面向21世纪课程教材

普通高等教育“九五”
国家级重点教材

面向21世纪课程
Textbook Series for 21st C

数据库系统概论

（第三版）

萨师煊　王　珊

高等教育出版社
HIGHER EDUCATION PRESS

(京) 112 号

内容提要

本书是教育部“高等教育面向 21 世纪教学内容和课程体系改革计划”的研究成果，是面向 21 世纪课程教材和教育部高等学校计算机科学与技术学科“九五”规划教材。

本书是在第二版基础上修订而成的，与第二版相比较，在整体结构上进行了适当的调整，增加了数据库新技术方面的内容。全书内容包括：数据库模型、数据库系统结构、关系数据库系统、SQL 语言、复杂数据理论、数据库维护、数据库设计、关系数据库管理系统、数据库技术新进展、面向对象数据库系统、分布式数据库系统、并行数据库系统等。本书曾获国家优秀教材奖，并且是教育部“九五”重点教材。

本书可作为高等学校计算机有关专业的数据库课程教材，也可供从事计算机软件工作的科技人员和工程技术人员以及其他有关部门人员参阅。

图书在版编目(CIP)数据

数据库系统概论/萨师煊,王珊编著.—3 版.—北京:高等教育出版社,2000.2

ISBN 7-04-007494-X

Ⅰ.数… Ⅱ.①萨… ②王… 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 (2000) 第 01158 号

数据库系统概论(第三版)
萨师煊 王珊

出版发行	高等教育出版社		
社　　址	北京市东城区沙滩后街 55 号	邮政编码	100009
电　　话	010-64054588	传　　真	010-64014048
网　　址	http://www.hep.edu.cn		
经　　销	新华书店北京发行所		
印　　刷	中国科学院印刷厂		
纸张供应	山东高唐纸业集团总公司		
开　　本	787×960　1/16	版　　次	1984 年 3 月第 1 版 2000 年 2 月第 3 版
印　　张	30	印　　次	2000 年 2 月第 1 次印刷
字　　数	540 000	定　　价	28.00 元

凡购买高等教育出版社图书,如有缺页、倒页、脱页等质量问题,请在所购图书销售部门联系调换。

作者简介

萨师煊教授，我国最早从事数据库教学和科研的专家之一。是中国数据库学科的奠基人。在国内外发表许多论文和著作，历任中国人民大学信息系主任、中国计算机学会常务理事，数据库学组组长等职务。

萨师煊教授是我国数据库学术活动的倡导者和领导者，对我国数据库技术的发展、应用和学术交流起了很大的推动作用，对我国数据库技术跟踪国际前沿，缩短我国数据库技术与国际差距，做出了很大贡献。

王珊教授，中国人民大学信息学院院长，中科院计算所兼职研究员，博士生导师。中国计算机学会常务理事，中国计算机学会数据库专委会主任。

自 1981 年以来一直从事数据库方向的教学、科研和开发工作。1984~1986 年在美国马里兰大学工作，设计和开发可扩展的关系数据库管理系统 XDB。主持和承担了七五、八五、九五国家科技攻关项目、863 高科技计划项目、国家自然科学基金项目等 20 多项；在国内外杂志发表论文百余篇，有数据库方面的著作十余部。获多项国家级和省部级科技进步奖以及全国优秀教师和全国五一劳动奖章等荣誉称号。

第三版前言

数据库技术从 20 世纪 60 年代中期产生到今天仅仅 30 多年的历史，已经历了三代演变，造就了 C.W.Bachman、E.F.Codd 和 James Gray 三位图灵奖获得者；发展了以数据建模和 DBMS 核心技术为主，内容丰富的一门学科；带动了一个巨大的软件产业 DBMS 产品及其相关工具和解决方案。30 多年成就辉煌。

数据库技术是计算机科学技术中发展最快的领域之一，也是应用最广的技术之一，它已成为计算机信息系统与应用系统的核心技术和重要基础。本书系统地阐述数据库系统的理论、技术和方法，是《数据库系统概论》的第三版。第一版 1983 年出版(1987 年获国家级优秀教材奖)，第二版 1991 年出版。

针对数据库技术的发展和我国应用水平的提高，我们对第二版做了较大的调整、修改和增删，但原书的基本宗旨和风格不变，保持讲述数据库的基本概念、基本理论和基本技术为主的特点。第三版主要的修改是：

1. 根据我国实际情况，网状、层次数据库系统已很少使用，因此把它们删去了，有关的主要概念放在第一章数据模型中介绍。

2. 进一步加强了关系数据库系统的讲解，特别是 SQL 语言的介绍，以适应当前广泛使用关系数据库系统的需要。

3. 把第二版第八章“数据库保护”中的安全性，完整性，并发控制和恢复四节扩展为本书“系统篇”中的四章，内容作了适度的加宽和加深。随着大型数据库系统的普遍使用，这些知识和技术是运行和维护数据库系统必不可少的。本书从使用和管理的角度讲解这些知识而不是讨论实现这些功能的内部技术。

4. 为了反映数据库技术的发展，增加了新技术篇。第十二章全面介绍了数据库发展的总体轮廓，从数据模型、新技术内容、应用领域三个方面，阐述了新一代数据库系统及其相互关系。并选择当前较重要的新技术在后面三章中介绍，它们是面向对象数据库系统，分布式数据库系统和并行数据库系统。

5. 每章后配有小结，习题及阅读参考文献，并对许多文献作了简要的注释，以便读者进一步参考。

此外，为了辅助教学和加强上机实习，人大金仓信息技术有限公司开发了与本书配套的 CAI 软件和 Easybase 数据库系统（个人版、网络版和企业版），

有兴趣的读者和单位可以向金仓公司购买（电话 62515575）。

全书分为四篇共十五章。第一章至第十章，是计算机软件专业本科生的基本教程(书中有*号的部分除外)，第十一章至第十五章是高级教程。

下图给出了本书各章之间的联系和读者对象示意。

本书内容丰富，讲授时可根据学生及专业情况酌情取舍。例如，对于计算机专业本科学生，第二章 2.5 节“关系演算”，第五章 5.4 节“模式分解”可适当压缩，新技术篇中的章节，教师可以选择部分内容进行讲解。

本书由王珊教授执笔，萨师煊教授审定，陈红副教授和研究生曾会萍、王静等参与了内容讨论和书稿校阅工作。

中国科技大学研究生院邵佩英教授和北京大学杨冬青教授详细审阅了全稿并提出了许多有益的意见，在此向他们表示衷心的感谢。

我们在编写本书的过程中，努力跟踪数据库学科的新发展、新技术，把它们纳入到教材中来，力求反映当代新技术，以保持本书的先进性和实用性。但由于学识浅陋，见闻不广，必有许多不足之处，希学术同仁不吝赐教。

萨师煊 王珊

1999 年仲夏于中国人民大学

第二版前言

《数据库系统概论》第一版出版于 1983 年，距今已 6 年。在这 6 年中，不仅数据库技术有了很大进展，而且国内计算机专业的学生和技术人员的水平也有显著提高。因此在第二版中，我们针对这些情况对原书从结构到内容做了较大的调整、修改和增删；但原书的基本宗旨和风格不变，仍以国家教育委员会颁布的《数据库系统概论教学大纲》作为本书编写的基本依据，保持讲述数据库的基本概念、基本理论和基本技术为主的特点。

全书分为两大部分，共十一章。第一部分，第一至九章，是计算机软件专业本科生的基本教程。第二部分，第十、十一章是高级教程。

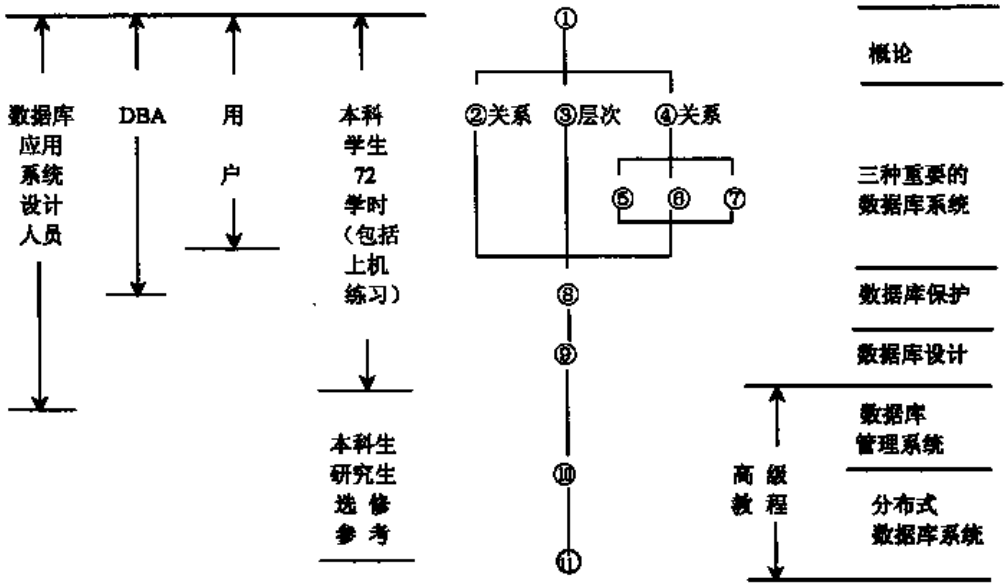
第一章绪论，概述了数据管理的进展、数据模型和数据库系统构成的一般概念。第二至七章介绍三种重要的数据库系统。它们是，网状数据库 DBTG 系统、层次数据库 IMS 和关系数据库系统。鉴于关系数据库具有许多优点并已在应用中日趋成熟，我们把重点放在这一部分，进一步充实了关系数据库的内容，共计有四章（第四至七章）。第四章概述关系模型的基本概念、关系代数和关系演算；第五章详细介绍关系数据库标准语言 SQL；第六章讨论关系系统及其查询优化；第七章讲述关系数据理论。

第八章“数据库保护”（包括数据库的安全性、完整性、并发控制和恢复）和第九章“数据库设计”都作了较大的变动，充实了内容，增强了实用性。

第二部分是新增的。包括第十章“数据库管理系统”和第十一章“分布式数据库系统”。这是为了加强读者对 DBMS 的了解，适应对“分布处理”日益普遍的需要，引导读者从学习本书开始向某些数据库的重要新领域过渡。

第一版中有附录 I “文件组织”与附录 II “IMAGE 数据库系统”。前者是基础性教材，由于读者水平的提高已无存在的必要；后者原作为实习课的参考教材，鉴于近几年引进多种商品化的关系数据库系统且应用甚为广泛，因此我们已另行编写了一本《实用数据库系统汇编》作为实习教材（高等教育出版社出版，1990 年出版），其中介绍了五种关系系统和 IMAGE 系统。因此本书删去了第一版中的附录。

下图给出了本书各章之间的联系和读者对象示意。



本书由王珊执笔，萨师煊审定，杜小勇、李曦老师和研究生唐元昌、武志文、任永杰等参与了内容讨论和校阅工作。

中国科技大学研究生院罗晓沛教授、清华大学俞盘祥副教授和北京大学杨冬青副教授详细审阅了全稿并提出了许多有益的意见，在此向他们致以衷心的感谢。

我们在改编过程中，尽可能引入新的观点和方法，力求能反映当代技术水平。但由于学识浅陋，见闻不广，必有许多不足之处，希同行指正。

萨师煊　王珊

1989 年 12 月

于中国人民大学数据工程与知识工程研究所

第一版前言

数据库是数据管理的最新技术，是计算机科学的重要分支。十余年来，数据库管理系统已从专用的应用程序包发展成为通用的系统软件。由于数据库具有数据结构化、最低冗余度、较高的程序与数据独立性、易于扩充、易于编制应用程序等优点，较大的信息系统都是建立在数据库设计之上的。因此，不仅大型计算机及中小型计算机，甚至微型机都配有数据库管理系统。目前，对数据库各种模型的研究以及理论上的探讨都还在蓬勃开展，其应用也从一般管理扩大到计算机辅助设计、人工智能以及科技计算等领域。国外高等学校计算机科学系、计算机应用与信息系统系等都开设有数据库系统方面的课程。今年来，我国在计算机科学教育中也对数据库予以应有的重视。1983 年教育部部属高等学校计算机软件专业教学方案将数据库概论列为四年制本科的必修课程，并已于 1983 年 6 月通过了教学大纲，我们这本书便是按照该大纲编写的一本教材。

全书共分十章和两个附录。附录 I 的“文件组织”系学习数据库的预备知识，但已散见于数据结构等先行课中，列在这里，仅供参阅用，一般不必作为正式讲授内容。第一、二、三章为有关数据库的一般概念；第四至八章介绍三种重要数据库模型。其中第八章的公理系统与模式分解的大部分内容已超出大纲范围，讲授时可酌情取舍。第九章为数据库控制；第十章为数据库设计。附录 II 的“IMAGE3000 数据库系统”介绍了如何在 HP-3000 计算机的 IMAGE3000 数据库管理系统上建立数据库和使用数据库。对于能够用到 HP-3000 计算机的单位，这一部分可作为实习教材；对于不能用上 HP-3000 计算机的单位则可作为参考，借以从中了解如何在一个实际的数据库管理系统上建立一个数据库应用系统。

本书是编者根据多年来进行数据库讲演及在高等学校为研究生、本科生开设数据库系统概论课程的经验，并在多次编写的内部交流讲义的基础上修改而成的，由于我们水平不高，而数据库作为计算机科学中的一个新兴分支发展又非常迅速，因此，在本书中存在的问题一定不少，希望读者提出批评意见

中国科技大学研究生院罗晓沛同志、清华大学计算机科学与工程系杨德元同志、余盘祥同志与沈金发同志详细阅读了全稿，并提出许多有益的意见，在此谨向他们致以衷心的感谢。

作为内部交流讲义阶段，有 50 余所兄弟院校采用作为教本，其中不少同

志在使用以后曾以口头或书面形式提出意见，我们也对他们表示感谢。

在讲稿整理过程中，我校信息管理系的刘怡同志、前 78 级、79 级研究生吴鹏琦等同志，82 级研究生张秉训、刘伶、杨明伟等同志以及资料室的王小平等同志，都从不同方面分别做了一些工作，对此作者一并表示诚挚的谢意。

萨师煊　王珊

1983 年 4 月于中国人民大学

目　录

第一篇　基础篇

第二篇　设计篇

第三篇　系统篇

第四篇　新技术篇

第一篇　基　础　篇

第一章 绪 论

数据库是数据管理的最新技术，是计算机科学的重要分支。今天，信息资源已成为各个部门的重要财富和资源。建立一个满足各级部门信息处理要求的行之有效的信息系统也成为一个企业或组织生存和发展的重要条件。因此，作为信息系统核心和基础的数据库技术得到越来越广泛的应用，从小型单项事务处理系统到大型信息系统，从联机事务处理到联机分析处理，从一般企业管理到计算机辅助设计与制造(CAD/CAM)、计算机集成制造系统(CIMS)、办公信息系统(OIS)、地理信息系统(GIS)等，越来越多新的应用领域采用数据库存储和处理他们的信息资源。对于一个国家来说，数据库的建设规模、数据库信息量的大小和使用频度已成为衡量这个国家信息化程度的重要标志。因此，数据库课程是计算机科学与技术专业、信息管理专业的重要课程。

1.1 数据库系统概述

在系统地介绍数据库的基本概念之前，这里首先介绍一些数据库最常用的术语和基本概念。

1.1.1 数据、数据库、数据库管理系统、数据库系统

数据、数据库、数据库管理系统和数据库系统是与数据库技术密切相关的四个基本概念。

一、数据（Data）

数据是数据库中存储的基本对象。数据在大多数人头脑中的第一个反应就是数字。其实数字只是最简单的一种数据，是数据的一种传统和狭义的理解。广义的理解，数据的种类很多，文字、图形、图像、声音、学生的档案记录、货物的运输情况等，这些都是数据。

可以对数据做如下定义：描述事物的符号记录称为数据。描述事物的符号可以是数字，也可以是文字、图形、图像、声音、语言等，数据有多种表现形

式，它们都可以经过数字化后存入计算机。

为了了解世界，交流信息，人们需要描述这些事物。在日常生活中直接用自然语言（如汉语）描述。在计算机中，为了存储和处理这些事物，就要抽出对这些事物感兴趣的特征组成一个记录来描述。例如：在学生档案中，如果人们最感兴趣的是学生的姓名、性别、年龄、出生年月、籍贯、所在系别、入学时间，那么可以这样描述：

（李明，男，21，1972，江苏，计算机系，1990）

因此这里的学生记录就是数据。对于上面这条学生记录，了解其含义的人会得到如下信息：李明是个大学生，1972 年出生，男，江苏人，1990 年考入计算机系；而不了解其语义的人则无法理解其含义。可见，数据的形式还不能完全表达其内容，需要经过解释。所以数据和关于数据的解释是不可分的，数据的解释是指对数据含义的说明，数据的含义称为数据的语义，数据与其语义是不可分的。

二、数据库（DataBase，简称 DB）

数据库，顾名思义，是存放数据的仓库。只不过这个仓库是在计算机存储设备上，而且数据是按一定的格式存放的。

人们收集并抽取出一个应用所需要的大量数据之后，应将其保存起来以供进一步加工处理，进一步抽取有用信息。在科学技术飞速发展的今天，人们的视野越来越广，数据量急剧增加。过去人们把数据存放在文件柜里，现在人们借助计算机和数据库技术科学地保存和管理大量的复杂的数据，以便能方便而充分地利用这些宝贵的信息资源。

所谓数据库是长期储存在计算机内的、有组织的、可共享的数据集合。数据库中的数据按一定的数据模型组织、描述和储存，具有较小的冗余度、较高的数据独立性和易扩展性，并可为各种用户共享。

三、数据库管理系统（DataBase Management System，简称 DBMS）

了解了数据和数据库的概念，下一个问题就是如何科学地组织和存储数据，如何高效地获取和维护数据。完成这个任务的是一个系统软件——数据库管理系统。

数据库管理系统是位于用户与操作系统之间的一层数据管理软件。它的主要功能包括以下几个方面：

1. 数据定义功能

DBMS 提供数据定义语言(Data Definition Language，简称 DDL)，用户通过它可以方便地定义对数据库中的数据对象进行定义。

2. 数据操纵功能

DBMS 还提供数据操纵语言(Data Manipulation Language，简称 DML)，用户可以使用 DML 操纵数据实现对数据库的基本操作，如查询、插入、删除和修改等。

3. 数据库的运行管理

数据库在建立、运用和维护时由数据库管理系统统一管理、统一控制，以保证数据的安全性、完整性、多用户对数据的并发使用及发生故障后的系统恢复。

4. 数据库的建立和维护功能

它包括数据库初始数据的输入、转换功能，数据库的转储、恢复功能，数据库的重组织功能和性能监视、分析功能等。 这些功能通常是由一些实用程序完成的。

数据库管理系统是数据库系统的一个重要组成部分。

四、数据库系统（DataBase System，简称 DBS）

数据库系统是指在计算机系统中引入数据库后的系统，一般由数据库、数据库管理系统（及其开发工具）、应用系统、数据库管理员和用户构成。应当指出的是，数据库的建立、使用和维护等工作只靠一个 DBMS 远远不够，还要有专门的人员来完成，这些人被称为数据库管理员（DataBase Administrator，简称 DBA）。

在一般不引起混淆的情况下常常把数据库系统简称为数据库。

数据库系统可以用图 1.1 表示。数据库系统在整个计算机系统中的地位如图 1.2 所示。

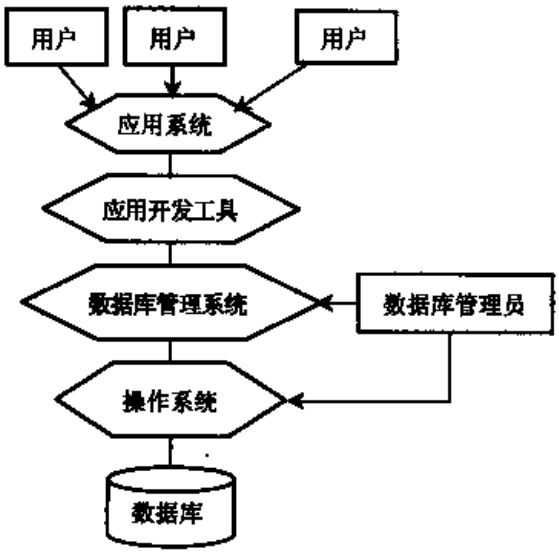


图 1.1 数据库系统

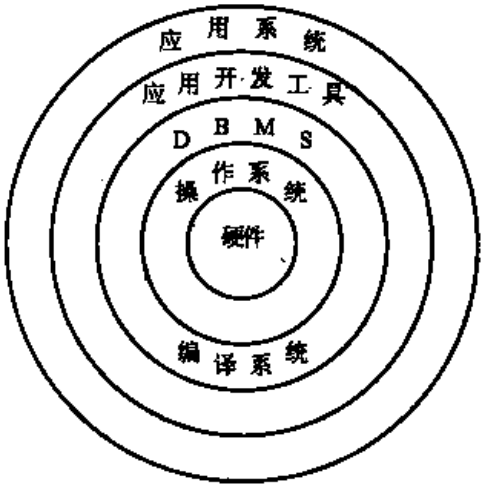


图 1.2 数据库在计算机系统中的地位

1.1.2 数据管理技术的产生和发展

数据库技术是应数据管理任务的需要而产生的。

数据的处理是指对各种数据进行收集、存储、加工和传播的一系列活动的总和。数据管理则是指对数据进行分类、组织、编码、存储、检索和维护，它是数据处理的中心问题。

人们借助计算机进行数据处理是近三十年的事。研制计算机的初衷是利用它进行复杂的科学计算。随着计算机技术的发展，其应用远远地超出了这个范围。在应用需求的推动下，在计算机硬件、软件发展的基础上，数据管理技术经历了人工管理、文件系统、数据库系统三个阶段。这三个阶段的特点及其比较如表 1.1 所示。

表 1.1 数据管理三个阶段的比较

		人工管理阶段	文件系统阶段	数据库系统阶段
背景	应用背景	科学计算	科学计算、管理	大规模管理
	硬件背景	无直接存取存储设备	磁盘、磁鼓	大容量磁盘
	软件背景	没有操作系统	有文件系统	有数据库管理系统
	处理方式	批处理	联机实时处理、批处理	联机实时处理、分布处理、批处理
特点	数据的管理者	用户（程序员）	文件系统	数据库管理系统
	数据面向的对象	某一应用程序	某一应用	现实世界
	数据的共享程度	无共享，冗余度极大	共享性差，冗余度大	共享性高，冗余度小
	数据的独立性	不独立，完全依赖于程序	独立性差	具有高度的物理独立性和一定的逻辑独立性
	数据的结构化	无结构	记录内有结构、整体无结构	整体结构化，用数据模型描述
	数据控制能力	应用程序自己控制	应用程序自己控制	由数据库管理系统提供数据安全性、完整性、并发控制和恢复能力

一、人工管理阶段

20 世纪 50 年代中期以前，计算机主要用于科学计算。当时的硬件状况是，外存只有纸带、卡片、磁带，没有磁盘等直接存取的存储设备；软件状况是，没有操作系统，没有管理数据的软件；数据处理方式是批处理。人工管理数据具有如下特点：

（1）数据不保存

由于当时计算机主要用于科学计算，一般不需要将数据长期保存，只是在计算某一课题时将数据输入，用完就撤走。不仅对用户数据如此处置，对系统软件有时也是这样。

（2）应用程序管理数据

数据需要由应用程序自己管理，没有相应的软件系统负责数据的管理工作。应用程序中不仅要规定数据的逻辑结构，而且要设计物理结构，包括存储结构、存取方法、输入方式等。因此程序员负担很重。

（3）数据不共享

数据是面向应用的，一组数据只能对应一个程序。当多个应用程序涉及某些相同的数据时，由于必须各自定义，无法互相利用、互相参照，因此程序与程序之间有大量的冗余数据。

（4）数据不具有独立性

数据的逻辑结构或物理结构发生变化后，必须对应用程序做相应的修改，这就进一步加重了程序员的负担。

在人工管理阶段，程序与数据之间的一一对应关系可用图 1.3 表示。

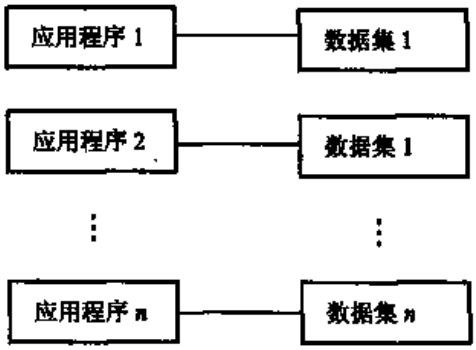


图 1.3 人工管理阶段应用程序与数据之间的对应关系

二、文件系统阶段

20 世纪 50 年代后期到 60 年代中期，这时硬件方面已有了磁盘、磁鼓等直接存取存储设备；软件方面，操作系统中已经有了专门的数据管理软件，一般称为文件系统；处理方式上不仅有了批处理，而且能够联机实时处理。

用文件系统管理数据具有如下特点：

（1）数据可以长期保存

由于计算机大量用于数据处理，数据需要长期保留在外存上反复进行查询、修改、插入和删除等操作。

（2）由文件系统管理数据

由专门的软件即文件系统进行数据管理，文件系统把数据组织成相互独立的数据文件，利用“按文件名访问，按记录进行存取”的管理技术，可以对文件进行修改、插入和删除的操作。文件系统实现了记录内的结构性，但整体无结构。程序和数据之间由文件系统提供存取方法进行转换，使应用程序与数据

之间有了一定的独立性，程序员可以不必过多地考虑物理细节，将精力集中于算法。而且数据在存储上的改变不一定反映在程序上，大大节省了维护程序的工作量。但是，文件系统仍存在以下缺点。

（3）数据共享性差，冗余度大

在文件系统中，一个文件基本上对应于一个应用程序，即文件仍然是面向应用的。当不同的应用程序具有部分相同的数据时，也必须建立各自的文件，而不能共享相同的数据，因此数据的冗余度大，浪费存储空间。同时由于相同数据的重复存储、各自管理，容易造成数据的不一致性，给数据的修改和维护带来了困难。

（4）数据独立性差

文件系统中的文件是为某一特定应用服务的，文件的逻辑结构对该应用程序来说是优化的，因此要想对现有的数据再增加一些新的应用会很困难，系统不容易扩充。一旦数据的逻辑结构改变，必须修改应用程序，修改文件结构的定义。应用程序的改变，例如应用程序改用不同的高级语言等，也将引起文件的数据结构的改变。因此数据与程序之间仍缺乏独立性。可见，文件系统仍然是一个不具有弹性的无结构的数据集合，即文件之间是孤立的，不能反映现实世界事物之间的内存联系。在文件系统阶段，程序与数据之间的关系如图 1.4 所示。

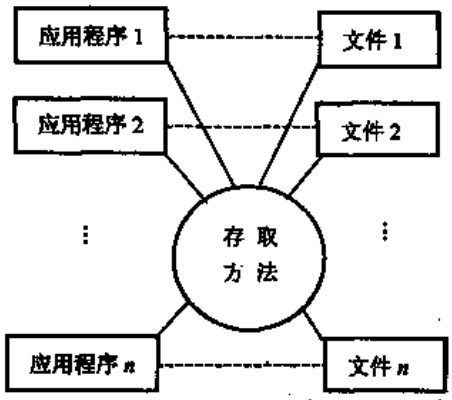


图 1.4 文件系统阶段应用程序与数据之间的对应关系

三、 数据库系统阶段

20 世纪 60 年代后期以来，计算机用于管理的规模越来越大，应用越来越广泛，数据量急剧增长，同时多种应用、多种语言互相覆盖地共享数据集合的要求越来越强烈。

这时硬件已有大容量磁盘，硬件价格下降；软件则价格上升，为编制和维护系统软件及应用程序所需的成本相对增加；在处理方式上，联机实时处理要

求更多，并开始提出和考虑分布处理。在这种背景下，以文件系统作为数据管理手段已经不能满足应用的需求，于是为解决多用户、多应用共享数据的需求，使数据为尽可能多的应用服务，数据库技术便应运而生，出现了统一管理数据的专门软件系统——数据库管理系统。

用数据库系统来管理数据比文件系统具有明显的优点，从文件系统到数据库系统，标志着数据管理技术的飞跃。下面来详细地讨论数据库系统的特点及其带来的优点。

1.1.3 数据库系统的特点

与人工管理和文件系统相比，数据库系统的特点主要有以下几个方面。

一、数据结构化

数据结构化是数据库与文件系统的根本区别。

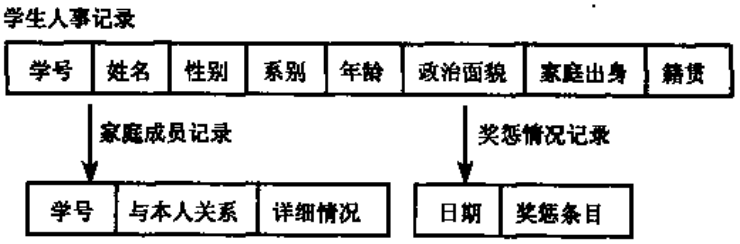
在文件系统中，相互独立的文件的记录内部是有结构的。传统文件的最简单形式是等长同格式的记录集合。例如：一个学生人事记录文件，每个记录都有如图 1.5 的记录格式：

学生人事记录

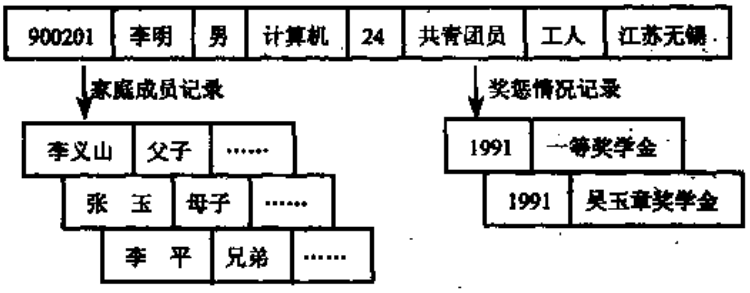
学号	姓名	性别	系别	年龄	政治面貌	家庭出身	籍贯	家庭成员	奖惩情况

图 1.5 学生记录格式示例

其中前八项数据是任何学生必须具有的而且基本上是等长的，而各个学生的后两项数据其信息量大小变化较大。如果采用等长记录形式存储学生数据，为了建立完整的学生档案文件，每个学生记录的长度必须等于信息量最多的记录的长度，因而会浪费大量的存储空间。所以最好是采用变长记录或主记录与详细记录相结合的形式建立文件。也就是将学生人事记录的前八项作为主记录，后两项作为详细记录，则每个记录为如图 1.6(a)所示的记录格式，学生李明的记录如图 1.6(b)所示。



(a) 主记录——详细记录格式示例



(b) 学生李明记录示例

图 1.6 主记录——详细记录

这样可以节省许多存储空间，灵活性也相对提高。

但这样建立的文件还有局限性，因为这种结构上的灵活性只是针对一个应用而言。一个学校或一个组织涉及许多应用，在数据库系统中不仅要考虑某个应用的数据结构，还要考虑整个组织的数据结构。例如一个学校的信息管理系统中不仅要考虑学生的人事管理，还要考虑学籍管理、选课管理，同时还要考虑教员的人事管理、科研管理等应用，可按图 1.7 方式为该校的信息管理系统组织其中的学生数据：

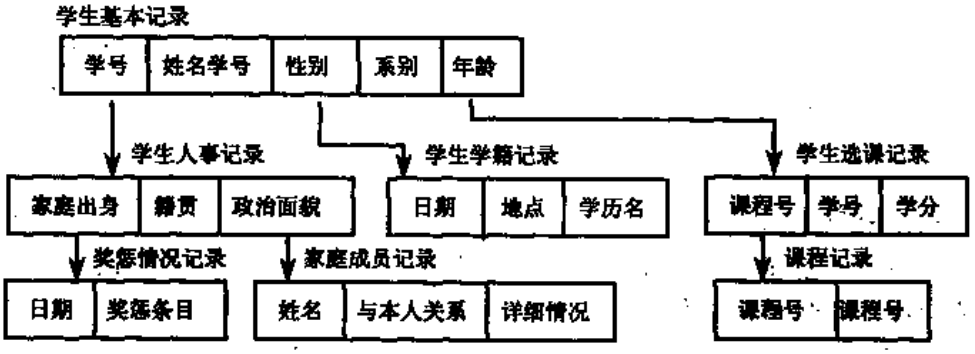


图 1.7 学生数据的组织

这种数据组织方式为各部分的管理提供了必要的记录，使数据结构化了。这就要求在描述数据时不仅要描述数据本身，还要描述数据之间的联系。

在文件系统中，尽管其记录内部已有了某些结构，但记录之间没有联系。

数据库系统实现整体数据的结构化，是数据库的主要特征之一，也是数据库系统与文件系统的本质区别。

在数据库系统中，数据不再针对某一应用，而是面向全组织，具有整体的结构化。不仅数据是结构化的，而且存取数据的方式也很灵活，可以存取数据库中的某一个数据项、一组数据项、一个记录或一组记录。而在文件系统中，数据的最小存取单位是记录，粒度不能细到数据项。

二、数据的共享性高，冗余度低，易扩充

数据库系统从整体角度看待和描述数据，数据不再面向某个应用而是面向整个系统，因此数据可以被多个用户、多个应用共享使用。数据共享可以大大减少数据冗余，节约存储空间。数据共享还能够避免数据之间的不相容性与不一致性。

所谓数据的不一致性是指同一数据不同拷贝的值不一样。采用人工管理或文件系统管理时，由于数据被重复存储，当不同的应用使用和修改不同的拷贝时就很容易造成数据的不一致。在数据库中数据共享，减少了由于数据冗余造成的不一致现象。

由于数据面向整个系统，是有结构的数据，不仅可以被多个应用共享使用，而且容易增加新的应用，这就使得数据库系统弹性大，易于扩充，可以适应各种用户的要求。可以取整体数据的各种子集用于不同的应用系统，当应用需求改变或增加时，只要重新选取不同的子集或加上一部分数据便可以满足新的需求。

三、数据独立性高

数据独立性是数据库领域中一个常用术语，包括数据的物理独立性和数据的逻辑独立性。

物理独立性是指用户的应用程序与存储在磁盘上的数据库中数据是相互独立的。也就是说，数据在磁盘上的数据库中怎样存储是由 DBMS 管理的，用户程序不需要了解，应用程序要处理的只是数据的逻辑结构，这样当数据的物理存储改变了，应用程序不用改变。

逻辑独立性是指用户的应用程序与数据库的逻辑结构是相互独立的，也就是说，数据的逻辑结构改变了，用户程序也可以不变。

数据独立性是由 DBMS 的二级映象功能来保证的，将在下面讨论。

数据与程序的独立，把数据的定义从程序中分离出去，加上数据的存取又由 DBMS 负责，从而简化了应用程序的编制，大大减少了应用程序的维护和修改。

四、数据由 DBMS 统一管理和控制

数据库的共享是并发的（Concurrency）共享，即多个用户可以同时存取数据库中的数据甚至可以同时存取数据库中同一个数据。

为此，DBMS 还必须提供以下几方面的数据控制功能：

（1）数据的安全性（Security）保护

数据的安全性是指保护数据以防止不合法的使用造成的数据的泄密和破坏。使每个用户只能按规定，对某些数据以某些方式进行使用和处理。

（2）数据的完整性（Integrity）检查

数据的完整性指数据的正确性、有效性和相容性。完整性检查将数据控制在有效的范围内，或保证数据之间满足一定的关系。

（3）并发（Concurrency）控制

当多个用户的并发进程同时存取、修改数据库时，可能会发生相互干扰而得到错误的结果或使得数据库的完整性遭到破坏，因此必须对多用户的并发操作加以控制和协调。

（4）数据库恢复（Recovery）

计算机系统的硬件故障、软件故障、操作员的失误以及故意的破坏也会影响数据库中数据的正确性，甚至造成数据库部分或全部数据的丢失。DBMS必须具有将数据库从错误状态恢复到某一已知的正确状态（亦称为完整状态或一致状态）的功能，这就是数据库的恢复功能。

数据库管理阶段应用程序与数据之间的对应关系可用图 1.8 表示。

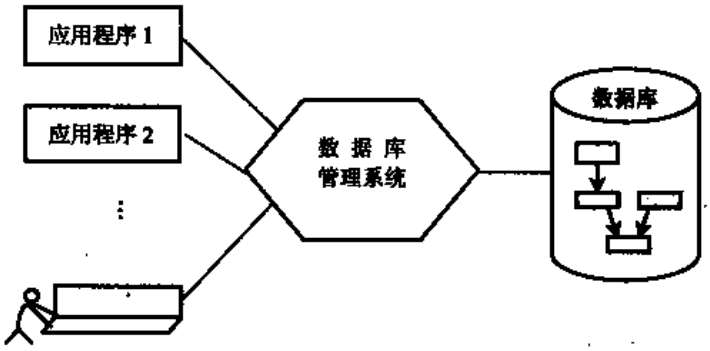


图 1.8 数据库系统阶段应用程序与数据之间的对应关系

综上所述，数据库是长期存储在计算机内有组织的大量的共享的数据集合。它可以供各种用户共享，具有最小冗余度和较高的数据独立性。DBMS在数据库建立、运用和维护时对数据库进行统一控制，以保证数据的完整性、安全性，并在多用户同时使用数据库时进行并发控制，在发生故障后对系统进行恢复。

数据库系统的出现使信息系统从以加工数据的程序为中心转向围绕共享的数据库为中心的新阶段。这样既便于数据的集中管理，又有利于应用程序的研制和维护，提高了数据的利用率和相容性，提高了决策的可靠性。

目前，数据库已经成为现代信息系统的不可分离的重要组成部分。具有数百万甚至数十亿字节信息的数据库已经普遍存在于科学技术、工业、农业、商业、服务业和政府部门的信息系统。20 世纪 80 年代后不仅在大型机上，在多

数微机上也配置了 DBMS，使数据库技术得到更加广泛的应用和普及。

数据库技术是计算机领域中发展最快的技术之一。数据库技术的发展是沿着数据模型的主线展开的。下面讨论数据模型。

1.2 数 据 模 型

模型，特别是具体模型，人们并不陌生。一张地图，一组建筑设计沙盘，一架精致的航模飞机都是具体的模型。一眼望去，就会使人联想到真实生活中的事物。模型是现实世界特征的模拟和抽象。数据模型（Data Model）也是一种模型，它是现实世界数据特征的抽象。

数据库是某个企业、组织或部门所涉及的数据的综合，它不仅要反映数据本身的内容，而且要反映数据之间的联系。由于计算机不可能直接处理现实世界中的具体事物，所以人们必须事先把具体事物转换成计算机能够处理的数据。在数据库中用数据模型这个工具来抽象、表示和处理现实世界中的数据和信息。通俗地讲数据模型就是现实世界的模拟。

现有的数据库系统均是基于某种数据模型的。因此，了解数据模型的基本概念是学习数据库的基础。

数据模型应满足三方面要求：一是能比较真实地模拟现实世界；二是容易为人所理解；三是便于在计算机上实现。一种数据模型要很好地满足这三方面的要求在目前尚很困难。在数据库系统中针对不同的使用对象和应用目的，采用不同的数据模型。

不同的数据模型实际上是提供给我们模型化数据和信息的不同工具。根据模型应用的不同目的，可以将这些模型划分为两类，它们分属于两个不同的层次。

第一类模型是概念模型，也称信息模型，它是按用户的观点来对数据和信息建模，主要用于数据库设计。另一类模型是数据模型，主要包括网状模型、层次模型、关系模型等，它是按计算机系统的观点对数据建模，主要用于 DBMS 的实现。

数据模型是数据库系统的核心和基础。各种机器上实现的 DBMS 软件都是基于某种数据模型的。

为了把现实世界中的具体事物抽象、组织为某一 DBMS 支持的数据模型，人们常常首先将现实世界抽象为信息世界，然后将信息世界转换为机器世界。也就是说，首先把现实世界中的客观对象抽象为某一种信息结构，这种信息结构并不依赖于具体的计算机系统，不是某一个 DBMS 支持的数据模型，而是

概念级的模型；然后再把概念模型转换为计算机上某一 DBMS 支持的数据模型，这一过程如图 1.9 所示。

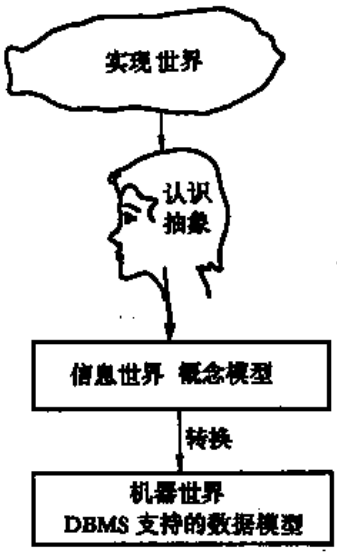


图 1.9 现实世界中客观对象的抽象过程

下面首先介绍数据模型的共性——数据模型的组成要素，然后分别介绍两类不同的数据模型——概念模型和数据模型。

1.2.1 数据模型的组成要素

一般地讲，数据模型是严格定义的一组概念的集合。这些概念精确地描述了系统的静态特性、动态特性和完整性约束条件。因此数据模型通常由数据结构、数据操作和完整性约束三部分组成。

一、数据结构

数据结构是所研究的对象类型的集合。这些对象是数据库的组成成分，它们包括两类，一类是与数据类型、内容、性质有关的对象，例如网状模型中的数据项、记录，关系模型中的域、属性、关系等；一类是与数据之间联系有关的对象，例如网状模型中的系型（Set Type）。

数据结构是刻画一个数据模型性质最重要的方面。因此在数据库系统中，人们通常按照其数据结构的类型来命名数据模型。例如层次结构、网状结构和关系结构的数据模型分别命名为层次模型、网状模型和关系模型。

数据结构是对系统静态特性的描述。

二、数据操作

数据操作是指对数据库中各种对象（型）的实例（值）允许执行的操作的集合，包括操作及有关的操作规则。数据库主要有检索和更新（包括插入、删除、修改）两大类操作。数据模型必须定义这些操作的确切含义、操作符号、操作规则（如优先级）以及实现操作的语言。数据操作是对系统动态特性的描述。

三、数据的约束条件

数据的约束条件是一组完整性规则的集合。完整性规则是给定的数据模型中数据及其联系所具有的制约和依存规则，用以限定符合数据模型的数据库状态以及状态的变化，以保证数据的正确、有效、相容。

数据模型应该反映和规定本数据模型必须遵守的基本的通用的完整性约束条件。例如，在关系模型中，任何关系必须满足实体完整性和参照完整性两个条件（第二章将详细讨论这两个完整性约束条件）。

此外，数据模型还应该提供定义完整性约束条件的机制，以反映具体应用所涉及的数据必须遵守的特定的语义约束条件。例如，在学校的数据库中规定大学生入学年龄不得超过 30 岁，硕士研究生入学年龄不得超过 38 岁，学生累计成绩不得有三门以上不及格等。

1.2.2 概念模型

由图 1.9 可以看出，概念模型实际上是现实世界到机器世界的一个中间层次。

概念模型用于信息世界的建模，是现实世界到信息世界的第一层抽象，是数据库设计人员进行数据库设计的有力工具，也是数据库设计人员和用户之间进行交流的语言，因此概念模型一方面应该具有较强的语义表达能力，能够方便、直接地表达应用中的各种语义知识，另一方面它还应该简单、清晰、易于用户理解。

一、信息世界中的基本概念

信息世界涉及的概念主要有：

（1）实体（Entity）

客观存在并可相互区别的事物称为实体。实体可以是具体的人、事、物，也可以是抽象的概念或联系，例如，一个职工、一个学生、一个部门、一门课、学生的一次选课、部门的一次订货、老师与系的工作关系（即某位老师在某系

工作）等都是实体。

（2）属性（Attribute）

实体所具有的某一特性称为属性。一个实体可以由若干个属性来刻画。例如学生实体可以由学号、姓名、性别、出生年份、系、入学时间等属性组成。（94002268，张山，男，1976，计算机系，1994）这些属性组合起来表征了一个学生。

（3）码（Key）

唯一标识实体的属性集称为码。例如学号是学生实体的码。

（4）域（Domain）

属性的取值范围称为该属性的域。例如，学号的域为 8 位整数，姓名的域为字符串集合，年龄的域为小于 38 的整数，性别的域为（男，女）。

（5）实体型（Entity Type）

具有相同属性的实体必然具有共同的特征和性质。用实体名及其属性名集合来抽象和刻画同类实体，称为实体型。例如，学生（学号，姓名，性别，出生年份，系，入学时间）就是一个实体型。

（6）实体集（Entity Set）

同型实体的集合称为实体集。例如，全体学生就是一个实体集。

（7）联系（Relationship）

在现实世界中，事物内部以及事物之间是有联系的，这些联系在信息世界中反映为实体（型）内部的联系和实体（型）之间的联系。实体内部的联系通常是指组成实体的各属性之间的联系。实体之间的联系通常是指不同实体集之间的联系。

两个实体型之间的联系可以分为三类：

① 一对一联系（1 : 1）

如果对于实体集 A 中的每一个实体，实体集 B 中至多有一个（也可以没有）实体与之联系，反之亦然，则称实体集 A 与实体集 B 具有一对一联系，记为 1 : 1。

例如，学校里面，一个班级只有一个正班长，而一个班长只在一个班中任职，则班级与班长之间具有一对一联系。

② 一对多联系（1 : n）

如果对于实体集 A 中的每一个实体，实体集 B 中有 n 个实体（$n \geq 0$）与之联系，反之，对于实体集 B 中的每一个实体，实体集 A 中至多只有一个实体与之联系，则称实体集 A 与实体集 B 有一对多联系，记为 1 : n。

例如，一个班级中有若干名学生，而每个学生只在一个班级中学习，则班级与学生之间具有一对多联系。

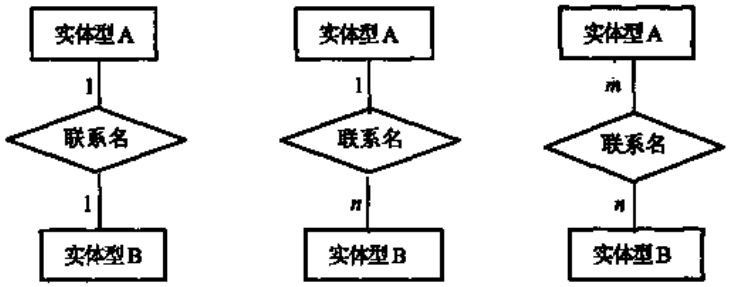
③ 多对多联系（$m:n$）

如果对于实体集 A 中的每一个实体，实体集 B 中有 n 个实体（$n\geqslant 0$）与之联系，反之，对于实体集 B 中的每一个实体，实体集 A 中也有 m 个实体（$m\geqslant 0$）与之联系，则称实体集 A 与实体集 B 具有多对多联系，记为 $m:n$。

例如，一门课程同时有若干个学生选修，而一个学生可以同时选修多门课程，则课程与学生之间具有多对多联系。

实际上，一对一联系是一对多联系的特例，而一对多联系又是多对多联系的特例。

可以用图形来表示两个实体型之间的这三类联系，如图 1.10 所示。



（a）1:1 联系　（b）1:n 联系　（c）m:n 联系

图 1.10　两个实体型之间的三类联系

一般地，两个以上的实体型之间也存在着一对一、一对多、多对多联系。

若实体集 E_1，E_2，…，E_n 存在联系，对于实体集 E_j（j=1，2，…，i-1，i+1，…，n）中的给定实体，最多只和 E_i 中的一个实体相联系，则说 E_i 与 E_1，E_2，…，E_{i-1}，E_{i+1}，…，E_n 之间的联系是一对多的。请读者给出多实体型之间一对一、多对多联系的定义。

例如，对于课程、教师与参考书三个实体型，如果一门课程可以有若干个教师讲授，使用若干本参考书，而每一个教师只讲授一门课程，每一本参考书只供一门课程使用，则课程与教师、参考书之间的联系是一对多的，如图 1.11（a）所示。

又如，有三个实体型：供应商、项目、零件，一个供应商可以供给多个项目多种零件，而每个项目可以使用多个供应商供应的零件，每种零件可由不同供应商供给，由此看出供应商、项目、零件三者之间是多对多的联系，如图 1.11（b）所示。要注意，三个实体型之间多对多的联系和三个实体型两两之间的（三个）多对多联系的语义是不同的。请读者给出供应商、项目、零件三个实体型两两之间的多对多联系的语义，并画出相应的 E-R 图。

同一个实体集内的各实体之间也可以存在一对一、一对多、多对多的联系。例如职工实体集内部具有领导与被领导的联系，即某一职工（干部）“领导”

若干名职工，而一个职工仅被另外一个职工直接领导，因此这是一对多的联系，如图 1.12 所示。

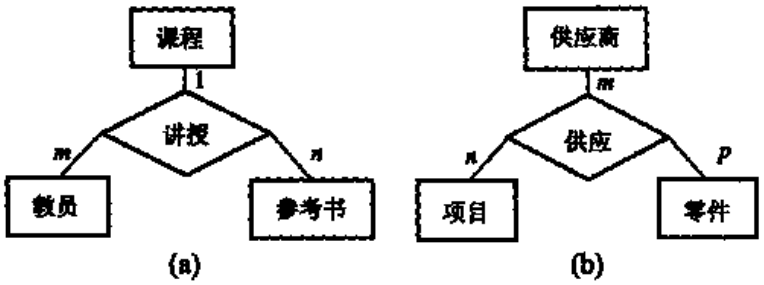


图 1.11 三个实体型之间的联系示例

图 1.12 一个实体型之间一对多联系示例

二、概念模型的表示方法

概念模型是对信息世界建模，所以概念模型应该能够方便、准确地表示出上述信息世界中的常用概念。概念模型的表示方法很多，其中最为著名最为常用的是 P.P.S.Chen 于 1976 年提出的实体-联系方法（Entity-Relationship Approach）。该方法用 E-R 图来描述现实世界的概念模型，E-R 方法也称为 E-R 模型。

这里介绍 E-R 图的要点。有关如何认识和分析现实世界，从中抽取实体和实体之间的联系，建立概念模型的方法将在第六章讲解。

E-R 图提供了表示实体型、属性和联系的方法：

- 实体型：用矩形表示，矩形框内写明实体名。
- 属性：用椭圆形表示，并用无向边将其与相应的实体连接起来。

例如：学生实体具有学号、姓名、性别、出生年份、系、入学时间等属性，用 E-R 图表示如图 1.13 所示。

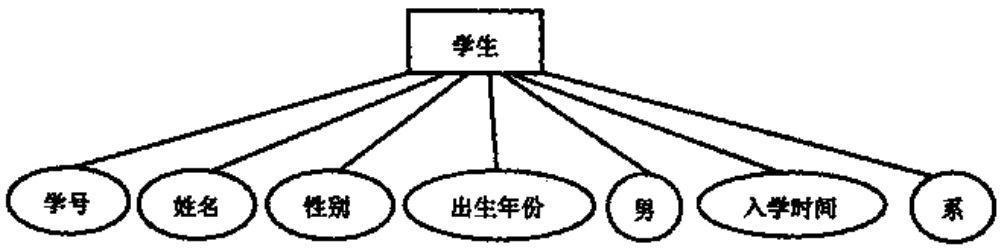


图 1.13 学生实体及属性

- 联系：用菱形表示，菱形框内写明联系名，并用无向边分别与有关实体连接起来，同时在无向边旁标上联系的类型（1 : 1，1 : n 或 m : n）。

需要注意的是，如果一个联系具有属性，则这些属性也要用无向边与该联系连接起来。

例如图 1.11(b)中，用“供应量”来描述联系“供应”的属性，表示某供应商供应了多少数量的零件给某个项目。那么这三个实体及其之间联系的 E-R 图表示可如图 1.14 所示。

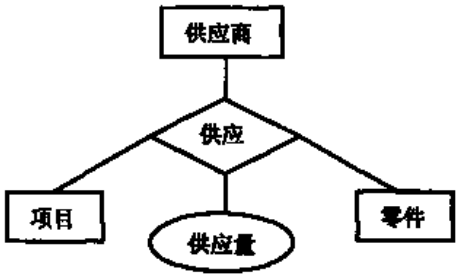


图 1.14 联系的属性

下面用 E-R 图来表示某个工厂物资管理的概念模型。

物资管理涉及的实体有：

- 仓库 属性有仓库号、面积、电话号码。
- 零件 属性有零件号、名称、规格、单价、描述。
- 供应商 属性有供应商号、姓名、地址、电话号码、账号。
- 项目 属性有项目号、预算、开工日期。
- 职工 属性有职工号、姓名、年龄、职称。

这些实体之间的联系如下：

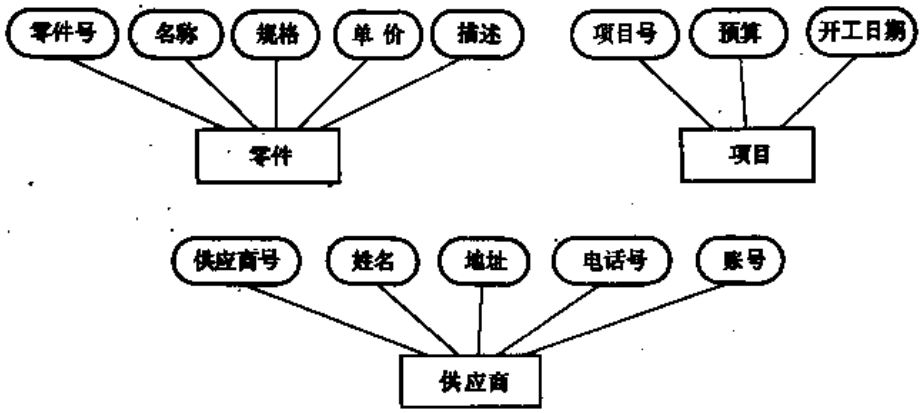
（1）一个仓库可以存放多种零件，一种零件可以存放在多个仓库中，因此仓库和零件具有多对多的联系。用库存量来表示某种零件在某个仓库中的数量。

（2）一个仓库有多个职工当仓库保管员，一个职工只能在一个仓库工作，因此仓库和职工之间是一对多的联系。

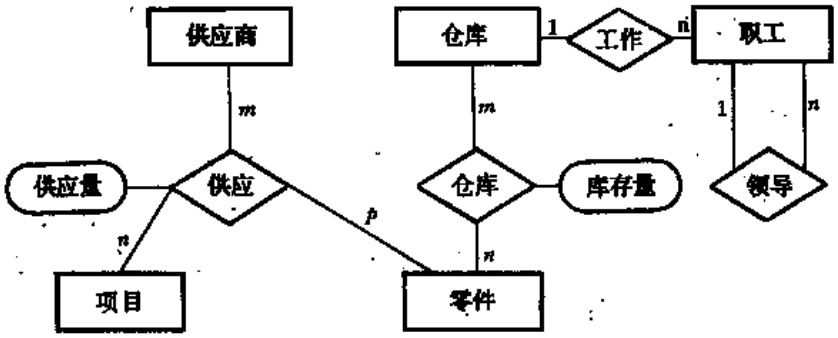
（3）职工之间具有领导-被领导关系。即仓库主任领导若干保管员，因此职工实体集中具有一对多的联系。

（4）供应商、项目和零件三者之间具有多对多的联系。即一个供应商可以供给若干项目多种零件，每个项目可以使用不同供应商供应的零件，每种零件可由不同供应商供给。

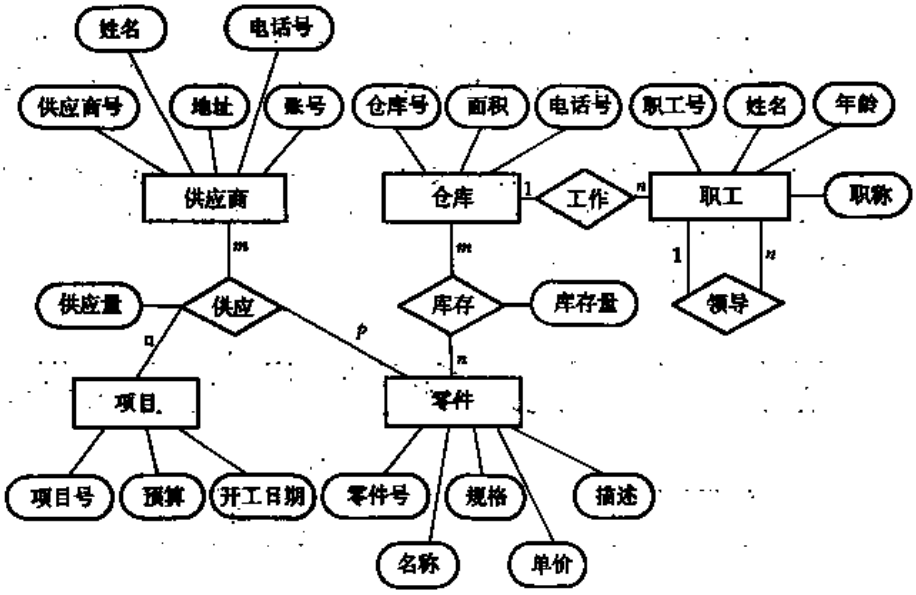
下面给出此工厂的物资管理 E-R 图(如图 1.15(c))。图 1.15(a)为实体及其属性图，图 1.15(b)为实体及其联系图。我们把实体的属性用另图画出仅仅是为了更清晰地表示实体及其实体之间的联系。



(a) 实体及其属性图



(b) 实体及其联系图



(c) 完整的实体联系图

图 1.15 工厂物资管理 E-R 图

实体-联系方法是抽象和描述现实世界的有力工具。用 E-R 图表示的概念模型独立于具体的 DBMS 所支持的数据模型，它是各种数据模型的共同基础，因而比数据模型更一般、更抽象、更接近现实世界。

1.2.3 最常用的数据模型

目前，数据库领域中最常用的数据模型有四种，它们是：

- 层次模型 (Hierarchical Model)
- 网状模型 (Network Model)
- 关系模型 (Relational Model)
- 面向对象模型 (Object Oriented Model)

其中层次模型和网状模型统称为非关系模型。

非关系模型的数据库系统在 20 世纪 70 年代至 80 年代初非常流行，在数据库系统产品中占据了主导地位，现在已逐渐被关系模型的数据库系统取代，但在美国等一些国家里，由于早期开发的应用系统都是基于层次数据库或网状数据库系统的，因此目前仍有不少层次数据库或网状数据库系统在继续使用。

20 世纪 80 年代以来，面向对象的方法和技术在计算机各个领域，包括程序设计语言、软件工程、信息系统设计、计算机硬件设计等各方面都产生了深远的影响，也促进数据库中面向对象数据模型的研究和发展。

本章简要介绍层次模型、网状模型和关系模型。面向对象的数据模型将在第十三章中介绍。

数据结构、数据操作和完整性约束条件这三个方面的内容完整地描述了一个数据模型，其中数据结构是刻画模型性质的最基本的方面。为了使读者对数据模型有一个基本认识，下面着重介绍三种模型的数据结构。

注意：这里讲的数据模型都是逻辑上的，也就是说是用户眼中看到的数据范围。同时它们又都是能用某种语言描述，使计算机系统能够理解，被数据库管理系统支持的数据视图。这些数据模型将以一定的方式存储于数据库系统中，这是 DBMS 的功能，是 DBMS 中的存储模型。

在非关系模型中，实体用记录表示，实体的属性对应记录的数据项（或字段）。实体之间的联系在非关系模型中转换成记录之间的两两联系。

非关系模型中数据结构的单位是基本层次联系。所谓**基本层次联系**是指两个记录以及它们之间的一对多（包括一对一）的联系，如图 1.16 所示。

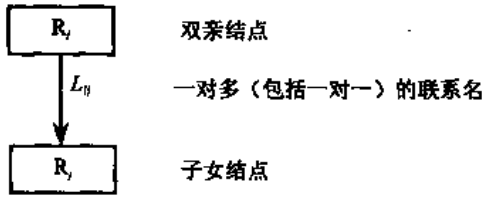


图 1.16 基本层次联系

图中 R_i 位于联系 L_{ij} 的始点，

称为**双亲结点**（Parent），R_j位于联系L_{ij}的终点，称为**子女结点**（Child）。

1.2.4 层次模型

层次模型是数据库系统中最早出现的数据模型，层次数据库系统采用层次模型作为数据的组织方式。层次数据库系统的典型代表是 IBM 公司的 IMS（Information Management System）数据库管理系统，这是 1968 年 IBM 公司推出的第一个大型的商用数据库管理系统，曾经得到广泛的使用。

层次模型用树形结构来表示各类实体以及实体间的联系。现实世界中许多实体之间的联系本来就呈现出一种很自然的层次关系，如行政机构、家族关系等。

一、层次数据模型的数据结构

在数据库中定义满足下面两个条件的基本层次联系的集合为层次模型。

（1）有且只有一个结点没有双亲结点，这个结点称为根结点；

（2）根以外的其他结点有且只有一个双亲结点。

在层次模型中，每个结点表示一个记录类型，记录（类型）之间的联系用结点之间的连线（有向边）表示，这种联系是父子之间的一对多的联系。这就使得层次数据库系统只能处理一对多的实体联系。

每个记录类型可包含若干个字段，这里，记录类型描述的是实体，字段描述的是实体的属性。各个记录类型及其字段都必须命名。各个记录类型、同一记录类型中各个字段不能同名。每个记录类型可以定义一个排序字段，也称为码字段，如果定义该排序字段的值是唯一的，则它能唯一地标识一个记录值。

一个层次模型在理论上可以包含任意有限个记录型和字段，但任何实际的系统都会因为存储容量或实现复杂度而限制层次模型中包含的记录型个数和字段的个数。

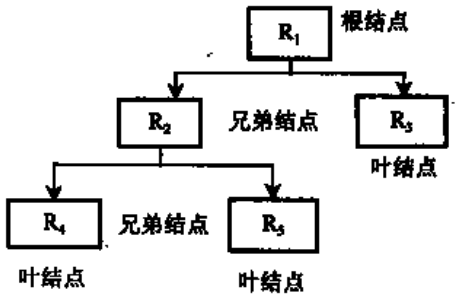


图 1.17 一个层次模型的示例

在层次模型中，同一双亲的子女结点称为兄弟结点（Twin 或 Sibling），没有子女结点的结点称为叶结点。图 1.17 给出了一个层次模型的例子。其中 R_1 为根结点；R_2 和 R_3 为兄弟结点，是 R_1 的子女结点；R_4 和 R_5 为兄弟结点，是 R_2 的子女结点；R_3，R_4 和 R_5 为叶结点。

从图上可以看出层次模型像一棵倒立的树，结点的双亲是唯一的。

层次模型的一个基本的特点是，任何一个给定的记录值只有按其路径查看时，才能显出它的全部意义，没有一个子女记录值能够脱离双亲记录值而独立

存在。

图 1.18 是一个教员 学生层次数据库。该层次数据库有四个记录型。记录型系是根结点，由系编号、系名、办公地点三个字段组成。它有两个子女结点教研室和学生。记录型教研室是系的子女结点，同时又是教员的双亲结点，它由教研室编号、教研室名两个字段组成。记录类型学生由学号、姓名、成绩三个字段组成。记录教员由职工号、姓名、研究方向三个字段组成。学生与教员是叶结点，它们没有子女结点。由系到教研室、由教研室到教员、由系到学生均是一对多的联系。

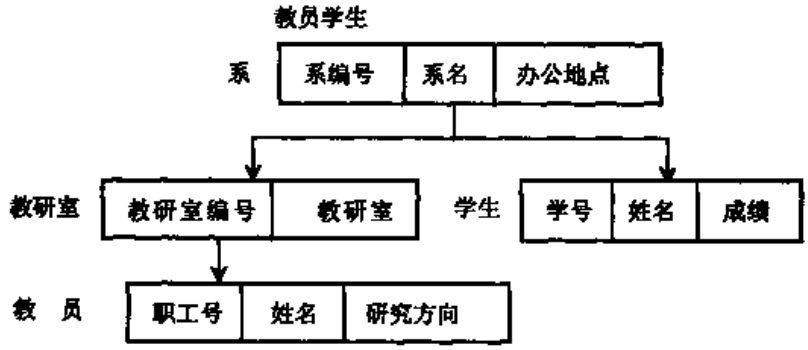


图 1.18 教员 学生数据库模型

图 1.19 是图 1.18 数据模型对应的一个值。该值是 D02 系（计算机科学系）记录值及其所有后代记录值组成的一棵树。D02 系有三个教研室子女记录值：R01，R02，R03 和三个学生记录值：S63871，S63874，S63876。教研室 R01 有三个教员记录值：E2101，E1709，E3501；教研室 R03 有两个教员记录值：E1101，E3102。

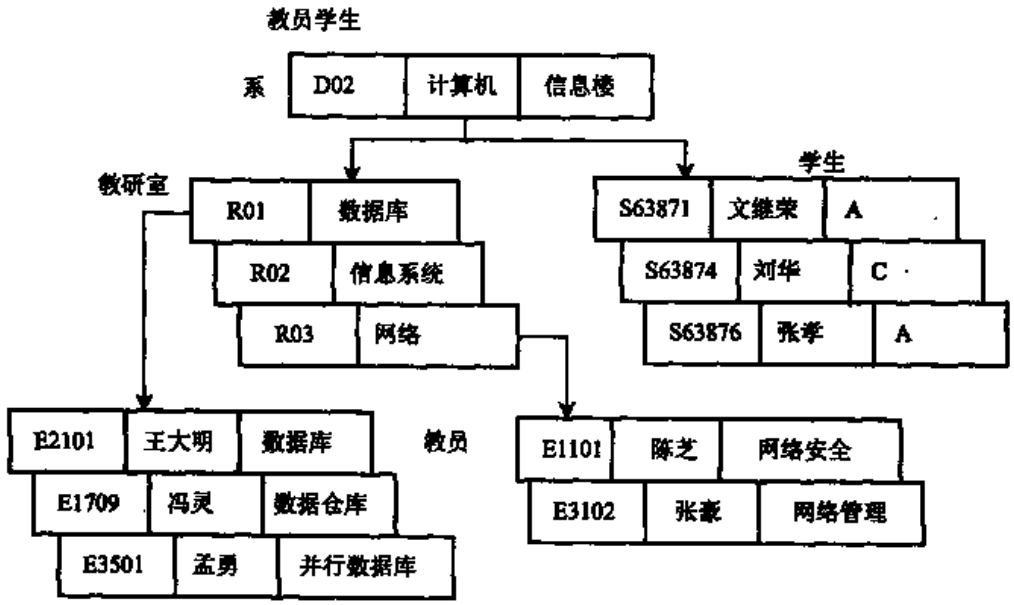


图 1.19 教员学生数据库的一个值

二、多对多联系在层次模型中的表示

前面已经说过，层次数据模型只能直接表示一对多（包括一对一）的联系，那么另一种常见联系-多对多联系能否在层次模型中表示呢？答案是肯定的，否则层次模型就无法真正反映现实世界了。但是用层次模型表示多对多联系，必须首先将其分解成一对多联系。分解方法有两种：冗余结点法和虚拟结点法。下面用一个例子来说明这两种分解方法。

图 1.20(a)是一个简单的多对多联系：一个学生可以选修多门课程，一门课程可由多个学生选修。学生的字段有学号、姓名、成绩三个字段组成，课程由课程号和课程名两个字段组成。

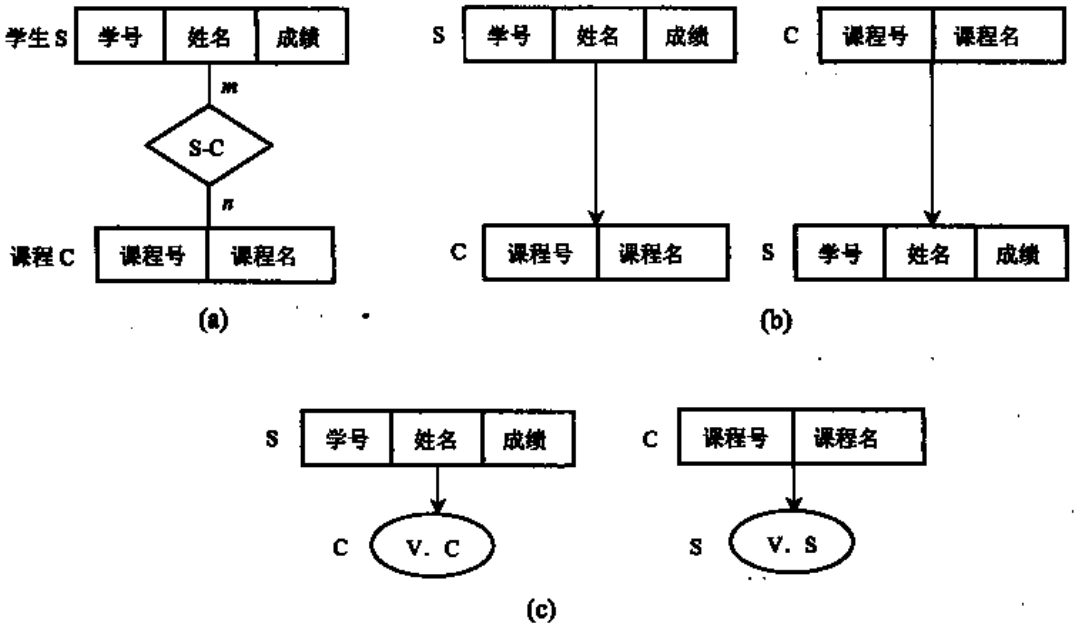


图 1.20 用层次模型表示多对多联系

图 1.20 (b)采用冗余结点法，即通过增设两个冗余结点将图 1.20 (a)的多对多联系转换成两个一对多联系，图 1.20 (c)采用虚拟结点的分解方法，即将图(b)中的冗余结点换为虚拟结点，所谓虚拟结点就是一个指引元，指向所替代的结点。

冗余结点法的优点是结构清晰，允许结点改变存储位置，缺点是需要额外占用存储空间，有潜在的不一致性。虚拟结点法的优点是减少对存储空间的浪费，避免产生潜在的不一致性，缺点是结点改变存储位置可能引起虚拟结点中指针的修改。

二、层次模型的数据操纵与完整性约束

层次模型的数据操纵主要有查询、插入、删除和修改。进行插入、删除、修改操作时要满足层次模型的完整性约束条件。

进行插入操作时，如果没有相应的双亲结点值就不能插入子女结点值。例如在图 1.18 的层次数据库中，若新调入一名教师，但尚未分配到某个教研室，这时就不能将新教员插入到数据库中。

进行删除操作时，如果删除双亲结点值，则相应的子女结点值也被同时删除。例如在图 1.18 的层次数据库中，若删除网络教研室，则该教研室所有老师的数据将全部丢失。

进行修改操作时，应修改所有相应记录，以保证数据的一致性。例如在图 1.20(b)的层次模型中，如果一个学生要改姓名，则两处学生记录值的姓名字段都需修改。

三、层次数据模型的存储结构

层次数据库中不仅要存储数据本身，还要存储数据之间的层次联系。层次模型数据的存储常常是和数据之间联系的存储结合在一起的。常用的实现方法有两种：

1. 邻接法

按照层次树前序穿越的顺序把所有记录值依次邻接存放，即通过物理空间的位置相邻来体现（或隐含）层次顺序。例如对于图 1.21（a）的数据库，按邻接法存放图 1.21(b)中以根记录 A1 为首的层次记录实例集，则应如图 1.22 所示存放（为简单起见，仅用记录值的第一个字段来代表该记录值）：

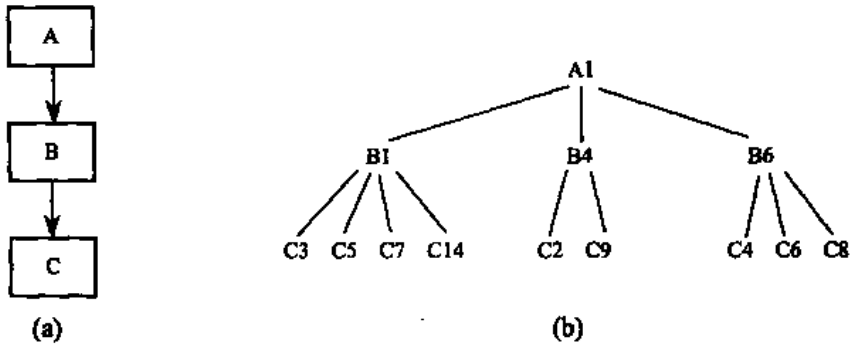


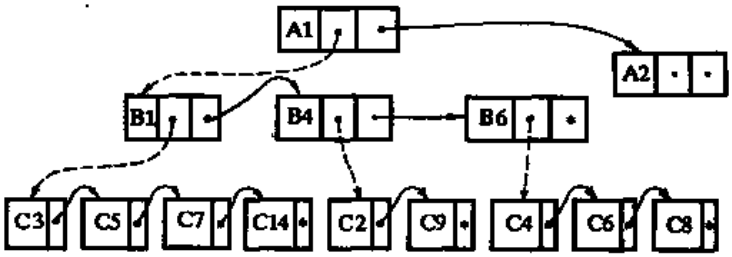
图 1.21 层次数据库及其实例

A1	B1	C3	C5	C7	C14	B4	C2	C9	B6	C4	C6	C8	A2	...

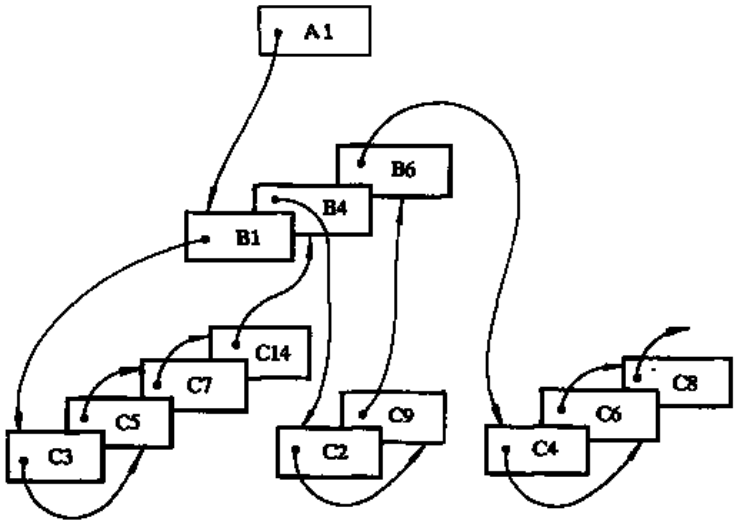
图 1.22 邻接法

2. 链接法

用指引元来反映数据之间的层次联系，如图 1.23 所示。其中图 1.23(a)每个记录设两类指引元，分别指向最左边的子女（每个记录型对应一个）和最近的兄弟，这种链接方法称为子女-兄弟链接法；图 1.23(b)是按树的前序穿越顺序链接各记录值，这种链接方法称为层次序列链接法。



(a)



(b)

图 1.23 链接法

四、层次模型的优缺点

层次模型的优点主要有：

- 层次数据模型本身比较简单。
- 对于实体间联系是固定的，且预先定义好的应用系统，采用层次模型来实现，其性能优于关系模型，不低于网状模型。
- 层次数据模型提供了良好的完整性支持。

层次模型的缺点主要有：

- 现实世界中很多联系是非层次性的，如多对多联系、一个结点具有多个

双亲等，层次模型表示这类联系的方法很笨拙，只能通过引入冗余数据（易产生不一致性）或创建非自然的数据组织（引入虚拟结点）来解决。

- 对插入和删除操作的限制比较多。
- 查询子女结点必须通过双亲结点。
- 由于结构严密，层次命令趋于程序化。

可见用层次模型对具有一对多的层次关系的部门描述非常自然、直观，容易理解。这是层次数据库的突出优点。

1.2.5 网状模型

在现实世界中事物之间的联系更多的是非层次关系的，用层次模型表示非树形结构是很不直接的，网状模型则可以克服这一弊病。

网状数据库系统采用网状模型作为数据的组织方式。网状数据模型的典型代表是 DBTG 系统，亦称 CODASYL 系统。这是 20 世纪 70 年代数据系统语言研究会 CODASYL（Conference On Data System Language）下属的数据库任务组（Data Base Task Group，简称 DBTG）提出的一个系统方案。DBTG 系统虽然不是实际的软件系统，但是它提出的基本概念、方法和技术具有普遍意义。它对于网状数据库系统的研制和发展起了重大的影响。后来不少的系统都采用 DBTG 模型或者简化的 DBTG 模型。例如，Cullinet Software 公司的 IDMS、Univac 公司的 DMS1100、Honeywell 公司的 IDS/2、HP 公司的 IMAGE 等。

一、网状数据模型的数据结构

在数据库中，把满足以下两个条件的基本层次联系集合称为网状模型：

（1）允许一个以上的结点无双亲；

（2）一个结点可以有多于一个的双亲。

网状模型是一种比层次模型更具普遍性的结构，它去掉了层次模型的两个限制，允许多个结点没有双亲结点，允许结点有多个双亲结点，此外它还允许两个结点之间有多种联系（称之为复合联系）。因此网状模型可以更直接地去描述现实世界。而层次模型实际上是网状模型的一个特例。

与层次模型一样，网状模型中每个结点表示一个记录类型（实体），每个记录类型可包含若干个字段（实体的属性），结点间的连线表示记录类型（实体）之间一对多的父子联系。

从定义可以看出，层次模型中子女结点与双亲结点的联系是唯一的，而在网状模型中这种联系可以不唯一。因此，要为每个联系命名，并指出与该联系有关的双亲记录和子女记录。例如图 1.24(a)中 R_3 有两个双亲记录 R_1 和 R_2，

因此把 R_1 与 R_3 之间的联系命名为 L_1，R_2 与 R_3 之间的联系命名为 L_2。图 1.24(a)，(b)，(c)都是网状模型的例子。

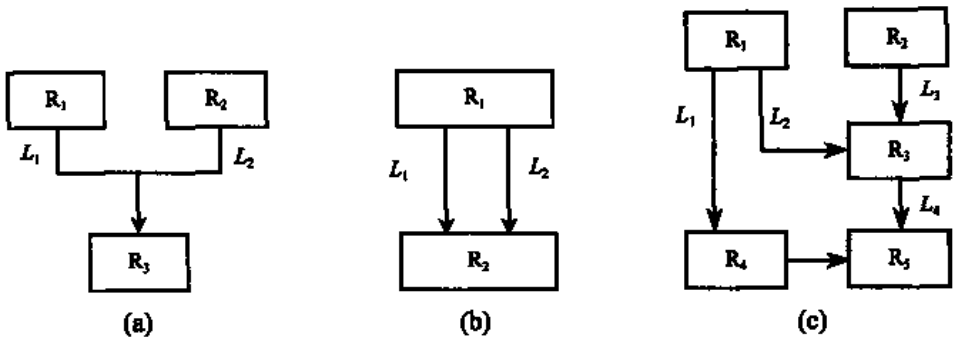


图 1.24 网状模型的例子

实际的商品化网状数据库系统对网状数据结构都有不同的限制，例如 HP 公司的 IMAGE3000 数据库管理系统是一个网状数据库系统，但是限制网状结构的层次只有两层，是一个简化了的网状结构；这时就需要把现实世界一般的网状结构转换成系统所能处理的结构。

下面以学生选课为例，看一看网状数据库模式是怎样来组织数据的。

按照常规语义，一个学生可以选修若干门课程，某一课程可以被多个学生选修，因此学生与课程之间是多对多联系。这样的实体联系图不能直接用 DBTG 模型来表示。因为 DBTG 模型中不能表示记录之间多对多的联系。为此引进一个学生选课的联结记录，它由三个数据项组成，即学号、课程号、成绩，表示某个学生选修某一门课程及其成绩。

这样，学生选课数据库包括三个记录：学生、课程和选课。

每个学生可以选修多门课程，显然对学生记录中的一个值，选课记录中可以有多个值与之联系，而选课记录中的一个值，只能与学生记录中的一个值联系。学生与选课之间的联系是一对多的联系，联系名为 S-SC。同样，课程与选课之间的联系也是一对多的联系，联系名为 C-SC。图 1.25 为学生选课数据库的网状数据库模式。

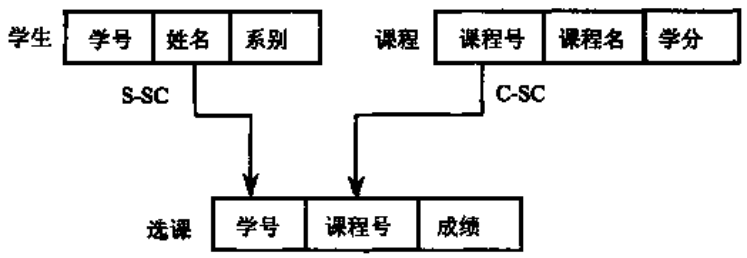


图 1.25 学生/选课/课程的网状数据库模式

二、网状数据模型的操纵与完整性约束

网状数据模型一般来说没有层次模型那样严格的完整性约束条件，但具体的网状数据库系统（如 DBTG）对数据操纵都加了一些限制，提供了一定的完整性约束。

DBTG 在模式 DDL 中提供了定义 DBTG 数据库完整性的若干概念和语句，主要有：

（1）支持记录码的概念，码即唯一标识记录的数据项的集合。例如，学生记录（如图 1.25）中学号是码，因此数据库中不允许学生记录中学号出现重复值。

（2）保证一个联系中双亲记录和子女记录之间是一对多的联系。

（3）可以支持双亲记录和子女记录之间某些约束条件。例如，有些子女记录要求双亲记录存在才能插入，双亲记录删除时也连同删除。例如图 1.26 中 SC 记录就应该满足这种约束条件，学生选课记录值必须是数据库中存在的某一学生，某一门课的选修记录。DBTG 提供了“属籍类别”的概念来描述这类约束条件。

三、网状数据模型的存储结构

网状数据模型的存储结构中关键是如何实现记录之间的联系。常用的方法是链接法，包括单向链接、双向链接、环状链接、向首链接等，此外还有其他实现方法，如指引元阵列法、二进制阵列法、索引法等依具体系统不同而不同。

图 1.26 为学生选课网状数据库的一个存储示意图。

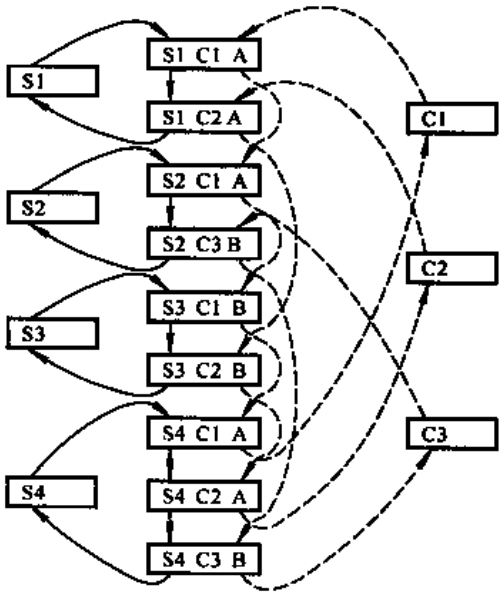


图 1.26 学生/选课/课程的网状数据库实例

学生选课数据库中学生、课程和选课三个记录的值可以分别按某种文件组织方式存储，记录之间的联系用单向链接法实现。图中实线链表示 S-SC 联系，虚线链表示 C-SC 联系。即把 S1 学生和他的选课记录（选修的 C1，C2 两门课程的选课记录）链接起来，同样把 S2，S3，S4 学生和他们的选课记录链接起来；把 C1 课程和选修了 C1 课程的学生记录（有 S1，S2，S3，S4 学生选修了 C1）链接起来，同样把 C2，C3 课程和选修了这些课程的学生记录链接起来。

四、网状数据模型的优缺点

网状数据模型的优点主要有：

- 能够更为直接地描述现实世界，如一个结点可以有多个双亲。
- 具有良好的性能，存取效率较高。

网状数据模型的缺点主要有：

- 结构比较复杂，而且随着应用环境的扩大，数据库的结构就变得越来越复杂，不利于最终用户掌握。
- 其 DDL，DML 语言复杂，用户不容易使用。

由于记录之间联系是通过存取路径实现的，应用程序在访问数据时必须选择适当的存取路径，因此，用户必须了解系统结构的细节，加重了编写应用程序的负担。

1.2.6 关系模型

关系模型是目前最重要的一种数据模型。关系数据库系统采用关系模型作为数据的组织方式。

1970 年美国 IBM 公司 San Jose 研究室的研究员 E.F.Codd 首次提出了数据库系统的关系模型，开创了数据库关系方法和关系数据理论的研究，为数据库技术奠定了理论基础。由于 E.F.Codd 的杰出工作，他于 1981 年获得 ACM 图灵奖。

20 世纪 80 年代以来，计算机厂商新推出的数据库管理系统几乎都支持关系模型，非关系系统的产品也大都加上了关系接口。数据库领域当前的研究工作也都是以关系方法为基础。因此本书的重点也将放在关系数据库上，下面四章将详细讲解关系数据库。

一、关系数据模型的数据结构

关系模型与以往的模型不同，它是建立在严格的数学概念的基础上的。严

格的定义将在下一章给出。这里只简单勾画一下关系模型。在用户观点下，关系模型中数据的逻辑结构是一张二维表，它由行和列组成。现在以学生登记表（如图 1.27 所示）为例，介绍关系模型中的一些术语。

学生登记表

学 号	姓 名	年 龄	性 别	系 名	年 级
95004	王小明	19	女	社会学	95
95006	黄大鹏	20	男	商品学	95
95008	张文斌	18	女	法律学	95
…	…	…	…	…	…

图 1.27 关系模型的数据结构

• 关系（Relation）：一个关系对应通常说的一张表，如图 1.27 中的这张学生登记表；

• 元组（Tuple）：表中的一行即为一个元组；

• 属性（Attribute）：表中的一列即为一个属性，给每一个属性起一个名称即属性名。如上表有六列，对应六个属性（学号，姓名，年龄，性别，系和年级）；

• 主码（Key）：表中的某个属性组，它可以唯一确定一个元组，如图 1.27 中的学号，可以唯一确定一个学生，也就成为本关系的主码；

• 域（Domain）：属性的取值范围，如人的年龄一般在 1～150 岁之间，大学生年龄属性的域是（14～38），性别的域是（男，女），系别的域是一个学校所有系名的集合；

• 分量：元组中的一个属性值；

• 关系模式：对关系的描述，一般表示为：

关系名（属性 1，属性 2，…，属性 n）

例如上面的关系可描述为：

学生（学号，姓名，年龄，性别，系和年级）

在关系模型中，实体以及实体间的联系都是用关系来表示。例如学生、课程、学生与课程之间的多对多联系在关系模型中可以如下表示：

学生（学号，姓名，年龄，性别，系和年级）

课程（课程号，课程名，学分）

选修（学号，课程号，成绩）

关系模型要求关系必须是规范化的，即要求关系必须满足一定的规范条件，这些规范条件中最基本的一条就是，关系的每一个分量必须是一个不可分的数据项，也就是说，不允许表中还有表，图 1.28 中工资和扣除是可分的数

据项，工资又分为基本工资、工龄工资和职务工资，扣除又分为房租和水电。因此，图 1.28 的表就不符合关系模型要求。

职工号	姓名	职称	工资			扣除		实发
			基本	工龄	职务	房租	水电	
86051	陈平	讲师	805	20	50	60	12	803
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

图 1.28 表中有表示例

二、关系数据模型的操纵与完整性约束

关系数据模型的操作主要包括查询、插入、删除和修改数据。这些操作必须满足关系的完整性约束条件。关系的完整性约束条件包括三大类：实体完整性、参照完整性和用户定义的完整性。其具体含义将在后面介绍。

关系模型中的数据操作是集合操作，操作对象和操作结果都是关系，即若干元组的集合，而不像非关系模型中那样是单记录的操作方式。另一方面，关系模型把存取路径向用户隐蔽起来，用户只要指出“干什么”或“找什么”，不必详细说明“怎么干”或“怎么找”，从而大大地提高了数据的独立性，提高了用户生产率。

三、关系数据模型的存储结构

在关系数据模型中，实体及实体间的联系都用表来表示。在数据库的物理组织中，表以文件形式存储，有的系统一个表对应一个操作系统文件，有的系统自己设计文件结构。

四、关系数据模型的优缺点

关系数据模型具有下列优点：

- 关系模型与非关系模型不同，它是建立在严格的数学概念的基础上的。
- 关系模型的概念单一。无论实体还是实体之间的联系都用关系表示。对数据的检索结果也是关系（即表）。所以其数据结构简单、清晰，用户易懂易用。
- 关系模型的存取路径对用户透明，从而具有更高的数据独立性、更好的安全保密性，也简化了程序员的工作和数据库开发建立的工作。

所以，关系数据模型诞生以后发展迅速，深受用户的喜爱。

当然，关系数据模型也有缺点，其中最主要的缺点是，由于存取路径对用户透明，查询效率往往不如非关系数据模型。因此为了提高性能，必须对用户

的查询请求进行优化，增加了开发数据库管理系统的难度。

1.3 数据库系统结构

考察数据库系统的结构可以有多种不同的层次或不同的角度。

从数据库管理系统角度看，数据库系统通常采用三级模式结构；这是数据库管理系统内部的系统结构。

从数据库最终用户角度看，数据库系统的结构分为集中式结构（又可有单用户结构、主从式结构）、分布式结构、客户／服务器结构和并行结构。这是数据库系统外部的体系结构。

本章介绍数据库系统的模式结构。

1.3.1 数据库系统模式的概念

在数据模型中有“型”（Type）和“值”（Value）的概念。型是指对某一类数据的结构和属性的说明，值是型的一个具体赋值。例如：学生记录定义为（学号，姓名，性别，系别，年龄，籍贯）这样的记录型，而（900201，李明，男，计算机，22，江苏）则是该记录型的一个记录值。

模式（Schema）是数据库中全体数据的逻辑结构和特征的描述，它仅仅涉及到型的描述，不涉及到具体的值。模式的一个具体值称为模式的一个实例（Instance）。同一个模式可以有很多实例。模式是相对稳定的，而实例是相对变动的，因为数据库中的数据是在不断更新的。模式反映的是数据的结构及其联系，而实例反映的是数据库某一时刻的状态。

虽然实际的数据库管理系统产品种类很多，它们支持不同的数据模型，使用不同的数据库语言，建立在不同的操作系统之上，数据的存储结构也各不相同，但它们在体系结构上通常都具有相同的特征，即采用三级模式结构（早期微机上的小型数据库系统除外）并提供两级映象功能。

1.3.2 数据库系统的三级模式结构

数据库系统的三级模式结构是指数据库系统是由外模式、模式和内模式三级构成，如图 1.29 所示。

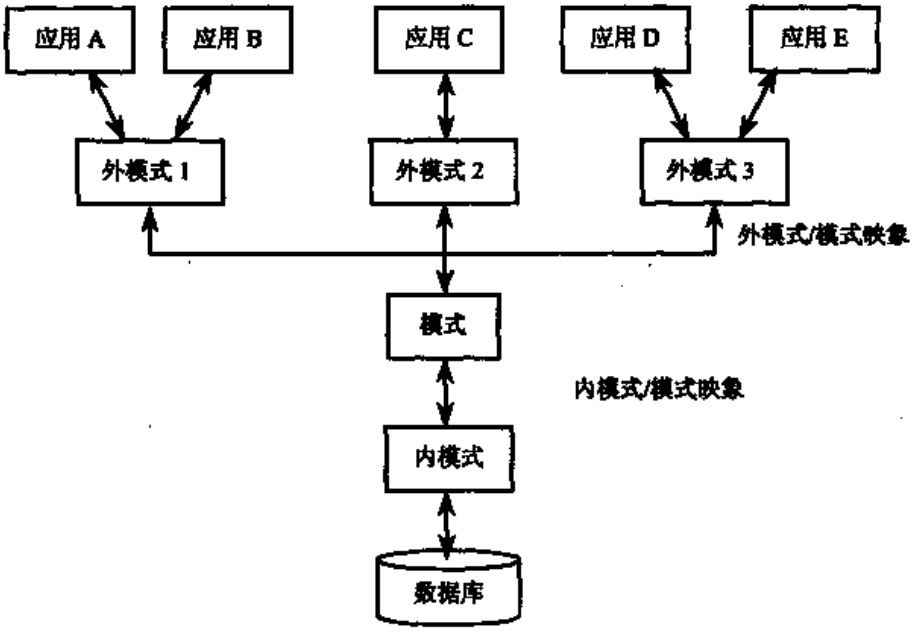


图 1.29 数据库系统的三级模式结构

一、模式（Schema）

模式也称逻辑模式，是数据库中全体数据的逻辑结构和特征的描述，是所有用户的公共数据视图。它是数据库系统模式结构的中间层，既不涉及数据的物理存储细节和硬件环境，也与具体的应用程序，与所使用的应用开发工具及高级程序设计语言（如 C，COBOL，FORTRAN）无关。

模式实际上是数据库数据在逻辑级上的视图。一个数据库只有一个模式。数据库模式以某一种数据模型为基础，统一综合地考虑了所有用户的需求，并将这些需求有机地结合成一个逻辑整体。定义模式时不仅要定义数据的逻辑结构，例如数据记录由哪些数据项构成，数据项的名字、类型、取值范围等，而且要定义数据之间的联系，定义与数据有关的安全性、完整性要求。

DBMS 提供模式描述语言（模式 DDL）来严格地定义模式。

二、外模式（External Schema）

外模式也称子模式（Subschema）或用户模式，它是数据库用户（包括应用程序员和最终用户）能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。

外模式通常是模式的子集。一个数据库可以有多个外模式。由于它是各个用户的数据视图，如果不同的用户在应用需求、看待数据的方式、对数据保密的要求等方面存在差异，则其外模式描述就是不同的。即使对模式中同一数据，

在外模式中的结构、类型、长度、保密级别等都可以不同。另一方面，同一外模式也可以为某一用户的多个应用系统所使用，但一个应用程序只能使用一个外模式。

外模式是保证数据库安全性的一个有力措施。每个用户只能看见和访问所对应的外模式中的数据，数据库中的其余数据是不可见的。

DBMS 提供子模式描述语言（子模式 DDL）来严格地定义子模式。

三、内模式（Internal Schema）

内模式也称存储模式（Storage Schema），一个数据库只有一个内模式。它是数据物理结构和存储方式的描述，是数据在数据库内部的表示方式。例如，记录的存储方式是顺序存储、按照 B 树结构存储还是按 hash 方法存储；索引按照什么方式组织；数据是否压缩存储，是否加密；数据的存储记录结构有何规定等。

DBMS 提供内模式描述语言（内模式 DDL，或者存储模式 DDL）来严格地定义内模式。

1.3.3 数据库的二级映象功能与数据独立性

数据库系统的三级模式是对数据的三个抽象级别，它把数据的具体组织留给 DBMS 管理，使用户能逻辑地抽象地处理数据，而不必关心数据在计算机中的具体表示方式与存储方式。为了能够在内部实现这三个抽象层次的联系和转换，数据库管理系统在这三级模式之间提供了两层映象：

- 外模式 / 模式映象
- 模式 / 内模式映象

正是这两层映象保证了数据库系统中的数据能够具有较高的逻辑独立性和物理独立性。

一、外模式 / 模式映象

模式描述的是数据的全局逻辑结构，外模式描述的是数据的局部逻辑结构。对应于同一个模式可以有任意多个外模式。对于每一个外模式，数据库系统都有一个外模式 / 模式映象，它定义了该外模式与模式之间的对应关系。这些映象定义通常包含在各自外模式的描述中。

当模式改变时（例如增加新的关系、新的属性、改变属性的数据类型等），由数据库管理员对各个外模式 / 模式的映象作相应改变，可以使外模式保持不变。应用程序是依据数据的外模式编写的，从而应用程序不必修改，保证了数

据与程序的逻辑独立性，简称数据的逻辑独立性。

二、模式 / 内模式映象

数据库中只有一个模式，也只有一个内模式，所以模式 / 内模式映象是唯一的，它定义了数据库全局逻辑结构与存储结构之间的对应关系。例如，说明逻辑记录和字段在内部是如何表示的。该映象定义通常包含在模式描述中。当数据库的存储结构改变了（例如选用了另一种存储结构），由数据库管理员对模式 / 内模式映象作相应改变，可以使模式保持不变，从而应用程序也不必改变。保证了数据与程序的物理独立性，简称数据的物理独立性。

在数据库的三级模式结构中，数据库模式即全局逻辑结构是数据库的中心与关键，它独立于数据库的其他层次。因此设计数据库模式结构时应首先确定数据库的逻辑模式。

数据库的内模式依赖于它的全局逻辑结构，但独立于数据库的用户视图即外模式，也独立于具体的存储设备。它是将全局逻辑结构中所定义的数据结构及其联系按照一定的物理存储策略进行组织，以达到较好的时间与空间效率。

数据库的外模式面向具体的应用程序，它定义在逻辑模式之上，但独立于存储模式和存储设备。当应用需求发生较大变化，相应外模式不能满足其视图要求时，该外模式就得做相应改动，所以设计外模式时应充分考虑到应用的扩充性。

特定的应用程序是在外模式描述的数据结构上编制的，它依赖于特定的外模式，与数据库的模式和存储结构独立。不同的应用程序有时可以共用同一个外模式。数据库的二级映象保证了数据库外模式的稳定性，从而从底层保证了应用程序的稳定性，除非应用需求本身发生变化，否则应用程序一般不需要修改。

数据与程序之间的独立性，使得数据的定义和描述可以从应用程序中分离出去。另外，由于数据的存取由 DBMS 管理，用户不必考虑存取路径等细节，从而简化了应用程序的编制，大大减少了应用程序的维护和修改。

1.4 数据库系统的组成

在本章一开始介绍了数据库系统一般由数据库、数据库管理系统（及其开发工具）、应用系统、数据库管理员和用户构成。下面分别介绍这几个部分的内容。

一、硬件平台及数据库

由于数据库系统数据量都很大，加之 DBMS 丰富的功能使得自身的规模也很大，因此整个数据库系统对硬件资源提出了较高的要求，这些要求是：

（1）要有足够大的内存，存放操作系统、DBMS 的核心模块、数据缓冲区和应用程序。

（2）有足够大的磁盘等直接存取设备存放数据库，有足够的磁带（或微机软盘）作数据备份。

（3）要求系统有较高的通道能力，以提高数据传送率。

二、软件

数据库系统的软件主要包括：

（1）DBMS。DBMS 是为数据库的建立、使用和维护配置的软件。

（2）支持 DBMS 运行的操作系统。

（3）具有与数据库接口的高级语言及其编译系统，便于开发应用程序。

（4） 以 DBMS 为核心的应用开发工具。

应用开发工具是系统为应用开发人员和最终用户提供的高效率、多功能的应用生成器、第四代语言等各种软件工具。它们为数据库系统的开发和应用提供了良好的环境。

（5） 为特定应用环境开发的数据库应用系统。

三、人员

开发、管理和使用数据库系统的人员主要是：数据库管理员、系统分析员和数据库设计人员、应用程序员和最终用户。不同的人员涉及不同的数据抽象级别，具有不同的数据视图，如图 1.30 所示，其各自的职责分别是：

1. 数据库管理员（DataBase Administrator，简称 DBA)

在数据库系统环境下，有两类共享资源。一类是数据库，另一类是数据库管理系统软件。因此需要有专门的管理机构来监督和管理数据库系统。DBA 则是这个机构的一个（组）人员，负责全面管理和控制数据库系统。具体职责包括：

（1）决定数据库中的信息内容和结构

数据库中要存放哪些信息，DBA 要参与决策。因此 DBA 必须参加数据库设计的全过程，并与用户、应用程序员、系统分析员密切合作共同协商，搞好数据库设计。

（2）决定数据库的存储结构和存取策略

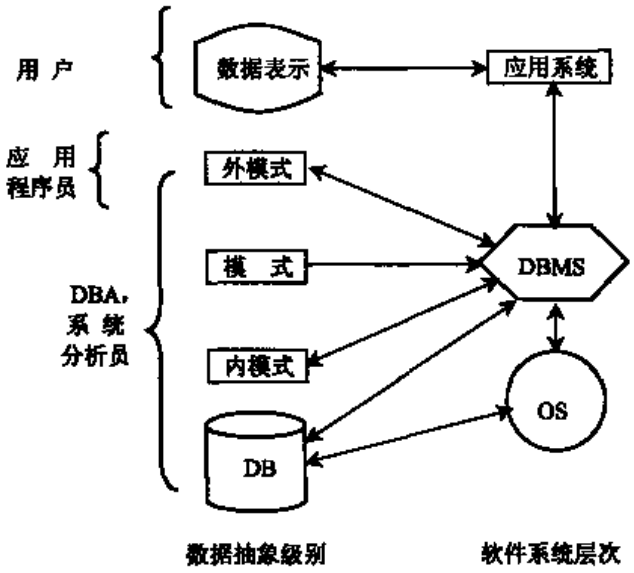


图 1.30 各种人员的数据视图

DBA 要综合各用户的应用要求，和数据库设计人员共同决定数据的存储结构和存取策略以求获得较高的存取效率和存储空间利用率。

（3）定义数据的安全性要求和完整性约束条件

DBA 的重要职责是保证数据库的安全性和完整性。因此 DBA 负责确定各个用户对数据库的存取权限、数据的保密级别和完整性约束条件。

（4）监控数据库的使用和运行

DBA 还有一个重要职责就是监视数据库系统的运行情况，及时处理运行过程中出现的问题。比如系统发生各种故障时，数据库会因此遭到不同程度的破坏，DBA 必须在最短时间内将数据库恢复到正确状态，并尽可能不影响或少影响计算机系统其他部分的正常运行。为此，DBA 要定义和实施适当的后备和恢复策略。如周期性的转储数据、维护日志文件等。有关这方面的内容将在下面做进一步讨论。

（5）数据库的改进和重组重构

DBA 还负责在系统运行期间监视系统的空间利用率、处理效率等性能指标，对运行情况进行记录、统计分析，依靠工作实践并根据实际应用环境，不断改进数据库设计。不少数据库产品都提供了对数据库运行状况进行监视和分析的实用程序，DBA 可以使用这些实用程序完成这项工作。

另外，在数据运行过程中，大量数据不断插入、删除、修改，时间一长，会影响系统的性能。因此，DBA 要定期对数据库进行重组织，以提高系统的性能。

当用户的需求增加和改变时，DBA 还要对数据库进行较大的改造，包括

修改部分设计，即数据库的重构造。

2. 系统分析员和数据库设计人员

系统分析员负责应用系统的需求分析和规范说明，要和用户及 DBA 相结合，确定系统的硬件软件配置，并参与数据库系统的概要设计。

数据库设计人员负责数据库中数据的确定、数据库各级模式的设计。数据库设计人员必须参加用户需求调查和系统分析，然后进行数据库设计。在很多情况下，数据库设计人员就由数据库管理员担任。

3. 应用程序员

应用程序员负责设计和编写应用系统的程序模块，并进行调试和安装。

4. 用户

这里用户是指最终用户（End User）。最终用户通过应用系统的用户接口使用数据库。常用的接口方式有浏览器、菜单驱动、表格操作、图形显示、报表书写等，给用户提供简明直观的数据表示。

最终用户可以分为如下三类：

（1）偶然用户。这类用户不经常访问数据库，但每次访问数据库时往往需要不同的数据库信息，这类用户一般是企业或组织机构的高中级管理人员。

（2）简单用户。数据库的多数最终用户都是简单用户。其主要工作是查询和修改数据库，一般都是通过应用程序员精心设计并具有友好界面的应用程序存取数据库。银行的职员、航空公司的机票预定工作人员、旅馆总台服务员等都属于这类用户。

（3）复杂用户。复杂用户包括工程师、科学家、经济学家、科学技术工作者等具有较高科学技术背景的人员。这类用户一般都比较熟悉数据库管理系统的各种功能，能够直接使用数据库语言访问数据库，甚至能够基于数据库管理系统的 API 编制自己的应用程序。

1.5 数据库技术的研究领域

数据库学科的研究范围是十分广泛的，概括地讲可以包括以下三个领域：

一、数据库管理系统软件的研制

DBMS 是数据库系统的基础。DBMS 的研制包括研制 DBMS 本身及以 DBMS 为核心的一组相互联系的软件系统，包括工具软件和中间件。研制的目标是提高系统的可用性、可靠性、可伸缩性；提高性能和提高用户的生产率。

DBMS 核心技术的研究和实现是三十余年来数据库领域所取得的主要成

就。DBMS 是一个基础软件系统，它提供了对数据库中的数据进行存储、检索和管理的功能。

二、数据库设计

数据库设计的主要任务是在 DBMS 的支持下，按照应用的要求，为某一部门或组织设计一个结构合理、使用方便、效率较高的数据库及其应用系统。其中主要的研究方向是数据库设计方法学和设计工具，包括数据库设计方法、设计工具和设计理论的研究，数据模型和数据建模的研究，计算机辅助数据库设计方法及其软件系统的研究，数据库设计规范和标准的研究等。

三、数据库理论

数据库理论的研究主要集中于关系的规范化理论、关系数据理论等。近年来，随着人工智能与数据库理论的结合、并行计算技术等的发展，数据库逻辑演绎和知识推理、数据库中的知识发现（Knowledge Discovery from Database，简称 KDD）、并行算法等成为新的理论研究方向。

计算机领域中其他新兴技术的发展对数据库技术产生了重大影响。数据库技术和其他计算机技术的互相结合、互相渗透，使数据库中新的技术内容层出不穷。数据库的许多概念、技术内容、应用领域，甚至某些原理都有了重大的发展和变化。建立和实现了一系列新型数据库系统，如分布式数据库系统、并行数据库系统、知识库系统、多媒体数据库系统等。它们共同构成了数据库系统大家族，使数据库技术不断地涌现新的研究方向。

本书介绍的数据库系统的基本概念、基本技术和基本知识是进一步进行上述三个领域研究和开发的基础。

1.6 小 结

本章概述了数据库的基本概念，并通过对数据管理进展情况的介绍，阐述了数据库技术产生和发展的背景，也说明了数据库系统的优点。

数据模型是数据库系统的核心和基础。本章介绍了组成数据模型的三个要素、概念模型和三种主要的数据库模型。

概念模型也称信息模型，用于信息世界的建模，E-R 模型是这类模型的典型代表，E-R 方法简单、清晰，应用十分广泛。

数据模型的发展经历了格式化数据模型（包括层次模型和网状模型）、关系模型，正在走向面向对象等非传统数据模型。由于层次数据库和网状数据库

已逐渐被关系数据库代替，本书不再用单独的章节讲解，因此本章较为详细地讲解了层次模型和网状模型，而关系模型只是简单介绍，后面会详细讲解。

数据库系统三级模式和两层映象的系统结构保证了数据库系统中能够具有较高的逻辑独立性和物理独立性。

最后介绍了数据库系统的组成，使读者了解数据库系统不仅是一个计算机系统，而是一个人-机系统，人的作用特别是 DBA 的作用尤为重要。

学习这一章应把注意力放在掌握基本概念和基本知识方面，为进一步学习下面章节打好基础。本章新概念较多，如果是刚开始学习数据库，可在学习后面章节后再回来理解和掌握这些概念。

习　题

1. 试述数据、数据库、数据库管理系统、数据库系统的概念。

2. 使用数据库系统有什么好处？

3. 试述文件系统与数据库系统的区别和联系。

4. 举出适合用文件系统而不是数据库系统的应用例子； 再举出适合用数据库系统的应用例子。

5. 试述数据库系统的特点。

6. 数据库管理系统的主要功能有哪些？

7. 试述数据模型的概念、数据模型的作用和数据模型的三个要素。

8. 试述概念模型的作用。

9. 定义并解释概念模型中以下术语:

实体，实体型，实体集，属性，码，实体联系图（E-R 图）

10. 试给出三个实际部门的 E-R 图，要求实体型之间具有一对一，一对多，多对多各种不同的联系。

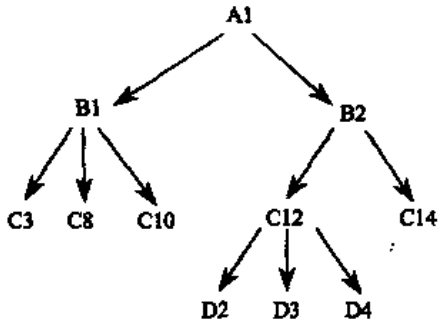
11. 试给出一个实际部门的 E-R 图，要求有三个实体型，而且三个实体型之间有多对多联系。三个实体型之间的多对多联系和三个实体型两两之间的三个多对多联系等价吗？为什么？

12. 学校中有若干系，每个系有若干班级和教研室，每个教研室有若干教员，其中有的教授和副教授每人各带若干研究生，每个班有若干学生，每个学生选修若干课程，每门课可由若干学生选修。请用 E-R 图画出此学校的概念模型。

13. 某工厂生产若干产品，每种产品由不同的零件组成，有的零件可用在不同的产品上。这些零件由不同的原材料制成，不同零件所用的材料可以相同。这些零件按所属的不同产品分别放在仓库中，原材料按照类别放在若干仓库中。请用 E-R 图画出此工厂产品，零件，材料，仓库的概念模型。

14. 试述层次模型的概念，举出三个层次模型的实例。

15. 今有一个层次数据库实例，试用子女-兄弟链接法和层次序列链接法画出它的存储结构示意图。



16. 试述网状模型的概念，举出三个网状模型的实例。

17. 试述网状、层次数据库的优缺点。

18. 试述关系模型的概念，定义并解释以下术语：

（1）关系 （2）属性 （3）域 （4）元组

（5）主码 （6）分量 （7）关系模式

19. 试述关系数据库的特点。

20. 试述数据库系统三级模式结构，这种结构的优点是什么？

21. 定义并解释以下术语：

模式、外模式、内模式、DDL、DML

22. 什么叫数据与程序的物理独立性？什么叫数据与程序的逻辑独立性？ 为什么数据库系统具有数据与程序的独立性？

23. 试述数据库系统的组成。

24. DBA 的职责是什么？

25. 系统分析员、数据库设计人员、应用程序员的职责是什么？

本章参考文献

1 Stonebraker M R. A Functional View of Data Independence. Proceedings of ACM SIGMOD Workshop on Data Description, Access and Control, May 1974

2 Michael L.Brodie and Joachim W.Schmidt(eds.). Final Report of the ANSI/X3/SPARC DBS-SG Relational Database Task Group, ACM SIGMOD Record 12, No.4, July 1982

3 ANSI/X3/SPARC Study Group on Data Base Management Systems. Interim Report, FDT(ACM SIGMOD bulletin)7.No.2, 1975

4 Information on IMS is available from IBM Corporation, Armonk, New York 10504

（文献[4]，[5]全面介绍了 IMS。）

5 McGee W C. The Information Management System IMS/VS. Part I: General Structure and

Operation. IBM Systems Journal, 6:2, June 1977

6 Kaoo D, Leben J F. IMS Programming Techniques: A Guide to Using DL/I. New York, Van Nostrand Reinhold, 1978

（本文全面介绍了 IMS 系统的语言 DL/1。）

7 Bjorner D, Lovengren H. Formalization of Database Systems and a Formal Definition of IMS. in Proceedings of VLDB, 1982

（本文把 IMS 的数据模型进行了形式化，给出了比较严格的层次数据模型的定义。）

8 Tsichritzis D C, Lochovsky F H. Hierarchical Data Base Management: A Survey. ACM Computing Surveys, 8:1, March 1976

（本文介绍了层次数据库系统的一般原理。）

9 萨师煊，王珊. 数据库系统概论（第二版）. 高等教育出版社，1991

（本书第二、第三章介绍了层次数据库系统和网状数据库系统的基本概念和一般原理。）

10 Bachman C, Williams S. A General Purpose Programming System for Random Access Memories. Proceedings of the Fall Joint Computer Conference, AFIPS, 26, 19641.11

（本文献描述的是 Charles Bachman 在第一 个商业 DBMS 的开发期间进行的网状数据模型早期研究工作。）

11 achman C. Data Structure Diagrams. Data Base (Bulletin of ACM SIGFIDET), 1:2, March 1969

12 Bachman C. The Programmer as a Navigator. CACM, 16:1, November 1973

（本文献最早提出了用数据结构简图表示数据库之间联系的思想。）

13 Bachman C. The Data Structure Set Model. In Proceedings of the ACM SIGMOD Debate on Data Model: Data Structure Set Versus Relation, 1974

（Bachman 的成就使他于 1973 年荣获了 ACM 最高荣誉——图灵奖。他在图灵奖演说中把数据库看作是基本资源，把程序设计者看作是数据库中的领航员。）

14 A Survey of Generalized Data Base Management Systems. CODASYL Systems Committee Technical Report(ACM and IAG.), May 1969

（在 1974 年支持和反对关系模型研究的讨论中， Bachman 持反对态度。）

15 Data Base Task Group of CODASYL Programming Language Committee Report, April 1971

16 Schenk H. Implementation Aspects of the DBTG Proposal. Proceedings of the IFIP Working Conference on Database Management Systems, 1974

（CODASYL 的数据库任务组(DBTG)对网状数据库系统进行了系统研究，在 1971 年的报告[15]中提出了模式 DDL、子模式 DDL 和嵌入 COBOL 的 DML，以后又发表了多篇报告。）

17 Taylor R, Frank R. CODASYL Data Base Management Systems. ACM Computing Surveys, 8:1, March 1976

18 CODASYL Data Description Language Committee Journal of Development, Canadian Government Publishing Center, 1978

（本文对 CODASYL 网状数据库管理进行了综述。）

19 Dahl R, Bubenko J. IDBD: An Interactive Design Tool for CODASYL DBTG Type Databases. In Proceedings of VLDB, 1982

（本文讨论了网状数据库语言。）

20 Cardenas A. Data Base Management Systems(Second Edition). Allyn and Bacon, 1985

（本文讨论了网状数据库设计。）

21 Olle T. The CODASYL Approach to Data Base Management. Wiley, 1978

22 Griethuysen J J Van (ed.), Concepts and Terminology for the Conceptual Schema and the Information Base. International Standards Organization Document No.ISO/TC97/SC5N695, March 1982

23 Jay-Louise Weldon. Data Base Administration. New York, N Y: Plenum Press, 1981

第二章　关系数据库

关系数据库应用数学方法来处理数据库中的数据。最早将这类方法用于数据处理的是 1962 年 CODASYL 发表的“信息代数”，之后有 1968 年 David Child 在 7090 机上实现的集合论数据结构，但系统地、严格地提出关系模型的是美国 IBM 公司的 E.F.Codd。

1970 年 E.F.Codd 在美国计算机学会会刊《Communication of the ACM》上发表的题为“A Relational Model of Data for Shared Data Banks”的论文，开创了数据库系统的新纪元。以后，他连续发表了多篇论文，奠定了关系数据库的理论基础。

20 世纪 70 年代末，关系方法的理论研究和软件系统的研制均取得了很大成果，IBM 公司的 San Jose 实验室在 IBM370 系列机上研制的关系数据库实验系统 System R 获得成功。1981 年 IBM 公司又宣布了具有 System R 全部特征的新的数据库软件产品 SQL/DS 问世。

与 System R 同期，美国加州大学柏克利分校也研制了 INGRES 关系数据库实验系统，并由 INGRES 公司发展成为 INGRES 数据库产品。

30 年来，关系数据库系统的研究取得了辉煌的成就。关系方法从实验室走向了社会，涌现出许多性能良好的商品化关系数据库管理系统（简称 RDBMS），如著名的 DB2，Oracle，Ingres，Sybase，Informix 等。数据库的应用领域迅速扩大。

本书第二、三、四、五章将集中讨论关系数据库的有关问题。其中，第二章介绍关系模型的基本概念，即关系模型的数据结构、关系操作和关系的完整性；第三章介绍关系数据库的标准语言 SQL；第四章介绍关系系统及其查询优化；第五章介绍关系数据理论。这是关系数据库的理论基础，也是关系数据库系统逻辑设计的工具。

2.1　关系模型概述

关系数据库系统是支持关系模型的数据库系统。

关系模型由关系数据结构、关系操作集合和关系完整性约束三部分组成。

一、单一的数据结构——关系

关系模型的数据结构非常单一。在关系模型中，现实世界的实体以及实体间的各种联系均用关系来表示。在用户看来，关系模型中数据的逻辑结构是一张二维表。

二、关系操作

关系模型给出了关系操作的能力，但不对 RDBMS 语言给出具体的语法要求。

关系模型中常用的关系操作包括：选择（Select）、投影（Project）、连接（Join）、除（Divide）、并（Union）、交（Intersection）、差（Difference）等查询（Query）操作和增加（Insert）、删除（Delete）、修改（Update）操作两大部分。查询的表达能力是其中最主要的部分。

关系操作的特点是集合操作方式，即操作的对象和结果都是集合。这种操作方式也称为一次一集合（set-at-a-time）的方式。相应地，非关系数据模型的数据操作方式则为一次一记录（record-at-a-time）的方式。

早期的关系操作能力通常用代数方式或逻辑方式来表示，分别称为关系代数和关系演算。关系代数是用对关系的运算来表达查询要求的方式。关系演算是用谓词来表达查询要求的方式。关系演算又可按谓词变元的基本对象是元组变量还是域变量分为元组关系演算和域关系演算。关系代数、元组关系演算和域关系演算三种语言在表达能力上是完全等价的。

关系代数、元组关系演算和域关系演算均是抽象的查询语言，这些抽象的语言与具体的 DBMS 中实现的实际语言并不完全一样。但它们能用作评估实际系统中查询语言能力的标准或基础。实际的查询语言除了提供关系代数或关系演算的功能外，还提供了许多附加功能，例如集函数、关系赋值、算术运算等。

关系语言是一种高度非过程化的语言，用户不必请求 DBA 为其建立特殊的存取路径，存取路径的选择由 DBMS 的优化机制来完成，此外，用户不必求助于循环结构就可以完成数据操作。

另外还有一种介于关系代数和关系演算之间的语言 SQL（Structurel Query Language）。SQL 不仅具有丰富的查询功能，而且具有数据定义和数据控制功能，是集查询、DDL、DML 和 DCL 于一体的关系数据语言。它充分体现了关系数据语言的特点和优点，是关系数据库的标准语言。

因此，关系数据语言可以分为三类：

关系数据语言
- 关系代数语言　例如 ISBL
- 关系演算语言
  - 元组关系演算语言　例如 APLHA，QUEL
  - 域关系演算语言　例如 QBE
- 具有关系代数和关系演算双重特点的语言　例如 SQL

这些关系数据语言的共同特点是，语言具有完备的表达能力，是非过程化的集合操作语言，功能强，能够嵌入高级语言中使用。

三、关系的三类完整性约束

关系模型允许定义三类完整性约束：实体完整性、参照完整性和用户定义的完整性。其中实体完整性和参照完整性是关系模型必须满足的完整性约束条件，应该由关系系统自动支持。用户定义的完整性是应用领域需要遵循的约束条件，体现了具体领域中的语义约束。

下面将分别介绍关系模型的三个方面。其中 2.2 节介绍关系数据结构，包括关系的形式化定义及有关概念；2.3 节介绍关系的三类完整性；2.4 节介绍关系代数；2.5 节介绍关系演算；第三章将专门介绍 SQL 语言。

2.2 关系数据结构及形式化定义

在关系模型中，无论是实体还是实体之间的联系均由单一的结构类型即关系（表）来表示。前面已经非形式化地介绍了关系模型及有关的基本概念。关系模型是建立在集合代数的基础上的，这里从集合论角度给出关系数据结构的形式化定义。

2.2.1 关系

1. 域（Domain）

定义 2.1 域是一组具有相同数据类型的值的集合。

例如，自然数、整数、实数、长度小于 25 字节的字符串集合、{0，1}、大于等于 0 且小于等于 100 的正整数等，都可以是域。

2. 笛卡尔积（Cartesian Product）

定义 2.2 给定一组域 $D_1, D_2, \cdots, D_n$，这些域中可以有相同的。$D_1, D_2, \cdots, D_n$ 的笛卡尔积为：

$$D_1 \times D_2 \times \cdots \times D_n = \{ (d_1, d_2, \cdots, d_n) \mid d_i \in D_i, \ i=1, 2, \cdots, n \}$$

其中每一个元素(d_1, d_2, …, d_n)叫作一个 n 元组(n-tuple)或简称元组(Tuple)。

元素中的每一个值 d_i 叫作一个分量(Component)。

若 D_i(i=1, 2, …, n)为有限集,其基数(Cardinal number)为 m_i(i=1, 2, …, n),则 $D_1\times D_2\times\cdots\times D_n$ 的基数 M 为:

$$M=\prod_{i=1}^{n}m_i$$

笛卡尔积可表示为一个二维表。表中的每行对应一个元组,表中的每列对应一个域。例如给出三个域:

D_1=导师集合 SUPERVISOR=张清玫,刘逸

D_2=专业集合 SPECIALITY=计算机专业,信息专业

D_3=研究生集合 POSTGRADUATE=李勇,刘晨,王敏

则 D_1, D_2, D_3 的笛卡尔积为:

$D_1\times D_2\times D_3$ = {(张清玫,计算机专业,李勇),(张清玫,计算机专业,刘晨),
(张清玫,计算机专业,王敏),(张清玫,信息专业,李勇),
(张清玫,信息专业,刘晨),(张清玫,信息专业,王敏),
(刘逸,计算机专业,李勇),(刘逸,计算机专业,刘晨),
(刘逸,计算机专业,王敏),(刘逸,信息专业,李勇),
(刘逸,信息专业,刘晨),(刘逸,信息专业,王敏)}

其中(张清玫,计算机专业,李勇)、(张清玫,计算机专业,刘晨)等都是元组。张清玫、计算机专业、李勇、刘晨等都是分量。

该笛卡尔积的基数为 2×2×3=12,也就是说,$D_1\times D_2\times D_3$ 一共有 2×2×3=12 个元组。这 12 个元组可列成一张二维表(如表 2.1)。

3. 关系(Relation)

定义 2.3 $D_1\times D_2\times\cdots\times D_n$ 的子集叫作在域 D_1, D_2, …, D_n 上的**关系**,表示为

$$R(D_1, D_2, \cdots, D_n)$$

这里 R 表示关系的名字,n 是关系的目或度(Degree)。

关系中的每个元素是关系中的元组,通常用 t 表示。

当 n=1 时,称该关系为**单元关系**(Unary relation)。

当 n=2 时,称该关系为**二元关系**(Binary relation)。

关系是笛卡尔积的有限子集,所以关系也是一个二维表,表的每行对应一个元组,表的每列对应一个域。由于域可以相同,为了加以区分,必须对每列起一个名字,称为属性(Attribute)。n 目关系必有 n 个属性。

若关系中的某一属性组的值能唯一地标识一个元组,则称该属性组为**候选码**(Candidate key)。

表 2.1 D_1，D_2，D_3 的笛卡尔积

SUPERVISOR	SPECIALITY	POSTGRADUATE
张清玫	计算机专业	李勇
张清玫	计算机专业	刘晨
张清玫	计算机专业	王敏
张清玫	信息专业	李勇
张清玫	信息专业	刘晨
张清玫	信息专业	王敏
刘逸	计算机专业	李勇
刘逸	计算机专业	刘晨
刘逸	计算机专业	王敏
刘逸	信息专业	李勇
刘逸	信息专业	刘晨
刘逸	信息专业	王敏

若一个关系有多个候选码，则选定其中一个为**主码**（Primary key）。主码的诸属性称为**主属性**（Prime attribute）。不包含在任何候选码中的属性称为**非码属性**（Non-key attribute）。 在最简单的情况下，候选码只包含一个属性。在最极端的情况下，关系模式的所有属性组是这个关系模式的候选码，称为**全码**（All-key）。

例如，可以在表 2.1 的笛卡尔积中取出一个子集来构造一个关系。由于一个研究生只师从于一个导师，学习某一个专业，所以笛卡尔积中的许多元组是无实际意义的，从中取出有实际意义的元组来构造关系。该关系的名字为 SAP，属性名就取域名，即 SUPERVISOR，SPECIALITY 和 POSTGRADUATE。则这个关系可以表示为：

SAP(SUPERVISOR，SPECIALITY，POSTGRADUATE)

假设导师与专业是一对一的，即一个导师只有一个专业；导师与研究生是一对多的，即一个导师可以带多名研究生，而一名研究生只有一个导师。这样 SAP 关系可以包含三个元组，如表 2.2 所示。

表 2.2 SAP 关系

SUPERVISOR	SPECIALITY	POSTGRADUATE
张清玫	信息专业	李勇
张清玫	信息专业	刘晨
刘逸	信息专业	王敏

假设研究生不会重名（这在实际当中是不合适的，这里只是为了举例方

便），则 POSTGRADUATE 属性的每一个值都唯一地标识了一个元组，因此可以作为 SAP 关系的主码。

关系可以有三种类型：基本关系（通常又称为基本表或基表）、查询表和视图表。基本表是实际存在的表，它是实际存储数据的逻辑表示。查询表是查询结果对应的表。视图表是由基本表或其他视图表导出的表，是虚表，不对应实际存储的数据。

按照定义 2.2，关系可以是一个无限集合。由于笛卡尔积不满足交换律，所以按照数学定义，$(d_1, d_2, \cdots, d_n) \neq (d_2, d_1, \cdots, d_n)$。当关系作为关系数据模型的数据结构时，需要给予如下的限定和扩充：

（1）无限关系在数据库系统中是无意义的。因此，限定关系数据模型中的关系必须是有限集合。

（2）通过为关系的每个列附加一个属性名的方法取消关系元组的有序性，即 $(d_1, d_2, \cdots, d_i, d_j, \cdots, d_n) = (d_1, d_2, \cdots, d_j, d_i, \cdots, d_n)$ $(i, j = 1, 2, \cdots, n)$。

因此，基本关系具有以下六条性质：

① 列是同质的（Homogeneous），即每一列中的分量是同一类型的数据，来自同一个域。

② 不同的列可出自同一个域，称其中的每一列为一个属性，不同的属性要给予不同的属性名。

例如在上面的例子中，也可以只给出两个域：

人（PERSON）=张清玫，刘逸，李勇，刘晨，王敏

专业（SPECIALITY）=计算机专业，信息专业

SAP 关系的导师属性和研究生属性都从 PERSON 域中取值。为了避免混淆，必须给这两个属性取不同的属性名，而不能直接使用域名。例如定义导师属性名为 SUPERVISOR-PERSON（或 SUPERVISOR），研究生属性名为 POSTGRADUATE-PERSON（或 POSTGRADUATE）。

③ 列的顺序无所谓，即列的次序可以任意交换。

由于列顺序是无关紧要的，因此在许多实际关系数据库产品中（例如 Oracle），增加新属性时，永远是插至最后一列。

④ 任意两个元组不能完全相同。

⑤ 行的顺序无所谓，即行的次序可以任意交换。

⑥ 分量必须取原子值，即每一个分量都必须是不可分的数据项。

注意：在许多实际关系数据库产品中，基本表并不完全具有这六条性质，例如，有的数据库产品（如 FoxPro）仍然区分了属性顺序和元组的顺序；许多关系数据库产品中，例如 Oracle，FoxPro 等，它们都允许关系表中存在两

个完全相同的元组，除非用户特别定义了相应的约束条件。

关系模型要求关系必须是规范化的，即要求关系模式必须满足一定的规范条件。这些规范条件中最基本的一条就是，关系的每一个分量必须是一个不可分的数据项。规范化的关系简称为范式（Normal Form）。例如，表 2.3 虽然很好地表达了导师与研究生之间的一对多关系，但由于 POSTGRADUATE 分量取了两个值，不符合规范化的要求，因此这样的关系在数据库中是不允许的。

表 2.3 非规范化关系

SUPERVISOR	SPECIALITY	POSTGRADUATE	
		PG1	PG2
张清玫	信息专业	李勇	刘晨
刘逸	信息专业	王敏	

2.2.2 关系模式

在数据库中要区分型和值。关系数据库中，关系模式是型，关系是值。关系模式是对关系的描述，那么一个关系需要描述哪些方面呢？

首先，应该知道，关系实质上是一张二维表，表的每一行为一个元组，每一列为一个属性。一个元组就是该关系所涉及的属性集的笛卡尔积的一个元素。关系是元组的集合，因此关系模式必须指出这个元组集合的结构，即它由哪些属性构成，这些属性来自哪些域，以及属性与域之间的映象关系。

其次，一个关系通常是由赋予它的元组语义来确定的。元组语义实质上是一个 n 目谓词（n 是属性集中属性的个数）。凡使该 n 目谓词为真的笛卡尔积中的元素（或者说凡符合元组语义的那部分元素）的全体就构成了该关系模式的关系。

现实世界随着时间在不断地变化，因而在不同的时刻，关系模式的关系也会有所变化。但是，现实世界的许多已有事实限定了关系模式所有可能的关系必须满足一定的完整性约束条件。这些约束或者通过对属性取值范围的限定，例如职工年龄小于 65 岁（65 岁以后必须退休），或者通过属性值间的相互关连（主要体现于值的相等与否）反映出来。关系模式应当刻划出这些完整性约束条件。

因此一个关系模式应当是一个 5 元组。

定义 2.4 关系的描述称为**关系模式**（Relation Schema）。它可以形式化地表示为：

$$R(U, D, \mathrm{dom}, F)$$

其中 R 为关系名，U 为组成该关系的属性名集合，D 为属性组 U 中属性所来

自的域，dom 为属性向域的映象集合，F 为属性间数据的依赖关系集合。

属性间的数据依赖将在第五章讨论，本章中关系模式仅涉及关系名、各属性名、域名、属性向域的映象四部分。

例如，在上面例子中，由于导师和研究生出自同一个域——人，所以要取不同的属性名，并在模式中定义属性向域的映象，即说明它们分别出自哪个域，如：

dom（SUPERVISOR-PERSON）= dom（POSTGRADUATE-PERSON）=PERSON

关系模式通常可以简记为

$R(U)$

或 $R(A_1, A_2, \cdots, A_n)$

其中 R 为关系名，$A_1, A_2, \cdots, A_n$ 为属性名。而域名及属性向域的映象常常直接说明为属性的类型、长度。

关系是关系模式在某一时刻的状态或内容。关系模式是静态的、稳定的，而关系是动态的、随时间不断变化的，因为关系操作在不断地更新着数据库中的数据。但在实际当中，人们常常把关系模式和关系都称为关系，这不难从上下文中加以区别。

2.2.3 关系数据库

在关系模型中，实体以及实体间的联系都是用关系来表示的。例如导师实体、研究生实体、导师与研究生之间的一对多联系都可以分别用一个关系来表示。在一个给定的应用领域中，所有实体及实体之间联系的关系的集合构成一个关系数据库。

关系数据库也有型和值之分。关系数据库的型也称为关系数据库模式，是对关系数据库的描述，它包括若干域的定义以及在这些域上定义的若干关系模式。关系数据库的值是这些关系模式在某一时刻对应的关系的集合，通常就称为关系数据库。

2.3 关系的完整性

关系模型的完整性规则是对关系的某种约束条件。关系模型中可以有三类完整性约束：实体完整性、参照完整性和用户定义的完整性。其中实体完整性和参照完整性是关系模型必须满足的完整性约束条件，被称作是关系的两个不变性，应该由关系系统自动支持。

一、实体完整性（Entity Integrity）

规则 2.1 实体完整性规则 若属性 A 是基本关系 R 的主属性，则属性 A 不能取空值。

例如在关系“SAP(SUPERVISOR，SPECIALITY，POSTGRADUATE)”中，“研究生姓名 POSTGRADUATE”属性为主码（假设研究生不会重名），则“研究生姓名”不能取空值。

实体完整性规则规定基本关系的所有主属性都不能取空值，而不仅是主码整体不能取空值。例如学生选课关系“选修（学号，课程号，成绩）”中，“学号、课程号”为主码，则“学号”和“课程号”两个属性都不能取空值。

对于实体完整性规则说明如下：

（1）实体完整性规则是针对基本关系而言的。一个基本表通常对应现实世界的一个实体集。例如学生关系对应于学生的集合。

（2）现实世界中的实体是可区分的，即它们具有某种唯一性标识。

（3）相应地，关系模型中以主码作为唯一性标识。

（4）主码中的属性即主属性不能取空值。所谓空值就是“不知道”或“无意义”的值。如果主属性取空值，就说明存在某个不可标识的实体，即存在不可区分的实体，这与第（2）点相矛盾，因此这个规则称为实体完整性。

二、参照完整性（Referential Integrity）

现实世界中的实体之间往往存在某种联系，在关系模型中实体及实体间的联系都是用关系来描述的。这样就自然存在着关系与关系间的引用。先来看三个例子。

例 1 学生实体和专业实体可以用下面的关系表示，其中主码用下划线标识：

学生（<u>学号</u>，姓名，性别，专业号，年龄）

专业（<u>专业号</u>，专业名）

这两个关系之间存在着属性的引用，即学生关系引用了专业关系的主码“专业号”。显然，学生关系中的“专业号”值必须是确实存在的专业的专业号，即专业关系中有该专业的记录。这也就是说，学生关系中的某个属性的取值需要参照专业关系的属性取值。

例 2 学生、课程、学生与课程之间的多对多联系可以如下三个关系表示：

学生（<u>学号</u>，姓名，性别，专业号，年龄）

课程（<u>课程号</u>，课程名，学分）

选修（<u>学号</u>，<u>课程号</u>，成绩）

这三个关系之间也存在着属性的引用，即选修关系引用了学生关系的主码“学号”和课程关系的主码“课程号”。同样，选修关系中的“学号”值必须是确实存在的学生的学号，即学生关系中有该学生的记录；选修关系中的“课程号”值也必须是确实存在的课程的课程号，即课程关系中有该课程的记录。换句话说，选修关系中某些属性的取值需要参照其他关系的属性取值。

不仅两个或两个以上的关系间可以存在引用关系，同一关系内部属性间也可能存在引用关系。

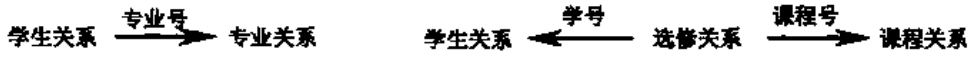
例 3 在关系学生 2（学号，姓名，性别，专业号，年龄，班长）中，“学号”属性是主码，“班长”属性表示该学生所在班级的班长的学号，它引用了本关系“学号”属性，即“班长”必须是确实存在的学生的学号。

定义 2.5 设 F 是基本关系 R 的一个或一组属性，但不是关系 R 的码。如果 F 与基本关系 S 的主码 K_s 相对应，则称 F 是基本关系 R 的外码(Foreign Key)，并称基本关系 R 为参照关系（Referencing Relation），基本关系 S 为被参照关系（Referenced Relation）或目标关系（Target Relation）。关系 R 和 S 不一定是不同的关系。

显然，目标关系 S 的主码 K_s 和参照关系的外码 F 必须定义在同一个（或一组）域上。

在例 1 中，学生关系的“专业号”属性与专业关系的主码“专业号”相对应，因此“专业号”属性是学生关系的外码。这里专业关系是被参照关系，学生关系为参照关系。如图 2.1(a)所示。

在例 2 中，选修关系的“学号”属性与学生关系的主码“学号”相对应，“课程号”属性与课程关系的主码“课程号”相对应，因此“学号”和“课程号”属性是选修关系的外码。这里学生关系和课程关系均为被参照关系，选修关系为参照关系。如图 2.1(b)所示。



(a) (b)

图 2.1 关系的参照图

在例 3 中，“班长”属性与本身的主码“学号”属性相对应，因此“班长”是外码。这里学生 2 关系既是参照关系也是被参照关系。

需要指出的是，外码并不一定要与相应的主码同名（如例 3）。不过，在实际应用当中，为了便于识别，当外码与相应的主码属于不同关系时，往往给它们取相同的名字。

参照完整性规则就是定义外码与主码之间的引用规则。

规则 2.2 参照完整性规则 若属性（或属性组）F 是基本关系 R 的外码，它与基本关系 S 的主码 K_s 相对应（基本关系 R 和 S 不一定是不同的关系），则对于 R 中每个元组在 F 上的值必须为：

- 或者取空值（F 的每个属性值均为空值）；
- 或者等于 S 中某个元组的主码值。

例如，对于例 1，学生关系中每个元组的“专业号”属性只能取下面两类值：

（1）空值，表示尚未给该学生分配专业；

（2）非空值，这时该值必须是专业关系中某个元组的“专业号”值，表示该学生不可能分配到一个不存在的专业中。即被参照关系“专业”中一定存在一个元组，它的主码值等于该参照关系“学生”中的外码值。

对于例 2，按照参照完整性规则，“学号”和“课程号”属性也可以取两类值：空值或目标关系中已经存在的值。但由于“学号”和“课程号”是选修关系中的主属性，按照实体完整性规则，它们均不能取空值。所以选修关系中的“学号”和“课程号”属性实际上只能取相应被参照关系中已经存在的主码值。

参照完整性规则中，R 与 S 可以是同一个关系。例如对于例 3，按照参照完整性规则，“班长”属性值可以取两类值：

（1）空值，表示该学生所在班级尚未选出班长；

（2）非空值，这时该值必须是本关系中某个元组的学号值。

三、用户定义的完整性（User-defined Integrity）

任何关系数据库系统都应该支持实体完整性和参照完整性。除此之外，不同的关系数据库系统根据其应用环境的不同，往往还需要一些特殊的约束条件，用户定义的完整性就是针对某一具体关系数据库的约束条件。它反映某一具体应用所涉及的数据必须满足的语义要求。例如某个属性必须取唯一值、某些属性值之间应满足一定的函数关系、某个属性的取值范围在 0～100 之间等。关系模型应提供定义和检验这类完整性的机制，以便用统一的系统的方法处理它们，而不要由应用程序承担这一功能。

2.4 关 系 代 数

关系代数是一种抽象的查询语言，是关系数据操纵语言的一种传统表达方式，它是用对关系的运算来表达查询的。

任何一种运算都是将一定的运算符作用于一定的运算对象上，得到预期的运算结果。所以运算对象、运算符、运算结果是运算的三大要素。

关系代数的运算对象是关系，运算结果亦为关系。关系代数用到的运算符包括四类：集合运算符、专门的关系运算符、算术比较符和逻辑运算符，如表2.4所示。

表2.4 关系代数运算符

运算符		含义	运算符		含义
集合运算符	∪	并	比较运算符	>	大于
	−	差		≥	大于等于
	∩	交		<	小于
				≤	小于等于
				=	等于
				≠	不等于
专门的关系运算符	×	广义笛卡尔积	逻辑运算符	¬	非
	σ	选择		∧	与
	π	投影		∨	或
	⋈	连接			
	÷	除			

关系代数的运算按运算符的不同可分为传统的集合运算和专门的关系运算两类。

其中传统的集合运算将关系看成元组的集合，其运算是从关系的“水平”方向即行的角度来进行。而专门的关系运算不仅涉及行而且涉及列。比较运算符和逻辑运算符是用来辅助专门的关系运算符进行操作的。

2.4.1 传统的集合运算

传统的集合运算是二目运算，包括并、差、交、广义笛卡尔积四种运算。

设关系 R 和关系 S 具有相同的目 n（即两个关系都有 n 个属性），且相应的属性取自同一个域，则可以定义并、差、交运算如下：

1. 并（Union）

关系 R 与关系 S 的并记作：

$$R \cup S = \{ t \mid t \in R \vee t \in S \}$$

其结果仍为 n 目关系，由属于 R 或属于 S 的元组组成。

2. 差（Difference）

关系 R 与关系 S 的差记作：

$$R - S = \{ t \mid t \in R \wedge t \notin S \}$$

其结果关系仍为 n 目关系，由属于 R 而不属于 S 的所有元组组成。

3. 交（Intersection）

关系 R 与关系 S 的交记作：

$$R\cap S=\{t|t\in R\wedge t\in S\}$$

其结果关系仍为 n 目关系，由既属于 R 又属于 S 的元组组成。关系的交可以用差来表示，即 $R\cap S=R-(R-S)$。

4. 广义笛卡尔积（Extended Cartesian Product）

两个分别为 n 目和 m 目的关系 R 和 S 的广义笛卡尔积是一个（$n+m$）列的元组的集合。元组的前 n 列是关系 R 的一个元组，后 m 列是关系 S 的一个元组。若 R 有 k_1 个元组，S 有 k_2 个元组，则关系 R 和关系 S 的广义笛卡尔积有 $k_1\times k_2$ 个元组。记作：

$$R\times S=\{\overset{\frown}{t_r t_s}|t_r\in R\ \wedge\ t_s\in S\}$$

图 2.2(a)、图 2.2(b)分别为具有三个属性列的关系 R，S。图 2.2(c)为关系 R 与 S 的并。图 2.2(d)为关系 R 与 S 的交。图 2.2(e)为关系 R 和 S 的差。图 2.2(f)为关系 R 和 S 的广义笛卡尔积。

R

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_2	c_1

(a)

S

A	B	C
a_1	b_2	c_2
a_1	b_3	c_2
a_2	b_2	c_1

(b)

$R\cup S$

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_2	c_1
a_1	b_3	c_2

(c)

$R\cap S$

A	B	C
a_1	b_2	c_2
a_2	b_2	c_1

(d)

$R-S$

A	B	C
a_1	b_1	c_1

(e)

$R\times S$

A	B	C	A	B	C
a_1	b_2	c_1	a_1	b_2	c_2
a_1	b_2	c_1	a_1	b_3	c_2
a_1	b_2	c_1	a_2	b_2	c_1
a_1	b_2	c_2	a_1	b_2	c_2
a_1	b_2	c_2	a_1	b_3	c_2
a_1	b_2	c_2	a_2	b_2	c_1
a_2	b_2	c_1	a_1	b_2	c_2
a_2	b_2	c_1	a_1	b_3	c_2
a_2	b_2	c_1	a_2	b_2	c_1

(f)

图 2.2 传统集合运算举例

2.4.2 专门的关系运算

专门的关系运算包括选择、投影、连接、除等。为了叙述上的方便，先引入几个记号。

（1）设关系模式为 $R(A_1, A_2, \cdots, A_n)$。它的一个关系设为 R。$t \in R$ 表示 t 是 R 的一个元组。$t[A_i]$则表示元组 t 中相应于属性 A_i 的一个分量。

（2）若 $A=\{A_{i1}, A_{i2}, \cdots, A_{ik}\}$，其中 $A_{i1}, A_{i2}, \cdots, A_{ik}$ 是 $A_1, A_2, \cdots, A_n$ 中的一部分，则 A 称为属性列或域列。$t[A]=(t[A_{i1}], t[A_{i2}], \cdots, t[A_{ik}])$表示元组 t 在属性列 A 上诸分量的集合。$\bar{A}$ 则表示$\{A_1, A_2, \cdots, A_n\}$中去掉$\{A_{i1}, A_{i2}, \cdots, A_{ik}\}$后剩余的属性组。

（3）R 为 n 目关系，S 为 m 目关系。$t_r \in R$，$t_s \in S$，$\widehat{t_r t_s}$ 称为元组的连接（Concatenation）。它是一个 $n+m$ 列的元组，前 n 个分量为 R 中的一个 n 元组，后 m 个分量为 S 中的一个 m 元组。

（4）给定一个关系 $R(X, Z)$，X 和 Z 为属性组。定义，当 $t[X]=x$ 时，x 在 R 中的象集（Images Set）为：

$$Z_x=\{t[Z] | t \in R, t[X]=x\}$$

它表示 R 中属性组 X 上值为 x 的诸元组在 Z 上分量的集合。

下面给出这些关系运算的定义：

1. 选择（Selection）

选择又称为限制（Restriction）。它是在关系 R 中选择满足给定条件的诸元组，记作：

$$\sigma_F(R) = \{t | t \in R \wedge F(t)='真'\}$$

其中 F 表示选择条件，它是一个逻辑表达式，取逻辑值“真”或“假”。

逻辑表达式 F 由逻辑运算符$\neg$，$\wedge$，$\vee$连接各算术表达式组成。算术表达式的基本形式为：

$$X_1 \theta Y_1$$

其中 θ 表示比较运算符，它可以是$>$，$\geqslant$，$<$，$\leqslant$，$=$或$\neq$。X_1，Y_1 等是属性名，或为常量，或为简单函数；属性名也可以用它的序号来代替。

选择运算实际上是从关系 R 中选取使逻辑表达式 F 为真的元组。这是从行的角度进行的运算。

设有一个学生-课程数据库，包括学生关系 Student、课程关系 Course 和选修关系 SC，如图 2.3 所示。下面的许多例子将对这三个关系进行运算。

Student

学号 Sno	姓名 Sname	性别 Ssex	年龄 Sage	所在系 Sdept
95001	李勇	男	20	CS
95002	刘晨	女	19	IS
95003	王敏	女	18	MA
95004	张立	男	19	IS

(a)

Course

课程号 Cno	课程名 Cname	先行课 Cpno	学分 Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL 语言	6	4

(b)

SC

学号 Sno	课程号 Cno	成绩 Grade
95001	1	92
95001	2	85
95001	3	88
95002	2	90
95002	3	80

(c)

图 2.3 学生-课程数据库

例 1 查询信息系（IS 系）全体学生

$\sigma_{Sdept = 'IS'}(Student)$

或 $\sigma_{5 = 'IS'}(Student)$

其中下角标“5”为 Sdept 的属性序号。结果如图 2.4(a)所示。

例 2 查询年龄小于 20 岁的学生

$$\sigma_{Sage<20}(Student)$$

或 $$\sigma_{4<20}(Student)$$

结果如图 2.4(b)所示。

Sno	Sname	Ssex	Sage	Sdept
95002	刘晨	女	19	IS
95004	张立	男	19	IS

(a)

Sno	Sname	Ssex	Sage	Sdept
95002	刘晨	女	19	IS
95003	王敏	女	18	MA
95004	张立	男	19	IS

(b)

图 2.4 选择运算举例

2. 投影（Projection）

关系 R 上的投影是从 R 中选择出若干属性列组成新的关系。记作：

$$\pi_A(R) = \{ t[A] \mid t \in R \}$$

其中 A 为 R 中的属性列。

投影操作是从列的角度进行的运算。

例 3 查询学生的姓名和所在系，即求 Student 关系在学生姓名和所在系两个属性上的投影。

$$\pi_{Sname,\ Sdept}(Student)$$

或 $$\pi_{2,\ 5}(Student)$$

结果如图 2.5(a)。

投影之后不仅取消了原关系中的某些列，而且还可能取消某些元组，因为取消了某些属性列后，就可能出现重复行，应取消这些完全相同的行。

例 4 查询学生关系 Student 中都有哪些系，即查询关系 Student 在所在系属性上的投影。

$$\pi_{Sdept}(Student)$$

结果如图 2.5(b)。Student 关系原来有四个元组，而投影结果取消了重复的 IS 元组，因此只有三个元组。

Sname	Sdept
李勇	CS
刘晨	IS
王敏	MA
张立	IS

(a)

Sdept
CS
IS
MA

(b)

图 2.5 投影运算举例

3. 连接（Join）

连接也称为 θ 连接。它是从两个关系的笛卡尔积中选取属性间满足一定条件的元组。记作：

$$R \underset{A\theta B}{\bowtie} S = \{ \widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[A]\ \theta\ t_s[B] \}$$

其中 A 和 B 分别为 R 和 S 上度数相等且可比的属性组。θ 是比较运算符。连接运算从 R 和 S 的广义笛卡尔积 $R\times S$ 中选取（R 关系）在 A 属性组上的值与（S 关系）在 B 属性组上值满足比较关系 θ 的元组。

连接运算中有两种最为重要也最为常用的连接，一种是等值连接（equijoin），另一种是自然连接（Natural join）。

θ 为“＝”的连接运算称为**等值连接**。它是从关系 R 与 S 的广义笛卡尔积中选取 A，B 属性值相等的那些元组，即等值连接为：

$$R \underset{A=B}{\bowtie} S = \{ \widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[A] = t_s[B] \}$$

自然连接（Natural join）是一种特殊的等值连接，它要求两个关系中进行比较的分量必须是相同的属性组，并且在结果中把重复的属性列去掉。即若 R 和 S 具有相同的属性组 B，则自然连接可记作：

$$R \bowtie S = \{ \widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[B] = t_s[B] \}$$

一般的连接操作是从行的角度进行运算。但自然连接还需要取消重复列，所以是同时从行和列的角度进行运算。

例 5 设图 2.6(a)和(b)分别为关系 R 和关系 S，图 2.6(c) 为 $R \underset{C<E}{\bowtie} S$ 的结果，图 2.6(d) 为等值连接 $R \underset{R.B=S.B}{\bowtie} S$的结果，图 2.6(e)为自然连接 $R \bowtie S$ 的结果。

R

A	B	C
a_1	b_1	5
a_1	b_2	6
a_2	b_3	8
a_2	b_4	12

(a)

S

B	E
b_1	3
b_2	7
b_3	10
b_3	2
b_5	2

(b)

$R \underset{C<E}{\bowtie} S$

A	$R.B$	C	$S.B$	E
a_1	b_1	5	b_2	7
a_1	b_1	5	b_3	10
a_1	b_2	6	b_2	7
a_1	b_2	6	b_3	10
a_2	b_3	8	b_3	10

(c)

A	$R.B$	C	$S.B$	E
a_1	b_1	5	b_1	3
a_1	b_2	6	b_2	7
a_2	b_3	8	b_3	10
a_2	b_3	8	b_5	2

(d)

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2

(e)

图 2.6 连接运算举例

4. 除（Division）

给定关系 $R(X, Y)$ 和 $S(Y, Z)$，其中 X，Y，Z 为属性组。R 中的 Y 与 S 中的 Y 可以有不同的属性名，但必须出自相同的域集。R 与 S 的除运算得到一个新的关系 $P(X)$，P 是 R 中满足下列条件的元组在 X 属性列上的投影：元组在 X 上分量值 x 的象集 Y_x 包含 S 在 Y 上投影的集合。记作：

$$R \div S = \{t_r[X] \mid t_r \in R \wedge \pi_y(S) \subseteq Y_x\}$$

其中 Y_x 为 x 在 R 中的象集，$x = t_t[X]$。

除操作是同时从行和列角度进行运算。

例 6 设关系 R，S 分别为图 2.7 中的(a)和(b)，$R \div S$ 的结果为图 2.7(c)。

在关系 R 中，A 可以取四个值$\{a_1, a_2, a_3, a_4\}$。其中：

a_1 的象集为$\{(b_1, c_2), (b_2, c_3), (b_2, c_1)\}$

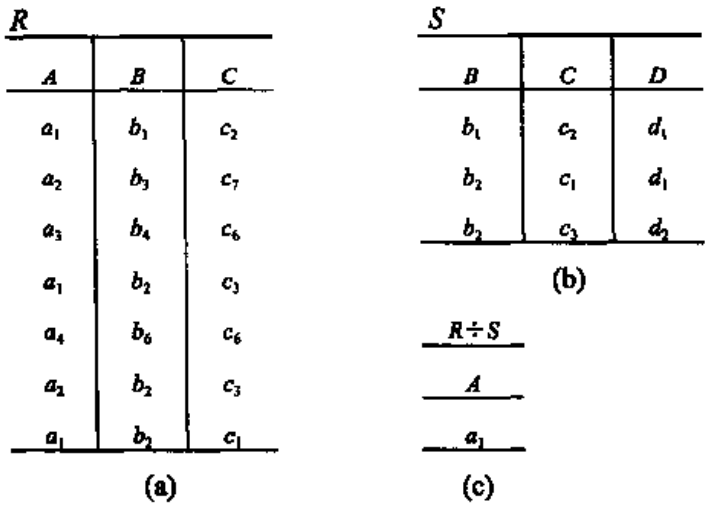
a_2 的象集为$\{(b_3, c_7), (b_2, c_3)\}$

a_3 的象集为$\{(b_4, c_6)\}$

a_4 的象集为$\{(b_6, c_6)\}$

S 在(B, C)上的投影为$\{(b_1, c_2), (b_2, c_1), (b_2, c_3)\}$

显然只有 a_1 的象集$(B, C)_{a1}$ 包含了 S 在(B, C)属性组上的投影，所以 $R \div S = \{a_1\}$



R

A	*B*	*C*
a_1	b_1	c_2
a_2	b_3	c_7
a_3	b_4	c_6
a_1	b_2	c_3
a_4	b_6	c_6
a_2	b_2	c_3
a_1	b_2	c_1

(a)

S

B	*C*	*D*
b_1	c_2	d_1
b_2	c_1	d_1
b_2	c_3	d_2

(b)

$R \div S$

A
a_1

(c)

图 2.7 除运算举例

下面再以学生-课程数据库为例，给出几个综合应用多种关系代数运算进行查询的例子。

例 7 查询至少选修 1 号课程和 3 号课程的学生号码。

首先建立一个临时关系 *K*：

Cno
1
3

然后求：$\pi_{Sno,\ Cno}(SC) \div K$

结果为{95001}。

求解过程与例 6 类似，先对 SC 关系在 Sno 和 Cno 属性上投影，然后对其中每个元组逐一求出每一学生的象集，并依次检查这些象集是否包含 K。

例 8 查询选修了 2 号课程的学生的学号。

$$\pi_{Sno}(\sigma_{Cno='2'}(SC)) = \{95001, 95002\}$$

例 9 查询至少选修了一门其直接先行课为 5 号课程的学生姓名。

$$\pi_{Sname}(\sigma_{Cpno='5'}(Course) \bowtie SC \bowtie \pi_{Sno,\ Sname}(Student))$$

或

$$\pi_{Sname}(\pi_{Sno}(\sigma_{Cpno='5'}(Course) \bowtie SC) \bowtie \pi_{Sno,\ Sname}(Student))$$

例 10 查询选修了全部课程的学生号码和姓名。

$$\pi_{Sno,\ Cno}(SC) \div \pi_{Cno}(Course) \bowtie \pi_{Sno,\ Sname}(Student)$$

本节介绍了 8 种关系代数运算，其中并、差、笛卡尔积、投影和选择 5 种运算为基本的运算。其他 3 种运算，即交、连接和除，均可以用这 5 种基本运算来表达。引进它们并不增加语言的能力，但可以简化表达。

关系代数中，这些运算经有限次复合后形成的式子称为关系代数表达式。

关系代数语言中比较典型的例子是查询语言 ISBL（Information System Base Language）。ISBL 语言由 IBM United Kingdom 研究中心研制，用于 PRTV（Peterlee Relational Test Vehicle）实验系统。

2.5 关 系 演 算

关系演算是以数理逻辑中的谓词演算为基础的。按谓词变元的不同，关系演算可分为元组关系演算和域关系演算。本节先介绍元组关系演算，然后介绍域关系演算。在讲解时都是先介绍一种具体的语言，再讲解抽象的语言。

2.5.1 元组关系演算语言 ALPHA

元组关系演算以元组变量作为谓词变元的基本对象。一种典型的元组关系演算语言是 E.F.Codd 提出的 ALPHA 语言。这一语言虽然没有实际实现，但关系数据库管理系统 INGRES 所用的 QUEL 语言是参照 ALPHA 语言研制的，与 ALPHA 十分类似。

ALPHA 语言主要有 GET，PUT，HOLD，UPDATE，DELETE，DROP 6 条语句，语句的基本格式是：

操作语句 工作空间名（表达式）：操作条件

其中表达式用于指定语句的操作对象，它可以是关系名或（和）属性名，一条语句可以同时操作多个关系或多个属性。操作条件是一个逻辑表达式，用于将操作结果限定在满足条件的元组中，操作条件可以为空。除此之外，还可以在基本格式的基础上加上排序要求，定额要求等。

一、检索操作

检索操作用 GET 语句实现。

（1）简单检索（即不带条件的检索）

例 1 查询所有被选修的课程号码。

```
GET  W  (SC.Cno)
```

W 为工作空间名。这里条件为空，表示没有限定条件。

例 2 查询所有学生的数据。

```
GET  W  (Student)
```

（2）限定的检索（即带条件的检索）

例 3 查询信息系(IS)中年龄小于 20 岁的学生的学号和年龄。

```
GET W (Student.Sno，Student.Sage): Student.Sdept='IS'∧Student.Sage<20
```

（3）带排序的检索

例 4 查询计算机科学系(CS)学生的学号、年龄，结果按年龄降序排序。

```
GET W (Student.Sno，Student.Sage): Student.Sdept='CS' DOWN Student.Sage
```

DOWN 表示降序排序。

（4）带定额的检索

例 5 取出一个信息系学生的学号。

```
GET W (1) (Student.Sno): Student.Sdept='IS'
```

所谓带定额的检索是指规定了检索出元组的个数，方法是在 W 后括号中加上定额数量。

排序和定额可以一起使用。

例 6 查询信息系年龄最大的三个学生的学号及其年龄，结果按年龄降序排序。

```
GET W (3) (Student.Sno，Student.Sage): Student.Sdept='IS' DOWN Student.Sage
```

（5）用元组变量的检索

前面已讲到，元组关系演算是以元组变量作为谓词变元的基本对象。元组变量是在某一关系范围内变化（所以也称为范围变量 Range Variable），一个关系可以设多个元组变量。

元组变量主要有两方面的用途：

① 简化关系名。如果关系的名字很长，使用起来就会感到不方便，这时我们可以设一个较短名字的元组变量来代替关系名。

② 操作条件中使用量词时必须用元组变量。

例 7 查询信息系学生的名字。

```
RANGE Student X
GET W (X.Sname): X.Sdept='IS'
```

ALPHA 语言用 RANGE 来说明元组变量。本例中 X 是关系 Student 上的元组变量，用途是简化关系名，即用 X 代表 Student。

（6）用存在量词的检索

操作条件中使用量词时必须用元组变量。

例 8 查询选修 2 号课程的学生名字。

```
RANGE SC X
GET W (Student.Sname): ∃X(X.Sno=Student.Sno∧X.Cno='2')
```

例 9 查询选修了这样课程的学生学号，其直接先行课是 6 号课程。

```
RANGE Course CX
```

GET W (SC.Sno): ∃CX (CX.Cno=SC.Cno∧CX.Pcno='6')

例 10 查询至少选修一门其先行课为 6 号课程的学生名字。

RANGE Course CX
SC SCX
GET W (Student.Sname): ∃SCX (SCX.Sno=Student.Sno∧
∃CX (CX.Cno=SCX.Cno∧CX.Pcno='6'))

在例 10 中的元组关系演算公式可以变换为前束范式（Prenex normal form）的形式：

GET W (Student.Sname): ∃SCX∃CX (SCX.Sno=Student.Sno∧
CX.Cno=SCX.Cno∧CX.Pcno='6')

例 8、例 9、例 10 中的元组变量都是为存在量词而设的。其中例 10 需要对两个关系作用存在量词，所以设了两个元组变量。

（7）带有多个关系的表达式的检索

上面所举的各个例子中，虽然查询时可能会涉及多个关系，即公式中可能涉及多个关系，但查询结果表达式中只有一个关系。实际上表达式中是可以有多个关系的。

例 11 查询成绩为 90 分以上的学生名字与课程名字。

本查询所要求的结果是学生名字和课程名字，分别在 Student 和 Course 两个关系中。

RANGE SC SCX
GET W (Student.Sname，Course.Cname): ∃SCX (SCX.Grade≥90∧
SCX.Sno=Student.Sno∧Course.Cno=SCX.Cno)

（8）用全称量词的检索

例 12 查询不选 1 号课程的学生名字。

RANGE SC SCX
GET W (Student.Sname): ∀SCX (SCX.Sno≠Student.Sno∨SCX.Cno≠'1')

本例也可以用存在量词来表示：

RANGE SC SCX
GET W (Student.Sname): ¬∃SCX (SCX.Sno=Student.Sno∧SCX.Cno='1')

（9）用两种量词的检索

例 13 查询选修了全部课程的学生姓名。

RANGE Course CX
SC SCX
GET W (Student.Sname): ∀CX∃SCX (SCX.Sno=Student.Sno∧
SCX.Cno=CX.Cno)

（10）用蕴函（Implication）的检索

例 14 查询最少选修了 95002 学生所选课程的学生学号。

本例题的求解思路是，对 Course 中的所有课程，依次检查每一门课程，看 95002 是否选修了该课程，如果选修了，则再看某一个学生是否也选修了该门课。如果对于 95002 所选的每门课程该学生都选修了，则该学生为满足要求的学生。把所有这样的学生全都找出来即完成了本题。

```
RANGE  Couse  CX
       SC     SCX  (注意，这里 SC 设了两个元组变量)
       SC     SCY
GET  W  (Student.Sno):  ∀ CX(∃SCX (SCX.Sno='95002'∧SCX.Cno=CX.Cno)
                              ⇨ ∃SCY(SCY.Sno=Student.Sno∧
                                SCY.Cno= CX.Cno))
```

（11）集函数

用户在使用查询语言时，经常要作一些简单的计算，例如要求符合某一查询要求的元组数，求某个关系中所有元组在某属性上的值的总和或平均值等。为了方便用户，关系数据语言中建立了有关这类运算的标准函数库供用户选用。这类函数通常称为集函数（Aggregation function）或内部函数（Build-in function）。关系演算中提供了 COUNT，TOTAL，MAX，MIN，AVG 等集函数，其含义如表 2.5 所示。

表 2.5 关系演算中的集函数

函 数 名	功 能
COUNT	对元组计数
TOTAL	求 总 和
MAX	求最大值
MIN	求最小值
AVG	求平均值

例 15 查询学生所在系的数目。

```
GET  W  (COUNT(Student.Sdept))
```

COUNT 函数在计数时会自动排除重复值。

例 16 查询信息系学生的平均年龄。

```
GET  W  (AVG(Student.Sage): Student.Sdept='IS')
```

二、更新操作

（1）修改操作

修改操作用 UPDATE 语句实现。其步骤是：

① 首先用 HOLD 语句将要修改的元组从数据库中读到工作空间中；

② 然后用宿主语言修改工作空间中元组的属性；

③ 最后用 UPDATE 语句将修改后的元组送回数据库中。

需要注意的是，单纯检索数据使用 GET 语句即可，但为修改数据而读元组时必须使用 HOLD 语句，HOLD 语句是带上并发控制的 GET 语句。有关并发控制的概念将在第八章中详细介绍。

例 17 把 95007 学生从计算机科学系转到信息系。

```
HOLD  W  (Student.Sno,  Student.Sdetp): Student.Sno='95007'
                              (从 Student 关系中读出 95007 学生的数据)
MOVE  'IS' TO W.Sdept         (用宿主语言进行修改)
UPDATE W                      (把修改后的元组送回 Student 关系)
```

在该例中用 HOLD 语句来读 95007 的数据，而不是用 GET 语句。

如果修改操作涉及到两个关系的话，就要执行两次 HOLD－MOVE－UPDATE 操作序列。

在 ALPHA 语言中，修改关系主码的操作是不允许的，例如不能用 UPDATE 语句将学号 95001 改为 95102。如果需要修改主码值，只能先用删除操作删除该元组，然后再把具有新主码值的元组插入到关系中。

（2）插入操作

插入操作用 PUT 语句实现。其步骤是：

① 首先用宿主语言在工作空间中建立新元组；

② 然后用 PUT 语句把该元组存入指定的关系中。

例 18 学校新开设了一门 2 学分的课程“计算机组织与结构”，其课程号为 8，直接先行课为 6 号课程。插入该课程元组。

```
MOVE '8' TO W.Cno
MOVE '计算机组织与结构' TO W.Cname
MOVE '6' TO W.Cpno
MOVE  '2' TO W.Ccredit
PUT  W  (Course)          (把 W 中的元组插入指定关系 Course 中)
```

PUT 语句只对一个关系操作，也就是说表达式必须为单个关系名。

（3）删除

删除操作用 DELETE 语句实现。其步骤为：

① 用 HOLD 语句把要删除的元组从数据库中读到工作空间中；

② 用 DELETE 语句删除该元组。

例 19 95110 学生因故退学，删除该学生元组。

```
HOLD  W  (Student): Student.Sno='95110'
DELETE  W
```

例 20　将学号 95001 改为 95102。

```
HOLD  W  (Student): Student.Sno='95001'
DELETE  W
MOVE '95102' TO W.Sno
MOVE '李勇' TO W.Sname
MOVE '男'  TO W.Ssex
MOVE '20' TO W.Sage
MOVE 'CS' TO W.Sdept
PUT W (Student)
```

例 21　删除全部学生。

```
HOLD  W  (Student)
DELETE  W
```

由于 SC 关系与 Student 关系之间的具有参照关系，为保证参照完整性，删除 Student 中元组时相应地要删除 SC 中的元组（手工删除或由 DBMS 自动执行）：

```
HOLD  W  (SC)
DELETE  W
```

2.5.2　元组关系演算

上一节讲解了一种具体的元组关系演算语言，这一节讲解抽象的元组关系演算。

为了讨论方便，先允许关系（的基数）是无限的。然后再对这种情况下定义的演算作适当的修改，保证关系演算中的每一个公式表示的是有限关系。

在元组关系演算系统中，称

$$\{t \mid \phi(t)\}$$

为**元组演算表达式**。其中 t 是元组变量，$\phi(t)$为**元组关系演算公式**，简称**公式**。它由原子公式和运算符组成。

原子公式有三类：

1. $R(t)$

R 是关系名，t 是元组变量。$R(t)$表示 t 是 R 中的元组。于是，关系 R 可表示为：

$$\{t \mid R(t)\}$$

2. $t[i]\ \theta\ u[j]$

t 和 u 是元组变量，θ 是算术比较运算符。$t[i]\ \theta\ u[j]$表示断言“元组 t 的第 i 个分量与元组 u 的第 j 个分量满足比较关系 θ”。例如，$t[2]<u[3]$ 表示元组 t 的第 2 个分量小于元组 u 的第 3 个分量。

3. $t[i]\ \theta\ c$ 或 $c\ \theta\ t[i]$

这里 c 是常量，该公式表示“t 的第 i 个分量与常量 c 满足比较关系 θ”。例如，$t[4]=3$ 表示元组 t 的第 4 个分量等于 3。

在关系演算中定义了“自由元组变量”和“约束元组变量”的概念。这些概念和谓词演算中的概念完全一样。若公式中的一个元组变量前有“全称量词”或“存在量词”，则称该变量为约束元组变量，否则称自由元组变量。

公式可以递归定义如下：

（1）每个原子公式是公式。

（2）如果 ϕ_1 和 ϕ_2 是公式，则 $\phi_1\wedge\phi_2$，$\phi_1\vee\phi_2$，$\neg\phi_1$ 也是公式。分别表示：

- 如果 ϕ_1 和 ϕ_2 同时为真，则 $\phi_1\wedge\phi_2$ 才为真，否则为假；
- 如果 ϕ_1 和 ϕ_2 中一个或同时为真，则 $\phi_1\vee\phi_2$ 为真，仅当 ϕ_1 和 ϕ_2 同时为假时，$\phi_1\vee\phi_2$ 才为假；
- 若 ϕ_1 为真，则 $\neg\phi_1$ 为假。

（3）若 ϕ 是公式，则 $\exists t(\phi)$ 也是公式。其中符号 $\exists$ 是存在量词符号，$\exists t(\phi)$ 表示：

若有一个 t 使 ϕ 为真，则 $\exists t(\phi)$ 为真，否则 $\exists t(\phi)$ 为假。

（4）若 ϕ 是公式，则 $\forall t(\phi)$ 也是公式。其中符号 $\forall$ 是全称量词符号，$\forall t(\phi)$ 表示：

如果对所有 t，都使 ϕ 为真，则 $\forall t(\phi)$ 为真，否则 $\forall t(\phi)$ 为假。

（5）在元组演算公式中，各种运算符的优先次序为：

① 算术比较运算符最高；

② 量词次之，且 $\exists$ 的优先级高于 $\forall$ 的优先级；

③ 逻辑运算符最低，且 $\neg$ 的优先级高于 $\wedge$ 的优先级，$\wedge$ 的优先级高于 $\vee$ 的优先级；

④ 加括号时，括号中运算符优先，同一括号内的运算符之优先级遵循①，②，③各项。

（6）有限次地使用上述五条规则得到的公式是元组关系演算公式，其他公式不是元组关系演算公式。

一个元组演算表达式 $\{t\mid\phi(t)\}$ 表示了使 $\phi(t)$ 为真的元组集合。

关系代数的运算均可以用关系演算表达式来表示（反之亦然）。下面用关系演算表达式来表示五种基本运算：

（1）并

$$R \cup S = \{ t \mid R(t) \vee S(t) \}$$

（2）差

$$R - S = \{ t \mid R(t) \wedge \neg S(t) \}$$

（3）笛卡儿积

$$R \times S = \{ t^{(n+m)} \mid (\exists u^{(n)})(\exists v^{(m)})(R(u) \wedge S(v) \wedge t[1]=u[1] \wedge \cdots \wedge t[n]=u[n] \wedge t[n+1]=v[1] \wedge \cdots \wedge t[n+m]=v[m]) \}$$

这里 $t^{(n+m)}$表示 t 有目数（$n+m$）

（4）投影

$$\pi_{i1, i2, \ldots, ik}(R) = \{ t^{(k)} \mid (\exists u)(R(u) \wedge t[1]=u[i_1] \wedge \cdots t[k]=u[i_k]) \}$$

（5）选择

$$\sigma_{F'}(R) = \{ t \mid R(t) \wedge F' \}$$

F'是公式 F 用 $t[i]$代替运算对象 i 得到的等价公式。

下面用关系演算来表达查询的例子。

例 1 查询信息系（IS 系）全体学生。

$$S_{IS} = \{ t \mid \text{Student}(t) \wedge t[5] = \text{'IS'} \}$$

例 2 查询年龄小于 20 岁的学生。

$$S_{20} = \{ t \mid \text{Student}(t) \wedge t[4] < 20 \}$$

例 3 查询学生的姓名和所在系。

$$S_1 = \{ t^{(2)} \mid (\exists u)(\text{Student}(u) \wedge t[1]=u[2] \wedge t[2]=u[5]) \}$$

上面定义的关系演算允许出现无限关系。例如，$\{ t \mid \neg R(t) \}$表示所有不属于 R 的元组（元组的目数等于 R 的目数）。要求出这些可能的元组是做不到的，所以必须排除这类无意义的表达式。把不产生无限关系的表达式称为安全表达式，所采取的措施称为安全限制。安全限制通常是定义一个有限的符号集 dom(ϕ)，dom(ϕ)一定包括出现在 ϕ 以及中间结果和最后结果的关系中的所有符号（实际上是各列中值的汇集）。dom(ϕ)不必是最小集。

当满足下列条件时，元组演算表达式$\{ t \mid \phi(t) \}$是安全的：

（1）如果 t 使ϕ（t）为真，则 t 的每个分量是 dom（ϕ）中的元素。

（2）对于 ϕ 中每一个形如$(\exists u)(W(u))$的子表达式，若 u 使 $W(u)$为真，则 u 的每个分量是 dom（ϕ）中的元素。

（3）对于ϕ 中每一个形如$(\forall u)(W(u))$的子表达式，若 u 使 $W(u)$为假，则 u 的每个分量必属于 dom（ϕ）。换言之，若 u 某一分量不属于 dom（ϕ），则 W（u）为真。

例 4 设有关系 R 如图 2.8（a），$S = \{ t \mid \neg R(t) \}$，若不进行安全限制，则可能是一个无限关系。所以定义

$\mathrm{dom}(\phi) = \pi_A(R) \cup \pi_B(R) \cup \pi_C(R)$

$=\{\{a1, a2\}, \{b1, b2\}, \{c1, c2\}\}$

则 S 是 dom(ϕ)中各域值中元素的笛卡儿积与 R 的差集。结果如图 2.8(b)。注意，在做笛卡儿积时各个域中的元素不能搞混。

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

(a)

A	B	C
a_1	b_1	c_2
a_1	b_2	c_1
a_1	b_2	c_2
a_2	b_1	c_1
a_2	b_1	c_2
a_2	b_2	c_1

(b)

图 2.8

2.5.3 域关系演算语言 QBE

关系演算的另一种形式是域关系演算。域关系演算以元组变量的分量即域变量作为谓词变元的基本对象。1975 年由 M.M.Zloof 提出的 QBE 就是一个很有特色的域关系演算语言，该语言于 1978 年在 IBM370 上得以实现。QBE 也指此关系数据库管理系统。

QBE 是 Query By Example（即通过例子进行查询）的简称，其最突出的特点是它的操作方式。它是一种高度非过程化的基于屏幕表格的查询语言，用户通过终端屏幕编辑程序以填写表格的方式构造查询要求，而查询结果也是以表格形式显示，因此非常直观，易学易用。

QBE 中用示例元素来表示查询结果可能的情况，示例元素实质上就是域变量。QBE 操作框架如图 2.9 所示。

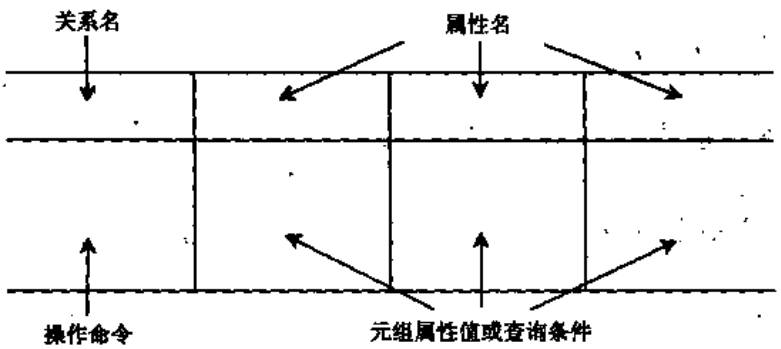


图 2.9 QBE 操作框架

下面以学生-课程关系数据库为例，说明 QBE 的用法。

一、检索操作

1. 简单查询

例 1　求信息系全体学生的姓名。

操作步骤为：

（1）用户提出要求；

（2）屏幕显示空白表格；

（3）用户在最左边一栏输入关系名 Student；

Student					

（4）系统显示该关系的属性名；

Student	Sno	Sname	Ssex	Sage	Sdept

（5）用户在上面构造查询要求。

Student	Sno	Sname	Ssex	Sage	Sdept
		P.T			IS

这里 T 是示例元素，即域变量。QBE 要求示例元素下面一定要加下划线。IS 是查询条件，不用加下划线。P.是操作符，表示打印（Print），实际上是显示。

查询条件中可以使用比较运算符>，≥，<，≤，＝和≠。其中＝可以省略。

示例元素是这个域中可能的一个值，它不必是查询结果中的元素。比如要求信息系的学生，只要给出任意的一个学生名即可，而不必真是信息系的某个学生名。

对于例 4，可如下构造查询要求：

Student	Sno	Sname	Ssex	Sage	Sdept
		P. 李勇			IS

这里的查询条件是 Sdept='IS'，其中“＝”被省略。

（6）屏幕显示查询结果

Student	Sno	Sname	Ssex	Sage	Sdept
		李勇			IS
		张立			

根据用户的查询要求，求出了信息系的学生姓名。

例 2 查询全体学生的全部数据。

Student	Sno	Sname	Ssex	Sage	Sdept
	P.95001	P. 李勇	P.男	P.20	P.CS

显示全部数据也可以简单地把 P.操作符作用在关系名上。因此本查询也可以简单地表示如下：

Student	Sno	Sname	Ssex	Sage	Sdept
P.					

2. 条件查询

例 3 求年龄大于 19 岁的学生的学号。

Student	Sno	Sname	Ssex	Sage	Sdept
	P.95001			>19	

例 4 求计算机科学系年龄大于 19 岁的学生的学号。

本查询的条件是 Sdept='CS'和 Sage>19 两个条件的“与”。在 QBE 中，表示两个条件的“与”有两种方法：

（1）把两个条件写在同一行上；

Student	Sno	Sname	Ssex	Sage	Sdept
	P.95001			>19	CS

（2）把两个条件写在不同行上，但使用相同的示例元素值。

Student	Sno	Sname	Ssex	Sage	Sdept
	P.95001				CS
	P.95001			>19	

例 5 查询计算机科学系或者年龄大于 19 岁的学生的学号。

本查询的条件是 Sdept='CS'和 Sage>19 两个条件的“或”。在 QBE 中把两个条件写在不同行上，并且使用不同的示例元素值，即表示条件的“或”。

Student	Sno	Sname	Ssex	Sage	Sdept
	P.95001				CS
	P.95002			>19	

对于多行条件的查询，如例 4 中的（2）和例 5，先输入哪一行是任意的，查询结果相同。这就允许查询者以不同的思考方式进行查询，十分灵活、自由。

例 6　查询既选修了 1 号课程又选修了 2 号课程的学生的学号。

本查询条件是在一个属性中的“与”关系，它只能用“与”条件的第（2）种方法表示，即写两行，但示例元素相同。

SC	Sno	Cno	Grade
	P.95001	1	
	P.95001	2	

例 7　查询选修 1 号课程的学生姓名。

本查询涉及两个关系：SC 和 Student。在 QBE 中实现这种查询的方法是通过相同的连接属性值把多个关系连接起来。

Student	Sno	Sname	Ssex	Sage	Sdept
	P.95001	P.李勇			

SC	Sno	Cno	Grade
	95001	1	

这里示例元素 Sno 是连接属性，其值在两个表中要相同。

例 8　查询未选修 1 号课程的学生姓名

这里的查询条件中用到逻辑非。在 QBE 中表示逻辑非的方法是将逻辑非写在关系名下面。

Student	Sno	Sname	Ssex	Sage	Sdept
	95001	P.李勇			

SC	Sno	Cno	Grade
¬	95001	1	

这个查询就是显示学号为 95001 的学生名字，而该学生选修 1 号课程的情况为假。

例 9 查询有两个人以上选修的课程号。

本查询是在一个表内连接。

这个查询就是要显示这样的课程 1，它不仅被 95001 选修，而且也被另一个学生（¬95001）选修了。

SC	Sno	Cno	Grade
	95001	P.1	
	¬95001	1	

3. 集函数

为了方便用户，QBE 提供了一些集函数，主要包括 CNT，SUM，AVG，MAX，MIN 等，其含义如表 2.6 所示。

表 2.6 QBE 中的集函数

函 数 名	功 能
CNT	对元组计数
SUM	求 总 和
AVG	求平均值
MAX	求最大值
MIN	求最小值

例 10 查询信息系学生的平均年龄。

Student	Sno	Sname	Ssex	Sage	Sdept
				P.AVG.ALL.	IS

4. 对查询结果排序

对查询结果按某个属性值的升序排序，只需在相应列中填入“AO.”，按降序排序则填“DO.”。如果按多列排序，用“AO(i).”或“DO(i).”表示，其中 i 为排序的优先级，i 值越小，优先级越高。

例 11 查全体男生的姓名，要求查询结果按所在系升序排序，对相同系的学生按年龄降序排序。

Student	Sno	Sname	Ssex	Sage	Sdept
		P.李勇	男	DO (2).	AO (1).

二、更新操作

1. 修改操作

修改操作符为“U.”。在 QBE 中，关系的主码不允许修改，如果需要修改某个元组的主码，只能先删除该元组，然后再插入新的主码的元组。

例 12 把 95001 学生的年龄改为 18 岁。

这是一个简单修改操作，不包含算术表达式，因此可以有两种表示方法。

（1）将操作符“U.”放在值上；

Student	Sno	Sname	Ssex	Sage	Sdept
	95001			U.18	

（2）将操作符“U.”放在关系上。

Student	Sno	Sname	Ssex	Sage	Sdept
U.	95001			18	

这里，码 95001 标明要修改的元组。“U.”标明所在的行是修改后的新值。由于主码是不能修改的，所以即使在第(2)种写法中，系统也不会混淆要修改的属性。

例 13 把 95001 学生的年龄增加 1 岁。

这个修改操作涉及表达式，所以只能将操作符“U.”放在关系上。

Student	Sno	Sname	Ssex	Sage	Sdept
	95001			17	
U.	95001			17+1	

例 14 将计算机科学系所有学生的年龄都增加 1 岁。

Student	Sno	Sname	Ssex	Sage	Sdept
	95008			18	CS
U.	95008			18+1	

2. 插入操作

插入操作符为“I.”。新插入的元组必须具有码值，其他属性值可以为空。

例 15 把信息系女生 95701，姓名张三，年龄 17 岁存入数据库中。

Student	Sno	Sname	Ssex	Sage	Sdept
I.	95701	张三	女	17	IS

3. 删除操作

删除操作符为“D.”。

例 16 删除学生 95089。

Student	Sno	Sname	Ssex	Sage	Sdept
D.	95089				

由于 SC 关系与 Student 关系之间具有参照关系，为保证参照完整性，删除 95089 学生后，通常还应删除 95089 学生选修的全部课程。

SC	Sno	Cno	Grade
D.	95089		

2.5.4 域关系演算

上一节讲解了一种实际的域关系演算语言，这一节讲解抽象的域关系演算。

域关系演算的谓词变元是域变量。

域演算表达式的定义类似于元组演算表达式的定义，所不同的是公式中的元组变量由域变量替代。域变量是表示域的变量。关系的属性名可以视为域变量。

域演算表达式的一般形式为：

$$\{t_1t_2\cdots t_k \mid \phi(t_1, t_2, \cdots, t_k)\}$$

其中 t_1，t_2，…，t_k 分别是域变量，ϕ 是域演算公式。域关系演算公式由原子公式和运算符组成。原子公式有三类：

1. $R(t_1, t_2, \cdots, t_k)$

R 是 k 元关系，t_i 是域变量或常量。$R(t_1, t_2, \cdots, t_k)$ 表示由分量 t_1，t_2，…，t_k 组成的元组属于关系 R。于是，关系 R 可表示为：

$$\{t_1t_2\cdots t_k \mid R(t_1, t_2, \cdots, t_k)\}$$

2. $t_i\,\theta\,u_j$

t_i，u_j 为域变量，θ 为算术比较运算符。$t_i\,\theta\,u_j$ 表示 t_i，u_j 满足比较关系 θ。

3. $t_i\,\theta\,c$ 或 $c\,\theta\,t_i$

t_i 为域变量，c 为常量，θ 为算术比较符。该公式表示 t_i 与常量 c 满足比较关系 θ。

域演算和元组关系演算具有相同的运算符，也有“自由域变量”和“约束域变量”的概念。公式可以递归定义如下：

（1）每个原子公式是公式。

（2）如果 ϕ_1 和 ϕ_2 是域关系演算公式，则 $\phi_1 \wedge \phi_2$，$\phi_1 \vee \phi_2$，$\neg \phi_1$ 也是域关系演算公式。

（3）如果 ϕ 是域关系演算公式，则 $\exists t_i(\phi)$ $(i=1, 2, \cdots, k)$ 也是域关系

演算公式。

（4）如果 ϕ 是域关系演算公式，则 $\forall t_i(\phi)$ (i=1，2，…，k)也是域关系演算公式。

（5）域关系演算公式中运算符优先级与元组关系演算公式中运算符优先级的规定相同。

（6）域关系演算公式是有限次应用上述规则得到的公式，其他公式不是域关系演算公式。

域演算表达式 $\{t_1t_2\cdots t_k \mid \phi(t_1, t_2, \cdots, t_k)\}$ 表示所有使得 ϕ 为真的那些 t_1，t_2，…，t_k 组成的元组集合。同元组演算表达式一样，域演算表达式也必须满足类似的三个条件才是安全的。

可以证明：

① 每一个关系代数表达式有一个等价的安全的元组演算表达式。

② 每一个安全的元组演算表达式有一个等价的安全的域演算表达式。

③ 每一个安全的域演算表达式有一个等价的关系代数表达式。

因此这三类关系运算的表达能力是等价的，可以互相转换。

2.6 小 结

关系数据库系统是本书的重点。这是因为关系数据库系统是目前使用最广泛的数据库系统。20 世纪 70 年代以后开发的数据库管理系统产品几乎都是基于关系的。更进一步，数据库领域近 30 年来的研究工作也主要是关系的。在数据库发展的历史上，最重要的成就是关系模型。

关系数据库系统与非关系数据库系统的区别是，关系系统只有“表”这一种数据结构；而非关系数据库系统还有其他数据结构，对这些数据结构有其他的操作。

关系理论的确立标志着关系数据库系统的基础研究已经接近顶峰。关系数据库系统已经占据了数据库系统的市场。

本章系统讲解了关系数据库的重要概念，包括关系模型的数据结构、关系的完整性以及关系操作。介绍了用代数方式或逻辑方式来表达的关系语言即关系代数、元组关系演算和域关系演算。在这一章从具体到抽象，先讲解了实际的语言 ALPHA（元组关系演算语言）和 QBE（域关系演算语言），然后讲解了抽象的元组关系演算和域关系演算。

习 题

1. 试述关系模型的三个组成部分。

2. 试述关系数据语言的特点和分类。

3. 定义并理解下列术语，说明它们之间的联系与区别:

（1）域，笛卡尔积，关系，元组，属性

（2）主码，候选码，外部码

（3）关系模式，关系，关系数据库

4. 试述关系模型的完整性规则。在参照完整性中，为什么外部码属性的值也可以为空？什么情况下才可以为空？

5. 设有一个 SPJ 数据库，包括 S，P，J，SPJ 四个关系模式：

S(SNO，SNAME，STATUS，CITY);

P(PNO，PNAME，COLOR，WEIGHT);

J(JNO，JNAME，CITY);

SPJ(SNO，PNO，JNO，QTY);

供应商表 S 由供应商代码（SNO）、供应商姓名（SNAME）、供应商状态（STATUS）、供应商所在城市（CITY）组成；

零件表 P 由零件代码（PNO）、零件名（PNAME）、颜色（COLOR）、重量（WEIGHT）组成；

工程项目表 J 由工程项目代码（JNO）、工程项目名（JNAME）、工程项目所在城市（CITY）组成；

供应情况表 SPJ 由供应商代码（SNO）、零件代码（PNO）、工程项目代码（JNO）、供应数量（QTY）组成，表示某供应商 供应某种零件 给某工程项目的数量为 QTY。

今有若干数据如下：

S 表

SNO	SNAME	STATUS	CITY
S1	精 益	20	天津
S2	盛 锡	10	北京
S3	东方红	30	北京
S4	丰泰盛	20	天津
S5	为 民	30	上海

P表

PNO	PNAME	COLOR	WEIGHT
P1	螺　母	红	12
P2	螺　栓	绿	17
P3	螺丝刀	蓝	14
P4	螺丝刀	红	14
P5	凸　轮	蓝	40
P6	齿　轮	红	30

J表

JNO	JNAME	CITY
J1	三　建	北京
J2	一　汽	长春
J3	弹簧厂	天津
J4	造船厂	天津
J5	机车厂	唐山
J6	无线电厂	常州
J7	半导体厂	南京

SPJ表

SNO	PNO	JNO	QTY
S1	P1	J1	200
S1	P1	J3	100
S1	P1	J4	700
S1	P2	J2	100
S2	P3	J1	400
S2	P3	J2	200
S2	P3	J4	500
S2	P3	J5	400
S2	P5	J1	400
S2	P5	J2	100
S3	P1	J1	200
S3	P3	J1	200
S4	P5	J1	100
S4	P6	J3	300
S4	P6	J4	200
S5	P2	J4	100
S5	P3	J1	200
S5	P6	J2	200
S5	P6	J4	500

试用关系代数、ALPHA 语言、QBE 语言完成如下查询：

（1） 求供应工程 J1 零件的供应商号码 SNO；

（2） 求供应工程 J1 零件 P1 的供应商号码 SNO；

（3） 求供应工程 J1 零件为红色的供应商号码 SNO；

（4） 求没有使用天津供应商生产的红色零件的工程号 JNO；

（5） 求至少用了供应商 S1 所供应的全部零件的工程号 JNO。

6. 试述等值连接与自然连接的区别和联系。

7. 关系代数的基本运算有哪些？如何用这些基本运算来表示其他运算？

本章参考文献

1 Codd E F. A Relational Model of Data for Large Shared Data Banks. CACM 13 No.6, 1970

（1970 年，E.F.Codd 在[1]中首先提出了关系数据模型，以后 Codd 又提出了关系代数和关系演算的概念，提出了函数依赖的概念，1972 年提出了关系的第一、第二、第三范式，1974 年提出了 BC(Boyce-Codd)范式，为关系数据库系统奠定了理论基础。由于对关系数据库理论的突出贡献，Codd 获得了 ACM 的最高奖——图灵奖。）

2 Codd E F. A Data Base Sublanguage Founded on the Relational Calculus. Proceedings of ACM SIGFIDET Workshop on Data Description. Access and Control, November 1971

3 Codd E F. Relational Completeness of Data Base Sublanguages in Data Base Systems. Courant Computer Science Symposia Series, Vol.6, 1972

4 Codd E F. Further Normalization of the Data Base Relational Model. In Data Base Systems, Prentice-Hall, 1972

5 Codd E F. Recent Investigations in Relational Database Systems. in Proceedings of the IFIP Congess, 1974

6 Ullman J. Principles of Database Systems, Second Edition, Computer Science Press, 1982

（Ullman 在文献[6]中给出了关系代数、元组关系演算和域关系演算的等价性证明。）

7 Hall P A V, Hitchcock P, Todd S J P. An Algebra of Relations for Machine Computation. Conference Record of the Second ACM Symposium on Principles of Programming Languages. 1975

8 Furtado A L, Kerschberg L. An Algebra of Quotient Relations. Proceedings of ACM SIGMOD International Conference on Management of Data. August 1977

9 Zloof M. M. Query By Example. Proc. NCC, AFIPS, 44, May 1975

（域关系演算的思想出现在文献[9]描述的 QBE 语言中，形式化定义由 Lacroix 和 Pirotte 在文献[10]中给出。文献[11]给出了一个域关系演算语言 ILL。）

10 Lacroix M, Pirotte A. Domain-Oriented Relational Languages. Proc. 3rd International

Conference on Very Large Data Bases. October 1977

11 Lacroix M, Pirotte A. ILL: An English Structured Query Language for Relational Data Bases. in Nijssen, 1977

12 Stonebraker M, Wong E, Kreps P, et al. The Design and Implementation of INGRES, TODS, 1:3, September 1976

（文献[12]讨论的 QUEL 语言是建立在元组关系演算基础上的查询语言。QUEL 语言是 INGRES 数据库系统的原始查询语言，文献[13]描述了大学版 INGRES 系统的 QUEL 语言，文献[14]介绍了商品化 INGRES 数据库系统的 QUEL 语言。）

13 Zook W, et al. INGRES Reference Manual. Department of EECS, University of California at Berkeley, 1977

14 Zook W, et al. RTI INGRES Reference Manual. Relational Technology Inc., 1983

15 Date C J. Referential Integrity. Proc.7th International Conference on Very Large Data Bases, September 1981

16 Date C J. Why Relations Should Have Exactly One Primary Key. ANS Database Committee(X3H2) Working Paper X3H2-84-118, October 1984

第三章　关系数据库标准语言 SQL

SQL (Structured Query Language) 语言是 1974 年由 Boyce 和 Chamberlin 提出的。1975 年～1979 年 IBM 公司 San Jose Research Laboratory 研制了著名的关系数据库管理系统原型 System R 并实现了这种语言。由于它功能丰富，语言简捷倍受用户及计算机工业界欢迎，被众多计算机公司和软件公司所采用。经各公司的不断修改、扩充和完善，SQL 语言最终发展成为关系数据库的标准语言。

1986 年 10 月美国国家标准局（American National Standard Institute，简称 ANSI）的数据库委员会 X3H2 批准了 SQL 作为关系数据库语言的美国标准。同年公布了 SQL 标准文本（简称 SQL-86）。1987 年国际标准化组织（International Organization for Standardization，简称 ISO）也通过了这一标准。此后 ANSI 不断修改和完善 SQL 标准，并于 1989 年公布了 SQL-89 标准，1992 年又公布了 SQL-92 标准。目前 ANSI 即将公布正在酝酿新的标准 SQL-99，亦称 SQL3。

自 SQL 成为国际标准语言以后，各个数据库厂家纷纷推出各自的 SQL 软件或与 SQL 的接口软件。这就使大多数数据库均用 SQL 作为共同的数据存取语言和标准接口，使不同数据库系统之间的互操作有了共同的基础。这个意义十分重大。因此，有人把确立 SQL 为关系数据库语言标准及其后的发展称为是一场革命。

SQL 成为国际标准，对数据库以外的领域也产生了很大影响，有不少软件产品将 SQL 语言的数据查询功能与图形功能、软件工程工具、软件开发工具、人工智能程序结合起来。SQL 已成为数据库领域中一个主流语言。这一章详细介绍 SQL 语言，并进一步讲述关系数据库的基本概念。

3.1　SQL　概　述

SQL 是一种介于关系代数与关系演算之间的结构化查询语言，其功能并不仅仅是查询。SQL 是一个通用的、功能极强的关系数据库语言。

3.1.1 SQL 的特点

SQL 语言之所以能够为用户和业界所接受，并成为国际标准，是因为它是一个综合的、功能极强同时又简捷易学的语言。SQL 语言集数据查询（Data Query）、数据操纵（Data Manipulation）、数据定义（Data Definition）和数据控制（Data Control）功能于一体，主要特点包括：

一、综合统一

数据库系统的主要功能是通过数据库支持的数据语言来实现的。

非关系模型（层次模型、网状模型）的数据语言一般都分为模式数据定义语言（Schema Data Definition Language，简称模式 DDL）、外模式数据定义语言（Subschema Data Definition Language，简称外模式 DDL 或子模式 DDL）、与数据存储有关的描述语言（Data Storage Description Language，简称 DSDL）及数据操纵语言（Data Manipulation Language，简称 DML），分别用于定义模式、外模式、内模式和进行数据的存取与处置。当用户数据库投入运行后，如果需要修改模式，必须停止现有数据库的运行，转储数据，修改模式并编译后再重装数据库，十分麻烦。

SQL 语言则集数据定义语言 DDL、数据操纵语言 DML、数据控制语言 DCL 的功能于一体，语言风格统一，可以独立完成数据库生命周期中的全部活动，包括定义关系模式、插入数据建立数据库、查询、更新、维护、数据库重构、数据库安全性控制等一系列操作要求，这就为数据库应用系统的开发提供了良好的环境。用户在数据库系统投入运行后，还可根据需要随时地逐步地修改模式，且并不影响数据库的运行，从而使系统具有良好的可扩展性。

另外，在关系模型中实体和实体间的联系均用关系表示，这种数据结构的单一性带来了数据操作符的统一，查找、插入、删除、修改等每一种操作都只需一种操作符，从而克服了非关系系统由于信息表示方式的多样性带来的操作复杂性。例如，在 DBTG 中，需要两种插入操作符：STORE 用来把记录存入数据库，CONNECT 用来把记录插入系值以建立数据之间的联系。

二、高度非过程化

非关系数据模型的数据操纵语言是面向过程的语言，用其完成某项请求，必须指定存取路径。而用 SQL 语言进行数据操作，只要提出“做什么”，而无须指明“怎么做”，因此无需了解存取路径，存取路径的选择以及 SQL 语句的操作过程由系统自动完成。这不但大大减轻了用户负担，而且有利于提高

数据独立性。

三、面向集合的操作方式

非关系数据模型采用的是面向记录的操作方式，操作对象是一条记录。例如查询所有平均成绩在 80 分以上的学生姓名，用户必须一条一条地把满足条件的学生记录找出来（通常要说明具体处理过程，即按照哪条路径，如何循环等）。而 SQL 语言采用集合操作方式，不仅操作对象、查找结果可以是元组的集合，而且一次插入、删除、更新操作的对象也可以是元组的集合。

四、以同一种语法结构提供两种使用方式

SQL 语言既是自含式语言，又是嵌入式语言。作为自含式语言，它能够独立地用于联机交互的使用方式，用户可以在终端键盘上直接键入 SQL 命令对数据库进行操作；作为嵌入式语言，SQL 语句能够嵌入到高级语言（例如 C，COBOL，FORTRAN，PL/1）程序中，供程序员设计程序时使用。而在两种不同的使用方式下，SQL 语言的语法结构基本上是一致的。这种以统一的语法结构提供两种不同的使用方式的做法，提供了极大的灵活性与方便性。

五、语言简捷，易学易用

SQL 语言功能极强，但由于设计巧妙，语言十分简捷，完成核心功能只用了 9 个动词，如表 3.1 所示。SQL 语言接近英语口语，因此容易学习，容易使用。

表 3.1 SQL 语言的动词

SQL 功能	动 词
数据查询	SELECT
数据定义	CREATE，DROP，ALTER
数据操纵	INSERT，UPDATE，DELETE
数据控制	GRANT，REVOKE

3.1.2 SQL 语言的基本概念

SQL 语言支持关系数据库三级模式结构，如图 3.1 所示。其中外模式对应于视图（View）和部分基本表（Base Table），模式对应于基本表，内模式对应于存储文件。

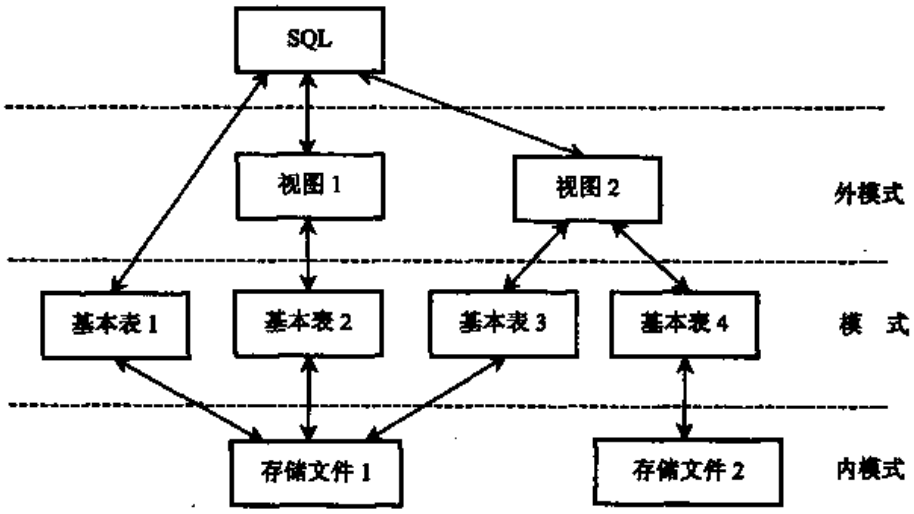


图 3.1 SQL 对关系数据库模式的支持

用户可以用 SQL 语言对基本表和视图进行查询或其他操作，基本表和视图一样，都是关系。

基本表是本身独立存在的表，在 SQL 中一个关系就对应一个表。一个（或多个）基本表对应一个存储文件，一个表可以带若干索引，索引也存放在存储文件中。

存储文件的逻辑结构组成了关系数据库的内模式。存储文件的物理结构是任意的，对用户是透明的。

视图是从一个或几个基本表导出的表。它本身不独立存储在数据库中，即数据库中只存放视图的定义而不存放视图对应的数据，这些数据仍存放在导出视图的基本表中，因此视图是一个虚表。视图在概念上与基本表等同，用户可以在视图上再定义视图。

下面将逐一介绍各 SQL 语句的功能和格式。为了突出基本概念和基本功能，略去了许多语法细节。各个 DBMS 产品在实现标准 SQL 语言时也各有差别，一般都做了某种扩充。因此，具体使用某个 DBMS 产品时，还应参阅系统提供的有关手册。

3.2 数 据 定 义

关系数据库系统支持三级模式结构，其模式、外模式和内模式中的基本对象有表、视图和索引。因此 SQL 的数据定义功能包括定义表、定义视图和定

义索引，如表 3.2 所示。

视图是基于基本表的虚表，索引是依附于基本表的，因此 SQL 通常不提供修改视图定义和修改索引定义的操作。用户如果想修改视图定义或索引定义，只能先将它们删除掉，然后再重建。不过有些关系数据库产品如 Oracle 允许直接修改视图定义。

表 3.2 SQL 的数据定义语句

操作对象	操作方式		
	创建	删除	修改
表	CREATE TABLE	DROP TABLE	ALTER TABLE
视图	CREATE VIEW	DROP VIEW	
索引	CREATE INDEX	DROP INDEX	

本节只介绍如何定义基本表和索引，视图的概念及其定义方法将在 3.4 节专门讨论。

3.2.1 定义、删除与修改基本表

一、定义基本表

建立数据库最重要的一步就是定义一些基本表。SQL 语言使用 CREATE TABLE 语句定义基本表，其一般格式如下：

```
CREATE TABLE <表名> (<列名> <数据类型>[ 列级完整性约束条件 ]
                    [, <列名> <数据类型>[ 列级完整性约束条件] ]…
                    [, <表级完整性约束条件> ]) ;
```

其中<表名>是所要定义的基本表的名字，它可以由一个或多个属性（列）组成。

建表的同时通常还可以定义与该表有关的完整性约束条件，这些完整性约束条件被存入系统的数据字典中，当用户操作表中数据时由 DBMS 自动检查该操作是否违背这些完整性约束条件。如果完整性约束条件涉及到该表的多个属性列，则必须定义在表级上，否则既可以定义在列级也可以定义在表级。

例 1 建立一个“学生”表 Student，它由学号 Sno、姓名 Sname、性别 Ssex、年龄 Sage、所在系 Sdept 五个属性组成。其中学号不能为空，值是唯一的，并且姓名取值也唯一。

```
CREATE TABLE Student
```

```
(Sno   CHAR(5) NOT NULL UNIQUE,    /* 列级完整性约束条件，Sno
 Sname  CHAR(20) UNIQUE,             取值唯一，不许取空值 */
 Ssex   CHAR(1) ,
 Sage   INT,
 Sdept  CHAR(15));
```

系统执行上面的 CREATE TABLE 语句后，就在数据库中建立一个新的空的“学生”表 Student，并将有关“学生”表的定义及有关约束条件存放在数据字典中。

定义表的各个属性时需要指明其数据类型及长度。不同的数据库系统支持的数据类型不完全相同，例如 IBM DB2 SQL 主要支持以下等数据类型：

SMALLINT	半字长二进制整数。
INTEGER 或 INT	全字长二进制整数。
DECIMAL(p[，q]) 或 DEC(p[，q])	压缩十进制数，共 p 位，其中小数点后有 q 位。$0 \leqslant q \leqslant p \leqslant 15$，$q=0$ 时可以省略不写。
FLOAT	双字长浮点数。
CHARTER(n)或 CHAR(n)	长度为 n 的定长字符串。
VARCHAR(n)	最大长度为 n 的变长字符串。
GRAPHIC(n)	长度为 n 的定长图形字符串。
VARGRAPHIC(n)	最大长度为 n 的变长图形字符串。
DATE	日期型，格式为 YYYY-MM-DD。
TIME	时间型，格式为 HH.MM.SS。
TIMESTAMP	日期加时间。

二、修改基本表

随着应用环境和应用需求的变化，有时需要修改已建立好的基本表，SQL 语言用 ALTER TABLE 语句修改基本表，其一般格式为：

```
ALTER TABLE <表名>
[ ADD <新列名> <数据类型> [ 完整性约束 ] ]
[ DROP <完整性约束名> ]
[ MODIFY <列名> <数据类型> ];
```

其中<表名>是要修改的基本表，ADD 子句用于增加新列和新的完整性约束条件，DROP 子句用于删除指定的完整性约束条件，MODIFY 子句用于修改原有的列定义，包括修改列名和数据类型。

例 2　向 Student 表增加“入学时间”列，其数据类型为日期型。

```
ALTER TABLE Student ADD Scome DATE;
```

不论基本表中原来是否已有数据，新增加的列一律为空值。

例 3 将年龄的数据类型改为半字长整数。

```
ALTER TABLE Student MODIFY Sage SMALLINT;
```

修改原有的列定义有可能会破坏已有数据。

例 4 删除学生姓名必须取唯一值的约束。

```
ALTER TABLE Student DROP UNIQUE(Sname);
```

SQL 没有提供删除属性列的语句，用户只能间接实现这一功能，即先把表中要保留的列及其内容复制到一个新表中，然后删除原表，再将新表重命名为原表名。

三、删除基本表

当某个基本表不再需要时，可以使用 DROP TABLE 语句删除它。其一般格式为：

```
DROP TABLE <表名>
```

例 5 删除 Student 表。

```
DROP TABLE  Student
```

基本表定义一旦删除，表中的数据、此表上建立的索引和视图都将自动被删除掉。因此执行删除基本表的操作一定要格外小心。

注意：有的系统，如 Oracle，删除基本表后建立在此表上的视图定义仍然保留在数据字典中。但是，当用户引用时就报错。

3.2.2 建立与删除索引

建立索引是加快查询速度的有效手段。用户可以根据应用环境的需要，在基本表上建立一个或多个索引，以提供多种存取路径，加快查找速度。一般说来，建立与删除索引由数据库管理员 DBA 或表的属主（即建立表的人）负责完成。系统在存取数据时会自动选择合适的索引作为存取路径，用户不必也不能选择索引。

一、建立索引

在 SQL 语言中，建立索引使用 CREATE INDEX 语句，其一般格式为：

```
CREATE [ UNIQUE ] [ CLUSTER ] INDEX <索引名>
        ON <表名>（<列名>[<次序>] [，<列名>[<次序>] ]…）；
```

其中，<表名>是要建索引的基本表的名字。索引可以建立在该表的一列或多

列上，各列名之间用逗号分隔。每个<列名>后面还可以用<次序>指定索引值的排列次序，可选 ASC（升序）或 DESC（降序），缺省值为 ASC。

UNIQUE 表明此索引的每一个索引值只对应唯一的数据记录。

CLUSTER 表示要建立的索引是聚簇索引。所谓聚簇索引是指索引项的顺序与表中记录的物理顺序一致的索引组织。例如，执行下面的 CREATE INDEX 语句：

```
CREATE CLUSTER INDEX Stusname ON Student(Sname);
```

将会在 Student 表的 Sname（姓名）列上建立一个聚簇索引，而且 Student 表中的记录将按照 Sname 值的升序存放。

用户可以在最常查询的列上建立聚簇索引以提高查询效率。显然在一个基本表上最多只能建立一个聚簇索引。建立聚簇索引后，更新索引列数据时，往往导致表中记录的物理顺序的变更，代价较大，因此对于经常更新的列不宜建立聚簇索引。

例 6　为学生-课程数据库中的 Student，Couse，SC 三个表建立索引。其中 Student 表按学号升序建唯一索引，Couse 表按课程号升序建唯一索引，SC 表按学号升序和课程号降序建唯一索引。

```
CREATE UNIQUE INDEX  Stusno ON Student(Sno);
CREATE UNIQUE INDEX Coucno ON Couse(Cno);
CREATE UNIQUE INDEX SCno ON SC(Sno ASC，Cno DESC);
```

二、删除索引

索引一经建立，就由系统使用和维护它，不需用户干预。建立索引是为了减少查询操作的时间，但如果数据增加删改频繁，系统会花费许多时间来维护索引。这时，可以删除一些不必要的索引。

在 SQL 语言中，删除索引使用 DROP INDEX 语句，其一般格式为：

```
DROP INDEX <索引名>;
```

例 7　删除 Student 表的 Stusname 索引。

```
DROP INDEX Stusname;
```

删除索引时，系统会同时从数据字典中删去有关该索引的描述。

3.3 查　询

数据库查询是数据库的核心操作。SQL 语言提供了 SELECT 语句进行数据库的查询，该语句具有灵活的使用方式和丰富的功能。其一般格式为：

SELECT [ALL|DISTINCT] <目标列表达式>[，<目标列表达式>]…
FROM <表名或视图名>[，<表名或视图名>] …
[WHERE <条件表达式>]
[GROUP BY <列名 1> [HAVING <条件表达式>]]
[ORDER BY <列名 2> [ASC|DESC]];

整个 SELECT 语句的含义是，根据 WHERE 子句的条件表达式，从 FROM 子句指定的基本表或视图中找出满足条件的元组，再按 SELECT 子句中的目标列表达式，选出元组中的属性值形成结果表。如果有 GROUP 子句，则将结果按<列名 1>的值进行分组，该属性列值相等的元组为一个组。通常会在每组中作用集函数。如果 GROUP 子句带 HAVING 短语，则只有满足指定条件的组才予输出。如果有 ORDER 子句，则结果表还要按<列名 2>的值的升序或降序排序。

SELECT 语句既可以完成简单的单表查询，也可以完成复杂的连接查询和嵌套查询。下面我们仍以学生-课程数据库为例说明 SELECT 语句的各种用法。

学生-课程数据库中包括三个表。

- 学生表：Student(Sno，Sname，Ssex，Sage，Sdept)
  Student 由学号（Sno）、姓名（Sname）、性别（Ssex）、年龄（Sage）、所在系（Sdept）五个属性组成，其中 Sno 为主码。
- 课程表：Course(Cno，Cname，Cpno，Ccredit)
  Course 由课程号（Cno）、课程名（Cname）、选修课号（Cpno）、学分（Ccredit）四个属性组成，其中 Cno 为主码。
- 学生选课表：SC(Sno，Cno，Grade)
  由学号（Sno）、课程号（Cno）、成绩（Grade）三个属性组成，主码为 (Sno，Cno)。

3.3.1 单表查询

单表查询是指仅涉及一个表的查询。

一、选择表中的若干列

选择表中的全部列或部分列，这就是投影运算。

1. 查询指定列

在很多情况下，用户只对表中的一部分属性列感兴趣，这时可以通过在 SELECT 子句的<目标列表达式>中指定要查询的属性。

例 1　查询全体学生的学号与姓名。

```
SELECT Sno，Sname
FROM Student;
```

例 2　查询全体学生的姓名、学号、所在系。

```
SELECT Sname，Sno，Sdept
FROM Student;
```

<目标列表达式>中各个列的先后顺序可以与表中的顺序不一致。用户可以根据应用的需要改变列的显示顺序。本例中先列出姓名，再列学号和所在系。

2. 查询全部列

将表中的所有属性列都选出来，可以有两种方法。一种方法就是在 SELECT 关键字后面列出所有列名。如果列的显示顺序与其在基表中的顺序相同，也可以简单地将<目标列表达式>指定为 * 。

例 3　查询全体学生的详细记录。

```
SELECT *
FROM Student;
```

等价于：

```
SELECT Sno，Sname，Ssex，Sage，Sdept
FROM Student;
```

3. 查询经过计算的值

SELECT 子句的<目标列表达式>不仅可以是表中的属性列，也可以是表达式。

例 4　查全体学生的姓名及其出生年份。

```
SELECT Sname，1999-Sage，
FROM Student;
```

在例 4 中，<目标列表达式>中第 2 项不是列名，而是一个计算表达式，是用当前的年份（假设为 1996 年）减去学生的年龄，这样，所得的即是学生的出生年份。输出的结果为：

Sname	1996-Sage
李勇	1976
刘晨	1977
王敏	1978
张立	1977

<目标列表达式>不仅可以是算术表达式，还可以是字符串常量、函数等。

例 5　查询全体学生的姓名、出生年份和所有系，要求用小写字母表示所有系名。

```
SELECT Sname，'Year of Birth: '，1996-Sage，ISLOWER(Sdept)
FROM Student;
```

结果为：

Sname	'Year of Birth: '	1996-Sage	ISLOWER(Sdept)
李勇	Year of Birth:	1976	cs
刘晨	Year of Birth:	1977	if
王敏	Year of Birth:	1978	ma
张立	Year of Birth:	1977	if

用户可以通过指定别名来改变查询结果的列标题，这对于含算术表达式、常量、函数名的目标列表达式尤为有用。例如对于上例，可以定义如下列别名：

```
SELECT Sname NAME, 'Year of Birth: ' BIRTH, 1996-Sage BIRTHDAY,
        ISLOWER(Sdept) DEPARTMENT
FROM Student;
```

结果为：

NAME	BIRTH	BIRTHDAY	DEPARTMENT
李勇	Year of Birth:	1976	cs
刘晨	Year of Birth:	1977	if
王敏	Year of Birth:	1978	ma
张立	Year of Birth:	1978	if

二、选择表中的若干元组

1. 消除取值重复的行

两个本来并不完全相同的元组，投影到指定的某些列上后，可能变成相同的行了。

例 6 查询选修了课程的学生学号。

```
SELECT Sno
FROM SC;
```

假设 SC 表中有下列数据

Sno	Cno	Grade
95001	1	92
95001	2	85
95001	3	88
95002	2	90
95002	3	80

执行上面的 SELECT 语句后，结果为：

Sno
95001
95001
95001
95002
95002

该查询结果里包含了许多重复的行。如果想去掉结果表中的重复行，必须指定 DISTINCT 短语：

```
SELECT DISTINCT Sno
FROM SC;
```

执行结果为：

Sno
95001
95002

如果没有指定 DISTINCT 短语，则缺省为 ALL，即保留结果表中取值重复的行。

```
SELECT Sno
FROM SC;
```

等价于

```
SELECT ALL Sno
FROM SC;
```

2. 查询满足条件的元组

查询满足指定条件的元组可以通过 WHERE 子句实现。WHERE 子句常用的查询条件如表 3.3 所示。

表 3.3　常用的查询条件

查 询 条 件	谓　词
比　较	=, >, <, >=, <=, !=, <>, !>, !<; NOT + 上述比较运算符
确定范围	BETWEEN AND，NOT BETWEEN AND
确定集合	IN，NOT IN
字符匹配	LIKE，NOT LIKE
空　值	IS NULL，IS NOT NULL
多重条件	AND，OR

（1）比较大小

用于进行比较的运算符一般包括：

=（等于），>（大于），<（小于），>=（大于等于），<=（小于等于），!=或 <>（不等于）。

有些产品中还包括：!>（不大于），!<（不小于）。

逻辑运算符 NOT 可与比较运算符同用，对条件求非。

例 7 查询计算机系全体学生的名单。

```
SELECT Sname
FROM Student
WHERE Sdept = 'CS';
```

例 8 查询所有年龄在 20 岁以下的学生姓名及其年龄。

```
SELECT Sname, Sage
FROM Student
WHERE Sage < 20;
```

或

```
SELECT Sname, Sage
FROM Student
WHERE NOT Sage >= 20;
```

例 9 查询考试成绩有不及格的学生的学号。

```
SELECT DISTINCT Sno
FROM Course
WHERE Grade < 60;
```

这里使用了 DISTINCT 短语，当一个学生有多门课程不及格，他的学号也只列一次。

（2）确定范围

谓词 BETWEEN…AND…和 NOT BETWEEN…AND…可以用来查找属性值在（或不在）指定范围内的元组，其中 BETWEEN 后是范围的下限（即低值），AND 后是范围的上限（即高值）。

例 10 查询年龄在 20~23 岁（包括 20 岁和 23 岁）之间的学生的姓名、系别和年龄。

```
SELECT Sname, Sdept, Sage
FROM Student
WHERE Sage BETWEEN 20 AND 23;
```

与 BETWEEN…AND…相对的谓词是 NOT BETWEEN…AND…。

例 11 查询年龄不在 20~23 岁之间的学生姓名、系别和年龄。

```
SELECT Sname, Sdept, Sage
```

```
FROM Student
WHERE Sage NOT BETWEEN 20 AND 23;
```

（3）确定集合

谓词 IN 可以用来查找属性值属于指定集合的元组。

例 12　查询信息系（IS）、数学系（MA）和计算机科学系（CS）学生的姓名和性别。

```
SELECT Sname，Ssex
FROM Student
WHERE Sdept IN ( 'IS'，'MA'，'CS' )
```

与 IN 相对的谓词是 NOT IN，用于查找属性值不属于指定集合的元组。

例 13　查询既不是信息系、数学系，也不是计算机科学系的学生的姓名和性别。

```
SELECT Sname，Ssex
FROM Student
WHERE Sdept NOT IN ( 'IS'，'MA'，'CS' )
```

（4）字符匹配

谓词 LIKE 可以用来进行字符串的匹配。其一般语法格式如下：

```
[ NOT ] LIKE '<匹配串>' [ESCAPE ' <换码字符>' ]
```

其含义是查找指定的属性列值与<匹配串>相匹配的元组。<匹配串>可以是一个完整的字符串，也可以含有通配符 % 和 _ 。其中：

• % (百分号)　代表任意长度（长度可以为 0）的字符串。例如 a%b 表示以 a 开头，以 b 结尾的任意长度的字符串。如 acb，addgb，ab 等都满足该匹配串。

• _ (下横线)　代表任意单个字符。

例如 a_b 表示以 a 开头，以 b 结尾的长度为 3 的任意字符串。如 acb，afb 等都满足该匹配串。

例 14　查询学号为 95001 的学生的详细情况。

```
SELECT *
FROM   Student
WHERE  Sno LIKE '95001';
```

等价于：

```
SELECT  *
FROM  Student
WHERE Sno = '95001';
```

如果 LIKE 后面的匹配串中不含通配符，则可以用 =（等于）运算符取代 LIKE 谓词，用 != 或 <>（不等于）运算符取代 NOT LIKE 谓词。

例 15 查询所有姓刘的学生的姓名、学号和性别。

```
SELECT Sname，Sno，Ssex
FROM Student
WHERE  Sname LIKE '刘%';
```

例 16 查询姓“欧阳”且全名为三个汉字的学生的姓名。

```
SELECT Sname
FROM   Student
WHERE  Sname LIKE '欧阳_';
```

注意：一个汉字要占两个字符的位置，所以匹配串欧阳后面需要跟两个_。

例 17 查询名字中第 2 个字为“阳”字的学生的姓名和学号。

```
SELECT Sname，Sno
FROM Student
WHERE Sname LIKE '__阳%';
```

例 18 查询所有不姓刘的学生姓名。

```
SELECT Sname
FROM Student
WHERE Sname NOT LIKE '刘%';
```

如果用户要查询的字符串本身就含有 % 或 _ ，这时就要使用 ESCAPE '<换码字符>' 短语对通配符进行转义了。

例 19 查询 DB_Design 课程的课程号和学分。

```
SELECT Cno，Ccredit
FROM Course
WHERE Cname LIKE 'DB\_Design' ESCAPE '\'
```

ESCAPE '\' 短语表示 \ 为换码字符，这样匹配串中紧跟在 \ 后面的字符“_”不再具有通配符的含义，转义为普通的“_”字符。

例 20 查询以“DB_”开头，且倒数第 3 个字符为 i 的课程的详细情况。

```
SELECT  *
FROM   Course
WHERE  Cname LIKE  'DB\_%i_ _' ESCAPE '\';
```

这里的匹配串为“DB_%i_ _”。第 1 个 _ 前面有换码字符 \ ，所以它被转义为普通的 _ 字符。而%前、第 2 和第 3 个 _ 前面均没有换码字符 \ ，所以它们仍作为通配符。

（5）涉及空值的查询

例 21 某些学生选修课程后没有参加考试，所以有选课记录，但没有考试成绩。查询缺少成绩的学生的学号和相应的课程号。

```
SELECT Sno，Cno
FROM SC
WHERE Grade IS NULL;
```

注意这里的“IS”不能用等号（=）代替。

例 22　查询所有有成绩的学生学号和课程号。

```
SELECT Sno，Cno
FROM  SC
WHERE  Grade IS NOT NULL;
```

（6）多重条件查询

逻辑运算符 AND 和 OR 可用来联结多个查询条件。AND 的优先级高于 OR，但用户可以用括号改变优先级。

例 23　查询计算机系年龄在 20 岁以下的学生姓名。

```
SELECT Sname
FROM  Student
WHERE Sdept= 'CS' AND Sage<20;
```

在例 12 中的 IN 谓词实际上是多个 OR 运算符的缩写，因此例 12 中的查询也可以用 OR 运算符写成如下等价形式：

```
SELECT Sname，Ssex
FROM   Student
WHERE  Sdept= ' IS ' OR Sdept= ' MA' OR Sdept= ' CS ';
```

三、对查询结果排序

用户可以用 ORDER BY 子句对查询结果按照一个或多个属性列的升序（ASC）或降序（DESC）排列，缺省值为升序。

例 24　查询选修了 3 号课程的学生的学号及其成绩，查询结果按分数的降序排列。

```
SELECT Sno，Grade
FROM  SC
WHERE  Cno= ' 3 '
ORDER BY Grade DESC;
```

对于空值，若按升序排，含空值的元组将最后显示。若按降序排，空值的元组将最先显示。

例 25　查询全体学生情况，查询结果按所在系的系号升序排列，同一系中的学生按年龄降序排列。

```
SELECT  *
```

```
FROM   Student
ORDER BY Sdept, Sage DESC;
```

四、使用集函数

为了进一步方便用户，增强检索功能，SQL 提供了许多集函数，主要有：

COUNT（[DISTINCT|ALL] * ）	统计元组个数
COUNT（[DISTINCT|ALL] <列名>）	统计一列中值的个数
SUM（[DISTINCT|ALL] <列名>）	计算一列值的总和（此列必须是数值型）
AVG（[DISTINCT|ALL] <列名>）	计算一列值的平均值（此列必须是数值型）
MAX（[DISTINCT|ALL] <列名>）	求一列值中的最大值
MIN（[DISTINCT|ALL] <列名>）	求一列值中的最小值

如果指定 DISTINCT 短语，则表示在计算时要取消指定列中的重复值。如果不指定 DISTINCT 短语或指定 ALL 短语（ALL 为缺省值），则表示不取消重复值。

例 26 查询学生总人数。

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

例 27 查询选修了课程的学生人数。

```
SELECT COUNT(DISTINCT Sno)
FROM SC;
```

学生每选修一门课，在 SC 中都有一条相应的记录。一个学生要选修多门课程，为避免重复计算学生人数，必须在 COUNT 函数中用 DISTINCT 短语。

例 28 计算 1 号课程的学生平均成绩。

```
SELECT AVG(Grade)
FROM SC
WHERE Cno='1';
```

例 29 查询选修 1 号课程的学生最高分数。

```
SELECT MAX(Grade)
FROM SC
WHERE Cno='1';
```

五、对查询结果分组

GROUP BY 子句将查询结果表按某一列或多列值分组，值相等的为一组。

对查询结果分组的目的是为了细化集函数的作用对象。如果未对查询结果分组，集函数将作用于整个查询结果，如上面的例 26、例 27、例 28、例 29。分

组后集函数将作用于每一个组，即每一组都有一个函数值。

例 30　求各个课程号及相应的选课人数。

```
SELECT Cno，COUNT(Sno)
FROM SC
GROUP BY Cno;
```

该语句对查询结果按 Cno 的值分组，所有具有相同 Cno 值的元组为一组，然后对每一组作用集函数 COUNT 计算，以求得该组的学生人数。查询结果可能为：

Cno	COUNT(Sno)
1	22
2	34
3	44
4	33
5	48

如果分组后还要求按一定的条件对这些组进行筛选，最终只输出满足指定条件的组，则可以使用 HAVING 短语指定筛选条件。

例 31　查询选修了 3 门以上课程的学生学号。

```
SELECT Sno
FROM SC
GROUP BY Sno
HAVING COUNT(*) >3;
```

这里先用 GROUP BY 子句按 Sno 进行分组，再用集函数 COUNT 对每一组计数。HAVING 短语指定选择组的条件，只有满足条件（即元组个数>3，表示此学生选修的课超过 3 门）的组才会被选出来。

WHERE 子句与 HAVING 短语的区别在于作用对象不同。WHERE 子句作用于基本表或视图，从中选择满足条件的元组。HAVING 短语作用于组，从中选择满足条件的组。

3.3.2　连接查询

前面的查询都是针对一个表进行的。若一个查询同时涉及两个以上的表，则称之为连接查询。连接查询是关系数据库中最主要的查询，包括等值连接、自然连接、非等值连接查询、自身连接查询、外连接查询和复合条件连接查询。

一、等值与非等值连接查询

连接查询中用来连接两个表的条件称为**连接条件或连接谓词**，其一般格式为：

[<表名 1>.]<列名 1> <比较运算符> [<表名 2>.]<列名 2>

其中比较运算符主要有：=、>、<、>=、<=、!=。

此外连接谓词还可以使用下面形式：

[<表名 1>.]<列名 1> BETWEEN [<表名 2>.]<列名 2> AND [<表名 2>.]<列名 3>

当连接运算符为 = 时，称为**等值连接**。使用其他运算符称为**非等值连接**。

连接谓词中的列名称为**连接字段**。连接条件中的各连接字段类型必须是可比的，但不必是相同的。

从概念上讲，DBMS 执行连接操作的过程是:首先在表 1 中找到第 1 个元组，然后从头开始扫描表 2，逐一查找满足连接条件的元组，找到后就将表 1 中的第 1 个元组与该元组拼接起来，形成结果表中一个元组。表 2 全部查找完后，再找表 1 中第 2 个元组，然后再从头开始扫描表 2，逐一查找满足连接条件的元组，找到后就将表 1 中的第 2 个元组与该元组拼接起来，形成结果表中一个元组。重复上述操作，直到表 1 中的全部元组都处理完毕为止。

例 32 查询每个学生及其选修课程的情况。

学生情况存放在 Student 表中，学生选课情况存放在 SC 表中，所以本查询实际上涉及 Student 与 SC 两个表。这两个表之间的联系是通过公共属性 Sno 实现的。

```
SELECT Student.*, SC.*
FROM Student, SC
WHERE Student.Sno=SC.Sno;    /* 将 Student 与 SC 中同一学生的元组连接起来 */
```

假设 Student 表、SC 表分别有下列数据：

Student 表

Sno	Sname	Ssex	Sage	Sdept
95001	李勇	男	20	CS
95002	刘晨	女	19	IS
95003	王敏	女	18	MA
95004	张立	男	19	IS

SC 表

Sno	Cno	Grade
95001	1	92
95001	2	85
95001	3	88
95002	2	90
95002	3	80

该查询的执行结果为：

Student.Sno	Sname	Ssex	Sage	Sdept	SC.Sno	Cno	Grade
95001	李勇	男	20	CS	95001	1	92
95001	李勇	男	20	CS	95001	2	85
95001	李勇	男	20	CS	95001	3	88
95002	刘晨	女	19	IS	95002	2	90
95002	刘晨	女	19	IS	95002	3	80

本例中，SELECT 子句与 WHERE 子句中的属性名前都加上了表名前缀，这是为了避免混淆。如果属性名在参加连接的各表中是唯一的，则可以省略表名前缀。

连接运算中有两种特殊情况，一种为自然连接，另一种为广义笛卡尔积（连接）。

广义笛卡尔积是不带连接谓词的连接。两个表的广义笛卡尔积即是两表中元组的交叉乘积，其连接的结果会产生一些没有意义的元组，所以这种运算实际很少使用。

若在等值连接中把目标列中重复的属性列去掉则为自然连接。

例 33　对例 32 用自然连接完成。

```
SELECT Student.Sno，Sname，Ssex，Sage，Sdept，Cno，Grade
FROM Student，SC
WHERE Student.Sno=SC.Sno;
```

例中，由于 Sname，Ssex，Sage，Sdept，Cno 和 Grade 属性列在 Student 表与 SC 表中是唯一的，因此引用时可以去掉表名前缀。而 Sno 在两个表都出现了，因此引用时必须加上表名前缀。

二、自身连接

连接操作不仅可以在两个表之间进行，也可以是一个表与其自己进行连接，称为表的自身连接。

例 34　查询每一门课的间接先修课（即先修课的先修课）。

在 Course 表关系中，只有每门课的直接先修课信息，而没有先修课的先修课。要得到这个信息，必须先对一门课找到其先修课，再按此先修课的课程

号，查找它的先修课程。这就要将 Course 表与其自身连接。

为清楚起见，可以为 Course 表取两个别名，一个是 FIRST，另一个是 SECOND。

FIRST 表（Course 表）

Cno	Cname	Cpno	Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL 语言	6	4

SECOND 表（Course 表）

Cno	Cname	Cpno	Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL 语言	6	4

完成该查询的 SQL 语句为：

```
SELECT FIRST.Cno，SECOND.Cpno
FROM Course FIRST，Course SECOND
WHERE FIRST.Cpno = SECOND.Cno;
```

结果为：

Cno	Pcno
1	7
3	5
5	6

三、外连接

在通常的连接操作中，只有满足连接条件的元组才能作为结果输出。如例 32 和例 33 的结果表中没有 95003 和 95004 两个学生的信息，原因在

于他们没有选课，在 SC 表中没有相应的元组。但是有时我们想以 Student 表为主体列出每个学生的基本情况及其选课情况，若某个学生没有选课，只输出其基本情况信息，其选课信息为空值即可，这时就需要使用外连接（Outer Join）。可以如下改写例 33：

```
SELECT  Student.Sno，Sname，Ssex，Sage，Sdept，Cno，Grade
FROM  Student，SC
WHERE  Student.Sno = SC.Sno(*);
```

外连接的表示方法为，在连接谓词的某一边加符号 *（有的数据库系统中用 +）。外连接就好象是为符号 * 所在边的表（本例是 SC 表）增加一个“万能”的行，这个行全部由空值组成。它可以和另一边的表（本例是 Student 表）中所有不满足连接条件的元组进行连接。本例中即与 Student 表的 95003 和 95004 元组进行连接。由于这个“万能”行的各列全部是空值，因此在连接结果中，95003 和 95004 两行中来自 SC 表的属性值全部是空值。

执行结果如下：

Student.Sno	Sname	Ssex	Sage	Sdept	Cno	Grade
95001	李勇	男	20	CS	1	92
95001	李勇	男	20	CS	2	85
95001	李勇	男	20	CS	3	88
95002	刘晨	女	19	IS	2	90
95002	刘晨	女	19	IS	3	80
95003	王敏	女	18	MA		
95004	张立	男	19	IS		

如果外连接符出现在连接条件的右边，称其为**右外连接**（如本例）。如果外连接符出现在连接条件的左边，则称为**左外连接**。

四、复合条件连接

上面各个连接查询中，WHERE 子句中只有一个条件，即连接谓词。WHERE 子句中可以有多个连接条件，称为**复合条件连接**。

例 35 查询选修 2 号课程且成绩在 90 分以上的所有学生。

```
SELECT Student.Sno，Sname
FROM   Student，SC
WHERE  Student.Sno = SC.Sno AND    /* 连接谓词 */
       SC.Cno='2' AND              /* 其他限定条件 */
       SC.Grade>90;                /* 其他限定条件 */
```

连接操作除了可以是两表连接，一个表与其自身连接外，还可以是两个以上的表进行连接，后者通常称为多表连接。

例 36 查询每个学生的学号、姓名、选修的课程名及成绩。

本查询涉及到三个表，完成该查询的 SQL 语句如下：

```
SELECT Student.Sno，Sname，Cname，Grade
FROM Student，SC，Course
WHERE Student.Sno = SC.Sno and SC.Cno = Course.Cno;
```

执行该查询后结果表为：

Student.Sno	Sname	Cname	Grade
95001	李勇	数据库	92
95001	李勇	数学	85
95001	李勇	信息系统	88
95002	刘晨	数学	90
95002	刘晨	信息系统	80

3.3.3 嵌套查询

在 SQL 语言中，一个 SELECT-FROM-WHERE 语句称为一个查询块。将一个查询块嵌套在另一个查询块的 WHERE 子句或 HAVING 短语的条件中的查询称为嵌套查询。例如：

```
SELECT Sname
FROM Student
WHERE Sno IN
    SELECT Sno
    FROM SC
    WHERE Cno='2';
```

本例中，下层查询块 SELECT Sno FROM SC WHERE Cno= '2'是嵌套在上层查询块 SELECT Sname FROM Student WHERE Sno IN 的 WHERE 条件中的。上层的查询块称为外层查询或父查询，下层查询块称为内层查询或子查询。SQL 语言允许多层嵌套查询。即一个子查询中还可以嵌套其他子查询。需要特别指出的是，子查询的 SELECT 语句中不能使用 ORDER BY 子句，ORDER BY 子句只能对最终查询结果排序。

嵌套查询一般的求解方法是由里向外处理。即每个子查询在上一级查询处理之前求解，子查询的结果用于建立其父查询的查找条件。

嵌套查询使我们可以用多个简单查询构成复杂的查询，从而增强 SQL 的查询能力。以层层嵌套的方式来构造程序正是 SQL(Structurred Query Language)

中“结构化”的含义所在。

一、带有 IN 谓词的子查询

在嵌套查询中，子查询的结果往往是一个集合，所以谓词 IN 是嵌套查询中最经常使用的谓词。

例 37　查询与“刘晨”在同一个系学习的学生。

先分步来完成此查询，然后再构造嵌套查询。

① 确定“刘晨”所在系名

```
SELECT Sdept
FROM   Student
WHERE  Sname='刘晨';
```

结果为：

Sdept
IS

② 查找所有在 IS 系学习的学生。

```
SELECT Sno，Sname，Sdept
FROM Student
WHERE Sdept=' IS ';
```

结果为：

Sno	Sname	Sdept
95001	刘晨	IS
95004	张立	IS

将第 1 步查询嵌入到第 2 步查询的条件中，构造嵌套查询，SQL 语句如下：

```
SELECT Sno，Sname，Sdept
FROM Student
WHERE Sdept IN
      （SELECT Sdept
       FROM Student
       WHERE Sname='刘晨'）；
```

DBMS 求解该查询时，实际上也是分步去做的，类似于上面写的分步过程。

本例中的查询也可以用自身连接来完成：

```
SELECT  S1.Sno，S1.Sname，S1.Sdept
FROM Student S1，Student S2
```

```
WHERE S1.Sdept = S2.Sdept AND
      S2.Sname = '刘晨';
```

可见，实现同一个查询可以有多种方法，当然不同的方法其执行效率可能会有差别，甚至会差别很大。

本例中父查询和子查询均引用了 Student 表，可以像自身连接那样用别名将父查询中的 Student 表与子查询中的 Student 表区分开：

```
SELECT Sno，Sname，Sdept
FROM   Student S1
WHERE  S1.Sdept IN
       (SELECT Sdept
        FROM Student S2
        WHERE S2.Sname='刘晨')；
```

例 38 查询选修了课程名为“信息系统”的学生学号和姓名。

本查询涉及学号、姓名和课程名三个属性。学号和姓名存放在 Student 表中，课程名的存放在 Course 表中，但 Student 与 Course 两个表之间没有直接联系，必须通过 SC 表建立它们二者之间的联系。所以本查询实际上涉及三个关系。

```
SELECT Sno，Sname                    ③ 最后在 Student 关系中
FROM Student                           取出 Sno 和 Sname
WHERE Sno IN
      (SELECT Sno                    ② 然后在 SC 关系中找出
       FROM   SC                       选修了 3 号课程的学生
       WHERE  Cno IN                   学号
              (SELECT Cno            ① 首先在 Course 关系中
               FROM Course               找出“信息系统”的课程
               WHERE Cname='信息系统')；   号，结果为 3 号
```

结果为：

Sno	Sname
95001	李勇
95002	刘晨

本查询同样可以用连接查询实现：

```
SELECT Sno，Sname
FROM   Student，SC，Course
WHERE Student.Sno = SC.Sno AND
      SC.Cno = Course.Cno AND
```

```
Course.Cname='信息系统';
```

从例 37 和例 38 可以看到，查询涉及多个关系时，用嵌套查询逐步求解，层次清楚，易于构造，具有结构化程序设计的优点。

有些嵌套查询可以用连接运算替代，有些是不能替代的。到底采用哪种方法用户可以根据自己的习惯确定。

例 37 和例 38 中的各个子查询都只执行一次，其结果用于父查询。子查询的查询条件不依赖于父查询，这类子查询称为**不相关子查询**。不相关子查询是最简单的一类子查询。

二、带有比较运算符的子查询

带有比较运算符的子查询是指父查询与子查询之间用比较运算符进行连接。当用户能确切知道内层查询返回的是单值时，可以用>，<，=，>=，<=，!=或<>等比较运算符。

例如在例 37 中，由于一个学生只可能在一个系学习，也就是说内查询的结果是一个值，因此可以用 = 代替 IN ，其 SQL 语句如下：

```
SELECT Sno，Sname，Sdept
FROM Student
WHERE Sdept =
      (SELECT Sdept
       FROM Student
       WHERE Sname='刘晨')；
```

需要注意的是，子查询一定要跟在比较符之后，下列写法是错误的：

```
SELECT Sno，Sname，Sdept
FROM Student
WHERE ( SELECT Sdept
          FROM Student
          WHERE Sname=' 刘晨 ')= Sdept;
```

三、带有 ANY 或 ALL 谓词的子查询

子查询返回单值时可以用比较运算符 ，而使用 ANY 或 ALL 谓词时则必须同时使用比较运算符。其语义为：

> ANY	大于子查询结果中的某个值
> ALL	大于子查询结果中的所有值
< ANY	小于子查询结果中的某个值
< ALL	小于子查询结果中的所有值

>= ANY	大于等于子查询结果中的某个值
>= ALL	大于等于子查询结果中的所有值
<= ANY	小于等于子查询结果中的某个值
<= ALL	小于等于子查询结果中的所有值
= ANY	等于子查询结果中的某个值
=ALL	等于子查询结果中的所有值（通常没有实际意义）
!=（或<>）ANY	不等于子查询结果中的某个值
!=（或<>）ALL	不等于子查询结果中的任何一个值

例 39 查询其他系中比信息系某一学生年龄小的学生姓名和年龄

```
SELECT Sname, Sage
FROM Student
WHERE Sage < ANY (SELECT Sage
                  FROM Student
                  WHERE Sdept='IS')
      AND Sdept <> 'IS'          /* 注意这是父查询块中的条件 */
```

结果如下：

Sname	Sage
王敏	18

DBMS 执行此查询时，首先处理子查询，找出 IS 系中所有学生的年龄，构成一个集合(19，18)。然后处理父查询，找所有不是 IS 系且年龄小于 19 或 18 的学生。

本查询也可以用集函数来实现。首先用子查询找出 IS 系中最大年龄（19），然后在父查询中查所有非 IS 系且年龄小于 19 岁的学生姓名及年龄。SQL 语句如下：

```
SELECT Sname, Sage
FROM Student
WHERE Sage<
          (SELECT MAX(Sage)
           FROM Student
           WHERE Sdept='IS')
      AND Sdept<>'IS'
```

例 40 查询其他系中比信息系所有学生年龄都小的学生姓名及年龄。

```
SELECT Sname, Sage
FROM Student
```

```
WHERE Sage < ALL
        (SELECT Sage
         FROM Student
         WHERE Sdept= ' IS ')
      AND Sdept <> ' IS '
```

查询结果为空表。本查询同样也可以用集函数实现。SQL 语句如下：

```
SELECT Sname，Sage
FROM Student
WHERE Sage <
        (SELECT MIN(Sage)
         FROM Student
         WHERE Sdept= ' IS ')
      AND Sdept <>' IS '
```

事实上，用集函数实现子查询通常比直接用 ANY 或 ALL 查询效率要高。ANY 与 ALL 与集函数的对应关系如表 3.4 所示。

表 3.4 ANY，ALL 谓词与集函数及 IN 谓词的等价转换关系

	=	<>或!=	<	<=	>	>=
ANY	IN	—	< MAX	<= MAX	> MIN	>= MIN
ALL	—	NOT IN	< MIN	<= MIN	> MAX	>= MAX

四、带有 EXISTS 谓词的子查询

EXISTS 代表存在量词 ∃。带有 EXISTS 谓词的子查询不返回任何数据，只产生逻辑真值“true”或逻辑假值“false”。

例 41 查询所有选修了 1 号课程的学生姓名。

本查询涉及 Student 和 SC 关系。我们可以在 Student 中依次取每个元组的 Sno 值，用此值去检查 SC 关系。若 SC 中存在这样的元组，其 Sno 值等于此 Student.Sno 值，并且其 Cno= '1'，则取此 Student.Sname 送入结果关系。将此想法写成 SQL 语句是：

```
SELECT Sname
FROM Student
WHERE EXISTS
      (SELECT *
       FROM SC
       WHERE Sno=Student.Sno AND Cno= ' 1 ');
```

使用存在量词 EXISTS 后，若内层查询结果非空，则外层的 WHERE 子句返回真值，否则返回假值。

由 EXISTS 引出的子查询，其目标列表达式通常都用 * ，因为带 EXISTS 的子查询只返回真值或假值，给出列名无实际意义。

这类查询与前面讲解的不相关子查询有一个明显区别，即子查询的查询条件依赖于外层父查询的某个属性值（在本例中是 Student 的 Sno 值），称这类查询为**相关子查询**（Correlated Subquery）。求解相关子查询不能像求解不相关子查询那样，一次将子查询求解出来，然后求解父查询。内层查询由于与外层查询有关，因此必须反复求值。从概念上讲，相关子查询的一般处理过程是：

首先取外层查询中（Student）表的第 1 个元组，根据它与内层查询相关的属性值（Sno 值）处理内层查询，若 WHERE 子句返回值为真，则取此元组放入结果表；然后再取（Student）表的下一个元组；重复这一过程，直至外层（Student）表全部检查完为止。

本例中的查询也可以用连接运算来实现，读者可以参照有关的例子，自己给出相应的 SQL 语句。

与 EXISTS 谓词相对应的是 NOT EXISTS 谓词。使用存在量词 NOT EXISTS 后，若内层查询结果为空，则外层的 WHERE 子句返回真值，否则返回假值。

例 42 查询没有选修 1 号课程的学生姓名。

```
SELECT Sname
FROM Student
WHERE NOT EXISTS
      (SELECT *
       FROM SC
       WHERE Sno = Student.Sno AND Cno='1');
```

一些带 EXISTS 或 NOT EXISTS 谓词的子查询不能被其他形式的子查询等价替换，但所有带 IN 谓词、比较运算符、ANY 和 ALL 谓词的子查询都能用带 EXISTS 谓词的子查询等价替换。例如带有 IN 谓词的例 37 可以用如下带 EXISTS 谓词的子查询替换：

```
SELECT Sno，Sname，Sdept
FROM Student S1
WHERE EXISTS
      SELECT *
      FROM Student S2
      WHERE S2.Sdept = S1.Sdept AND
```

```
S2.Sname = '刘晨';
```

由于带 EXISTS 量词的相关子查询只关心内层查询是否有返回值，并不需要查具体值，因此其效率并不一定低于不相关子查询，有时是高效的方法。

SQL 语言中没有全称量词∀（For all）。但是，总可以把带有全称量词的谓词转换为等价的带有存在量词的谓词：

$$(\forall x)P \equiv \neg(\exists x(\neg P))$$

例 43 查询选修了全部课程的学生姓名。

由于没有全称量词，可将题目的意思转换成等价的用存在量词的形式：查询这样的学生，没有一门课程是他不选修的。其 SQL 语句为：

```
SELECT Sname
FROM Student
WHERE NOT EXISTS
        (SELECT *
         FROM Course
         WHERE NOT EXISTS
              (SELECT *
               FROM SC
               WHERE Sno= Student.Sno
                  AND Cno= Course.Cno);
```

SQL 语言中也没有蕴函（Implication）逻辑运算。但是，可以利用谓词演算将一个逻辑蕴函的谓词等价转换为：

$$p \rightarrow q \equiv \neg p \vee q$$

例 44 查询至少选修了学生 95002 选修的全部课程的学生号码。

本查询可以用逻辑蕴函来表达：查询学号为 x 的学生，对所有的课程 y，只要 95002 学生选修了课程 y，则 x 也选修了 y。形式化表示如下：

用 P 表示谓词 “学生 95002 选修了课程 y”

用 q 表示谓词 “学生 x 选修了课程 y”

则上述查询为:

$$(\forall y)\, p \rightarrow q$$

该查询可以转换为如下等价形式：

$$(\forall y)p \rightarrow q \equiv \neg(\exists y(\neg(p \rightarrow q)) \equiv \neg(\exists y(\neg(\neg p \vee q) \equiv \neg \exists y(p \wedge \neg q)$$

它所表达的语义为：不存在这样的课程 y，学生 95002 选修了 y，而学生 x 没有选。用 SQL 语言表示如下：

```
SELECT DISTINCT Sno
FROM SC SCX
WHERE NOT EXISTS
      (SELECT *
       FROM SC SCY
       WHERE SCY.Sno = '95002'  AND
             NOT EXISTS
            (SELECT *
             FROM SC SCZ
             WHERE SCZ.Sno=SCX.Sno AND
                   SCZ.Cno=SCY.Cno);
```

3.3.4 集合查询

SELECT 语句的查询结果是元组的集合，所以多个 SELECT 语句的结果可进行集合操作。集合操作主要包括并操作 UNION、交操作 INTERSECT 和差操作 MINUS。

例 45 查询计算机科学系的学生及年龄不大于 19 岁的学生。

```
SELECT *
FROM Student
WHERE Sdept= 'CS'
UNION
SELECT *
FROM Student
WHERE Sage<=19;
```

本查询实际上是求计算机科学系的所有学生与年龄不大于 19 岁的学生的并集。使用 UNION 将多个查询结果合并起来时，系统会自动去掉重复元组。注意，参加 UNION 操作的各结果表的列数必须相同；对应项的数据类型也必须相同。

例 46 查询选修了课程 1 或者选修了课程 2 的学生。

即查选修课程 1 的学生集合与选修课程 2 的学生集合的并集。

```
SELECT Sno
FROM SC
WHERE Cno=' 1 '
UNION
```

```
SELECT Sno
FROM SC
WHERE Cno='2';
```

标准 SQL 中没有直接提供集合交操作和集合差操作，但可以用其他方法来实现。

例 47　查询计算机科学系的学生与年龄不大于 19 岁的学生的交集，这实际上就是查询计算机科学系中年龄不大于 19 岁的学生。

```
SELECT *
FROM Student
WHERE Sdept= 'CS' AND
      Sage<=19;
```

例 48　查询选修课程 1 的学生集合与选修课程 2 的学生集合的交集

本例实际上是查询既选修了课程 1 又选修了课程 2 的学生。

```
SELECT Sno
FROM SC
WHERE Cno='1' AND Sno IN
                    (SELECT Sno
                     FROM SC
                     WHERE Cno='2');
```

例 49　查询计算机科学系的学生与年龄不大于 19 岁的学生的差集。

本查询换种说法就是，查询计算机科学系中年龄大于 19 岁的学生。

```
SELECT *
FROM Student
WHERE Sdept= 'CS' AND
      Sage>19;
```

3.3.5　SELECT 语句的一般格式

SELECT 语句是 SQL 的核心语句，从上面的例子可以看到其语句成分丰富多样，下面我们总结一下它们的一般格式。

SELECT 语句的一般格式：

```
SELECT [ALL|DISTINCT] <目标列表达式> [别名] [ ，<目标列表达式> [别名]] …
       FROM <表名或视图名> [别名] [ ，<表名或视图名> [别名]] …
       [WHERE <条件表达式>]
```

[GROUP BY <列名 1> [HAVING <条件表达式>]]

[ORDER BY <列名 2> [ASC|DESC]]；

1. 目标列表达式有以下可选格式：

（1） *

（2）<表名>.*

（3）COUNT([DISTINCT|ALL] *)

（4）[<表名>.]<属性列名表达式>[，[<表名>.]<属性列名表达式>] …

其中<属性列名表达式>可以是由属性列、作用于属性列的聚集函数和常量的任意算术运算（+，-，*，/）组成的运算公式。

2. 集函数的一般格式为：

{COUNT / SUM / AVG / MAX / MIN} ([DISTINCT|ALL] <列名>)

3. WHERE 子句的条件表达式有以下可选格式：

（1）

<属性列名> θ {<属性列名> / <常量> / [ANY|ALL] (SELECT 语句)}

（2）

<属性列名> [NOT] BETWEEN {<属性列名> / <常量> / (SELECT 语句)} AND {<属性列名> / <常量> / (SELECT 语句)}

（3）

<属性列名> [NOT] IN {(<值 1>[，<值 2>] …) / (SELECT 语句)}

（4）<属性列名> [NOT] LIKE <匹配串>

（5）<属性列名> IS [NOT] NULL

（6）[NOT] EXISTS (SELECT 语句)

（7）

<条件表达式> {AND / OR} <条件表达式> [{AND / OR} <条件表达式>] …

3.4 数 据 更 新

SQL 中数据更新包括插入数据、修改数据和删除数据三条语句。

3.4.1 插入数据

SQL 的数据插入语句 INSERT 通常有两种形式。一种是插入一个元组，另一种是插入子查询结果。后者可以一次插入多个元组。

一、插入单个元组

插入单个元组的 INSERT 语句的格式为：

```
INSERT
INTO <表名> [(<属性列 1>[，<属性列 2 >…)]
VALUES (<常量 1> [，<常量 2>]…);
```

其功能是将新元组插入指定表中。其中新记录属性列 1 的值为常量 1，属性列 2 的值为常量 2，…。INTO 子句中没有出现的属性列，新记录在这些列上将取空值。

但必须注意的是，在表定义时说明了 NOT NULL 的属性列不能取空值。否则会出错。

如果 INTO 子句中没有指明任何列名，则新插入的记录必须在每个属性列上均有值。

例 1 将一个新学生记录（学号：95020；姓名：陈冬；性别：男；所在系：IS；年龄：18 岁）插入到 Student 表中。

```
INSERT
INTO Student
VALUES ('95020'，'陈冬'，'男'，'IS'，18);
```

例 2 插入一条选课记录('95020'，'1 ')。

```
INSERT
INTO SC(Sno，Cno)
VALUES (' 95020 '，' 1 ');
```

新插入的记录在 Grade 列上取空值。

二、插入子查询结果

子查询不仅可以嵌套在 SELECT 语句中，用以构造父查询的条件（如 3.3.3 节所述），也可以嵌套在 INSERT 语句中，用以生成要插入的批量数据。

插入子查询结果的 INSERT 语句的格式为：

```
INSERT
INTO <表名> [(<属性列 1> [，<属性列 2>…)]
子查询;
```

例 3 对每一个系，求学生的平均年龄，并把结果存入数据库。

首先在数据库中建立一个新表，其中一列存放系名，另一列存放相应的学生平均年龄。

```
CREATE TABLE Deptage
      (Sdept CHAR(15)
       Avgage SMALLINT);
```

然后对 Student 表按系分组求平均年龄，再把系名和平均年龄存入新表中。

```
INSERT
INTO Deptage(Sdept，Avgage)
      SELECT Sdept，AVG(Sage)
      FROM Student
      GROUP BY Sdept;
```

3.4.2 修改数据

修改操作语句的一般格式为：

```
UPDATE <表名>
SET <列名>=<表达式>[，<列名>=<表达式>]…
[WHERE <条件>];
```

其功能是修改指定表中满足 WHERE 子句条件的元组。其中 SET 子句给出<表达式>的值用于取代相应的属性列值。如果省略 WHERE 子句，则表示要修改表中的所有元组。

一、修改某一个元组的值

例 4 将学生 95001 的年龄改为 22 岁。

```
UPDATE Student
```

```
SET Sage=22
WHERE  Sno=' 95001 ';
```

二、修改多个元组的值

例 5　将所有学生的年龄增加 1 岁。

```
UPDATE Student
SET Sage= Sage+1;
```

三、带子查询的修改语句

子查询也可以嵌套在 UPDATE 语句中，用以构造修改的条件。

例 6　将计算机科学系全体学生的成绩置零。

```
UPDATE SC
SET  Grade=0
WHERE  'CS'=
       (SELETE Sdept
        FROM  Student
        WHERE  Student.Sno = SC.Sno);
```

3.4.3 删除数据

删除语句的一般格式为：

```
DELETE
FROM <表名>
[WHERE <条件>];
```

DELETE 语句的功能是从指定表中删除满足 WHERE 子句条件的所有元组。如果省略 WHERE 子句，表示删除表中全部元组，但表的定义仍在字典中。也就是说，DELETE 语句删除的是表中的数据，而不是关于表的定义。

一、删除某一个元组的值

例 7　删除学号为 95019 的学生记录。

```
DELETE
FROM Student
WHERE Sno='95019';
```

二、删除多个元组的值

例 8 删除所有的学生选课记录。

```
DELETE
FROM SC;
```

这条 DELETE 语句将使 SC 成为空表，它删除了 SC 的所有元组。

三、带子查询的删除语句

子查询同样也可以嵌套在 DELETE 语句中，用以构造执行删除操作的条件。

例 9 删除计算机科学系所有学生的选课记录。

```
DELETE
FROM SC
WHERE  'CS'=
    (SELETE Sdept
     FROM Student
     WHERE Student.Sno=SC.Sno);
```

四、更新操作与数据库的一致性

增删改操作只能对一个表操作。这会带来一些问题。例如，比如 95019 学生被删除后，有关其选课信息也应同时删除，而这只能通过两条语句进行。

第 1 条语句删除 95019 学生：

```
DELETE
FROM   Student
WHERE  Sno='95019';
```

第 2 条语句删除 95019 学生的选课记录：

```
DELETE
FROM  SC
WHERE  Sno='95019';
```

在执行了第 1 条 DELETE 语句之后，数据库中的数据已处于不一致状态，因为这时实际上已没有学号为 95019 的学生了，但 SC 表中仍然记录着关于 95019 学生的选课信息，即数据的参照完整性受到破坏。只有执行了第 2 条 DELETE 语句之后，数据才重新处于一致状态。但如果执行完一条语句之后，

机器突然出现故障，无法再继续执行第 2 条 UPDATE 语句，则数据库中的数据将永远处于不一致状态。因此必须保证这两条语句要么都做，要么都不做。为解决这一问题，数据库系统通常都引入了事务（Transaction）的概念，将在第七章详细介绍。

在 2.3 节“关系的完整性”中已讲到，实体完整性和参照完整性是关系模型必须满足的完整性约束条件，应该由关系系统自动支持。

当向参照表中插入元组，如往 SC 中插入选课记录('95020', '1')，系统将自动地检查被参照表（Student 和 Course）中是否存在相应的元组(95020 号学生的记录，1 号课程的记录)，如果都存在则操作成功，否则操作失败。

当删除主表中的元组（如学生 95019），可以有以下的策略：

（1）系统自动地删除参照表（如 SC）中相应的元组（学生 95019 的选课记录）。

（2）系统检查参照表中是否存在相应的元组，如果存在，则操作失败。

各种数据库产品提供了不同的实现策略。

3.5 视　图

视图是关系数据库系统提供给用户以多种角度观察数据库中数据的重要机制。

视图是从一个或几个基本表（或视图）导出的表，它与基本表不同，是一个虚表。数据库中只存放视图的定义，而不存放视图对应的数据，这些数据仍存放在原来的基本表中。所以基本表中的数据发生变化，从视图中查询出的数据也就随之改变了。从这个意义上讲，视图就像一个窗口，透过它可以看到数据库中自己感兴趣的数据及其变化。

视图一经定义，就可以和基本表一样被查询、被删除，我们也可以在一个视图之上再定义新的视图，但对视图的更新（增加、删除、修改）操作则有一定的限制。

本节专门讨论视图的定义、操作及优点。

3.5.1 定义视图

一、建立视图

SQL 语言用 CREATE VIEW 命令建立视图，其一般格式为：

CREATE　VIEW　<视图名>[(<列名>[，<列名>]…)]

```
AS <子查询>
[WITH CHECK OPTION];
```

其中子查询可以是任意复杂的 SELECT 语句，但通常不允许含有 ORDER BY 子句和 DISTINCT 短语。

WITH CHECK OPTION 表示对视图进行 UPDATE，INSERT 和 DELETE 操作时要保证更新、插入或删除的行满足视图定义中的谓词条件（即子查询中的条件表达式）。

组成视图的属性列名或者全部省略或者全部指定，没有第 3 种选择。如果省略了视图的各个属性列名，则隐含该视图由子查询中 SELECT 子句目标列中的诸字段组成。但在下列三种情况下必须明确指定组成视图的所有列名：

（1）某个目标列不是单纯的属性名，而是集函数或列表达式

（2）多表连接时选出了几个同名列作为视图的字段

（3）需要在视图中为某个列启用新的更合适的名字

例 1 建立信息系学生的视图。

```
CREATE VIEW IS_Student
        AS
        SELECT Sno，Sname，Sage
        FROM Student
        WHERE Sdept= 'IS';
```

本例中省略了视图 IS_Student 的列名，隐含了由子查询中 SELECT 子句中的三个列名组成。

DBMS 执行 CREATE VIEW 语句的结果只是把视图的定义存入数据字典，并不执行其中的 SELECT 语句。只是在对视图查询时，才按视图的定义从基本表中将数据查出。

例 2 建立信息系学生的视图，并要求进行修改和插入操作时仍需保证该视图只有信息系的学生。

```
CREATE VIEW IS_Student
        AS
        SELECT Sno，Sname，Sage
        FROM Student
        WHERE Sdept= 'IS'
        WITH CHECK OPTION;
```

由于在定义 IS_Student 视图时加上了 WITH CHECK OPTION 子句，以后对该视图进行插入、修改和删除操作时，DBMS 会自动加上 Sdept= 'IS'的

条件。

若一个视图是从单个基本表导出的，并且只是去掉了基本表的某些行和某些列，但保留了码，我们称这类视图为**行列子集视图**。IS_Student 视图就是一个行列子集视图。

视图不仅可以建立在单个基本表上，也可以建立在多个基本表上。

例 3　建立信息系选修了 1 号课程的学生的视图。

```
CREATE VIEW IS_S1(Sno，Sname，Grade)
        AS
        SELECT   Student.Sno，Sname，Grade
        FROM     Student，SC
        WHERE    Sdept= 'IS' AND
                 Student.Sno=SC.Sno AND
                 SC.Cno= '1';
```

由于视图 IS_S1 的属性列中包含了 Student 表与 SC 表的同名列 Sno，所以必须在视图名后面明确说明视图的各个属性列名。

视图不仅可以建立在一个或多个基本表上，也可以建立在一个或多个已定义好的视图上，或建立在基本表与视图上。

例 4　建立信息系选修了 1 号课程且成绩在 90 分以上的学生的视图。

```
CREATE VIEW IS_S2
        AS
        SELECT   Sno，Sname，Grade
        FROM     IS_S1
        WHERE    Grade>=90;
```

这里的视图 IS_S2 就是建立在视图 IS_S1 之上的。

定义基本表时，为了减少数据库中的冗余数据，表中只存放基本数据，由基本数据经过各种计算派生出的数据一般是不存储的。但由于视图中的数据并不实际存储，所以定义视图时可以根据应用的需要，设置一些派生属性列。这些派生属性由于在基本表中并不实际存在也称它们为**虚拟列**。带虚拟列的视图也称为**带表达式的视图**。

例 5　定义一个反映学生出生年份的视图。

```
CREATE   VIEW BT_S(Sno，Sname，Sbirth)
         AS   SELECT Sno，Sname，1996-Sage
              FROM   Student;
```

BT_S 视图是一个带表达式的视图。视图中的出生年份值是通过计算得到的。

还可以用带有集函数和 GROUP BY 子句的查询来定义视图，这种视图称为分组视图。

例 6 将学生的学号及他的平均成绩定义为一个视图。

假设 SC 表中“成绩”列 Grade 为数字型，否则无法求平均值。

```
CREAT  VIEW S_G(Sno，Gavg)
       AS  SELECT Sno，AVG(Grade)
           FROM  SC
           GROUP BY Sno;
```

由于 AS 子句中 SELECT 语句的目标列平均成绩是通过作用集函数得到的，所以 CREATE VIEW 中必须明确定义组成 S_G 视图的各个属性列名，S_G 是一个分组视图。

例 7 将 Student 表中所有女生记录定义为一个视图。

```
CREATE VIEW F_Student(stdnum，name，scx，age，dept)
       AS  SELECT *
           FROM  Student
           WHERE Ssex='女';
```

这里视图 F_Student 是由子查询“SELECT *”建立的。如果以后修改了基本表 Student 的结构，则 Student 表与 F_Student 视图的映象关系被破坏，因而该视图就不能正确工作了。为避免出现这类问题，最好在修改基本表之后删除由该基本表导出的视图，然后重建（同名）视图。

二、删除视图

该语句的格式为：

```
DROP  VIEW  <视图名>;
```

视图删除后视图的定义将从数据字典中删除。但是由该视图导出的其他视图定义仍在数据字典中，不过该视图已失效。用户使用时会出错，要用 DROP VIEW 语句将它们一一删除。

就像基本表删除后，由该基本表导出的所有视图（定义）没有被删除，但均已无法使用了。删除这些视图（定义）需要显式地使用 DROP VIEW 语句。

例 8 删除视图 IS_S1

```
DROP VIEW IS_S1;
```

执行此语句后，IS_S1 视图的定义将从数据字典中删除。由 IS_S1 视图导出 IS_S2 视图的定义虽然仍在数据字典中，但是该视图已无法使用了，因此应该同时删除。

3.5.2 查询视图

视图定义后，用户就可以象对基本表一样对视图进行查询了。

例 1　在信息系学生的视图中找出年龄小于 20 岁的学生。

```
SELECT  Sno，Sage
FROM  IS_Student
WHERE  Sage<20;
```

DBMS 执行对视图的查询时，首先进行有效性检查，检查查询的表、视图等是否存在。如果存在，则从数据字典中取出视图的定义，把定义中的子查询和用户的查询结合起来，转换成等价的对基本表的查询，然后再执行修正了的查询。这一转换过程称为视图消解（View Resolution）。

本例转换后的查询语句为：

```
SELECT  Sno，Sage
FROM  Student
WHERE  Sdept='IS'  AND  Sage<20;
```

例 2　查询信息系选修了 1 号课程的学生

```
SELECT  Sno，Sname
FROM  IS_Student，SC
WHERE IS_Student.Sno=SC.Sno AND
                SC.Cno='1';
```

本查询涉及虚表 CS_Student 和基本表 SC，通过这两个表的连接来完成用户请求。

在一般情况下，视图查询的转换是直接了当的。但有些情况下，这种转换不能直接进行，查询时就会出现问题。如例 3。

例 3　在 S_G 视图中查询平均成绩在 90 分以上的学生学号和平均成绩，语句为：

```
SELECT *
FROM  S_G
WHERE  Gavg>=90;
```

S_G 视图定义为：

```
SELECT  Sno，AVG(Grade)
FROM  SC
GROUP BY Sno;
```

将上面查询语句与子查询结合后，形成下列查询语句：

```
SELECT Sno，AVG(Grade)
```

```
FROM  SC
WHERE  AVG(Grade)>=90
GROUP BY Sno;
```

前面讲过 WHERE 子句中是不能用集函数作为条件表达式的，因此执行此修正后的查询将会出现语法错误。正确的查询语句应该是：

```
SELECT  Sno，AVG(Grade)
FROM  SC
GROUP BY Sno
HAVING AVG(Grade)>=90;
```

目前多数关系数据库系统对行列子集视图的查询均能进行正确转换。但对非行列子集的查询(如例 3)就不一定能做转换了，因此这类查询应该直接对基本表进行。

3.5.3 更新视图

更新视图是指通过视图来插入（INSERT）、删除（DELETE）和修改（UPDATE）数据。由于视图是不实际存储数据的虚表，因此对视图的更新，最终要转换为对基本表的更新。

为防止用户通过视图对数据进行增加、删除、修改时，有意无意地对不属于视图范围内的基本表数据进行操作，可在定义视图时加上 WITH CHECK OPTION 子句。这样在视图上增删改数据时，DBMS 会检查视图定义中的条件，若不满足条件，则拒绝执行该操作。

例 1 将信息系学生视图 IS_Student 中学号为 95002 的学生姓名改为“刘辰”。

```
UPDATE  IS_Student
SET  Sname='刘辰'
WHERE  Sno='95002';
```

转换后的更新语句为：

```
UPDATE Student
SET Sname='刘辰'
WHERE Sno='95002' AND Sdept='IS';
```

例 2 向信息系学生视图 IS_S 中插入一个新的学生记录，其中学号为 95029，姓名为赵新，年龄为 20 岁。

```
INSERT
INTO IS_Student
```

```
VALUES('95029', '赵新', 20);
```

转换为对基本表的更新：

```
INSERT
INTO  Student(Sno, Sname, Sage, Sdept)
VALUES('95029', '赵新', 20, 'IS');
```

这里系统自动将系名'IS'放入 VALUES 子句中。

例 3　删除计算机系学生视图 CS_S 中学号为 95029 的记录

```
DELETE
FROM IS_Student
WHERE Sno='95029';
```

转换为对基本表的更新：

```
DELETE
FROM Student
WHERE Sno='95029' AND Sdept='IS';
```

在关系数据库中，并不是所有的视图都是可更新的，因为有些视图的更新不能唯一地有意义地转换成对相应基本表的更新。

例如前面定义的视图 S_G 是由“学号”和“平均成绩”两个属性列组成的，其中平均成绩一项是由 Student 表中对元组分组后计算平均值得来的。如果我们想把视图 S_G 中学号为 95001 的学生的平均成绩改成 90 分，SQL 语句如下：

```
UPDATE S_G
SET Gavg=90
WHERE Sno='95001';
```

但这个对视图的更新是无法转换成对基本表 SC 的更新的，因为系统无法修改各科成绩，以使平均成绩成为 90。所以 S_G 视图是不可更新的。

一般地，行列子集视图是可更新的。除行列子集视图外，还有些视图理论上是可更新的，但它们的确切特征还是尚待研究的课题。还有些视图从理论上是不可更新的。

目前各个关系数据库系统一般都只允许对行列子集视图进行更新，而且各个系统对视图的更新还有更进一步的规定，由于各系统实现方法上的差异，这些规定也不尽相同。

例如 DB2 规定：

（1）若视图是由两个以上基本表导出的，则此视图不允许更新。

（2）若视图的字段来自字段表达式或常数，则不允许对此视图执行 INSERT 和 UPDATE 操作，但允许执行 DELETE 操作。

（3）若视图的字段来自集函数，则此视图不允许更新。

（4）若视图定义中含有 GROUP BY 子句，则此视图不允许更新。

（5）若视图定义中含有 DISTINCT 短语，则此视图不允许更新。

（6）若视图定义中有嵌套查询，并且内层查询的 FROM 子句中涉及的表也是导出该视图的基本表，则此视图不允许更新。例如将 SC 中成绩在平均成绩之上的元组定义成一个视图 GOOD_SC：

```
CREATE VIEW GOOD_SC
        AS SELECT  Sno，Cno，Grade
           FROM SC
           WHERE Grade >
                (SELECT AVG(Grade)
                 FROM  SC)；
```

导出视图 GOOD_SC 的基本表是 SC，内层查询中涉及的表也是 SC，所以视图 GOOD_SC 是不允许更新的。

（7）一个不允许更新的视图上定义的视图也不允许更新。

应该指出的是，不可更新的视图与不允许更新的视图是两个不同的概念。前者指理论上已证明其是不可更新的视图。后者指实际系统中不支持其更新，但它本身有可能是可更新的视图。

3.5.4 视图的作用

视图最终是定义在基本表之上的，对视图的一切操作最终也要转换为对基本表的操作。而且对于非行列子集视图进行查询或更新时还有可能出现问题。既然如此，为什么还要定义视图呢？这是因为合理使用视图能够带来许多好处。

1. 视图能够简化用户的操作

视图机制使用户可以将注意力集中在所关心的数据上。如果这些数据不是直接来自基本表，则可以通过定义视图，使数据库看起来结构简单、清晰，并且可以简化用户的数据查询操作。例如，那些定义了若干张表连接的视图，就将表与表之间的连接操作对用户隐蔽起来了。换句话说，用户所做的只是对一个虚表的简单查询，而这个虚表是怎样得来的，用户无需了解。

2. 视图使用户能以多种角度看待同一数据

视图机制能使不同的用户以不同的方式看待同一数据，当许多不同种类的用户共享同一个数据库时，这种灵活性是非常重要的。

3. 视图对重构数据库提供了一定程度的逻辑独立性

第一章中已经介绍过数据的物理独立性与逻辑独立性的概念。数据的物理独立性是指用户和用户程序不依赖于数据库的物理结构。数据的逻辑独立性是指当数据库重构造时，如增加新的关系或对原有关系增加新的字段等，用户和用户程序不会受影响。层次数据库和网状数据库一般能较好地支持数据的物理独立性，而对于逻辑独立性则不能完全地支持。

在关系数据库中，数据库的重构往往是不可避免的。重构数据库最常见的是将一个基本表“垂直”地分成多个基本表。例如：将学生关系

Student(Sno，Sname，Ssex，Sage，Sdept)

分为 SX(Sno，Sname，Sage)

和 SY(Sno，Ssex，Sdept)

两个关系。这时原表 Student 为 SX 表和 SY 表自然连接的结果。如果建立一个视图 Student：

```
CREATE VIEW   Student(Sno，Sname，Ssex，Sage，Sdept)
        AS
        SELECT   SX.Sno，SX.Sname，SY.Ssex，SX.Sage，SY.Sdept
        FROM    SX，SY
        WHERE   SX.Sno=SY.Sno;
```

这样尽管数据库的逻辑结构改变了，但应用程序不必修改，因为新建立的视图定义了用户原来的关系，使用户的外模式保持不变，用户的应用程序通过视图仍然能够查找数据。

当然，视图只能在一定程度上提供数据的逻辑独立性，比如由于对视图的更新是有条件的，因此应用程序中修改数据的语句可能仍会因基本表结构的改变而改变。

4. 视图能够对机密数据提供安全保护

有了视图机制，就可以在设计数据库应用系统时，对不同的用户定义不同的视图，使机密数据不出现在不应看到这些数据的用户视图上，这样视图机制就自动提供了对机密数据的安全保护功能。例如 Student 表涉及三个系的学生数据，可以在其上定义三个视图，每个视图只包含一个系的学生数据，并只允许每个系的系主任查询自己系的学生视图。

3.6 数 据 控 制

由 DBMS 提供统一的数据控制功能是数据库系统的特点之一。SQL 中数据控制功能包括事务管理功能和数据保护功能，即数据库的恢复、并发控制；

数据库的安全性和完整性控制。这些概念和技术将在后面章节详细讨论。

SQL 语言定义完整性约束条件的功能主要体现在 CREATE TABLE 语句和 ALTER TABLE 中，可以在这些语句中定义码、取值唯一的列、不允许空值的列、外码（参照完整性）及其他一些约束条件。

SQL 语言也提供了并发控制及恢复的功能，支持事务、提交、回滚等概念，SQL 语言在这方面的能力将在后面章节做进一步介绍。

这里主要讨论 SQL 语言的安全性控制功能。

某个用户对某类数据具有何种操作权力是个政策问题而不是技术问题。数据库管理系统的功能是保证这些决定的执行。为此 DBMS 必须具有以下功能：

（1）把授权的决定告知系统，这是由 SQL 的 GRANT 和 REVOKE 语句来完成的。

（2）把授权的结果存入数据字典。

（3）当用户提出操作请求时，根据授权情况进行检查，以决定是否执行操作请求。

3.6.1 授权

SQL 语言用 GRANT 语句向用户授予操作权限，GRANT 语句的一般格式为：

```
GRANT <权限>[，<权限>]…
    [ON <对象类型> <对象名>]
    TO <用户>[，<用户>]…
    [WITH GRANT OPTION];
```

其语义为：将对指定操作对象的指定操作权限授予指定的用户。

对不同类型的操作对象有不同的操作权限，常见的操作权限如表 3.4 所示。

表 3.4 不同对象类型允许的操作权限

对 象	对象类型	操 作 权 限
属性列	TABLE	SELECT，INSERT，UPDATE，DELETE ，ALL PRIVILEGES
视 图	TABLE	SELECT，INSERT，UPDATE，DELETE ，ALL PRIVILEGES
基本表	TABLE	SELECT，INSERT，UPDATE，DELETE ，ALTER，INDEX，ALL PRIVILEGES
数据库	DATABASE	CREATETAB

对属性列和视图的操作权限有：查询（SELECT）、插入（INSERT）、修改（UPDATE）、删除（DELETE）以及这四种权限的总和（ALL PRIVILEGES）。

对基本表的操作权限有：查询（SELECT）、插入（INSERT）、修改（UPDATE）、删除（DELETE）、修改表（ALTER）和建立索引（INDEX）以及这六种权限的总和（ALL PRIVILEGES）。

对数据库可以有建立表（CREATETAB）的权限，该权限属于 DBA，可由 DBA 授予普通用户，普通用户拥有此权限后可以建立基本表，基本表的属主（Owner）拥有对该表的一切操作权限。

接受权限的用户可以是一个或多个具体用户，也可以是 PUBLIC，即全体用户。

如果指定了 WITH GRANT OPTION 子句，则获得某种权限的用户还可以把这种权限再授予其他的用户。如果没有指定 WITH GRANT OPTION 子句，则获得某种权限的用户只能使用该权限，但不能传播该权限。

例 1　把查询 Student 表的权限授给用户 U1。

```
GRANT SELECT
ON TABLE Student
TO U1;
```

例 2　把对 Student 表和 Course 表的全部操作权限授予用户 U2 和 U3。

```
GRANT ALL PRIVILEGES
ON TABLE Student，Course
TO U2，U3;
```

例 3　把对表 SC 的查询权限授予所有用户。

```
GRANT SELECT
ON TABLE SC
TO PUBLIC;
```

例 4　把查询 Student 表和修改学生学号的权限授给用户 U4。

```
GRANT UPDATE(Sno)，SELECT ON TABLE Student TO U4;
```

这里实际上要授予 U4 用户的是对基本表 Student 的 SELECT 权限和对属性列 Sno 的 UPDATE 权限。授予关于属性列的权限时必须明确指出相应属性列名。

例 5　把对表 SC 的 INSERT 权限授予 U5 用户，并允许将此权限再授予其他用户。

```
GRANT INSERT ON TABLE SC TO U5 WITH GRANT OPTION;
```

执行此 SQL 语句后，U5 不仅拥有了对表 SC 的 INSERT 权限，还可以传

播此权限，即由 U5 用户使用上述 GRANT 命令给其他用户授权。例如 U5 可以将此权限授予 U6：

```
GRANT INSERT ON TABLE SC TO U6 WITH GRANT OPTION;
```

同样，U6 还可以将此权限授予 U7：

```
GRANT INSERT ON TABLE SC TO U7;
```

因为 U6 未给 U7 传播的权限，因此 U7 不能再传播此权限。

例 6 DBA 把在数据库 S_C 中建立表的权限授予用户 U8。

```
GRANT CREATETAB ON DATABASE S_C TO U8;
```

由上面的例子可以看到，GRANT 语句可以一次向一个用户授权，如例 1 所示，这是最简单的一种授权操作；也可以一次向多个用户授权，如例 2、例 3 等所示；还可以一次传播多个同类对象的权限，如例 2 所示；甚至一次可以完成对基本表、视图和属性列这些不同对象的授权，如例 4 所示；但授予关于 DATABASE 的权限必须与授予关于 TABLE 的权限分开，因为对象类型不同。

3.6.2 收回权限

授予的权限可以由 DBA 或其他授权者用 REVOKE 语句收回，REVOKE 语句的一般格式为：

```
REVOKE <权限>[ ，<权限>]…
    [ON <对象类型> <对象名>]
    FROM <用户>[ ，<用户>]…;
```

例 7 把用户 U4 修改学生学号的权限收回

```
REVOKE UPDATE(Sno) ON TABLE Student FROM U4;
```

例 8 收回所有用户对表 SC 的查询权限

```
REVOKE SELECT ON TABLE SC FROM PUBLIC;
```

例 9 把用户 U5 对 SC 表的 INSERT 权限收回

```
REVOKE INSERT ON TABLE SC FROM U5;
```

在例 5 中，U5 将对 SC 表的 INSERT 权限授予了 U6，而 U6 又将其授予了 U7。执行例 9 REVOKE 语句后，DBMS 在收回 U5 对 SC 表的 INSERT 权限的同时，还会自动收回 U6 和 U7 对 SC 表的 INSERT 权限，即收回权限的操作会级联下去的。但如果 U6 或 U7 还从其他用户处获得对 SC 表的 INSERT 权限，则他们仍具有此权限，系统只收回直接或间接从 U5 处获得的权限。

可见，SQL 提供了非常灵活的授权机制。DBA 拥有对数据库中所有对象

的所有权限，并可以根据应用的需要将不同的权限授予不同的用户。

用户对自己建立的基本表和视图拥有全部的操作权限，并且可以用GRANT语句把其中某些权限授予其他用户。被授权的用户如果有“继续授权”的许可，还可以把获得的权限再授予其他用户。

所有授予出去的权力在必要时又都可以用REVOKE语句收回。

3.7 嵌 入 式 SQL

以上介绍的SQL语言是作为独立语言在终端交互方式下使用的。这是面向集合的描述性语言，是非过程性的。即大多数语句都是独立执行，与上下文无关的。而许多事务处理应用都是过程性的，需要根据不同的条件来执行不同的任务，因此单纯用SQL语言是很难实现这类应用的。

为了解决这一问题，SQL语言提供了另一种使用方式，即将SQL语言嵌入到某种高级语言中使用，利用高级语言的过程性结构来弥补SQL语言实现复杂应用方面的不足。这种方式下使用的SQL语言称为嵌入式SQL（Embedded SQL），而嵌入SQL的高级语言称为主语言或宿主语言。

前面已经讲到，SQL的特点之一是，在两种不同的使用方式下，SQL语言的语法结构基本上是一致的。当然细节上会有许多差别，在程序设计的环境下，SQL语句要做某些必要的扩充。

3.7.1 嵌入式SQL的一般形式

对宿主型数据库语言SQL，DBMS可采用两种方法处理，一种是预编译，另一种是修改和扩充主语言使之能处理SQL语句。目前采用较多的是预编译的方法。即由DBMS的预处理程序对源程序进行扫描，识别出SQL语句，把它们转换成主语言调用语句，以使主语言编译程序能识别它，最后由主语言的编译程序将整个源程序编译成目标码。

在嵌入式SQL中，为了能够区分SQL语句与主语言语句，所有SQL语句都必须加前缀EXEC SQL。SQL语句的结束标志则随主语言的不同而不同。

例如在PL/1和C中以分号（；）结束：

```
EXEC SQL <SQL 语句>;
```

在COBOL中以END-EXEC结束：

```
EXEC SQL <SQL 语句> END-EXEC
```

例如一条交互形式的SQL语句：

```
DROP TABLE  Student;
```

嵌入到 C 程序中，应写作：

```
EXEC SQL DROP TABLE  Student;
```

嵌入 SQL 语句根据其作用的不同，可分为可执行语句和说明性语句两类。可执行语句又分为数据定义、数据控制、数据操纵三种。

在宿主程序中，任何允许出现可执行的高级语言语句的地方，都可以写可执行 SQL 语句；任何允许出现说明性高级语言语句的地方，都可以写说明性 SQL 语句。

3.7.2 嵌入式 SQL 语句与主语言之间的通信

将 SQL 嵌入到高级语言中混合编程，SQL 语句负责操纵数据库，高级语言语句负责控制程序流程。这时程序中会含有两种不同计算模型的语句，一种是描述性的面向集合的 SQL 语句，一种是过程性的高级语言语句，它们之间应该如何通信呢？

数据库工作单元与源程序工作单元之间的通信主要包括：

（1）向主语言传递 SQL 语句的执行状态信息，使主语言能够据此信息控制程序流程，主要用 SQL 通信区（SQL Communication Area，简称 SQLCA）实现；

（2）主语言向 SQL 语句提供参数，主要用主变量（Host Variable）实现；

（3）将 SQL 语句查询数据库的结果交主语言进一步处理，主要用主变量和游标（Cursor）实现。

一、SQL 通信区

SQL 语句执行后，系统要反馈给应用程序若干信息，主要包括描述系统当前工作状态和运行环境的各种数据。这些信息将送到 SQL 通信区 SQLCA 中。应用程序从 SQLCA 中取出这些状态信息，据此决定接下来执行的语句。

SQLCA 是一个数据结构，在应用程序中用 EXEC SQL INCLUDE SQLCA 加以定义。SQLCA 中有一个存放每次执行 SQL 语句后返回代码的变量 SQLCODE。应用程序每执行完一条 SQL 语句之后都应该测试一下 SQLCODE 的值，以了解该 SQL 语句执行情况并做相应处理。如果 SQLCODE 等于预定义的常量 SUCCESS，则表示 SQL 语句成功，否则在 SQLCODE 中存放错误代码。

例如，在执行删除语句 DELETE 后，不同的执行情况 SQLCA 中有下列不同的信息：

- 成功删除，并有删除的行数（SQLCODE=SUCCESS）
- 无条件删除警告信息
- 违反数据保护规则，拒绝操作
- 没有满足条件的行，一行也没有删除
- 由于各种原因，执行出错

二、主变量

嵌入式 SQL 语句中可以使用主语言的程序变量来输入或输出数据。我们把 SQL 语句中使用的主语言程序变量简称为**主变量**。

主变量根据其作用的不同，分为输入主变量和输出主变量。**输入主变量**由应用程序对其赋值，SQL 语句引用；**输出主变量**由 SQL 语句对其赋值或设置状态信息，返回给应用程序。一个主变量有可能既是输入主变量又是输出主变量。利用输入主变量，可以指定向数据库中插入的数据，可以将数据库中的数据修改为指定值，可以指定执行的操作，可以指定 WHERE 子句或 HAVING 子句中的条件。利用输出主变量，可以得到 SQL 语句的结果数据和状态。

一个主变量可以附带一个任选的指示变量（Indicator Variable）。**指示变量**是一个整型变量，用来“指示”所指主变量的值或条件。指示变量可以指示输入主变量是否为空值，可以检测输出主变量是否空值，值是否被截断。

所有主变量和指示变量必须在 SQL 语句 BEGIN DECLARE SECTION 与 END DECLARE SECTION 之间进行说明。说明之后，主变量可以在 SQL 语句中任何一个能够使用表达式的地方出现，为了与数据库对象名（表名、视图名、列名等）区别，SQL 语句中的主变量名前要加冒号（:）作为标志。同样，SQL 语句中的指示变量前也必须加冒号，并且要紧跟在所指主变量之后。而在 SQL 语句之外，主变量和指示变量均可以直接引用，不必加冒号。

三、游标

SQL 语言与主语言具有不同的数据处理方式。SQL 语言是面向集合的，一条 SQL 语句原则上可以产生或处理多条记录。而主语言是面向记录的，一组主变量一次只能存放一条记录。所以仅使用主变量并不能完全满足 SQL 语句向应用程序输出数据的要求，为此嵌入式 SQL 引入了游标的概念，用游标来协调这两种不同的处理方式。游标是系统为用户开设的一个数据缓冲区，存放 SQL 语句的执行结果 ，每个游标区都有一个名字。用户可以通过游标逐一

获取记录，并赋给主变量，交由主语言进一步处理。

四、程序实例

可以看到，在嵌入式 SQL 中，SQL 语句与主语言语句分工非常明确。SQL 语句用来直接与数据库打交道，主语言语句用来控制程序流程以及对 SQL 语句的执行结果做进一步加工处理。SQL 语句用主变量从主语言中接收执行参数，操纵数据库；SQL 语句的执行状态由 DBMS 送至 SQLCA 中；主语言程序从 SQLCA 中取出状态信息，据此决定下一步操作；如果 SQL 语句从数据库中成功地检索出数据，则通过主变量传给主语言做进一步处理。SQL 语言和主语言的不同数据处理方式通过游标来协调。这实际上反映了嵌入式 SQL 的工作原理。

为了能够更好地理解上面的概念，下面给出带有嵌入式 SQL 的一小段 C 程序。

```
.....
.....
EXEC SQL INCLUDE SQLCA; .....(1) 定义 SQL 通信区
EXEC SQL BEGIN DECLARE SECTION; .....(2) 主变量说明开始
    CHAR   Sno(5);
    CHAR   Cno(3);
    INT    Grade;
EXEC SQL END DECLARE SECTION; .....主变量说明结束
main()
{
    EXEC SQL DECLARE C1 CURSOR FOR.....(3) 游标操作（定义游标）
        SELECT Sno，Cno，Grade
        FROM SC;              /*从表中查询 Sno，Cno，Grade */
    EXEC SQL OPEN C1; .....(4) 游标操作（打开游标）
    for(；；)
    {
      EXEC SQL FETCH C1 INTO :Sno，: Cno，: Grade;
                                  .....(5) 游标操作（推进游标指针并
                                           将当前数据放入主变量）
        if (sqlca.sqlcode < > SUCCESS)  .....(6) 利用 SQLCA 中的状态信息
                                           决定何时退出循环
```

```
            break;
        printf("Sno: %s, Cno: %s Grade: %d", :Sno, :Cno, :Grade);
                                                /* 打印查询结果 */
    }
    EXEC SQL CLOSE C1; .....(7) 游标操作（关闭游标）
}
```

3.7.3 不用游标的 SQL 语句

不用游标的 SQL 语句有：

- 说明性语句
- 数据定义语句
- 数据控制语句
- 查询结果为单记录的 SELECT 语句
- 非 CURRENT 形式的 UPDATE 语句
- 非 CURRENT 形式的 DELETE 语句
- INSERT 语句

所有的说明性语句及数据定义与控制语句都不需要使用游标。它们是嵌入式 SQL 中最简单的一类语句，不需要返回结果数据，也不需要使用主变量。在主语言中嵌入说明性语句及数据定义与控制语句，只要给语句加上前缀 EXEC SQL 和语句结束符即可。

INSERT 语句也不需要使用游标，但通常需要使用主变量。

SELECT 语句、UPDATE 语句、DELETE 语句则要复杂些。

一、说明性语句

说明性语句是专为在嵌入式 SQL 中说明主变量等而设置的，主要有两条语句：

```
EXEC SQL BEGIN DECLARE SECTION;
```

和

```
EXEC SQL END DECLARE SECTION;
```

两条语句必须配对出现，相当于一个括号，两条语句中间是主变量的说明。

二、数据定义语句

例 1 建立一个“学生”表 Student。

```
EXEC SQL CREATE TABLE Student
            (Sno    CHAR(5) NOT NULL UNIQUE,
             Sname  CHAR(20),
             Ssex   CHAR(1),
             Sage   INT,
             Sdept  CHAR(15));
EXEC SQL DROP TABLE Student;
```

数据定义语句中不允许使用主变量。例如下列语句是错误的：

```
EXEC SQL DROP TABLE :table_name;
```

三、数据控制语句

例 2 把查询 Student 表权限授给用户 U1。

```
EXEC SQL GRANT SELECT ON TABLE Student TO U1;
```

四、查询结果为单记录的 SELECT 语句

在嵌入式 SQL 中，查询结果为单记录的 SELECT 语句需要用 INTO 子句指定查询结果的存放地点。该语句的一般格式为：

```
EXEC SQL SELECT  [ALL|DISTINCT] <目标列表达式>[，<目标列表达式>]…
         INTO <主变量>[<指示变量>] [，<主变量>[<指示变量>] ]…
         FROM  <表名或视图名>[，<表名或视图名>]…
         [WHERE <条件表达式>]
         [GROUP BY <列名 1> [HAVING <条件表达式>] ]
         [ORDER BY <列名 2> [ASC|DESC]];
```

该语句对交互式 SELECT 语句的扩充就是多了一个 INTO 子句。把从数据库中找到的符合条件的记录，放到 INTO 子句指出的主变量中去。其他子句的含义不变。使用该语句需要注意以下几点：

（1）INTO 子句、WHERE 子句的条件表达式、HAVING 短语的条件表达式中均可以使用主变量。

（2）查询返回的记录中，可能某些列为空值 NULL。如果 INTO 子句中主变量后面跟有指示变量，则当查询得出的某个数据项为空值时，系统会自动将相应主变量后面的指示变量置为负值，而不再向该主变量赋值，即主变量值

仍为执行 SQL 语句之前的值。所以当指示变量值为负值时，不管主变量为何值，均应认为主变量值为 NULL。指示变量只能用于 INTO 子句中。

（3）如果数据库中没有满足条件的记录，则 DBMS 将 SQLCODE 的值置为 100。

（4）如果查询结果实际上并不是单条记录，而是多条记录，则程序出错，DBMS 会在 SQLCA 中返回错误信息。

例 3 根据学生号码查询学生信息。假设已将要查询的学生的学号赋给了主变量 givensno。

```
EXEC SQL SELECT Sno，Sname，Ssex，Sage，Sdept
         INTO  :Hsno，:Hname，:Hsex，:Hage，:Hdept
         FROM  Student
         WHERE Sno=:givensno;
```

例 4 查询某个学生选修某门课程的成绩。假设已将要查询的学生的学号赋给了主变量 givensno，将课程号赋给了主变量 givencno。

```
EXEC SQL SELECT Sno，Cno，Grade
         INTO :Hsno，:Hcno，:Hgrade:Gradeid
         FROM SC
         WHERE Sno=:givensno AND  Cno=:givencno;
```

由于学生选修一门课后有可能没有参加考试，也就是说其成绩为空值，所以在 INTO 子句中加了指示变量 Gradeid，用于指示主变量 Hgrade 是否为空值。执行此语句后，如果 Gradeid 小于 0，则不论 Hgrade 为何值，均认为该学生成绩为空值。

五、非 CURRENT 形式的 UPDATE 语句

在 UPDATE 语句中，SET 子句和 WHERE 子句中均可以使用主变量，其中 SET 子句中还可以使用指示变量。

例 5 将全体学生 1 号课程的考试成绩增加若干分。假设增加的分数已赋给主变量 Raise。

```
EXEC SQL UPDATE SC
         SET Grade=Grade+:Raise
         WHERE Cno='1';
```

例 6 修改某个学生 1 号课程的成绩。假设该学生的学号已赋给主变量 givensno，修改后的成绩已赋给主变量 newgrade。

```
EXEC SQL UPDATE SC
```

```
        SET Grade=:newgrade
        WHERE Sno=:givensno;
```

例 7 将计算机系全体学生年龄置 NULL 值。

```
Sageid=-1;
EXEC SQL UPDATE Student
        SET Sage=:Raise:Sageid
        WHERE Sdept= 'CS';
```

将指示变量 Sageid 赋一个负值后，无论主变量 Raise 为何值，DBMS 都会将 CS 系所有学生的年龄置空值。它等价于：

```
EXEC SQL UPDATE Student
        SET Sage=NULL
        WHERE Sdept= 'CS';
```

六、非 CURRENT 形式的 DELETE 语句

DELETE 语句的 WHERE 子句中可以使用主变量指定删除条件。

例 8 某个学生退学了，现要将有关他的所有选课记录删除掉。假设该学生的姓名已赋给主变量 stdname。

```
EXEC SQL DELETE
        FROM SC
        WHERE Sno=
                (SELECT Sno
                 FROM Student
                WHERE Sname=:stdname);
```

另一种等价实现方法为：

```
EXEC SQL DELETE
        FROM SC
        WHERE :stdname=
                (SELECT Sname
                 FROM Student
                WHERE Student.Sno=SC.Sno);
```

第 1 种方法更直接，从而也更高效些。

七、INSERT 语句

INSERT 语句的 VALUES 子句中可以使用主变量和指示变量。

例 9 某个学生新选修了某门课程，将有关记录插入 SC 表中。假设学生

的学号已赋给主变量 stdno，课程号已赋给主变量 couno。

```
gradeid=-1;
EXEC SQL INSERT
        INTO SC(Sno，Cno，Grade)
        VALUES(:stdno :couno，:gr:gradeid);
```

由于该学生刚选修课程，尚未考试，成绩应为空，所以要把指示变量:gradeid 赋为负值。

3.7.4 使用游标的 SQL 语句

必须使用游标的 SQL 语句有：

- 查询结果为多条记录的 SELECT 语句
- CURRENT 形式的 UPDATE 语句
- CURRENT 形式的 DELETE 语句

一、查询结果为多条记录的 SELECT 语句

一般情况下，SELECT 语句查询结果都是多条记录，而高级语言一次只能处理一条记录，因此需要用游标机制，将多条记录一次一条送至宿主程序处理，从而把对集合的操作转换为对单个记录的处理。

使用游标的步骤为：

（1）说明游标。用 DECLARE 语句为一条 SELECT 语句定义游标。DECLARE 语句的一般形式为：

```
EXEC SQL DECLARE <游标名> CURSOR FOR <SELECT 语句>;
```

其中 SELECT 语句可以是简单查询，也可以是复杂的连接查询和嵌套查询。

定义游标仅仅是一条说明性语句，这时 DBMS 并不执行 SELECT 语句。

（2）打开游标。用 OPEN 语句将定义的游标打开。OPEN 语句的一般形式为：

```
EXEC SQL OPEN <游标名>;
```

打开游标实际上是执行相应的 SELECT 语句，把查询结果取到缓冲区中。这时游标处于活动状态，指针指向查询结果集中第 1 条记录。

（3）推进游标指针并取当前记录。用 FETCH 语句把游标指针向前推进一条记录，同时将缓冲区中的当前记录取出来送至主变量供主语言进一步处理。FETCH 语句的一般形式为：

```
EXEC SQL FETCH <游标名>
        INTO <主变量>[<指示变量>] [，<主变量>[<指示变量>] ]…;
```

其中主变量必须与 SELECT 语句中的目标列表达式具有一一对应关系。

FETCH 语句通常用在一个循环结构中，通过循环执行 FETCH 语句逐条取出结果集中的行进行处理。

为进一步方便用户处理数据，现在许多关系数据库管理系统对 FETCH 语句做了扩充，允许向任意方向以任意步长移动游标指针，而不仅仅是把游标指针向前推进一行了。

（4）关闭游标。用 CLOSE 语句关闭游标，释放结果集占用的缓冲区及其他资源。CLOSE 语句的一般形式为：

```
EXEC SQL CLOSE <游标名>;
```

游标被关闭后，就不再和原来的查询结果集相联系。但被关闭的游标可以再次被打开，与新的查询结果相联系。

例 1 查询某个系全体学生的信息。要查询的系名由用户在程序运行过程中指定，放在主变量 deptname 中。

```
......
......
EXEC SQL BEGIN DECLARE SECTION;
......
/* 说明主变量 deptname，HSno，HSname，HSsex，HSage 等*/
......
EXEC SQL END DECLARE SECTION;
......
gets(deptname);                                  /* 为主变量 deptname 赋值 */
......
EXEC SQL DECLARE SX CURSOR FOR          /* 说明游标 SX，将 SX
        SELECT Sno，Sname，Ssex，Sage      与查询结果集相联系*/
        FROM    Student
        WHERE   SDept=:deptname;
EXEC SQL OPEN SX                                 /* 打开游标 */
WHILE(1)                                  /* 用循环结构逐条处理结果集中的记录 */
{
  EXEC SQL FETCH SX INTO :HSno，:HSname，:HSsex，:HSage;
            /* 游标指针向前推进一行，从结果集中取当前行，送相应主变量 */
  if (sqlca.sqlcode < > SUCCESS)              /* 若所有查询结果均已处理完或
    break;                                      出现错误，则退出循环 */
    ......                                    /* 由主语言语句进行进一步处理 */
```

```
};
EXEC SQL CLOSE SX;                    /* 关闭游标 SX */
......
```

被关闭的游标 SX 实际上可以再次被打开，与新的查询结果相联系。例如，可以在例 1 中再加上一层外循环，每次对 deptname 赋新的值，这样 SX 就每次和不同的系的学生集合相联系。如例 2 所示。

例 2　查询某些系全体学生的选课信息。

```
......
EXEC SQL BEGIN DECLARE SECTION;
    ......
        /* 说明主变量 deptname，HSno，HSname，HSsex，HSage 等*/
    ......
EXEC SQL END DECLARE SECTION;
......
EXEC SQL DECLARE SX CURSOR FOR          /* 说明游标 SX，将 SX
        SELECT Sno，Sname，Ssex，Sage       与查询结果集相联系*/
        FROM Student
        WHERE SDept=:deptname;
WHILE (gets(deptname)!=NULL)            /* 接收主变量 deptname 的值 */
{   /* 处理 deptname 指定系的学生信息，每次循环中 deptname 可具有不同的值 */
    EXEC SQL OPEN SX                    /* 每次打开游标 SX 就和
                                           不同系的学生集合相联系*/
    WHILE (1) {                         /* 用循环结构逐条处理结果集中的记录 */
        EXEC SQL FETCH SX INTO :HSno，:HSname，:HSsex，:HSage;
      /* 游标指针向前推进一行，然后从结果集中取当前行，送相应主变量 */

      if (sqlca.sqlcode <> SUCCESS)     /* 若所有查询结果均已处理完或
        break;                             出现错误，则退出循环 */
    ......                              /* 由主语言语句进行进一步处理 */
    };                                  /* 内循环结束 */
    EXEC SQL CLOSE SX;                  /* 关闭游标 */
};                                      /* 外循环结束 */
......
```

二、CURRENT 形式的 UPDATE 语句和 DELETE 语句

UPDATE 语句和 DELETE 语句都是集合操作，如果只想修改或删除其中某个记录，则需要用带游标的 SELECT 语句查出所有满足条件的记录，从中进一步找出要修改或删除的记录，然后用 CURRENT 形式的 UPDATE 和 DELETE 语句修改或删除之。具体步骤是：

（1）用 DECLARE 语句说明游标。如果是为 CURRENT 形式的 UPDATE 语句作准备，则 SELECT 语句中要用

FOR UPDATE OF <列名>

用来指明检索出的数据在指定列是可修改的。如果是为 CURRENT 形式的 DELETE 语句作准备，则不必使用上述子句。

（2）用 OPEN 语句打开游标，把所有满足查询条件的记录从指定表取到缓冲区中。

（3）用 FETCH 语句推进游标指针，并把当前记录从缓冲区中取出来送至主变量。

（4）检查该记录是否是要修改或删除的记录。如果是，则用 UPDATE 语句或 DELETE 语句修改或删除该记录。这时 UPDATE 语句和 DELETE 语句中要用子句

WHERE CURRENT OF <游标名>

来表示修改或删除的是最近一次取出的记录，即游标指针指向的记录。

第(3)和(4)步通常用在一个循环结构中，通过循环执行 FETCH 语句，逐条取出结果集中的行进行判断和处理。

（5）处理完毕用 CLOSE 语句关闭游标，释放结果集占用的缓冲区和其他资源。

例 3 查询某个系全体学生的信息（要查询的系名由主变量 deptname 指定），然后根据用户的要求修改其中某些记录的年龄字段。

```
.....
EXEC SQL BEGIN DECLARE SECTION;
    .....
/* 说明主变量 deptname，HSno，HSname，HSsex，HSage，NEWAge 等*/
    .....
EXEC SQL END DECLARE SECTION;
.....
gets(deptname);                          /* 为主变量 deptname 赋值 */
EXEC SQL DECLARE SX CURSOR FOR
```

```
        SELECT Sno，Sname，Ssex，Sage
        FROM Student
        WHERE SDept=:deptname
        FOR UPDATE OF Sage;          /* 说明游标，为 CURRENT UPDATE
                                     作准备 */
EXEC SQL OPEN SX                     /* 打开游标 */
WHILE(1)  {                          /* 用循环结构逐条处理结果集中的记录 */
    EXEC SQL FETCH SX INTO :HSno，:HSname，:HSsex，:HSage;
        /* 游标指针向前推进一行，然后从结果集中取当前行，送相应主变量 */
    if (sqlca.sqlcode <> SUCCESS)
        break;
          /* 若所有查询结果均已处理完或出现 SQL 语句错误，则退出循环 */
    printf( "%s，%s，%s，%d"，Sno，Sname，Ssex，Sage);    /* 显示该记录 */
    printf("UPDATE AGE ? ");                    /* 问用户是否要修改 */
    scanf("%c"，&yn);
    if (yn= 'y' or yn= 'Y')                     /* 需要修改 */
    {
        printf("INPUT NEW AGE: ");
        scanf( "%d"，&NEWAge);                  /* 输入新的年龄值 */
        EXEC SQL UPDATE Student
            SET Sage = :NEWAge
            WHERE CURRENT OF SX;    /* 修改当前记录的年龄字段 */
    };
    ……
};
EXEC SQL CLOSE SX;                   /* 关闭游标 */
……
```

例 4　查询某个系全体学生的信息（要查询的系名由主变量 deptname 指定），然后根据用户的要求修改删除其中某些记录。

```
……
EXEC SQL BEGIN DECLARE SECTION;
    ……
    /* 说明主变量 deptname，HSno，HSname，HSsex，HSage 等*/
    ……
EXEC SQL END DECLARE SECTION;
```

```
.....
gets(deptname);                          /* 为主变量 deptname 赋值 */
.....
EXEC SQL DECLARE SX CURSOR FOR
        SELECT Sno，Sname，Ssex，Sage
        FROM  Student
        WHERE SDept=:deptname;                    /* 说明游标 */
EXEC SQL OPEN SX                                  /* 打开游标 */
WHILE(1){                              /* 用循环结构逐条处理结果集中的记录 */
    EXEC SQL FETCH SX INTO :HSno，:HSname，:HSsex，:HSage;
        /* 游标指针向前推进一行，然后从结果集中取当前行，送相应主变量 */

    if (sqlca.sqlcode <> SUCCESS)
        break;
        /* 若所有查询结果均已处理完或出现 SQL 语句错误，则退出循环 */
        printf("%s，%s，%s，%d"，Sno，Sname，Ssex，Sage);
                                          /* 显示该记录 */
        printf("DELETE ? ");                /* 问用户是否要删除 */
        scanf("%c"，&yn);
        if (yn= 'y' or yn= 'Y')             /* 需要删除 */
          EXEC SQL DELETE
                FROM Student
                WHERE CURRENT OF SX;        /* 删除当前记录 */
    .....
};
EXEC SQL CLOSE SX;                                /* 关闭游标 */
.....
```

注意：当游标定义中的 SELECT 语句带有 UNION 或 ORDER BY 子句时，或者该 SELECT 语句相当于定义了一个不可更新的视图时，不能使用 CURRENT 形式的 UPDATE 语句和 DELETE 语句。

3.7.5 动态 SQL 简介

在 3.7.3 节和 3.7.4 节中介绍的嵌入式 SQL 语句为编程提供了一定的灵活性，使用户可以在程序运行过程中根据实际需要输入 WHERE 子句或 HAVING 子句中某些变量的值。这些 SQL 语句的共同特点是，语句中主变量的个数与

数据类型在预编译时都是确定的，只有主变量的值是程序运行过程中动态输入的，称这类嵌入式 SQL 语句为静态 SQL 语句。

静态 SQL 语句提供的编程灵活性在许多情况下仍显得不足，有时候需要编写更为通用的程序。例如，对 SC 表，任课教师想查询选修了他所授课程的学生学号及其成绩；班主任想查询某个学生选修的课程号及相应成绩；学生想查询自己某门课程的成绩。也就是说查询条件、要查询的属性列是不确定的，这时就无法用一条静态 SQL 语句实现了。

如果在预编译时下列信息不能确定，就必须使用动态 SQL 技术 ，如：

（1）SQL 语句正文

（2）主变量个数

（3）主变量的数据类型

（4）SQL 语句中引用的数据库对象（例如列、索引、基本表、视图等）

动态 SQL 方法允许在程序运行过程中临时“组装”SQL 语句，主要有三种形式：

1. 语句可变

允许用户在程序运行时临时输入完整的 SQL 语句。

2. 条件可变

对于非查询语句，条件子句有一定的可变性。例如删除学生选课记录，既可以是因某门课临时取消，需要删除有关该课程的所有选课记录，也可以是因为某个学生退学，需要删除该学生的所有选课记录。

对于查询语句，SELECT 子句是确定的，即语句的输出是确定的，其他子句（如 WHERE 子句、HAVING 短语）有一定的可变性。例如查询学生人数，可以是查询某个系的学生总数，查询某个性别的学生人数，查询某个年龄段的学生人数，查询某个系某个年龄段的学生人数等，这时 SELECT 子句的目标列表达式是确定的（COUNT（*）），但 WHERE 子句的条件是不确定的。

3. 数据库对象、查询条件均可变

对于查询语句，SELECT 子句中的列名、FROM 子句中的表名或视图名、WHERE 子句和 HAVING 短语中的条件等均可由用户临时构造，即语句的输入和输出可能都是不确定的。例如前面查询学生选课关系 SC 的例子。对于非查询语句，涉及的数据库对象及条件也是可变的。

这几种动态形式几乎可覆盖所有的可变要求。为了实现上述三种可变形式，SQL 提供了相应的语句，例如 EXECUTE IMMEDIATE，PREPARE，EXECUTE，DESCRIBE 等。使用动态 SQL 技术更多的是涉及程序设计方面的知识，而不是 SQL 语言本身，所以这里就不详细介绍了，有兴趣的读者可以参阅有关书籍。

3.8 小 结

本章系统而详尽地讲解了 SQL 语言。SQL 是关系数据库语言的工业标准。各个数据库厂商支持的 SQL 语言在遵循标准的基础上常常作不同的扩充或修改。本章介绍的是标准 SQL，因此，本章的绝大部分例子应能在不同的系统如 Oracle，Sybase，DB2， Informix，SQL Server 和 Easybase，Openbase，DM2，Cobase 等众多系统上运行，也许有的例子在某些系统上需要稍作修改后才能运行。

在讲解了 SQL 语言的同时进一步讲解了关系数据库系统的基本概念，使这些概念更加具体、更加丰富。

SQL 语言可以分为数据定义、数据查询、数据更新、数据控制四大部分，有时人们把数据更新称为数据操纵，或把数据查询与数据更新合称为数据操纵。本章用很大的篇幅介绍了四部分的内容。

视图是关系数据库系统中的重要概念，这是因为合理使用视图具有许多优点。

SQL 语言的数据查询功能是最丰富，也是最复杂的，读者应加强练习。

习 题

1. 试述 SQL 语言的特点。
2. 试述 SQL 的定义功能。
3. 用 SQL 语句建立第二章习题 5 中的四个表。
4. 针对上题中建立的四个表试用 SQL 语言完成第二章习题 5 中的查询。
5. 针对习题 3 中的四个表试用 SQL 语言完成以下各项操作：
   （1）找出所有供应商的姓名和所在城市。
   （2）找出所有零件的名称、颜色、重量。
   （3）找出使用供应商 S1 所供应零件的工程号码。
   （4）找出工程项目 J2 使用的各种零件的名称及其数量。
   （5）找出上海厂商供应的所有零件号码。
   （6）找出使用上海产的零件的工程名称。
   （7）找出没有使用天津产的零件的工程号码。
   （8）把全部红色零件的颜色改成蓝色。
   （9）由 S5 供给 J4 的零件 P6 改为由 S3 供应，请作必要的修改。

（10）从供应商关系中删除 S2 的记录，并从供应情况关系中删除相应的记录。

（11）请将 (S2，J6，P4，200) 插入供应情况关系。

6. 什么是基本表？什么是视图？两者的区别和联系是什么？

7. 试述视图的优点。

8. 所有的视图是否都可以更新？为什么？

9. 哪类视图是可以更新的，哪类视图是不可更新的？ 各举一例说明。

10. 试述某个你熟悉的实际系统中对视图更新的规定。

11. 请为三建工程项目建立一个供应情况的视图，包括供应商代码（SNO）、零件代码（PNO）、供应数量（QTY）。针对该视图完成下列查询：

（1）找出三建工程项目使用的各种零件代码及其数量。

（2）找出供应商 S1 的供应情况。

12. 针对习题 3 建立的表，用 SQL 语言完成以下各项操作：

（1）把对表 S 的 INSERT 权限授予用户张勇，并允许他再将此权限授予其他用户。

（2）把查询 SPJ 表和修改 QTY 属性的权限授给用户李天明。

13. 在嵌入式 SQL 中是如何区分 SQL 语句和主语言语句的？

14. 在嵌入式 SQL 中是如何解决数据库工作单元与源程序工作单元之间通讯的？

15. 在嵌入式 SQL 中是如何协调 SQL 语言的集合处理方式和主语言的单记录处理方式的？

本章参考文献

1 Boyce R，Chamberlin D D，Hammer M，et al．Specifying Queries as Relational Expressions．CACM，18:11，November 1975

（SQL 语言原名 SEQUEL，是 Boyce 等人 1975 年在文献[1]提出的 SQUARE (Specifying Queries as Relational Expressions)语言的基础上发展起来的。文献[2]对 SQUARE 语言的语法进行了修改，形成了 SEQUEL 语言。文献[3]对 SEQUEL 语言进行了进一步的改进，最后形成了 SQL 语言。SEQUEL 语言最早用于 IBM San Jose 研究中心设计的关系数据库系统 SYSTEM R。）

2 Chamberlin D D，Boyce R．SEQUEL：A Structured English Query Language. Proceedings of ACM SIGMOD Workshop on Data Description．Access and Control，May 1974

3 Chamberlin D D，et al．SEQUEL 2: A Unified Approach to Data Definition,Manipulation and Control,IBM Journal of Research and Development．20，No.6．November 1976

4 Reisner P．Use of Psychological Experimentation as an Aid to Development of a Query Language．TSE，3：3，May 1977

（文献[4]和[5]分析了 SQL 语言的优缺点。）

5 Date C J．A Critique of the SQL Database Language．ACM SIGMOD Record 14，No.3，

November 1984

6 Date C J. A Guide to DB2. Reading, Mass, Addison-Wesley, 1984

（C.J.Date 在文献[6]中描述了 DB2 系统中的 SQL 语言。）

7 [美]Date C J. 刘伶，张素训译. DB2 指南. 北京：科学出版社，1990

8 X3H2(American National Standards Database Committee). American National Standard Database Language SQL: Working Draft. Document X3H2-85-1, December 1984

9 ANSI: The Database Language SQL, Document ANSI X3.315, 1986

（1986 年美国国家标准局 ANSI 定义了 SQL 语言标准，详见文献[8]。）

10 Grifiths P P, Wade B W. An Authorization Mechanism for a Relational Database System. ACM TODS, Vol.1, No.3, Sept. 1976

11 Chamberlin D D. A Summary of User Experience with the SQL Data Sublanguage. Proc. International Conference on Databases, Aberdeen, Scotland, 1980

12 Date C J. The Outer Join. Proc.2nd International Conference on Database(ICOD-2), Cambridge, England, September 1983

13 Dayal U, Bernstein P. On the Updatability of Relational Views. in Proceedings of VLDB, 1978

（[12]和[13]讨论了 SQL 语言的视图修改问题。）

14 Keller A M. Updates to Relational Databases Through Views Involving Joins. In Improving Database Usability and Responsiveness(ed. P.Scheuermann). New York: Academic Press, 1982

第四章　关系系统及其查询优化

这一章包括两个内容，一是关系系统（关系数据库系统的简称），二是关系系统的查询优化。第一部分讨论关系系统的定义和分类；第二部分讨论关系系统中查询优化的概念、查询优化的基本原理和技术。

4.1　关 系 系 统

在第二章比较详细地讨论了关系模型的三个基本要素：关系数据结构、关系的完整性和三类等价的关系操作。

关系系统和关系模型是两个密切相关而又不同的概念。支持关系模型的数据库管理系统称为关系系统。这种说法很笼统，因为关系模型中并非每一部分都是同等重要的，所以并不苛求一个实际的关系数据库管理系统必须完全支持关系模型，也不苛求完全支持关系模型的系统才能称为关系系统。因此，可以给出一个关系系统的最小要求以及分类的定义。

4.1.1　关系系统的定义

一个系统可定义为关系系统，当且仅当它：

（1）支持关系数据库（关系数据结构）。

从用户观点看，数据库由表构成，并且只有表这一种结构。

（2）支持选择、投影和（自然）连接运算，对这些运算不必要求定义任何物理存取路径。

一个系统仅支持关系数据库而没有选择、投影和连接运算功能的，不能称为关系系统。

一个系统虽然支持这三种运算，但要求定义物理存取路径，如要求用户建立索引才能按索引字段检索记录，也不能称为关系系统。

当然并不要求关系系统的选择、投影、连接运算和关系代数的相应运算完全一样，而只要求有等价的这三种运算功能就行。

下面对关系系统的定义做几点解释：

（1）为什么关系系统除了要支持关系数据结构外，还必须支持选择、投影、连接运算呢？因为不支持这三种关系运算的系统，用户使用仍不方便，不能提高用户的生产率，而提高用户生产率正是关系系统的主要目标之一。

（2）为什么要求这三种运算不能依赖于物理存取路径呢？因为依赖物理存取路径来实现关系运算就降低或丧失了数据的物理独立性。不依赖物理存取路径来实现关系运算就要求关系系统自动地选择路径。为此，系统要进行查询优化，以获得较好的性能。这正是关系系统实施的关键技术。

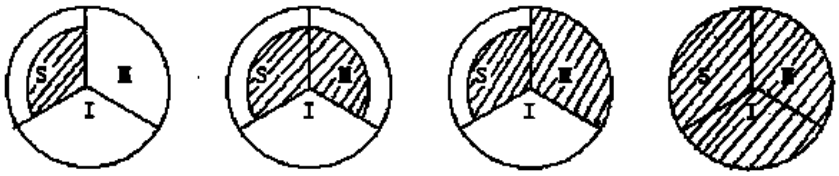
（3）要求关系系统支持这三种最主要的运算而不是关系代数的全部运算功能，是因为它们是最有用的运算功能，能解决绝大部分的实际问题。

4.1.2 关系系统的分类

有了关系系统的定义，就可以根据它来区分哪些系统是贴了“关系型”标签的非关系系统，哪些是关系系统。当前，许多产品如 DB2，Oracle，Sybase，Informix，Ingres 是关系系统。我国自主开发的 Pbase 和 Easybase（中国人民大学）、Cobase（北京大学、中国人民大学与中国软件总公司）、OPENBASE（东大阿尔派）、DM2（华中理工大学）等数据库管理系统是关系系统，这里不一一列举了。

在 4.1.1 节中定义的关系系统是关系系统的最小要求，许多实际系统，如上面提到的那些系统都不同程度地超过了这些要求。

按照 E.F.Codd 的思想，可以把关系系统分类，如图 4.1 所示。



(a) 表式系统　(b) (最小)关系系统　(c) 关系完备的　(d) 全关系的

图 4.1 关系系统的分类

图中的圆表示关系数据模型。每个圆分为三部分，分别表示关系模型的三个组成部分：结构 S(Structure)、完整性 I(Integrity)、数据操纵 M(Manipulation)。图中阴影部分表示各类系统支持模型的程度。

1. 表式系统

这类系统仅支持关系(即表)数据结构，不支持集合级的操作。表式系统不能算关系系统。倒排表列(Inverted List)系统就属于这一类。

2.（最小）关系系统

4.1.1 节中定义的关系系统，它们仅支持关系数据结构和三种关系操作。许多微机关系数据库系统如 FoxBASE，FoxPro 等就属于这一类。

3. 关系完备的系统

这类系统支持关系数据结构和所有的关系代数操作(功能上与关系代数等价)。20 世纪 90 年代初的许多关系数据库管理系统属于这一类。

4. 全关系系统

这类系统支持关系模型的所有特征。即不仅是关系上完备的而且支持数据结构中域的概念，支持实体完整性和参照完整性。目前，大多数关系系统已不同程度上接近或达到了这个目标。

4.1.3 全关系系统的十二条基本准则

全关系型的系统应该完全地支持关系模型的所有特征，这是个原则。关系模型的奠基人 E.F.Codd 具体地给出了全关系型的关系系统应遵循的十二条基本准则。从实际意义上看，这十二条准则可以作为评价或购买关系型产品的标准。从理论意义上看，它是对关系数据模型的具体而又深入的论述，是从理论和实际紧密结合的高度对关系型 DBMS 的评述。

准则 0 一个关系型的 DBMS 必须能完全通过它的关系能力来管理数据库。

准则 0 是下面十二条准则的基础。准则 0 的一个推论是：任何声称是关系型的 DBMS 必须在关系这个级别上支持数据的插入、修改和删除(即一次多个记录的操作级别)。

不管一个系统是否还具有非关系的管理数据的能力，它必须满足准则 0。任何不满足准则 0 的 DBMS 不配称为关系型 DBMS。

在一些“关系型”DBMS 产品的说明书中，常常建议用户“为取得可接受的性能”使用某些非关系的能力。这实际上是一种借口，表明这些系统没有为改善关系型系统的性能而进行必要的优化工作。违反准则 0 的“关系型”DBMS 给用户带来的危害是：用户不能得到真正的关系型 DBMS 所具有的优点。

准则 0 的另一个推论是：关系型 DBMS 必须遵循信息准则和保证访问(存取)准则。

准则 1 信息准则。关系型 DBMS 的所有信息都应在逻辑一级上用一种方法即表中的值显式地表示。

进一步地，表名、列名和域名等都用系统内部表(即数据字典表)中的值表示。数据字典本身是一个动态的用来描述元数据的关系数据库。

满足信息准则不仅是为了提高用户的生产率，而且也是为了使软件厂商在

定义其他软件包(例如应用软件辅助开发工具、专家系统等)时更加简便合理。这些软件包可以与关系型 DBMS 接口，并与 DBMS 融合成一体。这样，这些软件包能够检索存储在数据字典中的信息，需要的话可以借助 DBMS 的操作把新的信息存入字典中。

要求满足信息准则的另一个原因是使得 DBA 维护数据库的工作更简单、更有效。因为 DBA 感到最困难的是用户向他们询问数据库中是否存有某种信息。一些非关系型系统的 DBA 往往无法对此作出确定的回答，而关系型 DBMS 完全可以解答这些查询。

准则 2 保证访问准则。依靠表名、主码和列名的组合，保证能以逻辑方式访问关系数据库中的每个数据项(分量值)。

访问关系数据库中的数据项可以有许多不同的逻辑方式(可能上千种)，但必须保证有一种方式是可以访问每个数据单元的。这种方式独立于关系数据库的物理结构。这一点十分重要，因为在关系数据库中许多面向计算机的概念(如扫描下一个地址)已经去掉了。

保证访问准则表明关系系统所采用的是关联寻址(Association Addressing)的访问模式，而不是那种面向机器的寻址方法，这是关系系统独有的方式。

准则 3 空值的系统化处理。全关系型的 DBMS 应支持空值的概念，并用系统化的方式处理空值。

空值是“不知道”或“无意义”的值。在关系模型中，空值的概念很重要。例如，为了支持关系模型的完整性，要求关系的主码不能取空值。用户应该了解空值的概念和处理空值的策略。例如，当求总和、平均值等集函数时，如何处理空值将直接影响到结果。

以往处理空值的办法常常是对每个允许取空值的字段定义一种特殊的值来表示空值。这不是系统化的好办法。因为这样的话，用户必须对每个字段或域采用不同的方法来处理空值。这种方法必然会大大降低用户生产率。而且关系语言是一种非常高级的语言，要这样做常常十分困难甚至做不到。因此必须用系统化的办法来处理空值。

准则 4 基于关系模型的动态的联机数据字典。数据库的描述在逻辑级上应该和普通数据采用同样的表示方式，使得授权用户可以使用查询一般数据所用的关系语言来查询数据库的描述信息。

这一准则的一个推论是：每个用户(无论是应用程序员还是最终用户)只需学习一种数据模型。而非关系系统常常不具备这个优点。

另一个推论是授权用户可以很容易地扩充数据字典，使之变成完备的主动的关系数据字典，即使厂商有时没有提供这样的数据字典。

准则 5 统一的数据子语言准则。一个关系系统可以具有几种语言和多种

终端使用方式(如表格填空方式、命令方式等)。但必须有一种语言，它的语句可以表示为具有严格语法规定的字符串，并能全面地支持以下功能：

- 数据定义、视图定义
- 数据操作(交互式或程序式)
- 完整性约束
- 授权
- 事务处理功能(事务开始、提交、回滚)

关系方法是高度动态的，即很少需要停止数据库的活动。非关系系统则不是这样。例如，非关系系统在建立、修改数据库模式时就必须停止存取数据库的操作。因此，关系系统没有必要把上面的功能分为若干种语言(DDL，DML等)来实现。关系数据语言是一体化的统一的数据子语言，它使程序员可以首先交互地调试数据库语句，调试正确后再嵌入程序中。这样便大大提高了程序员的生产率。

准则 6 视图更新准则。所有理论上可更新的视图也应该允许由系统更新。

什么叫“一个视图是理论上可更新的视图”呢？它是指对此视图的更新要求，存在一个与时间无关的算法，该算法可以无二义性地把更新要求转换为对基本表的更新序列。

目前，有的系统只允许对基本表更新，不支持对视图的更新。有的只允许对有限的一类视图如行列子集视图更新。

视图更新准则对于系统支持数据逻辑独立性是不可缺少的。

准则 7 高级的插入、修改和删除操作。关系系统的操作对象是单一的关系。以关系为操作对象不仅简化了用户查询，提高了用户生产率，而且也为系统提供了很大的余地来进行查询优化，提高了系统的运行效率。它允许系统来选择存取路径，以便得到最有效的运行代码。

这条准则对于获得有效的分布式事务处理也是十分重要的。这样可以避免从远程结点传送一个记录就要发出一次请求。实现一次请求传送一个关系，从而节省了通信代价。

准则 8 数据物理独立性。无论数据库的数据在存储表示或存取方法上作任何变化，应用程序和终端活动都保持逻辑上的不变性。

为了保证这一独立性。DBMS 必须清楚明确地区分基本关系的逻辑和语义方面及其物理和效率方面，应用程序仅涉及逻辑方面。

准则 9 数据逻辑独立性。当对基本关系进行理论上信息不受损害的任何改变时，应用程序和终端活动都保持逻辑上的不变性。

对基本关系来讲，什么样的变化在理论上是保持信息不受损害的变化呢？

举两个例子来说明。把一个基本表按行的内容或者按列名分解为两个表，若这两个表都保留原基本表的主码，则这种变换在理论上是保持信息不受损失的。用无损连接方法(见 5.4 节)把两个表合成一个表也不会破坏信息的变换。

为了尽可能提高数据逻辑独立性，DBMS 必须能对理论上可更新的视图执行插入、修改和删除操作，即必须满足准则 6。

数据库数据物理独立性和数据逻辑独立性给数据库设计带来了很大的好处。它使得关系数据库设计易做、易改。设计人员发现错误可随时修改而不会像非关系型 DBMS 中由于设计错误而造成很大损失。在开始进行关系数据库设计时，不必对应用变化、系统效率等各方面考虑周全后才进行实现工作，而可以先实施并运行数据库，当需要时再改变数据库的逻辑结构甚至物理结构。准则 8 和准则 9 保证了应用程序和终端活动都保持逻辑上的不变性。

准则 10 数据完整性的独立性。关系数据库的完整性约束条件必须是用数据库语言定义并存储在数据字典中的，而不是在应用程序中加以定义的。

任何关系数据库必须满足两类完整性约束条件，即实体完整性和参照完整性。此外，对于某一具体的关系数据库还需要反映业务政策和管理规章的完整性约束条件。对于一个全关系型的 DBMS，这些完整性约束条件应该能用数据库语言加以定义，并能把定义存入数据字典中。它们应独立于应用程序。这样，当业务政策或管理规章变化时，只要改变存储在数据字典中的完整性定义语句而不会在逻辑上影响应用程序和终端活动。

准则 11 分布独立性。关系型 DBMS 具有分布独立性。

所谓分布独立性是指关系型 DBMS 具有这样的数据库语言，它使应用程序和终端活动在下列情况下保持逻辑不变性：

- 在第一次引入分布式数据时，即如果原来的 DBMS 只管理非分布式的数据，而现在引入了分布式数据。
- 当数据重新分布时，即如果 DBMS 能管理分布式数据，现在要改变原来的数据分布。

这两种情况概括了无论是分布的或非分布的关系型 DBMS 都能完全地支持这条准则。

一个不支持分布式数据库的全关系型的 DBMS 能够加以扩充提供以上的支持，使应用程序和终端活动无论在最初的数据还是以后的数据重新分布时都能在逻辑上保持不变。关系型 DBMS 之所以具有这个优点，其原因是：

（1）在分布数据时具有分解灵活性。

（2）在组合各个结点上子事务作业的结果时具有关系操作的重组能力。

（3）当从远程结点检索数据时可以一次请求传送一个关系，从而节省了通信代价。

（4）由于关系数据库语言是非常高级的语言，系统可以分析查询语义，执行查询优化。而对于一次处理一个记录的数据语言，系统是无法分析查询语义的。

准则 12 无破坏准则。如果一个关系系统具有一个低级(指一次一个记录)语言，则这个低级语言不能违背或绕过完整性准则。

在关系方法中，为获得数据完整性的独立性(准则 10)就要让完整性约束条件和数据的逻辑结构相独立。用准则 12 就很容易帮助识别那些贴着“关系”标签的非关系系统。因为这一系统已经在关系接口之下有一个应用接口，因此很难满足准则 12。这时 DBA 不能控制他们的数据库，不能保证数据库的完整性状态。

以上是 E.F.Codd 给出的衡量一个全关系型 DBMS 的十二条准则。这十二条准则都以准则 0 为基础。但仅有准则 0 是不够的。不支持信息准则，不保证访问准则，不支持空值准则和数据字典准则就不能保证数据库的完整性。和早期的 DBMS 相比，这四条准则使数据的管理和控制(授权和完整性控制)达到了更高的标准。准则 8～11 要求全关系型 DBMS 具有四种独立性。其中数据的物理独立性和逻辑独立性已为大家所熟知，而数据完整性的独立性和分布独立性也已为大家重视。准则 10、准则 11 已变得和准则 8、准则 9 同等重要。

4.2 关系数据库系统的查询优化

查询优化在关系数据库系统中有着非常重要的地位。关系数据库系统和非过程化的 SQL 语言能够取得巨大的成功，关键是得益于查询优化技术的发展。关系查询优化是影响 RDBMS 性能的关键因素。

优化对关系系统来说既是挑战又是机遇。所谓挑战是指关系系统为了达到用户可接受的性能必须进行查询优化。由于关系表达式的语义级别很高，使关系系统可以从关系表达式中分析查询语义，提供了执行查询优化的可能性。这就为关系系统在性能上接近甚至超过非关系系统提供了机遇。

4.2.1 关系系统及其查询优化

关系系统的查询优化既是 RDBMS 实现的关键技术又是关系系统的优点所在。它减轻了用户选择存取路径的负担。用户只要提出“干什么”，不必指出“怎么干”。对比一下非关系系统中的情况：用户使用过程化的语言表达查询要求，执行何种记录级的操作，以及操作的序列是由用户而不是由系统来决定

的。因此用户必须了解存取路径，系统要提供用户选择存取路径的手段，查询效率由用户的存取策略决定。如果用户做了愚蠢的选择，系统是无能为力对此加以改进的。这就要求用户有较高的数据库技术和程序设计水平。

查询优化的优点不仅在于用户不必考虑如何最好地表达查询以获得较好的效率，而且在于系统可以比用户程序的“优化”做得更好。这是因为：

（1）优化器可以从数据字典中获取许多统计信息，例如关系中的元组数、关系中每个属性值的分布情况等。优化器可以根据这些信息选择有效的执行计划，而用户程序则难以获得这些信息。

（2）如果数据库的物理统计信息改变了，系统可以自动对查询进行重新优化以选择相适应的执行计划。在非关系系统中必须重写程序，而重写程序在实际应用中往往是不太可能的。

（3）优化器可以考虑数百种不同的执行计划，而程序员一般只能考虑有限的几种可能性。

（4）优化器中包括了很多复杂的优化技术，这些优化技术往往只有最好的程序员才能掌握。系统的自动优化相当于使得所有人都拥有这些优化技术。

关系数据库查询优化的总目标是：选择有效的策略，求得给定关系表达式的值。

实际系统对查询优化的具体实现不尽相同，但一般来说，可以归纳为四个步骤：

① 将查询转换成某种内部表示，通常是语法树。

② 根据一定的等价变换规则把语法树转换成标准（优化）形式。

③ 选择低层的操作算法。对于语法树中的每一个操作需要根据存取路径、数据的存储分布、存储数据的聚簇等信息来选择具体的执行算法。

④ 生成查询计划。查询计划也称查询执行方案，是由一系列内部操作组成的。这些内部操作按一定的次序构成查询的一个执行方案。通常这样的执行方案有多个，需要对每个执行计划计算代价，从中选择代价最小的一个。在集中式关系数据库中，计算代价时主要考虑磁盘读写的 I/O 次数，也有一些系统还考虑了 CPU 的处理时间。

步骤③和④实际上没有清晰的界限，有些系统是作为一个步骤处理的。对于一个查询可能会有很多候选的查询计划，因此应采取适当的启发式技术来缩减查询计划的搜索空间。另外由于统计信息的不精确性，中间结果的大小难以预计等因素使得代价的精确估计常常比较困难。

目前的商品化 RDBMS 大都采用基于代价的优化算法。这种方法要求优化器充分考虑系统中的各种参数（如缓冲区大小、表的大小、数据的分布、存取路径等），通过某种代价模型计算出各种查询执行方案的执行代价，然后选取

代价最小的执行方案。在集中式数据库中，查询的执行开销主要包括：

总代价 = I/O 代价 + CPU 代价

在多用户环境下，内存在多个用户间的分配情况会明显地影响这些用户查询执行的总体性能。例如，如果系统分配给某个用户大量的内存用于其查询处理，这固然会加速该用户查询的执行，但是却可能使系统内的其他用户得不到足够的内存而影响其查询处理速度。因此，多用户数据库还应考虑查询的内存开销，即：

总代价 = I/O 代价 + CPU 代价 + 内存代价

4.2.2 一个实例

首先来看一个简单的例子，说明为什么要进行查询优化。

例 求选修了 2 号课程的学生姓名。用 SQL 语言表达：

```
SELECT  Student.Sname
FROM    Student, SC
WHERE   Student.Sno=SC.Sno AND SC.Cno='2';
```

假定学生-课程数据库中有 1000 个学生记录，10000 个选课记录，其中选修 2 号课程的选课记录为 50 个。

系统可以用多种等价的关系代数表达式来完成这一查询

$Q1=\pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge SC.Cno='2'}(Student \times SC))$

$Q2=\pi_{Sname}(\sigma_{SC.Cno='2'}(Student \bowtie SC))$

$Q3=\pi_{Sname}(Student \bowtie \sigma_{SC.Cno='2'}(SC))$

还可以写出几种等价的关系代数表达式，但分析这三种就足以说明问题了。后面将看到由于查询执行的策略不同，查询时间相差很大。

一、第一种情况

1. 计算广义笛卡尔积

把 Student 和 SC 的每个元组连接起来。一般连接的做法是：在内存中尽可能多地装入某个表(如 Student 表)的若干块元组，留出一块存放另一个表(如 SC 表)的元组。然后把 SC 中的每个元组和 Student 中每个元组连接，连接后的元组装满一块后就写到中间文件上，再从 SC 中读入一块和内存中的 Student 元组连接，直到 SC 表处理完。这时再一次读入若干块 Student 元组，读入一块 SC 元组，重复上述处理过程，直到把 Student 表处理完。

设一个块能装10个Student元组或100个SC元组，在内存中存放5块Student元组 和1块 SC 元组，则读取总块数为：

$$\frac{1\,000}{10}+\frac{1\,000}{10\times5}\times\frac{10\,000}{100}=100+20\times100=2\,100\text{ 块}$$

其中读 Student 表 100 块。读 SC 表 20 遍，每遍 100 块。若每秒读写 20 块，则总计要花 105 s。

连接后的元组数为 $10^3\times10^4=10^7$ 。设每块能装 10 个元组，则写出这些块要用 $10^6/20=5\times10^4$ s。

2. 作选择操作

依次读入连接后的元组，按照选择条件选取满足要求的记录。假定内存处理时间忽略。这一步读取中间文件花费的时间(同写中间文件一样)需 5×10^4 s。满足条件的元组假设仅 50 个，均可放在内存。

3. 作投影

把第 2 步的结果在 Sname 上作投影输出，得到最终结果。

因此第一种情况下执行查询的总时间$\approx105+2\times5\times10^4\approx10^5$ s。这里，所有内存处理时间均忽略不计。

二、第二种情况

1. 计算自然连接

为了执行自然连接，读取 Student 和 SC 表的策略不变，总的读取块数仍为 2 100 块花费 105s。但自然连接的结果比第一种情况大大减少，为 10^4 个。因此写出这些元组时间为 $10^4/10/20=50$ s，仅为第一种情况的千分之一。

2. 读取中间文件块，执行选择运算，花费时间也为 50 s。

3. 把第 2 步结果投影输出。

第二种情况总的执行时间$\approx105+50+50\approx205$ s。

三、第三种情况

1. 先对 SC 表作选择运算，只需读一遍 SC 表，存取 100 块花费时间为 5 s，因为满足条件的元组仅 50 个，不必使用中间文件。

2. 读取 Student 表，把读入的 Student 元组和内存中的 SC 元组作连接。也只需读一遍 Student 表共 100 块花费时间为 5 s。

3. 把连接结果投影输出。

第三种情况总的执行时间$\approx5+5\approx10$ s。

假如 SC 表的 Cno 字段上有索引，第 1 步就不必读取所有的 SC 元组而只需读取 Cno='2'的那些元组(50 个)。存取的索引块和 SC 中满足条件的数据块大约总共 3～4 块。若 Student 表在 Sno 上也有索引，则第 2 步也不必读取所有

的 Student 元组，因为满足条件的 SC 记录仅 50 个，涉及最多 50 个 Student 记录，因此读取 Student 表的块数也可大大减少。总的存取时间将进一步减少到数秒。

这个简单的例子充分说明了查询优化的必要性，同时也给出一些查询优化方法的初步概念。如当有选择和连接操作时，应当先做选择操作，这样参加连接的元组就可以大大减少。下面给出优化的一般策略。

4.2.3 查询优化的一般准则

下面的优化策略一般能提高查询效率，但不一定是所有策略中最优的。其实“优化”一词并不确切，也许“改进”或“改善”更恰当些。

1. 选择运算应尽可能先做。在优化策略中这是最重要、最基本的一条。它常常可使执行时间降低几个数量级，因为选择运算一般使计算的中间结果大大变小。

2. 在执行连接前对关系适当地预处理。预处理方法主要有两种，在连接属性上建立索引和对关系排序，然后执行连接。第一种称为**索引连接方法**；第二种称为**排序合并（SORT-MERGE）连接方法**。

例如 Student ⋈ SC 这样的自然连接，用索引连接方法的步骤是：

① 在 SC 上建立 Sno 的索引；

② 对 Student 中每一个元组，由 Sno 值通过 SC 的索引查找相应的 SC 元组；

③ 把这些 SC 元组和 Student 元组连接起来。

这样 Student 表和 SC 表均只要扫描一遍。处理时间只是两个关系大小的线性函数。

用排序合并连接方法的步骤是：

① 首先对 Student 表和 SC 表按连接属性 Sno 排序(如图 4.2 所示)；

② 取 Student 表中第 1 个 Sno， 依次扫描 SC 表中具有相同 Sno 的元组；

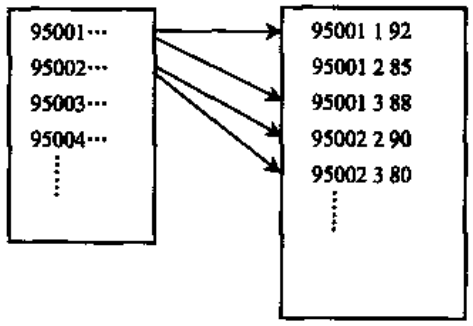


图 4.2 排序合并连接方法示意图

把它们连接起来：

③ 当扫描到 Sno 不相同的第 1 个 SC 元组时，返回 Student 表扫描它的下一个元组，再扫描 SC 表中具有相同 Sno 的元组，把它们连接起来。

重复上述步骤直到 Student 表扫描完。

这样 Student 表和 SC 表也只要扫描一遍。当然，执行时间要加上对两个表的排序时间。即使这样，使用预处理方法执行连接的时间一般仍大大减少。

3. 把投影运算和选择运算同时进行。如有若干投影和选择运算，并且它们都对同一个关系操作，则可以在扫描此关系的同时完成所有的这些运算以避免重复扫描关系。

4. 把投影同其前或其后的双目运算结合起来，没有必要为了去掉某些字段而扫描一遍关系。

5. 把某些选择同在它前面要执行的笛卡尔积结合起来成为一个连接运算，连接特别是等值连接运算要比同样关系上的笛卡尔积省很多时间(如 4.2.2 节中的实例)。

6. 找出公共子表达式。如果这种重复出现的子表达式的结果不是很大的关系，并且从外存中读入这个关系比计算该子表达式的时间少得多，则先计算一次公共子表达式并把结果写入中间文件是合算的。当查询的是视图时，定义视图的表达式就是公共子表达式的情况。

4.2.4 关系代数等价变换规则

上面的优化策略大部分都涉及到代数表达式的变换。在第二、三章中介绍了各种查询语言，这些语言都可以转换成关系代数表达式。因此关系代数表达式的优化是查询优化的基本课题。而研究关系代数表达式的优化最好从研究关系代数表达式的等价变换规则开始。所谓关系代数表达式的等价是指用相同的关系代替两个表达式中相应的关系所得到的结果是相同的。

两个关系表达式 E_1 和 E_2 是等价的，可记为 $E_1 \equiv E_2$。

常用的等价变换规则有：

1. 连接、笛卡尔积交换律

设 E_1 和 E_2 是关系代数表达式，F 是连接运算的条件，则有：

$$E_1 \times E_2 \equiv E_2 \times E_1$$

$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$$

$$E_1 \underset{F}{\bowtie} E_2 \equiv E_2 \underset{F}{\bowtie} E_1$$

2. 连接、笛卡尔积的结合律

设 E_1，E_2，E_3 是关系代数表达式，F_1 和 F_2 是连接运算的条件，则有：

$$(E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3)$$

$$(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$$

$$(E_1 \underset{F_1}{\bowtie} E_2) \underset{F_2}{\bowtie} E_3 \equiv E_1 \underset{F_1}{\bowtie} (E_2 \underset{F_2}{\bowtie} E_3)$$

3. 投影的串接定律

$$\pi_{A_1,A_2,\cdots,A_n}(\pi_{B_1,B_2,\cdots,B_m}(E)) \equiv \pi_{A_1,A_2,\cdots,A_n}(E)$$

这里，E 是关系代数表达式，$A_i(i=1, 2, \cdots, n)$，$B_j(j=1, 2, \cdots, m)$是属性名且$\{A_1, A_2, \cdots, A_n\}$构成$\{B_1, B_2, \cdots, B_m\}$的子集。

4. 选择的串接定律

$$\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_1 \wedge F_2}(E)$$

这里，E 是关系代数表达式，F_1，F_2 是选择条件。选择的串接律说明选择条件可以合并。这样一次就可检查全部条件。

5. 选择与投影的交换律

$$\sigma_F(\pi_{A_1,A_2,\cdots,A_n}(E)) \equiv \pi_{A_1,A_2,\cdots,A_n}(\sigma_F(E))$$

这里，选择条件 F 只涉及属性 A_1，…，A_n。若 F 中有不属于 A_1，…，A_n 的属性 B_1，…，B_m 则有更一般的规则：

$$\pi_{A_1,A_2,\cdots,A_n}(\sigma_F(E)) \equiv \pi_{A_1,A_2,\cdots,A_n}(\sigma_F(\pi_{A_1,A_2,\cdots,A_n,B_1,B_2,\cdots,B_m}(E)))$$

6. 选择与笛卡尔积的交换律

如果 F 中涉及的属性都是 E_1 中的属性，则

$$\sigma_F(E_1 \times E_2) \equiv \sigma_F(E_1) \times E_2$$

如果 $F=F_1 \wedge F_2$，并且 F_1 只涉及 E_1 中的属性，F_2 只涉及 E_2 中的属性，则由上面的等价变换规则 1，4，6 可推出：

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F_1}(E_1) \times \sigma_{F_2}(E_2)$$

若 F_1 只涉及 E_1 中的属性，F_2 涉及 E_1 和 E_2 两者的属性，则仍有

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F_2}(\sigma_{F_1}(E_1) \times E_2)$$

它使部分选择在笛卡尔积前先做。

7. 选择与并的交换

设 $E=E_1 \cup E_2$，E_1，E_2 有相同的属性名，则

$$\sigma_F(E_1 \cup E_2) \equiv \sigma_F(E_1) \cup \sigma_F(E_2)$$

8. 选择与差运算的交换

若 E_1 与 E_2 有相同的属性名，则

$$\sigma_F(E_1 - E_2) \equiv \sigma_F(E_1) - \sigma_F(E_2)$$

9. 投影与笛卡尔积的交换

设 E_1 和 E_2 是两个关系表达式，A_1，…，A_n 是 E_1 的属性，B_1，…，B_m 是 E_2 的属性，则

$$\pi_{A1,A2,\cdots,An,B1,B2,\cdots,Bm}(E_1\times E_2)\equiv \pi_{A1,A2,\cdots,An}(E_1)\times \pi_{B1,B2,\cdots,Bm}(E_2)$$

10. 投影与并的交换

设 E_1 和 E_2 有相同的属性名，则

$$\pi_{A1,A2,\cdots,An}(E_1\cup E_2)\equiv \pi_{A1,A2,\cdots,An}(E_1)\cup \pi_{A1,A2,\cdots,An}(E_2)$$

4.2.5 关系代数表达式的优化算法

应用上面的变换法则来优化关系表达式，使优化后的表达式能遵循 4.2.3 节中的一般原则。例如把选择和投影尽可能地早做(即把它们移到表达式语法树的下部)。下面给出关系表达式的优化算法。

算法：关系表达式的优化。

输入：一个关系表达式的语法树。

输出：计算该表达式的程序。

方法：

（1）利用规则 4 把形如 $\sigma_{F1\wedge F2\wedge\cdots\wedge Fn}(E)$ 变换为

$$\sigma_{F1}(\sigma_{F2}(\cdots(\sigma_{Fn}(E))\cdots))$$

（2）对每一个选择，利用规则 4～8 尽可能把它移到树的叶端。

（3）对每一个投影利用规则 3，9，10，5 中的一般形式尽可能把它移向树的叶端。

注意：法则 3 使一些投影消失，而一般形式的规则 5 把一个投影分裂为两个，其中一个有可能被移向树的叶端。

（4）利用规则 3～5 把选择和投影的串接合并成单个选择、单个投影或一个选择后跟一个投影。使多个选择或投影能同时执行，或在一次扫描中全部完成，尽管这种变换似乎违背“投影尽可能早做”的原则，但这样做效率更高。

（5）把上述得到的语法树的内节点分组。每一双目运算(×，⋈，∪，-)和它所有的直接祖先为一组(这些直接祖先是σ，π运算)。如果其后代直到叶子全是单目运算，则也将它们并入该组，但当双目运算是笛卡尔积(×)，而且其后的选择不能与它结合为等值连接时除外。把这些单目运算单独分为一组。

（6）生成一个程序，每组结点的计算是程序中的一步。各步的顺序是任意的，只要保证任何一组的计算不会在它的后代组之前计算。

4.2.6 优化的一般步骤

各个关系系统的优化方法不尽相同，大致的步骤可以归纳如下:

（1）把查询转换成某种内部表示

通常用的内部表示是语法树，例如 4.2.2 节中的实例可表示为图 4.3。

为了使用关系代数表达式的优化法，不仿假设内部表示是关系代数语法树，则上面的语法树变成图 4.4。

（2）把语法树转换成标准(优化)形式

利用优化算法，把原始的语法树转换成优化的形式。

各个 DBMS 优化算法不尽相同，这里利用前面讨论的关系代数表达式的优化算法进行优化。

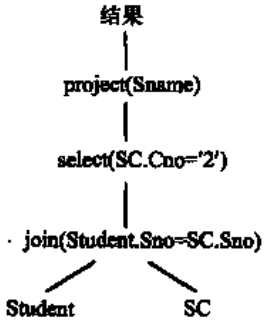


图 4.3 语法树

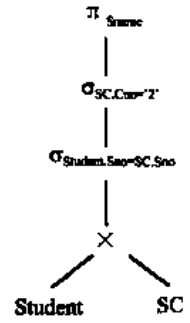


图 4.4 关系代数语法树

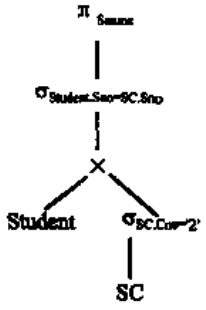


图 4.5 优化后的语法树

利用规则 4，6 把选择$\sigma_{SC.Cno='2'}$移到叶端，图 4.4 语法树便转换成图 4.5。这就是 4.2.2 节中 $Q3$ 的语法树表示。上面已经分析了 $Q3$ 比 $Q1$，$Q2$ 查询效率要高得多。

（3）选择低层的存取路径

根据第（2）步得到的优化了的语法树计算关系表达式值的时候要充分考虑索引、数据的存储分布等存取路径，利用它们进一步改善查询效率。这就要求优化器去查找数据字典，获得当前数据库状态的信息。例如选择字段上是否有索引，连接的两个表是否有序，连接字段上是否有索引等，然后根据一定的优化规则选择存取路径。如本例中若 SC 表上建有 Cno 的索引，则应该利用这个索引，而不必顺序扫描 SC 表。

（4）生成查询计划，选择代价最小的

查询计划是由一组内部过程组成的，这组内部过程实现按某条存取路径计算关系表达式的值。常有多个查询计划可供选择。例如在作连接运算时，若两个表(设为 R1，R2)均无序，连接属性上也没有索引，则可以有下面几种查询

计划：

- 对两个表作排序预处理
- 对 R1 在连接属性上建索引
- 对 R2 在连接属性上建索引
- 在 R1，R2 的连接属性上均建索引

对不同的查询计划计算代价，选择代价最小的一个。在计算代价时主要考虑磁盘读写的 I/O 数，内存 CPU 处理时间在粗略计算时可不考虑。

注意：对某一查询可以有很多不同的查询计划。不可能生成所有的查询计划。因为对这些查询计划进行代价估计本身要花费一定的代价，弄不好就会得不偿失。生成查询计划的方法和技术这里不细述，有兴趣的读者可阅读 SYSTEM R 和 INGRES 优化技术的有关文献。

4.3 小 结

支持关系模型的数据库系统是关系系统，由于支持关系模型的程度不同，实际的数据库系统可以分为表式系统、(最小)关系系统、关系完备的和全关系的四类系统。

完全地支持关系模型的系统应遵循十二条基本准则，这是关系模型的奠基人 E.F.Codd 提出的。这十二条准则也可以作为评价或购买关系型产品的标准。

查询处理是数据库管理系统的核心，而查询优化技术又是查询处理的关键技术。查询优化一般可分为代数优化和物理优化。代数优化是指关系代数表达式的优化；物理优化则是指存取路径和低层操作算法的选择。

习 题

1. 试给出各类关系系统的定义：最小关系的系统；关系上完备的系统；全关系型的关系系统。

*2. 试述全关系型系统应满足的十二条准则，以及十二条基本准则的实际意义和理论意义。

3. 试述查询优化在关系数据库系统中的重要性和可能性。

4. 对学生-课程数据库有如下的查询:

```
SELECT  Cname
FROM  Student, Course, SC
```

```
WHERE   Student.Sno=SC.Sno   AND
        SC.Cno=Course.Cno   AND
        Student.Sdept ='IS';
```

此查询要求信息系学生选修了的所有课程名称。

试画出用关系代数表示的语法树，并用关系代数表达式优化算法对原始的语法树进行优化处理，画出优化后的标准语法树。

5. 试述查询优化的一般准则。

6. 试述查询优化的一般步骤。

本章参考文献

1 Codd E F. Relational Database: A Practical Foundation for Productivity. CACM 25, No.2, February 1982

（本文阐述了关系数据库系统能极大地提高用户生产率这一重要观点。）

2 Schmidt Joachim W, Brodie Michael L(eds.). Relational Database Systems: Analysis and Comparison. New York: Springer-Verlag, 1983

3 Frank W King III. Relational Data Base Systems, Where We Stand Today. IFIP, 1980

4 Codd F. Is Your DBMS Really Relational. Computer World, No.10, 1985

5 Special Issue on Query Optimization. IEEE Database Engineering 5. No.3.September 1982

（[5]是关于查询优化的专辑。）

6 Smith J M, Chang P Y T. Optimizing the Performance of a Relational Algebra Database Interface. CACM 18, No.10, October 1975

（本文研究了关系代数的优化，给出了详细的算法。）

7 Wong E, Youssefi K. Decomposition: A Strategy for Query Processing. ACM TODS 1, NO.3, September 1976

（[7]和[8]介绍了 INGRES 采用的查询分解优化方法。）

8 Youssefi X, Wong E. Query Processing in a Relational Database Management System. Proc.5th International Conference on Very Large Data Bases. September 1979

9 Youssefi K, Wong E. Query Processing in a Relational Database Management System. in Proceedings of VLDB, 1979

（本文讨论了 INGRES 查询分解优化算法的性能。）

10 Aho A V, Sagiv Y, Ullman J D. Efficient Optimization of a Class of Relational Expressions. ACM TODS 4, No.4, December 1979

（本文讨论了关系表达式的优化方法。）

11 Astrahan M et al. System R: A Relational Approach to Data Base Management. ACM TODS, 1:2, June 1976

（本文介绍了 SYSTEM R 的基于代价估算的查询优化算法。）

12 Selinger P，et al. Access Path Selection in a Relational Database Management System. in Proceedings of ACM SIGMOD，1979

（本文讨论了 SYSTEM R 中多表连接操作的查询优化问题。）

13 Astrahan M M，Schkolnick M. Performance of the System R Access Path Selection Mechanism. IFIP，1980

（本文介绍了 SYSTEM R 的存取路径选择机制及其性能。）

14 Kim W. On Optimizing An SQL-like Nested Query. TODS，3:3，September 1982

（本文提出了 SQL 嵌套查询的优化方法。）

15 Whang K. Query Optimization in Office By Example. IBM Research Report RC 11571，December 1985

（本文讨论了 OBE 语言的查询优化问题，OBE 是一个基于 QBE 的数据库系统语言。）

16 DeWitt D et al. Implementation Techniques for Main Memory Databases. in Proceedings of ACM SIGMOD，1984

（本文研究了主存数据库上的查询优化问题。）

17 Yao S B. Optimization of Query Evaluation Algorithms. ACM TODS，4:2，June 1979

（本文对多个查询优化算法进行了比较和分析。）

18 Jarke M， Koch J. Query Optimization in Database Systems. ACM Computing Surveys，16:2，June 1984

（本文系统地综述了查询优化的主要研究成果，并列出了这一研究领域的主要文献。）

19 Kim W， Reiner D，Batory D. Query Processing in Database Systems. Springer-Verlag，1985

（本文讨论了查询优化研究领域中的新问题。）

20 King J. QUIST：A System for Semantic Query Optimization in Relational Databases. In Proceedings of VLDB，1981

（本文讨论了在数据库中利用语义知识进行查询优化的问题。）

21 Malley C，Zdonick S. A Knowledge-Based Approach to Query Optimization. in Proceedings of the 1st Inter. Conference on Expert Database Systems. 1986

（本文也是讨论在数据库中利用语义知识进行查询优化的问题。）

22 Beck H，Gala S，Navathe S. Classification as a Query Processing Technique in the CANDIDE Semantic Data Model. in Proceedings of IEEE Inter. Conference on Data Engineering. 1989

（本文讨论了一种实现查询优化的分类技术，这种技术适用于基于语义数据模型的数据库系统。）

第五章　关系数据理论

这一章讨论关系数据理论。5.1 节从数据库逻辑设计中如何构造一个好的数据库模式这一问题出发，阐明了关系规范化理论研究的实际背景。5.2 节介绍规范化理论，讨论各种范式及可能存在的插入、删除等毛病，并直观地描述解决办法。5.3 节和 5.4 节进一步讨论关系数据理论。5.3 节讨论函数依赖的推理规则。5.4 节给出了模式等价的不同定义及模式分解算法。5.1 和 5.2 两节内容是基本的，作为本科学生的要求，5.3 和 5.4 两节则可作为研究生的学习内容。

5.1　问题的提出

前面已经讨论了数据库系统的一般概念，介绍了关系数据库的基本概念、关系模型的三个部分以及关系数据库的标准语言。但是还有一个很基本的问题尚未涉及，针对一个具体问题，应该如何构造一个适合于它的数据模式，即应该构造几个关系模式，每个关系由哪些属性组成等。这是数据库设计的问题，确切地讲是关系数据库逻辑设计问题。

实际上设计任何一种数据库应用系统，不论是层次的、网状的还是关系的，都会遇到如何构造合适的数据模式即逻辑结构的问题。由于关系模型有严格的数学理论基础，并且可以向别的数据模型转换，因此，人们就以关系模型为背景来讨论这个问题，形成了数据库逻辑设计的一个有力工具——关系数据库的规范化理论。规范化理论虽然是以关系模型为背景，但是它对于一般的数据库逻辑设计同样具有理论上的意义。

下面首先回顾一下关系模型的形式化定义。

在 2.2.2 节中已经讲过，一个关系通常是由赋予它的元组语义来确定的。元组语义实质上是一个 n 目谓词（n 是属性集中属性的个数）。凡使该 n 目谓词为真的笛卡尔积中的元素（或者说凡符合元组语义的那部分元素）的全体就构成了该关系模式的关系。

现实世界随着时间在不断地变化，因而在不同的时刻，关系模式的关系也会有所变化。但是，现实世界的许多已有事实限定了关系模式所有可能的关系必须满足一定的完整性约束条件。这些约束或者通过对属性取值范围的限定，

或者通过属性值间的相互关连（主要体现于值的相等与否）反映出来。后者称为数据依赖，它是数据模式设计的关键。关系模式应当刻划这些完整性约束条件，于是一个关系模式应当是一个五元组。

R（U，D，dom，F）

这里：

（1）关系名 R，它是符号化的元组语义；

（2）一组属性 U；

（3）属性组 U 中属性所来自的域 D；

（4）属性到域的映射 dom；

（5）属性组 U 上的一组数据依赖 F。

由于（3），（4）对模式设计关系不大，因此在本章中把关系模式看作是一个三元组：

R（U，F）

当且仅当 U 上的一个关系 r 满足 F 时，r 称为关系模式 R（U，F）的一个关系。

关系，作为一张二维表，对它有一个最起码的要求：每一个分量必须是不可分的数据项。满足了这个条件的关系模式就属于**第一范式**（1NF）。

在模式设计中，假设已知一个模式 Sϕ，它仅由单个关系模式组成，问题是要设计一个模式 SD，它与 Sϕ“等价”，但在某些指定的方面“更好”一些。这里通过一个例子来说明一个“不好”的模式会有些什么毛病，分析它们产生的原因，从中找出设计一个“好”的关系模式的办法。

在举例之前，先非形式地讨论一下数据依赖的概念。

数据依赖是通过一个关系中属性间值的相等与否体现出来的数据间的相互关系。它是现实世界属性间相互联系的抽象，是数据内在的性质，是语义的体现。现在人们已经提出了许多种类型的数据依赖，其中最重要的是**函数依赖**（Functional Dependency，简称 FD）和**多值依赖**（Multivalued Dependency，简称 MVD）。

函数依赖极为普遍地存在于现实生活中。比如描述一个学生的关系，可以有学号（SNO），姓名（SNAME），系名（SDEPT）等几个属性。由于一个学号只对应一个学生，一个学生只在一个系学习。因而当“学号”值确定之后，姓名和该生所在系的值也就被唯一地确定了。就象自变量 x 确定之后，相应的函数值 $f(x)$ 也就唯一地确定了一样，称 SNO 函数决定 SNAME 和 SDEPT，或者说 SNAME，SDEPT 函数依赖于 SNO，记为：SNO→SNAME，SNO→SDEPT。

现在要建立一个数据库来描述学生的一些情况，面临的对象有：学生（用

学号 SNO 描述），系（用系名 SDEPT 描述），系负责人（用其姓名 MN 描述），课程（用课程名 CNAME 描述）和成绩（G）。于是得到一组属性。

$U=\{$ SNO，SDEPT，MN，CNAME，G $\}$

由现实世界的已知事实得知：

① 一个系有若干学生，但一个学生只属于一个系；

② 一个系只有一名（正职）负责人；

③ 一个学生可以选修多门课程，每门课程有若干学生选修；

④ 每个学生学习每一门课程有一个成绩；

于是得到属性组 U 上的一组函数依赖：

$F=\{$ SNO→SDEPT，SDEPT→MN，（SNO，CNAME）→G $\}$

这组函数依赖如图 5.1 所示。

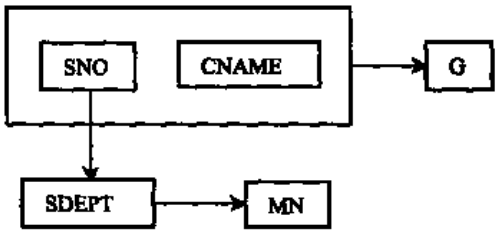


图 5.1

如果只考虑函数依赖这一种数据依赖，就得到了一个描述学校的数据库模式 S（U，F，它由一个单一的关系模式构成。这个模式有下述三个“毛病”：

（1）如果一个系刚成立尚无学生，或者虽然有了学生但尚未安排课程。那么就无法把这个系及其负责人的信息存入数据库。称为**插入异常**。

（2）反过来，如果某个系的学生全部毕业了，在删除该系学生选修课程的同时，把这个系及其负责人的信息也丢掉了，称为**删除异常**。

（3）冗余太大。比如，每一个系负责人的姓名要与该系每一个学生的每一门功课的成绩出现的次数一样多。这样，一方面浪费存储，另一方面系统要付出很大的代价来维护数据库的完整性。比如某系负责人更换后，就必须逐一修改有关的每一个元组。

由于上述三个“毛病”，它是一个“不好”的数据库模式。一个“好”的模式应当不会发生插入异常和删除异常、冗余应尽可能少。

为什么会发生插入异常和删除异常呢?

这是因为这个模式中的函数依赖存在某些不好的性质。假如把这个单一的模式改造一下，分成三个关系模式：

S（SNO，SDEPT，SNO→SDEPT）；

SG（SNO，CNAME，G，（SNO，CNAME）→G）；

DEPT（SDEPT，MN，SDEPT→MN）；

这三个模式都不会发生插入异常、删除异常的毛病，数据的冗余也得到了控制。

一个模式的函数依赖会有哪些不好的性质，如何改造一个不好的模式，这就是下一节规范化理论讨论的内容。

5.2 规 范 化

为了使数据库设计的方法走向完备，人们研究了规范化理论。从 1971 年起 E.F.Codd 就提出了这一理论，规范化理论的研究已经取得了很多成果。

前面已经提过关系必须是规范化的（Normalization），即每一个分量必须是不可分的数据项。但是这只是最基本的规范化。上节的例子说明并非所有这样规范化的关系都能很好地描述现实世界，必须作进一步的分析，以确定如何设计一个好的、反映现实世界的模式。

本节将讨论下述内容：首先讨论一个关系属性间不同的依赖情况，讨论如何根据属性间依赖情况来判定关系是否具有某些不合适的性质。通常按属性间依赖情况来区分关系规范化的程度为第一范式、第二范式、第三范式和第四范式等。然后直观地描述如何将具有不合适性质的关系转换为更合适的形式。

5.2.1 函数依赖

定义 5.1 设 $R(U)$ 是属性集 U 上的关系模式。X，Y 是 U 的子集。若对于 $R(U)$ 的任意一个可能的关系 r，r 中不可能存在两个元组在 X 上的属性值相等，而在 Y 上的属性值不等，则称 **X 函数确定 Y 或 Y 函数依赖于 X**，记作 $X \rightarrow Y$。

函数依赖和别的数据依赖一样是语义范畴的概念。只能根据语义来确定一个函数依赖。例如 姓名→年龄 这个函数依赖只有在该部门没有同名人的条件下成立。如果允许有同名人，则年龄就不再函数依赖于姓名了。设计者也可以对现实世界作强制的规定。例如规定不允许同名人出现，因而使 姓名→年龄 函数依赖成立。这样当插入某个元组时这个元组上的属性值必须满足规定的函数依赖，若发现有同名人存在，则拒绝插入该元组。

注意，函数依赖不是指关系模式 R 的某个或某些关系满足的约束条件，而是指 R 的一切关系均要满足的约束条件。

下面介绍一些术语和记号：

- $X \rightarrow Y$，但 $Y \not\subseteq X$ 则称 $X \rightarrow Y$ **是非平凡的函数依赖**。若不特别声明，总是

讨论非平凡的函数依赖。

- $X \rightarrow Y$，但 $Y \subseteq X$ 则称 $X \rightarrow Y$ 是**平凡的函数依赖**。
- 若 $X \rightarrow Y$，则 X 叫做**决定因素**（Determinant）。
- 若 $X \rightarrow Y$，$Y \rightarrow X$，则记作 $X \longleftrightarrow Y$。
- 若 Y 不函数依赖于 X，则记作 $X \nrightarrow Y$。

定义 5.2 在 $R(U)$ 中，如果 $X \rightarrow Y$，并且对于 X 的任何一个真子集 X'，都有 $X' \nrightarrow Y$，则称 Y 对 X **完全函数依赖**，记作：

$$X \xrightarrow{F} Y$$

若 $X \rightarrow Y$，但 Y 不完全函数依赖于 X，则称 Y 对 X **部分函数依赖**，记作：

$$X \xrightarrow{P} Y$$

举一个简单的例子。在关系 S（SNO，SNAME，SDEPT，SAGE）中，SNO →SDEPT，SNO→SAGE，SNO←→SNAME（若无重名）。

在关系 SC（SNO，CNO，G）中，SNO ↛CNO，SNO ↛G。
在这里单个的属性不能作为决定因素，但属性的组合可以作为决定因素，即：

（SNO，CNO）$\xrightarrow{F}$G，（SNO，CNO）就是决定因素。

定义 5.3 在 $R(U)$ 中，如果 $X \rightarrow Y$，（$Y \nsubseteq X$），$Y \nrightarrow X$，$Y \rightarrow Z$，则称 Z 对 X **传递函数依赖**。

加上条件 $Y \nrightarrow X$，是因为如果 $Y \rightarrow X$，则 $X \longleftrightarrow Y$，实际上是 $X \underset{\text{直接}}{\rightarrow} Z$，是**直接函数依赖**而不是传递函数依赖。

5.2.2 码

在第二章中已给出了有关码的若干定义。这里用函数依赖的概念来定义码。

定义 5.4 设 K 为 $R(U, F)$ 中的属性或属性组合，若 $K \xrightarrow{F} U$ 则 K 为 R 的**候选码**（Candidate key）。若候选码多于一个，则选定其中的一个为**主码**（Primary key）。

包含在任何一个候选码中的属性，叫做**主属性**（Prime attribute）。不包含在任何码中的属性称为**非主属性**（Nonprime attribute）或**非码属性**（Non-key attribute）。最简单的情况，单个属性是码。最极端的情况，整个属性组是码，称为**全码**（All-key）。如在关系模式 S（SNO，SDEPT，SAGE）中 SNO 是码，而在关系模式 SC（SNO，CNO，G）中属性组合（SNO，CNO）是码。下面举一个全码的例子。

关系模式 $R(P, W, A)$，属性 P 表示演奏者，W 表示作品，A 表示听众。假设一个演奏者可以演奏多个作品，某一作品可被多个演奏者演奏。听众也可以欣赏不同演奏者的不同作品，这个关系模式的码为（P，W，A），即 All-key。

定义 5.5 关系模式 R 中属性或属性组 X 并非 R 的码，但 X 是另一个关系模式的码，则称 X 是 R 的**外部码**（Foreign key），也称**外码**。

如在 SC（SNO，CNO，G）中，SNO 不是码，但 SNO 是关系模式 S（SNO，SDEPT，SAGE）的码，则 SNO 是关系模式 SC 的外部码。

主码与外部码提供了一个表示关系间联系的手段。如关系模式 S 与 SC 的联系就是通过 SNO 来体现的。

5.2.3 范式

关系数据库中的关系是要满足一定要求的，满足不同程度要求的为不同范式。满足最低要求的叫第一范式，简称 1NF。在第一范式中满足进一步要求的为第二范式，其余以此类推。

从范式来讲，主要是 E.F.Codd 做的工作，1971 年~1972 年他系统地提出了 1NF，2NF，3NF 的概念，讨论了规范化的问题。1974 年，Codd 和 Boyce 又共同提出了一个新范式，即 BCNF。1976 年 Fagin 又提出了 4NF。后来又有人提出了 5NF。

本来，所谓“第几范式”，是表示关系的某一种级别。所以经常称某一关系模式 R 为第几范式。现在把范式这个概念理解成符合某一种级别的关系模式的集合，则 R 为第几范式就可以写成 $R \in x$NF。

对于各种范式之间的联系有

$$5NF \subset 4NF \subset BCNF \subset 3NF \subset 2NF \subset 1NF$$

成立，如图 5.2 所示。

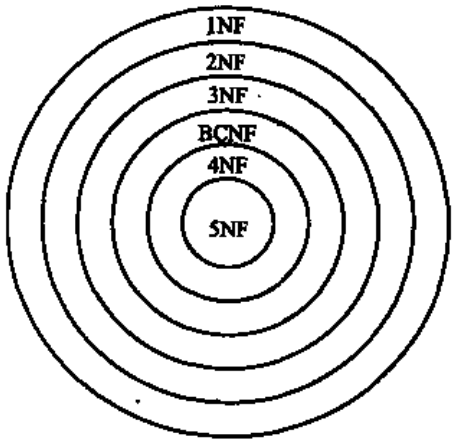


图 5.2 各种范式之间的关系

一个低一级范式的关系模式，通过模式分解可以转换为若干个高一级范式的关系模式的集合，这种过程就叫规范化。

5.2.4 2NF

定义 5.6 若 $R \in 1NF$，且每一个非主属性完全函数依赖于码，则 $R \in 2NF$。

下面举一个不是 2NF 的例子。

关系模式 S-L-C（SNO，SDEPT，SLOC，CNO，G）

某中 SLOC 为学生的住处，并且每个系的学生住在同一个地方。

这里码为（SNO，CNO）。函数依赖有：

$(SNO, CNO) \xrightarrow{F} G$

$SNO \rightarrow SDEPT, (SNO, CNO) \xrightarrow{P} SDEPT$

$SNO \rightarrow SLOC, (SNO, CNO) \xrightarrow{P} SLOC,$

SDEPT→SLOC（因为每个系的学生只住一个地方），如图 5.3 所示。

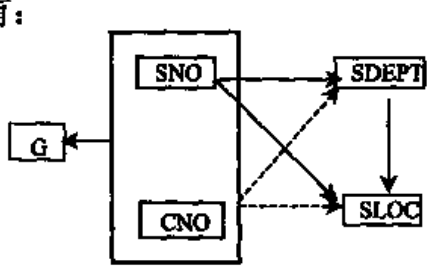


图 5.3 函数依赖示例

图中用虚线表示部分函数依赖。另外，SDEPT 还函数决定 SLOC，这一点在讨论第二范式时暂不考虑。可以看到非主属性 SDEPT，SLOC 并不完全函数依赖于码。因此 S-L-C（SNO，SDEPT，SLOC，CNO，G）不符合 2NF 定义，即 S-L-C ∉ 2NF。

一个关系模式 R 不属于 2NF，就会产生以下几个问题：

1. 插入异常。假若要插入一个学生 SNO=S7，SDEPT=PHY，SLOC=BLD2，但该生还未选课，即这个学生无 CNO，这样的元组就插不进 S-L-C 中。因为插入元组时必须给定码值，而这时码值的一部分为空，因而学生的固有信息无法插入。

2. 删除异常。假定某个学生只选一门课，如 S4 就选了一门课 C3。现在 C3 这门课他也不选了，那么 C3 这个数据项就要删除。而 C3 是主属性，删除了 C3，整个元组就必须跟着删除，使得 S4 的其他信息也被删除了，从而造成删除异常，即不应删除的信息也删除了。

3. 修改复杂。某个学生从数学系（MA）转到计算机科学系（CS），这本来只需修改此学生元组中的 SDEPT 分量。但因为关系模式 S-L-C 中还含有系的住处 SLOC 属性，学生转系将同时改变住处，因而还必须修改元组中的 SLOC 分量。另外，如果这个学生选修了 k 门课，SDEPT，SLOC 重复存储了 k 次，不仅存储冗余度大，而且必须无遗漏地修改 k 个元组中全部 SDEPT，SLOC 信息，造成修改的复杂化。

分析上面的例子，可以发现问题在于有两种非主属性。一种如 G，它对码是完全函数依赖。另一种如 SDEPT，SLOC，对码不是完全函数依赖。解决的办法是用投影分解把关系模式 S-L-C 分解为两个关系模式。

SC（SNO，CNO，G）

S-L（SNO，SDEPT，SLOC）

关系模式 SC 与 S-L 中属性间的函数依赖可以用图 5.4、图 5.5 表示如下：

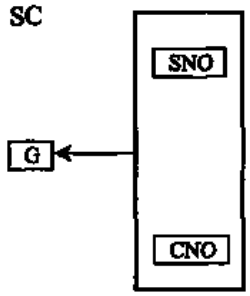


图 5.4 SC 中的函数依赖

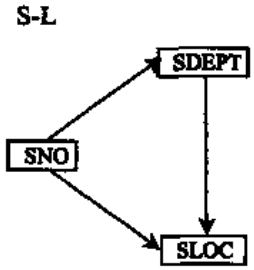


图 5.5 S-L 中的函数依赖

关系模式 SC 的码为（SNO，CNO），关系模式 S-L 的码为 SNO，这样就使得非主属性对码都是完全函数依赖了。

5.2.5 3NF

定义 5.7 关系模式 R（U，F）中若不存在这样的码 X，属性组 Y 及非主属性 Z（$Z \subsetneq Y$）使得 $X \to Y$，（$Y \not\to X$）$Y \to Z$ 成立，则称 R（U，F）$\in$ 3NF。

由**定义 5.7** 可以证明，若 $R \in$ 3NF，则每一个非主属性既不部分依赖于码也不传递依赖于码。

在图 5.4 中关系模式 SC 没有传递依赖，而图 5.5 中关系模式 S-L 存在非主属性对码传递依赖。在 S-L 中，由 SNO→SDEPT，（SDEPT↛SNO），SDEPT→SLOC，可得 SNO $\overset{\text{传递}}{\to}$ SLOC。因此 SC$\in$3NF，而 S-L $\notin$ 3NF。

一个关系模式 R 若不是 3NF，就会产生与 5.2.4 节中 2NF 相类似的问题。读者可以类比 2NF 的反例加以说明。

解决的办法同样是将 S-L 分解为：

S-D（SNO，SDEPT）

D-L（SDEPT，SLOC）

分解后的关系模式 S-D 与 D-L 中不再存在传递依赖。

5.2.6 BCNF

BCNF（Boyce Codd Normal Form）是由 Boyce 与 Codd 提出的，比上述的 3NF 又进了一步，通常认为 BCNF 是修正的第三范式，有时也称为扩充的确第三范式。

定义 5.8 关系模式 R（U，F）$\in$1NF。若 $X \to Y$ 且 $Y \subsetneq X$ 时 X 必含有码，则 R（U，F）$\in$ BCNF。

也就是说，关系模式 R（U，F）中，若每一个决定因素都包含码，则 R（U，F）$\in$BCNF。

由 BCNF 的定义可以得到结论，一个满足 BCNF 的关系模式有：

- 所有非主属性对每一个码都是完全函数依赖。
- 所有的主属性对每一个不包含它的码，也是完全函数依赖。
- 没有任何属性完全函数依赖于非码的任何一组属性。

由于 R∈BCNF，按定义排除了任何属性对码的传递依赖与部分依赖，所以 R∈3NF。严格的证明留给读者完成。但是若 R∈3NF，则 R 未必属于 BCNF。

下面用几个例子说明属于 3NF 的关系模式有的属于 BCNF，但有的不属于 BCNF。

先考察关系模式 C（CNO，CNAME，PCNO），它只有一个码 CNO，这里没有任何属性对 CNO 部分依赖或传递依赖，所以 C∈3NF。同时 C 中 CNO 是唯一的决定因素，所以 C∈BCNF。对于关系模式 SC（SNO，CNO，G）可作同样分析。

对于关系模式 S（SNO，SNAME，SDEPT，SAGE），假定 SNAME 也具有唯一性，那么 S 就有两个码，这两个码都由单个属性组成，彼此不相交。其他属性不存在对码的传递依赖与部分依赖，所以 S∈3NF。同时 S 中除 SNO，SNAME 外没有其他决定因素，所以 S 也属于 BCNF。

以下再举几个例子

例 1 关系模式 SJP（S，J，P）中，S 是学生，J 表示课程，P 表示名次。每一个学生选修每门课程的成绩有一定的名次，每门课程中每一名次只有一个学生（即没有并列名次）。由语义可得到下面的函数依赖：

（S，J）→P；（J，P）→S

所以（S，J）与（J，P）都可以作为候选码。这两个码各由两个属性组成，而且它们是相交的。这个关系模式中显然没有属性对码传递依赖或部分依赖。所以 SJP∈3NF，而且除（S，J）与（J，P）以外没有其他决定因素，所以 SJP∈BCNF。

例 2 关系模式 STJ（S，T，J）中，S 表示学生，T 表示教师，J 表示课程。每一教师只教一门课。每门课有若干教师，某一学生选定某门课，就对应一个固定的教师。由语义可得到如下的函数依赖。

（S，J）→T；（S，T）→J；T→J，用图 5.6 表示如下。

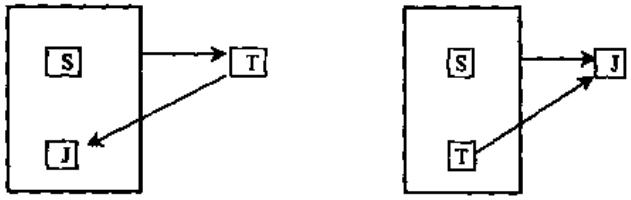


图 5.6 STJ 中的函数依赖

这里（S，J），（S，T）都是候选码。

STJ 是 3NF，因为没有任何非主属性对码传递依赖或部分依赖。但 STJ 不是 BCNF 关系，因为 T 是决定因素，而 T 不包含码。

对于不是 BCNF 的关系模式，仍然存在不合适的地方。读者可自己举例指出 STJ 的不合适之处。非 BCNF 的关系模式也可以通过分解成为 BCNF。例如 STJ 可分解为 ST（S，T）与 TJ（T，J），它们都是 BCNF。

3NF 和 BCNF 是在函数依赖的条件下对模式分解所能达到的分离程度的测度。一个模式中的关系模式如果都属于 BCNF，那么在函数依赖范畴内，它已实现了彻底的分离，已消除了插入和删除的异常。3NF 的“不彻底”性表现在可能存在主属性对码的部分依赖和传递依赖。

5.2.7 多值依赖

以上完全是在函数依赖的范畴内讨论问题。属于 BCNF 的关系模式是否就很完美了呢？下面来看一个例子。

例 1 学校中某一门课程由多个教员讲授，他们使用相同的一套参考书。每个教员可以讲授多门课程，每种参考书可以供多门课程使用。可以用一个非规范化的关系来表示教员 T，课程 C 和参考书 B 之间的关系（如表 5.1 所示）。

表 5.1

课 程 C	教 员 T	参 考 书 B
物理	李 勇 王 军	普通物理学 光学原理 物理习题集
数学	李 勇 张 平	数学分析 微分方程 高等代数
计算数学	张 平 周 峰	数学分析
⋮	⋮	⋮

把这张表变成一张规范化的二维表，如表 5.2 所示。

表 5.2 Teaching

课程 C	教员 T	参考书 B
物 理	李 勇	普通物理学
物 理	李 勇	光学原理
物 理	李 勇	物理习题集
物 理	王 军	普通物理学
物 理	王 军	光学原理
物 理	王 军	物理习题集
数 学	李 勇	数学分析
数 学	李 勇	微分方程
数 学	李 勇	高等代数
数 学	张 平	数学分析
数 学	张 平	微分方程
数 学	张 平	高等代数
⋮	⋮	⋮

关系模型 TEACHING（C，T，B）的码是（C，T，B），即 All-Key。因而 TEACHING∈BCNF。但是当某一课程（如物理）增加一名讲课教员（如周英）时，必须插入多个（这里是三个）元组：（物理，周英，普通物理学）；（物理，周英，光学原理）；（物理，周英，物理习题集）。

同样，某一门课（如数学）要去掉一本参考书（如微分方程），则必须删除多个（这里是两个）元组：（数学，李勇，微分方程）；（数学，张平，微分方程）。

对数据的增删改很不方便，数据的冗余也十分明显。仔细考察这类关系模式，发现它具有一种称之为多值依赖（MVD）的数据依赖。

定义 5.9 设 $R(U)$ 是属性集 U 上的一个关系模式。X，Y，Z 是 U 的子集，并且 $Z=U-X-Y$。关系模式 $R(U)$ 中多值依赖 $X\rightarrow\rightarrow Y$ 成立，当且仅当对 $R(U)$ 的任一关系 r，给定的一对（x，z）值，有一组 Y 的值，这组值仅仅决定于 x 值而与 z 值无关。

例如，在关系模式 TEACHING 中，对于一个（物理，光学原理）有一组 T 值{李勇，王军}，这组值仅仅决定于课程 C 上的值（物理）。也就是说对于另一个（物理，普通物理学）它对应的一组 T 值仍是{李勇，王军}，尽管这时参考书 B 的值已经改变了。因此 T 多值依赖于 C，即 C→→T。

对于多值依赖的另一个等价的形式化的定义是：

在 $R(U)$ 的任一关系 r 中，如果存在元组 t，s 使得 $t[X]=s[X]$，那么就必然存在元组 $w, v \in r$，(w, v 可以与 s, t 相同)，使得 $w[X]=v[X]=t[X]$，而 $w[Y]=t[Y]$，$w[Z]=s[Z]$，$v[Y]=s[Y]$，$v[Z]=t[Z]$（即交换 s，t 元组的 Y 值所得的两个新元组必在 r 中），则 Y 多值依赖于 X，记为 $X \rightarrow\rightarrow Y$。这里，$X$，$Y$ 是 U 的子集，$Z=U-X-Y$。

若 $X \rightarrow\rightarrow Y$，而 $Z=\phi$ 即 Z 为空，则称 $X \rightarrow\rightarrow Y$ 为**平凡的多值依赖**。

下面再举一个具有多值依赖的关系模式的例子。

例 2 关系模式 WSC（W，S，C）中，W 表示仓库，S 表示保管员，C 表示商品。假设每个仓库有若干个保管员，有若干种商品。每个保管员保管所在的仓库的所有商品，每种商品被所有保管员保管。列出关系如下：

W	S	C
W_1	S_1	C1
W_1	S_1	C_2
W_1	S_1	C_3
W_1	S_2	C_1
W_1	S_2	C_2
W_1	S_2	C_3
W_3	S_3	C_4
W_2	S_3	C_5
W_2	S_4	C_4
W_2	S_4	C_5

按照语义对于 W 的每一个值 W_i，S 有一个完整的集合与之对应而不论 C 取何值。所以 $W \rightarrow\rightarrow S$。

如果用图来表示这种对应，则对应 W 的某一个值 W_i 的全部 S 值记作 $\{S\}_{W_i}$（表示此仓库工作的全部保管员），全部 C 值记作 $\{C\}_{W_i}$（表示在此仓库中存放的所有商品）。应当有 $\{S\}_{W_i}$ 中的每一个值和 $\{C\}_{W_i}$ 中的每一个 C 值对应。于是 $\{S\}_{W_i}$ 与 $\{C\}_{W_i}$ 之间正好形成一个完全二分图，因而 $W \rightarrow\rightarrow S$。

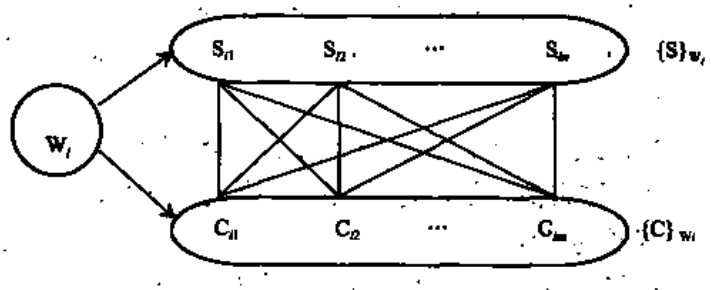


图 5.7

由于 C 与 S 的完全对称性，必然有 W→→C 成立。

多值依赖具有以下性质：

（1）多值依赖具有对称性。即若 $X\rightarrow\rightarrow Y$，则 $X\rightarrow\rightarrow Z$，其中 $Z=U-X-Y$。

从例 2 容易看出，因为每个保管员保管所有商品，同时每种商品被所有保管员保管，显然若 W→→S，必然有 W→→C。

（2）多值依赖的传递性。即若 $X\rightarrow\rightarrow Y$，$Y\rightarrow\rightarrow Z$，则 $X\rightarrow\rightarrow Z-Y$。

（3）函数依赖可以看作是多值依赖的特殊情况。即若 $X\rightarrow Y$，则 $X\rightarrow\rightarrow Y$。这是因为当 $X\rightarrow Y$ 时，对 X 的每一个值 x，Y 有一个确定的值 y 与之对应，所以 $X\rightarrow\rightarrow Y$。

（4）若 $X\rightarrow\rightarrow Y$，$X\rightarrow\rightarrow Z$，则 $X\rightarrow\rightarrow YZ$。

（5）若 $X\rightarrow\rightarrow Y$，$X\rightarrow\rightarrow Z$，则 $X\rightarrow\rightarrow Y\cap Z$。

（6）若 $X\rightarrow\rightarrow Y$，$X\rightarrow\rightarrow Z$，则 $X\rightarrow\rightarrow Y-Z$，$X\rightarrow\rightarrow Z-Y$。

多值依赖与函数依赖相比，具有下面两个基本的区别：

（1）多值依赖的有效性与属性集的范围有关。

若 $X\rightarrow\rightarrow Y$ 在 U 上成立则在 W（$XY\subseteq W\subseteq U$）上一定成立；反之则不然，即 $X\rightarrow\rightarrow Y$ 在 W（$W\subset U$）上成立，在 U 上并不一定成立。这是因为多值依赖的定义中不仅涉及属性组 X 和 Y，而且涉及 U 中其余属性 Z。

一般地，在 R（U）上若有 $X\rightarrow\rightarrow Y$ 在 W（$W\subset U$）上成立，则称 $X\rightarrow\rightarrow Y$ 为 R（U）的嵌入型多值依赖。

但是在关系模式 R（U）中函数依赖 $X\rightarrow Y$ 的有效性仅决定于 X，Y 这两个属性集的值。只要在 R（U）的任何一个关系 r 中，元组在 X 和 Y 上的值满足定义 5.1，则函数依赖 $X\rightarrow Y$ 在任何属性集 W（$XY\subseteq W\subseteq U$）上成立。

2. 若函数依赖 $X\rightarrow Y$ 在 R（U）上成立，则对于任何 $Y'\subset Y$ 均有 $X\rightarrow Y'$ 成立。而多值依赖 $X\rightarrow\rightarrow Y$ 若在 R（U）上成立，却不能断言对于任何 $Y'\subset Y$ 有 $X\rightarrow\rightarrow Y'$ 成立。

5.2.8 4NF

定义 5.10 关系模式 R（U，F）$\in$ 1NF，如果对于 R 的每个非平凡多值依赖 $X\rightarrow\rightarrow Y$（$Y\subseteq X$），$X$ 都含有码，则称 R（U，F）$\in$ 4NF。

4NF 就是限制关系模式的属性之间不允许有非平凡且非函数依赖的多值依赖。因为根据定义，对于每一个非平凡的多值依赖 $X\rightarrow\rightarrow Y$，$X$ 都含有候选码，于是就有 $X\rightarrow Y$，所以 4NF 所允许的非平凡的多值依赖实际上是函数依赖。

显然，如果一个关系模式是 4NF，则必为 BCNF。

在前面讨论的关系模式 WSC 中，W→→S，W→→C，它们都是非平凡的多值依赖。而 W 不是码，关系模式 WSC 的码是（W，S，C），即 All-Key。

因此 WSC∉4NF。

一个关系模式如果已达到了 BCNF 但不是 4NF，这样的关系模式仍然具有不好的性质。以 WSC 为例，WSC∉4NF，但是 WSC∈BCNF。对于 WSC 的某个关系，若某一仓库 W_i 有 n 个保管员，存放 m 件物品，则关系中分量为 W_i 的元组数目一定有 $m \times n$ 个。每个保管员重复存储 m 次，每种物品重复存储 n 次，数据的冗余度太大，因此还应该继续规范化使关系模式 WSC 达到 4NF。

可以用投影分解的方法消去非平凡且非函数依赖的多值依赖。例如可以把 WSC 分解为 WS（W，S），WC（W，C）。在 WS 中虽然有 W→→S，但这是平凡的多值依赖。WS 中已不存在非平凡的非函数依赖的多值依赖。所以 WS∈4NF，同理 WC∈4NF。

函数依赖和多值依赖是两种最重要的数据依赖。如果只考虑函数依赖，则属于 BCNF 的关系模式规范化程度已经是最高的了。如果考虑多值依赖，则属于 4NF 的关系模式规范化程度是最高的。事实上，数据依赖中除函数依赖和多值依赖之外，还有其他数据依赖。例如有一种连接依赖。函数依赖是多值依赖的一种特殊情况，而多值依赖实际上又是连接依赖的一种特殊情况。但连接依赖不像函数依赖和多值依赖可由语义直接导出，而是在关系的连接运算时才反映出来。存在连接依赖的关系模式仍可能遇到数据冗余及插入、修改、删除异常等问题。如果消除了属于 4NF 的关系模式中存在的连接依赖，则可以进一步达到 5NF 的关系模式。这里不再讨论连接依赖和 5NF，有兴趣的读者可以参阅有关书籍。

5.2.9 规范化小结

在关系数据库中，对关系模式的基本要求是满足第一范式。这样的关系模式就是合法的、允许的。但是，人们发现有些关系模式存在插入、删除异常、修改复杂，数据冗余等毛病。人们寻求解决这些问题的方法，这就是规范化的目的。

规范化的基本思想是逐步消除数据依赖中不合适的部分，使模式中的各关系模式达到某种程度的“分离”，即“一事一地”的模式设计原则。让一个关系描述一个概念、一个实体或者实体间的一种联系。若多于一个概念就把它“分离”出去。因此所谓规范化实质上是概念的单一化。

人们认识这个原则是经历了一个过程的。从认识非主属性的部分函数依赖的危害开始，2NF，3NF，BCNF，4NF 的提出是这个认识过程逐步深化的标志。如图 5.8 所示可概括这个过程。

关系模式的规范化过程是通过对关系模式的分解来实现的。把低一级的关系模式分解为若干个高一级的关系模式。这种分解不是唯一的。下面就将进一

步讨论分解后的关系模式与原关系模式“等价”的问题以及分解的算法。

图 5.8 各种范式及规范化过程

5.3 数据依赖的公理系统

数据依赖的公理系统是模式分解算法的理论基础，下面首先讨论函数依赖的一个有效而完备的公理系统——Armstrong 公理系统。

定义 5.11 对于满足一组函数依赖 F 的关系模式 $R\langle U, F\rangle$，其任何一个关系 r，若函数依赖 $X\rightarrow Y$ 都成立（即 r 中任意两元组 t，s，若 $t[X]=s[X]$，则 $t[Y]=s[Y]$），则称 F **逻辑蕴含** $X\rightarrow Y$。

为了求得给定关系模式的码，为了从一组函数依赖求得蕴含的函数依赖，例如已知函数依赖集 F，要问 $X\rightarrow Y$ 是否为 F 所蕴含，就需要一套推理规则，这组推理规则是 1974 年首先由 Armstrong 提出来的。

Armstrong 公理系统 设 U 为属性集总体，F 是 U 上的一组函数依赖，于是有关系模式 $R\langle U, F\rangle$。对 $R\langle U, F\rangle$ 来说有以下的推理规则：

- A1 自反律（Reflexivity）：若 $Y\subseteq X\subseteq U$，则 $X\rightarrow Y$ 为 F 所蕴含。
- A2 增广律（Augmentation）：若 $X\rightarrow Y$ 为 F 所蕴含，且 $Z\subseteq U$，则 $XZ\rightarrow YZ$[①] 为 F 所蕴含。
- A3 传递律（Transitivity）：若 $X\rightarrow Y$ 及 $Y\rightarrow Z$ 为 F 所蕴含，则 $X\rightarrow Z$ 为 F 所蕴含。

注意：由自反律所得到的函数依赖均是平凡的函数依赖，自反律的使用并不依赖于 F。

定理 5.1 Armstrong 推理规则是正确的。

① 为了简单起见，用 XZ 代表 $X\cup Z$。

下面从定义出发证明推理规则的正确性。

证

（1）设 $Y\subseteq X\subseteq U$ 。

对 $R\langle U, F\rangle$ 的任一关系 r 中的任意两个元组 t，s：

若 $t[X]=s[X]$，由于 $Y\subseteq X$，有 $t[y]=s[y]$，

所以 $X\rightarrow Y$ 成立，自反律得证①。

（2）设 $X\rightarrow Y$ 为 F 所蕴含，且 $Z\subseteq U$。

设 $R\langle U, F\rangle$ 的任一关系 r 中任意的两个元组 t，s：

若 $t[XZ]=s[XZ]$，则有 $t[X]=s[X]$和 $t[Z]=s[Z]$；

由 $X\rightarrow Y$，于是有 $t[Y]=s[Y]$，所以 $t[YZ]=s[YZ]$，所以 $XZ\rightarrow YZ$ 为 F 所蕴含，增广律得证。

（3）设 $X\rightarrow Y$ 及 $Y\rightarrow Z$ 为 F 所蕴含。

对 $R\langle U, F\rangle$ 的任一关系 r 中的任意两个元组 t，s。

若 $t[X]=s[X]$，由于 $X\rightarrow Y$，有 $t[Y]=s[Y]$；

再由 $Y\rightarrow Z$，有 $t[Z]=s[Z]$，所以 $X\rightarrow Z$ 为 F 所蕴含，传递律得证。

根据 A1，A2，A3 这三条推理规则可以得到下面三条很有用的推理规则：

- **合并规则**：由 $X\rightarrow Y$，$X\rightarrow Z$，有 $X\rightarrow YZ$。
- **伪传递规则**：由 $X\rightarrow Y$，$WY\rightarrow Z$，有 $XW\rightarrow Z$。
- **分解规则**：由 $X\rightarrow Y$ 及 $Z\subseteq Y$，有 $X\rightarrow Z$。

根据合并规则和分解规则，很容易得到这样一个重要事实：

引理 5.1 $X\rightarrow A_1A_2\cdots A_k$ 成立的充分必要条件是 $X\rightarrow A_i$ 成立（$i=1, 2, \cdots, k$）。

定义 5.12 在关系模式 $R\langle U, F\rangle$ 中为 F 所逻辑蕴含的函数依赖的全体叫做 F 的**闭包**，记为 F^+。

人们把自反律，传递律和增广律称为 Armstrong 公理系统。Armstrong 公理系统是有效的、完备的。Armsttrong 公理的**有效性**指的是：由 F 出发根据 Armstrong 公理推导出来的每一个函数依赖一定在 F^+中；**完备性**指的是 F^+中的每一个函数依赖，必定可以由 F 出发根据 Armstrong 公理推导出来。

要证明完备性，就首先要解决如何判定一个函数依赖是否属于由 F 根据 Armstrong 公理推导出来的函数依赖的集合。当然，如果能求出这个集合，问题就解决了。但不幸的是，这是一个 NP 完全问题。比如从 $F=\{X\rightarrow A_1, \cdots, X\rightarrow A_n\}$出发，至少可以推导出 2^n个不同的函数依赖为此引入了下面的概念：

定义 5.13 设 F 为属性集 U 上的一组函数依赖，$X\subseteq U$，$X_F^+=\{A|X\rightarrow A$ 能由 F 根据 Armstrong 公理导出$\}$，X_F^+ 称为属性集 X 关于函数依赖集 F 的闭包。

① $t[X]$表示元组 t 在属性（组）X 上的分量，等价于 $t.X$ 。

由引理 5.1 容易得出：

引理 5.2 设 F 为属性集 U 上的一组函数依赖，X，$Y \subseteq U$，$X \rightarrow Y$ 能由 F 根据 Armstrong 公理导出的充分必要条件是 $Y \subseteq X_F^+$。

于是，判定 $X \rightarrow Y$ 是否能由 F 根据 Armstrong 公理导出的问题，就转化为求出 X_F^+，判定 Y 是否为 X_F^+的子集的问题。这个问题由**算法 5.1** 解决了。

算法 5.1 求属性集 X（$X \subseteq U$）关于 U 上的函数依赖集 F 的闭包 X_F^+。

输入：X，F

输出：X_F^+

步骤：

（1）令 $X^{(0)}=X$，$i=0$

（2）求 B，这里 $B=\{A \mid (\exists V)\ (\exists W)\ (V \rightarrow W \in F \wedge V \subseteq X^{(i)} \wedge A \in W)\}$；

（3）$X^{(i+1)}=B \cup X^{(i)}$

（4）判断 $X^{(i+1)}=X^{(i)}$ 吗？

（5）若相等或 $X^{(i)}=U$ 则 $X^{(i)}$ 就是 X_F^+，算法终止。

（6）若否，则 $i=i+1$，返回第（2）步。

例 1 已知关系模式 $R\langle U, F\rangle$，

其中 $U=\{A, B, C, D, E\}$；$F=\{AB \rightarrow C, B \rightarrow D, C \rightarrow E, EC \rightarrow B, AC \rightarrow B\}$。

求 $(AB)_F^+$。

解 由算法 5.1，设 $X^{(0)}=AB$；

计算 $X^{(1)}$；逐一扫描 F 集合中各个函数依赖，找左部为 A，B 或 AB 的函数依赖。得到两个：$AB \rightarrow C$，$B \rightarrow D$。于是 $X^{(1)}=AB \cup CD=ABCD$。

因为 $X^{(0)} \neq X^{(1)}$，所以再找出左部为 $ABCD$ 子集的那些函数依赖，又得到 $C \rightarrow E$，$AC \rightarrow B$，于是 $X^{(2)}=X^{(1)} \cup BE=ABCDE$。

因为 $X^{(2)}$ 已等于全部属性集合，所以 $(AB)_F^+=ABCDE$。

对于**算法 5.1**，令 $a_i=|X^{(i)}|$，$\{a_i\}$形成一个步长大于 1 的严格递增的序列，序列的上界是 $|U|$，因此该算法最多 $|U|-|X|$ 次循环就会终止。

定理 5.2 Armstrong 公理系统是有效的、完备的。

Armstrong 公理系统的有效性可由**定理 5.1** 得到证明。这里给出完备性的证明。

证明完备性的逆否命题，即若函数依赖 $X \rightarrow Y$ 不能由 F 从 Armstrong 公理导出，那么它必然不为 F 所蕴含，它的证明分三步。

（1）若 $V \rightarrow W$ 成立，且 $V \subseteq X_F^+$，则 $W \subseteq X_F^+$。

证 因为 $V \subseteq X_F^+$，所以有 $X \rightarrow V$ 成立；于是 $X \rightarrow W$ 成立（因为 $X \rightarrow V$，$V \rightarrow W$），

所以 $W \subseteq X_F^+$。

（2）构造一张二维表 r，它由下列两个元组构成，可以证明 r 必是 $R\langle U, F\rangle$ 的一个关系，即 F 中的全部函数依赖在 r 上成立。

X_F^+	$U-X_F^+$
11.....1	00.....0
11.....1	11.....1

若 r 不是 $R\langle U, F\rangle$ 的关系，则必由于 F 中有函数依赖 $V\to W$ 在 r 上不成立所致。由 r 的构成可知，V 必定是 X_F^+ 的子集，而 W 不是 X_F^+ 的子集，可是由第（1）步，$W\subseteq X_F^+$，矛盾。所以 r 必是 $R\langle U, F\rangle$ 的一个关系。

（3）若 $X\to Y$ 不能由 F 从 Armstrong 公理导出，则 Y 不是 X_F^+的子集，因此必有 Y' 的子集 Y' 满足 $Y\subseteq U-X_F^+$，则 $X\to Y$ 在 r 中不成立，即 $X\to Y$ 必不为 $R\langle U, F\rangle$ 蕴含。

Armstrong 公理的完备性及有效性说明了“导出”与“蕴含”是两个完全等价的概念。于是 F^+ 也可以说成是由 F 出发借助 Armstrong 公理导出的函数依赖的集合。

从蕴含（或导出）的概念出发，又引出了两个函数依赖集等价和最小依赖集的概念。

定义 5.14 如果 $G^+=F^+$，就说函数依赖集 F 覆盖 G（F 是 G 的覆盖，或 G 是 F 的覆盖），或 F 与 G 等价。

引理 5.3 $F^+=G^+$ 的充分必要条件是 $F\subseteq G^+$，和 $G\subseteq F^+$。

证 必要性显然，只证充分性。

（1）若 $F\subseteq G^+$，则 $X_F^+\subseteq X_{G^+}^+$。

（2）任取 $X\to Y\in F^+$ 则有 $Y\subseteq X_F^+\subseteq X_{G^+}^+$。

所以 $X\to Y\in (G^+)^+=G^+$。即 $F^+\subseteq G^+$。

（3）同理可证 $G^+\subseteq F^+$，所以 $F^+=G^+$。

而要判定 $F\subseteq G^+$，只须逐一对 F 中的函数依赖 $X\to Y$，考察 Y 是否属于 $X_{G^+}^+$就行了。因此引理 5.3 给出了判断两个函数依赖集等价的可行算法。

定义 5.15 如果函数依赖集 F 满足下列条件，则称 F 为一个极小函数依赖集。亦称为最小依赖集或最小覆盖。

（1）F 中任一函数依赖的右部仅含有一个属性。

（2）F 中不存在这样的函数依赖 X→A，使得 F 与 F-{X→A}等价。

（3）F 中不存在这样的函数依赖 X→A，X 有真子集 Z 使得 F-{X→A}∪{Z→A}与 F 等价。

例 2 考察 5.1 节中的关系模式 $S\langle U, F\rangle$，其中：

U={SNO，SDEPT，MN，CNAME，G}，

F={SNO→SDEPT，SDEPT→MN，（SNO，CNAME）→G}

设 F′ ={SNO→SDEPT，SNO→MN，SDEPT→MN，（SNO，CNAME）→G，（SNO，SDEPT）→SDEPT}

根据定义 5.15 可以验证 F 是最小覆盖，而 F' 不是。因为 $F'-\{SNO\rightarrow MN\}$ 与 F' 等价，$F'-\{(SNO，SDEPT)\rightarrow SDEPT\}$ 与 F' 等价。

定理 5.3 每一个函数依赖集 F 均等价于一个极小函数依赖集 F_m。此 F_m 称为 F 的最小依赖集。

证 这是一个构造性的证明，分三步对 F 进行“极小化处理”，找出 F 的一个最小依赖集来。

（1）逐一检查 F 中各函数依赖 FD_i：$X\rightarrow Y$，若 $Y=A_1A_2\cdots A_k$，$k>2$，则用 $\{X\rightarrow A_j|j=1，2，\cdots，k\}$ 来取代 $X\rightarrow Y$。

（2）逐一检查 F 中各函数依赖 FD_i：$X\rightarrow A$，令 $G=F-\{X\rightarrow A\}$，若 $A\in X_G^+$，则从 F 中去掉此函数依赖（因为 F 与 G 等价的充要条件是 $A\in X_G^+$）。

（3）逐一取出 F 中各函数依赖 FD_i：$X\rightarrow A$，设 $X=B_1B_2\cdots B_m$，逐一考查 B_i（$i=1，2，\cdots，m$），若 $A\in(X-B_i)_F^+$，则以 $X-B_i$ 取代 X（因为 F 与 $F-\{X\rightarrow A\}\cup\{Z\rightarrow A\}$ 等价的充要条件是 $A\in Z_F^+$ 其中 $Z=X-B_i$）。

最后剩下的 F 就一定是极小依赖集，并且与原来的 F 等价。因为对 F 的每一次“改造”都保证了改造前后的两个函数依赖集等价。这些证明很显然，请读者自行补上。

应当指出，F 的最小依赖集 F_m 不一定是唯一的，它与对各函数依赖 FD_i 及 $X\rightarrow A$ 中 X 各属性的处置顺序有关。

例 3 $F=\{A\rightarrow B，B\rightarrow A，B\rightarrow C，A\rightarrow C，C\rightarrow A\}$

$F_{m1}=\{A\rightarrow B，B\rightarrow C，C\rightarrow A\}$

$F_{m2}=\{A\rightarrow B，B\rightarrow A，A\rightarrow C，C\rightarrow A\}$

这里给出了 F 的两个最小依赖集 F_{m1}，F_{m2}。

若改造后的 F 与原来的 F 相同，说明 F 本身就是一个最小依赖集，因此定理 5.3 的证明给出的极小化过程也可以看成是检验 F 是否为极小依赖集的一个算法。

两个关系模式 R_1（U，F），R_2（U，G），如果 F 与 G 等价，那么 R_1 的关系一定是 R_2 的关系。反过来，R_2 的关系也一定是 R_1 的关系。所以在 R（U，F）中用与 F 等价的依赖集 G 来取代 F 是允许的。

*5.4 模式的分解

在对函数依赖的基本性质有了初步的了解之后，可以具体地来讨论模式的分解了。

定义 5.16 关系模式 $R\langle U, F\rangle$ 的一个分解是指 $\rho=\{R_1\langle U_1, F_1\rangle, R_2\langle U_2, F_2\rangle, \cdots, R_n\langle U_n, F_n\rangle\}$

某中 $U=\bigcup_{i=1}^{n} U_i$，并且没有 $U_i\subseteq U_j$，$1\leqslant i, j\leqslant n$，$F_i$ 是 F 在 U_i 上的投影。

所谓“F_i 是 F 在 U_i 上的投影”的确切定义是：

定义 5.17 函数依赖集合 $\{X\rightarrow Y|X\rightarrow Y\in F^+\wedge XY\subseteq U_i\}$ 的一个覆盖 F_i 叫做 F 在属性 U_i 上的投影。

5.4.1 模式分解的三个定义

对于一个模式的分解是多种多样的，但是分解后产生的模式应与原模式等价。

人们从不同的角度去观察问题，对“等价”的概念形成了三种不同的定义：

- 分解具有“无损连接性”（Lossless join）。
- 分解要“保持函数依赖”（Preserve dependency）。
- 分解既要“保持函数依赖”，又要具有“无损连接性”。

这三个定义是实行分解的三条不同的准则。按照不同的分解准则，模式所能达到的分离程度各不相同，各种范式就是对分离程度的测度。

这一节要讨论的问题是：

（1）“无损连接性”和“保持函数依赖”的含义是什么？如何判断？

（2）对于不同的分解等价定义，究竟能达到何种程度的分离，即分离后的关系模式是第几范式。

（3）如何实现分离，即给出分解的算法。

先来看两个例子，说明按**定义 5.16**，若只要求 $R\langle U, F\rangle$ 分解后的各关系模式所含属性的“并”等于 U，这个限定是很不够的。

一个关系分解为多个关系，相应地原来存储在一张二维表内的数据就要分散存储到多张二维表中，要使这个分解有意义，起码的要求是后者不能丢失前者的信息。

例 4 已知关系模式 $R\langle U, F\rangle$，其中 U={SNO, SDEPT, MN}，F={SNO→SDEPT, SDEPT→MN}。$R\langle U, F\rangle$ 的元组语义是学生 SNO 正在 SDEPT

系学习，其系主任是 MN。并且一个学生（SNO）只在一个系学习，一个系只有一名系主任。R 的一个关系见表 5.3。

由于 R 中存在传递函数依赖 SNO→MN，它会发生更新异常。例如，如果 S4 毕业，则 D3 系的系主任是王一的信息也就丢掉了。反过来，如果一个系 D5 尚无在校学生，那么这个系的系主任是赵某的信息也无法存入。于是进行了如下分解：

$$\rho_1=\{R_1(\text{SNO},\ \phi),\ R_2(\text{SDEPT},\ \phi),\ R_3(\text{MN},\ \phi)\}$$

表 5.3

SNO	SDEPT	MN
S1	D1	张 五
S2	D1	张 五
S3	D2	李 四
S4	D3	王 一

分解后诸 R_i 的关系 r_i 是 R 在 U_i 上的投影，即 $r_i=R[U_i]$

r_1={S1，S2，S3，S4}，r_2={D1，D2，D3}，r_3={张五，李四，王一}。

对于分解后的数据库，要回答“S1 在哪个系学习”也不可能了。这样的分解还有什么用呢。

如果分解后的数据库能够恢复到原来的情况，不丢失信息的要求也就达到了。R_i 向 R 的恢复是通过自然连接来实现的，这就产生了无损连接性的概念。显然，本例的分解 ρ_1 所产生的诸关系自然连接的结果实际上是它们的笛卡尔积，元组增加了，信息丢失了。

于是对 R 又进行另一种分解：

ρ_2={R_1（{SNO，SDEPT}，{SNO→SDEPT}），R_2（{SNO，MN}，{SNO→MN}）}

以后可以证明 ρ_2 对 R 的分解是可恢复的，但是前面提到的插入和删除异常仍然没有解决，原因就在于原来在 R 中存在的函数依赖 SDEPT→MN，现在在 R_1 和 R_2 中都不再存在了。因此人们又要求分解具有“保持函数依赖”的特性。

最后对 R 进行了以下一种分解：

ρ_3={R_1（{SNO，SDEPT}，{SNO→SDEPT}），R_2（{SDEPT，MN}，{SDEPT→MN}）}

可以证明分解 ρ_3 既具有无损连接性，又保持函数依赖。它解决了更新异常，又没有丢失原数据库的信息，这是所希望的分解。

由此，可以看出为什么要提出对数据库模式“等价”的三个不同定义的

原因。

下面严格地定义分解的无损连接性和保持函数依赖性并讨论它们的判别算法。

5.4.2 分解的无损连接性和保持函数依赖性

先定义一个记号：设$\rho=\{R_1\langle U_1, F_1\rangle, \cdots, R_k\langle U_k, F_k\rangle\}$是$R\langle U, F\rangle$的一个分解，$r$是$R\langle U, F\rangle$的一个关系。定义$m_\rho(r)=\bowtie_{i=1}^{k}\pi_{R_i}(r)$，即$m_\rho(r)$是$r$在$\rho$中各关系模式上投影的连接。这里$\pi_{R_i}(r)=\{t.U_i|t\in r\}$。

引理 5.4 设$R\langle U, F\rangle$是一个关系模式，$\rho=\{R_1\langle U_1, F_1\rangle, \cdots, R_k\langle U_k, F_k\rangle\}$是$R$的一个分解，$r$是$R$的一个关系，$r_i=\pi_{R_i}(r)$，则

（1）$r\subseteq m_\rho(r)$

（2）若$s=m_\rho(r)$，则$\pi_{R_i}(s)=r_i$

（3）$m_\rho(m_\rho(r))=m_\rho(r)$

证

（1）证明r中的任何一个元组属于$m_\rho(r)$。

任取r中的一个元组t，$t\in r$，设$t_i=t.U_i$（$i=1, 2, \cdots, k$）。对k进行归纳可以证明$t_1t_2\cdots t_k\in\bowtie_{i=1}^{k}\pi_{R_i}(r)$，所以$t\in m_\rho(r)$，即$r\subseteq m_\rho(r)$。

（2）由（1）得到$r\subseteq m_\rho(r)$，已设$s=m_\rho(r)$，所以，$r\subseteq s$，$\pi_{R_i}(r)\subseteq\pi_{R_i}(s)$现只需证明$\pi_{R_i}(s)\subseteq\pi_{R_i}(r)$，就有$\pi_{R_i}(s)=\pi_{R_i}(r)=r_i$。

任取$S_i\in\pi_{R_i}(s)$，必有S中的一个元组v，使得$v.U_i=S_i$。根据自然连接的定义$v=t_1t_2\cdots t_k$，对于其中每一个t_i必存在r中的一个元组t，使得$t.U_i=t_i$。由前面$\pi_{R_i}(r)$的定义即得$t_i\in\pi_{R_i}(r)$。又因$v=t_1t_2\cdots t_k$，故$v.U_i=t_i$。又由上面证得：$v.U_i=S_i$，$t_i\in\pi_{R_i}(r)$，故$S_i\in\pi_{R_i}(r)$。即$\pi_{R_i}(s)\subseteq\pi_{R_i}(r)$。故$\pi_{R_i}(s)=\pi_{R_i}(r)$。

（3）$m_\rho(m_\rho(r))=\bowtie_{i=1}^{k}\pi_{R_i}(m_\rho(r))=\bowtie_{i=1}^{k}\pi_{R_i}(s)=\bowtie_{i=1}^{k}\pi_{R_i}(r)=m_\rho(r)$

定义 5.18 $\rho=\{R_1\langle U_1, F_1\rangle, \cdots, R_k\langle U_k, F_k\rangle\}$是$R\langle U, F\rangle$的一个分解，若对$R\langle U, F\rangle$的任何一个关系$r$均有$r=m_\rho(r)$成立，则称分解$\rho$具有无损连接性。简称$\rho$为无损分解。

直接根据定义 5.18 去鉴别一个分解的无损连接性是不可能的，算法 5.2 给出了一个判别的方法。

算法 5.2 判别一个分解的无损连接性。

$\rho=\{R_1\langle U_1, F_1\rangle, \cdots, R_k\langle U_k, F_k\rangle\}$是$R\langle U, F\rangle$的一个分解，$U=\{A_1, \cdots, A_n\}$，$F=\{FD_1, FD_2, \cdots, FD_p\}$，不妨设$F$是一极小依赖集，记$FD_i$为$X_i\to A_{1i}$。

（1）建立一张n列k行的表。每一列对应一个属性，每一行对应分解中

的一个关系模式。若属性 A_j 属于 U_i，则在 j 列 i 行交叉处填上 a_j，否则填上 b_{ij}；

（2）对每一个 FD_i 做下列操作：找到 X_i 所对应的列中具有相同符号的那些行。考察这些行中 l_i 列的元素，若其中有 a_{li}，则全部改为 a_{li}；否则全部改为 b_{mli}；m 是这些行的行号最小值。

应当注意的是，若某个 b_{tli} 被更动，那么该表的 li 列中凡是 b_{tli} 的符号（不管它是开始找到的那些行）均应作相应的更改。

如在某次更改之后，有一行成为 a_1，a_2，…，a_n。则算法终止。ρ具有无损连接性，否则ρ不具有无损连接性。

对 f 中 p 个 FD 逐一进行一次这样的处置，称为对 F 的一次扫描。

（3）比较扫描前后，表有无变化。如有变化，则返回第（2）步，否则算法终止。

如果发生循环，那么前次扫描至少应使该表减少一个符号，表中符号有限，因此循环必然终止。

定理 5.4 ρ为无损连接分解的充分必要条件是**算法 5.2** 终止时，表中有一行为 a_1，a_2，…，a_n。

证明从略。

例 5 已知 $R\langle U, F\rangle$，$U=\{A, B, C, D, E\}$，$F=\{AB\rightarrow C, C\rightarrow D, D\rightarrow E\}$，$R$ 的一个分解为 $R_1(A, B, C)$，$R_2(C, D)$，$R_3(D, E)$。

（1）首先构造初始表，如图 5.9（a）。

（2）对 $AB\rightarrow C$，因各元组的第 1、2 列没有相同的分量，所以表不改变。由 $C\rightarrow D$ 可以把 b_{14} 改为 a_4，再由 $D\rightarrow E$ 可使 b_{15}，b_{25} 全改为 a_5。最后结果为图 5.9（b）。表中第 1 行成为 a_1，a_2，a_3，a_4，a_5，所以此分解具有无损连接性。

A	B	C	D	E
a_1	a_2	a_3	b_{14}	b_{15}
b_{21}	b_{22}	a_3	a_4	b_{25}
b_{31}	b_{32}	b_{33}	a_4	a_5

（a）

A	B	C	D	E
a_1	a_2	a_3	a_4	a_5
b_{21}	b_{22}	a_3	a_4	a_5
b_{31}	b_{32}	b_{33}	a_4	a_5

（b）

图 5.9 分解具有无损连接的一个实例

当关系模式 R 分解为两个关系模式 R_1，R_2 时有下面的判定准则。

定理 5.5 $R\langle U, F\rangle$ 的一个分解 $\rho=\{R_1\langle U_1, F_1\rangle R_2\langle U_2, F_2\rangle\}$ 具有无损连接性的充分必要条件是：$U_1\cap U_2\rightarrow U_1-U_2\in F^+$ 或 $U_1\cap U_2\rightarrow U_2-U_1\in F^+$。

定理的证明留给读者完成。

定义 5.19 若 $F^+=(\bigcup_{i=1}^{k}F_i)^+$，则 $R\langle U, F\rangle$ 的分解 $\rho=\{R_1\langle U_1, F_1\rangle, \cdots, R_k\langle U_k, F_k\rangle\}$ 保持函数依赖。

引理 5.3 给出了判断两个函数依赖集等价的可行算法。因此**引理 5.3** 也给出了判别 R 的分解 ρ 是否保持函数依赖的方法。

5.4.3 模式分解的算法

关于模式分解的几个重要事实是：

（1）若要求分解保持函数依赖，那么模式分离总可以达到 3NF，但不一定能达到 BCNF；

（2）若要求分解既保持函数依赖，又具有无损连接性，可以达到 3NF，但不一定能达到 BCNF；

（3）若要求分解具有无损连接性，那一定可达到 4NF。

它们分别由**算法 5.3**、**算法 5.4**、**算法 5.5**、**算法 5.6** 来实现。

算法 5.3 （合成法）转换为 3NF 的保持函数依赖的分解。

（1）对 $R\langle U, F\rangle$ 中的函数依赖集 F 进行“极小化处理”（处理后得到的依赖集仍记为 F）。

（2）找出不在 F 中出现的属性，把这样的属性构成一个关系模式。把这些属性从 U 中去掉，剩余的属性仍记为 U。

（3）若有 $X\rightarrow A\in F$，且 $XA=U$，则 $\rho=\{R\}$，算法终止。

（4）否则，对 F 按具有相同左部的原则分组（假定分为 k 组），每一组函数依赖 F_i' 所涉及的全部属性形成一个属性集 U_i。若 $U_i\subseteq U_j$ $(i\neq j)$ 就去掉 U_i。由于经过了步骤（2），故 $U=\bigcup_{i=1}^{k}U_i$，于是 $\rho=\{R_1\langle U_1, F_1\rangle, \cdots, R_k\langle U_k, F_k\rangle\}$ 构成 $R\langle U, F\rangle$ 的一个保持函数依赖的分解，并且每个 $R_i\langle U_i, F_i\rangle$ 均属 3NF。这里 F_i 是 F 在 U_i 上的投影，并且 F_i 不一定与 F_i' 相等，但 F_i' 一定被 F_i 所包含，因此分解 ρ 保持函数依赖是显然的。

下面证明每一个 $R_i\langle U_i, F_i\rangle$ 一定属于 3NF。

设 $F_i'=\{X\rightarrow A_1, X\rightarrow A_2, \cdots, X\rightarrow A_k\}$，$U_i=\{X, A_1, A_2, \cdots, A_k\}$

（1）$R_i\langle U_i, F_i\rangle$ 一定以 X 为码。

（2）若 $R_i\langle U_i, F_i\rangle$ 不属于 3NF，则必存在非主属性 A_m $(1\leqslant m\leqslant k)$ 及

属性组合 Y，$A_m \notin Y$，使得 $X \to Y$，$Y \to A_m \in F_i^+$，而 $Y \to X \notin F_i^+$。

若 $Y \subset X$，则与 $X \to A_m$ 属于最小依赖集 F 相矛盾，因而 $Y \not\subset X$。不妨设 $Y \cap X = X_1$，$Y - X = \{A_1, \cdots, A_\rho\}$，令 $G = F - \{X \to A_m\}$，显然 $Y \subseteq X_G^+$，即 $X \to Y \in G^+$。

可以断言 $Y \to A_m$ 也属于 G^+。因为 $Y \to A_m \in F_i^+$，所以 $A_m \in F^+$。若 $Y \to A_m$ 不属于 G^+，则在求 Y_F^+的算法中，只有使用 $X \to A_m$ 才能将 A_m 引入。于是按**算法 5.1** 必有 j，使得 $X \subseteq Y^{(j)}$，于是 $Y \to X$ 成立是矛盾的。

于是 $X \to A_m$ 属于 G^+，与 F 是最小依赖集相矛盾。所以 $R_i \langle U_i, F_i \rangle$ 一定属于 3NF。

算法 5.4 转换为 3NF 既有无损连接性又保持函数依赖的分解。

(1) 设 X 是 $R \langle U, F \rangle$ 的码。$R \langle U, F \rangle$ 已由**算法 5.3** 分解为 $\rho = \{R_1 \langle U_1, F_1 \rangle, R_2 \langle U_2, F_2 \rangle, \cdots, R_k \langle U_k, F_k \rangle\}$，令 $\tau = \rho \cup \{R^* \langle X, F_x \rangle\}$。

(2) 若有某个 U_i，$X \subseteq U_i$，将 $R^* \langle X, F_x \rangle$ 从 τ 中去掉，

(3) τ 就是所求的分解。

$R^* \langle X, F_x \rangle$ 显然属于 3NF，而 τ 保持函数依赖也很显然，只要判定 τ 的无损连接性就行了。

由于 τ 中必有某关系模式 $R(T)$ 的属性组 $T \supseteq X$。由于 X 是 $R \langle U, F \rangle$ 的码，任取 $U - T$ 中的属性 B，必存在某个 i，使 $B \in T^{(i)}$（按**算法 5.1**）。对 i 施行归纳法可以证明由**算法 5.2**，表中关系模式 $R(T)$ 所在的行一定可成为 $a_1, a_2, \cdots, a_n$。τ 的无损连接性得证。

算法 5.5 转换为 BCNF 的无损连接分解（分解法）。

(1) 令 $\rho = \{R \langle U, F \rangle\}$

(2) 检查 ρ 中各关系模式是否均属于 BCNF。若是，则算法终止。

(3) 设 ρ 中 $R_i \langle U_i, F_i \rangle$ 不属于 BCNF，那么必有 $X \to A \in F_i^+$（$A \notin X$），且 X 非 R_i 的码。因此，XA 是 U_i 的真子集。对 R_i 进行分解：$\sigma = \{S_1, S_2\}$，$U_{S1} = XA$，$U_{S2} = U_i - \{A\}$，以 σ 代替 $R_i \langle U_i, F_i \rangle$ 返回第 (2) 步。

由于 U 中属性有限，因而有限次循环后**算法 5.5** 一定会终止。

这是一个自顶向下的算法。它自然地形成一棵对 $R \langle U, F \rangle$ 的二叉分解树。应当指出，$R \langle U, F \rangle$ 的分解树不一定是唯一的。这与步骤 (3) 中具体选定的 $X \to A$ 有关。

算法 5.5 最初令 $\rho = \{R \langle U, F \rangle\}$，显然 ρ 是无损连接分解，而以后的分解则由下面的**引理 5.5** 保证了它的无损连接性。

引理 5.5 若 $\rho = \{R_1 \langle U_1, F_1 \rangle, \cdots, R_k \langle U_k, F_k \rangle\}$ 是 $R \langle U, F \rangle$ 的一个无损连接分解，$\sigma = \{S_1, S_2, \cdots, S_m\}$ 是 ρ 中 $R_i \langle U_i, F_i \rangle$ 的一个无损连接分解，那么

$$\rho' = \{R_1, R_2, \cdots, R_{i-1}, S_1, \cdots, S_m, R_{i+1}, \cdots, R_k\},$$

$\rho''=\{R_1, \cdots, R_k, R_{k+1}, \cdots, R_n\}$（$\rho''$ 是 $R\langle U, F\rangle$ 包含 ρ 的关系模式集合的分解），均是 $R\langle U, F\rangle$ 的无损连接分解。

证明的关键是自然连接的结合律，给出结合律的证明，其他部分留给读者。

引理 5.6 $(R_1 \bowtie R_2) \bowtie R_3 = R_1 \bowtie (R_2 \bowtie R_3)$

证

设 r_i 是 $R_i\langle U_i, F_i\rangle$ 的关系，i=1，2，3

设 $U_1 \cap U_2 \cap U_3 = V$；

$U_1 \cap U_2 - V = X$；

$U_2 \cap U_3 - V = Y$；

$U_1 \cap U_3 - V = Z$（如图 5.10 所示）。

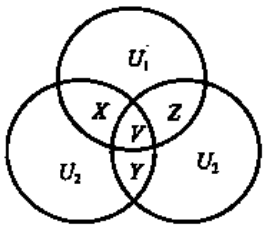


图 5.10

容易证明 t 是 $(R_1 \bowtie R_2) \bowtie R_3$ 中的一个元组的充要条件是：

T_{R_1}，T_{R_2}，T_{R_3} 是 t 的连串，这里

$T_{R_i} \in r_i$（i=1，2，3），$T_{R_1}[V]=T_{R_2}[V]=T_{R_3}[V]$，$T_{R_1}[X]=T_{R_2}[X]$，$T_{R_1}[Z]=T_{R_3}[Z]$，$T_{R_2}[Y]=T_{R_3}[Y]$。而这也是 t 为 $R \bowtie (R_2 \bowtie R_3)$ 中的元组的充要条件。于是有

$$(R_1 \bowtie R_2) \bowtie R_3 = R_1 \bowtie (R_2 \bowtie R_3)$$

在 5.2.8 节中已经指出，一个关系模式中若存在多值依赖（指非平凡的非函数依赖的多值依赖），则数据的冗余度大而且存在插入、修改、删除异常等问题。为此要消除这种多值依赖，使模式分离达到一个新的高度 4NF。下面讨论达到 4NF 的具有无损连接性的分解。

定理 5.6 关系模式 $R\langle U, D\rangle$ 中，D 为 R 中函数依赖 FD 和多值依赖 MVD 的集合。则 $X \rightarrow\rightarrow Y$ 成立的充要条件是 R 的分解 $\rho=\{R_1(X, Y), R_2(X, Z)\}$ 具有无损连接性，其中 $Z=U-X-Y$。

证 先证充分性。

若 ρ 是 R 的一个无损连接分解，则对 $R\langle U, F\rangle$ 的任一关系 r 有：

$$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r)$$

设 $t, s \in r$，且 $t[X]=s[X]$，于是 $t[XY]$，$s[XY] \in \pi_{R_1}(r)$，$t[XZ]$，$s[XZ] \in \pi_{R_2}(r)$。由于 $t[X]=s[X]$，所以 $t[XY] \cdot s[XZ]$ 与 $t[XZ] \cdot s[XY]$ 均属于 $\pi_{R_1}(r) \bowtie \pi_{R_2}(r)$，也即属于 r。令 $u=t[XY] \cdot s[XZ]$，$v=t[XZ] \cdot s[XY]$，就有 $u[X]=v[X]=t[X]$，$u[Y]=t[Y]$，$u[Z]=s[Z]$，$v[Y]=s[Y]$，$v[Z]=t[Z]$，所以 $X \rightarrow\rightarrow Y$ 成立。

再证必要性。

若 $X \rightarrow\rightarrow Y$ 成立，对于 $R\langle U, D\rangle$ 的任一关系 r，任取 $\omega \in \pi_{R_1}(r) \bowtie \pi_{R_2}(r)$，则必有 $t, s \in r$，使得 $\omega = t[XY] \cdot s[XZ]$，由于 $X \rightarrow\rightarrow Y$ 对 $R\langle U, D\rangle$ 成立，ω 应当属于 r，所以 ρ 是无损连接分解。

定理 5.6 给出了对 $R\langle U, D\rangle$ 的一个无损的分解方法。若 $R\langle U, D\rangle$ 中 $X\rightarrow\rightarrow Y$ 成立，则 R 的分解 $\rho=\{R_1\langle X, Y\rangle, R_2\langle X, Z\rangle\}$ 具有无损连接性。

算法 5.6 达到 4NF 的具有无损连接性的分解。

首先使用**算法 5.5**，得到 R 的一个达到了 BCNF 的无损连接分解 ρ。然后对某一 $R_i\langle U_i, D_i\rangle$，若不属于 4NF，则可按**定理 5.6** 的作法进行分解。直到每一个关系模式均属于 4NF 为止。**定理 5.6** 和**引理 5.5** 保证了最后得到的分解的无损连接性。

关系模式 $R\langle U, D\rangle$，U 是属性总体集，D 是 U 上的一组数据依赖（函数依赖和多值依赖），对于包含函数依赖和多值依赖的数据依赖有一个有效且完备的公理系统。

A1：若 $Y\subseteq X\subseteq U$，则 $X\rightarrow Y$；

A2：若 $X\rightarrow Y$，且 $Z\subseteq U$，则 $XZ\rightarrow YZ$；

A3：若 $X\rightarrow Y$，$Y\rightarrow Z$，则 $X\rightarrow Z$；

A4：若 $X\rightarrow\rightarrow Y$，$V\subseteq W\subseteq U$，则 $XW\rightarrow\rightarrow YV$；

A5：若 $X\rightarrow\rightarrow Y$，则 $X\rightarrow\rightarrow U-X-Y$；

A6：若 $X\rightarrow\rightarrow Y$，$Y\rightarrow\rightarrow Z$，则 $X\rightarrow\rightarrow Z-Y$；

A7：若 $X\rightarrow Y$，则 $X\rightarrow\rightarrow Y$；

A8：若 $X\rightarrow\rightarrow Y$，$W\rightarrow Z$，$W\cap Y=\phi$，$Z\subseteq Y$，则 $X\rightarrow Z$。

公理系统的有效性是指从 D 出发根据 8 条公理推导出的函数依赖或多值依赖一定为 D 蕴含；完备性是指凡 D 所蕴含的函数依赖或多值依赖均可以从 D 根据 8 条公理推导出来。也就是说，在函数依赖和多值依赖的条件下，“蕴含”与“导出”仍是等价的。

A1，A2，A3 的有效性前面 5.3 节已证明，其余的有效性证明，留给读者。

由 8 条公理可得如下 4 条有用的推理规则：

（1）合并规则：$X\rightarrow\rightarrow Y$，$X\rightarrow\rightarrow Z$，则 $X\rightarrow\rightarrow YZ$；

（2）伪传递规则：$X\rightarrow\rightarrow Y$，$WY\rightarrow Z$，则 $WX\rightarrow\rightarrow Z-WY$；

（3）混合伪传递规则：$X\rightarrow\rightarrow Y$，$XY\rightarrow Z$，则 $X\rightarrow Z-Y$；

（4）分解规则：$X\rightarrow\rightarrow Y$，$X\rightarrow Z$，则 $X\rightarrow\rightarrow Y\cap Z$，$X\rightarrow\rightarrow Y-Z$，$X\rightarrow\rightarrow Z-Y$。

5.5 小 结

在函数依赖、多值依赖的范畴内讨论了关系模式的规范化，其基本思想可用图 5.11 表示。

在整个讨论过程中，只采用了两种关系运算——投影和自然连接，并且总

是从一个关系模式出发，而不是从一组关系模式出发实行分解。“等价”的定义也是一组关系模式与一个关系模式的“等价”，这就是说，在开始讨论问题时事实上已经假设了存在着一个单一的关系模式，这就是泛关系假设。

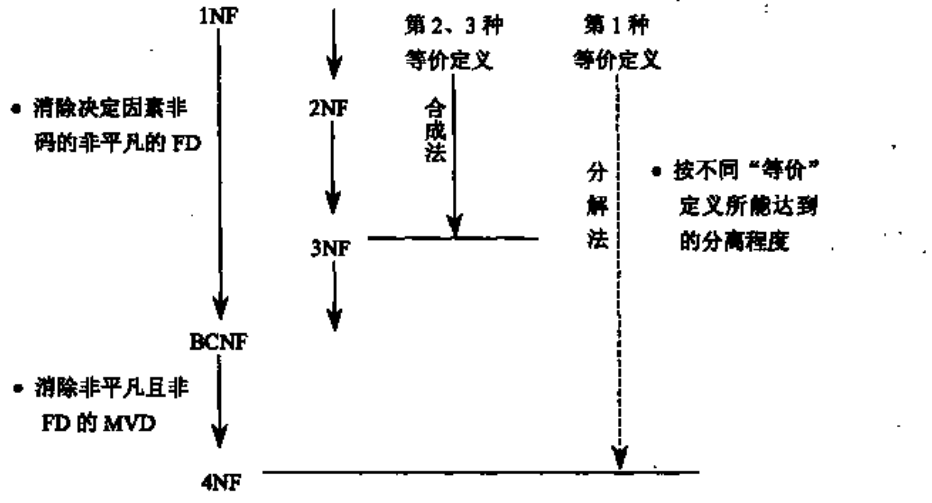


图 5.11 关系模式的规范化小结

本章一开始，就指出这是研究模式设计的一种特殊情况：“假设已知一个模式 Sϕ，它仅由单个关系模式组成，问题是要设计一个模式 SD，它与 Sϕ‘等价’，但在某些方面更好一些”。

泛关系假设是运用规范化理论时的障碍，因为承认了泛关系假设，就等于承认了现实世界各实体间只能有一种联系，而这常常是办不到的。比如工人与机器之间就可以存在“使用”、“维护”等多种联系。对此，人们提出了一些办法，希望解决这个矛盾。例如建立一个不受泛关系假设限制的理论，或者采用某些手段使现实世界向信息世界转换时适合于泛关系的要求。

最后应当强调的是，规范化理论为数据库设计提供了理论的指南和工具，但仅仅是指南和工具。并不是规范化程度越高，模式就越好，而必须结合应用环境和现实世界的具体情况合理地选择数据库模式。

习 题

1. 理解并给出下列术语的定义：

函数依赖、部分函数依赖、完全函数依赖、传递依赖、候选码、主码、外码、全码（All-key）、INF、2NF、3NF、BCNF、多值依赖、4NF。

2. 建立一个关于系、学生、班级、学会等诸信息的关系数据库。

描述学生的属性有：学号、姓名、出生年月、系名、班号、宿舍区。

描述班级的属性有：班号、专业名、系名、人数、入校年份。

描写系的属性有：系名、系号、系办公室地点、人数。

描述学会的属性有：学会名、成立年份、地点、人数。

有关语义如下：一个系有若干专业，每个专业每年只招一个班，每个班有若干学生。一个系的学生住在同一宿舍区。每个学生可参加若干学会，每个学会有若干学生。学生参加某学会有一个入会年份。

请给出关系模式，写出每个关系模式的极小函数依赖集，指出是否存在传递函数依赖，对于函数依赖左部是多属性的情况讨论函数依赖是完全函数依赖，还是部分函数依赖。

指出各关系的候选码、外部码，有没有全码存在？

*3. 试由 Armostrong 公理系统推导出下面三条推理规则：

（1）合并规则：若 $X\to Z$，$X\to Y$，则有 $X\to YZ$

（2）伪传递规则：由 $X\to Y$，$WY\to Z$ 有 $XW\to Z$

（3）分解规则：$X\to Y$，$Z\subseteq Y$，有 $X\to Z$

4. 关于多值依赖的另一种定义是：

给定一个关系模式 $R(X, Y, Z)$，其中 X，Y，Z 可以是属性或属性组合。

设 $x\in X$，$y\in Y$，$z\in Z$，xz 在 R 中的像集为：

$$Y_{xz}=\{r.Y \mid r.X=x\wedge r.Z=z\wedge r\in R\}$$

定义 $R(X, Y, Z)$ 当且仅当 $Y_{xz}=Y_{xz'}$ 对于每一组 (x, z, z') 都成立，则 Y 对 X 多值依赖，记作 $X\to\to Y$。这里，允许 Z 为空集，在 Z 为空集时，称为平凡的多值依赖。

请证明这里的定义和 5.2.7 节中定义 5.9 是等价的。

5. 试举出三个多值依赖的实例。

*6. 试证明给出的关于 FD 和 MVD 公理系统的 A_4，A_6 和 A_8。

*7. 设关系模式为 $R(U, F)$，X，Y 为属性集，X，$Y\subseteq U$。

证明：（1）$X\subseteq X_F^+$

（2）$(X_F^+)_F^+=X_F^+$

（3）若 $X\subseteq Y$ 则 $X_F^+\subseteq Y_F^+$

（4）$U_F^+=U$

*8. 设关系模式为 $R(U, F)$，若 $X_F^+=X$，则称 X 相对于 F 是饱和的。定义饱和集 $\varphi_F=\{X \mid X=X_F^+\}$，试证明 $\varphi_F=\{X_F^+ \mid X\subseteq U\}$。

*9. 图 5.12 表示一个公司各部门的层次结构。

对每个部门，数据库中包含部门号（唯一的）D#、预算费（BUGET）以及此部门领导人员的职工号 E#（唯一的）信息。

对每一个部门，还存有关于此部门的全部职工、生产与科研项目以及办公室的信息。

职工信息包括：职工号、他所参加的生产与科研项目号（J#）、他所在办公室的电话号码（PHONE#）。

生产科研项目包含：项目号（唯一的）、预算费。

办公室信息包含办公室房间号（唯一的）、面积。

对每个职工，数据库中有他曾担任过的职务以及担任某一职务时的工资历史。

对每个办公室包含此办公室中全部电话号码的信息。

请给出你认为合理的数据依赖，把这个层次结构转换成一组规范化的关系。

提示：此题可分步完成，第 1 步先转换成一组 1NF 的关系，然后逐步转换为 2NF，3NF，…。

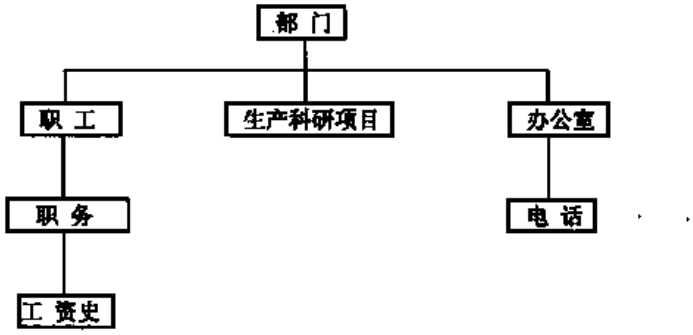


图 5.12

*10. 在一个订货系统的数据库中，存有顾客、货物和订货单的信息。

每个顾客包含顾客号（唯一的）、收货地址（一个顾客可有几个地址）、赊购限额、余额以及折扣。

每个订货单包含顾客号、收货地址、订货日期、订货细则（每个订货单有若干条）、每条订货细则内容为货物号以及订货数量。

每种货物包含货物号（唯一的）、制造厂商、每个厂商的实际存货量、规定的最低存货量和货物描述。

由于处理上的要求，每个订货单的每一订货细则中还应有一个未发货量（此值初始时为订货数量，随着发货将减为零）。

为这些数据设计一个数据库，如前题那样，给出认为合理的数据依赖。

*11. 设在第 10 题中实际上只有很少量的顾客（例如 1 %）有多个发货地址，由于这些少数的而又不能忽视的情形使得不能按一般的方式来处理问题。你能发现第 10 题答案中的问题吗？能设法改进吗？

12. 下面的结论哪些是正确的？哪些是错误的？对于错误的请给出一个反例说明之。

（1）任何一个二目关系是属于 3NF 的。

（2）任何一个二目关系是属于 BCNF 的。

（3）任何一个二目关系是属于 4NF 的。

（4）当且仅当函数依赖 $A\rightarrow B$ 在 R 上成立，关系 R（A，B，C）等于其投影 R_1（A，B）和 R_2（A，C）的连接。

（5）若 $R.A\rightarrow R.B$，$R.B\rightarrow R.C$，则 $R.A\rightarrow R.C$

(6) 若 $R.A \rightarrow R.B$，$R.A \rightarrow R.C$，则 $R.A \rightarrow R.(B, C)$

(7) 若 $R.B \rightarrow R.A$，$R.C \rightarrow R.A$，则 $R.(B, C) \rightarrow R.A$

(8) 若 $R.(B, C) \rightarrow R.A$，则 $R.B \rightarrow R.A$，$R.C \rightarrow R.A$

本章参考文献

1 Codd E F. Normalized Data Base Structure: A Brief Tutorial. Proc. of ACM SIGFIDET Workshop on Data Description, Access and Control. November 1971

2 Codd E F. Further Normalized Data Base Relational Model. In Data Base Systems, Courant Computer Science Symposia Series, Vol.6, Englewood Ciffs, N.J.Prentice-Hall, 1972

3 Armstrong W W. Dependency Structures of Data Base Relationships. Proceedings of the IFIP Congress, 1974

(Armstrong 在本文中提出了函数依赖公理系统，并给出了正确性和完备性证明。)

4 Bernstein P. Synthesizing Third Normal Form Relations from Functional Dependencies. ACM TODS, 1:4, December 1976

(本文给出了关系数据库设计的综合算法(达到第三范式的综合算法)。)

5 Zaniolo C. Analysis and Design of Relational Schemata for Database Systems. Ph.D. Thesis, University of California, Los Angeles, 1976

(Zaniolo 在他的博士论文中提出了多值依赖的概念。)

6 Fagin R. Multivalued Dependencies and a New Normal Form for Relational Database. ACM TODS, 2:3, September 1977

(Fagin 在本文中讨论并定义了第四范式。)

7 Beeri C, Fagin R, Howard J H. A Complete Axiomatization for Functional and Multivalued Dependencies. in Proceedings of ACM SIGMOD, 1977

(本文研究了函数依赖和多值依赖的公理系统及其正确性和完备性。)

8 Bernstein Philip A. Goodman Nathan. What Does Boyce-Codd Normal Form Do. in Proceedings of VLDB, 1980

9 Aho A, Beeri C, Ullman J. The Theory of Joins in Relational Databases. ACM TODS, 4:3, September 1979

(本文分析了无损连接性的性质。)

10 Delobel C. An Overview of the Relational Data Theory. IFIP, 1980

11 Maier D. The Theory of Relational Databases. Rockville, Md.: Computer Science Press, 1983

(本文献中 Maier 对关系依赖理论进行了综合讨论，给出了许多有关函数依赖的定理及其证明，介绍了求与给定函数依赖集等价的最小函数依赖集的方法等。)

12 Kent W. Consequences of Assuming a Universal Relation. ACM TODS 6, No.4, December

1981

13 Ullman J D. On Kent's Consequences of Assuming a Universal Relation. ACM TODS 8, No.4, December 1983

第二篇　设　计　篇

第六章　数据库设计

6.1　数据库设计概述

人们在总结信息资源开发、管理和服务的各种手段时，认为最有效的是数据库技术。数据库的应用已越来越广泛。从小型的单项事务处理系统到大型复杂的信息系统大都用先进的数据库技术来保持系统数据的整体性、完整性和共享性。目前，一个国家的数据库建设规模（指数据库的个数、种类）、数据库信息量的大小和使用频度已成为衡量这个国家信息化程度的重要标志之一。

数据库设计是建立数据库及其应用系统的技术，是信息系统开发和建设中的核心技术，具体说，数据库设计是指对于一个给定的应用环境，构造最优的数据库模式，建立数据库及其应用系统，使之能够有效地存储数据，满足各种用户的应用需求（信息要求和处理要求）。这个问题是数据库在应用领域的主要研究课题。

在数据库领域内，常常把使用数据库的各类系统统称为数据库应用系统。

6.1.1　数据库和信息系统

从使用者角度看，信息系统是提供信息、辅助人们对环境进行控制和进行决策的系统。数据库是信息系统的核心和基础。它把信息系统中大量的数据按一定的模型组织起来，提供存储、维护、检索数据的功能，使信息系统可以方便、及时、准确地从数据库中获得所需的信息。一个信息系统的各个部分能否紧密地结合在一起以及如何结合，关键在数据库。因此只有对数据库进行合理的逻辑设计和有效的物理设计才能开发出完善而高效的信息系统。数据库设计是信息系统开发和建设的重要组成部分。

大型数据库的设计和开发是一项庞大的工程，是涉及多学科的综合性技术。其开发周期长、耗资多、失败的风险也大。必须把软件工程的原理和方法应用到数据库建设中来。对于从事数据库设计的专业人员来讲，应该具备多方面的技术和知识。主要有：

- 数据库的基本知识和数据库设计技术；

- 计算机科学的基础知识和程序设计的方法和技巧；
- 软件工程的原理和方法；
- 应用领域的知识。

其中应用领域的知识随着应用系统所属的领域不同而不同。数据库设计人员必须深入实际与用户密切结合，对应用环境、专业业务有具体深入的了解才能设计出符合具体领域要求的数据库应用系统。

6.1.2 数据库设计的特点

数据库设计既是一项涉及多学科的综合性技术，又是一项庞大的工程项目。有人讲“三分技术，七分管理，十二分基础数据”是数据库建设的基本规律，这是有一定道理的。技术与管理的界面（称之为“干件”）十分重要。数据库建设是硬件、软件和干件的结合。这是数据库设计的特点之一。这里着重讨论软件设计的技术。

数据库设计应该和应用系统设计相结合，也就是说，整个设计过程中要把结构（数据）设计和行为（处理）设计密切结合起来。这是数据库设计的特点之二。

传统的软件工程忽视对应用中数据语义的分析和抽象。例如结构化设计（Structure Design，简称 SD 方法）和逐步求精的方法着重于处理过程的特性，只要有可能就尽量推迟数据结构设计的决策。这种方法显然对于数据库应用系统是不妥的。数据库模式是各应用程序共享的结构，是稳定的，永久的，不像以文件系统为基础的应用系统，文件是某一应用程序私用的。数据库设计质量的好坏直接影响系统中各个处理过程的性能和质量。

早期的数据库设计致力于数据模型和建模方法研究，着重结构特性的设计而忽视了对行为的设计。也就是说比较重视在给定的应用环境下，采用什么原则、方法来建造数据库的结构，而没有考虑应用环境要求与数据库结构的关系，因此结构设计与行为设计是分离的（如图 6.1 所示）。

在数据库设计中如何把结构特性和行为特性相结合，许多学者和专家进行了探讨和实践。

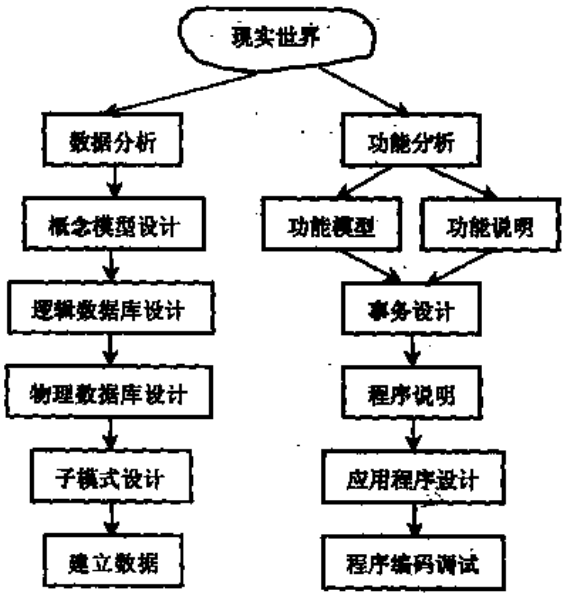


图 6.1 结构和行为分离的设计

6.1.3 数据库设计方法简述

由于信息结构复杂，应用环境多样，在相当长的一段时期内数据库设计主要采用手工试凑法。使用这种方法与设计人员的经验和水平有直接关系，数据库设计成为一种技艺而不是工程技术，缺乏科学理论和工程方法的支持，工程的质量难以保证，常常是数据库运行一段时间后又不同程度地发现各种问题，增加了系统维护的代价。十余年来，人们努力探索，提出了各种数据库设计方法，这些方法运用软件工程的思想和方法，提出了各种设计准则和规程，都属于规范设计法。

规范设计法中比较著名的有新奥尔良（New Orleans）方法。它将数据库设计分为四个阶段：需求分析（分析用户要求）、概念设计（信息分析和定义）、逻辑设计（设计实现）和物理设计（物理数据库设计）。其后，S.B.Yao 等又将数据库设计分为五个步骤。又有 I.R.Palmer 等主张把数据库设计当成一步接一步的过程，并采用一些辅助手段实现每一过程。

基于 E-R 模型的数据库设计方法，基于 3NF（第三范式）的设计方法，基于抽象语法规范的设计方法等，是在数据库设计的不同阶段上支持实现的具体技术和方法。

规范设计法从本质上看仍然是手工设计方法，其基本思想是过程迭代和逐步求精。

数据库工作者和数据库厂商一直在研究和开发数据库设计工具。经过十多年的努力，数据库设计工具已经实用化和产品化。例如 Design 2000 和 PowerDesigner 分别是 ORACLE 公司和 SYBASE 公司推出的数据库设计工具软件。这些工具软件可以自动地或辅助设计人员完成数据库设计过程中的很多任务。人们已经越来越认识到自动数据库设计工具的重要性。特别是大型数据库的设计需要自动设计工具的支持。人们也日益认识到数据库设计和应用设计应该同时进行，目前许多计算机辅助软件工程（Computer Aided Sofeware Engineering，简称 CASE）工具已经开始强调这两个方面。

6.1.4 数据库设计的基本步骤

按照规范设计的方法，考虑数据库及其应用系统开发全过程，将数据库设计分为以下六个阶段(如图 6.2 所示)：

- 需求分析；
- 概念结构设计；
- 逻辑结构设计；

- 物理结构设计；
- 数据库实施；
- 数据库运行和维护。

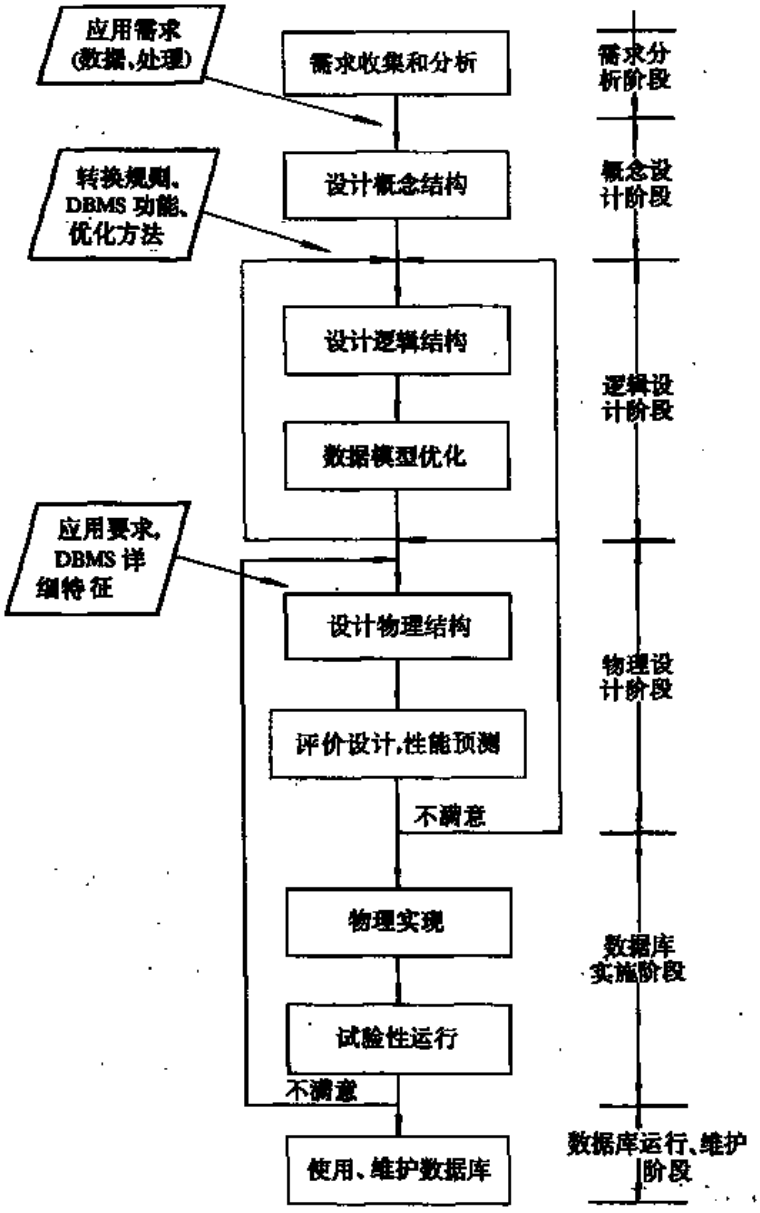


图 6.2 数据库设计步骤

数据库设计开始之前，首先必须选定参加设计的人员，包括系统分析人员、数据库设计人员和程序员、用户和数据库管理员。系统分析和数据库设计人员是数据库设计的核心人员，他们将自始至终参与数据库设计，他们的水平决定了数据库系统的质量。用户和数据库管理员在数据库设计中也是举足轻重的，

他们主要参加需求分析和数据库的运行维护，他们的积极参与不但能加速数据库设计，而且也是决定数据库设计的质量的重要因素。程序员则在系统实施阶段参与进来，分别负责编制程序和准备软硬件环境。

如果所设计的数据库应用系统比较复杂，还应该考虑是否需要使用数据库设计工具和 CASE 工具以提高数据库设计质量并减少设计工作量，以及选用何种工具。

1. 需求分析阶段

进行数据库设计首先必须准确了解与分析用户需求（包括数据与处理）。需求分析是整个设计过程的基础，是最困难、最耗费时间的一步。作为地基的需求分析是否做得充分与准确，决定了在其上构建数据库大厦的速度与质量。需求分析做得不好，甚至会导致整个数据库设计返工重做。

2. 概念结构设计阶段

概念结构设计是整个数据库设计的关键，它通过对用户需求进行综合、归纳与抽象，形成一个独立于具体 DBMS 的概念模型。

3. 逻辑结构设计阶段

逻辑结构设计是将概念结构转换为某个 DBMS 所支持的数据模型，并对其进行优化。

4. 数据库物理设计阶段

数据库物理设计是为逻辑数据模型选取一个最适合应用环境的物理结构（包括存储结构和存取方法）。

5. 数据库实施阶段

在数据库实施阶段，设计人员运用 DBMS 提供的数据语言及其宿主语言，根据逻辑设计和物理设计的结果建立数据库，编制与调试应用程序，组织数据入库，并进行试运行。

6. 数据库运行和维护阶段

数据库应用系统经过试运行后即可投入正式运行。在数据库系统运行过程中必须不断地对其进行评价、调整与修改。

设计一个完善的数据库应用系统是不可能一蹴而就的，它往往是上述六个阶段的不断反复。

需要指出的是，这个设计步骤既是数据库设计的过程，也包括了数据库应用系统的设计过程。在设计过程中把数据库的设计和对数据库中数据处理的设计紧密结合起来，将这两个方面的需求分析、抽象、设计、实现在各个阶段同时进行，相互参照，相互补充，以完善两方面的设计。事实上，如果不了解应用环境对数据的处理要求，或没有考虑如何去实现这些处理要求，是不可能设计一个良好的数据库结构的。按照这个原则，设计过程各个阶段的设计描述，

可用图 6.3 概括地给出。

图 6.3 的有关处理特性的设计描述中，其设计原理、采用的设计方法、工具等在软件工程和信息系统设计的课程中有详细介绍，这里不再讨论。这里主要讨论关于数据特性的描述以及如何在整个设计过程中参照处理特性的设计来完善数据模型设计等问题。

设计阶段	设计描述	
	数据	处理
需求分析	数据字典、全系统中数据项、数据流、数据存储的描述	数据流图和判定表（判定树）、数据字典中处理过程的描述
概念结构设计	概念模型（E-R 图） 数据字典	系统说明书包括： ①新系统要求、方案和概图 ②反映新系统信息流的数据流图
逻辑结构设计	某种数据模型 关系　非关系	系统结构图（模块结构）
物理设计	存储安排 方法选择 存取路径建立 分区 1 分区 2	模块设计 IPO 表 IPO 表…… 输入： 输出： 处理：
实施阶段	编写模式 装入数据 数据库试运行 Creat…… Load……	程序编码、编译联结、测试 Main( ) …… if…… then …… end
运行维护	性能监测、转储/恢复 数据库重组和重构	新旧系统转换、运行、维护（修正性、适应性、改善性维护）

图 6.3 数据库结构设计阶段

按照这样的设计过程，数据库结构设计的不同阶段形成数据库的各级模式，如图 6.4 所示。需求分析阶段，综合各个用户的应用需求；在概念设计阶段形成独立于机器特点，独立于各个 DBMS 产品的概念模式，在本篇中就是 E-R 图；在逻辑设计阶段将 E-R 图转换成具体的数据库产品支持的数据模型，

如关系模型，形成数据库逻辑模式；然后根据用户处理的要求、安全性的考虑，在基本表的基础上再建立必要的视图(View)，形成数据的外模式；在物理设计阶段，根据 DBMS 特点和处理的需要，进行物理存储安排，建立索引，形成数据库内模式。

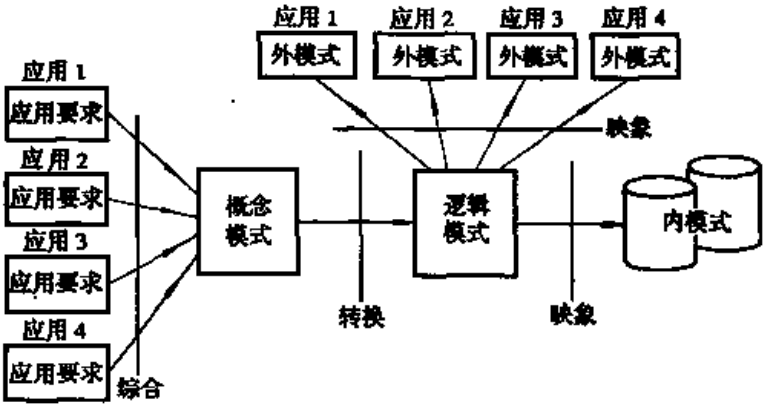


图 6.4 数据库的各级模式

下面就以图 6.2 的设计过程为主线，讨论数据库设计各个阶段的设计内容、设计方法和工具。

6.2 需 求 分 析

需求分析简单地说就是分析用户的要求。需求分析是设计数据库的起点，需求分析的结果是否准确地反映了用户的实际要求，将直接影响到后面各个阶段的设计，并影响到设计结果是否合理和实用。

6.2.1 需求分析的任务

需求分析的任务是通过详细调查现实世界要处理的对象（组织、部门、企业等），充分了解原系统（手工系统或计算机系统）工作概况，明确用户的各种需求，然后在此基础上确定新系统的功能。新系统必须充分考虑今后可能的扩充和改变，不能仅仅按当前应用需求来设计数据库。

调查的重点是“数据”和“处理”，通过调查、收集与分析，获得用户对数据库的如下要求：

（1）信息要求。指用户需要从数据库中获得信息的内容与性质。由信息要求可以导出数据要求，即在数据库中需要存储哪些数据。

（2）处理要求。指用户要完成什么处理功能，对处理的响应时间有什么

要求，处理方式是批处理还是联机处理。

（3）安全性与完整性要求。

确定用户的最终需求是一件很困难的事，这是因为一方面用户缺少计算机知识，开始时无法确定计算机究竟能为自己做什么，不能做什么，因此往往不能准确地表达自己的需求，所提出的需求往往不断地变化。另一方面，设计人员缺少用户的专业知识，不易理解用户的真正需求，甚至误解用户的需求。因此设计人员必须不断深入地与用户交流，才能逐步确定用户的实际需求。

6.2.2 需求分析的方法

进行需求分析首先是调查清楚用户的实际要求，与用户达成共识，然后分析与表达这些需求。

调查用户需求的具体步骤是：

（1）调查组织机构情况。包括了解该组织的部门组成情况、各部门的职责等，为分析信息流程做准备。

（2）调查各部门的业务活动情况。包括了解各个部门输入和使用什么数据，如何加工处理这些数据，输出什么信息，输出到什么部门，输出结果的格式是什么，这是调查的重点。

（3）在熟悉了业务活动的基础上，协助用户明确对新系统的各种要求，包括信息要求、处理要求、完全性与完整性要求，这是调查的又一个重点。

（4）确定新系统的边界。对前面调查的结果进行初步分析，确定哪些功能由计算机完成或将来准备让计算机完成，哪些活动由人工完成。由计算机完成的功能就是新系统应该实现的功能。

在调查过程中，可以根据不同的问题和条件，使用不同的调查方法。常用的调查方法有：

（1）跟班作业。通过亲身参加业务工作来了解业务活动的情况。这种方法可以比较准确地理解用户的需求，但比较耗费时间。

（2）开调查会。通过与用户座谈来了解业务活动情况及用户需求。座谈时，参加者之间可以相互启发。

（3）请专人介绍。

（4）询问。对某些调查中的问题，可以找专人询问。

（5）设计调查表请用户填写。如果调查表设计得合理，这种方法是很有效，也易于为用户接受。

（6）查阅记录。查阅与原系统有关的数据记录。

做需求调查时，往往需要同时采用上述多种方法。但无论使用何种调查方法，都必须有用户的积极参与和配合。

调查了解了用户的需求以后，还需要进一步分析和表达用户的需求。在众多的分析方法中结构化分析方法（Structured Analysis，简称 SA 方法）是一种简单实用的方法。SA 方法从最上层的系统组织机构入手，采用自顶向下、逐层分解的方式分析系统。SA 方法把任何一个系统都抽象为图 6.5 的形式。

图 6.5 给出的只是最高层次抽象的系统概貌，要反映更详细的内容，可将处理功能分解为若干子功能，每个子功能还可以继续分解，直到把系统工作过程表示清楚为止。在处理功能逐步分解的同时，它们所用的数据也逐级分解，形成若干层次的数据流图。

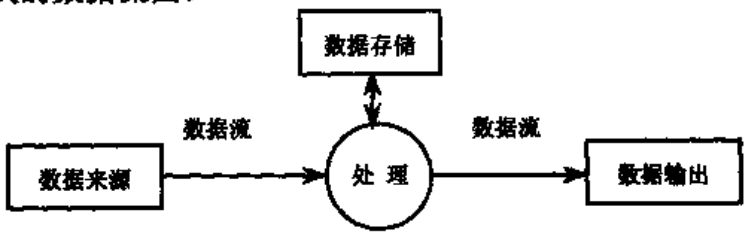


图 6.5 系统高层抽象图

数据流图表达了数据和处理过程的关系。在 SA 方法中，处理过程的处理逻辑常常借助判定表或判定树来描述。系统中的数据则借助数据字典（Data Dictionary，简称 DD）来描述。

对用户需求进行分析与表达后，必须提交给用户，征得用户的认可。

图 6.6 描述了需求分析的过程。

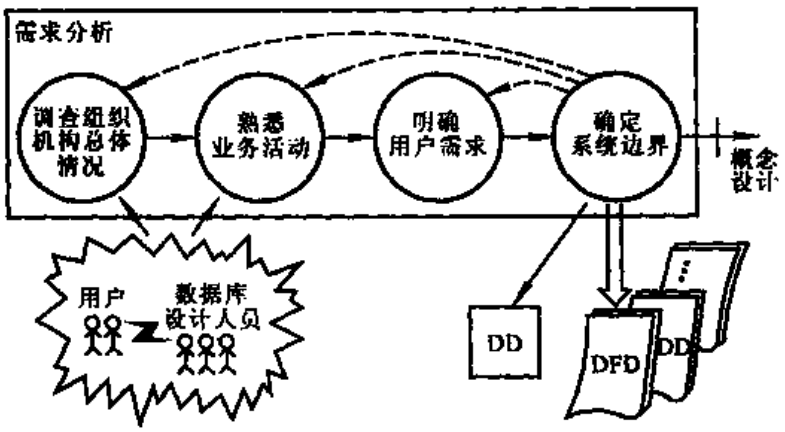


图 6.6 需求分析过程

6.2.3 数据字典

数据流图表达了数据和处理的关系，数据字典则是系统中各类数据描述的集合，是进行详细的数据收集和数据分析所获得的主要成果。数据字典在数据库设计中占有很重要的地位。

数据字典通常包括数据项、数据结构、数据流、数据存储和处理过程五个部

分。其中数据项是数据的最小组成单位，若干个数据项可以组成一个数据结构，数据字典通过对数据项和数据结构的定义来描述数据流、数据存储的逻辑内容。

1. 数据项

数据项是不可再分的数据单位。对数据项的描述通常包括以下内容：

数据项描述＝{数据项名，数据项含义说明，别名，数据类型，长度，取值范围，
取值含义，与其他数据项的逻辑关系，数据项之间的联系}

其中“取值范围”、“与其他数据项的逻辑关系”（例如该数据项等于另几个数据项的和，该数据项值等于另一数据项的值等）定义了数据的完整性约束条件，是设计数据检验功能的依据。

可以用关系规范化理论为指导，用数据依赖的概念分析和表示数据项之间的联系。即按实际语义，写出每个数据项之间的数据依赖，它们是数据库逻辑设计阶段数据模型优化的依据。

2. 数据结构

数据结构反映了数据之间的组合关系。一个数据结构可以由若干个数据项组成，也可以由若干个数据结构组成，或由若干个数据项和数据结构混合组成。对数据结构的描述通常包括以下内容：

数据结构描述＝{数据结构名，含义说明，组成：{数据项或数据结构}}

3. 数据流

数据流是数据结构在系统内传输的路径。对数据流的描述通常包括以下内容：

数据流描述＝{数据流名，说明，数据流来源，数据流去向，
组成：{数据结构}，平均流量，高峰期流量}

其中“数据流来源”是说明该数据流来自哪个过程。“数据流去向”是说明该数据流将到哪个过程去。“平均流量”是指在单位时间（每天、每周、每月等）里的传输次数。“高峰期流量”则是指在高峰时期的数据流量。

4. 数据存储

数据存储是数据结构停留或保存的地方，也是数据流的来源和去向之一。它可以是手工文档或手工凭单，也可以是计算机文档。对数据存储的描述通常包括以下内容：

数据存储描述＝{数据存储名，说明，编号，输入的数据流，输出的数据流，
组成：{数据结构}，数据量，存取频度，存取方式}

其中“存取频度”指每小时或每天或每周存取几次、每次存取多少数据等信息。“存取方式”包括是批处理还是联机处理；是检索还是更新；是顺序检索还是随机检索等。另外，“输入的数据流”要指出其来源，“输出的数据流”要指出其去向。

5. 处理过程

处理过程的具体处理逻辑一般用判定表或判定树来描述。数据字典中只需要描述处理过程的说明性信息，通常包括以下内容：

处理过程描述＝｛处理过程名，说明，输入：｛数据流｝，输出：｛数据流｝，
处理：｛简要说明｝｝

其中“简要说明”中主要说明该处理过程的功能及处理要求。功能是指该处理过程用来做什么（而不是怎么做），处理要求包括处理频度要求，如单位时间里处理多少事务、多少数据量、响应时间要求等。这些处理要求是后面物理设计的输入及性能评价的标准。

可见，数据字典是关于数据库中数据的描述，即元数据，而不是数据本身。

数据字典是在需求分析阶段建立，在数据库设计过程中不断修改、充实、完善的。

明确地把需求收集和分析作为数据库设计的第一阶段是十分重要的。这一阶段收集到的基础数据（用数据字典来表达）和一组数据流程图（Data Flow Diagram，简称 DFD）是下一步进行概念设计的基础。

最后，要强调两点：

（1）需求分析阶段的一个重要而困难的任务是收集将来应用所涉及的数据，设计人员应充分考虑到可能的扩充和改变，使设计易于更改，系统易于扩充，这是第一点。

（2）必须强调用户的参与，这是数据库应用系统设计的特点。数据库应用系统和广泛的用户有密切的联系，许多人要使用数据库。数据库的设计和建立又可能对更多人的工作环境产生重要影响。因此用户的参与是数据库设计不可分割的一部分。在数据分析阶段，任何调查研究没有用户的积极参加是寸步难行的。设计人员应该和用户取得共同的语言，帮助不熟悉计算机的用户建立数据库环境下的共同概念，并对设计工作的最后结果承担共同的责任。

6.3 概念结构设计

将需求分析得到的用户需求抽象为信息结构即概念模型的过程就是概念结构设计。它是整个数据库设计的关键。

6.3.1 概念结构

在需求分析阶段所得到的应用需求应该首先抽象为信息世界的结构，才能更好地、更准确地用某一 DBMS 实现这些需求。

概念结构的主要特点是：

（1）能真实、充分地反映现实世界，包括事物和事物之间的联系，能满足用户对数据的处理要求。是对现实世界的一个真实模型。

（2）易于理解，从而可以用它和不熟悉计算机的用户交换意见，用户的积极参与是数据库的设计成功的关键。

（3）易于更改，当应用环境和应用要求改变时，容易对概念模型修改和扩充。

（4）易于向关系、网状、层次等各种数据模型转换。

概念结构是各种数据模型的共同基础，它比数据模型更独立于机器、更抽象，从而更加稳定。

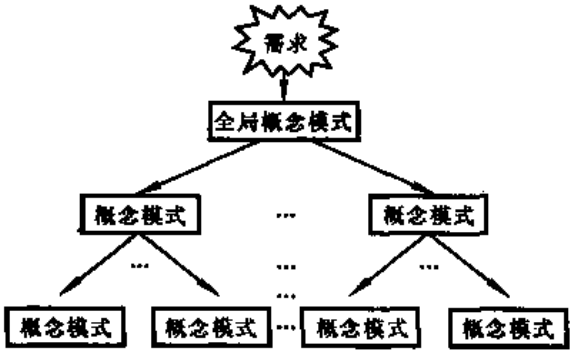
描述概念模型的有力工具是 E-R 模型。有关 E-R 模型的基本概念已在第一章介绍。下面将用 E-R 模型来描述概念结构。

6.3.2 概念结构设计的方法与步骤

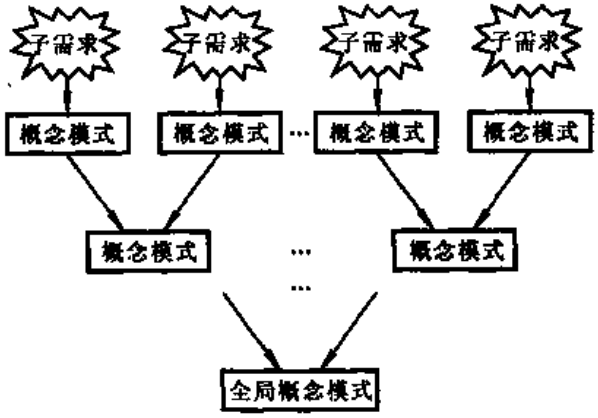
设计概念结构通常有四类方法：

- 自顶向下。即首先定义全局概念结构的框架，然后逐步细化，如图 6.7(a)所示。
- 自底向上。即首先定义各局部应用的概念结构，然后将它们集成起来，得到全局概念结构，如图 6.7(b)所示。
- 逐步扩张。首先定义最重要的核心概念结构，然后向外扩充，以滚雪球的方式逐步生成其他概念结构，直至总体概念结构，如图 6.7(c)所示。
- 混合策略。即将自顶向下和自底向上相结合，用自顶向下策略设计一个全局概念结构的框架，以它为骨架集成由自底向上策略中设计的各局部概念结构。

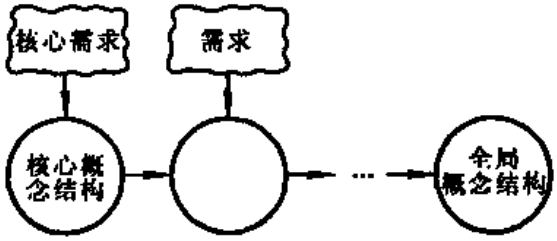
其中最经常采用的策略是自底向上方法。即自顶向下地进行需求分析，然后再自底向上地设计概念结构。如图 6.8 所示。这里只介绍自底向上设计概念结构的方法。它通常分为两步：第 1 步是抽象数据并设计局部视图，第 2 步是集成局部视图，得到全局的概念结构，如图 6.9 所示。



(a) 自顶向下策略



(b) 自底向上策略



(c) 逐步扩张策略

图 6.7 设计概念结构的策略

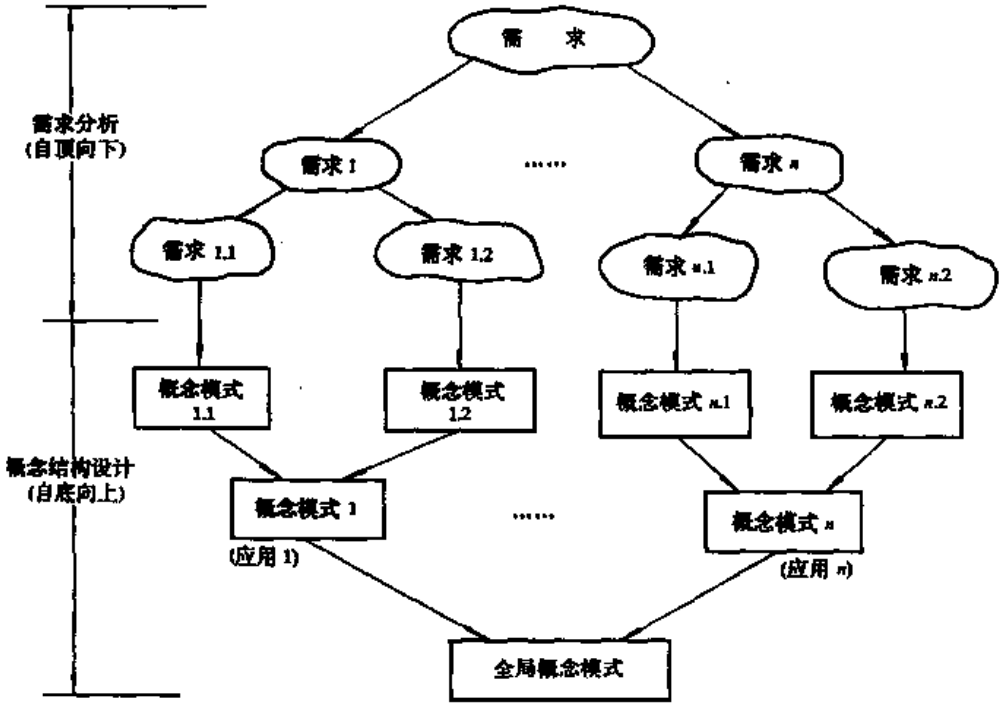


图 6.8 自顶向下分析需求与自底向上设计概念结构

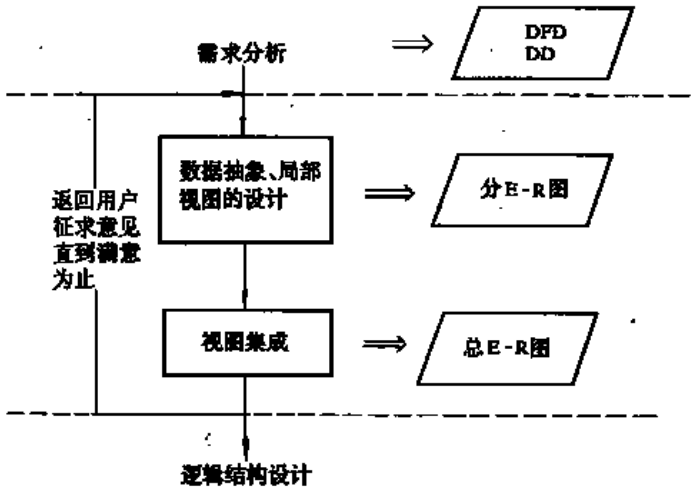


图 6.9 概念结构设计步骤

6.3.3 数据抽象与局部视图设计

概念结构是对现实世界的一种抽象。所谓抽象是对实际的人、物、事和概念进行人为处理，抽取所关心的共同特性，忽略非本质的细节，并把这些特性用各种概念精确地加以描述，这些概念组成了某种模型。

一般有三种抽象：

1. 分类（Classification）

定义某一类概念作为现实世界中一组对象的类型。这些对象具有某些共同的特性和行为。它抽象了对象值和型之间的“is member of”的语义。在 E-R 模型中，实体型就是这种抽象。例如在学校环境中，张英是学生（如图 6.10 所示），表示张英是学生中的一员（is member of 学生），具有学生们共同的特性和行为：在某个班学习某种专业，选修某些课程。

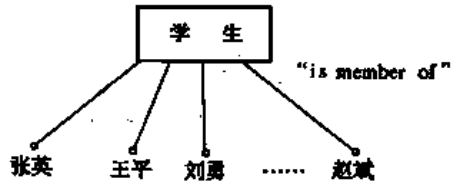


图 6.10 分类

2. 聚集（Aggregation）

定义某一类型的组成成分。它抽象了对象内部类型和成分之间“is part of”的语义。在 E-R 模型中若干属性的聚集组成了实体型，就是这种抽象，如图 6.11 所示。

更复杂的聚集如图 6.12 所示，即某一类型的成分仍是一个聚集。

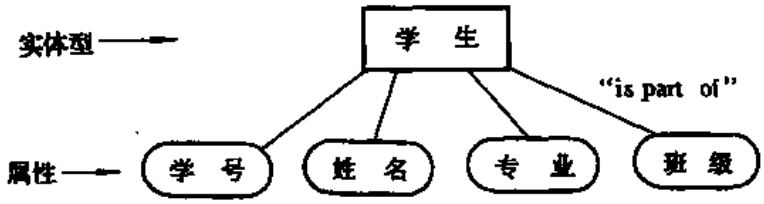


图 6.11 聚集

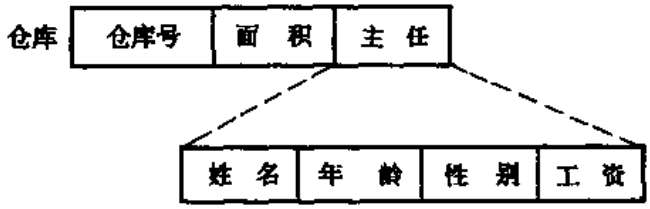


图 6.12 更复杂的聚集

3. 概括（Generalization）

定义类型之间的一种子集联系。它抽象了类型之间的“is subset of”的语义。例如学生是一个实体型，本科生、研究生也是实体型。本科生、研究生均是学生的子集。把学生称为超类（Superclass），本科生、研究生称为学生的子类（Subclass）。

原 E-R 模型不具有概括，本书对 E-R 模型作了扩充，允许定义超类实体型和子类实体型。并用双竖边的矩形框表示子类，用直线加小圆圈表示超类-子类的联系（如图 6.13 所示）。

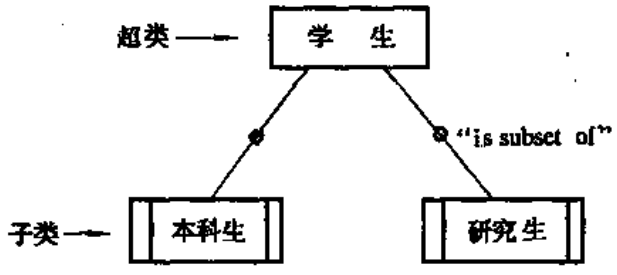


图 6.13 概括

概括有一个很重要的性质：继承性。子类继承超类上定义的所有抽象。这样，本科生、研究生继承了学生类型的属性。当然，子类可以增加自己的某些特殊属性。

概念结构设计的第一步就是利用上面介绍的抽象机制对需求分析阶段收集到的数据进行分类、组织（聚集），形成实体、实体的属性，标识实体的码，确定实体之间的联系类型(1∶1，1∶n，m∶n)，设计分 E-R 图。具体做法是：

1. 选择局部应用

根据某个系统的具体情况，在多层的数据流图中选择一个适当层次的数据流图，作为设计分 E-R 图的出发点。让这组图中每一部分对应一个局部应用。

由于高层的数据流图只能反映系统的概貌，而中层的数据流图能较好地反映系统中各局部应用的子系统组成，因此人们往往以中层数据流图作为设计分 E-R 图的依据(如图 6.14 所示)。

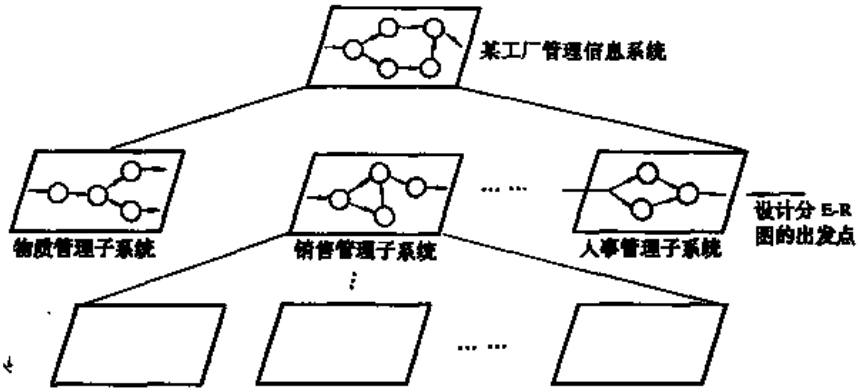


图 6.14 设计分 E-R 图的出发点

2. 逐一设计分 E-R 图

选择好局部应用之后，就要对每个局部应用逐一设计分 E-R 图，亦称局部 E-R 图。

在前面选好的某一层次的数据流图中，每个局部应用都对应了一组数据流图，局部应用涉及的数据都已经收集在数据字典中了。现在就是要将这些数据从数据字典中抽取出来，参照数据流图，标定局部应用中的实体、实体的属性、标识实体的码，确定实体之间的联系及其类型。

事实上，在现实世界中具体的应用环境常常对实体和属性已经作了大体的自然的划分。在数据字典中，“数据结构”、“数据流”和“数据存储”都是若干属性有意义的聚合，就体现了这种划分。可以先从这些内容出发定义 E-R 图，然后再进行必要的调整。在调整中遵循的一条原则是：

为了简化 E-R 图的处置，现实世界的事物能作为属性对待的，尽量作为属性对待。

那么符合什么条件的事物可以作为属性对待呢？本来，实体与属性之间并没有形式上可以截然划分的界限，但可以给出两条准则：

（1）作为“属性”，不能再具有需要描述的性质。“属性”必须是不可分的数据项，不能包含其他属性。

（2）“属性”不能与其他实体具有联系，即 E-R 图中所表示的联系是实体之间的联系。

凡满足上述两条准则的事物，一般均可作为属性对待。

例如：职工是一个实体，职工号、姓名、年龄是职工的属性，职称如果没有与工资、福利挂钩，换句话说，没有需要进一步描述的特性，则根据准则(1)可以作为职工实体的属性。但如果不同的职称有不同的工资、住房标准和不同的附加福利，则职称作为一个实体看待就更恰当，如图 6.15 所示。

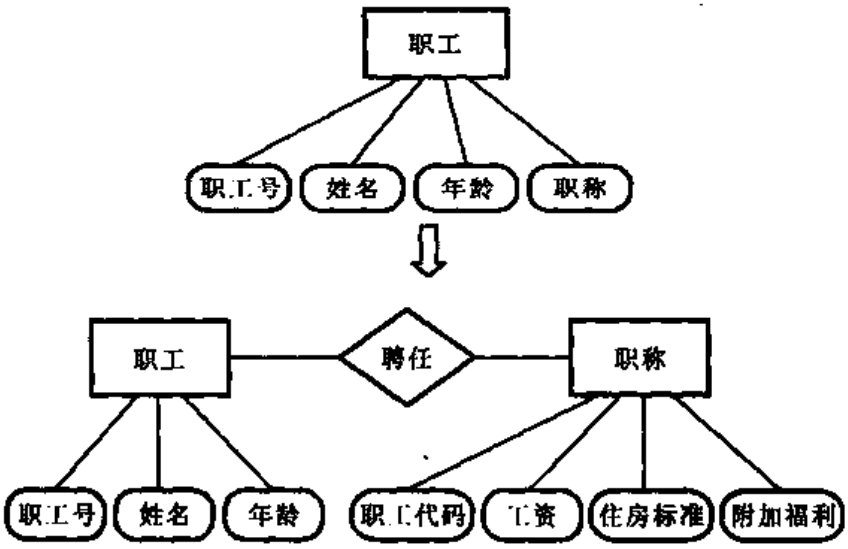


图 6.15 职称作为一个实体

再如，在医院中，一个病人只能住在一个病房，病房号可以作为病人实体的一个属性。但如果病房还要与医生实体发生联系，即一个医生负责几个病房的病人的医疗工作，则病房根据准则(2)应作为一个实体，如图 6.16 所示。

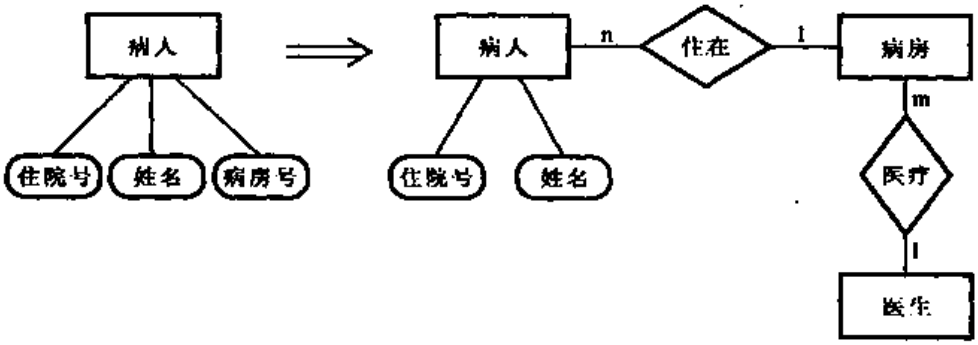


图 6.16 病房作为属性或一个实体

又如，如果一种货物只存放在一个仓库，那么就可以把存放货物的仓库的仓库号作为描述货物存放地点的属性。但如果一种货物可以存放在多个仓库中，或者仓库本身又用面积作为属性，或者与职工发生管理上的联系，那么就应把仓库作为一个实体，如图 6.17 所示。

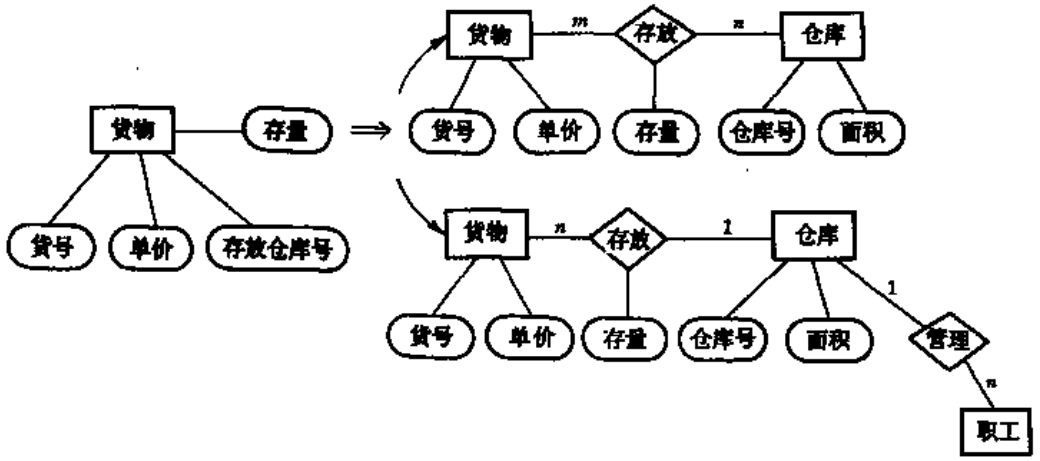


图 6.17 仓库作为一个实体

实例 销售管理子系统分 E-R 图的设计。

某工厂开发管理信息系统，经过可行性分析和详细调查，确定了该系统由物资管理、销售管理、劳动人事管理等子系统组成。为每个子系统组成了开发小组。

销售管理子系统开发小组的成员经过调查研究、信息流程分析和数据收集，明确了该子系统的主要功能是：处理顾客和销售员送来的订单；工厂是根据订货安排生产的；交出货物同时开出发票；收到顾客付款后，根据发票存根和信贷情况进行应收款处理，得到了该子系统二层数据流图（共 5 张）和数据字典，其中包括 14 个数据结构和 29 个数据流。

图 6.18 是第一层数据流图。虚线部分划出了系统边界。图中把系统功能又分为四个子系统。图 6.19 至图 6.22 是第二层数据流图。

由于该子系统不太复杂，设计分 E-R 图可以从图 6.18 第一层数据流图入手。若某一局部应用仍比较复杂，则可以从更下层的数据流图入手。例如，从图 6.19 至图 6.22 开始，分别设计它们的分 E-R 图，再汇总成该局部应用的分 E-R 图。

分析图 6.18 和数据字典，知道整个系统功能围绕了“订单”和“应收账款”的处理。数据结构中订单、顾客、顾客应收账目用得最多，是许多子功能、数据流共享的数据，因此先设计该分 E-R 图的草图（如图 6.23 所示）。

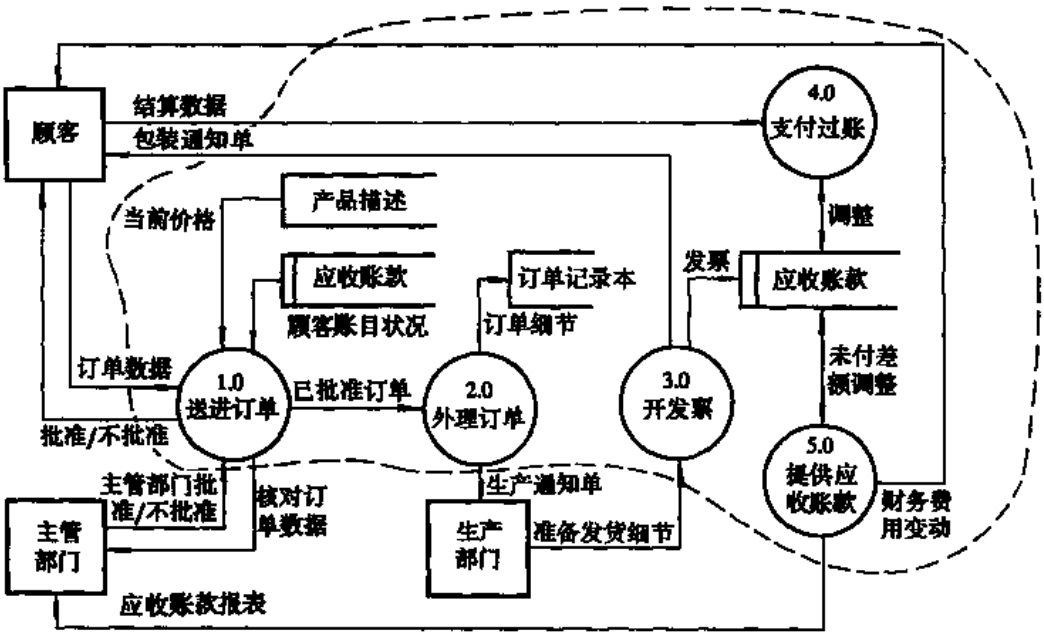


图 6.18 销售管理子系统第一层数据流图

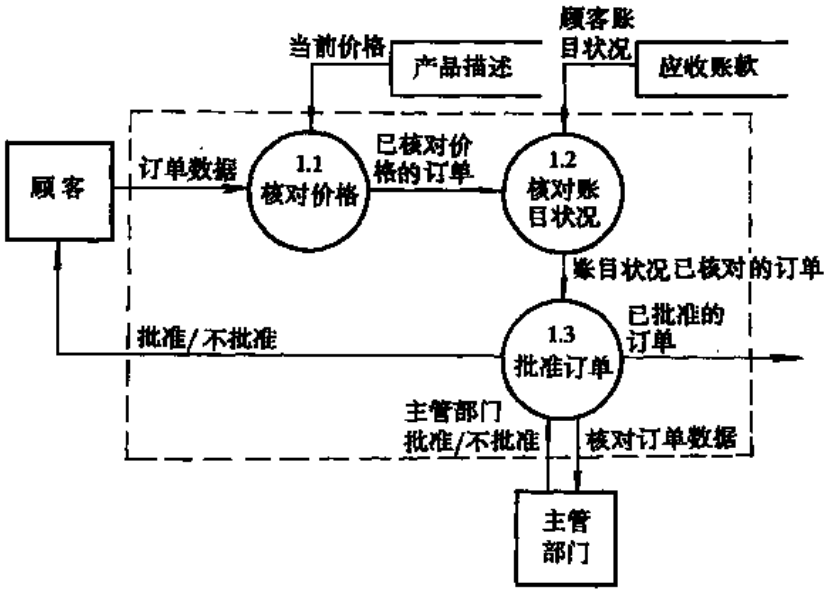


图 6.19 接收订单

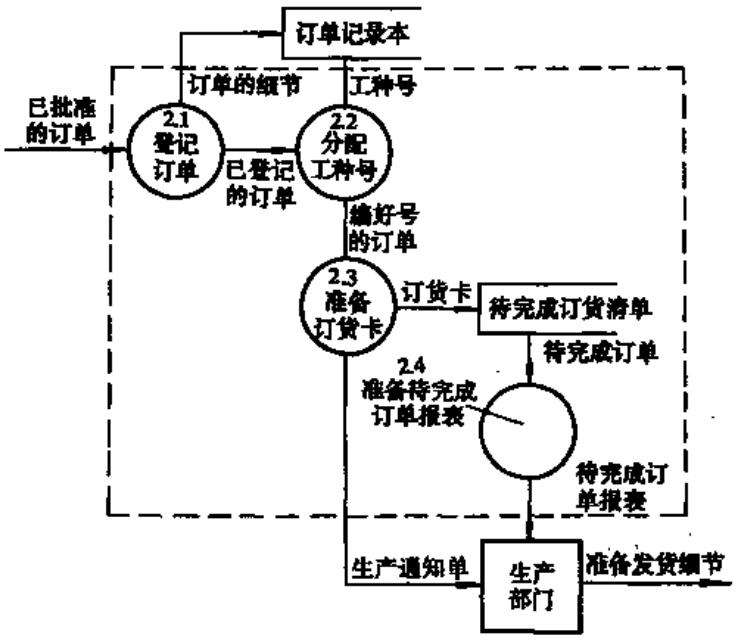


图 6.20 处理订单

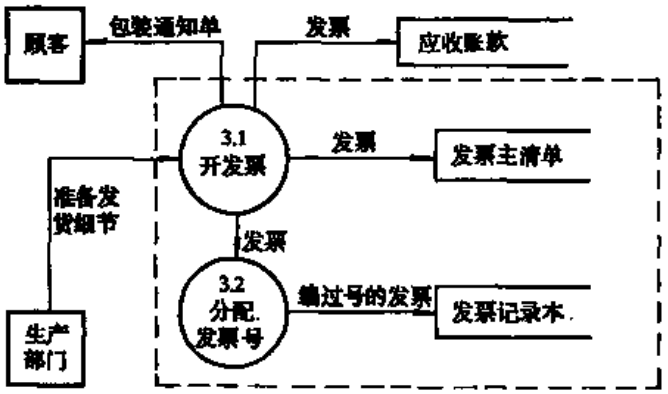


图 6.21 开发票

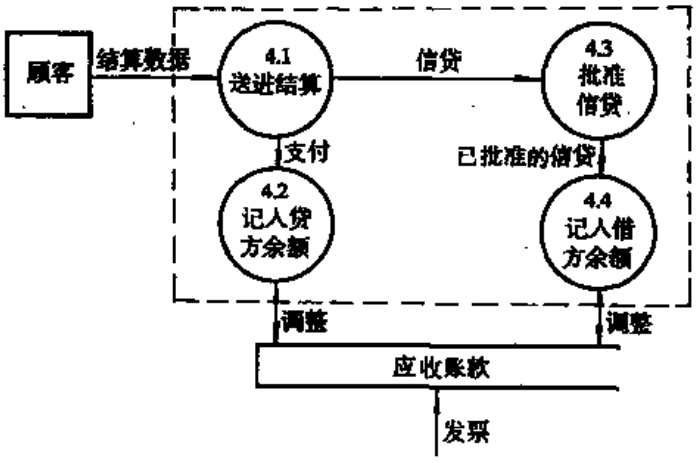


图 6.22 支付过账

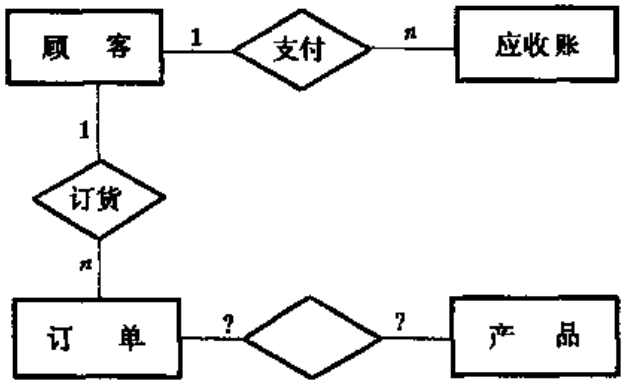


图 6.23 分 E-R 图的框架

然后参照第二层数据流图和数据字典中的详尽描述，遵循前面给出的两个准则，进行了如下调整：

（1）每张订单由订单号、若干头信息和订单细节组成。订单细节又有订货的零件号、数量等来描述。按照准则（2），订单细节就不能作订单的属性处理而应该上升为实体。一张订单可以订若干产品，所以订单与订单细节两个实体之间是 1：*n* 的联系。

（2）原订单和产品的联系实际上是订单细节和产品的联系。每条订货细节对应一个产品描述，订单处理时从中获得当前单价、产品重量等信息。

（3）图 6.21 中“发票清单”是一个数据存储，是否应作为实体加入分 E-R 图呢？答案是不必。这里的数据存储对应手工凭证，发票上的信息在开具发票的同时已及时存入应收账款中了。

（4）工厂对大宗订货给予优惠。每种产品都规定了不同订货数量的折扣，应增加一个“折扣规则”实体存放这些信息，而不应把它们放在产品描述实体中。

最后得到分 E-R 图如图 6.24 所示。

对每个实体定义的属性如下：

顾客：{顾客号，顾客名，地址，电话，信贷状况，账目余额}

订单：{订单号，顾客号，订货项数，订货日期，交货日期，工种号，生产地点}

订单细则：{订单号，细则号，零件号，订货数，金额}

应收账款：{顾客号，订单号，发票号，应收金额，支付日期，支付金额，
当前余额，货款限额}

产品描述：{产品号，产品名，单价，重量}

折扣规则：{产品号，订货量，折扣}

（注意：为了节省篇幅，实体与属性的关系没有用图形表示，实体的标识码用下横线划出）。

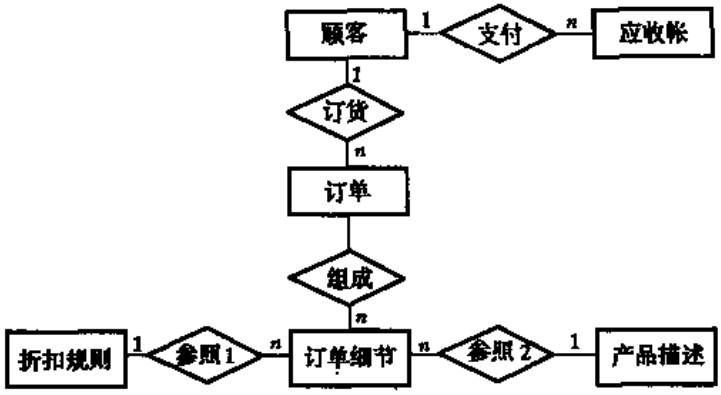


图 6.24 销售管理子系统的分 E-R 图

6.3.4 视图的集成

各子系统的分 E-R 图设计好以后，下一步就是要将所有的分 E-R 图综合成一个系统的总 E-R 图。一般说来，视图集成可以有两种方式：

- 多个分 E-R 图一次集成，如图 6.25(a)所示。
- 逐步集成，用累加的方式一次集成两个分 E-R 图，如图 6.25(b)所示。

第一种方式比较复杂，做起来难度较大。

第二种方式每次只集成两个分 E-R 图，可以降低复杂度。

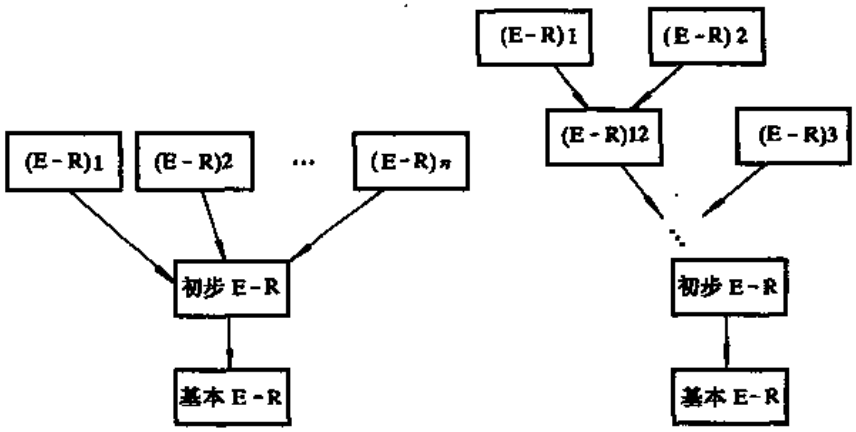


图 6.25 视图集成的两种方式

无论采用哪种方式，每次集成局部 E-R 图时都需要分两步走，如图 6.26 所示。

（1）合并。解决各分 E-R 图之间的冲突，将各分 E-R 图合并起来生成初步 E-R 图。

（2）修改和重构。消除不必要的冗余，生成基本 E-R 图。

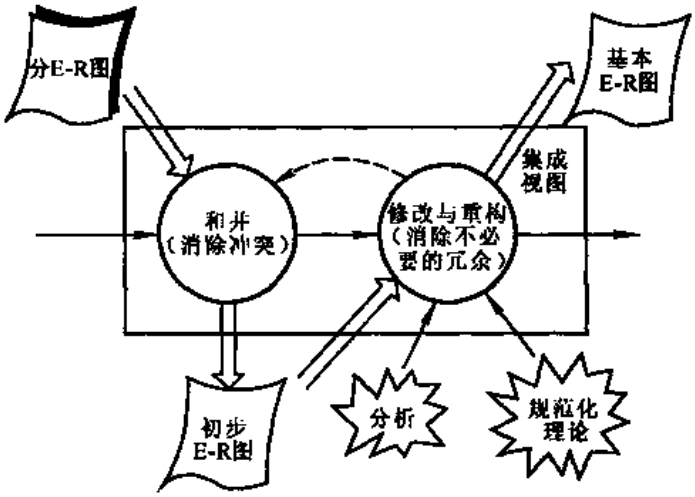


图 6.26 视图集成

一、合并分 E-R 图，生成初步 E-R 图

各个局部应用所面向的问题不同，且通常是由不同的设计人员进行局部视图设计，这就导致各个分 E-R 图之间必定会存在许多不一致的地方，称之为冲突。因此合并分 E-R 图时并不能简单地将各个分 E-R 图画到一起，而是必须着力消除各个分 E-R 图中的不一致，以形成一个能为全系统中所有用户共同理解和接受的统一的概念模型。合理消除各分 E-R 图的冲突是合并分 E-R 图的主要工作与关键所在。

各分 E-R 图之间的冲突主要有三类：属性冲突、命名冲突和结构冲突。

1. 属性冲突

（1）属性域冲突，即属性值的类型、取值范围或取值集合不同。例如零件号，有的部门把它定义为整数，有的部门把它定义为字符型。不同的部门对零件号的编码也不同。又如年龄，某些部门以出生日期形式表示职工的年龄，而另一些部门用整数表示职工的年龄。

（2）属性取值单位冲突。例如零件的重量，有的以公斤为单位，有的以斤为单位，有的以克为单位。

属性冲突理论上好解决，但实际上需要各部门讨论协商，解决起来并非易事。

2. 命名冲突

（1）同名异义，即不同意义的对象在不同的局部应用中具有相同的名字。

（2）异名同义（一义多名），即同一意义的对象在不同的局部应用中具有不同的名字。

如对科研项目，财务科称为项目，科研处称为课题，生产管理处称为工程。

命名冲突可能发生在实体、联系一级上，也可能发生在属性一级上。其中

属性的命名冲突更为常见。处理命名冲突通常也像处理属性冲突一样，通过讨论、协商等行政手段加以解决。

3. 结构冲突

（1）同一对象在不同应用中具有不同的抽象。例如职工在某一局部应用中被当作实体，而在另一局部应用中则被当作属性。

解决方法通常是把属性变换为实体或把实体变换为属性，使同一对象具有相同的抽象。但变换时仍要遵循 6.3.3 节中讲述的两个准则。

（2）同一实体在不同分 E-R 图中所包含的属性个数和属性排列次序不完全相同。

这是很常见的一类冲突，原因是不同的局部应用关心的是该实体的不同侧面。解决方法是使该实体的属性取各分 E-R 图中属性的并集，再适当调整属性的次序。

实体间的联系在不同的分 E-R 图中为不同的类型，如实体 E1 与 E2 在一个分 E-R 图中是多对多联系，在另一个分 E-R 图中是一对多联系；又如在一个分 E-R 图中 E1 与 E2 发生联系，而在另一个分 E-R 图中 E1，E2，E3 三者之间有联系。

解决方法是根据应用的语义对实体联系的类型进行综合或调整。

例如，图 6.27 中零件与产品之间存在多对多的联系——“构成”。产品、零件与供应商三者之间还存在多对多的联系——“供应”，这两个联系互相不能包含，在合并两个分 E-R 图时就应把它们综合起来。

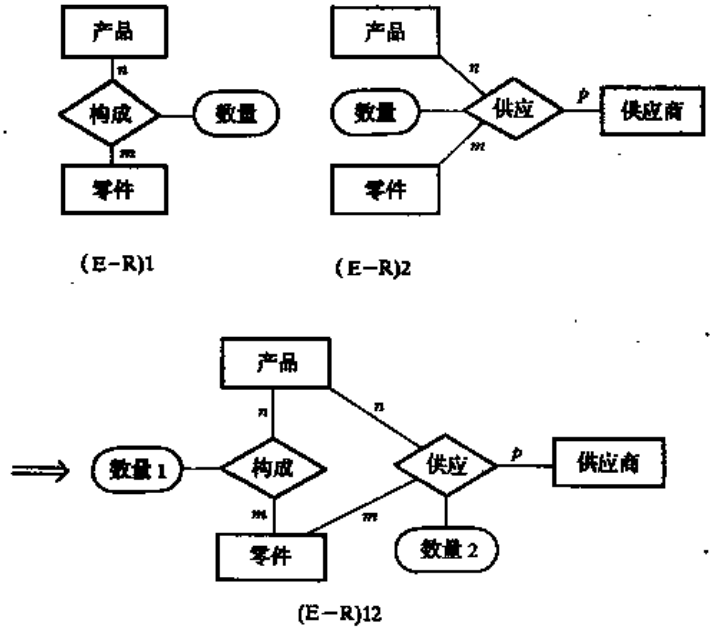


图 6.27 合并两个分 E-R 图时的综合

二、消除不必要的冗余，设计基本 E-R 图

在初步 E-R 图中，可能存在一些冗余的数据和实体间冗余的联系。所谓冗余的数据是指可由基本数据导出的数据，冗余的联系是指可由其他联系导出的联系。冗余数据和冗余联系容易破坏数据库的完整性，给数据库的维护增加困难，应当予以消除。消除了冗余后的初步 E-R 图称为基本 E-R 图。

消除冗余主要采用分析方法，即以数据字典和数据流图为依据，根据数据字典中关于数据项之间逻辑关系的说明来消除冗余。如图 6.28 中，$Q_3=Q_1\times Q_2$，$Q_4=\Sigma Q_5$。所以 Q_3 和 Q_4 是冗余数据，可以消去。并且由于 Q_3 消去，产品与材料间 m ：n 的冗余联系也应消去。

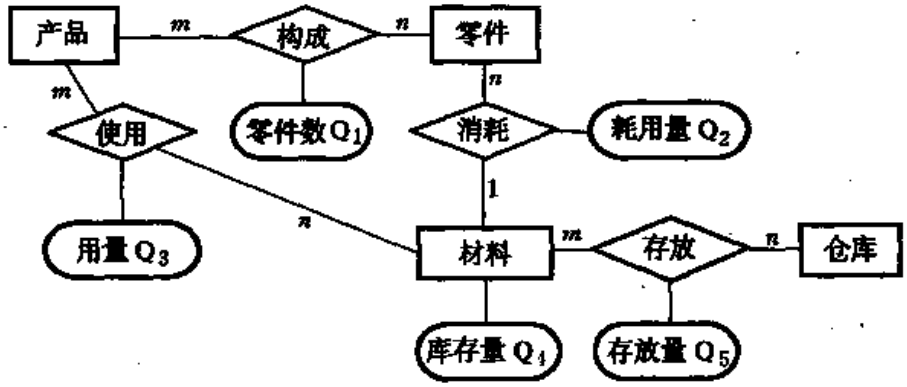


图 6.28 消除冗余

但并不是所有的冗余数据与冗余联系都必须加以消除，有时为了提高效率，不得不以冗余信息作为代价。因此在设计数据库概念结构时，哪些冗余信息必须消除，哪些冗余信息允许存在，需要根据用户的整体需求来确定。如果人为地保留了一些冗余数据，则应把数据字典中数据关联的说明作为完整性约束条件。

例如，若物种部门经常要查询各种材料的库存量，如果每次都要查询每个仓库中此种材料的库存，再对它们求和，查询效率就太低了。所以应保留 Q_4，同时把 $Q_4=\Sigma Q_5$ 定义为 Q_4 的完整性约束条件。每当 Q_5 修改后，就触发该完整性检查例程，对 Q_4 作相应的修改。

除分析方法外，还可以用规范化理论来消除冗余。在规范化理论中，函数依赖的概念提供了消除冗余联系的形式化工具。具体方法如下：

1. 确定分 E-R 图实体之间的数据依赖。实体之间一对一、一对多、多对多的联系可以用实体码之间的函数依赖来表示。如图 6.29 中：

部门和职工之间一对多的联系可表示为

职工号→部门号

职工和产品之间多对多的联系可表示为（职工号，产品号）→工作天数等。

于是有函数依赖集 F_L。

2. 求 F_L 的最小覆盖 G_L，差集为

$D = F_L - G_L$.

逐一考察 D 中的函数依赖，确定是否是冗余的联系，若是，就把它去掉。由于规范化理论受到泛关系假设的限制，应注意下面两个问题：

（1）冗余的联系一定在 D 中，而 D 中的联系不一定是冗余的；

（2）当实体之间存在多种联系时要将实体之间的联系在形式上加以区分。如图 6.30 中部门和职工之间另一个一对一的联系就要表示为：

负责人.职工号→部门号，部门号→负责人.职工号。

[实例] 某工厂管理信息系统的视图集成。

图 1.15(c)、图 6.24、图 6.29 分别为该厂物资、销售和劳动人事管理的分 E-R 图。图 6.30 为该系统的基本 E-R 图。基本 E-R 图中各实体的属性因篇幅有限这里从略。 集成过程中，解决了以下问题：

（1）异名同义，项目和产品含义相同。某个项目实质上是指某个产品的生产。统一用产品作实体名。

（2）库存管理中职工与仓库的工作关系已包含在劳动人事管理的部门与职工之间的联系之中，所以可以取消。职工之间领导与被领导关系可由部门与职工（经理）之间的领导关系、部门与职工之间的从属关系两者导出，所以也可以取消。

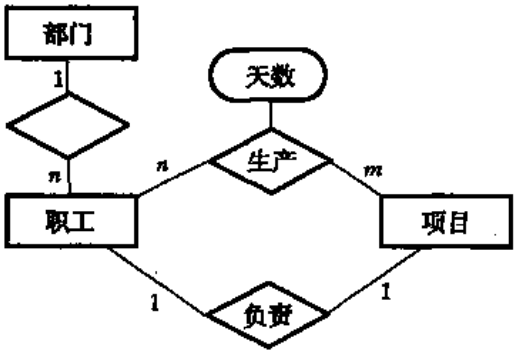


图 6.29 劳动人事管理的分 E-R 图

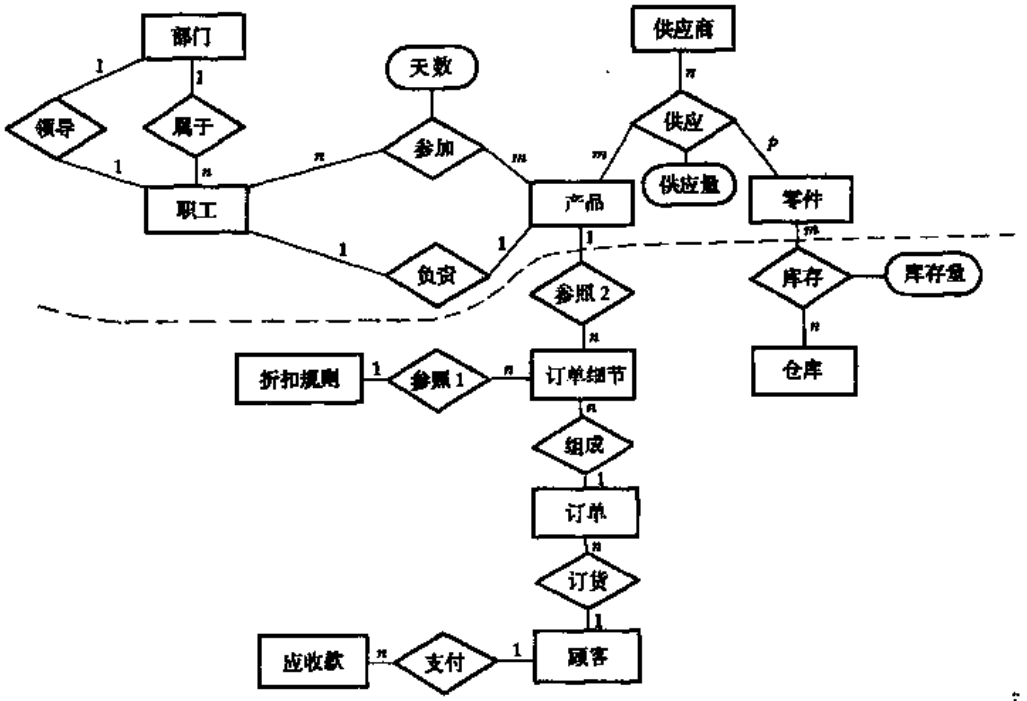


图 6.30 某工厂管理信息系统的基本 E-R 图

6.4 逻辑结构设计

概念结构是独立于任何一种数据模型的信息结构。逻辑结构设计的任务就是把概念结构设计阶段设计好的基本 E-R 图转换为与选用 DBMS 产品所支持的数据模型相符合的逻辑结构。

从理论上讲，设计逻辑结构应该选择最适于相应概念结构的数据模型，然后对支持这种数据模型的各种 DBMS 进行比较，从中选出最合适的 DBMS。但实际情况往往是已给定了某种 DBMS，设计人员没有选择的余地。目前 DBMS 产品一般支持关系、网状、层次三种模型中的某一种，对某一种数据模型，各个机器系统又有许多不同的限制，提供不同的环境与工具。所以设计逻辑结构时一般要分三步进行（如图 6.31 所示）：

（1）将概念结构转换为一般的关系、网状、层次模型；

（2）将转换来的关系、网状、层次模型向特定 DBMS 支持下的数据模型转换；

（3）对数据模型进行优化。

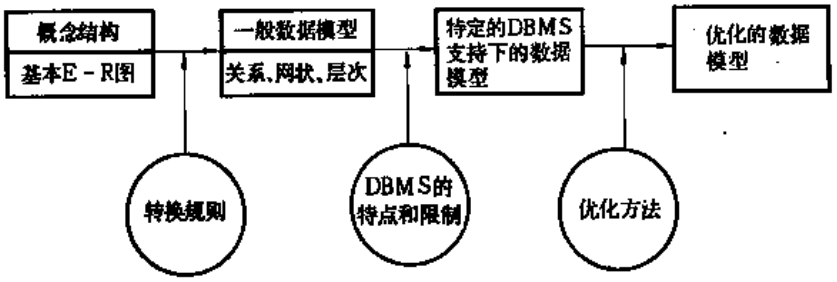


图 6.31 逻辑结构设计时的三个步骤

某些早期设计的应用系统中还在使用网状或层次数据模型，而新设计的数据库应用系统都普遍采用支持关系数据模型的 RDBMS，所以这里只介绍 E-R 图向关系数据模型的转换原则与方法。

6.4.1 E-R 图向关系模型的转换

E-R 图向关系模型的转换要解决的问题是如何将实体和实体间的联系转换为关系模式，如何确定这些关系模式的属性和码。

关系模型的逻辑结构是一组关系模式的集合。E-R 图则是由实体、实体的属性和实体之间的联系三个要素组成的。所以将 E-R 图转换为关系模型实际上就是要将实体、实体的属性和实体之间的联系转换为关系模式，这种转换一般遵循如下原则：

（1）一个实体型转换为一个关系模式。实体的属性就是关系的属性，实体的码就是关系的码。

对于实体间的联系则有以下不同的情况：

（2）一个 1：1 联系可以转换为一个独立的关系模式，也可以与任意一端对应的关系模式合并。如果转换为一个独立的关系模式，则与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，每个实体的码均是该关系的候选码。如果与某一端实体对应的关系模式合并，则需要在该关系模式的属性中加入另一个关系模式的码和联系本身的属性。

（3）一个 1：n 联系可以转换为一个独立的关系模式，也可以与 n 端对应的关系模式合并。如果转换为一个独立的关系模式，则与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，而关系的码为 n 端实体的码。

（4）一个 m：n 联系转换为一个关系模式。与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，而关系的码为各实体码的组合。

（5）三个或三个以上实体间的一个多元联系可以转换为一个关系模式。与该多元联系相连的各实体的码以及联系本身的属性均转换为关系的属性，而关系的码为各实体码的组合。

（6）具有相同码的关系模式可合并。

下面把图 6.30 中虚线上部的 E-R 图转换为关系模型。 关系的码用下横线标出。

部门（部门号，部门名，经理的职工号，…）

此为部门实体对应的关系模式。该关系模式已包含了联系——“领导”所对应的关系模式。经理的职工号是关系的候选码

职工（职工号，部门号，职工名，职务，…）

此为职工实体对应的关系模式。该关系模式已包含了联系——“属于”所对应的关系模式。

产品（产品号，产品名，产品组长的职工号，…）

此为产品实体对应的关系模式。

供应商（供应商号，姓名，…）

此为供应商实体对应的关系模式。

零件（零件号，零件名，…）

此为零件实体对应的关系模式。

职工工作（职工号，产品号，工作天数，…）

此为联系“参加”所对应的关系模式。

供应（产品号，供应商号，零件号，供应量）

此为联系“供应”所对应的关系模式。

形成了一般的数据模型后，下一步就是向特定的 RDBMS 的模型转换。设计人员必须熟悉所用 RDBMS 的功能与限制。这一步是依赖于机器的，不能给出一个普遍的规则，但对于关系模型来说，这种转换通常都比较简单，不会有太多的困难。

6.4.2 数据模型的优化

数据库逻辑设计的结果不是唯一的。为了进一步提高数据库应用系统的性能，还应该根据应用需要适当地修改、调整数据模型的结构，这就是数据模型的优化。关系数据模型的优化通常以规范化理论为指导，方法为：

1. 确定数据依赖。在 6.2.3“数据字典”一节中已讲到用数据依赖分析和表示数据项之间的联系，写出每个数据项之间的数据依赖。如果需求分析阶段没有来得及做，可以现在补做，即按需求分析阶段所得到的语义，分别写出每个关系模式内部各属性之间的数据依赖以及不同关系模式属性之间的数据依赖。

2. 对于各个关系模式之间的数据依赖进行极小化处理，消除冗余的联系，具体方法已在 6.3.4 节中讲解。

3. 按照数据依赖的理论对关系模式逐一进行分析，考察是否存在部分函数依赖、传递函数依赖、多值依赖等，确定各关系模式分别属于第几范式。

4. 按照需求分析阶段得到的处理要求，分析这些模式对于这样的应用环境是否合适，确定是否要对某些模式进行合并或分解。

必须注意的是，并不是规范化程度越高的关系就越优。例如，当查询经常涉及到两个或多个关系模式的属性时，系统经常进行连接运算。连接运算的代价是相当高的，可以说关系模型低效的主要原因就是连接运算引起的。这时可以考虑将这几个关系合并为一个关系。因此在这种情况下，第二范式甚至第一范式也许是合适的。

又如，非 BCNF 的关系模式虽然从理论上分析会存在不同程度的更新异常或冗余，但如果在实际应用中对此关系模式只是查询，并不执行更新操作，则就不会产生实际影响。所以对于一个具体应用来说，到底规范化到什么程度，需要权衡响应时间和潜在问题两者的利弊决定。

5. 对关系模式进行必要的分解，提高数据操作的效率和存储空间的利用率。常用的两种分解方法是水平分解和垂直分解。

水平分解是把(基本)关系的元组分为若干子集合，定义每个子集合为一个子关系，以提高系统的效率。根据“80/20 原则”，一个大关系中，经常被使用的数据只是关系的一部分，约 20%，可以把经常使用的数据分解出来，形成一个子关系。 如果关系 R 上具有 n 个事务，而且多数事务存取的数据不相交，则 R 可分解为少于或等于 n 个子关系，使每个事务存取的数据对应一个关系。

垂直分解是把关系模式 R 的属性分解为若干子集合，形成若干子关系模式。垂直分解的原则是，经常在一起使用的属性从 R 中分解出来形成一个子关系模式。垂直分解可以提高某些事务的效率，但也可能使另一些事务不得不执行连接操作，从而降低了效率。因此是否进行垂直分解取决于分解后 R 上的所有事务的总效率是否得到了提高。垂直分解需要确保无损连接性和保持函数依赖，即保证分解后的关系具有无损连接性和保持函数依赖性。这可以用第五章中的模式分解算法对需要分解的关系模式进行分解和检查。

规范化理论为数据库设计人员判断关系模式的优劣提供了理论标准，可用来预测模式可能出现的问题，使数据库设计工作有了严格的理论基础。

6.4.3 设计用户子模式

将概念模型转换为全局逻辑模型后，还应该根据局部应用需求，结合具体 DBMS 的特点，设计用户的外模式。

目前关系数据库管理系统一般都提供了视图（View）概念，可以利用这

一功能设计更符合局部用户需要的用户外模式。

定义数据库全局模式主要是从系统的时间效率、空间效率、易维护等角度出发。由于用户外模式与模式是相对独立的，因此在定义用户外模式时可以注重考虑用户的习惯与方便。包括：

（1）使用更符合用户习惯的别名

在合并各分 E-R 图时，曾做了消除命名冲突的工作，以使数据库系统中同一关系和属性具有唯一的名字。这在设计数据库整体结构时是非常必要的。用 View 机制可以在设计用户 View 时重新定义某些属性名，使其与用户习惯一致，以方便使用。

（2）可以对不同级别的用户定义不同的 View，以保证系统的安全性。

假设有关系模式——产品（产品号，产品名，规格，单价，生产车间，生产负责人，产品成本，产品合格率，质量等级）。

可以在产品关系上建立两个视图：

为一般顾客建立视图：

产品 1（产品号，产品名，规格，单价）

为产品销售部门建立视图：

产品 2（产品号，产品名，规格，单价，车间，生产负责人）

顾客视图中只包含允许顾客查询的属性；销售部门视图中只包含允许销售部门查询的属性。生产领导部门则可以查询全部产品数据。这样就可以防止用户非法访问本来不允许他们查询的数据，保证了系统的安全性。

（3）简化用户对系统的使用

如果某些局部应用中经常要使用某些很复杂的查询，为了方便用户，可以将这些复杂查询定义为视图，用户每次只对定义好的视图进行查询，大大简化了用户的使用。

6.5 数据库的物理设计

数据库在物理设备上的存储结构与存取方法称为数据库的物理结构，它依赖于给定的计算机系统。为一个给定的逻辑数据模型选取一个最适合应用要求的物理结构的过程，就是数据库的物理设计。

数据库的物理设计通常分为两步：

（1）确定数据库的物理结构，在关系数据库中主要指存取方法和存储结构；

（2）对物理结构进行评价，评价的重点是时间和空间效率。

如果评价结果满足原设计要求，则可进入到物理实施阶段，否则，就需要

重新设计或修改物理结构，有时甚至要返回逻辑设计阶段修改数据模型。

6.5.1 数据库的物理设计的内容和方法

不同的数据库产品所提供的物理环境、存取方法和存储结构有很大差别，能供设计人员使用的设计变量、参数范围也很不相同，因此没有通用的物理设计方法可遵循，只能给出一般的设计内容和原则。希望设计优化的物理数据库结构，使得在数据库上运行的各种事务响应时间小、存储空间利用率高、事务吞吐率大。为此首先对要运行的事务进行详细分析，获得选择物理数据库设计所需要的参数。其次，要充分了解所用的 RDBMS 的内部特征，特别是系统提供的存取方法和存储结构。

对于数据库查询事务，需要得到如下信息：

- 查询的关系；
- 查询条件所涉及的属性；
- 连接条件所涉及的属性；
- 查询的投影属性。

对于数据更新事务，需要得到如下信息：

- 被更新的关系；
- 每个关系上的更新操作条件所涉及的属性；
- 修改操作要改变的属性值。

除此之外，还需要知道每个事务在各关系上运行的频率和性能要求。例如，事务 T 必须在 10 秒钟内结束，这对于存取方法的选择具有重大影响。

上述这些信息是确定关系的存取方法的依据。

应注意的是，数据库上运行的事务会不断变化、增加或减少，以后需要根据上述设计信息的变化调整数据库的物理结构。

通常对于关系数据库物理设计的内容主要包括 ：

- 为关系模式选择存取方法；
- 设计关系、索引等数据库文件的物理存储结构。

下面就介绍这些设计内容和方法。

6.5.2 关系模式存取方法选择

数据库系统是多用户共享的系统，对同一个关系要建立多条存取路径才能满足多用户的多种应用要求。物理设计的任务之一就是要确定选择哪些存取方法，即建立哪些存取路径。

存取方法是快速存取数据库中数据的技术。数据库管理系统一般都提供多

种存取方法。常用的存取方法有三类。第一类是索引方法，目前主要是 B+树索引方法；第二类是聚簇（Cluster）方法；第三类是 HASH 方法。

B+树索引方法是数据库中经典的存取方法，使用最普遍。

一、索引存取方法的选择

所谓选择索引存取方法实际上就是根据应用要求确定对关系的哪些属性列建立索引、哪些属性列建立组合索引、哪些索引要设计为唯一索引等。一般来说：

1. 如果一个(或一组)属性经常在查询条件中出现，则考虑在这个(或这组)属性上建立索引(或组合索引)；

2. 如果一个属性经常作为最大值和最小值等聚集函数的参数，则考虑在这个属性上建立索引；

3. 如果一个(或一组)属性经常在连接操作的连接条件中出现，则考虑在这个(或这组)属性上建立索引；

关系上定义的索引数并不是越多越好，系统为维护索引要付出代价，查找索引也要付出代价。例如，若一个关系的更新频率很高，这个关系上定义的索引数不能太多。因为更新一个关系时，必须对这个关系上有关的索引做相应的修改。

二、聚簇存取方法的选择

为了提高某个属性（或属性组）的查询速度，把这个或这些属性（称为聚簇码）上具有相同值的元组集中存放在连续的物理块称为聚簇。

聚簇功能可以大大提高按聚簇码进行查询的效率。例如要查询信息系的所有学生名单，设信息系有 500 名学生，在极端情况下，这 500 名学生所对应的数据元组分布在 500 个不同的物理块上。尽管对学生关系已按所在系建有索引，由索引很快找到了信息系学生的元组标识，避免了全表扫描，然而再由元组标识去访问数据块时就要存取 500 个物理块，执行 500 次 I/O 操作。如果将同一系的学生元组集中存放，则每读一个物理块可得到多个满足查询条件的元组，从而显著地减少了访问磁盘的次数。

聚簇功能不但适用于单个关系，也适用于经常进行连接操作的多个关系。即把多个连接关系的元组按连接属性值聚集存放，聚簇中的连接属性称为聚簇码。这就相当于把多个关系按“预连接”的形式存放，从而大大提高连接操作的效率。

一个数据库可以建立多个聚簇，一个关系只能加入一个聚簇。

选择聚簇存取方法，即确定需要建立多少个聚簇，每个聚簇中包括哪些

关系。

下面先设计候选聚簇，一般来说：

（1）对经常在一起进行连接操作的关系可以建立聚簇；

（2）如果一个关系的一组属性经常出现在相等比较条件中，则该单个关系可建立聚簇；

（3）如果一个关系的一个(或一组)属性上的值重复率很高，则此单个关系可建立聚簇。即对应每个聚簇码值的平均元组数不是太少。太少了，聚簇的效果不明显。

然后检查候选聚簇中的关系，取消其中不必要的关系：

（1）从聚簇中删除经常进行全表扫描的关系；

（2）从聚簇中删除更新操作远多于连接操作的关系；

（3）不同的聚簇中可能包含相同的关系，一个关系可以在某一个聚簇中，但不能同时加入多个聚簇。要从这多个聚簇方案(包括不建立聚簇)中选择一个较优的，即在这个聚簇上运行各种事务的总代价最小。

必须强调的是，聚簇只能提高某些应用的性能，而且建立与维护聚簇的开销是相当大的。对已有关系建立聚簇，将导致关系中元组移动其物理存储位置，并使此关系上原有的索引无效，必须重建。当一个元组的聚簇码值改变时，该元组的存储位置也要做相应移动，聚簇码值要相对稳定，以减少修改聚簇码值所引起的维护开销。

因此，当通过聚簇码进行访问或连接是该关系的主要应用，与聚簇码无关的其他访问很少或者是次要的，这时可以使用聚簇。尤其当 SQL 语句中包含有与聚簇码有关的 ORDER BY，GROUP BY，UNION，DISTINCT 等子句或短语时，使用聚簇特别有利，可以省去对结果集的排序操作；否则很可能会适得其反。

三、HASH 存取方法的选择

有些数据库管理系统提供了 HASH 存取方法。选择 HASH 存取方法的规则如下：

如果一个关系的属性主要出现在等连接条件中或主要出现在相等比较选择条件中，而且满足下列两个条件之一，则此关系可以选择 HASH 存取方法：

（1）如果一个关系的大小可预知，而且不变；

（2）如果关系的大小动态改变，而且数据库管理系统提供了动态 HASH 存取方法。

6.5.3 确定数据库的存储结构

确定数据库物理结构主要指确定数据的存放位置和存储结构，包括确定关系、索引、聚簇、日志、备份等的存储安排和存储结构；确定系统配置等。

确定数据的存放位置和存储结构要综合考虑存取时间、存储空间利用率和维护代价三方面的因素。这三个方面常常是相互矛盾的，因此需要进行权衡，选择一个折中方案。

1. 确定数据的存放位置

为了提高系统性能，应该根据应用情况将数据的易变部分与稳定部分、经常存取部分和存取频率较低部分分开存放。

例如，目前许多计算机都有多个磁盘，因此可以将表和索引放在不同的磁盘上，在查询时，由于两个磁盘驱动器并行工作，可以提高物理 I/O 读写的效率；也可以将比较大的表分放在两个磁盘上，以加快存取速度，这在多用户环境下特别有效；还可以将日志文件与数据库对象（表、索引等）放在不同的磁盘上以改进系统的性能。此外，数据库的数据备份和日志文件备份等只在故障恢复时才使用，而且数据量很大，可以存放在磁带上。

由于各个系统所能提供的对数据进行物理安排的手段、方法差异很大，因此设计人员应仔细了解给定的 RDBMS 提供的方法和参数，针对应用环境的要求，对数据进行适当的物理安排。

2. 确定系统配置

DBMS 产品一般都提供了一些系统配置变量、存储分配参数，供设计人员和 DBA 对数据库进行物理优化。初始情况下，系统都为这些变量赋予了合理的缺省值。但是这些值不一定适合每一种应用环境，在进行物理设计时，需要重新对这些变量赋值，以改善系统的性能。

系统配置变量很多，例如：同时使用数据库的用户数，同时打开的数据库对象数，内存分配参数，缓冲区分配参数（使用的缓冲区长度、个数），存储分配参数，物理块的大小，物理块装填因子，时间片大小，数据库的大小，锁的数目等。这些参数值影响存取时间和存储空间的分配，在物理设计时就要根据应用环境确定这些参数值，以使系统性能最佳。

在物理设计时对系统配置变量的调整只是初步的，在系统运行时还要根据系统实际运行情况做进一步的调整，以期切实改进系统性能。

6.5.4 评价物理结构

数据库物理设计过程中需要对时间效率、空间效率、维护代价和各种用户

要求进行权衡，其结果可以产生多种方案，数据库设计人员必须对这些方案进行细致的评价，从中选择一个较优的方案作为数据库的物理结构。

评价物理数据库的方法完全依赖于所选用的 DBMS，主要是从定量估算各种方案的存储空间、存取时间和维护代价入手，对估算结果进行权衡、比较，选择出一个较优的合理的物理结构。如果该结构不符合用户需求，则需要修改设计。

6.6 数据库的实施和维护

完成数据库的物理设计之后，设计人员就要用 RDBMS 提供的数据定义语言和其他实用程序将数据库逻辑设计和物理设计结果严格描述出来，成为 DBMS 可以接受的源代码，再经过调试产生目标模式。然后就可以组织数据入库了，这就是数据库实施阶段。

6.6.1 数据的载入和应用程序的调试

数据库实施阶段包括两项重要的工作，一项是数据的载入，另一项是应用程序的编码和调试。

一般数据库系统中，数据量都很大，而且数据来源于部门中的各个不同的单位，数据的组织方式、结构和格式都与新设计的数据库系统有相当的差距，组织数据录入就要将各类源数据从各个局部应用中抽取出来，输入计算机，再分类转换，最后综合成符合新设计的数据库结构的形式，输入数据库。因此这样的数据转换、组织入库的工作是相当费力费时的工作。

特别是原系统是手工数据处理系统时，各类数据分散在各种不同的原始表格、凭证、单据之中。在向新的数据库系统中输入数据时，还要处理大量的纸质文件，工作量就更大。

由于各个不同的应用环境差异很大，不可能有通用的转换器，DBMS 产品也不提供通用的转换工具。为提高数据输入工作的效率和质量，应该针对具体的应用环境设计一个数据录入子系统，由计算机来完成数据入库的任务。

由于要入库的数据在原来的系统中的格式结构与新系统中不完全一样，有的差别可能还比较大，不仅向计算机内输入数据时发生错误，转换过程中也有可能出错。因此在源数据入库之前要采用多种方法对它们进行检验，以防止不正确的数据入库，这部分的工作在整个数据输入子系统中是非常重要的。

在设计数据输入子系统时还要注意原有系统的特点，例如对原有系统是人工数据处理系统的情况，尽管新系统的数据结构可能与原系统有很大差别，在

设计数据输入子系统时，尽量让输入格式与原系统结构相近，这不仅使处理手工文件比较方便，更重要的是减少用户出错的可能性，保证数据输入的质量。现有的 DBMS 一般都提供不同 DBMS 之间数据转换的工具，若原来是数据库系统，就可以利用新系统的数据转换工具，先将原系统中的表转换成新系统中相同结构的临时表，再将这些表中的数据分类、转换、综合成符合新系统的数据模式，插入相应的表中。

数据库应用程序的设计应该与数据库设计同时进行，因此在组织数据入库的同时还要调试应用程序。应用程序的设计、编码和调试的方法、步骤在软件工程等课程中有详细讲解，这里就不赘述了。

6.6.2 数据库的试运行

在原有系统的数据有一小部分已输入数据库后，就可以开始对数据库系统进行联合调试，这又称为数据库的试运行。

这一阶段要实际运行数据库应用程序，执行对数据库的各种操作，测试应用程序的功能是否满足设计要求。如果不满足，对应用程序部分则要修改、调整，直到达到设计要求为止。

在数据库试运行时，还要测试系统的性能指标，分析其是否达到设计目标。在对数据库进行物理设计时已初步确定了系统的物理参数值，但一般的情况下，设计时的考虑在许多方面只是近似的估计，和实际系统运行总有一定的差距，因此必须在试运行阶段实际测量和评价系统性能指标。事实上，有些参数的最佳值往往是经过运行调试后找到的。如果测试的结果与设计目标不符，则要返回物理设计阶段，重新调整物理结构，修改系统参数，某些情况下甚至要返回逻辑设计阶段，修改逻辑结构。

这里特别要强调两点，第一，上面已经讲到组织数据入库是十分费时费力的事，如果试运行后还要修改数据库的设计，还要重新组织数据入库。因此应分期分批地组织数据入库，先输入小批量数据做调试用，待试运行基本合格后，再大批量输入数据，逐步增加数据量，逐步完成运行评价。

第二，在数据库试运行阶段，由于系统还不稳定，硬、软件故障随时都可能发生。而系统的操作人员对新系统还不熟悉，误操作也不可避免，因此应首先调试运行 DBMS 的恢复功能，做好数据库的转储和恢复工作。一旦故障发生，能使数据库尽快恢复，尽量减少对数据库的破坏。

6.6.3 数据库的运行和维护

数据库试运行合格后，数据库开发工作就基本完成，即可投入正式运行了。

但是，由于应用环境在不断变化，数据库运行过程中物理存储也会不断变化，对数据库设计进行评价、调整、修改等维护工作是一个长期的任务，也是设计工作的继续和提高。

在数据库运行阶段，对数据库经常性的维护工作主要是由 DBA 完成的，它包括：

1. 数据库的转储和恢复

数据库的转储和恢复是系统正式运行后最重要的维护工作之一。DBA 要针对不同的应用要求制定不同的转储计划，以保证一旦发生故障能尽快将数据库恢复到某种一致的状态，并尽可能减少对数据库的破坏。

2. 数据库的安全性、完整性控制

在数据库运行过程中，由于应用环境的变化，对安全性的要求也会发生变化，比如有的数据原来是机密的，现在是可以公开查询的了，而新加入的数据又可能是机密的了。系统中用户的密级也会改变。这些都需要 DBA 根据实际情况修改原有的安全性控制。同样，数据库的完整性约束条件也会变化，也需要 DBA 不断修正，以满足用户要求。

3. 数据库性能的监督、分析和改造

在数据库运行过程中，监督系统运行，对监测数据进行分析，找出改进系统性能的方法是 DBA 的又一重要任务。目前有些 DBMS 产品提供了监测系统性能参数的工具，DBA 可以利用这些工具方便地得到系统运行过程中一系列性能参数的值。DBA 应仔细分析这些数据，判断当前系统运行状况是否是最佳，应当做哪些改进。例如调整系统物理参数，或对数据库进行重组织或重构造等。

4. 数据库的重组织与重构造

数据库运行一段时间后，由于记录不断增、删、改，会使数据库的物理存储情况变坏，降低了数据的存取效率，数据库性能下降，这时 DBA 就要对数据库进行重组织，或部分重组织（只对频繁增、删的表进行重组织）。DBMS 一般都提供数据重组织用的实用程序。在重组织的过程中，按原设计要求重新安排存储位置、回收垃圾、减少指针链等，提高系统性能。

数据库的重组织，并不修改原设计的逻辑和物理结构，而数据库的重构造则不同，它是指部分修改数据库的模式和内模式。

由于数据库应用环境发生变化，增加了新的应用或新的实体，取消了某些应用，有的实体与实体间的联系也发生了变化等，使原有的数据库设计不能满足新的需求，需要调整数据库的模式和内模式。例如，在表中增加或删除某些数据项，改变数据项的类型，增加或删除某个表，改变数据库的容量，增加或删除某些索引等。当然数据库的重构也是有限的，只能做部分修改。如果应用

变化太大，重构也无济于事，说明此数据库应用系统的生命周期已经结束，应该设计新的数据库应用系统了。

6.7 小　结

数据库设计这一章主要讨论数据库设计的方法和步骤，列举了较多的实例，详细介绍了数据库设计各个阶段的目标、方法、应注意的事项。其中的重点是概念结构的设计和逻辑结构的设计，这也是数据库设计过程中最重要的两个环节。

学习这一章，要努力掌握书中讨论的基本方法，还要能在实际工作中运用这些思想，设计符合应用需求的数据库应用系统。

习　题

1. 试述数据库设计过程。
2. 试述数据库设计过程的各个阶段上的设计描述。
3. 试述数据库设计过程中结构设计部分形成的数据库模式。
4. 试述数据库设计的特点。
5. 需求分析阶段的设计目标是什么？调查的内容是什么？
6. 数据字典的内容和作用是什么？
7. 什么是数据库的概念结构试述其特点和设计策略。
8. 什么叫数据抽象？试举例说明。
9. 试述数据库概念结构设计的重要性和设计步骤。
10. 什么是 E-R 图？构成 E-R 图的基本要素是什么？
11. 为什么要视图集成？视图集成的方法是什么？
12. 什么是数据库的逻辑结构设计？试述其设计步骤。
13. 试述把 E-R 图转换为 DBTG 模型和关系模型的转换规则。
14. 你能给出由 E-R 图转换为 IMS 模型的转换规则吗？
15. 试把第一章习题 7 和习题 8 中的 E-R 图转换为 DBTG 模型、IMS 模型、关系模型。
16. 试述逻辑设计阶段中运用 LRA 方法优化模型的方法和步骤。
17. 试用规范化理论中有关范式的概念分析习题 15 中你设计的关系模型中各个关系模式的码，它们属于第几范式？会产生什么更新异常？
18. 规范化理论对数据库设计有什么指导意义？
19. 试述数据库物理设计的内容和步骤。

20. 你能给出关系数据库物理设计的主要内容吗？例如 ORACLE 数据库物理设计的内容。

21. 数据输入在实施阶段的重要性是什么？如何保证输入数据的正确性？

22. 什么是数据库的再组织和重构造？为什么要进行数据库的再组织和重构造？

23. 现有一局部应用，包括两个实体："出版社"和"作者"，这两个实体是多对多的联系，请读者自己设计适当的属性，画出 E-R 图，再将其转换为关系模型（包括关系名、属性名、码和完整性约束条件）。

24. 请设计一个图书馆数据库，此数据库中对每个借阅者保存读者记录，包括：读者号，姓名，地址，性别，年龄，单位。对每本书存有：书号，书名，作者，出版社。对每本被借出的书存有读者号、借出日期和应还日期。要求：给出 E-R 图，再将其转换为关系模型。

本章参考文献

1 Wiederhold G. Database Design(Second Edition). McGraw-Hill，1983

（本书是一本综合性教科书，覆盖了数据库设计的各个阶段，重点强调了物理设计阶段。）

2 Albano A(ed.). Computer-Aided Database Design：The DATAID Project. North-Holland，1985

（本书讲解了数据库设计及计算机辅助数据库设计的方法和工具。）

3 Yao S B(ed.). Principles of Database Design：Logical Organization. Vol.1，Prentice-Hall，1985

（本书收集了从需求说明技术到模式调整等方面的研究成果。）

4 Yao S B(ed.). 罗大卫、刘化明译. 数据库设计 I. 计算机工程与应用，1985,7

5 姚卿达. 数据库设计. 北京：高等教育出版社，1987

（本书系统讲解了数据库设计的方法和工具，覆盖了数据库设计的所有阶段，提出了适合中小型数据库设计的多级方法 DBD/MLM，并设计了一个处理要求模型化工具 PSSB 作为处理模式的描述语言，把结构和处理设计相结合。）

6 王珊，冯念真. 计算机应用系统的设计和开发. 北京：高等教育出版社，1989

（本书从应用系统的开发和研制的角度讲解了数据库设计。）

7 Chen P P S. The Entity-Relationship Model：Towards a Unified View of Data. ACM TODS Vol. 1，No.1，March 1976

（Peter Chen 在本文中首次提出了著名的实体-联系模型，在以后的论文中他对 E-R 模型作了深入的研究。）

8 Chen P P S. The Entity-Relationship Model：A Basis for the Enterprise View of Data. AFIPS Conf. Proc.Vol.46，AFIPS Press，NJ，1977

9 Chen P P S. The Entity-Relationship Approach to Logical DataBase Design. Q.E.D. MonographSeries, 1977

10 Chen P P S. Recent Literature of the Entity-Relationship Approach. Proceedings of International Conference on Entity-Relationship Approach to Systems Analysis and Design, 1979

11 Smith J, Smith D. Database Abstractions: Aggregation and Generalization. TODS, 2:2, June 1977

（本文在 E-R 模型中引入了概括(Generalization)和聚集(Aggregation)的概念。）

12 Hammer M, McLeod D. Database Description with SDM: A Semantic Data Model. TODS, 6:3, September 1980

（本文提出了扩展的 ER 模型，引进了超类/子类等概念。）

13 Ng P. Further Analysis of the Entity-Relationship Approach to Database Design. TSE, 7:1, January 1981

（本文将 E-R 模型形式化。）

14 Elmasri R, Weeldreyer J, Hevner A. The Category Concept: An Extension to the Entity-Relationship Model. International Journal on Data and Knowledge Engineering, 1:1, May 1985

（本文对 E-R 模型进行了扩充。）

15 Teorey T, Yang D, Fry J. A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. ACM Computing Surveys, 18:2, June 1986

16 王珊，吴鹤琦. E-R 图/数据模型转换的一点注记. 小型微型计算机系统，1983.6

（本文研究了从 E-R 模型向数据模型转换的问题。）

17 萨师煊，王珊. N-E-R Model: A New Method to Design Enterprise Schema. 第一届国际计算机应用大会论文集，1984

18 Schkolnick M. A Survey of Physical Database Design Methodology and Techniques. in Proceedings of VLDB, September 1978

（本文综述了物理数据库设计方法。）

19 Yeh R T et al. Software and Data Base Engineering: Towards a Common Design Methodology. Issues in Data Base Management, 1979

（本文指出数据库设计是一项庞大的软件工程。）

20 Teorey T. The Logical Record Access Approach to Data Base Design. ACM computing Surveys, Vol.12, No.12, 1980

21 Gan e Chris, Sarson Trish. Structured Systems Analysis, 1979

22 Hainut J L. Theoretical and Practical Tools for Data Base Design. in Proceedings of VLDB, 1981

23 萨师煊. 数据库的理论与实践. 计算机科学，1983.3

24 萨师煊. 数据库系统和系统分析. 计算机工程与设计，1984.3

25 萨师煊，王珊. 数据库设计理论和实践. 计算机应用与软件，VOL.2，NO.4，1984

26 Ceri S，Navathe S，Wiederhold G. Distribution Design of Logical Database Schemas. TSE，9:4，July 1983

（本文全面讨论了分布数据库的逻辑设计。）

27 Batini C，Lenzerini M. and Navathe S. A Comparative Analysis of Methodologies for Database Schema Integration. ACM Computing Surveys，18:4，December 1986

（本文研究和比较了数据库设计中各种模式集成方法。）

28 Navathe S，Gadgil S. A Methodology for View Integration in Logical Database Design. in Proceedings of VLDB，1982

（本文提出了数据库设计中视图集成方法。）

29 Navathe S，Elmasri R，Larson J. Integrating User Views in Database Design，IEEE Computer，19:1，January 1986

30 Mannino M，Effelsberg W. Matching Techniques in Global Schema Design. in Proceedings of IEEE Inter. Conference on Data Engineering，1984

31 Navathe S，Sashidhar T，Elmasri R. Relationship Merging in Schema Integration，in Proceedings of VLDB，1984

32 Larson J，Navathe S，Elmasri R. Attribute Equivalence and its Use in Schema Integration. in TSE，1989

（文讨论了视图集成中的问题。）

33 Curtice R. Data Dictionaries：An Assessment of Current Practice and Problems. in Proceedings of VLDB，1981

（本文给出了关于数据字典的综述。）

34 Goldfine A， Konig P. A Technical Overview of the Information Resource Dictionary System (IRDS)（Second Edition）. NBS IR 88-3700，National Bureau of Standards

（本文讨论了数据字典在数据库设计中的作用。）

35 Navathe S，Kerschberg L. Role of Data Dictionaries in Database Design. Information and Management，10:1，January 1986

（本文讨论了数据库设计的各个阶段，指出了数据字典在数据库设计中的的中心地位。）

36 ANSI(American National Standards Institute)，Information Resource Dictionary Systems. Document ANSI X3.138，1989

第三篇　系　统　篇

第七章　数据库恢复技术

本书第七章、第八章讨论事务处理技术。事务是一系列的数据库操作，是数据库应用程序的基本逻辑单元。事务处理技术主要包括数据库恢复技术和并发控制技术。数据库恢复机制和并发控制机制是数据库管理系统的重要组成部分。本章讨论数据库恢复的概念和常用技术。

7.1　事务的基本概念

在讨论数据库恢复技术之前先讲解事务的基本概念和事务的性质。

一、事务(Transaction)

所谓事务是用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位。例如，在关系数据库中，一个事务可以是一条 SQL 语句、一组 SQL 语句或整个程序。

事务和程序是两个概念。一般地讲，一个程序中包含多个事务。

事务的开始与结束可以由用户显式控制。如果用户没有显式地定义事务，则由 DBMS 按缺省规定自动划分事务。在 SQL 语言中，定义事务的语句有三条：

```
BEGIN TRANSACTION
COMMIT
ROLLBACK
```

事务通常是以 BEGIN TRANSACTION 开始，以 COMMIT 或 ROLLBACK 结束。COMMIT 表示提交，即提交事务的所有操作。具体地说就是将事务中所有对数据库的更新写回到磁盘上的物理数据库中去，事务正常结束。ROLLBACK 表示回滚，即在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作全部撤销，滚回到事务开始时的状态。这里的操作指对数据库的更新操作。

二、事务的特性

事务具有四个特性：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持续性（Durability）。这个四个特性也简称为 ACID 特性。

1. 原子性

事务是数据库的逻辑工作单位，事务中包括的诸操作要么都做，要么都不做。

2. 一致性

事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态。因此当数据库只包含成功事务提交的结果时，就说数据库处于一致性状态。如果数据库系统运行中发生故障，有些事务尚未完成就被迫中断，这些未完成事务对数据库所做的修改有一部分已写入物理数据库，这时数据库就处于一种不正确的状态，或者说是不一致的状态。例如某公司在银行中有 A，B 两个账号，现在公司想从账号 A 中取出一万元，存入账号 B。那么就可以定义一个事务，该事务包括两个操作，第 1 个操作是从账号 A 中减去一万元，第 2 个操作是向账号 B 中加入一万元。这两个操作要么全做，要么全不做。全做或者全不做，数据库都处于一致性状态。如果只做一个操作则用户逻辑上就会发生错误，少了一万元，这时数据库就处于不一致性状态。可见一致性与原子性是密切相关的。

3. 隔离性

一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对其他并发事务是隔离的，并发执行的各个事务之间不能互相干扰。

4. 持续性

持续性也称永久性（Permanence），指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其执行结果有任何影响。

事务是恢复和并发控制的基本单位。所以下面的讨论均以事务为对象。

保证事务 ACID 特性是事务处理的重要任务。事务 ACID 特性可能遭到破坏的因素有：

（1）多个事务并行运行时，不同事务的操作交叉执行；

（2）事务在运行过程中被强行停止。

在第一种情况下，数据库管理系统必须保证多个事务的交叉运行不影响这些事务的原子性。在第二种情况下，数据库管理系统必须保证被强行终止的事务对数据库和其他事务没有任何影响。

这些就是数据库管理系统中恢复机制和并发控制机制的责任。

7.2 数据库恢复概述

尽管数据库系统中采取了各种保护措施来防止数据库的安全性和完整性被破坏，保证并发事务的正确执行，但是计算机系统中硬件的故障、软件的错误、操作员的失误以及恶意的破坏仍是不可避免的，这些故障轻则造成运行事务非正常中断，影响数据库中数据的正确性，重则破坏数据库，使数据库中全部或部分数据丢失，因此数据库管理系统必须具有把数据库从错误状态恢复到某一已知的正确状态（亦称为一致状态或完整状态）的功能，这就是数据库的恢复。恢复子系统是数据库管理系统的一个重要组成部分，而且还相当庞大，常常占整个系统代码的百分之十以上。数据库系统所采用的恢复技术是否行之有效，不仅对系统的可靠程度起着决定性作用，而且对系统的运行效率也有很大影响，是衡量系统性能优劣的重要指标。

7.3 故障的种类

数据库系统中可能发生各种各样的故障，大致可以分以下几类：

一、事务内部的故障

事务内部的故障有的是可以通过事务程序本身发现的（见下面转账事务的例子），有的是非预期的，不能由事务程序处理的。

例如：银行转账事务，这个事务把一笔金额从一个账户甲转给另一个账户乙。

```
BEGIN TRANSACTION
  读账户甲的余额 BALANCE;
  BALANCE = BALANCE-AMOUNT;  (AMOUNT 为转账金额)
  IF (BALANCE < 0 )  THEN
    { 打印'金额不足，不能转账' ;
      ROLLBACK;  (撤销刚才的修改，恢复事务) }
  ELSE
    { 读账户乙的余额 BALANCE1;
      BALANCE1 = BALANCE1+AMOUNT;
```

写回 BALANCE1；

COMMIT；}

这个例子所包括的两个更新操作要么全部完成要么全部不做。否则就会使数据库处于不一致状态，例如只把账户甲的余额减少了而没有把账户乙的余额增加。

在这段程序中若产生账户甲余额不足的情况，应用程序可以发现并让事务滚回，撤销已作的修改，恢复数据库到正确状态。

事务内部更多的故障是非预期的，是不能由应用程序处理的。如运算溢出、并发事务发生死锁而被选中撤销该事务、违反了某些完整性限制等。以后，事务故障仅指这类非预期的故障。

事务故障意味着事务没有达到预期的终点（COMMIT 或者显式的 ROLLBACK），因此，数据库可能处于不正确状态。恢复程序要在不影响其他事务运行的情况下，强行回滚（ROLLBACK）该事务，即撤销该事务已经作出的任何对数据库的修改，使得该事务好像根本没有启动一样。

这类恢复操作称为事务撤销（UNDO）。

二、系统故障

系统故障是指造成系统停止运转的任何事件，使得系统要重新启动。例如，特定类型的硬件错误（CPU 故障）、操作系统故障、DBMS 代码错误、突然停电等。这类故障影响正在运行的所有事务，但不破坏数据库。这时主存内容，尤其是数据库缓冲区（在内存）中的内容都被丢失，所有运行事务都非正常终止。发生系统故障时，一些尚未完成的事务的结果可能已送入物理数据库，从而造成数据库可能处于不正确的状态。为保证数据一致性，需要清除这些事务对数据库的所有修改。

恢复子系统必须在系统重新启动时让所有非正常终止的事务回滚，强行撤销（UNDO）所有未完成事务。

另一方面，发生系统故障时，有些已完成的事务可能有一部分甚至全部留在缓冲区，尚未写回到磁盘上的物理数据库中，系统故障使得这些事务对数据库的修改部分或全部丢失，这也会使数据库处于不一致状态，因此应将这些事务已提交的结果重新写入数据库。所以系统重新启动后，恢复子系统除需要撤销所有未完成事务外，还需要重做（REDO）所有已提交的事务，以将数据库真正恢复到一致状态。

三、介质故障

系统故障常称为软故障（Soft Crash），介质故障称为硬故障（Hard Crash）。

硬故障指外存故障，如磁盘损坏、磁头碰撞，瞬时强磁场干扰等。这类故障将破坏数据库或部分数据库，并影响正在存取这部分数据的所有事务。这类故障比前两类故障发生的可能性小得多，但破坏性最大。

四、计算机病毒

计算机病毒是一种人为的故障或破坏，是一些恶作剧者研制的一种计算机程序。这种程序与其他程序不同，它像微生物学所称的病毒一样可以繁殖和传播，并造成对计算机系统包括数据库的危害。

病毒的种类很多，不同病毒有不同的特征。小的病毒只有 20 条指令，不到 50B。大的病毒像一个操作系统，由上万条指令组成。

有的病毒传播很快，一旦侵入系统就马上摧毁系统；有的病毒有较长的潜伏期，机器在感染后数天或数月才开始发病；有的病毒感染系统所有的程序和数据；有的只对某些特定的程序和数据感兴趣。多数病毒一开始并不摧毁整个计算机系统，它们只在数据库中或其他数据文件中将小数点向左或向右移一移，增加或删除一两个“0”。

计算机病毒已成为计算机系统的主要威胁，自然也是数据库系统的主要威胁。为此计算机的安全工作者已研制了许多预防病毒的“疫苗”，检查、诊断、消灭计算机病毒的软件也在不断发展。但是，至今还没有一种使得计算机“终生”免疫的疫苗。因此数据库一旦被破坏仍要用恢复技术把数据库加以恢复。

总结各类故障，对数据库的影响有两种可能性。一是数据库本身被破坏。二是数据库没有破坏，但数据可能不正确，这是因为事务的运行被非正常终止造成的。

恢复的基本原理十分简单。可以用一个词来概括：冗余。这就是说，数据库中任何一部分被破坏的或不正确的数据可以根据存储在系统别处的冗余数据来重建。尽管恢复的基本原理很简单但实现技术的细节却相当复杂，下面略去一些细节，介绍数据库恢复的实现技术。

7.4 恢复的实现技术

恢复机制涉及的两个关键问题是：第一，如何建立冗余数据；第二，如何利用这些冗余数据实施数据库恢复。

建立冗余数据最常用的技术是数据转储和登录日志文件。通常在一个数据库系统中，这两种方法是一起使用的。

7.4.1 数据转储

数据转储是数据库恢复中采用的基本技术。所谓转储即 DBA 定期地将整个数据库复制到磁带或另一个磁盘上保存起来的过程。这些备用的数据文本称为后备副本或后援副本。

当数据库遭到破坏后可以将后备副本重新装入，但重装后备副本只能将数据库恢复到转储时的状态，要想恢复到故障发生时的状态，必须重新运行自转储以后的所有更新事务。例如在图 7.1 中，系统在 T_a 时刻停止运行事务进行数据库转储，在 T_b 时刻转储完毕，得到 T_b 时刻的数据库一致性副本。系统运行到 T_f 时刻发生故障。为恢复数据库，首先由 DBA 重装数据库后备副本，将数据库恢复至 T_b 时刻的状态，然后重新运行自 T_b～T_f 时刻的所有更新事务，这样就把数据库恢复到故障发生前的一致状态。

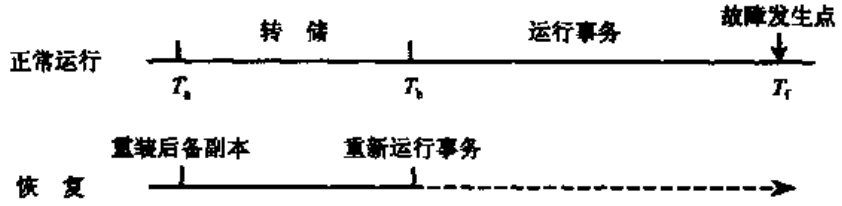


图 7.1 转储和恢复

转储是十分耗费时间和资源的，不能频繁进行。DBA 应该根据数据库使用情况确定一个适当的转储周期。

转储可分为静态转储和动态转储。

静态转储是在系统中无运行事务时进行的转储操作。即转储操作开始的时刻，数据库处于一致性状态，而转储期间不允许（或不存在）对数据库的任何存取、修改活动。显然，静态转储得到的一定是一个数据一致性的副本。

静态转储简单，但转储必须等待正运行的用户事务结束才能进行，同样，新的事务必须等待转储结束才能执行。显然，这会降低数据库的可用性。

动态转储是指转储期间允许对数据库进行存取或修改。即转储和用户事务可以并发执行。

动态转储可克服静态转储的缺点，它不用等待正在运行的用户事务结束，也不会影响新事务的运行。但是，转储结束时后援副本上的数据并不能保证正确有效。例如，在转储期间的某个时刻 T_c，系统把数据 A=100 转储到磁带上，而在下一时刻 T_d，某一事务将 A 改为 200。转储结束后，后备副本上的 A 已是过时的数据了。

为此，必须把转储期间各事务对数据库的修改活动登记下来，建立日志文

件（log file）。这样，后援副本加上日志文件就能把数据库恢复到某一时刻的正确状态。

转储还可以分为海量转储和增量转储两种方式。海量转储是指每次转储全部数据库。增量转储则指每次只转储上一次转储后更新过的数据。从恢复角度看，使用海量转储得到的后备副本进行恢复一般说来会更方便些。但如果数据库很大，事务处理又十分频繁，则增量转储方式更实用更有效。

数据转储有两种方式，分别可以在两种状态下进行，因此数据转储方法可以分为四类：动态海量转储、动态增量转储、静态海量转储和静态增量转储，如表 7.1 所示。

表 7.1 数据转储分类

		转储状态	
		动态转储	静态转储
转储方式	海量转储	动态海量转储	静态海量转储
	增量转储	动态增量转储	静态增量转储

7.4.2 登记日志文件（Logging）

一、日志文件的格式和内容

日志文件是用来记录事务对数据库的更新操作的文件。不同数据库系统采用的日志文件格式并不完全一样。概括起来日志文件主要有两种格式：以记录为单位的日志文件和以数据块为单位的日志文件。

对于以记录为单位的日志文件，日志文件中需要登记的内容包括：

- 各个事务的开始(BEGIN TRANSACTION)标记
- 各个事务的结束(COMMIT 或 ROLLBACK)标记
- 各个事务的所有更新操作

这里每个事务开始的标记、每个事务的结束标记和每个更新操作均作为日志文件中的一个日志记录(log record)。

每个日志记录的内容主要包括：

- 事务标识（标明是哪个事务）
- 操作的类型（插入、删除或修改）
- 操作对象(记录内部标识)
- 更新前数据的旧值（对插入操作而言，此项为空值）
- 更新后数据的新值（对删除操作而言，此项为空值）

对于以数据块为单位的日志文件，日志记录的内容包括事务标识和被更新的数据块。由于将更新前的整个块和更新后的整个块都放入日志文件中，操作的类型和操作对象等信息就不必放入日志记录中。

二、日志文件的作用

日志文件在数据库恢复中起着非常重要的作用。可以用来进行事务故障恢复和系统故障恢复，并协助后备副本进行介质故障恢复。具体作用是：

（1）事务故障恢复和系统故障恢复必须用日志文件。

（2）在动态转储方式中必须建立日志文件，后援副本和日志文件综合起来才能有效地恢复数据库。

（3）在静态转储方式中，也可以建立日志文件。当数据库毁坏后可重新装入后援副本把数据库恢复到转储结束时刻的正确状态，然后利用日志文件，把已完成的事务进行重做处理，对故障发生时尚未完成的事务进行撤销处理。这样不必重新运行那些已完成的事务程序就可把数据库恢复到故障前某一时刻的正确状态,如图 7.2 所示。

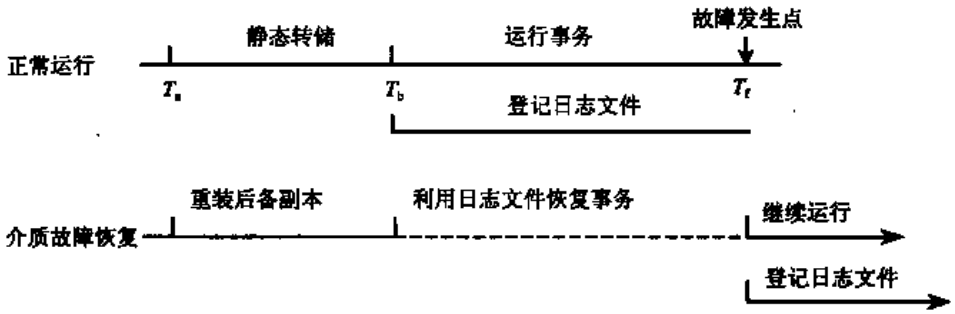


图 7.2 利用日志文件恢复

三、登记日志文件（Logging）

为保证数据库是可恢复的，登记日志文件时必须遵循两条原则：

（1）登记的次序严格按并发事务执行的时间次序。

（2）必须先写日志文件，后写数据库。

把对数据的修改写到数据库中和把表示这个修改的日志记录写到日志文件中是两个不同的操作。有可能在这两个操作之间发生故障，即这两个写操作只完成了一个。如果先写了数据库修改，而在运行记录中没有登记这个修改，则以后就无法恢复这个修改了。如果先写日志，但没有修改数据库，按日志文件恢复时只不过是多执行一次不必要的 UNDO 操作，并不会影响数据库的正确

性。所以为了安全，一定要先写日志文件，即首先把日志记录写到日志文件中，然后写数据库的修改。这就是“先写日志文件”的原则。

7.5 恢 复 策 略

当系统运行过程中发生故障，利用数据库后备副本和日志文件就可以将数据库恢复到故障前的某个一致性状态。不同故障其恢复策略和方法也不一样。

7.5.1 事务故障的恢复

事务故障是指事务在运行至正常终止点前被终止，这时恢复子系统应利用日志文件撤销（UNDO）此事务已对数据库进行的修改。事务故障的恢复是由系统自动完成的，对用户是透明的。系统的恢复步骤是：

（1）反向扫描文件日志（即从最后向前扫描日志文件），查找该事务的更新操作。

（2）对该事务的更新操作执行逆操作。即将日志记录中“更新前的值”写入数据库。这样，如果记录中是插入操作，则相当于做删除操作（因此时“更新前的值”为空）；若记录中是删除操作，则做插入操作；若是修改操作，则相当于用修改前值代替修改后值。

（3）继续反向扫描日志文件，查找该事务的其他更新操作，并做同样处理。

（4）如此处理下去，直至读到此事务的开始标记，事务故障恢复就完成了。

7.5.2 系统故障的恢复

前面已讲过，系统故障造成数据库不一致状态的原因有两个，一是未完成事务对数据库的更新可能已写入数据库，二是已提交事务对数据库的更新可能还留在缓冲区没来得及写入数据库。因此恢复操作就是要撤销故障发生时未完成的事务，重做已完成的事务。

系统故障的恢复是由系统在重新启动时自动完成的，不需要用户干预。

系统的恢复步骤是：

（1）正向扫描日志文件（即从头扫描日志文件），找出在故障发生前已经提交的事务(这些事务既有 BEGIN TRANSACTION 记录，也有 COMMIT 记录)，将其事务标识记入重做(REDO)队列。同时找出故障发生时尚未完成的事务(这些事务只有 BEGIN TRANSACTION 记录，无相应的 COMMIT 记录)，将其事务标识记入撤销（UNDO）队列。

（2）对撤销队列中的各个事务进行撤销(UNDO)处理。

进行 UNDO 处理的方法是，反向扫描日志文件，对每个 UNDO 事务的更新操作执行逆操作，即将日志记录中“更新前的值”写入数据库。

（3）对重做队列中的各个事务进行重做(REDO)处理。

进行 REDO 处理的方法是：正向扫描日志文件，对每个 REDO 事务重新执行日志文件登记的操作。即将日志记录中“更新后的值”写入数据库。

7.5.3 介质故障的恢复

发生介质故障后，磁盘上的物理数据和日志文件被破坏，这是最严重的一种故障，恢复方法是重装数据库，然后重做已完成的事务。具体地说就是：

1. 装入最新的数据库后备副本（离故障发生时刻最近的转储副本），使数据库恢复到最近一次转储时的一致性状态。

对于动态转储的数据库副本，还需同时装入转储开始时刻的日志文件副本，利用恢复系统故障的方法（即 REDO+UNDO），才能将数据库恢复到一致性状态。

2. 装入相应的日志文件副本（转储结束时刻的日志文件副本），重做已完成的事务。即首先扫描日志文件，找出故障发生时已提交的事务的标识，将其记入重做队列。然后正向扫描日志文件，对重做队列中的所有事务进行重做处理。即将日志记录中“更新后的值”写入数据库。

这样就可以将数据库恢复至故障前某一时刻的一致状态了。

介质故障的恢复需要 DBA 介入。但 DBA 只需要重装最近转储的数据库副本和有关的各日志文件副本，然后执行系统提供的恢复命令即可，具体的恢复操作仍由 DBMS 完成。

7.6 具有检查点的恢复技术

利用日志技术进行数据库恢复时，恢复子系统必须搜索日志，确定哪些事务需要 REDO，哪些事务需要 UNDO。一般来说，需要检查所有日志记录。这样做具有两个问题。一是搜索整个日志将耗费大量的时间。二是很多需要 REDO 处理的事务实际上已经将它们的更新操作结果写到数据库中了，然而恢复子系统又重新执行了这些操作，浪费了大量时间。为了解决这些问题，又发展了具有检查点的恢复技术。这种技术在日志文件中增加一类新的记录——检查点（checkpoint）记录，增加一个重新开始文件，并让恢复子系统在登录日

志文件期间动态地维护日志。

检查点记录的内容包括：

① 建立检查点时刻所有正在执行的事务清单。

② 这些事务最近一个日志记录的地址。

重新开始文件用来记录各个检查点记录在日志文件中的地址。图 7.3 说明了建立检查点 C_i 时对应的日志文件和重新开始文件。

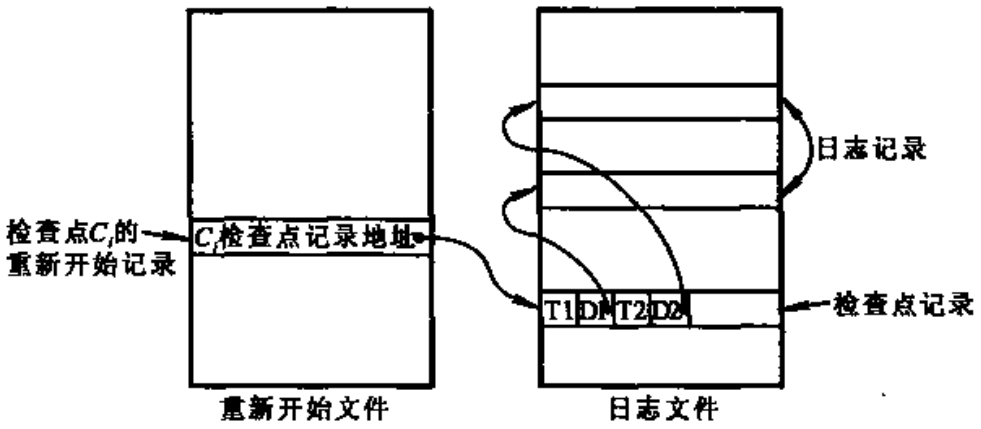


图 7.3 具有检查点的日志文件和重新开始文件

动态维护日志文件的方法是周期性地执行如下操作：建立检查点，保存数据库状态。具体步骤是：

① 将当前日志缓冲中的所有日志记录写入磁盘的日志文件上；

② 在日志文件中写入一个检查点记录；

③ 将当前数据缓冲的所有数据记录写入磁盘的数据库中；

④ 把检查点记录在日志文件中的地址写入一个重新开始文件。

恢复子系统可以定期或不定期地建立检查点保存数据库状态。检查点可以按照预定的一个时间间隔建立，如每隔一小时建立一个检查点；也可以按照某种规则建立检查点，如日志文件已写满一半建立一个检查点。

使用检查点方法可以改善恢复效率。当事务 T 在一个检查点之前提交，T 对数据库所做的修改一定都已写入数据库，写入时间是在这个检查点建立之前或在这个检查点建立之时。这样，在进行恢复处理时，没有必要对事务 T 执行 REDO 操作。

系统出现故障时恢复子系统将根据事务的不同状态采取不同的恢复策略，如图 7.4 所示。

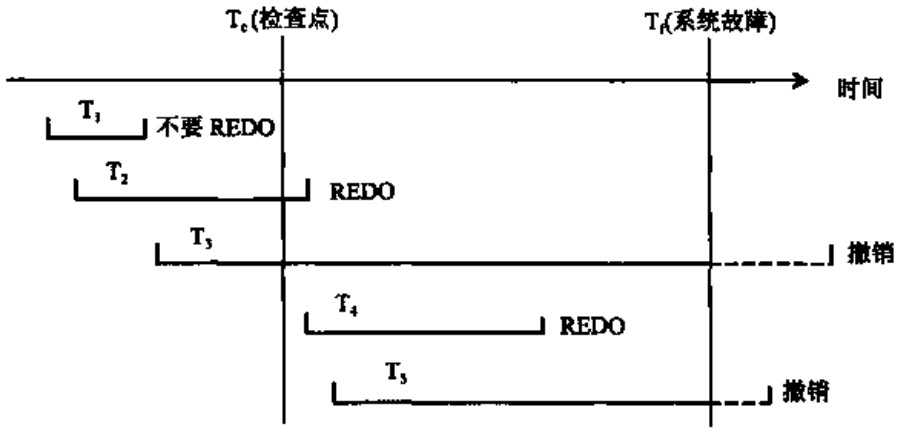


图 7.4 恢复子系统采取的不同策略

T_1：在检查点之前提交。

T_2：在检查点之前开始执行，在检查点之后故障点之前提交。

T_3：在检查点之前开始执行，在故障点时还未完成。

T_4：在检查点之后开始执行，在故障点之前提交。

T_5：在检查点之后开始执行，在故障点时还未完成。

T_3和T_5在故障发生时还未完成，所以予以撤销；T_2和T_4在检查点之后才提交，它们对数据库所做的修改在故障发生时可能还在缓冲区中，尚未写入数据库，所以要 REDO；T_1在检查点之前已提交，所以不必执行 REDO 操作。

系统使用检查点方法进行恢复的步骤是：

① 从重新开始文件中找到最后一个检查点记录在日志文件中的地址，由该地址在日志文件中找到最后一个检查点记录。

② 由该检查点记录得到检查点建立时刻所有正在执行的事务清单 ACTIVE-LIST。

这里建立两个事务队列：

- UNDO-LIST: 需要执行 UNDO 操作的事务集合；
- REDO-LIST: 需要执行 REDO 操作的事务集合；

把 ACTIVE-LIST 暂时放入 UNDO-LIST 队列，REDO 队列暂为空。

③ 从检查点开始正向扫描日志文件

- 如有新开始的事务 T_i，把 T_i暂时放入 UNDO-LIST 队列；
- 如有提交的事务 T_j，把 T_j从 UNDO-LIST 队列移到 REDO-LIST 队列；

直到日志文件结束。

④ 对 UNDO-LIST 中的每个事务执行 UNDO 操作，对 REDO-LIST 中的每个事务执行 REDO 操作。

7.7 数据库镜像

根据前面所述，介质故障是对系统影响最为严重的一种故障。系统出现介质故障后，用户应用全部中断，恢复起来也比较费时。而且 DBA 必须周期性地转储数据库，这也加重了 DBA 的负担。如果不及时而正确地转储数据库，一旦发生介质故障，会造成较大的损失。

随着磁盘容量越来越大，价格越来越便宜，为避免磁盘介质出现故障影响数据库的可用性，许多数据库管理系统提供了数据库镜像（Mirror）功能用于数据库恢复。即根据 DBA 的要求，自动把整个数据库或其中的关键数据复制到另一个磁盘上。每当主数据库更新时，DBMS 自动把更新后的数据复制过去，即 DBMS 自动保证镜像数据与主数据的一致性(如图 7.5(a)所示)。这样，一旦出现介质故障，可由镜像磁盘继续提供使用，同时 DBMS 自动利用镜像磁盘数据进行数据库的恢复，不需要关闭系统和重装数据库副本(如图 7.5(b)所示)。在没有出现故障时，数据库镜像还可以用于并发操作，即当一个用户对数据加排它锁修改数据时，其他用户可以读镜像数据库上的数据，而不必等待该用户释放锁。

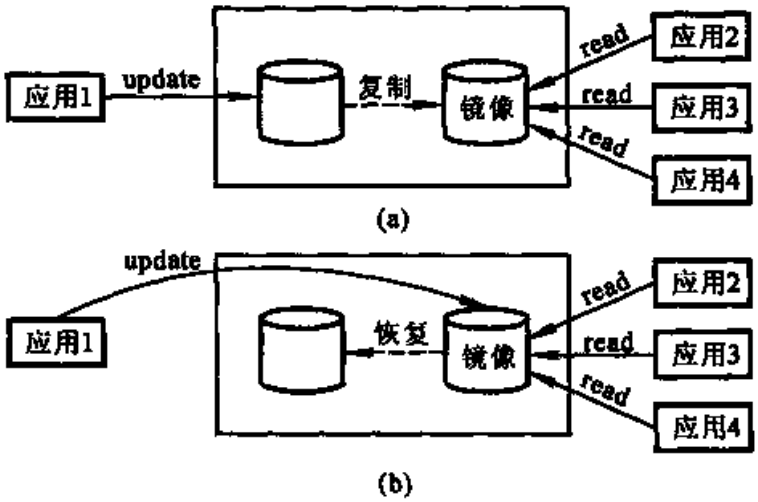


图 7.5 数据库镜像

由于数据库镜像是通过复制数据实现的，频繁地复制数据自然会降低系统运行效率，因此在实际应用中用户往往只选择对关键数据和日志文件镜像，而不是对整个数据库进行镜像。

7.8 Oracle 的恢复技术

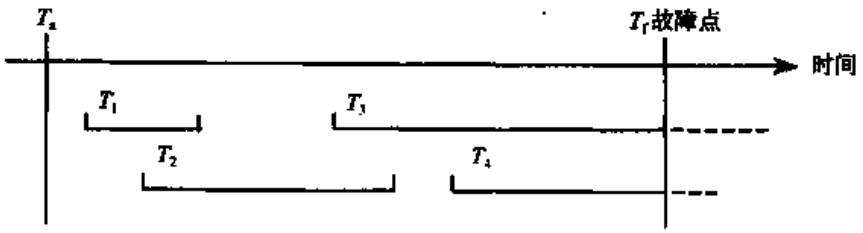
上面介绍了恢复技术的一般原理，实际 DBMS 产品中的恢复策略往往都有自己的特色。下面简单介绍一下 Oracle 的恢复技术，以加深对数据库恢复技术的理解。

Oracle 中恢复机制也采用了转储和登记日志文件两个技术。

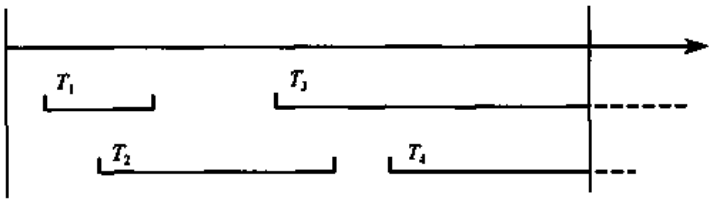
Oracle 向 DBA 提供了多种转储后备副本的方法，如文件拷贝、利用 Oracle 的 Export 实用程序、用 SQL 命令 Spool 以及自己编程实现等。相应地，Oracle 也提供了多种重装后备副本的方法，如文件拷贝、利用 Oracle 的 Import 实用程序、利用 SQL*LOADER 以及自己编程实现等。

在 Oracle 早期版本（V.5）中，日志文件以数据块为单位，也就是说，Oracle 的恢复操作不是基于操作，而是基于数据块的。Oracle 将更新前的旧值与更新后的新值分别放在两个不同的日志文件中。记录数据库更新前的旧值的日志文件称为数据库前像文件（Before Image，简称 BI 文件），记录数据库更新后的新值的日志文件称为数据库的后像文件（After Image，简称 AI 文件）。由于 BI 文件关系到能否将数据库恢复到一致性状态，因此 BI 文件是必须的。而 AI 文件的作用是减少必须重新运行的事务程序，尽可能多地恢复数据库，所以在 Oracle 中 AI 文件是任选的。日志文件是十分占据空间的，为节省存储空间和操作时间，DBA 可以不配置 AI 文件。没有 AI 文件，恢复机制进行故障恢复时只能执行 UNDO 处理，不能执行 REDO 处理。

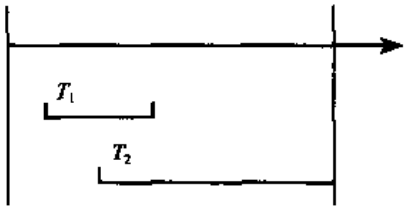
Oracle7 为了能够在出现故障时更有效地恢复数据，也为了解决读“脏”数据问题（参阅第八章），提供了 REDO 日志文件和回滚段(Rollback Segment)。REDO 日志文件中记录了被更新数据的前像和后像，设在数据库缓冲区中的回滚段记录更新数据的前像。在利用日志文件进行故障恢复时，为减少扫描日志文件的遍数，Oracle7 首先扫描 REDO 日志文件，重做所有操作，包括未正常提交的事务的操作，然后再根据回滚段中的数据，撤销未正常提交的事务的操作，如图 7.6 所示。



(a) 发生故障，事务非正常终止



(b) 利用 REDO 文件，重做所有操作



(c) 利用回滚段撤销未提交的事务数据库恢复到一致性状态

图 7.6　Oracle 的恢复过程

7.9　小　　结

保证数据一致性是对数据库的最基本的要求。事务是数据库的逻辑工作单位，只要 DBMS 能够保证系统中一切事务的原子性、一致性、隔离性和持续性，也就保证了数据库处于一致状态。为了保证事务的原子性、一致性与持续性，DBMS 必须对事务故障、系统故障和介质故障进行恢复。数据库转储和登记日志文件是恢复中最经常使用的技术。恢复的基本原理就是利用存储在后备副本、日志文件和数据库镜像中的冗余数据来重建数据库。

事务不仅是恢复的基本单位，也是并发控制的基本单位，为了保证事务的隔离性，保证事务的一致性，DBMS 需要对并发操作进行控制。

习　　题

1. 试述事务的概念及事务的四个特性。

2. 为什么事务非正常结束时会影响数据库数据的正确性，请列举一例说明之。

3. 数据库中为什么要有恢复子系统？它的功能是什么？

4. 数据库运行中可能产生的故障有哪几类？哪些故障影响事务的正常执行？哪些故障破坏数据库数据？

5. 数据库恢复的基本技术有哪些？

6. 数据库转储的意义是什么？ 试比较各种数据转储方法。

7. 什么是日志文件？为什么要设立日志文件？

8. 登记日志文件时为什么必须先写日志文件，后写数据库？

9. 针对不同的故障，试给出恢复的策略和方法。（即如何进行事务故障的恢复？系统故障的恢复？介质故障恢复？）

10. 什么是检查点记录，检查点记录包括哪些内容？

11. 具有检查点的恢复技术有什么优点？试举一个具体的例子加以说明。

12. 试述使用检查点方法进行恢复的步骤。

13. 什么是数据库镜像？它有什么用途？

*14. 试述你了解的某一个实际的 DBMS 产品中采用的恢复策略。

*15. 试用恢复的基本技术设计一个恢复子系统，给出这个子系统的恢复策略，包括：

(a) 当产生某一类故障时如何恢复数据库的方法；

(b) 日志文件的结构；

(c) 登记日志文件的方法；

(d) 利用日志文件恢复事务的方法；

(e) 转储的类型；

(f) 转储的后备副本和日志文件如何配合使用。

本章参考文献

1 Davies，Jr C T. Recovery Semantics for a DB/DC System. Proceedings of the ACM Annual Conference，1973

2 Bjork L A. Recovery Scenario for a DB/DC System. Proceedings of the ACM Annual Conference. 1973

（文献[1]和[2]是系统恢复领域中两篇最早的论文。）

3 Theo Haerder and Andreas Reuter. Principles of Transaction-Oriented Database Recovey. ACM Computing Surveys，Vol. 15，No.4，December 1983

4 Gray Jim，et al. The Recovery Manager of the System R Data Manager. ACM Computing Surveys，Vol.13，No.2，June 1981

（本文概述了 IBM 的数据库管理系统(实验系统)System R 的恢复技术。）

5 Crus R A. Data Recovery in IBM Database 2. IBM Sys，J.23，No.2，1984

（本文介绍了 IBM DB2 的恢复技术。）

6 Lorie R A. Physical Integrity in a Large Segmented Database. ACM Transactions on Database Systems，Volume 2，Number 1，March 1977

（本文研究了 System R 的影子页面技术。）

7 Chandy K M，Browne J C，Dissley C W，et al. Analytic Models for Rollback and Recovery

Strategies in Database Systems. IEEE Transactions on Software Engineering, Vol.SE-1, Number 1, March 1975

（本文提出了数据库系统恢复和回滚的分析模型 。）

8 Reuter A. A Fast Transaction-Oriented Logging Scheme for UNDO Recovery. IEEE Transactions on Software Engineering, Volume SE-6, Number 4, July 1980

（本文给出了一个快速处理 UNDO 操作的日志模式。）

9 Lilien L, Bhargava B. Database Integrity Block Construct: Concept and Design Issues. IEEE Transactions on Software Engineering, Vol.SE-11, No.9, 1985

（本文介绍了完整性、并发控制和系统恢复相结合的问题。）

10 Verhofstad J S M. Recovery Techniques for Database Systems. ACM Computing Surveys, Volume 10, Number 2, June 1978

（本文系统地综述了数据库管理系统的恢复技术。）

11 Bernstein A, Hadzilacos V, Goodman W. Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987

（本书包含了有关 DBMS 恢复的全面介绍。）

12 Mohan C, Haderle D, Lindsay R et al. ARIES: A Transaction Recovery Method Supporting Fine Granularity Locking and Partial Rollback Using Write-Ahead Logging. ACM Transactions on Database Systems, 1991.

（本文讨论了 ARIES 系统中事务恢复技术。）

13 Mohan C. Commit-LSN: A Novel and Simple Method for Reducing Locking and Latching in Transaction Processing Systems. Proceedings of the International Conference on Very Large Data Bases, 1990

14 Korth H F, Speegle G. Long Duration Transactions in Software Design Projects. Proceedings of the International Conference on Data Engineering, 1990

（本文讨论了长事务的问题。）

15 Korth H F, Levy E. Silberschatz A. A Formal Approach to Recovery by Compensating Transactions. Proceedings of the International Conference on Very Large Data Bases, 1990

（本文讨论了长事务的恢复问题。）

第八章 并发控制

数据库是一个共享资源，可以供多个用户使用。允许多个用户同时使用的数据库系统称为多用户数据库系统。例如飞机定票数据库系统、银行数据库系统等都是多用户数据库系统。在这样的系统中，在同一时刻并行运行的事务数可达数百个。

事务可以一个一个地串行执行，即每个时刻只有一个事务运行，其他事务必须等到这个事务结束以后方能运行。事务在执行过程中需要不同的资源，有时需要 CPU，有时需要存取数据库，有时需要 I/O，有时需要通信。如果事务串行执行，则许多系统资源将处于空闲状态。因此，为了充分利用系统资源，发挥数据库共享资源的特点，应该允许多个事务并行地执行。

在单处理机系统中，事务的并行执行实际上是这些并行事务的并行操作轮流交叉运行。这种并行执行方式称为交叉并发方式（Interleaved Concurrency）。虽然单处理机系统中的并行事务并没有真正地并行运行，但是减少了处理机的空闲时间，提高了系统的效率。

在多处理机系统中，每个处理机可以运行一个事务，多个处理机可以同时运行多个事务，实现多个事务真正的并行运行。这种并行执行方式称为同时并发方式（Simultaneous Concurrency）。本章讨论的数据库系统并发控制技术是以单处理机系统为基础的。这些理论可以推广到多处理机的情况。

当多个用户并发地存取数据库时就会产生多个事务同时存取同一数据的情况。若对并发操作不加控制就可能会存取和存储不正确的数据，破坏数据库的一致性。所以数据库管理系统必须提供并发控制机制。并发控制机制是衡量一个数据库管理系统性能的重要标志之一。

8.1 并发控制概述

在第七章中已经讲到，事务是并发控制的基本单位，保证事务 ACID 特性是事务处理的重要任务，而事务 ACID 特性可能遭到破坏的原因之一是多个事务对数据库的并发操作造成的。为了保证事务的隔离性更一般，为了保证数据库的一致性，DBMS 需要对并发操作进行正确调度。这些就是数据库管理系

统中并发控制机制的责任。

下面先来看一个例子，说明并发操作带来的数据的不一致性问题。

考虑飞机订票系统中的一个活动序列:

① 甲售票点（甲事务）读出某航班的机票余额 A，设 A=16;

② 乙售票点（乙事务）读出同一航班的机票余额 A，也为 16;

③ 甲售票点卖出一张机票，修改余额 A←A-1，所以 A 为 15，把 A 写回数据库;

④ 乙售票点也卖出一张机票，修改余额 A←A-1，所以 A 为 15，把 A 写回数据库。

结果明明卖出两张机票，数据库中机票余额只减少 1。

这种情况称为数据库的不一致性。这种不一致性是由并发操作引起的。在并发操作情况下，对甲、乙两个事务的操作序列的调度是随机的。若按上面的调度序列执行，甲事务的修改就被丢失。这是由于第④步中乙事务修改 A 并写回后覆盖了甲事务的修改。

仔细分析并发操作带来的数据不一致性包括三类：丢失修改、不可重复读和读“脏”数据，如图 8.1 所示。

1. 丢失修改（Lost Update）

两个事务 T_1 和 T_2 读入同一数据并修改，T_2 提交的结果破坏了 T_1 提交的结果，导致 T_1 的修改被丢失，如图 8.1(a)所示。上面飞机订票例子就属此类。

T_1	T_2
①读 A=16	
②	读 A=16
③A←A-1	
写回 A=15	
④	A←A-1
	写回 A=15

(a) 丢失修改

T_1	T_2
①读 A=50	
读 B=100	
求和=150	
②	读 B=100
	B←B*2
	写回 B=200
③读 A=50	
读 B=200	
和=250	
(验算不对)	

(b) 不可重复读

T_1	T_2
①读 C=100	
C←C*2	
写回 C	
②	读 C=200
③ROLLBACK	
C 恢复为 100	

(c) 读“脏”数据

图 8.1 三种数据不一致性

2. 不可重复读（Non-Repeatable Read）

不可重复读是指事务 T_1 读取数据后，事务 T_2 执行更新操作，使 T_1 无法再现前一次读取结果。具体地讲，不可重复读包括三种情况：

（1）事务 T_1 读取某一数据后，事务 T_2 对其做了修改，当事务 T_1 再次读该数据时，得到与前一次不同的值。例如在图 8.1(b)中，T_1 读取 B=100 进行运算，T_2 读取同一数据 B，对其进行修改后将 B=200 写回数据库。T_1 为了对读取值校对重读 B，B 已为 200，与第 1 次读取值不一致。

（2）事务 T_1 按一定条件从数据库中读取了某些数据记录后，事务 T_2 删除了其中部分记录，当 T_1 再次按相同条件读取数据时，发现某些记录神秘地消失了。

（3）事务 T_1 按一定条件从数据库中读取某些数据记录后，事务 T_2 插入了一些记录，当 T_1 再次按相同条件读取数据时，发现多了一些记录。

后两种不可重复读有时也称为幻影（Phantom Row）现象。

3. 读“脏”数据（Dirty Read）

读“脏”数据是指事务 T_1 修改某一数据，并将其写回磁盘，事务 T_2 读取同一数据后，T_1 由于某种原因被撤销，这时 T_1 已修改过的数据恢复原值，T_2 读到的数据就与数据库中的数据不一致，则 T_2 读到的数据就为“脏”数据，即不正确的数据。例如在图 8.1(c)中 T_1 将 C 值修改为 200，T_2 读到 C 为 200，而 T_1 由于某种原因撤销，其修改作废，C 恢复原值 100，这时 T_2 读到的 C 为 200，与数据库内容不一致就是“脏”数据。

产生上述三类数据不一致性的主要原因是并发操作破坏了事务的隔离性。并发控制就是要用正确的方式调度并发操作，使一个用户事务的执行不受其他事务的干扰，从而避免造成数据的不一致性。

另一方面，对数据库的应用有时允许某些不一致性，例如有些统计工作涉及数据量很大，读到一些“脏”数据对统计精度没什么影响，这时可以降低对一致性的要求以减少系统开销。

并发控制的主要技术是封锁（Locking）。例如在飞机订票例子中，甲事务要修改 A，若在读出 A 前先锁住 A，其他事务就不能再读取和修改 A 了，直到甲修改并写回 A 后解除了对 A 的封锁为止。这样，就不会丢失甲的修改。

8.2 封 锁 (Locking)

封锁是实现并发控制的一个非常重要的技术。所谓封锁就是事务 T 在对某个数据对象例如表、记录等操作之前，先向系统发出请求，对其加锁。加锁后事务 T 就对该数据对象有了一定的控制，在事务 T 释放它的锁之前，其他的事务不能更新此数据对象。

确切的控制由封锁的类型决定。基本的封锁类型有两种：排它锁(Exclusive

Locks，简称 X 锁) 和共享锁(Share Locks，简称 S 锁)。

排它锁又称为写锁。若事务 T 对数据对象 A 加上 X 锁，则只允许 T 读取和修改 A，其他任何事务都不能再对 A 加任何类型的锁，直到 T 释放 A 上的锁。这就保证了其他事务在 T 释放 A 上的锁之前不能再读取和修改 A。

共享锁又称为读锁。若事务 T 对数据对象 A 加上 S 锁，则事务 T 可以读 A 但不能修改 A，其他事务只能再对 A 加 S 锁，而不能加 X 锁，直到 T 释放 A 上的 S 锁。这就保证了其他事务可以读 A，但在 T 释放 A 上的 S 锁之前不能对 A 做任何修改。

排它锁与共享锁的控制方式可以用图 8.2 的相容矩阵来表示。

T_1 \ T_2	X	S	-
X	N	N	Y
S	N	Y	Y
-	Y	Y	Y

Y=Yes，相容的请求

N=No，不相容的请求

图 8.2 封锁类型的相容矩阵

在图 8.2 的封锁类型相容矩阵中，最左边一列表示事务 T_1 已经获得的数据对象上的锁的类型，其中横线表示没有加锁。最上面一行表示另一事务 T_2 对同一数据对象发出的封锁请求。T_2 的封锁请求能否被满足用矩阵中的 Y 和 N 表示，其中 Y 表示事务 T_2 的封锁要求与 T_1 已持有的锁相容，封锁请求可以满足。N 表示 T_2 的封锁请求与 T_1 已持有的锁冲突，T_2 的请求被拒绝。

8.3 封 锁 协 议

在运用 X 锁和 S 锁这两种基本封锁，对数据对象加锁时，还需要约定一些规则，例如何时申请 X 锁或 S 锁、持锁时间、何时释放等。称这些规则为封锁协议（Locking Protocol）。对封锁方式规定不同的规则，就形成了各种不同的封锁协议。下面介绍三级封锁协议。对并发操作的不正确调度可能会带来丢失修改、不可重复读和读“脏”数据等不一致性问题，三级封锁协议分别在不同程度上解决了这一问题。为并发操作的正确调度提供一定的保证。不同级别的封锁协议达到的系统一致性级别是不同的。

一、一级封锁协议

一级封锁协议是：事务 T 在修改数据 R 之前必须先对其加 X 锁，直到事

务结束才释放。事务结束包括正常结束（COMMIT）和非正常结 ROLLBACK）。

一级封锁协议可防止丢失修改，并保证事务 T 是可恢复的。例如图 8.3(a)使用一级封锁协议解决了图 8.1(a)中的丢失修改问题。

图 8.3(a)事务 T_1 在读 A 进行修改之前先对 A 加 X 锁，当 T_2 再请求对 A 加 X 锁时被拒绝，T_2 只能等待 T_1 释放 A 上的锁后 T_2 获得对 A 的 X 锁，这时它读到的 A 已经是 T_1 更新过的值 15，再按此新的 A 值进行运算，并将结果值 A=14 送回到磁盘。这样就避免了丢失 T_1 的更新。

在一级封锁协议中，如果仅仅是读数据不对其进行修改，是不需要加锁的，所以它不能保证可重复读和不读“脏”数据。

二、二级封锁协议

二级封锁协议是：一级封锁协议加上事务 T 在读取数据 R 之前必须先对其加 S 锁，读完后即可释放 S 锁。

二级封锁协议除防止了丢失修改，还可进一步防止读“脏”数据。例如图 8.3(c)使用二级封锁协议解决了图 8.1(c)中的读“脏”数据问题。

图 8.3(c)中，事务 T_1 在对 C 进行修改之前，先对 C 加 X 锁，修改其值后写回磁盘。这时 T_2 请求在 C 上加 S 锁，因 T_1 已在 C 上加了 X 锁，T_2 只能等待。T_1 因某种原因被撤销，C 恢复为原值 100，T_1 释放 C 上的 X 锁后 T_2 获得 C 上的 S 锁，读 C=100。这就避免了 T_2 读“脏”数据。

在二级封锁协议中，由于读完数据后即可释放 S 锁，所以它不能保证可重复读。

三、三级封锁协议

三级封锁协议是：一级封锁协议加上事务 T 在读取数据 R 之前必须先对其加 S 锁，直到事务结束才释放。

三级封锁协议除防止了丢失修改和不读“脏”数据外，还进一步防止了不可重复读。例如图 8.3(b)使用三级封锁协议解决了图 8.1(b)不可重复读问题。

图 8.3(b)中，事务 T_1 在读 A，B 之前，先对 A，B 加 S 锁，这样其它事务只能再对 A，B 加 S 锁，而不能加 X 锁，即其他事务只能读 A，B，而不能修改它们。所以当 T_2 为修改 B 而申请对 B 的 X 锁时被拒绝只能等待 T_1 释放 B 上的锁。T_1 为验算再读 A，B，这时读出的 B 仍是 100，求和结果仍为 150，即可重复读。T_1 结束才释放 A，B 上的 S 锁。T_2 才获得对 B 的 X 锁。

上述三级协议的主要区别在于什么操作需要申请封锁，以及何时释放锁（即持锁时间）。三个级别的封锁协议可以总结为表 8.1。

T_1	T_2
① Xlock A	
获得	
② 读 A=16	
	Xlock A
③A←A-1	等待
写回 A=15	等待
Commit	等待
Unlock A	等待
④	获得 Xlock A
	读 A=15
	A←A-1
⑤	写回 A=14
	Commit
	Unlock A

(a) 没有丢失修改

T_1	T_2
①Slock A	
Slock B	
读 A=50	
读 B=100	
求和=150	
②	Xlock B
	等待
	等待
③读 A=50	等待
读 B=100	等待
求和=150	等待
Commit	等待
Unlock A	等待
Unlock B	等待
④	获得 Xlock B
	读 B=100
	B←B*2
⑤	写回 B=200
	Commit
	Unlock B

(b) 可重复读

T_1	T_2
①Xlock C	
读 C= 100	
C←C*2	
写回 C=200	
②	Slock C
	等待
③ROLLBACK	等待
(C 恢复为 100)	等待
Unlock C	等待
④	获得 Slock C
	读 C=100
⑤	Commit C
	Unlock C

(c) 不读“脏”数据

图 8.3 用封锁机制解决三种数据不一致性的示例

表 8.1 不同级别的封锁协议

	X 锁		S 锁		一致性保证		
	操作结束释放	事务结束释放	操作结束释放	事务结束释放	不丢失修改	不读“脏”数据	可重复读
一级封锁协议		√			√		
二级封锁协议		√	√		√	√	
三级封锁协议		√		√	√	√	√

8.4 活锁和死锁

和操作系统一样，封锁的方法可能引起活锁和死锁。

一、活锁

如果事务 T_1 封锁了数据 R，事务 T_2 又请求封锁 R，于是 T_2 等待。T_3 也请求封锁 R，当 T_1 释放了 R 上的封锁之后系统首先批准了 T_3 的请求，T_2 仍然等待。然后 T_4 又请求封锁 R，当 T_3 释放了 R 上的封锁之后系统又批准了 T_4 的请求……T_2 有可能永远等待，这就是活锁的情形，如图 8.4(a)所示。

T_1	T_2	T_3	T_4
lock R			
⋮	lock R		
	等待	Lock R	
Unlock	等待		Lock R
⋮	等待	Lock R	等待
	等待		等待
	等待	Unlock	等待
	等待		Lock R
	等待		

(a) 活锁

T_1	T_2
lock R_1	
	Lock R_2
	⋮
Lock R_2	
等待	
等待	Lock R_1
等待	等待
等待	等待

(b) 死锁

图 8.4 活锁与死锁

避免活锁的简单方法是采用先来先服务的策略。当多个事务请求封锁同一数据对象时，封锁子系统按请求封锁的先后次序对事务排队，数据对象上的锁一旦释放就批准申请队列中第 1 个事务获得锁。

二、死锁

如果事务 T_1 封锁了数据 R_1，T_2 封锁了数据 R_2，然后 T_1 又请求封锁 R_2，因 T_2 已封锁了 R_2，于是 T_1 等待 T_2 释放 R_2 上的锁。接着 T_2 又申请封锁 R_1，因 T_1 已封锁了 R_1，T_2 也只能等待 T_1 释放 R_1 上的锁。这样就出现了 T_1 在等待 T_2，而 T_2 又在等待 T_1 的局面，T_1 和 T_2 两个事务永远不能结束，形成死锁。如

图 8.4(b)所示。

死锁的问题在操作系统和一般并行处理中已做了深入研究，目前在数据库中解决死锁问题主要有两类方法，一类方法是采取一定措施来预防死锁的发生，另一类方法是允许发生死锁，采用一定手段定期诊断系统中有无死锁，若有则解除之。

1. 死锁的预防

在数据库中，产生死锁的原因是两个或多个事务都已封锁了一些数据对象，然后又都请求对已为其他事务封锁的数据对象加锁，从而出现死等待。防止死锁的发生其实就是要破坏产生死锁的条件。预防死锁通常有两种方法：

（1）一次封锁法

一次封锁法要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行。如图 8.4(b)的例子中，如果事务 T_1 将数据对象 R_1 和 R_2 一次加锁，T_1 就可以执行下去，而 T_2 等待。T_1 执行完后释放 R_1，R_2 上的锁，T_2 继续执行。这样就不会发生死锁。

一次封锁法虽然可以有效地防止死锁的发生，但也存在问题。第一，一次就将以后要用到的全部数据加锁，势必扩大了封锁的范围，从而降低了系统的并发度。第二，数据库中数据是不断变化的，原来不要求封锁的数据，在执行过程中可能会变成封锁对象，所以很难事先精确地确定每个事务所要封锁的数据对象，为此只能扩大封锁范围，将事务在执行过程中可能要封锁的数据对象全部加锁，这就进一步降低了并发度。

（2）顺序封锁法

顺序封锁法是预先对数据对象规定一个封锁顺序，所有事务都按这个顺序实行封锁。例如在 B 树结构的索引中，可规定封锁的顺序必须是从根结点开始，然后是下一级的子女结点，逐级封锁。

顺序封锁法可以有效地防止死锁，但也同样存在问题。第一，数据库系统中封锁的数据对象极多，并且随数据的插入、删除等操作而不断地变化，要维护这样的资源的封锁顺序非常困难，成本很高。第二，事务的封锁请求可以随着事务的执行而动态地决定，很难事先确定每一个事务要封锁哪些对象，因此也就很难按规定的顺序去施加封锁。

可见，在操作系统中广为采用的预防死锁的策略并不很适合数据库的特点，因此 DBMS 在解决死锁的问题上普遍采用的是诊断并解除死锁的方法。

2. 死锁的诊断与解除

数据库系统中诊断死锁的方法与操作系统类似，一般使用超时法或事务等待图法。

（1）超时法

如果一个事务的等待时间超过了规定的时限，就认为发生了死锁。超时法实现简单，但其不足也很明显。一是有可能误判死锁，事务因为其他原因使等待时间超过时限，系统会误认为发生了死锁。二是时限若设置得太长，死锁发生后不能及时发现。

（2）等待图法

事务等待图是一个有向图 $G=(T, U)$。T 为结点的集合，每个结点表示正运行的事务；U 为边的集合，每条边表示事务等待的情况。若 T_1 等待 T_2，则 T_1，T_2 之间划一条有向边，从 T_1 指向 T_2。事务等待图动态地反映了所有事务的等待情况。并发控制子系统周期性地（比如每隔 1 min）检测事务等待图，如果发现图中存在回路，则表示系统中出现了死锁。

DBMS 的并发控制子系统一旦检测到系统中存在死锁，就要设法解除。通常采用的方法是选择一个处理死锁代价最小的事务，将其撤销，释放此事务持有的所有的锁，使其他事务得以继续运行下去。当然，对撤销的事务所执行的数据修改操作必须加以恢复。

8.5 并发调度的可串行性

计算机系统对并发事务中并发操作的调度是随机的，而不同的调度可能会产生不同的结果，那么哪个结果是正确的，哪个是不正确的呢？

如果一个事务运行过程中没有其他事务同时运行，也就是说它没有受到其他事务的干扰，那么就可以认为该事务的运行结果是正常的或者预想的。因此将所有事务串行起来的调度策略一定是正确的调度策略。虽然以不同的顺序串行执行事务可能会产生不同的结果，但由于不会将数据库置于不一致状态，所以都是正确的。

定义 多个事务的并发执行是正确的，当且仅当其结果与按某一次序串行地执行它们时的结果相同，我们称这种调度策略为**可串行化**（Serializable）的调度。

可串行性（Serializability）是并发事务正确性的准则。按这个准则规定，一个给定的并发调度，当且仅当它是可串行化的，才认为是正确调度。

例如，现在有两个事务，分别包含下列操作：

事务 T_1：读 B；A=B+1；写回 A；

事务 T_2：读 A；B=A+1；写回 B；

假设 A，B 的初值均为 2。按 $T_1 \rightarrow T_2$ 次序执行结果为 A=3，B=4；按 $T_2 \rightarrow T_1$ 次序执行结果为 B=3，A=4。

图 8.5 给出了对这两个事务的三种不同的调度策略。

图 8.5(a)和(b)为两种不同的串行调度策略，虽然执行结果不同，但它们都是正确的调度；图 8.5(c)中两个事务是交错执行的，由于其执行结果与(a)，(b)的结果都不同，所以是错误的调度；图 8.5(d)中两个事务也是交错执行的，其执行结果与串行调度(a)的执行结果相同，所以是正确的调度。

T_1	T_2
Slock B	
Y=B=2	
Unlock B	
Xlock A	
A=Y+1	
写回 A(=3)	
Unlock A	
	Slock A
	X=A=3
	Unlock A
	Xlock B
	B=X+1
	写回 B(=4)
	Unlock B

(a) 串行调度

T_1	T_2
	SlockA
	X=A=2
	Unlock A
	Xlock B
	B=X+1
	写回 B(=3)
	Unlock B
Slock B	
Y=B=3	
Unlock B	
Xlock A	
A=Y+1	
写回 A(=4)	
Unlock A	

(b) 串行调度

T_1	T_2
Slock B	
Y=B=2	
	Slock A
	X=A=2
Unlock B	
	Unlock A
Xlock A	
A=Y+1	
写回 A(=3)	
	Xlock B
	B=X+1
	写回 B(=3)
Unlock A	
	Unlock B

(c) 不可串行化的调度

T_1	T_2
Slock B	
Y=B=2	
Unlock B	
Xlock A	
	Slock A
A=Y+1	等待
写回 A(=3)	等待
Unlock A	等待
	X=A=3
	Unlock A
	Xlock B
	B=X+1
	写回 B(=4)
	Unlock B

(d) 可串行化的调度

图 8.5 并发事务的不同调度

为了保证并发操作的正确性，DBMS 的并发控制机制必须提供一定的手段来保证调度是可串行化的。

从理论上讲，在某一事务执行时禁止其他事务执行的调度策略一定是可串行化的调度，这也是最简单的调度策略，但这种方法实际上是不可取的，这使用户不能充分共享数据库资源。目前 DBMS 普遍采用封锁方法实现并发操作调度的可串行性，从而保证调度的正确性。

两段锁（Two-Phase Locking，简称 2PL）协议就是保证并发调度可串行性的封锁协议。

除此之外还有其他一些方法，如时标方法、乐观方法等来保证调度的正确性。

8.6 两段锁协议

所谓两段锁协议是指所有事务必须分两个阶段对数据项加锁和解锁。

- 在对任何数据进行读、写操作之前，首先要申请并获得对该数据的封锁；
- 在释放一个封锁之后，事务不再申请和获得任何其他封锁。

所谓“两段”锁的含义是，事务分为两个阶段，第一阶段是获得封锁，也称为扩展阶段。在这阶段，事务可以申请获得任何数据项上的任何类型的锁，但是不能释放任何锁。第二阶段是释放封锁，也称为收缩阶段。在这阶段，事务可以释放任何数据项上的任何类型的锁，但是不能再申请任何锁。

例如事务 T_1 遵守两段锁协议，其封锁序列是：

Slock A　Slock B　Xlock C　Unlock B　Unlock A　Unlock C；

|←——　扩展阶段　——→|　|←——　收缩阶段　——→|

又如事务 T_2 不遵守两段锁协议，其封锁序列是：

Slock A　Unlock A　Slock B　Xlock C　Unlock C　Unlock B；

可以证明，若并发执行的所有事务均遵守两段锁协议，则对这些事务的任何并发调度策略都是可串行化的。

需要说明的是，事务遵守两段锁协议是可串行化调度的充分条件，而不是必要条件。也就是说，若并发事务都遵守两段锁协议，则对这些事务的任何并发调度策略都是可串行化的；若对并发事务的一个调度是可串行化的，不一定所有事务都符合两段锁协议。

在图 8.6 中，(a)和(b)都是可串行化的调度，但(a)中 T_1 和 T_2 都遵守两段锁协议，(b)中 T_1 和 T_2 不遵守两段协议。又如图 8.5 中（d）是可串行化的调度，但 T_1 和 T_2 也不遵守两段协议。

另外要注意两段锁协议和防止死锁的一次封锁法的异同之处。一次封锁法要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行，因此一次封锁法遵守两段锁协议；但是两段锁协议并不要求事务必须一次将所有要使用的数据全部加锁，因此遵守两段锁协议的事务可能发生死锁，如图 8.7 所示。

T_1	T_2
Slock B	
读 B=2	
Y=B	
Xlock A	
	Slock A
	等待
A=Y+1	等待
写回 A=3	等待
Unlock B	等待
Unlock A	等待
	Slock A
	读 A=3
	Y=A
	Xlock B
	B=Y+1
	写回 B=4
	Unlock B
	Unlock A

(a) 遵守两段锁协议

T_1	T_2
Slock B	
读 B=2	
Y=B	
Unlock B	
Xlock A	
	Slock A
	等待
A=Y+1	等待
写回 A=3	等待
Unlock A	等待
	Slock A
	读 A=3
	X=A
	Unlock A
	Xlock B
	B=X+1
	写回 B=4
	Unlock B

(b) 不遵守两段锁协议

图 8.6 可串行化调度

T_1	T_2
Slock B	
读 B=2	
	Slock A
	读 A=2
Xlock A	
等待	Xlock B
等待	等待

图 8.7 遵守两段锁协议的事务发生死锁

8.7 封锁的粒度

封锁对象的大小称为封锁粒度（Granularity）。封锁对象可以是逻辑单元，也可以是物理单元。以关系数据库为例，封锁对象可以是这样一些逻辑单元：属性值、属性值的集合、元组、关系、索引项、整个索引直至整个数据库；也可以是这样一些物理单元:页(数据页或索引页)、块等。

封锁粒度与系统的并发度和并发控制的开销密切相关。直观地看，封锁的粒度越大，数据库所能够封锁的数据单元就越少，并发度就越小，系统开销也越小；反之，封锁的粒度越小，并发度较高，但系统开销也就越大。

例如，若封锁粒度是数据页，事务 T_1 需要修改元组 L_1，则 T_1 必须对包含 L_1 的整个数据页 A 加锁。如果 T_1 对 A 加锁后事务 T_2 要修改 A 中元组 L_2，则 T_2 被迫等待，直到 T_1 释放 A。如果封锁粒度是元组，则 T_1 和 T_2 可以同时对 L_1 和 L_2 加锁，不需要互相等待，提高了系统的并行度。又如，事务 T 需要读取整个表，若封锁粒度是元组，T 必须对表中的每一个元组加锁，显然开销极大。

因此， 如果在一个系统中同时支持多种封锁粒度供不同的事务选择是比较理想的，这种封锁方法称为多粒度封锁（Multiple Granularity Locking）。选择封锁粒度时应该同时考虑封锁开销和并发度两个因素，适当选择封锁粒度以求得最优的效果。一般说来，需要处理大量元组的事务可以以关系为封锁粒度；需要处理多个关系的大量元组的事务可以以数据库为封锁粒度；而对于一个处理少量元组的用户事务，以元组为封锁粒度就比较合适了。

8.7.1 多粒度封锁

下面讨论多粒度封锁，首先定义多粒度树。多粒度树的根结点是整个数据库，表示最大的数据粒度。叶结点表示最小的数据粒度。图 8.8 给出了一个三级粒度树。根结点为数据库，数据库的子结点为关系，关系的子结点为元组。

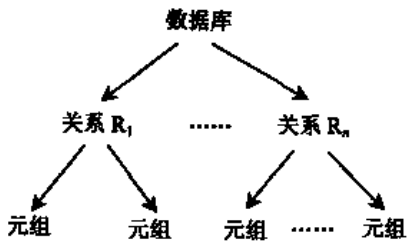


图 8.8 三级粒度树

然后，来讨论多粒度封锁的封锁协议。多粒度封锁协议允许多粒度树中的每个结点被独立地加锁。对一个结点加锁意味着这个结点的所有后裔结点也被加以同样类型的锁。因此，在多粒度封锁中一个数据对象可能以两种方式封锁，显式封锁和隐式封锁。

显式封锁是应事务的要求直接加到数据对象上的封锁；隐式封锁是该数据对象没有独立加锁，是由于其上级结点加锁而使该数据对象加上了锁。

多粒度封锁方法中，显式封锁和隐式封锁的效果是一样的，因此系统检查封锁冲突时不仅要检查显式封锁还要检查隐式封锁。例如事务 T 要对关系 R_1 加 X 锁。系统必须搜索其上级结点数据库、关系 R_1 以及 R_1 中的每一个元组，如果其中某一个数据对象已经加了不相容锁，则 T 必须等待。

一般地，对某个数据对象加锁，系统要检查该数据对象上有无显式封锁与之冲突；还要检查其所有上级结点，看本事务的显式封锁是否与该数据对象上的隐式封锁（即由于上级结点已加的封锁造成的）冲突；还要检查其所有下级结点，看上面的显式封锁是否与本事务的隐式封锁（将加到下级结点的封锁）冲突。显然，这样的检查方法效率很低。为此人们引进了一种新型锁，称为意向锁（Intention Lock）。

8.7.2 意向锁

意向锁的含义是如果对一个结点加意向锁，则说明该结点的下层结点正在被加锁；对任一结点加锁时，必须先对它的上层结点加意向锁。

例如，对任一元组加锁时，必须先对它所在的关系加意向锁。

于是，事务 T 要对关系 R_1 加 X 锁时，系统只要检查根结点数据库和关系 R_1 是否已加了不相容的锁，而不再需要搜索和检查 R_1 中的每一个元组是否加了 X 锁。

下面介绍三种常用的意向锁：意向共享锁(Intent Share Lock，简称 IS 锁)；意向排它锁(Intent Exclusive Lock，简称 IX 锁)；共享意向排它锁(Share Intent Exclusive Lock，简称 SIX 锁)。

1. IS 锁

如果对一个数据对象加 IS 锁，表示它的后裔结点拟（意向）加 S 锁。例如，要对某个元组加 S 锁，则要首先对关系和数据库加 IS 锁。

2. IX 锁

如果对一个数据对象加 IX 锁，表示它的后裔结点拟（意向）加 X 锁。例如，要对某个元组加 X 锁，则要首先对关系和数据库加 IX 锁。

3. SIX 锁

如果对一个数据对象加 SIX 锁，表示对它加 S 锁，再加 IX 锁，即 SIX = S

+IX。例如对某个表加 SIX 锁，则表示该事务要读整个表（所以要对该表加 S 锁），同时会更新个别元组（所以要对该表加 IX 锁）。

图 8.9（a）给出了这些锁的相容矩阵，从中可以发现这 5 种锁的强度如图 8.9（b）所示的偏序关系。所谓锁的强度是指它对其他锁的排斥程度。一个事务在申请封锁时以强锁代替弱锁是安全的，反之则不然。

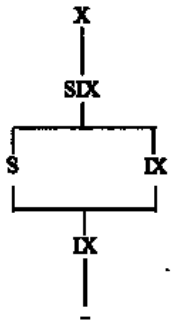
具有意向锁的多粒度封锁方法中任意事务 T 要对一个数据对象加锁，必须先对它的上层结点加意向锁。申请封锁时应该按自上而下的次序进行；释放封锁时则应该按自下而上的次序进行。

具有意向锁的多粒度封锁方法提高了系统的并发度，减少了加锁和解锁的开销，它已经在实际的数据库管理系统产品中得到广泛应用，例如新版的 Oracle 数据库系统就采用了这种封锁方法。

T_1 \ T_2	S	X	IS	IX	SIX	-
S	Y	N	Y	N	N	Y
X	N	N	N	N	N	Y
IS	Y	N	Y	Y	Y	Y
IX	N	N	Y	Y	N	Y
SIX	N	N	Y	N	N	Y
-	Y	Y	Y	Y	Y	Y

Y=Yes，表示相容的请求　　N=No，表示不相容的请求

(a) 数据锁的相容矩阵



(b) 锁的强度的偏序关系

图 8.9　锁的相容矩阵

8.8　Oracle 的并发控制

前面讨论了并发控制的一般原则与方法，下面简单介绍 Oracle 数据库系统中的并发控制机制。

Oracle 采用封锁技术保证并发操作的可串行性。Oracle 的锁分为两大类：数据锁（亦称 DML 锁）和字典锁。字典锁是 Oracle DBMS 内部用于对字典表的封锁。字典锁包括语法分析锁和 DDL 锁，由 DBMS 在必要的时候自动加锁和释放锁，用户无权控制。

Oracle 主要提供了 5 种数据锁：共享锁（S 锁）、排它锁（X 锁）、行级共享锁（RS 锁）、行级排它锁（RX 锁）和共享行级排它锁（SRX 锁）。其封锁粒度包括行级和表级。数据锁的相容矩阵如图 8.10 所示。

T_1 \ T_2	S	X	RS	RX	SRX	-
S	Y	N	Y	N	N	Y
X	N	N	N	N	N	Y
RS	Y	N	Y	Y	Y	Y
RX	N	N	Y	Y	N	Y
SRX	N	N	Y	N	N	Y
-	Y	Y	Y	Y	Y	Y

Y=Yes，表示相容的请求

N=No，表示不相容的请求

图 8.10 Oracle 数据锁的相容矩阵

可以看出 Oracle 的 RS 锁、RX 锁、SRX 锁实际上就是上面介绍的 IS 锁、IX 锁、SIX 锁。

在通常情况下，数据封锁由系统控制，对用户是透明的。但 Oracle 也允许用户用 LOCK TABLE 语句显式对封锁对象加锁。

Oracle 数据锁的一个显著特点是，在缺省情况下，读数据不加锁。也就是说，当一个用户更新数据时，另一个用户可以同时读取相应数据，反之亦然。Oracle 通过回滚段（Rollback Segment）的内存结构来保证用户不读“脏”数据和可重复读。这样做的好处是提高了数据的并发度。

Oracle 提供了有效的死锁检测机制，周期性诊断系统中有无死锁；若存在死锁，则撤销执行更新操作次数最少的事务。

8.9 小 结

数据库的重要特征是它能为多个用户提供数据共享。数据库管理系统允许共享的用户数目是数据库管理系统重要标志之一。数据库管理系统必须提供并发控制机制来协调并发用户的并发操作以保证并发事务的隔离性，保证数据库的一致性。

数据库的并发控制以事务为单位，通常使用封锁技术实现并发控制。本章介绍了两类最常用的封锁和三级封锁协议。不同的封锁和不同级别的封锁协议所提供的系统一致性保证是不同的，提供数据共享度也是不同的。

并发控制机制调度并发事务操作是否正确的判别准则是可串行性，两段锁协议是可串行化调度的充分条件，但不是必要条件。因此，两段锁协议可以保证并发事务调度的正确性。

对数据对象施加封锁，会带来活锁和死锁问题，并发控制机制必须提供适

合数据库特点的解决方法。

不同的数据库管理系统提供的封锁类型、封锁协议、达到的系统一致性级别不尽相同。但是其依据的基本原理和技术是共同的。

习 题

1. 在数据库中为什么要并发控制?

2. 并发操作可能会产生哪几类数据不一致?用什么方法能避免各种不一致的情况?

3. 什么是封锁?

4. 基本的封锁类型有几种?试述它们的含义。

5. 如何用封锁机制保证数据的一致性?

6. 什么是封锁协议?不同级别的封锁协议的主要区别是什么?

7. 不同封锁协议与系统一致性级别的关系是什么?

8. 什么是 活锁?什么是死锁?

9. 试述活锁的产生原因和解决方法。

10. 请给出预防死锁的若干方法。

11. 请给出检测死锁发生的一种方法,当发生死锁后如何解除死锁?

12. 什么样的并发调度是正确的调度?

13. 设 T_1, T_2, T_3 是如下的三个事务:

T_1: A: =A+2;

T_2: A: =A*2;

T_3: A: =A**2; (A ← A^2)

设 A 的初值为 0;

(1) 若这三个事务允许并发执行,则有多少种可能的正确结果,请一一列举出来;

(2) 请给出一个可串行化的调度,并给出执行结果;

(3) 请给出一个非串行化的调度,并给出执行结果;

(4) 若这三个事务都遵守两段锁协议,请给出一个不产生死锁的可串行化调度;

(5) 若这三个事务都遵守两段锁协议,请给出一个产生死锁的调度。

14. 试述两段锁协议的概念。

15. 试证明,若并发事务遵守两段锁协议,则对这些事务的并发调度是可串行化的。

16. 举例说明,对并发事务的一个调度是可串行化的,而这些并发事务不一定遵守两段锁协议。

17. 为什么要引进意向锁?意向锁的含义是什么?

18. 试述常用的意向锁:IS 锁、IX 锁、SIX 锁、给出这些锁的相容矩阵。

19. 理解并解释下列术语的含义:封锁、活锁、死锁、排它锁、共享锁、并发事务的调度、可串行化的调度、两段锁协议。

*20. 试述你了解的某一个实际的 DBMS 产品的并发控制机制。

本章参考文献

1 Bernstein P A. Concurrency Control and Recovery In Database Systems. 1986

2 Eswaran K P, Gray J N, Lorie R A, et al. The Notions of Consistency and Predicate Locks in a Data Base System. CACM 19, No.11, November 1976

（本文给出了可串行性概念的形式化定义。）

3 Gray J N, Lorie R A, Putzolu G R. Granularity of Locks in a Large Shared Data Base. in Proceedings of VLDB, September 1975

4 Papadimitriou C H, Bernstein P A, Rothnie J B. Some Computational Problems Related to Database Concurrency Control. Proceedings of the Conference on Theoretical Compute Science, 1977

5 Papadimitriou C H. The Serializability of Concurrent Database Updates. Journal of the ACM, Volume 26, Number 4, October 1979

（本文给出了与可串行性相关的研究结果。）

6 Soisalon-Soininen E, Wood D. An Optimal Algorithm for Testing Safety and Detecting Deadlocks. Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, 1982

7 Yannakakis M, Papadimitriou C H, Kung H T. Locking Protocols: Safety and Freedom from Deadlock. Proceedings of the IEEE Symposium on the Foundations of Computer Science, 1979

（本文讨论了封锁协议。）

8 Kedem Z M, Silberschatz A. Locking Protocols: From Exclusive to Shared Locks. Journal of the ACM, Volume 30, Number 4, 1983

（本文讨论封锁协议。）

9 Korth H F. Locking Primitives in a Database System. Journal of the ACM, Volume 30, Number 1, January 1983

（本文探讨了从基本共享锁和独占锁模型获得的各种锁模型。）

10 Papadimitriou C H. A Theorem in Database Concurrency Control. Journal of the ACM, Volume 29, Number 5, 1982

11 Bayer R, Schkolnick M. Concurrency of Operating on B-trees. Acys Informatica, Volume 9, Number 1, 1977

（本文提出了对 B-树进行并发存取的算法。）

12 Ellis C S. Concurrent Search And Insertion in 2-3 Trees. Acta Informatica, Volume 14, 1980

（本文提出了对 2-3 树进行并发存取的算法。）

13 Ellis C S. Concurrent Search And Insertion in AVL Trees. IEEE Transactions on Computers, Volume E-29, Number 3, September 1980

（本文提出了对 AVL 树进行并发存取的算法。）

14 Lehman P L, Yao S B. Efficient Locking for Concurrent Operations on B-trees. ACM Transactions on Database Systems, Volume 6, Number 4, 1981

15 Manber U, Ladner R E. Concurrency Control in a Dynamic Search Structure. ACM Transactions on Database Systems, Volume 9, Number 3, 1984

（本文讨论了对动态搜索树进行并发存取的各种算法。）

16 Buckley G, Silberschatz A. Concurrency Control in Graph Protocols by Using Edge Locks. Proceedings of the ACM SIGACT-SIGMOD Symposium on the Principles of Database Systems, 1984

（本文给出了扩展的封锁协议。）

17 Bernstein P A, Goodman N. Timestamp-based Algorithms for Concurrency Control in Distributed Database Systems. Proceedings of the International Conference on Very Large Data Bases, 1980

（本文讨论了各种基于时间戳的并发控制算法。）

18 Gray J N, Lorie R A, Putzolu G R. Granularity of Locks and Degrees of Consistency in a Shared Data Base. in Proceedings of VLDB, 1975

（本文综述了多粒度封锁协议。）

19 Korth H F. Deadlock Freedom Using Edge Locks. ACM Transactions on Database Systems, Volume 7, Number 4, December 1982

第九章　数据库安全性

在第一章中已经讲到数据库的特点之一是 DBMS 提供统一的数据保护功能来保证数据的安全可靠和正确有效，数据库的数据保护主要包括数据的安全性和数据的完整性。这一章讨论数据的安全性，下一章讨论数据的完整性。

9.1　计算机安全性概论

数据库的安全性是指保护数据库以防止不合法的使用所造成的数据泄露、更改或破坏。

安全性问题不是数据库系统所独有的，所有计算机系统都有这个问题。只是在数据库系统中大量数据集中存放，而且为许多最终用户直接共享，从而使安全性问题更为突出。系统安全保护措施是否有效是数据库系统的主要指标之一。

数据库的安全性和计算机系统的安全性，包括操作系统、网络系统的安全性是紧密联系、相互支持的，因此在讨论数据库的安全性之前首先讨论计算机系统安全性的一般问题。

9.1.1　计算机系统的三类安全性问题

所谓计算机系统安全性，是指为计算机系统建立和采取的各种安全保护措施，以保护计算机系统中的硬件、软件及数据，防止其因偶然或恶意的原因使系统遭到破坏，数据遭到更改或泄露等。计算机安全不仅涉及到计算机系统本身的技术问题、管理问题，还涉及法学、犯罪学、心理学的问题。其内容包括了计算机安全理论与策略；计算机安全技术、安全管理、安全评价、安全产品以及计算机犯罪与侦察、计算机安全法律、安全监察等。概括起来，计算机系统的安全性问题可分为三大类，即技术安全类、管理安全类和政策法律类。

技术安全是指计算机系统中采用具有一定安全性的硬件、软件来实现对计算机系统及其所存数据的安全保护，当计算机系统受到无意或恶意的攻击时仍能保证系统正常运行，保证系统内的数据不增加、不丢失、不泄露。技术安全之外的，诸如软硬件意外故障、场地的意外事故、管理不善导致的计算机设备

和数据介质的物理破坏、丢失等安全问题，视为管理安全。而政策法律类则指政府部门建立的有关计算机犯罪、数据安全保密的法律道德准则和政策法规、法令，本书只讨论技术安全类。

随着计算机资源共享和网络技术的应用日益广泛和深入，特别是 Internet 技术的发展，计算机安全性问题越来越得到人们的重视。对各种计算机及其相关产品、信息系统的安全性要求越来越高。为此，在计算机安全技术方面逐步发展建立了一套可信（Trusted）计算机系统的概念和标准。只有建立了完善的可信或安全标准，才能规范和指导安全计算机系统部件的生产，比较准确地测定产品的安全性能指标，满足民用和国防的需要。

9.1.2 可信计算机系统评测标准

为降低进而消除对系统的安全攻击，尤其是弥补原有系统在安全保护方面的缺陷，在计算机安全技术方面逐步建立了一套可信标准。在目前各国所引用或制定的一系列安全标准中，最重要的当推 1985 年美国国防部（DoD）正式颁布的《DoD 可信计算机系统评估标准》（Trusted Computer System Evaluation Criteria，简称 TCSEC 或 DoD85）。

制定这个标准的目的主要有：

（1）提供一种标准，使用户可以对其计算机系统内敏感信息安全操作的可信程度做评估。

（2）给计算机行业的制造商提供一种可循的指导规则，使其产品能够更好地满足敏感应用的安全需求。

TCSEC 又称桔皮书，1991 年 4 月美国 NCSC（国家计算机安全中心）颁布了《可信计算机系统评估标准关于可信数据库系统的解释》（Trusted Database Interpretation，简称 TDI，即紫皮书）。将 TCSEC 扩展到数据库管理系统。TDI 中定义了数据库管理系统的设计与实现中需满足和用以进行安全性级别评估的标准。

以下着重介绍 TDI/TCSEC 标准的基本内容。

TDI 与 TCSEC 一样，从以下四个方面来描述安全性级别划分的指标：安全策略、责任、保证和文档。每个方面又细分为若干项。这些指标的具体内容如下：

- R1 安全策略（Security Policy）
  - R1.1 自主存取控制（Discretionary Access Control，简称 DAC）
  - R1.2 客体重用（Object Reuse）
  - R1.3 标记（Labels）
    - R1.3.1 标记完整性（Label Integrity）

R1.3.2 标记信息的扩散（Labeled Information Exploration）
R1.3.3 主体敏感度标记（Subject Sensitivity Labels）
R1.3.4 设备标记（Device Labels）
R1.4 强制存取控制（Mandatory Access Control，简称 MAC）
• R2 责任（Accountability）
R2.1 标识与鉴别（Identification & Authentication）
R2.1.1 可信路径（Trusted Path）
R2.2 审计（Audit）
• R3 保证（Assurance）
R3.1 操作保证（Operational Assurance）
R3.1.1 系统体系结构（System Architecture）
R3.1.2 系统完整性（System Integrity）
R3.1.3 隐蔽信道分析（Covert Channel Analysis）
R3.1.4 可信设施管理（Trusted Facility Management）
R3.1.5 可信恢复（Trusted Recovery）
R3.2 生命周期保证（Life Cycle Assurance）
R3.2.1 安全测试（Security Testing）
R3.2.2 设计规范和验证（Design Specification & Verification）
R3.2.3 配置管理（Configuration Management）
R3.2.4 可信分配（Trusted Distribution）
• R4 文档（Documentation）
R4.1 安全特性用户指南（Security Features User's Guide）
R4.2 可信设施手册（Trusted Facility Manual）
R4.3 测试文档（Test Documentation）
R4.4 设计文档（Design Documentation）

根据计算机系统对上述各项指标的支持情况，TCSEC（TDI）将系统划分为四组(division)七个等级，依次是 D；C（C1，C2）；B（B1，B2，B3）；A（A1），按系统可靠或可信程度逐渐增高，如表 9.1 所示。

在 TCSEC 中建立的安全级别之间具有一种偏序向下兼容的关系，即较高安全性级别提供的安全保护要包含较低级别的所有保护要求，同时提供更多或更完善的保护能力。

下面，简略地对各个等级作一介绍。

D 级 D 级是最低级别。保留 D 级的目的是为了将一切不符合更高标准的系统，统统归于 D 组。如 DOS 就是操作系统中安全标准为 D 的典型例子。它具有操作系统的基本功能，如文件系统，进程调度等，但在安全性方面几乎

没有什么专门的机制来保障。

表 9.1 TCSEC/TDI 安全级别划分

安全级别	定义
A1	验证设计（Verified Design）
B3	安全域（Security Domains）
B2	结构化保护（Structural Protection）
B1	标记安全保护（Labeled Security Protection）
C2	受控的存取保护（Controlled Access Protection）
C1	自主安全保护（Discretionary Security Protection）
D	最小保护（Minimal Protection）

C1 级 只提供了非常初级的自主安全保护。能够实现对用户和数据的分离，进行自主存取控制（DAC），保护或限制用户权限的传播。现有的商业系统往往稍作改进即可满足要求。

C2 级 实际是安全产品的最低档次，提供受控的存取保护，即将 C1 级的 DAC 进一步细化，以个人身份注册负责，并实施审计和资源隔离。很多商业产品已得到该级别的认证。达到 C2 级的产品在其名称中往往不突出“安全”(Security)这一特色，如操作系统中 Microsoft 的 Windows NT 3.5，数字设备公司的 Open VMS VAX 6.0 和 6.1。数据库产品有 Oracle 公司的 Oracle 7，Sybase 公司的 SQL Server 11.0.6 等。

B1 级 标记安全保护。对系统的数据加以标记，并对标记的主体和客体实施强制存取控制（MAC）以及审计等安全机制。B1 级能够较好地满足大型企业或一般政府部门对于数据的安全需求，这一级别的产品才认为是真正意义上的安全产品。满足此级别的产品前一般多冠以“安全”(Security)或“可信的”(Trusted)字样，作为区别于普通产品的安全产品出售。例如，操作系统方面，典型的有数字设备公司的 SEVMS VAX Version 6.0，惠普公司的 HP-UX BLS release 9.0.9+。 数据库方面则有 Oracle 公司的 Trusted Oracle 7， Sybase 公司的 Secure SQL Server version 11.0.6，Informix 公司的 Incorporated INFORMIX-OnLine / Secure 5.0 等。

B2 级 结构化保护。建立形式化的安全策略模型并对系统内的所有主体和客体实施 DAC 和 MAC。

从互联网上的最新资料[2]看，经过认证的、B2 级以上的安全系统非常稀少。例如，符合 B2 标准的操作系统只有 Trusted Information Systems 公司的 Trusted XENIX 一种产品，符合 B2 标准的网络产品只有 Cryptek Secure Communications 公司的 LLC VSLAN 一种产品，而数据库方面则没有符合 B2

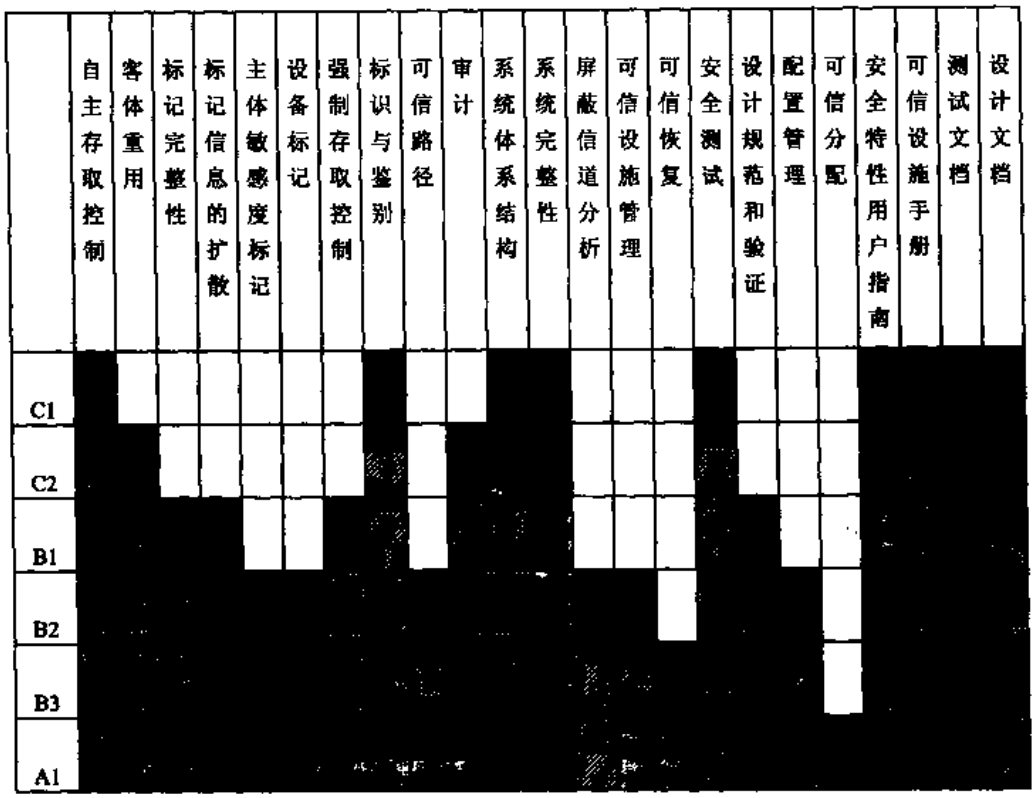
标准的产品。

B3 级 安全域。该级的 TCB 必须满足访问监控器的要求，审计跟踪能力更强，并提供系统恢复过程。

A1 级 验证设计，即提供 B3 级保护的同时给出系统的形式化设计说明和验证以确信各安全保护真正实现。

表 9.2 列出了各安全等级对安全指标的支持情况。随着安全级别的由低到高，系统对各项安全指标的支持也从无到有。其中，□表示该级不提供对该指标的支持；■表示该级新增的对该指标的支持；▩表示该级对该指标的支持与相邻低一级的等级一样；▨表示该级对该指标的支持较下一级有所增加或改动。

表 9.2 不同安全级别对安全指标的支持情况



B2 级以上的系统标准更多地还处于理论研究阶段，产品化以至商品化的程度都不高，其应用也多限于一些特殊的部门，如军队等。但美国正在大力发展安全产品，试图将目前仅限于少数领域应用的 B2 级安全级别或更高安全级别下放到商业应用中来，并逐步成为新的商业标准。

有关各安全等级对安全策略、责任、保证及文档四个方面安全指标的支持情况这里就不详细展开了，有兴趣的读者请参阅[1]。

可以看出，支持自主存取控制的 DBMS 大致属于 C 级，而支持强制存取

控制的 DBMS 则可以达到 B1 级。当然，存取控制仅是安全性标准的一个重要方面（即安全策略方面），不是全部。为了使 DBMS 达到一定的安全级别，还需要在其他三个方面提供相应的支持。例如审计功能就是 DBMS 达到 C2 以上安全级别必不可少的一项指标。

9.2 数据库安全性控制

在一般计算机系统中，安全措施是一级一级层层设置的。例如可以有如下的模型：

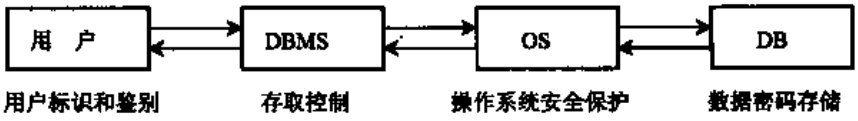


图 9.1 计算机系统的安全模型

在图 9.1 的安全模型中，用户要求进入计算机系统时，系统首先根据输入的用户标识进行用户身份鉴定，只有合法的用户才准许进入计算机系统。对已进入系统的用户，DBMS 还要进行存取控制，只允许用户执行合法操作。操作系统一级也会有自己的保护措施。数据最后还可以以密码形式存储到数据库中。操作系统一级的安全保护措施可参考操作系统的有关书籍，这里不再详叙。另外对于强力逼迫透露口令、盗窃物理存储设备等行为而采取的保安措施，例如出入机房登记、加锁等，也不在这里讨论之列。

这里只讨论与数据库有关的用户标识和鉴定、存取控制、视图和密码存储等安全技术。

9.2.1 用户标识与鉴别（Identification & Authentication）

用户标识和鉴别是系统提供的最外层安全保护措施。其方法是由系统提供一定的方式让用户标识自己的名字或身份。每次用户要求进入系统时，由系统进行核对，通过鉴定后才提供机器使用权。

对于获得上机权的用户若要使用数据库时数据库管理系统还要进行用户标识和鉴定。

用户标识和鉴定的方法有很多种，而且在一个系统中往往是多种方法并举，以获得更强的安全性。常用的方法有：

用一个用户名或者用户标识号来标明用户身份。系统内部记录着所有合法用户的标识，系统鉴别此用户是否是合法用户，若是，则可以进入下一步的核

实；若不是，则不能使用系统。

口令（Password） 为了进一步核实用户，系统常常要求用户输入口令。为保密起见，用户在终端上输入的口令不显示在屏幕上。系统核对口令以鉴别用户身份。

通过用户名和口令来鉴定用户的方法简单易行，但用户名与口令容易被人窃取，因此还可以用更复杂的方法。例如每个用户都预先约定好一个计算过程或者函数，鉴别用户身份时，系统提供一个随机数，用户根据自己预先约定的计算过程或者函数进行计算，系统根据用户计算结果是否正确进一步鉴定用户身份。用户可以约定比较简单的计算过程或函数，以便计算起来方便；也可以约定比较复杂的计算过程或函数，以便安全性更好。

用户标识和鉴定可以重复多次。

9.2.2 存取控制

数据库安全性所关心的主要是 DBMS 的存取控制机制。数据库安全最重要的一点就是确保只授权给有资格的用户访问数据库的权限，同时令所有未被授权的人员无法接近数据，这主要通过数据库系统的存取控制机制实现。

存取控制机制主要包括两部分：

1. 定义用户权限，并将用户权限登记到数据字典中。用户权限是指不同的用户对于不同的数据对象允许执行的操作权限。系统必须提供适当的语言定义用户权限，这些定义经过编译后存放在数据字典中，被称作安全规则或授权规则。

2. 合法权限检查，每当用户发出存取数据库的操作请求后（请求一般应包括操作类型、操作对象和操作用户等信息），DBMS 查找数据字典，根据安全规则进行合法权限检查，若用户的操作请求超出了定义的权限，系统将拒绝执行此操作。

用户权限定义和合法权检查机制一起组成了 DBMS 的安全子系统。

前面已经讲到，当前大型的 DBMS 一般都支持 C2 级中的自主存取控制（DAC），有些 DBMS 同时还支持 B1 级中的强制存取控制（MAC）。

这两类方法的简单定义是：

（1）在**自主存取控制**方法中，用户对于不同的数据对象有不同的存取权限，不同的用户对同一对象也有不同的权限，而且用户还可将其拥有的存取权限转授给其他用户。因此自主存取控制非常灵活。

（2）在**强制存取控制**方法中，每一个数据对象被标以一定的密级，每一个用户也被授予某一个级别的许可证。对于任意一个对象，只有具有合法许可证的用户才可以存取。强制存取控制因此相对比较严格。

下面介绍这两种存取控制方法。

9.2.3 自主存取控制（DAC）方法

大型数据库管理系统几乎都支持自主存取控制，目前的 SQL 标准也对自主存取控制提供支持，这主要通过 SQL 的 GRANT 语句和 REVOKE 语句来实现。

用户权限是由两个要素组成的：数据对象和操作类型。定义一个用户的存取权限就是要定义这个用户可以在哪些数据对象上进行哪些类型的操作。在数据库系统中，定义存取权限称为授权（Authorization）。

在非关系系统中，用户只能对数据进行操作，存取控制的数据对象也仅限于数据本身。而关系数据库系统中，DBA 可以把建立、修改基本表的权限授予用户，用户获得此权限后可以建立和修改基本表、索引、视图。因此，关系系统中存取控制的数据对象不仅有数据本身，如表、属性列等，还有模式、外模式、内模式等数据字典中的内容，如表 9.3 所示。

表 9.3 关系系统中的存取权限

	数据对象	操作类型
模 式	模 式	建立、修改、检索
	外模式	建立、修改、检索
	内模式	建立、修改、检索
数 据	表	查找、插入、修改、删除
	属性列	查找、插入、修改、删除

在第三章 3.6 节中讨论了 SQL 语言的安全性控制功能，即 SQL 的 GRANT 语句和 REVOKE 语句，这里简单回顾一下。首先，数据对象的创建者自动获得对于该数据对象的所有操作权限。例如，表 Student 的创建者自动获得对该表的 SELECT，INSERT，UPDATE 和 DELETE 等权限，这些权限可以通过 GRANT 语句转授给其他用户。例如：

```
GRANT SELECT, INSERT ON Student
TO 王平
WITH GRANT OPTION;
```

就将表 Student 的 SELECT 和 INSERT 权限授予了用户王平，后面的“WITH GRANT OPTION”子句表示用户王平同时也获得了“授权”的权限，即可以把得到的权限继续授予其他用户。

当用户将某些权限授给其他用户后，他可以使用 REVOKE 语句将权限收

回，例如：

REVOKE INSERT ON Student FROM 王平 CASCADE;

就将表 Student 的 INSERT 权限从用户王平处收回，选项 CASCADE 表示，如果用户王平将 Student 的 INSERT 权限又转授给了其他用户，那么这些权限也将从其他用户处收回。

例如，在学生-课程数据库中一个可能的用户权限定义如表 9.4 所示。

表 9.4 学生-课程数据库中的用户权限定义表

用户名	数据对象名	允许的操作类型
王 平	关系 Student	SELECT
张明霞	关系 Student	UPDATE
张明霞	关系 Course	ALL
张明霞	SC. Grade	UPDATE
张明霞	SC. Sno	SELECT
张明霞	SC. Cno	SELECT

用户权限定义中数据对象范围越小授权子系统就越灵活。例如上面的授权定义可精细到字段级，而有的系统只能对关系授权。授权粒度越细，授权子系统就越灵活，但系统定义与检查权限的开销也会相应地增大。

衡量授权子系统精巧程度的另一个尺度是能否提供与数据值有关的授权。上面的授权定义是独立于数据值的，即用户能否对某类数据对象执行的操作与数据值无关，完全由数据名决定。如用户王平能检索关系 Student，不管学生是哪个系的。反之，若授权依赖于数据对象的内容，则称为是与数据值有关的授权。例如，用户王平只能检索计算机系学生的信息。这就要求系统能支持存取谓词。例如，把表 9.4 的用户权限定义表扩充为表 9.5。

表 9.5 支持存取谓词的用户权限定义

用户名	数据对象名	允许的操作类型	存取谓词
王平	关系 Student	SELECT	Sdept='CS'
张明霞	关系 Student	UPDATE	Sname='张明霞'
张明霞	关系 Course	ALL	空

用户张明霞只能更新自己的数据，张明霞可以在 Course 表上执行全部操作（检索、更新、插入、修改），并且存取谓词为空。

另外，还可以在存取谓词中引用系统变量，如一天的某个时刻或某段时

间，某台终端设备号，系统时钟等，这就是与时间地点有关的存取权限，这样用户只能在某段时间内，某台终端上存取有关数据。例如规定“教师只能在每周星期一至星期五上午 8 点到下午 5 点处理学生成绩数据”。

自主存取控制能够通过授权机制有效地控制其他用户对敏感数据的存取。但是由于用户对数据的存取权限是“自主”的，用户可以自由地决定将数据的存取权限授予何人、决定是否也将“授权”的权限授予别人，而系统对此无法控制。在这种授权机制下，仍可能存在数据的“无意泄露”。比如，甲将自己权限范围内的某些数据存取权限授权给乙，甲的意图是只允许乙本人操纵这些数据。但甲的这种安全性要求并不能得到保证，因为乙一旦获得了对数据的权限，就可以将数据备份，获得自身权限内的副本，并在不征得甲同意的前提下传播副本。造成这一问题的根本原因就在于，这种机制仅仅通过对数据的存取权限来进行安全控制，而数据本身并无安全性标记。要解决这一问题，就需要对系统控制下的所有主客体实施强制存取控制策略。

9.2.4 强制存取控制（MAC）方法

所谓 MAC 是指系统为保证更高程度的安全性，按照 TDI/TCSEC 标准中安全策略的要求，所采取的强制存取检查手段。它不是用户能直接感知或进行控制的。MAC 适用于那些对数据有严格而固定密级分类的部门，例如军事部门或政府部门。

在 MAC 中，DBMS 所管理的全部实体被分为主体和客体两大类。

主体是系统中的活动实体，既包括 DBMS 所管理的实际用户，也包括代表用户的各进程。客体是系统中的被动实体，是受主体操纵的，包括文件、基表、索引、视图等。对于主体和客体，DBMS 为它们每个实例（值）指派一个敏感度标记（Label）。

敏感度标记被分成若干级别，例如绝密（Top Secret）、机密（Secret）、可信（Confidential）、公开（Public）等。主体的敏感度标记称为许可证级别（Clearance Level），客体的敏感度标记称为密级（Classification Level）。MAC 机制就是通过对比主体的 Label 和客体的 Label，最终确定主体是否能够存取客体。

当某一用户（或某一主体）以标记 label 注册入系统时，系统要求他对任何客体的存取必须遵循如下规则：

（1）仅当主体的许可证级别大于或等于客体的密级时，该主体才能读取相应的客体；

（2）仅当主体的许可证级别等于客体的密级时，该主体才能写相应的

客体。

规则（1）的意义是明显的。而规则（2）需要解释一下。在某些系统中，第（2）条规则与这里的规则有些差别。这些系统规定：仅当主体的许可证级别小于或等于客体的密级时，该主体才能写相应的客体，即用户可以为写入的数据对象赋予高于自己的许可证级别的密级。这样一旦数据被写入，该用户自己也不能再读该数据对象了。这两种规则的共同点在于它们均禁止了拥有高许可证级别的主体更新低密级的数据对象，从而防止了敏感数据的泄漏。

强制存取控制（MAC）是对数据本身进行密级标记，无论数据如何复制，标记与数据是一个不可分的整体，只有符合密级标记要求的用户才可以操纵数据，从而提供了更高级别的安全性。

前面已经提到，较高安全性级别提供的安全保护要包含较低级别的所有保护，因此在实现 MAC 时要首先实现 DAC，即 DAC 与 MAC 共同构成 DBMS 的安全机制，如图 9.2 所示。系统首先进行 DAC 检查， 对通过 DAC 检查的允许存取的数据对象再由系统自动进行 MAC 检查，只有通过 MAC 检查的数据对象方可存取。

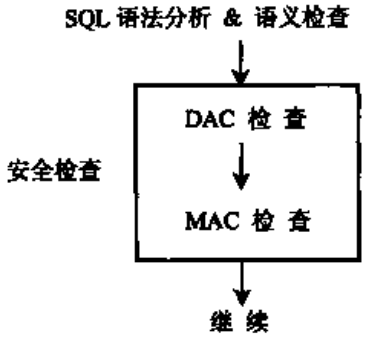


图 9.2 DAC + MAC 安全检查示意图

9.2.5 视图机制

进行存取权限控制时可以为不同的用户定义不同的视图，把数据对象限制在一定的范围内，也就是说，通过视图机制把要保密的数据对无权存取的用户隐藏起来，从而自动地对数据提供一定程度的安全保护。

视图机制间接地实现了支持存取谓词的用户权限定义，例如表 9.5 中王平只能检索计算机系学生的信息，这就要求系统能支持存取谓词。在不直接支持存取谓词的系统中，可以先建立计算机系学生的视图 CS_Student，然后在视图上进一步定义存取权限，把对 CS_Student 的 SELECT 权限授予王平。

```
CREATE VIEW CS_Student
    AS
```

```
        SELECT *
        FROM  Student
        WHERE  Sdept='CS';
GRANT  SELECT
ON  CS_Student
TO 王平;
```

9.2.6 审计（Audit）

前面讲的用户标识与鉴别、存取控制仅是安全性标准的一个重要方面（安全策略方面），不是全部。为了使 DBMS 达到一定的安全级别，还需要在其他方面提供相应的支持。例如按照 TDI/TCSEC 标准中安全策略的要求，“审计”功能就是 DBMS 达到 C2 以上安全级别必不可少的一项指标。

因为任何系统的安全保护措施都不是完美无缺的，蓄意盗窃、破坏数据的人总是想方设法打破控制。审计功能把用户对数据库的所有操作自动记录下来放入审计日志（Audit Log）中。DBA 可以利用审计跟踪的信息，重现导致数据库现有状况的一系列事件，找出非法存取数据的人、时间和内容等。

审计通常是很费时间和空间的，所以 DBMS 往往都将其作为可选特征，允许 DBA 根据应用对安全性的要求，灵活地打开或关闭审计功能。审计功能一般主要用于安全性要求较高的部门。

9.2.7 数据加密

对于高度敏感性数据，例如财务数据、军事数据、国家机密，除以上安全性措施外，还可以采用数据加密技术。

数据加密是防止数据库中数据在存储和传输中失密的有效手段。加密的基本思想是根据一定的算法将原始数据（术语为明文，Plain text）变换为不可直接识别的格式（术语为密文，Cipher text），从而使得不知道解密算法的人无法获知数据的内容。

加密方法主要有两种，一种是替换方法，该方法使用密钥（Encryption Key）将明文中的每一个字符转换为密文中的一个字符。另一种是置换方法，该方法仅将明文的字符按不同的顺序重新排列。单独使用这两种方法的任意一种都是不够安全的。但是将这两种方法结合起来就能提供相当高的安全程度。采用这种结合算法的例子是美国 1977 年制定的官方加密标准，数据加密标准（Data Encryption Standard，简称 DES）。

有关 DES 密钥加密技术及密钥管理问题等已超出本书范围，这里不再讨论。

目前有些数据库产品提供了数据加密例行程序，可根据用户的要求自动对存储和传输的数据进行加密处理。另一些数据库产品虽然本身未提供加密程序，但提供了接口，允许用户用其他厂商的加密程序对数据加密。

由于数据加密与解密也是比较费时的操作，而且数据加密与解密程序会占用大量系统资源，因此数据加密功能通常也作为可选特征，允许用户自由选择，只对高度机密的数据加密。

9.3 统计数据库安全性

一般地，统计数据库允许用户查询聚集类型的信息（例如合计、平均值等），但是不允许查询单个记录信息。例如，查询“程序员的平均工资是多少？”是合法的，但是查询“程序员张勇的工资是多少？”就不允许。

在统计数据库中存在着特殊的安全性问题，即可能存在着隐蔽的信息通道，使得可以从合法的查询中推导出不合法的信息。例如下面两个查询都是合法的：

- 本公司共有多少女高级程序员？
- 本公司女高级程序员的工资总额是多少？

如果第 1 个查询的结果是“1”，那么第 2 个查询的结果显然就是这个程序员的工资数。这样统计数据库的安全性机制就失效了。为了解决这个问题，可以规定任何查询至少要涉及 N 个以上的记录(N 足够大)。但是即使这样，还是存在另外的泄密途径，看下面的例子：

某个用户 A 想知道另一用户 B 的工资数额，他可以通过下列两个合法查询获取：

- 用户 A 和其他 N 个程序员的工资总额是多少？
- 用户 B 和其他 N 个程序员的工资总额是多少？

假设第 1 个查询的结果是 X，第 2 个查询的结果是 Y，由于用户 A 知道自己的工资是 Z，那么他可以计算出用户 B 的工资=Y-(X-Z)。

这个例子的关键之处在于两个查询之间有很多重复的数据项（即其他 N 个程序员的工资）。因此可以再规定任意两个查询的相交数据项不能超过 M 个。这样就使得获取他人的数据更加困难了。可以证明，在上述两条规定下，如果想获知用户 B 的工资额，用户 A 至少需要进行 $1+(N-2)/M$ 次查询。

当然可以继续规定任一用户的查询次数不能超过 $1+(N-2)/M$，但是如果两

个用户合作查询就可以使这一规定仍然失效。

另外还有其他一些方法用于解决统计数据库的安全性问题，例如数据污染。但是无论采用什么安全性机制，都仍然会存在绕过这些机制的途径。好的安全性措施应该使得那些试图破坏安全的人所花费的代价远远超过他们所得到的利益，这也是整个数据库安全机制设计的目标。

9.4 Oracle 数据库的安全性措施

下面介绍 Oracle 的安全性措施，使读者能进一步理解前面讨论的数据库安全性的基本方法。

Oracle 的安全措施主要有三个方面，一是用户标识和鉴定；二是授权和检查机制；三是审计技术（是否使用审计技术可由用户灵活选择）；除此之外，Oracle 还允许用户通过触发器灵活定义自己的安全性措施。

一、用户标识和鉴定

在 Oracle 中，最外层的安全性措施是让用户标识自己的名字，然后由系统进行核实。Oracle 允许用户重复标识三次，如果三次未通过，系统自动退出。

二、授权与检查机制

Oracle 的授权和检查机制与前面讲的存取控制方法基本相同。

Oracle 的权限包括系统权限和数据库对象的权限两类，采用非集中的授权机制，即 DBA 负责授予与回收系统权限，每个用户授予与回收自己创建的数据库对象的权限。

Oracle 允许重复授权，即可将某一权限多次授予同一用户，系统不会出错。Oracle 也允许无效回收，即用户没有某种权限，但回收此权限的操作仍算成功。

1. 系统权限

Oracle 提供了 80 多种系统权限，如创建会话、创建表、创建视图、创建用户等。DBA 在创建一个用户时需要将其中的一些权限授予该用户。

Oracle 支持角色的概念。所谓角色就是一组系统权限的集合，目的在于简化权限管理。Oracle 除允许 DBA 定义角色外，还提供了预定义的角色，如 CONNECT，RESOURCE 和 DBA。

具有 CONNECT 角色的用户可以登录数据库，执行数据查询和操纵。即可以执行 ALTER TABLE，CREATE VIEW，CREATE INDEX，DROP TABLE，DROP VIEW，DROP INDEX，GRANT，REVOKE，INSERT，SELECT，

UPDATE，DELETE，AUDIT，NOAUDIT 等操作。

RESOURCE 角色可以创建表，即执行 CREATE TABLE 操作。创建表的用户将拥有对该表的所有权限。

DBA 角色可以执行某些授权命令，创建表，对任何表的数据进行操纵。它涵盖了前两种角色，此外还可以执行一些管理操作，DBA 角色拥有最高级别的权限。

例如 DBA 建立一用户 U1 后，欲将 ALTER TABLE，CREATE VIEW，CREATE INDEX，DROP TABLE，DROP VIEW，DROP INDEX，GRANT，REVOKE，INSERT ，SELECT，UPDATE，DELETE，AUDIT，NOAUDIT 等系统权限授予 U1，则可以只简单地将 CONNECT 角色授予 U1 即可：

GRANT CONNECT TO U1;

这样就可以省略十几条 GRANT 语句。

2. 数据库对象的权限

在 Oracle 中，可以授权的数据库对象包括基本表、视图、序列、同义词、存储过程、函数等，其中最重要的是基本表。

对于基本表 Oracle 支持三个级别的安全性：表级、行级和列级。

（1）表级安全性

表的创建者或 DBA 可以把表级权限授予其他用户，表级权限包括：

ALTER：修改表定义

DELETE：删除表记录

INDEX：在表上建索引

INSERT：向表中插入数据记录

SELECT：查找表中记录

UPDATE：修改表中的数据

ALL：上述所有权限

表级授权使用 GRANT 和 REVOKE 语句，其语法与第三章中介绍的基本一致。

（2）行级安全性

Oracle 行级安全性由视图实现。用视图定义表的水平子集，限定用户在视图上的操作，就为表的行级提供了保护。视图上的授权与回收与表级完全相同。

例如，只允许用户 U2 查看 Student 表中信息系学生的数据，则首先创建信息系学生视图 S_IS，然后将该视图的 SELECT 权限授予 U2 用户。

（3）列级安全性

Oracle 列级安全性可以由视图实现，也可以直接在基本表上定义。

用视图定义表的垂直子集就可以实现列级安全性，方法与上面类似。

直接在基本表上定义和回收列级权限也是使用 GRANT 和 REVOKE 语句。目前 Oracle 的列级权限只有 UPDATE，回收列级 UPDATE 权限时，Oracle 不允许一列一列地回收，只能回收整个表的 UPDATE 权限。例如，

```
GRANT  UPDATE(Sno，Cno)
ON  SC
TO  U2;
```

把对 SC 表中 Sno 列和 Cno 列的 UPDATE 权限授予 U2 用户。

```
REVOKE  UPDATE  ON  SC  FROM  U2;
```

回收了 U2 用户对 SC 表中 Sno 列和 Cno 列的 UPDATE 权限。

在 Oracle 中，表、行、列三级对象自上而下构成一个层次结构，其中上一级对象的权限制约下一级对象的权限。例如当一个用户拥有了对某个表的 UPDATE 权限，即相当于在表的所有列都拥有了 UPDATE 权限。

Oracle 对数据库对象的权限采用分散控制方式，允许具有 WITH GRANT OPTION 的用户把相应权限或其子集传递授予其他用户，但不允许循环授权，即被授权者不能把权限再授回给授权者或其祖先，如图 9.3 所示。

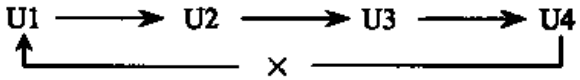


图 9.3 循环授权

Oracle 把所有权限信息记录在数据字典中。当用户进行数据库操作时，Oracle 首先根据数据字典中的权限信息，检查操作的合法性。在 Oracle 中，安全性检查是任何数据库操作的第一步。

三、Oracle 的审计技术

在 Oracle 中，审计分为用户级审计和系统级审计。用户级审计是任何 Oracle 用户可设置的审计，主要是用户针对自己创建的数据库表或视图进行审计，记录所有用户对这些表或视图的一切成功和（或）不成功的访问要求以及各种类型的 SQL 操作。

系统级审计只能由 DBA 设置，用以监测成功或失败的登录要求、监测 GRANT 和 REVOKE 操作以及其他数据库级权限下的操作。

Oracle 的审计功能很灵活，是否使用审计，对哪些表进行审计，对哪些操作进行审计等都可以由用户选择。为此，Oracle 提供了 AUDIT 语句设置审计功能，NOAUDIT 语句取消审计功能。设置审计时，可以详细指定对哪些 SQL 操作进行审计。例如，

对修改 SC 表结构或数据的操作进行审计可使用如下语句：

```
AUDIT ALTER, UPDATE ON SC;
```

取消对 SC 表的一切审计可使用如下语句：

```
NOAUDIT ALL ON SC;
```

在 Oracle 中，审计设置以及审计内容均存放在数据字典中。其中审计设置记录在数据字典表 SYS.TABLES 中，审计内容记录在数据字典表 SYS.AUDIT_TRAIL 中。

四、用户定义的安全性措施

除了系统级的安全性措施外，Oracle 还允许用户用数据库触发器定义特殊的更复杂的用户级安全性措施。例如，规定只能在工作时间内更新 Student 表，可以定义如下触发器，其中 sysdate 为系统当前时间：

```
CREATE OR REPLACE TRIGGER secure_student
BEFORE INSERT OR UPDATE OR DELETE ON Student
BEGIN
    IF (TO_CHAR(sysdate, 'DY') IN ('SAT', 'SUN'))
        OR (TO_NUMBER(sysdate, 'HH24') NOT BETWEEN 8 AND 17)
    THEN
        RAISE_APPLICATION_ERROR(-20506,
            'You may only change data during normal business hours.')
    END IF;
END;
```

触发器一经定义便存放在数据字典中。用户每次对 Student 表执行 INSERT，UPDATE 或 DELETE 操作时都会自动触发该触发器，由系统检查当时的系统时间，如果是周六或周日，或者不是 8 点至 17 点，系统会拒绝执行用户的更新操作，并提示出错信息。

类似地，用户还可以利用触发器进一步细化审计规则，使审计操作的粒度更细。

综上所述，Oracle 提供了多种安全性措施，提供了多级安全性检查，其安全性机制与操作系统的安全机制彼此独立，数据字典在 Oracle 的安全性授权和检查以及审计技术中起着重要作用。

9.5 小 结

随着计算机特别是计算机网络的发展，数据的共享日益加强，数据的安全

保密越来越重要。DBMS 是管理数据的核心，因而其自身必须具有一整套完整而有效的安全性机制。

《可信计算机系统评测标准》TCSEC/TDI 是目前各国所引用或制定的一系列安全标准中最重要的一个。TCSEC/TDI 从安全策略、责任、保证和文档四个方面描述了安全性级别的指标。按照这些指标，目前许多大型 DBMS 达到了 C2 级，其安全版本达到了 B1。

实现数据库系统安全性的技术和方法有多种，最重要的是存取控制技术和审计技术。C2 级的 DBMS 必须具有自主存取控制功能和初步的审计功能，B1 的 DBMS 必须具有强制存取控制和增强的审计功能。自主存取控制功能一般是通过 SQL 的 GRANT 语句和 REVOKE 语句来实现的。

习 题

1. 什么是数据库的安全性？

2. 数据库安全性和计算机系统的安全性有什么关系？

3. 试述可信计算机系统评测标准的情况，试述 TDI/TCSEC 标准的基本内容。

4. 试述 TCSEC（TDI）将系统安全级别划分为四组七个等级的基本内容。

5. 试述实现数据库安全性控制的常用方法和技术。

6. 什么是数据库中的自主存取控制方法和强制存取控制方法？

7. SQL 语言中提供了哪些数据控制（自主存取控制）的语句？请试举几例说明它们的使用方法。

8. 今有两个关系模式：

职工（职工号，姓名，年龄，职务，工资，部门号）

部门（部门号，名称，经理名，地址，电话号）

请用 SQL 的 GRANT 和 REVOKE 语句（加上视图机制）完成以下授权定义或存取控制功能：

(a) 用户王明对两个表有 SELECT 权力；

(b) 用户李勇对两个表有 INSERT 和 DELETE 权力；

(c)*每个职工只对自己的记录有 SELECT 权力；

(d) 用户刘星对职工表有 SELECT 权力，对工资字段具有更新权力；

(e) 用户张新具有修改这两个表的结构的权力；

(f) 用户周平具有对两个表所有权力(读，插，改，删数据)，并具有给其他用户授权的权力；

(g) 用户杨兰具有从每个部门职工中 SELECT 最高工资，最低工资，平均工资的权力，他不能查看每个人的工资。

9. 把习题 8 中(a)~(g) 的每一种情况，撤销各用户所授予的权力。

10. 为什么强制存取控制提供了更高级别的数据库安全性?

11. 理解并解释 MAC 机制中主体、客体、敏感度标记的含义。

12. 举例说明 MAC 机制如何确定主体能否存取客体。

13. 什么是数据库的审计功能，为什么要提供审计功能?

14. 统计数据库中存在何种特殊的安全性问题?

*15. 试述你了解的某一个实际的 DBMS 产品的安全性措施。

本章参考文献

1 Department of Defense(USA). Department of Defense Trusted Computer System Evaluation Criteria (Orange Book), Rainbow Series-12/85 (DoD 5200.28-std)

2 Evaluated Products List Indexed by Rating. www.radium.ncsc/tpep/library/rainbow

3 Department of Defense. Trusted Database Management System of TCSEC, 1991

4 Fernfandez E B, Summers R C, Wood C. Database Security and Integrity. Reading Mass, Addison-Wesley, 1981

（本文献详细地讨论了数据库的安全性和完整性问题。）

5 Denning D E. Cryptography and Data Security, Reading Mass, Addison Wesley, 1983

6 Zloof M M. Security and Integrity Within the Query-By-Example Data Base Management Language. IBM Research Report RC 6982, February 1978

7 Jajodia Sushil, Sandhu Ravi. Toward a Multilevel Secure Relation Data Model. Proc. ACM SIGMOD International Conference on Management Data, May 1991

8 Griffiths P, Wade B. An Authorization Mechanism for a Relational Database System. ACM TODS, 1:3, September 1976

（本文讨论了 SYSTEM R 的授权机制。关系数据库系统原型 SYSTEM R 首先提出了授权和撤销权力的概念。）

9 Leiss E. Principles of Data Security, Plenum Press, 1982

（本文献详细地讨论了计算机系统和数据库的安全性问题。）

10 Chin F. Security in Statistical Databases for Queries with Small Counts. ACM TODS, 3: 1, March 1978

（本文讨论了统计数据库的安全性和保护技术。）

11 Chin F, Ozsoyoglu G. Statistical Database Design. ACM TODS, 6: 1, March 1981

12 Leiss E, Randomizing A. A Practical Method for Protecting Statistical Database Against Compromise. in Proceedings of VLDB, 1982

13 Rivest R, Shamir A, Adelman L. On Digital Signatures and Public Key Cryptosystems. CACM, 21: 2, February 1978

（本文讨论了数据加密技术。）

14 Akl S. Digital Signatures: A Tutorial Survey. IEEE Computer, 16:2, February 1983

15 刘启原. 数据库与信息系统的安全. 数据库技术丛书之一，北京：科学出版社，1999

（本书系统讲解了信息系统的安全性问题，包括安全模型、与安全有关的标准、密码学与信息保密、攻击检测技术和理论、数据库的安全性、数据仓库的安全问题、网络安全问题、Java 语言及其安全性问题、信息安全和病毒问题等。）

16 张孝. 可信 COBASE 的系统强制存取控制的设计与实现. 中国人民大学硕士论文，1998

（本文系统讨论了数据库系统的安全性问题，给出了可信 COBASE 的系统设计和实现技术。可信 COBASE 是 COBASE V2.0 的安全版本。COBASE V2.0 是由北京大学、中国人民大学、中国软件总公司研制的 RDBMS。可信 COBASE 依据 TCSEC 和 TDI 对 B1 级的要求进行设计，实现了强制存取控制（MAC）和安全审计。）

17 文继荣、张孝、罗立等. 可信 COBASE 的设计策略：数据强制存取控制机制及其实现. 第十四届全国数据库学术会议论文集，1997

第十章　数据库完整性

数据库的完整性是指数据的正确性和相容性。例如，学生的学号必须唯一；性别只能是男或女；本科学生年龄的取值范围为 14~30 的整数；学生所在的系必须是学校已开设的系；等。数据库是否具备完整性关系到数据库系统能否真实地反映现实世界，因此维护数据库的完整性是非常重要的。

数据的完整性和安全性是两个不同的概念。

前者是为了防止数据库中存在不符合语义的数据，防止错误信息的输入和输出，即所谓垃圾进垃圾出（Garbage In Garbage Out）所造成的无效操作和错误结果。而后者是保护数据库防止恶意的破坏和非法的存取。也就是说，安全性措施的防范对象是非法用户和非法操作，完整性措施的防范对象是不合语义的数据。当然，完整性和安全性是密切相关的。

为维护数据库的完整性，DBMS 必须提供一种机制来检查数据库中的数据，看其是否满足语义规定的条件。这些加在数据库数据之上的语义约束条件称为**数据库完整性约束条件**，它们作为模式的一部分存入数据库中。而 DBMS 中检查数据是否满足完整性条件的机制称为**完整性检查**。

下面首先讨论完整性约束条件的概念，然后介绍完整性保证的方法。

10.1　完整性约束条件

完整性检查是围绕完整性约束条件进行的，因此完整性约束条件是完整性控制机制的核心。

完整性约束条件作用的对象可以是关系、元组、列三种。其中列约束主要是列的类型、取值范围、精度、排序等约束条件。元组的约束是元组中各个字段间的联系的约束。关系的约束是若干元组间、关系集合上以及关系之间的联系的约束。

完整性约束条件涉及的这三类对象，其状态可以是静态的，也可以是动态的。所谓静态约束是指数据库每一确定状态时的数据对象所应满足的约束条件，它是反映数据库状态合理性的约束，这是最重要的一类完整性约束。

动态约束是指数据库从一种状态转变为另一种状态时，新、旧值之间所应

满足的约束条件，它是反映数据库状态变迁的约束。

综合上述两个方面，可以将完整性约束条件分为六类，如图 10.1 所示。

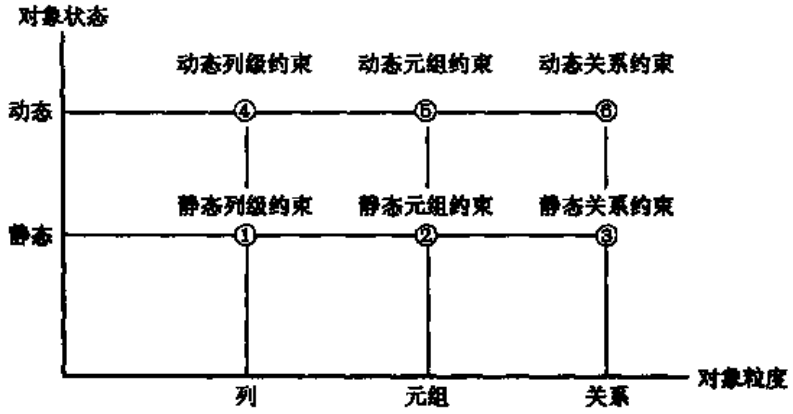


图 10.1 完整性约束条件分类

一、静态列级约束

静态列级约束是对一个列的取值域的说明，这是最常用也最容易实现的一类完整性约束，包括以下几方面：

1. 对数据类型的约束（包括数据的类型、长度、单位、精度等）

例如，中国人民大学数据库中学生姓名的数据类型规定为字符型，长度为 8。中央民族大学数据库中学生姓名的数据类型规定为字符型，长度为 20，因为少数民族的姓名较长。

2. 对数据格式的约束

例如，规定学号的前两位表示入学年份，中间两位表示系的编号，后三位为顺序编号。出生日期的格式为 YY.MM.DD。

3. 对取值范围或取值集合的约束

例如，规定学生成绩的取值范围为 0~100，大学本科学生年龄的取值范围为 14~29，性别的取值集合为[男，女]。

4. 对空值的约束

空值表示未定义或未知的值，它与零值和空格不同。有的列允许空值，有的则不允许。例如学生学号不能取空值，成绩可以为空值。

5. 其他约束

例如关于列的排序说明，组合列等。

二、静态元组约束

一个元组是由若干个列值组成的，静态元组约束就是规定元组的各个列之

间的约束关系。例如订货关系中包含发货量、订货量等列，规定发货量不得超过订货量；又如教师关系中包含职称、工资等列，规定教授的工资不低于1000元。

三、静态关系约束

在一个关系的各个元组之间或者若干关系之间常常存在各种联系或约束。常见的静态关系约束有：

（1）实体完整性约束。

（2）参照完整性约束。

实体完整性约束和参照完整性约束是关系模型的两个极其重要的约束，称为关系的两个不变性。

（3）函数依赖约束。大部分函数依赖约束都在关系模式中定义。

（4）统计约束。即字段值与关系中多个元组的统计值之间的约束关系。例如规定部门经理的工资不得高于本部门职工平均工资的5倍，不得低于本部门职工平均工资的2倍。这里，本部门职工的平均工资是一个统计值。

四、动态列级约束

动态列级约束是修改列定义或列值时应满足的约束条件，包括下面两方面：

（1）修改列定义时的约束

例如，将允许空值的列改为不允许空值时，如果该列目前已存在空值，则拒绝这种修改。

（2）修改列值时的约束

修改列值有时需要参照其旧值，并且新旧值之间需要满足某种约束条件。例如，职工工资调整不得低于其原来工资，学生年龄只能增长等。

五、动态元组约束

动态元组约束是指修改元组的值时元组中各个字段间需要满足某种约束条件。例如职工工资调整时新工资不得低于原工资+工龄*1.5等。

六、动态关系约束

动态关系约束是加在关系变化前后状态上的限制条件，例如事务一致性、原子性等约束条件。

以上六类完整性约束条件的含义可用表10.1进行概括。

当然完整性的约束条件可以从不同角度进行分类。因此会有多种分类方法。

表 10.1 完整性约束条件

状态 \ 粒度	列 级	元 组 级	关 系 级
静 态	列定义 • 类型 • 格式 • 值域 • 空值	元组值应 满足的条件	实体完整性约束 参照完整性约束 函数依赖约束 统计约束
动 态	改变列定义或列值	元组新旧值之间应满足的约束条件	关系新旧状态间应满足的约束条件

10.2 完整性控制

DBMS 的完整性控制机制应具有三个方面的功能：

• 定义功能，提供定义完整性约束条件的机制。

• 检查功能，检查用户发出的操作请求是否违背了完整性约束条件。

• 如果发现用户的操作请求使数据违背了完整性约束条件，则采取一定的动作来保证数据的完整性。

完整性约束条件包括有六大类，约束条件可能非常简单，也可能极为复杂。一个完善的完整性控制机制应该允许用户定义所有这六类完整性约束条件。

检查是否违背完整性约束的时机通常是在一条语句执行完后立即检查，称这类约束为立即执行约束（Immediate Constraints）。有时完整性检查需要延迟到整个事务执行结束后再进行，检查正确方可提交，称这类约束为延迟执行约束（Deferred Constraints）。例如银行数据库中“借贷总金额应平衡”的约束就应该是延迟执行的约束，从账号 A 转一笔钱到账号 B 为一个事务，从账号 A 转出去钱后账就不平了，必须等转入账号 B 后账才能重新平衡，这时才能进行完整性检查。

如果发现用户操作请求违背了完整性约束条件，系统将拒绝该操作，但对于延迟执行的约束，系统将拒绝整个事务，把数据库恢复到该事务执行前的状态。

一条完整性规则可以用一个五元组 (D，O，A，C，P) 来表示，其中：

• D（Data） 约束作用的数据对象；

• O（Operation） 触发完整性检查的数据库操作，即当用户发出什么操作请求时需要检查该完整性规则，是立即检查还是延迟检查；

- A（Assertion） 数据对象必须满足的断言或语义约束，这是规则的主体；
- C（Condition） 选择 A 作用的数据对象值的谓词；
- P（Procedure） 违反完整性规则时触发的过程。

例如，在“学号不能为空”的约束中

D 约束作用的对象为 Sno 属性

O 插入或修改 Student 元组时

A Sno 不能为空

C 无（A 可作用于所有记录的 Sno 属性）

P 拒绝执行该操作

又如，在“教授工资不得低于 1000 元”的约束中

D 约束作用的对象为工资 Sal 属性

O 插入或修改职工元组时

A Sal 不能小于 1000

C 职称='教授'（A 仅作用于职称='教授'的记录）

P 拒绝执行该操作

在关系系统中，最重要的完整性约束是实体完整性和参照完整性，其他完整性约束条件则可以归入用户定义的完整性。

目前许多关系数据库管理系统都提供了定义和检查实体完整性、参照完整性和用户定义的完整性的功能。对于违反实体完整性和用户定义的完整性的操作一般都采用拒绝执行的方式进行处理。而对于违反参照完整性的操作，并不都是简单地拒绝执行，有时要根据应用语义执行一些附加的操作，以保证数据库的正确性。

在 2.3 节中已讨论了关系系统中的实体完整性、参照完整性和用户定义的完整性的含义，下面详细讨论实现参照完整性要考虑的几个问题。

1. 外码能否接受空值问题

例如，职工-部门数据库包含职工表 EMP 和部门表 DEPT，其中 DEPT 关系的主码为部门号 Deptno，EMP 关系的主码为职工号 Empno，外码为部门号 Deptno，称 DEPT 为被参照关系或目标关系，EMP 为参照关系。

EMP 中，某一元组的 Deptno 列若为空值，表示这个职工尚未分配到任何具体的部门工作。这和应用环境的语义是相符的，因此 EMP 的 Deptno 列可以取空值。但在学生一选课数据库中，Student 关系为被参照关系，其主码为 Sno。SC 为参照关系，外码为 Sno。若 SC 的 Sno 为空值，则表明尚不存在的某个学生，或者某个不知学号的学生，选修了某门课程，其成绩记录在 Grade 列中。这与学校的应用环境是不相符的，因此 SC 的 Sno 列不能取空值。

因此在实现参照完整性时，系统除了应该提供定义外码的机制，还应提供

定义外码列是否允许空值的机制。

2. 在被参照关系中删除元组的问题

例如要删除 Student 关系中 Sno=95001 的元组，而 SC 关系中又有 4 个元组的 Sno 都等于 95001，一般地，当删除被参照关系的某个元组，而参照关系存在若干元组，其外码值与被参照关系删除元组的主码值相同，这时可有三种不同的策略：

（1）级联删除（CASCADES）

将参照关系中所有外码值与被参照关系中要删除元组主码值相同的元组一起删除。例如将上面 SC 关系中 4 个 Sno=95001 的元组一起删除。如果参照关系同时又是另一个关系的被参照关系，则这种删除操作会继续级联下去。

（2）受限删除（RESTRICTED）

仅当参照关系中没有任何元组的外码值与被参照关系中要删除元组的主码值相同时，系统才执行删除操作，否则拒绝此删除操作。例如对于上面的情况，系统将拒绝删除 Student 关系中 Sno=95001 的元组。

（3）置空值删除（NULLIFIES）

删除被参照关系的元组，并将参照关系中相应元组的外码值置空值。例如将上面 SC 关系中所有 Sno=950001 的元组的 Sno 值置为空值。

这三种处理方法，哪一种是正确的呢？这要依应用环境的语义来定。在学生-选课数据库中，显然第 1 种方法是对的。因为当一个学生毕业或退学后，他的个人记录从 Student 表中删除了，他的选课记录也应随之从 SC 表中删除。

3. 在参照关系中插入元组时的问题

例如向 SC 关系插入（99001，1，90）元组，而 Student 关系中尚没有 Sno=99001 的学生，一般地，当参照关系插入某个元组，而被参照关系不存在相应的元组，其主码值与参照关系插入元组的外码值相同，这时可有以下策略：

（1）受限插入

仅当被参照关系中存在相应的元组，其主码值与参照关系插入元组的外码值相同时，系统才执行插入操作，否则拒绝此操作。例如对于上面的情况，系统将拒绝向 SC 关系插入（99001，1，90）元组。

（2）递归插入

首先向被参照关系中插入相应的元组，其主码值等于参照关系插入元组的外码值，然后向参照关系插入元组。例如对于上面的情况，系统将首先向 Student 关系插入 Sno=99001 的元组，然后向 SC 关系插入（99001，1，90）元组。

4. 修改关系中主码的问题

（1）不允许修改主码

在有些 RDBMS 中，修改关系主码的操作是不允许的，例如不能用 UPDATE

语句将学号 95001 改为 95102。如果需要修改主码值，只能先删除该元组，然后再把具有新主码值的元组插入到关系中。

（2）允许修改主码

在有些 RDBMS 中，允许修改关系主码，但必须保证主码的唯一性和非空，否则拒绝修改。

当修改的关系是被参照关系时，还必须检查参照关系，是否存在这样的元组，其外码值等于被参照关系要修改的主码值。例如要将 Student 关系中 Sno=950001 的 Sno 值改为 960123，而 SC 关系中有 4 个元组的 Sno=950001，这时与（上面第 2 点）在被参照关系中删除元组的情况类似，可以有：级联修改、拒绝修改、置空值修改三种策略加以选择。

当修改的关系是参照关系时，还必须检查被参照关系，是否存在这样的元组，其主码值等于被参照关系要修改的外码值。例如要把 SC 关系中（95001，1，90）元组修改为（99001，1，90），而 Student 关系中尚没有 Sno=99001 的学生，这时与（上面第 3 点）在参照关系中插入元组时情况类似，可以有：受限插入和递归插入两种策略加以选择。

从上面的讨论看到 DBMS 在实现参照完整性时，除了要提供定义主码、外码的机制外，还需要提供不同的策略供用户选择。选择哪种策略，都要根据应用环境的要求确定。

10.3 Oracle 的完整性

上面介绍了完整性控制的一般方法，下面介绍 Oracle 的完整性控制策略。

一、Oracle 中的实体完整性

Oracle 在 CREATE TABLE 语句中提供了 PRIMARY KEY 子句，供用户在建表时指定关系的主码列。例如，在学生-选课数据库中，要定义 Student 表的 Sno 为主码，可使用如下语句：

```
CREATE TABLE Student
      (Sno   NUMBER(8),
       Sname   VARCHAR(20),
       Sage   NUMBER(20),
       CONSTRAINT PK_SNO PRIMARY KEY (Sno));
```

其中，PRIMARY KEY(Sno)表示 Sno 是 Student 表的主码。PK_SNO 是此主码约束名。

若要在 SC 表中定义(Sno， Cno)为主码，则用下面语句建立 SC 表：

```
CREATE TABLE SC
      (Sno  NUMBER(8),
       Cno  NUMBER(2),
       Grade NUMBER(2),
       CONSTRAINT  PK_SC  PRIMARY KEY (Sno，Cno) );
```

用 PRIMARY KEY 语句定义了关系的主码后，每当用户程序对主码列进行更新操作时，系统自动进行完整性检查，凡操作使主码值为空值或使主码值在表中不唯一，系统拒绝此操作，从而保证了实体完整性。

二、Oracle 中的参照完整性

Oracle 的 CREATE TABLE 语句也可以定义参照完整性规则，即用 FOREIGN KEY 子句定义哪些列为外码列，用 REFERENCES 子句指明这些外码相应于哪个表的主码，用 ON DELETE CASCADE 子语指明在删除被参照关系的元组时，同时删除参照关系中相应的元组。

例如，使用如下 SQL 语句建立表 EMP 表：

```
CREATE  TABLE  EMP
     (Empno  NUMBER(4),
      Ename  VARCHAR(10),
      Job  VARCHAR2(9),
      Mgr  NUMBER(4),
      Sal  NUMBER(7，2),
      Deptno  NUMBER(2),
      CONSTRAINT  FK_DEPTNO
         FOREIGN  KEY (Deptno)
         REFERENCES  DEPT(Deptno) );
```

则表明 EMP 是参照表，DEPT 为其被参照表，EMP 表中 Deptno 为外码，它相应于 DEPT 表中的主码 Deptno。当删除或修改 DEPT 表中某个元组的主码时要检查 EMP 中是否有元组的 DEPTNO 值等于 DEPT 中要删除的元组的 Deptno 值，如没有，接受此操作；否则系统拒绝这一更新操作。

如果用如下 SQL 语句建立 EMP 表：

```
CREATE TABLE  EMP
      (Empno  NUMBER(4),
       Ename  VARCHAR(10),
       Job  VARCHAR2(9),
```

```
    Mgr  NUMBER(4),
    Sal  NUMBER(7, 2),
    Deptno  NUMBER(2),
    CONSTRAINT  FK_DEPTNO
       FOREIGN KEY (Deptno)
       REFERENCES DEPT(Deptno)
       ON DELETE CASCADE);
```

当要修改DEPT表中的Deptno值时，先要检查EMP表中有无元组的Deptno值与之对应，若没有，系统接受这个修改操作，否则，系统拒绝此操作。

当要删除 DEPT 表中某个元组时，系统也要检查 EMP 表，若找到相应元组则将它们也随之删除。

三、Oracle 中用户定义的完整性

除实体完整性和参照完整性外，应用系统中往往还需要定义与应用有关的完整性限制。例如：要求某一列的值不能取空值，要在表中是唯一的，要在某个取值范围中等。Oracle 允许用户在建表时定义下列完整性约束：

- 列值非空（NOT NULL 短语）
- 列值唯一（UNIQUE 短语）
- 检查列值是否满足一个布尔表达式（CHECK 短语）

例 1　建立部门表 DEPT，要求部门名称 Dname 列取值唯一，部门编号 Deptno 列为主码。

```
CREATE TABLE DEPT
    (Deptno  NUMBER,
     Dname  VARCHAR(9)  CONSTRAINT  U1  UNIQUE,
     Loc  VARCHAR(10),
     CONSTRAINT  PK_DEPT PRIMARY KEY (Deptno));
```

其中 CONSTRAINT　U1　UNIQUE 表示约束名为 U1，该约束要求 Dname 列值唯一。

例 2　建立学生登记表 Student，要求学号在 90000~99999 之间，年龄<29，性别只能是“男”或“女”，姓名非空。

```
CREATE TABLE Student
    (Sno  NUMBER(5)
          CONSTRAINT C1 CHECK (Sno BETWEEN 90000 AND 99999),
     Sname  VARCHAR(20)  CONSTRAINT C2 NOT NULL,
     Sage  NUMBER(3)
```

```
                CONSTRAINT C3   CHECK (Sage < 29),
        Ssex    VARCHAR(2)
                CONSTRAINT C4   CHECK (Ssex IN ('男', '女') );
```

例 3 建立职工表 EMP，要求每个职工的应发工资不得超过 3000 元。应发工资实际上就是实发工资列 Sal 与扣除项 Deduct 之和。

```
CREATE TABLE EMP
    (Eno   NUMBER(4)
     Ename   VARCHAR(10),
     Job   VARCHAR(8),
     Sal   NUMBER(7，2),
    Deduct   NUMBER(7，2)
    Deptno   NUMBER(2),
    CONSTRAINTS   C1   CHECK (Sal + Deduct <=3000) );
```

在 Oracle 中，除列值非空、列值唯一、检查列值是否满足一个布尔表达式外，用户还可以通过触发器（Trigger）来实现其他完整性规则。所谓数据库触发器，就是一类靠事件驱动的特殊过程，一旦由某个用户定义，任何用户对该数据的增、删、改操作均由服务器自动激活相应的触发器，在核心层进行集中的完整性控制。

定义数据库触发器的语句是 CREATE TRIGGER。

例 4 为教师表 Teacher 定义完整性规则“教授的工资不得低于 1000 元，如果低于 1000 元，自动改为 1000 元”。

```
CREATE   TRIGGER   UPDATE_SAL
    BEFORE   INSERT OR UPDATE OF Sal，  Pos ON Teacher
    FOR EACH ROW
    WHEN (：new.Pos='教授')      /* 某教员晋升为教授 */
    BEGIN
        IF ：new.sal < 1000   THEN
            ：new.Sal ：=1000;
        END IF;
    END;
```

综上所述，Oracle 提供了 CREATE TABLE 语句和 CREATE TRIGGER 语句定义完整性约束条件，其中用 CREATE TRIGGER 语句可以定义很复杂的完整性约束条件。完整性约束条件一旦定义好，Oracle 会自动执行相应的完整性检查，对于违反完整性约束条件的操作或者拒绝执行或者执行事先定义的操作。

10.4 小 结

数据库的完整性是为了保证数据库中存储的数据是正确的，所谓正确的是指符合现实世界语义的。本章讲解了 DBMS 完整性实现的机制，包括完整性约束定义机制、完整性检查机制和违背完整性约束条件时 DBMS 应采取的动作等。

完整性机制的实施会极大地影响系统性能，因此，许多数据库管理系统对完整性机制的支持比对安全性的支持要晚得多也弱得多。随着硬件性能和容量的提高，数据库技术的发展，许多数据库厂商对完整性的支持越来越好，不仅能保证实体完整性和参照完整性，而且能在 DBMS 核心定义、检查和保证用户定义的完整性约束条件。读者应注意，不同的数据库产品对完整性的支持策略和支持程度是不同的。

习 题

1. 什么是数据库的完整性？

2. 数据库的完整性概念与数据库的安全性概念有什么区别和联系？

3. 什么是数据库的完整性约束条件？可分为哪几类？

4. DBMS 的完整性控制机制应具有哪些功能？

5. RDBMS 在实现参照完整性时需要考虑哪些方面？

6. 假设有下面两个关系模式：

职工（职工号，姓名，年龄，职务，工资，部门号），其中职工号为主码；

部门（部门号，名称，经理名，电话），其中部门号为主码；

用 SQL 语言定义这两个关系模式，要求在模式中完成以下完整性约束条件的定义：

（1）定义每个模式的主码；（2）定义参照完整性；（3）定义职工年龄不得超过 60 岁。

7. 在关系系统中，当操作违反实体完整性、参照完整性和用户定义的完整性约束条件时，一般是如何分别进行处理的？

*8. 试述你了解的某一个实际的 DBMS 产品的完整性控制策略。

本章参考文献

1 Hammer M，McLeod D. Semantic Integrity in a Relational Data Base System. in Proc. of VLDB，1975

（本文讨论数据库管理系统中语义完整性的概念。）

2 Schmid J，Swenson J. On the Semantics of the Relational Model. in SIGMOD，1975

3 Codd E F. Extending the Database Relational Model to Capture More Meaning. TODS，4:4，December 1979

（本文扩展了关系模型的语义表达能力。）

4 Stonebraker M，Wong E. Access Control in a Relational Database Management System by Query Modification. Proceedings of the ACM Annual Conference，1974

（本文讨论 INGRES 数据库管理系统的完整性约束机制。）

5 Stonebraker M R. implementation of Integrity Constrains and Views by Query Modification. in Proceedings of SIGMOD，1975

（本文讨论 INGRES 数据库管理系统的完整性约束机制。）

6 Chamberlin D，et al. SEQUEL 2：A Unified Approach to Data Definition，Manipulation，and Control. IBM Journal of Research and Development，20：6，November 1976

（本文提出了 SYSTEM R 中 SQL 语言的 ASSERT(完整性断言)和 TRIGGER(触发器)语句。）

7 Hammer M，Sarin S. Efficient Monitoring of Database Assertions. in SIGMOD，1978

（本文介绍了完整性断言的有效验证方法。）

8 Bernstein P，Blaustein B，Clarke E. Fast Maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data. in Proceedings of VLDB，1980

9 Badal D，Popek G. Cost and Performance Analysis of Semantic Integrity Validation Methods. in SIGMOD，1979

10 Hsu A，Imielinsky T. Integrity Checking for Multiple Updates. in SIGMOD，1985

*第十一章　数据库管理系统

在第一章中，已简单介绍了数据库管理系统（以下简称 DBMS）的功能。这一章进一步阐述 DBMS 的基本功能，系统结构及主要的实现技术。使 DBA 或数据库应用系统开发人员（本章不是针对 DBMS 设计人员写的）能从宏观和总体的角度掌握 DBMS 的基本概念和基本原理，以便更好地使用和维护 DBMS。

11.1　DBMS 概 述

这一节概述 DBMS 的目标和基本功能，简述 DBMS 向下与操作系统的接口，向上为第四代应用开发环境提供的支持，从而阐明 DBMS 在整个计算机系统中的地位及作用。

11.1.1　DBMS 的目标

从计算机软件系统的构成看，DBMS 是介于用户和操作系统之间的一组软件，它实现对共享数据的有效组织、管理和存取。

由于 DBMS 实现的硬件资源、软件环境不同，DBMS 的功能和性能便有差异。但不管有多少差异，均应该尽量满足以下系统目标，它们是：

1. 用户界面友好

对一个实用 DBMS 来说，用户界面的质量直接影响其生命力。DBMS 的用户接口应面向应用，面向多种用户。一般地说，用户界面应具有可靠性、易用性、立即反馈和多样性等特性。

（1）可靠性。指界面要具有一定的容错能力，能及时正确地给出运行状态指示和出错信息，并引导用户改正错误。

（2）易用性。尽量减少用户负担，操作方式简单，容易记忆，输入/输出容易理解。

（3）立即反馈。对用户的应用请求都应在用户心理许可的时间范围内给出响应，即使不能得到结果，也应给出某种信息以缓和用户等待心理。

（4）多样性。根据用户背景的不同，提供多种用户接口，以适应不同层次用户的需要。

对于初级用户或初学者，提供以机器为主导的会话形式，DBMS 应给出较多的提示信息；对于熟练用户应采用以人为主导的对话形式，无需机器过多的提示，这样可提高处理效率；对于应用程序员等高级用户则应提供应用开发接口，其功能要全面，方式要灵活，以满足应用开发的需要。目前，许多 DBMS 都提供交互式 SQL、嵌入式 SQL、表格语言接口、命令式语言接口和函数调用接口等多种形式以方便用户使用。

2. 功能完备

DBMS 核心功能随系统的大小而异。大型 DBMS 功能完全，小型 DBMS 功能弱一些。DBMS 主要功能包括数据库定义、数据库数据存取、数据库运行管理、数据库组织和存储管理、数据库建立和维护等。这些功能的内容将在 11.1.2 节讨论。

3. 效率高

DBMS 应该具有高的系统效率和高的用户生产率。

系统效率包括两个方面：一是计算机系统内部资源的利用率，即能充分利用资源（包括盘空间、设备、CPU 等），并注意使各种资源的负载均衡以提高整个系统的效率。二是 DBMS 本身的运行效率，根据系统目标确定恰当的体系结构、数据结构和算法，保证 DBMS 运行的高效率。由于关系数据库管理系统（RDBMS）承担了为用户查询自动选择存取路径的任务，RDBMS 必须进行查询优化，即采取各种策略，寻找最佳查询执行方案，提高查询效率。这是 RDBMS 实现的关键和困难所在。

另一个方面是提高用户生产率。所谓用户生产率是指用户设计和开发应用程序的效率。关系数据库系统由于提供了严格而简明的数据模型，高度非过程化的数据语言，把基于数据的应用程序设计提高到了一个新的水平。此外，RDBMS 还为用户提供了一个良好的应用开发环境，进一步提高了应用开发的效率和质量。正是由于 RDBMS 大大提高了用户生产率，才使得它获得了强大的生命力。

4. 结构清晰

DBMS 是一个复杂的系统软件，涉及面广，包括向上与用户的接口，向下与操作系统及其他软、硬件资源的接口。其实现技术复杂，需要程序设计、编译原理、操作系统、数据结构等许多软件知识和技术的支持。因此，应该使 DBMS 内部结构清晰、层次分明。这样既便于 DBMS 支持其外层开发环境的构造，也便于自身的设计、开发和维护。清晰的结构也是 DBMS 具有开放性的一个必要条件。

5. 开放性

所谓开放性是指符合标准和规范，例如 SQL 标准、ODBC 标准等。遵循标准可以大大提高 DBMS 的互操作性和可扩展性，从而为建立以 DBMS 为核心的软件开发环境和企业级大规模的（跨地区、跨国家）信息系统提供基础。

11.1.2 DBMS 的基本功能

前面已讲过，DBMS 主要是实现对共享数据有效的组织、管理和存取。围绕数据，DBMS 应具有如下几个方面的基本功能。

1. 数据库定义

对数据库的结构进行描述，包括外模式、模式、内模式的定义；数据库完整性的定义；安全保密定义（如用户口令、级别、存取权限）；存取路径（如索引）的定义。这些定义存储在数据字典（亦称为系统目录）中，是 DBMS 运行的基本依据。

2. 数据存取

提供用户对数据的操作功能，实现对数据库数据的检索、插入、修改和删除。一个好的 DBMS 应该提供功能强易学易用的数据操纵语言（DML）、方便的操作方式和较高的数据存取效率。DML 有两类：一类是宿主型语言，一类是自立（独立）型语言。前者的语句不能独立使用而必须嵌入某种主语言，如 C 语言、PASCAL 语言、COBOL 语言中使用。而后者可以独立使用，通常供终端用户使用。

3. 数据库运行管理

这是指 DBMS 运行控制和管理功能。包括多用户环境下的事务的管理和自动恢复、并发控制和死锁检测（或死锁防止）、安全性检查和存取控制、完整性检查和执行、运行日志的组织管理等。这些功能保证了数据库系统的正常运行。

4. 数据组织、存储和管理

DBMS 要分类组织、存储和管理各种数据，包括数据字典、用户数据、存取路径等。要确定以何种文件结构和存取方式在存储级上组织这些数据，如何实现数据之间的联系。数据组织和存储的基本目标是提高存储空间利用率和方便存取，提供多种存取方法（如索引查找、Hash 查找、顺序查找等）提高存取效率。

5. 数据库的建立和维护

包括数据库的初始建立、数据的转换、数据库的转储和恢复、数据库的重组织和重构造以及性能监测分析等功能。

6. 其他功能

包括 DBMS 与网络中其他软件系统的通信功能；一个 DBMS 与另一个 DBMS 或文件系统的数据转换功能；异构数据库之间的互访和互操作功能等。

11.1.3 DBMS 和操作系统

从计算机软件系统的构成看，DBMS 是建立在操作系统之上的软件系统，是操作系统的用户。操作系统负责计算机系统的进程管理、作业管理、存储器管理、设备管理、文件管理等，因此 DBMS 对共享数据的组织、管理和存取离不开操作系统的支持。DBMS 遇到创建和撤销进程、进程通信、读写磁盘、分配内存等要求时必须请求操作系统的服务。例如，DBMS 读取数据库中记录时就要调用操作系统读取磁盘块的操作，操作系统从磁盘取来的是一个物理块，对物理块的解释通常由 DBMS 来完成。

不同操作系统提供了不同的服务方式，例如现代操作系统不仅提供进程管理还进一步发展了线程的概念和技术。操作系统不同的服务方式，导致了 DBMS 在实现技术和实现方法上的诸多差异。对于同一个操作系统，DBMS 也可以有不同的实现选择。

在诸多影响 DBMS 性能的因素中，DBMS 在操作系统上的运行方式尤为突出，例如 DBMS 进程结构方案的选择是 DBMS 实现中一个关键问题，它影响 DBMS 的整体设计思想和 DBMS 的性能，影响 DBMS 实现的难易程度。这些内容将在 11.2 节中详细讨论。

11.1.4 DBMS 和第四代应用开发环境

应该指出，DBMS 本身并不能全部解决数据管理的问题，DBMS 的设计目标也不是要解决所有问题。这些问题包括数据的存储、访问、处理和表示等。但是，它确实应该而实际上也已经为解决这些问题奠定了基础。DBMS 提供了丰富的数据管理的基本功能。以 DBMS 这些功能为核心，可以进一步开发工具软件，如报表书写、屏幕表格软件、菜单生成系统、图形显示系统等，从而构成了一组内部相互联系的 DBMS 家族，为用户提供了一个良好的第四代应用开发环境。这些软件系统拓广了数据库的应用领域，大大提高了应用开发的生产率。

上面对 DBMS 的系统目标、基本功能、与操作系统的关系以及与开发环境的关系等四个方面进行了简要的论述。可以看到，和操作系统、编译系统等系统软件相比，DBMS 具有跨度大、功能多的特点。从最底层的外存管理、缓冲区管理、数据存取操作、语言翻译处理到最外层的用户接口、数据表示、开发环境的支持都是它要实现的功能。

DBMS 的实现，既要利用操作系统、编译系统和网络通信的技术，又要突出对大量结构数据进行存储、管理和处理的特点，并要保证其存取数据和运行事务的高效率，是一个复杂而综合的软件设计过程。在下面几节，将逐步深入地讨论 DBMS 的进程结构、基本功能及语言处理等实现方法和技术。

11.2 DBMS 的进程结构和多线索机制

实际应用中，DBMS 以及应用程序都必须进入某个具体的操作系统环境。应用程序通过 DBMS，DBMS 通过 OS，与数据库中的数据打交道。根据三者的关系，一般有下述四种进程组织方案：

11.2.1 *N* 方案：DBMS 与应用程序相融合的方案

在这种方案中，*N* 个 DB 用户仅需 *N* 个进程，DBMS 作为应用程序的子程序被连入应用程序中，成为应用程序的一部分，因此这种方案也称为连入式方案，如图 11.1 所示。

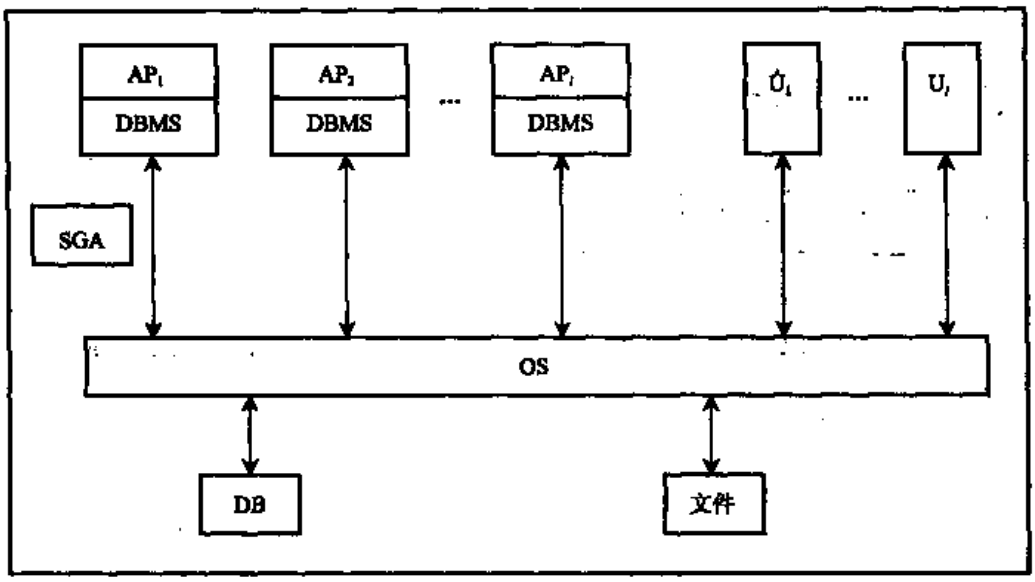


图 11.1 *N* 方案

其中 AP_i 为数据库系统应用程序；U_i 为非数据库用户（或应用程序）；SGA 为共享全局区（Shared Global Area），存放字典定义信息、数据和索引的缓冲块、日志缓冲块以及封锁控制块等全局资源信息。

N 方案最明显的特点是能用子程序调用方式把控制迅速直接地传送给

DBMS，从而减少应用程序的运行时间。与后面的几种方案相比，*N* 方案调用一次 DBMS 至少省掉两次进程切换开销。而在大多数操作系统中进程切换通常是很费时间的，大约需要数千条指令。

作为一种子程序技术，*N* 方案对内存的需求量比较大。因为用户程序与 DBMS 功能的融合使得 DBMS 的代码段无法共享，导致内存中保留了 DBMS 代码的许多副本。随着 DBMS 功能的日益增强，DBMS 代码会不断扩大，*N* 方案造成的代码冗余会使系统性能下降，甚至达到令人无法忍受的程度。

N 方案 DBMS 实现起来比较简单，不用考虑用户进程与 DBMS 进程间的通讯，因此在用户数少的小型 DBMS 中可以采用该方法。

解决 *N* 方案 DBMS 代码冗余的办法是将图 11.1 中的 AP_i 与其 DBMS 副本分开，让 DBMS 的代码段在内存中被共享，这就引出了下面的方案。

11.2.2 2*N* 方案：一个 DBMS 进程对应一个用户进程

每个用户进程均有一个 DBMS 进程为之服务，称这种 DBMS 进程为影子（Shadow）进程。在这种方案中，有 *N* 个用户进程就要启动 *N* 个 DBMS 进程（共 2*N* 个进程），因此这种进程结构方案称为 2*N* 方案，如图 11.2 所示。

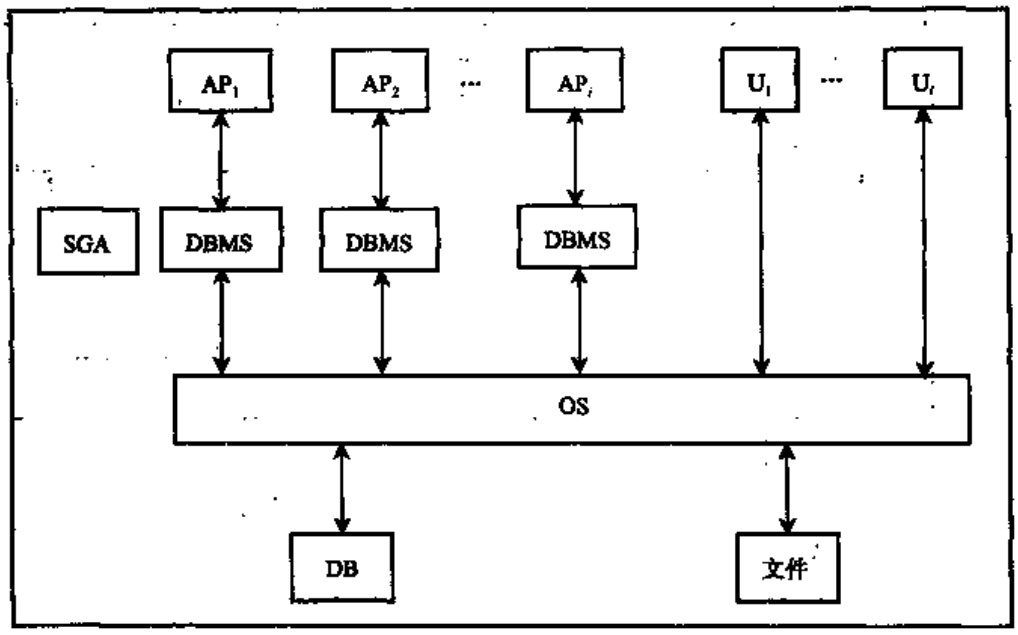


图 11.2 2*N* 方案

在这种情况下，数据库系统的各个活跃进程都是独立运行的。因此，用户进程与 Shadow 进程之间以及各 Shadow 进程相互间都要通信。

系统的全局管理数据驻留在 SGA 中，各 DBMS 进程靠操作系统提供的封

锁原语（信号量）实现同步或实现对 SGA 的互斥访问。因此 DBMS 的这种进程组织方式使进程间总的通信开销上升很多。

在 2*N* 结构的方案中，操作系统的负担增大了，主要表现在空间和时间两个方面。

在空间上，虽然各个 DBMS 进程的代码段（即 UNIX 中的正文段）可以共享，但数据段和栈段的空间还是各自的，再加上 OS 要为多出来的 *N* 个 Shadow 进程分配进程控制块（PCB）等诸多内部空间，因此 2*N* 结构对内存的需要量亦很大。这极易使内存页面状况变坏，并可能因物理内存远远小于各进程虚空间之和而导致各进程被操作系统频繁地换入换出（Swap in / Swap out），使系统出现颠簸。

在时间方面，由于 2*N* 方案中进程数目过多，许多 CPU 时间会白白浪费在进程切换这种昂贵的工作上。颠簸及进程切换的高额开销使系统性能严重下降。

2*N* 方案中增加一个用户所需要的资源很大，通常增加两三个用户就需要 1MB 的内存量，再加上操作系统所支持的进程总数有限，导致了 2*N* 方案不可能支持很多的 DBMS 用户，因此它不大适合于大量用户的 OLTP 应用。

除了每个用户进程需要一个 DBMS 服务进程外，DBMS 在全局上还要维护若干个后台服务进程。由这些后台进程通过 SGA 完成诸如监控、写数据库、预读、写日志、淘汰页面、死锁检测和解除等全局工作。使用后台进程大大增加了进程间通讯的开销，而频繁的进程通讯意味着频繁的进程切换。少数几个后台进程还可能成为整个系统的瓶颈。

此外，2*N* 方案至少还会碰到数据不在内存时所造成的性能问题和临界区（Critical Section）问题，这里就不再详细讨论了。

把多个 DBMS 进程合为一个 DBMS 进程可以有效地解决上述问题，这就是 *N*+1 方案。与 *N*+1 方案相比较，2*N* 方案的最大优势是它省掉了 DBMS 对各数据库用户的多任务调度，而把这个任务交给了操作系统，从而简化了用户进程与 DBMS 的接口。这就使得用 2*N* 方案开发 DBMS 更简单，更直观。

Oracle 7 之前的版本、Ingres 和 Informix 的早期版本使用 2*N* 方案（最早的 Ingres 采用 4*N* 方案：每个用户进程有三个分别完成不同任务的 DBMS 进程为之服务）。

11.2.3 *N*+1 方案：一个 DBMS 进程对应所有用户进程

进程结构的另一种组织方式是整个 DBMS 仅使用一个进程，该进程的行

为类似于一个服务器（Server）。多个数据库用户向 Server 发消息以申请数据库服务。Server 要用自己的机制来调度这些申请，以支持一个多任务的数据库系统。由于 N 个用户仅需要一个 DBMS 进程，因此称这种进程结构方案为 N+1 方案，如图 11.3 所示，这时 SGA 已不再需要了。

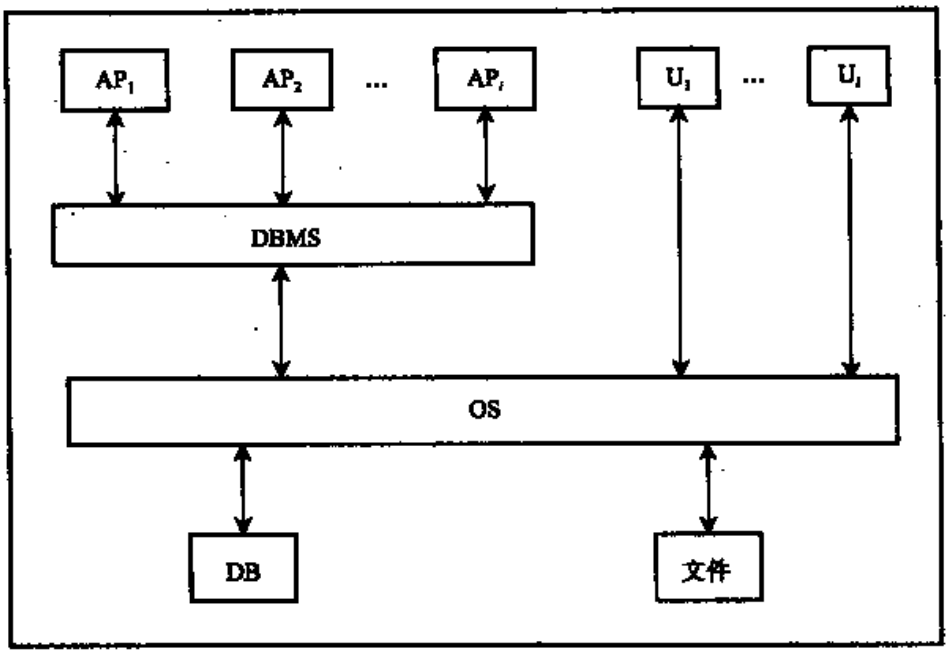


图 11.3 N+1 方案

由于要设计自己的多任务处理机制和调度算法，使得 N+1 方案的 DBMS 设计在整体上比 2N 方案要复杂。Server 要处理所有用户的申请，如果调度策略不当或不能使数据库进程获得较高的优先权，这种方案将导致瓶颈。

另外，N+1 方案中要用到操作系统级的消息机制以实现多进程向单进程以及单进程向多进程的数据传送。许多文献指出消息系统是一种昂贵的设施，大多数操作系统中消息往返一次的开销是几千条指令。

尽管如此，N+1 方案的许多诱人的优点仍使许多 RDBMS 厂家纷纷朝这个方向靠拢。

这种方案的另一个优点是可以采用多线索（Multi_Threaded）技术来实现 N+1 方案，从而可大大提高系统性能，降低系统资源的开销，简化 DBMS 许多部分的设计，这是因为线索机制使得 N+1 方案能用统一的“线索”思想处理所有执行流，其中包括 DBMS 核心的执行流，这将省去所有后台进程并能适应系统软件设计与开发的微内核（Micro_Kernel）要求，使 DBMS 的核心代码简明精巧、易于移植。

在 11.3 中将讲解多线索的概念和技术。

11.2.4 *N*+*M* 方案：*M* 个 DBMS 进程对应 *N* 个用户进程

N+*M* 方案采用 *M* 个 DBMS 进程为 *N* 个用户进程提供服务（一般 $M<N$）。如图 11.4 所示。

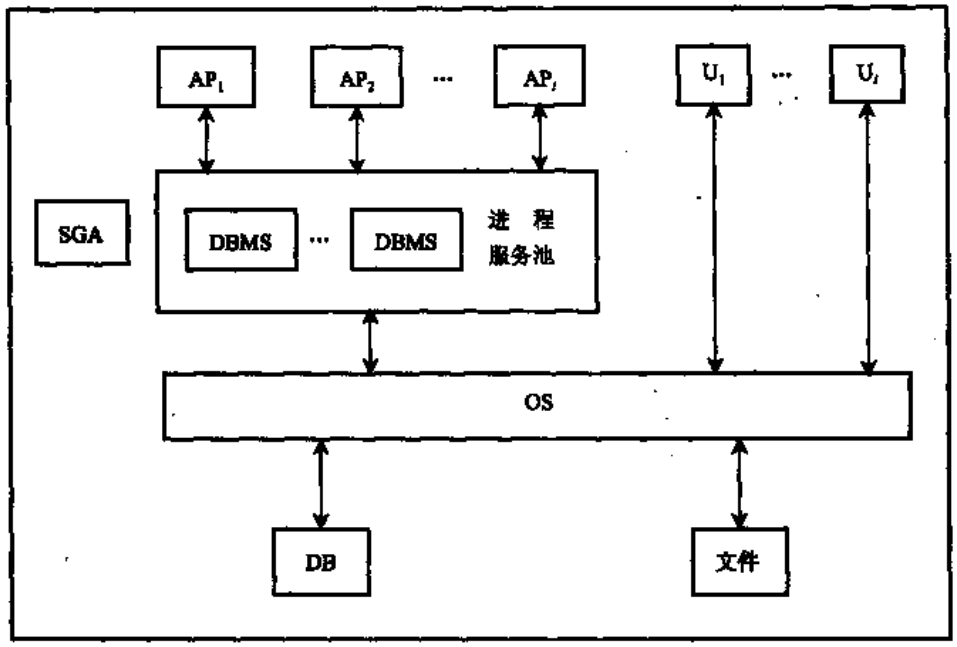


图 11.4 *N*+*M* 方案

在该方案中，DBMS 进程不负责多任务调度，同时每个用户进程也不固定地对应于某个 DBMS 进程。用户的数据库请求将会被动态地分配给某个 DBMS 进程来处理（例如最不忙的那个 DBMS 进程）。

DBMS 进程的分派由分派程序完成。分派程序可以是一个单独的后台进程，也可以将其代码连入用户程序（AP_i）中。分派程序通过 SGA 管理用户的数据库请求队列，并分配某个 DBMS 进程服务于某用户进程。分派程序还应监测整个 DBMS 的运行状况并根据用户请求队列的情况动态增减 DBMS 进程的个数。如果用户进程增加，则 DBMS 进程的个数也动态地增多，但后者一般总小于前者（即 $M<N$）。

N+*M* 方案基于的指导思想是：用户进程可能并不总是同时提出数据库请求，一个用户一个 DBMS 进程的方案太浪费了，几个用户轮流使用一个 DBMS 服务进程也许会节约一些。

N+*M* 方案是 2*N* 方案的一种改进，提高了内存资源的利用率，但它没有克服 2*N* 方案的本质弱点。此外，分派程序给系统增加了开销并可能成为瓶颈，而且 DBMS 进程动态增减的开销亦很大。

11.2.5 多线索（Multi_Threaded）DBMS 的概念

一、线程的概念

数据库系统中线索（Thread）的概念来源于操作系统中“线程”（Thread）的概念。这里先介绍操作系统中线程的概念。

随着多处理器系统以及并行计算技术的发展，进程概念被进一步细划为“任务”（Task）与“线程”的概念。

UNIX 核心中最关键的概念就是进程。进程既是资源分配的最小单位也是运行的最小单位。与一个进程相联系的资源有：进程虚拟空间、进程控制块、数据段、正文段、堆栈段、共享内存、程序计数器、状态寄存器及通用寄存器等。进程建立及切换的花销很大，核心维护每一个进程的虚拟空间花销也很大。进程概念在处理微粒度问题（一个问题可划分成多个工作量很小的并行计算单位）时显得很吃力，不能充分发挥多机的优点。出于以上考虑，1988 年卡内基-梅隆大学在其研制的多机操作系统 MACH 中实现了 Task 及 Thread 这两个概念。

他们将进程的概念一分为二：Task 是申请资源的最小单位，而 Thread 是调度和运行的最小单位。一个 Task 中可有多个 Thread，这些 Thread 共享 Task 的所有资源，共同完成一个任务。一个 Thread 只能存在于某个 Task 中。在多处理机上，这些 Thread 可真正地并行执行。UNIX 中的进程概念相当于 MACH 中的只包括一个 Thread 的 Task。图 11.5 说明了它们之间的区别。

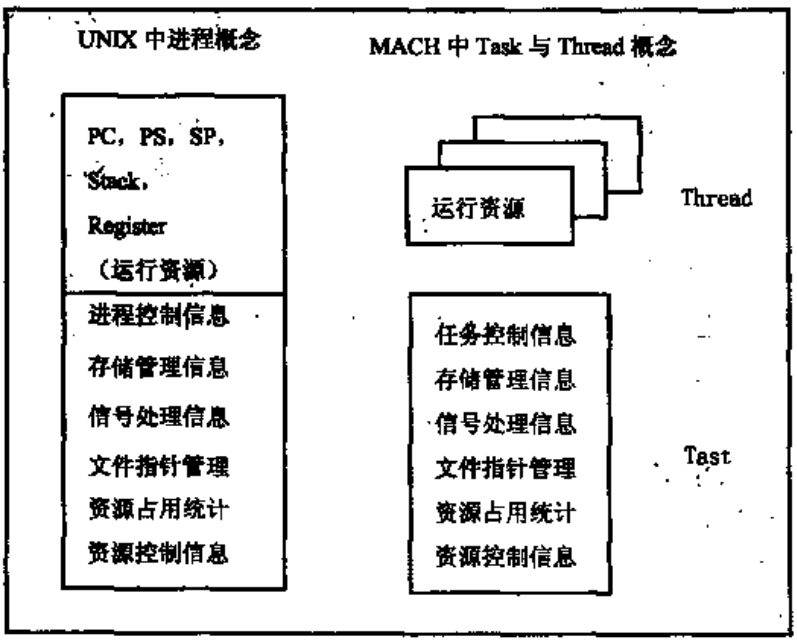


图 11.5 进程、Task 与 Thread

从图 11.5 中可以看出，由于多个线程运行于同一实体（任务或进程）中并且共享该实体所拥有的所有资源，使得传统 UNIX 中属于某一任务的多个进程（特别是个数较多情况下）以线程的方式并行执行时效率大大提高（所有线程共享一个地址空间等）。线程之间的切换效率更高，因此线程又被称为“轻权”或“轻量”进程(light_weighted process)。

可见，线程的基本概念就是将进程中的程序代码与进程所占资源相分离，从而在一个地址空间可以运行多个指令流。线程概念在表达应用问题本身的并行性方面较传统进程有较大优势，因而可以更好地发挥多处理机系统的能力。

二、多线索（Multi_Threaded）DBMS

数据库系统中的“线索”概念借鉴了操作系统中“线程”概念的含义：整个 DBMS 可以看作是一个 Task，当有一个用户申请数据库服务时，Task 分配至少一个 Thread 为之服务，多个 Thread 并行工作，共享资源。

一般地讲，DBMS 中的线索是 DBMS 的一个执行流，它服务于整个 DBMS 系统或 DBMS 中的某个用户；DBMS 服务器响应客户请求是通过为每个用户创建线索（而不是创建进程）来完成的。DBMS 的各个线索能在逻辑上并行执行；它们共存于一个进程中，共享 DBMS 的所有资源，如数据库缓冲区和 CPU 时间；线索是 DBMS 的调度单位，服务器进程能按一定的调度算法调度用户请求，由于调度优化策略是由数据库服务器执行，因此会比操作系统直接对这些请求进行调度高效得多。

多线索机制可以减少每个用户需要的系统资源，如内存要求，从而可以增加并发用户数，提高服务器的运行效率，更好地支持 OLTP 应用。

在支持线程的操作系统（如 MACH，Chorus，Amoeba 等）上实现单进程多线索 DBMS 是比较容易的，这种操作系统能给予 DBMS 较好的支持。

三、线索与进程的比较

为了进一步说明线索概念对于 DBMS 设计和实现的重要意义，下面对线索机制与进程机制作一个比较。

1. 线索比进程占用较少的资源

进程是程序的动态概念，指正在执行的程序。进程有其自身的程序地址空间，除正文段以外，进程间一般不能共享彼此资源。线索则是程序中的一串指令流，同一进程内的多个线索共享该进程内的一切资源。因此，线索比进程占用更少的系统资源，是更小的调度单位，可支持较细粒度的并行。

2. 线索调度比较灵活，可控制性强

线索的调度方式比较灵活，可以实现“智能”化的调度，提高并行效率。

例如，可以让紧急事务优先执行，可以设置线索优先级，可以使某一线索主动出让 CPU 等。

进程调度由于是由操作系统控制的，DBMS 无法干涉，而且进程调度是非“智能”的，特定进程可能会被频繁挂起。例如，假定一个进程要反复地读取数据、处理数据并回写数据，这样该进程会因等待 I/O 而被频频挂起，效率很低；但如果将该进程划分为读、处理、写等三个线索，这样当其中某个线索被阻塞时，其他线索可被调度执行，效率就会提高。而且，可以通过合理地调度一个进程内的多条并发线索，而使该进程长久地占用 CPU 。从这一意义上讲，无论单处理机还是多处理机，多线索结构均能提高执行效率。

3. 线索切换开销较小

就进程切换而言，由于进程的地址空间是私有的，因此进程切换必须保持进程的正文、数据、栈段及地址映射、进程状态、寄存器值等系统信息，一般需要花费数千条机器指令；而线索由于共享同一地址空间，因此当切换同一进程内的不同线索时，只需保存线索状态、寄存器值（程序计数器、堆栈指针、通用寄存器）等，由于切换较少的信息，线索切换的代码量明显减少，一般需数十条指令。

4. 线索间通信简便

UNIX 操作系统将进程的程序地址空间相隔离是为了避免并发进程间的相互干扰，以后为了支持进程间通信又先后引入了管道、通道、新 IPC 机制（信号量、消息及共享存储器）等通信手段。这些通信机制或者编程麻烦、不易使用，或者有一定局限性、效率不高。同一进程内的线索由于共享地址空间，因而可以借助全局变量名或指向局部变量名的指针来进行通信，这种通信当然简单而直接。不过，由于线索间可以互访数据，因此在利用线索编程时应注意保持数据的一致性和完整性。

总之，采用多线索机制的 DBMS 具有运行效率高、消耗系统资源少等显著优点。

11.3 DBMS 系统结构

在这一节中，首先从程序功能的角度分析 DBMS 的模块构成，然后用层次结构的观点介绍 DBMS 的系统结构，最后用一个例子说明 DBMS 的运行过程。

11.3.1 DBMS 程序模块的组成

作为一个庞大的系统软件，DBMS 由众多程序模块组成，它们分别实现 DBMS 复杂而繁多的功能。正如在前面所说的，不同的 DBMS 功能并不完全相同。大型系统功能完备，小型系统则常常对系统功能作了裁减。但大致说来，DBMS 的程序模块按功能划分如图 11.6 所示。

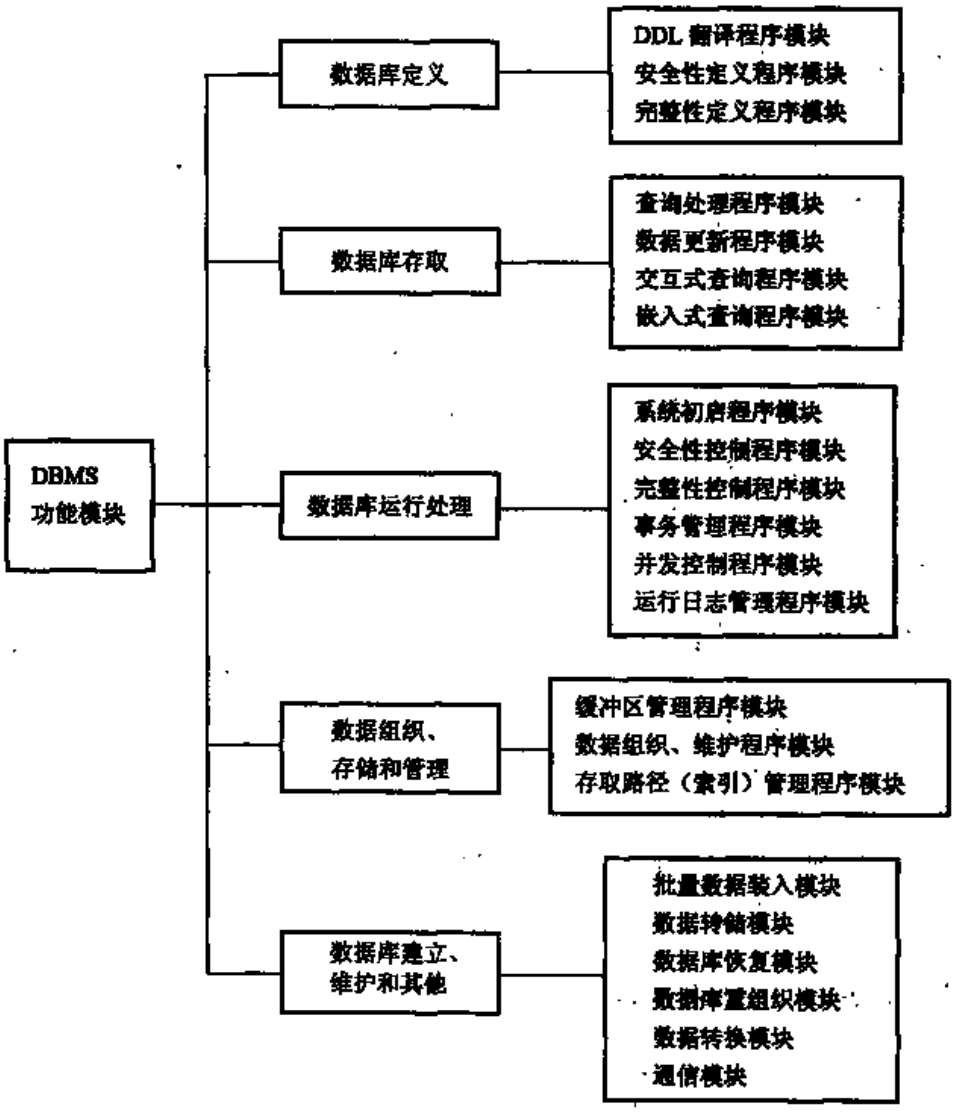


图 11.6 DBMS 程序模块结构

一、数据定义方面的程序模块

数据定义的程序模块主要包括：

- 模式、外模式、存储模式的定义模块，在 RDBMS 中就是创建数据库、

创建表、创建视图、创建索引等定义模块；

- 安全性定义，如授权定义及处理模块；
- 完整性定义，如主码、外码、其他完整性约束定义及处理模块。

这些 DDL 程序模块接收相应的定义，进行语法、语义检查，把它们翻译为内部格式存储在数据字典中。创建数据库的模块还根据定义，建立数据库的框架（即形成一个空库），等待装入数据。

二、数据操纵方面的程序模块

数据操纵的程序模块主要包括：

- 查询（Select 语句）处理程序模块
- 数据更新（增加、删除、修改）程序模块
- 交互式查询程序模块
- 嵌入式查询程序模块

这些程序模块对用户的数据操纵请求进行语法分析、语义检查，生成某种内部表示，通常是语法树。对于查询语句，要由查询优化器（模块）进行优化，如根据一定的等价变换规则把语法树转换成标准（优化）形式；对于语法树中的每一个操作根据存取路径、数据的存储分布、数据的聚簇等信息来选择具体的执行算法；最后生成查询计划(生成代码)交查询执行模块执行，完成对数据库的存取操作。

三、数据库运行管理方面的程序模块

主要有系统初启程序，负责初始化 DBMS，建立 DBMS 的系统缓冲区，系统工作区，打开数据字典等。还有安全性控制，完整性检查、并发控制、事务管理、运行日志管理等程序模块，在数据库运行过程中监视对数据库的所有操作，控制管理数据库资源，处理多用户的并发操作等。它们一方面保证用户事务的正常运行及其原子性，一方面保证数据库的安全性和完整性。

四、数据库组织、存储和管理方面的程序模块

有文件读写与维护程序、存取路径（如索引）管理和维护程序、缓冲区管理程序（包括缓冲区读、写、淘汰模块）等。这些程序负责维护数据库的数据和存取路径，提供有效的存取方法。

五、数据库建立、维护和其他方面的程序模块

有数据库初始装入程序、转储程序、恢复程序、数据库重构造程序、数据转换程序、通信程序等。

DBMS 的这些组成模块互相联系，互相依赖，共同完成 DBMS 的复杂功能。这些模块之间的联系也不是平面的，散乱无章的，正如下面要讲的，它们具有一定的层次联系。

11.3.2 DBMS 的层次结构

和操作系统一样，可以将 DBMS 划分成若干层次。清晰、合理的层次结构不仅可以更清楚地认识 DBMS，更重要的是大大有助于 DBMS 的设计和维护。许多 DBMS 实际上就是分层实现的。

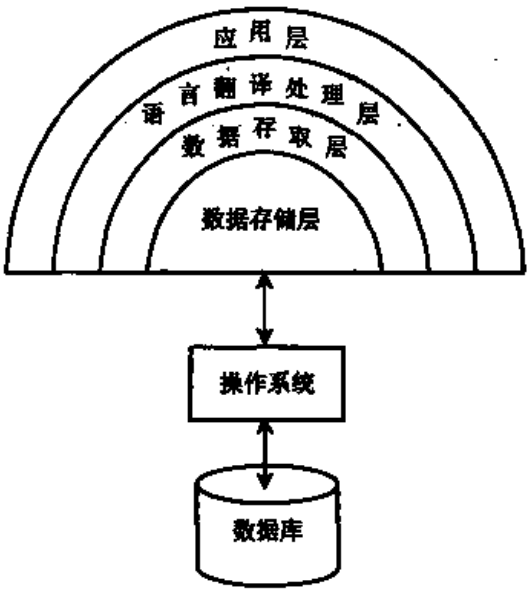


图 11.7 RDBMS 的层次结构

例如 IBM 公司最早研制的著名的 DBMS 实验系统 System R，其核心分为两层，即底层的关系存储系统 RSS 和上层的关系数据系统 RDS。RDS 本质上是一个语言翻译和执行层，完成包括语法检查与分析、优化、代码生成、视图实现、合法性检查等功能。RSS 则是一个存取方法层，其功能包括空间、设备管理，索引、存取路径管理，并发控制，运行日志管理和恢复等。

图 11.7 给出一个 RDBMS 的层次结构示例。这个层次结构是按照处理对象的不同，依最高级到最低级的次序来划分的，具有普遍性。图中包括了与 RDBMS 密切相关的应用层和操作系统。

最上层是应用层，位于 DBMS 核心之外。它处理的对象是各种各样的数据库应用，如用开发工具开发的或用宿主语言编写的应用程序，终端用户通过应用接口发出事务请求或各种查询要求。该层是 DBMS 与终端用户和应用程序的界面层。

第 2 层是语言翻译处理层。它处理的对象是数据库语言，如 SQL。向上提供的数据接口是关系、视图，即元组的集合。其功能是对数据库语言的各类语句进行语法分析、视图转换、授权检查、完整性检查、查询优化等。通过对下层基本模块的调用，生成可执行代码。这些代码的运行即可完成数据库语句的功能要求。

第 3 层是数据存取层。该层处理的对象是单个元组。它把上层的集合操

作转换为单记录操作。执行扫描（如表扫描）、排序、元组的查找、插入、修改、删除、封锁等基本操作。完成数据记录的存取、存取路径维护、并发控制、事务管理等工作。

第 4 层是数据存储层。该层处理的对象是数据页和系统缓冲区。执行文件的逻辑打开、关闭、读页、写页、缓冲区读和写、页面淘汰等操作，完成缓冲区管理、内外存交换、外存的数据管理等功能。

操作系统是 DBMS 的基础。它处理的对象是数据文件的物理块。执行物理文件的读写操作，保证 DBMS 对数据逻辑上的读写真实地映射到物理文件上。操作系统提供的存取原语和基本的存取方法通常作为和 DBMS 存储层的接口。

以上所述的 DBMS 层次结构划分的思想具有普遍性。当然具体系统在划分细节上会是多种多样的，这可以根据 DBMS 实现的环境，系统的规模灵活地处理。

下面将按照由外及内的次序在下面 11.4 节至 11.7 节中依次介绍图 11.7 中的各个层次。

11.3.3 DBMS 的运行过程示例

DBMS 是一个复杂而有序的整体，应该用动态的观点看待 DBMS 各个功能模块。为了加深对 DBMS 的了解，来考察一个应用程序或用户通过 DBMS 读取数据库中数据的过程。这个过程显示在图 11.8 中。

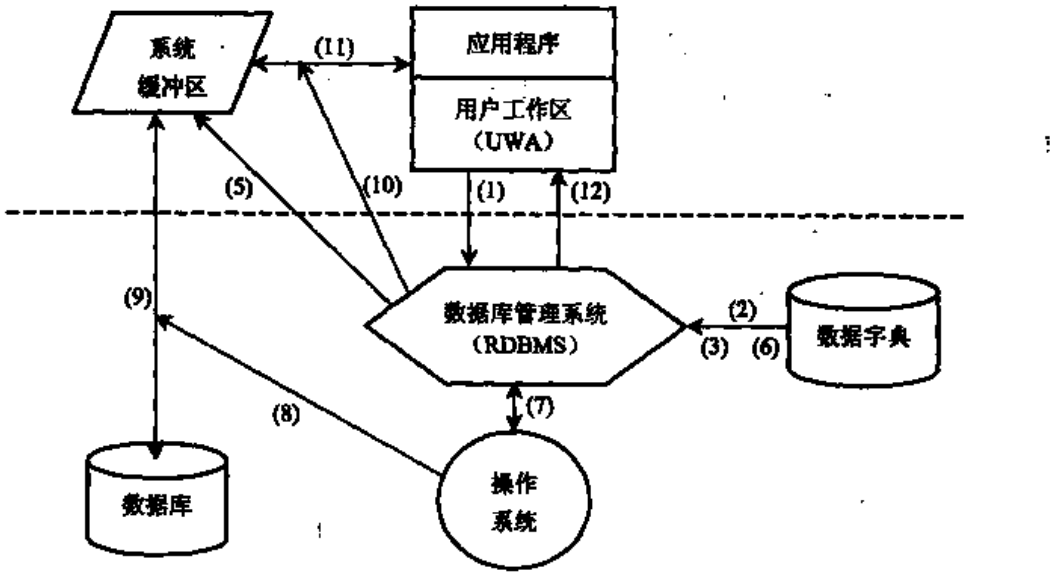


图 11.8 DBMS 的运行过程示例

图中的数据字典存储数据库定义，如子模式、模式、存储模式定义（关系数据库中相应于视图、基本表、索引、存储方式等的定义）。

(1) 用户 A 通过应用程序向 DBMS 发出调用数据库数据的命令。如 SELECT 命令。命令中给出了一个关系名和查找条件。

(2) DBMS 先对命令进行语法检查。语法检查通过后进行语义检查和用户存取权限检查。具体做法是，DBMS 读取数据字典，检查是否存在该关系及相应的字段，该用户能否读取它们等。确认语义正确、存取权限合法后便决定执行该命令。否则拒绝执行，返回错误信息。

(3) DBMS 执行查询优化。优化器要依据数据字典中的信息进行优化，并把该命令转换成一串单记录的存取操作序列。

(4) DBMS 执行存取操作序列（反复执行以下各步，直至结束）。

(5) DBMS 首先在缓冲区中查找记录，若找到满足条件的记录则转到 10，否则转到 6。

(6) DBMS 查看存储模式，决定从哪个文件，用什么方式读取哪个物理记录。

(7) DBMS 根据 6 的结果，向操作系统发出读取记录的命令。

(8) 操作系统执行读数据的有关操作。

(9) 操作系统将数据从数据库的存储区送至系统缓冲区。

(10) DBMS 根据查询命令和数据字典的内容导出用户所要读取的记录格式。

(11) DBMS 将数据记录从系统缓冲区传送到应用程序 A 的用户工作区。

(12) DBMS 将执行状态信息，如成功读取或不成功的错误指示、例外状态信息等返回给应用程序 A。

对照在 11.3.2 节中给出的层次结构模型，可以大致作如下的对应：

动作 1 属于第 1 层——应用层。

动作 2、3 由第 2 层——语言翻译处理层来完成。

动作 4、10、11、12 由第 3 层——数据存取层来完成。

动作 5、6、7 由第 4 层——数据存储层来进行。

动作 8、9 由操作系统执行。

整个 DBMS 各层模块互相配合、互相依赖共同完成对数据库的操纵。

对其他一些操作，如插入、删除、修改，其过程与读一个记录是类似的。例如修改记录，首先把要修改的记录找到，并把它放到用户工作区进行修改，然后把修改好的记录写回。DBMS 首先把记录写回系统缓冲区，交数据存储层的缓冲区管理模块处理。该模块按一定的调度策略在适当的时候（例如缓冲区已满需淘汰此块，或该用户事务结束）调操作系统执行物理写回操作。

插入和删除操作也都是先找到相应记录所在位置，再执行插入、删除操作。

11.4 语 言 处 理

11.4.1 语言翻译处理层的任务和工作步骤

一个好的数据库管理系统不仅应该在执行效率、功能齐全等方面是突出的，而且提供给用户的接口也应方便、灵活而且友好。使用户仅集中于解决自己的问题，而不必考虑数据管理等细节。这种要求已由非过程化的数据库语言达到了。前面已经介绍，目前的 DBMS 一般向用户提供多种形式的语言，如交互式命令语言（如 Oracle 的 SQL Plus）、嵌入一个至多个主语言的嵌入式语言(如嵌入式 SQL)、过程式语言（如 Oracle 的 PL/SQL）。这些语言都是直接由 DBMS 的语言翻译处理层来支持的。

语言翻译处理层的任务就是把用户在这各种方式下提交给 DBMS 的数据库语句转换成对 DBMS 内层可执行的基本存取模块的调用序列。

数据库语言通常包括 DDL，DML，DCL 三部分语句。DDL 语句的处理相对独立和简单。DML 和 DCL 则较为复杂。

具体来说，对 DDL 语句，语言翻译处理层首先把它翻译成内部表示，然后把它存储在系统的数据字典中。对 DCL 语句的定义部分，如安全保密定义，存取权限定义，完整性约束定义等的处理与 DDL 相同。

在 RDBMS 中数据字典通常采用和普通数据同样的表示方式，即也用 Table 来表示。数据字典包括关系定义表、属性表、视图表、视图属性表、视图表达式表、用户表、存取权限表等。图 11.9 给出了一个 RDBMS 数据字典的一部分示意图。

数据字典是 DML 语句处理、执行以及 DBMS 运行管理的基本依据。

对 DML 语句，语言翻译处理层要做的工作比较多，图 11.10 给出了 RDBMS 中 DML 语句处理过程的示意。

首先，对 DML 语句进行词法分析和语法分析，并把外部关系名、属性名转换为内部名。外部名便于用户记忆和使用，内部名则整齐划一。在符号名转换过程中需存取数据字典。词法和语法分析通过后便生成语法分析树。

接着，根据数据字典中的内容进行查询检查，包括审核用户的存取权限和完整性检查。

对那些具有存取谓词的存取权限，它们可能与数据的值有关，则此时不能确定该语句能否执行，于是还要生成相应的动作，以便运行时检查。

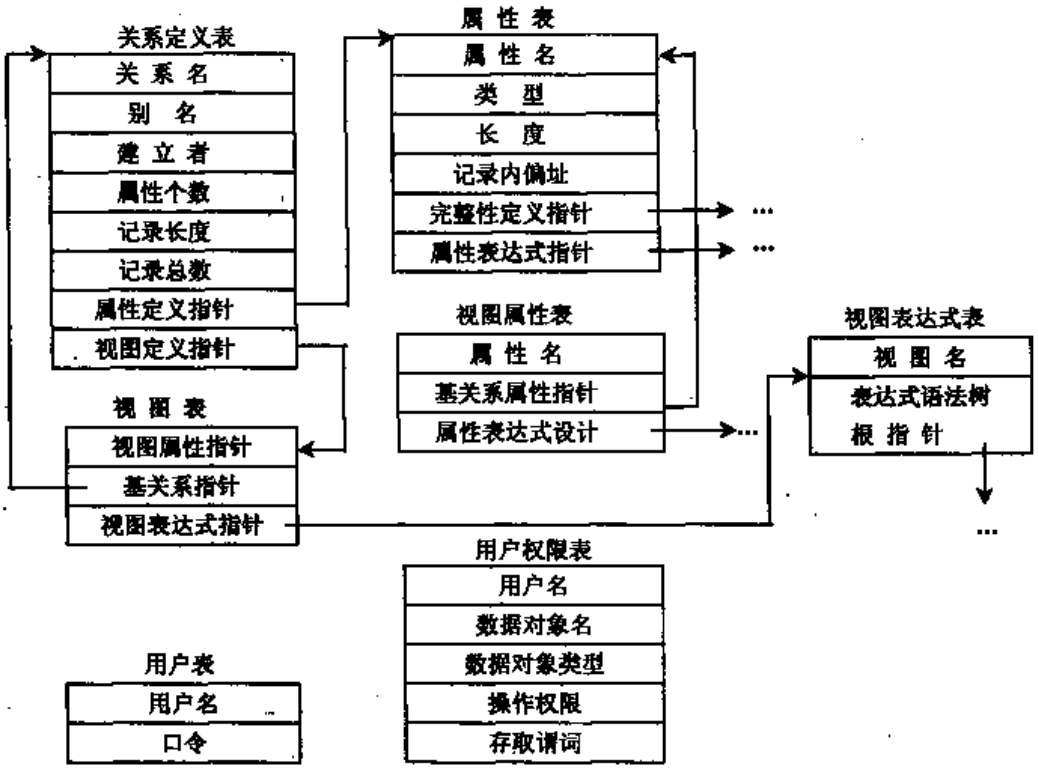


图 11.9 RDBMS 数据字典的部分示意图

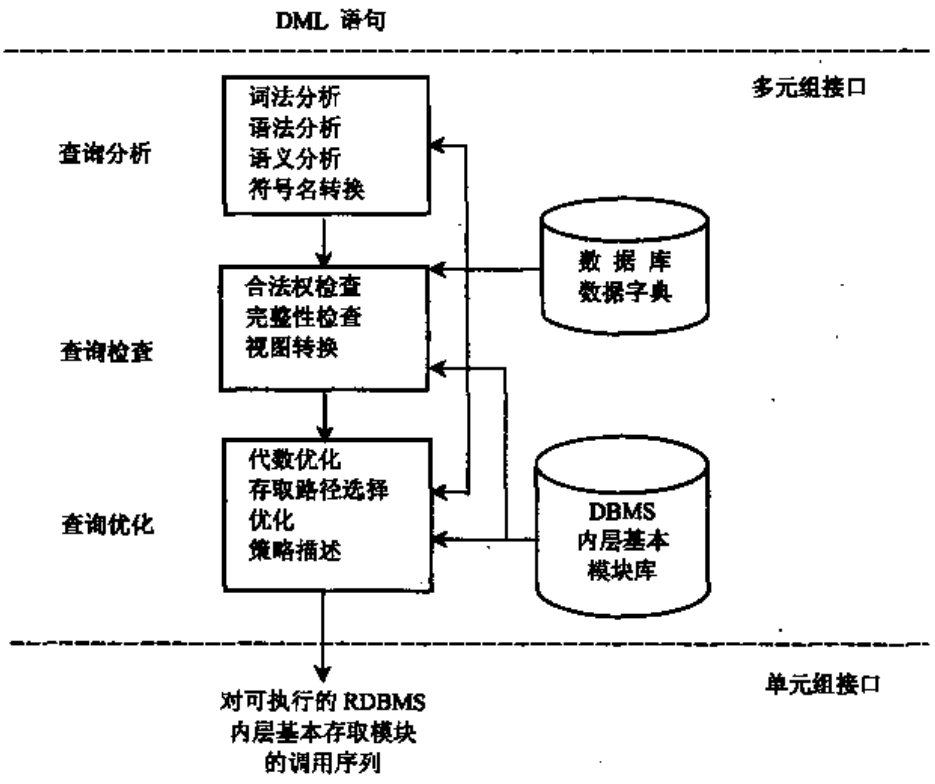


图 11.10 RDBMS 中 DML 语句的处理过程——束缚过程

完整性检查是查询检查的重要内容。DBMS 参照数据字典中完整性约束规则，这时只是进行部分的静态约束检查，如检查数据的类型、范围是否符合数据定义。许多完整性约束条件是在执行时检查的。例如，实体完整性约束，是在执行数据插入时检查的，即检查插入的元组其主码是否已经存在，以保证主码的唯一性。同样，这时也要检查参照完整性约束（若有的话）。还有，对某些动态完整性规则，它们与数据值和执行过程有关，则也要在操作执行时进行检查。

查询检查还包括视图消解，也称为视图转换。视图转换是指，若 DML 语句涉及对视图的操纵，则首先要从数据字典中取出视图的定义，根据该定义把对视图的操作转换为对基本表的操作。

然后，对查询进行优化。优化分为两类，一类为代数优化，另一类为存取路径优化。后者要根据数据字典中记载的各种信息，按照一定的优化策略选择一个系统认为是“较好”的存取方案，并把选中的方案描述出来。

综上所述，将数据库 DML 语句转换成一串可执行的存取动作的这一过程，称为一个逐步**束缚的**（Bind）过程。它将 DML 高级的描述型语句（集合操作）转换为系统内部的低级的单元组操作，将具体的数据结构、存取路径、存储结构等结合起来，构成了一串确定的存取动作。

在各种具体的 RDBMS 中，这一束缚过程基本一致，但进行这一过程的时间则有所不同。有的在运行中进行，有的在运行前进行，于是形成了两种基本的翻译方法：解释方法和预编译方法。

11.4.2 解释方法

早期的一些数据库系统（如 ADABAS 系统）采用一种系统处理效率最高的方法来处理对数据库的存取。这种方法束缚时间最早，事实上是在应用程序设计时由应用程序员进行束缚。它要求应用程序员为每个存取请求列出详细的参数，如数据的内部名、类型操作码等，因此程序员的负担重、效率低、数据独立性差。这种方法已被淘汰。

随后的一些数据库系统（如 DBASE III）对上述方法进行了改进，通过尽量推迟束缚过程来赢得数据独立性。具体做法是：直到执行前，数据库 DML 语句都以原始字符串的形式保存。当执行到该语句时，才利用解释程序去完成图 11.8 中所示的全部过程，同时予以执行。这种方法被称为解释执行方法。

解释方法具有灵活、应变性强的优点，甚至能适应在解释过程中发生的数据结构、存储结构等的变化，因此能保持较高的数据独立性。但由于每次执行一个 DML 语句时都要执行图 11.10 中的所有步骤，尤其当这样的语句位于一个循环体内时，就要多次重复解释一个 DML 语句，开销会很大，因此效率比较低。

随着数据库技术的发展，这种方法已逐步被预编译方式所取代。

11.4.3 预编译方法

由上述可见，将束缚过程提前固然可达到系统的高效率，但失去了数据库系统的一个主要优点：数据独立性；将束缚时间推迟，赢得了数据独立性，却牺牲了执行高效率的优点。

预编译方法就是为了克服它们的缺点，保持两者优点而提出的。其基本思想时：在用户提交了 DML 语句之后，在运行之前对它进行翻译处理，保存产生好的可执行代码。当需要运行时，取出保存的可执行代码加以执行。

下面以 DB2 为例说明这种方法。图 11.11 描述了 DB2 处理带有 SQL 语句的应用程序的过程。

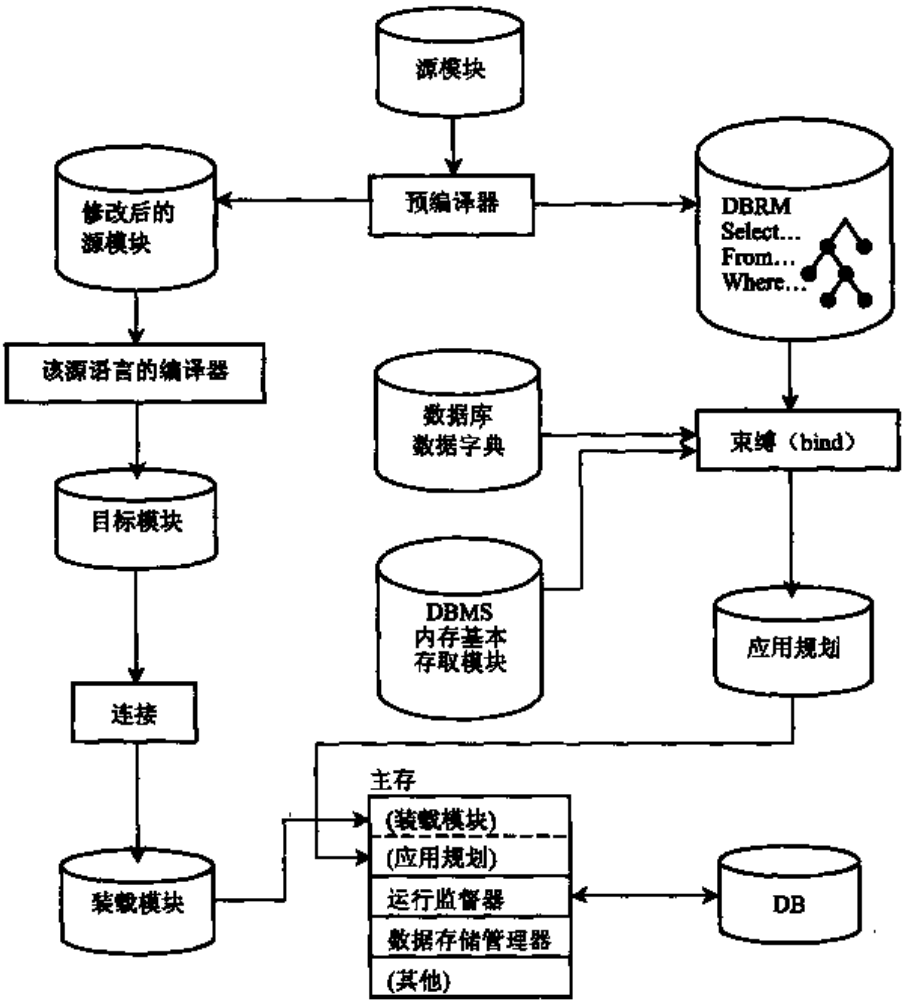


图 11.11 DB2 处理带有 SQL 语句的宿主语言源程序的过程

假定被处理的对象是一个 C 语言源程序模块 P，它首先被一个预编译器处

理。这个预编译器的功能是识别 SQL 语句，把它们替换成源语言的 CALL 语句（原来的 SQL 语句作为注解放在修改后的源程序中，以备恢复和阅读）。预编译器还取出 SQL 语句，将每个 SQL 语句生成一个数据库请求模块 DBRM（Data Base Request Module），它是下一步束缚过程的输入。

修改后的 C 源程序按通常方式被编译、连接（Linkage Editor）处理，生成装载模块。

束缚过程类似图 11.10 执行的功能。它实际上是一个 SQL 语句的优化编译器，编译一个至多个 DBRM 生成应用规划。应用规划包含了对基本存取模块的调用，完成 DBRM 对应的源 SQL 语句的功能。

这样，一个 C 源程序现在分成了两部分：装载模块和应用规划。

在运行时，当装载模块执行过程中遇到数据库调用的 CALL 语句后，就把控制权交给运行监督器。运行监督器找到相应的应用规划后将其调入内存并转让控制权，然后由应用规划去启用数据存储管理器执行实际的数据存取，并将结果返回给 C 程序。

但是，使用这种方法会遇到这样的问题：在束缚过程中进行优化所依据的条件可能在运行前已不存在，或者数据库结构已被修改，因而导致已作出的应用规划在执行时不再有效。例如，假设在束缚过程中决定使用某一索引来加快存取速度，而在该程序编译完成之后，运行之前，该索引被删除了，那么，运行时就会出现不可预测的现象。

为了解决这类问题，采用了重编译方法。即当数据库中因某些成分的改变而使一些程序（语句）的编译结果无效时，再对它们执行一次编译。重编译可在不同时刻进行。为了提高整个系统的效率，不应在数据库某一成分改变后就马上对受影响的那些源程序重编译，较好的方法是将受影响的编译结果置“无效”标志，在其被执行时才进行自动重编译。

自动重编译技术使得预编译方法既拥有了编译时进行束缚所带来的高效率，又具备了执行时束缚带来的数据独立性。实践证明，预编译方法的效率比其他方法高两倍以上。

有些语言的编译器将主语言和嵌入的数据库语言进行一体化编译，即没有 DB2 系统中分离的预处理部分（预编译器和束缚过程）。这种方法常称为增强编译法，究其实质，它和 DB2 中所用的预编译方法是一样的。

11.5 数据存取层

按照 11.3.2 节中 DBMS 的层次结构，下面介绍数据存取层的结构以及它

的各个功能子系统。

数据存取层介于语言翻译处理层和数据存储层之间。它向上提供单元组接口，即导航式的一次一个元组的存取操作。向下则以系统缓冲区的存储器接口作为实现基础，其接口关系如图 11.12 所示。

图中给出了每个层次中操作对象的数据结构。例如，数据存取层所涉及的主要数据结构为逻辑数据记录、逻辑块、逻辑存取路径。

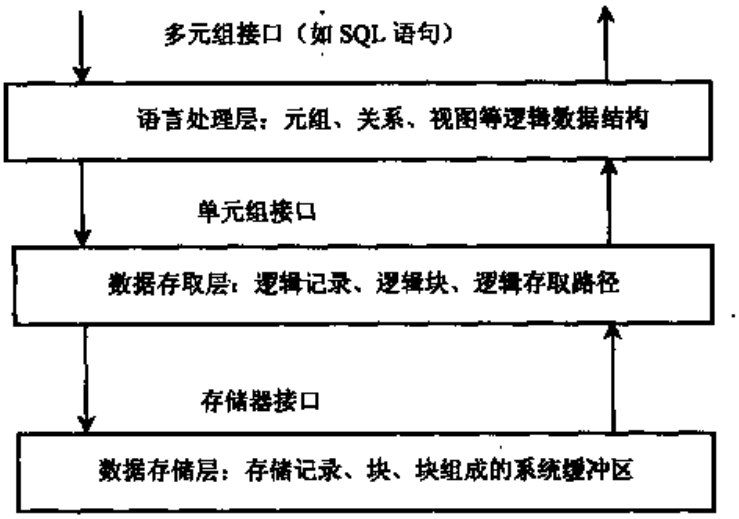


图 11.12 数据存取层和它的上下接口关系

存取层的任务主要包括：

（1）提供一次一个元组的查找、插入、删除、修改等基本操作。

（2）提供元组查找所循的存取路径以及对存取路径的维护操作。如对索引记录的查找、插入、删除、修改。若索引是采用 B+ 树，则应提供 B+ 树的建立、查找、插入、删除、修改等功能。

（3）对记录和存取路径的封锁、解锁操作。

（4）日志文件的登记和读取操作。

（5）其他辅助操作，如扫描、合并/排序，其操作对象有关系、有序表、索引等。

为了完成上述功能，通常把数据存取层又划分为若干功能子系统加以实现。

11.5.1 数据存取层的系统结构

数据存取层包括许多功能，在实际的 DBMS 中由多个功能子系统来完成。图 11.13 是数据存取层的系统结构，它包括下列子系统和模块：

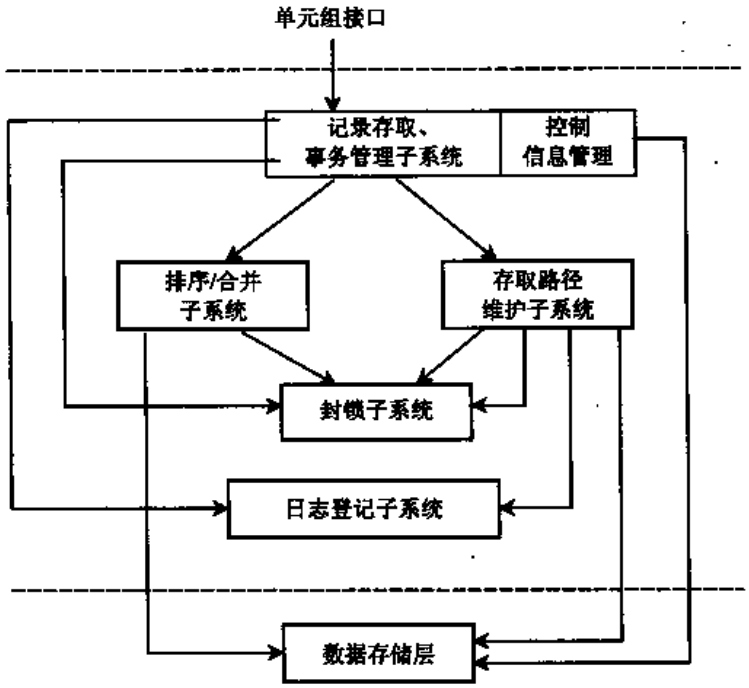


图 11.13 数据存取层的结构

- 记录存取、事务管理子系统。
- 控制信息管理模块。
- 排序/合并子系统。
- 存取路径维护子系统。
- 封锁子系统，执行并发控制。
- 日志登记子系统，用以执行恢复任务。

这些子系统相互配合，紧密联系，构成了一个完整的数据存取系统。

11.5.2 数据存取层的功能子系统

数据存取层中许多子系统的功能，如封锁子系统、日志登记子系统等已在第七、八章中做了介绍，下面只做扼要介绍。

一、记录存取、事务管理子系统

数据存取层不涉及存储分配、存储结构及有关参数，只在数据的逻辑结构上操作，因而可以把各种物理实现形态隐蔽起来。

记录存取子系统提供按某个属性值直接取一个元组和顺序取一个元组的存取原语。这种存取运算是按已选定的某个逻辑存取路径进行的，如某个数据文

件或某一个索引。这类存取操作的例子有：

- 在某个存取路径上按属性值找元组（FIND）。
- 按相对位置找元组（NEXT，PRIOR，FIRST，LAST）。
- 给某关系增加一个元组（INSERT）。
- 从找到的元组中取某个属性值（GET）。
- 从某关系中删去一个元组（DELETE）。
- 把某修改完的元组写回关系中（REPLACE）。

事务管理子系统提供定义和控制事务的操作。数据库中事务是并发控制和恢复的单位。事务管理的基本操作有：

- 定义事务开始（BEGIN TRANSACTION）。
- 事务提交（COMMIT）。
- 事务滚回（ROLLBACK）。

事务管理子系统提供的这些操作将登记进日志文件中。

二、日志登记子系统

日志登记子系统和事务管理子系统紧密配合，完成 DBMS 对事务和数据库的恢复任务，它把事务开始、滚回、提交；对元组的插入、删除、修改；对索引记录的插入、删除、修改等每一个操作作为一个日志记录存入日志文件中。当事务或系统软、硬件发生故障时利用日志文件执行恢复。与日志文件有关的主要操作有：

- 写日志记录（WRITELOG）。
- 读日志记录（READLOG）。
- 扫描日志文件（SCANLOG）。
- 撤销尚未结束的事务（UNDO）。
- 重做已经结束的事务（REDO）。

三、控制信息管理模块

该模块利用专门的数据区（内存中）登记不同记录类型以及不同存取路径的说明信息（取自数据字典）和控制信息。这些信息是存取元组和管理事务的依据。它和事务管理、记录存取子系统一起保证事务的正常运行。

该模块提供对数据字典中说明信息的读取、增加、删除和修改操作。

四、排序/合并子系统

在语言翻译处理层中，描述性语言表达的集合级操作被转换成一系列的对数据存取层所提供的存取原语的调用。为了得到用户所要求的有序输出，为了

加速关系运算（如自然连接）的中间步骤，为了提高效率，常常需要对关系元组重新排序。这一工作由排序/合并子系统来完成。下面列举排序操作的若干主要用途。

1. 输出有序结果

例如，用户提出如下查询要求：

```
SELECT  Eno，Salary
FROM    EMP
ORDER BY  Salary  DESC;
```

若 EMP 表上的 Salary 属性已建有索引(如 B+树)，则可以顺序扫描索引获得要求的输出。若 Salary 上没有索引，则必须对 EMP 表按 Salary 的属性值重新降序排序以得到所要的结果。

2. 删去重复值

删去重复值（简称删重）是并、交、差、属于、不属于等关系运算中常常首先要做的预处理。对无序数据，删重的代价一般为 O（n^2）数量级，比较昂贵。因此，用先排序后删重或边排序边删重的方法可以降低开销。

3. 支持合并扫描的连接操作

对两个无序关系的连接运算有时可以用合并扫描法来提高连接运算速度。即先对两个关系按连接属性值排序，然后对两个有序关系执行连接。

4. 支持动态建立索引结构

数据库中常用的索引结构有 B+树。B+树的叶页的索引记录形式为（码值，TID）。其中 TID 为元组标识符。TID 可用元组逻辑记录号，或主码值，或数据块号加位移等来表示。索引记录在 B+树的叶页上是顺序存储的，因此在初建 B+树索引时首先要对（码值，TID）排序。

5. 减少数据块的存取次数

通过 B+树索引存取元组时，首先得到（码值，TID）集合，然后根据 TID 存取相应的元组。当 TID 是用数据块号加位移来表示时，可以首先对 TID 排序，使相同或临近块号的 TID 聚集在一起，然后按数据块号顺序存取物理数据块，避免无序状态下重复读块的情况，减少数据块的存取次数。

排序操作的用途还很多，这里不一一列举了。

由此可见，排序操作是数据存取子系统和存取路径维护子系统都要经常调用的操作。它对提高系统效率具有关键的作用。因此，排序子系统的设计十分重要，应尽量采用高效的排序算法。

五、存取路径维护子系统

对数据执行插入、删除、修改操作的同时要对相应的存取路径进行维护。

例如，若用 B^+树索引作为存取路径，则对元组进行插入、删除、修改操作时要对该表上已建立的所有 B^+ 树索引进行动态维护，插入、删除相应的索引项。否则，就会造成 B+树索引与数据库表的不一致，当再通过 B+树索引结构存取元组时便会造成操作失败或错误结果。B+树的维护算法已广为人知，读者可参考数据结构教科书的有关部分。

六、封锁子系统

封锁子系统完成并发控制功能。有关封锁的概念、封锁的类型、封锁相容矩阵、死锁处理、可串行性准则、两段锁协议等已在第八章详细讨论，这里只说明两点：

第一，在操作系统中也有并发控制问题，其实现并发控制的方法通常也采用封锁技术。DBMS 封锁技术和操作系统的封锁技术相比（表 11.1），内容更加丰富，技术更加复杂。

第二，DBMS 中封锁子系统设计的难点不仅在于技术复杂，而且在于其实现手段依赖于操作系统提供的环境。例如，封锁表的设计。由于封锁表必须能为多个进程共享，能动态建立和释放，因此封锁表的设计就随操作系统环境而异。它是封锁子系统设计的关键。

表 11.1 操作系统和数据库管理系统封锁技术的比较

	操作系统	数据库管理系统
封锁对象	单一，系统资源（包括 CPU、设备、表格等）	多样，数据库中各种数据对象（包括用户数据、索引（存取路径）、数据字典等）
封锁对象的状态	静态、确定、各种封锁对象在封锁表中占有一项。封锁对象数是不变的	动态，不确定。封锁对象动态改变着、常常在执行前不能确定。一个封锁对象只有当封锁时才在封表中占据一项
封锁的粒度	不变，由于封锁对象单一、固定，封粒度不会改变	可变，封锁可加到或大或小的数据单位上，封锁粒度可以是整个数据库、记录或字段
封锁的类型	单一，排它锁	多样，一般有共享锁（S Lock）、排它锁（X Lock）或其他类型的封锁，随系统而异

11.6 缓冲区管理

数据存取层的下面是数据存储区（简称存储层）。存储层的主要功能是存储管理，包括缓冲区管理、内外存交换、外存管理等。其中缓冲区管理是最主要的。存储层向存取层提供的接口是由定长页面组成的系统缓冲区。

系统缓冲区的设立是出于两方面的原因：一是它把存储层以上各系统成分

和实在的外存设备隔离。外存设备的变更不会影响其他系统成分，使 DBMS 具有设备独立性。二是提高存取效率。DBMS 利用系统缓冲区滞留数据。当存取层需要读取数据时存储子系统首先到系统缓冲区中查找。只有当缓冲区中不存在该数据时才真正从外存读入该数据所在的页面。当存取层写回一元组到数据库中时，存储子系统并不把它立即写回外存，仅把该元组所在的缓冲区页面作一标志，表示可以释放。只有当该用户事务结束或缓冲区已满需要调入新页时才按一定的淘汰策略把缓冲区中已有释放标志的页面写回外存。这样可以减少内外存交换的次数，提高存取效率。

系统缓冲区可由内存或虚存组成。由于内存空间紧张，缓冲区的大小、缓冲区内存和虚存部分的比例要精心设计。针对不同的应用和环境按一定的模型进行调整。既不能让缓冲区占据太大内存空间，也不能因其空间太小而频频缺页调页，造成“抖动”，影响效率。

图 11.14 给出了缓冲区及上下接口示意图。

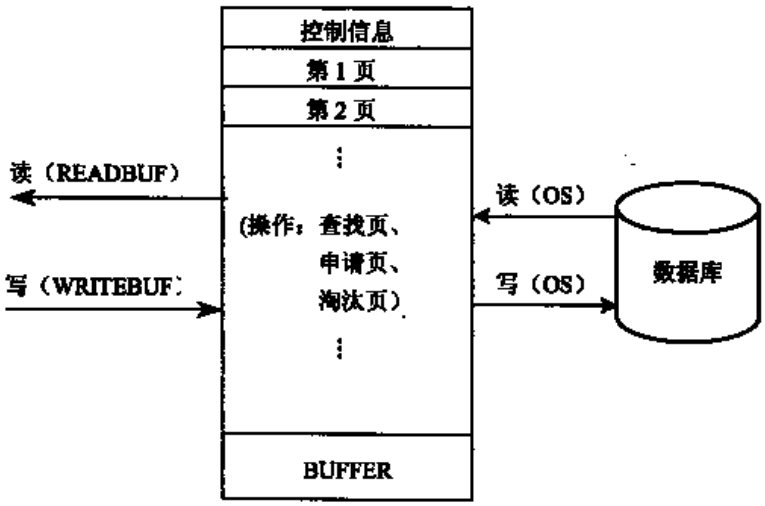


图 11.14 数据库缓冲区及上下接口

缓冲区由控制信息和若干定长页面组成。缓冲区管理模块向上层提供的操作是缓冲区的读（READBUF）、写（WRITEBUF）。缓冲区内部的管理操作有：查找页、申请页、淘汰页。缓冲区管理调用操作系统的操作有：读（READ）、写（WRITE）。以读操作为例，缓冲区管理的大致过程如图 11.15 所示。可以看到缓冲区管理中主要算法是淘汰算法和查找算法。操作系统中许多淘汰算法可以借鉴，如 FIFO（先进先出算法），LRU（最近最少使用的先淘汰算法）以及它们的各种改进算法。查找算法用来确定所求的页是否在内存，可采用顺序扫描、折半查找、Hash 查找算法等。

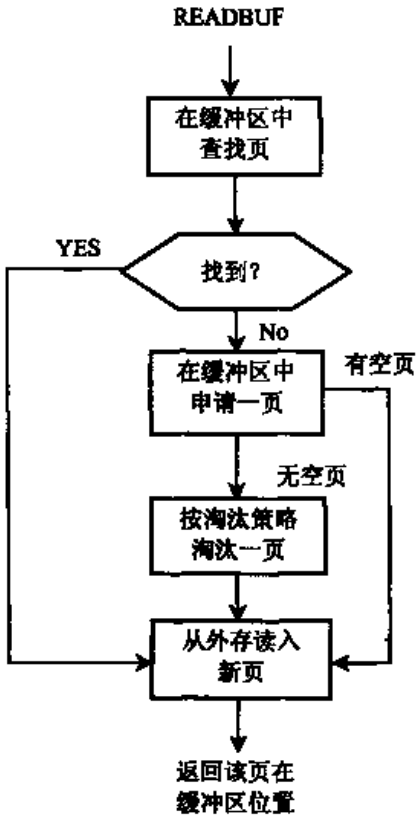


图 11.15 缓冲区管理示意图（读操作）

11.7 数据库物理组织

从 DBMS 最外层语言翻译处理层开始逐步深入地讨论了数据存取层、数据存储层各层的系统结构、功能、任务以及实现技术。现在进入数据库本身，介绍数据库的物理组织。

数据库是大量数据的有结构的综合性集合。如何将这样一个庞大的数据集合以最优的形式组织起来存放在外存上是一个非常重要的问题。所谓“优”应包括两方面：一是存储效率高，节省存储空间；二是存取效率高，速度快，代价小。

数据库实现的基础是文件，对数据库的任何操作最终要转化为对文件的操作。所以在数据库物理组织中，基本的问题是如何设计文件组织或者利用操作系统提供的基本的文件组织方法。

这里不打算讨论文件组织的基本方法和常用的文件组织形式，这些内容在数据结构课程中已介绍。这一节主要讨论如何利用基本的文件组织方法来实现数据库组织。

数据库系统是文件系统的发展。文件系统中每个文件存储同质实体的数据，各文件是孤立的，没有体现实体之间的联系。数据库系统中数据的物理组织必须体现实体之间的联系，支持数据库的逻辑结构——各种数据模型。因此数据库中要存储四个方面的数据：

- 数据描述。即数据外模式、模式、内模式；
- 数据本身；
- 数据之间的联系；
- 存取路径。

这四个方面的数据内容都要采用一定的文件组织方式组织、存储起来。

一、数据字典（DD）的组织

有关数据的描述存储在数据库的数据字典中。数据字典的特点是数据量比较小（与数据本身比）、使用频繁，因为任何数据库操作都要参照数据字典的内容。数据字典在网状、层次数据库中常常用一个特殊的文件来组织。所有关于数据的描述信息存放在一个文件中，例如 HP3000 计算机系统中 IMAGE 网状数据库的模式就是用一个称为“根文件”的特权文件来存放的。关系数据库中数据字典的组织通常与数据本身的组织相同。数据字典按不同的内容在逻辑上组织为若干张表（如图 11.9 所示），在物理上就对应若干文件而不是一个文件。由于每个文件中存放数据量不大，可简单地用顺序文件来组织。

二、数据及数据联系的组织

关于数据自身的组织，DBMS 可以根据数据和处理的要求自己设计文件结构，也可以从操作系统提供的文件结构中选择合适的加以实现。目前，操作系统提供的常用文件结构有：顺序文件、索引文件、索引顺序文件、HASH 文件（杂凑文件）和 B 树类文件等。

数据库中数据组织与数据之间联系是紧密结合的。在数据的组织和存储中必须直接或间接、显示或隐含地体现数据之间的联系，这是数据库物理组织中主要考虑和设计的内容。

在网状和层次数据库中，常用邻接法和链接法实现数据之间的联系。对应到物理组织方式中，就要在操作系统已有的文件结构上实现数据库的存储组织和存取方法。例如 IMS 数据库中，操作系统提供的低级的存取方法有 SAM，ISAM，VSAM，OSAM。IMS 数据库管理系统在此基础上设计了 HSAM，HISAM，HDAM，HIDAM 四种数据库的存储组织和相应的存取方法。其中 HSAM 层次顺序存取方法按照片段值的层次序列码的次序顺序存放各片段值。而层次序列码正体现了数据之间的父子和兄弟联系。这是一种典型的按物理邻

接方式实现数据之间联系的方法。在这种存储方法中，整个数据库中不同片段型的数据均存储在一个 SAM 文件中。

网状数据库中最常用的组织策略是：各记录型分别用某种文件结构组织，记录型之间的联系——SET 用指引元方式实现。即在每个记录型中，增加 DBMS 控制和维护的系统数据项——指引元，它和用户数据项并存于同一个记录中。

关系数据库中实现了数据表示的单一性。实体及实体之间的联系都用一种数据结构——“表”来表示。在数据库的物理组织中，每一个表通常对应一种文件结构。因此数据和数据之间的联系两者组织方式相同。

三、存取路径的组织

在网状和层次数据库中，存取路径是用数据之间的联系来表示的，因此已与数据结合并固定下来。

关系数据库中，存取路径和数据是分离的，对用户是隐蔽的。存取路径可以动态建立、删除。存取路径的物理组织通常采用 B 树类文件结构和 HASH 文件结构。在一个关系上可以建立若干个索引。有的系统支持组合属性索引，即在两个或两个以上属性上建立索引。索引可以由用户用 CREATE INDEX 语句建立，用 DROP INDEX 删除。在执行查询时 DBMS 查询优化模块也会根据优化策略自动地建立索引，以提高查询效率。由此可见，关系数据库中存取路径的建立是十分灵活的。

四、一个实际系统的例子

数据库物理组织除了要把数据字典、数据、数据之间的联系和存取路径以一定的文件结构组织存储起来之外，还要考虑数据库存储空间的组织。

在一些微机 RDBMS 中，一个关系或一个 B 树索引就对应一个文件，文件存储空间的组织和管理由操作系统负责。而在一些大型的 DBMS 中存储空间的组织和管理并不是这样简单。这里举一个 Oracle 关系数据库管理系统的例子来说明这个问题。

在 Oracle 系统中（无论是大型、中型或微型机版本），数据库存储空间由 RDBMS 来管理。一个 Oracle 数据库逻辑上由一个或多个分区（Partition）组成。分区中至少有一个系统分区（System Partition）。数据字典存储在系统分区中。一个分区物理上由一个或多个物理文件组成。一个表可以分跨在多个物理文件上。表是数据库中存放数据的基本单位。数据段和索引段是表建立后，Oracle 自动地根据表建立时给出的参数（或缺省参数）为该表的数据部分和索引部分保留的空间。其中每个段又有若干区间（extent）。一个区间由若干自由块组成，以便今后插入新数据用。

图 11.16 给出了 Oracle 数据库存储空间组织的示意图。

Oracle 中数据字典是由 DBMS 自动建立和维护的一组表和视图。用户可以查询、参考数据字典。

Oracle 中一个数据库表由若干元组组成，它们按图 11.17 的格式存放在一个数据块中。一个数据块可存放多个元组，一个元组只存放在一个块中。

Oracle 中的索引是 B+树结构的，在 Oracle 中称为 B+树。Oracle 的索引在逻辑上和物理存储上都独立于数据。

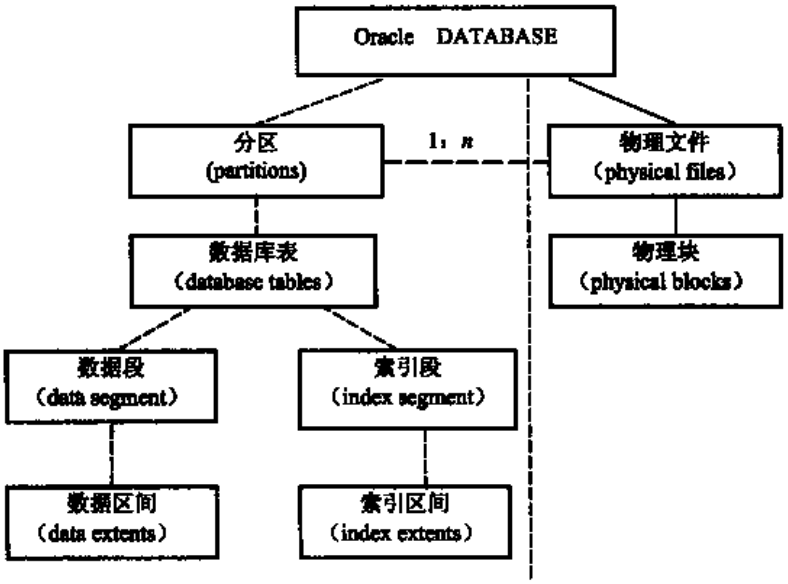


图 11.16 Oracle 数据库组织示意图

时间印（Timestamp）	
下一块的地址	
上一块的地址	
块头信息（Header）	
行号	该行长度
列 a 标识	列 a 长度
列 a 数据	
列 b 标识	列 b 长度
列 b 数据	
⋮	
备用空间（由 PCT FREE 参数确定）	

图 11.17 Oracle 数据块的格式

11.8 小　结

本章主要讨论 DBMS 的基本功能，系统结构及主要的实现技术，是 DBA 和数据库应用系统开发人员应该掌握的内容。

按照 DBMS 的层次结构依次介绍了语言翻译处理层、数据存取层、缓冲区管理层、数据物理组织层等各个层次的主要任务和功能，涉及的主要概念和问题。这里没有讲解 DBMS 具体的实现技术、实现算法和数据结构，这些内容应该在专门的一本 DBMS 原理和设计中讲解。

数据库领域过去所取得的主要成就，就是数据建模和计算技术。三十多年来数据库研究人员深入研究了这些技术并且取得了辉煌的成就。例如人们已经清楚地知道如何在外部存储设备上存储数据、如何分片；如何使用各种复杂的存取方法、缓冲策略和索引技术访问外部存储设备上的数据；数据库恢复、并发控制、完整性和安全性的实施、查询处理和优化等技术也得到深入了解，并在大多数商用的集中式和分布式 DBMS 中得以实现。

从这一章看到了在数据库领域中从理论的创立和工作原理的推出，到实验研究和原型系统研制，最后演化为商品的成功过程。数据库系统研究的历史有力地说明进行基础研究可以带来商业上的成功。

习　题

1. 试述 DBMS 的基本功能？

2. 试述 DBMS 四种进程组织方案，并分析各种方案的特点和优缺点。

3. 理解并解释下列术语的含义：进程、任务、“轻权”进程、线程、线索。

4. 什么是 DBMS 的多线索机制？有什么优点？

5. DBMS 由哪些主要的程序模块组成？

6. DBMS 工作过程是什么？给出 DBMS 插入一个记录的活动过程，画出类似图 11.8 的活动过程示意图。

7. DBMS 的语言翻译层是如何处理一个 DDL 语句的？

8. 试述 DBMS 的语言翻译层处理一个 DML 语句的大致过程。

9. 什么是处理 DML 语句的解释方法和预编译方法？试述二者的区别，联系，比较各自的优缺点。

10. 试述数据存取层主要的子系统及其功能。

11. 在操作系统中也有并发控制问题，为什么 DBMS 还要并发控制机制？

12. 试比较 DBMS 与操作系统的封锁技术。

13. DBMS 中为什么要设置系统缓冲区？

14 .数据库中要存储和管理的数据内容包括哪些方面？

*15. 请给出缓冲区管理中的一个淘汰算法，并上机实现(提示：你首先要设计缓冲区的数据结构，然后写出算法)。

*16. 请写出对一个文件按某一个属性的排序算法(设该文件的记录是定长的)，并上机实现。若要按多个属性排序，你能写出改进的算法吗？

*17. 请给出 B+ 树文件的创建和维护(增、删、改)算法并上机实现(提示：设 B+ 树叶结点上仅存放索引项[码值，TID]，你首先要设计索引项，B 树叶页和非叶页的数据结构，然后写出算法)。注：习题 15、16、17 可作为学生 DBMS 课程的实习课题。这些模块是 DBMS 中必不可少的基本模块。

本章参考文献

1 Gray Jamed. Notes on Data Base Operation Systems. In Operating Systems: An Advanced Course. Springer Verlag, 1978

（本文献系统地综述了数据库系统及其与操作系统有关的问题。）

2 Stonebraker M R. Operating System Support for Database Management. CACM VOL.24, No.7, July 1981

（本文提出了操作系统应提供对数据库系统更好支持的问题。）

3 Traiger I L. Virtual Memory for Database Systems. ACM SIGOPS Operating Systems Review, Volume 16, Number 4, 1982

（本文讨论了数据库系统中与操作系统有关的问题。）

4 马文骞. DBMS 进程结构研究及多线索 DBMS 的设计与实现. 中国人民大学硕士论文, 1993

（本文详细讨论了数据库管理系统的进程结构和多线索 DBMS 的概念、优点和实现技术。）

5 Astrahan M M et al. System R: Relational Approach to Database Management. ACM TODS 1, No.2, June 1976

（本文详细介绍了 IBM 公司的 System R 的实现技术。）

6 Blasgen M M et al. System R: An Architectural Overview. IBM Sys.J.20, No.1, February 1981

（本文详细介绍了 IBM 公司的 System R 的系统结构。）

7 Selinger P G et al. Access Path Selection in a Relational Database Management System. Proc. of ACM SIGMOD International Conference on Management of Data, June 1979

8 Blasgen M M. Eswaran K P. Storage and Access in Relational Database. IBM Sys.J. 16,

No.4，1977

9 Merrett T H. Why Sort/Merger Gives the Best Implementation of the Natural Join. ACM SIGMOD Record 13，No.2，January 1983

（本文阐述了自然连接的实现技术之一，合并扫描方法及其优点。）

10 Bayer R，McCreight C. Organization and Maintenance of Large Ordered Indexes. Acta informatica I，No.3，1972

（Bayer 和 McCreight 在本文中首先提出了 B 树存储结构，现在 B 树已广泛用于数据库的物理组织之中。）

11 Comer D. The Ubiquitous B-tree. ACM Computing Surveys 11，No.2，June 1979

（本文对 B 树的历史和发展进行了系统的综述，并介绍了各种 B 树的变种。）

12 王珊，吴鸥琦. B 树及其在文件组织中的应用. 电子计算机动态. 1981，9

（本文讨论 B 树的优缺点及其作用。）

13 王珊，吴鸥琦. B+ 树效率分析和组织聚集索引的算法——B+ 树的进一步研究和应用. 计算机研究与发展，1982，11

（本文讨论了 B+树的组织和算法。）

14 Wagner R E. Indexing Design Considerations. IBM sys. J. 12，No.4，1973

15 Knott G D. Expandable Open Addressing Hash Table Storage and Retrieval. Proc. of ACM SIGFIDET Workshop on Data Description，Access and Control，November 1971

16 Fagin R，Nievergelt J，Pippenger N et al. Extendible Hashing：A Fast Access Method for Dynamic Files. ACM TODS 4，No.3，September 1979

17 Lum V. Multi-Attribute Retrieval with Combined Indexes. CACM 13，No.11，November 1971

（本文研究了在多属性上建立组合属性索引的问题。）

18 Knuth D E. The Art of Computer Programming. Vol.III，1973

（本书是一套著名丛书中的一本，在本书中 Knuth 对多种数据存取方法进行了详细地分析，包括对 B+树及其相关存储结构的分析。）

19 Wirth W. Programming = Algorithm + Data Structures. 1976

（本书是一本数据结构教科书，其中详细给出了 B 树的增加、删除、修改算法。）

20 王珊. 关系数据库产品透析. 中国计算机报专家评述，1993，11

（本文分析并给出了关系数据库产品的三个发展阶段，总结了各个阶段的特点。）

21 孟小峰，王珊. 嵌套查询的非嵌套化处理研究. 计算机学报. Vol.18，No.4，1995，4

（本文研究了 JA 类嵌套查询的优化问题，提出了把嵌套查询非嵌套化的算法。）

22 项明峰，王珊. 数据库字典的封锁技术. 第 10 届全国数据库会议论文集，1992

（本文提出了一种有效的数据库字典的封锁方法，并成功地用于 COBASE 数据库

管理系统的实现之中。COBASE 是由北京大学、中国人民大学、中国软件总公司等单位联合研制的关系数据库管理系统。）

第四篇　新技术篇

第十二章　数据库技术新发展

了解当前数据库技术的进展，研究数据库发展的动向，分析各种新型数据库的特点，对数据库技术的研究和应用具有重大的意义。本章以数据模型、数据库应用、数据库管理系统开发技术三个方面为主线概述数据库发展历程，展示数据库学科在理论、应用和系统开发等研究和应用领域的主要内容和发展方向。目的在于提供一个宏观的、总体的数据库学科的视图，使读者既能了解数据库的新进展又能了解数据库技术的来龙去脉；既能了解新的数据库分支的基础，又能了解这些分支之间的相互联系。

数据库技术从20世纪60年代中期产生到今天仅仅几十年的历史。其发展速度之快，使用范围之广是其他技术所远不及的。数据库系统已从第一代的网状、层次数据库系统，第二代的关系数据库系统，发展到第三代以面向对象模型为主要特征的数据库系统。

数据库技术与网络通信技术、人工智能技术、面向对象程序设计技术、并行计算技术等互相渗透，互相结合，成为当前数据库技术发展的主要特征。

12.1　数据库技术三个发展阶段

前面已经讲到，数据模型是数据库系统的核心和基础。数据模型的发展经历了格式化数据模型（包括层次数据模型和网状数据模型）、关系数据模型两个阶段，正在走向面向对象的数据模型等非传统数据模型的阶段。按照数据模型的进展，数据库技术可以相应地分为三个发展阶段。

层次数据库系统和网状数据库系统的数据模型虽然分别为层次模型和网状模型，但实质上层次模型是网状模型的特例。它们都是格式化模型。它们从体系结构、数据库语言到数据存储管理均具有共同特征，是第一代数据库系统。

关系数据库系统支持关系模型。关系模型不仅简单、清晰，而且有关系代数作为语言模型，有关系数据理论作为理论基础。因此，关系数据库系统具有形式基础好、数据独立性强、数据库语言非过程化等特色，标志着数据库技术发展到了第二代。

第一和第二代数据库系统的数据模型虽然描述了现实世界数据的结构和一

些重要的相互联系，但是仍不能捕捉和表达数据对象所具有的丰富而重要的语义，因此尚只能属于语法模型。

第三代的数据库系统将是以更加丰富的数据模型和更强大的数据管理功能为特征，从而满足传统数据库系统难以支持的新的应用要求。

12.1.1 第一代数据库系统

第一代数据库系统指层次和网状数据库系统，其代表是：

（1）1969 年 IBM 公司研制的层次模型的数据库管理系统 IMS（Information Management System）。

（2）美国数据库系统语言协商会 CODASYL（Conference On Data System Language）下属的数据库任务组 DBTG（Data Base Task Group）对数据库方法进行了系统的研究、探讨，于 20 世纪 60 年代末 70 年代初提出了若干报告，称为 DBTG 报告。DBTG 报告确定并建立了数据库系统的许多概念、方法和技术。DBTG 所提议的方法是基于网状结构的。它是数据库网状模型的典型代表。

在 DBTG 方法和思想的指引下数据库系统的实现技术不断成熟，开发了许多商品化的数据库管理系统，它们都是基于网状模型或层次模型的。

可以说，层次数据库是数据库系统的先驱，而网状数据库则是数据库概念、方法、技术的奠基。它们是数据库技术中研究得最早的两种数据库系统。两者的区分是以数据模型为基础，层次数据库的数据模型是分层结构的，而网状数据库的数据模型是网状的，它们的数据结构都可以用图来表示。层次数据模型对应于有根定向有序树，而网状模型对应的是有向图。所以，这两种数据模型可以统称为格式化数据模型。

这两种数据库系统具有下列共同特点：

1. 支持三级模式的体系结构

三级模式通常指外模式、模式、内模式。模式之间具有转换(或称为映射)功能。

层次数据库和网状数据库均支持三级模式结构，通过外模式与模式、模式与内模式之间的映象，保证了数据库系统具有数据与程序的物理独立性和一定的逻辑独立性。

2. 用存取路径来表示数据之间的联系

这是数据库系统和文件系统的主要区别之一。

数据库不仅存储数据而且存储数据之间的联系。数据之间的联系在层次和网状数据库系统中是用存取路径来表示和实现的。例如，DBTG 中一对多的联系用系(Set)来表示，而系一般是用指引元的方法实现的，因此系值就是一种数

据的存取路径。

3. 独立的数据定义语言

层次数据库系统和网状数据库系统有独立的数据定义语言，用以描述数据库的外模式、模式、内模式以及相互映象。诸模式一经定义，就很难修改。修改模式必须首先把数据全部卸出，然后重新定义诸模式，重新生成诸模式，最后编写实用程序把卸出的数据按新模式的定义装入新数据库中。因此在许多实际运行的层次、网状数据库系统中，模式是不轻易重构的。这就要求数据库设计人员在建立数据库应用系统时，不仅充分考虑用户的当前需求，还要充分了解需求可能的变化和发展。对数据库设计的要求比较高。

4. 导航的数据操纵语言

层次和网状数据库的数据查询和数据操纵语言是一次一个记录的导航式的过程化语言。这类语言通常嵌入某一种高级语言如 COBOL，FORTRAN，PL/1 中。

所谓导航就是指用户不仅要了解“要干什么”，而且要指出“怎么干”。用户必须使用某种高级语言编写程序，一步一步地“引导”程序按照数据库中某一条预先定义的存取路径来访问数据库，最终达到要访问的数据目标。在访问数据库时，每次只能存取一条记录值。若该记录值不满足要求就沿着存取路径查找下一条记录值。

导航式的数据操纵语言其优点是存取效率高。应用程序员可以根据对数据库逻辑模式和存储模式的了解选取一条较优的存取路径，从而优化了存取效率。缺点是编程繁琐。用户既要掌握高级语言又要掌握数据库的逻辑结构和物理结构。导航式的数据访问方式，给用户编写应用程序带来了困难，其设计在很大程度上依赖于设计者自己的经验和实践，因而只有具有计算机专业水平的应用程序员才能掌握和使用这类数据库操纵语言。还有，应用程序的可移植性较差，数据的逻辑独立性也较差。

12.1.2 第二代数据库系统——关系数据库系统

支持关系数据模型的关系数据库系统是第二代数据库系统。

1970 年 IBM 公司 San Jose 研究室的研究员 E.F.Codd 发表了题为“大型共享数据库数据的关系模型”论文，提出了数据库的关系模型，开创了数据库关系方法和关系数据理论的研究，为关系数据库技术奠定了理论基础。

20 世纪 70 年代是关系数据库理论研究和原型开发的时代。其中以 IBM San Jose 研究室开发的 System R 和 Berkeley 大学研制的 INGRES 为典型代表。经过大量的高层次的研究和开发取得了一系列的成果。主要是：

（1）奠定了关系模型的理论基础，给出了人们一致接受的关系模型的规

范说明。

（2）研究了关系数据语言，有关系代数、关系演算、SQL 语言及 QBE 等。这些描述性语言一改以往程序设计语言和网状、层次数据库系统中数据库语言的风格，以其易学易懂的优点得到了最终用户的喜爱，为 20 世纪 80 年代数据库语言标准化打下了基础。

（3）研制了大量的 RDBMS 的原型，攻克了系统实现中查询优化、并发控制、故障恢复等一系列关键技术。不仅大大丰富了 DBMS 实现技术和数据库理论，更重要的是促进了 RDBMS 产品的蓬勃发展和广泛应用。

关系数据库系统从实验室走向了社会，因此，在计算机领域中有人把 20 世纪 70 年代称为数据库时代。

20 世纪 80 年代几乎所有新开发的系统均是关系的。这些商用数据库系统的运行，特别是微机 RDBMS 的使用，使数据库技术日益广泛地应用到企业管理、情报检索、辅助决策等各个方面，成为实现和优化信息系统的基本技术。

关系数据库是以关系模型为基础的。关系模型概括地讲由三部分组成：

1. 数据结构

数据结构包括域及域上定义的关系等。关系模型中，实体、实体与实体之间的联系都通过关系这种单一的结构类型来表示。

2. 关系操作

关系操作的能力可用关系代数(或等价的关系演算)中并、交、差、广义笛卡尔积、选择、投影、连接、除等操作来表示。关系操作的特点是集合操作。无论是操作的对象还是操作的结果都是集合。这种操作方式被称为一次一集合(set-at-a-time)的方式，与非关系型的一次一记录(record-at-a-time)的方式相对照。

3. 数据完整性

数据完整性包括实体完整性、参照完整性以及与应用有关的完整性。

实体完整性和参照完整性是关系模型必须满足的完整性约束条件，应由关系数据库系统自动支持；与应用有关的完整性由用户定义，它是针对某一具体数据库的约束条件，由应用环境决定，关系数据库系统应提供定义和检验这类完整性的机制。

关系模型建立在严格的数学概念基础上，概念简单、清晰，易于用户理解和使用，大大简化了用户的工作。正因为如此，关系模型提出以后，便迅速发展，并在实际的商用数据库产品中得到了广泛应用，成为深受广大用户欢迎的数据模型。总的来看，关系模型具有以下主要特点：

（1）关系模型的概念单一，实体以及实体之间的联系都用关系来表示；

（2）以关系代数为基础，形式化基础好；

（3）数据独立性强，数据的物理存储和存取路径对用户隐蔽；

（4）关系数据库语言是非过程化的，将用户从编程数据库记录的导航式检索中解脱出来，大大降低了用户编程的难度。

12.1.3 新一代数据库技术的研究和发展

从20世纪80年代以来，数据库技术在商业领域的巨大成功刺激了其他领域对数据库技术需求的迅速增长。这些新的领域为数据库应用开辟了新的天地，另一方面在应用中提出的一些新的数据管理的需求也直接推动了数据库技术的研究与发展，尤其是面向对象数据库系统（Object Oriented DataBase System，简称OODBS）的研究与发展。

一、新应用领域的需求

新的数据库应用领域，如计算机辅助设计/管理（CAD/CAM）、计算机集成制造（CIM）、办公信息系统（OIS）、地理信息系统（GIS）、知识库系统和实时系统等，需要数据库的支持，而其所需的数据管理功能有相当一部分是传统的数据库系统所不能支持的。

例如，在计算机辅助超大规模集成电路设计（VLSI CAD）中，其所需的数据管理就有许多不同于传统数据管理的地方：

（1）VLSI设计中要涉及大量的数据，许多具有复杂结构。如几何模板数据，无论是用基本的原始几何体组合来描述还是用几何边界来描述，其结构都将是复杂的、有层次的。

（2）有大量的历史数据必须保存，直到设计完成和稳定。这就需要管理不同版本的设计数据。

（3）在一个复杂芯片设计中存在大量的可重用的原始部件，对这些原始部件提供数据库管理将极大地支持对已开发的部件库的重用。

（4）VLSI基本单元模型有许多不同方面的表示，每一方面都有自己独立的描述。例如，功能描述、芯片描述、布线模板描述等。

许许多多这样的应用需求都要求新的数据库系统有比传统数据库系统更强大的数据管理能力。例如它们通常需要数据库系统支持以下功能：

（1）存储和处理复杂对象。这些对象不仅内部结构复杂，很难用普通的关系结构来表示，而且相互之间的联系也有复杂多样的语义。

（2）支持复杂的数据类型。包括抽象数据类型、半结构或无结构的超长数据、时间和版本数据等。还要具备支持用户自定义类型的可扩展能力。

（3）需要常驻内存的对象管理以及支持对大量对象的存取和计算。

（4）实现程序设计语言和数据库语言无缝地集成。

（5）支持长事务和嵌套事务的处理。

二、传统数据库系统的局限性

新的应用领域需要数据库的支持，但是当试图将传统的数据库系统运用到前面所提到的那些新的应用领域时，就暴露出传统数据库系统的局限性。

这里应说明的是，分析传统数据库系统的局限性并不是批评或否定传统数据库系统。前面已经阐明了传统数据库系统，尤其是关系数据库系统的优点及其在传统应用领域中取得的成就。它们适合处理格式化数据，较好地满足了商业事务处理的需求。现在人们要把它们应用到新的应用领域，可以说是对传统数据库的苛求。不过，正是应用的驱动，推动了数据库技术的进步，同样也促进了传统数据库系统的提高和发展。

传统数据库系统的局限性主要表现在以下几个方面：

1. 面向机器的语法数据模型

传统数据库中采用的数据模型强调数据的高度结构化，是面向机器的语法数据模型。它们只能存储离散的数据和有限的数据与数据之间的关系，语义表示能力差，无法表示客观世界中的复杂对象，即结构复杂、相互联系的语义也十分复杂的对象。从而限制了数据库处理文本、超文本、图形、图像、CAD图件、声音等多种复杂对象，以及工程、地理、测绘等领域中的非格式化、非经典数据的能力。此外，传统数据模型无法揭示数据之间的深层含义和内在联系，缺乏数据抽象。

例如，关系模型只有一个非常简单的结构性概念：关系。

在传统应用中，数据对象具有同形结构，这样它们很容易映射到关系来表示；而工程对象拥有许多异形结构，一个复杂对象可能由许多具有不同结构的子对象组成。对这些复杂对象比较自然的（即对于用户来说是友好的）表示，就需要有比关系模型更复杂的抽象机制。

有两个主要的数据抽象：

- 聚合(Aggregation)

聚合也称为聚集，是用来抽象由子对象聚集而成的合成对象。

- 泛化(Generalization)和特化(Specialization)

泛化也称为普遍化或概括。泛化是指将相似对象分类，抽象成一个一般化类型。

特化也称为特殊化。特殊化是与普遍化互补的概念，是指一个对象类型可精化到一个只涉及更特殊属性的实例。

在关系数据库系统中必须将在逻辑上是一个整体的复杂对象分解为好几个

基本关系。在这种结构下内部数据库结构与外部对象不再是一一对应的。对许多操作来说，人们更希望把所操作的部分抽象为一个逻辑单位，而关系模型不支持这一点。于是人们必须从关系模式的片段中构造复杂对象，其结果常常是带有许多冗余数据的不自然的复杂查询。进一步讲，重构复杂对象还带来另一个问题，那就是，由连接构造的视图一般是不可更新的。

关系数据库系统是基于值的，即用码来标识元组。采用这种用值来标识一个对象的方法，一旦其值变化，所有对该对象的引用将变为无效。

2. 数据类型简单、固定

传统的 DBMS 只能理解、存储和处理简单的数据类型。如整数、浮点数、字符串、日期、货币等。传统的 RDBMS 只支持某一固定的类型集，不能依据某一应用所需的特定数据类型来扩展其类型集。例如，不能定义包含三个实数分量的数据类型 vector 来表示三维向量。

在传统 DBMS 中，复杂数据类型只能由用户编写程序借助高级语言功能用简单的数据类型来构造、描述和处理，这就加重了用户的负担，也不能保证数据的一致性。

此外，如果在客户/服务器环境下，由于应用程序在客户端，势必加大了客户与服务器之间的网络通信开销，没有充分发挥服务器的性能，从而降低了整个系统的效率。

3. 结构与行为分离

从应用程序员角度来看，在某一应用领域内标识的对象应包含两个方面的内容：

- 结构表示
- 行为规格说明

前者可映射到数据库模式（带着前面所提到的缺陷），而后者在传统数据库系统中则完全失去了。

传统数据库主要关心数据的独立性以及存取数据的效率，是语法数据库，语义表达差，难以抽象化地去模拟行为。例如，用户在 CAD 设计中用某些数据结构来表示的对象，对它们的操作（如成形、显示和组合等）就无法存放到数据库中，即便按记录存放进去了，这些操作也成了毫无意义的编码(或字符)，对象中与应用相关的大量语义在数据库中无法从无意义的编码中恢复。这样，对象的行为特征在传统数据库系统中最多只能由应用程序来表示。因此在传统数据模型中，结构与行为被完全分割开了。

4. 阻抗失配

在关系数据库系统中，数据操纵语言如 SQL 与通用程序设计语言之间的失配称为阻抗失配。这种不匹配表现在两个方面：一是编程模式不同，描述性

的 SQL 语言与指令式的编程语言如 C 语言不同；二是类型系统不匹配，编程语言不能直接表示诸如关系这样的数据库结构，在其界面就会丢失信息。进一步地，由于是两个类型系统，自动的类型检查也成了问题。

5. 被动响应

传统数据库管理系统只能响应和重做用户要求它们做的事情，从这种意义上说，它们是被动的。而在实际应用中，往往要求一个系统能够管理它本身的状态，在发现异常情况时及时通知用户；能够主动响应某些操作或外部事件，自动采取规定的行动；应该能够在一些预定的（或动态计算的）时间间隔中自动执行某些操作。就象在现实生活中，一个好的决策者（个人或组织）周围必定有一群人负责主动、及时地提供各种有用信息和提出建议（专家知识），决策者不需要总是向别人或机器询问信息。这就是说，要求系统更加主动、更加智能化，而传统的数据库系统显然不能适应这一要求。

6. 存储、管理的对象有限

传统的 DBMS 只存储和管理数据，缺乏知识管理和对象管理的能力。

传统数据管理中，主要进行的是数据的存储、管理、查询、排序和报表生成等比较简单的、离散化的信息处理工作。数据库反映的是客观世界中静态、被动的事实。

上面已经论述了传统的 DBMS 缺乏对象管理的能力。此外，传统的 DBMS 还缺乏描述和表达知识的能力，缺乏对知识的处理能力，不具有演绎和推理的功能，因而无法满足 MIS，DSS，OA 和 AI 等领域中进行高层管理和决策的要求，从而限制了数据库技术的高级应用。

7. 事务处理能力较差

传统数据库只能支持非嵌套事务。对长事务的响应较慢，而且在长事务发生故障时恢复也比较困难。

面对数据库应用领域的不断扩展和用户要求的多样化、复杂化，传统的数据库技术遇到了严峻的挑战，它所固有的这些局限性使其不能适应新的要求。正是这些缺陷决定了当前数据库的研究方向与未来的努力方向，新一代数据库技术应运而生。

三、新一代数据库技术的特点

由于传统数据库在新应用中存在的种种缺陷，数据库技术遇到了挑战。人们问："下一代数据库系统是什么"？"数据库将向何处去"？数据库工作者为了给应用建立合适的数据库系统，进行了艰苦的探索，从多方面发展了现行的数据库系统技术，如：

- 针对不同的应用，对传统的 DBMS，主要是 RDBMS 进行不同层次上

的扩充。

• 与其他学科的新技术紧密结合，丰富和发展数据库系统的概念、功能和技术。

以上两方面的努力是立足于数据库已有的成果和技术，加以发展进化，有人称之为“进化论”的观点和方法。

另一方面的努力是立足于新的应用需求和计算机未来的发展，研究全新的数据库系统，有人称之为“革新论”的观点和方法。可以说新一代数据库技术的研究，新一代数据库系统的发展呈现了百花齐放的局面。其特点是：

1. 面向对象的方法和技术对数据库发展的影响最为深远

20 世纪 80 年代出现的面向对象的方法和技术对计算机各个领域，包括程序设计语言、软件工程、信息系统设计以及计算机硬件设计等都产生了深远的影响，也给面临新挑战的数据库技术带来了机会和希望。数据库研究人员借鉴和吸收了面向对象的方法和技术，提出了面向对象数据模型(简称对象模型)。该模型克服了传统数据模型的局限性，为新一代数据库系统的探索带来了希望，促进了数据库技术在一个新的技术基础上继续发展。

2. 数据库技术与多学科技术的有机结合

数据库技术与多学科技术的有机结合是当前数据库技术发展的重要特征。

计算机领域中其他新兴技术的发展对数据库技术产生了重大影响。传统的数据库技术和其他计算机技术的互相结合、互相渗透，使数据库中新的技术内容层出不穷。数据库的许多概念、技术内容、应用领域，甚至某些原理都有了重大的发展和变化。建立和实现了一系列新型数据库系统，如分布式数据库系统、并行数据库系统、演绎数据库系统、知识库系统、多媒体数据库系统等。它们共同构成了数据库系统大家族。

传统的数据库系统仅是数据库大家族的一员，当然，它也是最成熟的和应用最广泛的一员。它的核心理论、应用经验、设计方法等仍然是整个数据库技术发展和应用开发的先导和基础。

3. 面向应用领域的数据库技术的研究

为了适应数据库应用多元化的要求，在传统数据库基础上，结合各个应用领域的特点，研究适合该应用领域的数据库技术，如数据仓库、工程数据库、统计数据库、科学数据库、空间数据库、地理数据库等，这是当前数据库技术发展的又一重要特征。

研究和开发面向特定应用领域的数据库系统的基本方法是以传统数据库技术为基础，针对某一领域的数据对象的特点，建立特定的数据模型，它们有的是关系模型的扩展和修改，有的是具有某些面向对象特征的数据模型。

四、第三代数据库系统

新一代数据库技术的研究和发展导致了众多不同于第一、二代数据库的系统诞生，构成了当今数据库系统的大家族。这些新的数据库系统无论它是基于扩展关系数据模型的、还是 OO 模型的；是分布式、客户/服务器或混合式体系结构的；是在 SMP 还是在 MPP 并行机上运行的并行数据库系统；是用于某一领域(如工程、统计、GIS)的工程数据库、统计数据库、空间数据库等，都可以广泛地称之为新一代数据库系统。

但是，在学术上讨论第三代数据库系统时，它应该和第一、二代数据库系统一样有严格的含义。尽管众多学者认为第三代数据库系统尚未成熟，但这并不妨碍人们来讨论和研究什么样的数据库系统可称之为第三代数据库系统。

1990 年高级 DBMS 功能委员会[①]发表了“第三代数据库系统宣言”的文章（以下简称《宣言》），提出了第三代 DBMS 应具有的三个基本特征（《宣言》中称为三条基本原则)。《宣言》从三个基本特征导出了十三个具体的特征和功能（《宣言》称为十三个命题）。

经过多年的研究和讨论，对第三代数据库系统的基本特征已有了共识。

1. 第三代数据库系统应支持数据管理、对象管理和知识管理

除提供传统的数据管理服务外，第三代数据库系统将支持更加丰富的对象结构和规则，应该集数据管理、对象管理和知识管理为一体。由此可以导出，第三代数据库系统必须支持 OO 数据模型。

关于什么是 OO 数据模型，至今尚没有一个统一的严格的定义，因此第三代数据库系统不像第二代关系数据库那样有一个统一的关系模型。但是，有一点应该是统一的，即无论该数据库系统支持何种复杂的，非传统的数据模型，它应该具有 OO 模型的基本特征。关于 OO 模型的基本特征，将在第十三章中讨论。

数据模型是划分数据库发展阶段的基本依据。因此第三代数据库系统应该是以支持面向对象数据模型为主要特征的数据库系统。但是，只支持 OO 模型的系统不能称为第三代数据库系统。第三代数据库系统还应具备其他特征。

2. 第三代数据库系统必须保持或继承第二代数据库系统的技术

即必须保持第二代数据库系统的非过程化数据存取方式和数据独立性。第三代数据库系统应继承第二代数据库系统已有的技术。不仅能很好的支持对象

① 该委员会成员包括： 加州 Berkeley 分校的 Michael Stonebraker 和 Lawrence A.Rowe；IBM 研究所的 Rruce Lindsay；Tandem 计算机公司的 James Gray；Wisconsi 大学的 Mchael Carey；GTE 实验室的 Michael Brodie；DEC 公司的 Philip Bernstacin；Oracle 公司的 David Beech。

管理和规则管理，而且能更好地支持原有的数据管理，支持多数用户需要的即席查询等。

3. 第三代数据库系统必须对其他系统开放

数据库系统的开放性表现在：支持数据库语言标准；在网络上支持标准网络协议；系统具有良好的可移植性、可连接性、可扩展性和可互操作性等。

12.2 数据库新技术

前面提到，当今数据库系统是个大家族，数据模型丰富多样，新技术内容层出不穷，应用领域日益广泛，当读者步入数据库领域时，面对众多复杂的数据库系统会产生迷惑和混乱。

图 12.1 从数据模型、新技术内容、应用领域三个方面，通过一个三维空间的视图，阐述了新一代数据库系统及其相互关系。

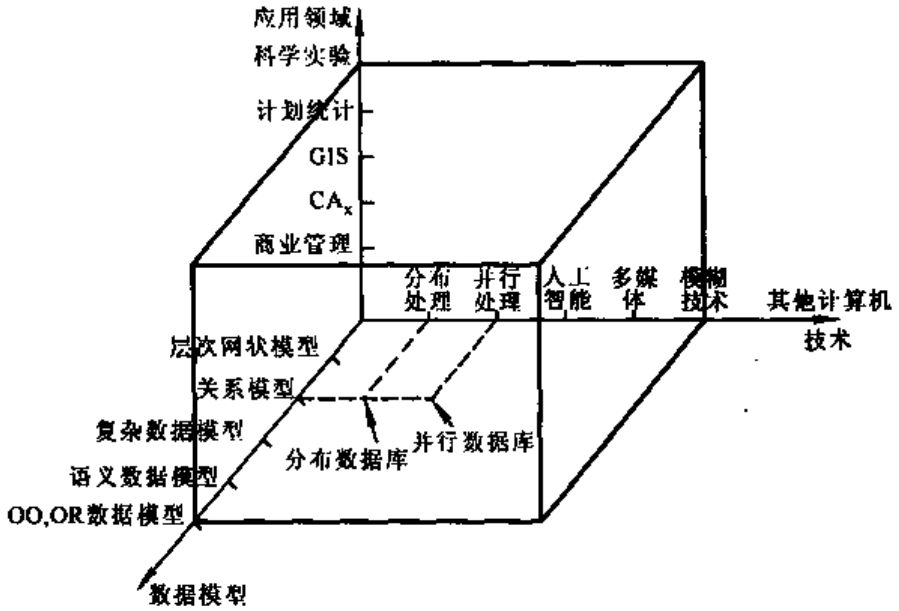


图 12.1 新一代数据库系统及其相互关系示意图

下面就从数据模型、新技术内容、应用领域三个方面来讨论数据库的新技术和新进展。

12.2.1 数据模型的发展

在讨论数据模型发展之前，首先回顾关于数据模型的概念和定义。

数据库是某个企业、组织或部门所涉及到的数据的综合。计算机不可能直接处理现实世界中的具体事物，需要把具体事物转换成计算机能够处理的数

据。在数据库中用数据模型（Data Model)这个工具来抽象、表示和处理现实世界中的数据和信息。数据模型是数据特征的抽象，描述的是数据的共性。

数据模型应满足三方面要求：一是能比较真实的模拟现实世界；二是容易为人们所理解；三是便于在计算机上实现。一种数据模型要很好地满足这三方面的要求在目前尚很困难。在数据库系统中针对不同的使用对象和应用目的，采用逐步抽象的方法，在不同层次采用不同的数据模型。一般分为三层，如图12.2 所示。

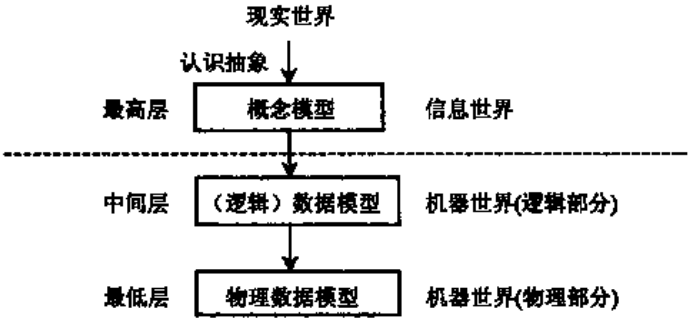


图 12.2 数据抽象的三个层次

一、物理层

数据抽象的最低层，用来描述数据物理存储结构和存储方法。例如一个数据库中数据和索引是存放在不同的数据段上还是同一数据段中。数据的物理记录格式是变长的还是定长的，数据是压缩的还是非压缩的，索引结构是 B+树还是 HASH 结构等。这一层的数据抽象称为物理数据模型，它不但由 DBMS 的设计决定，而且与操作系统、计算机硬件密切相关。物理数据结构一般都向用户隐蔽，用户不必了解其细节。

二、逻辑层

数据抽象的中间层，描述数据库数据整体的逻辑结构。这一层的数据抽象称为逻辑数据模型（简称数据模型）。它是用户通过数据库管理系统看到的现实世界，是数据的系统表示。因此它既要考虑用户容易理解，又要考虑便于 DBMS 实现。不同的 DBMS 提供不同的逻辑数据模型，传统的数据模型有层次、网状、关系模型，非传统的数据模型有面向对象数据模型（简称 OO 模型）。

三、概念层

概念层次的数据模型称为概念数据模型，简称概念模型。概念模型离机器最远，从机器立场看是抽象级别的最高层。目的是按用户的观点来对世界建模，

因此它应该是：

（1）语义表达能力强。能够方便、直接地表达各种语义。

（2）易于用户理解。概念模型是用户与数据库设计人员之间交流的语言。用户一般缺乏计算机知识，因此概念模型应当简单、清晰、易于用户理解。

（3）独立于任何 DBMS

（4）容易向 DBMS 所支持的逻辑数据模型转换。

概念模型的例子有实体-联系模型（Entity-Relational Model，简称 ER 模型）。

数据库的发展集中表现在数据模型的发展。从最初的层次、网状数据模型发展到关系数据模型，数据库技术产生了巨大的飞跃。关系模型的提出，是数据库发展史上具有划时代意义的重大事件。关系理论研究和关系数据库管理系统研制的巨大成功进一步促进了关系数据库的发展，使关系数据模型成为具有统治地位的数据模型。20 世纪 80 年代，几乎所有的数据库系统都是关系的，它的应用遍布各个领域。

然而，随着数据库应用领域对数据库需求的增多，传统的关系数据模型开始暴露出许多弱点，如对复杂对象的表示能力较差，语义表达能力较弱，缺乏灵活丰富的建模能力等。为了使数据库用户能够直接以他们对客观世界的认识方式来表达他们所要描述的世界，人们提出并发展了许多新的数据模型。这些尝试是沿着如下几个方向进行的：

1. 对传统的关系模型(1NF)进行扩充，引入了少数构造器，使它能表达比较复杂的数据类型，增强其结构建模能力，这样的数据模型称为复杂数据模型。按照它们进行扩充的侧重点，复杂数据模型可分为两种：

一种是偏重于结构的扩充。首先出现的这类模型是嵌套关系模型(NF2)。它能表达“表中表”，并且表中的一个域可以是一个函数(称为虚域)。

另一种是侧重于语义的扩充。像 U.C.Berkeley 大学的 POSTGRES 系统。它支持关系之间的继承，也支持在关系上定义函数和运算符。但关系的结构仍然是一张平面表。“表中表”只能通过关系上定义的函数来模拟。

总的来说，在复杂数据模型和支持它们的数据库系统里，客观世界中的每一个实体都用一个元组和它的码(KEY)来表示。不支持太多的语义关联，不区分类和型。

这种数据模型和数据库系统的主要缺点是，不能保证客观世界中实体的确定性；实体的引用只能通过码和数据冗余来达到。其主要优点是，支持这类模型的系统实现起来相对比较容易。

2. 提出和发展相比关系模型来说全新的数据构造器和数据处理原语，以表达复杂的结构和丰富的语义。这类模型比较有代表性的是函数数据模型(FDM)、语义数据模型(SDM)、RM/T 模型以及 E-R 模型等，常常统称它们为

语义数据模型。它们的特点是引入了丰富的语义关联(如 ISA，ISP)。能更自然、更恰当地表达客观世界中实体间的联系。加上比较丰富的结构构造器(如 TUPLE，LIST，SET 等)，因此它们也具有很强的结构表达能力。

也许是由于它们比较复杂，在程序设计语言和技术方面没有相应的支持，计算机硬件也没有发展到一定的程度，因此，它们都没有在数据库系统实现方面有重大的突破。至多被当做数据库设计中概念建模的一种工具(如 E-R 模型)。

3. 是将上述语义数据模型和 OO 程序设计方法结合起来提出了面向对象的数据模型。面向对象的数据模型吸收了面向对象程序设计方法学的核心概念和基本思想。

一个面向对象数据模型是用面向对象观点来描述现实世界实体（对象）的逻辑组织、对象间限制、联系等的模型。一系列面向对象核心概念构成了面向对象数据模型的基础。

有关面向对象数据模型的核心概念以及面向对象数据系统将在第十三章详细讨论。

12.2.2 数据库技术与其他相关技术相结合

数据库技术与其他学科的内容相结合，是新一代数据库技术的一个显著特征，涌现出各种新型的数据库系统（如图 12.3 所示），例如：

- 数据库技术与分布处理技术相结合，出现了分布式数据库系统；
- 数据库技术与并行处理技术相结合，出现了并行数据库系统；
- 数据库技术与人工智能技术相结合，出现了知识库系统和主动数据库系统；
- 数据库技术与多媒体技术相结合，出现了多媒体数据库系统；
- 数据库技术与模糊技术相结合，出现了模糊数据库系统等。

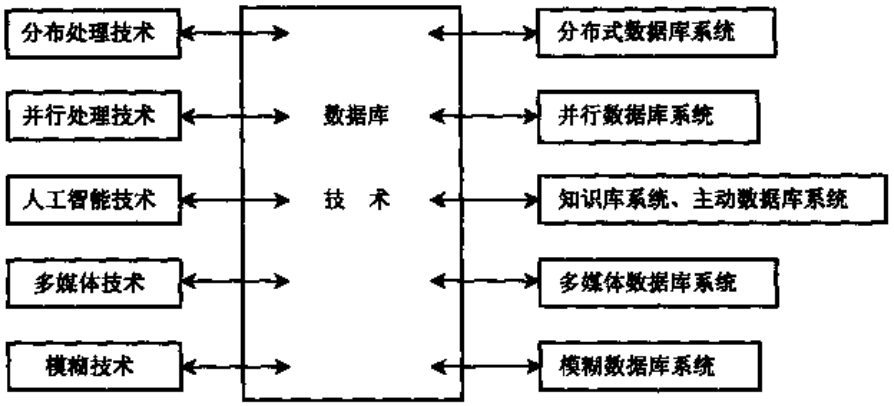


图 12.3 数据库技术与其他计算机技术的相互渗透

这里以几个新型数据库系统为例，描述数据库技术是如何吸收、结合其他计算机技术，从而形成了数据库领域的众多分支和研究课题，极大地丰富和发展了数据库技术。

一、分布式数据库系统

随着地理上分散的用户对数据共享的要求日益增强，以及计算机网络技术的发展，在传统的集中式数据库系统基础上产生和发展了分布式数据库系统。

分布式数据库系统并不是简单地把集中式数据库安装在不同场地，用网络连接起来便实现了（这是分散的数据库系统），而是具有自己的性质和特征。集中式数据库系统中的许多概念和技术，如数据独立性的概念；数据共享和减少冗余的控制策略；并发控制和事务恢复的概念和实现技术等在分布式数据库中有了不同的、更加丰富的内容。

分布式数据库应具有以下特点：

（1）数据的物理分布性。数据库中的数据不是集中存储在一个场地的一台计算机上，而是分布在不同场地的多台计算机上。它不同于通过计算机网络共享的集中式数据库系统。

（2）数据的逻辑整体性。数据库虽然在物理上是分布的，但这些数据并不是互不相关的，它们在逻辑上是相互联系的整体。它不同于通过计算机网络互连的多个独立的数据库系统。

（3）数据的分布独立性（也称分布透明性）。分布式数据库中除了数据的物理独立性和数据的逻辑独立性外，还有数据的分布独立性。即在用户看来，整个数据库仍然是一个集中的数据库，用户不必关心数据的分片，不必关心数据物理位置分布的细节，不必关心数据副本的一致性，分布的实现完全由分布式数据库管理系统来完成。

（4）场地自治和协调。系统中的每个结点都具有独立性，能执行局部的应用请求；每个结点又是整个系统的一部分，可通过网络处理全局的应用请求。

（5）数据的冗余及冗余透明性。与集中式数据库不同，分布式数据库中应存在适当冗余以适合分布处理的特点，提高系统处理效率和可靠性。因此，数据复制技术是分布式数据库的重要技术。但分布式数据库中的这种数据冗余对用户是透明的，即用户不必知道冗余数据的存在，维护各副本的一致性也由系统来负责。

20 世纪 80 年代，研制了许多分布式数据库的原型系统，攻克了分布式数据库中许多理论和技术难点。20 世纪 90 年代开始，主要的数据库厂商对集中式数据库管理系统的核心加以改造，逐步向分布式数据库管理系统发展。目前，分布式数据库管理系统已进入市场。

一个大的部门可能会同时使用多个不同的 DBMS，以支持各种不同特点和功能的应用系统。例如，一个大公司有许多部门，每个部门使用的 DBMS 不尽相同，当进行全公司效益评估、财务分析、经营决策等高层管理时，往往需要访问由异构 DBMS 管理的数据库。

现有的分布式数据库技术尚不能解决异构数据和系统的许多问题。异构分布式数据库技术还有待进一步研究。

有关分布式数据库技术和系统，将在第十四章详细讨论，这里就不赘述了。

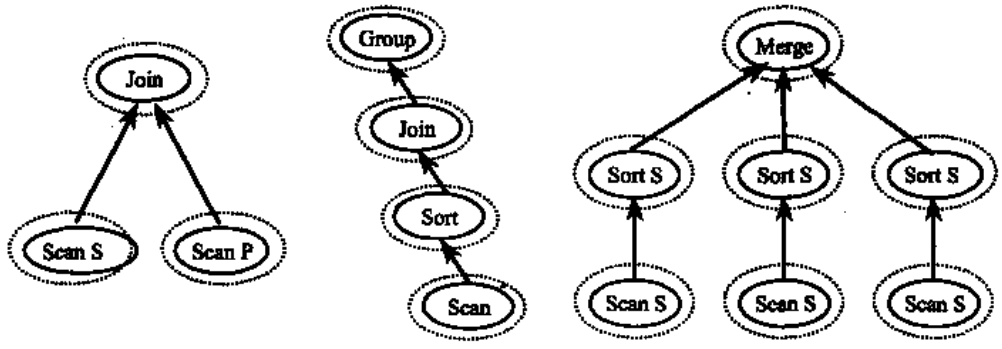
二、并行数据库系统

近年来，数据库系统的应用已经从商业数据处理迅速拓展到诸如超大型数据检索、数据仓库、联机数据分析、数据挖掘以及高吞吐量 OLTP 等许多应用领域。这些应用领域的特点是数据量大、复杂度高、用户数目多，对数据库系统的处理能力提出了非常高的要求，这些应用需求直接驱动了新一代高性能数据库系统—并行数据库系统的研制。

并行数据库系统试图通过充分利用通用并行计算机的处理机、磁盘等硬件设备的并行数据处理能力来提高数据库系统的性能。并行数据库系统自出现以来，已经取得了一系列引人注目的进展，主要的原因有两个：

第一，关系数据模型和 SQL 语言的广泛采纳。关系查询存在着内在的三种并行性：**操作间独立并行**、**操作间流水线并行**和**操作内并行**。

例如，图 12.4（a）中的 Scan S 操作和 Scan P 操作可以独立地并行执行；如果一个操作的输出是另一个操作的输入，这两个操作之间就存在着流水线并行性，例如图 12.4（b）中的四个操作就可以采用流水线方式并行执行；如果把操作的输入数据分解为多个子集，该操作就可以分成多个独立的子操作，每个子操作的并行执行称为操作内并行，图 12.4（c）中的 Scan S 和 Sort S 都被分解成三个子操作，每个子操作并行执行。



(a) 操作间独立并行　(b) 操作间流水线并行　(c) 操作内并行

图 12.4　数据库查询内在的三种并行性

需要注意的是，上述并行性并不是关系数据库系统特有的。一般来说，无论数据库系统采用什么数据模型，如果查询能够采用面向集合的代数进行表达，都蕴含着这三种并行性。

第二，近十几年来，计算机的处理能力得到了非常迅速的提高，高性能CPU、大容量内存、高速磁盘阵列以及高带宽通信网络的出现为高性能数据处理提供了充分的硬件支持。

国外 Intel，NCR，nCUBE，Sequent，Tandem，Teradata 公司和国内曙光公司等许多供应商已经能够提供基于快速、廉价的处理器、内存、磁盘和通信网络的多处理机系统。这些机器以较低的价格提供了比大型机更为强大的整体处理能力。并且，这些并行计算机具有很好的伸缩性，其模块化体系结构使得系统能够不断增长，通过增加处理机、内存和磁盘使得系统对于给定的任务能够加速完成，或者使得系统在相同的时间内能够处理大规模的任务。尤其是 Shared-Nothing 结构的并行计算机可以通过对多个磁盘的并行读写有效地克服数据 I/O 瓶颈。

自并行数据库的概念提出以来，国外的许多研究机构相继研制出了多种结构平台上的并行数据库原型系统，例如加州 Berkeley 大学的 XPRS 系统、Colorado 大学的 Volcano 系统、Wisconsin 大学的 Gamma 系统等。这些系统探讨了并行数据库理论和实现中的若干重要问题，为并行数据库的进一步发展奠定了坚实的基础。与此同时，Teradata，Tandem 以及 Oracle，Sybase，Informix 等传统数据库厂商都正在或已经开发出自己的并行数据库系统。

从 1993 年起在国家 863 高科技计划和国家自然科学基金的支持下，中国人民大学开展了对并行数据库理论和系统实现技术的研究，于 1996 年开发了基于曙光 1 号并行机（SMP 结构或 Shared-Memory 结构）平台的 PBASE/1 并行数据库管理系统。

但是，SM 结构的并行机不能从根本上解决 I/O 瓶颈和存储器瓶颈问题，处理器个数也受到系统收缩性的限制。因此，在 PBASE/1 研制成功后继续研制针对 MPP，Cluster 等 Shared-Nothing 平台的并行数据库系统。

与 SM 相比，SN 在并行查询优化、并行查询执行、数据存储管理、事务管理等方面都要复杂得多。在 SN 中采用了基于“流”的查询模型、先进的查询优化方案和执行机制等先进技术，于 1998 年研制了国内第一个有自主版权的实用而先进的并行数据库系统 PBASE/2。它的完成对推动我国并行数据库系统的研究和开发，推动国产并行计算机的发展都有着重要的意义。

有关并行数据库技术和系统，将在第十五章详细讨论，这里就不赘述了。

三、知识库系统

知识库系统是数据库和人工智能两种技术结合的产物。目前，已成为数据库研究的热门课题之一。

数据库和人工智能是计算机科学两个十分重要的领域，它们相互独立发展，在各自的领域取得了突出成就并获得广泛应用。然而，它们各自都存在着突出的问题和矛盾。

一方面现有的 AI 系统 (如专家系统) 可以使用成百上千条基于规则的知识去进行启发式搜索与推理，却缺乏高效检索访问事实库和管理大量数据和规则的能力；另一方面现有的 DBMS 已发展到可以处理海量数据和大量商业事务的水平，却缺乏表达和处理 AI 系统中常见的规则和知识，以提高数据库的演绎、推理能力。人们认识到 AI 和 DB 技术相结合可以克服单方面研究的局限性，彼此取长补短具有共同发展的广阔前景。

在数据库技术中引入人工智能技术，多年来是沿着数据库的智能化和智能化的数据库这两个途径发展的。

1. 数据库的智能化

所谓数据库的智能化是指把数据库视为一个 AI 系统或专家系统，借鉴 AI 技术来提高 DBMS 的表达、推理和查询能力，例如：

（1）用知识表示方法来描述 DBMS 中的数据模型、完整性约束条件、安全性约束条件。

（2）将更多的语义信息以知识表示形式存入系统，利用知识表示与搜索方法去描述和开发复杂对象的数据库系统，以扩展数据库的应用领域。

（3）用户界面的智能化，它能以自然语言理解的形式为用户使用数据库提供灵活、方便、友好的用户界面，建立数据库的用户界面管理系统，提供用户使用数据库的经验和知识。

2. 智能化的数据库

所谓智能化的数据库是指扩大数据库的功能，使其不但具有传统数据库的现有功能，还具有一些 AI 能力，以提高数据库的演绎、推理功能和智能化的程度。例如：

（1）数据库的演绎功能：能从现有的数据库的数据中演绎和导出一些新数据。具有这种功能的数据库称为演绎数据库系统。

（2）数据库的搜索功能，将数据库中的操作与 AI 中的问题求解—搜索技术结合，适当扩充其功能使数据库具有智能搜索能力。

（3）数据库的问题求解能力：扩大数据库功能，使之成为能共享信息的面向知识处理的问题求解系统，这种系统称为专家数据库系统。

（4）数据库的归纳功能：能将数据库中的数据归纳成规则，存入知识库，进一步可使数据库具有学习能力。

（5）数据库的知识管理能力：在数据库管理事实（数据）的基础上，加以扩充使其具有管理规则的功能，即管理知识的能力，从而形成一个知识库及知识库管理系统。

总之，在数据库领域引进 AI 的思想和方法，并将这两种技术有效结合起来，必将为解决传统数据库技术的缺陷、不足和各种难题提供新的方法与手段。

这里要说明一点，由于知识库的概念是来自数据库和人工智能（及其分支—知识工程）这两个不同的领域，因此数据库界（从数据库的角度引入 AI 技术）和人工智能界（从 AI 的角度引入数据库技术）对知识库有不同的理解和定义。

通常，从数据库角度引入 AI 技术来开发具有智能的数据库系统，主要是从逻辑程序设计的观点出发进行知识库系统的研究，以改进和扩充数据库的功能和执行效率。其功能体现在演绎(推理)能力的扩充、语义知识的引入、知识的获取、知识和数据的有效组织及管理等方面；而效率则指数据库对用户查询的快速响应与查询优化上。

例如，J.D.Ullman 定义一个知识库系统是具有如下两种特征的逻辑程序设计系统：一是有一个既作为查询语言又作为宿主语言的描述性语言；另一个是支持数据库系统的主要功能，如支持大批量的数据的高效存取、数据共享、并发控制及错误恢复等。另一个知识库系统的定义是 D.H.Warren 给出的，他认为，一个知识库系统应能够有效地处理中等规模的知识库(由 3 000 个谓词、3 万条规则和 300 万个事实组成，总存储量达 30 MB)的逻辑程序设计系统。

以上知识库系统的定义代表了数据库专家对知识库的理解。

从 AI 角度所理解的知识库系统则更为广泛，通常是指利用人类所认识的各种知识进行推理、联想、学习和问题求解的智能计算机信息系统。在 AI 研究中，将数据定义为特定实例(事实)的信息，而知识则定义为一般(抽象)概念信息。从 AI 角度所看待的知识库系统主要操纵和管理的对象是知识。

目前，多数数据库工作者是将知识库系统视为智能数据库系统来看待。但多数 AI 工作者认为，由于知识库和智能数据库操纵和管理的对象不同（是数据还是知识），加之目前在 AI 领域知识库已成为实现专家系统及其他知识处理系统的主要组成部分而独立出来的一门学科，因此他们主张将知识库系统与智能数据库系统相区别。

有关知识库系统的详细讨论，可参阅数据库丛书的《知识库系统》一书或《数据库综合大词典》中的第 10 篇等文献。

四、主动数据库

主动数据库（Active Data Base）是相对于传统数据库的被动性而言的。许多实际的应用领域，如计算机集成制造系统、管理信息系统、办公室自动化系统中常常希望数据库系统在紧急情况下能根据数据库的当前状态，主动适时地做出反应，执行某些操作，向用户提供有关信息。传统数据库系统是被动的系统，它只能被动地按照用户给出的明确请求执行相应的数据库操作，很难充分适应这些应用的主动要求，因此在传统数据库基础上，结合人工智能技术和面向对象技术提出了主动数据库。

主动数据库的主要目标是提供对紧急情况及时反应的能力，同时提高数据库管理系统的模块化程度。主动数据库通常采用的方法是在传统数据库系统中嵌入 ECA（即事件-条件-动作）规则，在某一事件发生时引发数据库管理系统去检测数据库当前状态，看是否满足设定的条件，若条件满足，便触发规定动作的执行。

为了有效地支持 ECA 规则，主动数据库的研究主要集中于解决以下问题：

（1）主动数据库的数据模型和知识模型。即如何扩充传统的数据库模型，使之能描述、存储、管理 ECA 规则，适应于主动数据库的要求。

（2）执行模型。即 ECA 规则的处理和执行方式，是对传统数据库系统事务模型的发展和扩充。

（3）条件检测。是主动数据库系统实现的关键技术之一，由于条件的复杂性，如何高效地对条件求值是对提高系统效率有很大的影响。

（4）事务调度。 与传统数据库系统中的数据调度不同，它不仅要满足并发环境下的可串行化要求而且要满足对事务时间方面的要求。目前，对执行时间估计的代价模型是有待解决的难题。

（5）体系结构。目前，主动数据库的体系结构大多是在传统数据库管理系统的基础上，扩充事务管理部件和对象管理部件以支持执行模型和知识模型，并增加事件侦测部件、条件检测部件和规则管理部件。

（6）系统效率。系统效率是主动数据库研究中的一个重要问题，是设计各种算法和选择体系结构时应主要考虑的设计目标。

主动数据库是目前数据库技术中一个活跃的研究领域，近年来的研究已取得了很大的成果。当然，主动数据库还是一个正在研究的领域，许多问题还有待进一步研究解决。

有关主动数据库的详细讨论，可参阅数据库丛书的《特种数据库》一书或《数据库综合大词典》中的第 22 篇等文献。

五、多媒体数据库

媒体是信息的载体。多媒体是指多种媒体，如数字、正文、图形、图像和声音的有机集成，而不是简单的组合。其中数字、字符等称为格式化数据；文本、图形、图像、声音、视频等称为非格式化数据，非格式化数据具有数据量大、处理复杂等特点。多媒体数据库实现对格式化和非格式化的多媒体数据的存储、管理和查询，其主要特征有：

（1）多媒体数据库应能够表示多种媒体的数据。非格式化数据表示起来比较复杂，需要根据多媒体系统的特点来决定表示方法。如果感兴趣的是它的内部结构且主要是根据其内部特定成份来检索，则可把它按一定算法映射成包含它所有子部分的一张结构表，然后用格式化的表结构来表示它。如果感兴趣的是它本身的内容整体，要检索的也是它的整体，则可以用源数据文件来表示它，文件由文件名来标记和检索。

（2）多媒体数据库应能够协调处理各种媒体数据，正确识别各种媒体数据之间在空间或时间上的关联。例如，关于乐器的多媒体数据包括乐器特性的描述，乐器的照片，利用该乐器演奏某段音乐的声音等，这些不同媒体数据之间存在着自然的关联，比如多媒体对象在表达时必须保证时间上的同步特性。

（3）多媒体数据库应提供比传统数据管理系统更强的适合非格式化数据查询的搜索功能。例如可以对 Image 等非格式化数据作整体和部分搜索。

多媒体数据库应提供特种事务处理与版本管理能力。

有关多媒体数据库的讨论，可参阅《数据库综合大词典》中的第 14 篇及其他文献。

六、模糊数据库系统

什么是模糊数据库？要了解模糊数据库概念首先要了解“模糊”一词的概念。现实世界中对象的模糊性是指其表露的不清楚性和不完全性。它不同于事物或事件的随机性，一般来说随机性所描述的事件或现象本身含义是清楚的，而模糊性所描述的现象或概念本身的界限是不清楚的。例如，一张白纸上的一片墨迹，由于墨水外渗，墨迹边缘不清楚，要判定边缘上的一些位置是否已被墨迹污染了，还是没有被污染都不能明确回答，它只能用一些“肯定不可能，极不可能，很小可能，较小可能，可能，较大可能，很大可能，极大可能，肯定可能”等词语来描述该位置是否在该墨迹之中。

模糊性是客观世界的一个重要属性。不但事物的静态结构方面存在着模糊性，而且事物间互相作用的动态行为方面也存在着模糊性。

传统的数据库系统描述和处理的是精确的或确定的客观事物，但不能描述

和处理模糊性和不完全性等概念，这是一个很大的不足。为此，有必要把不完全性、不确定性和模糊性引进数据库，并且开展模糊数据库的理论和实现技术的研究。从20世纪80年代开始人们就陆续提出了模糊数据库。但含义比较简单，仅仅是允许在数据库中存放一些模糊数据和不确定数据而已。

由于理论和实现技术上的困难，模糊数据库技术，十多年来的发展并不算快，使得至今尚无商品化的模糊数据库管理系统推出。

从20世纪80年代中后期开始，国内外出现了一系列对模糊关系数据库的有关概念和方法的研究，建立了一套较合理的理论体系，包括各种模糊数据表示、模糊距离的度量、模糊数据模型、模糊语言、模糊查询方法等，并且摸索了一些较实用的实现技术，在计算机上实现了一些基于上述理论和技术的模糊数据库管理系统的原型。

模糊数据库系统可以定义为：存储、组织、管理和操作模糊数据的数据库系统。它除了具有普通数据库系统的公共特性外，在以下方面体现了模糊性：

- 存储的是以各种形式表示的模糊数据；
- 数据结构和数据之间的联系是模糊的；
- 数据上的运算和操作也是模糊的；
- 对数据的约束，包括完整性和安全性，是模糊的；
- 用户使用数据库的窗口—用户视图是模糊的；
- 数据的一致性和无冗余性的定义也是模糊的。

精确数据可以看成是模糊数据的特例。

由于模糊数据库和普通数据库的主要区别是数据的模糊性和对数据操作的模糊性，因此对它的研究内容和关键技术主要是围绕这些问题提出的。例如：

（1）模糊数据的表示和语义解释

任何包含模糊性的数据都称为模糊数据，由于模糊性可以有多种表示形式，所以模糊数据也有多种表示形式。例如，用模糊集表示的模糊数、正态模糊数、模糊区间数、模糊中心数和模糊语言值等。还可以构造更复杂的结构模糊数据，如模糊复数、模糊数集合、模糊元组、模糊关系等。

（2）模糊数据模型，包括结构模糊数据的构造、模糊数据上的模糊操作和运算的定义。

目前主要有模糊关系模型、模糊网络模型、模糊层次模型、模糊实体联系模型、模糊面向对象数据模型等，它们大多是从相应的非模糊数据模型经过模糊化得到的。

（3）模糊数据库管理系统的实现技术。因为模糊数据模型和模糊数据库语言比精确时复杂得多，所以在实现模糊数据库管理系统时，无论在功能上、逻辑结构上、在物理组织和存储形式上都相应地要复杂得多。

（4）模糊数据库和模糊知识库的结合。

模糊数据库的理论和技术经过二十多年来的研究和发展，一方面已经取得了不少成果，另一方面关于模糊数据库的理论和技术的研究，无论在国内国外，都仍处于发展阶段，尚未达到成熟和完善，还有许多问题有待进一步研究。我国何新贵研究员和他的研究小组在十多年的研究中做出了不少很好的贡献。

尽管模糊数据库系统如今还不够完善，但它已在模式识别、过程控制、案情侦破、医疗诊断、工程设计、营养咨询、公共服务以及专家系统等领域得到较好的应用，显示了广阔的应用前景。

在未来的发展中，除在理论和方法上仍需做各种必须的改进和完善之外，以下几方面的发展将具有特别的意义：

（1）把模糊数据库技术有机地与模糊知识处理相结合，建立模糊专家系统开发工具和环境，开发各种具体领域的模糊专家系统。特别在模糊决策、模糊推理、模糊控制、模糊综合评判和模糊诊断等应用领域发挥其特有的作用。

（2）把模拟计算技术纳入模糊数据库的操作和处理中。若能用一些模拟计算部件来完成各种模糊运算，可以充分发挥模拟计算快速和逼真的长处。

（3）把模糊数据库技术与主动处理技术有机结合，发展模糊主动数据库乃至模糊主动知识库系统也将是一个很值得研究的发展方向。

（4）借助模糊数学的理论和方法，研究更符合实际的模糊算术和模糊代数理论以及模糊集间的近似程度的度量方法等也应是今后理论研究工作的重点。

有关模糊数据库的详细讨论，可参阅数据库丛书的《特种数据库》一书或其他文献。

12.2.3 面向应用领域的数据库新技术

数据库技术被应用到特定的领域中，出现了数据仓库、工程数据库、统计数据库、空间数据库、科学数据库等多种数据库（如图 12.5），使数据库领域的应用范围不断扩大。

这些数据库系统都明显地带有某一领域应用需求的特征。由于传统数据库系统的局限性，无法直接使用当前 DBMS 市场上销售的通用的 DBMS 来管理和处理这些领域内的数据对象。因而广大数据库工作者针对各个领域的数据库特征探索和研制了各种特定的数据库系统，取得了丰硕的成果。不仅为这些应用领域建立了可供使用的数据库系统，有的已实用化，而且为新一代数据库技术的发展作出了贡献。

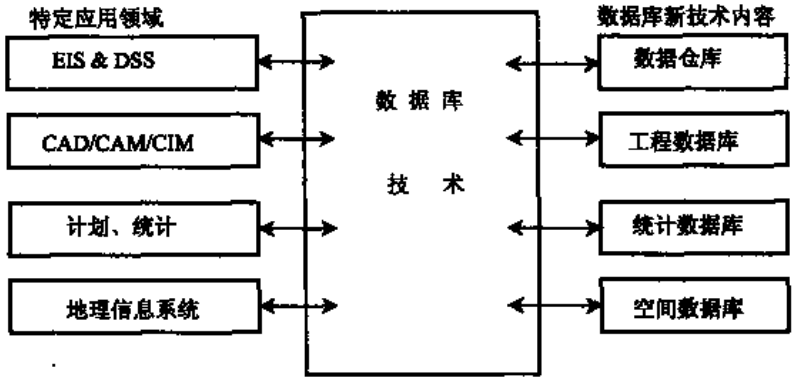


图 12.5 特定应用领域中的数据库技术

一、数据仓库(Data Warehouse)

数据仓库是信息领域中近年来迅速发展起来的数据库新技术。数据仓库的建立能充分利用已有的数据资源，把数据转换为信息，从中挖掘出知识，提炼成智慧，最终创造出效益。所以，越来越多的企业开始认识到数据仓库应用所带来的好处。

传统的数据库技术是以单一的数据资源，即数据库为中心，进行从事务处理、批处理到决策分析等各种类型的数据处理工作。然而，不同类型的数据处理有着其不同的处理特点，以单一的数据组织方式进行组织的数据库并不能反映这种差异，满足不了数据处理多样化的要求。近年来，随着数据库应用的广泛普及，人们对数据处理的这种多层次特点有了更清晰的认识，逐渐认识到计算机系统中存在着两类不同的处理：操作型处理和分析型处理。

操作型处理也叫事务处理，是指对数据库联机的日常操作，通常是对一个或一组记录的查询和修改，主要是为企业的特定应用服务的，人们关心的是响应时间，数据的安全性和完整性。

分析型处理则用于管理人员的决策分析。例如：DSS，EIS 和多维分析等，经常要访问大量的历史数据。两者之间的巨大差异使得操作型处理和分析型处理的分离成为必然。

这种分离，划清了数据处理的分析型环境与操作型环境之间的界限，从而由原来的以单一数据库为中心的数据环境发展为一种新环境：体系化环境。体系化环境由操作型环境和分析型环境（包括全局级数据仓库、部门级数据仓库、个人级数据仓库）构成。

数据仓库是体系化环境的核心，它是建立决策支持系统（DSS）的基础。

1. 从数据库到数据仓库

数据库系统作为数据管理的主要手段，主要用于事务处理。在这些数据库中已经保存了大量的日常业务数据。传统的 DSS 一般是直接建立在这种事务处理环境上的。

数据库技术一直力图使自己能胜任从事务处理、批处理到分析处理的各种类型的信息处理任务。尽管数据库技术在事务处理方面的应用获得了巨大的成功，但它对分析处理的支持一直不能令人满意，尤其是当以事务处理为主的联机事务处理（OLTP）应用与以分析处理为主的 DSS 应用共存于同一个数据库系统中时，这两种类型的处理发生了明显的冲突。人们逐渐认识到事务处理和分析处理具有极不相同的性质，直接使用事务处理环境来支持 DSS 是不合适的。

具体来说，有如下原因使得事务处理环境不适宜 DSS 应用：

（1）事务处理和分析处理的性能特性不同

在事务处理环境中，用户的行为特点是数据的存取操作频率高而每次操作处理的时间短，因此系统可以允许多个用户按分时方式使用系统资源，同时保持较短的响应时间，OLTP（联机事务处理）是这种环境下的典型应用。

在分析处理环境中，用户的行为模式与此完全不同，某个 DSS 应用程序可能需要连续运行几个小时，消耗大量的系统资源。将具有如此不同处理性能的两种应用放在同一个环境中运行显然是不适当的。

（2）数据集成问题

DSS 需要集成的数据。全面而正确的数据是有效的分析和决策的首要前提，相关数据收集得越完整，得到的结果就越可靠。因此，DSS 不仅需要企业内部各部门的相关数据，还需要企业外部、竞争对手等处的相关数据。

而事务处理的目的在于使业务处理自动化，一般只需要与本部门业务有关的当前数据，对整个企业范围内的集成应用考虑很少。当前绝大部分企业内数据的真正状况是分散而非集成的，尽管每个单独的事务处理应用可能是高效的，能产生丰富的细节数据，但这些数据却不能成为一个统一的整体。对于需要集成数据的 DSS 应用来说，必须自己在应用程序中对这些纷杂的数据进行集成。

可是，数据集成是一项十分繁杂的工作，都交给应用程序完成会大大增加程序员的负担。并且，如果每做一次分析，都要进行一次这样的集成，将会导致极低的处理效率。DSS 对数据集成的迫切需要可能是数据仓库技术出现的最重要动因。

（3）数据动态集成问题

由于每次分析都进行数据集成的开销太大，一些应用仅在开始对所需的数据进行了集成，以后就一直以这部分集成的数据作为分析的基础，不再与数据

源发生联系，称这种方式的集成为静态集成。静态集成的最大缺点在于如果在数据集成后数据源中数据发生了改变，这些变化将不能反映给决策者，导致决策者使用的是过时的数据。对于决策者来说，虽然并不要求随时准确地探知系统内的任何数据变化，但也不希望他所分析的是几个月以前的情况。因此，集成数据必须以一定的周期（例如几天或几周）进行刷新，称为动态集成。显然，事务处理系统不具备动态集成的能力。

（4）历史数据问题

事务处理一般只需要当前数据，在数据库中一般也只存储短期数据，且不同数据的保存期限也不一样，即使有一些历史数据保存下来了，也被束之高阁，未得到充分利用。但对于决策分析而言，历史数据是相当重要的，许多分析方法必须以大量的历史数据为依托。没有对历史数据的详细分析，是难以把握企业的发展趋势的。

综上所述，可以看出 DSS 对数据在空间和时间的广度上都有了更高的要求。而事务处理环境难以满足这些要求。

（5）数据的综合问题

在事务处理系统中积累了大量的细节数据，一般而言，DSS 并不对这些细节数据进行分析，原因一是细节数据数量太大，会严重影响分析的效率；原因二是太多的细节数据不利于分析人员将注意力集中于有用的信息上。因此，在分析前，往往需要对细节数据进行不同程度的综合。而事务处理系统不具备这种综合能力，而且根据规范化理论，这种综合还往往因为是一种数据冗余而加以限制。

以上这些问题表明在事务型环境中直接构建分析型应用是一种失败的尝试。数据仓库本质上是对这些存在问题的回答。但是数据仓库的主要驱动力并不是过去的缺点，而是市场商业经营行为的改变，市场竞争要求捕获和分析事务级的业务数据。

建立在事务处理环境上的分析系统无法达到这一要求。要提高分析和决策的效率和有效性，分析型处理及其数据必须与操作型处理和数据相分离。必须把分析数据从事务处理环境中提取出来，按照 DSS 处理的需要进行重新组织，建立单独的分析处理环境，数据仓库正是为了构建这种新的分析处理环境而出现的一种数据存储和组织技术。

2. 数据仓库的定义和特点

事务处理和分析处理具有极不相同的性质，因而两者对数据也有着不同的要求。数据仓库概念的创始人 W.H. Inmon 在其《Building Data Warehouse》一书中，列出了操作型数据与分析型数据之间的区别(如表 12.1)。

表 12.1 操作型数据和分析型数据的区别

操作型数据	分析型数据
细节的	综合的，或提炼的
在存取瞬间是准确的	代表过去的数据
可更新	不更新
操作需求事先可知道	操作需求事先不清楚
生命周期符合 SDLC	完全不同的生命周期
对性能要求高	对性能要求宽松
一个时刻操作一单元	一个时刻操作一集合
事务驱动	分析驱动
面向应用	面向分析
一次操作数据量小	一次操作数据量大
支持日常操作	支持管理需求

表 12.1 所列出的分析型数据的特点可以概括为四点，也就是数据仓库数据的四个基本特征：

- 数据仓库的数据是面向主题的
- 数据仓库的数据是集成的
- 数据仓库的数据是不可更新的
- 数据仓库的数据是随时间不断变化的

那么什么是数据仓库呢？ 数据仓库就是一个面向主题的、集成的、不可更新的、随时间不断变化的数据集合，用以支持企业或组织的决策分析处理。下面简要讨论数据仓库的四个基本特征。

（1）主题与面向主题

与传统数据库面向事务处理应用进行数据组织的特点相对应， 数据仓库中的数据是面向主题进行组织的。什么是主题呢？主题是一个抽象的概念，是在较高层次上将企业信息系统中的数据综合、归类并进行分析利用的抽象；在逻辑意义上，它对应企业中某一宏观分析领域所涉及的分析对象。所谓较高层次是相对面向应用的数据组织方式而言的， 是指按照主题进行数据组织的方式具有更高的数据抽象级别。

比如一家商场，概括分析领域的对象，应有的主题包括供应商、商品、顾客等。可以看出，基于主题组织的数据被划分为各自独立的领域，每个领域有自己的逻辑内涵而不相交叉。而按业务建立的是销售、采购、库存管理以及人事管理子系统，数据库模式有：订单、订单细则、供应商、顾客、销售、领料单、进料单、库存、库房、员工、部门等。可以发现这些数据是为处理具体应

用而组织的，它对于数据内容的划分并不适合分析的需求。

“主题”在数据仓库中是由一系列表（table）实现的。一个主题之下表的划分可以按数据的综合、数据所属时间段进行划分。但无论如何，基于一个主题的所有表都含有一个称为公共码键的属性作为其主码的一部分。公共码键将一个主题的各个表联系起来。

同时，由于数据仓库中的数据都是同某一时刻联系在一起的，所以每个表除了其公共码键之外，还必然包括时间成分作为其码键的一部分。

有一点需说明的是，同一主题的表未必存在同样的介质中，根据数据被关心的程度不同，不同的表分别存储在磁盘、磁带、光盘等不同介质中。一般而言，年代久远的、细节的或查询频率低的数据存储在廉价慢速设备（如磁带）上，而近期的、综合的或查询频率高的数据则保存在磁盘上。

（2）数据仓库是集成的

前面已经讲到，操作型数据与分析型数据之间差别甚大，数据仓库的数据是从原有的分散的数据库数据中抽取来的，因此数据在进入数据仓库之前，必然要经过加工与集成，统一与综合。这一步实际是数据仓库建设中最关键、最复杂的一步。

首先，要统一原始数据中所有矛盾之处，如字段的同名异义、异名同义，单位不统一，字长不一致等；然后将原始数据结构作一个从面向应用到面向主题的大转变；还要进行数据综合和计算；数据仓库中的数据综合工作可以在抽取数据时生成， 也可以在进入数据仓库以后进行综合时生成。

（3）数据仓库是不可更新的

数据仓库主要是供决策分析之用的，所涉及的数据操作主要是数据查询，一般情况下并不进行修改操作。数据仓库存储的是相当长一段时间内的历史数据，是不同时点数据库快照的集合，以及基于这些快照进行统计、综合和重组的导出数据，不是联机处理的数据。因而，数据一经集成进入数据库后是极少或根本不更新的，是稳定的。

（4）数据仓库是随时间变化的

数据仓库中的数据不可更新是指数据仓库的用户进行分析处理时是不进行数据更新操作的。但并不是说在数据仓库的整个生存周期中数据集合是不变的。

数据仓库的数据是随时间的变化不断变化的，这一特征表现在以下三方面：

第一，数据仓库随时间变化不断增加新的数据内容。数据仓库系统必须不断捕捉 OLTP 数据库中新的数据，追加到数据仓库中去， 也就是要不断地生成 OLTP 数据库的快照， 经统一集成后增加到数据仓库中去；但对于每次的数据库快照确实是不再变化了。捕捉到的新数据只是又生成一个数据库的快照增加进数据仓库，而不会覆盖原来的快照。

第二，数据仓库随时间变化不断删去旧的数据内容。数据仓库的数据也有存储期限，一旦超过了这一期限， 过期数据就要被删除。只是数据仓库内的数据时限要远远长久于操作型数据的时限。在操作型环境中一般只保存 60~90 天的数据，而在数据仓库中则需要保存较长时限的数据（如 5~20 年），以适应 DSS 进行趋势分析的要求。

第三，数据仓库中包含大量的综合数据，这些综合数据中很多与时间有关，如数据按照某一时间段进行综合，或隔一定的时间片进行抽样等，这些数据就会随着时间的变化不断地进行重新综合。

3. 分析工具——数据仓库系统的重要组成部分

有了数据就如同有了矿藏，而要从大量数据中获得决策所需的数据就如同开采矿藏一样，必须要有工具。仅拥有数据仓库而没有高效的数据分析工具就只能望“矿”兴叹。

20 世纪 80 年代，随着数据库技术的发展开发了一整套以数据库管理系统(DBMS)为核心的第四代开发工具产品，如 FORMS，REPORTS，MENUS，GRAPHICS 等。这些第四代开发工具有效地帮助了应用开发人员快速建立数据库应用系统，使数据库技术获得了广泛的应用，有效地支持了 OLTP 应用，人们从中认识到，仅有引擎（DBMS）是不够的，工具同样重要。数据分析工具的迅速发展正是得益于这一经验。

（1）联机分析处理技术及工具

联机分析处理（OLAP）应用是不同于与联机事务处理（OLTP）的一类应用。

从 1991 年 W.H.Inmon 提出 DW 概念到 E.F.Codd 于 1993 年提出 OLAP 概念仅仅两年，而从 OLAP 概念提出到 OLAP 工具的推出则时间更短，几乎是同时的。

E.F.Codd 在“Providing OLAP to User Analysts”一文（见《PC World》1993，9）中完整地定义了 OLAP 的概念，多维分析的概念，并给出了数据分析从低级到高级的四种模型以及 OLAP 的十二条准则，这些都对 OLAP 技术的发展，产品的功能产生了重大影响。短短的几年， OLAP 技术发展迅速，产品越来越丰富。它们具有灵活的分析功能，直观的数据操作和可视化的分析结果表示等突出优点，从而使用户对基于大量数据的复杂分析变得轻松而高效。

在 OLAP 中，特别应指出的是多维数据视图的概念和多维数据库(MDB)的实现。维是人们观察现实世界的角度，决策分析需要从不同的角度观察分析数据，以多维数据为核心的多维数据分析是决策的主要内容。早期的决策分析程序中分析方法和数据结构是紧密捆绑在一个应用程序当中的，因此，对数据施加不同的分析方法就十分困难了。

数据仓库技术把决策分析中数据结构和分析方法相分离，这才可能研制出通用而灵活的分析工具，使分析工具的产品化成为可能。

目前 OLAP 工具产品的实现可分为两大类，一类是基于多维数据库的，一类是基于关系数据库的。两者相同之处是基本数据源仍是基于关系数据模型的，向用户呈现的也都是多维数据视图。不同之处是前者把分析所需的数据从数据库或数据仓库中抽取出来物理地组织成多维数据库，后者则利用关系表来模拟多维数据，并不物理地生成多维数据库。

（2）数据挖掘技术和工具

数据挖掘（Data Mining，简称 DM）是从超大型数据库（VLDB）或数据仓库中发现并提取隐藏在内部的信息的一种新技术。目的是帮助决策者寻找数据间潜在的关联，发现经营者被忽略的要素，而这些要素对预测趋势、决策行为也许是十分有用的信息。

数据挖掘技术涉及数据库技术、人工智能技术、机器学习、统计分析等多种技术，它使 DSS 系统跨入了一个新阶段。传统的 DSS 系统通常是在某个假设的前提下通过数据查询和分析来验证或否定这个假设，而数据挖掘技术则能够自动分析数据，进行归纳性推理，从中发掘出数据间潜在的模式；或产生联想，建立新的业务模型帮助决策者调整市场策略，找到正确的决策。

有关数据挖掘技术的研究仅有四五年时间，已从理论研究走向产品开发，这实在是十分惊人的速度。而且据国外报导，虽然数据挖掘工具产品尚不成熟，但其市场份额却在增加，越来越多的大中型企业开始利用数据挖掘工具分析公司的数据，而且认为“如果不在竞争对手之前使用数据挖掘工具，等待你的将是失败！”

综上所述，数据仓库系统是多种技术的综合体，它由数据仓库(DW)、数据仓库管理系统(DWMS)、数据仓库工具三个部分组成（如图 12.6）。在整个系统中，数据仓库居于核心地位，是信息挖掘的基础； 数据仓库管理系统负责管理整个系统的运转，是整个系统的引擎； 而数据仓库工具则是整个系统发挥作用的关键，只有通过高效的工具，数据仓库才能真正发挥出数据宝库的作用。

4. 基于数据库技术的 DSS 解决方案

技术的进步，不懈的努力使人们终于找到了基于数据库技术的 DSS 的解决方案，这就是：

DW＋OLAP＋DM ⟶ DSS 的可行方案

数据仓库、OLAP 和数据挖掘是作为三种独立的信息处理技术出现的。数据仓库用于数据的存储和组织，OLAP 集中于数据的分析，数据挖掘则致力于知识的自动发现。它们都可以分别应用到信息系统的设计和实现中，以提高

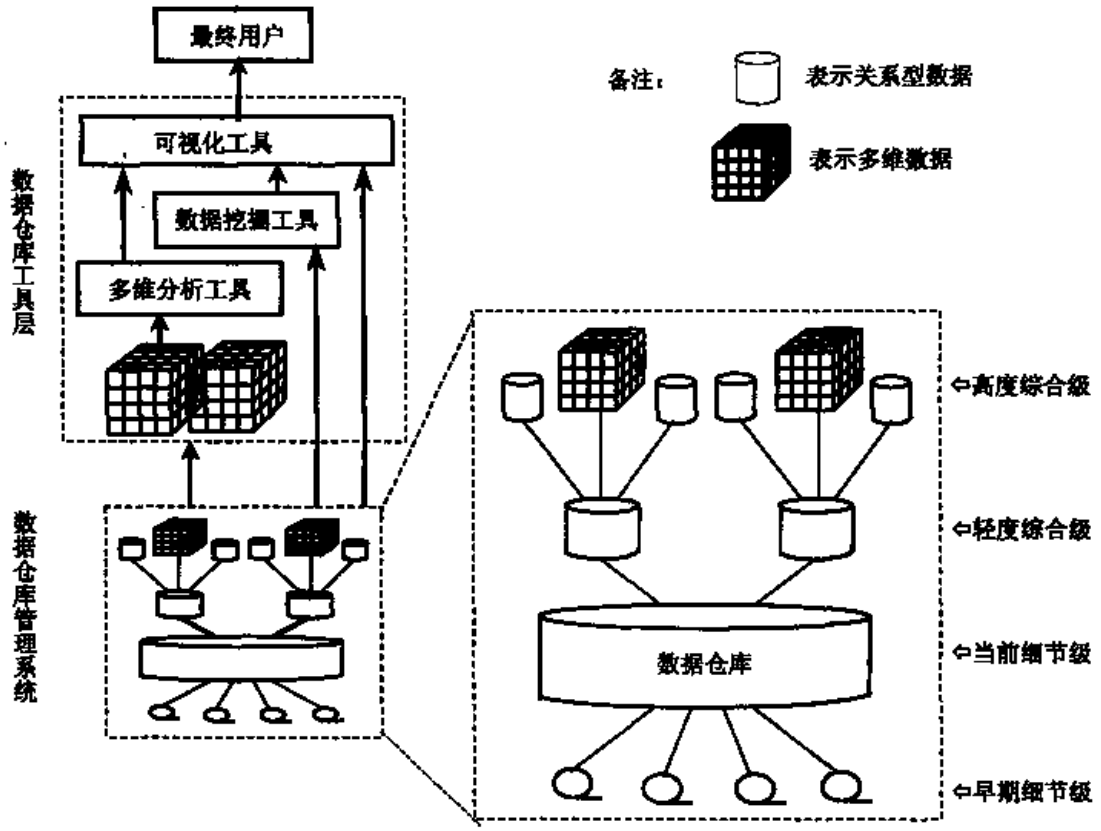


图 12.6 数据仓库体系结构图

相应部分的处理能力。但是，由于这三种技术内在的联系性和互补性，将它们结合起来即是一种新的 DSS 构架。这一构架以数据库中的大量数据为基础，其特点是：

（1）在底层的数据库中保存了大量的事务级细节数据。这些数据是整个 DSS 系统的数据来源。

（2）数据仓库对底层数据库中的事务级数据进行集成、转换、综合，重新组织成面向全局的数据视图，为 DSS 提供数据存储和组织的基础。

（3）OLAP 从数据仓库中的集成数据出发，构建面向分析的多维数据模型，再使用多维分析方法从多个不同的视角对多维数据进行分析、比较，分析活动从以前的方法驱动转向了数据驱动，分析方法和数据结构实现了分离。

（4）数据挖掘以数据仓库和多维数据库中的大量数据为基础，自动地发现数据中的潜在模式，并以这些模式为基础自动地作出预测。数据挖掘表明知识就隐藏在日常积累下来的大量数据之中，仅靠复杂的算法和推理并不能发现知识，数据才是知识的真正源泉。数据挖掘为 AI 技术指出了一条新的发展道路。

有关数据仓库的详细讨论，可参阅数据库丛书的《数据仓库技术和联机分析处理》一书或其他文献。

二、工程数据库(Engineering Data Base)

工程数据库是一种能存储和管理各种工程设计图形和工程设计文档，并能为工程设计提供各种服务的数据库。传统的数据库能很好地存储规范数据和进行事务处理，而在 CAD/CAM，CIM，CASE 等 CAx 的工程应用领域，对具有复杂结构和工程设计内涵的工程对象以及工程领域中的大量“非经典”应用，传统的数据库则无能为力。工程数据库正是针对工程应用领域的需求而提出来的，目的是利用数据库技术对工程对象有效地加以管理，并提供相应的处理功能及良好的设计环境。

工程数据库管理系统是用于支持工程数据库的数据库管理系统，基于工程数据库中数据结构复杂，相互联系紧密，数据存储量大的特点，工程数据库管理系统的功能与传统数据库管理系统有很大不同，主要应具有以下功能：

（1）支持复杂对象(如图形数据、工程设计文档)的表示和处理；

（2）可扩展的数据类型；

（3）支持复杂多样的工程数据的存储和集成管理；

（4）支持变长结构数据实体的处理；

（5）支持工程长事务和嵌套事务的并发控制和恢复；

（6）支持设计过程中多个不同数据版本的存储和管理；

（7）支持模式的动态修改和扩展；

（8）支持多种工程应用程序等。

在工程数据库的设计过程中，由于传统的数据模型难以满足工程应用的要求，需要运用当前数据库研究中的一些新的模型技术，如扩展的关系模型、语义模型、面向对象的数据模型。

目前的工程数据库的研究和开发虽然已取得了很大的成绩，但要全面达到应用所要求的目标仍有待进一步努力。

有关工程数据库的讨论，可参阅《数据库综合大词典》中的第 17 篇及其他文献。

三、统计数据库(Statistical Data Base)

统计数据库是一种用来对统计数据进行存储、统计、分析的数据库系统。

现实社会中对国民经济、科学技术、文化教育、国防军事、日常生活的大量调查数据是人类社会活动结果的实际反映，是重要的信息资源。采用数据库技术实现对统计数据的管理，对于充分发挥统计信息的作用具有重大的意义。

统计数据具有层次型特点，但并不完全是层次型结构。统计数据也有关系型特点，但关系型也不完全满足需要。概括起来，统计数据具有以下的基本

特性：

（1）分类属性和统计属性

统计数据库中的数据按语义可以分为两类：一类是用于统计分析的计量数据；另一类是用于说明计量数据的参数数据。例如，某系、某班、某年龄段的学生人数这一统计数据中，某系、某班、某年龄段这三个数据是参数数据，学生人数是计量数据。

（2）多维性

虽然一般统计表都是二维表，但统计数据实际上是多维的。例如经济统计信息，由统计指标名称（如工业总产值）、统计时间（1999 年）、统计空间范围（全国）、统计分组特性（按行业）、统计度量种类（万元）等相互独立的多种因素方可确切地定义出一批数据。反映在数据结构上就是一种多维性。

表示参数数据的属性称为分类属性，计量数据的属性称为统计属性。分类属性的笛卡尔积构成了一个多维空间，计量数据是多维空间中的点，统计数据库可以看成是多维空间中的点集。

多维性是统计数据最基本的特点。统计数据库的多维性确定了它的其他属性。

（3）分类属性的层次结构

分类属性具有复杂的层次结构。例如，上例中每个系有多个班，下面包括多个年龄段。这种层次结构可以用有向树来表示。每条从根到叶的路径对应一组分类属性的值，确定一个统计属性值。

（4）微数据(Micro Data)和宏数据(Macro Data)

统计数据库中的数据可分为两类：微数据(Micro Data)和宏数据(Macro Data)。微数据描述的是个体或事件的信息，而宏数据是综合统计数据，它可以直接来自应用领域，也可以是微数据的综合分析结果。

统计数据库中常用的操作有：抽样、邻近搜索、估计与插值、转置、聚集及复杂的分析操作。这些操作不同于关系数据库中传统的查询、增加、删除、修改操作。若把这些操作放在应用程序一级的统计软件包来实现，则系统效率会较低，人们希望统计数据库能从 DMBS 一级来支持以上的数据特性和操作。

因此，统计数据库研究的问题和技术有：

（1）数据模型

人们已提出了概念统计模型、统计对象描述模型、统计关系表模型、代数语义数据模型等多种具有更丰富语义的数据模型，为了能够较好地描述统计数据及其语义。

（2）查询语言和用户接口

研究支持统计数据操作的各种查询语言。例如：

① 人们研究了基于扩展关系模型的查询语言，设计了关系代数、集合代数、矩阵代数和向量代数操作；

② 人们研究了统计报表查询语言，这类查询语言把统计报表作为数据模型的数据对象，提供了直接在统计报表上进行统计分析的各种操作。

③ 人们研究了友善的用户接口，如基于自然语言的统计分析请求和直观清晰的统计分析结果显示。

（3）统计数据的物理组织

统计数据的多维性和稀疏性对数据的物理组织提出了很多新的要求，其中主要的技术包括：

- 数据压缩
- 多维数据的物理组织

统计数据库需要数据压缩的原因是统计数据库中分类属性数常常很多，有时可达数百个，而且分类属性值重复出现的频率极高，使得分类属性的多维空间十分稀疏，因此数据压缩和解压缩是多维数据物理组织的重要技术。

统计数据库的多维性使得多维数据结构成为统计数据库中重要的物理组织方法。人们提出了 GRID 文件、QUAD 树、K-D-B 树等多种多维数据结构。

（4）时序数据

统计数据的任何统计量都离不开时间因素，时间属性是统计数据一个最基本的属性，从这个角度看，统计数据是时序数据。人们对时序数据的许多研究成果如时序数据操作（时序连接），时序数据模型等可以用到统计数据库之中。

（5）统计数据库安全性

统计数据库在安全性方面有特殊的要求，要防止某些用户在统计数据库中利用对统计数据（如综合数据）的合法查询推导出该用户无权了解的某一个体的具体数据。已在本书 9.3 节中介绍了统计数据库的安全性问题。

有关统计数据库的讨论，可参阅《数据库综合大词典》中的第 15 篇及其他文献。

四、空间数据库(Spatial Data Base)

空间数据是用于表示空间物体的位置、形状、大小和分布特征等诸方面信息的数据，适用于描述所有二维、三维和多维分布的关于区域的现象。

空间数据的特点是不仅包括物体本身的空间位置及状态信息，还包括表示物体的空间关系（即拓扑关系）的信息。属性数据为非空间数据，用于描述空间物体的性质，对空间物体进行语义定义。

空间数据库系统是描述、存储和处理空间数据及其属性数据的数据库系统。

空间数据库是随着地理信息系统（Geographical Information System）的开

发和应用而发展起来的数据库新技术。目前，空间数据库系统不是独立存在的系统，它与应用紧密结合，大多数作为地理信息系统的基础和核心的形式出现。

空间数据库的研究始于20世纪70年代的地图制图与遥感图像处理领域，其目的是为了有效地利用卫星遥感资源迅速绘制出各种经济专题地图。由于传统数据库在空间数据的表示、存储、管理和检索上存在许多缺陷，从而形成了空间数据库这一新的数据库研究领域，它涉及计算机科学、地理学、地图制图学、摄影测量与遥感、图像处理等多个学科。

空间数据库技术研究的主要内容包括：

（1）空间数据模型

空间数据模型是描述空间实体和空间实体关系的数据模型。一般来说可以用传统的数据模型加以扩充和修改来实现，也可以用面向对象的数据模型来实现。

空间数据库中常用的空间数据结构有矢量数据结构和栅格数据结构两种（如图12.7）。

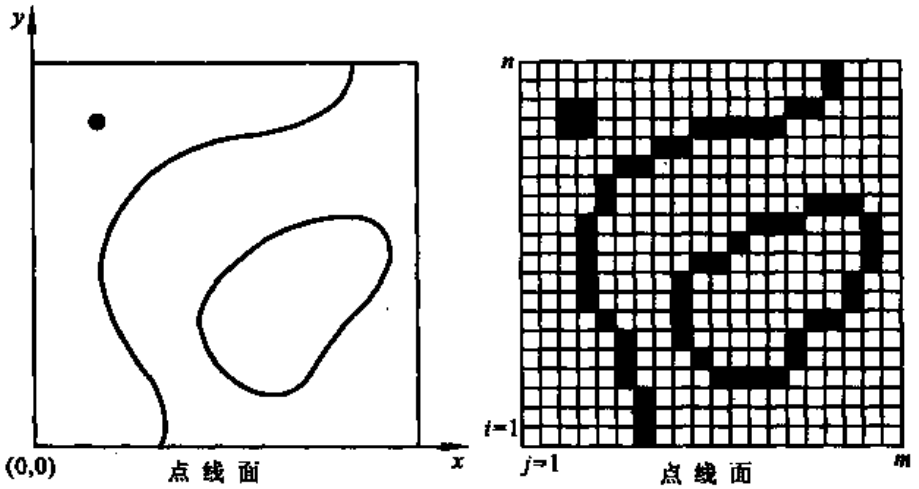


图12.7 空间数据结构

矢量数据结构中，一个区域或一个地图划分为若干个多边形，每个多边形由若干条线段或弧组成。每条线段或弧包含两个结点，结点的位置用X，Y坐标表示。空间关系用点和边，边和面，面和岛之间的关系隐含或显式表示。矢量数据结构数据存储量小，图形精度高。容易定义单个空间对象，但是处理空间关系比较费时，常用于描述图形数据。

栅格数据结构中，地理实体用格网单元的行和列作为位置标识，栅格数据的每个元素（灰度）与地理实体的特征相对应。行和列的数目决定于栅格的分辨率（大小）。栅格数据简单，

容易处理空间位置关系，但数据存储量大，图形精度低。常用于描述图像（和影像）数据。

（2）空间数据查询语言

空间数据查询包括位置查询、空间关系查询和属性查询，前两种查询是空间数据库特有的，基本方式有：

- 面-面查询：例如查询与某一面状物 A 相邻的多边形。
- 线-线查询：例如查询某一河流 A 的所有支流。
- 点-点查询：例如查询两点之间的距离。
- 线-面查询：例如查询铁路 A 穿过的所有城镇。
- 点-线查询：例如查询某一河流 A（线）上的所有桥梁（点）。
- 点-面查询：例如查询某一城市 A 地图中的所有医院（点）。

空间数据查询语言是为了正确表达以上查询请求。

（3）空间数据库管理系统

空间数据库管理系统的主要功能是提供对空间数据和空间关系的定义和描述；提供空间数据查询语言，实现对空间数据的高效查询和操作；提供对空间数据的存储和组织；提供对空间数据的直观显示等。可以想象空间数据库管理系统比传统的数据库管理系统在数据的查询、操作、存储和显示等方面要复杂许多。

目前以空间数据库为核心的地理信息系统的应用已经从解决道路、输电线路等基础设施的规划和管理，发展到更加复杂的领域，地理信息系统已经广泛应用于环境和资源管理、土地利用、城市规划、森林保护、人口调查、交通、商业网络等各个方面的管理与决策。例如，我国已建立了国土资源管理信息系统、黄土高原信息系统、洪水灾情预报和分析系统等。许多公司和研究所已研制了许多地理信息系统 GIS 软件产品。

有关空间数据库的讨论，可参阅《数据库综合大词典》中的第 26 篇及其他文献。

面向特定领域的数据库系统（这里权且称它们为特种数据库系统）还有很多，就不一一介绍了。

人们知道，推动数据库技术前进的原动力是应用需求和硬件平台的发展。正是这些应用需求的提出，特种数据库系统的研究，推动了新一代数据库技术的产生和发展。而新一代数据库技术也首先在这些特种数据库中发挥了作用，得到了应用。

从这些特种数据库系统的实现情况来分析，可以发现它们虽然采用不同的数据模型，但都带有 OO 模型的特征。具体实现时，有的是对关系数据库系统进行扩充，有的则是从头做起。

人们会问，难道不同的应用领域就要研制不同的数据库管理系统吗?能否像第一、二代数据库管理系统那样研制一个通用的能适合各种应用需求的数据

库管理系统呢?这实际上正是第三代数据库系统研究探索的问题，或者说是第三代数据库系统的数据模型即面向对象数据模型研究探索的问题。

人们期望第三代数据库系统能够提供丰富而又灵活的造模能力，强大而又容易剪裁、扩充的系统功能，从而能针对不同应用领域的特点，利用通用的系统模块比较容易地构造出多种多样的特种数据库。

12.3 小　结

本章概要地回顾了数据库技术发展的三个阶段，着重介绍了数据库新技术的主要内容和发展方向。

数据库系统已经发展成为一个大家族，当人们步入数据库领域时，面对众多不同的数据库系统一定会产生迷惑和混乱，本章以数据模型、新技术内容、应用领域为三条主线，阐述了新一代数据库系统及其相互关系，达到纲举目张。

然后，就从这三个方面介绍了数据库技术的进展：

（1）数据库技术发展的核心是数据模型的发展；

（2）传统的数据库技术和其他计算机技术的互相结合、互相渗透是数据库新技术的主要特征；

（3）数据库技术在特定领域的应用使数据库的应用范围不断扩大，从而为数据库技术的发展提供源源不断的动力。

本章以精练的言语，正确的概念，介绍了数据库大家庭中的主要成员，包括分布式数据库系统、并行数据库系统、主动数据库系统、多媒体数据库系统、模糊数据库系统以及面向特定领域的数据仓库系统、工程数据库系统、统计数据库系统和空间数据库系统。

习　题

1. 试述数据库技术的发展过程。
2. 当前数据库技术发展的主要特征是什么？
3. 试述第一、二代数据库系统的主要成就。
4. 第三代数据库系统的主要特点是什么？
5. 试述数据模型在数据库系统发展中的作用和地位。
6. 请用实例阐述数据库技术与其他学科的技术相结合的成果。
7. 请阐述以下数据库系统的主要概念、研究的主要问题及其发展过程：

分布式数据库系统、并行数据库系统、主动数据库系统、多媒体数据库系统、模糊数

据库系统。

8. 试述数据仓库的产生背景。

9. 数据仓库数据的基本特征是什么？

10. 什么是联机分析处理？什么是数据挖掘？

11. 基于数据库技术的D S S解决方案是什么？

12. 什么是工程数据库？

13. 什么是统计数据库？

14. 什么是空间数据库？

本章参考文献

1 Silberschatz Avi，Stonebraker Mike，Ullman Jeff et al. Database Research：Achievements and Opportunities into 21th Century. NSF Workshop Report on the Database System Research in the future，1995，5

（此篇研究报告是美国国家科学基金（NSF）支持的一个研究项目，研讨 21 世纪数据库系统未来的研究方向。参加研讨会的有二十多位著名的数据库专家，如中国数据库界熟悉的 Phil Bernstein，Mike Stonebraker，Jeff Ullman 和 Gio Wiederhold 等。这篇报告内容丰富，对今后数据库系统的研究具有很好的指导意义和参考价值。）

2 李昭原主编. 数据库技术新进展. 北京：清华大学出版社，1997

（本书第一章“数据库技术的发展”（王珊编写）系统地讲解了数据库技术进展的三个阶段，讨论了新一代数据库技术的主要研究内容和发展方向。）

3 王珊等. 数据仓库技术和联机分析处理. 数据库丛书之一. 北京：科学出版社，1998

（本书全面系统地讲解了数据仓库的概念、技术和数据仓库系统建设方法，介绍了以数据仓库为基础的联机分析处技术及其应用实例。）

4 Wang Shan，Lou Wenwu and Feng Yu. PSCARG：A Fast Parallel Clustering Algorithm on Large Datasets，in Proc. of PAKDD'99，Spinger-Verlag，Beijing，April 1999

5 Meng Xiaofeng，Zhou Yong，Wang Shan. Domain Knowledge Extracting in a Chinese Natural Language Interface to Databases：Nchiql. in Proc. of PAKDD'99，Spinger-Verlag，Beijing，April 1999

6 何新贵，唐常杰，李霖等. 特种数据库技术. 数据库丛书之一，北京：科学出版社，1999

（本书主要包括“时态数据库技术”、“移动数据库技术”、“主动数据库技术”和“模糊数据库技术”和“实时数据库技术”五大部分，每部分都讲解了各自领域的基本概念、基本理论和处理技术，以及国内外有关学者们在相应领域中的最新研究成果等。）

7 Shoshani A. Statistical Database：Characteristics，Problems and Some Solutions. in Proc. of VLDB，1982

（本文讨论了统计数据库的问题、研究领域和进展情况。）

8 Shoshani A, Olken F and Wong H K T. Characteristics of Scientific Database. in Proceedings of VLDB, 1984

（本文讨论了科学数据库的问题、研究领域和研究进展情况。）

9 Li Jianzhong(李建中), Rotem D and Wong H K T. A new compression method with fast searching on large database. in Proceedings of VLDB, 1987

10 李建中，孙文隽. 一个统计与科学数据库的语义数据模型. 计算机学报，Vol.14，No. 10，1991

11 李建中，孙文隽，丁华福. 统计与科学数据库上的代数操作. 软件学报，Vol.4，No.2，1993

12 王珊，苏俊等. DKBS/IDKE 的总体设计与实现. 计算机科学，NO.4，1992

（本期《计算机科学》的一组论文，全面介绍了中国人民大学数据与知识工程研究所研制的分布式知识库系统 DKBS/IDKE 的总体设计、知识处理、实现协商策略及其实现。）

13 何守才主编. 数据库综合大词典. 上海：上海科学技术文献出版社，1995

14 徐洁磐，马玉书，范明. 数据库系统导论. 数据库丛书之一. 北京：科学出版社，1999

15 刘启原，刘怡. 知识库与信息系统的安全. 数据库丛书之一. 北京：科学出版社，1999

第十三章　面向对象数据系统

面向对象数据库系统(Object Oriented DataBase System，简称 OODBS)是数据库技术与面向对象程序设计方法相结合的产物。

由于数据库技术在商业领域的巨大成功，使得数据库的应用领域迅速扩展。20 世纪 80 年代以来，出现了大量的新一代数据库应用。在第十二章中已经指出，由于层次、网络和关系数据库系统的设计目标源于商业事务处理，它们面对层出不穷的新一代数据库应用显得力不从心。人们一直在研究支持新一代数据库应用的数据库技术和方法，试图研制和开发新一代数据库管理系统。

面向对象程序设计方法在计算机的各个领域，包括程序设计语言、人工智能、软件工程、信息系统设计以及计算机硬件设计等都产生了深远的影响。自然地，也给遇到挑战的数据库技术带来了机会和希望。人们发现，把面向对象程序设计方法和数据库技术相结合能够有效地支持新一代数据库应用。于是，面向对象数据库系统研究领域应运而生，吸引了相当多的数据库工作者，获得了大量的研究成果，开发了很多面向对象数据库管理系统，包括实验系统和产品。

有关 OO 数据模型和面向对象数据库系统的研究在数据库研究领域是沿着三条路线展开的：

一条是以关系数据库和 SQL 为基础的扩展关系模型。例如美国加州伯克利分校的 POSTGRES 就是以 INGRES 关系数据库系统为基础，扩展了抽象数据类型 ADT(Abstract Data Type)，使之具有面向对象的特性（POSTGRES 后来成为 Illustra 公司的产品，该公司又被 Informix 公司收购）。目前，Informix，DB2，Oracle，Sybase 等关系数据库厂商，都在不同程度上扩展了关系模型，推出了对象关系数据库产品。

一条是以面向对象的程序设计语言为基础，研究持久的程序设计语言，支持 OO 模型，例如美国 Ontologic 公司的 Ontos 是以面向对象程序设计语言 C++ 为基础的；Servialogic 公司的 GemStone 则是以 Smalltalk 为基础的。

一条是建立新的面向对象数据库系统，支持 OO 数据模型，例如法国 O2 Technology 公司的 O2，美国 Itasca System 公司的 Itasca 等。

本章介绍面向对象数据库系统的基本概念。在介绍面向对象数据库系统之前，先简要介绍面向对象程序设计方法。

13.1 面向对象程序设计方法

面向对象是一种新的程序设计方法学，也是一种认知方法学。

面向对象数据库系统源于面向对象的程序设计语言。第一个面向对象程序设计语言是 SIMULA 67。20 世纪 80 年代以来，Smalltalk 和 C++成为被人们普遍接受的面向对象程序设计语言。

面向对象程序设计方法是一种支持模块化设计和软件重用的实际可行的编程方法。它把程序设计的主要活动集中在建立对象和对象之间的联系(或通信)上，从而完成所需要的计算。一个面向对象的程序就是相互联系(或通信)的对象集合。由于现实世界可以抽象为对象和对象联系的集合，所以面向对象的程序设计方法学是一种更接近现实世界的、更自然的程序设计方法学。

面向对象程序设计的基本思想是**封装**和**可扩展性**。

传统的程序设计为**数据结构 + 算法**。面向对象程序设计就是把数据结构和数据结构上的操作算法封装在一个对象之中。对象是以对象名封装的数据结构和可施加在这些数据上的私有操作。对象的数据结构描述了对象的状态，对象的操作是对象的行为。例如，定义一个学生对象，其状态由“学号，姓名，性别，专业号，年龄”等属性值组成，其行为由：“显示学生信息”、“增加一个学生”、“删除一个学生”等操作组成。

面向对象程序设计中，操作名列在封装对象的界面上，当其他对象要启动它的某个操作时，以操作名发一条消息，该对象接收消息，操作动作起来，完成对私有数据的加工。例如，在学生管理应用系统中，以“显示学生”发一条消息，就把相应的代码激活，完成对学生数据的显示。

当一个面向对象的程序运行完毕时，各对象也就达到了各自的终态。输入、输出也由对象自己完成。这种全封装的计算实体给软件带来了模块性、安全性等显著优点。因为它基本没有数据耦合，对象间没有因操作而产生的边界效应，出了错可以很快找到原因，所以易于维护和修改。

面向对象程序设计的**可扩展性**体现在**继承性**和**行为扩展**两个方面。

对象具有一种层次关系。每个对象可以有子对象。子对象可以继承父对象（及其祖先对象）的数据结构和操作，继承的部分就是**重用**的成分。

另一方面，子对象还可以增加新的数据结构和新的操作。子对象新增加的部分就是子对象对父对象发展的部分。

面向对象程序设计的**行为扩展**是指可以方便地增加程序代码来扩展对象的行为而不会影响该对象上的其他操作。

由封装和继承还可以导出面向对象程序设计方法的其他优良特性，如多态性、动态联编等。

面向对象程序设计方法所支持的封装、继承等特性提供了同时表示、同时管理程序和数据的统一框架，数据库研究人员借鉴和吸收了面向对象的方法和技术，提出了面向对象数据模型(简称对象模型)，把面向对象方法和数据库技术结合起来产生了面向对象数据库系统。

13.2 面向对象数据模型

面向对象数据库系统支持面向对象数据模型（以下简称 OO 模型）。

也就是说，一个面向对象数据库系统是一个持久的、可共享的对象库的存储和管理者；而一个对象库是由一个 OO 模型所定义的对象的集合体。

就关系数据库系统而言，E.F.Codd 在其开创性论文中就已给出了关系模型清晰的规范说明。关系数据库系统乃是建立在统一的数据模型基础上的，对关系数据库系统的研究主要集中在实现技术方面而非基本原理方面。

面向对象数据库系统尚不存在这样的规范说明。虽然有很多论文对 OO 模型进行过讨论，但目前仍缺少关于 OO 模型的统一的规范说明，即对 OO 模型缺少一个统一的严格的定义。尽管如此，有关 OO 模型的许多核心概念已取得了高度的共识。

13.2.1 OO 模型的核心概念

一个 OO 模型是用面向对象观点来描述现实世界实体（对象）的逻辑组织、对象间限制、联系等的模型。一系列面向对象核心概念构成了 OO 模型的基础。概括起来，OO 模型的核心概念有如下一些。

1. 对象(Object)与对象标识 OID(Object IDentifier)

现实世界的任一实体都被统一地模型化为一个对象，每个对象有一个唯一的标识，称为对象标识(OID)。

注意，OID 与关系数据库中码(Key)的概念和某些关系系统中支持的记录标识（RID）、元组标识（TID）是有本质区别的。OID 是独立于值的、系统全局唯一的，下面会进一步讲解。

2. 封装(Encapsulation)

每一个对象是其状态与行为的封装，其中状态是该对象一系列属性(Attribute)值的集合，而行为是在对象状态上操作的集合，操作也称为方法

(Method)。

3. 类（Class）

共享同样属性和方法集的所有对象构成了一个**对象类**（简称类），一个对象是某一类的一个**实例**（instance）。例如，学生是一个类，李勇、刘晨、王敏、张立等是学生类中的对象。在数据库系统中，一再要大家注意区分“型”和“值”的概念。在 OODB 中，类是“型”，对象是某一类的一个“值”。类属性的定义域可以是任何类，即可以是基本类，如整数、字符串、布尔型，也可以是包含属性和方法的一般类。特别地，一个类的某一属性的定义也可是这个类自身。

4. 类层次（结构）

在一个面向对象数据库模式中，可以定义一个类（如 C1）的子类（如 C2），类 C1 称为类 C2 的**超类**（或父类）。子类（如 C2）还可以再定义子类（如 C3）。这样，面向对象数据库模式的一组类形成一个有限的层次结构，称为**类层次**。

一个子类可以有多个超类，有的是直接的，有的是间接的。例如，C2 是 C3 的直接超类，C1 是 C3 的间接超类。一个类可以继承类层次中其直接或间接超类的属性和方法。

5. 消息（Message）

由于对象是封装的，对象与外部的通信一般只能通过显式的消息传递，即消息从外部传送给对象，存取和调用对象中的属性和方法，在内部执行所要求的操作，操作的结果仍以消息的形式返回。

下面分别介绍这些概念。

13.2.2 对象与对象标识

一、对象结构

对象是由一组数据结构和在这组数据结构上的操作的程序代码封装起来的基本单位。对象之间的界面由一组消息定义。一个对象包括以下几个部分：

- 属性集合：所有属性合起来构成了对象数据的数据结构（有的书上称为变量集合）。属性描述对象的状态、组成和特性。对象的某一属性可以是单值的或值的集合，进一步地，一个对象的属性也可以是一个对象，即对象可以嵌套。这种嵌套可以继续，从而组成各种复杂对象。
- 方法集合：方法描述了对象的行为特性。方法的定义包括两部分，一是方法的接口，二是方法的实现。方法的接口用以说明方法的名称、参数和结果返回值的类型，也称之为调用说明。方法的实现是一段程序编码，用以实现方法的功能，即对象操作的算法。

• 消息集合：消息是对象向外提供的界面，消息由对象接收和响应。面向对象数据模型中的“消息”与计算机网络中传输的消息的含义不同。它是指对象之间的操作请求的传递，而不考虑操作实现细节。

二、对象标识

面向对象数据库中的每个对象都有一个唯一的不变的标识称为对象标识（OID）。对象通常与实际领域的实体对应。在现实世界中，实体中的属性值可能随着时间的推移会发生改变，但是每个实体的标识始终保持不变。相应的，对象的部分（或全部）属性、对象的方法会随着时间的推移发生变化，但对象标识不会改变。两个对象即使属性值和方法都完全相同，如果 OID 不同则认为是两个不同的对象，它们只是值相等而已。对象标识的概念比程序设计语言或传统数据模型中所用到的标识概念更强。下面是常用的几种标识。

1. **值标识**。用值来表示标识。例如，关系数据库中使用的就是值标识，在关系数据库中，一个关系的元组是用它们的码值来区分的，例如学号 8202110 标识了中国人民大学信息系 82 级学生张雪冰。

2. **名标识**。用一个名字来表示标识。例如，程序变量使用的就是名标识，程序中的每个变量被赋予一个名字，变量名唯一地标识每个变量。

3. **内标识**。以上两种标识是由用户建立的，内标识是建立在数据模型或程序设计语言中，不要求用户给出标识。例如，面向对象数据库系统使用的就是内标识。

不同的标识其持久性程度是不同的，若标识只能在程序或查询的执行期间保持不变，则称该标识具有**程序内持久性**。例如，程序设计语言中的变量名和 SQL 语句中的元组标识符，就是具有程序内持久性的标识。若标识在从一个程序的执行到另一个程序的执行期间能保持不变，则称该标识具有**程序间持久性**。例如，在 SQL 语言中的关系名是具有程序间持久性的标识。若标识不仅在程序执行过程中而且在对数据的重组重构过程中一直保持不变，则称该标识具有**永久持久性**。例如，面向对象数据库系统中对象标识具有永久持久性，而 SQL 语言中的关系名不具有永久持久性，因为数据的重构可能修改关系名。

对象标识具有永久持久性的含义是，一个对象一经产生系统就给它赋于一个在全系统中唯一的对象标识符，直到它被删除。对象标识是由系统统一分配的，用户不能对对象标识符进行修改。对象标识是稳定的，它不会因为对象中某个值的修改而改变。

面向对象的数据库系统在逻辑上和物理上从面向记录（或元组）上升为面向对象、面向可具有复杂结构的一个逻辑整体。它允许用自然的方法，并结合数据抽象机制在结构和行为上对复杂对象建立模型，从而大幅度提高管理效

率，降低用户使用复杂性，并为版本管理、动态模式修改等功能的实现创造了条件。

三、封装

OO 模型的一个关键概念就是封装。每一个对象是其状态与行为的封装。

封装是对象的外部界面与内部实现之间实行清晰隔离的一种抽象，外部与对象的通信只能通过消息，这是 OO 模型的主要特征之一。

封装的意义在于将对象的实现与对象应用互相隔离，从而允许对操作的实现算法和数据结构进行修改，而不影响接口，不必修改使用它们的应用，这有利于提高数据独立性。由于封装，对用户而言这些实现是不可见的，这就隐藏了在实现中使用的数据结构与程序代码等细节。此外对象封装后成为一个自含的单元，对象只接受已定义好的操作，其他程序不能直接访问对象中的属性，从而可以提高程序的可靠性。

但是，对象封装之后查询属性值必须通过调用方法，不能像关系数据库系统那样（用 SQL）进行即席的（随机的）、按内容的查询，这就不够方便灵活，失去了关系数据库的重要优点，因此在 OODB 中必须在对象封装方面作必要的修改或妥协。

13.2.3 类和类层次

一、类的概念

在 OO 数据库中相似对象的集合称为类。每一个对象称为它所在类的一个实例。一个类中的所有对象共享一个定义，它们的区别仅在于属性的取值不同。

可以看到，类的概念类似关系模式，类的属性类似关系模式中的属性；对象类似元组的概念，类的一个实例对象类似关系中的一个元组。

可以把类本身也看作一个对象，称为类对象(Class Object)。

面向对象数据库模式是类的集合。在一个面向对象数据库模式中，会出现多个相似但又有所不同的类。例如，一个有关学校应用的面向对象数据库，其中有教职员工和学生两个类。这两个类都有身份证号，姓名、年龄、性别、住址等属性，也有一些相同的方法和消息。当然，教职员工对象中有一些特殊的属性、方法和消息，如工龄、工资、办公室电话号码、家庭成员数。用户希望统一定义教职员工和学生的公共属性、方法和消息部分，分别定义各自的特殊属性、方法和消息部分。为此，面向对象的数据模型提供了一种类层次结构。类层次结构可以实现上述要求。

二、类的层次结构

在面向对象数据库模式中，一组类可形成一个类层次。一个面向对象数据库模式可能有多个类层次。在一个类层次中，一个类继承其所有超类的全部属性、方法和消息。

回到上面的学校应用的面向对象数据库的例子，可以定义一个类“人”。人的属性、方法和消息的集合是教职员工和学生的公共属性、公共方法和公共消息的集合。教职员工类和学生类定义为人的子类。教职员工类只包含教职员工的特殊属性、特殊方法和特殊消息的集合。学生类也只包含学生的特殊属性、特殊方法和特殊消息集合。图 13.1 给出了学校数据库的一个类层次以及各类对应的属性。

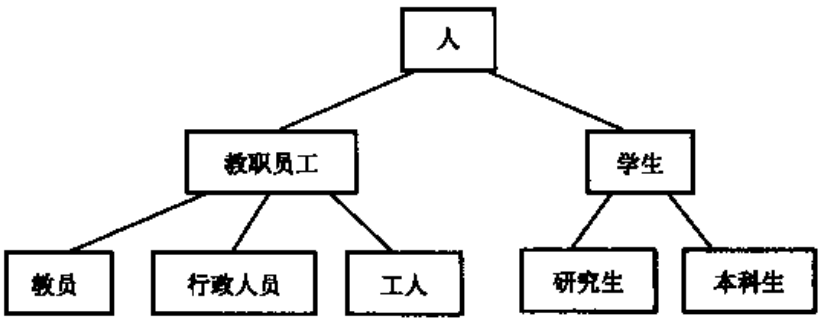


图 13.1 某学校数据库的类层次结构图

其中：

- 人(Person)：身份证号、姓名、年龄、性别、住址
- 教职员工(Employee)：工龄、工资、办公室电话号码、家庭成员数
- 教员 (Teacher)：职称、职务、专长
- 行政人员(Officer)：职务、职责、办公室地址
- 工人(Worker)：工种、级别、所属部门
- 学生 (Student)：入学年份、专业
- 本科生(Undergraduate Student)：已修学分、平均成绩
- 研究生(Graduate Student)：研究方向、导师

为了叙述简单，没有给出这些类的方法。在这个类层次中教职员工和学生是人的子类，教员、行政人员、工人是教职员工的子类。教员、行政人员、工人中实际只有它本身的特殊属性、方法和消息，同时它们又继承教职员工类和人的所有属性、方法和消息。因此，逻辑上它们具有人、教职员工和本身的所有属性、方法和消息。同样，本科生和研究生是学生的子类。

超类/子类之间的关系体现了“ISA”的语义，例如，图 13.1 中教员“ISA”教职员工（即“教员 is a 教职员工”），教职员工“ISA”人。因此，超类是

子类的抽象（Generalization）或普通化，子类是超类的特殊化（Specialization）或具体化。

类层次可以动态扩展，一个新的子类能从一个或多个已有类导出。根据一个类能否继承多个超类的特性将继承分成了单继承和多重继承。

13.2.4 继承

在 OO 模型中常用的有两种继承，**单继承与多重继承**。若一个子类只能继承一个超类的特性（包括属性、方法和消息），这种继承称为单继承；若一个子类能继承多个超类的特性，这种继承称为多重继承。例如，在学校中实际上还有“在职研究生”，他们既是教员又是学生，在职研究生继承了教职员工和学生两个超类的所有属性、方法和消息，如图 13.2 所示。

单继承的层次结构图是一棵树（如图 13.1），多继承的层次结构图是一个带根的有向无回路图（如图 13.2）。

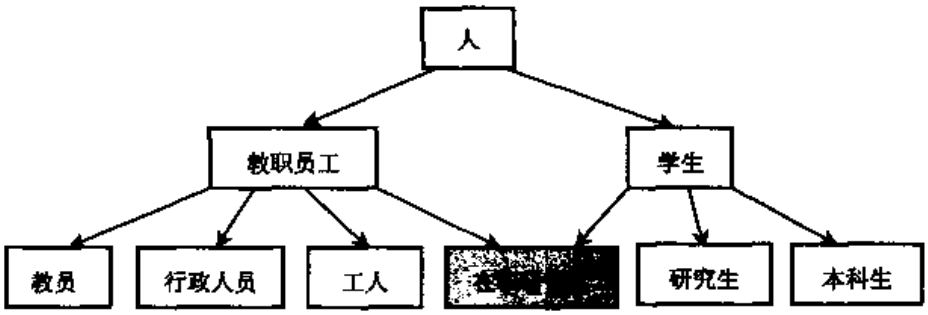


图 13.2 具有多继承的类层次结构图

继承性有两个优点，第一，它是建模的有力工具，提供了对现实世界简明而精确的描述。第二，它提供了信息重用机制。由于子类可以继承超类的特性，这就可以避免许多重复定义。当然子类除了继承超类的特性外还要定义自己特殊的属性、方法和消息。在定义这些特殊属性、方法和消息时可能与继承下来的超类的属性、方法和消息发生冲突，例如在教职员工类中已经定义了一个操作“打印”，在教员子类又要定义一个操作“打印”，用来打印教员的姓名、年龄、性别、职称和专长，这就产生了同名冲突。这类冲突可能发生在子类与超类之间，也可能发生在子类的多个直接超类之间。这类冲突通常由系统解决，在不同的系统中使用不同的冲突解决方法，便产生了不同的继承性语义。例如对于子类与超类之间的同名冲突，一般是以子类定义的为准，即子类的定义取代或替换由超类继承而来的定义；对于子类的多个直接超类之间的同名冲突，有的系统是在子类中规定超类的优先次序，首先继承优先级最高的超类的定义；有的系统则指定继承其中某一个超类的定义。

子类对父类即有继承又有发展，继承的部分就是重用的成分。由封装和继承还导出面向对象的其他优良特性，如多态性、动态联编等。

13.2.5 滞后联编

在 OO 模型中，当子类定义的方法与继承下来的超类的方法发生同名冲突，即子类只继承了超类中操作的名称而自己实现操作的算法，有自己的数据结构和程序代码。这样，同一个操作名就与不同的实现方法，与不同的参数相联系。例如，图 13.2 的例子中定义的“打印”操作，在不同的类中有不同的打印格式和打印参数。

一般地，在 OO 模型中对于同一个操作，可以按照类的不同，重新定义操作的实现，这称为操作的重载（Overriding）。这就导致一个名字表示不同的程序，这称为过载（Overloading ）。例如，要显示（Display）三个不同的对象——人、图片、图形，这里要用不同的显示机制。对于人，要按某种格式显示一个人的有关信息如身份证号、姓名、年龄、性别、住址等；对于图片要显示的是一幅图像；对于图形则要显示的是它的形状。

在传统的程序设计中，要编写三个操作 display-person，display-bitmap，display-graph，在应用程序中，程序员要判断对象的类型，再调用相应的操作。

```
begin
case of  type(x)
person: display-person(x);
bitmap: display-bitmap(x);
graph: display-graph(x);
end
```

这就要求程序员必须知道应用中所有的对象类型、知道相应的显示操作才能编写上面的应用程序。

在一个 OODB 系统中对人、图片、图形三个不同的对象可以用一个 display 操作名，针对不同对象编写不同的实现算法，让系统来选择。

虽然系统设计人员仍要为不同的类编写不同的 display 程序，但是对于应用程序员来讲就不必为选择这些操作费心了，也没有上面关于选择的 case 语句，应用程序的代码也简单了。

为了提供这个功能，OODBMS 不能在编译时就把操作名联编到程序上，必须在运行时根据实际请求中的对象类型和操作选择相应的程序，把操作名与它联编上（即把操作名转换成该程序的地址），这个推迟的转换称为滞后联编（Late Binding）。

13.2.6 对象的嵌套

前面已经讲到，在一个面向对象数据库模式中，对象的某一属性可以是单值的或值的集合。进一步地，一个对象的属性也可以是一个对象，这样对象之间产生一个**嵌套层次结构**。

设 Obj_1 和 Obj_2 是两个对象。如果 Obj_2 是 Obj_1 的某个属性的值，称 Obj_2 属于 Obj_1，或 Obj_1 包含 Obj_2。一般的，如果对象 Obj_1 包含对象 Obj_2，则称 Obj1 为**复杂对象或复合对象**。

Obj_2 是 Obj_1 的组成成分，也可称是 Obj_1 **子对象**。Obj_2 还可以包含对象 Obj_3，这样 Obj_2 也是复杂对象，从而形成一个嵌套层次结构。

例如，每辆汽车包括：汽车型号、汽车名称、发动机、车体、车轮、内部设备等属性。其中汽车型号和汽车名称的数据类型是字符串，发动机不是一个标准数据类型，而是一个对象，包括：发动机型号、马力等属性；车体也是一个对象，包括：钢板厚度、钢板型号、车体形状等属性；内部设备也是一个对象，包括：车座、音响设备、安全设备等属性；音响设备也可以是一个对象，包括：VCD、喇叭等属性，如图 13.3 所示。

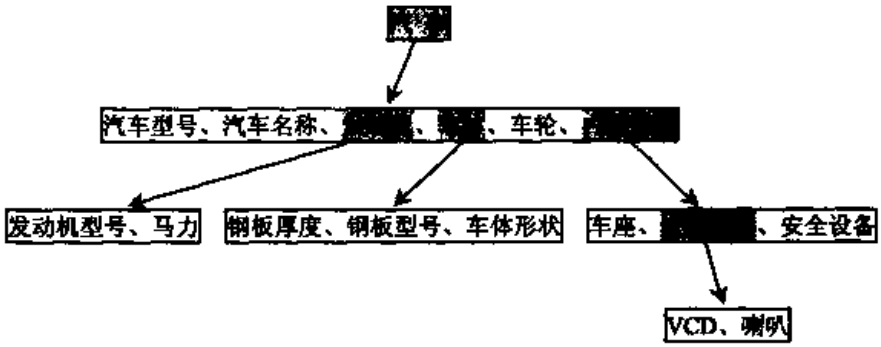


图 13.3 汽车的嵌套层次图

对象嵌套概念是面向对象数据库系统中又一个重要概念，它允许不同的用户采用不同的粒度来观察对象。一个发动机设计者一般只把注意力集中在发动机对象上，顾客在选购买汽车时，则关心汽车的实例。

对象嵌套层次结构和类层次结构形成了对象横向和纵向的复杂结构。这是说，不仅各种类之间具有层次结构，而且某一个类内部也具有嵌套层次结构，它不再像一个关系模式那样是平面结构的了，一个类的属性可以是一个基本类，也可以是一个一般类。

13.3 面向对象数据库语言

在 OODB 中，与对象模型密切相关的是面向对象数据库语言（OODB 语言）。OODB 语言用于描述面向对象数据库模式，说明并操纵类定义与对象实例。OODB 语言主要包括对象定义语言（ODL）和对象操纵语言（OML），对象操纵语言中一个重要子集是对象查询语言（OQL）。

OODB 语言一般应具备下列功能。

（1）类的定义与操纵

面向对象数据库语言可以操纵类，包括定义、生成、存取、修改与撤销类。其中类的定义包括定义类的属性、操作特征、继承性与约束等。

（2）操作/方法的定义

面向对象数据库语言可用于对象操作/方法的定义与实现。在操作实现中，语言的命令可用于操作对象的局部数据结构。对象模型中的封装性允许操作/方法由不同程序设计语言来实现，并且隐藏不同程序设计语言实现的事实。

（3）对象的操纵

面向对象数据库语言可以用于操纵（即生成、存取、修改与删除）实例对象。

数据库系统从网状模型到关系模型的进步使数据库查询语言从用户导航式的过程性语言进入到了由系统自动选择查询路径的非过程性语言。但是非过程性语言的面向集合的操作方式又与高级程序设计语言的面向单个数据的操作方式之间产生了不协调现象，俗称阻抗失配。阻抗失配的根本原因在于关系数据库的数据模型和程序设计语言不一致。因而，对嵌入式数据库语言来说不可避免地会产生阻抗失配。

但 OODB 不同，它的数据模型的概念来自面向对象的程序设计方法 OOP，因此作为某一面向对象的程序设计语言（OOPL）扩充的 OODB 语言，有望解决数据库系统中的阻抗失配问题。

但 OODB 语言又不同于 OOPL，它是对数据库操作的语言，要提供对数据库的操作功能。而 OOPL 的查询功能很弱，这是因为 OOPL 的查询采用的是导航方式。如定义图书为类，作者为属性。一般的 OOP 可从每一本书（图书类的对象实例）找到它的所有作者，但不支持另一方向的查询，即从作者查询他的所有作品，除非把作者也定义成一个类而把作品设置成属性。但这样的设计将在数据库中造成冗余。

此外，OOPL 要求所有对象之间的相互通信都通过发送消息来实现。这种

要求严重地限制了数据库应用。例如“查找作者为萨师煊的所有著作”，按照OOPL 规范，就要向图书类的每个实例发送一个消息，检查它的作者。如果按照关系数据库来处理这个查询，可以在图书类上建立一个索引，索引字段为属性“作者”，这样就可以通过索引快速地查到“萨师煊的所有著作”。

为此，面向对象数据库语言的研制是 OODB 系统开发中的重要部分，人们试图扩充 OOPL 的查询语言或者扩充 SQL 的功能。

目前，还没有像 SQL 那样的关于面向对象数据库语言的标准，因此不同的 OODBMS 其具体的数据库语言各不相同。

13.4 面向对象数据库的模式演进

面向对象数据库模式是类的集合。模式为适应需求的变化而随时间的变化称为模式演进。模式演进包括创建新的类、删除旧的类、修改类的属性和操作等。在关系数据库系统中，模式的修改比较简单，主要有如下的模式修改操作：

- 创建或删除一个关系
- 在关系模式中增加或删除一个属性
- 在关系模式中修改完整性约束条件

OODB 应用环境对 OODB 模式演进提出了许多新的要求，使得面向对象数据库模式的修改要比关系模式的修改复杂得多，其主要原因是：

（1）模式改变频繁

使用 OODB 系统的应用通常需要频繁地改变 OODB 数据库模式。例如 OODB 经常运用于工程设计环境中，设计环境特征之一就是不断变化。设计自身在不断变化，以纠正错误或修改设计使之更完美、更适合于实际；而当设计者对问题及其解决有更深刻理解时也会修改模式。

（2）模式修改复杂

从上面讲解的 OO 模型特征可以看到 OO 模型具有很强的建模能力和丰富的语义，包括类本身的语义、类属性之间和类之间丰富的语义联系，这使得模式修改操作的类型复杂多样。此外，OODB 中模式演进往往是动态的，动态模式演进的实现技术更加复杂。

13.4.1 模式的一致性

模式的演进必须要保持模式的一致性。模式一致性是指模式自身内部不能出现矛盾和错误，它由模式一致性约束来刻画。模式一致性约束可分为唯一性

约束、存在性约束和子类型约束等，满足所有这些一致性约束的模式则称为是一致的。

（1）唯一性约束

这一类约束条件要求名字的唯一性，例如：

- 在同一模式中所有类的名字必须唯一。
- 类中属性名和方法名必须唯一，包括从超类中继承的属性和方法。

模式的不同种类的成分可以同名，例如属性的名字和方法可以同名。

（2）存在性约束

存在性约束是指显式引用的某些成分必须存在，例如：

- 每一个被引用的类必须在模式中定义。这种引用包括出现在属性说明、变量说明、变量定义等之中。
- 某操作代码中调用的操作必须给出说明。
- 对每一说明的操作必须存在一个实现程序。

（3）子类型约束

- 子类/超类联系不能有环。
- 不能有从多继承带来的任何冲突。
- 如果只支持单继承，则子类的单一超类必须加以标明。

13.4.2 模式演进操作

下面给出一些主要的模式演进操作，OODBMS 应该支持这些模式演进。

（1）类集的改变

- 增加一个新的类
- 删除一个已有类
- 改变一个已有类名字

（2）已有类的成分的改变

- 增加新的属性或新的操作/方法
- 删除已有的属性或操作
- 改变已有属性的名字或类型
- 改变一操作的名称或操作的实现

（3）子类/超类联系的改变

- 增加一新的超类
- 删除一已有超类

13.4.3 模式演进的实现

模式演进主要的困难是模式演进操作可能影响模式一致性。面向对象数据库中类集的改变比关系数据库中关系模式的改变要复杂得多。例如：增加一新的类可能违背类名唯一的约束。为了保持一致性可能要做大量的一致性验证工作和修改工作；新增加的类必须放到相关类层次图中，如果新增加的类不是类层次图中的叶结点，新增加类的所有后裔子类需要继承新类的属性和方法，因此要检查是否存在继承冲突问题。

又如删除一个类，在面向对象数据库中删除一个类要执行多个操作。被删除类的所有子类继承的变量和方法必须被重新检查，撤销从被删除类继承的变量和方法。被删除类的实例或者改为其他类的实例，如改为被删除类的超类的实例，或者与类一起被删除。

与上面的情况类似，类的修改操作可能会影响到其他类的定义。例如，改变一个类的属性名，这需要所有使用该属性的地方都要改名。

因此，在 OODB 模式演进的实现中必须具有模式一致性验证功能，这部分的功能类似编译器的语义分析。

进一步，任何一个面向对象数据库模式修改操作不仅要改变有关类的定义，而且要修改相关类的所有对象，使之与修改后的类定义一致。

例如，要在一个类中增加一个属性，简单地保持一致性的方法是首先删除该类所有的实例，然后修改类结构。这对于已花费了大量人力物力保存起来的有价值的信息来说，这种做法显然是不可取的。一般采用转换的机制来实现模式演进。

所谓转换方法是指在 OO 数据库中，已有的对象将根据新的模式结构进行转换以适应新的模式。例如，在某类中增加一属性时，所有的实例都将增加该属性。这时还要处理新属性的初值，例如给定一缺省初值，或提供一算法来自动计算新属性初值，还可以让用户设定初值。删除某类中一属性时只需从该类的所有实例中删除相应属性值即可。

根据转换发生的时间可有不同的转换方式：

- 立即转换方式。一旦模式变化立即执行所有变换；
- 延迟转换方式。模式变化后不立即执行，延迟到低层数据库载入时，或者延迟到该对象被存取时才执行变换。

前者的缺点在于系统为了执行转换将停顿一些时间；后者的缺点在于以后应用程序存取一个对象时，要把它的结构与其所属类的定义比较，完成必须的修改，运行效率将受到影响。

实际系统中，GemStone 采用立即转换方式；Orion 则采用延迟转换方式，

开始只执行逻辑变换，使用该对象时才变换成新的定义。

此外，通过多模式版本方式来完成转换也是一种有力的手段。当修改面向对象的数据库模式时，建立一个数据库模式新版本，保留旧版本，不废弃原数据库模式。这样，系统中同时存在多个数据库模式版本。这对于历史数据库的存取非常有利。但是，这种方法无疑导致存储空间开销的增大。

如何实现面向对象数据库模式的演进是面向对象数据库系统研究的一个重要方向。

13.5 对象-关系数据库

本章开始讲到 OO 数据模型和面向对象数据库系统的研究在数据库研究领域是沿着三条路线展开的。其中很重要的一条是研制对象-关系数据库系统。

1990 年以 Michael Stonebraker 为首的高级 DBMS 功能委员会发表了“第三代数据库系统宣言”的文章，按照文章的思想，一个面向对象数据库系统必须满足两个条件：

- 支持一核心的面向对象数据模型；
- 支持传统数据库系统所有的数据库特征。

也就是说 OODBS 必须保持第二代数据库系统的非过程化数据存取方式和数据独立性，即应继承第二代数据库系统已有的技术，不仅能很好地支持对象管理和规则管理，而且能更好地支持原有的数据管理。

对象-关系数据库系统就是按照这样的目标将关系数据库系统与面向对象数据库系统两方面的特征相结合。

对象-关系数据库系统除了具有原来关系数据库的各种特点外，还应该提供以下特点：

（1）扩充数据类型

目前的商品化 RDBMS 只支持某一固定的类型集，不能依据某一应用所需的特定数据类型来扩展其类型集。对象-关系数据库系统允许用户在关系数据库系统中扩充数据类型，即允许用户根据应用需求自己定义数据类型、函数和操作符。例如，某些应用涉及三维向量，系统就允许用户定义一个新的数据类型三维向量 vector，它包含三个实数分量。而且一经定义，这些新的数据类型、函数和操作符将存放在数据库管理系统核心中，可供所有用户共享，如同基本数据类型一样。例如可以定义数组、向量、矩阵、集合等数据类型以及这些数据类型上的操作。

（2）支持复杂对象

能够在 SQL 中支持复杂对象。复杂对象是指由多种基本数据类型或用户自定义的数据类型构成的对象。

（3）支持继承的概念

能够支持子类、超类的概念，支持继承的概念，包括属性数据的继承和函数及过程的继承；支持单继承与多重继承；支持函数重载（操作的重载）。

（4）提供通用的规则系统

能够提供强大而通用的规则系统。规则在 DBMS 及其应用中是十分重要的，在传统的 RDBMS 中用触发器来保证数据库数据的完整性。触发器可以看成规则的一种形式。对象-关系数据库系统要支持的规则系统将更加通用，更加灵活，并且与其他的对象-关系能力是集成为一体的，例如规则中的事件和动作可以是任意的ＳＱＬ语句，可以使用用户自定义的函数，规则能够被继承等。这就大大增强了对象-关系数据库的功能，使之具有主动数据库和知识库的特性。

实现对象-关系数据库系统的方法主要有以下五类。

1. 从头开发对象-关系 DBMS。这种方法费时费力，一般不采用。

2. 在现有的关系型 DBMS 基础上进行扩展。扩展方法有两种：

（1）对关系型 DBMS 核心进行扩充，逐渐增加对象特性。这是一种比较安全的方法，新系统的性能往往也比较好。目前许多著名的关系数据库系统厂商都采用这种方法。他们推出的最新版本都已声称是对象-关系数据库系统。

（2）不修改现有的关系型 DBMS 核心，而是在现有关系型 DBMS 外面加上一个包装层，由包装层提供对象-关系型应用编程接口，并负责将用户提交的对象-关系型查询映象成关系型查询，送给内层的关系型 DBMS 处理。这种方法系统效率会因包装层的存在受到影响。

3. 将现有的关系型 DBMS 与其他厂商的对象-关系型 DBMS 连接在一起，使现有的关系型 DBMS 直接而迅速地具有了对象-关系特征。连接方法主要有两种：

（1）关系型 DBMS 使用网关技术与其他厂商的对象-关系型 DBMS 连接。但网关这一中介手段会使系统效率降低。

（2）将对象-关系型引擎与关系型存储管理器结合起来，即以关系型 DBMS 作为系统的最底层，具有兼容的存储管理器的对象-关系型系统作为上层。

4. 将现有的面向对象型 DBMS 与其他厂商的对象-关系型 DBMS 连接在一起，使现有的面向对象型 DBMS 直接而迅速地具有了对象-关系特征。连接方法是将面向对象型 DBMS 引擎与持久语言系统结合起来。即以面向对象的 DBMS 作为系统的最底层，具有兼容的持久语言系统的对象-关系型系统作为上层。

5. 扩充现有的面向对象的 DBMS，使之成为对象-关系型 DBMS。

以上五类实现方法中最主要和最有效的是对现有的关系型 DBMS 核心进行扩展，增加对象特性，使之逐渐成为对象-关系数据库系统。

13.6 小 结

本章介绍面向对象数据库系统的基本概念和一般特征。面向对象数据库是数据库技术与面向对象技术相结合的产物。

面向对象数据库的产生和发展受应用需求的直接驱动，其目标在于满足不断增长的新应用需求。

OODB 的研究实际上是沿着不同的方向、采用不同的方法进行的。因此在 20 世纪 80 年代中后期，关于 OO 概念、OODB 概念、OODB 数据模型和语言等一系列基本概念、术语的定义和理解呈现了百花齐放、百家争鸣的情景。经过反复的探索，OODB 专家们现在已对 OODB 中的核心概念、OODBMS 的基本目标取得了共识。本章讲解了这些基本概念。

人们认为 OODBS 将成为下一代数据库系统的典型代表，但是，OODBS 在奠定其新一代数据库系统地位之前要清除两个主要障碍：标准化和性能。

关系数据库的成功不仅在于其简单的数据模型和高度的数据独立性，还在于其标准化。标准化对 OODBMS 实际上更为重要，因为 OODBMS 涉及范围远远超出 RDBMS，应用更加复杂，为了使应用实际可行，为了使应用可以在不同环境间移植和互操作，以及为了使用户容易学习新的 OODBS，标准尤其重要。即将颁布的 SQL99(俗称 SQL3)增加了 OODB 的许多特征，会对 OODB 的标准化有所贡献。

在 OODBS 中为了获得满意的性能，还需要在辅助索引、物理聚集、并发控制、查询优化、对象目录管理等方面不断改进技术。

习 题

1. 面向对象程序设计的基本思想是什么？它的主要特点是什么？
2. 定义并解释 OO 模型中以下核心概念：对象与对象标识、封装、类、类层次。
3. OO 模型中对象标识与关系模型中的“码”有什么区别？
4. 请给出一个 OO 数据库类层次（包括各类对应的属性）实例。
5. 举例说明超类和子类的概念。

6. 什么是单继承？什么是多重继承？继承性有什么优点？

7. 什么操作的重载？在 OODB 中为什么要滞后联编？

8. 什么是 OODB 模式演进？为什么面向对象数据库模式的修改要比关系模式的修改复杂得多？

9. 什么是对象-关系数据库？它的主要特点是什么？常用的实现方法有哪些？

本章参考文献

1 Goldberg A，Robson D. Smalltalk-80：The Language and Its Implementation. Addison-Wesley，1983

（本文以及[2]讨论了面向对象的程序设计方法学。）

2 Cox B J. Object-Oriented Programming: An Evolutionary Approach. Addison-Wesley，1986

3 Dittrich K(Editor). Advances in Object-Oriented Database Systems. Lecture Notes in Computer Science 334，Springer-Verlag，1988

（本文献综述了面向对象数据库系统各方面的研究进展情况和主要研究成果。）

4 Kim W，Lochovsky F(Editors). Object-Oriented Concepts，Databases and Applications. Addison-Wesley，1989

（本书讲解了面向对象数据库系统的基本概念、技术方法和应用。）

5 Kim W. Introduction to Object-Oriented Databases. MIT Press，Cambridge，1990

（本书讲解了面向对象数据库系统的基本概念和实现技术。）

6 Zdonik S，Maier D. Readings in Object-Oriented Database Systems. Morgan Kaufmann Publishers，Inc.，1990

（本书讲解了面向对象数据库系统的基本概念和技术方法。）

7 王珊. 面向对象的数据库系统. 计算机世界专题综述，1990，1

8 王珊等. An Object-Oriented Model for GIS，第三届泛太平洋计算机国际会议 PPCC-3 论文集，1989，8

9 王珊. 面向对象数据库系统的回顾和现状. 计算机世界，1992，12

10 王珊，唐元昌，任永杰，武志文等. OBMS/IDKE 的数据模型和无缝的编程语言. 计算机科学，No.2，1992

（本期的一组论文，全面介绍了中国人民大学数据与知识工程研究所研制的对象管理系统 OBMS/IDKE 的对象数据模型、无缝的编程语言、对象存储管理和动态模式管理。）

11 Stonbraker M，Rowe L. The Design of POSTGERS. Proceedings of the ACM SIGMOD，1986

（本文介绍了 POSTGRES 系统，是众多介绍 POSTGRES 系统的论文之一。）

12 Kim W. Architectural Issues in Object-Oriented Databases. Journal of Object-Oriented

Programming, March 1990

（本文介绍了 Orion 系统的各个方面，是众多介绍 Orion 系统的论文之一。）

13 Lecluse C, Richard P, Velez F. O_2: An Object-Oriented Data Model. Proceedings of the ACM SIGMOD, 1988

（本文介绍了 O_2 系统，介绍 O_2 系统的论文很多，这只是其中之一。）

14 Maier D, Stein J, et al. Development of an Object-Oriented DBMS. Proceedings of the Inter. Conference on Object-Oriented Programming Systems, Languages and Applications, 1986

（本文介绍了 Gemstone 系统，介绍 Gemstone 系统的论文很多，这只是其中之一。）

15 Fishman D, Beech D et al. IRIS: An Object-Oriented Database Management System. in Readings in Oriented-Oriented Database Systems, 1990

（本文介绍了 Iris 系统，介绍 Iris 系统的论文很多，这只是其中之一。）

16 Carey M L, DeWitt D et al. The EXODUS Extensible DBMS Project: An Overview, in Readings in Oriented-Oriented Database Systems. Morgan Kaufmann Publishers, 1990

（本文介绍了 EXODUS 系统，介绍 EXODUS 系统的论文很多，这只是其中之一。）

17 Hudson S E, King R. Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System. ACM TODS, Volume 14, Number 3, September 1989

（本文介绍了 Cactis 系统）

18 Abiteboul S, Kanellakis P. Object Identity as a Query Language Primitive. Proceedings of the ACM SIGMOD, 1989

（本文详细讨论了对象标识问题。）

19 Penney J, Stein J. Class Modification in the GemStone Object-Oriented DBMS. Proceedings of the International Conference on Object-Oriented Programming Systems, Languages and Applications, 1987

（本文研究了面向对象数据库模式修改的问题，这些方法已经用于 Orion 和 Gemstone 系统。）

20 Skarra A, Zdonik S. Concurrency Control in Object_Oriented Databases. in Objected-Oriented Concepts, Databases and Applications, Addison-Wesley, 1989

（文献[20]讨论了面向对象数据库系统的并发控制问题。）

21 Maier D, Stein J. Indexing in an Object-Oriented DBMS. Proceedings of the International Workshop on Object-Oriented Database Systems, 1986

（本文研究了面向对象数据库系统的查询处理问题。）

22 QSQL: A Language for Migrating from SQL to Object Databases. Proceedings of the International Conference on Extending Database Technology, 1988

（本文研究了面向对象数据库系统的查询语言。）

23 Stonbraker M M，Moore D. Object-Relational DBMSs：The Next Great Ware，Morgan Kaufmann Publisher，Inc. 1996

（本书作者是是美国著名的数据库专家。本书已由杨冬青、唐世渭、裴芳等译校，《对象-关系数据库管理系统——下一个浪潮》，北京大学出版社出版，1997）

24 Malcolm Atkinson et al. The Object-Oriented Database System Manifesto. Proc. First International Conference on Deductive and Object-Oriented Databases. Kyoto，Japan 1989

25 Malcolm Atkinson et al. The Object-Oriented Database System Manifesto. Proc. First International Conference on Deductive and Object-Oriented Databases. New York，N.Y.：Elsevier Science，1990

（这篇文章试图为面向对象数据库系统（以及面向对象数据模型）作出定义。它描述了作为一个合格的面向对象数据库系统所应具备的主要特征和性质。这些特性被分为三类：

• 必备的：所有被称作面向对象的数据库系统所必须满足的特性。包括：复杂对象、对象标识、封装性、类型和类、继承性、结合滞后联编的重载、可扩充性、计算完备性、持久性、辅存管理、并发性、恢复和即席查询功能。

• 可选的：可以改进系统但并非必需的特性。包括：多重继承性、类型检查和推理、分布、设计事务处理（长事务或嵌套事务）、以及版本控制。

• 开放的：设计人员可选择的特性。包括：程序设计风格、表示系统、类型系统以及一致性。）

26 中国人民大学数据与知识工程研究所编译：X3/SPARC/DBSSG/OODBTG 最终报告. 计算机工程与应用，No.6，1992

（本期是“OODBTG 最终报告”的专辑。1989 年 1 月，在美国 ANSI 所属 ASC/X3/SPARC 的数据库系统研究组（DBSSG）下，成立了面向对象数据库任务组（OODBTG），专门就对象数据管理（简称 ODM）为面向对象数据库管理系统（OODBMS）的标准化问题进行研究。

该组织不是制定标准的技术委员会，而是研究如何实现新标准的工作组。OODBTG 的目标是对下面的问题进行调查、研究和取得共识：

(1)提出能被大家接受的对象数据库的可行定义。

(2)确定对象数据库技术与相关领域中面向对象技术之间的联系。这些相关领域包括程序设计语言、用户界面、方法学、信息建模方法等。

(3)为将来在对象信息管理领域内的标准化工作建立一个框架。

该任务组于 1991 年 8 月发表了“OODBTG 最终报告”。该报告在大量调查研究的基础上，阐述了许多新的技术概念，提出了对象数据管理（ODM）参考模型和实现对象信息管理标准化的建议。上述工作对以后面向对象数据库系统的研究和系统开发产生了深远影响。

*第十四章　分布式数据库系统

20 世纪 70 年代以来，由于计算机网络通信的迅速发展，以及地理上分散的公司、团体和组织对数据库更为广泛应用的需求，在集中式数据库系统成熟技术的基础上产生和发展了分布式数据库系统（Distributed Database System）。分布式数据库系统是数据库技术和网络技术两者相互渗透和有机结合的结果。经过 20 多年的发展，分布式数据库已经发展得相当成熟，推出了很多实用化系统，从早期的 INGRES/STAR（原 INGRES 公司），SQL*STAR（ORACLE 公司）到现在的 SQL REPLICATION SERVER（SYBASE 公司）等。

这一章介绍分布式数据库系统的基本概念，包括什么是分布式数据库，为什么要分布式数据库，分布式数据库的特点（14.1 节）；分布式数据库的模式结构，分布透明的层次，分布式数据库管理系统（14.2 节）；查询处理和优化（14.3 节）；分布事务管理（恢复和并发控制，14.4 节）；发展前景和应用趋势（14.5 节）。

由于分布式数据库系统比集中式数据库系统更复杂，在有限的篇幅中只能论述最基本的概念和技术，作为今后进一步学习或研究分布式数据库系统的基础。

14.1　概　　述

这一节介绍分布式数据库系统的定义和特点。

14.1.1　什么是分布式数据库系统

什么样的一个数据库系统才算是分布式数据库系统呢？一个粗略的定义是：

“分布式数据库由一组数据组成，这些数据物理上分布在计算机网络的不同结点（亦称场地或站点）上，逻辑上是属于同一个系统”。

这个定义强调了下面两点：

（1）分布性。数据库中的数据不是存储在同一场地（更确切地讲，不存储在同一计算机的存储设备）上，这就可以和集中式数据库相区别。

（2）逻辑整体性。这些数据逻辑上是互相联系的，是一个整体（逻辑上如同集中数据库）。这就可以和分散在计算机网络不同结点上的那些集中式数据库或文件的集合相区别。后者各结点的数据之间没有内在的逻辑联系。所以在讨论分布式数据库时就有了全局数据库（逻辑上）和局部数据库（物理上）的概念。

但是，这个定义仍然是不精确的，下面来分析两个例子。

例 1　如图 14.1 所示，这个系统中有三台服务器，每台服务器有自己的数据库系统，三台服务器之间通过网络（局域网或广域网）相连，每台服务器有自己的若干客户机。用户可以通过客户机对本地服务器中的数据库执行某些应用（称之为局部应用），也可以通过客户机对两个或两个以上结点中的数据库执行某些应用（称之为全局应用或分布应用）。这样的系统是分布式数据库系统，而不支持全局应用的系统不能称为分布式数据库系统。一个典型的全局应用的例子是银行转账。这个应用要求从一个分行的账户（设在 DB1 数据库）中转让若干金额到另一个分行的账户（设在 DB3 数据库）中去，因此要同时更新两个结点上的数据库。

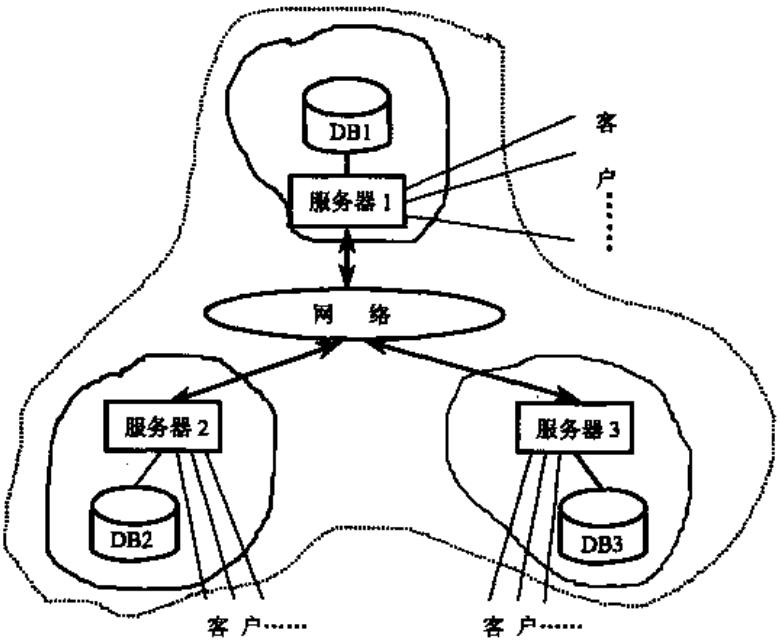


图 14.1　一个分布式数据库系统

这个例子说明，区分一个系统是若干集中式数据库的简单连网还是分布式数据库系统的技术要点在于，系统是否支持全局应用。所谓全局应用是指应用中涉及两个或两个以上结点的数据库。它和只存取本结点数据库的应用不同，和只存取另一个结点（或远程结点）数据库的应用也是不同的。支持全局应用

的技术要复杂困难得多。在图 14.1 中用实线示意局部数据库及局部应用，用虚线示意全局数据库及全局应用。

例 2 如图 14.2 所示，这个系统的配置和图 14.1 的不同点是：三个数据库通过后台服务器和网络相连。服务器执行数据库管理功能。所有的应用都由客户机处理。尽管发出这些应用的客户可以分布在不同的场地，数据库也物理地分布在各个不同的服务器上，但这样的系统不是分布式数据库系统而是一个多处理机系统（SN 并行结构）（请参见本书第十五章）。其原因是没有局部应用，即每个后台服务器不能由它本身来执行自己的局部应用，数据的物理分布不是对应用观点而言的分布，而分布式数据库不仅要求数据的物理分布，而且要求这种分布是面向处理、面向应用的。

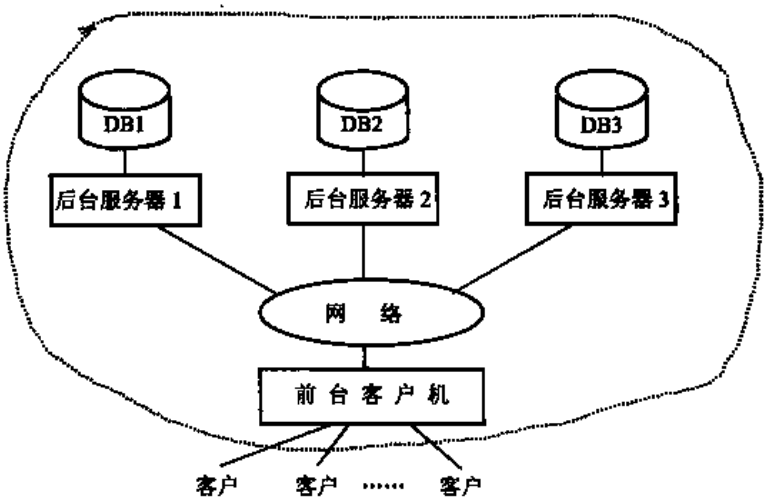


图 14.2 一个多处理机系统（SN 并行结构）

总结上面两个例子，可以得出分布数据库的一个更确切的定义：

分布式数据库是由一组数据组成的，这组数据分布在计算机网络的不同计算机上，网络中的每个结点具有独立处理的能力（称为场地自治），可以执行局部应用。同时，每个结点也能通过网络通信子系统执行全局应用。

这个定义比前一个定义更强调了场地自治性以及自治场地之间的协作性。这就是说，每个场地是独立的数据库系统，它有自己的数据库、自己的用户、自己的 CPU，运行自己的 DBMS，执行局部应用，具有高度的自治性。同时各个场地的数据库系统又相互协作组成一个整体。这种整体性的含义是，对于用户来说，一个分布式数据库系统逻辑上看如同一个集中式数据库系统一样，用户可以在任何一个场地执行全局应用。

14.1.2 分布式数据库系统的特点

分布式数据库系统是在集中式数据库系统技术的基础上发展起来的，但不是简单地把集中式数据库分散地实现，它是具有自己的性质和特征的系统。集中式数据库的许多概念和技术，如数据独立性、数据共享和减少冗余度、并发控制、完整性、安全性和恢复等，在分布式数据库系统中都有了不同的更加丰富的内容。

一、数据独立性

数据独立性是数据库方法追求的主要目标之一。在集中式数据库系统中，数据独立性包括两个方面：数据的逻辑独立性与数据的物理独立性。其含义是用户程序与数据的全局逻辑结构及数据的存储结构无关。

在分布式数据库系统中，数据独立性这一特性更加重要，并具有更多的内容。除了数据的逻辑独立性与物理独立性外，还有数据分布独立性亦称分布透明性（Distribution Transparency）。分布透明性指用户不必关心数据的逻辑分片，不必关心数据物理位置分布的细节，也不必关心重复副本（冗余数据）一致性问题，同时也不必关心局部场地上数据库支持哪种数据模型。分布透明性也可以归入物理独立性的范围。

分布透明性的优点是很明显的。有了分布透明性，用户的应用程序书写起来就如同数据没有分布一样。当数据从一个场地移到另一个场地时不必改写应用程序。当增加某些数据的重复副本时也不必改写应用程序。数据分布的信息由系统存储在数据字典中。用户对非本地数据的访问请求由系统根据数据字典予以解释、转换和传送。

在集中式数据库系统中，数据独立性是通过系统的三级模式（外模式、模式、内模式）和它们之间的二级映象得到的。在分布式数据库系统中，分布透明性则是由于引入了新的模式和模式间的映象得到的。这将在 14.2.1 节中详细讨论。

二、集中与自治相结合的控制结构

数据库是多个用户共享的资源。在集中式数据库系统中，为了保证数据库的安全性和完整性，对共享数据库的控制是集中的，并设有 DBA 负责监督和维护系统的正常运行。

在分布式数据库系统中，数据的共享有两个层次：

（1）局部共享。即在局部数据库中存储局部场地上各用户的共享数据，

这些数据是本场地用户常用的。

（2）全局共享。即在分布式数据库系统的各个场地也存储供其他场地的用户共享的数据，支持系统的全局应用。

因此，相应的控制机构也具有两个层次：集中和自治。分布式数据库系统常常采用集中和自治相结合的控制机构。各局部的 DBMS 可以独立地管理局部数据库，具有自治的功能。同时，系统又设有集中控制机制，协调各局部DBMS的工作，执行全局应用。当然，不同的系统，集中和自治的程度不尽相同。有些系统高度自治，连全局应用事务的协调也由局部 DBMS、局部 DBA 共同承担，而不要集中控制，不设全局 DBA。有些系统则集中控制程度较高，场地自治功能较弱。

三、适当增加数据冗余度

在集中式数据库系统中，尽量减少冗余度是系统目标之一。其原因是，冗余数据不仅浪费存储空间，而且容易造成各数据副本之间的不一致性，为了保证数据的一致性，系统要付出一定的维护代价。减少冗余度的目标是用数据共享来达到的。

而在分布式数据库系统中却希望增加冗余数据，在不同的场地存储同一数据的多个副本，其原因是：

1. 提高系统的可靠性、可用性

当某一场地出现故障时，系统可以对另一场地上的相同副本进行操作，不会因一处故障而造成整个系统的瘫痪。

2. 提高系统性能

系统可以选择用户最近的数据副本进行操作，减少通信代价，改善整个系统的性能。

但是，数据冗余同样会带来和集中式数据库系统中一样的问题。不过，冗余数据增加存储空间的问题随着硬件磁盘价格的下降得到解决。而冗余副本之间数据不一致的问题是分布式数据库系统必须着力解决的问题。

一般地讲，增加数据冗余度方便了检索，提高了系统的查询速度、可用性和可靠性，但不利于更新，增加了系统维护的代价。因此应在这些方面作出权衡，进行优化。

四、全局的一致性、可串行性和可恢复性

分布式数据库系统中各局部数据库应满足集中式数据库的一致性、并发事务的可串行性和可恢复性。除此以外还应保证数据库的全局一致性、全局并发事务的可串行性和系统的全局可恢复性。这是因为在分布式数据库系统中全局

应用要涉及两个以上结点的数据，全局事务可能由不同场地上的多个操作组成。例如 14.1.1 节的例 1 中银行转账事务包括两个结点上的更新操作。这样，当其中某一个结点出现故障，操作失败后如何使全局事务回滚呢？如何使另一个结点撤销（UNDO）已执行的操作（若操作已完成或完成一部分）或者不必再执行事务的其他操作（若操作尚未执行）？这些技术要比集中式数据库系统复杂和困难得多，分布式数据库系统必须解决这些问题。

14.1.3 分布式数据库系统的目标

分布式数据库系统的目标，也就是研制分布式数据库系统的目的和动机，主要包括技术和组织两方面的目标。许多论文和专著主要讨论技术目标。而研制分布式系统的主要动机应该说是组织目标，是由应用驱动的。

1. 适应部门分布的组织结构，降低费用

使用数据库的单位在组织上常常是分布的（如分为部门、科室、车间等），在地理上也是分布的。分布式数据库系统的结构符合部门分布的组织结构，允许各个部门对自己常用的数据存储在本地，在本地录入、查询、维护，实行局部控制。由于计算机资源靠近用户，因而可以降低通信代价，提高响应速度，使这些部门使用数据库更方便更经济。

2. 提高系统的可靠性和可用性

改善系统的可靠性和可用性是分布式数据库系统的主要目标。将数据分布于多个场地，并增加适当的冗余度可以提供更好的可靠性。对于那些可靠性要求较高的系统，这一点尤其重要。一个场地出了故障不会引起整个系统崩溃，因为故障场地的用户可以通过其他场地进入系统，而其他场地的用户可以由系统自动选择存取路径，避开故障场地，利用其他数据副本执行操作，不影响事务的正常执行（例如 SYBASE REPLICATION SERVER）。

3. 充分利用数据库资源，提高现有集中式数据库的利用率

当在一个大企业或大部门中已建成若干个数据库之后，为了利用相互的资源，为了开发全局应用，就要研制分布式数据库系统。这种情况可称为自底向上的建立分布式系统。这种方法虽然也要对各现存的局部数据库系统作某些改动、重构，但比起把这些数据库集中起来重建一个集中式数据库，则无论从经济还是从组织上考虑，分布式数据库均是较好的选择。

4. 逐步扩展处理能力和系统规模

当一个单位扩大规模，要增加新的部门（如银行增加新的分行，工厂增加新的科室、车间）时，分布式数据库系统的结构为扩展系统的处理能力提供了较好的途径，即在分布式数据库系统中增加一个新的结点。这样比在集中式系统中扩大系统规模要方便、灵活、经济得多。在集中式系统中为了扩大规模常

用的方法有两种：一种是在开始设计时留有较大的余地，这容易造成浪费，而且由于预测困难，设计结果仍可能不适应情况的变化。另一种方法是系统升级，这会影响现有应用的正常运行。并且当升级涉及不兼容的硬件或系统软件有了重大修改而要相应地修改已开发的应用软件时，升级的代价就十分昂贵而常常使得升级的方法不可行。分布式使数据库系统能方便地将一个新的结点纳入系统，不影响现有系统的结构和系统的正常运行，提供了逐渐扩展系统能力的较好途径，有时甚至是唯一的途径。

14.2 分布式数据库系统的体系结构

这一节讨论满足 14.1 节中给出的分布式数据库系统特点的系统结构（包括模式结构和分布式数据库管理系统的结构）。同时进一步阐述分布式数据库系统的基本概念。

14.2.1 分布式数据库系统的模式结构

图 14.3 是分布式数据库系统一种模式结构的示意图，实际的系统并非都具有这种结构。在这个结构中各级模式的层次清晰，可以概括和说明分布式数据库系统的概念和结构。

图 14.3 的模式结构从整体上可以分为两大部分：下部是集中式数据库系统的模式结构，代表了各局部场地上局部数据库系统的基本结构；上部是分布式数据库系统增加的模式级别，其中包括：

（1）全局外模式（Global External Schema），它们是全局应用的用户视图，是全局概念模式的子集。

（2）全局概念模式（Global Conceptual Schema），它定义分布式数据库中数据的整体逻辑结构，使得数据如同没有分布一样。全局概念模式中所用的数据模型应该易于向其他模式映象，通常采用关系模型，因此也选用关系模型。这样，全局概念模式包括一组全局关系的定义。

（3）分片模式（Fragmentation Schema），每一个全局关系可以分为若干不相交的部分，每一部分称为一个片段（Fragment）。分片模式定义片段以及全局关系到片段的映象。这种映象是一对多的，一个全局关系可对应多个片段，而一个片段只来自一个全局关系。

（4）分布模式（Allocation Schema），片段是全局关系的逻辑部分，一个片段在物理上可以分配到网络的不同结点上。分布模式定义片段的存放结

点。分布模式的映象类型确定了分布式数据库是冗余的还是非冗余的。若映象是一对多的，即一个片段可分配到多个结点上存放，则是冗余的分布数据库，若映象是一对一的，则是非冗余的分布数据库。

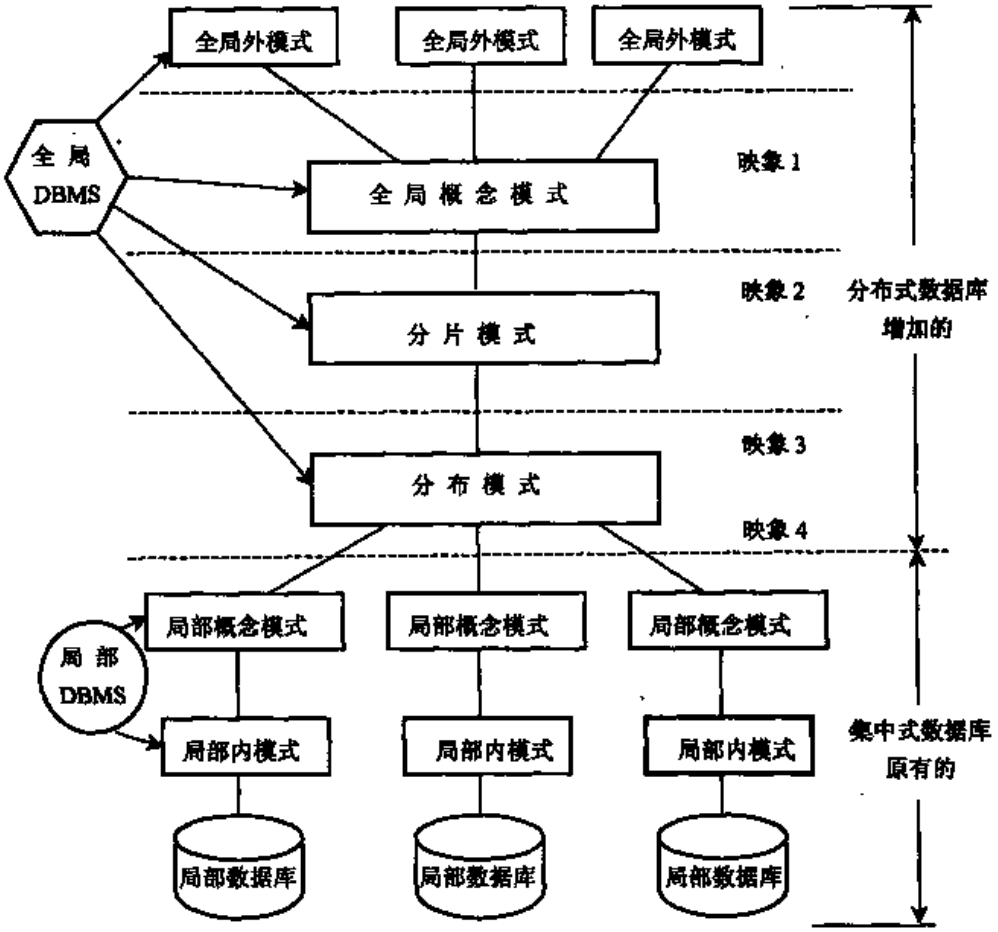


图 14.3 分布式数据库系统的模式结构

根据分布模式提供的信息，一个全局查询可分解为若干子查询，每一子查询要访问的数据属于同一场地的局部数据库。

由分布模式到各局部数据库的映象（映象 4）把存储在局部场地的全局关系或全局关系的片段映象为各局部概念模式（Local Conceptual Schema），局部概念模式采用局部场地的 DBMS 所支持的数据模型。

分片模式和分布模式均是全局的，分布式数据库系统中增加的这些模式和相应的映象使分布式数据库系统具有了分布透明性。

14.2.2 数据分片

将数据分片，使数据存放的单位不是关系而是片段，这既有利于按照用户

的需求较好地组织数据的分布，也有利于控制数据的冗余度。

分片的方式有多种，水平分片和垂直分片是两种基本的分片方式，混合分片和导出分片是较复杂的分片方式。

水平分片是指按一定的条件将关系按行（水平方向）分为若干不相交的子集，每个子集为关系的一个片段。

垂直分片是指将关系按列（垂直方向）分为若干子集。垂直分片的诸片段必须能够重构原来的全局关系，即可以用连接的方法恢复原关系，因此垂直分片的诸片段通常都包含关系的码。

导出分片是指导出水平分片，即水平分片的条件不是本身属性的条件而是其他关系的属性的条件。例如学生选课关系 SC（Sno，Cno，Grade），若不是按照学号或课程号或成绩的某类条件分片，而是按照学生年龄＞18 岁和≤18 岁来分片。（学生年龄是学生关系 Student 的属性）则这类分片称导出分片，可以用 SQL 语句来表达这两个片段。

年龄＞18 岁的学生选课片段 SC_A（Sno，Cno，Grade）由下面的查询结果组成：

```
SELECT  Sno，Cno，Grade
FROM  S，SC
WHERE  S.Sno=SC.Sno  AND  S.Sage＞18；
```

年龄≤18 岁的片段 SC_B 由下面的查询结果组成：

```
SELECT  Sno，Cno，Grade
FROM  S，SC
WHERE  S.Sno=SC.Sno  AND  S.Sage≤18；
```

混合分片是指按上述三种分片方式得到的片段继续按另一种方式分片。例如先按水平分片得到的某一片段再进行垂直分片（如图 14.4（a）所示），或者按垂直分片得到的某一片段又按水平分片（如图 14.4（b）所示）方式继续分为若干片段。

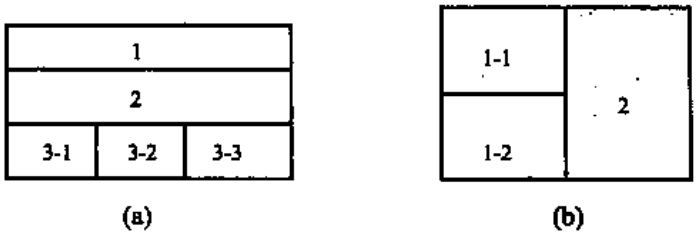


图 14.4 混合分片

无论哪种分片方式都应满足以下条件：

（1）完备性。一个全局关系中的数据必须完全地划分为若干片段，不允许某些数据属于全局关系但不属于任何一个片段。

（2）不相交性。不允许一个全局关系的某些数据既属于该全局关系的某一个片段又属于该全局关系的另一个片段（垂直分片中的主码属性除外）。

（3）可重构性。可以由片段重构全局关系，对于垂直分片可以用连接操作重构全局关系。例如，R1，R2，R3 为全局关系 R 的垂直分片，R 的码属性 A，则

$$R = R_1 \underset{R_1.A=R_2.A}{\bowtie} R_2 \underset{R_2.A=R_3.A}{\bowtie} R_3$$

对于水平分片，可以用并操作重构全局关系，例如，SC_A，SC_B 为 SC 的两个水平分片，则

$$SC = SC\_A \cup SC\_B$$

14.2.3 分布透明性

从图 14.3 的模式结构可以看到分布透明性包括：分片透明性、位置透明性和局部数据模型透明性。

分片透明性是分布透明性的最高层次。所谓分片透明性是指用户或应用程序只对全局关系进行操作而不必考虑关系的分片。当分片模式改变了，由于全局模式到分片模式的映象（映象 2），全局模式不变，应用程序不必改写，这就是分片透明性。

位置透明是分布透明的下一层次。所谓位置透明性是指，用户或应用程序不必了解片段的存储场地，当存储场地改变了，由于分片模式到分布模式的映象（映象 3），应用程序不必改变。同时，若片段的重复副本数目改变了，数据的冗余度改变了，用户也不必关心如何保持各副本的一致性，这就是重复副本的透明性。

局部数据模型透明性是指用户或用户程序不必了解局部场地上使用的是哪种数据模型，模型的转换以及数据库语言的转换均由映象 4 完成。

下面讨论一个简单的应用，用它来说明对于不同层次上的透明性，应如何编写应用程序。可以发现，透明性层次越高，应用程序的编写越简单。

例 设在分布式数据库系统中有全局关系 Student（Sno，Sname，Sdept，Sage），Student 关系被划分为两个片段 S_A 和 S_B。S_A 代表理学院的学生，S_B 代表文学院的学生。S_A 存储在场地 1（Site1），S_B 冗余地存储在场地 2 和场地 3 上（如图 14.5 所示）。

这里有一个查询，从终端读入一个学号，查找该学号的学生姓名、年龄，并把它们显示在屏幕上。

设应用程序是用嵌入 SQL 语句的 C 语言写的。这里略去了程序细节，仅给出查询部分的算法思想。

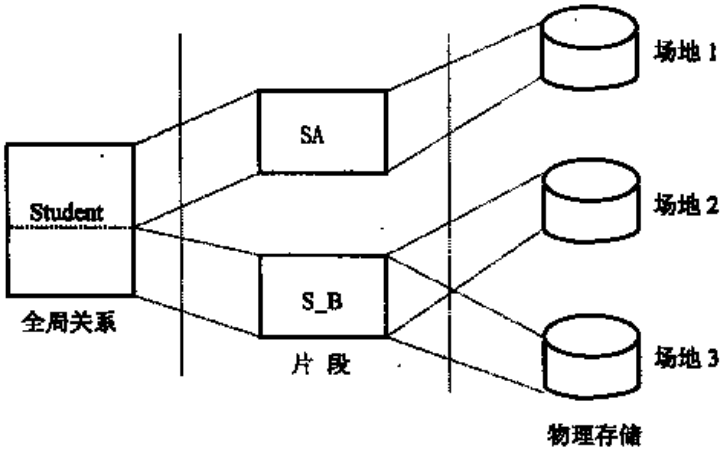


图 14.5 全局关系、片段及物理存储的关系

情况 1 若系统具有分片透明性，则

```
Scanf（" %s"，Snumber）；          /* 从终端读入学号到变量 Snumber 中 */
EXEC SQL SELECT Sname，Sage INTO ：NAME，：AGE
                                   /* NAME，AGE 为程序变量 */
FROM  Student                      /* 在全局关系 Student 中查找 */
WHERE Sno =：Snumber；
Printf（" %s，%d"，NAME，AGE）；  /* 把 NAME，AGE 打印在屏幕上 */
```

情况 2 若系统具有位置透明性，但不具有分片透明性，则

```
Scanf（" %s"，Snumber）；
EXEC SQL SELECT Sname，Sage INTO ：NAME，：AGE
FROM S_A                           /* 先在片段 S_A 中查找 */
WHERE Sno =：Snumber；
If（! FOUND）｛
EXEC SQL SELECT Sname，Sage INTO ：NAME，：AGE
FROM S_B                           /* 再在片段 S_B 中查找 */
WHERE Sno =：Snumber；
｝
Printf（" %s，%d"，NAME，AGE）；
```

情况 3 若系统只具有局部数据模型透明性，不具有位置透明性（当然也就不具有分片透明性），则

```
Scanf（" %s"，Snumber）；
EXEC SQL SELECT Sname，Sage INTO ：NAME，：AGE
        FROM S_A AT Site1          /* 先在场地 1 的片段 S_A 中查找 */
        WHERE Sno =：Snumber；
```

```
If (! FOUND) {
EXEC SQL SELECT Sname, Sage INTO :NAME, :AGE
        FROM S_B AT Site2      /* 再在场地 2 的片段 S_B 中查找 */
        WHERE Sno =: Snumber ; /* 也可以在场地 3 的片段 S_B 中查找 */
}
Printf ("%s, %d", NAME, AGE);
```

因为 S_B 重复存放在场地 2 和场地 3，所以用户程序可以对场地 2，或场地 3 访问，用户可选择离他较近的某个场地进行访问，这样可以提高查询效率。

从这里看到，若没有分片透明性，用户必须了解分片的情况。若没有位置透明性，场地的选择由用户程序负责，用户不仅要了解分片的情况，还必须了解片段存放的位置，场地分布的远近，以及通信线路的质量。若系统具有位置透明性，则这些任务均由系统承担，并且当访问场地 2 时，若该线路繁忙或有了通信故障，则系统可以自动地改为访问场地 3，而这种优化工作用户程序是做不到的。

14.2.4 分布式数据库管理系统

分布式数据库管理系统（Distributed DataBase Management System，简称 D-DBMS）是建立、管理和维护分布式数据库的一组软件。这里给出一种 D-DBMS 的结构，分析它的主要成分和功能（如图 14.6 所示）。

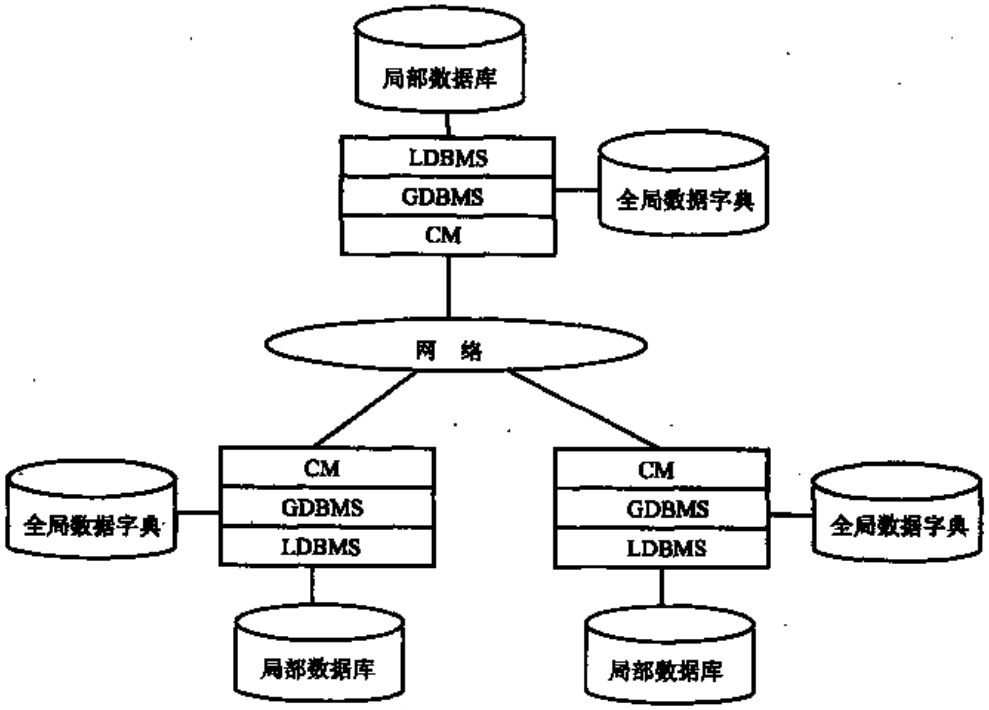


图 14.6 分布式数据库管理系统的结构

D-DBMS 由四部分组成：

（1）LDBMS（Local DBMS），局部场地上的数据库管理系统，其功能是建立和管理局部数据库，提供场地自治能力，执行局部应用及全局查询的子查询。

（2）GDBMS（Global DBMS），全局数据库管理系统，主要功能是提供分布透明性（如图 14.3 所示），协调全局事务的执行，协调各局部 DBMS 以完成全局应用，保证数据库的全局一致性，执行并发控制，实现更新同步，提供全局恢复功能等。

（3）全局数据字典（Global Data Directory，简称 GDD），存放全局概念模式、分片模式、分布模式的定义以及各模式之间映象的定义，存放有关用户存取权限的定义，以保证全用户的合法权限和数据库的安全性，存放数据完整性约束条件的定义，其功能与集中式数据库的数据字典类似。

（4）通信管理（Communication Management，简称 CM），通信管理系统在分布数据库各场地之间传送消息和数据，完成通信功能。

D-DBMS 功能的分割和重复以及不同的配置策略就导致了各种不同的体系结构。

一、按全局控制方式分类

1. 全局控制集中的 D-DBMS

这种结构的特点是全局控制成分 GDBMS 集中在某一结点上，由该结点完成全局事务的协调和局部数据库转换等一切控制功能。全局数据字典只有一个，也存放在该结点上，它是 GDBMS 执行控制的主要依据。

这种结构的优点是控制简单，容易实现更新一致性，但由于控制集中在某一特定的结点上，不仅容易形成瓶颈而且系统较脆弱，一旦该结点出故障，整个系统就将瘫痪。

2. 全局控制分散的 D-DBMS

这种结构的特点是全局控制成分 GDBMS 分散在网络的每一个结点上，全局数据字典也在每个结点上存放一份。每个结点都能完成全局事务的协调和局部数据库转换的控制功能，每个结点既是全局事务的参与者又是全局事务的协调者。图 14.6 就是这种体系结构。一般称这类结构为完全分布的 D-DBMS。

它的优点是结点独立，自治性强，单个结点退出或进入系统均不会影响整个系统的运行，但是全局控制的协调机制和一致性的维护都比较复杂。

3. 全局控制部分分散的 D-DBMS

这种结构是根据应用的需要将 GDBMS 和全局数据字典分散在某些结点上，是介于前两种情况的体系结构。

二、按局部 DBMS 的类型分类

区分不同 D-DBMS 的一个重要性质是：局部 DBMS 是同构的还是异构的。

同构和异构的级别可以有三级：硬件、操作系统和局部 DBMS。其中最主要的是局部 DBMS 这一级，因为硬件和操作系统的不同将由通信软件处理和管理。所以，定义同构型 D-DBMS 为：每个结点的局部数据库具有相同的 DBMS，如都是 Oracle 关系数据库管理系统，即使操作系统和计算机硬件并不相同；异构型 D-DBMS 为：各结点的局部数据库具有不同的 DBMS，如有的是 Oracle，有的是 Sybase，有的是 IMS 层次数据库管理系统。

异构型 D-DBMS 的设计和实现比同构型 D-DBMS 更加复杂。因为各结点的局部数据库可能采用不同的数据模型（层次、网状或关系），或者虽然模型相同但它们是不同厂商的 DBMS（如 DB2，Oracle，Sybase，Informix），它要解决不同的 DBMS 之间以及不同的数据模型之间的转换，要解决异构数据模型的同种化问题。这是研制异构型 D-DBMS 的关键问题，所谓同种化就是寻找合适的公共数据模型，采用公共数据模型与异构（局部）数据模型之间的转换，而不采用各结点之间的一对一转换。这样可以减少转换次数。设有 N 个结点，用公共数据模型时转换次数为 $2N$，而各结点之间一对一转换则需 $N(N-1)$ 次。

现在的分布式数据库系统产品大都提供了集成异构数据库的功能，以 Sybase Replication Server 为例，任何数据存储系统只要遵循基本的数据操作和事务处理规范，都可以充当局部数据库管理系统。Sybase Replication Server 提供开放接口以支持异构数据库，它包括以下四个功能组件：

（1）Sybase 客户/服务器接口（C/SI），负责完成 Replication Server 与各异构数据库服务器间的通信。如果局部数据服务器不支持 C/SI，系统管理人员还可以为该服务器创建 Open Server Gateway，完成通信任务。

（2）Replication Agent 或 Log Transfer Manager（LTM），作为 Replication Server 在各局部数据库服务器的代理，监控局部数据操作。

（3）错误处理，统一处理各局部数据库服务器返回的错误。

（4）Functions，Funcition strings 和 Function string classes，负责完成各数据库服务器间数据模型和数据操作的转换。

14.3 查询处理和优化

分布式数据库系统中的查询处理较集中式数据库系统复杂，查询优化较集

中数据库系统更重要，效果更显著。下面首先讨论一个实例，说明在分布环境下选择一个好的处理查询策略是多么重要，并了解查询处理和优化涉及的问题，然后讨论查询优化的一般目标和策略。

14.3.1 一个实例

下面是一个很有说服力的例子，说明存取策略优化的重要性。

- 数据库，简化了的供应商和零件数据库：

  S（Sno，City） 10^4个元组，存放在场地 A；

  P（Pno，Color） 10^5个元组，存放在场地 B；

  SP（Sno，Pno） 16^6个元组，存放在场地 A；

  设每个关系的元组均为 100 字节长。

- 查询：求供应红色零件的、北京的供应商号：

```
SELECT  S.Sno
FROM  S，P，SP
WHERE  S.City ='北京'  AND
       SP.Pno = P.Pno  AND
       P.Color ='红色'
```

- 估算值（某些中间结果的元组数）：

  红色零件数=10

  北京供应商的装运单数=10^5

- 对通信系统的假定

  数据传输速度=10^4 字节/秒

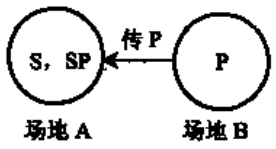
  传输延迟=1 秒

现在，考虑下面 6 种可能的查询存取策略，对各种策略 i，根据下面的公式分别计算通信时间 T[i]：

T[i]＝总传输延迟+总数据量 / 数据传输速度

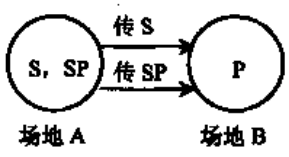
策略 1 把关系 P 传送到场地 A，在 A 地进行查询处理，所以，

T[1]=1+10^5×100/10^4 =10^3秒(16.7 分)

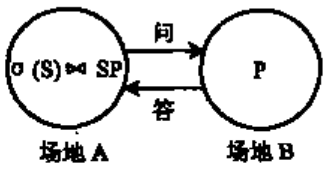


策略 2 把关系 S，SP 传到场地 B，在 B 地执行查询处理，所以

T[2]=2+(10^4+10^6)×100/10^4≈10 100 秒(2.8 小时)



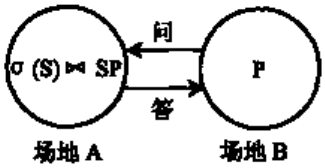
策略 3 在场地 A 站连接关系 S 和 SP，选出城市为北京的元组（10^5 个）然后对这些元组中的每个元组的 Pno，询问场地 B 看



此零件是否红色。所以共问答 10^5 次，由于不是传送数据，只是消息的问答，所以

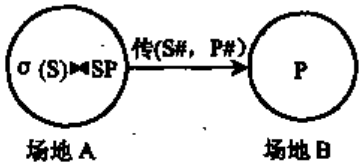
T[3]=2×10^5s(2.3 天)

策略 4 在 B 站选出红色零件的元组(10 个)，然后对每一个元组逐一检查 A 站，看北京供应商的装运单中是否有这个零件装运单（若有则选出 S#）每做这样一次检查包括 2 次消息，共问一答 10 次，所以



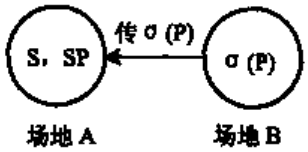
T[4]=2×10=20 秒

策略 5 在 A 站选出北京的供应商的装运单把结果送到 B 站，在 B 站完成最后处理，所以



T[5]=1+(10^5×100)/10^4≈1 000 秒(16.7 分)

策略 6 在 B 站的关系 P 中选出红色的元组（10 个），把结果送到 A 站完成最终处理。所以



T[6]=1+(10×100)/10^4≈1 秒

表 14-1 综合了上述结果。

表 14-1 分布环境下查询策略实例比较

策 略	通信时间	方 法
1	16.7 分	把 P 传到 A 站
2	2.8 小时	把 S，SP 传到 B 站
3	2.3 天	对每一北京的装运单，检查相应零件是否红色
4	20 秒	对每一红色零件，检查北京供应商中是否有人供应
5	16.7 分	把北京供应商的装运单传送到 B
6	1 秒	把红色零件传送到 A

上面的时间只是查询处理中的通信时间。查询处理时间还应包括在某一场地上的处理时间。这和集中数据库系统中查询处理时间计算是一样的，主要包括 I/O 时间和 CPU 时间。由于在分布数据库中通信时间是最主要的开销，这里仅仅为说明问题就只比较了查询处理中的通信时间。注意，在这个例子中，忽略了发出查询的地点，即最后结果传送到哪个站的问题。在实际系统中还应加上传送结果的开销。

这个例子说明了：

（1）不同的存取策略通信时间相差很大，达多个数量级！因此必须进行优化。

（2）在有些策略中数据传输速度和传输延迟都要考虑；而在有些策略中（如例子中策略 3、策略 4）主要考虑传输延迟，还有一些策略中（如例子中策略 1、策略 2、策略 5）数据传输量大，主要考虑传输时间。

此外，有些策略还允许在两个场地上并行处理。例如策略 6，可以在 A 场地完成对北京供应商装运单选择、连接的同时，在 B 场地选出红色零件的元组，因此总的查询时间还可能比集中数据库系统中所花费的要少。

14.3.2 查询处理和优化要解决的问题

绝大多数分布式数据库系统都是关系型的，由于关系查询的语义级别较高，为查询优化提供了可能。系统执行查询可以有多种策略，而且彼此之间性能会有很大差别。分布式 DBMS 极其复杂，系统开销非常可观，为了提供能够接受的效率，查询优化是十分必要的。

在分布式数据库系统中有三类查询：局部查询、远程查询和全局查询。局部查询和远程查询都只涉及单个结点上的数据（本地的或远程的），所以查询优化采用的技术就是集中式数据库的查询优化技术（代数优化和非代数优化）。全局查询涉及多个结点的数据，因此查询处理和优化要复杂得多，下面要讨论的是全局查询处理和优化涉及的问题，优化的目标及连接查询的优化方法。

为了执行全局查询和确定一个好的查询处理策略，要做许多判断、计算工作，但总体上可分为三类。

1. 查询分解

对于全局查询必须把它们分为若干子查询，每个查询只涉及某一结点的数据，可以由局部 DBMS 处理。在分布数据库中一个关系可分为若干逻辑片段，这些片段又可以在系统的多个结点上存放。所以，对一个查询中所涉及的关系需要确定一个物理片段，选择不同的物理片段执行查询操作会直接影响查询执行的效率。因此必须选择查询开销最省的那些物理片段。

2. 选择操作执行的次序

主要是确定连接和并操作的次序，其他的操作顺序是不难确定的，例如选择和投影操作总是应尽量提前执行。这个原则和集中数据库中代数优化的策略相同。但是，涉及不同结点上关系的连接和并操作的次序是必须认真考虑的。

3. 选择执行操作的方法

这包括将若干操作组合在对数据库的一次存取中执行；选择可用的存取路径（如索引）以及选择某一种算法等问题。连接（Join）是查询中最费时的操作，因此连接的执行方法是研究的重点。在分布式数据库中提出了半连接（Semi-Join）的方法。

显然，这三个问题不是独立的。为了找到一个全局查询的最优分解，确定执行查询的物理片段，必须了解查询中各操作执行的次序，而这又依赖于对每个操作的执行方法。而且，这三个问题之间的联系不是单向的或线性的，而是互相制约的。不过，这三个问题中关键是第 2 个问题。因为第 1、第 3 个问题可将“使应用局部性最大化”和“使应用的并发性最大化”作为准则，而第 2 个问题则需要大量计算比较，需要关于数据库的环境信息（如关系的大小，连接属性上的值的分布，通信线路的有关指标等）才能解决。

14.3.3 查询优化的目标

无论是在集中式数据库中还是在分布式数据库中一个查询处理策略的选择都是以执行查询的预期代价为依据的。不同的只是构成一个查询代价的主要因素在这两类系统中不完全一样。

在集中式数据库中，查询执行开销主要是：

I / O 代价 + CPU 代价

而在分布式数据库中，除上面两种开销外还有数据在网络上的传输代价：

I / O 代价 + CPU 代价 + 通信代价

对于分布式数据库中数据传输代价应区别不同情况考虑。在远程通信网或数据传输率较低的系统中，通信代价可能会比查询执行中的 I / O 及 CPU 开销大得多，因而作为首要的优化目标来考虑。在局域网且传输率高的系统中，通信代价和局部处理的开销差不多，此时就应在优化中将它们平等对待。

但是，在查询优化过程中将通信代价作为一个首要问题单独列出来研究，从以下两点看来似乎更合理：

首先，通信代价很好估计，通常是数据传输量的一个函数，这个特点是 I / O 开销所不具备的。

其次，分布查询可分为两部分：存取策略的分布优化和局部优化，二者可以分别予以解决。其中，局部优化可以采用集中式数据库中的技术，而分布优化是分布式系统要考虑的。一般来说比局部优化更重要。

通信代价可以用下面的公式粗略计算：

$$TC(X) = C_0 + X * C_1$$

其中 X 是数据传输量，这里以 bit（位）为单位计算；C_0 为两结点之间初始化一次传输所花费的开销，它由通信系统决定，近似为一个常数，单位为 s（秒）；C_1 为单位数据（bit）传输的代价，单位为（秒/bit）。

总之，在分布式数据库中查询优化的首要目标是：使该查询执行时其通信代价最省。

不同结点之间的连接操作和并操作是数据传输的主要原因，因此连接查询的优化在分布优化中是举足轻重的。

14.3.4 连接查询的优化

对连接查询的优化可以采取两种策略，一种是使用半连接来缩减关系（或片段）进而节省传输开销；另一种是不采用半连接，而是用直接连接的优化方案。这里重点讨论半连接的方法。

一、用半连接来优化连接的方法

设 R，S 是两个关系，A，B 分别是 R，S 上的两个属性。

首先引入半连接的记号：$\ltimes$

R，S 在 A，B 上的半连接可表示为：

$$R \underset{A=B}{\ltimes} S = R \underset{A=B}{\bowtie} (\pi_B(S))$$

一个连接运算可以用半连接作为中间步骤来实现：

$$R \underset{A=B}{\bowtie} S = (R \underset{A=B}{\ltimes} S) \underset{A=B}{\bowtie} S$$

具体操作过程和执行代价如下：

设关系 R，S 分别存放在结点 r，s 上（如图 14.7 所示）。

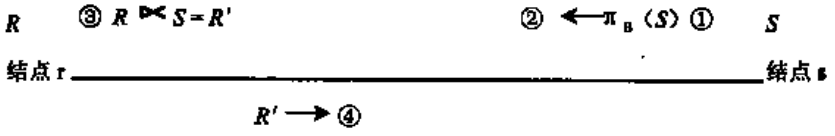


图 14.7

① 在结点 s 作关系 S 在属性 B 上的投影 $\pi_B(S)$

② 把 $\pi_B(S)$ 送到结点 r，代价为：

$$C_0 + C_1 \times size(B) \times val(B[S])$$

其中 size（B）表示属性 B 的长度，val（B[S]）表示关系 S 中属性 B 上不

同值的个数。

③在结点 r 计算半连接，设结果为 R′，即

$$R' = R \underset{A=B}{\ltimes} S$$

④ 把 R′ 从结点 r 送到结点 s，代价为：

$C_0 + C_1 \times \text{size}(R) \times \text{card}(R')$

card（R'）表示 R' 的元组数。

⑤ 在结点 s 执行连接操作：

$$R' \underset{A=B}{\bowtie} S$$

在上面的操作过程中，忽略了局部场地上执行操作的代价，仅考虑了通信代价。

直观地看，使用半连接可以缩减关系 R 的大小使得从结点 r 传送的关系由 R 缩减为 R'（一般 card（R）>> card（R'）），从而减少了通信代价，从中获得好处。

但是，由于执行半连接本身要传输数据（$\pi_B(S)$），需要通信代价，因而究竟是否采用半连接还要计算和比较其好处和开销之后再作决定。

采用半连接方案的总代价为：

$C_{SJ} = 2C_0 + C_1(\text{size}(B) \times \text{val}(B[S]) + \text{size}(R) \times \text{card}(R'))$

不用半连接，直接把 R 送到 S 执行连接的代价为：

$C_{JN} = C_0 + C_1 \times \text{size}(R) \times \text{card}(R)$

只有当 $C_{SJ} < C_{JN}$ 时，采用半连接方案才是合适的。

上面分析的是简单的连接查询，对于复杂的连接查询，即包括多个关系的连接，则可能存在许多可应用的半连接方案，而其中总有一个方案的效果最佳，因此优化的第一步是从所有半连接方案中找出一个最优的。第二步是与不用半连接的方案进行比较，选出较优者。

由美国计算机公司（Computer Corporation of America）研制的 SDD-1 分布式数据库系统就是采用的半连接的优化方案。

二、用直接连接的方案

上面已经讲过，用半连结方案并不总是有利可图的，因此有些系统如 R*（美国 IBM San Jose 研究室研制的系统 R 的分布版本）就采用了直接做连接的方案。

直接做连接也有不同的操作方法可选用。R*给出了以下几种可选的方法，它们的算法思想与集中库中的相同：

- 嵌套循环法。

- 合并扫描法

R* 支持两种传输方式：

- 整体传输：把整个关系传至指定结点再作处理。
- 按需传输：逐个取元组，只传所需要的元组。

以上两种连接算法和传输方式的组合构成了 R* 优化过程中的候选方案，经过代价计算选出其中最省的方案作为最终的查询计划。

14.4 分布事务管理

一个事务的执行必须保持其原子性，即它所包含的所有更新操作要么全部都做，要么都不做。

在分布式数据库系统中，一个全局事务会涉及多个场地上的数据更新。因此，事务是分布执行的。可以把一个事务看成由不同场地上的若干子事务组成。分布事务的原子性就应该是：组成该事务的所有子事务要么一致地全部提交，要么一致地全部滚回。

在多用户系统中，还必须保证分布事务的可串行性。因此，分布事务管理主要包括两个方面：事务的恢复和并发控制。这一节简要地讨论这两个问题。

14.4.1 分布事务的恢复

数据库在运行过程中故障和错误总是难免的。不同的故障会造成数据库不同程度的损害，轻则事务不能正常运行，重则造成数据库数据不一致性。更严重的情况，会使部分或整个数据库遭到破坏。已在第七章中讨论了集中式数据库系统中的故障类型和恢复技术。在集中式数据库系统中利用日志文件（必要时加上后备副本）总可以把数据库恢复到某种正确的状态。

分布式数据库系统中，各个场地除了可能发生如同集中式数据库的那些故障外，还会出现通信网络中通信信息丢失、长时间延迟、网络线路中断等事故。因此，情况比集中式数据库更复杂，相应的恢复过程也就更复杂些。

为了执行分布事务，通常在每个场地都有一个局部事务管理器，用来管理局部子事务的执行，保证子事务的完整性。同时这些局部管理器之间还必须相互协调，保证所有场地对它们所处理的子事务采取同样的策略：要么都提交，要么都滚回。为了保证这一策略，最常用的技术是两段提交协议（2-Phase-Commitment Protocol，简称 2PC）。

两段提交协议把一个分布事务的事务管理分为两类：一个是协调者，所有

其他的是参与者。协调者负责作出该事务是提交还是撤销（Abort）的最后决定。所有参与者负责管理相应子事务的执行及在各自局部数据库上执行写操作。

两段提交协议的内容是：

（1）第一阶段：协调者向所有参与者发出“准备提交”信息。如果某个参与者准备提交，就回答“就绪”（Ready）信息，否则回答“撤销”信息。参与者在回答前，应把有关信息写入自己的日志中。协调者在发出准备提交信息前也要把有关信息写入自己的日志中。

如果在规定时间内协调者收到了所有参与者“就绪”的信息，则将作出提交的决定，否则将作出撤销的决定。

（2）第二阶段：协调者将有关决定的信息先写入日志，然后把这个决定发送给所有的参与者。所有参与者收到命令之后首先往日志中写入“收到提交（或撤销）”决定的信息，并向协调者发送“应答（ACK）”消息，最后执行有关决定。协调者收到所有参与者的应答消息后，一个事务的执行到此结束，有关日志信息可以脱机保存。

采用两段提交协议后，当系统发生故障时，各场地利用各自有关的日志信息便可执行恢复操作，恢复操作的执行类似集中式数据库。

可以看出，在两阶段提交协议中，各结点采取的是完全同步的方法来保证数据库的一致性，称为紧密一致性（Tight Consistency）。

紧密一致性能绝对保证任何时刻整个分布式数据库系统的数据一致性和全局事务的原子性，但从实际应用的角度来说，它的缺陷也是非常明显的。

（1）全局事务可靠性低，任何一个参与结点的失败或网络的中断都将导致整个事务的回滚；

（2）系统效率低下，由于采用完全同步的方法来保证数据库的一致性，系统的性能将取决于系统中最慢的结点。

为了解决紧密一致性的缺陷，相应地提出了松散一致性（Loose Consistency）的概念。数据各副本的修改是异步的，也就是说各副本将不保证任何时刻数据库绝对的一致性；各副本同步的延迟时间可长可短，视系统的具体情况而定。与紧密一致性相比，松散一致性比较灵活，系统的可用性大为提高，适用于对数据一致性要求不是很高的应用，但对于银行转账这样的关键性应用显然是不适合的。

14.4.2 并发控制

集中式数据库系统中并发控制一般采用封锁技术。

锁可分为不同类型，常用的是共享锁（SLock）、排它锁（XLock）。封锁的对象可以是表一级的或记录一级的。

为了保证并发事务的可串行性，必须做到：

- 遵守锁的相容性规则
- 遵守两段锁（2-Phase Lock）协议，即如果一个事务已释放了一个锁，那么它就不能再请求新的封锁。

在分布式数据库系统中并发控制也可采用封锁技术，不过与集中式数据库系统相比，由于下面两个原因，使并发控制更为复杂。

（1）分布式数据库系统支持多副本的特点。

（2）由于事务的分布执行，封锁的方法会引起全局死锁。

因此分布式数据库系统中基于封锁技术的并发控制必须解决这两个问题。

分布式数据库系统支持多副本的特点给并发控制带来了困难。例如事务 T_1 在场地 1 获得了对数据 d 的 X 封锁，而事务 T_2 在场地 2 对数据 d 的副本申请 X 封锁，场地 2 的事务管理器并不知道场地 1 上的 d 已被 T_1 加了 X 封锁，因此事务 T_2 仍获得对 d 的 X 封锁，这样，封锁机制就失去了原来的功能而完全无用了。

T_1 在场地 1	T_2 在场地 2
Xlock（d_{site1}）	Xlock（d_{site2}）

为了避免这个问题，分布事务管理就要把“事务 T_1 对 d 的 X 封锁”这件事让 d 副本所在场地上的事务管理器都知道，一个简单的方法是向这些场地的事务管理器发出局部封锁请求，这个办法是有效的。但封锁的冗余度很大，局部封锁的数目和副本数相同。为了减少系统开销，处理多副本的封锁可采取如下几种方法：

（1）对写操作，要申请对所有副本的 X 锁。对于读操作，只要申请对某个副本的 S 锁。

（2）无论是写操作还是读操作都要对多数（大于半数）副本申请 X 锁或 S 锁。

（3）规定某个场地上的副本为主副本，所有的读写操作均申请对主副本的封锁。

这三个方法均可有效地发现冲突，协调并发事务的执行。

基于封锁的并发控制方法在分布环境下还必须解决全局死锁的问题，所谓全局死锁即包括两个以上场地上的死锁。

和集中式数据库系统相似，通常采用分布等待图的方法来检测死锁。不同的是，分布情况下死锁检测涉及多个场地，多个局部数据库，需要较多的通信和验证，开销较大。

除了用死锁检测及解除方式来解决死锁问题外，还可以采用死锁预防方法，即不让死锁发生。典型的一种解决方法是对事务按某一标准进行排序，只允许它们沿着这一次序单向等待，这就不会发生死锁了。

在分布式数据库中还研究了基于时标（Time Stamp，亦称为时间戳）的方法和乐观方法等并发控制技术，但实际系统则大都是基于封锁的方法。

14.5 分布式数据库的发展前景和应用趋势

分布式数据库兴起于20世纪70年代，繁荣于80年代，而在90年代分布式数据库更以其在分布性和开放性方面的优势重又获得了青睐。其应用领域已不再局限于 OLTP 应用，从分布式计算、Internet 应用、数据仓库到高效的数据复制都可以看到分布式数据库系统的影子。

这些新型的应用与传统的分布式数据库应用相比不再具有理论上强调的所有特点和功能，而是根据应用环境和需求的不同在保留基本结构的基础上作出了适当的修改或放松。

以 SYBASE 公司的 Replication Server 为例，该产品是一种典型的分布式数据库系统，具有分布式体系结构、支持场地自治和全局应用、实现了分布透明性。但出于实用的考虑，Replication Server 并没有采用两段提交的方式管理全局事务。原因很简单，Replication Server 强调系统的可用性、可靠性、场地自治，而两段提交协议在具体实施时有诸多缺陷（见 14.4.1 节），所以在 Replication Server 中采用松散一致性来完成全局事务，以满足实际应用的性能要求。

随着网络环境的日益普及，新的应用呈现出许多相似的特点，那就是开放性和分布性。以数据仓库为例，它建立在原有多个数据库系统的基础之上，需要能从异构数据库中提取数据并且能逐步扩展处理能力和系统规模；而对于 Internet 商业应用来说分布性和开放性更是最基本的要求。并且随着人们对电子商务、安全防范等较复杂的 Web 应用需求的增加，Web 应用不仅仅是对只读信息的存取，面向商业活动的读/写存取将迅速增加。也就是说，从简单的数据系统全球连网查询，逐渐转向更具有分布式数据库系统特色的应用环境。

正如在 14.1.3“分布式数据库系统的目标”中提到的，研制分布式数据库系统的目的、动机主要是组织目标，即为应用所驱动。开放性、分布性不仅是当前新应用的要求，也同样是分布式数据库系统的优势所在。可以想见在当前的网络、分布、开放的大环境下，分布式数据库系统将会有更加长足的发展和应用。

14.6 小 结

由于应用的驱动，数据库技术和网络技术的进展，分布式数据库技术得到了长足的发展。

这一章讨论了分布式数据库系统的基本概念、这一领域内的某些主要问题以及解决这些问题所采用的基本技术和基本方法。

分布式数据库系统以集中式数据库系统技术为基础，但不是简单地把集中式数据库连网就能实现的。

分布式数据库系统分布在网络的不同计算机（结点）上，既具有高度的自治性，更强调各场地系统之间的协作性。对于用户来说，一个分布式数据库系统逻辑上看如同一个集中式数据库系统一样 ，用户可以在任何一个场地执行全局应用和局部应用。

分布式数据库系统具有自己的性质和特征。集中式数据库的许多概念和技术，如数据独立性、数据共享和减少冗余度、并发控制、完整性、安全性和恢复等在分布式数据库系统中都有了不同的更加丰富的内容。例如，在分布式数据库系统中，数据独立性包括数据的逻辑独立性、物理独立性以及数据分布独立性（亦称分布透明性）。分布式数据库系统中，分布查询处理和优化、分布事务的恢复、并发控制等技术比集中式数据库更加复杂，是分布式数据库系统重要的技术内容。

在当前网络、分布、开放的大环境下，分布式数据库系统的应用更加广泛。

习 题

1. 什么样的数据库系统是分布式数据库系统？图 14.1 的系统配置在什么情况下只能算分散的数据库系统？在什么条件下才是分布式数据库系统？

2. 分布式数据库系统有什么特点？

3. 试述研制分布式数据库系统的目的和动机。

4. 试述分布式数据库系统的模式结构。

5. 什么是数据分片？有几种分片方式？数据分片的目的是什么？有什么优点？

6. 试述分布透明性的内容。

7. 什么是同构型 D-DBMS？ 什么是异构型 D-DBMS？

8. 设在 14.2.3 节的分布式数据库系统例子中，还有全局关系 SC(SNO，CNO，G)，它具有两个导出分片 SC_A，SC_B，分别存储理学院和文学院学生的选课记录。SC_A

存放在场地 4，SC_B 存放在场地 5。今有一个稍复杂的查询，从终端输入一个课程号，查找选修该课程的学生学号和姓名，并把它们显示在屏幕上。请写出具有不同层次分布透明性(类比例子中的三种情况)的应用程序。不必给出细节，只需写出算法思想。

9. 对 14.3.1 节的例子中介绍的六种策略改用下列估算值后分别计算通信时间：

红色零件数=1 000，

北京供应商的装运单=10 000。

10. 设关系 R，S 存储在不同的场地，已知：

card(R)=100，size(R)=50

card(S)=50，size(S)=5

今要计算 $R \underset{A=B}{\bowtie} S$，其中 A，B 分别为 R，S 上的属性，并且

Val(A[R])=50，size(A)=3

Val(B[S])=50，size(B)=3

若 $R'=R \underset{A=B}{\ltimes} S$，card(R')=0.2*card(R)

$S'=S \ltimes R$，card (S')=0.8*card(s)

请比较采用以下不同策略时的通信代价：

(a) 在 R 所在场地执行连接操作，利用半连接方法；

(b) 在 S 所在场地执行连接操作，利用半连接方法；

(c) 在 R 所在场地执行连接操作，不用半连接方法；

(d) 在 S 所在场地执行连接操作，不用半连接方法；

哪种情况代价最省(不考虑把结果传送到某一场地的代价)?

11. 试述下列概念：

两段提交协议(2PC)；分布事务的原子性；全局死锁。

12. 在分布式数据库系统中，对多副本的封锁有几种解决方法?

本章参考文献

1 Rothnie J B，Goodman N，A Survey of Research and Development in Distributed Database Management. in Proceedings of LDB，October 1977

（本文综述了分布式数据库系统早期的研究结果。）

2 Ceri S et al. Distributed Databases：Principles and Systems. McGraw Hill，NY，1984

（本书介绍了分布式数据库系统的基本概念、原理和系统。）

3 周龙骧等. 分布式数据库管理系统实现技术. 数据库丛书之一，北京：科学出版社，1998

4 郑振楣，于戈，郭敏. 分布式数据库. 数据库丛书之一，北京：科学出版社，1998

5 Tamer Ozsu M，Valduriez P. Principles of Distributed Database Systems. Prentice-Hall International Inc. 1991

6 石树刚，郑振楣，阮春等．分布式数据库系统的实用化．微型计算机，1989，3

（本期为分布式数据库的专刊，刊登了十余篇论文（如文献[6]，[7]，[8][9]，[10]，[11]），反映了国内在这一领域中当时的研究的情况和部分成果。）

7 李曦，王珊，萨师煊．一个基于大型机远程通讯的分布数据库查询系统 DQS/SEIS．微型计算机，1989，3

8 杜小勇，王珊，萨师煊．分布式数据库 DQS/SEIS 设计实现中若干问题的探讨．微型计算机，1989，3

9 张霞，郑怀远．面向客体和语义关联的异构分布式数据库系统 OHDDB-DGI．微型计算机，1989，3

10 王能斌，徐立臻，程琪．SUNDDB：一个分布式数据库管理系统．微型计算机，1989，3

11 周晓方，金志权，柳诚飞．LSZTM：一个分布式数据库系统的管理．微型计算机，1989，3

12 王珊等．The Global Conceptual Model Design in DQS/SEIS．第三届泛太平洋计算机国际会议 PPCC-3 论文集．1989.8

（本文讨论分布数据库系统全局概念模式的设计，以中国人民大学为国家经济信息系统研制的分布查询系统 DQS/SEIS 为实际背景。）

13 周龙骧，顾君忠．POREL：一个分布式关系型数据库系统．计算机科学，1982

14 Ceri S，Navathe B，Wiederhold G．Distribution Design of Logical Database Schemas．IEEE Transactions on Software Engineering，Vol.SE-9，No.4，July 1983

（本文讨论了分布式数据库系统的设计问题。）

15 Mahmoud S，Riordon J S．Optimal Allocation of Resources in Distributed Information Networks．ACM TODS，Vol.1，No.1，January 1976

（本文讨论了分布数据库系统中资源的优化分配问题。）

16 Epstein R，Stonebraker M R．Analysis of Distributed Database Processing Strategies．in Proceedings of VLDB，1980

（本文讨论了分布式数据库中的查询处理策略。）

17 Apers P M G．Yao S B．Optimization Algorithms for Distributed Queries，IEEE Transactions on Software Engineering，Volume SE-9，Number 1，January 1983

（本文讨论了分布式查询处理的优化算法。）

18 Daniels D，Selinger P G，et al．An Introduction to Distributed Query Compilation in R*，in Distributed Data Bases．Proceedings of International Symposium on Distributed Databases，1982

（本文全面介绍了分布式数据库系统 R*，介绍 R*的论文很多，这只是其中之一。）

19 Berntstein P A，Chiu D W．Using Semijoins to Solve Relational Queries．Journal of the ACM，Volume 28，Number 1，January 1981

（本文讨论了分布式查询处理的半连接优化方法。）

20 Kambayashi Y，Yoshikawa M，Yajima S. Query Processing for Distributed Databases Using Generalized Semi-joins. Proceedings of the ACM SIGMOD，1982

（本文讨论了分布式查询处理的半连接优化方法。）

21 Traiger I L，et al. Transactions and Consistency in Distributed Database Management Systems. ACM TODS，Volume 7，Number 3，September 1982

（本文讨论了分布式数据库系统中事务及其一致性的概念。）

22 Bernstein P A，et al. Concurrency Control in a System for Distributed Databases(SDD-1). ACM TODS，Volume 5，Number 1，March 1980

（本文讨论了分布式数据库的并发控制问题。）

23 Bernstein P A，Goodman N. Timestamp-based Algorithms for Concurrency Control in Distributed Database Systems. in Proceedings of VLDB，1980

（本文讨论了布式数据库系统中基于时间戳的并发控制方法。）

24 Kohler W H. A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems. ACM Computing Surveys，Volume 13，Number 2，June 1981

（本文讨论了分布数据库系统中的同步和数据库恢复问题。）

25 Jajodia S，Mutchler D. Dynamic Voting Algorithms for Maintaining the Consistency of a Replicated Database. ACM TODS，Volume 15，Number 2，June 1990

（本文讨论了分布数据库系统中建立在选举概念之上的冗余数据的并发控制。）

26 Bhargava B (Editor)，Concurrency and Reliability in Distributed Systems. Van Nostrand Reinhold，New York，1987

（本文献编辑了多篇论述分布式数据库并发控制和可靠性问题的论文。）

27 Lampson B，Sturgis H. Crash Recovery in a Distributed Data Storage System. Technical Report，Computer Science Laboratory，Xerox，Palo Alto Research Center，Palo Alto，CA，1976

（本文提出了分布式数据库系统中事务的两段式提交协议。）

28 Skeen D. Non-blocking Commit Protocols. Proceedings of the ACM SIGMOD，1982

（本文提出了分布式数据库系统中事务的三段式提交协议。）

29 Obermarck R. Distributed Deadlock Detection Algorithm. ACM TODS，Volume 7，Number 2，1982

（本文讨论了分布式数据库系统中的死锁检测问题。）

30 Litwin W，Abdellatif A. Multidatabase Interoperability. IEEE Computer，Volume 12，Number 1986

（本文讨论了多数据库系统的互操作问题。）

*第十五章　并行数据库系统

15.1　并行数据库系统概述

并行数据库系统是在并行机上运行的具有并行处理能力的数据库系统。并行数据库系统是数据库技术与并行计算技术相结合的产物。

近年来，计算机系统性能价格比的不断提高迫切要求硬件和软件结构的改进。硬件方面，传统的靠提高微处理机速度和缩小体积来提高性能价格比的方法正趋于物理的极限；磁盘技术的发展滞后于微处理机的发展速度，使得磁盘I/O瓶颈问题日益突出。软件方面，随着应用领域数据库规模的急剧膨胀，数据库服务器对大型数据库各种复杂查询响应时间和联机事务处理(OLTP)吞吐量的要求顾此失彼。数据库应用（如DSS，OLAP等）的发展对数据库的性能和可用性提出了更高的要求，能否为越来越多的用户维持高事务吞吐量和低响应时间已成为衡量DBMS性能的重要指标。

随着微处理机技术和磁盘阵列技术的进步，并行计算机的发展十分迅速，出现了像Sequent，Tandem，Teradata和曙光机这样一些商品化的并行计算机系统。这些系统使用数个、数十个、甚至成千上百个廉价的微处理机协同工作，性能价格比远比中大型计算机系统高；而且，更为吸引人的是，这些系统由于广泛地采用了磁盘阵列技术，增加了I/O的带宽， 能有效地缓解应用中的I/O瓶颈问题。

并行计算技术利用多处理机并行处理产生的规模效益来提高系统的整体性能，为数据库系统提供了一个良好的硬件平台。研究和开发适应于并行计算机系统的并行数据库系统成为数据库学术界和工业界的研究热点，形成了并行处理技术与数据库技术相结合的并行数据库新技术。

并行处理技术与数据库技术的结合，具有潜在的可行性。因为关系数据库模型本身就有极大的并行可能性。关系数据模型中，数据库是元组的集合，数据库操作实际是集合操作，许多情况下可分解为一系列对子集的操作，许多子操作不具有数据相关性，因而具有潜在的并行性。

15.2 并行数据库系统的目标

一个并行数据库系统应该实现如下目标：

1. 高性能

并行数据库系统通过将数据库管理技术与并行处理技术有机结合，发挥多处理机结构的优势，从而提供比相应的大型机系统要高得多的性能价格比和可用性。例如，通过将数据库在多个磁盘上分布存储，利用多个处理机对磁盘数据进行并行处理，从而解决磁盘“I/O”瓶颈问题。通过开发查询间并行性（不同查询并行执行）、查询内并行性（同一查询内的操作并行执行）以及操作内并行性（子操作并行执行）大大提高查询效率。

2. 高可用性

并行数据库系统可通过数据复制来增强数据库的可用性。这样，当一个磁盘损坏时，该盘上的数据在其他磁盘上的副本仍可供使用，且无需额外开销（与基于日志的恢复不同）。数据复制还应与数据划分技术相结合以保证当磁盘损坏时系统仍能并行访问数据。

3. 可扩充性

这里，数据库系统的可扩充性指系统通过增加处理和存储能力而平滑地扩展性能的能力。理想情况下，并行数据库系统应具有两个方面的可扩充性优势：线性伸缩和线性加速。

其中，线性伸缩比（Linear Scale up）是指当任务扩大 N 倍，系统处理和存储能力也扩大 N 倍时系统性能不变。

$$\text{线性伸缩比}=\frac{\text{小任务在小系统上的运行时间}}{\text{大（}N\text{倍）任务在大系统上的运行时间}}=1$$

线性加速比（Linear Speedup）是指当任务不变而系统处理和存储能力扩大 N 倍时，系统性能也提高 N 倍。

$$\text{线性加速比}=\frac{\text{小系统上执行一个任务的时间}}{\text{大（}N\text{倍）系统上执行同一任务的时间}}=N$$

15.3 支持并行数据库的并行结构

从硬件结构来看，根据处理机与磁盘及内存的相互关系可以将并行计算机

分为三种基本的体系结构，并行数据库系统研究一直以三种并行计算结构为基础，下面分别介绍这三种基本结构，并从性能、可用性和可扩充性等三个方面来加以比较。

- 共享内存（主存储器）结构 (Shared_Memory)（以下简称 SM 结构）；
- 共享磁盘结构(Shared_Disk)（以下简称 SD 结构）；
- 无共享资源结构(Shared_Nothing)（以下简称 SN 结构）。

15.3.1 SM 并行结构

SM 并行结构由多个处理机、一个共享内存（主存储器）和多个磁盘存储器构成。多处理机和共享内存由高速通信网络连接，每个处理机可直接存取一个或多个磁盘，即所有内存与磁盘为所有处理机共享。SM 并行计算机结构如图 15.1 所示。IBM/370 多处理机系统、VAX 多处理机系统和 Sequent 系统是具有 SM 结构的并行计算机系统。

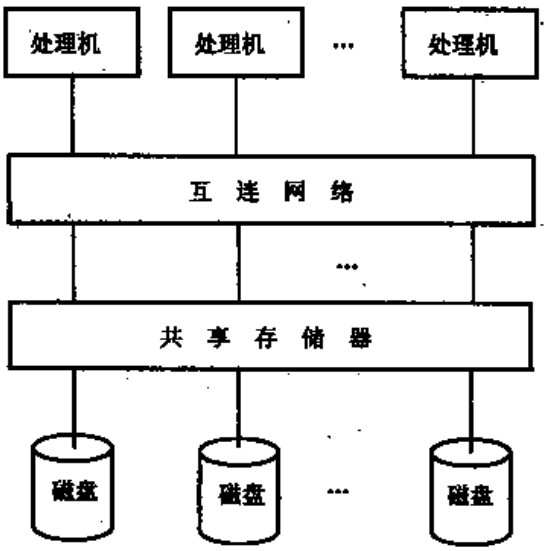


图 15.1 SM 结构并行计算机

并行数据库系统中，XPRS，DBS3，Volcano 以及中国人民大学研制的 PBASE/1 都是基于 SM 结构的。

SM 方案的优势在于实现简单和负载均衡。

在该结构中，共同执行一条 SQL 语句的多个数据库构件通过共享内存来交换消息与数据。数据库中的数据划分在多个局部磁盘上，并可以为所有处理机访问。由于元信息（目录表）和控制信息（锁表等）可被所有处理机共享，因此数据库软件的编制与单处理机情形区别不大。特别是，查询间并行性的实现不需额外开销，查询内并行性的实现也不太困难。这种系统可以基于实际负

荷来给各处理机动态地分配任务，因而可以很好地实现负荷均衡。但是这种结构的系统由于硬件成员之间的互连很复杂（每个处理机要与每个内存模块和每个磁盘相连），因而成本高。由于访问共享内存和磁盘会成为瓶颈，为了避免访问冲突增多而导致系统性能下降，结点数目必须限制在几十个以内，可扩充性较差。此外，内存的任何错误都将影响到多个处理机，因此系统可用性不是很好。

15.3.2 SD 并行结构

SD 并行结构由多个具有独立内存（主存储器）的处理机和多个磁盘构成。每个处理机都可以读写任何磁盘。多个处理机和磁盘存储器由高速通信网络连接。SD 结构并行计算机如图 15.2 所示。IBM 的 Sysplex 系统就是一个基于 SD 结构的并行计算机系统。

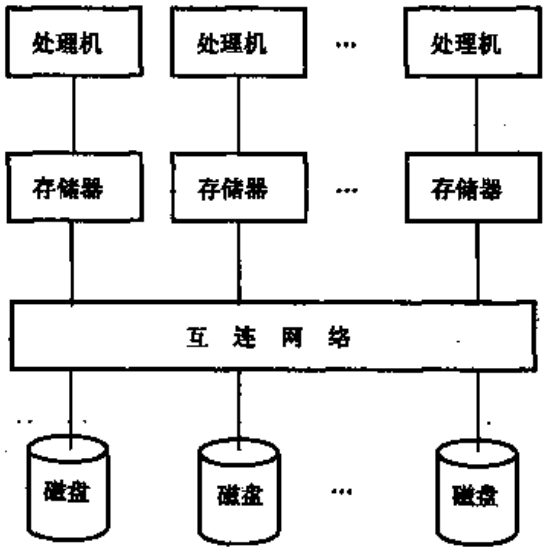


图 15.2 SD 结构并行计算机

采用这一结构的数据库系统有 IBM 的 IMS/VS Data Sharing 和 Dec 的 VAX DBMS 和 Rdb 产品。在 DEC 的 VAX 群集机和 nCUBE 机上实现的 Oracle 系统也采用这一结构。

SD 方案具有成本低、可扩充性好、可用性强、容易从单处理机系统迁移以及负载均衡等优点。

首先，具有这种结构的系统可以使用标准总线互连，因而成本较之 SM 结构要低；其次，当每个处理机有足够大的高速缓存空间，对共享磁盘的访问冲突可降低，因而可扩充性较好，结点数目可达数百个；单个处理机结点的失败不致影响其他处理机的工作，因而系统可用性高；由于磁盘数据不需重新组织，

所以从一个单处理机的集中系统向一个 SD 系统迁移简便而直接；至于负荷均衡的理由则与 SM 结构相同。

该结构的不足在于实现起来复杂以及存在潜在的性能问题。这是由于 SD 方案中每一处理机可以访问共享磁盘上的数据库页，但它们无共享内存。因此数据被拷贝到各自的高速缓冲区中。为避免对同一磁盘页的访问冲突，该结构需要一个分布式缓存管理器来对各处理机（结点）并发访问进行全局控制与管理，并保持数据的一致性。维护数据一致性会带来额外的通信开销，此外对共享磁盘的访问是潜在的“瓶颈”。

15.3.3 SN 并行结构

SN 并行结构由多个处理结点构成。每个处理结点具有自己独立的处理机、内存（主存储器）和磁盘存储器。多个处理机结点由高速通信网络连接。SN 结构并行计算机如图 15.3 所示。 具有 SN 结构的并行计算机系统包括 nCUB 系统、Teradata 系统、Tandem 系统和 VAXcluster 系统等。

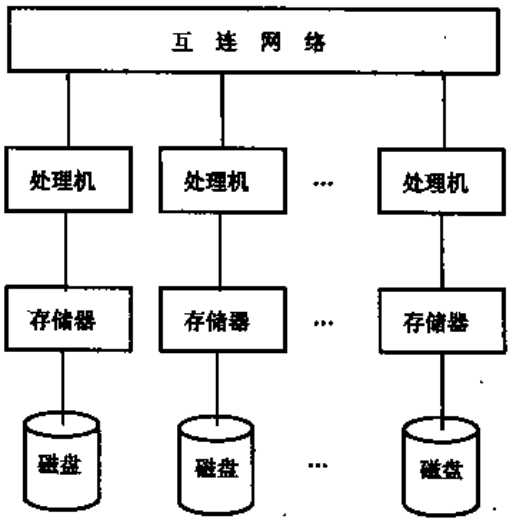


图 15.3 SN 结构并行计算机

SN 方案中，由于每一结点可视为分布式数据库系统中的局部场地（拥有自己的数据库软件），因此分布式数据库设计中的多数设计思路，如数据库分片、分布事务管理和分布查询处理等，都可以借鉴。

采用 SN 方案的有 Teradata 的 DBC 和 Tandem 的 NonStop SQL 以及中国人民大学的 PBASE/2 等系统。

SN 结构成本较低，它最大限度地减少了共享资源，具有极佳的可伸缩性，

结点数目可达数千个，并可获得接近线性的伸缩比。而通过在多个结点上复制数据又可实现高可用性。

该方案的不足在于实现复杂以及结点负荷难以均衡。往往只是根据数据的物理位置而不是系统的实际负载来分配任务。并且，系统中新结点的加入将导致重新组织数据库以均衡负载。

15.3.4 三种并行结构比较

下面列出对以上三种并行结构从性能、可用性和可扩充性等方面的比较。

表 15.1 三种并行数据库体系结构比较

	Shared_Memory	Shared_Disk	Shared_Nothing
性能	最佳	较佳	较佳
可用性	低	较高	高
可扩充性	差	较好	好
负载均衡	易做到	易做到	难做到
实现技术	容易	较复杂	复杂
成本	高	较低	低
处理机数	数十个	数百个	数千个
规模	中小系统	中小系统	大系统

简而言之，在可扩充性与可用性方面 SN 结构显然要优于其他两种结构；而在负载均衡、设计的简单性等方面，则是 SM 结构优点突出一些。对于结点数目较多的配置，SN 结构比较好地适应了高伸缩性的要求，它通过最小化共享资源来最小化资源竞争带来的系统干扰。而对于中小型系统的配置，SM 结构由于其设计的简单性和负载易于均衡也许就更为合适一些。此外还有混合结构，即整个系统是 Shared_Nothing 结构而每个结点是 Shared_Memory 结构。这种结构综合了 SM 与 SN 的优点。

15.4 并行数据库的并行查询处理技术

在数据库管理系统中，查询处理的主要任务是翻译用户所给出的查询语句并对若干可能有效的查询处理计划进行评价、比较，从中选出能优化其性能的一个方案。查询处理的效率对整个系统的性能有很大的影响，因此并行数据库系统中查询处理的并行实现也就成为并行数据库管理系统的一个重要指标

首先介绍并行数据库系统中关于并行查询处理的一些基本概念。

对应于传统的顺序查询处理计划（Sequential Plan，简称 SP），称相应于并行处理环境下的执行计划为并行执行计划（Parallel Plan，简称 PP）。

如果查询 Q 的某个并行执行计划 PP 与 Q 的一个顺序执行计划 SP 对应于相同的操作树，则称 PP 为 SP 的一个并行化方案；而由顺序执行计划 SP 得到某个 PP 的过程称为并行化（Parallelization）。

一个串行执行计划可以通过不同的并行化过程得到不同的并行执行计划。并行化过程可以由下述一些基本概念刻画。

15.4.1 并行粒度

并行粒度指的是查询执行的并行程度，并行粒度可分为如下四种。

不同用户事务间的并行性、同一事务内不同查询间的并行性、同一查询内不同操作间的并行性和同一操作内的并行性。

（1）事务间（inter-Transaction）并行性

事务间的并行性是粒度最粗也是最易实现的并行性。由于这种并行性允许多个进程或线索同时处理多个用户的请求，因此可以显著增加系统吞吐量，支持更多的并发用户。

（2）查询间（inter-Query）并行性（也就是事务内并行性）

同一事务内的不同查询如果是不相关的，它们并行执行必将提高效率。但是，同一事务内的不同查询如果是相关的，它们并行执行比较复杂，系统必须进行相关性控制。

（3）操作间（inter-Operation）并行性（也就是查询内并行性）

同一查询内的不同操作往往可以并行执行。例如，一条 SQL 查询语句可以分解成多个子操作，由多个处理机执行。例如下列查询：

```
SELECT 部门号，职工号
FROM   部门，职工
WHERE  部门.部门号=职工.部门号
GROUP BY 部门号
ORDER BY 部门号
```

可以分解为扫描部门表和职工表、对两表进行连接、对连接结果排序以及分组和输出五个子任务。前一操作的输出即是下一操作的输入。如果后一操作等待前一操作产生一定量的输出后（而不必等待前一操作执行完毕）即可在另一处理机上开始执行。这种并行方式称为垂直并行或流水线并行。

（4）操作内（intra-Operation）并行性

操作内并行性的粒度最细，它将同一操作（如扫描操作、连接操作、排序

操作等）分解成多个独立的子操作，由不同的处理机同时执行。例如，如果部门表被划分到四个不同的磁盘上，则扫描部门表的操作就可以分解成四个子操作同时执行，从而大大加快了扫描部门表的操作。这种并行方式称为水平并行。

从广义上讲，事务间和查询间的并行性也属于水平并行。

图 15.4 给出了这四种并行性的相互关系。一般来说，并行粒度越细，并行化程度就越高，而实现就越为复杂。并行数据库系统通过开发事务间、查询间（同一事务内）、操作间（同一查询内）及操作内四种不同粒度的并行性，满足 OLTP 及 DSS 两种应用环境中对事务吞吐量和响应时间的要求。

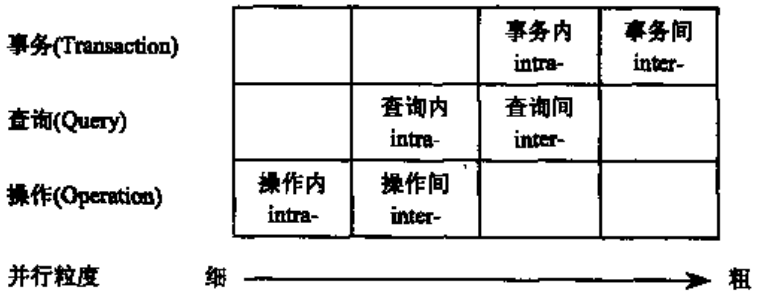


图 15.4 四种并行粒度

一个好的并行数据库系统应能充分利用硬件平台的并行性，提供四种不同粒度的并行性。

15.4.2 并行化形式

并行化可分为水平并行化（也称独立并行化，Independent Parallelism）和垂直并行化（也称流水线并行化，Pipelining Parallelism）两种形式，图 15.5 给出了这两种形式的示意图。

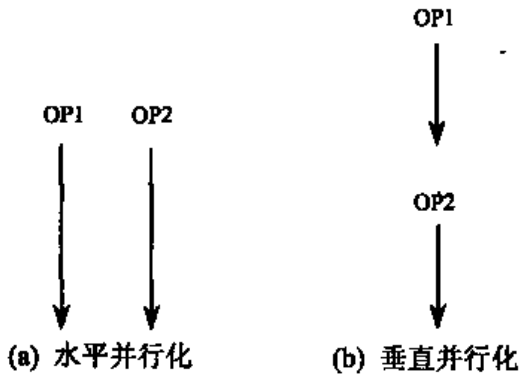


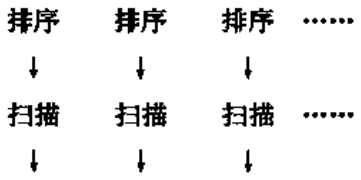
图 15.5 并行化的两种形式

如果两个操作 OP1，OP2 无相互依赖关系，则称这两个操作相互独立。水平并行化指的是互相独立的多个操作或者一个操作内互相独立的多个子

操作分别由不同的处理机并行执行的形式；

如果操作 OP2 直接依赖于 OP1，并且 OP2 必须等待 OP1 处理完所有元组后方可开始执行，则称 OP2 以阻塞方式直接依赖于 OP1；如果 OP2 无需等待 OP1 执行完毕即可在另一处理机上开始执行，则称 OP2 以流水线方式直接依赖于 OP1。垂直并行化则是指存在流水线方式依赖关系的操作分别由不同处理机并行执行的形式。

例如：排序操作、扫描操作由不同的处理机并行执行就是水平并行化的实例。



例如：扫描操作→排序操作→连接操作→分组操作由不同的处理机并行执行就是垂直并行化的实例。

由于关系代数的封闭性和数据操作的相对独立性，关系查询具有三种固有并行性，即操作间的流水线并行性、操作间的独立并行性以及操作内的独立并行性(参见本书 12.2.2 节中图 12.4)，这为关系代数的并行化提供了现实基础。

15.4.3 并行操作算法

使用并行数据操作算法实现查询的并行处理可以更充分地发挥多处理器的并行性，更大地提高系统的查询处理效率。并行数据操作算法的研究已经成为最近几年一个非常活跃的并行数据库系统研究领域。并行操作算法有并行连接算法，并行扫描算法，并行排序算法等。

由于连接是关系数据库系统中最耗时且最常用的操作，对并行连接操作的研究最多。提出了基于嵌套循环的并行连接算法、基于合并扫描的并行连接算法、基于 HASH 的并行连接算法，基于索引的并行连接算法等。

一、基于嵌套循环的并行连接算法

在顺序计算机环境中，嵌套循环连接算法一直被认为是效率最低的连接算法。然而，嵌套循环连接算法很容易并行化，并且可以通过附加的计算减少多处理机间的通信开销。下边的算法是一个典型的并行嵌套循环连接算法(简称PNLJ 算法)：

输入：R，S：分布在 P 个处理机上的关系

A：连接属性

P：处理机数

输出： 关系 R 和 S 的连接结果(连接属性为 A)；

方法：

(1) 把 S 均匀地分布到 P 个处理机，设 S_i 是 S 在结点 i 上的子集合；

(2) FOR i=1 TO P DO (并行地)

(3) 处理机 i 按照连接属性值排序 S_i；

(4) ENDFOR；

(5) 各处理机以流水线方式向 P 个处理机广播 R 的元组；

(6) FOR i=1 TO P DO (流水线方式并行地)

(7) 处理机 i 以流水线方式接收并按连接属性值排序其他处理机发来的 R 元组；

(8) 对磁盘上的 S_i 中和内存中的 R 元组做合并连接；

(9) ENDFOR

理论和实验结果表明，在具有大量处理结点的系统中，或在 R 和 S 的大小差别很大的情况下，PNLJ 算法的效率远高于其他并行连接算法。

二、基于 SORT-MERGE 的并行连接算法

基于 SORT-MERGE 的并行连接算法(简称 PSMJ 算法)由两个阶段组成，排序阶段和连接阶段。在排序阶段，它按照连接属性的值排序每个连接关系；在连接阶段，使用合并算法完成两个排序关系的连接。下边是一个具有代表性的 PSMJ 算法：

输入：R，S：分布在 P 个处理机上的关系

A：连接属性

P：处理机数

H：HASH 函数

输出：关系 R 和 S 的连接结果(连接属性为 A)；

方法：

(1) 使用 H 在 P 个处理结点间分布 R 和 S 的元组，设 S_i 和 R_i 是 S 和 R 在结点 i

上的子集：

（2）FOR i=1 TO P DO (并行地)

处理结点 i 排序 R_i 和 S_i；

（3）FOR i=1 TO P DO (并行地)

结点 i 使用合并算法完成 R_i 与 S_i 的连接；

（4）输出 R 和 S 的连接结果。

理论和实验结果表明，当两个连接关系的元组数相差较小时，PSMJ 算法的效率高于 PNLJ 算法。

三、基于 HASH 的并行连接算法

基于 HASH 的连接算法很容易并行化。它通过一个定义在连接属性上的 HASH 函数把连接关系分解为 P 个子集合，然后使用 P 个处理机并行地完成连接操作。简称并行 HASH-连接算法为 PHJ 算法。如果 HASH 函数能够把连接关系均匀地分布为 N 个子集合，则 PHJ 算法可以具有线性时间复杂性。这里介绍两个具有代表性的 PHJ 算法。

1983 年，Kitsyregawa 提出了第 1 个 PHJ 算法，称为 GRACE-HASH-连接算法，其算法如下：

输入：关系 R 和 S，HASH 函数 H_1 和 H_2，处理结点数 P，子集合数 M。

输出：关系 R 和 S 的连接结果。

（1）使用 H_1 把 R 划分为 M 个子集合，HASH 值为 i 的元组送入子集合 R_i，并存储到所有可用的磁盘；

（2）使用 H_1 把 S 划分为 N 个子集合，HASH 值为 i 的元组送入子集合 S_i，并存储到所有可用的磁盘；

（3）FOR i=1 TO M DO (并行地)

使用 H_2 把 R_i 分布到 P 个处理结点，在每个结点的内存中建立 R_i 元组的 HASH 表；

使用 H_2 以流水线方式向 N 个处理结点发送 S_i 的元组：

FOR j=1 TO P DO (并行地)

结点 j 用收到的 S_i 的元组匹配 R_i 的 HASH 表，进行 S_i 和 R_i 的连接；

（4）输出 R 和 S 的连接结果。

算法中的 M 应该充分大，以减少 R 的子集合对应的 HASH 表超过可用内存容量的概率。Schneder 和 DeWitt 改进了 GRACE-HASH-连接算法，提出了一个称为 HYBRID-HASH-连接的 PHSJ 算法。

实验表明， 使用并行数据操作算法实现查询的并行处理可以充分地发挥多处理机的并行性，从而改善关系运算的效率，提高查询处理的速度。

15.4.4 并行查询优化

查询优化始终是数据库管理系统的重要组成部分，查询优化的目标在于提高执行效率。

并行数据库查询优化问题与顺序数据库查询问题有所不同。在顺序数据库系统中，给定一个查询 Q，查询优化算法只需找到 Q 的一个具有最小工作量的执行计划。这样的计划必然具有最快的响应时间。这是因为在具有单处理机的计算机系统中，一个计算任务的响应时间与这个任务的工作量成正比。在并行数据库系统中，情况就不同了。Q 的具有最小工作量的执行计划可能具有很强的固有顺序性，难以并行化，因而不具有最小响应时间。在并行数据库系统中，查询优化的目标是寻找 Q 的具有最小响应时间的执行计划，执行计划的工作量不是第一重要的。于是，在并行数据库系统中，需要新的查询处理算法和新的查询优化技术。

由于并行数据库环境中存在多个处理机，并行查询优化应尽可能地使每个操作并行处理，充分利用系统资源提高并行度来达到提高系统性能的目的。

并行查询优化面临的两大困难在于：

（1）执行计划搜索空间的庞大

在传统的顺序查询优化中，往往采取穷尽的方法或者某种半穷尽的方法搜索整个查询计划空间，对每个查询计划估算出代价，而后找出代价最小的查询计划。然而，在并行环境下，并行执行计划空间成指数增长。假设 SPLAN(Q)是查询 Q 的顺序执行计划空间，P 是一个顺序执行计划，PARALLEL(P)是该计划的所有并行化方案。那么查询 Q 的并行执行计划空间 PPLAN(Q)则可以由下述公式来表示：

$$PPLAN(Q)=\bigcup_{P\in SPLAN(Q)} PARALLEL(P)$$

由此可见，依靠传统的穷尽方法进行并行查询优化是不现实的。应该提供某种启发式的方法对并行执行计划空间作裁剪以减少搜索空间的代价。

在并行数据库中，查询优化的研究主要围绕着具有多个连接操作的复杂关系数据库查询的优化问题进行。不少学者相继提出了基于左线性树的查询优化算法、基于右线性树的查询优化算法、基于片段式右线性树的查询优化算法、基于浓密树的查询优化算法、基于操作森林的查询优化算法等。这些算法在搜索代价和最终获得的查询计划的效率之间有着不同的权衡。

（2）执行时的某些系统参数比如 CPU 数目、内存大小在优化时是未知的

在多用户环境下，系统参数如 CPU 数目、内存大小只能到计划执行时才能确定。有的学者提出了两阶段优化的思想，将查询处理分为静态顺序优化和

动态并行化两个阶段进行：

• 阶段 1：在编译阶段，假设全部内存大小可为查询所用，利用传统查询优化策略得到最优顺序执行计划。

• 阶段 2：执行阶段，根据阶段 1 的顺序执行计划得到给定缓冲区大小和处理机数目条件下的最优并行化方案。

将并行查询优化分解为两个阶段进行，有效地解决了并行查询面临的两大困难，同时又降低了并行查询优化算法的复杂性。但问题是，两阶段优化是否能保证并行执行计划的最优性。有关并行查询优化的策略和查询优化算法的详细介绍可参考“全书参考文献”[24]。

15.5 并行数据库的物理组织

DBMS 的操作是 I/O 密集型的操作，I/O 瓶颈问题是长期阻碍 DBMS 性能提高的一个重要原因，不真正利用多磁盘技术，扩大 I/O 带宽，DBMS 的性能不能实现质的飞跃。在磁盘阵列技术下如何进行数据库的合理设计，就成为并行数据库要研究的问题。

15.5.1 并行数据库的数据划分

目前，并行计算系统中这个问题的普遍解决方法是通过数据分布(Data Declustering)将每个关系划分为小的片段，并把这些小片段均匀地分布在系统的多个磁盘驱动器上。由于能够并行地读写磁盘，相当于增加了 I/O 带宽，从而能够在很大程度上消除 I/O“瓶颈”。

数据分布方法降低了数据的聚集度(Cluster Degree)，每个操作能够由多个处理机来承担，因此能够减少查询的响应时间并提高整个系统的吞吐量。

并行数据库是以提高系统性能为宗旨的，即尽可能地使每个操作并行处理，因此就十分强调数据分布的均匀性。若负载不均，往往会造成多个处理机结点能力的浪费。所以，使用正确的数据分布算法以达到负载均衡是并行数据库中数据分布的关键问题。

数据划分是并行查询处理的重要基础。许多研究表明，数据划分对于并行数据库系统的性能具有很大的影响。

可以依据数据的某一个属性并基于一个函数（Hash 函数或范围索引）将关系进行水平划分，然后将得到的各个独立的数据集分配到不同的磁盘上，类似于分布式数据库中的水平分片技术。这种数据分布技术不仅有利于均衡负

荷，也有益于查询间 / 查询内并行性，因为不相关的查询或者同一个查询内的独立操作可以同时处理不同磁盘数据集中的数据。

依照存放关系的结点数目的不同，数据划分技术可分为完全划分（亦称为完全分布）和变量划分（称为部分分布）两种类型。前者将每一个关系分布存储到所有结点上，这种方法不适合于小关系以及结点数目大的系统；后者将每一关系只分布存储到部分结点上，其中结点数目是关系大小和访问频率的一个函数，从而使数据分布更为灵活。

划分数据时可以依据一个属性的值，也可以同时依据多个属性的值，前者称为一维数据划分，后者则称为多维数据划分。一维数据划分方法相对比较简单，常用的数据划分方法有：

- 轮转法(round_robin)
- Hash 法(hashed declustering)
- 值域划分法(range_partition declustering)

此外，还有用户定义（user-defined）划分法、模式（schema）划分法、Hybrid_Range 划分法等。

（1）轮转法。将前 N 个元组依次分配给 N 个磁盘后，再从第 N+1 个元组开始依次分配，如此往复至所有元组被分配完，相当于将所有元组进行了 N 等分划分。

轮转法是最简单的划分策略，它保证了数据均匀分布在所有磁盘上，极适合于进行全关系顺序扫描的查询应用，因为它的最大片段和最小片段至多相差一个元组。在这种查询应用下，并行扫描算法同时读取不同磁盘中的数据，各处理结点之间达到最大限度的负载平衡，从而使响应时间最短。但对于需要查询具有某一属性值的元组的应用，由于系统无法推算出数据的存放位置，还使用所有的处理机来处理此类查询，导致许多不必要的开销。

（2）Hash 法。在每个元组的某个属性（如主属性）上使用一个随机函数进行 Hash，根据 Hash 所得结果来决定元组所在的片段，属于同一片段的元组存放在同一个处理机结点上。

该方法可以通过设计一个优良的函数来达到数据分布比较均匀。

Hash 方法和值域划分法都依赖于属性的取值来决定片段，因而对于包含该属性的查询是十分高效的。例如，系统能够利用 Hash 码计算出该行位于哪个结点，并利用该结点上的局部索引快速访问到该行。

两种方法都可以将精确匹配(exact_match)类型查询或划分属性上的部分匹配(partial_match)查询直接引导到某一个（或几个）处理机结点磁盘上进行。

但对于选择谓词是划分属性上的一个区间查询，Hash 方法与轮转法一样，将使用所有的处理机来处理此类区间查询，导致开销增加。

（3）值域划分法。按照某个属性的取值范围将数据文件划分为 *N* 段，分别存放到 *N* 个处理机结点上。例如可以按职工名字的第 1 个字母划分职工表，A~E 放在磁盘 1 上，G~I 放在磁盘 2 上等。该方法极适合于在划分属性上进行范围查询，这时它可以跳过所有无关的数据片段，而直接访问与查询有关的片段。因为将查询的执行仅局限于包含相关元组的处理机上，省去了许多并行处理机的通信，启动和结束的开销，从而减少执行时间。

user-defined 划分法是按照用户定义的表达式对表进行划分。它使数据的划分更为灵活，可以满足不同用户的特殊应用需要。

schema 划分法是指不对表进行划分，以表为基本单位分布数据。这种方法适合于以查询为主很少进行修改的小表，如州代码表。将表放在一个磁盘上，不仅复制时效率高，而且维护副本的工作量小。

有关关系的划分信息被存放在数据字典中。例如，对于值域划分法和 Hash 法，将把划分属性名，划分属性的值域范围，Hash 函数定义等存放在数据字典中。

以上这些方法有各自的优点和弱点。其弱点如，轮转法不能很好支持具有低选择性谓词的查询，Hash 方法不能很好支持区间查询等。除此之外，一维数据划分方法具有一个共同的问题，它们不能有效地支持在关系的非划分属性上具有条件谓词的数据查询。这样的查询必须被送到所有存储操作关系的处理结点上进行处理。显然，把一个数据操作送到不存储所需元组的处理结点是完全没有必要的，必然浪费大量系统资源，降低系统的处理能力。为了解决这个问题，不少学者提出了一些多维数据划分方法，例如：

- CMD 多维数据划分法(黑龙江大学李建中教授提出)；
- BERD 多维数据划分法(S.Ghandeharizadeh 和 DeWitt 提出)；
- MAGIC 多维数据划分法。

然而，这些方法假定数据的分布是已知的。如果一个关系的分布不可预知，则这些方法仍有可能造成数据的分布不均而产生所谓的数据扭曲现象（Data Skew）。因此，人们又研究了动态数据划分方法。李建中教授等在这方面已做了一些工作，提出了 DCMD 方法。

15.5.2 并行数据库的数据复制

与数据分布相关联的一个复杂问题是数据复制。

数据库应用中系统的高可用性是十分重要的，并行数据库中由于结点失败，网络故障会引起数据或资源丢失。

对于结点失败有三种解决方法：数据冗余、资源冗余和恢复机制。

系统的高可用性可通过镜像和事务恢复来实现。镜像提供数据和资源冗余

的机制，事务恢复则保证了结点失败时数据的正确性。实现镜像有两种方法：基于硬件(磁盘)的方法和基于软件(数据块)的方法。

并行数据库借鉴了分布数据库中数据复制技术，以提高系统的可获得性。基本方法是维护同一数据的两个拷贝： 一个主拷贝和一个备份拷贝，这两个拷贝分别存放在不同的结点上。当一个结点失败时，其上的数据仍可通过另一个结点上的数据拷贝来获得。但是该结点就要承担原属于失败结点的那部分操作，它上面的负载相当于原来负载的 2 倍，从而破坏了负载平衡，这是并行数据库中需要特殊考虑的问题。另外，维护拷贝的一致性也是十分重要的。

总之在设计数据分布时必须考虑数据复制，而且应把它作为一个优化问题加以研究。

15.5.3 并行数据库的数据重组

通常，数据分布是通过一定的规则(例如在某属性值上使用 Hash 函数)来使每个结点上的数据负载与其处理能力相适应的。既便能够找到一个很好的数据划分算法使系统在初始情况下达到各结点间的负载平衡，但时间长了以后，随着数据的不断更新，这种初始的负载平衡必将被打乱，从而产生数据扭曲，这时就应当进行数据重组。

一般有静态重组和动态重组两种方法。静态重组是在某一段时间内，中断其他事务的执行进行数据重分布；动态重组是通过并行操作，联机的，不中断其他事务进行数据重分布。

现有的许多数据库系统一般使用静态重组。如何进行动态数据重组以及如何选择进行重组的时间仍是一个重要研究课题。

许多系统采取的策略是在系统运行的非高峰时间进行重组，因为数据重组本身的代价是巨大的，所以有时宁愿在数据扭曲的情况下继续执行操作而不愿进行重组。有的系统采取的方法就是首先估计数据重组的开销，然后再决定是进行重组还是继续忍受数据扭曲。

此外，重组应当对运行在并行数据库系统中已编译好的程序保持透明性，即程序不应由于重组而需要重新编译。因此已编译好的程序应当与数据位置保持独立性，这无疑给查询处理和查询优化器的设计增加了难度。

15.6 并行数据库系统与分布式数据库系统的区别

分布式数据库系统与并行数据库系统特别是与 SN 结构的并行数据库系统

具有很多相似点。它们都是用网络连接各个数据处理结点；整个网络中的所有结点构成一个逻辑上统一的整体；用户可以对各个结点上的数据进行透明存取等等。但是由于分布式数据库系统和并行数据库系统的应用目标和具体实现方法不同，使得它们具有很大的不同。

1. 应用目标不同

并行数据库系统的目标是充分发挥并行计算机的优势，利用系统中的各个处理机结点并行地完成数据库任务，提高数据库系统的整体性能。而分布式数据库系统主要目的在于实现场地自治和数据的全局透明共享，而不要求利用网络中的各个结点来提高系统处理性能（实际上，由于网络带宽的限制，这点也不容易做到）。

2. 实现方式不同

由于应用目标的差异，在具体的实现方法上，并行数据库系统与分布数据库系统也有着较大的不同 。

在并行数据库系统中，为了充分利用各个结点的处理能力，各结点间采用高速网络互连（结点间数据传输率可达 100MB/s 以上），结点间的数据传输代价相对较低，因此当某些结点处于空闲状态时，可以将工作负载过大的结点上的部分任务通过高速网传送给空闲结点处理，从而实现系统的负载平衡。

但是在分布式数据库系统中，为了适应应用的需要，满足部门分布的要求，各结点间一般采用局域网或广域网相连，网络带宽较低，点到点的通信开销较大，因此在查询处理时一般应尽量减少结点间的数据传输量。

3. 各结点的地位不同

在并行数据库系统中，不存在全局应用和局部应用的概念。各结点是完全非独立的，在数据处理中只能发挥协同作用，而不可能有局部应用。

而在分布式数据库系统中，各结点除了能通过网络协同完成全局事务外，更重要的一点还在于，各结点具有场地自治性。也就是说，每个场地是独立的数据库系统：它有自己的数据库，自己的客户，自己的 CPU，运行自己的 DBMS，执行局部应用，具有高度的自治性。

15.7 小 结

并行数据库系统把数据库技术与并行技术结合，发挥多处理机结构的优势，采用先进的并行查询技术和并行数据管理技术，其目标是提供一个高性能、高可用性、高扩展性的数据库管理系统，而在性能价格比方面，较相应大型机上的 DBMS 高得多。

并行数据库系统通过将数据库在多个磁盘上分布存储，可以利用多个处理机对磁盘数据进行并行处理，从而解决了磁盘“I/O”瓶颈问题。同样，通过开发查询间并行性、查询内并行性以及操作内并行性，大大提高查询效率。

并行数据库系统是新兴的数据库研究领域。国外已有一些并行数据库的原型系统如 Arbre，Bubba，Gamma，ERADAT 及 XPRS 等。

自并行数据库的概念提出以来，国外的许多研究机构相继研制出了多种结构平台上的并行数据库原型系统，其中比较著名的有 Tandem 的 NonStop SQL、Teradata 的 DBC/1012、加州 Berkeley 大学的 XPRS 系统、Colorado 大学的 Volcano 系统、Wisconsin 大学的 Gamma 等。这些系统探讨了并行数据库理论和实现中的若干重要问题，为并行数据库的进一步发展奠定了坚实的基础。与此同时，Teradata，Tandem 以及 Oracle，Sybase，Informix 等传统数据库厂商都正在或已经开发自己的并行数据库系统。

在“863 计划”的资助下，中国人民大学数据与知识工程研究所研制和开发了并行数据库系统 PBASE。PBASE/1 是 SM 结构的并行数据库原型系统；PBASE/2 是 SN 结构的。PBASE/2 系统立足于理论研究和技术攻关，并以我国的曙光并行系列机为基础，是自主研制的实用化的并行数据库管理系统。

PBASE/2 采用具有特色的并行查询技术来实现 SQL 功能。PBASE/2 提出了“流”(Stream)的重要思想，并将这一基本思想贯穿于 PBASE/2 整个查询处理过程中，设计和实现了一套完整的“流”的处理机制，该机制在不同平台上采用一组统一的函数调用接口，使系统具有很强的可扩展性。

PBASE/2 采用了先进的单进程多线索（在 SM 结构上）、多进程多线索（在 SN 结构）的体系结构。这种体系结构可以实现事务间并行、操作间独立并行、操作间流水线并行、操作内并行等多种粒度的并行性，达到并行数据库管理系统所追求的高事务吞吐量和低响应时间的目标。PBASE/2 实现了一个先进的并行查询处理器，包括：

（1）基于代价的查询优化器；

（2）灵活的并行查询执行引擎；

（3）高效的并行操作算法；

（4）轮转法、值域划分法和 Hash 三种数据分片方法。

并行数据库系统作为一个新兴的方向，需要深入研究的问题还很多，是目前数据库研究领域热门的课题之一。可以预见，由于并行数据库系统可以充分地利用并行计算机强大的处理能力，必将成为并行计算机最重要的支撑软件之一。

习 题

1. 什么是并行数据库系统?
2. 试述并行数据库系统的研制目标。
3. 什么是并行数据库系统的伸缩比和加速比?
4. 并行数据库系统有哪几种体系结构?试比较它们的特点。
5. 阐述并行数据库系统中并行粒度的概念。
6. 举例说明水平并行和垂直并行的概念。
7. 并行数据库系统中并行查询优化的必要性和困难何在?
8. 试述数据划分在并行查询处理中的重要性。
9. 并行数据库系统中有哪几种常用的数据划分方法?
10. 试述并行数据库系统与分布式数据库系统的区别。

本章参考文献

1 DeWitt D J, Gray J. Parallel Database Systems: The Future of High Performance Database Systems. Communications of The ACM, 35 (6): 85-98, June 1992.

(本文阐述了研究和开发并行数据库的必要性。)

2 Valduriez P. Parallel Database Systems: Open Problems and New Issues. Distributed and Parallel Databases, Vol.1, No.2, 1993

(本文阐述了研究和开发并行数据库的前景,以及需要研究的问题。)

3 Yu C, Meng W. Principles of Database Query Processing for Advanced Applications. Morgan Kaufmann Publishers, Inc. 1997

4 Valduriez Patrick, Zait Mohamed, Ziane Mikal. Designing an Optimizer for Parallel Relational Systems. Parallel Database Techniques, Chapter 1. IEEE Computer Society, 1998

(本文讨论了并行数据库系统的查询优化问题。)

5 Li Jianzhong, Srivastava, Rotem D. CMD: A Multi-dimensional Declustering Method for Parallel Database Systems. In Proceedings of VLDB, 1992

6 李建中. 并行数据库的查询并行化技术与物理设计方法. 软件学报, Vol.5, No.10, 1994

(本文系统地综述了目前并行数据库研究领域已经取得的重要研究成果,并提出了需要继续研究的问题。)

7 李建中. 并行数据操作算法和查询优化技术. 软件学报, Vol.5, No.10, 1994

8 Pirahesh H, et al. Parallelism in Relational Data Base Systems: Architectural Issues and Design Approaches. Proc. Second Int'l Symp. Databases in Parallel and Distributed Systems, 1990

(本文讨论了并行关系数据库系统的结构和设计方法。)

9 Stonebraker M，et al. The Design of XPRS. in Proceedings of VLDB，1988

（本文介绍了并行数据库原型系统 XPRS 的设计。）

10 DeWitt D J，et al. The Gamma Database Machine Project. IEEE Transactions on Knowledge and Data Eng.. March 1990

（本文介绍了并行数据库原型系统 Gamma 的系统结构。）

11 Boral Haran，et al. Prototyping Bubba: A Highly Parallel Database System. IEEE Knowledge and Data Engineering，Vol. 2，No. 1，March 1990

12 Query Processing in Parallel Relational Database systems. IEEE Computer Society Press，1994

（本文献集中了有关并行数据库系统的并行操作算法和并行查询处理的部分主要论文。）

13 文继荣. 并行数据库系统 PBASE/2 查询处理技术. 中国人民大学博士论文，1999，5

（该博士论文系统阐述了当前并行数据库的研究状况，深入探讨了并行查询处理中存在的若干关键理论问题，给出了一套完整而高效的并行查询处理方案，并且在中国人民大学研制的并行数据库系统 PBASE/2 中进行了具体实现。）

14 周胜，王珊. PBASE/2 全局系统资源信息管理系统的管理和利用. 计算机学报，1999

15 王珊，朱鸿雁，孟小峰. PBASE 的并行查询优化方案. 软件学报，1996.10

16 王珊，祝远玲，冯玉. PBASE 的并行查询处理策略. 软件学报，1996.10

17 王珊，陈红等. PBASE 体系结构与并行性. 计算机学报，Vol.19，No.3，1996

18 文继荣，陈红，王珊. Shared-nothing 并行数据库系统 PBASE/2 及其负载平衡策略. 97' 北京香港国际计算机会议，1997.10

19 陈红，文继荣，王珊. PQE：一个并行查询执行器. 第 16 届全国数据库会议论文集，1999

20 李建中，孙文隽. 并行关系数据库管理系统引论. 数据库丛书之一，北京：科学出版社，1998

（本书系统综述了并行数据库系统的基本概论和基本技术）

全书参考文献

1 萨师煊，王珊．数据库系统概论（第一版）．北京：高等教育出版社，1983
(本书于1987年获国家级优秀教材奖。)

2 萨师煊，王珊．数据库系统概论（第二版）．北京：高等教育出版社，1991

3 萨师煊，王珊．实用数据库系统汇编．北京：高等教育出版社，1990

4 王珊，陈红，文继荣．数据库和数据库管理系统．北京：电子工业出版社，1995

5 冯玉才．数据库基础（第一版）．武汉：华中理工大学出版社，1984

6 冯玉才．数据库基础（第二版）．武汉：华中理工大学出版社，1993

7 施伯乐，何继潮，崔靖．关系数据库的理论及应用．郑州：河南科技出版社，1990

8 俞盘祥，沈金发．数据库系统原理．北京：清华大学出版社，1988

9 石树刚，郑振楣．关系数据库．北京：清华大学出版社，1993

10 Date C J. An Introduction to Database System(Ed.6). Addison-Wesley, 1995
(本书是一本经典的英文教材，从1975年第一版起已修订到1995年的第六版。本书作者是一位著名的数据库专家，编著有数十本数据库领域的著作。)

11 Ullman J D. Principles of Database Systems.Computer Science Press, 1980

12 Yao S B. 赵延光，刑俊英等编译．数据库系统基础．计算机工程与应用．8~10，1981

13 Martin J. Principles of Data Base Management. 1976

14 Sibley E H. The Development of Data Base Technology. ACM Computing Surveys, Vol.8, No.1, 1976

15 Fry James P, Sibley Edgor H. Evolution of Data Base Management Systems. ACM Computing Surveys, Vol.8, No.1, 1976

16 Chamberlin Donald D. Relational Data Base Management Systems. ACM Computing Surveys, Vol.8, No.1, 1976

17 Taylor Robert W, Frank Randall L. CODASYL Data Base Management Systems. ACM Computing Surveys, Vol.8, No.1, 1976

18 Tsichritzis D C, Lachovsky F H. Hierarchical Data-base Management. ACM Computing Surveys, Vol.8, No.1, 1976

19 Michaels A S, Mittman B, Carlson C R. Relational and CODASYL Approaches. ACM Computing Surveys, Vol.8, No.1, 1976

20 Deen S M. Fundamentals of Data Base Systems. 1977

21 Martin J. Computer Data Base Organization. 1977

22 Dionysios Tsichritzis C. Data Base Management Systems. 1977

23 Wiederhold Gio. Data Base Design. McGraw-Hill, 1977, 1983

24 Cardenas A F. Data Base Management Systems. 1979, 1985

25 Bracchi G. Data Base Architecture. 1979

26 Brodie ML. On Conceptual Modeling. Springer Verlag, 1984

27 Korth H, Silberschatz A. Database System Concepts. McGraw-Hill, 1986

28 Ullman J D. Principles of Database and Knowledge-base Systems. Vol.I-II, Computer Science Press, Rockville, MD, 1988

29 Tsichritzis D C, Lochovsky F H. Data Models. Prentice-Hall, 1982

30 王珊，陈红. 数据库系统原理教程. 北京：清华大学出版社，1998
（本书是高等学校信息管理专业系列教材之一。）

31 王珊. 数据库系统基础. 北京：铁道出版社，1998

责任编辑　柳秀丽
封面设计　张　楠
责任绘图　黄建英
责任印制　宋克学