

Customer ID	Customer Name	Zip
1003	Lara	60555
1005	Sally	60555
1007	Adam	60555

图 6-25　使用图 6-24 中的表格进行搜索返回的结果表

注意，以上的简单例子只是对表格基础功能的简要演示。数据库前端应用可以包含复杂的表格形式，能够结合多种功能同时从多张表中检索数据。

报表（report）也是一种数据库前端应用的组件，其作用是以某种格式化的形式呈现数据以及对来自一张或多张表的数据进行计算。表格的目的是便于终端用户和

数据库之间进行交互，如允许用户进行更新与检索操作，而报表检索的数据则是格式化的、经专业排版的，以便于浏览，可在显示器上显示或者打印出来。

图 6-26 所示是关系 CUSTOMER 的一个报表，其中记录根据 Zip 进行分组和汇总。

报表：根据 Zip 进行分组和汇总

Zip	Customer ID	Customer Name
60111	1000	Zach
	1010	Emma
Zip 为 60111 的客户总数：2		
60222	1005	Matt
	1008	Steve
Zip 为 60222 的客户总数：2		
60333	1001	Ana
	1006	Bob
	1009	Pam
Zip 为 60333 的客户总数：3		
60444	1004	Pam
	1012	Fiona
Zip 为 60444 的客户总数：2		
60555	1003	Lana
	1005	Sally
	1007	Adam
Zip 为 60555 的客户总数：3		
60666	1011	Peter
Zip 为 60666 的客户总数：1		
所有 Zip 中的客户总数：13		

图 6-26　一个来自关系 CUSTOMER 的报表的例子

和表格的情况一样，数据库前端应用也可以包含复杂的报表。这些报表可以结合多种不同类型的计算以及从多张表中检索数据。

作为表格和报表的补充，数据库前端应用还包含多种其他组件，如菜单、图表、图片、示意图。如何选择和使用不同的组件将根据终端用户的需求而定。

[194]

一个数据库可以有多个前端应用用于不同的目的或者用户。这些前端应用可以分别独立地访问，如图 6-27 所示，并允许用户通过接口根据自己的需求来选择应用。

图 6-27　一个数据库前端应用程序集合界面的例子

例如，如果用户选择了“客户管理”选项，就会出现如图 6-28 所示的应用。

现在，用户可以利用多种不同的前端组件访问客户相关数据。比如，如果用户选择“搜索客户”选项，就会出现图 6-24 所示的表格；如果用户选择“按 Zip 列出所有客户”，则会出现图 6-26 所示的报表。

图 6-28　一个数据库前端应用程序界面的例子

一个实际应用中可以包含大量精心设计的表格、图表或其他前端组件，并以复杂的方式组合。例如，在初始菜单上选择某一项后连接到另一个包含多个选择列表的菜单。

现有多数 DBMS 包都带有用于创建数据库前端接口的功能或者附加软件。当然，一些特定的第三方软件工具也可用来创建实现数据库数据访问的前端组件。另外，还可以利用编程语言编写代码来实现前端组件。

195

数据库前端接口也可以是基于 Web 的，此时，终端用户需要使用 Web 浏览器来连接数据库。许多 Web 网站都有连接数据库的接口。在这种情况下，Web 网页的内容是从数据库中提取数据并生成的，如图 6-29 所示为 Web 零售商的某一特定网页。

图 6-29　一个网页的例子

图 6-30 展示的是相应的关系表，用来存储显示在网页上的婴儿玩具信息。换句话说，图 6-29 所示网页的中间界面即为图 6-30 所示关系数据库的前端。

数据库关系表的集合如图 6-30 所示，它提供了一种结构化且非冗余的方式来存储呈现在整个网站上的所有数据的方法。例如，对一个 ProductID 值为 1 的产品来说，其 ProductName

为 Rainbow Baby Violet 的产品可以出现在零售网站的多个网页。将产品名从 Rainbow Baby Violet 修改为 Rainbow Baby Purple 时，只需要执行一次更新操作，数据库中新的 ProductName 版本将会覆盖旧的版本，其结果是，更新的版本自动传播到所有网页，并使用新的 ProductName。

ProductID	ProductName	Price	Picture
1	Rainbow Baby Violet	$16.00	Picture1.gif
2	Rainbow Baby Red	$16.00	Picture2.gif
3	Rainbow Baby Green	$16.00	Picture3.gif
4	Rainbow Baby Light Green	$16.00	Picture4.gif
5	Rainbow Baby Blue	$16.00	Picture5.gif
6	Ikbab Rainbow Baby Blue	$16.00	Picture6.gif
7	Alba Grabbing Rattle	$20.00	Picture7.gif
8	Musical Alba Owl	$22.00	Picture8.gif
9	Ikbab Musical Giraffe	$46.00	Picture9.gif
		...	
...	...	...	...

图 6-30　存储图 6-29 中网站内容的数据库关系表

6.6　数据质量问题

在使用数据的过程中，需要始终重视**数据质量**（data quality）问题。如果设计的数据库所代表的内容能够准确、清晰地反映真实世界，则其数据即为高质量的。有多种不同的定义数据质量的数据属性，最基本的数据质量特征包括：

- 准确性（accuracy）。
- 唯一性（uniqueness）。
- 完整性（completeness）。
- 一致性（consistency）。
- 及时性（timeliness）。
- 统一性（conformity）。

196
-
197

数据的**准确性**关系到该数据刻画真实实例的正确程度。比如，数据库中客户的姓名拼写错误，这样的数据即存在准确性数据质量问题，纠正拼写可以解决这类问题。

数据的**唯一性**要求每一真实实例在数据收集时只能被描述一次。唯一性数据质量问题有时也称为**数据重复**（data duplication）。例如，如果关系表中有两条记录包含的数据为同一客户的信息，就会出现唯一性数据质量问题。解决唯一性问题首先需识别出这两条相同的记录，然后删除其中一条或者将二者合并为一条记录。

数据的**完整性**关系到在进行数据收集时需求数据被描述的完整程度。例如，如果包含病人数据的关系表中，病人的体重信息是缺失的，这种疏忽即为数据完整性问题。每位病人都有体重，没有体重的病人记录即为不完整的。另一方面，保险单号的缺失则不一定是数据完整性问题。如果病人没有参保，保险单号的缺失是真实情况的正确反映；然而，若病人参保了，保险单号缺失即为完整性数据质量问题。

数据的**一致性**是指数据恰当地符合及匹配其他数据的程度。例如，某机构可能有两种不同的关于公司整体利润信息的来源。如果这两种来源指向不同的利润金额，则这两个金额互

不一致。

在数据集中，一致性数据质量问题是由其他数据质量问题引起的，如准确性或唯一性数据质量问题。例如，考察某警察局雇用了 50 名男警务人员和 50 名女警务人员的场景。根据人员情况，该警察局定制了男女制服各 50 套。假设 100 名警务人员的所有相关信息（包括性别）都输入到一张关系表中，同时假设其中有一条记录的性别信息输入错误，将一名男警务人员的性别信息错误地设为“女”。这条特定的记录就存在准确性数据质量问题。使用标准的公式统计男女警务人员总数的结果将会得到 49 名男性，51 名女性。但是，定制制服的信息显示为男女警务人员各 50 名。这两组数据即为不一致的。注意，通过将错误输入“女”修改为“男”，纠正准确性错误之后，也就自动解决了一致性问题。纠正后使用标准的公式统计男女警务人员总数的结果为男女各 50 名，这就和定制制服时的数字保持一致了。

当正确的数据出现在错误的位置时，也会出现一致性数据质量问题。例如，一张包含住院病人信息的表中，可能有一条记录真实、正确地反映了某门诊病人的信息（如姓名、出生日期、性别等）。这条记录的内容在准确性的范围内，因为其准确地反映了真实的病人信息。但是，这条记录错误地出现在住院病人信息表中，这可能会导致医院床位数据的不一致。将该记录从住院病人信息表移至门诊病人信息表即可解决这个问题。

数据的及时性关系到数据与其所表达的真实世界在时间维度上的相符程度。通常，及时性关系到数据的“新鲜”程度。如果数据是来自用户当前工作的及时数据，则不会存在及时性问题。反之，如果数据未得到及时有效的更新，则会存在及时性问题。例如，考察下面这种情况，某货运公司运出了一批货物，但是这条记录直到 3 天以后才输入到 DELIVERED_CARGO_SHIPMENTS 表中。在这 3 天的时间内，DELIVERED_CARGO_SHIPMENTS 表即存在及时性数据质量问题。解决这一问题的办法是要求数据输入人员在货物运出之后及时地将相关信息输入到数据库表中。

数据的统一性是指数据符合某种特定格式的程度。当某数据实例不符合预先定义的格式时，统一性数据质量问题即会出现。例如，假设银行事务中涉及美元交易事务存储时，需按规定在金额的前面加上标识符 $。如果某银行处理美元交易事务存储时将标识符 $ 加在了金额的后面，则这些记录即存在统一性数据质量问题。解决这一问题的方法是将所有标识符 $ 放在金额的前面。

198

为进一步演示准确性、唯一性、完整性、一致性、及时性、统一性问题，考察如下例子。假设 Albritco 公司需记录其经理的数据。Albritco 公司现有 5 名经理：2 名销售经理（Lilly 和 Carlos）、3 名财务经理（Emma、Robert 和 Vijay）。最近新招聘 1 名销售经理（Scarlett）。这一做法符合该公司在战略上的部署，即销售经理和财务经理的数目相等。在招聘了 1 名新的销售经理之后，如图 6-31 所示的消息被发送给了董事会主席。

致：Albritco 董事会主席

主题：实现表格规划目标（机密）

我们很荣幸地报告，销售经理和财务经理数目相同的战略目标已经达成，见下表：

经理人数一致：目标达成	
销售经理数目	3
财务经理数目	3

图 6-31　Albritco 公司经理人数的报表消息

这条消息是由招聘主管起草的，他知道之前的负责人统计的数据为 2 名销售经理、3 名

财务经理，以及该公司刚刚新招了1名销售经理。

图6-32展示的是数据库中存储该公司经理数据信息的关系表。为便于讨论，该关系表中设置了多种数据质量问题。

图6-32　一个有数据质量问题的数据库关系表

第一条记录中，经理Lilly的生日信息缺失。虽然经理Lilly有生日信息，但并没有输入到该记录中。这种疏忽即为完整性问题的例子。

199

第二条记录中，经理Emma的名字错误拼写为Emu，导致准确性问题。

第三条、第四条记录同时对应同一个人。在数据输入的过程中出现了错误，将同一个人的信息录入了两次，并赋予了两个不同的ManagerID值：一个为其真实姓名，另一个为昵称。这两条记录指向同一个人，导致唯一性问题。

第五条记录中经理Carlos生日信息的录入格式导致统一性问题。所有生日信息的录入格式为：/月份名称/2位日期/4位年份/。但是经理Carlos的生日信息录入格式为：/1或2位日期/1或2位月份/2位年份/。

新招的经理Scarlett已经在Albritco公司上班，但是她的信息并没有录入到数据库的关系表内，因为信息录入人员正在度假。直到经理Scarlett的信息录入之前，该关系表中的经理数据都存在及时性问题。

为阐明一致性问题，考虑图6-33所示的报表，它是基于图6-32中的关系的。

图6-33　一个基于图6-32中关系的报表

报表中的计算是基于正确的数学公式来统计Title为销售经理以及Title为财务经理的

记录总数。对销售经理的统计将会出现少统计的现象，因为存在及时性问题，经理 Scarlett 的信息未录入。而对财务经理的统计将会出现多统计的现象，因为存在唯一性问题，记录 Robert 的信息被录入了两次。这次统计数据（2 名销售经理以及 4 名财务经理）与图 6-31 所示的信息（3 名销售经理以及 3 名财务经理）之间存在一致性问题。

本例所描述的数据质量问题可以通过预防措施和纠正措施来规避。

排除数据质量问题的**数据质量预防措施**（preventive data quality action）包括：

- 为数据录入人员提供适当的专业培训和激励。
- 规定适当的数据录入时间，如规定及时输入数据以避免及时性问题。
- 制定用户自定义约束和机制。如实现某种数据库用户自定义规则来限制日期数据的插入，要求插入日期在合适的范围内或实现了某种特定的机制（如输入掩码），以确保正确的格式。

如果数据质量问题已经存在，则可以执行以下**数据质量校正措施**（corrective data quality action）：

- 检测和确认数据库有数据质量问题。
- 使用特定的软件和标准查找数据库来识别和校正特定的数据质量问题。

209

对于图 6-32 所示的情况来说，Albriteo 公司可以通过执行以下操作来解决数据质量问题。

- DateOfBirth 可以声明为强制列（如在 CREATE TABLE 语句中使用 NOT NULL 子句），这就可以规避第一条记录所示的完整性问题。
- 在 Name 列输入数据时，检查和核对标准姓名的数据库（如美国常用的 10 000 个人名），标记不常见的输入以便进一步查证。该操作将会在第二条记录输入 Emu 时标记记录列表以便进一步查证。查证方式包括搜索此人的其他已有信息，进而确认该人的真实名字为 Emma。这些操作可以提示我们将关系中的姓名从 Emu 改为 Emma，从而解决了准确性问题。
- E-mail 列在 CREATE TABLE 语句中应声明为唯一列，因为其实际上为候选码。这就可以防止第四条记录的生成，从而保留第三条记录作为有问题的经理信息的唯一信息记录，也因此规避了唯一性问题。
- 生日信息的输入格式可以作为一种机制统一指定为 /月份名称 /2 位日期/4 位年份/，以此规避如第五条记录所示的数据统一性问题。
- 制定并强制执行管理规则，要求每一位新经理的信息在其被招聘时就必须录入到数据库中，这就可以消除经理 Scarlett 已经在公司上班但其信息仍未录入数据库所造成的及时性问题。
- 对于图 6-33 所示的报表显示的一致性问题，可以通过上述操作校正表 6-32 所示的唯一性和及时性问题来解决，不需要额外的特定操作来解决这些特殊的一致性问题。

图 6-34 所示为图 6-32 所示数据质量问题解决之后的表。

图 6-35 所示为基于图 6-34 所示数据生成的报表。现在一致性问题已解决，这个报表显示有 3 名销售经理和 3 名财务经理，与图 6-31 保持一致且数据正确。

高质量的数据对于数据库的合理使用至关重要。低质量数据造成的后果可能非常糟糕，因为低质量的数据可能导致错误决策。错误的顾客统计、不准确的金融资产信息、缺失的病人诊断信息等现象，不但没有一点好处，而且必然会带来负面影响。因此，建立规则以确保数据质量十分必要。

ManagerID	Name	Title	DateOfBirth	E-mail
111	Lily	Sales Manager	May 21, 1971	lily@albritco.com
222	Emma	Financial Manager	June 01, 1966	emma@albritco.com
333	Robert	Financial Manager	December 25, 1973	robert@albritco.com
444	Carlos	Sales Manager	April 13, 1968	carlos@albritco.com
555	Vijay	Financial Manager	October 04, 1981	vijay@albritco.com
666	Scarlett	Sales Manager	July 17, 1984	scarlett@albritco.com

图 6-34　图 6-32 中的数据库关系表在解决数据异常问题后的结果

报表：经理

Title	Manager ID	Manager Name
销售经理	111	Lily
	444	Carlos
	666	Scarlett
销售经理总数：3		
财务经理	222	Emma
	333	Robert
	555	Vijay
财务经理总数：3		

图 6-35　一个基于图 6-34 中关系的报表

本章已经从数据库设计者和用户的角度介绍了关于数据库实现和使用的大部分基本问题。下一节将介绍关于断言和触发器的附加信息。

[201]

6.7　问题说明：断言和触发器

断言（assertion）是用于指定用户自定义约束的一种机制。为使下演示和描述这种机制，我们将使用图 6-36 所示的例子。

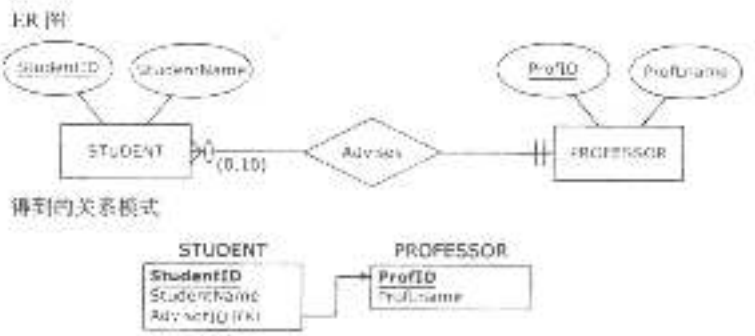


图 6-36　每个教授最多只能指导 10 个学生

在此例中，用户自定义约束规定每位教授最多可以指导 10 个学生。以下 CREATE ASSERTION SQL 语句指定了该规则：

```
CREATE ASSERTION profsadvisingupto10students
CHECK (
(SELECT MAX( totaladvised )
```

```
FROM (SELECT count(*) AS totaladvised
    FROM student
    GROUP BY advisorid) < 11);
```

CHECK 语句核查每位教授所指导的学生数是否小于 11。如果没有导师指导的学生数超过 10，则 CREATE ASSERTION 语句将会返回 TRUE；若有导师指导的学生数超过 10，则返回 FALSE。

尽管 CREATE ASSERTION 语句只是 SQL 标准中的一部分，但是许多 RDBMS 包并不能通过 CREATE ASSERTION 语句实现断言（然而，本节选择给出一个 CREATE ASSERTION 语句例子，是因为这刚好演示了实现用户自定义约束的过程）。大多数 RDBMS 包实现断言功能是通过不同且更复杂的机制（如触发器）来完成的。 202

触发器（trigger）是一种规则，当关系中删除、插入、修改（更新）一条记录时执行该规则。为演示触发器规则，下面将考察如何通过两个触发器来实现断言 ProfsAdvising-UpTo10students（在 MySQL 中书写）：

```
CREATE TRIGGER studentinsertTrigger
      BEFORE INSERT ON student
  FOR EACH ROW
  BEGIN
    DECLARE totaladvised INT DEFAULT 0;
    SELECT COUNT(*) INTO totaladvised
    FROM student
    WHERE advisorid = NEW.advisorid;
    IF (totaladvised >= 10) THEN
       SET NEW.advisorid = NULL;
    END IF;
  END;

CREATE TRIGGER studentupdatetrigger
      BEFORE UPDATE ON student
  FOR EACH ROW
  BEGIN
    DECLARE totaladvised INT DEFAULT 0;
    SELECT COUNT(*) INTO totaladvised
    FROM student
    WHERE advisorid = NEW.advisorid;
    IF (totaladvised >= 10) THEN
      SET NEW.Advisorid = NULL;
    END IF;
  END;
```

StudentInsertTrigger 使用了一个变量（触发器的局部变量）TotalAdvised，其初始值设为 0。该触发器在每一条 INSERT INTO Student 语句执行之前执行。NEW.AdvisorID 是指待插入学生的 AdvisorID 值。例如，假设向关系 STUDENT 发出以下 INSERT INTO 语句：

```
INSERT INTO student VALUES ('111', 'Mark', 'P11');
```

新插入记录的 AdvisorID 值为 P11，因此在这种情况下 NEW.AdvisorID 也为 P11。在每一次插入之前，SELECT 子句统计在关系 STUDENT 中有多少条记录和待插入记录的 AdvisorID 值是相同的（本例为 P11）。接下来这个统计值将会赋给变量 TotalAdvised。触发器中 IF 子句紧接着 SELECT 子句，检查这个统计值是否达到 10。如果达到，插入操作将会导致新的统计值超过 10，这是不允许的。触发器将会使 INSERT INTO 语句如下执行：

```
INSERT INTO student VALUES ('111', 'Mark', NULL);
```

这会导致这名新插入的学生没有指导老师。然而，在本例中，由于 STUDENT 实体强制性参

与到 Advises 联系中，AdvisorID 列被设置为 NOT NULL，所以整个 INSERT INTO 语句将会被 RDBMS 拒绝执行。然而，如果统计计数未超过 10，则此插入语句将会按照以下形式执行：

```
INSERT INTO student VALUES ('1111', 'Mark', 'Pill');
```

StudentUpdateTrigger 与 StudentInsertTrigger 的工作方式类似，只是它是由更新（修改）操作而不是插入操作触发执行的。在本例中，不需要 StudentDeleteTrigger，因为本例要求的是被指导的学生数不能超过某一上限，而删除操作不会增加任何导师所指导的学生数目。

203

关键术语

accuracy（准确性），197
assertion（断言），202
binary search（二分查找），188
CHECK，185
completeness（完整性），198
conformity（统一性），198
consistency（一致性），198
corrective data quality action（数据质量校正措施），200
database front-end（数据库前端），192
data duplication（数据重复），198
data quality（数据质量），197
delete cascade（级联删除），179
delete restrict（限制删除），178
delete set-to-default（删除设置为默认值），180
delete set-to-null（删除设置为空），180
form（表格），192
index（索引），187
linear/sequential search（线性/顺序查找），187
preventive data quality action（数据质量预防措施），200
report（报表），194
timeliness（及时性），198
trigger（触发器），203
uniqueness（唯一性），198
update cascade（级联更新），182
update restrict（限制更新），181
update set-to-default（更新设置为默认值），183
update set-to-null（更新设置为空），182

复习题

Q6.1　列出用于实现参照完整性约束的删除和更新操作。
Q6.2　索引的作用是什么？
Q6.3　数据库前端应用的作用是什么？
Q6.4　表格的作用是什么？
Q6.5　报表的作用是什么？
Q6.6　简述 Web 网站作为数据库接口的优势。
Q6.7　定义数据准确性。
Q6.8　定义数据唯一性。
Q6.9　定义数据完整性。
Q6.10　定义数据一致性。
Q6.11　定义数据及时性。
Q6.12　定义数据统一性。
Q6.13　给出一个数据质量预防措施的例子。
Q6.14　给出一个数据质量校正措施的例子。

练习

E6.1 考察图6-37所示的两张表。关系SALES_REP的外码列TerID参照关系TERRITORY的主码列TerID。

SALES_REP

SRID	SRName	TerID
1	Joe	E
2	Sue	E
3	Meg	C
4	Bob	S
5	Ike	N
6	Pat	N
7	Lee	N
8	Joe	E

TERRITORY

TerID	TerName
E	East
W	West
S	South
N	North
C	Central

图6-37 关系SALES_REP和TERRITORY

E6.1a 如果DBMS在关系SALES_REP和TERRITORY的参照完整性约束之上实施了限制删除操作，用户删除关系TERRITORY中第四条记录（N，North）之后，写出关系表SALES_REP和TERRITORY之中的记录。

E6.1b 如果DBMS在关系SALES_REP和TERRITORY的参照完整性约束之上实施了限制删除操作，用户删除关系TERRITORY中第二条记录（W，West）之后，写出关系表SALES_REP和TERRITORY之中的记录。

E6.1c 如果DBMS在关系SALES_REP和TERRITORY的参照完整性约束之上实施了限制删除操作，用户删除关系SALES_REP中第一条记录（1，Joe，E）之后，写出关系表SALES_REP和TERRITORY之中的记录。

E6.1d 如果DBMS在关系SALES_REP和TERRITORY的参照完整性约束之上实施了级联删除操作，用户删除关系TERRITORY中第四条记录（N，North）之后，写出关系表SALES_REP和TERRITORY之中的记录。 204

E6.1e 如果DBMS在关系SALES_REP和TERRITORY的参照完整性约束之上实施了级联删除操作，用户删除关系TERRITORY中第二条记录（W，West）之后，写出关系表SALES_REP和TERRITORY之中的记录。

E6.1f 如果DBMS在关系SALES_REP和TERRITORY的参照完整性约束之上实施了级联删除操作，用户删除关系SALES_REP中第一条记录（1，Joe，E）之后，写出关系表SALES_REP和TERRITORY之中的记录。

E6.1g 如果DBMS在关系SALES_REP和TERRITORY的参照完整性约束之上实施了删除设置为空操作，用户删除关系TERRITORY中第四条记录（N，North）之后，写出关系表SALES_REP和TERRITORY之中的记录。

E6.1h 如果DBMS在关系SALES_REP和TERRITORY的参照完整性约束之上实施了删除设置为空操作，用户删除关系TERRITORY中第二条记录（W，West）之后，写出关系表SALES_REP和TERRITORY之中的记录。

E6.1i 如果DBMS在关系SALES_REP和TERRITORY的参照完整性约束之上实施了删除设置为空操作，用户删除关系SALES_REP中第一条记录（1，Joe，E）之后，写出关系表SALES_REP和TERRITORY之中的记录。

E6.1j 如果DBMS在关系SALES_REP和TERRITORY的参照完整性约束之上实施了删除设置为空操作，用户删除关系TERRITORY中第四条记录（N，North）之后，写出关系表SALES_REP和TERRITORY之中的记录。假设关系SALES_REP中对DeptID的默认值设置为"E"。

E6.1k　如果 DBMS 在关系 SALES_REP 和 TERRITORY 的参照完整性约束之上实施了删除设置为默认值操作，用户删除关系 TERRITORY 中第二条记录（W，West）之后，写出关系表 SALES_REP 和 TERRITORY 之中的记录。假设关系 SALES_REP 中对 DeptID 的默认值设置为“E”。

E6.1l　如果 DBMS 在关系 SALES_REP 和 TERRITORY 的参照完整性约束之上实施了删除设置为默认值操作，用户删除关系 SALES_REP 中第一条记录（1，Joe，E）之后，写出关系表 SALES_REP 和 TERRITORY 之中的记录。假设关系 SALES_REP 中对 DeptID 的默认值设置为“E”。

E6.2　考察图 6-37 所示的两张表：

E6.2a　如果 DBMS 在关系 SALES_REP 和 TERRITORY 的参照完整性约束之上实施了限制更新操作，用户将关系 TERRITORY 中第四条记录（N，North）修改为（NO，North）之后，写出关系表 SALES_REP 和 TERRITORY 之中的记录。

E6.2b　如果 DBMS 在关系 SALES_REP 和 TERRITORY 的参照完整性约束之上实施了限制更新操作，用户将关系 TERRITORY 中第二条记录（W，West）修改为（WE，West）之后，写出关系表 SALES_REP 和 TERRITORY 之中的记录。

E6.2c　如果 DBMS 在关系 SALES_REP 和 TERRITORY 的参照完整性约束之上实施了限制更新操作，用户将关系 SALES_REP 中第一条记录（1，Joe，E）修改为（1，Joe，C）之后，写出关系表 SALES_REP 和 TERRITORY 之中的记录。

E6.2d　如果 DBMS 在关系 SALES_REP 和 TERRITORY 的参照完整性约束之上实施了级联更新操作，用户将关系 TERRITORY 中第四条记录（N，North）修改为（NO，North）之后，写出关系表 SALES_REP 和 TERRITORY 之中的记录。

E6.2e　如果 DBMS 在关系 SALES_REP 和 TERRITORY 的参照完整性约束之上实施了级联更新操作，用户将关系 TERRITORY 中第二条记录（W，West）修改为（WE，West）之后，写出关系表 SALES_REP 和 TERRITORY 之中的记录。

E6.2f　如果 DBMS 在关系 SALES_REP 和 TERRITORY 的参照完整性约束之上实施了级联更新操作，用户将关系 SALES_REP 中第一条记录（1，Joe，E）修改为（1，Joe，C）之后，写出关系表 SALES_REP 和 TERRITORY 之中的记录。

E6.2g　如果 DBMS 在关系 SALES_REP 和 TERRITORY 的参照完整性约束之上实施了更新设置为空操作，用户将关系 TERRITORY 中第四条记录（N，North）修改为（NO，North）之后，写出关系表 SALES_REP 和 TERRITORY 之中的记录。

E6.2h　如果 DBMS 在关系 SALES_REP 和 TERRITORY 的参照完整性约束之上实施了更新设置为空操作，用户将关系 TERRITORY 中第二条记录（W，West）修改为（WE，West）之后，写出关系表 SALES_REP 和 TERRITORY 之中的记录。

E6.2i　如果 DBMS 在关系 SALES_REP 和 TERRITORY 的参照完整性约束之上实施了更新设置为空操作，用户将关系 SALES_REP 中第一条记录（1，Joe，E）修改为（1，Joe，C）之后，写出关系表 SALES_REP 和 TERRITORY 之中的记录。

E6.2j　如果 DBMS 在关系 SALES_REP 和 TERRITORY 的参照完整性约束之上实施了更新设置为默认值操作，用户将关系 TERRITORY 中第四条记录（N，North）修改为（NO，North）之后，写出关系表 SALES_REP 和 TERRITORY 之中的记录。假设关系 SALES_REP 中对 DeptID 的默认值设置为“E”。

E6.2k　如果 DBMS 在关系 SALES_REP 和 TERRITORY 的参照完整性约束之上实施了更新设置为默认值操作，用户将关系 TERRITORY 中第二条记录（W，West）修改为（WE，West）之后，写出关系表 SALES_REP 和 TERRITORY 之中的记录。假设关系 SALES_REP 中对 DeptID 的默认值设置为“E”。

205

E6.2l　如果 DBMS 在关系 SALES_REP 和 TERRITORY 的参照完整性约束之上实施了更新设置为默认值操作，用户将关系 SALES_REP 中第一条记录（1，Joe，E）修改为（1，Joe，C）之后，

写出关系表 SALES_REP 和 TERRITORY 之中的记录。假设关系 SALES_REP 中对 DeptID 的默认值设置为“E”。

E6.3　使用图 6-15 演示查找客户名 Peter 的线性查找步骤。

E6.4　使用图 6-15 演示查找客户 1001 的二分查找步骤。

E6.5　为第 3 章中图 3-33 所示的表 PRODUCT 建立 PRODUCT_INDEX 索引表，使用 ProductID 作为索引指针。

E6.6　用户使用图 6-23 所示的表格将第五个客户的 zip code 从 60444 修改为 60999，写出其 SQL 指令。

E6.7　用户使用图 6-24 所示的表格在 Customer Name 编辑框内输入 Pam 并单击 Search 按钮，写出其 SQL 指令，同时写出执行之后的结果。

E6.8　考察如下场景：

在 Albus 国际学院，每位入学的学生都缴纳 $1000 的押金。Albus 国际学院共有 10 名新生，每位学生都缴纳了押金，这些信息都反映在图 6-38 所示的表 A 中。表 B 显示的是该学院入学学生的个人信息。假设如下：

表 A：入学学生存款

Deposit Date	Deposit Amount
2/12/2013	$1,000
2/12/2013	$1,000
2/13/2013	$1,000
2/13/2013	$1,000
2/14/2013	$1,000
2/14/2013	$1,000
2/15/2013	$1,000
2/15/2013	$1,000
2/16/2013	$1,000
2/16/2013	$1,000
学生总数（有存款）	10
存款总数	$10,000

表 B：入学学生信息

Student ID	Student Lname	Student Fname	Passport Number	Date of Deposit	Student Home Town	Student Home Country
200602	McCoy	Mark	US2000001	02/12/2013	Detroit	USA
200603	Suzuki	Akiko	JA7000001	02/12/2013	Tokyo	
200604	Jones	Mary	US2000002	02/13/2013	Chicago	USA
200605	Lalime	Patrick	CA3000001	February 13th, 2013	Montreal	Canada
200606	Van Basten	Robin	NE4000001	02/14/2013	Amsterdam	Netherlands
200607	Van Basten	Rob	NE4000001	02/14/2013	Amsterdam	Netherlands
200608	Vrabec	Alen	CR5000001	02/14/2013	Zagreb	Croatia
200609	Popov	Stilian	BU6000001	02/15/2013	Plovdiv	Bulgaria
200610	De Beers	Inge	NE4000002	02/15/2013	Amsterdam	Neverland
入学学生总数	9					

图 6-38　Albus 国际学院学生表

- 表 A 正确地反映了每位学生所缴纳的押金。
- 表 B 中每位学生的姓氏都拼写正确。
- 表 B 中每位学生的名字（或缩写）都拼写正确。
- 表 B 中押金缴纳的日期格式为 MM/DD/YYYY。
- 表 B 中的 Passport Number 号都是正确的。
- 表 B 中所有学生的家乡都拼写正确。

在图 6-38 所示的表 B 中检测和分类所有实例的数据质量问题（如准确性、唯一性、完整性、一致性、及时性、统一性问题）。

206

第二部分

Database Systems: Introduction to Databases and Data Warehouses

分析型数据库

第 7 章
Database Systems: Introduction to Databases and Data Warehouses

数据仓库概念

7.1 引言

一个典型的组织往往维护并使用着若干操作型数据源。这些操作型数据源包括关系型数据库及其他数据存储系统，用于保存组织机构的日常操作数据，如收银事务、信用卡购买、酒店预订或者学生成绩录入等。

数据仓库作为单独的数据存储创建于组织中，其主要目的是数据分析。一般情况下，同一个事务描述可以同时用作操作型和分析型目的。例如，数据描述“顾客 X 在商店 Z 处购买了产品 Y”，既可以作为诸如存货监管或财务交易记录等业务流程支持信息而存储到操作型数据库中，也可以保存到数据仓库中，结合一段时间累积记录下来的大量类似的事务描述，它们就可以反映出一些重要的趋势，如销售模式或顾客行为等。

将数据仓库的创建转换为单独的分析型数据库有两个主要原因。第一，如果操作型日常任务需要跟分析型查询竞争计算资源时，那么与数据相关的操作型日常任务的性能就会大幅降低。第二，即使性能不构成问题，构建一个可以同时用于操作型和分析型目的的数据库通常都不太可能（第 8 章会详细介绍）。因此，数据仓库是作为一个单独的数据存储而创建的，也是为适合分析查询而设计的。

要区分分析型数据存储系统和操作型数据存储系统，需要相应地区分分析型和操作型信息。本章我们将描述这两种信息的区别，同时还将给出数据仓库系统的主要组件，介绍数据集市与数据仓库的区别，本章末尾将给出创建数据仓库主要过程的概要描述。

7.2 操作型信息与分析型信息

正如第 1 章中所提到的，信息系统所收集和使用的信息通常可以用于两个目的：操作和分析。

回顾术语**操作型信息 / 事务型信息**（operational information/transactional information），它指的是为支持日常操作需要所收集和使用的信息。由于操作型信息源于个人事务，如利用 ATM 存取款或者购买机票，所以操作型信息有时候也称为事务信息。

另一方面，术语**分析型信息**（analytical information）指的是为支持数据分析任务的决策所收集和使用的信息。正如第 1 章所提及的分析型信息的例子，既可以是描述 ATM 使用模式的信息（比如一天中哪些时间段是存取款数量的最高和最低时期），也可以是揭示航空公司销售趋势的信息（比如在美国哪些路线拥有最多或最少的销售量）。重要的是，应该意识到分析型信息是以操作型信息为基础的。例如，分析型信息可以揭示出一天中不同时段的 ATM 使用模式，而这些信息则需要通过结合大量的个人 ATM 存取款事务信息实例来得到。

虽然分析型信息以操作型信息为基础，但是两者截然不同。分析型和操作型信息的差别可以分成三类：数据组成差别，技术差别，功能差别。

7.2.1 数据组成差别

数据组成差别是指构成分析型和操作型信息的数据所具有的特性上的差别。这里我们特别给出操作型和分析型数据的三个差别：数据时间范围，数据细节层次，数据时间表示。

1. 数据时间范围差别

操作型系统的时间范围比分析型系统更短。例如，在许多操作型系统中，数据的时间范围一般为 60 ～ 90 天，90 天以后的数据（通常）会从操作型系统中移除并进行存档。

以电信公司为例。公司的操作型数据库可能只保存 90 天的有用数据，代表客户产生的个人通话记录，并通过相关应用为客户代表提供服务。大部分客户会关心最近三个月的数据，因此当需要查询过去几周或几个月的电话费用时，客户代表使用操作型系统可以非常容易且快速地得到电话记录信息。如果该操作型系统包含了数年的有效个人通话记录，则基本数据库将会更大，因而支撑客户代表的操作型系统在实现传统查询时就会更慢。这可能导致系统对大部分客户查询的回复速度过慢，这也是为什么操作型系统（在本例中）的时间范围为 90 天。如果一个罕见查询需要讨论客户三年前的一条通话记录，此时公司可以通过查找存档数据来满足这个不寻常的请求。当然，这种查询请求处理起来可能很慢，但是比起大幅降低偶然出现的非典型请求的查询速度，快速处理大量出现的典型查询显然更有意义。大部分用于操作型目的的查询请求只在相对较短的时间跨度内查询数据。在操作型系统中保存各时间段内的所有数据可能造成不必要的混杂，继而由于这些多余且巨大的搜索空间，操作型查询会变得更加缓慢。

另一方面，分析型数据存储系统通常需要提供数年的数据趋势和模式分析。因此，分析型数据库比操作型数据库具有更长的时间范围。分析型数据库的数据时间范围常常跨越数年。不同的数据时间范围需求是将操作型和分析型数据进行物理分离的主要原因。时间范围较短的操作型数据库往往作为提供操作型数据使用功能的站点，而具有较长时间范围的分析型数据存储系统则作为提供分析型查询功能的站点。

2. 数据细节层次差别

操作型数据通常反映的是细节数据。换言之，它对每条个人交易事务的细节都进行了记录。这种层次的细节也称为详细细节层次。如电信公司的操作型数据库记录了每一条通话，通常包括主叫号码、被叫号码、通话起始时间以及通话时间。 [208]

汇总数据，如本周内所有客户通话记录的时间总和，可以从细节数据计算得到。这样的汇总数值通常不会存储在操作型数据库中，这是因为这些数值的计算速度几乎和存储后的检索速度一样快。同时，汇总数据会有频繁的改变，因此存储汇总数据的意义不大。例如，每次客户另外拨打一个电话时，客户的每周总通话时间就要相应地改变。

在操作型数据库中，这些汇总数据通常并不存储，而是作为根据公式得到的派生属性。因此，每次客户再拨打一个电话，每周的总通话时间就会自动重新计算并且在应用程序终端进行相应的更新。另一方面，分析型数据库可以同时包含物理存储的细节数据和汇总数据。分析型数据库的汇总数据，如过去 10 年各个星期顾客的通话时间，通常会事先计算好并进行物理存储。鉴于分析型数据库的历史特性，这些汇总值并不发生改变，因而在分析型数据库中物理存储这些汇总值是可行的。例如，过去几年的每周通话时间总长并不会随时间发生改变。如果分析师需要频繁地使用这些汇总数值，存储计算结果就比每次需要时重新计算更具意义。重复计算通常更适合操作型数据库，这是由于其动态特性、相对小的数据存储量以

及相对短的时间范围等特点所决定的。然而对于具有较长时间范围且存储了大量数据的分析型数据库，比起每次需要时重新计算，保存汇总数据能更有效地减少分析查询的时间开销。

3. 数据时间表示差别

操作型数据通常反映现实世界事务的当前状态，而分析型数据既能表示当前状态还能表示过去时刻的快照。例如，对于银行顾客通常会有一个数据表示其支票余额。如果顾客进行了一次取款或者存款操作，支票余额会相应改变，新的当前值则会替代旧的值。

分析型数据存储系统既可以包含顾客的当前余额，也可以包含顾客一段时间之前的余额，也称快照，如在某个时间范围内每个月末的支票余额。这些快照可以一次性计算好并存储起来以备分析师共分列用，而不是在分析师需要的每个时间点上重新计算。

7.2.2　技术差别

技术差别定义为DBMS及应用程序对操作型和分析型数据的处理与访问方式的差别。下文将描述查询数据总量、查询频度、数据更新、数据冗余几个方面的技术差别。

1. 查询数据总量和查询频度差别

比起分析型查询，操作型查询通常处理的数据量要小。另一方面，操作型查询通常比分析型查询出现的频率更高且用户数量更多。通过系统配置来优化处理少量数据的频繁查询操作或处理大量数据的不频繁操作通常是可行的。但要在技术上通过系统配置同时实现两种情况的优化通常是不可能的。这就是数据仓库必须作为物理上分离的数据存储系统的另一个原因。

209

2. 数据更新差别

操作型系统中的数据由用户进行有规律地更新。在操作型数据库中，除了提出检索数据的查询请求，用户往往还要按要例进行插入、修改以及删除数据的操作。而在分析型数据库中，终端用户只能检索数据，更新数据则是不允许的。

3. 数据冗余差别

正如本书前面章节所详细描述的，操作型数据库的主要目标就是减少信息冗余。使冗余最小化的主要原因是为了减小更新异常的可能。然而，分析型数据库并不允许终端用户进行数据更新，缺乏也就不存在数据更新异常的情况。因此，分析型数据库中的减少数据冗余必然不如操作型数据库中重要。

7.2.3　功能差别

功能差别定义为操作型数据和分析型数据使用原理上的不同，接下来我们将描述的两种功能差别与数据读者和数据定位有关。

1. 数据读者差别

操作型数据支撑着商业公司及组织机构的日常操作，这些数据会被大量雇员、顾客以及其他用户用于各种实际战略目的。例如，售货员可以使用操作型信息为购买商品的顾客进行合适的找零，旅行家可以使用操作型信息预订飞往目的地的机票。与操作型数据的广泛使用相比，分析型数据往往只被少量用户用来进行市场决策。例如，零售连锁店的CEO可以使用大量复杂的分析型信息来决定应该关闭哪些亏损的门店或应该在何处开新店。

2. 数据定位差别

这里所讨论的操作型和分析型信息的“定位”一词，意指信息是以何种目的组织起来的。操作型数据库通常是为了支持某个为业务操作及处理过程提供服务的应用而创建的。以操作

型数据库为例，船运公司的订单录入数据库或者牙医办公室的预约管理数据库通过为应用提供数据存储来支持相关操作。因此操作型数据库又称为**面向应用型**（application-oriented）数据库。

相反，分析型数据库是为特定的业务主题领域的分析任务而创建的，如销售、改善、成本、利润等。分析型数据库围绕待分析的主题而建，因此分析型数据库又称为**面向主题型**（subject-oriented）数据库。

下面的这个例子表明了面向应用型与面向主题型数据在组织上的差别。该例给出一个较简单的场景——活力健康俱乐部。健身中心拥有按月交会费的固定会员。会员可以分为两种，一种是 $100 会费的金牌会员，另一种是 $50 会费的普通会员。普通会员可以使用除游泳池以外的全部设备，金牌会员则可以使用包括游泳池在内的所有设备。俱乐部还提供非会员的每日消费形式，非会员顾客支付每天 $10 的费用就可以使用除游泳池以外的全部设备，支付每天 $15 的费用就可以使用包括游泳池在内的所有设备。图 7-1 展示了一个为活力健康俱乐部建立的面向应用型数据集，它可以为该公司的计算机应用提供支持，收银员通过该应用处理缴费操作以及会员和非会员的消费操作。

HEALTH CLUB MEMBER

MemberID	MemberName	MemberGender	MLevelID	DateMembershipPaid
111	Joe	M	A	1/1/2013
222	Sue	F	B	1/1/2013
333	Pam	F	A	1/2/2013
...	...	...	...	...

MEMBERSHIP LEVEL

MLevelID	MLevelType	MLevelFee	MLevelDescription
A	Gold	$100	Includes the Pool Usage
B	Basic	$50	No Pool Usage

DAILY VISIT FROM NONMEMBERS

DVisitTID	DVisitLevelID	DVisitDate	DVisitorGender
11xx22	YP	1/1/2013	M
11xx23	NP	1/2/2013	M
11xx24	YP	1/2/2013	F
...	...	...	...

VISIT LEVEL

DVisitLevelID	DVisitLevelFee	DVisitLevelType
YP	$15	With Pool Usage
NP	$10	Without Pool Usage

图 7-1　支持活力健康俱乐部访问和支付操作的面向应用型数据库

简单起见，我们只展示了 HEALTH CLUB MEMBER 以及 DAILY VISIT FROM NONMENMBERS 两个关系表的前面几行记录。

图 7-2 展示了一个为该俱乐部创建的面向主题型数据集，它可以支持税收主题的分析任务。

REVENUE

RevenueRecordID	Date	GeneratedBy	ClientGender	Pool Use Included in Purchase	Amount
1000	1/1/2013	Member	M	Yes	$100
1001	1/1/2013	Member	F	No	$50
1002	1/1/2013	Nonmember	M	Yes	$15
1003	1/2/2013	Member	F	Yes	$100
1004	1/2/2013	Nonmember	M	No	$10
1005	1/2/2013	Nonmember	F	Yes	$15
				...	...

图 7-2 为进行营业额分析而建立的面向主题型数据库

图 7-1 和图 7-2 的数据集包含的数据完全相同，只是组织方式不同而已。图 7-2 中的 RecordID 列是表的主码，用于区别不同的记录，它在数据库中保存为自动增加的整型数值。图 7-2 中的所有其他数据均来自图 7-1。例如，记录 1000 是将表 HEALTH CLUB MEMBER 首条记录的一部分与表 MEMBERSHIP LEVEL 首条记录的一部分进行组合之后得到的结果。由于表 HEALTH CLUB MEMBER 以及表 MEMBERSHIP LEVEL 都与会员相关，因此列 GeneratedBy 中有了 Member 一项。记录 1002 则是将表 DAILY VISIT FROM NONMEMBERS 的首条记录的一部分与表 VISIT LEVEL 的首条记录组合以后得到的结果。列 GeneratedBy 中的项 Nonmember 则是源于这两个关系表均是非会员相关的事实。

211

图 7-1 中的数据集合已经规范化成了多个关系表，很适合活力健康俱乐部的消费和支付应用。例如，当有新会员加入时，管理会员的工作人员只需要在 HEALTH CLUB MEMBER 表中使用合适的代号（A 或 B）输入一条新记录，这条记录就会自动与 MEMBERSHIP LEVEL 表中的相应记录进行关联。因此，工作人员只需要在 HEALTH CLUB MEMBER 表中添加一条新的记录，不需要再输入任何其他信息，之后便可收取会费（$50 或 $100）。该面向应用型数据库可以有效地支持相关应用的功能。

图 7-2 中的数据集合包含在一个以营业额分析为目的而建立起来的面向主题型关系表中。根据图 7-1 的操作型数据库可分析营业额相关数据的具体类别，图 7-2 的关系表中的列则相应包含了总营业额、日期、顾客类型、顾客性别以及营业额相关的使用类型等。在这张关系表上建立单表查询就可以直接根据日期、顾客类型、顾客性别进行营业额分析。

虽然活力健康俱乐部也可以利用图 7-1 中的数据库进行营业额分析，但是分析的过程一定不如利用图 7-2 中的关系表简单直接。为了在图 7-1 的数据库中得到所有的营业记录，分析人员必须首先将最上方的两个关系表进行连接，然后连接最下方的两个表，最后通过合并两个连接结果将所有营业记录展示出来。

实际上，像活力健康俱乐部这样的公司可以采用其他操作型数据表来描述额外的营业额来源。比如，用来描述俱乐部专卖店购买记录的关系表或描述俱乐部饮料能量吧购买记录的关系表。这时，将图 7-2 所示的所有营业额实例都合并到关系表的面向主题型数据集中，也可以直接用于营业额分析。另一方面，仅仅使用操作型数据集进行分析可能会涉及更多的复杂操作，如建立连接以及在连接之上建立连接。分析人员很快就会发现，他们用于集合必要数据的时间（即数据准备）将会比用于分析的时间更多。

正如上文所描述的，图 7-2 中的面向主题型数据集比图 7-1 中的面向应用型数据集更适合进行收入分析。

同样，图7-1中的面向应用型数据集比较适合支持操作型应用，而图7-2中的面向主题型数据集并不适合这个目的。

7.3 数据仓库定义

现在已经清楚了分析型信息和操作型信息之间的区别，接下来继续考察最基本的分析型数据存储系统：**数据仓库**（data warehouse）。

不管是从实践的角度还是理论的角度，都有许多描述术语“数据仓库”的定义。这些定义之间有差别，但是对通用概念保持一致。下面的这个一般性定义就包含了这些概念：

数据仓库

数据仓库是一种结构化的数据存储形式，其存储的数据具有集成性、面向主题、企业范围跨度、历史性以及时变性的特点。数据仓库的目的是用于对分析型数据进行检索。数据仓库可以保存细节型或汇总型的数据。

接下来我们将继续分析本定义中涉及的几个基本概念。 212

7.3.1 结构化数据存储

数据仓库是一种保存了可分析的有用数据的数据库。任何一个数据库都是一种结构化的数据存储，结构表示其元数据。因此，数据仓库作为数据库的一种，也是一种结构化的数据存储。换句话说，数据仓库并不是大量随机数据的杂乱集合。

7.3.2 集成性

数据仓库的目的是为分析型信息创建一种有序的存储形式，这种存储与已有的操作型数据库在组织上是物理分离的。数据仓库集成了操作型数据库中各种可分析的有用数据，集成就是将多个操作型数据库中的数据转换到一个单一数据仓库中的过程。当然，在这个过程中，并不是把数据从操作型数据库中移除，而是将各个操作型数据库中可分析的有用数据复制到数据仓库中。

7.3.3 面向主题

术语“面向主题”反映了操作型数据库系统与数据仓库在目的上的基本差别。操作型数据库系统是为了支持一些特殊的业务操作而创建的，而数据仓库是为了对这些特殊的业务主题领域进行分析而创建的。

7.3.4 企业范围

数据仓库是一种关于可分析的有用数据的存储形式。术语“企业范围”是指数据仓库可以为包含在其中的信息提供一个机构范围的全局视图。例如，如果数据仓库的主题之一为成本，那么整个机构的操作型数据源中所有有关成本的可分析的有用数据都会包含到该数据仓库中来。

7.3.5 历史性

与操作型信息相比，分析型信息具有更长的时间跨度。给定一个包含分析型信息的数据

仓库，该数据仓库的时间跨度比操作型数据库更长（通常如此）。术语“历史性”是指数据仓库所具有的比操作型数据库更长的时间范围。例如，许多传统操作型数据库的时间范围为60～90天，而对数据仓库来说，包含数年有效数据的情况更为常见。

7.3.6　时变性

术语“时变性”是指数据仓库包含来自长时间范围内不同区间的数据片段或子图快照。有了这些数据片段后，用户便可以为时间范围内各个时间段的数据创建报告。例如，如果分析主题为成本且时间跨度为数年，那么我们便可以进行一年前的一季度成本与两年前的一季度成本的比较分析。

7.3.7　分析型信息的检索

数据仓库正是为了分析型信息的检索而建的，而不是为用户直接输入数据而建的。对数据仓库用户来说，唯一可用的功能就是检索。数据仓库中的数据并不会受到终端用户的修改、插入或删除操作的影响。数据仓库中的新数据周期性地从操作型数据源中导入继而添加到已有数据中，该过程由系统自动完成。那些超过时间跨度的数据由于太陈旧则会自动被系统从数据仓库中净化掉（也可能进行存档或者归纳）。然而，数据仓库中的数据不会受到这些改变的影响，这就是为什么数据仓库中的数据具有非易失、稳定以及只读的特点。

[213]

7.3.8　细节数据和汇总数据

根据不同的目的，数据仓库可能包含细节数据、汇总数据或者两者皆有。细节数据又称为原子数据或事务级数据。例如，将每一条ATM交易事务都进行了单独记录的关系表包含的就是细节数据（原子数据、事务级数据）。另一方面，有些关系表中的记录是根据多个事务级实例数据计算得到的，这些表包含的就是更粗粒度的汇总数据。例如，汇总数据可以表示一台ATM在某个月的取款总量。

那些包含了最细粒度数据的数据仓库最为有用，因为通过这样的数据仓库可以计算得到所有汇总数据，如果需要重复使用则可以保存下来。然而，要将整个时间范围内的所有事务级分析型数据详细地记录下来，则会代价过高。例如，一些组织机构可能没有经济能力来购买能存储和处理巨大数据集的特殊硬件和软件。同时还有另一种情形，即机构自行对某些待分析主题的数据做出这样的决策：汇总型数据是合适的，事务级的细节数据更没有必要。当企业没有能力或者不愿意保存其所需要的最细粒度的细节数据时，那么数据仓库中部分或者所有（依据具体情况而定）待分析主题的数据都只保留到特定的汇总级别。

7.4　数据仓库组件

每个数据仓库系统的核心都有三个主要组件：源系统、提取、转换、加载（ETL）架构以及数据仓库本身。大部分数据仓库系统还有查询部分。本节我们将简要地概要并讨论这些组件。

图7-3给出了一个企业的示例场景，为完成日常操作，用户需要同时使用多个操作型数据存储。

[214]

然而对分析型目的来说，另一种方法就是使用如图

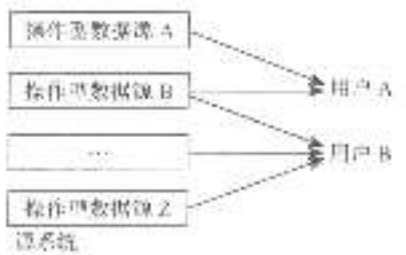


图7-3　一个企业将操作型数据源用于操作型目的的例子

7-4所示的数据仓库。在一系列面向应用的操作型数据源中进行搜寻，因此将会花费大量时间用于数据准备而不是数据分析。

7.4.1 源系统

在数据仓库领域中，源系统（source system）是指那些为数据仓库分析主题提供可分析的有用信息的操作型数据库以及其他操作型数据存储系统（即用于操作型目的的任何数据集）。用作数据仓库源系统的所有操作型数据存储系统都有两个目的：

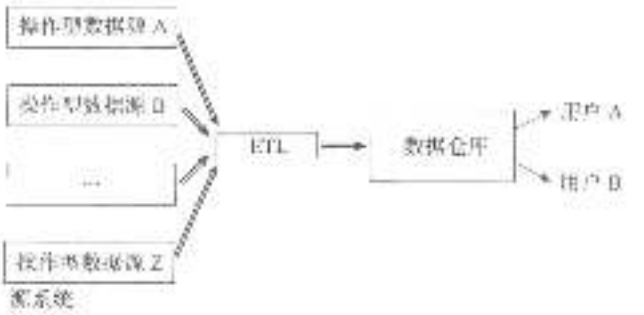


图7-4 数据仓库系统的核心组件

- 最初的操作型目的。
- 作为数据仓库的源系统。

操作型数据存储的最初目的就是为用户提供操作型功能。操作型数据存储也可以用作数据仓库的数据源，为数据仓库提供数据。注意，这些功能是同时进行的。换句话说，就算成为了数据仓库的源系统，操作型数据存储依然能一如既往地完成其操作型功能。

除组织机构中的操作型数据源之外，源系统还可以包含一些外部数据源来为一些实例命名，如第三方市场研究数据、人口数据、股市数据或者天气数据。图7-5给出了一个同时包括企业内部的操作型源系统和外部源系统的数据仓库例子。

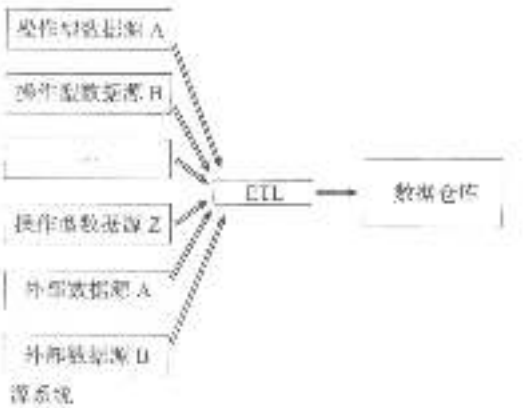


图7-5 同时包括内部与外部源系统的数据仓库

7.4.2 数据仓库

数据仓库本身就是为集成来自源系统的分析型数据提供存储的系统。数据仓库有时又称目标系统，意思是数据仓库是源系统中数据的目的地。典型的数据仓库会周期性地从操作型数据源中检索精选好的可分析的有用数据。在所谓的“活动的”数据仓库中，从操作型数据源中检索数据是一个持续不断的过程。

215

7.4.3 ETL

对任何一个数据仓库来说，完成从操作型数据库中检索数据功能的设备称为提取－转换－加载（extraction-transformation-load，ETL）设备。ETL包含如下几个任务：

- 从操作型数据源中提取可分析的有用数据。
- 转换数据使其满足面向主题目标数据仓库的结构，同时利用数据清洗和数据净化过程来确保转换后的数据质量。
- 将转换后且有质量保证的数据加载到目标数据仓库中。

源系统、ETL以及数据仓库，这三个组件形成了所有数据仓库系统的核心部分。

7.4.4　数据仓库前端（BI）应用

与操作型数据库一样，数据仓库系统通常还提供具有直接访问数据仓库功能的前端应用程序，以满足那些急切希望通过前端应用实现间接访问的用户需求。图 7-6 说明了这一情况。数据仓库前端应用也称为 BI（商务智能）应用

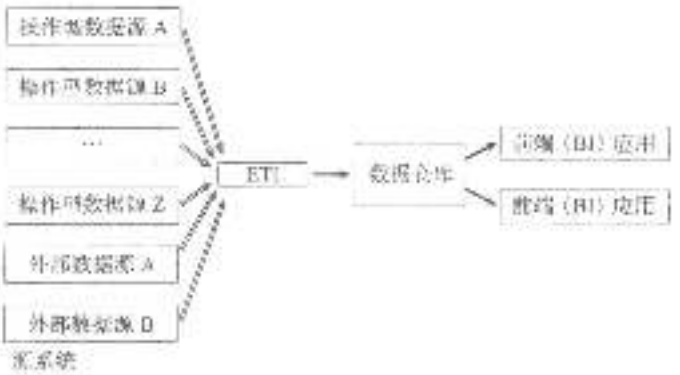


图 7-6　带有前端应用的数据仓库

7.5　数据集市

数据集市是一种与数据仓库具有相似准则的数据存储形式，但相比数据仓库，其范围更小。数据仓库包含的数据是检索自整个组织机构的操作型数据库的多主题分析数据，而数据集市通常较小，包含的数据只与一个主题相关，不涉及整个企业范围。图 7-7 中的表总结了术语“数据仓库”与“数据集市”之间的区别。

数据集市有两种主要类别：独立数据集市与非独立数据集市。

	数据仓库	数据集市
主题	多个	单个
数据源	较多	较少
数据大小	非常大（通常为 T 字节甚至更大）	没有那么大
实现时间	相对较长（数月、数年）	没有那么长
关注范围	整个组织范围	通常小于整个组织范围

图 7-7　数据仓库和数据集市性质

独立数据集市（independent data mart）是一种单独的数据集市，创建形式与数据仓库一致。它有自己的源系统和 ETL 架构。图 7-7 列出了独立数据集市与数据仓库的区别：单一主题、少量数据源、规模小、实现时间短、范围小。

非独立数据集市（dependent data mart）没有自己的源系统，它的数据来自数据仓库。这也是“非独立”的由来。当用户或者应用程序不需要、不必要或不允许用到整个数据仓库的数据时，非独立数据集市就可以简单地为用户提供数据仓库数据的一个子集。

第 8 章中将讨论独立与非独立数据集市用于多种数据仓库建模的过程。

7.6　数据仓库开发步骤

与数据库系统一样，数据仓库系统也需要经过一系列特定顺序的步骤才能建立起来。如

图7-8所示，图中描述了一个数据仓库（DWH）系统生命周期中最核心的开发活动。

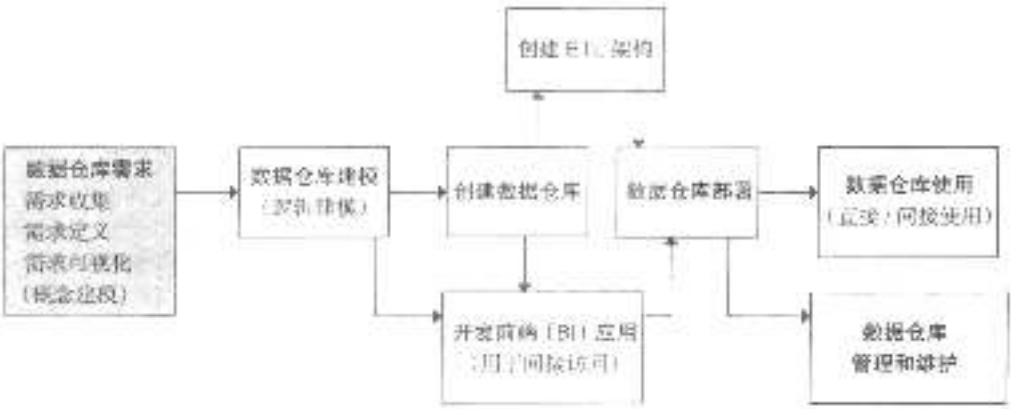


图7-8 数据仓库开发步骤

一旦数据仓库项目启动，包括计划、预算等在内的活动就都开始了。完成这些活动之后 [217]
就是数据仓库的实际开发过程，如图7-8所示。接下来，我们将阐述并讨论数据仓库开发过程中的每个步骤。

7.6.1 需求收集、定义与可视化

需求收集、定义与可视化（requirements collection, definition, and visualization）是数据仓库开发过程中的第一步也是最重要的步骤。如果该步骤成功了，接下来的步骤就可能比较顺利。相反，如果这个步骤出现错误，后面的所有步骤连同整个项目都会失败。该步骤在图7-8中用灰色背景进行了突出显示，以强调其重要性。

该步骤的结果是得到终端用户的需求，这些需求明确指出了未来数据仓库的理想特性和功能。需求以分析型的需要为基础，内部数据源系统和可用的外部数据源中的数据可以满足这些需要。需求收集过程希望可以考虑全部可用数据，但该过程不能以不可用或者不存在的数据为基础。

需求收集的过程需要采访多位数据仓库的相关人员，具体包括：

- 领导、经理人、组织机构中决定和安排最终数据仓库主题的其他决策人员。
- 为更好地理解这些数据源，还包括每个待考虑数据源的技术专家和业务专家。
- 数据仓库的最终使用人员，他们负责分析任务并决定分析主题的细节。

除采访以外，还有其他方法可用于从数据仓库的相关人员处明确具体需求，如小组讨论、提问、调查，或观察已有的分析型实践以了解用户利用这些数据真正做的事情以及用户真正需要的数据。

收集好的需求应该有明确的定义并以书面形式进行陈述，然后利用概念数据模型技术将需求描述成一个概念数据模型，概念数据模型技术可以是ER建模（详见第2章）或维度建模（详见第8章）。

数据仓库需求的收集和定义方法通常是一个迭代的过程。将一个较小的初始需求集合进行收集、定义并且可视化描述后，就可以进一步与开发人员和相关人员讨论，继而将讨论结 [218]
果并入下一步的收集、定义以及需求可视化的迭代过程中，逐步扩大初始需求集合。

即使一个需求集合在数据仓库的需求收集、定义以及可视化描述步骤中都得到了一致认可，它仍然有可能受到开发过程中其他步骤的影响而发生改变，如图 7-9 所示。

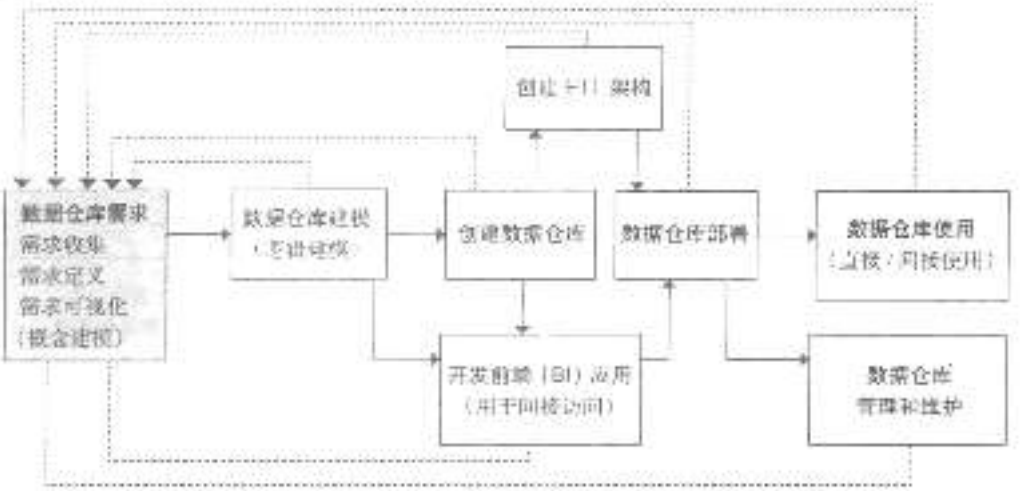


图 7-9　数据仓库需求收集、定义以及可视化描述过程的迭代性质

得到数据仓库需求集合的一种做法是试图在一个单独的过程中完成所有需求的收集、定义、可视化描述操作，然后与开发过程的所有其他步骤进行整合。然而通常更推荐的是另一种方法，即允许在数据仓库开发过程中的每个步骤完成之后进行完善和添加。图 7-9 中的虚线描述了这一点。例如，我们可以收集、定义、可视化描述一个初始的需求集合，继而创建一个初始的数据仓库模型，而其他需求则可以通过一系列类似的迭代过程添加进来。若其他步骤（前端应用或实际数据库使用）显示需要对原来的需求集合进行修改、扩大或减小等操作，则可以对这些需求进行迭代更新。

每次需求集合发生改变时，概念模型就要相应地进行改变，需求的改变要在所有相关步骤中进行传播：建模、创建数据仓库、创建 ETL 架构、创建前端应用、部署、使用以及管理和维护。

所有数据仓库开发步骤都不允许隐式地改变需求。例如，在数据仓库的创建过程中，不允许开发人员临时创建需求没有涉及的新数据仓库结构。相反，如果在数据仓库创建过程中发现有新的结构是系统确实需要用到的（如新的关系表或表中新的列），则需要将需求集合进行扩展，以包含这些新结构。新需求集合则应相应地反映到修改后的可视化描述以及数据仓库模型中去。只有这样，一个新的数据仓库结构才能得以实施。

279

得到数据仓库需求的过程，是整个数据仓库系统开发过程中最重要的步骤，这一点已经得到了广泛认可。该步骤的结果决定了整个数据仓库项目的成功与否。如果该步骤不能成功进行，得到的需求就不能正确地反映系统终端用户的分析要求，因此，最终得到的数据仓库就不能满足终端用户的分析要求。

7.6.2　数据仓库建模

需求收集、定义以及可视化完成之后的第一个步骤，就是**数据仓库建模**（data warehouse modeling）。我们使用术语“数据仓库建模”来表示由数据管理软件实现的数据仓库模型。正如第 1 章中所述，这种模型也称为逻辑数据建模或实施型数据建模，与概念数据建模相

对（概念数据模型不依赖特定数据管理软件的任何逻辑信息，只简单地将需求进行可视化描述）

7.6.3 创建数据仓库

一旦数据仓库模型建立起来，下一步就是**创建数据仓库**（creating the data warehouse）。该步骤利用数据管理软件将数据仓库模型初始化为一个内容为空的实际数据仓库。数据仓库通常被建模为关系数据库，因此常常会使用到关系型DBMS。常规DBMS软件包，如Microsoft SQL Server或Oracle等，都可以用于实现关系型数据库及关系型数据仓库。同时，那些专门用于处理大量数据的DBMS软件包（如Teradata）也可以用于实现数据仓库。

创建数据仓库是一个直观的过程：数据仓库开发人员利用DBMS的功能和性质将数据仓库模型实现为一个实际的功能型分析数据库。

与数据仓库的创建过程类似，数据集市也可以利用DBMS软件包来创建。在某些情况下，数据集市也可以实现为"立方体"，使用一种与关系型DBMS不同的数据管理技术（详见第9章）来创建。

7.6.4 创建ETL架构

创建ETL架构（creating ETL ingrastucture）就是为以下目的创建必要的程序和代码：

- 从操作型数据源中自动提取相关数据；
- 转换提取得到的数据，以确保数据质量并将其结构转换成模型和数据仓库的相应结构；
- 转换后数据与数据仓库的无缝接入。

ETL架构必须解释并协调操作型数据源与目标数据仓库在数据及元数据上的差别。在许多情况下，组织机构有多个独立的数据源，数据源中的数据存在重复现象。此时，创建ETL架构的过程还要考虑如何引入数据但不产生多余的数据复制（即如何引入所有有用信息，同时避免唯一性数据违反问题）。

由于需要考虑的数据细节数量较多，因此创建ETL架构往往是数据仓库开发过程中最耗费时间与资源的部分。

7.6.5 开发前端（BI）应用

开发前端应用（developing front-end/BI application）的过程就是设计和创建可供终端用户间接使用数据仓库的应用。大部分数据仓库系统中都包含前端应用，也称为商务智能（business intelligence，BI）应用。数据仓库前端应用通常包含接口（如表单）以及通过菜单等导航机制得到的报表等。

220

如图7-8所示，数据仓库前端应用的设计和创建过程可以和数据仓库的创建过程同时进行。例如，前端应用的外观及视觉效果、各种组件（表单和报表）的个数及其功能都可以在数据仓库实现之前确定。该步骤依据的是数据仓库的模型和需求，模型和需求反映了终端用户需要系统所具有的特性和功能。当然，前端应用的实际创建过程还包括将该应用连接到数据仓库，该过程需在数据仓库实现后才能进行。

7.6.6 数据仓库部署

数据仓库及其前端应用实现后的下一步，就是**数据仓库部署**（data warehouse deployment）。

该步骤就是将数据仓库及其前端应用交给用户使用。通常在该步骤之前要先执行初始导入步骤，即通过 ETL 架构完成数据仓库及初始数据集从操作型数据源中的初始导入。

7.6.7　数据仓库使用

一旦数据仓库部署完成，终端用户就可以进行**数据仓库使用**（data warehouse use）。数据仓库的使用即检索数据仓库中包含的数据。数据仓库由用户通过前端应用进行间接使用，也可以通过管理数据仓库的数据管理软件语言进行直接使用。例如，关系型数据仓库可以利用 SQL 语句直接查询。终端用户还可通过所谓的在线分析处理（OLAP）工具（或 BI 工具）对数据仓库进行特殊的分析查询，这是一种更通用的数据仓库直接使用方法。

7.6.8　数据仓库管理与维护

数据仓库管理与维护（data warehouse administration and maintenance）活动用来支持终端用户对数据仓库的使用。与操作型数据库的管理与维护活动类似，数据仓库的管理与维护活动包括各类技术问题的处理。如保证数据仓库的信息安全、确保数据仓库中的内容拥有充分的硬盘空间、实现数据的备份与恢复。

7.7　数据仓库的新版本

通常来说，数据仓库在投入使用之后就会有另外的新版本出现。新的数据仓库需要根据相同的开发步骤来创建，如图 7-10 所示。

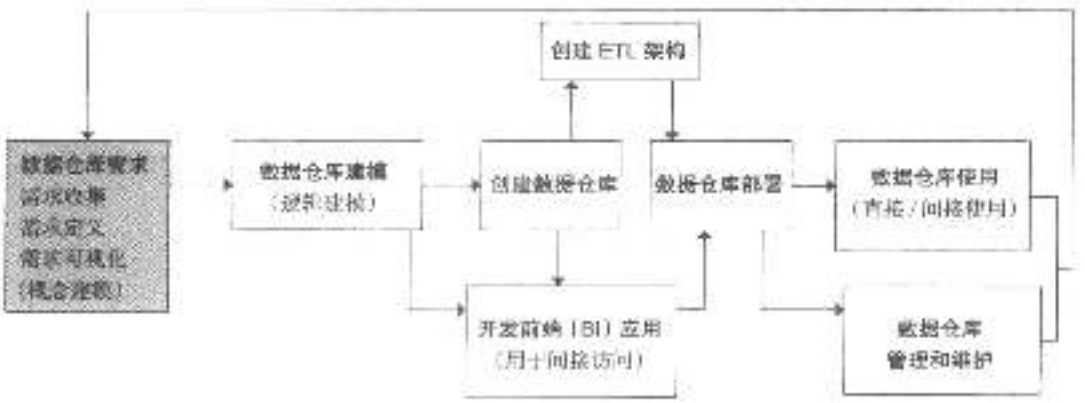


图 7-10　新版本数据仓库的开发

和原始版的数据仓库一样，新版本数据仓库的开发过程同样需要从需求的收集、定义、可视化描述步骤开始。当然，与原始版数据仓库不同的是，后面版本的数据仓库并不需要从头开始收集需求。原来的需求集合可以作为添加和修改的基础，许多添加和修改都来自于终端用户在前面版本使用过程中的观察和发现，进而证明了可以通过何种方式对数据仓库进行改善和扩展。还有一些新的需求则可能源自操作型数据源的改变或是内部技术的改变。和其他任何一种信息系统一样，数据仓库系统也不可避免地会不断有新版本出现。

[221]

本章给出了关于数据仓库基本概念的一个概述，第 8 章与第 9 章将会给出更多的细节。第 8 章将重点讨论数据仓库建模，第 9 章则包括数据仓库新版本开发的相关问题。

关键术语

analytical information（分析型信息），207
application-oriented（面向应用），210
creating ETL infrastructure（创建ETL架构），220
creating the data warehouse（创建数据仓库），220
data mart（数据集市），216
data warehouse（数据仓库），212
data warehouse administration and maintenance（数据仓库管理与维护），221
data warehouse deployment（数据仓库部署），221
data warehouse modeling（数据仓库建模），220
data warehouse use（数据仓库使用），221
dependent data mart（非独立数据集市），217
developing front-end BI application（开发前端（BI）应用），220
extraction-transformation-load, ETL（抽取－转换－加载），216
independent data mart（独立数据集市），216
operational information/transactional information（操作信息/事务信息），207
requirements collection, definition and visualization（需求收集、定义及可视化），218
source system（源系统），215
subject-oriented（面向主题），210

复习题

Q7.1 数据仓库的主要目的是什么？
Q7.2 为什么数据仓库要创建为单独的数据存储形式？
Q7.3 描述操作型与分析型数据在时间范围上的差别。
Q7.4 描述操作型与分析型数据在数据细节层次上的差别。
Q7.5 描述操作型与分析型数据在时间表示上的差别。
Q7.6 描述操作型与分析型数据在查询总量与查询频度上的差别。
Q7.7 描述操作型与分析型数据在数据更新上的差别。
Q7.8 描述操作型与分析型数据在数据冗余上的差别。
Q7.9 描述操作型与分析型数据在数据用户上的差别。
Q7.10 描述操作型与分析型数据在数据面向对象上的差别。
Q7.11 解释下面数据仓库中各部分的定义：结构化存储、集成性、面向主题、企业范围、历史性、时变性、为可分析信息的检索而开发，可能包括最细粒度的细节数据或汇总数据（或二者兼有）。 222
Q7.12 数据仓库系统的主要组成部分是什么？
Q7.13 什么是数据集市？
Q7.14 非独立数据集市与独立数据集市的区别是什么？
Q7.15 开发数据仓库的步骤是什么？
Q7.16 解释数据仓库需求收集、定义及可视化描述过程中的迭代特性。
Q7.17 简要描述创建数据仓库的过程。
Q7.18 简要描述创建ETL架构的过程。
Q7.19 简要描述开发数据仓库前端（BI）应用的过程。
Q7.20 数据仓库部署过程如何运行？
Q7.21 数据仓库的使用包括哪些方面？
Q7.22 给出一个数据仓库管理与维护活动的例子。 223
Q7.23 数据仓库初始版本与后续新版本的开发有什么异同？ 224

数据仓库与数据集市建模

8.1 引言

在本书的前面部分，我们详细介绍了操作型数据库的建模过程。第 2 章描述了 ER 建模的细节，它是一种可视化数据库需求的主要技术，大量用于操作型数据库的概念建模。第 3 章讲解了作为操作型数据库逻辑建模标准方法的关系建模。这两种技术都可用于数据仓库和数据集市的开发。此外，在数据仓库和数据集市的建模实践中，还经常使用一种专门针对分析型数据库设计的技术维度建模技术。

本章首先详细介绍维度建模，接着介绍最常用的数据仓库建模策略，包括如何利用 ER 建模、关系建模和维度建模。

8.2 维度建模基本概念

维度建模（dimensional modeling）是用于面向主题的分析型数据库（如数据仓库、数据集市）设计的一套数据设计方法。通常将维度建模作为一种关系数据建模技术，需要设计包含主码的关系表，且关系表之间通过外码相互连接，同时满足标准的关系完整性约束。除了使用常规的关系概念（主码、外码、完整性约束等）外，维度建模的不同之处在于，它有另外两种类型的表——维度和事实。

- **维度表**（dimensional table/dimension）包含对分析主题所属类型的描述，如商业、组织或企业。维度表中的列通常包含文本型的描述信息（如产品品牌、产品颜色、顾客性别、顾客受教育程度），但也可以是数值型的（如产品重量、顾客收入水平）。这些信息提供了对该主题进行分析的基础。例如，如果对主题商业进行销售方面的分析，则可以对产品品牌、顾客性别、顾客收入水平等进行分析。
- **事实表**（fact table）包含对于所分析主题的度量。另外，事实表还包含与维度表关联的外码。事实表中的度量通常是数值型的，用于数学计算和定量分析。例如，如果商业分析的主题是销售，那么销售事实表中的一个度量可能是销售金额。可以使用不同的数学函数对各种不同维度的列计算销售金额。例如，可以对每种产品品牌、顾客性别、顾客收入水平等计算总销售额和平均销售额。

[225

关系维度建模的结果是一个包含事实和维度的维度关系模式，如图 8-1 所示。维度模式通常是**星形模式**（star schema）。

维度表包含一个主码和多种属性，这些属性用于事实表中的度量分析。事实表包含事实度量属性和连接事实表与维度表的外码。在本书中，我们将事实表用加粗的边框表示，以便和维度表区分开来。事实表的主码是一个由外码列和（或）事实表中其他列构成的组合码（将在本章后面的部分详细说明）。

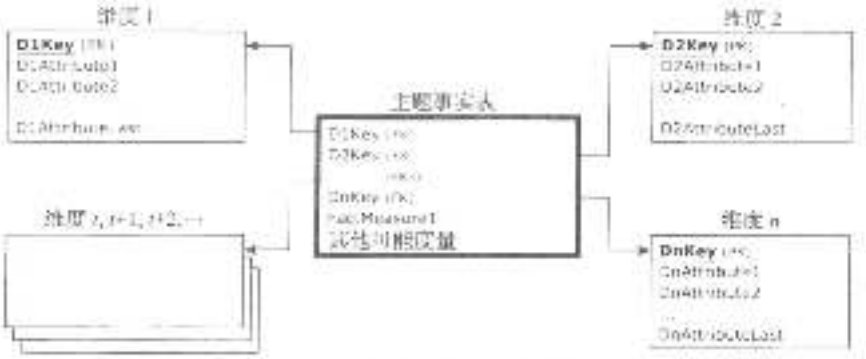


图8-1 一个维度模型（星形模式）

尽管事实表（或者任意表）中列的顺序都可以任意排列，但出于可读性的原因，在本书的星形模式的例子中，我们总是将事实表中的度量放到表的最后。

8.3 初始实例：基于单个数据源的维度模型

下面的例子说明了维度建模的基本概念。该例子使用本书第2章和第3章曾用过的ZAGI零售公司销售部门的业务场景。为了方便，我们重新把ZAGI零售公司销售部门的数据库及其中的记录列举出来。

图8-2（与图3-32相同）展示了ZAGI零售公司销售部门数据库的ER图和所得到的关系模式。

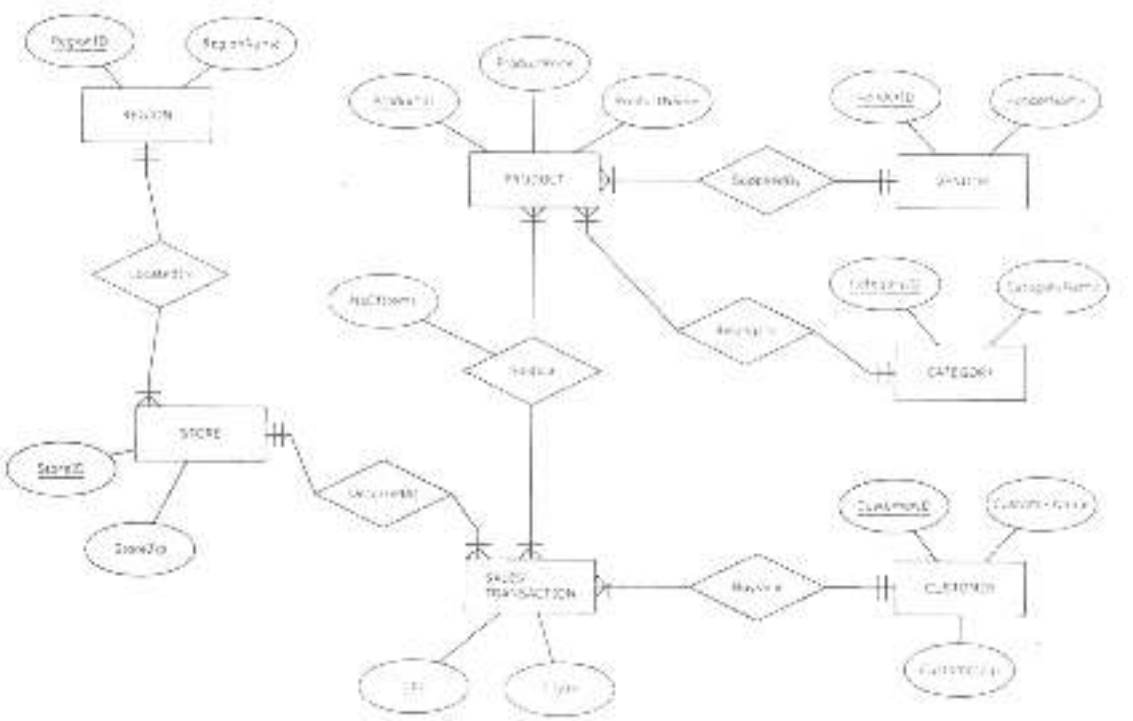


图8-2 ZAGI零售公司销售部门数据库：ER图和所得到的关系模式

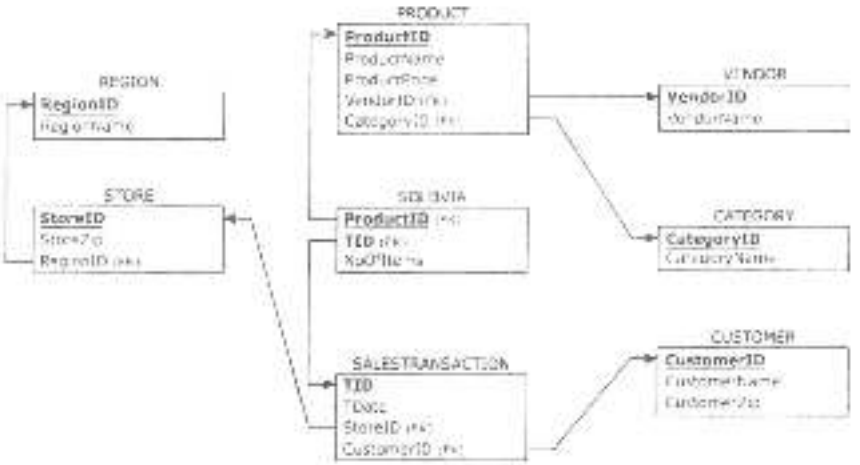


图 8-2（续）

图 8-3（与图 3-33 相同）展示了 ZAGI 零售公司销售部门数据库中的记录。

REGION

RegionID	RegionName
C	Chicagoland
T	Toledo

STORE

StoreID	StoreZip	RegionID
S1	60600	C
S2	60600	C
S3	35400	T

PRODUCT

ProductID	ProductName	ProductPrice	VendorID	CategoryID
1X1	Zoo Bag	$100	PG	CP
2X2	Easy Boot	$20	MK	FW
3X3	Easy Sock	$15	MK	FW
4X4	Dina Boot	$90	PG	FW
5X5	Tiny Tent	$150	MK	CP
6X6	Biggy Tent	$250	MK	CP

VENDOR

VendorID	VendorName
PG	Pacifica Gear
MK	Mountain King

CATEGORY

CategoryID	CategoryName
CP	Camping
FW	Footwear

SALESTRANSACTION

TID	CustomerID	StoreID	TDate
T111	1-2-333	S1	1-Jan-2013
T222	2-3-444	S2	1-Jan-2013
T333	1-2-333	S3	2-Jan-2013
T444	3-4-555	S3	2-Jan-2013
T555	2-3-444	S3	2-Jan-2013

SOLDVIA

ProductID	TID	NoOfItems
1X1	T111	1
2X2	T222	1
3X3	T333	3
1X1	T333	1
4X4	T444	1
2X2	T444	2
4X4	T555	4
5X5	T555	2
6X6	T555	1

CUSTOMER

CustomerID	CustomerName	CustomerZip
1-2-333	Tina	60137
2-3-444	Tony	60611
3-4-555	Pam	35401

图 8-3　图 8-2 所示的 ZAGI 零售公司销售部门数据库的数据记录

基于图 8-2 和图 8-3 中所示的操作型数据库，ZAGI 零售公司决定使用维度建模技术来设计一个分析型数据库，该数据库要分析的主题是销售。该过程的结果是图 8-4 所示的星形模式。

在星形模式中，选定的分析主题用事实表来表示。在本例中，选定的分析主题（销售）用 SALES 事实表来表示。

226

设计星形模式时，需要考虑使用事实表中的那些维度来表示选定的主题。对于每个要考虑的维度，需要回答以下两个问题：

问题 1：该维度对于选定主题的分析是否有用？

问题 2：该维度是否可以基于已有的数据源来创建？

在这个特殊的例子中，可用的数据源只有ZAGI零售公司销售部门数据库，如图8-2和图8-3所示。

图8-4中的星形模式包含四个维度，分别命名为PRODUCT、STORE、CUSTOMER和CALENDAR。对于每个维度，上面的问题1和问题2的回答都是关。 227

图8-4 ZAGI零售公司分析型数据库的维度模型，分析的主题是销售

特别地，当考虑第一个问题时，维度建模团队认为，根据产品、顾客、商店和日期对销售额进行分析是有用的。因此，该团队决定根据这些分析的领域（产品、顾客、商店和日期）来考虑创建维度。

考虑第二个问题时，该团队进一步确认，对于正在考虑之中的每个维度，已有的操作型数据库都可以提供数据来源。因此，该团队决定创建所有四个维度：PRODUCT、CUSTOMER、STORE和DATE。

维度PRODUCT是将操作型数据库中的表PRODUCT与表VENDOR和CATEGORY连接得到的，这是为了包含供应商名称和类别名称。它使我们既可以按照单个产品来分析销售额，也可以按照产品的供应商和类别来分析销售额。 228

维度CUSTOMER与操作型数据库中的表CUSTOMER相同。它使我们既可以按照单个顾客来分析销售额，也可以按照顾客的邮政编码来分析销售额。

维度STORE是将操作型数据库中的表STORE与表REGION连接得到的，这是为了包含地区名称。它使我们既可以按照单个商店来分析销售额，也可以按照销售地区和邮政编码来分析销售额。

维度CALENDAR记录了操作型数据库中表SALESTRANSACTION中的列TDate的日期范围。与日期相关的分析是最常见的主题分析类型之一。例如，按照月或者季度来分析销售额是一种常见的与日期相关的分析。实际上，每个星形模式中都包括一个与日期信息相关的维度。在本例中，维度CALENDAR充当了该角色。将完整的日期分解成各种与日期相关的单个成分（如季度、月份、年份和周中的天）来扩展维度CALENDAR的结构，这样将有助于分析。

图8-5为图8-4所示的星形模式填充后的表。图8-5每个表中的数据都来自图8-3所示的操作型数据库中的数据源（ZAGI零售公司销售部门数据库）。

SALES事实表中的每条记录代表某一天某一顾客在某一商店购买某一产品的购买记录。

在图8-5中，我们没有给出SALES事实表的主码。本章后面将会讨论事实表的主码，并给出SALES事实表的主码。

CALENDAR 维度

CalendarKey	FullDate	DayOf Week	DayOf Month	Month	Qtr	Year
1	1/1/2018	Tuesday	1	January	Q1	2018
2	1/2/2019	Wednesday	2	January	Q1	2019

STORE 维度

StoreKey	StoreID	StoreZip	StoreRegionName
1	S1	90630	Cinnagaland
2	S2	60625	Cinnagaland
3	S3	35620	Tuxtek

PRODUCT 维度

ProductKey	ProductID	Product Name	Product Price	Product Vendor Name	Product Category Name
1	1X1	Zzz Bag	$100	Pacific Gear	Camping
2	1X2	Fury Boot	$80	Mountain King	Footwear
3	3X3	Cody Boot	$75	Mountain King	Footwear
4	4X4	Dura Boot	$90	Pacific Gear	Footwear
5	1X1	Tiny Tent	$150	Mountain King	Camping
6	5X5	Bigge Tent	$250	Mountain King	Camping

CUSTOMER 维度

CustomerKey	CustomerID	CustomerName	CustomerZip
1	1-2-333	Tom	60137
2	2-3-444	Jim	60611
3	3-4-555	Pam	35401

SALES 事实表

CalendarKey	StoreKey	ProductKey	CustomerKey	DollarsSold	UnitsSold
1	1	1	1	$100	1
1	2	2	2	$80	1
2	3	3	1	$75	5
2	3	1	1	$100	1
2	3	4	3	$90	1
2	3	2	3	$140	2
2	3	4	2	$360	4
2	3	5	2	$300	2
2	3	6	2	$250	1

图 8-5　ZAGI 零售公司用于分析销售的维度模型。该维度模型已经填充了来自操作型数据源的数据

8.4　维度特性、事实特性及初始实例分析

在一个典型的设计良好的星形模式中，任何维度表中的记录（行）数都小于事实表中的记录数。典型的维度包含相对静态的数据，而事实表中的记录会不断添加，表的规模会迅速增长。因此，在一个典型的维度建模的分析型数据库中，维度表的记录数要比事实表少几个数量级。

229

例如，在 ZAGI 零售公司的例子中，如果图 8-4 和图 8-5 所示的维度建模的分析型数据库中保存了 10 年的有价值数据，那么维度 CALENDAR 将会有 3652 条记录（365 × 10+2 天，假设这 10 年间有两个闰年，每个闰年多加 1 天）。假设 ZAGI 零售公司有 100 家商店、100 000 位顾客和 5000 种不同的商品，则维度 STORE、CUSTOMER 和 PRODUCT 将分别有 100、100 000 和 5000 条记录。即使保守估计，每家商店每天只有 200 位顾客且平均购买 3 种产品，那么事实表将会有 219 120 000 条记录（200 × 3 × 100 × 3652 条记录），这比任何一个维度的记录数都要高出几个数量级。

在一个星形模式中，所有维度表通常都被分配了一个简单的、非复合的，由系统生成的码，称为**代理码**（surrogate key）。在图 8-4 中，维度 CALENDAR 的代理码为 CalendarKey，维度 PRODUCT 的代理码为 ProductKey，维度 STORE 的代理码为 StoreKey，维度 CUSTOMER 的代理码为 CustomerKey。这些码的值都是简单的自动分配的整数，如图 8-5 所示。代理码的值没有任何意义或者额外目的，它只是在维度模型中为每个维度增添一列作为主码，而不是采用原来的操作码。例如，不使用操作型数据库中的表 PRODUCT 的主码 ProductID 作为维度 PRODUCT 的主码，而是创建了新的代理码列。创建代理主码（如 ProductKey）而不使用操作型主码（如 ProductID）作为维度的主码，主要原因之一是为了能够处理所谓的缓慢变化的维度。我们将会在本章的后面部分讲述缓慢变化的维度。

在图 8-4 和图 8-5 所示的维度建模的数据库中，ZAGI 零售公司可以很容易地创建如下

分析型查询。

QueryA：比较已售出商品的数量，商品为2013年第一季度到第二季度之间有个周六，由Tristate地区的Pacifica Gear供应商提供的Camping类所有商品的售出数量。

在维度建模的数据库中，这样的查询可使用SQL语句或所谓的OLAP/BI工具（详见第9章）直接处理。

首先考虑下面几种使用维度数据库得到的QueryA的SQL版本（见图8-4和图8-5）。

SQL Query A－维度版本（使用图8-5）：

```
SELECT  SUM(SA.UnitsSold)
,       P.ProductCategoryName
,       P.ProductVendorName
,       C.DayofWeek
,       C.Qtr
FROM
        Calendar C
        Store    S
        Product  P
        Sales    SA
WHERE
        C.CalendarKey          =   SA.CalendarKey
AND     S.StoreKey             =   SA.StoreKey
AND     P.ProductKey               SA.ProductKey
AND     P.ProductVendorName    =   'Pacifica Gear'
AND     P.ProductCategoryName  =   'Camping'
AND     S.StoreRegionName      =   'Tristate'
AND     C.DayofWeek            =   'Saturday'
AND     C.Year                 =   2013
AND     C.Qtr                  IN  ( 'Q1' , 'Q2' )
GROUP BY
            P.ProductCategoryName,
            P.ProductVendorName,
            C.DayofWeek,
            C.Qtr;
```

230

现在考虑下面几种使用非维度数据库得到的QueryA的SQL版本（见图8-2和图8-3）。在下面的例子中，我们将使用这些函数从日期中提取年份、季度和星期[⊖]。

SQLQuery A－非维度版本（使用图8-3）：

```
SELECT  SUM( SV.NetUnits )
,       C.CategoryName
,       V.VendorName
,       EXTRACTWEEKDAY(ST.Date)
,       EXTRACTQUARTER(ST.Date)
FROM
        Region     R
        Store      S
        SalesTransaction ST
        SaleWin    SV
        Product    P
        Vendor     V
        Category   C
WHERE
        R.RegionID          =   S.RegionID
AND     S.StoreID           =   ST.StoreID
AND     SV.TID              =   SV.TID
AND     SV.ProductID        =   P.ProductID
```

⊖ 为简化该实例，本书使用虚拟的EXTRACTYEAR(Date)、EXTRACTQUARTER(Date)和EXTRACTWEEKDAY(Date)函数。实际上，本书的DBMS中用于提取年份、季度和星期的SQL函数的语法和复杂度是不同的。

```
AND  P.VendorID        =   V.VendorID
AND  P.CategoryID      =   C.CategoryID
AND  V.VendorName      =   'Pacifica Gear'
AND  C.CategoryName    =   'Camping'
AND  R.RegionName          'Tristate'
AND  EXTRACTWEEKDAY(ST.Date) =   'Saturday'
AND  EXTRACTYEAR(ST.Date)    =   2013
AND  EXTRACTQUARTER(ST.Date) IN ( 'Q1', 'Q2' )
GROUP BY
     C.CategoryName,
     V.VendorName,
     EXTRACTWEEKDAY(ST.Date),
     EXTRACTQUARTER(ST.Date);
```

比较上面两个查询，考察分析人员在进行查询时维度模型所具有的优势。注意，SQL Query A- 维度版本将事实表及其 3 个维度进行连接，而 SQL Query A- 非维度模型将 7 张表进行连接并使用函数从日期里提取年份、季度和星期。

这个小例子证明，即使对于维度模型是基于单一数据源建立起来的简单情况，维度模型数据库上的分析型查询也要比同等的非维度数据库上的查询更容易建立。在分析所需的数据存在于多个数据源的情况下，使用维度模型数据进行分析的便利则更为明显。可分析的有用数据存在于一个组织机构的多个独立数据系统及存储设备中的情况是十分常见的。下面的实例将说明维度建模在这些场景中的使用情况。

8.5　扩展实例：基于多个数据源的维度模型

图 8-4 和图 8-5 给出的初始维度建模实例描述的是基于单一数据源（ZAGI 零售公司销售部门数据库，见图 8-2 和图 8-3）的分析型数据库。

231

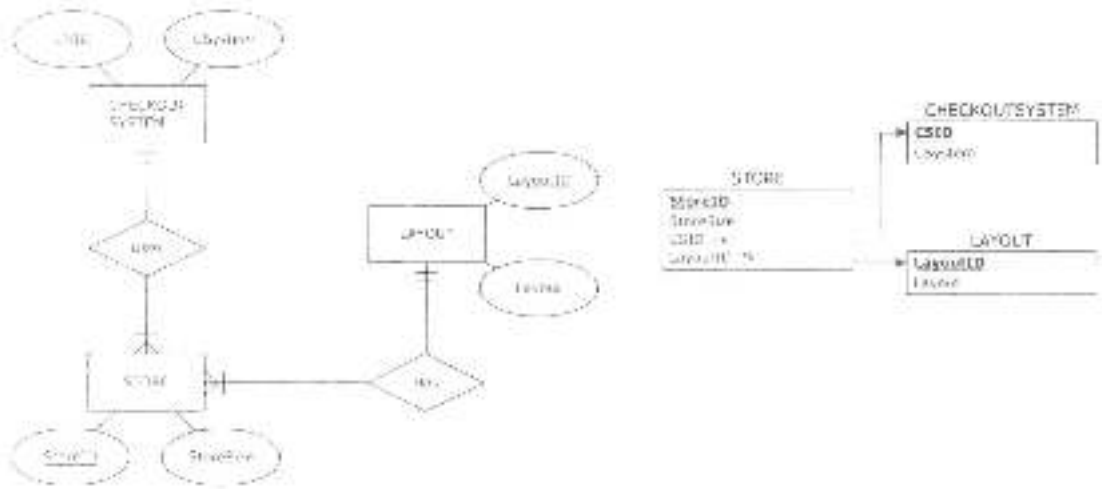
更为一般地，数据仓库与数据集市从多个源中得到数据。下面将使用 ZAGI 零售公司的场景对初始实例进行扩展，添加了两个数据源：

- 在 ZAGI 零售公司中，除了图 8-2 以及图 8-3 所示的销售部门数据库，设备部门还有一个数据库用于保存如下有关每个商店物理细节的信息：商店规模、商店布局、商店结账系统。ZAGI 零售公司设备部门数据库如图 8-6 所示。分析人员认为商店的物理细节可以作为分析销售数据的一个有用方面，因此同样对维度模型进行了扩展来说明该数据。
- ZAGI 零售公司分析人员认为用户信息统计数据可以作为销售数据分析的另一个有用方面。ZAGI 零售公司任何一个操作型数据库中都未包含用户信息统计的相关数据。因而，ZAGI 零售公司从一个市场研究公司获得顾客的用户信息统计数据。所得的外部数据源顾客用户信息数据表如图 8-7 所示。该外部数据源可用于进一步扩展维度模型。

232

图 8-8 给出了一个扩展维度模型的例子，该例子基于三个不同的数据源：两个内部的操作型数据库（图 8-2 与图 8-3 所示的 ZAGI 零售公司销售部门数据库、图 8-6 所示的 ZAGI 零售公司设备部门数据库）以及一个外部数据源（图 8-7 所示的顾客信息统计数据表）。注意，ZAGI 零售公司设备部门数据库使得 STORE 维度新增了列 StoreSize、StoreCSystem 以及 StoreLayout。同时还要注意，外部数据源顾客信息统计数据表使得维度 CUSTOMER 新增了列 CustomerGender、CustomerMaritalStatus、CustomerEducationLevel 以及 CustomerCreditScore。

图 8-9 给出了图 8-8 中维度模型对应的表。



CHECKOUTSYSTEM

CSID	CSystem
CR	Cashier
AC	Self Service
MX	Mixed

STORE

StoreID	StoreSize (sq.ft.)	CSID	LTID
S1	50000	CR	M
S2	35000	AC	T
S3	20000	MX	T

LAYOUT

LayoutID	Layout
M	Modern
T	Traditional

图8-6 ZAGI零售公司设备部门数据库ER模型、关系模型和表

CUSTOMER TABLE

CustomerID	Customer Name	Gender	Marital Status	Education Level	Credit Score
123-789	Tina	Female	Single	College	700
234-444	Tony	Male	Single	High School	650
345-555	Pam	Female	Married	College	620

图8-7 顾客用户数据表（从一个市场研究公司处得到的外部源）

图8-8 ZAGI零售公司销售数据分析的扩展维度模型，基于多个数据源建立

现在ZAGI零售公司可以考虑包含更多分析因素的更为精细的分析问题。例如，考虑下面几个问题：

CALENDAR 维度

CalendarKey	FullDate	DayOf Week	DayOf Month	Month	Qtr	Year
1	1/1/2013	Tuesday		January	Q1	2013
2	1/2/2013	Wednesday	2	January	Q1	2013

PRODUCT 维度

ProductKey	ProductID	Product Name	Product Price	Product Vendor Name	Product Category Name
1	1X1	Zip Gap	$100	Pacifica Gear	Camping
2	2X2	Easy Boot	$70	Mountain King	Footwear
3	3X3	Cozy Stock	$15	Mountain King	Footwear
4	4X4	Duro Rain	$90	Pacifica Gear	Footwear
5	5X5	Tiny Tent	$150	Mountain King	Camping
6	6X6	Bigby Tent	$250	Mountain King	Camping

STORE 维度

StoreKey	StoreID	StoreZip	StoreRegion Name	Store Size (m²)	Store OSystem	Store Layout
1	S1	60000	Chicagoland	51000	Cashwell	Western
2	S2	60505	Chicagoland	35000	Self Service	Traditional
3	S3	35400	Tristate	55000	Mundi	Traditional

CUSTOMER 维度

CustomerKey	CustomerID	Customer Name	Customer Zip	Customer Gender	Customer MaritalStatus	Customer EducationLevel	Customer CreditScore
1	1-2-331	Tina	60032	Female	Single	College	700
2	2-3-444	Tony	60611	Male	Single	High School	650
3	3-4-555	Pam	35401	Female	Married	College	600

SALES 事实表

CalendarKey	StoreKey	ProductKey	CustomerKey	DollarsSold	UnitsSold
1	1	1	1	$100	1
1	2	2	2	$70	1
2	3	3		$70	6
2	1	1	1	$100	1
2	3	4	3	$90	1
2	3	2	3	$140	2
2	3	4	2	$360	4
2	3	5	2	$900	2
2	3	6	2	$250	1

图 8-9　为 ZAGI 零售公司销售记录分析所建立的扩展维度模型，数据来自多个数据源

Query B：比较已售出商品的数量，商品为 2013 年第一季度到第二季度之间每个周六在 Modern 商店售出，由 Tristate 地区的 Pacifica Gear 供应商供货的 Camping 类且出售给男性顾客的所有商品。

为了从操作型数据源中获得 Query B 的答案，用户需要跨越所有 3 个操作型数据源进行查询，然后将结果结合起来获得最后的答案。

使用图 8-9 所示的维度模型来获得 Query B 的答案将更为直接。SQL Query B- 维度版本包括 SQL Query A- 维度版本的一个简单扩展。

SQL Query B– 维度版本（使用图 8-9）：

```
SELECT  SUM(SA.UnitsSold)
,       P.ProductCategoryName
,       P.ProductVendorName
,       C.DayofWeek
,       C.Qtr
FROM
        Calendar C
,       Store    S
,       Product  P
,       Customer CU
,       Sales    SA
WHERE
        C.CalendarKey        SA.CalendarKey
```

```
AND  S.StoreKey          =  SA.StoreKey
AND  P.ProductKey        =  SA.ProductKey
AND  CU.CustomerKey      =  SA.CustomerKey
AND  P.ProductVendorName     =    'Pacifica Gear'
AND  P.ProductCategoryName       'Camping'
AND  S.StateProvinceName     =    'Tristate'
AND  C.DayofWeek             =    'Saturday'
AND  C.Year                  =    2013
AND  C.Qtr                   IN   ( 'Q1', 'Q2' )
AND  S.StoreLayout               'Modern'
AND  CU.Gender               =    'Male'
GROUP BY
     P.ProductCategoryName,
     P.ProductVendorName,
     C.DayofWeek,
     C.Qtr;
```

233

对使用维度建模的数据查询，如上文的 Query B，可以利用 BI/OLAP 工具由更简单的版本实现（如第 9 章所述）。

8.6　其他可能的事实属性

正如本章前面部分所讨论的，事实表包含了将表连接到维度表的外码以及与分析主题相关的度量。例如，如果业务分析的主题是销售，销售事实表中的度量通常是销售金额和数量。在某些情况中，除了与分析主题相关的度量，事实表还包含其他非度量的属性，如图 8-10 给出的例子。SUBJECT 事实表中，"其他可能度量"表明事实表可以包含其他可能的非度量属性。

234

图 8-10　事实表包含额外属性的维度模型（星形模式）

出现在事实表中的两种最典型的其他属性是**事务标识码**（transaction identifier）以及**事务时间**（transaction time）。下面将用 ZAGI 零售公司的例子说明这两种属性。

8.7　事实表中的事务标识码

首先，我们将说明事务标识码的概念。注意图 8-2 与图 8-3 中所示的 ZAGI 零售公司销售部门数据库，SALES TRANSACTION 关系有一个表示事务标识码的列 TID。TID 值本身并没有任何分析价值，例如，分析 TID 值以偶数结尾的销售以及 TID 值以奇数结尾的销售并不能提供任何有价值的商业见解。然而，对某些特定类型的分析，TID 值可以提供其他见解。例如，在 ZAGI 零售公司的情景中，TID 值可以提供有关哪些产品在相同的交易事务中被出售的信息，这样的信息对很多分析任务都有用。有一种分析是试着建立那些经常被一

起出售的产品之间的关联，这种分析一般称为购物篮数据分析（又称为“关联规则挖掘”或“关联性分组”，见附录 A）。维度模型中包含了 TID 后，事务分析人员便可使用这种分析。

一旦决定在维度模型中包含 TID，剩下的问题则变成：TID 应该放在星形模式中的什么位置？初看之下，可以创建一个单独的 TRANSACTION 维度，该维度包含 TID 属性。然而，正如本章前面所述，通常在一个设计合理的星形模式中，与事实表的记录（行）数比起来，每个维度表中的记录（行）数要小得多。如果创建一个单独的 TANSACTION 维度，则该维度的记录数和 SALES 事实表中的记录数将在同一个数量级（如百万、十亿个记录）上，且同一时刻的行数数量级比任何其他维度的行数都要高。

一种更实用的方法是将 TID 作为事实表中额外的列包含进来，如图 8-11 所示。这种简单的方法同样可以分析哪些产品在同一个交易中被售出，且不需要新建记录数目巨大的维度。实际上，事务的标识码可以表示为销售事务标识码、顺序标识码、租赁标识码、票据标识码、订单标识码等。让事实表包含事务标识码的方法，在理论与实践中都常常称为**退化维度**（degenerate dimension），术语“退化”的意思是“算术上更简单”。该术语简单地反映了在事实表中包含事件标识码比为其创建一个单独的维度更简单。

235

图 8-11　ZAGI 零售公司分析任务相关的维度模型中包含了带事务标识码属性（TID）的事实表

图 8-12　展示了图 8-11 所示维度中的记录以及维度模型的事实表。

CALENDAR 维度

CalendarKey	FullDate	DayOf Week	DayOf Month	Month	Qtr	Year
1	1/1/2013	Tuesday	1	January	Q1	2013
2	1/2/2013	Wednesday	2	January	Q1	2013

PRODUCT 维度

ProductKey	ProductID	Product Name	Product Price	Product Vendor Name	Product Category Name
1	1X1	Zip Bag	$100	Pacifica Gear	Camping
2	2X2	Easy Boot	$70	Mountain King	Footwear
3	3X3	Cosy Sock	$15	Mountain King	Footwear
4	4X4	Jura Boot	$60	Pacifica Gear	Footwear
5	5X5	Tiny Tent	$150	Mountain King	Camping
6	6X6	Biggy Tent	$250	Mountain King	Camping

STORE 维度

StoreKey	StoreID	StoreZip	StoreRegion Name	Store Size (m^2)	Store CSystem	Store Layout
1	S1	60600	Chicagoland	51000	Cashiers	Modern
2	S2	60605	Chicagoland	35000	Self Service	Traditional
3	S3	85400	Tristate	55000	Mixed	Traditional

图 8-12　图 8-11 所示维度模型的记录（包含 TID 值）

CUSTOMER 缘度

CustomerKey	CustomerID	Customer Name	Customer Zip	Customer Gender	Customer MaritalStatus	Customer EducationLevel	Customer CreditScore
1	[illegible]	[illegible]	[illegible]	Female	Single	[illegible]	700
2	[illegible]	[illegible]	[illegible]	Male	Single	[illegible]	620
3	[illegible]	[illegible]	[illegible]	Female	Married	[illegible]	628

SALES 事实表

CalendarKey	StoreKey	ProductKey	CustomerKey	TID	DollarsSold	UnitsSold
			1	[illegible]	$[illegible]	[illegible]
1	2	2	2	T222	$70	[illegible]
2	3	3	1	T333	$[illegible]	[illegible]
3	3	1	1	T333	$100	[illegible]
2	3	4	3	T444	$30	1
2	3	3	3	T444	$140	2
1	3	4	2	T555	$360	4
2	3	5	2	T555	$360	9
3	3	5	2	T555	$250	1

图 8-12　（续）

8.8　事实表中的事务时间

为说明事务时间的概念，我们对 ZAGI 零售公司的例子稍作扩展。我们通过向 SALESTRANSACTION 实体以及图 8-13 所示的结果 SALESTRANSACTION 关系中添加事务时间（TTime）属性，来扩展图 8-2 所示的图表。图 8-13 所示 ER 图中所有其他元素与图 8-2 相同。

236
|
237

图 8-14 给出了图 8-13 所示 ZAGI 零售公司销售部门数据库中的数据记录，其 ER 图及关系模式见图 8-13。关系 SALESTRANSACTION 中的列 TTime 已经填有相应数据。图中所有其他列与图 8-3 相同。

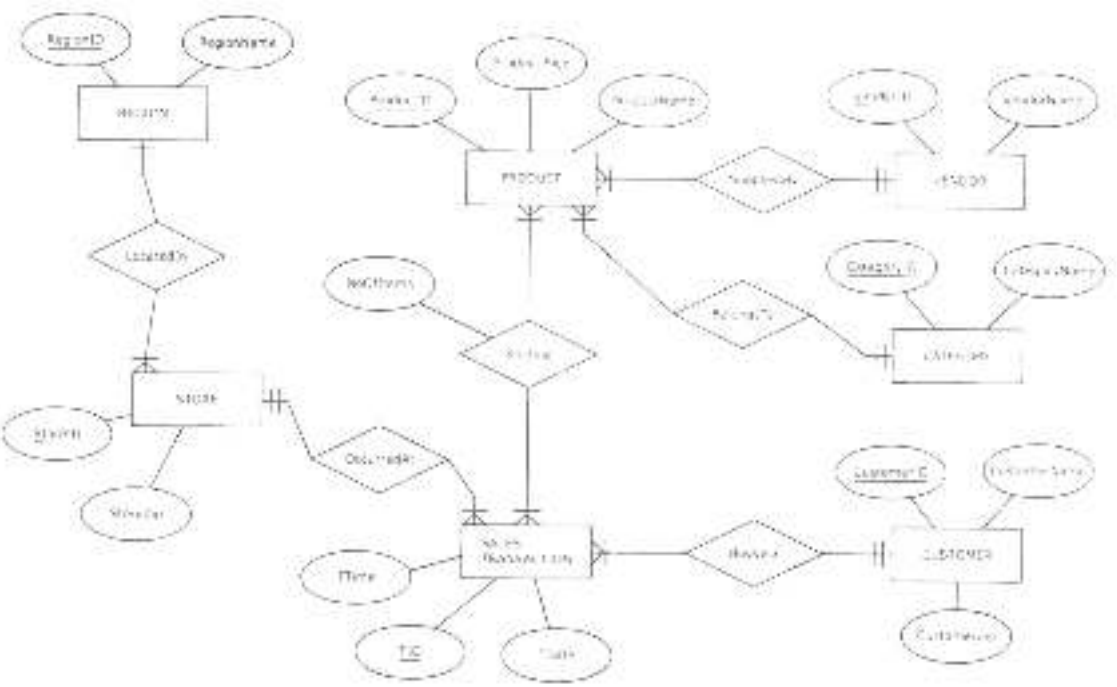


图 8-13　ZAGI 零售公司销售部门数据库：ER 图及结果关系模型（TTime 属性已添加）

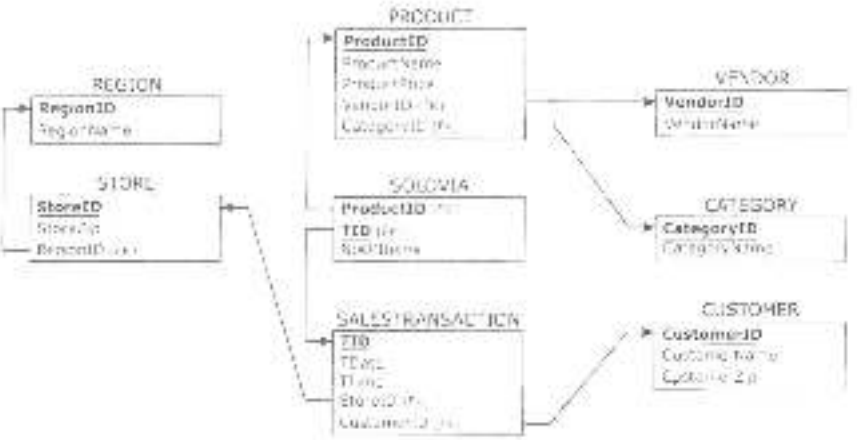


图 8-13（续）

REGION

RegionID	RegionName
C	Chicagoland
T	Tristate

PRODUCT

ProductID	ProductName	ProductPrice	VendorID	CategoryID
1X1	Zig Bag	$100	PG	OP
2X2	Easy Boot	$70	MK	PW
3X3	Easy Sons	$15	MK	PW
4X4	Cam boot	$80	PG	PW
5X5	Tiny Twin	$150	MK	OP
6X6	Bigger Twin	$260	MK	OP

VENDOR

VendorID	VendorName
PG	Pacifica Gear
MK	Mountain King

STORE

StoreID	StoreZip	RegionID
S1	60560	C
S2	60505	C
S3	95400	T

CATEGORY

CategoryID	CategoryName
OP	Camping
PW	Footwear

SALESTRANSACTION

TID	CustomerID	StoreID	TDate	TTime
1111	1-2-333	S1	1 Jan 2013	8:23:50 AM
2222	2-3-444	S1	1 Jan 2013	8:24:30 AM
7333	1-2-333	S3	2 Jan 2013	8:15:00 AM
7444	3-4-555	S3	2 Jan 2013	8:20:00 AM
7555	2-3-444	S3	2 Jan 2013	8:30:00 AM

SOLDVIA

ProductID	TID	NoOfName
1X1	T111	1
2X2	T222	1
3X3	T333	2
1X1	T333	1
4X4	T444	1
2X2	T444	2
4X4	T555	4
5X5	T555	2
5X6	T555	1

CUSTOMER

CustomerID	CustomerName	CustomerZip
1-2-333	Tina	60137
2-3-444	Tony	60611
3-4-555	Pam	36401

图 8-14　图 8-13 所示 ZAGI 零售公司销售部门数据库中的数据记录

假设业务分析人员在分析 ZAGI 零售公司的销售主题时，希望考虑一天中的时间，这时可扩展图 8-11 中的维度模型以包含这一信息。

一种方法是将一天中的时间作为一个独立的维度，或者作为 CALENDAR 维度的一部分。对当天的分析仅需要相对粗糙的时间段（即早上、中午、下午、晚上、夜晚），这种方法十分合理也比较实用，原因是该方法既不会产生一个只包含少数记录的单独时间维度（即 5 条记录，分别为早上、中午、下午、晚上、夜晚），同时还可确保 CALENDAR 维度以相对较小的比例被扩展（即 CALENDAR 维度中每天为 5 条记录而不是 1 条记录）。然而，如果时间分析需要以秒来表示，单独的 CALENDAR 维度则有 86400 条记录（24 × 60 × 60），或者扩展的 CALENDAR 维度可能包含上亿的记录（86400 × 表示的天数）。此时单独的 TIME 维度和扩展的 CALENDAR 维度都不再可行，而将时间添加为事实表的一个新属性将会是一

种更为简单的方法，如图 8-15 所示，该方法同样允许任何与天相关的业务分析。

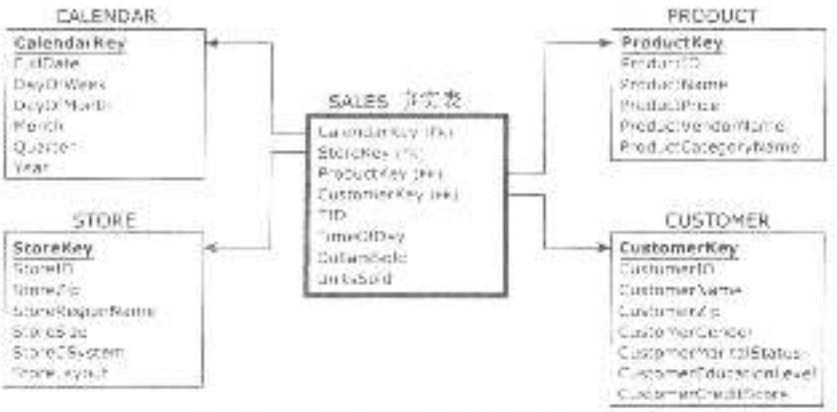


图 8-15　ZAGI 零售公司用于销售分析的分析型数据库相应的维度模型中，带有事务时间属性（TimeOfDay）的事实表

图 8-16 给出了图 8-15 中维度模型对应的填充数据表。

CALENDAR 维度

CalendarKey	FullDate	DayOf Week	DayOf Month	Month	Qtr	Year
1	1/1/2013	Tuesday	1	January	Q1	2013
2	1/2/2013	Wednesday	2	January	Q1	2013

PRODUCT 维度

ProductKey	ProductID	Product Name	Product Price	Product Vendor Name	Product Category Name
1	1X1	Zip Bag	$100	Pacifica Gear	Camping
2	2X2	Easy Boot	$70	Mountain King	Footwear
3	3X3	Cozy Sock	$10	Mountain King	Footwear
4	4X4	Duck Boot	$90	Pacifica Gear	Footwear
5	5X5	Tiny Tent	$100	Mountain King	Camping
6	6X6	Bigby Tent	$250	Mountain King	Camping

STORE 维度

StoreKey	StoreID	StoreZip	StoreRegion Name	Store Size (m²)	Store CSystem	Store Layout
1	S1	60500	Chicagoland	51000	Cashier	Modern
2	S2	60505	Chicagoland	35000	Self Service	Traditional
3	S3	35400	Tuscaloosa	55000	Mixed	Traditional

CUSTOMER 维度

CustomerKey	CustomerID	Customer Name	Customer Zip	Customer Gender	Customer MaritalStatus	Customer EducationLevel	Customer CreditScore
1	1-9-333	Tiny	60137	Female	Single	College	700
2	2-8-444	Tiny	60611	Male	Single	High School	650
3	3-4-555	Pam	35401	Female	Married	College	623

SALES 事实表

CalendarKey	StoreKey	ProductKey	CustomerKey	TID	TimeOfDay	DollarsSold	UnitsSold
1	1	1	1	T111	8:23:50 AM	$100	1
1	2	2	2	T222	8:24:30 AM	$70	1
2	2	3	1	T333	8:15:08 AM	$70	7
2	3	1	1	T333	8:15:08 AM	$100	1
2	3	4	3	T444	8:20:33 AM	$90	1
2	3	2	3	T444	8:20:33 AM	$140	2
2	3	4	2	T555	8:30:00 AM	$360	4
2	3	5	2	T555	8:30:00 AM	$800	2
2	3	6	2	T555	8:30:00 AM	$250	1

图 8-16　填有数据的事实表，包含事务标识码以及日期的数据

注意，事实表中可以包含事务标识码的列以及一天中的时间的列。这样的想法可以应用到任何一种实际的事件处理场景中，如购买事务、顺序、找零、票据、账单等，以此为事实表提供基础。

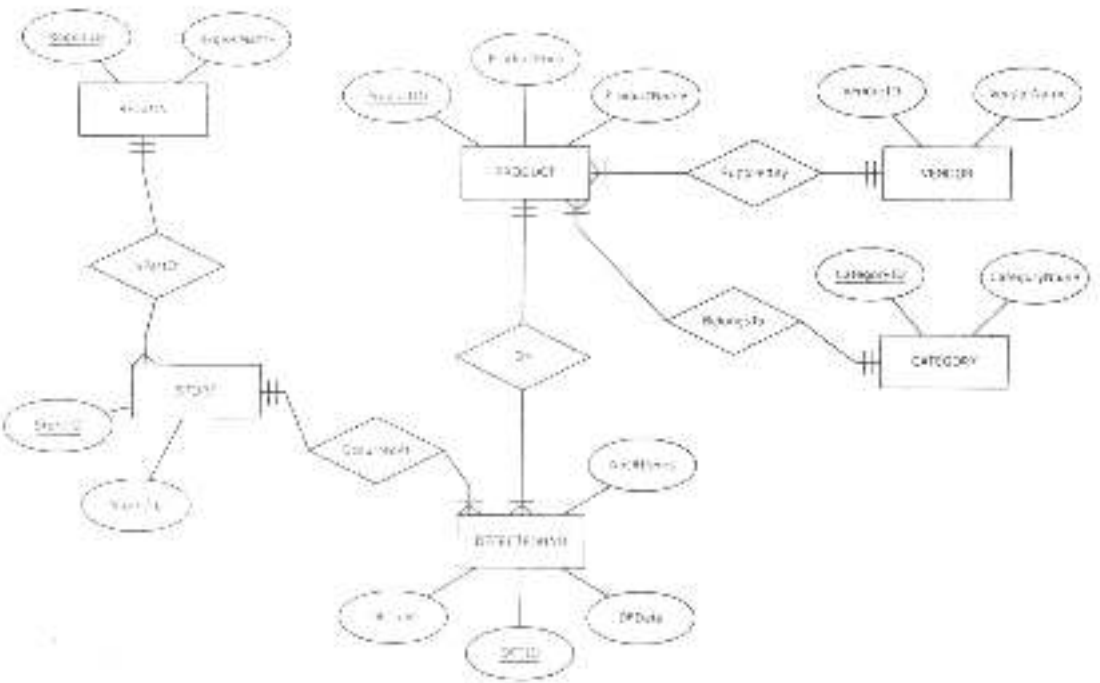
238
239

8.9　一个维度模型中的多个事实表

一个维度模型可以包含多个事实表，该情况出现在多个分析主题共享维度的时候。下面的例子将采用ZAGI零售公司的场景来说明这个概念。

在ZAGI零售公司中，除了销售部门需要跟踪销售记录外，质量监管部门需要跟踪商店中的劣质产品。质量监管部门定期检查商店的所有货架，发现劣质产品则将该产品从货架上移除。劣质产品的存货发现与移除过程，质量监管部门工作人员都会直接将时间、日期、劣质类型、产品信息记录到ZAGI零售公司质量监管部门的数据库中。

图8-17给出了ZAGI零售公司质量监管部门的ER图以及得到的关系模式。



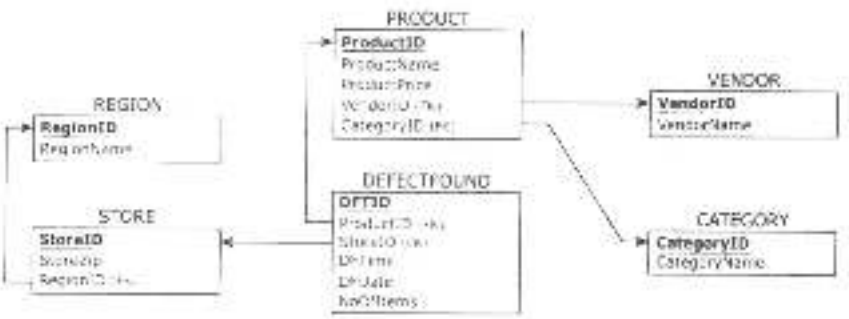


图8-17　ZAGI零售公司质量监管部门数据库的ER模型和关系模式

图8-18给出了ZAGI零售公司质量监管部门数据库的记录。该数据库的ER图与关系模式见图8-17。

REGION

RegionID	RegionName
[illegible]	[illegible]
[illegible]	[illegible]

STORE

StoreID	StoreZip	RegionID
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

PRODUCT

ProductID	ProductName	ProductPrice	VendorID	CategoryID
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

VENDOR

VendorID	VendorName
[illegible]	[illegible]
[illegible]	[illegible]

CATEGORY

CategoryID	CategoryName
CP	Camping
FW	Footwear

DEFECTEDUNIT

DFTID	ProductID	StoreID	DFTDate	DFTTime	NoOfItems
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

图8-18 ZAGI零售公司质量监管部门数据库的数据记录

在ZAGI零售公司中，质量监管部门数据库的部分信息与图8-13、图8-14所示的销售部门数据库重复。例如，两个数据库都包含了商店与产品的信息。正如本书第7章所述，组织机构拥有多个存在重复信息的单独操作型数据库，这种现象十分常见。 240

ZAGI零售公司希望用与分析销售额同样的方法分析劣质产品。公司决定创建一个维度模型来分析在商店中出现的劣质产品。分析劣质产品所需要的维度已经在分析销售的维度模型中创建完成。因此，ZAGI零售公司数据建模团队可以简单地创建另一个劣质产品的事实表，作为已有维度模型的一部分，而不是单独创建一个新的维度模型。如图8-19所示。 241

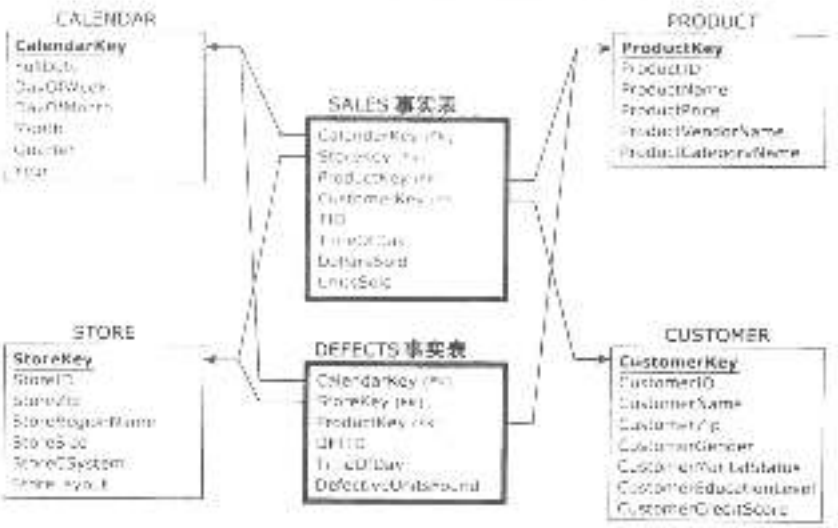


图8-19 拥有两个主题的扩展维度模型

图8-20给出了图8-19中维度模型对应的填充数据表。

有多个事实表的维度模型（如图8-19所示）又称为**星座/星系**（constellation/galaxy of stars）。该方法可以更快地开发用于多主题分析的分析型数据库，原因是对维度进行了重复利用而不是直接复制。同样，由于共享了维度，该方法还可以直接进行交叉事实分析，如比

较产品的日平均销售量与另一产品的日平均发现与移除量。比较对象可以是每个商店、区域、季度、产品类别等。

242

CALENDAR 维度

CalendarKey	FullDate	DayOf Week	DayOf Month	Month	Qtr	Year
1	1/1/2013	Tuesday	1	January	Q1	2013
2	1/2/2013	Wednesday	2	January	Q1	2013

PRODUCT 维度

ProductKey	ProductID	Product Name	Product Price	Product Vendor Name	Product Category Name
1	141	Zip Bag	$100	Pacifica Gear	Camping
2	242	Easy Boot	$70	Mountain King	Footwear
3	343	Cozy Sock	$15	Mountain King	Footwear
4	444	Cozy Boot	$80	Pacifica Gear	Footwear
5	555	Tiny Tent	$150	Mountain King	Camping
6	555	Giant Tent	$550	Mountain King	Camping

STORE 维度

StoreKey	StoreID	StoreZip	StoreRegion Name	Store Size (sq. ft.)	Store OSystem	Store Layout
1	S1	60600	Chicagoland	51000	Custom	Modern
2	S2	60605	Chicagoland	35000	Set Service	Traditional
3	S3	35401	Tuscaloosa	60000	Mixed	Traditional

CUSTOMER 维度

CustomerKey	CustomerID	Customer Name	Customer Zip	Customer Gender	Customer MaritalStatus	Customer EducationLevel	Customer CreditScore
1	1-2-333	Tina	60137	Female	Single	College	700
2	2-3-444	Tony	60611	Male	Single	High School	650
3	3-4-555	Pam	35401	Female	Married	College	675

SALES 事实表

CalendarKey	StoreKey	ProductKey	CustomerKey	TID	TimeOfDay	DollarSold	UnitsSold
1	1	1	1	T111	8:20:50 AM	$100	1
1	2	2	2	T222	8:34:30 AM	$70	1
2	3	3	1	T333	8:15:08 AM	$75	5
2	3	1	1	T333	8:15:08 AM	$120	1
2	3	4	3	T444	8:20:39 AM	$80	1
2	3	5	3	T444	8:20:39 AM	$140	2
2	3	4	2	T555	8:30:00 AM	$360	4
2	3	3	2	T555	8:30:00 AM	$300	2
2	3	4	2	T555	8:30:00 AM	$320	1

DEFECTS 事实表

CalendarKey	StoreKey	ProductKey	DFTID	TimeOfDay	DefectiveUnitsFound
1	1	1	DFT101	8:00:05 AM	1
1	2	2	DFT202	8:30:05 AM	2
2	3	3	DFT303	8:45:05 AM	6

图 8-20　图 8-19 中维度模型的记录

8.10　细节事实表与聚集事实表

维度模型中的事实表既可以包含**细节数据**（detailed data）也可以包含**聚集数据**（aggregated data）。在**细节事实表**（detailed fact tables）中，每条记录代表一个单一事实；而在**聚集事实表**（aggregated fact tables）中，每条记录概括了多个事实。下面的例子说明了两种事实表中数据的区别。为解释这些概念，我们将进一步扩展图 8-14 所示 ZAGI 零售公司销售部门操作型数据库的数据集，扩展后的数据集见图 8-21。

与图 8-14 中的数据集相比，图 8-21 中的数据集进行了扩展：在 SALESTANSACTION 表中添加了一条记录，在 SOLDVIA 表中添加了两条记录。在扩展之后的情形中，早上

8:30:00 已经在 S3 商店购买了两个 Tiny 帐篷以及一个 Biggy 帐篷的顾客 Tony，在一个小时后的 9:30:00 又返回到同一家商店购买了同样的商品。对数据集稍作扩展可以帮助我们更好地说明下面例子中的细节星形模式和聚集星形模式之间的区别。图 8-21 中的数据集将与图 8-6 和图 8-7 所示的数据集一起作为下面例子的数据源。

REGION

RegionID	RegionName
[illegible]	[illegible]
[illegible]	[illegible]

STORE

StoreID	StoreZip	RegionID
[illegible]	60606	[illegible]
[illegible]	60606	[illegible]
[illegible]	95400	[illegible]

PRODUCT

ProductID	ProductName	ProductPrice	VendorID	CategoryID
[illegible]	Tiny Bag	$100	[illegible]	[illegible]
[illegible]	Tiny Tent	$70	[illegible]	[illegible]
[illegible]	Cozy Sleep	$15	[illegible]	[illegible]
[illegible]	[illegible]	$80	[illegible]	[illegible]
[illegible]	Tiny Tent	$ 50	[illegible]	[illegible]
[illegible]	Biggy Tent	$250	[illegible]	[illegible]

VENDOR

VendorID	VendorName
[illegible]	Patrick Gear
[illegible]	Mountain King

CATEGORY

CategoryID	CategoryName
[illegible]	Camping
[illegible]	Footwear

SALESTRANSACTION

TID	CustomerID	StoreID	TDate	TTime
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

SOLDVIA

ProductID	TID	NoOfItems
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

CUSTOMER

CustomerID	CustomerName	CustomerZip
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

图 8-21 有两个主题的扩展维度模型

8.10.1 细节事实表

图 8-22 给出了一个包含销售主题细节事实表的维度模型。图 8-23 给出了图 8-22 所示的维度模型对应的数据填充表。

在图 8-21 所示的 ZAGI 零售公司销售部门数据库中，一个销售事实由 SOLDVIA 表中的一条记录表示。由于图 8-23 中的 SALES 表是细节级的，所以表中的每条记录是指向图 8-21 所示 SOLDVIA 表中的一条记录所表示的一个销售事实。因此，表 SALES 与表 SOLDVIA 都有 11 行。为了强调图 8-22 和图 8-23 中 SALES 事实表的行所表示的内容，我们对其主码进行了明确标记。主码由 TID 和 ProductKey 两个列组成，这是因为事实表中的每条记录表示一个特定销售事务中的特定行。由于同一个事务可以购买多个产品，因此表中的 TID 并不唯一。类似地，由于同一个产品可以与多个事务有关，因此表中的 ProductKey 也不唯一。因而需要 TID 与 ProductKey 一起组成主码。 243

8.10.2 聚集事实表

相应地，图 8-24 给出了包含销售主题的聚集事实表的一个维度模型。图 8-25 给出了图 8-24 所示维度模型对应的数据填充表。

图 8-22 带有细节事实表的维度模型

CALENDAR 维度

CalendarKey	FullDate	DayOf Week	DayOf Month	Month	Qtr	Year
1	1/1/2013	Tuesday	1	January	Q1	2013
2	1/2/2013	Wednesday	2	January	Q1	2013

PRODUCT 维度

ProductKey	ProductID	Product Name	Product Price	Product Vendor Name	Product Category Name
1	1X1	Zoo Bag	$100	Pacifica Gear	Camping
2	2X2	Day Boot	$70	Mountain King	Footwear
3	3X3	Daisy Sock	$15	Mountain King	Footwear
4	4X4	Dura Boot	$90	Pacifica Gear	Footwear
5	5X5	Tiny Tent	$150	Mountain King	Camping
6	6X6	Biggy Tent	$250	Mountain King	Camping

STORE 维度

StoreKey	StoreID	StoreZip	StoreRegion Name	Store Size (m^2)	Store CSystem	Store Layout
1	S1	60500	Chicagoland	51000	Cashiers	Modern
2	S2	60505	Chicagoland	35000	Self Service	Traditional
3	S3	35400	Trolyte	55000	Mixed	Traditional

CUSTOMER 维度

CustomerKey	CustomerID	Customer Name	Customer Zip	Customer Gender	Customer MaritalStatus	Customer EducationLevel	Customer CreditScore
1	1-2-939	Tony	60137	Female	Single	College	700
2	1-3-444	Tony	60611	Male	Single	High School	650
3	1-4-555	Dan	35401	Female	Married	College	633

SALES 事实表

CalendarKey	StoreKey	ProductKey	CustomerKey	TID	TimeOfDay	DollarsSold	UnitsSold
1	1	1	1	T111	8:23:59 AM	$100	1
1	2	2	2	T222	8:24:30 AM	$70	1
2	3	3	1	T333	8:15:05 AM	$75	5
2	3	1	1	T333	8:15:06 AM	$100	1
2	3	4	3	T444	8:20:35 AM	$90	1
2	3	2	2	T444	8:20:35 AM	$140	2
2	3	4	3	T555	8:30:00 AM	$360	4
2	3	5	2	T555	8:30:00 AM	$300	2
2	3	5	2	T555	8:30:00 AM	$250	1
2	3	6	3	T666	9:30:00 AM	$300	2
2	3	6	2	T666	9:30:00 AM	$250	1

图 8-23 图 8-22 对应的填充数据的维度模型

图 8-24　带有聚集销售事实表 SalesPerDPCS
（每天、每个产品、每个顾客和每个商店）的维度模型

CALENDAR 维度

CalendarKey	FullDate	DayOf Week	DayOf Month	Month	Qtr	Year
1	[illegible]	[illegible]	1	[illegible]	Q1	[illegible]
2	[illegible]	[illegible]	2	[illegible]	Q1	[illegible]

PRODUCT 维度

ProductKey	ProductID	Product Name	Product Price	Product Vendor Name	Product Category Name
1	[illegible]	Zip Dip	$100	Phoenix Gear	Camping
2	[illegible]	Easy Boot	$90	Mountain King	Footwear
3	[illegible]	Cozy Sock	$10	Mountain King	Footwear
4	[illegible]	Duck Boot	$80	Phoenix Gear	Footwear
5	[illegible]	Tent Tent	$150	Mountain King	Camping
6	[illegible]	Fuzzy Tent	$200	Mountain King	Camping

STORE 维度

StoreKey	StoreID	StoreZip	StoreRegion Name	Store Size (sf)	Store CSystem	Store Layout
1	S1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
2	S2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
3	S3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

CUSTOMER 维度

CustomerKey	CustomerID	Customer Name	Customer Zip	Customer Gender	Customer MaritalStatus	Customer EducationLevel	Customer CreditScore
1	[illegible]	[illegible]	[illegible]	Female	Single	College	700
2	[illegible]	[illegible]	[illegible]	Male	Single	High School	820
3	[illegible]	[illegible]	[illegible]	Female	Married	College	675

SALESPerDPCS 事实表

CalendarKey	StoreKey	ProductKey	CustomerKey	DollarsSold	UnitsSold
1	1	1	1	$100	1
1	2	2	2	$90	1
2	3	3	1	$75	5
2	3	1	1	$100	1
2	3	4	3	$80	1
2	3	2	3	$180	2
2	3	4	2	$360	4
3	3	5	2	$600	4
3	3	6	2	$800	2

图 8-21 的 SALES 事实表中的 8 条记录中 10 条以上的点击相加，产品的数量

图 8-23 的 SALES 事实表中的 9 条记录中 11 条记录汇总相加，后的数量

图 8-25　图 8-24 对应的填充数据的维度模型

为了强调图 8-24 和图 8-25 中 SALES 事实表的聚集特性，我们明确地标识了其主码。主码包含 CalendarKey、StoreKey、ProductKey、CustomerKey 四个列，这是因为 SALES 事实表中的每条记录表示一个汇总结果。该结果代表某天某个顾客在某个商店购买某种产品的总金额和单位。

如前所述，图 8-23 的细节 SALES 事实表有 11 条记录，与图 8-21 中 SOLDVIA 表中的记录数相同。这是因为两个表中的每条记录都代表一个事实。而另一方面，由于图 8-25 中的 SalesPerDPCS 表是一个聚集类型的事实表，因而它只有 9 条记录。特别地：

- 图 8-25 中聚集 SalesPerDPCS 事实表中的第 8 条记录汇总了（相加在一起）图 8-23 中 SALES 事实表中的第 8 条与第 10 条记录，也就是图 8-21 中 SOLDVIA 表中的第 8 条和第 10 条记录。
- 图 8-25 中聚集 SalesPerDPCS 事实表中的第 9 条记录汇总了（相加在一起）图 8-23 中 SALES 事实表中的第 9 条与第 11 条记录，也就是图 8-21 中 SOLDVIA 表中的第 9 条和第 11 条记录。

该聚集过程出现的原因是 TID 值没有被包含在聚集 SalesPerDPCS 事实表中。图 8-25 中聚集 SalesPerDPCS 事实表的前 7 条记录由一张单独的事实表汇总得到，因此这些值与图 8-23 中 SALES 事实表的前 7 条记录是相同的，与图 8-21 中 SOLDVIA 表的前 7 条记录也是相同的。这是由于在这 7 种情况中，某一天中某顾客在某个商店只购买了一个产品。第 8 条、第 10 条以及第 9 条、第 11 条记录源自两个事务，在这两个事务中，同一天中同一个顾客在同一个商店中购买了两个相同的产品。

244 ~ 245

将图 8-21 的数据源中的数据进行聚集有多种方式，上面的例子便是其中一种。图 8-26 与图 8-27 给出了另一个例子。图 8-26 中维度模型聚集的数据与图 8-24 聚集的数据相同，只是所用方式不同（图 8-26 的方式更精简）。

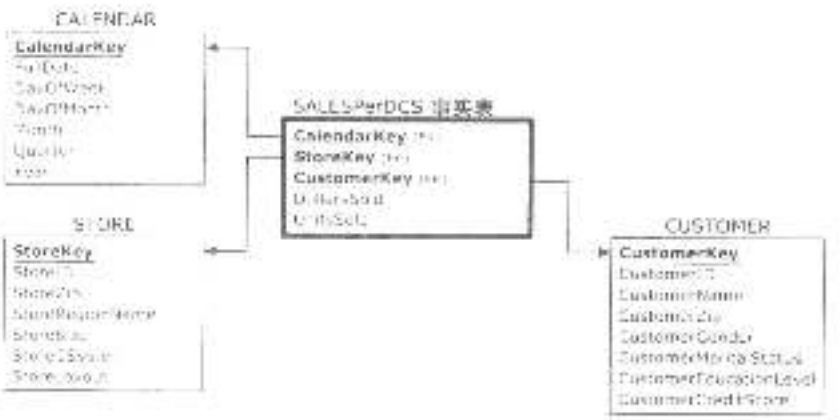


图 8-26　带有聚集事实表 SalesPerDCS（每天、每个顾客和每个商店）的维度模型

图 8-27 给出了图 8-26 的维度模型对应的数据填充表。

在图 8-26 与图 8-27 中，聚集事实表 SALES 的主码由 CalendarKey、StoreKey 以及 CustomerKey 几个列组成。表 SALES 中没有包含 ProductKey，因为表中包含了所有产品的聚集。聚集的 SALES 事实表中的每条记录是一个汇总，代表了某天当中某个顾客在某商店购买某产品的总金额与单位。

246

CALENDAR 维度

CalendarKey	FullDate	DayOf Week	DayOf Month	Month	Qtr	Year
1	1/1/2013	Tuesday	1	January	Q1	2013
2	1/2/2013	Wednesday	2	January	Q1	2013

STORE 维度

StoreKey	StoreID	StoreZip	StoreRegion Name	Store Size	Store CSystem	Store Layout
1	S1	60608	Chicagoland	54000	Customer	Modern
2	S2	60605	Chicagoland	55000	Self Service	Traditional
3	S3	30430	Indiana	55000	Mixed	Traditional

CUSTOMER 维度

CustomerKey	CustomerID	Customer Name	Customer Zip	Customer Gender	Customer MaritalStatus	Customer EducationLevel	Customer CreditScore
1	1-2-333	Lisa	60101	Female	Single	College	700
2	2-3-444	Tony	60611	Male	Single	High School	650
3	3-4-555	Pam	30401	Female	Married	College	623

SALESPerDCS 事实表

CalendarKey	StoreKey	CustomerKey	DollarsSold	UnitsSold
1	1	1	$100	1
1	2	2	$70	1
2	3	1	$175	6
2	3	3	$230	3
2	3	2	$1,480	10

图8-23的SALES事实表中第3条和第4条记录汇总（相加）后的数量

图8-23的SALES事实表中第5条和第6条记录汇总（相加）后的数量

图8-23的SALES表中第7条到第11条记录汇总（相加）后的数量

图8-27 图8-26对应的填充数据的维度模型

图8-27所示的聚集事实表SalesPerDCS有5条记录，最下面的3条记录包含了下面的汇总信息：

- SalesPerDCS的第3条记录汇总了图8-23中SALES事实表的第3条记录与第4条记录，即图8-21中SOLDVIA表中的第3条与第4条记录。
- SalesPerDCS的第4条记录汇总了图8-23中SALES事实表的第5条记录与第6条记录，即图8-21中SOLDVIA表中的第5条与第6条记录。
- SalesPerDCS的第5条记录汇总了图8-23中SALES事实表的第7条记录与第11条记录，即图8-21中SOLDVIA表中的第7条与第11条记录。

247

图8-27中聚集的SalesPerDCS事实表的前2条记录由一张单独的事实表汇总得到，因此该表与图8-23中SALES事实表的前2条记录取值相同，与图8-21中SOLDVIA表的前2条记录也相同。这是由于这两条记录都是表示同一天中同一个顾客在同一家商店只购买了一个产品的情况。若同一天中同一个顾客在同一个商店购买了两个或两个以上的产品，购买记录就会被聚集到图8-27所示的SalesPerDCS事实表中。

8.10.3 细节事实表与聚集事实表的其他实例

为了更要重述细节事实表与聚集事实表的概念，我们再来看看本章维度模型的其他事实表，并将这些表区分为细节型与聚集型。

在图 8-4、图 8-5 以及图 8-8、图 8-9 给出的例子中，事实表 SALES 是聚集型事实表，其主码由 CelandarKey、StoreKey、ProductKey 以及 CustomerKey 组成。在这些例子中，事实表 SALES 没有包含 TID，表明事实表是聚集型的，SALES 事实表与图 8-3 中的 SOLDVIA 表有相同的记录数目。然而，如果情形如同图 8-25 所示，即同一天当中同一个顾客在同一个商店购买了两次同样的产品，这时候就不属于聚集型事实表了。

在图 8-11、图 8-12 和图 8-15、图 8-16 以及图 8-19、图 8-20 给出的例子中，事实表 SALES 是细节类型的事实表，其主码由 ProductKey 和 TID 组成。

8.11　事实表的粒度

事实表的粒度（granularity）刻画的是事实表中每一行描述的信息。细节事实表具有较好的粒度，因为每条记录代表一个单独的事实。比起细节事实表，聚集事实表的粒度比较粗糙，这是由于聚集事实表通常表示多个事实的汇总。

例如，如前文所述，图 8-24 与图 8-25 中的 SalesPerDPCS 事实表比图 8-22 与图 8-23 中的 SALES 事实表粒度更粗糙，因为 SalesPerDPCS 事实表中的记录是对图 8-22、图 8-23 事实表 SALES 中记录的汇总。图 8-26、图 8-27 的事实表 SalesPerDCS 的粒度更为粗糙，因为它的记录是从图 8-24、图 8-25 的事实表 SalesPerDPCS 中汇总得到的。例如，SalesPerDPCS 的第 3 条记录汇总了图 8-23 中 SALES 事实表的第 3 条与第 4 条记录，同样也是图 8-24、图 8-25 中 SalesPerDPCS 事实表的第 3 条记录与第 4 条记录。

基于其聚密程度，粗糙粒度聚集的事实表查询起来比细节事实表更快。然而，粗粒度的表将受到其所能检索到信息的限制。聚集与需求相关，较细粒度使得分析相对不受限制。换
[248] 言之，用户往往可以从最细的粒度获得一个聚集，相反则不成立。

一种既能利用聚集事实表的查询性能又能保留细节事实表的分析优势的方法是，让两种类型的表同时存在于一个维度模型中，即在同一个星座模型中。例如，图 8-22、图 8-24 以及图 8-26 所示的图形，可以成为同一个模式中的一个部分，如图 8-28 所示。

如果一个用户需要分析某天中某顾客在某商店购买某产品的每个汇总销售数据，或某天中某顾客在某商店的每个汇总销售数据，用户可以快速查询聚集事实表 SALESPerDPCS 以及 SALESPerDCS，同时也可以从细节事实表 SALES-DETAILED 中进行任何其他类型的销售分析，虽然速度比较慢。

条目级细节事实表与事务级细节事实表

根据其基本数据源所描述的内容，细节事实表可以表示不同类型的信息。最常见的两种细节事实表是条目级细节事实表与事务级细节事实表。

在条目级细节事实表（line-item detailed fact table）中，每行表示一个事务中的一个条目行。图 8-22 与图 8-23 所示的 SALES 表就是条目事实表，因为每行表示一个销售事务中的一个条目行。其基本数据源（ZAGI 零售公司销售部门数据库）中关于销售主题的最细粒度事实表由一个销售事务中的一个条目行表示。

在事务级细节事实表（transaction-level detailed fact table）中，每行表示一个事务。考虑 ZippyZoom 汽车租赁商的例子，每个租赁事务在一个部门中进行，与一辆车和一个顾客相关。在 ZippyZoom 汽车租赁商的操作型数据库中，除了记录租赁日期、被租汽车、租车
[249] 顾客外，还可能记录唯一的 RentalID、租赁总量。基于该操作型数据库，ZippyZoom 汽车租

管理可以建立一个维度模型数据库来分析租金收益，如图8-29所示。

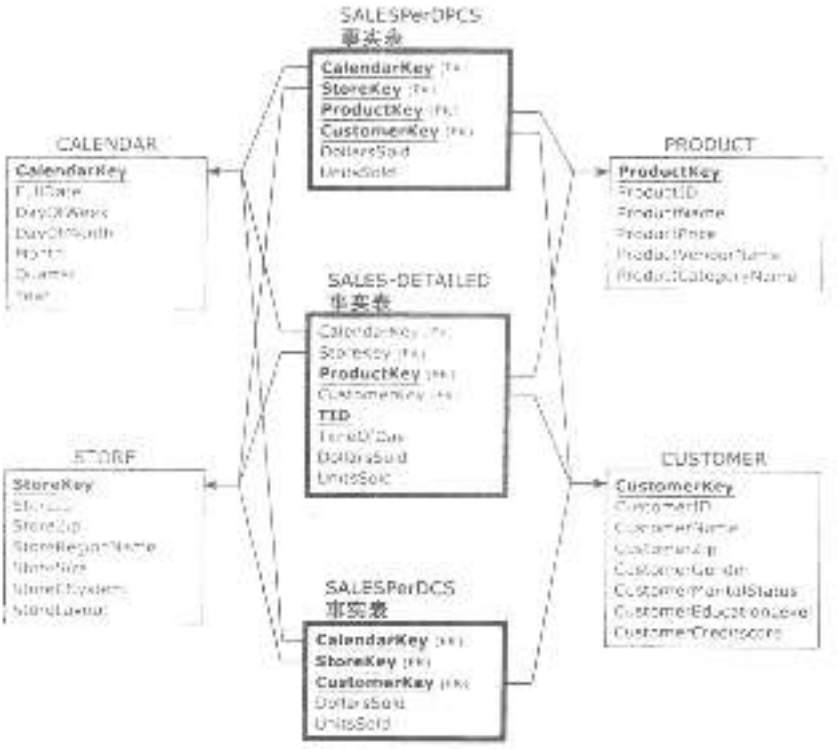


图8-28 一个细节和聚集事实表的星座模型

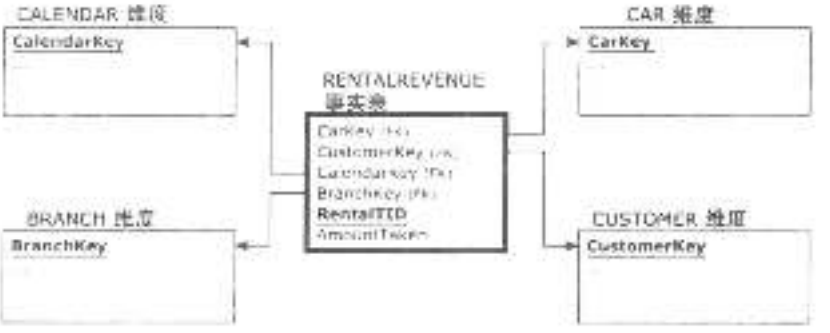


图8-29 一个包含事务级事实表的维度模型

图8-29所示的维度模型包括四个维度（CALENDAR、CAR、CUSTOMER以及BRANCH）及事实表RENTALREVENUE。事实表RENTALREVENUE是一个事务级的细节事实表。其基本数据源（ZippyZoom汽车租赁商操作型数据库）中关于租金收益主题的最细粒度事实表由每个租赁事务表示。一个租赁事务没有条目行，因为它只包含一辆车。因此，该事务本身就是租金收益主题事实表所能表示的最低细节级别。

8.12 缓慢变化维度与时间戳

一个星形模式中的典型维度要么包含值不发生改变（或基本很少改变）的属性，如商店

大小及顾客性别。要么包含随时间偶发性变化的属性，如顾客邮编及新水。包含可变属性的维度通常叫做缓慢变化维度（slowly changing dimension）。有几种不同的方法可用于处理缓慢变化维度，最常用的为 Type1、Type2 及 Type3。

8.12.1　Type1 方法

Type1 方法是一种最简单的方法，它常用于维度中由于错误而造成的属性值变化。Type1 方法简单地改变维度记录中的值，即用新值取代旧值。若使用了 Type1 方法，则不会保存任何历史信息。下面这个简单的例子说明了 Type1 方法。

250　假设下表是一个缓慢变化维度：

CUSTOMER

CustomerKey	CustomerID	CustomerName	TaxBracket
1	111	Linda	Low
2	222	Susan	Medium
3	333	William	High

假设 Susan 的 TaxBracket 属性值需要从 Medium 变为 High。若使用 Type1 方法，结果如下：

CUSTOMER

CustomerKey	CustomerID	CustomerName	TaxBracket
1	111	Linda	Low
2	222	Susan	High
3	333	William	High

第 2 行中的 TaxBracket 属性的旧值（Medium）被简单地用新值（High）替换。Type1 方法常适用于以下情形：若 Susan 的 TaxBracket 开始取值为 High，但被错误地记录为 Medium，改变之后使其正确化。

8.12.2　Type2 方法

Type2 方法用于需要保存历史信息的情况。每次维度中记录值发生变化时，Type2 方法使用新的值为代理码创建一个额外的维度记录。这里继续用说明 Type1 方法的例子说明 Type2。

假设下表是一个缓慢变化维度：

CUSTOMER

CustomerKey	CustomerID	CustomerName	TaxBracket
1	111	Linda	Low
2	222	Susan	Medium
3	333	William	High

假设 Susan 的 TaxBracket 属性值需要从 Medium 变为 High。若使用 Type2 方法，结果如下：

CUSTOMER

CustomerKey	CustomerID	CustomerName	TaxBracket
1	111	Linda	Low
2	222	Susan	Medium
3	333	William	High
4	222	Susan	High

一个包含 Susan 的 TaxBracket 属性新值（High）的新记录被创建，同时包含 Susan 的 TaxBracket 属性旧值的旧记录依然保留。Type2 方法适用于以下情形：Susan 的 Tax Bracket 开始确实为 Medium，后来变为 High。 [251]

注意 Type2 方法中代理码的重要作用。由于出现了具有相同操作码的多条记录（记录 2 与记录 4 中相同的 CustomerID），因而代理码才是主码，而原始操作码现在则用于连接代表相同实际对象的多个记录。如果我们想分析顾客 Susan 的购买记录，可以看到所有与 CustomerID 为 222 的维度记录相关的销售事实记录。然而，若希望观察 TaxBracket 为 High 的顾客的购买模式，我们可以将与 Customer 维度中第 3 条记录相关的事实表中的销售记录以及与 Customer 维度中第 4 条记录相关的销售记录结合起来。换言之，只能保留由 Susan 产生的购买记录，因为只有 Susan 的 TaxBracket 为 High。

Type2 方法是最常用的处理缓慢变化维度的方法。该方法可以直接处理维度属性改变较多的情况。

Type2 方法通常要结合维度中的附加列使用，即**时间戳**（timestamp）。时间戳表明了记录的值可以应用的时间段。时间戳列表明了因维度记录中值的改变而向每个列输入的起始和结束日期。另一个名为**行指示符**（row indicator）的列也可以添加到整个表中，该列的角色是提供一个快速标识，用以指示当前哪些记录是有效的。假设顾客表包含了自 2000 年 1 月 1 日起的数据，并假设 2008 年 1 月 1 日时 Susan 的 TaxBracket 值变为了 High，则下面的例子用 Type2 方法演示了时间戳以及行指示器的使用：

CUSTOMER

CustomerKey	CustomerID	CustomerName	TaxBracket	Effective StartDate	Effective EndDate	Row Indicator
1	111	Linda	Low	1.1.2000	n/a	Current
2	222	Susan	Medium	1.1.2000	12.31.2007	Not Current
3	333	William	High	1.1.2000	n/a	Current
4	222	Susan	High	1.1.2008	n/a	Current

EffectiveStartDate 以及 EffectiveEndDate 指示每行值的有效时间区间。“n/a”（not applicable）表明该行的值依然有效，目前没有终止时间。在实际实施过程中，“n/a”往往会记录为一个更远的日期来简化一些时间相关的查询。如用 9999 年 12 月 31 日代替“n/a”。行指示器能快速从历史数据行的所有列中区分出包含有效数据的行。

8.12.3　Type3 方法

Type3 方法可用于如下几种情况。一种情况是维度中每列可能发生改变的数量是确定的，另一种情况是只列有限的历史进行记录。Type3 方法会为维度表中每个发生改变的列创建一个“历史值”列以及一个“当前值”列。

这里同样用演示 Type1 与 Type2 的例子来说明 Type3。 [252]

假设下表是一张缓慢变化的维度表：

CUSTOMER

CustomerKey	CustomerID	CustomerName	TaxBracket
1	111	Linda	Low
2	222	Susan	Medium
3	333	William	High

假设 Susan 的 TaxBracket 属性值已经由 Medium 变为 High。若应用 Type3 方法，结果如下：

CUSTOMER

CustomerKey	CustomerID	CustomerName	Previous TaxBracket	Current TaxBracket
1	111	Linda	n/a	Low
2	222	Susan	Medium	High
3	333	William	n/a	High

有两个为 TaxBracket 属性创建的单独的列：一个为 TaxBracket 的当前值，一个为 TaxBracket 的历史值。Type3 方法适用于以下情形：顾客的 TaxBracket 只能改变一次，或在进行该组织机构的分析时只需要顾客 TaxBracket 属性的两个最近的值。

Type3 方法常与时间戳结合起来使用。下面是用 Type3 方法说明时间戳的例子：

CUSTOMER

CustomerKey	CustomerID	CustomerName	Previous TaxBracket	Previous TaxBracket EffectiveDate	Current TaxBracket	Current TaxBracket EffectiveDate
1	111	Linda	n/a	n/a	Low	1.1.2000
2	222	Susan	Medium	1.1.2000	High	1.1.2008
3	333	William	n/a	n/a	High	1.1.2000

处理缓慢变化维度时，最常用的选择是 Type1，Type2 或 Type3，或将这些方法结合使用，需要根据维度中多个属性可能发生改变的实际情况而定。

8.13 其他维度建模问题

253 我们简单地用与维度建模相关的几个其他问题来总结有关维度建模概念的讨论。

8.13.1 雪花模型

将星形模式中的维度设计为一个非规范的表，而表中的数据可以划分到若干规范表之间，这样的情况比较常见。例如，图 8-24 中的 PRODUCT 维度包含了有关产品、种类以及供应商的信息，这些信息可以被划分到 3 个规范表中：PROCUDT、CATEGORY 以及 VENDOR。若维度模型中的维度是规范的，这种模型称为**雪花模型**（snowflake model）。图 8-30 给出了图 8-24 中星形模式的一个雪花模型版本（即规范后的版本）。

在实际情况下，雪花模型通常并不用于维度建模中。维度建模中不使用雪花模型的原因之一在于，非规范（即非雪花模型）维度的目的是使分析过程更简单。而规范的星形模式在考察分析过程时将形成大量关系表，这增加了过程的复杂性。比较图 8-24 与图 8-30 的模式。若两种模式都已实现，它们最终都会包含相同的数据。然而，使用图 8-24 的分析人员仅利用 4 个维度便可进行销售相关的任务分析，而使用图 8-30 的分析人员进行同样的分析则需要 7 个维度。

维度建模过程中不使用雪花模型的原因之一在于，对于分析型数据库来说，规范化并不是必须的。回顾第 4 章的内容，对操作型数据库进行规范化的主要原因是为了避免更新异常。再回顾第 7 章的分析型数据库，如数据仓库和数据集市，它们并不会受到更新异常的影响，因为这些数据库是“附加且只读”的。由于维度建模主要用于设计不会被更新异常影响的分析型商务智能数据库，因而用于防止更新异常的规范化过程就不必要了。

图8-30 图8-24所示的维度模型的立花版本

8.13.2 立方体

关系模型并不是实现维度模型数据库的唯一方法，另一个方法是使用立方体。从概念上来说，使用关系模型或使用立方体实现的维度模型之间没有任何区别，概念设计依然与创建维度和事实表有关，而区别仅存在于物理实现过程中。第9章将给出立方体的概述，并说明维度模型的关系型实现与立方体实现之间的异同，以及相关的性能和容量问题。 254

8.14 数据仓库（数据集市）建模方法

现代数据建模方法为数据仓库和数据集市等分析型数据库的设计提供了若干选择。下面列出了最常见的三种方法：

- 规范化数据仓库。
- 维度建模数据仓库。
- 独立数据集市。

这些方法的区别在于如何利用本书前面章节介绍的数据建模技术。本章的余下部分将说明并演示这些技术的基本概念及其异同。

8.15 规范化数据仓库

建模数据仓库的方法之一，就是把数据仓库看作一个用ER建模以及关系型建模集成得到的分析型数据库，其结果就是一个规范化的关系型数据库模式。**规范化数据仓库**（normalized data warehouse）的数据来源有：操作型数据源中通过ETL过程得到的对分析有用的数据、提供维度建模数据集市的数据源以及其他无量纲的对分析有用的数据集。数据仓库作为一种规范化的集成分析型数据库，最早由Bill Inmon提出，因此规范化数据仓库方法通常指Inmon方法[3]。图8-31所示为一个规范化的数据仓库。 255

③ 可参考《Building the Data Warehouse》，第4版，W.H.Inmon（Wiley，2005）。

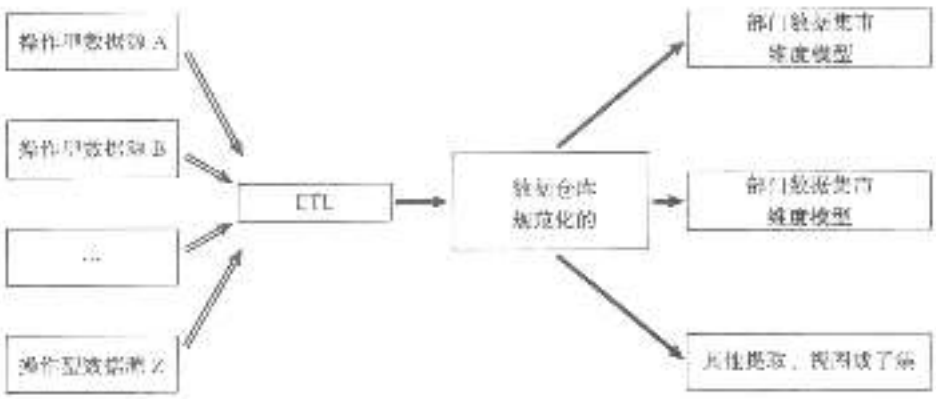


图 8-31　一个规范化的数据仓库

该方法的思路是将一个中心数据仓库建模为 ER 模型，因此可以实射为一个规范化的关系型数据库模型，规范化的关系型数据库则用作数据仓库的物理存储。所有基础的操作型数据源的集成都将发生在中心规范化数据库模式之中。一旦数据仓库已经构建完成且由基础源通过 ETL 架构得到的数据填充好，则基于这种完全集成的数据库，各种对分析有用的视图、子集以及提取等操作都是可实现的。规范化数据仓库得到的分析型数据集，其主要类型之一实际就是一种维度建模的数据集市，可以使用 OLAP/BI 工具（OLAP/BI 工具将在第 9 章中阐述）进行查询。这种由更大的数据仓库得到的数据集市，可用作视图（虚拟表）或者物理提取，该数据集市称为独立数据集市。

此外，当分析和决策支持需要用到无定纲数据集时，这种数据集也可以通过提取得到。例如，一些分析人员或数据挖掘工具等分析型应用（数据挖掘将在附录 G 中进行讨论）需要结合多个规范化数据仓库的数据来查询一个大的表。

8.16　规范化数据仓库实例

为了说明如何使用 ER 建模及关系型模式映射建模规范化数据仓库，我们再一次使用 ZAGI 零售公司的例子。

假设 ZAGI 零售公司需要使用 ER 建模技术为分析其销售额设计一个规范化数据仓库。设计该数据仓库的必要条件如下。

ZAGI 零售公司需要创建一个分析型数据仓库来分析销售额。

3 个可用数据源如下：

数据源 1：如图 8-13 和图 8-14 所示的 ZAGI 零售公司销售部门数据库。

数据源 2：如图 8-6 所示的 ZAGI 零售公司设备部门数据库。

数据源 3：如图 8-7 所示的顾客信息数据的额外表。

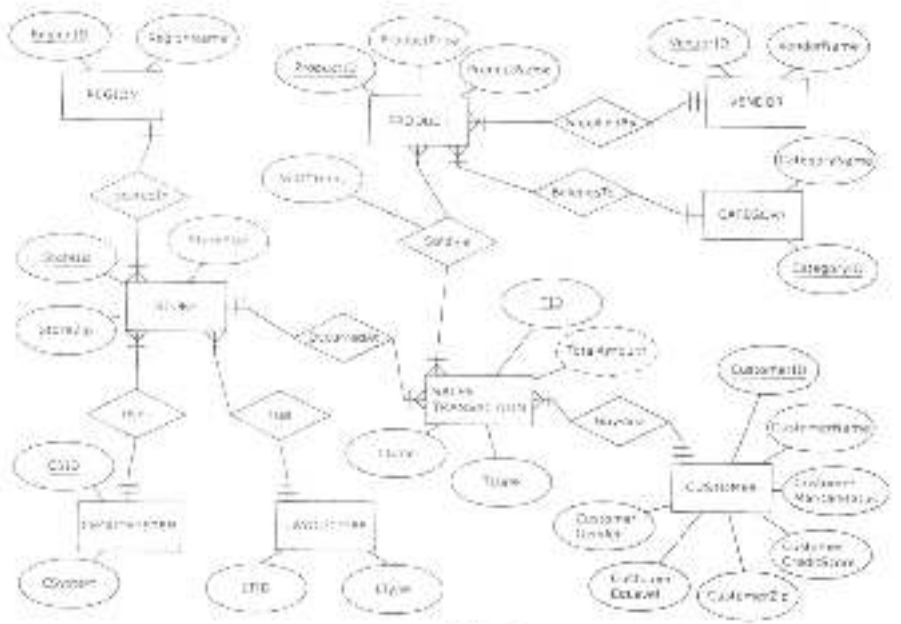
数据仓库需要能根据如下方面完成销售总额和销售总量的分析：

- 日期。
- 时间。
- 产品，包括：
  - 产品名称及价格；
  - 产品种类；

- 产品供货商
- 顾客，包括：
  - 顾客姓名、邮编、性别、婚姻状况、教育程度、信用评分
- 商店，包括：
  - 单个商店；
  - 商店规模和邮编；
  - 商店结账系统；
  - 商店布局；
  - 商店所在区域。

图 8-32 和图 8-33 演示了使用 ER 建模技术如何设计一个基于这些数据源和需求的销售分析数据仓库。

256



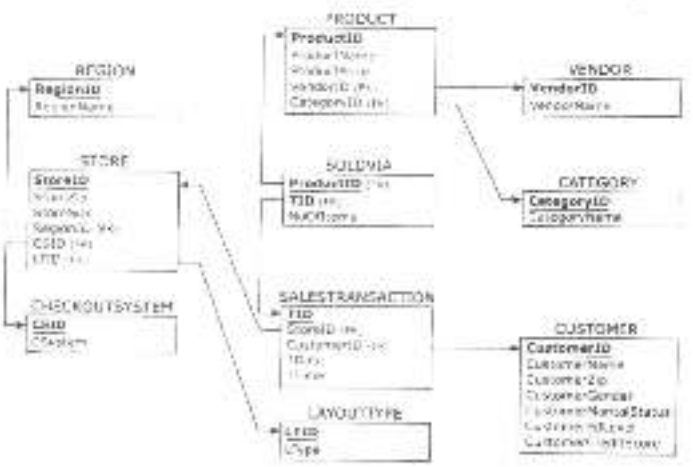


图 8-32　基于多个数据源建立的 ER 模型及关系型模式，用于 ZAGI 零售公司的销售额分析

PRODUCT

ProductID	ProductName	ProductPrice	VendorID	CategoryID
1X1	Zzz Bag	$100	PG	CP
2X2	Easy Boot	$70	MK	FW
3X3	Cosy Sock	$15	MK	FW
4X4	Dura Boot	$90	PG	FW
5X5	Tiny Tent	$150	MK	CP
6X6	Biggy Tent	$250	MK	CP

CATEGORY

CategoryID	CategoryName
CP	Camping
FW	Footwear

VENDOR

VendorID	VendorName
PG	Pacifica Gear
MK	Mountain King

STORE

StoreID	StoreZip	StoreSize (m²)	RegionID	CSID	LTID
S1	60600	51000	C	CR	M
S2	60605	35000	C	AC	T
S3	35400	55000	T	MX	T

REGION

RegionID	RegionName
C	Chicagoland
T	Tristate

LAYOUTTYPE

LTID	Layout
M	Modern
T	Traditional

SALESTRANSACTION

TID	CustomerID	StoreID	TDate	TTime
T111	1-2-333	S1	1-Jan-2013	8:23:59 AM
T222	2-3-444	S2	1-Jan-2013	8:24:30 AM
T333	1-2-333	S3	2-Jan-2013	8:15:08 AM
T444	3-4-555	S3	2-Jan-2013	8:20:33 AM
T555	2-3-444	S3	2-Jan-2013	8:30:00 AM

SOLDVIA

ProductID	TID	NoOfItems
1X1	T111	1
2X2	T222	1
3X3	T333	5
1X1	T333	1
4X4	T444	1
2X2	T444	2
4X4	T555	4
5X5	T555	2
6X6	T555	1

CHECKOUTSYSTEM

CSID	CSystem
CR	Cashiers
AC	Self Service
MX	Mixed

CUSTOMER

CustomerID	CustomerName	CustomerZip	Customer Gender	Customer Marital Status	Customer Education Level	Customer Credit Score
1-2-333	Tina	60137	Female	Single	College	700
2-3-444	Tony	60611	Male	Single	High School	650
3-4-555	Pam	35401	Female	Married	College	623

图 8-33　用 3 个源自数据源填充好的图 8-32 所示的关系模型

图 8-32 给出了通过合并数据源 1、2 的 ER 模型以及从数据源 3 中添加属性从而得到的 ER 模型。如需求所述，ER 模型提取了作为分析主题的销售总额和总量，以及所有跟销售相关的必需属性。图 8-32 还给出了与 ER 模型相对应的规范化关系模式。

227

注意图 8-32 ER 图中的几个数据源的合并。实体 STORE 在图 8-13（数据源 1）以及图 8-6（数据源 2）中都有出现。实体 CUSTOMER 在图 8-13（数据源 1）及图 8-7 的一张表（数据源 3）中都有出现。实体 REGION、PRODUCT、CATEGORY、VENDOR 以及 SALESTRANSACTION 只在图 8-13（数据源 1）中出现。实体 CHECKOUTSYSTEM 以及 LAYOUT 只在图 8-6（数据源 2）中出现。

实体 STORE 现在有 3 个属性：

- 商店编号，在图 8-13（数据源 1）的实体 STORE 和图 8-6（数据源 2）的实体 STORE 中出现。
- 商店邮编，只在图 8-13（数据源 1）的实体 STORE 中出现。
- 商店大小，只在图 8-6（数据源 2）的实体 STORE 中出现。

实体 CUSTOMER 现在有 7 个属性：

- 顾客编号和顾客姓名，在图 8-13（数据源 1）的实体 CUSTOMER 和图 8-7 的表 CUSTOMER（数据源 3）中出现。
- 顾客邮编，只在图 8-13（数据源 1）的实体 CUSTOMER 中出现。
- 顾客性别、婚姻状况、教育程度、信用评分，只在图 8-7 的表 CUSTOMER（数据源 3）中出现。

关系 Uses 和 Has 在图 8-6（数据源 2）中出现。关系 LocatedIn、OccurredAt、SoldVia、BuysVia、BelongsTo 及 SuppliedBy 在图 8-13（数据源 1）中出现。该结果大体上就是结合了操作型数据源相应模型中对分析有用的（由需求指出的）实体、关系及属性得到的一个集

成 ER 图。

一旦如图 8-32 所示的关系模型在 DBMS 中得到实现，它就会由 3 个基础数据源中通过 ETL 架构得到的数据进行填充。该过程的结果如图 8-33 所示。 258

且规范数据仓库（如图 8-32 和图 8-33 所示）创建成功，它就可作为集成数据库使用，任何需要的分析型数据集都可以在其中通过查看或提取得到。

例如，如图 8-15 和图 8-16 所示的维度建模分析型数据库就可以基于图 8-32 和图 8-33 所示的集成规范数据仓库，而不是直接基于基础操作型数据源。

8.17 维度建模数据仓库

另一种方法是将数据仓库看作是交错的维度建模数据集市的集合（如维度模型星座），这些数据集市集成了源自操作型数据源中的对分析有用的数据。该方法得到了 Ralph Kimball 的拥护，也常常被称为 Kimball 方法[2]。图 8-34 列举了一个**维度建模数据仓库**（dimensionally modeled data warehouse）的例子。

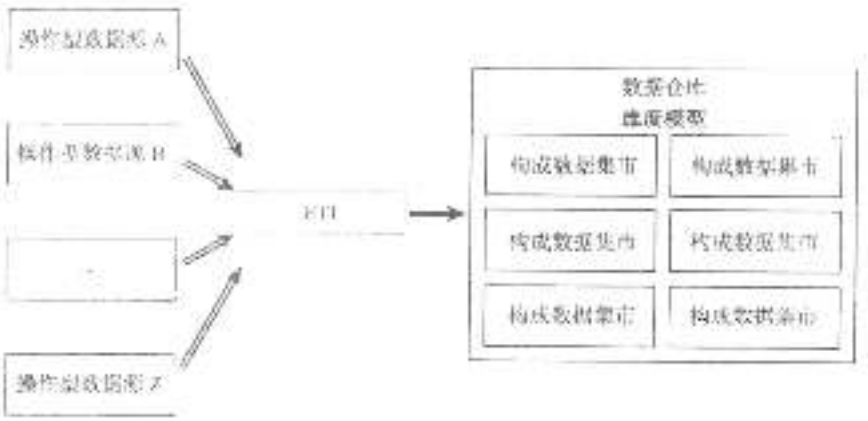


图 8-34 一个维度建模的数据仓库

如图 8-34 所示，对于操作型数据源的使用与 ETL 过程来说，该方法与规范化数据仓库方法一样。区别在于用于建模数据仓库的技术不同。在该方法中，首先设计一组常用的维度集合，称为**一致维度**（conformed dimension）。例如，在一个零售公司中，诸如 CALENDAR、PRODUCT、STORE 的一致性维度可以首先设计，这是因为它们通常会被分析主题所使用。之后可添加与分析主题相关的事实表。然后创建一个维度模型的集合，该集合中的每个事实表与多个维度相连，其中一些维度被多个事实表共享。除了原始创建的一致性维度集合，还要包含其他必需的维度。结果将得到一个数据仓库，该数据仓库是一系列交错的维度建模数据集市的集合（即一个星座），图 8-35 就是一个例子。

如图 8-35 所示，数据仓库设计人员可以首先创建如下维度作为一致性维度：CALENDAR、CUSTOMER、STORE 以及 PRODUCT，希望它们能被事实表表示的多个分析主题所用到。然后根据要求的两个分析主题（即销售和不合格品）创建两个事实表 SALES 和 DEFECTS。这些事实表可以方便地让已有的一致性维度 CALENDAR、CUSTOMER、STORE 以及 PRODUCT 随时与之相连。一旦事实表 SALES 和 DEFECTS 创建成功并连接到 259

[2] 可参考《The Data Warehouse Lifecycle Toolkit》，第 2 版，Ralph Kimball 等著。

维度（SALES 连接到 CALENDAR、CUSTOMER、STORE、PRODUCT，DEFECTS 连接到 CALENDAR、STORE 及 PRODUCT）后，数据仓库模型就大致形成了。

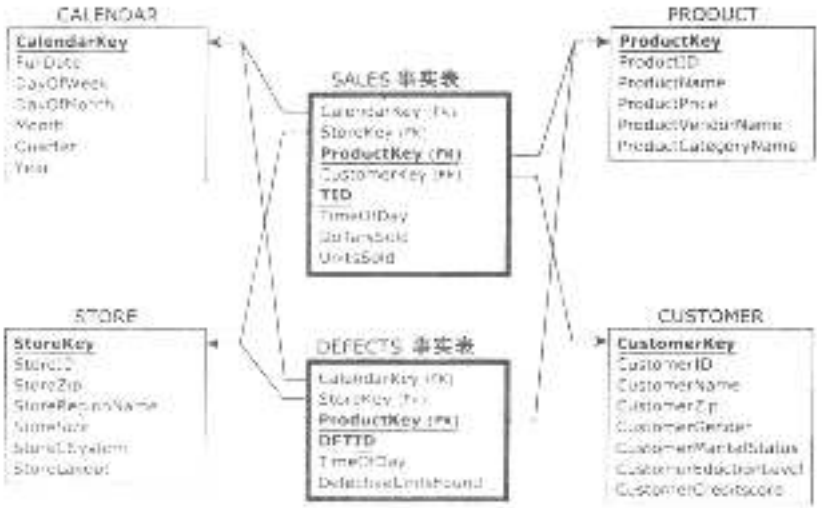


图 8-35　这两个数据集市组成的维度建模数据仓库

注意，除了直接使用 OLAP/BI 工具查询维度建模数据仓库，我们还可以创建独立数据集市以及其他小型视图、子集，或者从数据仓库中提取，这些信息可能会被各种分析人员或应用所需要。图 8-36 说明了这个过程。

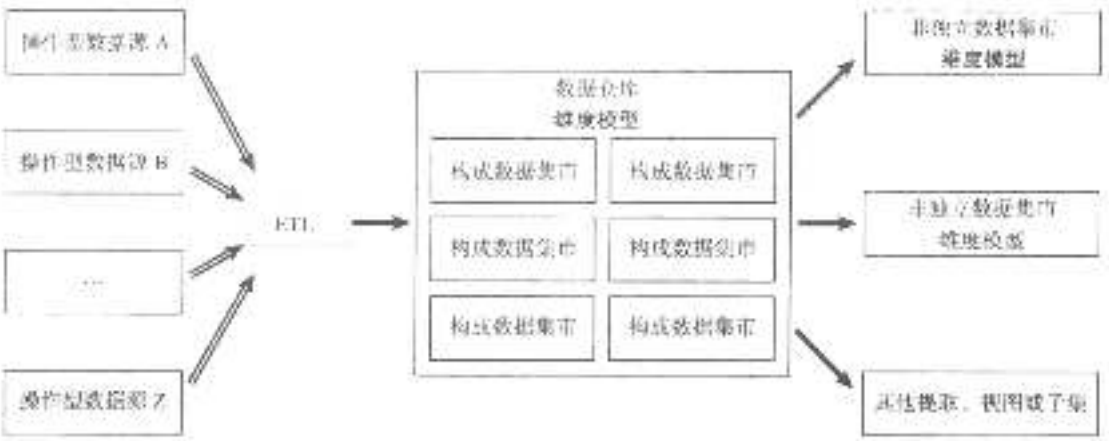


图 8-36　一个维度建模数据集市作为小型分析型数据集的数据源

8.18　维度建模数据仓库实例

为了说明维度建模如何用于建模数据仓库，我们将使用与规范数据仓库相同的 ZAGI 零售公司的场景来举例说明。

[260]

假设 ZAGI 零售公司需要使用维度建模技术为分析其销售情况设计一个数据仓库。该数据仓库的设计需求与本章前面部分规范数据仓库的需求一样。

ZAGI 零售公司需要创建分析型数据库来分析其销售情况。

可用的3个数据源为：

数据源1：如图8-13和图8-14所示的ZAGI零售公司销售部门数据库。

数据源2：如图8-6所示的ZAGI零售公司设备部门数据库。

数据源3：如图8-7所示的顾客信息数据的额外表。

数据仓库需要能根据如下方面分析销售总额和销售总量：

- 日期。
- 时间。
- 产品，包括：
  - 产品名称及价格；
  - 产品种类；
  - 产品供货商。
- 顾客，包括：
  - 顾客姓名、邮编、性别、婚姻状况、教育程度、信用评分。
  - 商店，包括：
  - 单个商店；
  - 商店大小和布局；
  - 商店结账系统；
  - 商店布局；
  - 商店所在区域

图8-37和图8-38说明了如何使用维度建模技术设计一个基于这些数据源与需求的销售分析数据仓库。

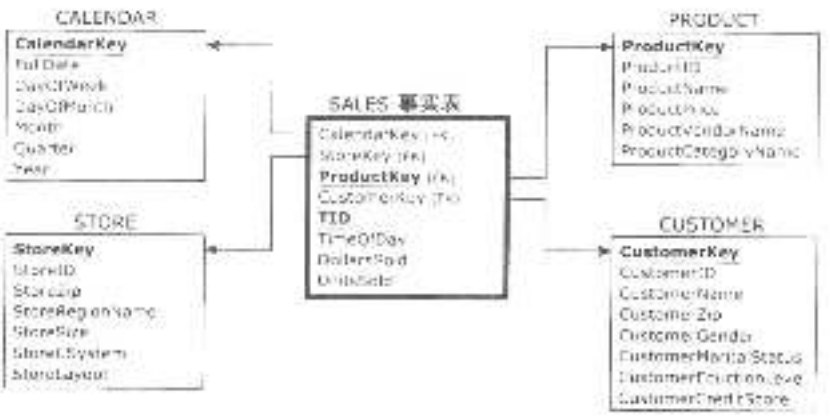


图8-37 带有一个主题（销售）的维度建模数据仓库

在图8-37中，维度CALENDAR、STORE、PRODUCT以及CUSTOMER包括需求中所有有关销售数据分析的必要属性，都可以首先创建。这些维度被连接到SALES事实表进行分析。如果之后确定了其他分析主题，这些维度也可以用作未来其他事实表的一致性维度。 261

在维度CALENDAR、STORE、PRODUCT以及CUSTOMER创建成功后，与两个分析主题销售总额和总量相关的SALES事实表随而被创建，SALES事实表通过外码与维度CALENDAR、STORE、PRODUCT以及CUSTOMER相连。

CALENDAR 维度

CalendarKey	Full Date	DayOf Week	DayOf Month	Month	Qtr	Year
1	1/1/2013	Tuesday	1	January	Q1	2013
2	1/2/2013	Wednesday	2	January	Q1	2013

PRODUCT 维度

ProductKey	ProductID	Product Name	Product Price	Product Vendor Name	Product Category Name
1	111	Zzz Bag	$100	Pacifica Gear	Camping
2	222	Easy Boot	$70	Mountain King	Footwear
3	333	Cozy Sock	$15	Mountain King	Footwear
4	444	Duo Boot	$90	Pacifica Gear	Footwear
5	555	Tiny Tent	$150	Mountain King	Camping
6	666	Biggy Tent	$250	Mountain King	Camping

STORE 维度

StoreKey	StoreID	StoreZip	StoreRegion Name	Store Size (m²)	Store Csystem	Store Layout
1	S1	60600	Chicagoland	51000	Custom	Modern
2	S2	60605	Chicagoland	35000	Self Service	Traditional
3	S3	35400	Tristate	55000	Mixed	Traditional

CUSTOMER 维度

CustomerKey	CustomerID	Customer Name	Customer Zip	Customer Gender	Customer MaritalStatus	Customer EducationLevel	Customer CreditScore
1	1-2-333	Tina	60132	Female	Single	College	700
2	2-3-444	Tony	60611	Male	Single	High School	650
3	3-4-555	Pam	35401	Female	Married	College	523

SALES 事实表

CalendarKey	StoreKey	ProductKey	CustomerKey	TID	TimeOfDay	DollarsSold	UnitsSold
1	1	1	1	T111	8:29:59 AM	$100	1
1	2	2	2	T222	8:34:30 AM	$70	1
2	3	3	3	T333	8:15:08 AM	$75	5
2	3	1	2	T333	8:15:08 AM	$100	1
2	3	4	3	T444	8:20:33 AM	$90	1
2	3	2	3	T444	8:20:33 AM	$140	2
2	3	4	3	T555	8:30:00 AM	$360	4
2	4	5	3	T555	8:30:00 AM	$300	2
2	3	6	3	T555	8:30:00 AM	$250	1

图 8-38　用 3 个基础数据源填充后的图 8-37 所示的关系模型

一旦图 8-37 所示的维度模型在 DBMS 中得到实现，它就会由 3 个基础数据源通过 ETL 架构得到的数据进行填充，该过程的结果如图 8-38 所示。

图 8-37 和图 8-38 所示的维度建模分析型数据仓库可以作为集成数据库使用，任何需要的分析型数据集都可以从中通过查看或提取得到。

[262]

8.19　独立数据集市

显然与数据仓库数据建模技术有关的大部分讨论都与上面列出的两个方法（规范化数据

仓库与维度建模数据仓库及其变本）相关，但还有第三种值得认可和讨论的方法。该方法涉及如图8-39所示的所谓独立数据集市的创建。

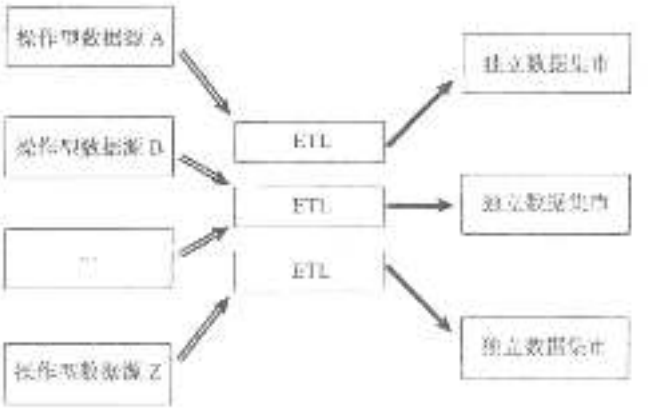


图8-39 独立数据集市

在该方法中，组织机构中的多个组都会创建单独的数据集市，这些数据集市与机构中的其他数据集市间相互独立，因此，需创建并维护多个ETL系统。

事实上，使用独立数据集市方法作为设计企业级分析型数据库的策略是不合适的，这一点在数据仓库技术社区中的所有成员之间已经达成了共识。认为独立数据集市是一种欠佳的策略，其原因是显而易见的。首先，该策略得到的不是一个数据仓库，而是一个不相关的独立数据集市的集合。而在实际的组织机构中，独立数据即使可能形成一个包含所有必需的可分析信息的系统，但这些信息却是分散的，要分析其中某一个单元往往很难甚至不太可能。不能横跨整个企业进行直接分析是该方法的主要缺陷。其次，独立数据集市方法的另一个缺陷在于存在多个不相干的ETL架构。正如第7章所述，ETL通常是数据仓库或数据集市项目中最为耗时且耗资源的部分。拥有多个不相干的ETL过程实际上会造成对多个耗费资源的相似过程进行复制，这些复制过程是不必要的，当然就会产生不必要的资源浪费。

尽管存在这些明显的缺点，但还是有大量企业将分析型数据存储作为独立数据集市进行开发。这看起来自相矛盾，其原因在于关注数据分析过程时不具备原始的企业级视野。

组织机构中的一些部门简单地采用“各做各的”方法来开发用于其分析需要的数据集市。这通常是由于企业“竞争”文化的影响，无起跨部门合作，每个部门更重视自己部门的利益。此时，拥有独立数据集市更多地反映了组织机构的管理及文化问题而不是放弃采用一种较差的数据仓库技术的结果。

此外，在一些实际情况中，组织方面、政治方面以及经济预算方面的因素可能促使机构中的某些部门采取单独行动。在这些场景中，组织机构中的部门或其他团体在考虑数据分析系统的开发问题时，要么创建独立数据集市，要么只能什么都不做。在给出的两个选择中，独立数据集市当然是一个更好的选择。 263

本章讨论了有关数据仓库及数据集市设计的最基础的问题。下一节给出了一些关于采用维度建模和ER建模作为数据仓库/数据集市设计技术的额外信息。

8.20 问题说明：维度建模与ER建模作为数据仓库/数据集市设计技术的比较

本章以一个简单的讨论作为结束，讨论的内容是比较维度建模与ER建模两种数据仓库/数据集市建模技术。

ER建模是一种帮助实现需求收集过程的技术，建模过程中将收集所有需求并可视化为

ER 模型结构：实体、属性及联系。ER 模型为待创建的数据仓库与数据集市的需求（即概念模型）提供了一种清晰的表示。之后数据仓库或数据集市的 ER 模型被映射到关系模式中，该关系模式继而被实现为一个规范化数据库从而形成数据仓库或数据集市。

维度建模既可用于表示数据仓库或数据集市（即概念建模），也可用于创建待实现的数据仓库或数据集市模型（即逻辑建模）。当维度建模技术用于帮助实现数据仓库或数据集市的需求收集过程时，这一过程会确定将那些主题表示为事实表，哪些维度和维度属性用于分析所选主题。一旦需求已表示为概念模型，维度模型就可以直接在 DBMS 中实现为数据仓库或数据集市的模式并得以运转。

这两种用于建模数据仓库或数据集市的方法（ER 建模与维度建模）并不一定存在初看之下那么大的差别。下面的例子说明了采用两种方法得到的最终结果的相关程度。

观察图 8-32，该图给出了使用 ER 建模技术进行数据仓库建模所得到的结果。若我们将 SOLDVIA、SALESTRANSACT 关系连接到 SALES 关系，为其他关系添加代理码，并在关系模式中添加 CALENDAR 关系，以此取代 SALE 表中的 Date 属性，则结果如图 8-40 所示。

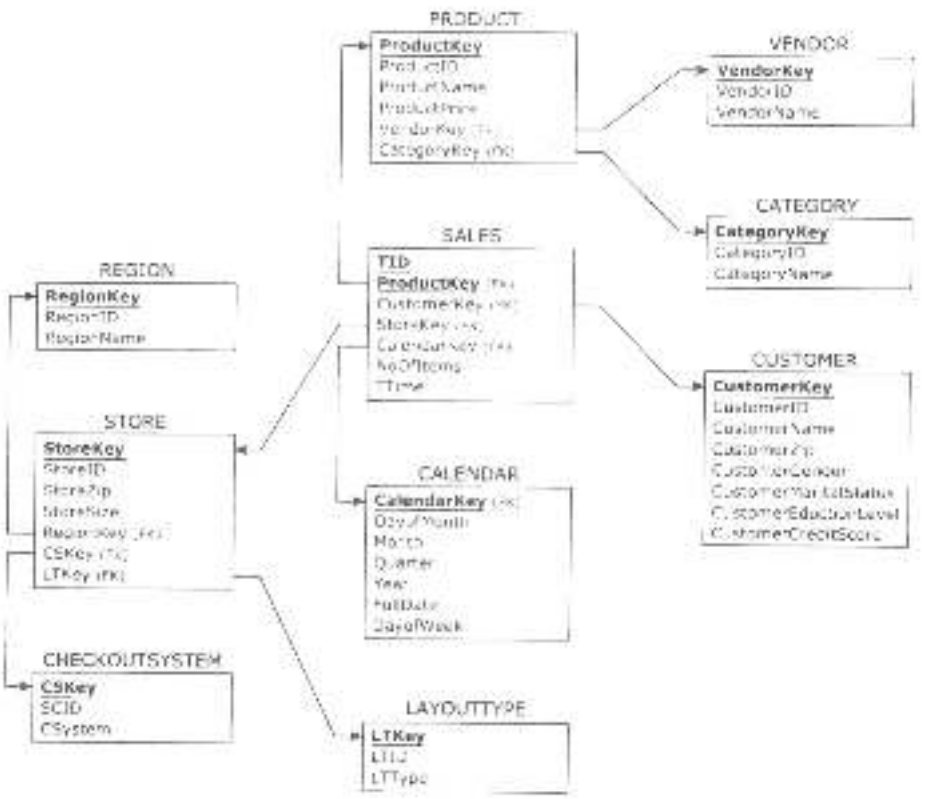


图 8-40　一个修改后的数据仓库模式

图 8-40 给出的模式与图 8-30 给出的雪花维度模式类似。若在图 8-40 中，关系 STORE、REGION、LAYOUT 以及 CHECKOUTSYSTEM 都已经连接，且关系 PRODUCT、VENDOR 和 CATEGORY 也已经连接，结果便会形成一个与单纯的非雪花维度模式类似的模式，非雪

花模式如图 8-22 和图 8-37 所示。

实际上，在建模分析型数据库时，ER 建模和维度建模方法可以在同一个项目中结合使用。换言之，使用一种技术来表示需求并不会妨碍使用其他技术进行建模并最终实现。例如，维度建模技术可用于需求收集过程以实现原始需求的收集、优化、可视示。根据得到的已表示为事实表和维度集合的需求，在更倾向于规范化数据仓库的情况下就可以创建规范化物理数据仓库的 ER 模型。一旦创建了规范化数据仓库，一系列独立数据集市都可以使用维度建模来进行创建。

在建模分析型数据库时，建模技术的选择在不同场景下有所不同。ER 建模和维度建模技术都是可行的选择，正如我们已经讨论的，它们可以在同一个项目中结合使用。

[264]

关键术语

aggregated data（聚集数据），243
aggregated fact table（聚集事实表），243
conformed dimension（一致维度），259
constellation/galaxy of star（星座/星系），242
degenerate dimension（退化维度），237
detailed data（细节数据），243
detailed fact table（细节事实表），243
dimension table/dimension（维度表/维度），225
dimensional modeling（维度建模），225
dimensional modeled data warehouse（维度建模数据仓库），259
fact table（事实表），225
granularity（粒度），248
line-item detailed fact table（条目级细节事实表），249
normalized data warehouse（规范化数据仓库），255
row indicator（行指示符），252
slowly changing dimension（缓慢变化的维度），250
snowflake model（雪花模型），254
star schema（星形模式），226
surrogate key（代理码），230
timestamp（时间戳），252
transaction identifier（事务标识码），235
transaction time（事务时间），235
transaction-level detailed fact table（事务级细节事实表），249
Type1 approach（Type1 方法），250
Type2 approach（Type2 方法），251
Type3 approach（Type3 方法），252

复习题

Q8.1 维度表在维度模型中扮演什么角色？
Q8.2 事实表在维度模型中扮演什么角色？
Q8.3 维度模型的使用如何简化分析查询？
Q8.4 维度模型中事务标识符一般如何表示？
Q8.5 维度模型中事务时间一般如何表示？
Q8.6 星座是什么？
Q8.7 细节事实表以及聚集事实表的区别是什么？
Q8.8 事实表的粒度是什么？
Q8.9 处理缓慢变化维度最常见的方法是什么？
Q8.10 为什么雪花模型通常不用于维度建模？
Q8.11 请描述规范化数据仓库的构成。
Q8.12 请描述维度建模数据仓库的构成。
Q8.13 请描述用于创建分析型数据库的独立数据集市方法。

练习

E8.1　考虑下面相作改变后的 ZAGI 零售公司场景。

ZAGI 零售公司需要创建一个分析型数据库来分析其销售情况。

数据源 1：如图 8-13 和图 8-41 所示的 ZAGI 零售公司销售部门数据库（模式与图 8-13 一样，但数据不同，如图 8-41 所示）。

数据源 2：如图 8-6 所示的 ZAGI 零售公司设备部门数据库。

数据源 3：如图 8-7 所示的顾客信息数据的额外表。

REGION

RegionID	RegionName
[illegible]	[illegible]
[illegible]	[illegible]

STORE

StoreID	StoreZip	RegionID
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

PRODUCT

ProductID	ProductName	ProductPrice	VendorID	CategoryID
[illegible]	[illegible]	[illegible]	PG	CP
[illegible]	[illegible]	$15	MK	FW
[illegible]	[illegible]	$10	MK	FW
[illegible]	[illegible]	$95	PG	FW
[illegible]	[illegible]	$140	MK	CP
[illegible]	[illegible]	$540	MK	CP

VENDOR

VendorID	VendorName
PG	Pacific Gear
MK	Mountain King

CATEGORY

CategoryID	CategoryName
CP	Camping
FW	[illegible]

SALESTRANSACTION

TID	CustomerID	StoreID	Date	TTime
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

SOLDVIA

ProductID	TID	NoOfItems
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

CUSTOMER

CustomerID	CustomerName	CustomerZip
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

图 8-41　数据源 1：ZAGI 零售公司销售部门数据库数据集（另一个不同的数据集）

数据仓库需要能根据如下方面分析销售总额和销售总量：

- 日期，包括：
  - 完整日期；
  - 星期；
  - 月份；
  - 季度；
  - 年份。
- 时间。
- 产品，包括：
  - 产品名称及价格；
  - 产品种类；
  - 产品供货商。
- 顾客，包括：
  - 顾客姓名、邮编、性别、婚姻状况、教育程度、信用评分。
- 商店，包括：

- 单个商店；
- 商店大小和形状；
- 商店收银系统；
- 商店布局；
- 商店区域。

图 8-36 说明了如何使用维度建模技术设计一个基于这些数据源及需求的销售分析数据仓库。

E8.1a 使用图 8-41 中的数据流明图 8-42 维度模型中的表将如何填充。

CALENDAR 维度

CalendarKey	FullDate	DayOf Week	DayOf Month	Month	Qtr	Year

PRODUCT 维度

ProductKey	ProductID	Product Name	Product Price	Product Vendor Name	Product Category Name

STORE 维度

StoreKey	StoreID	StoreZip	StoreRegion Name	Store Size (m²)	Store CSystem	Store Layout

CUSTOMER 维度

CustomerKey	CustomerID	Customer Name	Customer Zip	Customer Gender	Customer MaritalStatus	Customer EducationLevel	Customer CreditScore

SALES 事实表

CalendarKey	StoreKey	ProductKey	CustomerKey	TID	TimeOfDay	DollarsSold	UnitsSold

图 8-42

E8.1b 创建一个包含集成事实表的维度模型，事实表表示每个商店每个产品的日事务中售出产品与售出金额的汇总。

E8.1c 为 E8.1b 所创建的表填充数据，聚集基础与填充图 8-42 中表的数据相同。 266

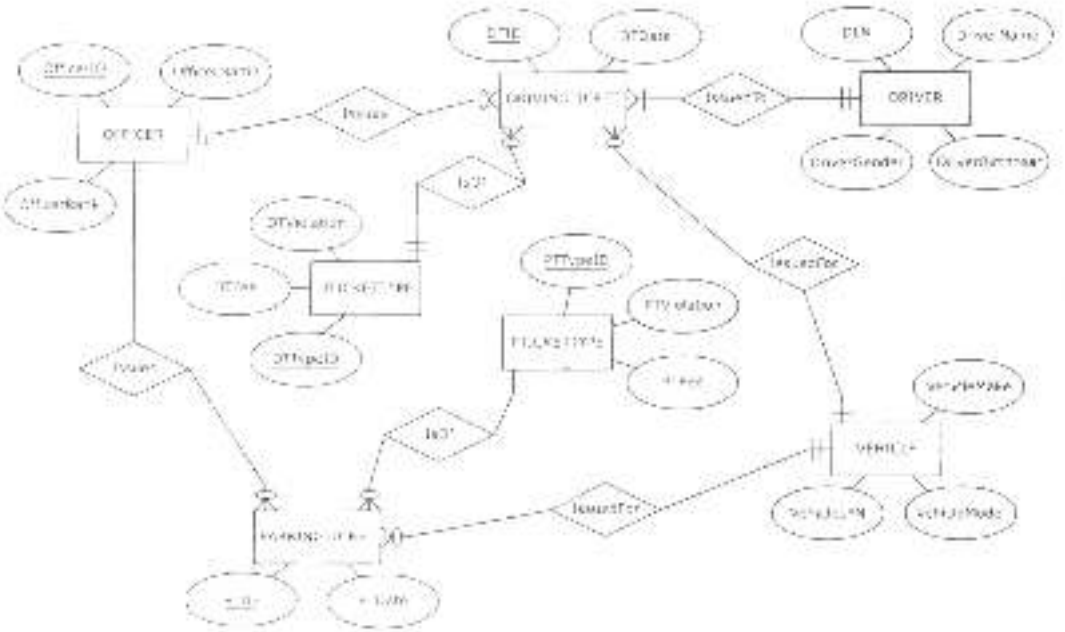
E8.2 考虑下面与城市警察局相关的例子：

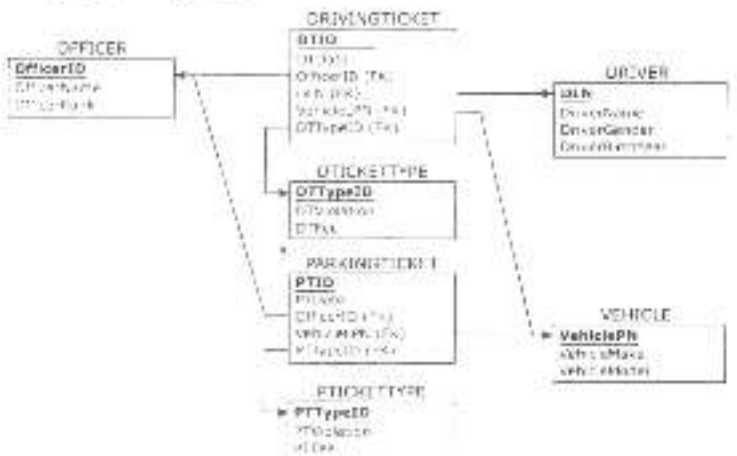
城市警察局需要创建一个分析型数据库来分析其罚单收入。

有两个可用数据源，数据源 1 和数据源 2，如下所示。

数据源 1：城市警察局维护的罚单违规记录数据库，如图 8-43 所示。

数据源 2：机动车辆部门（DMV）维护的车辆注册表，如图 8-44 所示。





OFFICER

OfficerID	OfficerName	OfficerRank
1	[illegible]	[illegible]
2	Mike	[illegible]
3	Rob	[illegible]

DRIVER

DLN	DriverName	DriverGender	DriverBirthYear
111	Bill	Male	1985
222	Sally	[illegible]	1977

VEHICLE

VehicleLPN	VehicleMake	VehicleModel
L1111	Honda	CRV
L2222	Honda	Civic
L3333	Honda	Civic

DRIVINGTICKET

DTID	DTDate	OfficerID	DLN	VehicleLPN	DTTypeID
[illegible]	[illegible] Jan 2013	[illegible]	111	L1111	D1
[illegible]	2 Jan 2013	3	222	L2222	D1
[illegible]	2 Jan 2013	3	111	L1111	D2

DTICKETTYPE

DTTypeID	DTViolation	DTFee
D1	Red Light	$200
D2	Speeding	$150

PTICKETTYPE

PTTypeID	PTViolation	DTFee
P1	Meter Expired	$100
P2	Hydrant	$100

PARKINGTICKET

PTID	PTDate	OfficerID	VehicleLPN	PTTypeID
PT1111	[illegible] Jan 2013	1	L1111	P1
PT2222	[illegible] Jan 2013	2	L2222	P1
PT3333	2 Jan 2013	3	L3333	P2

图 8-43 数据第 1：城市警察局维护的罚单违规记录数据库

VEHICLE REGISTRATION TABLE

VehicleLPN	VehicleMake	VehicleModel	VehicleYear	OwnerDLN	OwnerName	OwnerGender	OwnerBirthYear
IL1111	Honda	CRV	2007	111	Joe	Male	1968
IL2222	Honda	Civic	2012	222	Suzy	Female	1977
IL3333	Honda	Civic	2005	333	David	Male	1966

图 8-44　数据源 2：机动车辆部门（DMV）维护的车辆注册表

数据仓库需要根据如下方面来分析罚款收入：

- 日期，包括：
  - 完整日期；
  - 星期；
  - 月份；
  - 季度；
  - 年份；
- 警员，包括：
  - 警员编号；
  - 警员姓名；
  - 警员级别；
- 受罚人，包括：
  - 受罚人 DLN；
  - 受罚人姓名；
  - 受罚人性别；
  - 受罚人出生年份；
- 车辆，包括：
  - 车辆 LPN；
  - 车辆品牌；
  - 车辆型号；
  - 车辆年份；
  - 车主 DLN；
  - 车主姓名；
  - 车主性别；
  - 车主出生年份；
- 罚单类型，包括：
  - 罚单种类（行驶类或停车类）；
  - 罚单级别；
  - 罚单金额。

287
288

图 8-45 说明了如何使用维度建模技术设计一个基于这些数据源及需求的罚单收入分析数据仓库。

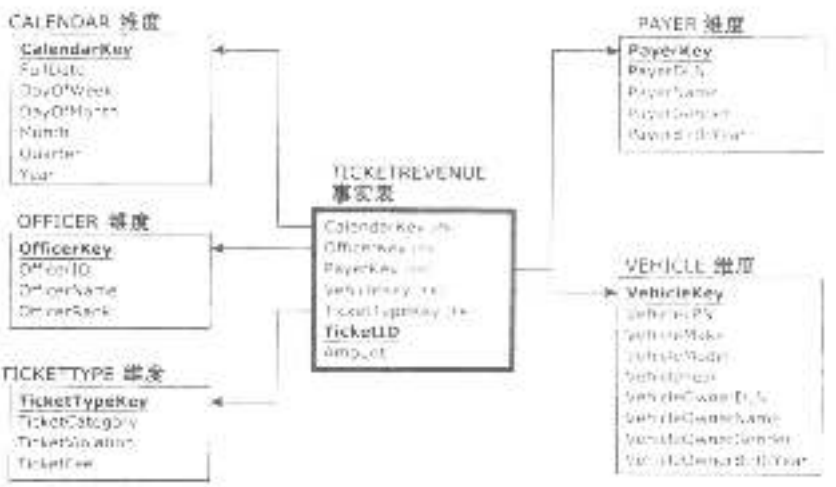


图 8-45　关于车辆罚单收入的维度建模数据仓库

E8.2a　使用图 8-43 与图 8-44 中的数据源说明图 8-46 中的空表应如何填充。

E8.2b　创建一个包含了集成事实表的维度模型，事实表表示每个警员的日罚单收入信息。

E8.2c　为 E8.2b 所创建的表填充数据，要求基础与填充图 8-46 中表的数据相同。

E8.3　考虑下面与 Big Z 公司相关的例子，Big Z 公司是一家汽车产品批发商。

Big Z 公司需要创建一个分析型数据库（数据仓库）来分析其订单数目。

269

CALENDAR 维度

CalendarKey	Full Date	DayOf Week	DayOf Month	Month	Qtr	Year

PAYER 维度

PayerKey	PayerDLN	PayerName	Payer Gender	Payer BirthYear

OFFICER 维度

OfficerKey	OfficerID	Officer Name	Officer Rank

VEHICLE 维度

Vehicle Key	Vehicle LPN	Vehicle Make	Vehicle Model	Vehicle Year	Vehicle OwnerDLN	Vehicle Owner Name	Vehicle Owner Gender	Vehicle Owner BirthYear

TICKETTYPE 维度

Ticket TypeKey	Ticket Category	Ticket Violation	Ticket Fee

TICKETREVENUE 事实表

CalendarKey	OfficerKey	PayerKey	VehicleKey	Ticket TypeKey	Ticket ID	Amount

图　8-46

有两个可用数据源，数据源 1 和数据源 2，描述如下。

数据源 1：Big Z 公司人力资源部门表，如图 8-47 所示。

数据源 2：Big Z 公司订单数据库，如图 8-48 所示。

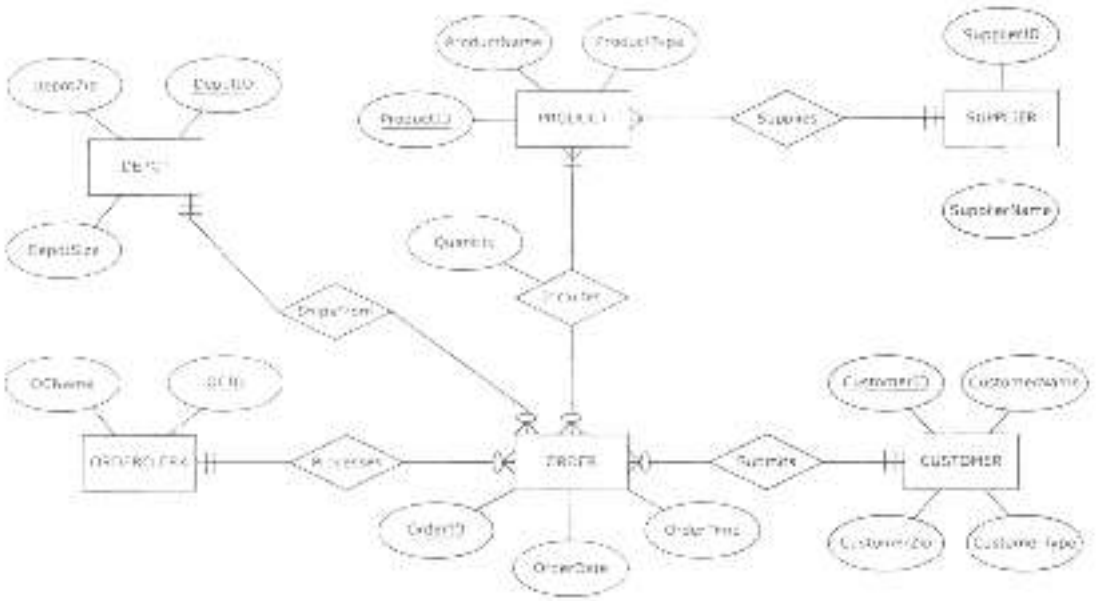
HUMAN RESOURCES DEPARTMENT TABLE (EMPLOYEE DATA)

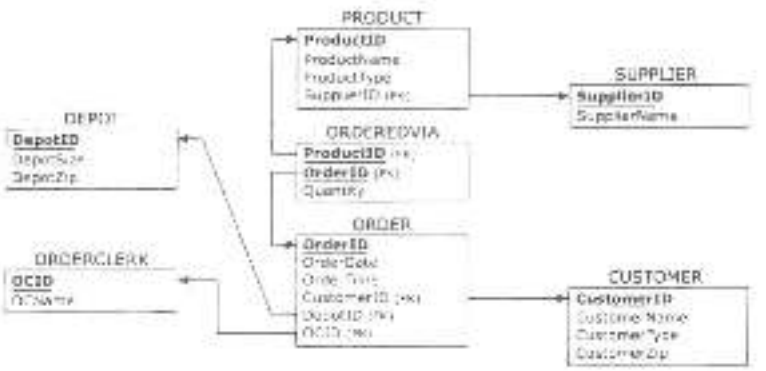
EmployeeID	Name	Title	EducationLevel	YearOfHire
OC1	Antonio	Order Clerk	High School	2001
OC2	Wesley	Order Clerk	College	2005
OC3	Lily	Order Clerk	College	2005

图 8-47　数据源 1:Big Z 公司人力资源部门表

数据仓库需要根据如下方面来分析订单数目：

- 日期，包括：
  - 完整日期；
  - 星期；
  - 月份；
  - 季度；
  - 年份。
- 时间。
- 产品，包括：
  - 产品编号；
  - 产品名称；
  - 产品类型；
  - 产品供应商名称。
- 顾客，包括：
  - 顾客 ID；
  - 顾客姓名；
  - 顾客类型；
  - 顾客邮编。
- 仓库，包括：
  - 仓库 ID；
  - 仓库大小；
  - 仓库邮编。
- 订单登记员，包括：
  - 登记员编号；
  - 登记员姓名；
  - 登记员头衔；
  - 登记员教育程度；
  - 登记员雇佣年份。





DEPOT

DepotID	DepotSize	DepotZip
D1	Small	60611
D2	Large	60660
D3	Large	60611

PRODUCT

ProductID	ProductName	ProductType	SupplierID
P1	BigGripper	Tire	ST
P2	TractionWiz	Tire	ST
P3	SureStart	Battery	BE

SUPPLIER

SupplierID	SupplierName
ST	Super Tires
BE	Batteries Etc

ORDERCLERK

OCID	OCName
OC1	Angela
OC2	Wesley
OC3	Lily

ORDER

OrderID	CustomerID	DepotID	OCID	OrderDate	OrderTime
O1	C1	D1	OC1	1-Jan-2013	9:00:00 AM
O2	C2	D1	OC2	2-Jan-2013	9:00:00 AM
O3	C3	D2	OC3	2-Jan-2013	9:10:00 AM
O4	C1	D2	OC1	3-Jan-2013	9:00:00 AM
O5	C2	D3	OC2	3-Jan-2013	9:15:00 AM
O6	C3	D3	OC3	3-Jan-2013	9:30:00 AM

ORDEREDVIA

ProductID	OrderID	Quantity
P1	O1	4
P2	O1	8
P1	O2	12
P2	O3	4
P3	O4	7
P3	O5	6
P2	O6	8
P1	O6	4

CUSTOMER

CustomerID	CustomerName	CustomerType	CustomerZip
C1	Auto Doc	Repair Shop	60137
C2	Bob's Car Repair	Repair Shop	60140
C3	JJ Auto Parts	Retailer	60605

图8-48 数据源2：Big Z公司订单数据库

E8.3a 使用以上数据源和需求创建一个用于 Big Z 公司维度建模数据仓库的维度模型。

E8.3b 使用图 8-47 与图 8-48 中的数据说明 E8.3a 中的维度模型表应如何填充。

270 - 271

E8.3c 基于以上数据源和需求为 Big Z 公司的规范化数据仓库创建一个 ER 模型和关系模型映射。

E8.3d 使用图 8-47 与图 8-48 中的数据说明练习 3c 中的关系模型表应如何填充。

E8.4 考虑下面缓慢变化维度 EMPLOYEE 的例子。

EMPLOYEE

EmployeeKey	EmployeeID	EmployeeName	EmployeeTitle
1	E101	Ava	Project Manager
2	E202	Sidney	Business Analyst
3	E303	Lorna	Senior Business Analyst

假设 Sidney 的头衔从业务分析员变为了高级业务分析员。

E8.4a 如果采用 Type1 方法来处理缓慢变化维度，给出维度 EMPLOYEE（包括所有记录）。

E8.4b 如果采用 Type2 方法来处理缓慢变化维度，给出维度 EMPLOYEE（包括所有记录）。

E8.4c 如果采用 Type3 方法来处理缓慢变化维度，给出维度 EMPLOYEE（包括所有记录）。

小案例

小案例 6 Jones Dozers

Jones Dozers 需要创建一个分析型数据库来分析其销售金额和租金收入。唯一可用的数据源是 Jones Dozers 销售和租赁数据库（第 2 章小案例 6 的图 2-59 为 ER 图，关系模式则在第 3 章小案例 6 中创建）。

创建一个维度模型，从以下方面实现销售和租赁收入分析：

- 日期。
- 收入类型（销售或租赁）。
- 顾客。
- 设备。
- 销售员。

事实表中的每行表示一个销售或租赁事务中发生的收入总金额。

小案例 7 Midtown Memorial

Midtown Memorial 医院需要创建一个分析型数据库，用于分析其病人的药物成分摄入量。唯一可用的数据源是 Midtown Memorial 病人药物成分数据库（第 2 章小案例 7 的图 2-60 为 ER 图，关系模式则在第 3 章小案例 7 中创建）。

创建一个维度模型，从以下方面实现摄入成分分析：

- 日期。
- 时间。
- 病人。
- 护士。
- 药物。
- 成分。

272

事实表中的每行表示一个病人在一次药物摄入过程中的成分摄入量（计算方法：次数 × 数量）。

数据仓库的实现与使用

9.1 引言

前一章介绍了数据仓库建模的相关内容，本章将进一步关注后续的数据仓库开发过程。本章不仅会描述和展示数据仓库的实现步骤：创建数据仓库，ETL 过程，开发数据仓库前端应用程序以及部署数据仓库，还会对数据仓库的使用进行探讨。本章将主要关注数据仓库在 OLAP 工具中的使用。

与第 8 章中演示数据仓库建模过程使用的实例类似，本章同样以 ZAGI 零售公司为实例来演示相关内容，这里稍微扩展了实例以展示本章包含的主题。

9.2 创建数据仓库

创建数据仓库涉及使用数据库管理软件的一些功能去实现数据仓库模型。数据仓库模型是物理上互相连接的数据库表的集合。多数情况下，数据仓库都被建模成关系数据库，因此就需要使用关系型 DBMS 来创建数据仓库。

例如，假设 ZAGI 零售公司已经完成数据仓库建模，创建了如图 9-1 所示的数据仓库模型。

273

图 9-1　一个数据仓库模型

创建本例中的数据仓库需要使用图 9-2 所示的 5 条 CREATE TABLE SQL 语句。当这 5 条语句执行完后，便会生成数据仓库表。（创建数据仓库时，除了要使用 CREATE TABLE 语句外，还包括一些与数据仓库物理实现相关的操作，如建立索引。）目前越来越流行使用 ETL 架构来创建数据仓库表。

274

```
CREATE TABLE calendar
        (       calendarkey                     INT,
                fulldate                        DATE,
                dayofweek                       CHAR(15),
                daytype                         CHAR(20),
                dayofmonth                      INT,
                month                           CHAR(10),
                quarter                         CHAR(2),
                year                            INT,
                PRIMARY KEY                     (calendarkey));

CREATE TABLE store
        (       storekey                        INT,
                storeid                         CHAR(5),
                storezip                        CHAR(5),
                storeregionname                 CHAR(15),
                storesize                       INT,
                storecsystem                    CHAR(15),
                storelayout                     CHAR(15),
                PRIMARY KEY                     (storekey));

CREATE TABLE product
        (       productkey                      INT,
                productid                       CHAR(5),
                productname                     CHAR(25),
                productprice                    NUMBER(7,2),
                productvendorname               CHAR(25),
                productcategoryname             CHAR(25),
                PRIMARY KEY                     (productkey));

CREATE TABLE customer
        (       customerkey                     INT,
                customerid                      CHAR(7),
                customername                    CHAR(15),
                customerzip                     CHAR(5),
                customergender                  CHAR(15),
                customermaritalstatus           CHAR(15),
                customereducationlevel          CHAR(15),
                customercreditscore             INT,
                PRIMARY KEY                     (customerkey));

CREATE TABLE sales
        (       calendarkey                     INT,
                storekey                        INT,
                productkey                      INT,
                customerkey                     INT,
                tid                             CHAR(15),
                timeofday                       TIME,
                dollarssold                     NUMBER(10,2),
                unitssold                       INT,
                PRIMARY KEY                     (productkey, tid),
                FOREIGN KEY (calendarkey)       REFERENCES calendar,
                FOREIGN KEY (storekey)          REFERENCES store,
                FOREIGN KEY (productkey)        REFERENCES product,
                FOREIGN KEY (customerkey)       REFERENCES customer);
```

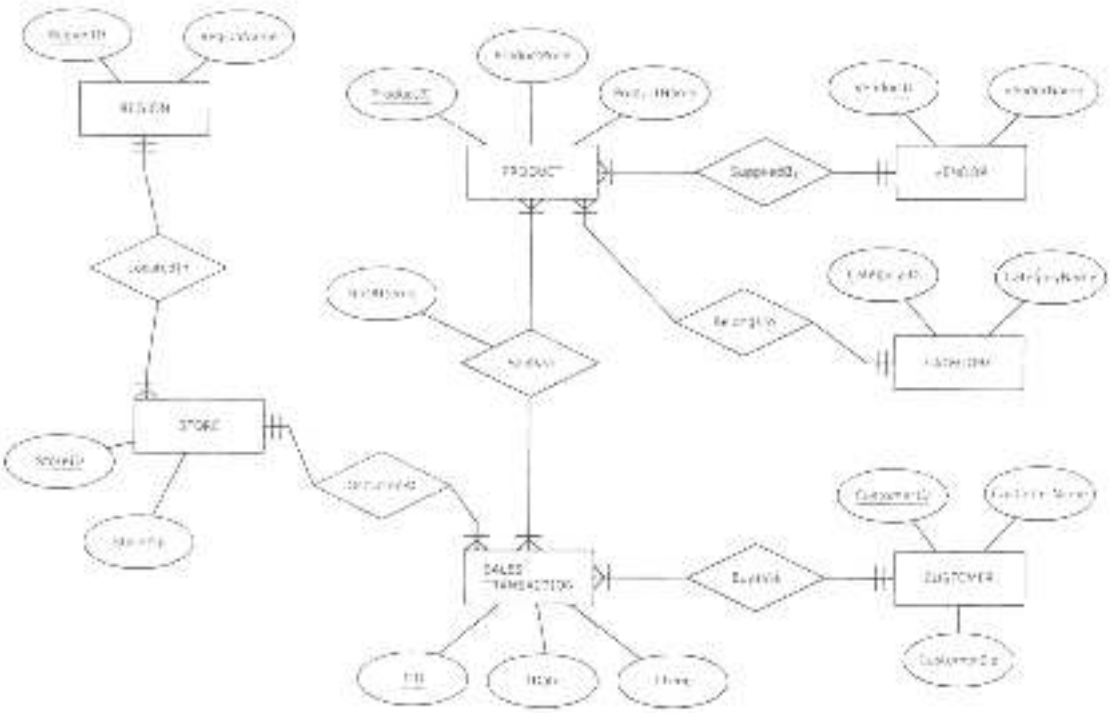
图 9-2 创建图 9-1 数据仓库模型中的表结构

9.3 ETL：提取、转换、加载

当数据仓库建模完成后，随后便是使用DBMS软件创建一系列的空数据仓库表（如图9-2所示）。接下来要向这些空表中插入数据。从操作型数据库中提取数据仓库表中的相关数据需要一套流程。这套流程便是ETL（提取、转换、加载）过程。

我们将扩展ZAGI零售公司实例来展示ETL过程。图9-1中表的数据来自下面二个数据源：

数据源1：图9-3和图9-4所示的ZAGI零售公司销售部门数据库。 275



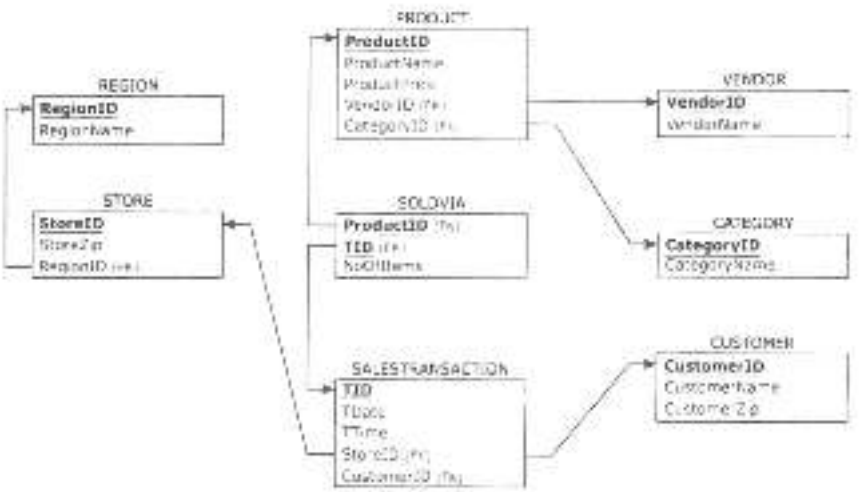


图9-3 数据源1：ZAGI零售公司销售部门数据库ER图以及相应的关系模型

REGION

RegionID	RegionName
C	Chicagoland
T	Texas

PRODUCT

ProductID	ProductName	ProductPrice	VendorID	CategoryID
1X1	Zip Bag	$100	AC	CP
2X2	Easy Boot	$70	MK	FW
3X3	Cozy Sock	$15	MK	FW
4X4	Dura Boot	$90	PG	FW
5X5	Trip Tent	$150	MK	CP
6X6	Super Tent	$250	MK	CP

VENDOR

VendorID	VendorName
PG	Pacifico Gear
MK	Mountain King

STORE

StoreID	StoreZip	RegionID
S1	60600	C
S2	60608	C
S3	75400	T

CATEGORY

CategoryID	CategoryName
CP	Camping
FW	Footwear

SALES TRANSACTION

TID	CustomerID	StoreID	TDate	TTime
111	1-2-333	S1	1-Jan-2013	8:23:55 AM
222	2-3-444	S1	1-Jan-2013	8:24:30 AM
7333	1-2-333	S3	1-Jan-2013	8:15:08 AM
444	3-4-555	S2	2-Jan-2013	8:25:33 AM
7555	2-3-444	S3	2-Jan-2013	8:31:00 AM
666	2-3-444	S1	3-Jan-2013	8:02:00 AM
7777	3-4-555	S3	3-Jan-2013	8:10:00 AM
7666	1-2-333	S2	4-Jan-2013	8:05:00 AM
1999	2-3-444	S2	4-Jan-2013	8:07:33 AM
7000	2-3-444	S2	4-Jan-2013	8:02:38 AM

SOLDWA

ProductID	TID	NoOfItems
1X	111	1
2X2	7222	1
3X3	1333	5
1X1	7333	1
4X4	1444	1
2X2	7444	2
4X4	555	4
5X5	7555	2
6X6	1666	1
1X1	7666	1
2X2	7777	1
3X3	1777	2
1X1	7888	1
1X1	1999	2
4X4	7999	1
4X4	1999	2
1X1	7000	3
3X3	11000	4
4X4	7000	3

CUSTOMER

CustomerID	CustomerName	CustomerZip
1-2-333	Ira	60137
2-3-444	Tony	60611
3-4-555	Pam	35401

图 9-4 数据源 1：图 9-3 所示 ZAGI 零售公司销售部门数据库中各个表的数据记录

数据源 2：图 9-5 所示的 ZAGI 零售公司设备部门数据库。

数据源 3：图 9-6 所示的顾客统计数据的额外表。

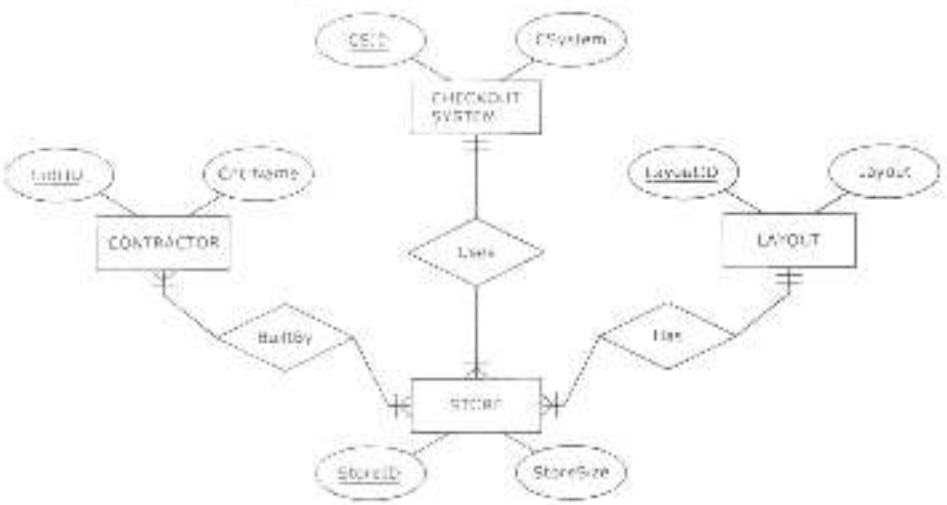
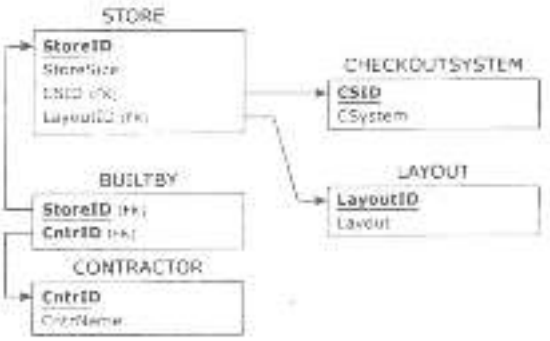


图 9-5 数据源 2：ZAGI 零售公司设备部门数据库



CONTRACTOR

CntrID	CntrName
C1	Anne Construction
C2	GIN Builders
C3	Rauker G.C.

CHECKOUTSYSTEM

CSID	CSystem
CR	Cashiers
AC	Self Service
MX	Mixed

LAYOUT

LayoutID	Layout
M	Modern
T	Traditional

BUILTBY

StoreID	CntrID
S1	C1
S1	C2
S1	C3
S2	C1
S2	C3
S3	C1
S3	C3

STORE TABLE

StoreID	StoreSize (m²)	CSID	LayoutID
S1	51000	CR	M
S2	35000	AC	T
S3	55000	MX	T

图 9-5（续）

ETL 架构从三个数据源（图 9-3、图 9-4、图 9-5 和图 9-6）中提取数据，对数据进行转换，并且将数据加载到数据仓库。最终生成如图 9-7 所示的填充后的数据仓库。为了更详细地演示 ETL 流程，我们将对图 9-3、图 9-4、图 9-5 以及图 9-6 所示的数据源系统如何产生如图 9-1（模型）以及图 9-7（成分表）所示的目标数据仓库的过程进行详细描述。

CUSTOMER TABLE

CustomerID	Customer Name	Gender	Marital Status	Education Level	Credit Score
1-2-333	Tina	Female	Single	College	700
2-3-444	Tony	Male	Single	High School	650
3-4-555	Penny	Female	Married	College	823

图 9-6 数据源 3：顾客统计数据的额外表

1. 提取

提取是指从操作型数据库中检索出最终将被加载到数据仓库的对分析有用的数据。在上述 ZAGI 零售公司的实例中，提取是指从三个数据源即数据源 1、数据源 2 以及数据源 3 中提取数据。在数据仓库中对分析有用的数据将被提取出来。在图 9-1 所示的数据仓库模型中，数据源 1 和数据源 3 中的所有数据都将被提取出来，而数据源 2 中有些数据将被提取，有些数据则不被提取。特别地，数据源 2 中的 CONTRACTOR 表和 BUILTBY 表中的数据没有被提取。在建模阶段，已经确定了目标数据仓库的分析主题是销售，所以建造商店的承包商与目标数据仓库的分析主题是不相关的。这个结论也可以从图 9-1 所示的模型中看出，图 9-1 所示的模型中不包括建造商店的承包商表中的相关属性。

CALENDAR 维度

Calendar Key	FullDate	DayOfWeek	DayType	DayOfMonth	Month	Qtr	Year
1	1/1/2013	Tuesday	Weekend/Holiday	1	January	Q1	2013
2	1/2/2013	Wednesday	Workday	2	January	Q1	2013
3	1/3/2013	Thursday	Workday	3	January	Q1	2013
4	1/4/2013	Friday	Workday	4	January	Q1	2013

PRODUCT 维度

ProductKey	ProductID	ProductName	ProductPrice	ProductVendor Name	ProductCategory Name
1	1X	Zzz Bag	$100	Pacifica Gear	Camping
2	2X2	Easy Boot	$70	Mountain King	Footwear
3	3X3	Cosy Sock	$15	Mountain King	Footwear
4	4X4	Dura Boot	$90	Pacifica Gear	Footwear
5	5X5	Try Tent	$150	Mountain King	Camping
6	6X6	Biggy Tent	$250	Mountain King	Camping

STORE 维度

StoreKey	StoreID	StoreZip	StoreRegion Name	Store Size (m^2)	Store CSystem	Store Layout
1	S1	60600	Chicagoland	61000	Cashiers	Modern
2	S2	60605	Chicagoland	35000	Self Service	Traditional
3	S3	35400	Tristate	55000	Mixed	Traditional

CUSTOMER 维度

Customer Key	CustomerID	Customer Name	Customer Zip	Customer Gender	Customer Marital Status	Customer Education Level	Customer Credit Score
1	1-2-333	Tina	60137	Female	Single	College	700
2	2-3-444	Tony	60611	Male	Single	High School	650
3	3-4-555	Pam	35401	Female	Married	College	623

SALES 事实表

CalendarKey	StoreKey	ProductKey	Customer Key	TID	TimeOfDay	Dollars Sold	UnitsSold
1	1	1	1	T111	8:23:59 AM	$100	1
1	2	2	2	T222	8:24:32 AM	$70	1
2	3	5	1	T333	8:15:06 AM	$75	5
2	3	1	1	T333	8:16:08 AM	$100	1
2	3	4	3	T444	8:20:33 AM	$90	1
2	3	2	3	T444	8:20:33 AM	$140	2
2	3	4	2	T555	8:30:00 AM	$360	4
2	3	5	2	T555	8:30:00 AM	$300	2
2	3	6	2	T555	8:30:00 AM	$250	1
3	1	1	2	T666	8:00:00 AM	$100	1
3	2	2	3	T777	8:10:00 AM	$70	1
3	2	3	2	T777	8:10:00 AM	$30	3
4	3	1	1	T888	8:05:00 AM	$200	2
4	2	1	2	T999	8:07:33 AM	$300	3
4	2	3	2	T999	8:07:33 AM	$60	4
4	2	4	3	T999	8:07:33 AM	$160	2

图 9-7　一个由数据第 1、数据第 2 和数据第 3 域充的数据仓库

即使承包商相关数据在一定程度上能够反映销售情况，但要将承包商相关数据包含到数据仓库中仍然会比较困难。首先，图 9-5 表明一个商店具有多个承包商，因此商店的承包商信息不能作为维度 STORE 的一个属性。其次，这个数据仓库分析的主题是销售，而某个承包商的工作对其所建造商店的销售情况的影响是不明确的。每一个商店都有多个承包商，并且底层数据源中没有包含一个承包商的工作量占建造这个商店工作总量的百分比信息。这就导致不能准确地计算承包商对商店销售的影响值。因此，CONTRACTOR 和 BUILTBY 表中的数据不能被加载到目标数据仓库中。

什么数据会被提取将由需求分析和建模阶段来决定。在数据仓库的需求分析和建模阶段，需要检查可用的数据源。在创建 ETL 架构的过程中，数据模型（已确定包含哪些底层数据）提供了提取程序的蓝本。数据仓库开发团队也可能会发现某些 ETL 操作与已经给定的数据模型的描述有所不同，在这种情况下，项目就需要重新返回到创建 ETL 架构的需求分析阶段，如第 7 章图 7-9 所示。正如前面所述，在数据仓库开发的任何阶段都不允许隐性地改变需求，创建 ETL 架构阶段也不例外。需求的任何改变都必须记录在文档中，并且必须反馈到后续的可视化和模型里。

2. 转换

转换是指对已经提取的数据的结构进行转换以满足目标数据仓库模型的过程。

在 ZAGI 零售公司实例中，转换过程涉及以下几步。图 9-4 数据源 1 中 CUSTOMER 表的记录与图 9-6 数据源 3 中 CUSTOMER 表的记录合并。换句话说，销售交易中的顾客数据被合并到了顾客统计数据中。然后在每条记录中加入代理码，这样，转换后的数据就适合目标数据仓库中 CUSTOMER 维度的结构，如图 9-7 所示。

除了改变提取数据的结构外，ETL 过程的转换部分也包括控制数据质量，甚至是提高数据质量的过程。尽管理想情况下底层的操作型数据源所包含的数据都是高质量的，但在实际中，一些数据源中的数据仍然会存在数据质量问题，如第 6 章所述。在第 6 章的那些实例中，转换过程包含检测和校正低质量的数据。这种转换操作称为**数据清洗 / 清理**（data cleansing/scrubbing）。

图 9-4 数据源 1 中有两个表中的记录具有唯一性数据质量问题。特别地，SALESTRANSACTION 表中的最后一条记录以及 SOLDVIA 表中的后三条记录都是唯一性数据质量问题导致的。数据源 1 中的数据实体对相同的交易记录了两次，使用了不同的 TID 值（T999 和 T1000）。这两条记录包含相同的顾客、商店、产品以及交易发生日期和时间。数据清洗行为发生在转换阶段，识别出唯一性数据质量问题，并将 TID 值为 T1000 的记录从已提取的数据集中排除。相应地，这个低质量数据不会在数据仓库 SALES 事实表中出现，如图 9-7 所示。

除了校正底层数据质量问题外，转换过程也可能涉及将不同版本数据源中的相同数据进行规范化。通常情况下，各个机构拥有各自单独的操作型数据库，这些数据库中往往会有重叠信息。例如，ZAGI 零售公司实例中，数据源 1 和数据源 2 数据库都包含商店（StoreID 和 StoreZIP）这个重叠信息。幸运的是，这个信息在数据源 1 和数据源 2 中的格式是完全相同的。然而，数据源 1 和数据源 3 中包含的顾客重叠信息却具有不同的格式。数据源 1 中，表 CUSTOMER 中的顾客 ID 是 3-4-555（如图 9-4 所示），她的名字是 Pam。然而同一个顾客在数据源 3 的表 CUSTOMER 中（如图 9-6 所示），其名字记录是 Pammy。转换阶段识别出这两个名字属于同一个人，将选择其中一个（本例中选用 Pam 这个名字）用于 CUSTOMER 维

度中，如图 9-7 所示。

3. 加载

加载是将已经被提取、转换并已保证质量的数据加载到目标数据仓库。ETL 工具将加载做成一个批处理程序，即将数据按照一定格式自动地插入到数据仓库表中。加载初期填充最初的空数据仓库表，称为**首次加载**（first load）。首次加载涉及大量数据，这将取决于新的初始数据仓库所需数据的时间范围。后续的加载都是**刷新加载**（refresh load）。**刷新周期**（refresh cycle）是指加载新的数据到数据仓库的频率。刷新周期是事先确定好的，取决于数据仓库商业用户分析的需要以及系统的技术可行性。例如，一个数据仓库可以每天更新一次，也可以每几小时就更新一次。在所谓的**动态数据仓库**（active data warehouse）中，常常持续地进行微批量加载，以确保数据仓库中数据接近实时更新（以确保能够分析最新数据）。

数据仓库首次加载数据源或源文件里的全部数据。之后，数据仓库按照确定的刷新周期重新加载数据源中的新数据。

ETL 架构使得 ETL 过程变得更容易。典型地，创建 ETL 架构过程包括使用专门的 ETL 软件工具或者自己编写代码。由于必须要考虑大量的细节，创建 ETL 架构常常是数据库开发过程中最耗时的部分。尽管创建 ETL 架构耗时耗力，但是正确地创建 ETL 架构是需求收集和数据仓库建模过程中指定数据源到目标数据源所必需的。

9.4 在线分析处理

OLAP 是在线分析处理（online analytical processing）的缩写。术语 OLAP 常常用来与另外一个缩写词 OLTP 作对比，OLTP 表示在线事务处理（online transaction processing）。在解释术语 OLAP 之前，我们将首先简要阐明 OLTP 这个术语。

在线事务处理（OLTP）是指以操作为目的的数据更新（如插入、修改以及删除）、查询和呈现。除了检索操作型数据库中的数据，OLTP 还包括操作型数据库中所有的日常事务更新。例如反映从活期账户取款的事务或者创建一个机票预订事务。操作型数据库也经常称为“OLTP 系统”。

在线分析处理（OLAP）是指以分析为目的对数据仓库或数据集市进行的数据查询和呈现。术语 OLTP 通常用于传统的以操作为目的的数据库（日常）事务中，而术语 OLAP 则与数据仓库和数据集市相关。OLTP 与 OLAP 的另一个不同之处是 OLTP 系统能够进行数据更新、查询和呈现，而 OLAP 工具只能进行数据查询和呈现。OLTP 系统会经常执行插入、修改以及删除数据等操作，而 OLAP 工具是只读的。OLAP 只用于从分析型数据存储中检索数据以进行决策。使用 OLAP 工具可以快速地读取和解释有组织的结构化数据，从而进行分析以及做出后续基于事实的决策。

280

术语 OLTP 和 OLAP 在互联网时代之前就出现了。词语“在线”在这两个术语中都有用到，但实际与互联网无关。相反，“在线”是指计算机立即响应用户需求的一种处理能力。如今，计算机能够快速地执行处理、更新和检索数据等操作，这已是我们所熟知的事实。然而，在创造术语 OLTP 的那个时代，多数电脑使用的仍然是硬盘驱动器出现之前的设备，例如磁带和卡片机。词语“在线”在于强调结果的即时性，即数据库系统使用直接访问的存储方式，例如硬盘驱动器，而不是顺序（慢速）访问的存储设备，例如磁带。

9.5　OLAP/BI 工具

用户通过数据库管理软件语言（如 SQL）可直接访问数据仓库。然而，另一个更常见的直接访问和分析数据的方法是使用在 OLAP 工具（OLAP Tool）。OLAP 工具也称为 BI 工具（BI Tool，商务智能工具）。在本书中，我们将这两个名字结合，命名为 OLAP/BI 工具。

OLAP/BI 工具用于维度模型数据的分析。正如第 8 章所述，无论选择哪种数据仓库方法，终端用户最终获取的数据都是按照维度模型结构组织的。因此，OLAP/BI 工具可以分析不同的建模方法所组织的数据，如图 9-8 所示。

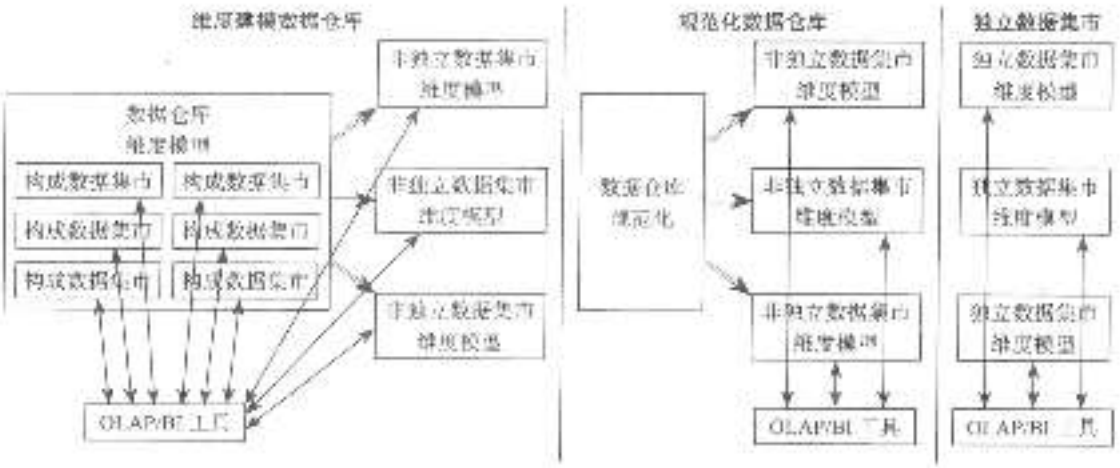


图 9-8　OLAP/BI 工具作为不同方法建模的数据仓库的接口

OLAP/BI 工具允许用户使用简单的单击式的查询程序来查询事实表和维度表。基于用户单击行为，OLAP/BI 工具使用数据仓库管理系统语言（如 SQL）编写和执行代码来支持数据仓库或数据集市查询。 [281]

9.6　OLAP/BI 工具功能

目前市场上有多种 OLAP/BI 工具（例如 Microstrategy、IBM 的 Cognos-acquired、SAP 的 Business Objects-acquired、Oracle 的 Hyperion-acquired、Microsoft 以及其他）。这里，我们给出所有 OLAP/BI 工具的一些常见功能概述。OLAP/BI 工具中分析人员经常使用的三个基本功能如下：

- 切片和切块。
- 旋转。
- 下钻和上卷。

我们将使用与 ZAGI 零售公司场景有关的实例来展示这些功能，这些实例基于图 9-7 所示维度建模数据集。

假设一个来自 ZAGI 零售公司的终端用户想使用 OLAP/BI 工具分析图 9-7 所示的涉及销售事实表中所有维度所有属性的 DollarsSold 和 UnitsSold。在这种情况下，OLAP/BI 工具会给用户提供如图 9-9 所示的接口。用户可以在维度中指定特定的 DollarsSold 或 UnitsSold 属性的具体查询分析操作，只需通过拖放方法来选择所需属性即可。 [282]

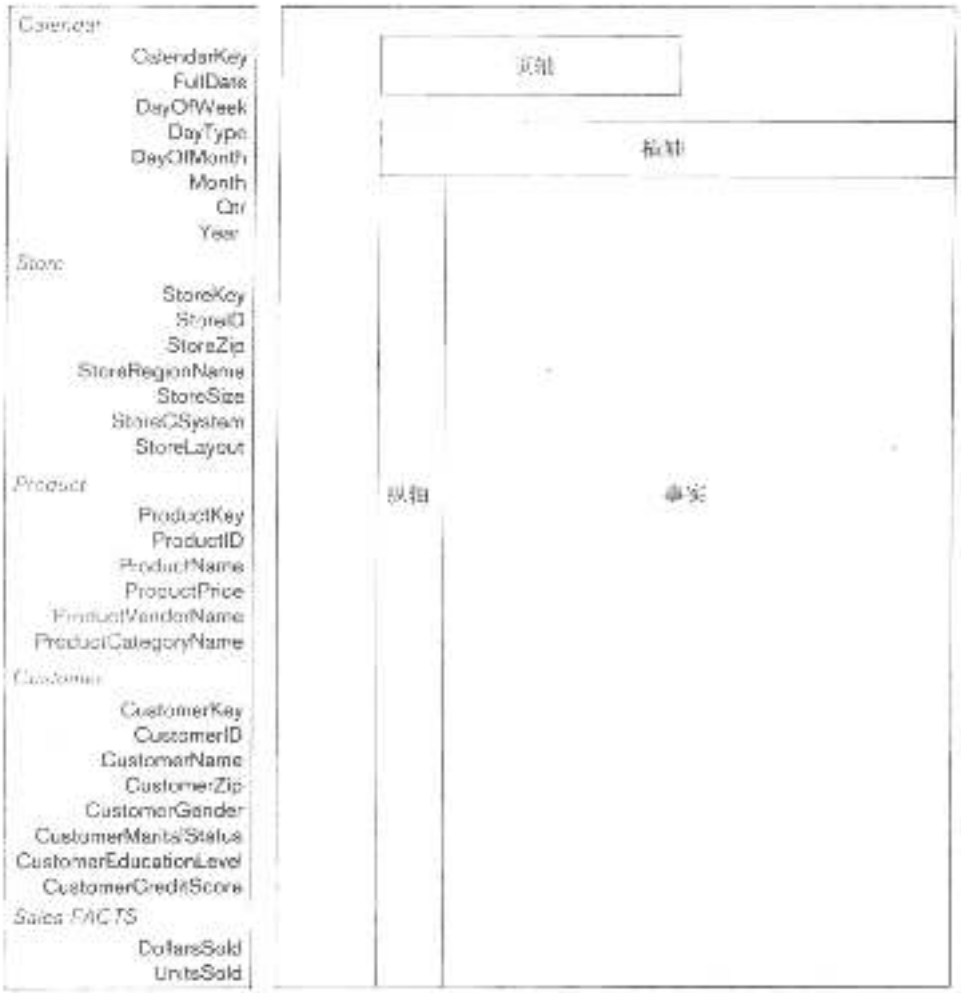


图9-9　一个典型的OLAP/BI查询构件空间

例如，假设分析人员想显示以下查询结果。

OLAP Query1：对每一个商店，分别显示每一类产品在男性（Male）和女性（Female）购物者中的销售数量。

在OLAP/BI工具中指定这种查询操作很简单。用户在图形界面接口中选择查询所需的属性，拖动并将其放到图形界面的查询构件空间中。OLAP/BI工具创建的典型的基本查询结果是一个表格报告，它的行和列标题来自维度属性的名字。数据内容来自事实表。这些数据按照满足查询条件的方式进行组织。

OLAP/BI工具的典型查询构件空间包含纵轴、横轴、页轴以及事实区域。通过拖动纵轴的维度属性，用户指定的与纵轴维度相关的结果就会在行上显示。通过拖动横轴的维度属性，用户所指定的与横轴维度相关的结果将会在列上显示。

我们来观察一下查询构件空间是如何使用的。在OLAP Query1中，用户可能会拖动STORE维度中的StoreID属性、CUSTOMER维度中的Gender属性、PRODUCT维度中的Category属性到一个轴上。例如，StoreID和Gender放到纵轴，Category放到横轴。同样，

SALES 事实表中的 UnitsSold 属性可以拖放到查询构件空间的事实区。如图 9-10 所示。

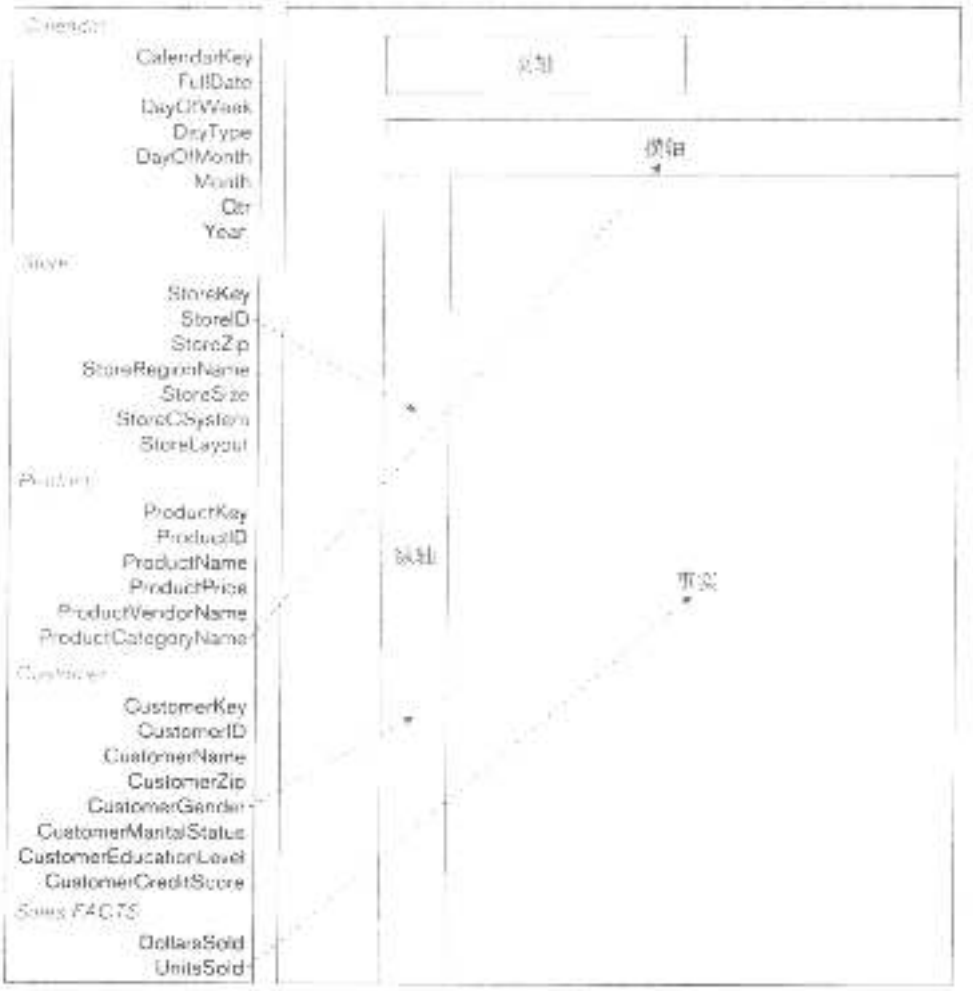


图 9-10　对于 OLAP Query1，OLAP/BI 工具的查询构件动作

查询一旦建立（如图 9-10 所示），紧接着便会执行，其结果会在屏幕上进行显示。图 9-11 展示了一个典型的 OLAP/BI 工具将如何显示 OLAP Query1 中的结果。

当屏幕显示出查询结果时，如图 9-11 所示，用户便可以选择执行 OLAP 的三个基本功能：切片和切块、旋转、下钻和上卷中的任意一个。

对于一个典型的 OLAP/BI 工具而言，切片和切块、旋转以及下钻和上卷功能是简单的单击和拖放操作的组合。现在我们将描述和说明这些功能。

		Sales-Units Sold	
		Camping	Footwear
Store 1			
	Female	1	0
	Male	1	0
Store 2			
	Female	0	3
	Male	5	7
Store 3			
	Female	3	8
	Male	3	4

图 9-11　OLAP Query1 的结果

9.6.1　切片和切块

切片和切块（slice and dice）操作是从已经显

示的结果里增加、替换或者消除指定的维度属性（或者是消除维度属性中的指定值）。例如，用户使用OLAP/BI工具对OLAP Query1进行修改，只显示商店1和商店2中的查询结果。换句话说，商店3中的查询结果将从图9-11所示的原始查询结果中“切除”。修改后的查询如下：

OLAP Query2：对商店1和商店2，分别显示每一类产品在男性和女性购物者中的销售数量。

图9-12展示了OLAP Query2的结果，针对图9-11的结果创建了一个切片和切块操作。

除了消除维度属性或者选择维度属性值，切片和切块操作还可以替换和增加维度属性。下面的例子说明了这一过程，用户通过用CALENDAR维度中的DayType属性替换PRODUCT维度中的ProductCategory属性来修改图9-11中的查询结果。修改后的查询语句如下：

283 / 284

OLAP Query3：对于每一个商店，分别显示男性和女性购物者在工作日和周末/节假日购买商品的数量。

图9-13展示了OLAP Query3的结果，该查询是通过对图9-11所示的查询进行另一种切片和切块操作得到的。

		Sales-Units Sold	
		Camping	Footwear
Store 1			
	Female	1	0
	Male	1	0
Store 2			
	Female	0	3
	Male	3	7

图9-12　切片和切块的第一个例子：OLAP Query2的结果

		Sales-Units Sold	
		Weekend/ Holiday	Workday
Store 1			
	Female	1	0
	Male	0	1
Store 2			
	Female	0	3
	Male	1	9
Store 3			
	Female	0	11
	Male	0	7

图9-13　切片和切块的第二个例子：OLAP Query3的结果

9.6.2　旋转

与切片和切块操作不同，**旋转**（pivot/rotate）操作不改变原始查询结果的值，它仅仅是对结果形式的重新组织。

在接下来的旋转操作的例子中，将使用图9-11所示的查询结果，交换ProductCategory属性和CustomerGender属性所在轴的位置。这种旋转操作的结果如图9-14所示。

由于旋转操作不改变用户看到的结果值，所以图9-11和图9-14所示的查询语句相同。

OLAP Query1：对每一个商店，分别显示每一类产品在男性和女性购物者中的销售数量。

		Sales-Units Sold	
		Female	Male
Store 1			
	Camping	1	1
	Footwear	0	0
Store 2			
	Camping	0	3
	Footwear	3	7
Store 3			
	Camping	3	3
	Footwear	6	4

图9-14　旋转的第一个例子

在图 9-11 中，ProductCategory 在横轴，CustomerGender 在纵轴。而在图 9-14 中，执行了旋转操作，ProductCategory 被旋转到纵轴，而 CustomerGender 被移到横轴。 285

除了纵轴和横轴，大多数 OLAP/BI 工具都有一个特性：所谓的“页轴”。这是一个在旋转操作中可以使用的额外的轴。图 9-15 展示了将图 9-14 所示查询中的属性 CustomerGender 从横轴旋转到页轴的结果。如图 9-15 所示，页轴允许查看页轴属性中的单值属性。在本例中，表中只给出了女性购物者购买不同商店每种产品的数量。

通过单击 OLAP/BI 工具上的菜单栏，用户可以轻松地将一个属性值（女性，如图 9-15 所示）切换到另一个属性值（男性，如图 9-16 所示）。相应地，图 9-16 只显示了男性购买不同商店每种产品的数量。

		Sales Units Sold Female
Store 1		
	Camping	1
	Footwear	2
Store 2		
	Camping	0
	Footwear	3
Store 3		
	Camping	5
	Footwear	6

图 9-15　旋转的第二个例子：页轴

		Sales Units Sold Male
Store 1		
	Camping	1
	Footwear	2
Store 2		
	Camping	3
	Footwear	7
Store 3		
	Camping	3
	Footwear	4

图 9-16　旋转的第二个例子：页轴（另一个值）

9.6.3　下钻和上卷

下钻（drill down）的目的是使得查询结果中的数据粒度更精细，而**上卷**（drill up）则是使其更粗略。 286

接下来使用图 9-11 所示的查询结果，将 RRODUCT 维度从商品类别（ProductCategory）下钻到商品名（ProductName）。展开后的查询语言表述如下。

OLAP Query4：对每一个商店，分别显示男性和女性购买者购买每一类商品中的每一种产品的数量。

图 9-17 给出了对图 9-11 所示结果进行下钻操作后得到的 OLAP Query4 的结果。

		Sales Units Sold					
			Camping			Footwear	
		Biggy Tent	Tiny Tent	Zzz Bag	Cosy Sock	Dora Boot	Easy Boot
Store 1							
	Female	0	0	1	0	0	0
	Male	0	0	1	0	0	0
Store 2							
	Female	0	0	0	2	0	1
	Male	0	0	3	4	2	1
Store 3							
	Female	0	0	3	5	1	2
	Male	1	2	0	0	4	2

图 9-17　下钻的例子：OLAP Query4 的结果

下钻操作允许用户通过不同维度层次观察数据。一个层次是维度中的一组属性，在较低层次时对应一个或多个属性，而在较高层次时只有一个属性。例如，ProductName 和 ProductCategory 在一个层次里，因为每个 ProductName 都属于一个 ProductCategory，而一个 ProductCategory 可以包含有多个 ProductName。另外，考虑 Store 维度的下列属性：StoreID、StoreZIP 以及 StoreRegionName。这三个属性存在如下层次结构 StoreRegion → StoreZIP → StoreID，因为：

- 一个 StoreID 对应一个 StoreZIP 和一个 StoreRegionName。
- 一个 StoreZIP 对应一个 StoreRegionName，但是可以有多个 StoreID 值。
- 一个 StoreRegionName 可以有多个 StoreZIP 值和一系列 StoreID 值。

多个商店可以有一样的邮政编码，同样，多个邮政编码可以有一样的区域码。一个邮政编码对应一个区域，一个商店有一个邮政编码，因此，它也属于一个区域。这样，一个邮政编码中的销售情况可以分散到每个商店的销售情况（下钻）或者将其合并到一个区域里（上卷）。

一些维度可以有多个层次，例如，STORE 维度有下列层次结构：

StoreRegionName → StoreZIP → StoreID

StoreSize → StoreID

StoreSystem → StoreID

StoreLayout → StoreID

上面描述的层次结构也称为钻层次结构。它们允许用户把一个层次的一个值在低层展开进行详细显示（下钻），或者将细节折叠只观察高层值（上卷）。

为了显示上卷操作，我们可以将图 9-17 作为查询的起点，然后将图 9-11 中的商品从商品名（ProductName）上卷到商品种类（ProductCategory）。

287

9.6.4　OLAP/BI 工具附加功能概述

进一步分析上述例子可以得出，维度模型是 OLAP 所必需的。底层数据必须按照维度结构进行组织，且事实表将作为连接其他一系列维度表的中心。若数据不按照维度方式组织，三个 OLAP 基本操作将不能有效地或者正确地执行。

除了切片和切块、旋转以及下钻和上卷等功能外，现代 OLAP/BI 工具还包含一些其他功能。现代 OLAP/BI 工具的一个标准特性便是通过图形化方式对结果进行可视化。例如，现在的任何一个 OLAP/BI 工具都可以将图 9-11 所示的查询结果按照图 9-18 的方式进行可视化展示。

除了一些标准操作（旋转、切片和切块、下钻和上卷）以及多样的图形可视化功能外，OLAP/BI 工具还可以创建和检查计算后的数据，确定相似或相对差异，进行异常分析、趋势分析、预测和回归分析，以及其他有用的分析功能。然而，这些功能在其他非 OLAP 应用程序里也包含有，例如，统计工具和电子制表软件。OLAP/BI 工具与其他应用程序的本质区别在于 OLAP/BI 工具很容易实现与维度建模的数据集市和数据仓库进行交互操作，并且，可在大规模数据上进行 OLAP 操作。

现在的电子制表软件包中包含一些功能，这些功能可以在它们能存储的有限数据集中执行基本的 OLAP 功能。电子制表软件可存储的数据比数据仓库或数据集市存储的数据量少得多。然而，一些电子制表工具通过 DBMS 工具配置后可以直接访问维度建模的数据集市和数据仓库。按照这种方式配置的电子制表软件实际上可以视为 OLAP/BI 工具。

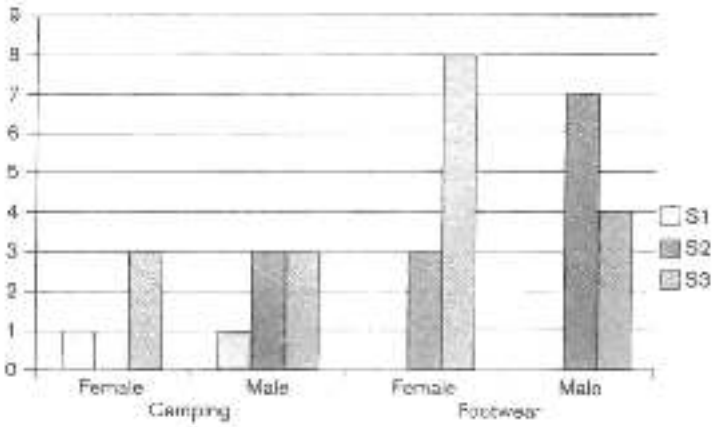


图 9-18 使用图表可视化 OLAP Query1 的结果

许多现存的 OLAP/BI 工具是基于网页的，它省去了在终端用户工作站上安装软件的过程，而是通过 Web 浏览器向终端用户提供查询构造空间，如图 9-9 所示。

OLAP/BI 工具可以基于不同的架构方法。在本章的结尾，我们将会简要地阐述最常见的 OLAP/BI 工具架构。

9.7 OLAP/BI 工具用途

OLAP/BI 工具有下列两个用途：特定地直接分析维度建模数据，以及创建前端应用程序间接地访问维度建模数据。 288

特定的直接分析发生在用户通过 OLAP/BI 工具访问数据仓库中的数据，以及执行旋转、切片和钻取等操作时。这个操作有时也称为数据探查，因为一个查询结果常常会引起另外一个新的问题。例如，分析员使用 OLAP/BI 工具创建一个查询，这个查询的结果可能会促使另外一个问题产生，随后通过对最初的结果进行旋转、切片或钻取操作以得到另一个结果，从而解决问题。

除了特定的直接分析，OLAP/BI 工具也常常用于创建数据仓库前端应用程序。数据仓库前端应用程序可以是由专业用户使用 OLAP/BI 工具创建的一系列查询的简单集合。这样的查询集合可以放在导航菜单栏中，提供给那些没有访问权限、没有专业知识或者没有时间直接使用 OLAP/BI 工具的数据仓库用户使用。

9.8 数据仓库 / 数据集市前端（BI）应用

大多数情况下，部分数据仓库的潜在用户（通常占用户的大多数）缺少时间或专业知识去使用开放式的数据分析工具。期望每一个在工作中使用数据仓库的用户都能够自己编写 SQL 代码或者使用专门的 OLAP/BI 工具是不可能的。相反，一些数据仓库和数据集市的用户常通过前端应用程序访问数据。数据仓库 / 数据集市前端应用程序通常涉及商务智能（BI）应用程序。图 9-19 展示了一个带有前端应用程序的数据仓库系统。

如图 9-19 所示，数据仓库系统的前端应用程序既可以检索数据仓库自身的数据，也可以检索依赖数据仓库中的数据子集的数据集市。当一个公司将其分析的数据存储在一个独立的数据集市中时，前端应用程序（BI）也可以检索独立数据集市中的数据。

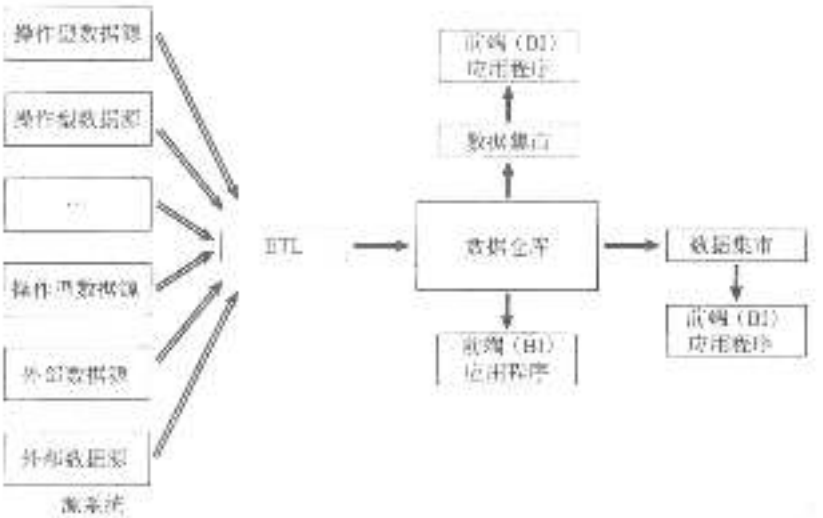


图 9-19　一个带有前端应用程序的数据仓库系统

289

数据仓库/数据集市前端应用程序是将预先开发好的查询集合组织起来，以供终端用户进行简单的访问、导航以及使用。图 9-20 展示了一个前端数据仓库应用程序集合的界面。

在这个例子中，提供的每一个应用程序都包含一系列预先开发的查询来检索数据，这些数据是应用程序指定的相关数据仓库中的一部分。例如，选择图 9-20 中销售比较分析选项，程序将给出图 9-21 所示的界面，该界面包含来自数据仓库的一系列查询检索后的销售分析数据。

图 9-20　一个数据仓库前端应用程序集合的界面

图 9-21　一个数据仓库前端应用程序的界面

选择图 9-21 中按季度进行商店销售比较选项，打开一个查询集合，这个集合以交互式的表单呈现出来，如图 9-22 所示。

图 9-22　预先开发的数据仓库查询界面

我们假设在图9-22中，用户选择如下所示的参数。

年份1: 2012
年份1中的季度: Q4（第4季度）
年份2: 2013
年份2中的季度: Q3（第3季度）
商店: 商店1
商店: 商店3
商店: 商店7
商店: <未选择>
商店: <未选择>

290

一旦图9-22中所示的参数选择好后，单击"运行"按钮，则将生成包含结果表以及图表的报告，如图9-23所示。

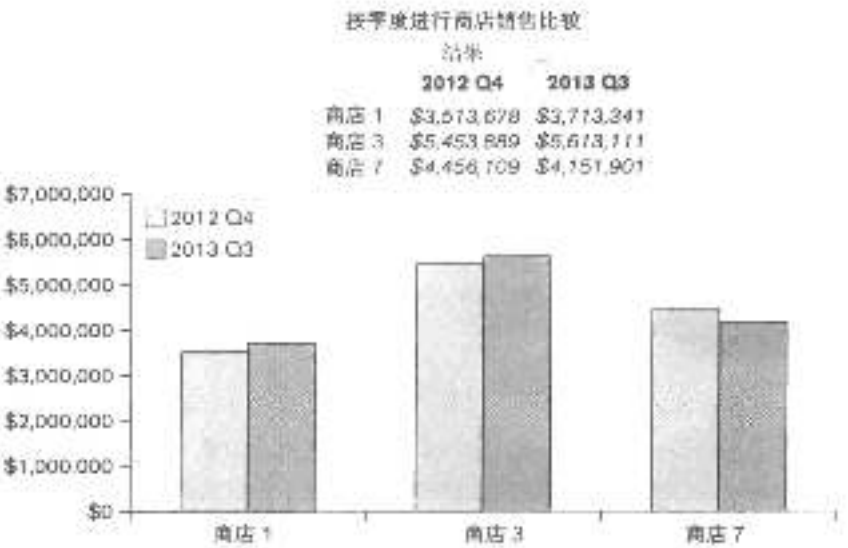


图9-23 指定图9-22中参数值后的查询结果

开发前端应用程序包含创建一系列预先开发的查询集合以及查询接口。开发前端应用程序过程可以结合使用OLAP/BI工具、其他报告工具或者从头开始编写代码。例如，多数OLAP/BI工具包含创建前端应用程序的功能，如上例所示。注意观察图9-22所示的界面，它能够查询基于维度建模的数据（按图9-1所示的星形模式建模）。图9-22中的查询界面是基于图9-9所示的OLAP/BI工具创建的。使用图9-9所示的界面，开发者可以拖拽属性Year和Quarter到横轴上，属性StoreID到纵轴上，以及Sales事实表中的属性DollarsSold到事实空间。这种查询建立好后可以得到图9-22所示的应用程序，使用这个应用程序可以切除所有未被用户选择的年、季度以及商店。

291

同OLAP/BI工具一样，访问前端应用程序也可以基于网页进行。基于网页的前端应用程序省去了终端用户在工作站上安装软件的步骤，而是通过网页浏览器向终端用户提供访问接口。

9.9 管理展示板

图 9-24 给出了数据仓库前端应用程序的一个特定形式的界面，称为管理展示板。

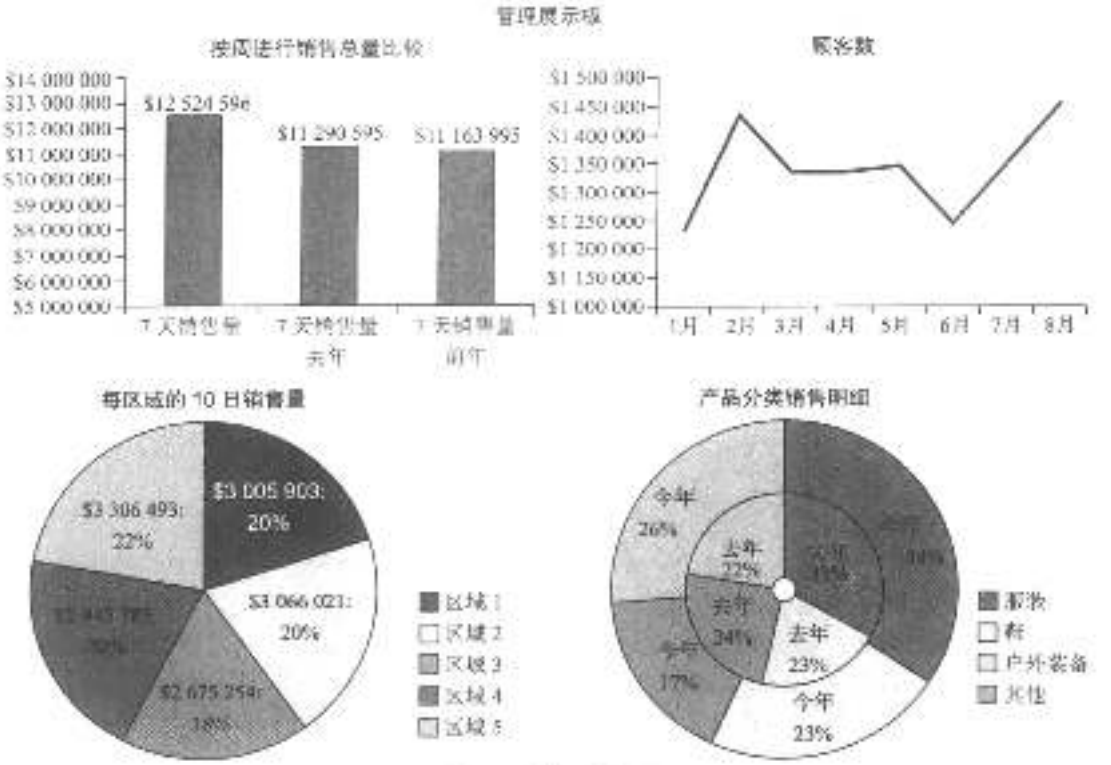


图 9-24 管理展示板

管理展示板（executive dashboard）这个术语表明它适合组织中的高层决策者使用。它包含一系列有组织的、易读的、描述组织性能的重要查询。一般情况下，使用展示板基本上不需要训练和学习。同其他前端数据仓库应用程序一样，管理展示板也可以是基于网页的。

9.10 数据仓库部署

数据仓库部署允许终端用户访问数据仓库中新创建和组织的数据以及使用数据仓库的前端应用程序。一旦数据仓库系统开发完成，终端用户就可以用它来完成从数据仓库中获取数据的任务。部署往往不会针对所有的潜在用户一次性地完成需求，相反，部署过程是逐渐进行的。

292

同其他多数信息系统相似，数据仓库系统在面向全部终端用户部署之前，通常要经历一个测试阶段。在发布所谓的 α 版（alpha release）期间，数据仓库以及相关的前端应用程序面向内部开发团队成员部署配置，对其功能进行初步测试。后续发布的数据仓库系统版本（所谓的 β 版（beta release））会选择一组用户对系统的可用性进行测试。在实际部署数据仓库系统之前，会根据测试阶段反馈回来的意见对系统进行修改。实际部署的数据仓库系统称为**发行版**（production release）。

本章从业务设计者和使用者的角度描述了大部分有关数据仓库实现和使用的基础性问

题。接下来将进一步加讲述一些有关 OLAP/BI 工具的数据库模型以及有关 OLAP/BI 架构选择的信息。

9.11　问题说明：OLAP/BI 工具数据库模型

如前文所述，OLAP/BI 工具是为访问维度模型数据而设计的。本节将给出两种不同的数据仓库模型的概要概述，它们用于存储维度模型数据：

- 关系数据库模型；
- 多维数据库模型。

回忆一下，关系数据库模型是现代关系 DBMS 软件包的基础，它被用来实现当前多数的操作型企业数据库。关系 DBMS 软件包括 Oracle、MySQL、Microsoft SQL Server、PostgreSQL、IBM DB2 以及 Teradata[⊖]。如第 3 章所述，关系数据库模型是一个二维表的集合，表中的每一行代表数据库中的一条记录。

关系模型不是实现维度建模数据仓库的唯一方式，另一个可以实现维度建模数据的方式是**多维数据库模型**（multidimensional database model）。在这个模型里，数据库是所谓的**立方体**（cube）的集合。从概念上来看，关系模型和数据立方体在维度模型中没有区别，概念设计仍然要创建具有事实表和维度表的星形模式，其区别在于物理实现阶段。为了展示多维数据模型并将其与关系模型进行比较，我们将使用图 9-25 所示的例子。 293

图 9-25　一个简单的维度模型

图 9-25 展示了一个具有二个维度表和一个事实表的星形模式。图 9-26 给出了图 9-25 中按关系方式组织的事实表。事实表包含所有连接维度表的码，以及数值度量 DollarsSold。一个记录对应特定日期特定商店特定产品的 DollarsSold 值。

SALES 事实表

CalendarKey	ProductKey	StoreKey	DollarsSold
1	1	1	$100
1	2	1	$70
1	3	1	$15

图 9-26　图 9-25 中事实表的关系型实现

多维数据库模型可以用具有多个维度的数据立方体来表示。与几何上的立方体不同，多维数据立方体可以多于三个维度。然而，为简单起见，我们的例子只使用三个维度。

图 9-27 展示了图 9-25 中按多维方式实现的事实表。立方体的每一格对应特定日期特定

⊖ Teradata 是专门针对大数据仓库而开发的 DBMS 的一个例子。而这里提及的其他 DBMS 既支持操作型数据库又支持数据仓库。

无法确定产品的 DollarsSold 值。为了更清楚地展示，只显示了数量值为 $100 的一格，它对应图 9-26 中 SALES 事实表的第一行记录。

两个模型本质上的不同点是数据存储方法。在关系模型中，存储一条记录至合找事实表，查找速度跟许多因素有关，例如如何存储记录或者表中有没有索引。在多维立方体中，每一条记录都可以直接查看，从而省去了搜索过程。因为每一格都有一个地址存储维度属性值，所以可以直接访问。为了找到数据，维度值可用于计算地址以及直接访问每一格的值。例如，假设进行以下两个检索：

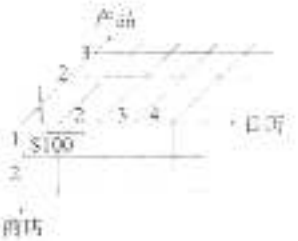


图 9-27　图 9-26 中 SALES 事实表的多维（立方体）实现

294

- 检索本例中商店 1，产品 1，日期（天）1——可以表示成（1，1，1）的 DollarsSold 值。
- 检索本例中商店 1，产品 3，日期（天）1——可以表示成（1，3，1）的 DollarsSold 值。

在立方体中，维度值（1，1，1）直接对应销售数量值 $15，换句话说，检索过程将不用搜索其他记录。另一方面，在关系型实现中，得到目标记录需要读值，并将值与其他记录进行比较。例如，在按照关系表组织的事实表中，如图 9-26 所示，线性搜索（1，3，1）涉及将值（1，3，1）与表中的第一条记录（1，1，1）、第二条记录（1，2，1）进行比较，直到找到匹配的第三条记录。

当然，在实际的数据仓库数据集中，搜索的空间更大，因此，搜索的速度是影响数据仓库性能和可用性的关键因素。立方体的直接访问方式是提高搜索速度的一种方法。如第 6 章所述，关系表的搜索速度可以通过使用排序/索引以及不同的搜索算法来得到显著提高。

9.12　问题说明：OLAP/BI 工具数据架构方法

根据实现维度表和事实表的数据仓库模型的不同，可以选择不同的 OLAP/BI 工具架构方法。本节将描述三种常见的方法：MOLAP、ROLAP 以及 HOLAP。

9.12.1　MOLAP

典型的**多维在线分析处理**（multidimensional online analytical processing，MOLAP）架构如图 9-28 所示。MOLAP 中的数据来自数据仓库或者来自存储在多维立方体中的操作型数据源。底层数据的复杂性对 MOLAP 工具使用者是隐藏的。换句话说，用户通过通常的单击方式便可使用其常用功能，如图 9-10 所示。用户不用了解立方体是如何建立的，也不用了解它与关系表的不同点。MOLAP 服务执行查询操作，并且将结果集合发送给终端用户的 OLAP/BI 工具，以图 9-11～图 9-18 所示的方式呈现给用户。如果 OLAP/BI 工具作为前端应用程序的基础，按图 9-22 所示指定查询操作，则将得到图 9-23 所示的显示结果。

图 9-28　一个典型的 MOLAP 架构

一般地，MOLAP服务包含有限的数据量。MOLAP的主要特性是可以快速进行分析。MOLAP服务会尽可能多地预先计算结果并且将这些结果存储在立方体中。

需要强调的是，在分析聚合数据时，MOLAP立方体的执行效果很好，但是它们不适合处理事务层次的细节数据。正如本书前面章节所述，事务层次的细节数据是最细粒度的数据，它的每一条记录对应现实生活中唯一确定的事务。立方体往往受到空间限制，一个典型的组织机构中事务层次的数据量常会超过立方体的存储容量。即使事务层次数据在某种程度上适合立方体，还会存在其他问题，如数据稀疏性，使得立方体不适合处理非聚合数据。在事务数据中，大多数维度的交集是空集，例如，如果商品码为2的商品没有在商店码为2、日期码为1的记录中出现，那么格（2，2，1）就是空的。 [295]

由于立方体的每一格对应一个直接地址并且常用的查询结果已被预先计算，所以MOLAP的查询速度往往很快。计算引擎通过公式和转换可以从存在的数据中提取新的信息。预先聚集的汇总数据和预先计算结果的方法使分析复杂的数据关系变得快速和简单。查询"处理"过程经常可简化为直接查找数据。

尽管MOLAP检索数据非常有效，但是立方体的更新非常慢。数据加载要花费几个小时，立方体的计算则要花费更多时间。这是因为，每当有新的有效数据时，立方体就需要重新加载和重新计算，如图9-28所示。然而，分析查询结果的速度往往比加载新数据和创建更新立方体的速度更为重要。

一般情况下，加载到MOLAP服务器上的数据来自关系DBMS平台的数据仓库。然而，存在这种情况，操作型数据源的数据直接被加载到立方体中以满足当前数据分析的需要。在这种情况下，立方体本身就是一个数据集市，没有数据仓库。

9.12.2 ROLAP

OLAP/BI工具的另一个主要种类是关系型OLAP/BI工具，通常称作**关系型在线分析处理**（relational online analytical processing，ROLAP）工具。典型的ROLAP框架的高层视图如图9-29所示。ROLAP工具也提供本章之前描述的常用OLAP功能。查询通过标准的单击方式进行，如图9-10所示。ROLAP服务将查询转换成SQL声明，SQL查询被发送到RDBMS支持的数据仓库。RDBMS执行查询，并将查询的结果集合发送到ROLAP服务器上，最终交给OLAP/BI工具终端用户。OLAP/BI工具按类似图9-11～图9-18的方式将结果呈现给用户。如果OLAP/BI工具作为终端用户应用程序的基础，可按图9-22指定查询，结果将按图9-23呈现。

图9-29 一个典型的ROLAP架构

ROLAP架构对数据库大小以及可能要执行的分析类型完全没有限制。然而，由于结果没有被预先执行，其查询性能没有MOLAP那么快。

选择MOLAP和ROLAP时需要在查询性能和存储之间进行权衡。ROLAP能够处理更大的数据量，这使得它更适合处理事务层次的详细数据。同时，RDBMS软件也在不断提升查询处理的速度以减小MOLAP与ROLAP之间的性能差距。 [296]

9.12.3 HOLAP

混合型在线分析处理（Hybrid online analytical processing，HOLAP）架构结合了 MOLAP 和 ROLAP 两种方法。典型的 HOLAP 架构如图 9-30 所示。

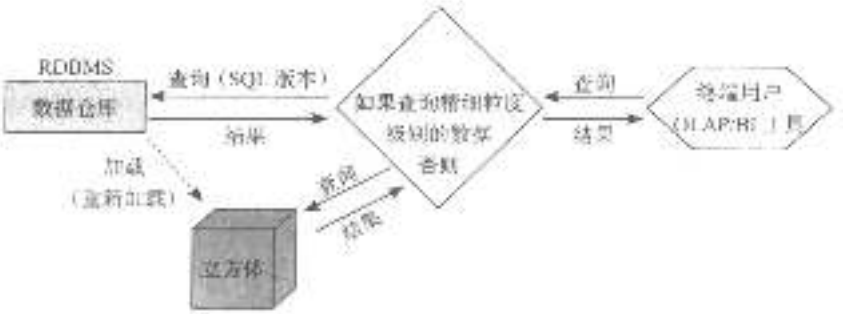


图 9-30 一个典型的 HOLAP 架构

HOLAP 结合了以上两种方法的优点。在混合解决方案中（如图 9-30 所示），关系数据库可以存储详细的事务层次数据，多维立方体可以存储汇总数据。如果 OLAP/BI 终端用户要查询精细粒度级别的细节数据，就可以针对关系数据库进行查询。另一方面，若查询存储在立方体中的汇总数据，就可以针对立方体进行查询。

关键术语

active data warehouse（动态数据仓库），280
alpha release（α 版），293
beta release（β 版），293
business intelligence (BI) tool（商务智能（BI）工具），281
cube（立方体），293
data cleansing/scrubbing（数据清洗/清理），279
drill down（下钻），286
drill up（上卷），286
executive dashboard（管理展示板），292
frist load（首次加载），280
hybrid online analytical processing, HOLAP（混合型在线分析处理），297
multidimensional database model（多维数据库模型），293
multidimensional online analytical processing, MOLAP（多维在线分析处理），295
OLAP/BI Tool（OLAP/BI 工具），281
online analytical processing, OLAP（在线分析处理），280
OLAP tool（在线分析处理工具），281
online transaction processing, OLTP（在线事务处理），280
pivot/rotate（旋转），285
production release（发行版），293
refresh cycle（刷新周期），280
refresh load（刷新装载），280
relational online analytical processing, ROLAP（关系型在线分析处理），296
slice and dice（切片和切块），283

复习题

Q9.1 在创建数据仓库的过程中如何使用数据库管理工具？
Q9.2 简要描述提取过程。
Q9.3 简要描述转换过程。
Q9.4 数据汇流的目的是什么？
Q9.5 简要描述加载过程。

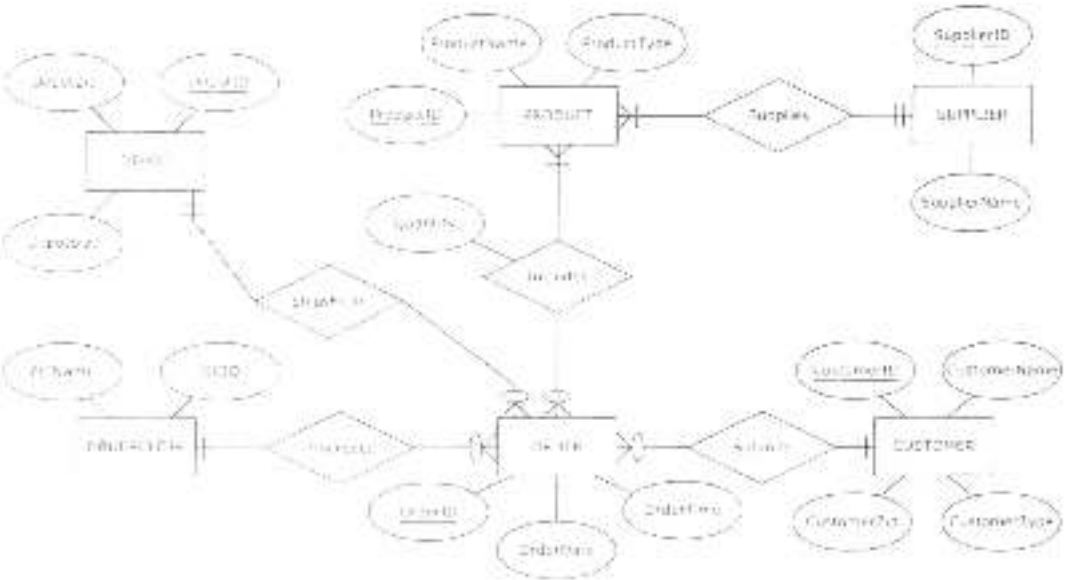
Q9.6 首次加载和刷新加载之间的区别是什么？
Q9.7 OLAP 和 OLTP 之间的区别是什么？
Q9.8 OLAP/BI 工具的一个基本功能是什么？
Q9.9 OLAP/BI 工具的两个主要用途是什么？
Q9.10 数据仓库/数据集市前端应用程序的用途是什么？
Q9.11 管理展示板是什么？
Q9.12 简要描述数据仓库部署过程。
Q9.13 关系和多维数据库模型之间的区别是什么？
Q9.14 简要描述 MOLAP 架构。
Q9.15 简要描述 ROLAP 架构。
Q9.16 简要描述 HOLAP 架构。

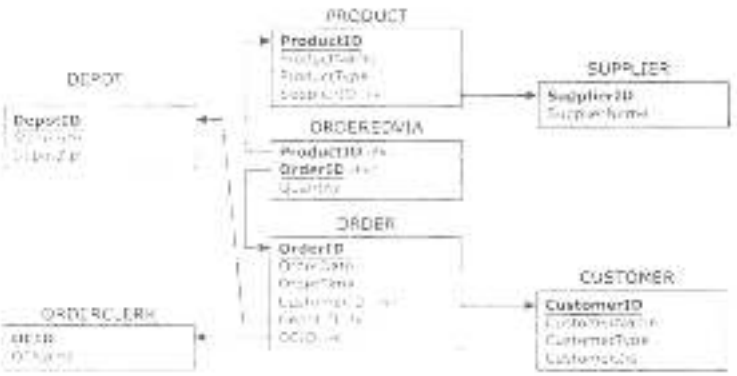
297

练习

E9.1 考虑以下场景：Big Z 公司是一家汽车产品批发商。
Big Z 公司想创建一个分析型数据库（数据仓库）来分析订单数量。
两个有效的数据源，数据源 1 和数据源 2，描述如下：
数据源 1：Big Z 公司订单数据库，如图 9-31 所示。
数据源 2：Big Z 公司人力资源部门表，如图 9-32 所示。
数据仓库必须对如下订单量进行分析：

- 日期，包括：
  - 完整的日期；
  - 星期；
  - 日；
  - 月；
  - 季度；
  - 年份。
- 时间
- 商品，包括：
  - 商品 ID；
  - 商品名；
  - 商品类型；
  - 商品供应商名；
- 顾客，包括：
  - 顾客 ID；
  - 顾客名；
  - 顾客类型；
  - 顾客邮政编码。
- 仓库，包括：
  - 仓库 ID；
  - 仓库大小；
  - 仓库邮政编码。
- 订单员，包括：
  - 订单员 ID；
  - 订单员名字；





DEPOT

DepotID	DepotSize	DepotZip
D1	[illegible]	[illegible]
D2	[illegible]	[illegible]
D3	[illegible]	[illegible]

PRODUCT

ProductID	ProductName	ProductType	SupplierID
P1	[illegible]	[illegible]	S1
P2	[illegible]	[illegible]	S1
P3	[illegible]	[illegible]	B1

SUPPLIER

SupplierID	SupplierName
S1	[illegible]
B1	[illegible]

ORDERCLERK

OCID	OCName
OC1	[illegible]
OC2	[illegible]
OC3	[illegible]

CUSTOMER

CustomerID	CustomerName	CustomerType	CustomerZip
C1	[illegible]	Repair Shop	[illegible]
C2	[illegible]	Repair Shop	[illegible]
C3	[illegible]	Retailer	[illegible]

ORDEREDVIA

ProductID	OrderID	Quantity
P1	O1	4
P2	O1	8
P1	O2	12
P2	O2	4
P3	O4	7
P3	O5	5
P2	O5	8
P1	O6	4
P1	O7	8
P3	O7	6
P1	O8	6
P2	O8	6

ORDER

OrderID	CustomerID	DepotID	OCID	OrderDate	OrderTime
O1	C1	D1	OC1	[illegible]	[illegible]
O2	C2	D1	OC1	[illegible]	[illegible]
O3	C1	D2	OC1	[illegible]	[illegible]
O4	C3	D2	OC1	[illegible]	[illegible]
O5	C2	D3	OC3	[illegible]	[illegible]
O6	C3	D3	OC3	[illegible]	[illegible]
O7	C1	D2	OC3	[illegible]	[illegible]
O8	C1	D2	OC3	[illegible]	[illegible]

图 9-33　数据源 1：Big Z 公司订单数据库

HUMAN RESOURCES DEPARTMENT TABLE (EMPLOYEE DATA)

EmployeeID	Name	Title	EducationLevel	YearofHire
OC1	Antonio	Order Clerk	High School	2001
OC2	Wesley	Order Clerk	College	2005
OC3	Liliana	Order Clerk	College	2005

图 9-32　数据源 2：Big Z 公司人力资源部门表

- 订单员头衔；
- 订单员教育级别；
- 订单员雇佣年限。

E9.1a　基于以上列出的数据源和需求，针对 Big Z 公司建立一个用于维度建模数据仓库的维度模型。（E9.1a 的结果与 E8.3 的结果相同——若你已经完成了 E8.3，则可以使用该结果作为 E9.1a 的结果。）

E9.1b　使用图 9-31 和图 9-32 的数据，讲述 E9.1a 维度模型中的表是如何组织的。给出以下假设：

- 在 ETL 过程中，若存在同一个人有多个名字的情况，使用最长的名字。
- 在 ETL 过程中，识别 Big Z 订单数据库中同一个记录拥有两个不同订单 ID（O7 和 O8）的情况。

E9.2　使用图 9-11 所示的 OLAP 查询 1（基于图 9-7 中的数据），给出一个旋转操作的例子。

E9.3　使用图 9-11 所示的 OLAP 查询 1（基于图 9-7 中的数据），给出一个切片和切块操作的例子。

E9.4　使用图 9-11 所示的 OLAP 查询 1（基于图 9-7 中的数据），给出一个下钻操作的例子。

E9.5　使用图 9-11 所示的 OLAP 查询 1（基于图 9-7 中的数据），给出一个上卷操作的例子。

更多练习可访问 dbtextbook.com。

298 ~ 300

第三部分

Database Systems: Introduction to Databases and Data Warehouses

其他主题

DBMS 功能与数据库管理概述

10.1 引言

本书的基本任务是讨论现代操作型与分析型数据库的设计与使用等相关问题。这一任务已经反映在了前面的各章内容中，关注重点是数据库的需求分析、实现及商业使用。

有关数据库软硬件性能、数据库物理存储细节、数据库系统管理及其他更多技术性问题的讨论已超出了本书的研究范围。而本章将仅给出 DBMS 组件及一些数据库最基本管理任务的简要概述。

10.2 DBMS 组件

DBMS 主要用于创建数据库，完成数据库中的数据操作（如数据插入、存储、恢复、更新及删除），以及对数据库的维护。一些 DBMS 包也可用于创建前端应用。图 10-1 中给出了典型 DBMS 软件的主要组件。

图 10-1 典型 DBMS 软件的组件

数据定义组件（data definition component）可供数据库设计者创建数据库的基础部件，如数据表以及连接数据表的参照完整性约束等。第 5 章中 DDL（数据定义语言）类型的 SQL 命令便可用于这一目的。

数据操纵组件（data manipulation component）允许终端用户插入、读取、更新及删除数据库。第 5 章中 DML（数据操纵语言）类型的 SQL 命令可用于这一目的。

数据操纵组件可以被终端用户直接使用（直接用户），或是通过前端应用间接使用（间接用户）。在间接使用时，终端用户应用与数据操纵组件间通过用户行为进行通信。在一些 DBMS 包中，数据操纵组件在同一时间往往仅由一个用户使用（包括直接或间接使用），这样的系统称为**单用户系统**（single-user system）（如 MS Acess）。在一些专业性更强的包中，数据操纵组件在同一时间能直接或间接地由多个用户使用。这样的系统称为**多用户系统**（multiuser system）（如 Oracle、MySQL、Microsoft SQL Server、PostgreSQL、IBM DB2 及 Teradata）。

数据库管理组件（database administration component）用于完成数据库系统的技术性、管理性或维护性任务，如确保系统安全、性能优化或实现系统的备份与恢复。DCL（数据控制语言）和 TCL（事务控制语言）类型的 SQL 命令将主要用于数据库管理以及其他相关的技术过程，本章将对此进行简要说明。

应用开发组件（application development component）用于前端应用的开发。在一些 DBMS 包中，该组件是 DBMS 的一部分；而在另一些包中，则是作为 DBMS 的添加组件。

10.3 数据库管理概述

数据库管理涵盖了确保一个已经部署的数据库系统正确运行的各种行为。为了实现数据库系统的监测与维护，数据管理行为包含数据安全性任务（如数据存取控制），数据备份与恢复、数据完整性保证等过程。数据库管理还包含与系统性能、标准管理、数据创建与使用政策相关的各种行为。

本章的后续部分将对以下数据管理基本任务做出简要说明：

- 数据库系统监测与维护。
- 防范非法存取的数据库安全性保护。
- 数据库备份与恢复。
- 数据完整性保护。
- 数据库性能优化。
- 数据库政策与标准的开发与实现。

10.4 数据库系统监测与维护

数据库系统发行后，控制系统的人员便由负责数据库设计、实现及部署的团队转变为了数据库管理员（database administrator，DBA）。DBA 的工作之一是监测数据库系统的使用和运行。

常规监测可以帮助 DBA 识别系统需要维护的各种情形。例如，监测结果可能让 DBA 注意到，以终端用户当前的数据输入速度，数据库系统将会在数周内耗尽硬盘空间。基于这一观测，DBA 则将做出增加数据库系统硬盘空间的决定。

另一个监测的实例是检查系统中谁正在使用什么表，以及如何使用。例如，DBA 可能会注意到有两个表被大部分用户非常频繁地加入到一个查询中，并重复地创建相同的报告。基于这一观测，DBA 则可建议数据库开发团队反规范化地将这两个表合并为一个表。这一问题的另一个处理办法是，创建一个物化视图来连接这两个表，使用户可以直接对视图进行查询。第 5 章曾讲过，普通的视图并非实际的数据表，且不对应物理存储数据。当用户存取一个视图时，用于创建视图的查询将被执行，而数据将从该查询所使用的物理表中得到。而**视图物化**（view materialization）则是指将视图创建为一个实际的物理表，这样的视图称为"物化视图"。视图物化的目的是提升被频繁使用的视图的查询性能。例如，DBA 可以建立一个自动化的视图物化，通过创建某种机制为每一个被查询了特定次数的视图再保存一个视图（作为有实际数据记录的真实表格）。物化视图可基于 DBMS 以及原始数据表的变化率进行周期性的更新。 [302]

维护行为包括对数据库软硬件资源的管理与升级。例如，DBA 会负责将数据库系统升级到一个新版的 DBMS 软件系统上，或是将数据库移动到一个新的硬件平台上。

维护行为也包括对数据库结构的升级。正如前面所提及的（图 1-6 及图 7-9），数据库需要随着其开发生命期内的任何一步改变而改变，包括在使用阶段中的改变。换言之，在数据

库系统的整个生命期内，可能会存在数据库模式改变的需要。例如，在数据库的使用阶段，一个新的关系可能需要加入到数据库中[⊖]，而是否添加该关系将取决于数据库的开发者。如果打算添加，则负责数据库系统的 DBA 将会直接参与到将这一新的数据表与现存数据库合并的工作之中。

数据字典

在数据库系统的监测与维护过程中，或是在系统的常规使用过程中，无论是对于存储在数据库中的数据还是元数据，均需要进行存取操作。其中，对于元数据的存取采用数据字典来实现。

数据字典（data dictionary）即数据库中元数据（关于数据的数据）的存储库。数据字典有时也称为元数据存储库[⊖]，包括表名、表中的列名、列的数据类型、关键字及参照完整性约束等信息。用于显示所运行数据库系统结构的元数据被 DBMS 自动地存储在数据字典中，此数据字典通常称为**目录**（catalog）。

[303] 图 10-2 的实例展示了存储在数据字典中的信息类型。

TableName	ColumnName	DataType	DataLength
Vendor	VendorId	Char	2
Vendor	VendorName	VarChar	25

图 10-2　数据字典中的一个实例

数据字典表可实现数据库关系中元数据的快速识别。图 10-2 给出的数据字典实例中，关系 Vendor 具有一个 VendorID 列，其数据类型为 Char(2)，还具有一个 VendorName 列，其数据类型为 Varchar(25)。

用户可使用 SQL 语句查询数据字典表。数据字典表中，USER_TABLES 用于提供用户可存取的所有表信息，USER_VIEWS 用于提供用户可存取的所有视图信息，USER_CONSTRAINTS 给出了用户存取的相关约束。尽管不同的 DBMS 对这些表的命名可能有所不同，但它们所提供的信息都是相似的，都具有相似的服务目标。

例如，假设使用 Oracle 创建 ZAGI 零售公司销售部的数据库（见图 3-32 或图 5-1a），则以下 SQL 查询结果将显示所有关系中所有列的列名及数据类型：

```
SELECT      table_name, column_name, data_type, data_length
FROM        user_tab_columns;
```

在 Oracle 中，USER_TAB_COLUMNS 包含了关系中列的元数据。上述查询结果显示在图 10-3 中。

对数据字典表的查询允许 DBA 及用户快速地获取数据库结构。普通用户对数据字典表的查询，其查询结果将仅包含该用户有权访问的表信息（访问控制将在后续部分讨论）。[304]

⊖ 复习一下，正确的方法程序：在实际创建新的数据表前，回退到需求阶段并完成图 1-6（用于标准数据库）或是图 7-9（用于数据仓库或数据集市）中显示的所有步骤。

⊖ 比较奇怪的是，术语“存储库”在实际中往往用于表示“元数据存储库”，但在某些时候也被简单地用于表示数据的存储。本书，我们采用“元数据存储库”的意思，以消除歧义。

TableName	ColumnName	DataType	DataLength
Vendor	VendorId	Char	2
Vendor	VendorName	VarChar2	25
Category	CategoryId	Char	2
Category	CategoryName	VarChar2	25
Product	ProductId	Char	3
Product	ProductName	VarChar2	25
Product	ProductPrice	Number	22
Product	VendorId	Char	2
Product	CategoryId	Char	2
Region	RegionId	Char	1
Region	RegionName	VarChar2	25
Store	StoreId	VarChar2	3
Store	StoreZip	Char	5
Store	RegionId	Char	1
Customer	CustomerId	Char	7
Customer	CustomerName	VarChar2	15
Customer	CustomerZip	Char	5
SalesTransaction	TId	VarChar2	8
SalesTransaction	CustomerId	Char	7
SalesTransaction	StoreId	Varchar2	3
SalesTransaction	TDate	Date	7
SoldVia	ProductId	Char	3
SoldVia	TId	VarChar2	8
SoldVia	NoOfItems	Number	22

图 10-3　ZAGI 零售公司销售部数据库的数据字典实例

10.5　数据库安全：防范非法存取

防范数据的非法存取是数据库管理中的首要任务，这一任务需要对数据的存取过程加以控制。在多用户数据库系统中，数据库存取的第一步是**认证**（authentication），它是利用用户账户标识符及密码进行登录来实现的。基于用户标识符，存取控制策略便可决定一个用户被授权存取的组件。受工作职责及职位的影响，不同的用户将可能被允许操作同一数据库的不同部分。在现代数据库系统中，这主要是通过对用户数据库**存取权限**（access privilege）的授权与权限回收来实现。一个存取权限可以分配给用户账户并包含诸如创建表与视图等行为。存取权限可以被指定到一个特定的关系或视图。存储权限包括以下行为：

SELECT　　完成选择（读）操作

UPDATE　　更新记录

ALTER　　修改数据表（如增加列、删除列等）

DELETE　　删除记录

INSERT　　插入记录

这些权限的实施主要是通过维护一个由主客体构成的**授权矩阵**（authorization matrix）来完成的。该矩阵常由 DBMS 提供，并可由 DBA 进行操纵。矩阵的主体为数据库的用户，而基本的客体则可以为关系、视图或属性。矩阵中的实体显示了一个用户对于特定客体所分配到的特定权限。例如，给定一个授权矩阵如图 10-4 所示，用户 Bob 可以读取（即

完成SELECT操作）关系VENDOR，并可读取与更新关系CATEGORY；用户Alice可以读取关系CATEGORY但不能存取关系VENDOR；而用户Lee则可对关系VENDOR与CATEGORY进行各种操作。

User	Relation VENDOR	Relation CATEGORY	...
Bob	SELECT	SELECT, UPDATE	...
Alice	–	SELECT	...
Lee	ALL	ALL	...
...			

图10-4 一个授权矩阵的例子

表的创建者将被当作该表的所有者并对该表具有所有权限。数据库管理者及表的所有者均可以为其他用户使用该表进行授权或收回权限，该过程将利用DCL（数据控制语言）的SQL语句GRANT及REVOKE来实现。例如，SQL语句

```
GRANT SELECT, UPDATE ON vendor TO alice;
```

表示授予用户Alice对关系VENDOR具有SELECT及UPDATE操作权限。类似地，以下语句将收回Alice的权限。

```
REVOKE UPDATE ON vendor FROM alice;
```

该语句执行后，Alice将不能再对关系表VENDOR进行更新，但她仍然可对该表进行SELECT操作。

这种存取控制方式将为每个用户分配一个特有的权限集以实现对特定客体的存取，便于在小型的多用户环境下实施及使用。但该方式无法随用户数与客体数的增加而很好地进行扩展。在很多情况下，为一个存在大量用户的数据库维护一个关于个体用户的特权矩阵将是不可行的。在这样的情况下，我们将不再对个体用户的特权加以识别，而是使用**基于角色的访问控制**（role-based access control）系统。在该系统中，用户被指定为角色，而角色具有预先定义的权限。由DBA对角色做出定义并为角色分配权限，之后，DBA再为用户进行角色分配。SQL中的GRANT及REVOKE语句可用于实现基于角色的访问控制。例如以下SQL语句序列：

[305]

```
CREATE ROLE accountant;
GRANT SELECT ON payroll TO accountant;
GRANT accountant TO brian;
```

角色Accountant被创建，并对表PAYROLL具有SELECT操作权限。用户Brian被分配到Accountant角色，并因此对表PAYROLL具有SELECT操作权限。

在基于角色的访问控制系统里，分配给角色的权限是是固定的，而角色中的用户成员关系则是可变的。当用户在组织中的角色发生变化时，则可将其成员关系从一个角色变为另一个角色。基于用户的职责，一个用户可以被分配到多个角色。被分配多个角色的用户将拥有每个角色所具有的所有权限。

当处理的数据特别敏感时，我们可对数据进行**加密**（encryption）处理，从而为防范数据的非法存取提供多一层的保护。数据加密即使用基于**加密密钥**（encryption key）的混乱算法来实现信息转换，从而使信息变得不可读取，而只有那些拥有**解密密钥**（decryption key）及解密算法的用户，才可将信息转换至原始状态并进行读取。对于特别机密的数据可以进行数据加密处理，且系统中只有具有访问授权的用户群体才能获得解密密钥。

10.6 数据库备份与恢复

备份（backup）与**恢复**（recovery）机制用于确保数据不会丢失。在计算机发展的早期年

代，很多公司均发生过因计算机失效而使全部顾客信息丢失的事故。在当今的商业数据库系统中，备份与恢复机制可确保数据库处于一种无数据丢失的连贯状态。

数据库中的数据存储在磁盘驱动器里。当用户发起对数据库中数据的读取或更新请求时，数据必将从磁盘进入内存。任何数据更新（即插入、删除、修改）都将首先写入内存的数据副本中，再最终写入磁盘。将数据更新写入磁盘的数据文件将带来不小的额外系统开销，因此数据更新往往会分批写入磁盘。即我们并不会在每次数据更新后便将更新写入磁盘的数据文件，而是将更新首先记录在一个**恢复日志**（recovery log）里。恢复日志可确保内存中已完成的数据更新在分批写入磁盘前，即便因某种原因而造成丢失，恢复日志中仍然保存有该次更新的相关信息。恢复日志的实体中包含以下信息：那个数据值被更新，更新前与更新后的值，请求数据更新的用户。由于更新前后的值均被记录，恢复日志将允许 DBMS 重做或是撤销更新操作。当恢复日志中记录下多个更新操作以后，日志中的更新将被写入数据文件的数据中。

利用恢复日志及**检查点**（checkpoint）可实现数据库的周期性副本备份。检查点属于恢复日志的一部分。创建检查点时，恢复日志中的所有更新操作及内存中各数据更新的结果值均将写入数据库的磁盘数据文件中。检查点创建完成后，恢复日志便会清空，从而可用于 DBMS 对数据库后续更新的记录。检查点的创建符合一定规则，可在指定的时间段后进行创建或在指定数据量被写入恢复日志后进行创建。

306

当失效事件发生后，DBMS 可通过以下方式进行系统恢复：“回滚”至检查点状态，对恢复日志中记录的检查点后系统所生成的全部更新进行重做。换言之，即是将磁盘数据文件中的数据库作为起始点，并将恢复日志中记录的所有更新作用于数据库中的数据。

SQL 的 TCL 命令，如 COMMIT 和 ROLLBACK，均可用于这一过程。COMMIT 命令会将数据库的所有更新记录到磁盘上。在检查点创建期间，该命令会自动执行。同时，该命令也可由用户或 DBA 来调用执行。ROLLBACK 命令则用于回滚最近的 COMMIT 命令后的所有更新。

以上过程是典型的数据库系统日常运行中的基本工作，它们提供了对数据更新失效的恢复，更新失效可能由断电、硬件故障、事务中止、软件故障及其他类似因素所导致。

除了恢复通常情况下数据更新失效的上述机制外，现代数据库系统还提供了数据库系统彻底毁坏情况下的应对策略。例如，由于自然灾害所导致的数据库系统毁坏。为此，我们通常采用在多个物理站点上分别保留一个**完全镜像备份**（complete mirrored backup）副本的方法。这些备份副本将持续地进行更新以保持与源数据库的完全一致。当源数据库遭到毁坏时，则可用备份副本对源数据库进行重置。

10.7　数据完整性保护

数据完整性保护用于防范将导致数据库中产生无效数据、损坏数据、低质量数据的各种非授权或意外的数据插入、修改或删除操作。保护数据库的完整性可结合与数据安全和数据使用相关的多种方法和途径来实现。

在数据库中，一种危及数据完整性的方式是**非授权的恶意数据更新**（unauthorized malicious data update）。例如，一个不道德的竞争对手可能企图错误地提升数据库中由某个零售商在线使用的产品价格。本章前面所给出的访问控制方法使可用于防范此类非授权更新，并确保数据的完整性。

另一种可能威胁数据完整性的方式是**更新失效**（update failure）。例如，一个银行数据库服务器可能在以下时刻突然崩溃：一个 ATM 在从账户 A 向账户 B 的转账过程中，钱已从账户 A 拨出，但没有到达账户 B。若没有采用备份与恢复机制，则客户端用户将会在本次转账中丢掉这笔款项。而前文中基于恢复日志的备份与恢复机制可确保数据（本例中的转款）不会丢失，因为事务可回滚至账户 A 还没进行扣款的状态。

数据完整性威胁也可能由数据库的意外误操作（accidental misuse）所导致。例如，数据录入人员可能因其错误的数据输入导致产品由一个并不存在的销售商来提供。此时，数据库服务器可利用约束条件，如第 6 章讨论的参照完整性约束及用户自定义完整性约束，对此时的数据完整性做出保护。在本例中，参照完整性约束将不允许把产品分派给一个并不存在的销售商。对于其他由终端用户的误操作导致的数据完整性问题，可利用第 6 章中基于防范与校正的数据质量保证行为来进行处理。

10.8 数据库性能优化

307

数据管理工作的另一个任务是数据库的性能优化。性能优化力图最小化从数据库中检索数据的查询响应时间。与性能优化相关的数据库管理工作包括：索引（第 6 章中讨论）、逆规范化（第 4 章中讨论）、视图物化（本章前面部分讨论）以及查询优化。

查询优化（query optimization）工作包括对同一查询任务的多种执行方案的审查，以及对最快速查询方案的选取。DBMS 的**查询优化器**（query optimizer）功能可决定如何有效地执行 SQL 语句。当用户向数据库提交一个 SQL 查询时，便指定了用户希望从数据库中检索的内容，但它并没指定如何从数据库中进行检索。事实上，这正是查询优化器的作用，即鉴别执行该查询的多种可能途径（称为查询计划），并选择最好的查询计划。

在查询优化问题中，术语**查询代价**（query cost）指的是执行时间长度。查询代价将受到各种因素的影响，如操作执行的顺序、索引的使用。操作执行的顺序可能会影响到查询效率。例如，假设一个查询需要在两个表上完成 JOIN 及 SELECT 操作，则首先执行 SELECT 操作通常会更加高效。假设 JOIN 操作对应于记录了所有年龄层次的学生表，且仅需对年龄为 18 岁的学生进行操作，那么，相比于在原学生表中先执行 JOIN 再执行 18 岁学生的 SELECT 操作，首先用 SELECT 将 18 岁的学生选取出来再执行 JOIN 操作必将更加高效。

为了正确地进行决策，查询优化器中将保存各种相关的信息块，例如，一个表中的记录个数、记录尺寸、一个列中可区分值的个数、列的最大与最小值。查询优化器也将为每个索引维护相似的信息。查询创建者虽不能告知优化器如何处理查询，但可以给出所谓的**查询提示**（query hint）。查询提示可以覆盖查询优化器的默认行为。通常而言，查询优化器无需查询提示便可得到最优查询方案。而查询提示往往仅用于有经验的数据库管理者的微调过程。

10.9 数据库政策与标准的开发与实施

本章最后讨论的数据库管理任务是**数据库政策与标准**（database policy and standard）的开发与实现。一个合理的商业数据库系统对于数据库的开发、使用及管理都有特定的政策与标准。

例如，数据库系统应该有一个命名约定来命名数据库的结构，如第 2 章所述。这些约定可包含各种规则，如数据库表的每个列名必须有一个前缀，且为数据库表名的前 3 个字符。这一规则应该在数据库设计的初始阶段进行强调，并贯穿于数据库生命期内所有数据库结构

的修改之中。DBA 将依据这一规则对数据库表中每个新的列名做出检查，验证其是否使用数据表名的前 3 个字符作为列名前缀。

除了关于数据库开发的政策与标准，数据库系统也提供了关于数据库使用的政策与标准。例如，一个数据库系统可以给出以下政策：只有那些可以进入数据库的供应商才是美国协会的注册供应商。这一政策可以作为一个商业规则加以实施，它能实现供应商关系中的每个新记录与查找表（查找表中存放的是美国协会的所有注册供应商）的对应，并可防止在查找表中不存在匹配信息的任一个体进入系统。

数据库系统也可以对一些政策与标准用于引导数据库的操控与行政管理。例如，管理政策可要求数据库管理表的开发包含特定的管理任务，如访问控制、数据库性能优化、在任务中扮演特定角色的所有数据管理人员的名字等。管理政策还可以进一步要求每个任务需指定一个主要负责人，并对该任务负有最终责任。 308

数据库的政策与标准覆盖的内容从各种数据库客体的实现到基于规则的基础而对数据库中数据实施规范化的行为和过程。尽管这些政策与标准具有各自的作用范围，但它们具有一个通用的目的，即反映开支持数据库的业务流程与业务逻辑。

关键术语

access privilege（访问权限），305
accidental misuse（意外误操作），307
application development component（应用开发组件），302
authentication（认证），305
authorization matrix（授权矩阵），305
backup（备份），306
catalog（目录），303
checkpoint（检查点），306
COMMIT，307
complete mirrored backup（完全镜像备份），307
data definition component（数据定义组件），301
data dictionary（数据字典），302
data manipulation component（数据操作组件），302
database administration component（数据库管理组件），302
database policy and standard（数据库政策与标准），308
decryption key（解密密钥），306
encryption（加密），306
encryption key（加密密钥），306
GRANT，305
multiuser system（多用户系统），302
query cost（查询代价），308
query hint（查询提示），308
query optimization（查询优化），308
query optimizer（查询优化器），308
recovery（恢复），306
recovery log（恢复日志），306
REVOKE，305
role-based access control（基于角色的访问控制），306
ROLLBACK，307
single-user system（单用户系统），302
unauthorized malicious data update（非授权的恶意数据更新），307
update failure（更新失效），307
view materialization（视图物化），303

复习题

Q10.1 DBMS 软件的 4 个典型的组件是什么？
Q10.2 DBMS 数据定义组件的作用是什么？
Q10.3 DBMS 数据操纵组件的作用是什么？

Q10.4 DBMS 数据管理组件的作用是什么？

Q10.5 DBMS 应用开发组件的作用是什么？

Q10.6 数据库管理包括哪些行为？

Q10.7 请给出几个数据库系统维护行为的例子。

Q10.8 什么是物化视图？

Q10.9 什么是数据字典？

Q10.10 列举访问权限的分配行为。

Q10.11 请简要描述基于角色的访问控制系统。

Q10.12 备份与恢复机制的作用是什么？

Q10.13 给出几个有损数据库完整性的例子。

309 ~ 310

Q10.14 数据库性能优化的目标是什么？

Q10.15 给出几个数据库政策与标准的例子。

附　　录

扩展的 ER

术语扩展的 ER（EER）模型涉及扩展的 ER 标记法，用于描述标准 ER 模型以外的其他数据库模型概念。添加到 ER 模型中并起着最重要作用的 EER 是概念**超类实体**（superclass entity）与**子类实体**（subclass entity）。本附录中，将使用几个实例对添加到 ER 模型的 EER 进行说明。

A.1 超类实体与子类实体

超类实体与子类实体的概念在以下情况下使用：除了被实体的所有实例进行分享的属性外，实体实例的某些特定群组还具有附加属性，且这些属性仅应用于这些特定群组。接下来的 3 个实例将对超类与子类实体的概念做出说明。

A.1.1 EER 实例 1– 非连接子类，完全特殊化

考虑以下实例：

- 对于每一个储存，公司 X 将保持对储存编号（唯一）及邮政编码的追踪。
- 公司 X 中，有些储存是属于公司所有的。对于这些储存，公司 X 将保持对其房产价值的追踪。
- 公司 X 中，有些储存是外部租用的。对于这些储存，公司 X 将保持对其年度租赁费用的追踪。

图 A-1 展示了如何使用 EER 标记法来对这一情形进行建模。在该图中，实体 STORE 代表一个超类，而实体 COMPANYSTORE 及 RENTEDSTORE 代表 STORE 超类的子类。所有子类实例均会继承其超类的属性。本例中，所有的公司储存以及租赁储存均会有一个 StoreID 及 ZipCode 值。此外，所有的公司储存还将包含一个 RealEstateValue 值，而所有的租赁储存则将包含一个 AnnualRent 值。

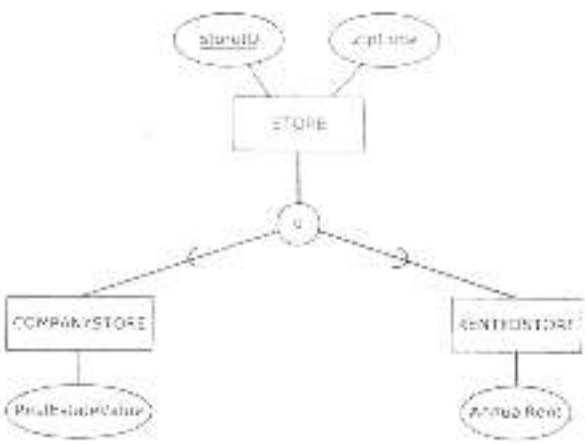


图 A-1 公司 X 中超类与子类的例子

在本标记法中[⊖]，超类与子类均使用一个圆圈进行标记。子类与超类的关

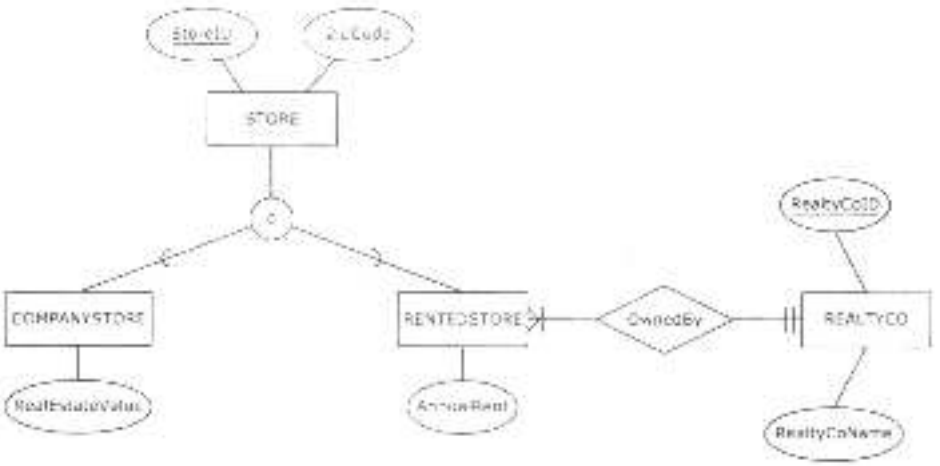
⊖ 正如我们在第 2 章所提到的，并不存在普遍采用的标准 ER 标记法。同样，也不存在通用的 EER 标记法。基于所使用的 EER 标记法，子类与超类可采用不同的方式来表示，但均具有相同的意义。

线将通过圆圈进行连接。超类与子类间的关系称为IS-A关系（在本例中对应为一个公司储存IS-A储存，以及一个租赁储存IS-A储存）。可利用加在子类连接线上的半椭圆形来识别子类。圆圈中的字母表明子类间是非连接（字母d）还是重叠（字母o）。在本例中，子类COMPANYSTORE和RENTEDSTORE是非连接子类（disjointed subclasses），这意味着子类COMPANYSTORE的实例不可能同时是子类RENTEDSTORE的实例。反之亦然。

311

用于连接超类STORE与圆圈的双线代表完全特殊化（total specialization）。在完全特殊化情形下，超类的每个实例必然也是子类的实例。本例中，完全特殊化意味着不可能存在既不是公司储存又不是租赁储存的储存。

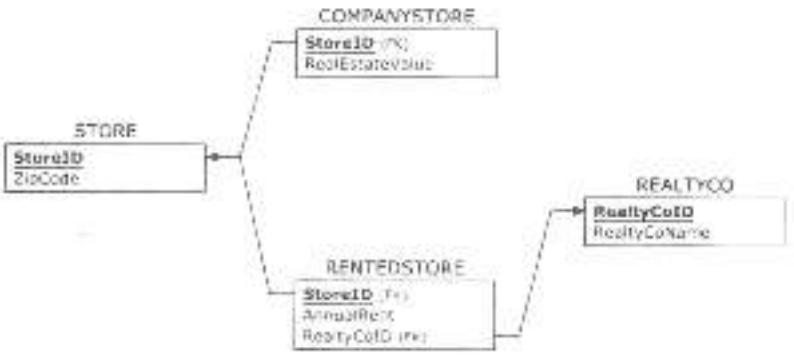
在EER图中，超类及子类都可关联在一般实体的一般关系中。这一问题在图A-2所示的扩展例子中进行了说明。该图中，子类RENTEDSTORE被关联在与一般实体REALTYCO间的一个二元的1：M关系中。



图A-2 公司X的EER图例

当EER图被映射为关系模式时，在超类与子类间的IS-A关系将被映射为超类与子类间一系列的1：1关系。图A-3显示了图A-2的EER图映射得到的关系模式。注意，在关系COMPANYSTORE及RENTEDSTORE中，列StoreID既是主码也是外码。

312



图A-3 公司X的EER图映射为关系模式

图A-4给出了公司X中由图A-2中EER图所创建的数据库的数据样本。

STORE

StoreID	ZipCode
S1	60600
S2	60605
S3	35400
S4	60611
S5	35405
S6	35405

COMPANY STORE

StoreID	RealEstateValue
S1	$9,000,000
S2	$11,000,000
S4	$8,000,000

RENTEDSTORE

StoreID	AnnualRent	RealtyCoID
S3	$450,000	R1
S5	$550,000	R2
S6	$400,000	R1

REALTYCO

RealtyCoID	RealtyCoName
R1	Brain CR
R2	Shea Prop

图 A-4 公司 X 的样本数据

注意，因为是完全特殊化，所以每个储存一定是或为公司储存，或为租赁储存。同时，由于子类是非重叠的，所以公司储存不可能同时是租赁储存，且租赁储存也不可能同时是公司储存。

A.1.2 EER 实例 2–重叠子类，完全特殊化

考虑以下实例：

- 对于每一本图书，出版商 X 保持对其图书编号（唯一）及图书书名的追踪。
- 对于出版商 X，一些图书是印刷版图书。对于这些图书，出版商 X 将保持对其印刷册数（PBookCopies）以及单本印刷价格（PBookPrice）的追踪。
- 对于出版商 X，一些图书是电子图书。对于这些图书，出版商 X 将保持对其价格（EBookPrice）的追踪。
- 每本电子图书可在多个平台上使用（每个平台具有平台编号（唯一）和平台名）。每个平台拥有大量的可用电子图书。

313 图 A-5 展示了如何使用 EER 标记法来对这一情形进行建模。

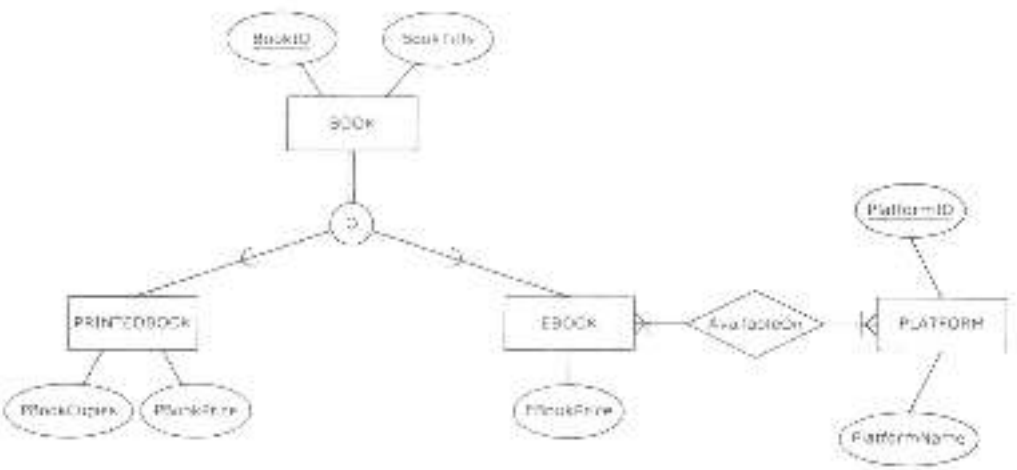


图 A-5 出版商 X 的 EER 实例

圆弧中的字母o表示子类PRINTEDBOOK与EBOOK是**重叠子类**（overlapping subclasses）。子类PRINTEDBOOK的实例可以同时是子类EBOOK的实例，反之亦然。换言之，一本图书可以既是印刷版图书又是电子图书。

由连接超类BOOK与圆圈的双线代表完全特殊化，所以不存在既不是印刷版图书也不是电子图书的图书。

图A-6 出版商X的EER图映射为关系模式

图A-6给出了图A-5的EER映射得到的关系模式。

图A-7给出了出版商X的EER图创建的数据库的数据样本。

314

BOOK

BookID	BookTitle
B1	Winter Game
B2	Solitude
B3	Code Q
B4	Pam & Sue
B5	Arrival
B6	My Mind

PRINTEDBOOK

BookID	PBookCopies	PBookPrice
B1	10,000	$90
B2	20,000	$70
B4	35,000	$65

EBOOK

BookID	EBookPrice
B2	$60
B3	$110
B4	$55
B5	$55
B6	$70

PLATFORM

PlatformID	PlatformName
P1	Bimple
P2	EyeGlad

AVAILABILITY

BookID	PlatformID
B2	P1
B3	P1
B3	P2
B4	P1
B4	P2
B5	P1
B6	P1

图A-7 出版商X的样本数据

注意，由于是完全特殊化，所以不存在既非印刷版图书也非电子图书的图书。同时，由于子类间是可以重叠的，所以一些图书可以既是印刷版图书又是电子图书。

A.1.3 EER 实例 3- 非连接子类，部分特殊化

考虑以下实例：

- 对于每一个雇员，航空公司 X 保持对其雇员编号（唯一）及雇员姓名的追踪。
- 在航空公司 X 中，一些雇员是飞行员。对于这些雇员，航空公司 X 将保持对其飞行时间（NoFHours）的追踪。
- 在航空公司 X 中，一些雇员是技术员。对于这些雇员，航空公司 X 将保持对其技术类型（MeType）的追踪。
- 在航空公司 X 中，一些雇员是空乘服务员。对于这些雇员，航空公司 X 将保持对其空乘服务水平（FALevel）的追踪。
- 还有在一些雇员，既非飞行员，也非技术员、空乘服务员。对于这些雇员，除追踪其雇员编号与雇员姓名外，航空公司 X 将不再对其他信息进行追踪。

图 A-8 展示了如何使用 EER 标记法来对这一情形进行建模。

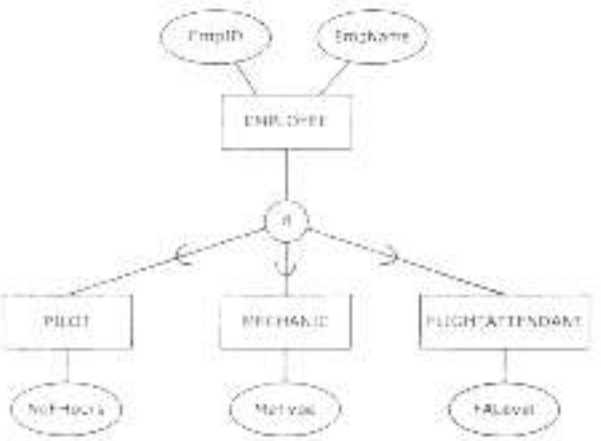


图 A-8 航空公司 X 的 EER 图例

部分特殊化（partial specialization）由连接超类 EMPLOYEE 与圆圈的单线表示。在本例中，部分特殊化表明，有些雇员既非飞行员、技术员，也非空乘服务员。

圆圈中的字母 d 代表子类 PILOT、MECHANIC、FLIGHTATTENDANT 是非连接子类。

图 A-9 给出了图 A-8 的 EER 映射得到的关系模式。

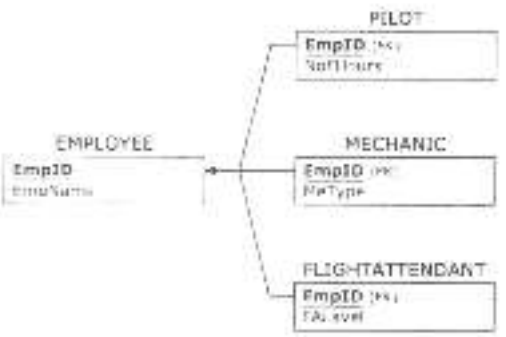


图 A-9 航空公司 X 的 EER 图映射为关系模式

图A-10给出了航空公司X的EER图对应数据库的数据样本。

EMPLOYEE

EmpID	EmpName
E1	Joe
E2	Sue
E3	Pat
E4	Lee
E5	Pam
E6	Bob
E7	Luc
E8	Stu

PILOT

EmpID	NoFHours
E1	17000
E2	20000

MECHANIC

EmpID	MeType
E3	Engine
E4	Wing

FLIGHTATTENDANT

EmpID	FALevel
E5	Level 1
E6	Level 2

图A-10　航空公司X的样本数据

注意，由于是部分特殊化，所以并不要求雇员一定是飞行员、技术员或空乘服务员之一。一些雇员（如Luc和Stu）只是一般雇员。同时，因为子类是非连接的，所以不会存在一个既是飞行员又是技术员，或既是飞行员又是空乘服务员，或既是技术员又是空乘服务员的雇员。

超类与子类可以是以下4种分类之一：

- 非连接子类，完全特殊化。
- 重叠子类，完全特殊化。
- 非连接子类，部分特殊化。
- 重叠子类，部分特殊化。

其中的前3类已经利用上述的3个实例（EER实例1、EER实例2、EER实例3）进行了说明。第4类（重叠子类、部分特殊化）将留作练习，请给出相应的实例说明。

315

316

关于规范化及更高范式

第 4 章介绍的函数依赖及规范化足以让我们很好地理解典型公司或组织机构的规范化过程，而本附录将对函数依赖以及规范化的研究做出延展。

B.1 候选码与函数依赖

除主码外，关系还可以有一个或多个额外的候选码。请看图 B-1 中给出的例子关系 CITY，它有一个主码 CityID，以及一个额外的候选码 CityName、State。

CITY

CityID	CityName	State	StatePopulation	CityPopulation
C1	Portland	ME	1,350,000	70,000
C2	Grand Rapids	MI	9,800,000	190,000
C3	Rockford	IL	12,900,000	340,000
C4	Spokane	WA	6,800,000	210,000
C5	Portland	OR	3,900,000	600,000
C6	Eugene	OR	3,900,000	360,000
C7	Grand Rapids	MN	5,400,000	11,000

图 B-1 具有一个主码和一个候选码的关系 CITY

图 B-2 中给出了关系 CITY 中的函数依赖。

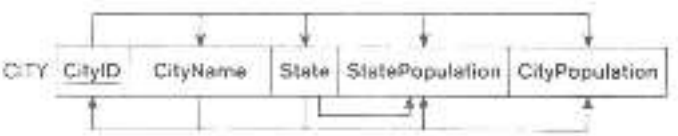


图 B-2 关系 CITY 中的函数依赖

若一个关系除主码外还有额外的候选码，以下将给出部分函数依赖与完全函数依赖的扩展定义：

部分函数依赖（Partial functional dependency）：关系中的非码属性存在对主码（或任一候选码）的某个真子集的函数依赖

完全函数依赖（Full key functional dependency）：关系中的一个属性存在对主码（或任一候选码）的函数依赖，且该属性不存在对主码（或候选码）的部分函数依赖。

注意，图 B-2 中的关系 CITY 具有一个主码 CityID，以及一个复合候选码 CityName，State。关系中的其余所有列（除 CityID 外的所有列）将完全函数依赖于主码 CityID。同时，这些列（除 CityName、State 外的所有列）也完全函数依赖于复合候选码 CityName，State。由于列 StatePopulation 函数依赖于 State，而 State 是复合候选码 CityName、State 的一个子集，因而列 StatePopulation 部分函数依赖于复合候选码 CityName、State。

回顾2NF的定义：

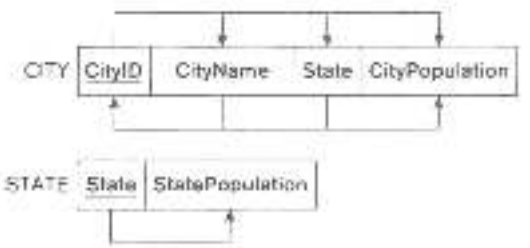
2NF：如果一个表满足1NF且不包含部分函数依赖，则这个表满足2NF。

关系CITY因包含部分函数依赖，所以不满足2NF。若将CITY规范化为2NF，则需要将CITY分解为两个关系从而消除部分函数依赖。如图B-3所示。

图B-4给出了已规范化的关系中的记录。

CITY

CityID	CityName	State	CityPopulation
C1	Portland	ME	70,000
C2	Grand Rapids	MI	190,000
C3	Rockford	IL	340,000
C4	Spokane	WA	210,000
C5	Portland	OR	600,000
C6	Eugene	OR	360,000
C7	Grand Rapids	MN	11,000



图B-3 对关系CITY的规范化

STATE

State	StatePopulation
ME	1,350,000
MI	9,900,000
IL	12,900,000
WA	6,800,000
OR	3,900,000
MN	5,400,000

图B-4 已规范化的关系CITY中的数据记录

回顾传递函数依赖的定义：

传递函数依赖：是指非码列函数确定关系中的其他非码列。

回顾3NF的定义。

3NF：如果一个表满足2NF且不包含传递函数依赖，则这个表满足3NF。 |318

候选码的存在将不会扩充传递函数依赖的定义，因为传递函数依赖是定义在非码属性间的函数依赖。

图B-3中的关系已满足3NF。

若我们并不知道CityName、State是候选码，则图B-2中的关系CITY将如何规范化？此时，函数依赖

State → StatePopulation

将被认为是一个传递函数依赖，因此，它会在规范化为3NF时进行消除，而不是在规范化为2NF时消除。但两个规范化过程的最终结果是相同的，同为图B-3所示。总之，无论CityName、State是否被当做候选码，关系CITY将得到同样的规范化结果。

B.2 BOYCE-CODD范式（BCNF）

BOYCE-CODD范式（Boyce-Codd normal form，BCNF）是对3NF的扩展。BCNF的定义如下。

BCNF：一个满足BCNF规范的关系，除具有完全主码函数依赖外，不再包含其他函数

依赖（即该主码或是候选码能够完全确定其他列）。

在很多情况下，满足3NF的关系同时也会满足BCNF。在满足3NF的关系中，那些仅有主码而没有候选码的关系将可对满足BCNF。但也存在以下情况：一些既有主码又有候选码的关系满足3NF但不满足BCNF。例如，考虑图B-5中的关系TEAMPLAYEROFTHEGAME，该关系中的数据记录了球队参加每场比赛的队员。在本例中，表中给出的是单个赛季的数据，不存在球队间的队员交换（在单个完整的赛季中，队员只能属于某一支球队），且不存在任何两个队员同名的情况（队员的姓名都是唯一的）。

TEAMPLAYEROFTHEGAME

GameOfSeason	Team	TeamPlayerOfTheGame
1st	Tigers	Joe Jones
2nd	Tigers	Tim Smith
3rd	Tigers	Joe Jones
1st	Sharks	Scott McHill
2nd	Sharks	Scott McHill
3rd	Sharks	Lee Hicks

图B-5 关系TEAMPLAYEROFTHEGAME（满足3NF但不满足BCNF）

图B-5中的关系TEAMPLAYEROFTHEGAME对应的函数依赖如图B-6所示。

图B-6 关系TEAMPLAYEROFTHEGAME（满足3NF但不满足BCNF）中的函数依赖

关系TEAMPLAYEROFTHEGAME满足3NF，因为它不包含部分或传递函数依赖。但它不满足BCNF，因为非码属性TeamPlayerOfTheGame可确定主码列Team的值。

为了规范化为BCNF，关系TEAMPLAYEROFTHEGAME将转变为两个关系，如图B-7所示。

图B-7 关系TEAMPLAYEROFTHEGAME规范化为BCNF

图B-8给出了已规范化的关系中的记录。

TEAMPLAYEROFTHEGAME

GameOfSeason	TeamPlayerOfTheGame
1st	Joe Jones
2nd	Tim Smith
3rd	Joe Jones
1st	Scott McHill
2nd	Scott McHill
3rd	Lee Hicks

PLAYERTEAM

Player	Team
Joe Jones	Tigers
Tim Smith	Tigers
Scott McHill	Sharks
Lee Hicks	Sharks

图B-8 已规范化的关系TEAMPLAYEROFTHEGAME中的数据

我们可通过选择不同的主码来避免关系TEAMPLAYEROFTHEGAME的BCNF规范化

过程，注意，如图 B-5 所示，关系 TEAMPLAYEROFTHEGAME 具有两个候选码：

- GameOfSeason, Team
- GameOfSeason, TeamPlayerOfTheGame

在图 B-5 中，GameOfSeason、Team 被选为了关系 TEAMPLAYEROFTHEGAME 的主码，而图 B-9 中则选择了另一候选码（GameOfSeason, TeamPlayerOfTheGame）作为关系 TEAMPLAYEROFTHEGAME 的主码。

TEAMPLAYEROFTHEGAME

GameOfSeason	TeamPlayerOfTheGame	Team
1st	Joe Jones	Tigers
2nd	Tim Smith	Tigers
3rd	Joe Jones	Tigers
1st	Scott McHill	Sharks
2nd	Scott McHill	Sharks
3rd	Lee Hicks	Sharks

图 B-9　主码替换后的关系 TEAMPLAYEROFTHEGAME

图 B-9 中的关系 TEAMPLAYEROFTHEGAME 无损的函数依赖如图 B-10 中所示。

图 B-10　主码替换后的关系 TEAMPLAYEROFTHEGAME 中的函数依赖

主码替换后的关系 TEAMPLAYEROFTHEGAME 将不再满足 2NF，因为它含有一个部分函数依赖。可通过为部分函数依赖新增一个关系（如图 B-11 所示），利用（第 4 章中的）标准方法来实现其 2NF 规范化过程。

图 B-11　主码替换后的关系 TEAMPLAYEROFTHEGAME 规范化至 2NF（及后续 3NF、BCNF）

注意，图 B-11 中的关系虽已规范化至 3NF 及 BCNF，事实上，图 B-11 中的关系与图 B-7 中的关系完全相同。此外，图 B-11 中关系的数据记录也与图 B-8 中的数据记录完全相同。

上述过程通过选取不同的主码，使规范化过程不再是处理主码列对非码属性的函数依赖，而是处理部分函数依赖关系。换言之，通过选取不同的主码，关系的规范化过程从规范化至 BCNF 转变为了规范化至 2NF。

B.3　第四范式（4NF）

考虑图 B-12 中的关系 ORGANIZATION_STUDENT_CHARITY。

在本例中，组织机构拥有学生成员，并支持各种慈善事业。关系 ORGANIZATION_STUDENT_CHARITY 中给出了每个组织机构的成员以及它所支持的慈善事业。列 StudentID 及 Charity 均将与列 OrgID 发生关联，但 StudentID 及 Charity 之间并不彼此关联。一个 OrgID 可以关联多个 StudentID 值，以及多个 Charity 值。形式上可描述为：

321

OrgID →→ StudentID 和 OrgID →→ Charity

ORGANIZATION_STUDENT_CHARITY

OrgID	StudentID	Charity
011	1111	Food Pantry
011	1111	Stop Diabetes
011	1111	River Care
022	1111	River Care
022	2222	River Care

图 B-12 关系 ORGANIZATION_STUDENT_CHARITY（不满足第四范式）

其中，符号 →→（双箭头）代表**多值依赖**（multivalued dependency）。多值依赖也可称之为"事实（行）依赖"。例如，因 OrgID →→ Charity，则每当有一个新学生加入某个组织机构时，则会产生多个新行（每行均与该组织机构中的一个慈善事业相关联）。同样，因 OrgID →→ StudentID，每当有一个新慈善事业分派给某个组织机构时，也会产生多个新行（每行均与该组织机构中的一个学生成员相关联）。

如前所述，多值依赖将产生于同一关系的独立列间包含无关值的情况下（即当同一关系代表了基数大于 1 的多个单独的关系时）。以下是 4NF 的定义。

4NF：如果一个关系满足 BCNF 且不包含多值依赖，则这个关系满足 4NF。

因此，一个关系中若包含多值依赖，则该关系将不满足 4NF。

因为多值依赖的存在，关系 ORGANIZATION_STUDENT_CHARITY 将不满足 4NF。要将其规范化为 4NF，只需为每个多值依赖创建一个单独的关系，如图 B-13 所示。

ORGANIZATION_STUDENT

OrgID	StudentID
011	1111
022	1111
022	2222

ORGANIZATION_CHARITY

OrgID	Charity
011	Food Pantry
011	Stop Diabetes
011	River Care
022	River Care

图 B-13 关系 ORGANIZATION_STUDENT_CHARITY 规范化为 4NF

B.4 其他范式

除了 4NF，还存在更高层的范式，如第五范式（5NF）、域键范式（DKNF）。这些范式主要用于一些实践上很少涉及的理论性概念之中。因而，这些概念超出了本书的研究范畴。

322

企业资源计划

"综合信息系统"是这样一个系统：系统内的数据可以由使用该系统的组织机构中的多个功能区进行共享。在公司这一组织机构内，其功能区包括市场、销售、人力资源等。以非冗余方式实现数据存储的任一数据库均可由构成综合信息系统的不同功能区来进行使用。

企业资源计划（ERP）系统（Enterprise resources planning system）是工业中用于描述预置的、综合性的（多功能的）公司信息系统的术语。ERP 系统由 ERP 销售商（如 SAP、Oracle、Lawson）设计、创建并销售。一个购买并安装了 ERP 系统的公司则可以使用自己的数据进行系统移植。

从数据库的角度看，ERP 代表一个预置的数据库，该数据库为各种用户群体预置了多个前端接口。ERP 概念之中隐含的思想是：创建一个空的数据库，其中的数据表以及列则用于获取公司中各种群体或员用户将使用的数据。ERP 数据库中带有大量的不同组件，容纳了为不同的用户群经所创建的用于数据库访问的前端应用。图 C-1 给出了一个 ERP 系统架构的高层视图。

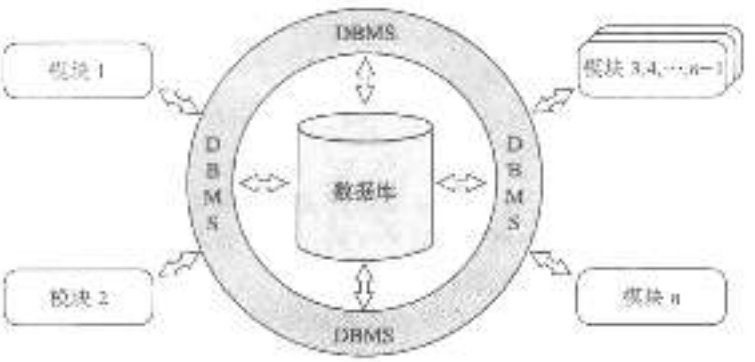


图 C-1　ERP 系统

用于典型 ERP 系统的模块反映了公司中的部门或组织结构。例如，一个 ERP 系统可以包含如下模块：

- 人力资源（HR）模块。
- 财务账目模块。
- 销售模块。
- 市场模块。
- 制造模块。
- 其他模块。

图 C-2 给出了一个带有大量可用模块的 ERP 系统。

当公司购买并安装了一个如图 C-2 所示的 ERP 系统时，便会将先前的数据表移植到 ERP 数据库中，这些数据表是该公司中有关人力资源、财务账目、销售、市场、制造等方面的数据。假如有的数据关系到不止一个部门，这些数据将仅需存储一次，并可由多个模块进行访问。例如，顾客数据被一次性储存在中心数据库中，并可由销售模块、市场模块或财务账目等多模块访问。

公司在购买ERP系统时，不必购买和安装所有的可用模块，而是可以使用一个或几个模块进行初始化配置，并在后期需要时再进行其他模块的加载。

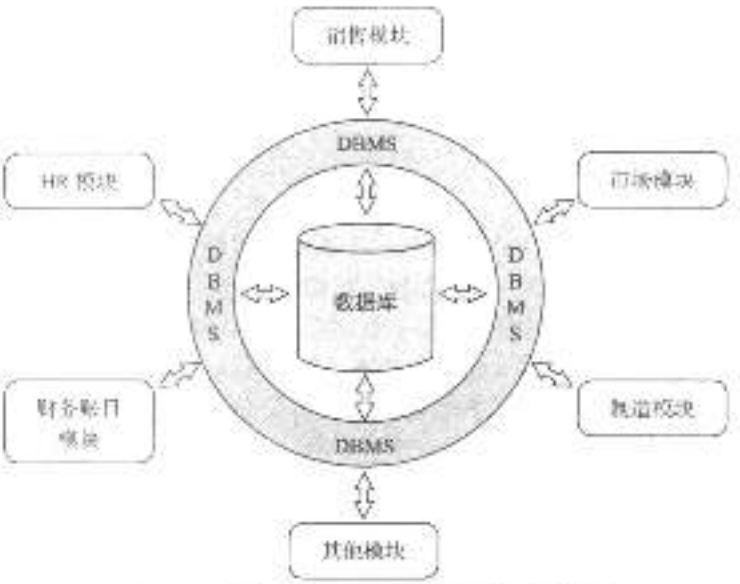


图 C-2 一个典型ERP系统中的各种可用模块

ERP系统可帮助公司统一整个业务流程。例如，在一个ERP系统中，HR模块包含了用于访问ERP数据库表及列的前端应用，ERP数据库中容纳了用于标准HR过程及步骤的相关数据，如招聘、雇用、工资单等。在HR模块及ERP数据库HR部分的设计阶段，ERP销售商运用HR的专家知识与经验，以专业的方式解释并帮助实施所有典型的HR功能。ERP系统中的其他模块将实施其他标准化过程，并以同样的方式进行创建。

设计ERP系统是为了反映一个特定商业过程的最佳工业实践方案。若公司拥有一个ERP系统，但其商业过程却与ERP系统执行方式有所不同时，公司可选择修正其操作过程以匹配ERP系统中提供的最佳实践方案，或对ERP系统做出可允许的调整以匹配公司的实践过程，或是将这些方法结合使用。

在公司中，并非所有商业过程都是标准化的商业过程。在运作成功的公司里，很多商业过程都是原创性的，尤其是那些用于创造或增加竞争优势的过程。这些过程的具体细节常常是需要严守机密信息，所以ERP销售商根本无法获取。支持非标准化过程的数据表与模块在ERP系统中是不可用的。为了处理这些过程，公司不得不开发自己的数据库与前端应用。因此，公司不能仅依赖于ERP系统去满足其信息系统的所有需要。

ERP的数据库与模块以特定的方式实现了每个过程的标准化。特定的方式可能并不意味着公司想要为某个过程而存储和使用数据。一些公司使用ERP系统简化部分或全部的标准化过程，而另一些公司则根本不使用ERP系统。

ERP系统的购买与实施似乎是一个简单而直接的任务，但是，ERP系统的部署时间可能长达几个月（甚至几年），且不少ERP项目竟以失败告终。导致ERP系统实施失败的一个主要原因是，客户缺乏对ERP系统前期细节的检查，并缺乏对ERP系统与公司商业过程之间的比较，这将可能导致ERP系统性能与公司需求和实践之间的不良匹配。

324

值得注意的是，ERP系统将迫使一个组织去适应该ERP，而非相反。那些需求性的调整并不总是顺利的（或算它们代表着对现有实践过程的一种提升），并可能导致冲突与阻碍。

325

数据管理与主数据管理

D.1 数据管理

数据管理（data governance）是一个广义的术语，用于组织机构中的权限定义，以正式规定组织中的数据与元数据在何时、以何种方式、由谁来进行创建、储存、更新（插入、修改、删除）及存档。

在很多组织机构中，尽管数据是最重要的资产之一（如果不是最重要的，也应是最重要的之一），但它往往并不具备与其他资产同等重要的地位，比如不动产、资金或是汽车，这些物件均是传统意义上的重要资源。不过，越来越多的公司已经逐渐意识到数据的重要意义，并开始实施数据管理工作。

数据管理的目标是建立与实施关于数据处理的正确规则与策略。这一做法使得组织机构在形式上可以规范化数据的管理，类似于规范化其他重要资产的管理，如金融资产。例如，很多公司都有严格的规定，确定谁有权使用公司的资金、为何使用、在哪些场合使用，由谁授权使用等。为了处理公司的数据，数据管理过程也制定了类似的规定。

关于数据管理的一个实例是第 10 章所述的数据库政策与标准的开发与实施，另一个实例则是第 6 章所述的通过制定政策对数据质量行为进行预防与校正。很多情况下，数据管理权将会关系到广泛的法律与组织中的行为管理规定。例如，美国卫生保健机构的数据管理实践活动必须与健康保险流通与责任法案（HIPAA）相一致，该法案由美国联邦政府执行。该法案的条款规定对患者信息的使用与披露必须具有严格的隐私性。在美国卫生保健机构中，这些法规必须在数据库系统中实施，并将患者信息组织成其整个管理框架中的一个部分。

在很多组织中，**数据专员**（data steward）角色将完成数据管理相关的任务。数据专员负责数据库中数据的正确使用，典型的管理活动包括：

- 为确保适当的数据录入、数据更新及数据使用，创建并实施相关的政策与标准。
- 创建并实施数据质量监控。
- 创建、实施并维护商业元数据（例如商业规则、表与列的描述等）。
- 其他可确保数据管理政策承诺的行为。

除了数据专员角色，很多组织中也设有数据管理者角色。**数据管理者**（data custodian）负责数据管理及使用中的技术问题，比如数据的保护、传输与存储。很多组织中，数据专员、数据管理者及 DBA 角色间具有明显的重叠。对于头衔与相关责任的分配，在不同的组织中差异很大。

D.2 主数据管理

主数据管理（master data management，MDM）是最常见的组织数据管理权之一。组织中的主数据包含主要数据的质量认证版本，这些数据将为组织机构的信息系统提供一个共同的参

326 要点。一旦主数据配置到位，则组织内的操作型信息系统将确保自己的数据与主数据相匹配。

例如，ZAGI零售公司需要维护一个数据表MASTERPRODUCT，该表中包含了由ZAGI零售公司销售的所有产品信息。ZAGI公司里使用产品信息的所有信息系统必须检查并确保其产品信息与数据表MASTERPRODUCT中的产品信息相一致。

主数据管理包括对主数据质量表的采集行为的创建与维护，并确保主数据的使用已嵌入到组织的操作型信息系统中。确保主数据的最高质量是一项基本要求。主数据必须展现高质量数据的所有特性（准确性、唯一性、完整性、一致性、及时性、统一性），正如第6章中所描述的。

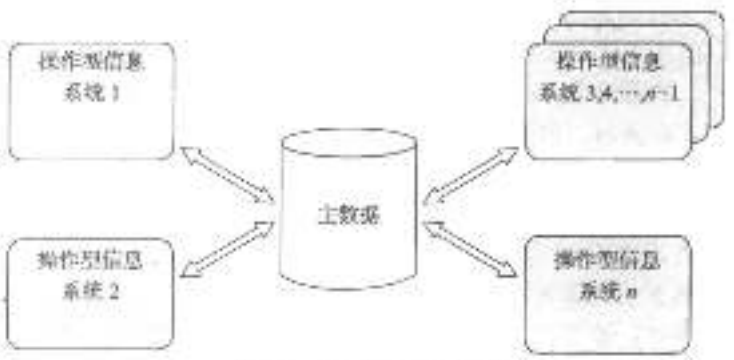
在操作型信息系统里，并非所有数据都是主数据。但是，对应于主数据的数据必须与主数据保持一致。考虑以下例子，在ZAGI零售公司存储部工作的某个职员记录了以下信息：在公司的销售管理信息系统中，某个顾客在上午8:00购买了某产品3个。产品数量（3）及购物时间（上午8:00）与主数据无关。但假若存在主数据表MASTERPRODUCT、MASTERCUSTOMER及MASTERSTORE，则销售信息系统中关于产品、顾客及存储的数据都需要进行验证，并与这些主数据表保持一致。

正确的主数据可以确保组织中所使用的关键信息具有一致性。操作型系统中的主数据概念类似于第8章所讨论的数据仓库的"一致维度"概念，只是一致维度所提供的一套质量参考数据用于一致性分析，而主数据所提供的一套质量参考数据则用于一致性操作。

对主数据的管理存在着多种体系，3种主要方法是：

- 中心法。
- 注册登记法。
- 混合法。

在中心法下，操作型系统将对主数据采用单一中心副本策略。非主数据依然可由系统来收集与维护，而主数据的所有实例却可以利用中心主数据副本进行恢复与更新。中心法如
327 图D-1所示。



图D-1 中心化的主数据

与中心法相反的一个方法，则是将所有的主数据存放在各个单独的操作信息系统中。这些分离的主数据将通过一个主数据中心注册表来进行连接。**主数据注册表**（master data registry）仅包含一个关键字列表，用于连接及融合位于操作型系统中的实际的主数据。主数据注册表允许单个信息系统从其他信息系统中访问主数据，用以匹配与支撑本系统中的主数据。主数据注册表如图D-2所示。

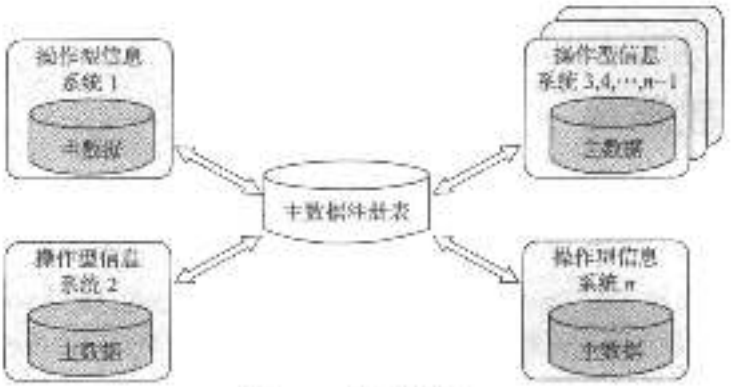


图 D-2 主数据注册表

在混合法中，主数据的存储结合了前述两个方法。存在一个实际的主数据中心副本，但也允许单个的操作型系统持有自己的主数据副本。主数据中心副本与其他的主数据副本将通过主数据中心注册表进行连接与整合。混合法如图 D-3 所示。

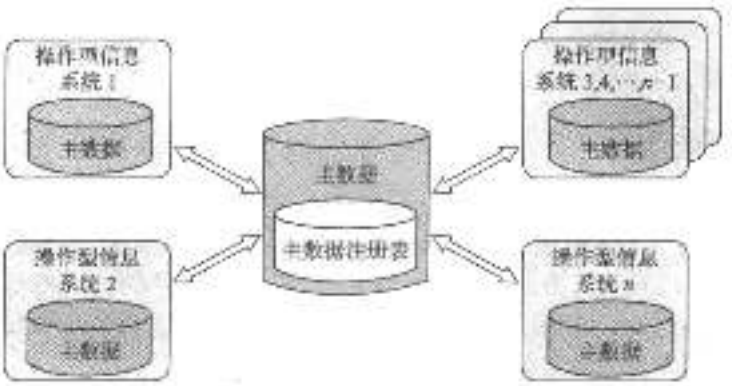


图 D-3 混合 MDM 方法

在混合法的部分实施过程中，主数据仅在中心副本中进行更新（插入、删除、修改），且所有更新将会传播到操作型系统的副本之中。而其他实施过程则允许主数据在单个的操作型系统中进行更新，这些数据改变将被传递到中心副本中，再进一步传播到其他操作型系统中。 328

此外，在部分实施过程中，整个主数据都将呈现在中心副本与所有的单个操作型系统中，而其他实施过程中则仅有部分主数据被呈现在中心副本与单个操作型系统中。 329

面向对象数据库

面向对象数据库（object-oriented database，OODB）系统也称作对象数据库系统（ODBS），是基于所谓的面向对象概念的数据库系统。本附录中将给出这些概念的简要概述，并说明它们将如何用于 OODB。面向对象的概念是在面向对象程序设计语言中首先引入的。一个面向对象的数据库管理系统（OODBMS）包含了数据库系统的特征，以及源自面向对象程序设计语言的面向对象特征。

OODBMS 是为了克服现有数据库（如关系数据库）的局限性而提出的，用于设计与执行复杂的数据库应用。复杂的数据库应用包括多媒体数据库、地理信息系统（GIS），用于计算机辅助设计（CAD）与计算机辅助制造（CAM）的数据库。这些数据库需要存储并操作各种对象，如图像、地图及视频等，它们都不是关系模式中所使用的典型数据类型，不符合关系表中的行、列概念，并需要有比典型的关系操作更多的操作。

E.1 面向对象概念

在 OODB 中，**对象**（object）对应于现实世界中的客体，这一概念与 ER 模型中的实体具有同样的意义。除了**对象属性**（object attribute）（等价于实体属性），对象还具有**对象操作**（object operation）（也称为“方法”，本附录的后续部分将进行说明），以及系统所生成的**对象标识**（object identifier，OID）。在 OODB 中，分享同种结构与操作的对象将形成一个**类**（class），特定类的对象称为该类的**实例**（class instance）。

OID 不同于主码，因为 OID 值由 OODB 系统自动分配且不可改变。然而，在关系数据库中，主码是基于值的（即在关系中，可使用其他列值的录入方式来录入主码值），且主码值是可以改变的。在整个面向对象的数据库中，OID 值是唯一的。而在关系数据库中，存在于两个独立关系中的两个主码值则可以是相同的。考虑图 E-1 中的例子。

图 E-1 的上半部分给出了 EMPLOYEE 及 CUSTOMER 两个关系表，其中 EMPLOYEE 具有主码列 EID 及列 EName，CUSTOMER 具有主码列 CID 及列 CName。该图的下半部分给出了 EMPLOYEE 及 CUSTOMER 两个对象的类，分别含有属性值 EName 及 CName，且每个对象有各自的 OID。在关系表中，给定的主码值是可以改变的，即便主码值变了，码值所在的行依然代表着先前的实例（例如，一个改变了 EID 值的行依然代表着同一个雇员）。而在 OODB 中，每个对象有只仅有一个 OID 值，其 OID 值不能被其他对象重用，即便是其

主码（关系型数据库）

EMPLOYEE

EID	EName
100	Anne
101	Bob
102	Cliff

CUSTOMER

CID	CName
100	Adam
101	Betty
102	Cindy

OID（面向对象数据库）

EMPLOYEE

OID	EName
100	Anne
101	Bob
102	Cliff

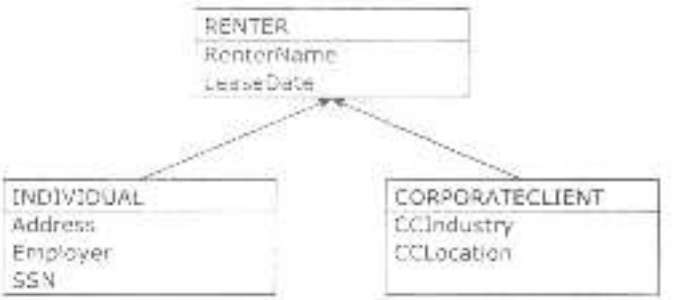
CUSTOMER

OID	CName
201	Adam
202	Betty
203	Cindy

图 E-1 主码与 OID 的对比

他类型的对象也不能出现重用。这也是图E-1下半部分的所有OID值均唯一的原因。注意，OID并非是对象的属性值，OID的唯一作用是将对象进行区分。它并未携带其他更多信息，且不会显示给数据库用户，而对象的属性值是可显示的。

如前所述，分享同样结构与操作的对象将形成一个类。例如，数据库HAFH（在第2、3、5章中曾使用）可扩展包含出租商的信息，且出租商可由个体出租商或企业出租商构成。这样，数据库中用于代表公寓出租商的数据对象可划分为INDIVIDUAL及CORPORATECLIENT两个类。在OODB中，类可以组织成**类层次/特化层次**（class hierarchy / specialization hierarchy）结构。图E-2中给出了由类RENTER、INDIVIDUAL及CORPORATECLIENT所形成的类层次结构。一个特化层次结构被描述为一个"IS-A"关系。图E-2中展现了公寓的一个CORPORATECLIENT *IS-A* RENTER关系与一个INDIVIDUAL *IS-A* RENTER关系。



图E-2 类的层次结构示例

在图E-2中，CORPORATECLIENT及INDIVIDUAL都是类RENTER的**子类**（subclass），而类RENTER则是类CORPORATECLIENT及INDIVIDUAL的**超类**（superclass）。在类层次结构中，每个类均有自己的属性。新类将从现有类中进行创建（即子类从超类中创建），且超类的属性将被子类继承。例如，在图E-2中，类RENTER的属性为RenterName及LeaseDate；而类INDIVIDUAL具有属性Address、Employer、SSN，以及从类RENTER中继承而来的属性RenterName、LeaseDate；类CORPORATECLIENT则具有属性CCIndustry、CCLocation，以及从类RENTER中继承而来的属性RenterName、LeaseDate。

除了传统的内置数据类型，如变长字符类型、日期类型等，OODB允许创建与使用用户自定义的数据类型。**用户自定义类型**（user-defined type，UDT）具有以下格式：[1]

```
CREATE TYPE type_name AS (component,...)
```

331

我们将利用图E-2中的类RENTER、CORPORATECLIENT及INDIVIDUAL对UDT概念加以说明。例如，renter_type将包含rentername及leasedate两个组件，并分别对应于数据类型varchar及date；individual_type将包含三个组件，其中address及employer对应于数据类型varchar，而ssn则对应于数据类型char；corporateclient_type将包含ccindustry及cclocation两个组件，其类型均为varchar。

关键字UNDER用于表明所定义的类型是一个子类，且将会继承对应超类的属性与方法。换言之，关键字UNDER表示被定义的类型在类的层次结构中位于超类的"下方"。

[1] 本附录的例子中采用了DDL与DML，这与关系-对象数据库（将在本附录的结尾处进行描述）中使用的UDT相一致。UDT提供了对现有SQL的扩展。

```
CREATE TYPE renter_type AS
  (rentername   VARCHAR (25),
  leasedate     DATE);

CREATE TYPE individual_type UNDER  renter_type AS
  (address      VARCHAR(25),
  employer      VARCHAR(20),
  ssn           CHAR (9));

CREATE TYPE corporateclient_type UNDER  renter_type AS
  (ccindustry   VARCHAR(25),
  cclocation    VARCHAR(25));
```

individual_type 是 renter_type 的一个子类，因而将继承 renter_type 的所有属性。这意味着 individual_type 将包含以下属性：rentername、leasedate、address、employer 及 ssn。类似地，corporateclient_type 也是 renter_type 的一个子类并继承了其所有属性，将包含以下属性：rentername、leasedate、ccindustry、cclocation。

新创建的类型可用于定义其他新类型。例如，我们可以创建一个 name_type，并包含三个组件：firstname、minitial 及 lastname。

```
CREATE TYPE name_type AS
  (firstname   VARCHAR(10),
  minitial     CHAR (1),
  lastname     VARCHAR (15));
```

然后我们可以为类 MANAGER 创建一个新类型，名为 mgr_type，并包含属性 mname，该属性的数据类型为 name_type。

```
CREATE TYPE mgr_type AS
  (managerid   CHAR (6),
  mname        name_type,
  msalary      NUMERIC (9,2));
```

则类型 mgr_type 所生成的对象将具有以下属性：managerid、firstname、minitial、lastname 及 msalary。

332

嵌套对象（nested object）即出现在另一对象中的对象。我们考虑与 HAFH 数据库相似的一个例子：一幢大楼由某个特定的管理员进行管理。我们为大楼对象创建一个新的类型 bldg_type_nest，其中包含了类型为 mgr_type 的嵌套对象 bmanager。

```
CREATE TYPE bldg_type_nest AS
  (buildingid  CHAR (3),
  bnoofflours  INT,
  bmanager     mgr_type);
```

类型 bldg_type_nest 的对象将包含属性：buildingid、bnoofflours、managerid、mname、msalary，这些属性均与大楼管理相关。另一个嵌套对象的例子是 mgr_type 类型中的 mname。

当一个对象被另一对象所参照时，便生成了**参照对象**（reference object）。为了表示对象的参照类型，我们采用关键字 REF 来指明对象的参照类型。参照类型将作为对象的 OID 来执行。将数据插入数据库时，其参照必须指定到一个特定的管理员。我们再次考虑大楼对象，这次不再定义大楼的管理员为一个嵌套类型，而是定义为一个参照类型。为此，我们创建新类型 bldg_type_ref，包含类型为 mgr_type 的参照对象 bmanager。

```
CREATE TYPE bldg_type_ref AS
  (buildingid  CHAR (3),
  bnoofflours  INT,
  bmanager     REF mgr_type)
```

参照对象的执行不同于嵌套对象。嵌套对象中含有该对象的确切值，这意味着

managerid，mname 及 msalary 的值将被复制到对象 building 的 bmanager 属性之中。而参照对象则不同，此时，只有 OID 会复制到对象 building 的 bmanager 属性之中。

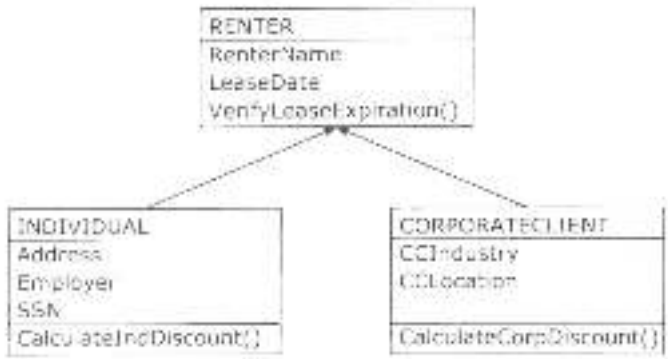


图 E-3 类的层次结构示例，用类方法进行扩展

除了自己特有的属性值外，类层次结构中的每个类均有自己的函数实现方法，这些方法将作用于该类所生成的对象，如图 E-3 所示。

在进行类型定义时，除了定义属性外，方法也需进行定义。例如图 E-3 的示例中，在创建 renter type 时，将定义函数 VerifyLeaseExpiration() 在以下情况返回 TRUE：从 LeaseDate 至今，时间尚未超过 1 年；而在以下情况返回 FALSE：从 LeaseDate 至今，时间已经达到或超过 1 年。在创建 individual_type 时，将定义函数 CalculateIndDiscount()，为特定的雇员返回其商品折扣值。类似地，在创建 corporateclient_type 时，也将对函数 CalculateCorpDiscount() 做出定义。在类层次结构中，超类的属性与方法均会由子类继承。例如，在图 E-3 中，类 RENTER 的属性是 RenterName、LeaseDate，方法是 VerifyLeaseExpiration()，则子类 INDIVIDUAL 的属性为 Address、Employer、SSN、RenterName 及 LeaseDate，方法则为 VerifyLeaseExpiration() 及 CalculateIndDiscount()；子类 CORPORATECLIENT 的属性为 CCIndustry、CCLocation、RenterName 及 LeaseDate，方法则为 VerifyLeaseExpiration() 及 CalculateCorpDiscount()。 333

E.2 面向对象查询

在一个类中，当对象类型已经定义后，以下表格创建语句可用于为类型 mgr_type 与 bldg_type 创建对象。

```
CREATE TABLE manager AS mgr_type;
CREATE TABLE building AS bldg_type nest;
```

一旦相应的对象（表）被创建，且使用其他命令向其中插入了数据实例，则对象便可以进行数据的更新与恢复操作。该语言[3]中涵盖了对嵌套对象及参照对象进行查询与更新的功能。

我们将使用以下例子来进行说明，该例子中考虑了管理员以及大楼对象，类似于 HAFH 数据库。

Query X 的文本描述：查询每栋大楼的 ID 以及大楼管理员的姓名。

使用上述已经创建好的类型与表，该查询可表示如下。

Query X（用于 OODB）：

```
SELECT      buildingid, bmanager.firstname,
            bmanager.minitial, bmanager.lastname
FROM        building;
```

注：对象查询语言（OQL）属于对象数据管理组（ODMG）所设计的一种类 SQL 查询语言，该管理组致力于为 ODBS 推出一种标准化查询语言。OQL 是复杂的，还没有一个商用数据库能够完全地执行 OQL 语言，但一个类 SQL 查询语言在 OODB 或在 ORDB 中则是使用可用的。

在 OODB 的 Query X 中，"." 用于限制嵌套对象中的属性名，以使其含有嵌套对象名。例如，嵌套对象 bmanager 中的属性 firstname 被指定为 bmanager.firstname。OODB 的 Query X 可称为"路径查询"，因为它使用 "." 来穿越嵌套对象。注意，查询中要求 building 表在 FROM 子句中单独指定，而 manager 表则不必指定。

我们将对 OODB 中的 Query X 与关系数据库中同样的查询进行比较。假设在关系数据库中，其 manager 表与 building 表定义如下：

```
CREATE TABLE manager
        (managerid        CHAR (4),
        mfirstname        VARCHAR(10),
        minitial          CHAR (1),
        mlastname         VARCHAR (10))
        msalary           NUMERIC (9,2)
        PRIMARY KEY       (managerid));

CREATE TABLE building
        (buildingid       CHAR (3),
        bnoofflooro       INT,
        bmanagerid        CHAR(4)
        PRIMARY KEY       (buildingid),
        FOREIGN KEY       (bmanagerid) REFERENCES
                                       manager(managerid) );
```

使用于前面例中的两个表，关系数据库中的 Query X 定义如下：

Query X（用于关系数据库）：

```
SELECT    buildingid, mfirstname, minitial, mlastname
FROM      manager, building
WHERE     managerid = bmanagerid;
```

关系数据库中的 Query X 需要在 MANAGER 与 BUILDING 两个表之间进行连接操作，该操作利用了 FROM 子句中罗列出的两个表，并通过 WHERE 子句中给出的连接条件 managerid = bmanagerid 来指定。而路径查询——已在基于 OODB 的 Query X 中给出——可消除连接操作的需要，因而，路径查询过程将比基于连接的查询过程速度更快。

在 OODB 中，连接操作的需要并不是在各种情况下均能完全消除。基于 OODB 的设计及相应的查询过程，可能依然需要连接操作。

E.3 对象－关系数据库

对象－关系数据库（object-relational database，ORDB）是一种关系数据库，该数据库同时具有一些面向对象的特性。ORDB 管理系统被认为是关系数据库系统的扩展。ORDB 主要包含一些使用最广泛的面向对象特征，如 OID、继承、任意数据类型、嵌套对象、参照对象及方法等。典型的 ORDB 允许用户创建对象类型以及对象表。ORDBMS 被成功地用于管理媒体对象及复杂数据的应用，如地理空间及金融时间序列数据系统等。ORDB 的一个缺陷是，在特定情形下可能会降低性能，这是因为面向对象特征是创建在现存关系数据库管理系统的顶层，这就使得性能优化比较困难。当用户已经熟悉了关系数据库，且又需要使用更多面向对象的特征时，ORDB 将为其提供方便。很多市面上出售的关系数据库都具有对象－关系的特征。同时，市面上也存在面向对象的数据库，但它们并没有获得与对象－关系数据库同样广泛的使用。

分布式数据库、并行数据库与云计算

分布式数据库系统（distributed database system，DDBS）是分布于不同计算机上的数据库系统。在 DDBS 系统中，允许进行数据分享，并提供了数据库的本地控制与自主权。分布式数据库反映了自然分布的组织结构。作为分布式数据库的一个例子，我们可考虑由一个总部与多个分部所形成的企业组织。该企业需要维护每个分部的顾客信息以及企业总部的管理信息。使用图 F-1 所示的分布式数据库，每个分部均可在本地数据库中维护自己的顾客信息，同时，一个分部无需在本地数据库中存储其他分部的顾客信息，也能实现其他分部顾客信息的访问。此外，公司总部可维护公司各分部的管理信息，分部则可从总部数据库所在位置访问到部分管理信息，而另一部分管理信息则可复制到每个分部的数据库所在位置。

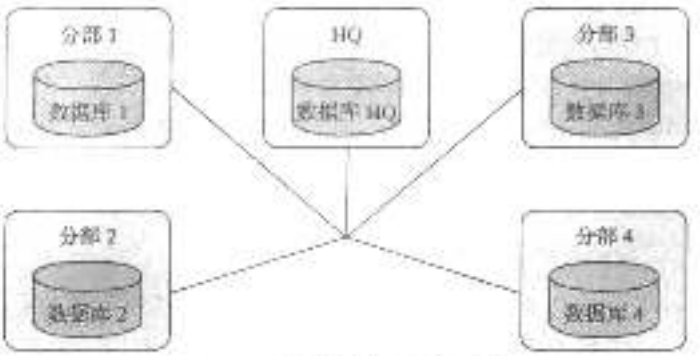


图 F-1 分布式数据库系统示例

终端用户往往并不关心系统分布的实际情况。换言之，用户在使用系统时，将不必知道所访问的数据是存放在本地，还是存放在一个地理位置完全不同的远端。在 DDBS 中，终端用户可采用与使用非分布式数据库相同的方式来使用分布式数据库，这一系统特性（即不必知道数据分布的细节，甚至不必知道数据的分布）称为**分布透明性**（distribution transparency）。

在 DDBS 中，数据库中的数据将分布在大量独立的计算机中，这些计算机彼此间并不分享任意的处理设备或存储设备，且每台计算机都拥有自己的 DBMS。这些计算机通过网络进行连接，因此，它们既可以位于同一建筑物内，也可以散布在不同的地域里。在**同构型** DDBS（homogeneous DDBS）中，所有的计算机将运行同样的 DBMS，而在**异构型** DDBS（heterogeneous DDBS）中，不同的计算机将运行不同的 DBMS（比如因近期两公司并购所致）。例如，异构型 DDBS 中可能存在一台计算机的 DBMS 使用的是 Oracle，而其他计算机使用的是 PostgreSQL。

分布式数据库具有特定的性能优势。分布式数据库允许用户彼此无干扰地在同一时间对数据库的不同部分进行访问，这将使以下情况显著受益：在整个 DDBS 中，大部分查询都

是关于本地用户对本地数据的查询。例如，在分部 1 中，大部分数据查询都是关于本地用户对分部 1 站点数据的查询；在分部 2 中，大部分数据查询都是关于本地用户对分部 2 站点数据的查询。与面向单数据库（非分布式数据库）的查询相比较，由储存在多个位置点的数据库所形成的分布式数据库可降低每台计算机上的查询量。因而，当系统中存在大量的本地查询时，DDBS 可实现更加快速的数据处理。此时，最经常访问的数据将由每台计算机进行本地存储，这将有效减少访问时间。此外，利用多于一个位置点的数据复制，分布式数据库还能提供增强的可靠性及可用性。当从某个位置点的数据库中访问数据出现问题时，还可以从其他复制点中访问数据。

注意：与非分布式数据库相比，分布式数据库将更加复杂。为了完成查询任务，分布式数据库需要更多的功能，如对数据以及所有副本的位置追踪，决定哪个数据副本用于查询，并确保更新被应用到数据的所有副本之中。同时，分布式字典中有关数据库的所有本地及远程数据也必须进行维护。以下是关于复杂性增加的一个例子，在处理一个跨越了整个分布式数据库的查询时，首先需要决定该查询将访问的数据的位置，以鉴别出查询中的哪些部分需要本地数据，哪些部分需要远端数据。若查询所需的数据存放在远端，则将此类查询送至合适的远端系统运行，而远端查询结果一旦返回，这些结果还将与本地结果进行结合。

数据库分片（database fragmentation）是一种基于不同位置点的数据分布策略。在分布式数据库中，一个完整的数据表可以存放于不同的位置点，换言之，表的分片可以存放于不同的位置点。数据库分片可以是水平分片或垂直分片。

在**水平分片**（horizontal fragmentation）中，DDBS 会将表的记录子集存于不同的位置点中。表中的所有列将存放在同一位置点，而只有表的记录子集存放于不同的位置点。要重建完整的记录表，需要在所有分片上执行联合操作。图 F-2 给出了数据表水平分片的例子。

[337]

EMPLOYEE（未分片）

EmpID	EmpName	EmpGender	EmpPhone	EmpBdate
0001	Joe	M	x234	1/11/1985
0002	Sue	F	x345	2/7/1983
0003	Amy	F	x456	8/4/1990
0004	Pat	F	x567	3/8/1971
0005	Mike	M	x678	5/5/1965

EMPLOYEE（水平分片－位置 A）

EmpID	EmpName	EmpGender	EmpPhone	EmpBdate
0001	Joe	M	x234	1/11/1985
0002	Sue	F	x345	2/7/1983
0003	Amy	F	x456	8/4/1990

EMPLOYEE（水平分片－位置 B）

EmpID	EmpName	EmpGender	EmpPhone	EmpBdate
0001	Joe	M	x234	1/11/1985
0004	Pat	F	x567	3/8/1971
0005	Mike	M	x678	5/5/1965

图 F-2 水平分片示例

在本例中，接下来的查询将为两个分片（每片中包含 3 个记录）创建一个联合操作并组

建出完整的数据表（共包含5个记录，因两个分片共享相同的Joe这一记录）：

```
SELECT * FROM employee_location_a
UNION
SELECT * FROM employee_location_b;
```

DDBS的分布透明性允许用户将这一查询简化为：

```
SELECT * FROM employee
```

在**垂直分片**（vertical fragmentation）中，DDBS则会将数据表的列的子集存于不同的位置点中。每一位置点中都会出现表的所有记录，只是记录值并不完整，而仅仅包含了某些列。因每个位置点仅存储了列的子集，因此，在每个垂直分片中，均需要包含表的关键字列。采用关键字作为连接条件连接所有分片，便可重建出完整的数据表。

图F-3给出了表的垂直分片示例。在该例中，以下的查询将实现两个分片的连接并重建出完整的数据表：

```
SELECT a.empid, a.empname, b.empgender,
       a.empphone, b.empbdate
FROM   employee_location_a a, employee_location_b b
WHERE  a.empid = b.empid;
```

EMPLOYEE（未分片）

EmpID	EmpName	EmpGender	EmpPhone	EmpBdate
0001	Joe	M	x234	1/11/1985
0002	Sue	F	x345	2/7/1983
0003	Amy	F	x456	8/4/1990
0004	Pat	F	x567	3/8/1971
0005	Mike	M	x678	5/5/1965

EMPLOYEE（垂直分片－位置A）

EmpID	EmpName	EmpPhone
0001	Joe	x234
0002	Sue	x345
0003	Amy	x456
0004	Pat	x567
0005	Mike	x678

EMPLOYEE（垂直分片－位置B）

EmpID	EmpGender	EmpBdate
0001	M	1/11/1985
0002	F	2/7/1983
0003	F	8/4/1990
0004	F	3/8/1971
0005	M	5/5/1965

图F-3 垂直分片示例

DDBS的分布透明性允许用户将这一查询简化为：

```
SELECT * FROM employee
```

338

在同一个数据表中，我们可以将水平分片与垂直分片结合使用，这一策略称为**混合分片**（mixed fragmentation）。图F-4给出了表的混合分片示例。

EMPLOYEE（未分片）

EmpID	EmpName	EmpGender	EmpPhone	EmpBdate
0001	Joe	M	x234	1/11/1985
0002	Sue	F	x345	2/7/1983
0003	Amy	F	x456	8/4/1990
0004	Pat	F	x567	3/8/1971
0005	Mike	M	x678	5/5/1965

EMPLOYEE（位置 A）

EmpID	EmpName	EmpPhone
0001	Joe	x234
0002	Sue	x345
0003	Amy	x456

EMPLOYEE（位置 B）

EmpID	EmpName	EmpPhone
0001	Joe	x234
0004	Pat	x567
0005	Mike	x678

EMPLOYEE（位置 C）

EmpID	EmpGender	EmpBdate
0001	M	1/11/1985
0002	F	2/7/1983
0003	F	8/4/1990
0004	F	3/8/1971
0005	M	5/5/1965

图 F-4 混合分片示例

在本例中，以下查询可重建出完整的数据表：

```
SELECT a.empid, a.empname, c.empgender,
       a.empphone, c.empbdate
FROM   employee_location_a a, employee_location_c c
WHERE  a.empid = c.empid
UNION
SELECT b.empid, b.empname, c.empgender,
       b.empphone, c.empbdate
FROM   employee_location_b b, employee_location_c c ,
WHERE  b.empid = c.empid;
```

DDBS 的分布透明性允许用户将这一查询简化为：

```
SELECT * FROM employee
```

分布式数据库的**数据复制**（data replication）发生在有多个数据副本需要存储于不同的位置点时。若整个数据库均在分布式系统的每个位置点上进行复制，则称为**完全复制分布式数据库**（fully replicated distributed database）。另一类复制则是仅对部分数据实施多位置点复制，这类复制称为**部分复制**（partial replication）。

339

数据复制的主要优势是可以加快查询速度（因为更多数据可以直接在本地进行处理），并提高数据的可用性，即使是当 DDBS 中某台计算机无法运行时（只要在别的计算机上存储了该计算机的数据副本，即可使用数据）。虽然数据复制可以提升查询性能以及数据的可用

性，但同时也带来了更新操作的复制性。当对一个数据副本进行更新时，必须采取策略来确保所有其他副本被同步更新。一种更新策略是，将数据副本中的某个副本指定为“可区别的”副本，用于实现其他所有副本的同步更新。对于进行更新的数据，只有当其所有副本均由同步器成功更新以后，数据才允许访问。另一种更新策略称为“法定数”策略或“投票”策略，无需同步器，而只需要在多数副本得到更新以后，数据便可允许访问。

联合数据库（federated database）是一类分布式数据库系统，由进入同一系统的一批现存数据库所组成。这些现存的数据库均保持其自主性，同时允许相互合作与数据分享，以形成一个新的数据库。关于联合数据库的一个示例是图书馆联合数据库，该数据库由全国各地不同大学的图书馆数据库所组成。每个图书馆有自己的数据库，存放着该图书馆的各种书目信息。各个图书馆联合起来便形成一个联合数据库，用户可以使用联合数据库实现对所有图书馆的信息访问。联合数据库不必提供透明性访问。当访问整个系统时，联合数据库需要具有全局联合的视角，因为每个数据库都有自己的工作模式。

F.1 并行数据库

在**并行数据库**（parallel database）系统中，各个计算机可以同时对同一数据集的不同部分实施同一任务并完成同样的操作。换言之，计算机将并行地进行工作。现代化的并行数据库采用了**大规模并行处理**（massively parallel processing，MPP）技术。MPP是一个行业术语，指的是大量独立工作的计算机处理器以并行方式执行单个的程序。并行计算存在着多种架构方法，这些架构方法的差异在于计算机处理器是共享或是具有各自的磁盘存储器与内存。基于此，并行系统中的处理器可以共享磁盘存储器与内存，或是具有各自的内存但共享磁盘存储器。当并行系统中的每个处理器都有自己的内存与磁盘存储器时，该系统称为**无共享MPP架构**（shared-nothing MPP architecture）。

当需要处理的数据量很大时，数据库操作的并行性能很好地提升系统的性能。这正是很多数据仓库系统实施中采用并行RDBMS的原因，如Teradata、Greenplum。

在附录J中，我们将介绍并行计算的使用，用以实现MapReduce方法对非结构化数据的处理。并行计算也常用于**云计算系统**（cloud computing system）中，使用网络实现数据存储与处理服务的交付，这些服务由服务提供商来进行引导。

F.2 云计算

云系统已经成为了一种计算模式以及一种商业模式，它能为用户按需提供计算资源，并允许用户按需增减资源数目。所提供的计算资源包含计算机的软硬件资源，甚至还可以是某个特定的计算平台。采用云技术可以使组织省去硬件成本投资以及雇用职员等方面的开销。 340]

云的主要特征为：按需访问计算资源、按次记费模式、灵活性、虚拟性、分布式/高度并行性。

云可以为消费者按需提供资源。消费者仅仅为所使用的资源付费，并按使用次数付费。因消费者可以按照所需来申请资源，因此不必为维护无用资源而花费代价。例如，在消费者市场进行促销、销售、庆典及一些特定的场合中，数据应用的工作负载将会激增，若公司调度可用资源以应对这一激增，则将是一个错误的决策，因为这些资源在其他时候将毫无用处。

云服务供应商所提供的服务可以划分为3类：基础设施即服务、平台即服务及软件即服务。

基础设施即服务（Infrastructure as a Service，IaaS）提供基于网络的硬件标准服务。消费者可以租用硬件资源，如服务器空间、网络设备、存储器、处理周期、存储空间等。

平台即服务（Platform as a Service，PaaS）提供消费者在创建应用时所需的各种资源。它将软件层压缩为一个平台，该平台集成了操作系统、中间件、应用软件及开发环境。消费者可以利用接口与平台实现互动，平台则将根据消费者的需求来实现自我维护与裁剪。

软件即服务（Software as a Service，SaaS）使应用具有服务器请求式特征：应用被放置在服务器上，而消费者则通过网络来使用这些应用。一个简单的示例是，一个运行在云中的软件可以同时被多个客户端使用。

数据库即服务（Database as a Service，DaaS）则是 SaaS 的一个特例，专门用于云中的数据库管理。

当需要存储的数据量急剧增加时，云架构可以解决大规模数据处理所面临的关键难题。云计算环境允许数据库基于动态负载自动地进行增减。云被视为适宜分析型数据管理应用的技术，这些应用将对多个操作型数据库中的历史数据进行处理，且仅需少量的甚至无需数据更新。不同于分析型数据库，操作型数据库需要进行数据更新。当存在数据更新时，维护连贯的数据值将会是一项复杂的工作，尤其是当数据在较大的地理范围内进行广泛复制时，维护连贯的数据值将可能会影响系统性能。

人们用“灵活”一词来描述云，这是由于它可以适应资源的需求增长与需求下降。云中的虚拟技术允许多个用户同时使用同一设备。云中的服务器提供了多个操作实例，使得多个用户可以在同一台计算机上使用不同的操作系统来运行一个应用。虚拟软件可以为用户模拟出多台物理机。例如，一个客户可选用一个特定的操作系统来运行其应用。

云是典型的分布式结构，可包含多个数据中心，这些数据中心可以跨越广泛的地理位置。以分布式的方式解决问题是云技术的优势所在，这相当于用多台机器以并行方式快速实现问题的处理。云中的服务器并不一定是昂贵的，很多云中包含的都是常见的计算机。云中任何组件失效的可能性将被创造成为云的计算模式。处理过程可以复制，从而确保任何失效处理都能够快速地进行恢复。类似地，通过在不同位置点分布式地创建一定数量的数据副本，可获得数据的可靠性及可用性保障。

云中的数据存储过程将关系到特定的安全与隐私问题。数据移出仓库并存放到第三方提供商的服务器中，这潜在地增大了安全隐患。企业若要在云中进行数据存储，则需要信任云主机。在云中，数据可以被存储在全球的任何地方。不同的国家有不同的法规来对待数据的合法存取。任一政府均可以利用法律方式强制访问存储在本国的数据，此时的数据移交将无需通知数据的持有者。在云系统中，关于异常账单与拒付款账单的责任问题也可能出现。虽然存在这些挑战，但云计算依然表现出增长的趋势，并被世界范围内越来越多的公司与组织采纳。

[341]
[342]

数据挖掘

数据挖掘（Data mining）被广义地定义为在大量数据中发现新奇而有趣的模式的过程。数据挖掘技术与算法起源于很多不同的领域，如数学、统计学、机器学习以及数据库。显然在标准数据分析（如回归分析、OLAP）与数据挖掘之间并没有明显的区分，但后者常常包含了大量可能的发现，且这些发现中仅有少量是合理的。

这些发现可能令人吃惊，前所未知，有时候甚至是反直觉的。例如，在大型超市链中发现畅销商品并不被看做数据挖掘，而发现常常一起销售的商品则是数据挖掘的典范。

数据仓库是数据挖掘的主要数据源之一。它提供了高质量的、完整的、企业范围的数据，可用于回答各种问题，并发现令人感兴趣的新商业模式。

一类最典型而普遍的数据挖掘应用称为**预测分析**（predictive analytics）。它利用以往的数据来预测将来的事件。其示例包括：使用过去的信誉历史来判断新贷款人的信誉情况；从典型的信用卡交易中获得使用模式，并对照有悖于这些使用模式的新购买行为，以标记出潜在的欺骗行为；使用遗传与环境信息来预测病人感染特定疾病的可能。很多技术可用于构造预测模型，如分类、聚类及回归等。

有关各种数据挖掘方法的具体描述已经超出了本书的研究范畴。但是，为了能够对数据挖掘相关机制与应用进行说明，我们将给出一个最为常用的方法概述，即关联规则挖掘。

G.1 关联规则挖掘

关联规则挖掘（association rule mining）也称为**购物篮分析**（market basket analysis），是从数据库视角进行数据挖掘的典范。该方法起源于20世纪90年代早期的数据库搜索，尤其适用于通过维度建模（带来了星型模式）实现数据仓库与数据集市的创建。关联规则挖掘的最简单定义是：找到那些常常可能一起销售的物品组合。术语“购物篮分析”也是因该类挖掘的原始数据为超市结账台数据而得名。其目标则是在所有商品随机购买的情况下，发现那些比我们预想的更有可能一起购买的商品。一些控制结果并不令人吃惊（如热狗与番茄酱经常一起销售，谷物与牛奶也常常一起销售），而引用在学术文献中的一个难以预料的事实是，啤酒与尿布也常会一起销售。这一结果已经被不同的研究先后多次证实。关联规则挖掘中的重要问题是发现示意式模式，这些模式可用于实现市场销售额度的上涨并增加利润。

G.2 关联规则挖掘实例

关联规则挖掘揭示了交易物品的相关性，该相关性采用以下形式进行描述：

一个包含X的交易很可能也包含Y

记为交易规则 X → Y，其中X与Y代表交易物品集。利用关联规则挖掘从超市结账台中发现那些商品被放在同一购物篮中，则该关联可采用以下规则形式来表示：

购买 X 的顾客在同一次购物活动中很可能也购买 Y

每个关联规则具有两个量化指标：支持度与置信度。规则 X → Y 的支持度定义为所有包含了 X 与 Y 的交易数与总交易数的比值：

支持度 = 同时包含了 X 与 Y 的交易数 / 总的交易数

规则 X → Y 的置信度定义为所有包含 X 与 Y 的交易数与所有包含 X 的交易数的比值：

置信度 = 同时包含了 X 与 Y 的交易数 / 包含了 X 的交易数

从直觉上分析，支持度量化了规则的意义，因而我们将对具有相对较高支持度的规则产生兴趣。置信度则量化了关联的强度，因此那些具有低置信度的规则是毫无意义的，即使它的支持度很高。由此可知，一个有意义的规则是那些具有高支持度与置信度（超过了特定的临界值）的交易物品关系，而临界值则由分析师来确定。

为了说明支持度与置信度规则如何促进关联规则挖掘，考虑以下的简单实例：

在 ZAGI 零售公司的零售链中，出售了上百种不同的商品，其中包括商品 A、B、C。

假设我们正在寻找支持度大于等于 0.5% 且信任度大于等于 50% 的关联规则。通常，这些临界值将由分析师来确定，并会在数据挖掘的后续迭代中进行修正。表 G-1 显示了所包含的交易数：

- 物品 A。
- 物品 B。
- 物品 C。
- 物品 A、B。
- 物品 A、C。
- 物品 B、C。
- 物品 A、B、C。

表 G-1 交易统计 （单位：个）

Total	A	B	C	A&B	A&C	B&C	A&B&C
500	10	7	11	4	5	2	1

交易总数为 500 000；10 000 包含物品 A；7 000 包含物品 B；11 000 包含物品 C；4 000 包含物品 A、B，等等。注意，包含物品 A、B 的交易是包含物品 A 的交易的子集。

物品 A、物品 B 这一组的支持度是 0.8%（规则 A → B 与规则 B → A 的支持度是相同的，同为 4/500=0.8%。因为包含同组物品的两个规则的支持度都是相同的）。物品 A 的交易数是 10 000，物品 B 的交易数是 7 000。因此，规则 A → B 的置信度是 40%，而规则 B → A 的置信度是 57.14%（A → B 的置信度是 4/10=40%，B → A 的置信度是 4/7=57.14%）。该例中所有可能规则的支持度及置信度见表 G-2。

344

表 G-2 支持度及置信度值

规则	支持度	置信度	规则	支持度	置信度
A → B	0.80%	40.00%	A, B → C	0.20%	25.00%
B → A	0.80%	57.15%	C → A, B	0.20%	9.09%
A → C	1.00%	50.00%	A, C → B	0.20%	20.00%
C → A	1.00%	45.45%	B → A, C	0.20%	14.29%
B → C	0.40%	28.57%	B, C → A	0.20%	50.00%
C → B	0.40%	18.18%	A → B, C	0.20%	10.00%

表G-2所示的12个可能的规则中，仅仅以下两个能满足支持度大于等于0.5%且置信度大于等于50%的临界值条件：

物品B → 物品A（支持度 = 0.80%，置信度 = 57%）

物品A → 物品C（支持度 = 1.00%，置信度 = 50%）

临界值的设定可以确保购物篮分析师在所有可能的规则中发现最有意义的规则。一旦有所发现，这些规则便可影响商业决策。例如，零售商发现有两种商品特别容易一起销售，他可以以几种方式来利用这一发现。他们可以将两种商品存放在一起，以方便顾客购物；也可以将两种商品分开存放，以使有需要购买这两种商品的顾客能看到商店中的更多商品；还可以在一种商品上实施促销活动但提高另一种商品的价格。在零售商对自身数据进行挖掘之前，这些改变了现有商业实践活动的方式还从未进入过人们的视野。

XML

H.1 标记语言

标记语言（markup language）是指在文档中进行文本注释（即“标记”）的语言，这些注释会将各种功能添加到文本中。基于标记语言的类型，注释可用于各种用途，如指明文档中已经注释的文本如何生成格式、如何显示、如何结构化等。

超文本标记语言（hypertext markup language，HTML）是标记语言的一种，HTML 提供了各种功能，以描述如何显示 Web 页面信息。HTML 的目的就是帮助完成数据的显示。

请看图 H-1 中的 HTML 代码，这些代码存放在标题为 HTMLExample.html 的文件中。代码中包含了可 HTML 标签进行注释的文本信息。例如，由起始标签 与结束标签 所注释的文本是被标记为分段显示的文本，而由起始标签 与结束标签 所注释的文本是被标记为粗体显示的文本。

```
<html>
   <head>
      <title> Web page EXAMPLE</title>
   </head>
   <body>
      <p>This is an example of a HTML Web page.</p>
      <p><b>This part is in bold font.</b></p>
      <p><i>This part is in italic font.</i></p>
   </body>
</html>
```

图 H-1 HTML 代码示例

HTMLExample.html 包含了图 H-1 中的 HTML 代码，它可以由 Web 浏览器恢复为图 H-2 中显示的 Web 页面。

图 H-2 Web 页面示例

H.2 XML

扩展的标记语言（extensible markup language，XML）是在文档中增加了结构与语义的标记语言。XML 通常的用法是帮助数据实现从数据库到电子商务应用的转换。此时，数据首先从数据库中提取出来，格式化为 XML 文档，再进行转换并显示在使用 HTML 的 Web 页面上。以下是关于 XML 的最基本性能的简要概述。

XML 文档的基础构件称为“元素”。XML 文档由“根元素”构成，根元素中包含了文

档的其他所有元素。XML 文档中的元素由起始标签 <元素名> 与结束标签 </元素名> 进行标识。"单元素"即为真实的数据值（如数值或字符），"复合元素"则是指包含了其他元素的元素。

请参图 H-3 中的 XML 代码。代码第一行是 XML 声明[1]，剩余部分则是 HAFH 数据库（曾用于第 3 章和第 5 章）的一个子集，描述了大楼、公寓与企业客户。

```
<?xml version="1.0" ?>
<HAFH>
<building>
            <BuildingID>B1</BuildingID>
            <BNoOfFloors>5</BNoOfFloors>
            <Apartment>
                        <AptNo>21</AptNo>
                        <ANoOfBedrooms>1</ANoOfBedrooms>
            </Apartment>
            <Apartment>
                        <AptNo>41</AptNo>
                        <ANoOfBedrooms>1</ANoOfBedrooms>
                        <CorporateClient>
                                    <CCID>C111</CCID>
                                    <CCName>BlingNotes</CCName>
                        </CorporateClient>
            </Apartment>
</building>
<building>
            <BuildingID>B2</BuildingID>
            <BNoOfFloors>6</BNoOfFloors>
            ...
            ...
</HAFH>
```

图 H-3 XML 代码示例

在这个简单实例中，树根是包含了大楼 Building 元素的 HAFH 复合元素。每个 Building 元素也是一个复合元素，包含了 BuildingID 及 BNoOfFloors 两个单元素，以及一个或多个 Apartment 元素。每个 Apartment 都是一个复合元素，包含 AptNo 及 ANoOfBedrooms 两个单元素。此外，一些 Apartment 中还包含了 CorporateClient 元素。而每个 CorporateClient 元素也是一个复合元素，并包含 CCID 及 CCName 两个单元素。

在 XML 中，所有的元素都是分层组织的。术语"父亲元素"、"孩子元素"、"孙子元素"及"兄弟元素"用于描述元素间的分层关系。例如，Building 是孩子元素 BuildingID、BNoOfFloors 及 Apartment 的父亲元素，而 Apartment 是孩子元素 AptNo、ANoOfBedrooms 及 CorporateClient 的父亲元素。同时，BuildingID、BNoOfFloors、Apartment、AptNo、ANoOfBedrooms、CorporateClient、CCID 及 CCName 都是 Building 的孙子元素。兄弟元素则是指共享同个父亲元素的元素。例如，BuildingID 和 BNoOfFloors 是兄弟元素。 347

因为 XML 的数据是分层的，所以 XML 的数据模式可描述为树。图 H-4 中给出了图 H-3 的 HAFHInfo.xml 文档的数据模式。

正如图 H-3 所示，数据库中所提取的数据可以由 XML 文档来抓取。此外，XML 文档自身也可被存储为数据库。将 XML 存入数据库可采用多种不同的方法来实现。XML 文档可以文本方式存入数据库系统中，此时需要 DBMS 具有文档处理模块。此外，如果所有的文档均具有相同的结构，则 XML 文档内容可存储为数据元素。此时，XML 模式将必然会映

[1] XML 声明用于确定该文档为 XML 文档，同时指定 XML 的标准版本（本例中为 1.0 版）。

射到数据库模式中，也可以从一个现有的关系数据库中创建出 XML 文档，再将其存放回数据库中。“原生 XML 数据库”是一类特殊的 DBMS，它被专门设计用于存储和处理 XML 数据。原生 XML 数据库具有一个 XML 文档作为基本的存储单元。很多 DBMS 中内置有相关功能，可使用典型的关系模式将 XML 元素中的数据呈现为数据表的行。

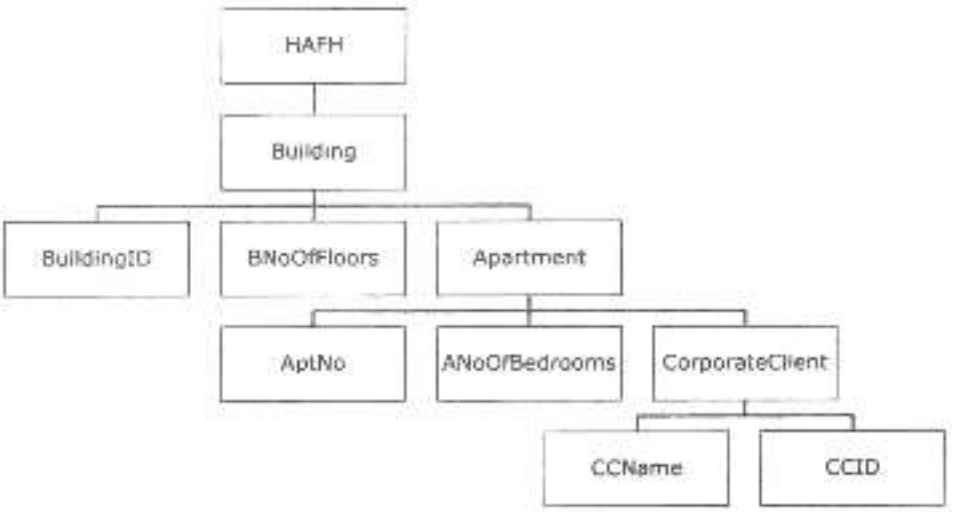


图 H-4 图 H-3 的 XML 数据模式

H.3 XML 查询

XML 通常用于数据库与应用之间的数据交换。此时，XML 文档中可以包含相当复杂的元素结构，因此有必要采用某种机制来遍历 XML 元素的树结构，并对其中包含的信息进行提取。

XML 路径语言（XML Path Language，XPath）是利用 XML 文档的树状表达式进行工作的一种简单语言，该语言允许用户遍历树。XPath 表达式将从树中返回一个元素节点集，这些元素节点能满足表达式所指定的模式。分隔符“/”与“//”用于指定一个标签的孩子及孙子。其描述如下。

/：该标签必须作为先前父亲标签的直接后裔（即孩子）而出现。

//：该标签可以作为先前标签的任意级别孙子标签而出现。

例如，要访问 HAFH 数据库中的 building，则可给出下述表达：

348

```
/HAFH/building
```

另一个例子，考虑获取 bedroom 大于 1 的公寓数的语句：

```
//apartment [ANoOfBedrooms > 1]/AptNo
```

XQuery 是一种更加广泛地用于 XML 文档查询的语言。XQuery 提供了对 XML 文档的查询功能，这就像 SQL 提供了对关系数据库的查询功能一样。XQuery 中使用 XPath 表达式，该表达式是一个“FLWOR”（For Let Where Order by Return）表达式，类似于 SQL 中的 SELECT…FROM…WHERE 表达式。其中，For 语句负责从 XML 文档中提取元素；Let 语句允许创建变量并为变量分配值；Where 语句基于逻辑表达式来完成元素的筛选；Order by 语句将列结果进行排序；Return 语句用于指定返回结果。在 FLWOR 表达式中，

For 及 Let 语句可以以任意次序出现任意次；Let、Where、Order by 语句是可选的，但 For 与 Return 语句则是必需的。为了对表达式进行说明，我们考虑以下例子，并假设图 H-3 所显示的 XML 文档被存为名为是 hafhinfo.xml 的文档，并存储在网络服务器 www.hafhrealty.com 中：

```
FOR $x in doc(www.hafhrealty.com/hafhinfo.xml)
RETURN <rec> $x/CorpClient/ccname, $x/CorpClient/ccid </rec>
```

该 XQuery 查询用于请求企业客户的姓名及 ID 列表。下面考虑其扩展示例：请求企业客户的姓名及 ID 列表，要求这些客户所租赁的公寓的 bedroom 不少于一间，并按照企业客户的 ID 进行排序：

```
FOR $x in doc(www.hafh.com/hafhinfo.xml)
LET $minbedroom := 2
WHERE $x/AMOUNTbedrooms >= $minbedroom
ORDER BY $x/CorpClient/ccid
RETURN <rec> $x/CorpClient/ccname, $x/CorpClient/ccid </rec>
```

除了一些用于提取 XML 文档中的特定元素与数据的有效机制外，我们还需要一些有效的方式，可基于数据库的数据查询自动地构造出 XML 文档。当今很多主流的数据库管理系统均已利用 SQL/XML 实现了 XML 功能的合并，这是 SQL 的扩展应用，指定了 SQL 与 XML 的结合使用。以下是用于说明 SQL/XML 绑定功能的一个简单实例。

考虑 SQL Query A，从 HAFH 数据库的 INSPECTOR 表中进行数据检索：

SQL Query A

```
SELECT  i.inscid, i.insname
FROM    inspector i;
```

该查询结果的输出如下：

SQL Query A 的结果

inscid	insname
I11	Jane
I22	Niko
I33	Mick

SQL/XML QueryAX 利用 SQL/XML 函数 xmlelement() 创建了一个 XML 元素。

SQL/XML Query AX

```
SELECT  xmlelement(name "inspector",
        xmlelement(name "inscid", i.inscid),
        xmlelement(name "insname", i.insname))
FROM    inspector i;
```

349

SQL/XML QueryAX 的结果输出如下：

SQL/XML Query AX 的结果

```
<inspector>
 <inscid>I11</inscid>
 <insname>Jane</insname>
</inspector>
```

```
<inspector>
 <inscid>I22</inscid>
 <insname>Niko</insname>
</inspector>
```

```
<inspector>
 <inscid>I33</inscid>
 <insname>Mick</insname>
</inspector>
```

SQL Query A 将从数据表 INSPECTOR 中生成数据的行与列，而 SQL/XML Query AX 则是基于数据表 INSPECTOR 中的数据生成 XML 元素。

XML 的一个典型应用是基于 Web 页面来呈现数据库中的信息。将数据库检索得到的 XML 元素呈现在 Web 页面的方式之一是使用可扩展样式语言（XSL）。

考虑图 H-5 中的 XML 文件 inspectors.xml。代码的第一行是 XML 声明。第二行则是指定 XSL 文件 insptoweb1.xsl（图 H-6 中所示）用于描述文件 inspectors.xml 如何显示为 Web 页面。XML 文件 inspectors.xml 的剩余代码则包含了 Query AX 从表 INSPECTOR 中检索到的相关内容。

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="insptoweb1.xsl"?>


<buildinginspectors>
            <inspector>
                    <insid>I11</insid>
                    <insname>Jane</insname>
            </inspector>
            <inspector>
                    <insid>I22</insid>
                    <insname>Nikon</insname>
            </inspector>
            <inspector>
                    <insid>I33</insid>
                    <insname>Rick</insname>
            </inspector>
</buildinginspectors>
```

图 H-5 XML 文件 inspectors.xml（将显示为 Web 页面）

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">

  <html>
  <head>
  <title> Web page XML EXAMPLE1</title>
  </head>
  <body>
    <b>Building Inspectors</b>
     <xsl:for-each select="buildinginspectors/inspector">
       <p> --- </p>
       <p>Inspector ID: <xsl:value-of select="insid" /></p>
       <p>Inspector Name: <xsl:value-of select="insname" /></p>
     </xsl:for-each>
  </body>
  </html>

</xsl:template>
</xsl:stylesheet>
```

图 H-6 XSL 文件 insptoweb1.xsl（用于描述 inspectors.xml 如何显示为 Web 页面）

图 H-6 中的 XSL 文件 insptoweb1.xsl 描述了文件 inspectors.xml 如何显示为 Web 页面。

图 H-6 的前四行与后两行代码包含了 XSL 文档中一些必要的实用信息。紧跟在前四行代码之后的标签 <html>、<title>、<body>、<b> 及 <p> 是标准的 HTML 标签，正如图 H-1 与 H-2 示例中说明的那样。中间部分显示为黑体的代码，将运用 XSL 元素 <xsl:for-each>（在参

考该 XSL 文件的 XML 文件中）来选择每个由 XPath 表达式创建的检查器 inspector 所识别出的 XML 树元素。当 XML 文件 inspectors.xml 在 Web 浏览器中打开时，它将显示为图 H-7。 350

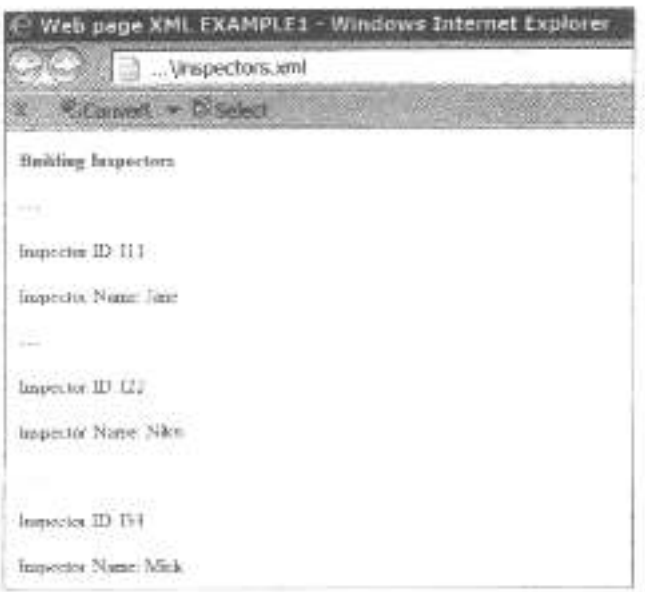


图 H-7 XML 文件 inspectors.xml(Web 页面显示结果)

为了说明同样的 XML 元素如何生成不同的格式，图 H-8 给出的文件 insptoweb2.xsl 为文件 inspectors.xml 的 XSL 代码的另一版本。该代码中使用 HTML 标签引入了着色与表格格式。例如，标签 <tr bgcolor="#808080"> 用于指定表头的背景色为灰色。

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">

  <html>
  <head>
  <title> Web page XML EXAMPLE2</title>
  </head>
  <body>
    <h>Building Inspectors</h>
    <table border="1">
      <tr bgcolor="#808080">
        <th>Inspector ID</th>
        <th>Inspector Name</th>
      </tr>
      <xsl:for-each select="buildinginspectors/inspector">
      <tr>
        <td><xsl:value-of select="inseid" /></td>
        <td><xsl:value of select="insname" /></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>

</xsl:template>
</xsl:stylesheet>
```

图 H-8 XSL 文件 insptoweb2.xsl(用于描述 inspectors.xml 如何显示为 Web 页面)

假设 XML 文件 inspectors.xml 将参考 XSL 文件 insptoweb2.xsl 而不是 insptoweb1.xsl，
351 那么，XML 文件 inspectors.xml 在使用 Web 浏览器打开时，将显示为图 H-9 中的 Web 页面。

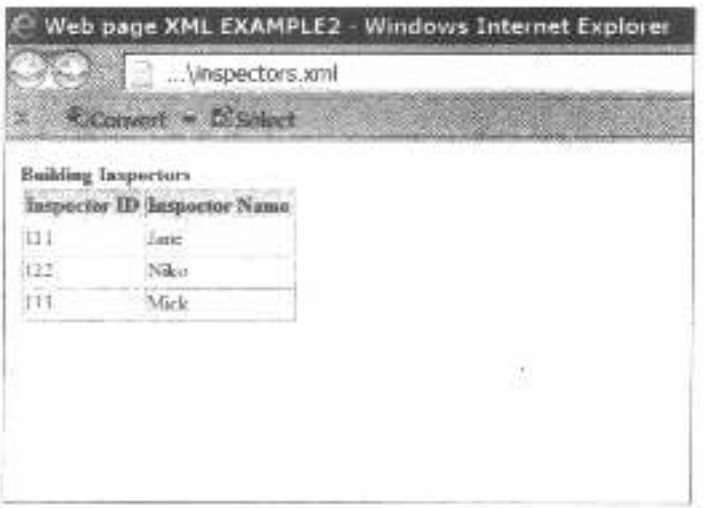


图 H-9 XML 文件 inspectors.xml（Web 页面另一显示结果）

如前所述，数据库中所提取的利用 XML 标签进行注释的 XML 文档内容，可以转换为不同的结果，并以不同的方式进行排列、格式设置和显示。

本附录对一些最基本的 XML 概念进行了简要的说明，着重于对数据库相关的 XML 功
352 能的介绍。

NoSQL 数据库

作为一个广义的术语，**NoSQL 数据库**（NoSQL database）是一种不基于关系模型且不使用 SQL 作为查询语言的数据库。除不基于 RDBMS 技术外，NoSQL 数据库的一个主要的区别性特征是它的灵活性以及可扩展的数据模型。在关系数据库中，数据库模式会预先捕捉数据库中元素的语义。而在 NoSQL 数据库中，数据则不必存储在由一个已经组建的数据库模式来描述的结构中。例如，在很多 NoSQL 数据库中，数据被组织成简单的码值对。在一个码值对中，数据库中所存储的数据实例则是一个通过给定码便可进行检索的数值，该数值可以是简单结构（如一个单词或数字），也可以是具有自身语义的复杂结构。这些值将由数据库之外的应用进行处理，而并不属于数据库系统本身。

我们将使用 NoSQL 数据库中最著名的例子——MongoDB——一个商务酒店相关的数据库，作为一个简单实例对 NoSQL 的数据模型、查询语言及扩展性做出说明。

I.1 NoSQL 数据库实例：MongoDB

MongoDB 是一个所谓的面向文档的 NoSQL 数据库，用于实现文档资料的收集。术语收集相当于 RDBMS 中表的概念，而文档则相当于 RDBMS 中行的概念。收集可以在运行时创建，任何给定收集采用的文档结构并不要求相同，虽然在很多情况下，这些结构是相似的。

考虑以下这个关于在线收集出租房信息（如旅店、汽车旅馆、私人出租房的床位、早餐、房间、以及露营地等）的公司实例。数据将在公司的 Web 站点进行收集并显示。包括各种出租房信息，如比率、房屋类型、舒适度等。要为这些不同类型的出租房设计一个统一的关系模式是行不通的。但我们可以做一些有意义的模拟研究，并产生一种可随着时间而进化的模式。

在 MongoDB 中，这种旅馆文档的收集可以渐进地进行创建，该创建过程不必定义任何模式，因为此时本就没有一个通用的模式。

例如，文档中可涵盖旅馆的最基本信息，如名字、地址、数字比率。以下语句实现了三个文档的创建并存于数据库中：

```
h1 = {name: "Metro Blu", address: "Chicago, IL", rating: 3.5}
db.hotels.save(h1)
h2 = {name: "Experientia", address: "New York, NY", rating: 4}
db.hotels.save(h2)
h3 = {name: "Xazu Hotel", address: "San Francisco, CA", rating: 4.5}
db.hotels.save(h3)
```

此时，数据库中包含了一个名为 hotels 的收集，该收集中包含三个文档。利用以下查询，所有的文档都可以在数据库中找到（等价于 SQL 中的“SELECT”语句）：

```
db.hotels.find()
```

注意，hotels 收集，或该收集中的文档模式不需要进行声明或定义。若想要增加一个不带比率的新旅馆，则可用以下语句创建出一个不带比率信息的新文档：

```
h4 = {name: "Solana", address: "Los Angeles, CA"}
db.hotels.save(h4)
```

所有在加利福尼亚的旅馆都可以用以下查询找到：

353

```
db.hotels.find( { address : { $regex : "CA" } } );
```

该查询使用了一个规范的表达式来查找地址中包含“CA”中的所有文档（类似于 SQL 中的查询“SELECT * FROM Hotels WHERE address LIKE '%CA%'”）。

若公司决定为一些旅馆增加额外信息，则以下更新便可应用于每个 hotel 文档：

```
db.hotels.update( { name:"Zazu Hotel" }, { $set : {wifi: "free"} } )
db.hotels.update( { name:"Zazu Hotel" }, { $set : {parking: 45} } )
```

这些语句将为 Zazu Hotel 条目增加一些新的特征，表明了 Wi-Fi（免费）的可用性与停车价格（$45）。此时我们将不必再转换模式并将数据迁入新转换的模式，这些只是关系数据库中的过程。

当然，获取这些灵活性所带来的代价则是有限的查询能力，尤其是文档的连接操作并没有被 MongoDB 直接地进行支持，因而，开发人员必须使用常规的编程语言（在 MongoDB 中为 JavaScript）来实现这些性能。其实现过程涵盖了在需要进行连接的收集上进行嵌套循环的过程。任何性能优化，例如索引的使用，对每个收集中的文档的物理布局的利用，都必须由开发人员进行处理。相比于 SQL 中两个表的连接查询 JOIN 的易操作性，上述过程给开发人员带来了一定的负担。

I.2 NoSQL 数据库与关系数据库

NoSQL 数据库与关系数据库在很多特征上均存在差异，此处我们主要总结一些最基本的差异。

NoSQL 数据库具有不固定的模式。正如上述例子中所展示的那样，它提供了比关系数据库更大的灵活性，但也致使查询过程实质上更加复杂。

NoSQL 的另一个普遍特征是分布式与“水平扩展”。NoSQL 数据库是典型的分布式数据存储，用来处理巨大的数据量（MongoDB 的名字就来自于术语“humongous”（巨大的））。NoSQL 数据库可以运行在大量相对廉价（或日用型）的服务器上，它们的性能以及处理大数据的能力可以通过按需增加更多的日用服务器来进行提升。这一特征称为水平扩展，是相对于中心型关系数据库的垂直扩展而言的。垂直扩展通过增加中心服务器的处理性能来提升对大数据的处理能力（不过关系数据库也能够配置为分布式与并行计算，如附录 F 所述）。NoSQL 数据库不能自如地使用所有数据库操作（可参考上述例子中关于 NoSQL 缺少 JOIN 操作的讨论），但还是存在一类相对简单的数据库任务适于利用 NoSQL 进行处理（例如，对大量简单文本文件的存储与检索，如微博）。对于这些任务，利用 NoSQL 数据库的水平扩展来分散数据与处理问题，比使用关系数据库更加简洁并且更能胜任处理任务。

NoSQL 数据库与关系数据库的另一点不同在于其处理数据库事务的方式。关系数据库中的 RDBMS 将确保数据库事务的 ACID 特性。首字母缩写 ACID 代表原子性、一致性、隔离性、持久性。事务的原子性可以确保数据库事务或者全部执行，或者根本不执行。例如，一个从账户 A 至账户 B 的银行转账事务，将包含从账户 A 中提款，并在账户 B 中存入，则

以上过程或者全部执行，或者全不执行。一致性用于确保每个事务都必须使数据库处于合法状态，所有约束、商业规则等都必须遵从一致性规定。隔离性用于确保每个事务不会被其他事务所干扰。持久性用于确保事务对数据库的作用是持久的。另一方面，NoSQL 数据库事务则通常表现出 BASE 特性（BASE 代表基本有效的软状态最终一致性）而不是 ACID 特性。在这种 NoSQL 数据库中，添加与修改操作是不会立即对用户生效，但最终都可以正确地反映在系统中。不依附于 ACID 特性可确保更好的简洁性与更低代价的可扩展性。 354

ACID 特性对于很多企业数据库而言通常是十分重要的，比如金融、计费以及供应链数据库，因为它们具有相同的准确性要求，以及正确指导商业行为所必需的时间线。但是，在一些其他场景中，比如使用 Web 数据库来处理用户邮件及微博时，BASE 特性就已经足够用了。例如，当一个挑剔的客户没有立即收到转账结果时，这可能会导致一个严重的商业后果；而另一方面，为最近所提交的微博显示在系统上而进行短暂的等待，则并不会带来极其严重的后果。

关系数据库与 NoSQL 数据库还有一点不同在于，关系数据库中采用 SQL 作为 RDBMS 的标准语言，而在 NoSQL 数据库所使用的各种语言与方法中，则不存在标准与统一的形式。

当前公司的状况是，用 RDBMS 实现的关系数据库应用于绝大多数的商业场景。但是，NoSQL 数据库为某些环境提供了有吸引力的且有效的可选方案，正如在以上示例和讨论中所展示的一样。 355

大 数 据

由商业机构、科研单位、政府及其他组织所产生的数据量正以不断增长的速率持续上涨。这一增长速率已经随着各种传感器及智能设备的更新换代而显著升高。这些设备产生大量数据的例子包括：

- 智能手机每隔几秒钟就会广播它们的位置
- 汽车芯片每秒钟需要完成数以千计的诊断测试。
- 摄像机将对公共及私人场所进行持续的记录。
- 可发送无线信号（这些信号在带有 RFID 标签的各个购物车站点被读取）的无线射频识别（RFID）标签将伴随着厂商及顾客所形成的供应链中。

大数据（big data）是一个广义的术语，指的是大量不同种类且快速增长的数据，这些数据无法正式地进行建模。大数据是非结构化的[⊖]，正因如此，它们不能直接组织成由行和列构成的表。大数据是特殊化的，因为它不仅缺乏结构且尺寸巨大。同时数据本身也是异质的。在一个典型的大公司里，大数据可以包含从智能设备、网络日志、公共记录、社会媒体及无数从其他数据出口所获得的各种数据。

企业已经认识到数据的价值，并建立了事件数据库和数据仓库来处理正式建模方式下可进行操作与解析的数据。如今，企业也正在考虑如何利用这些无结构性的数据，它们在数量上更加巨大且增长速度更快。标准数据库及数据仓库技术，如关系模型与维度模型，无法充分地采集大数据的多样性与异质性。此外，关系型 DBMS 在处理大数据的复杂查询时常常不能提供令人满意的性能。

这样，便出现了几种处理大数据的新方法，包括：

- MapReduce 框架（本附录中将进行描述），在 2004 年首先由 Google 引入。
- NoSQL 数据库，在其所管理的数据上并没有采用关系结构（已在附录 I 中进行描述）。
- 关系数据库技术的扩展，如为 RDBMS 设置大量的并行处理架构（已在附录 F 中进行描述）以及列存储。在列存储中，数据将以列而非行的方式实现物理存储，从而为一些特定类型的查询提供了性能优势，如列的聚集。

MAPREDUCE

为了对大数据的处理方法进行说明，我们将简要地给出 MapReduce 计算框架的讨论，并给出一个简单实例来进行说明。

大数据的一个典型实例是大量非结构化的文本数据，如文章、评论、微博等。为了以标准数据仓库的方式来管理这些数据，组织机构需要确定一个固定的 ETL。但是，由于数据的

⊖ “非结构化数据”、“半结构化数据”、“结构化数据”与“经型级结构化数据”是一些不同的术语，用于描述那些具有少量或甚不具有正式元数据的数据。

多样性，以及可能会有新的数据源持续地增加进来，所以不可能找到在不丢失信息情况下实现数据转换的方法。因而，这些数据最好是保留成原始形式。例如，给定某产品或服务的书面评论，我们可能并不清楚该评论的哪些特性需要提取、哪些特性需要丢弃，因而，最好对该评论完整地进行保留。因此，处理这些数据将不可能采用传统的数据库或数据仓库技术。即使这些评论可以存储为文本数据类型的列，但在这些列所对应的数以亿计的记录条目上的计算工作将会变得非常缓慢，因为普通的数据库与数据仓库并不是用来有效地处理巨大规模的数据的。

2004年由Google发明的MapReduce框架是一种解决大数据计算问题的方法。MapReduce框架使用了一个节点簇（分离的计算机）来并行地完成任务。MapReduce计算包括两个阶段："映射阶段"与"降低阶段"。

在映射阶段，原始数据（如包含了产品评论的文本）的每条记录将由一个程序（常由Java或其他通用语言来书写）映射为零个或多个码值对。一个码可以是一个简单的值（如一个整数或是一个URL），也可以是一个更加复杂的值。例如，给定一个产品的评论，则映射函数将可能为其输出几个码值对。每个码值对中包含了一个由产品与特征组合所构成的码，以及该产品与特征所对应的一个量化值。由于每对输出依赖于单个的输入记录，所以数据输入并创建相应输出对的工作可以分布式地在很多计算机节点上进行，而在不同节点中完成的所有映射结果，将利用框架进行统一收集。

下一过程则是降低阶段。这一阶段将收集具有相同码值的所有记录，并为每个码值产生单个的输出记录。降低阶段也可以是分布式的，基于码值在很多计算机节点上进行。考虑以下实例。

MapReduce的一个实例

将关于网球拍的一套书面评论转换为网球拍特征的量化评分，其输出为网球拍特征的量化均值。

评论R1：X球拍非常灵活，强劲有力，但控制能力中等。

评论R2：Y球拍力量中等，但控制能力极强。

评论R3：Y球拍具有很强的控制能力，但该球拍使用比较费劲，结构不是很灵活。

映射函数的输出结果：

map(R1) -> (<X, flexibility>, 9), (<X, power>, 8), (<X, control>, 5)

map(R2) -> (<Y, power>, 5), (<Y, control>, 10)

map(R3) -> (<Y, control>, 9), (<Y, power>, 3), (<Y, flexibility>, 2)

评论R1的映射函数返回了3个码值对，第1个码值对的码为<X, flexibility>，其值为9；第2个码值对的码为<X, power>，其值为8；第3个码值对的码为<X, control>，其值为5。

注意，不同的评论可由不同的节点处理，其映射函数的输出可由不同的节点创建，如图J-1所示。而各个映射函数的输出结果将会由框架进行统一。

下一步则是为每个独立的码值（球拍、特征）完成降低阶段，这一工作也可以分布式地在各个节点中进行。

降低函数的输出结果：

reduce((<X, flexibility>)) -> (<X, flexibility>, 9)

reduce((<X, power>)) -> (<X, power>, 8)

reduce((<X, control>)) -> (<X, control>, 5)

reduce({<Y, power >}) -> {<Y, power>, 4}

reduce({<Y, control >}) -> {<Y, control>, 9.5}

357 reduce({<Y, flexibility >}) -> {<Y, flexibility>, 2}

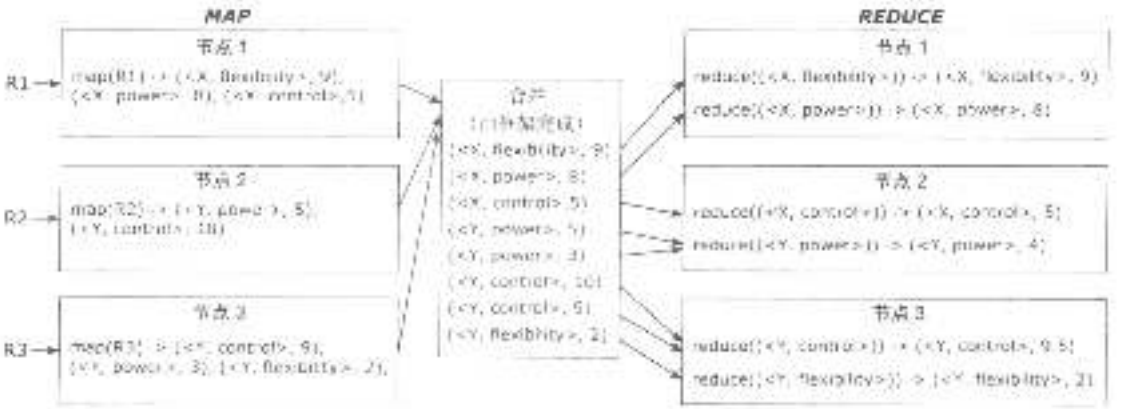


图 J-1 MapReduce 示例

例如，<Y, power> 的降低操作可以在某个节点中进行，该节点将合并映射函数的两个输出 (<Y, power>, 5) 及 (<Y, power>, 3)，并计算量化均值，产生一个降低结果值 (<Y, power>, 4)。同时，<Y, control> 的降低操作则可以在另一个节点中进行，该节点将合并映射函数的两个输出 (<Y, control>, 10) 及 (<Y, control>, 9)，并计算量化均值，产生一个降低结果值 (<Y, control>, 9.5)。这一过程已在图 J-1 中最出说明。

图 J-1 中的映射降低操作的结果值可以以表的方式来显示或存储，如图 J-2 所示。

球拍	特征	值
X	flexibity	6
X	power	8
X	control	5
Y	flexibity	2
Y	power	4
Y	control	9.5

图 J-2

给定映射、降低函数及输入数据集后，MapReduce 框架[⊖]将能确定如何在节点中分布数据，并以某种方式来执行映射及降低操作，从而确保有效的性能。例如，如果所有节点的执行力相同，则任务将进行平均分配。若有某个节点失效，它的工作则将被分配给其他节点。

注意，程序员仅需要书写映射与降低函数，其余工作——如启动计算节点，在计算节点中分布数据，收集映射操作的输出结果，对结果进行排序并将其分布到计算节点中，以及降低操作的执行——都由框架完成。即便映射与降低函数可能相对复杂，但一般来讲，它们往
358 往与上述实例中所展示的一样简单（或更加简单）。在某种意义上，开发工作的相对简易性类似于为处理关系数据而书写一个 SQL 查询。在 RDBMS 中，SQL 查询将不必指定数据表如何进行存取或用那种特定的算法来实现一个连接操作。在 MapReduce 中，程序员将不必

⊖ Apache Hadoop 是 MapReduce 框架中最流行（和开源）的实现。

关注底层细节，如数据分布、节点失效、排序算法等。因此，在这两种情况下，由于分别使用了RDBMS及MapReduce执行器（如Apache Hadoop），从而程序员的任务得到了显著的简化。

正如上图所示，处理这些非结构化的数据将产生一个有序的、易读的数据集，它可以直接报告给用户，或是存储到电子制表软件、数据库、数据集市或数据仓库中。注意，原始数据集（即网球拍书面评论的信息收集）仍然是完整的，并可用于其他类型的数据分析。

大数据技术增强了组织对其所拥有或可访问数据的分析能力。大数据方法（如MapReduce）并不是要替代数据库与数据仓库的相关方法（这些方法主要用于对可以正式建模的数据资产进行管理与利用），而是允许组织机构对那些不适于数据库与数据仓库技术处理的数据做出分析与理解。

359

术 语 表

access privilege（存取权限） 分配给用户账户的权限，用来确定该用户在某一数据库对象上可以进行哪些操作（SELECT，UPDATE，ALTER，DELETE，INSERT）。

accidental misuse（意外误操作） 由于意外而不是恶意导致的对数据库系统中数据条目的误操作。

accuracy（准确性）（数据质量）数据刻画真实实体正确性的程度。

active data warehouse（动态数据仓库） 持续地或批量加载的数据仓库，以确保数据仓库中的数据接近实时更新（以确保能够分析最新数据）。

aggregate function（聚集函数） 用于计算并统计查询结果的 SQL 函数，包括 COUNT，SUM，AVG，MIN 及 MAX。

aggregated data（聚集数据） 表示多个实例汇总的数据。

aggregated fact table（聚集事实表） 每条记录概括了多个事实的事实表。

alias（别名） 一个其他的名字（通常较短），可在查询中的任何地方代替完整的关系名；列也可以有别名，但只限制在查询内部使用。

alpha release（α 版） 面向内部开发团队成员部署的系统，用于对其功能进行初步测试。

ALTER TABLE（SQL） 用于修改已创建的关系结构。

analytical databases（分析型数据库） 包含分析信息的数据库。

analytical information（分析信息） 为支持分析任务而收集和使用的信息，基于的是操作（业务）信息。

application development component（应用开发组件） 向能够应用开发人员提供功能接口，在某些 DBMS 包中，该组件是 DBMS 的一部分；在有些 DBMS 包中，该组件是可选的。

application-oriented（面向应用） 支持为若干业务操作及处理过程提供服务的应用。

assertion（断言） 用于指定用户自定义约束的一种机制。

association rule mining（关联规则挖掘） 找出在事务中倾向于共同出现的项集的数据挖掘方法。

associated entity（关联实体） ER 图中用于描述 M：N 联系的另一种方式。

attribute（of an entity）（属性（实体的）） 实体特征的描述。

attribute（of a relation）（属性（关系的）） 关系的一列。

augmented functional dependency（增广函数依赖） 包含已有函数依赖的函数依赖，它们不能为已有的函数依赖增加更多的信息，因此可以省略。

authentication（认证） 利用用户账户标识符及密码进行登录以确认用户的身份。

authorization matrix（授权矩阵） 行和列分别代表数据库主客体的矩阵（由 DBMS 提供，并可由 DBA 进行操纵），矩阵中的交叉点显示了一个用户对于特定客体所分配到的特定权限。

autonumber data type（自动编号数据类型） 在一个列中自动产生连续数值的数据类型。

AVG 见 aggregate function。

backup（备份） 除了原始副本外，额外存储数据的物理副本。

beta release（β 版） 选择一组用户部署以对系统的可用性进行测试的系统（在 α 版之后）。

big data（大数据） 一个广义的术语，指的是大量不同种类且快速增长的数据，这些数据无法正式地进行建模。

binary relationship（二元联系） ER 图中涉及两个实体的联系，也称为度为 2 的联系。

binary search（二分查找） 利用排好序的列表的搜索方法。

Boyce-Codd normal form，BCNF（BCNF 范式）

如果一个表除具有完全函数依赖外，不再包含其他函数依赖（即仅主码或是候选码能够完全地确定其他列），则该表满足 BCNF 范式。

bridge relation（桥关系） 表示 M：N 联系的关系。

business intelligence (BI) tool（BI 工具） 见 online analytical processing (OLAP) tool。

business rule（业务规则） 一类在最终生成的数据库中指定的用户自定义约束，该约束并没有作为创建 ER 图标准符号的一部分。

candidate key（候选码） 当实体同时拥有多个唯一属性时，每个唯一属性称为一个候选码。

cardinality constraint（基数约束） 表示该实体可以有多少实例与另一实体的实例存在联系的 ER 图关系符号。

catalog（目录） 由 DBMS 创建的数据字典。

CHECK（SQL） 用于指定一个关系表中某一特定列的约束。

checkpoint（检查点） 恢复日志的一部分，当一个检查点被创建时，恢复日志中的所有更新记录，以及内存中各数据更新的结果值，均将被写入数据库的磁盘数据文件中。

circular foreign key dependency（循环外码依赖） 发生在表 A 有外码参照表 B 的主码，且表 B 也有外码参照表 A 的主码时。

class（类） 一个面向对象的概念，将一组分享同种结构与操作的对象。

class hierarchy / specialization hierarchy（类层次 / 特化层次） 包含由“IS-A”关系联系起来的超类和子类的面向对象的层次结构。

class instance（类的实例） 一个特定类的对象。

cloud computing system（云计算系统） 使用因特网实现数据存储与处理服务的系统，这些服务由服务提供商主导。

column（列） 见 attribute（of a relation）。

COMMIT（SQL） 将数据库的所有更新记录到磁盘上的 TCL 命令。

complete mirrored backup（完全镜像备份） 持续地进行更新以保持与源数据库完全一致的单独的完全物理副本。

completeness（完整性） （数据质量）需求数据被描述的完整程度。

composite attribute（复合属性） 由若干属性组合得到的实体属性。

composite primary key（复合主码） 由多个属性组合起来形成的关系的主码。

composite unique attribute（复合的唯一属性） 由若干属性组合得到的实体属性，对于每个实体，该属性的值都不相同。

conceptual database model（概念数据库模型） 可视化数据库需求，独立于特定 DBMS 所基于的逻辑数据库模型。

conformed dimension（一致维度） 开发星形模式之前的标准化维度，通常用于多个事实表。

conformity（统一性） （数据质量）数据符合某特定格式的程度。

consistency（一致性） （数据质量）数据恰当地符合及匹配其他数据的程度。

constellation/galaxy of stars（星座 / 星系） 包含多个事实表的星形模式。

corrective data quality action（数据质量校正措施） 校正数据质量问题的活动。

correlated subquery（相关子查询） 使用外查询 SELECT 部分列出的某关系中某些列的内查询（嵌套查询）。

COUNT 见 aggregate function。

CREATE TABLE（SQL） 用于关系数据库表的创建和连接。

CREATE VIEW（SQL） 用于创建视图。

creating ETL infrastructure（创建 ETL 架构） 为 ETL 创建必要的程序和代码。

creating the data warehouse（创建数据仓库） 使用 DBMS 将数据仓库模型实现为一个真实的数据仓库。

cube（立方体） 多维度数据库存储库，可用于存储维度建模的数据。

data（数据） 已记录或者可得到的事实。

data cleansing/scrubbing（数据清洗 / 清理） 检测并校正低质量的数据。

data control language，DCL（数据控制语言） SQL 语言的一部分，用于数据库维护及管理相关的过程，帮助实现数据的存取控制过程。

data custodian（数据管理者） 负责数据管理及使用中的技术问题，如数据的保护、传输与存储。

data definition component（数据定义组件） 数

据库设计者用于创建数据库结构（如数据库表以及连接已创建数据库表的参照完整性约束）的 DBMS 组件。

data definition language, DDL（数据定义语言） SQL 语言的一部分，用于创建和修改数据库结构。

data dictionary（数据字典） 数据库元数据的存储库。

data duplication（数据重复） 唯一性数据质量问题导致的结果，发生在同一数据收集过程中，一个真实实例被表示了多于一次时。

data governance（数据管理） 一个广义的术语，用于组织机构中的权限定义，以形成正式的规定，确定组织中的数据与元数据在何时、以何种方式、由谁来进行创建、储存、更新（插入、修改、删除）及存档。

data manipulation component（数据操纵组件） 终端用户与其向数据库中插入、读、更新和删除信息的 DBMS 组件。

data manipulation language, DML（数据操纵语言） SQL 语言的一部分，用于操作数据库中的数据，包括对数据库中数据的插入、修改、删除及检索等命令。

data mart（数据集市） 与数据仓库具有相似准则的数据存储形式，但相比数据仓库，其范围更小；数据仓库包含的数据是检索自整个组织机构的操作和数据库中的多主题分析数据，而数据集市通常较小，包含的数据只与一个主题相关，没有必要涉及整个企业范围。

data mining（数据挖掘） 在大量数据中发现新奇而有趣的模式的过程。

data quality（数据质量） 由不同的数据特性定义，包括准确性、唯一性、完整性、一致性、及时性和统一性。

data replication（数据复制） 分布式数据库中在不同的位置存储多个数据副本。

data steward（数据专员） 进行数据管理相关任务的人，负责数据库中数据的正确使用。

data warehouse（数据仓库） 一种有结构的数据存储形式，其存储的数据具有集成性、面向主题、企业范围粒度、历史性以及时变性的特点。数据仓库的目的是用于对分析型数据进行检索。数据仓库可以保存细节型或是汇总型的数据。

data warehouse administration and maintenance（数据仓库管理与维护） 支持终端用户使用数据仓库的相关活动，包括各类技术问题的处理，如保证数据仓库中信息的安全、确保数据仓库中的内容拥有充分的硬盘空间、实现数据的备份与恢复。

data warehouse deployment（数据仓库部署） 将数据仓库及其前端应用交给终端用户使用。

data warehouse modeling（数据仓库建模） 通过 DBMS 软件创建可实现的数据仓库模型，是需求收集、定义以及可视化表示完成之后的第一个步骤。

data warehouse use（数据仓库使用） 用户检索数据仓库中的数据。

database（数据库） 在计算机介质上按结构存储的相关数据，目的是方便用户有效地从数据中获取信息。

database administration and maintenance（数据库管理和维护） 支持终端用户使用数据库的相关活动，包括处理各类技术问题，比如为数据库中的信息提供安全支持、确保足够的硬盘空间来存储数据库的内容、实现数据备份和恢复等过程。

database administration component（数据库管理组件） 用于完成数据库系统技术性、管理性或维护性的任务，如确保系统安全、性能优化或实现系统备份与恢复的 DBMS 组件。

database administrator, DBA（数据库管理员） 负责处理与安全相关的技术问题（如创建、分配用户名和密码、数据库的权限管理等）的人，包括数据库系统的备份和恢复、使用监管、根据需要增加内存空间及其他数据库系统相关技术问题。

database analyst（数据库分析者） 参与需求的收集、定义和可视化阶段的人。

database as a service, DaaS（数据库即服务） SaaS 的一个特例，专门用于云中的数据库管理。

database deployment（数据库部署） 将数据库系统（如数据库及其前端应用）提供给终端用户使用。

database designer/dataset modeler or architect（数据库设计者/数据库建模者或构建者）

参与数据库建模阶段的人。

database developer（数据库开发者） 负责使用DBMS软件完成数据库模型中的数据库实现功能的人。

database fragmentation（数据库分片） 分布式系统中一种基于不同位置上的数据分布策略。

database front-end（数据库前端） 数据库系统的一个组件，提供了一种不需要用户书写命令的访问方式（即问接访问）。

database implementation（数据库实现） 使用DBMS将数据库模型实现为一个真正的数据库。

database management system, DBMS（数据库管理系统） 用于创建数据库，插入、存储、查询、更新和删除数据库中的数据，以及维护数据库的软件。

database metadata（数据库元数据） 数据的数据，不是数据本身的数据库内容（如表名、列名）。

database modeling /logical database modeling（数据库建模 / 逻辑数据库建模） 由DBMS软件执行的数据库模型创建过程，收集、定义和可视化需求之后的第一个步骤。

database policy and standard（数据库的政策与标准） 数据库开发、使用和管理中的政策与标准。所有数据库政策与标准的共同目的是反映并支持数据库的业务流程与业务逻辑。

database requirement（数据库需求） 定义待开发数据库的数据和元数据的细节和约束的语句。数据库需求说明将要记录和以什么方式记录数据。

database system（数据库系统） 基于计算机系统，其目的是在用户和数据库信息之间保证高效的交互。

database use（数据库使用） 终端用户对数据库系统中数据的插入、修改、删除以及查询。

decryption key（解密密钥） 将信息转换至原始状态的算法。

degenerate dimension（退化维度） 事实表中包含的时间标识符（而不是自己有一个单独的维度）。

degree of a relationship（联系的度） 表示有多少个实体参与到该联系中。

delete cascade（级联删除） 一个参照完整性约束选项，如果一条记录的主码值被另一条记录的外码值所参照，则允许删除该记录；所有外码值参照了被删除记录主码值的记录将会被删除。

delete operation（删除操作） 用于从关系中移除数据的数据库操作。

delete restrict（限制删除） 一个参照完整性约束选项，如果一条记录的主码值被另一条记录的外码值所参照，则不允许删除该记录。

delete set-to-default（删除设置为默认值） 一个参照完整性约束选项，如果一条记录的主码值被另一条记录的外码值所参照，则允许删除该记录；所有外码值参照了被删除记录主码值的记录，其外码值被设置为预先定义的默认值。

delete set-to-null（删除设置为空） 一个参照完整性约束选项，如果一条记录的主码值被另一条记录的外码值所参照，则允许删除该记录；所有外码值参照了被删除记录主码值的记录，其外码值被设置为空。

DELETE（SQL） 用于删除存储在数据库关系中的数据。

deletion anomaly（删除异常） 一种更新异常，发生在用户想要删除某一真实世界中的实体数据时，还必须删除另一个真实世界中的实体数据的情况下。

denormalization（逆规范化） 把多个规范化的表连接成一个非规范化的表，得到与规范化相反的效果。

dependent data mart（非独立数据集市） 没有自己的源系统的数据集市，其数据来自数据仓库。

derived attribute（派生属性） 属性值由计算得到且非永久性存于数据库的实体的属性。

designer-added entity/table（设计者添加的实体 / 表） 数据库设计者添加的、非原始需求要求的实体，结果会得到设计者添加的表。

designer-added key（设计者添加的码） 设计者添加的表中的主码，由数据库设计者创建。

designer-created primary key（设计者创建的主码） 数据库设计者向表中添加的、不是原始需求要求的主码。

detailed data（细节数据） 单个数据实例组成的

数据。

detailed fact table（细节事实表） 每条记录代表一个单一事实的事实表。

developing front-end application（开发前端应用） 设计和创建可供终端用户非直接使用数据库的应用。

dimensional table（dimension）（维度表） 维度模型中的表，包含对所要分析的主题属于商业、组织或企业的描述，且包含一个主码和事实表中度量分析所使用的属性。

dimensional modeling（维度建模） 用于面向主题的分析型数据库（如数据仓库或数据集市）设计的一种数据设计方法。

dimensionally modeled data warehouse（维度建模的数据仓库） 使用维度建模的数据仓库。

direct interaction（直接交互） 即终端用户直接与数据库管理系统进行通信的交互操作。

disjointed subclass（非连接子类） 没有普适的实例，共享同一超类实体的子类实体。

DISTINCT（SQL） 结合 SELECT 语句一起使用，用于清除查询结果中的重复值。

distributed database system, DDBS（分布式数据库系统） 分布于不同计算机上的数据库系统。

distribution transparency（分布透明性） 终端用户可采用同种方式来使用分布式数据库，好似数据库并非分布式的 DDBS 特性（即不必知道数据分布的细节，甚至不必知道数据是分布式的）。

domain constraint（域约束） 关系数据库约束，规定在一个关系中，每一列中的取值应该来自预定义的域。

drill down（下钻） 使得查询结果中的数据粒度更精细的 OLAP 操作。

drill up（上卷） 使得查询结果中的数据粒度更粗糙的 OLAP 操作。

DROP TABLE（SQL） 用于从数据库中移除一个关系表。

DROP VIEW（SQL） 用于移除一个视图。

encryption（加密） 使用置乱算法来实现信息的转换，从而使信息对于没有解密秘钥的人变得不可读。

encryption key（加密密钥） 用于使数据对于没有解密秘钥的人不可读的置乱算法。

end user/business user（终端用户/业务用户） 使用数据库系统来辅助处理与工作或者生活相关的任务和事务的人。

enhanced ER, EER（扩展的 ER 模型） 对 ER 符号体系的扩充，可以描述标准 ER 模型之外的一些概念。

enterprise resources planning（ERP）system（企业资源计划系统） 一个预置的、综合性的（多功能的）公司信息系统。

entity（实体） 描述数据库所记录内容的 ER 图组件。

entity instance/entity member（实体实例/实体成员） 实际中现的实体。

entity integrity constraint（实体完整性约束） 各关系数据库规则，所有的主码属性必须有非空取值。

entity-relationship（ER）modeling（实体－关系建模） 一种广泛使用的概念数据库建模方法，用于需求收集过程的构建和组织，并提供了一种用图形方式表示需求的方法。

equivalent functional dependency（等价函数依赖） 发生在互相函数依赖的两列（或两组列）确定其他列时。如果给出了等价函数依赖中的一个，那么另一个等价的函数依赖就不必给出了。

ER diagram, ERD（ER 图） 数据库需求的图形化表示，包括实体和联系。

exact minimum and/or maximum cardinality（最小基数和/或最大基数的确切值） 事先知道的最小基数和/或最大基数。

EXCEPT 见 MINUS。

executive dashboard（管理展示板） 包含一系列有组织的、易读的、描述组织绩效的重要参数的前端应用。

EXISTS（SQL） 用于检查内表查询的结果是否为空。

extensible markup language, XML（可扩展标记语言） 在文档中添加了结构与语义的标记语言。

extensible stylesheet language, XSL（可扩展样式表语言） 用于表示 Web 页面 XML 元素的语言。

extraction-transformation-load, ETL（提取－转换－加载） 从操作型数据源中提取可分析的有用

数据的过程；转换数据使其满足面向主题目标数据仓库的结构，同时确保转换后数据的质量；将转换后具有质量保证的数据加载到目标数据仓库中。

fact table（事实表） 维度模型中表，包含对于给分析主题的度量以及将于实表与维度表关联的外码。

federated database（联合数据库） 一类分布式数据库系统，由连入同一系统的一批现有数据库组成。

field（域） 见 column。

first load（首次加载） 加载的初期，即填充空的数据仓库。

first normal form, 1NF（第一范式） 如果表的每一行都是唯一的并且任何行都没有包含多个值的列，则这个表满足 1NF。

foreign key（外码） 某关系中的一列，这一列又恰好是另外一个（或多个）的关系中的主码。

form（表格） 用于支持终端用户输入和检索数据的数据库前端组件。

fourth normal form, 4NF（第四范式） 一个表若满足 BCNF 且不包含多值依赖，则它满足 4NF。因此，一个关系中若包含多值依赖，则该关系将不满足 4NF。

front-end application（前端应用程序） 一种为用户和数据库管理系统之间进行交互的应用程序。

front-end applications analyst（前端应用程序分析者） 负责为前端应用程序收集和定义需求的人。

front-end applications developer（前端应用程序开发者） 负责创建前端应用程序的人。

full key functional dependency（完全函数依赖） 发生在主码函数确定关系中的其他列，并且主码的任意一部分都不能单独确定同样的列时。

FULL OUTER JOIN 见 OUTER JOIN statement。

fully replicated distributed database（完全复制分布式数据库） 整个数据库均在分布式系统的每个位置点上进行复制的一类复制。

functional dependency（函数依赖） 发生在关系的每条记录中一列（或几列）的值唯一决定同一条记录中另一列的值时。

GRANT（SQL） 用来授予访问权限的语句。

granularity（粒度） 刻画事实表中每一行描述了什么。

GROUP BY（SQL） 可以对表中相关联的数据按组进行聚集操作。

HAVING（SQL） 决定查询中哪些组将在结果中显示，当然也决定了哪些组不在结果中显示。

heterogeneous DDBS（异构型 DDBS） 不同的计算机可以运行不同 DBMS 的 DDBS 架构。

homogeneous DDBS（同构型 DDBS） 所有的计算机运行相同 DBMS 的 DDBS 架构。

horizontal fragmentation（水平分片） 表的记录子集存储于 DDBS 的不同位置的分片。

hybrid online analytical processing, HOLAP（混合型在线分析处理） 结合了 MOLAP 和 ROLAP 的 OLAP 架构。

hypertext markup language, HTML（超文本标记语言） 具有描述如何显示 Web 页面信息功能的标记语言。

identifying relationship（标识性联系） ER 图中弱实体及其主实体之间的联系。

implicit constrain（隐含约束） 为使关系数据库有效而必须满足的隐含的关系数据库模型规则。

IN（SQL） 用于将一个值与一组值进行比较。

independent data mart（独立数据集市） 有自己的源系统和 ETL 架构的单独的数据集市；有一个单独的主题和（与数据仓库相比）更少的数据量、更小的规模、更短的实现时间以及通常更集中的关注点。

index（索引） 用于加快在包含大量记录的关系表中进行数据查询和检索速度的数据库机制。

indirect interaction（间接交互） 终端用户与数据库之间通过前端应用程序进行的交互操作类型。

information（信息） 用户出于某种目的而获取的数据。

infrastructure as a service, IaaS（基础设施即服务） 基于网络提供计算机硬件作为一种标准服务。

inner query（内部查询） 见 nested query。

INSERT INTO（SQL） 用于向创建的关系中填充数据。

插入操作（Insert operation） 用于向关系中输入新数据的数据库操作。

插入异常（insertion anomaly） 一种更新异常，发生在用户需要插入某一真实世界中的实体数据时，还必须输入另一个真实世界中的实体数据的情况下。

insertion, modification, deletion and retrieval（插入、修改、删除和检索） 终端用户用于处理数据库系统所包含数据的操作。

INTERSECT（SQL） 用来合并两个相容的SELECT语句结果的操作符，合并方式是列出所有在这两个SELECT语句的结果中都出现的行。

IS NULL（SQL） 用于包含与记录的列中的空值进行比较的查询。

JOIN（SQL） 用于帮助查询多个表。

LEFT OUTER JOIN（SQL） 见OUTER JOIN statement。

LIKE（SQL） 用于检索部分匹配某一标准的记录。

linear (sequential) search（线性（顺序）查找） 顺序地逐一匹配每一个元素的值，直到所要查找的内容被找到的查找方法。

line-item detailed fact table（条目级细节事实表） 得行表示一个事务中的一个条目行的事实表。

logical database model / implementational database model（逻辑数据库模型/实施型数据库模型） 由DBMS软件实现的数据库模型。

many-to-many relationship（M ∶ N）（多对多联系） ER图中两边的最大基数都为“多个”的联系。

market basket analysis（购物篮分析） 见association rule mining。

markup language（标记语言） 用于在文档中进行文本注释（即“标记”）的语言，这些注释会将各种功能添加到文本中。

massively parallel processing, MPP（大规模并行处理） 用大量独立工作的计算机处理器以并行方式执行单个程序的方法。

master data（主数据） 关键数据已经认证的质量版本，这些数据将为组织机构的信息系统提供一个共同的参考点。

master data management, MDM（主数据管理） 创建和维护一组优质的主数据表，并确保主数据的使用已经入到组织的操作型信息系统中。

master data registry（主数据注册表） 包含一个关键字列表，用于连接及整合操作型系统中实际的主数据；允许信息系统从别的信息系统中访问主数据，用于调整和并立本系统中的主数据。

MAX 见aggregate function。

maximum cardinality one or many（最大基数——一个或多个） 显近实体矩形一端的基数约束部分，表示该实体最多可以有多少实例与另一实体的实例存在联系。

metadata（元数据） 描述数据的结构和数据局部的数据。

MIN 见aggregate function。

minimum cardinality/participation-optional or mandatory（最小基数/参与－可选的或强制的） 远离实体矩形一端的基数约束部分，表示该实体至少要有多少实例与另一实体的实例存在联系。

MINUS（SQL） 用来合并两个相容的SELECT语句结果的操作符，合并方式是列出所有出现在第一个SELECT语句结果中但不出现在另一个SELECT语句结果中的行。

mixed fragmentation（混合分片） 水平分片与垂直分片的结合。

modification anomaly（修改异常） 一种更新异常，发生在用户要修改某一真实值时，同样的修改操作需要重复多次的情况下，是三种类型的更新异常之一。

modify operation（修改操作） 用来改变关系中已有数据数据库操作。

multidimensional database model（多维数据库模型） 用于实现维度建模数据的模型，在这个模型里数据库是数据立方体的集合。

multidimensional online analytical processing, MOLAP（多维在线分析处理） 数据存储在多维立方体中的OLAP架构。

multiuser system（多用户系统） 数据操纵组件在同一时间能直接或间接地由多个用户进行使用的数据库系统。

multivalued attribute（多值属性） 实体实例的同一属性可以有多个不同取值的实体属性。

multivalued dependency（多值依赖） 发生在

同一关系的多个单独的列包含不相关的值，即同一关系表示基数大于1的多个单独联系时。

necessary redundancy（必要的冗余） 指满足3NF的关系表中多次出现的外码值，对于连接数据库中的表来说是必要的。

nested object（嵌套对象） 发生在一个对象包含在另一个对象中时。

nested query（嵌套查询） 包含在其他查询里的查询，也称为内部查询。

nonkey column（非码列） 关系中既不是主码也不是候选码的列。

NoSQL database（NoSQL数据库） 一个广义的术语，描述不基于关系模型且不使用SQL作为查询语言的数据库。

normal form（范式） 表示一个表需要满足的一组特定条件，目的是减少数据冗余，从较低的范式到较高的范式，范式的条件越来越严格，冗余的可能性越来越小。

normalization（规范化） 用于改进包含冗余数据的关系数据库设计的过程，而包含冗余的关系数据库更容易发生更新异常。

normalized data warehouse（规范化数据仓库） 使用ER建模、关系建模和规范化等传统的数据库建模技术对数据仓库进行建模，得到的规范化的表的集合。

NOT（SQL） 与条件比较语句一起使用，当某一条件为FALSE时返回布尔值TRUE，条件为TRUE时返回布尔值FALSE。

object（对象） 面向对象的概念对应于现实世界中的一个客体，这一概念与ER模型中的实体具有同样的意义。

object attribute（对象属性） 描述对象特性的对象元素，等价于实体属性。

object identifier, OID（对象标识） 对象的标识符，其值由OODBS自动分配且不可改变。

object operation（对象操作） 描述对象功能的对象元素。

object-oriented database, OODB（面向对象数据库） 基于面向对象概念的数据库。

object-relational database（面向对象的关系型数据库） 包含面向对象特性的关系型数据库。

OLAP/BI tool（OLAP/BI工具） 见online analytical processing（OLAP）tool。

one-to-many relationship, 1：M（一对多联系） ER图中一侧的最大基数为"多个"，另一侧为"一个"的联系。

one-to-one relationship, 1：1（一对一联系） ER图中两边的最大基数均为"一个"的联系。

online analytical processing, OLAP（在线分析处理） 对数据仓库或数据集市进行以分析为目的查询和数据呈现。

online analytical processing（OLAP）tool（在线分析处理工具） 使终端用户能够对数据仓库进行分析型查询的工具。

online transaction processing, OLTP（在线事务处理） 对数据库中的数据进行更新（即插入、修改及删除）、查询和以操作为目的数据呈现等。

operational database（操作型数据库） 收集支持日常处理中的操作信息的数据库。

operational information/transactional information（操作信息/事务信息） 支撑商业和其他组织日常操作需要所收集和使用的信息。

optional attribute（可选属性） 允许没有取值的实体属性。

ORDER BY（SQL） SELECT查询中用于对查询结果按一列或多列（或表达式）进行排序的子句。

OUTER JOIN（SQL） JOIN操作的变体，包括LEFT、RIGHT和FULL，用于增加在一个（被连接的）关系中不能匹配另一个（被连接的）关系的记录。

outer query（外部查询） 包含嵌套（内部）查询的查询。

overlapping subclass（重叠子类） 有共同实例的子类实体（共享相同的超类实体）。

owner identity（主实体） 其唯一属性可以用来识别弱实体实例的实体。

parallel database（并行数据库） 多个计算机可以同时对同一数据集的不同部分实施同一个任务并完成同样操作的数据库。

partial functional dependency（部分函数依赖） 发生在关系的一列函数依赖于主码的一部分的情况下。

partial key（部分码） 弱实体的一个属性，与主实体的唯一属性结合起来可以唯一标识该弱实体的实例。

partial replication（部分复制） 一种复制类型，对部分数据实施多位置点复制，其余数据不复制。

partial specialization（部分特殊化） 发生在超类实体的一个或多个实例并不是其任何一个子类实体的实例的情况下。

pivot/rotate（旋转） OLAP 操作的一种，通过将维度列表值从一个坐标轴移到另一个坐标轴来重新组织原始查询结果的显示值。

platform as a service, PaaS（平台即服务） 在网络上提供创建应用时所需的各种资源并以计费为标准的服务。

predictive analytic（预测分析） 利用以往的数据预测将来的事件。

preventive data quality action（数据质量预防措施） 为避免数据质量问题而采取的措施。

primary key（主码） 关系的一列（或一组列），其值对于每一行是唯一的；在有多个候选的情况下，设计者从中选择一个作为主码。

primary key constrain（主码约束） 关系数据库规则，要求每一个关系必须有一个主码。

production release（发行版） 实际部署的系统。

query（查询） 数据检索表达式。

query cost（查询代价） 查询执行的时间长度。

query hint（查询提示） 覆盖查询优化器的默认行为的建议，仅用于有经验的数据库管理者的微调过程。

query optimization（查询优化） 对同一查询任务的多种执行方案的检查，以选取最快速的查询方案。

query optimizer（查询优化器） DBMS 的一个特性，可决定如何有效地执行 SQL 语句。

read operation（读操作） 用来从关系中读取数据的数据检索操作。

record（记录） 关系中数据的一行。

recovery（恢复） 在失败后恢复数据库的内容。

recovery log（恢复日志） 将数据库更新记录到磁盘上的文件，确保即使内存中的数据更新操作在写入磁盘前丢失，关于更新的信息仍然能被捕获。

redundant data（冗余数据） 同一数据的多个实例（存储在数据库中）。

reference object（参照对象） 发生在一个对象被另一对象所参照的情况下。

referential integrity constraint（参照完整性约束） 数据库规则，要求关系的每一行包含一个外码，外码的值要么匹配被参照关系中主码列中的一个值，要么为空。

referential integrity constraint line（参照完整性约束连线） 关系图中由外码到相应的主码连接关系的连线。

refresh cycle（刷新周期） 加载新的数据到数据仓库的频率。

refresh load（刷新加载） 首次加载之后的加载。

related multivalued column（相关多值列） 表中表示同一直实世界概念（实体）且每条记录有多个取值的列。

relation（关系） 关系数据库中的表，包含行和列。

relation database（关系数据库） 使用关系数据库模型建模的数据库，是具有唯一名称的相互关联的关系的集合。

relational DBMS, RDBMS（关系数据库管理系统） 关系型 DBMS 软件，基于关系数据库模型用于实现关系数据库。

relation database model（关系数据库模型） 最常用的逻辑数据库模型，用一组相互关联的关系表来表示数据库。

relational online analytical processing, ROLAP（关系型在线分析处理） 用关系表来存储数据的 OLAP 架构。

relational schema（关系模式） 关系数据库的可视化描述。

relation table（关系表） 见 relation。

relationship（联系） 描述实体间如何关联的 ER 图构件，在一个 ER 图内，每个实体必须通过该构件与其他至少一个实体相关联。

relationship attribute（联系属性） 联系的属性，可用于 M：N 联系。

relationship instance（联系实例） 发生在一个实体的实例通过联系与另一个实体的实例相关联的情况下。

relationship role（联系的角色） 在 ER 图中可用来表达额外的语义信息，数据建模者可以自行决定每个实体在联系中的角色。

report（报表） 数据库前端应用组件，其目的是以某种格式化的形式呈现数据以及来自一张或多张表的数据计算结果。

requirements collection, definition, and

visualization（需求收集、定义和可视化）数据库（数据仓库）开发过程中的第一步，也是最关键的步骤：详细说明数据库（数据仓库）系统将要保存的数据类型、保存方式、数据库（数据仓库）系统容量与性能需求。

REVOKE（SQL） 用于回收访问权限的语句。

RIGHT OUTER JOIN 见 OUTER JOIN statement。

role-based access control（基于角色的访问控制） 允许创建包含多个用户的组并为这些组分配访问权限的数据库访问控制方法。

ROLLBACK（SQL） 回滚自最近 COMMIT 后的所有更新的命令。

row（行） 见 Record。

row indicator（行指示符） 提供一个快速标识符的列，用来判断当前记录是否有效。

second normal form，2NF（第二范式） 如果一个表满足 1NF 且不包含部分函数依赖，则这个表满足 2NF。

SELECT（SQL） 使用最频繁的 SQL 语句，用于从数据库关系中检索数据。

self-JOIN（自连接） 连接语句，包含一个参照其本身主码关系的外码，并在查询中为它自己建立连接。

set operator（集合运算） 并运算、交运算和差运算，用于合并两条或者多条能相容的 SELECT 语句所产生的结果。

shared-nothing MPP architecture（无共享 MPP 架构） 所有处理器都有自己的内存与磁盘存储器的并行系统。

single-user system（单用户系统） 数据操纵组件在同一时间只能由一个用户使用的数据库系统。

slice and dice（切片和切块） 从已经显示的结果集增加、替换或者消除指定的维度属性（或消除维度属性中的指定项）的 OLAP 操作。

slowly changing dimension（缓慢变化的维度） 包含可变属性的维度。

snowflake model（雪花模型） 包含规范化维度的星形模式。

software as a service, SaaS（软件即服务） 计算机程序作为一种服务通过互联网进行请求、持有和操作。

source system（源系统） 为数据仓库分析主题提供可分析的有用信息的操作型数据库及其他数据集合。

SQL data type（SQL 数据类型） SQL 中创建表时可用的数据类型。

SQL standard（SQL 标准） 由标准化组织创建并达成一致，明确了 SQL 的逻辑框架和核心要素，以及如何将 SQL 与其他程序语言、外部数据、多媒体等联合使用，还包括其他方面能力的扩展。

SQL/XML 将 SQL 和 XML 结合使用的 SQL 扩展。

star schema（星形模式） 包含事实表和维度的模式。

structured query language, SQL（结构化查询语言） 用来查询数据库、创建数据库、增加、修改和删除数据库构件，以及插入、删除和修改数据库记录的语言。

subclass（子类） 拥有自己的属性和操作并且继承了其超类的属性和操作的类。

subclass entity（子类实体） 有两类属性的实体，分别为其本身的属性和从超类实体继承的属性。所有共享同一超类实体的子类实体共享从超类实体继承的相同的属性，子类实体的实例同时也是其超类实体的实例。

SUM 见 aggregate function。

superclass（超类） 包含其他类（子类）的类。

superclass entity（超类实体） 属性被其子类实体继承的实体。

subject-oriented（面向主题型） 为分析一个或多个具体商业主题领域而创建的。

surrogate key（代理码） 为星形模式的每个维度分配的由系统生成的非复合码。

table（表） 见 relation。

ternary relationship（三元联系） 涉及三个实体的 ER 图联系，也称为度为 3 的联系。

Third normal form，3NF（第三范式） 如果一个表满足第二范式且不包含传递函数依赖，则这个表满足 3NF。

timeliness（及时性，数据质量） 数据与其所表达的真实世界在时间维度上的相符程度。

timestamp（时间戳） 在表中用来说明记录值可用的时间区间的列。

total specialization（完全特殊化） 发生在超类实体的所有实例同时至少拥有一个子类实体的

实例的情况下。

Transaction Control Language, TCL（事务控制语言） SQL 语言的一部分，用于各种与数据库维护及管理相关的过程。TCL 用来管理数据库事务。

transaction identifier（事务标识码） 表示事务 ID 的列。

transaction time（事务时间） 表示事务时间的列。

transaction-level detailed fact table（事务级细节事实表） 每行表示某一事务的事实表。

transitive functional dependency（传递函数依赖） 发生在非码列函数确定关系中的其他非码列的情况下。

trigger（触发器） 使用 SQL 写成的规则，在关系中删除、插入、修改（更新）记录时被激活。

trivial functional dependency（平凡函数依赖） 发生在一个属性（或属性集）函数确定它自己或它的子集的情况下。

tuple（元组） 见 row。

Type1 approach（Type1 方法） 基于重写维度的记录值来处理缓慢变化维度的方法，最常用于由于错误而导致维度变化的情况。

Type2 approach（Type2 方法） 在需要保存历史信息时所使用的处理缓慢变化维度的方法。每当维度中的记录值发生变化时，它将使用新的值为代理码创建一个额外的维度记录。

Type3 approach（Type3 方法） 当维度中每列可能发生改变的数量是确定的，或者只对有限的历史进行记录时，用来处理缓慢变化维度的方法。它为维度表中每个发生改变的列创建一个“历史值”列以及一个“当前值”列。

unary relationship/recursive relationship（一元联系/递归联系） ER 图中只涉及一个实体自身的联系，也称为度为 1 的联系。

unauthorized malicious data update（非授权的恶意数据更新） 在数据录入期间由于恶意的企图而导致的数据库误用。

UNION（SQL） 用来合并两个相容的 SELECT 语句结果的操作符，合并方式是列出所有出自第一个 SELECT 语句结果的行和另一个 SELECT 语句结果的行，如果有两行或多行相同，将只显示其中的一行（从结果中删除重复的行）。

unique attribute（唯一属性） 对于每个实体实例取值都不相同的实体属性。

uniqueness（唯一性）（数据质量）要求每一真实实例在数据收集时只能被描述一次。

update anomaly（更新异常） 由更新操作引起的使关系中包含冗余数据的异常。见 Deletion anomaly，Insertion anomaly 和 Modification anomaly。

update cascade（级联更新） 参照完整性约束选项，如果记录的主码值被另一个关系中记录的外码值参照，则允许该记录的主码值被修改。所有参照被修改记录主码值的记录的外码值也会被修改。

update failure（更新失效） 在更新操作期间的数据库系统失效。

update operation（更新操作） 用于更新关系中数据内容的三个操作，见 delete operation，insert operation 和 modify operation。

update restrict（限制更新） 参照完整性约束选项，如果记录的主码值被另一个关系中记录的外码值参照，则不允许该记录主码值被修改。

update set-to-default（更新设置为默认值） 参照完整性约束选项，如果记录的主码值被另一个关系中记录的外码值参照，则允许该记录的主码值被修改，所有外码值参照被更新记录的主码值的记录，其外码值将会被设置为默认值。

updateset-to-null（更新设置为空） 参照完整性约束选项，如果记录的主码值被另一个关系中记录的外码值参照，则允许该记录的主码值被修改，所有外码值参照被更新记录的主码值的记录，其外码值将会被设置为空。

UPDATE（SQL） 用来修改存储在数据库表中的数据。

user-defined constraint（用户自定义约束） 由数据库设计者添加的数据库约束。

user-defined type, UDT（用户自定义类型） 用户创建的类，在另一个类中作为数据类型使用。

vertical fragmentation（垂直分片） 数据表的列的子集存于 DDBS 不同的位置点中的分片。

view（视图） 允许查询保存在 RDBMS 中的 SQL

机制，生称为虚表。被调用时，视图显示的是已保存的查询结果的表。

view materialization（视图物化） 将视图保存在一个实际的物理表中。

week entity（弱实体） 用于描述本身没有唯一属性的实体的数据库内容。

WHERE（SQL） 确定哪些行应该被返回以及哪些行不应该被返回。

write operations（写操作） 见 update operation。

XML path language, XPath（XML 路径语言） 使用 XML 文档树状表达式的语言，允许用户遍历树。XPath 表达式从树中返回一个元素节点集，这些元素节点能满足表达式所指定的模式。

XQuery 具备 XML 文档查询功能的语言，就像 SQL 提供了对关系表的查询功能一样。

索　引

索引中的页码为英文原书页码，与书中页边标注的页码一致。

A

access privilege（访问权限），305
accidental misuse（意外误操作），307
accuracy（data quality）（准确性（数据质量）），197，199，201s
ACID property（ACID特性），354-355
active data warehouse（动态数据仓库），280
aggregate function（聚集函数），139
aggregated fact table（聚集事实表），243，245-248
alias（别名），151-152
alpha release（α版），293
ALTER TABLE（修改表），153，162-163，171
American National Standards Institute，ANSI（美国国家标准学会），169
analytical database（分析型数据库），10
analytical information（分析型信息），10，207-208，213-214
analytical versus operational information（分析型与操作型信息）
　data audience difference（数据读者差别），210
　data makeup difference（数据组成差别），208
　data orientation difference（数据定位差别），210
　data redundancy difference（数据冗余差别），210
　data time-horizon difference（数据时间范围差别），208
　data time-representation difference（数据时间表示差别），209
　data update difference（数据更新差别），210
　functional difference（功能差别），210
　query amounts/frequency（查询数据总量/频度），209
　technical difference（技术差别），209
Apache Hadoop，358-359
application-oriented（面向应用），210
AS，170
assertion（断言），202-203
association rule mining（关联规则挖掘），343-345
associative entity（关联实体），43-45，52，88
atomicity（ACID property）（原子性（ACID特性）），354
attribute（as a relation concept）（属性（关系中的概念）），57
attribute（as an ER concept）（属性（ER概念））
　composite（复合），22-24，50，60-61，87
　composite unique（复合唯一），23-24，50，87
　defined（定义），14
　derived（派生），25-26，50，87
　in the ER model（在ER模型中），86
　graphical presentation（图形表示），50-51
　mapping composite（映射复合），60-61
　mapping derived（映射派生），73
　mapping multiple unique（映射多个唯一），71-72
　mapping multivalued（映射多值），72-73
　mapping unique composite（映射唯一复合）61
　multiple unique（多个唯一），24
　multivalued（多值），25，50，87
　naming convention（命名约定），33
　optional（选项），25，50，87
　regular（一般），50，87
　relationship（联系），19-20
　unique（唯一），14，50，87
augmented functional dependency（增广函数依赖），100
authentication（认证），305
authorization matrix（授权矩阵），305-306
autonumber（自动编号），84
AVG（求均值），139-140

B

backup（备份），302
BASE property（BASE特性），354-355
beta release（β版），293

big data（大数据）356-359
binary relationship（二元联系），28，30，51，88
binary search（二分查找），188-189
Boyce-Codd normal form，BCNF（Boyce-Codd 范式），319-321
bridge relation（桥关系），66
business intelligence（BI）tool（商务智能（BI）工具），281
business rule（业务规则），81
business user（业务用户），3

C

candidate key（候选码），24，71-72，317-319
cardinality constraint（基数约束），15-17，51
catalog（目录），303
centralized master data（中心化的主数据），327
CHAR，128
CHECK，185-186
checkpoint（检查点），306-307
Chen ER notation（Chen ER 符号体系），13，39
circular foreign key dependences（循环外码依赖），160
class（类），330-331
class hierarchy（类层次），331，333
class instance（类的实例），330
cloud computing（云计算），340-342
column（列），57-58，87
COMMIT（提交），307
complete mirrored backup（完全镜像备份），307
completeness（data quality）（完整性（数据质量）），197-199，201
composite attribute（复合属性），22-24，50，60-61，87
composite candidate key（复合候选码），72
composite primary key（复合主码），61，75
composite unique attribute（复合的非一属性），23-24，50，87
conceptual（database）model（概念（数据库）模型），4-6，8，218-220，222
conformed dimension（一致维度），259
conformity（data quality）（统一性（数据质量）），197-201
consistency（ACID property）（一致性（ACID 特性）），354
consistency（data quality）（一致性（数据质量）），197-201
constellation of stars（星系），242，249
constraint（约束），62，87
constraint management（约束管理），160-163，173
corrective data quality action（数据质量校正措施），200-202
correlated subquery（相关子查询），166
COUNT（计数），139-140
CREATE ASSERTION（创建断言），202-203
CREATE TABLE（创建表），129-131，157-160，166
CREATE TRIGGER（创建触发器），203
CREATE VIEW（创建视图），154-155
cube（立方体），254-255，293-295

D

data（数据），1-2，207
data audience（数据读者），210
data cleansing（数据清洗），279
data control language, DCL（数据控制语言），127-128
data custodian（数据管理者），326
data definition language, DDL（数据定义语言），127-128
data dictionary（数据字典），303-304
data duplication（数据重复），198
data governance（数据管理），326
data manipulation language, DML（数据操纵语言），127-128
data mart（数据集市）
  category（类别），216，217
  data modeling（数据建模），255，263-264
  defined（定义），216
  front-end（BI）application（前端（BI）应用），289
  property（属性），217
data mining（数据挖掘），343
data orientation（数据定位），210
data quality（数据质量），197-202
data redundancy（数据冗余），210
data replication（数据复制），339
data scrubbing（数据清理），279
data steward（数据专员），326

data time-horizon（数据时间范围），208
data warehouse（数据仓库）
  component（组件），214-216
  creating（创建），273-274
  defined（定义），207，212
  definition（定义），212-214
  front-end（BI）application（前端（BI）应用），218-221，289-292
  steps in development（开发步骤），217-222
data warehouse/data mart modeling（数据仓库/数据集市建模）
  defined（定义），220
  dimensionally modeled（维度建模），255，259-260
  independent data mart（独立数据集市），255，263-264
  modeling step（建模步骤），218-220
  normalized approach（规范化方法），255-259
data warehouse deployment（数据仓库部署），218-219，221，292-293
database（数据库），1-4
database administration（数据库管理）
  backup/recovery（备份/恢复），306
  DBMS component（DBMS组件），301-302
  development steps（开发步骤），4-5，7
  monitoring/maintenance task（监测/维护任务），302-304
  new version（新版本），8
  overview（概述），302，305-306
  security（安全），302，305-306
database administrator（DBA）（数据库管理员）
  role（角色），8-9
  task（任务），302-303
database analyst（数据库分析者），8
database architect（数据库架构师），8-9
database as a service, DaaS（数据库即服务），341
database deployment（数据库部署）
  deployment step（部署步骤），4-5，7
  new version（新版本），8
database designer（数据库设计者），8-9
database developer（数据库开发者），8-9
database end user（数据库终端用户），8-9
database fragmentation（数据库分片），337
database implementation（数据库实现），4-8，177
database integrity（数据库完整性），302，307
database management system, DBMS（数据库管理系统）
  application generation component（应用开发组件），301-302
  component（组件），301-302
  data definition component（数据定义组件），301
  data manipulation component（数据操作组件），301-302
  database administration component（数据库管理组件），301-302
  defined（定义），2-3
  relational dbms, RDBMS（关系数据管理系统），10，57
database metadata（数据库元数据），2
database modeler（数据库建模者），8-9
database modeling（数据库建模）
  defined（定义），6
  development step（开发步骤）4-6
  new version（新版本），8
database performance optimization（数据库性能优化），302，307-308
database policies and standards（数据库政策与标准），302，307-308
database requirements（数据库需求）
  defined（定义），13
  development step（开发步骤）4-6
  new version（新版本），8
  visualized as an ER model（可视化为ER模型），15-24，26-32，35-38
database scope（数据库使用范围），8
database system（数据库系统），3-11
database use（数据库使用）
  development step（开发步骤），4-5，7
  new version（新版本），8
DATE，170
decryption key（解密密钥），306
degenerate dimension（退化维度），237
degree of a relationship（联系的度），28-29
DELETE（删除），128，154
DELETE CASCADE（级联删除），179-180
delete restrict（限制删除），178-179
delete set-to-default（删除设置为默认值），183-184
delete set-to-null（删除设置为空），180

deletion anomaly（删除异常），93，95-96
Deletion, delete operation（删除操作），7，91-92
denormalization（逆规范化），116
dependent data mart（非独立数据集市），217
derived attribute（派生属性），25-26，50，87
DESC，138
designer-added entity（设计者添加的实体），117-119
designer-added key（设计者添加的码），117，119
designer-created primary key（设计者创建的主码），84-85
detailed data（细节数据），214
detailed fact table（细节事实表），243-245，248
difference（差），155
dimension table（维度表），225-226
dimensional model（维度模型）
  aggregated fact table（聚集事实表），245-248
  defined（定义），225
  detailed fact table（细节事实表），243-245
  dimension table（维度表），225
  and ER modeling（与 ER 建模），264
  fact table（事实表），225-226
  granularity of the fact table（事实表的粒度），248-249
  line-item fact table（条目级细节事实表），249
  multiple fact tables（多个事实表）240-241
  multiple sources（多个源），231-233
  OLAP/BI tool（OLAP/BI 工具），281
  single source（单个源），226-229
  slowly-changing dimension（缓慢变化的维度），250-253
  snowflake model（雪花模型），254
  table type（表的类型），225-226
  transaction identifier in fact table（事务标识码），235-237
  transaction level fact table（事务级细节事实表），249-250
  transaction time in fact table（事实表中的事务时间），237-239
dimensionally modeled data warehouse（维度建模的数据仓库），259-262
direct interaction（直接交互），3
disjointed subclass（非连接子类）
  partial specialization（部分特殊化），315-316
  total specialization（完全特殊化），312-315
DISTINCT（去重），137-138
distributed database system, DDBS（分布式数据库系统）
  basic description（基本描述），336-337
  data replication（数据复制），339-340
  database fragmentation（数据库分片），337-339
  federated database（联合数据库），340
distribution transparency（分布透明性），336
domain constraint（域约束），79，87
domain key normal form, DKNF（域键范式（DKNF）），322
drill down（下钻），286-287
drill hierarchy（钻层次结构），287
drill up（上卷），286-287
DROP TABLE（删除表），131-132，163
DROP VIEW（删除视图），155
durability（ACID property）（持久性（ACID 特性）），354

E

encryption（加密），306
encryption key（加密密钥），306
end user（终端用户），3
enhanced ER, EER（扩展的 ER 模型）
  defined（定义）40，311
  notation（符号），311
  subclass entity（子类实体），311-316
  superclass entity（超子类实体），311-316
enterprise resources planning（ERP）system（企业资源计划（ERP）系统），323-325
enterprise-wide data（企业范围的数据），213
entity（实体）
  associative（关联），43-45
  attribute（属性），14，50
  defined（定义），13-14
  ERD，13
  mapping into relational model（映射到关系模型），59-60
  naming convention（命名约定），33
entity instance（member）（实体实例（实体成员）），14，18
entity integrity constraint（实体完整性约束），62-63，87

entity-relationship (ER) modeling (实体－关系（ER）建模), 5-6, 13
entity type (实体类型), 14
equivalent functional dependency (等价函数依赖), 100-101
ER modeling (ER 建模)
advantages of (优势), 86
attribute (属性), 14, 22-27
cardinality constraint (基数约束), 15-17
construct (构件), 13
and data warehouse/mart technique (与数据仓库/数据集市技术), 264-265
ER diagram, ERD (ER 图), 21-22, 35-36, 117
exact cardinality (确切的基数), 27-28
mapping ER diagrams into relational schema (ER 图映射到关系模式), 60-78, 83-84
multiple diagrams (多个图), 33-35
multiple relationships (多种联系), 30
naming convention (命名约定), 33
and normalization (与规范化), 116-117
notation (符号体系), 13, 39
relationship role (联系的角色), 29-30
relationship (联系), 15-20, 27-30, 40-49
ER notation (ER 符号体系)
selection of (选择), 13
variation (变体), 39
exact maximum cardinality (最大基数的确切值), 27-28
exact minimum cardinality (最小基数的确切值), 27-28
except (差), 155
executive dashboard (管理展示板), 292
EXISTS statement (存在语句), 166-167
extensible markup language (扩展的标记语言), 见 XML
extensible stylesheet language (可扩展样式语言), 见 XSL
extraction-transformation-load, ETL (提取－转换－加载)
creating infrastructure (创建架构), 218-220
and data warehouse (与数据仓库), 214-216
ETL process (ETL 过程), 275-280

F

fact table (事实表)
additional attribute (额外的属性), 234-235
aggregated data (聚集数据), 243
characteristic (特性), 229-230
detailed data (细节数据), 243
and dimensional modeling (与维度建模), 225-226
granularity of (的粒度), 248-249
line-item (条目级), 249
multiple subjects (多主题), 240
transaction-level (事务级), 249
federated database (联合数据库), 340
field (域), 57
first load (首次加载), 288
first normal form, 1NF (第一范式), 104-107
foreign key (外码)
1 : 1 mapping (一对一映射), 68-69
1 : M mapping (一对多映射), 64-66
defined (定义), 63, 87
M : N relationship mapping (多对多关系映射), 66-68
referential integrity constraint (参照完整性约束), 69, 177
renaming (重命名), 66
SQL syntax (SQL 语法), 131, 170
unary relationships mapping (一元关系映射), 74-76
form (表格), 192-196
fourth normal form, 4NF (第四范式), 321-322
front-end application analyst (前端应用程序分析者), 8-9
front-end application developer (前端应用程序开发者), 8-9
front-end application development (前端应用程序开发)
development step (开发步骤), 4-5, 7
(new version 新版本), 8
front-end application (前端应用程序), 3, 192-197, 214, 288
front-end (BI) application (前端（BI）应用), 289-292
full key functional dependency (完全函数依赖), 101-102, 31
FULL OUTER JOIN (完全外连接), 164-166, 172
fully replicated distributed database (完全复制分布式数据库), 339

functional dependency（函数依赖）
augmented functional dependency（扩展函数依赖），100
candidate key（候选码），317
defined（定义），97
equivalent functional dependency(等价函数依赖)，100-101
full key functional dependency（完全函数依赖），101-102，317
partial functional dependency（部分函数依赖），101-102，107，317-318
transitive functional dependency（传递函数依赖），101-102，108，318-319
trivial functional dependency（平凡函数依赖），99-100
set of（的集合），98-101，103-104
streamlined set（简化后的集合），101

G

galaxy of stars（星系），242
GRANT（授权），305，306
granularity（fact table)(事实表粒度)，248-249
GROUP BY（分组）
aggregate function（聚集函数），140-144
and HAVING（与having），144-145
SQL syntax（SQL 语法），173-174

H

HAVING，144-146
heterogeneous DDBS（异构型 DDBS），336
historical data（历史数据），213
homogeneous DDBS（同构型 DDBS）
horizontal fragmentation（水平分片），337-338
horizontal scalability（水平扩展），354
hybridapproach（MDM)(混合方法)，327-329
hybrid online analytical processing（HOLAP）architecture（混合型在线分析处理结构），297
hypertext markup language, HTML（超文本标记语言），346

I

IBM DB2
RDMS（关系型数据库管理系统），10，57，127
SQL syntax（SQL 语法），171-172
identifier（标识符），14
identifying relationship（标识性联系），30，32
implementational database model（实现型数据库模型），6
implicit constrain（隐含约束），78-79
IN，147
independent data mart（独立数据集市），216-217
indexing（索引），187-192
indirect interaction（间接交互），3-4
information（信息）
analytical（分析型），10，207-208
database system（数据库系统），9
defined（定义），1-2
operational（操作型），10，207
information system（信息系统），3
infrastructure as a service, IaaS（基础设施即服务），341
Inmon，Bill，255
inner query（内部查询），146
INSERT INTO（插入），128，132-133，153，160，162
insertion anomaly（插入异常），93，95-96
insertion（insert operation)(插入操作)，7，91-92
INT，128
integrated data（集成数据），213
interface（接口），7
international organization for standardization, ISO（国际标准化组织），169
intersection（交），155-156
INVTERVAL，170
IS NULL，166，177
isolation（ACID property)(隔离性（ACID 特性)），354
iterative requirement phase（迭代的需求阶段），5-6

J

JOIN（连接），148-153，163-166

K

key attribute（主属性），14
Kimball，Ralph，259

L

LEFT OUTER JOIN（左外连接），164-165，172
level-of-detail（细节层次），208-209
LIKE，139
line-item detailed fact table（条目级细节事实表），249
linear (sequential) search（线性（顺序）查找），187-189，191
logical database modeling（逻辑数据库模型），4-6，57

M

maintenance（维护），302
mandatory participation（强制参与），15，64
many（cardinality）（多（基数）），15-18
many-to-many relationship（M：N）（多对多联系），40-42，44-45，66-68，75
map（映射），57-58
mapping（ER to relational model）（映射（ER图到关系模型））
1：1 relationships（一对一关系），68-69
1：1 unary relationships（一对一一元关系），76
1：M relationships（一对多关系），64-66
associative entity（关联实体），83
attribute（属性），74
composite attribute（复合属性），73
defined（定义），57
derived attribute（派生属性），73
entity（实体），59-60
entity with candidate attribute（带候选码的实体），71-72
ER diagram（ER图），78
M：N relationship（多对多关系），66-68
M：N unary relationship（多对多一元关系），75
multiple relationships（多元关系），76
multivalued attribute（多值属性），72-73
optional attributes（可选参与），62
optional attributes（可选属性），65
optional participation（可选参与），65
rule（规则），87-88
ternary relationship（三元关系），84
unary relationship（一元关系），74-75
unique composite attribute（唯一复合属性），61
weak entity（弱实体），77-78
MapReduce，356-359
market basket analysis（购物篮分析），343
markup language（标记语言）
defined（定义），346
HTML，346
XML，346-347
massively parallel processing, MPP（大规模并行处理），340
master data（主数据），326
master data management, MDM（主数据管理）
architecture（架构），327-329
data governance（数据管理），326-327
master data registry（主数据注册表），327-328
max（求最大值），139-140
maximum cardinality（最大基数），15，17
metadata（元数据），2
Microsoft access，57，185
Microsoft SQL server
RDBMS software（关系型数据库管理系统软件），10，57，127
SQL syntax（SQL语法），170-173
min（求最小值），139-140
minimum cardinality（最小基数）（参与），15
minus（差），155-157
mixed fragmentation（混合分片），339
modification anomaly（修改异常），93，96-97
modification（modify operation）（修改（修改操作））
monitoring（监测），302
multidimensional database model（多维数据库模型），293-295
multidimensional online analytical process（MOLAP）architecture（多维在线分析处理结构），295-296
multiple instance（多实例），40-42
multiple relationship（多关系）
ERD，30
mapping（映射），76
multiple unique attributes（多个唯一属性），25，31-32，50，87
multiuser system（多用户系统），302
multivalued attribute（多值属性），25，31-32，50，87
MySQL

RDBMS software（关系数据库管理系统软件），10，57，127
SQL syntax（SQL 语法），170-172，174

N

necessary redundancy（必要的冗余），111
nested object（嵌套对象），332
nested query（嵌套查询），146
nonkey column（非码列），104
normalization（规范化）
1NF（first normal form）（第一范式），104-107
2NF（second normal form）（第二范式），104，107-108
3NF（second normal form）（第三范式），104，109
4NF（fourth normal form）（第四范式），321-322
5NF（fifth normal form）（第五范式），322
BCNF（Boyce-Codd normal form）（BC 范式），319-321
candidates key（候选码），317-318
DKNF（domain key normal form）（DK 范式），322
example（例子），108-113，317-322
functional dependency（函数依赖），101-102
normal form（范式），104
relational database（关系数据库），91
normalized data warehouse（规范化的数据仓库）
NoSQL database（NoSQL 数据库），353-355
NOT（非），167
NUMERIC，128

O

object（对象）
object attribute（对象属性），330
object identifier（OID）（对象标识），330
object operation（对象操作），330
object query language, OQL（对象查询语言），334
object-oriented database, OODB（面向对象数据库），330-333
object-oriented query（面向对象查询），334-335
object-relational database, ORDB（对象关系型数据库），335
one（cardinality）（一（基数）），15，17-18，64-66，68-69，75-76
online analytical processing, OLAP（在线分析处理）
ad-hoc direct analysis（特定的直接分析），288-289
category（种类），295-297
defined（定义），280-281
front-end application（终端应用），288-289
online analytical processing（OLAP）tools（在线分析处理工具）
defined（定义），281
drill up/drill down（上卷/下钻），286-287
functionality（功能），282
functionality note（功能概述），288
pivot（rotate）（旋转），285-286
slice and dice（切片和切块），282-285
online transaction processing, OLTP（在线事务处理），280-281
operational database（操作型数据库）
defined（定义），10
modeling construct（建模构件），13-49
operational information（操作信息），10，207
operational versus analytical information（操作型与分析型信息）
data audience difference（数据读者差别），210
data makeup difference（数据构成差别），208
data orientation difference（数据导向差别），210
data redundancy difference（数据冗余差别），210
data time-horizon difference（数据时间范围差别），208
data time-representation difference（数据时间表示差别），209
data update difference（数据更新差别），210
functional differences（功能差别），210
query amounts/frequency（查询总量/频度），209
technical difference（技术差别），209
optional attribute（可选属性），26，50，87
optional participation（可选参与），15-16，65
Oracle
RDBMs（关系数据库管理系统），10，57，127
SQL syntax（SQL 语法），170-173
ORDER BY，138-139
OUTER JOIN（外连接），164
outer query（外部查询），146
overlapping subclass（重叠子类），314

owner identity（主实体），38

P

page axis（页轴），286
parallel database（并行数据库），340
partial functional dependency（部分函数依赖），101-102，107，317-318
partial key（部分码），31
partial replication（部分复制），339
partial specialization（部分特殊化），215
pivot/rotate（旋转），282，285-286
platform as a service, PaaS（平台即服务），341
PostgreSQL
  RDBMS software（关系数据库管理系统软件），10，57，127
  SQL syntax（SQL 语法），176-172
predictive analytic（预测分析），343
preventive data quality action（数据质量预防措施），200
primary key（主码）
  1：1 mapping（一对一映射），64-66
  candidate key（候选码），317
  defined（定义），59
  entity integrity constraint（实体完整性约束），62
  M：N relationship mapping（多对多关系映射），66-68
  and OID（与 OID），330-331
  referential integrity constraint（参照完整性约束），69，177
  SQL syntax（SQL 语法），131
  unary relationships mapping（一元关系映射），74-76
primary key constrain（主码约束），79，87
production release（发行版），293

Q

query（查询）
  OODB（面向对象数据库），334
  relational database（关系数据），334
  SQL，134-152，156-157，163-168
  XML，348
query cost（查询代价），308
query hint（查询提示），308
query optimization（查询优化），308

R

reading（read operation）（读操作），91
record（记录），57
recovery（恢复），302，306-307
recovery log（恢复日志），306-307
recursive relationship（递归关系），28，87
redundant data（冗余数据），91
reference object（参照对象），333
referential integrity constraint（参照完整性约束）
  defined（定义），177-181
  delete operation（删除操作），178-181
  implicit constraint（隐藏约束），79
  rule（规则），69
  SQL code（SQL 代码），184
  update operation（更新操作），178，181-184
referential integrity constraint line（参照完整性约束连线），69
refresh cycle（刷新周期），280
refresh load（刷新加载），280
related multivalued column（相关多值列），105
relation（关系）
  mapping（映射），59-60
  relational database model（关系数据库模型），57
  user-defined constraint（用户自定义约束），78，81-83
relational database（关系型数据库）
  basic concept（基本概念），57
  common type（通用型），37
  constraint（约束），78-79，81，87
  database model（数据库模型），6
  following ER modeling，85-86
  implicit constraint（隐式约束），78-79
  software package（软件包），10
  terminology（术语），57-58
  update operation（更新操作），91
  user-defined constraint（用户定义约束），78,81-83
relational online analytical processing, ROLAP（关系型在线分析处理），296-297
relational schema（关系模式）
  defined（定义），57
  mapping ER diagram（映射 ER 图），78
  mapping relationships（映射关系），63-64
relation table（关系表），57-58

relationship（联系）
attribute（属性），19-20
cardinality constraint（基数约束），15-17
degree（度），28-29
ERD，13，15
identifying（标识性），30
M：N（多对多），17-18，40-42，75
naming convention（命名约定），33
1：M（一对多），17-18，64-65，74
1：1（一对一），17-18，76
ternary（三元），45-49
relationship instance（联系实例）
defined（定义），8-19
multiple（多），40-42
relationship role（联系的角色），29-30
relationship type（联系的类型），15
repeating group（重复的组），105
requirement（需求）
data warehouse requirements collection, definition, visualization（数据仓库需求的收集，定义和可视化），218-220
database requirements collection, definition, visualization（数据库需求收集，定义和可视化），4-6，8
retrieval（检索），7
REVOKE（撤销权限），305-306
RIGHT OUTER JOIN（右外连接），164-165，172
role-based access control system（基于角色的访问控制系统），306
ROLLBACK（回滚），307
rotate (pivot)（旋转），282
row（行），57，58
row indicator（行指示符），252

S

scrubbing（清理），279
second normal form, 2NF（第二范式），104，107-108
security（安全），302，305
SELECT（选择），128，134-152，154，156-157，163-168
self-JOIN（自连接），163-164
set operator（集合运算），172
shared-nothing MPP architecture（无共享MPP架构），340
single-user system（单用户系统），302
slice and dice（切片和切块），282-285
slowly changing dimension（缓慢变化的维度）
type1 approach（类型1方法），250-251
type2 approach（类型2方法），251-252
type3 approach（类型3方法），252-253
snowflake model（雪花模型），254
software as a service, SaaS（软件即服务），341
source system（源系统），214-216
specialization hierarchy（特化层次），331
spreadsheets（电子制表软件），288
SQL 见 SQL
SQL/XML，349-350
star schema（星形模式）
additional attribute（额外属性），234-235
characteristic（特性），229-230
dimensional modeling（维度建模），226-229，231
slowly changing dimension（缓慢变化的维度），250-253
structured query language, SQL（结构化查询语言），7，127-173
structured repository（有结构的数据存储），213
subclass（子类）
category（种类），315
EER，311-316
OODB concept（OODB概念），331
subject oriented（面向主题的），210，213
SUM（求和），139-140，146
summarized data（汇总数据），214
superclass（超类）
category（种类），315-316
EER，311-316
surrogate key（代理码），230

T

table（表）
designer-added（设计者添加的），117-119
relational database model（关系数据模型），57-58
Teradata
relational DBMS software（关系数据库管理系统软件），10，57，127
SQL syntax（SQL语法），170-172
ternary relationship（三元关系）
defined（定义），45-49

graphical presentation（图形表示），52
mapping（映射），84
schema mapping rule（模式映射规则），88
third normal form, 3NF（第三范式），104，109
TIME data type（TIME 数据类型），170
time variant data（时变性数据），213
timeliness（data quality）（及时性（数据质量）），197-198，200
timestamp（时间戳），252-253
TIMESTAMP data type（TIMESTAMP 数据类型），170
total specialization（完全特殊化），312-315
transaction control language, TCL（事务控制语言），127-128
transaction identifier（事务标识码），235-237，239
transaction-level detailed fact table（事务级细节事实表），249-250
transaction time（事务时间），235
transactional information（事务信息），10，207
transitive functional dependency（传递函数依赖），101-102，108，318-319
trigger（触发器），203
trivial functional dependency（平凡函数依赖），99-100
tuple（元组），57

U

unary relationship（一元联系）
defined（定义），28-30，44
graphical presentation（图形表示），51
mapping（映射），74-75
mapping rule（映射规则），88
unauthorized malicious data update（非授权的恶意数据更新），307
UNDER，332
unified modeling language（UML）notation（统一建模语言（UML）符号体系），39
union（并），155-156
unique attribute（唯一属性）
defined（定义），14
graphical presentation（图形表示），50
relational database（关系数据库），87
unique composite attribute（唯一复合属性），61
uniqueness（data quality）（唯一性（数据质量）），197-198，200
UPDATE（更新），128，153-154，160-162
update anomalies（更新异常），91，93-96
UPDATE CASCADE（级联更新），182
update failure（更新失效），307
update operation（s）（更新操作）
defined（定义），91，93，177
delete operation（删除操作），92
insert operation（插入操作），92
modify operation（修改操作），92-93
type（类型），91
UPDATE RESTRICT（更新限制），181-182
UPDATE SET-TO-DEFAULT（更新设置为默认值），183
UPDATE SET-TO-NULL（更新设置为空），182-183
user-defined constraint（用户自定义约束）
assertion（断言），202-203
implementation（实现），185-192
relational database（关系数据库），78，81-83
user-defined type, UDT（用户自定义类型），331，332

V

VARCHAR，128
vertical fragmentation（垂直分片），338
view（视图），154
view materialization（视图物化），303

W

week entity（弱实体），30-33，42，52，77-78，88
Web page（Web 页面），197
WHERE，136-137，142-143
write operation（写操作），91

X

XML
document（文档），347
element（元素），347
functionality（功能），347-352
query（查询），348-352
SQL/XML，349-350
XPath（XML 路径语言），348
XSL，350-352
XQuery，349