
数据库系统实用教程

徐洁磐 柏文阳 刘奇志

高等教育出版社

2006.01 宁

内容简介

本书是一本实用性数据库教材，它重点突出应用性与新技术，它将数据库基本原理、技术与应用三者结合于一体，系统性强、基本概念与原理讲述清楚、内容深入浅出、文字浅显易懂、并配有大量辅助性材料。

本书由六部分组成，它们是基本原理部分（第一章——第二章），关系数据库系统的原理与技术（第三章——第七章），数据库的设计与管理（第八章——第十章），新型数据库（第十一章——第十三章），数据库应用（第十四章——十六章）最后是数据库实验指导书。

本书可作为高等学校计算机应用类专业以及计算机应用相关专业的大学本科数据库课程教材，也可作为数据库应用开发人员的参考资料及相关培训教材。

前 言

数据库技术在我国已日渐普及，数据库应用领域也日益广泛，数据库不仅在传统的事务处理领域发挥重要作用，同时也在非传统领域应用中也起到越来越大作用。近年来，数据库与网络的结合，数据库在决策分析中的应用已成为目前发展的新的趋势。

目前市场上相关教材很多，但是由于教材需求层次多、类型广，因此需要有适应不同需求特色的教材，本教材的特色如下：

1. 应用性

目前我国学校中计算机专业大致分研究型与应用型两种，而近年来**应用型计算机**专业发展很快，同时与计算机应用紧密相关专业（如信息管理、软件工程、信息安全、电子商务、工程管理、统计、金融、自动控制、GIS、通信等）也飞跃发展，而市场上以计算机研究型教材为多见，而缺少此类应用型专业的数据库教材。本教材面向应用，以数据库基本原理及应用技术为主，适应应用型计算机专业以及与计算机应用紧密相关专业的数据库教学需要，可作为此种类型本科数据库教材。

2. 新技术

由于数据库技术发展较快，目前数据库教材由于更新周期长大都存在一定程度的技术滞后，它与目前我国计算机应用开发技术存在一定差距。本教材紧跟数据库应用技术新发展，能适应国内数据库应用中新技术发展需要。

3. 适合教学需要

本教材将原理、技术及应用三者有机结合，系统性强，基本概念与原理讲述清楚，内容深入浅出，文字浅显易懂并配有大量辅助性材料：

- (1) 本教材配有大量应用性习题，可帮助学生理解课程内容。
- (2) 为便于学生复习在每章后配有复习指导。
- (3) 为配合数据库实验需要，在教材中配有实验教导书。
- (4) 本教材并提供电子教案及其它参考资料书目。

在本书中以教材为主配合多种教学手段，构成一个适应计算机应用型教学需求的整体教学平台。

本书可作为高等院校**计算机应用型**专业及计算机应用相关专业的大学本科数据库课程教材，也可作为数据库应用开发人员的参考材料以及相关应用培训教材，本书难度适中，面向基本原理、基本技术、面向应用，特别是新技术应用。

从内容上看，本书由下面几部分组成：

(1) 数据库基本概念与原理：由第一、二章组成，主要讲述数据库系统的最基本的概念及数据模型，它是本书的核心。

(2) 关系数据库系统的原理与技术：由第三章——第七章组成，主要介绍关系数据库系统基本理论、原理、操作及技术，它是本书的主要内容。

(3) 关系数据库的设计与管理：由第八章——第十章组成，主要介绍关系数据库的设计的理论、技术以及关系数据库系统的管理，它是开发应用的基础。

(4) 新型数据库：由第十一章——第十三章组成，主要介绍几种新模型的数据库系统以及分布式数据库与 Web 数据库。

(5) 数据库应用：由第十四章——第十六章组成，主要介绍数据库在事务处理、非事务处理领域及分析领域的应用。

(6) 实验指导书：在附录中给出，共六个实验及一个实验总结指导，它为全书提供实验环节指导。

本教材可为两种不同应用型专业的本科学生作教材使用，一种是高级计算机应用型专业，可使用本教材全部内容；另一种是普通计算机应用型专业，它可使用本教材的部分内容，在本书中凡带有“*”号的章节均可删减，它包括第七章、第十一章、第十二章以及第十五章等四章内容及 2.3.4、2.4.4 节等两节内容。

值本书付梓之际，作者首先要感谢山东大学董继润教授，他为审阅本书付出艰辛的劳动并提出了很多宝贵的意见，同时感谢南京大学计算机软件新技术重点实验室及费翔林教授为本书出版所作的支持，最后，在本书编写过程中得到了南京大学计算机科学与技术系多位老师的支持和帮助以及陈巧珍老师的具体帮助，在此一并表示感谢。

由于作者水平所限，书中错误与缺点在所难免，恳切希望读者批评指正。

作者

南京大学计算机软件新技术国家重点实验室

南京大学计算机科学与技术系

2006.01.宁

目 录

第一章 数据库系统概述.....	1
1.1 基本概念.....	1
1.2 数据库系统的发展及当前主流.....	7
1.3 数据库系统的特点.....	9
1.4 数据库内部结构体系.....	10
1.4.1 数据库三级模式.....	10
1.4.2 数据库两级映射.....	11
习题一.....	12
第一章复习指导.....	13
第二章 数据模型.....	15
2.1 数据模型的基本概念.....	15
2.2 数据模型的四个世界.....	15
2.3 概念世界与概念模型.....	16
2.3.1 E-R 模型.....	16
2.3.2 扩充的 E-R 模型——EE-R 模型.....	21
2.3.3 面向对象模型.....	22
*2.3.4 谓词模型.....	24
2.4 信息世界与逻辑模型.....	29
2.4.1 概述.....	29
2.4.2 关系模型与关系模型数据库管理系统.....	29
2.4.3 面向对象模型与面向对象数据库管理系统.....	34
*2.4.4 谓词模型及知识库系统.....	35
2.5 计算机世界与物理模型.....	37
2.5.1 计算机中的磁盘.....	37
2.5.2 文件系统.....	38
2.5.3 逻辑模型的物理存储结构.....	39
习题二.....	39
第二章复习指导.....	41
第三章 关系数据库系统.....	42
3.1 关系数据库系统概述.....	42
3.2 关系数据库系统的衡量准则.....	43
3.3 关系模型数学理论——关系代数.....	44
3.3.1 关系的表示.....	44
3.3.2 关系操纵的表示.....	45
3.3.3 关系模型与关系代数.....	47
3.3.4 关系代数中的扩充运算.....	48
3.3.5 关系代数实例.....	51

3.4 关系数据库语言 SQL'92.....	53
3.4.1 SQL 概貌.....	53
3.4.2 SQL 数据定义功能.....	55
3.4.3 SQL 数据操纵功能.....	58
3.4.4 SQL 的更新功能.....	69
3.4.5 视图.....	70
习题三.....	72
第三章复习指导.....	76
第四章 数据库的安全性完整性保护.....	78
4.1 数据库的安全性保护.....	78
4.1.1 数据库的安全与安全数据库.....	78
4.1.2 数据库安全的基本概念与内容.....	79
4.1.3 数据库的安全标准.....	81
4.1.3 SQL 对数据库安全的支持.....	83
4.2 数据库的完整性保护.....	85
4.2.1 数据库完整性保护的功能.....	85
4.2.2 完整性规则的三个内容.....	85
4.2.3 完整性约束的设置、检查与处理.....	86
4.2.4 触发器.....	88
习题四.....	89
第四章复习指导.....	91
第五章 事务处理、并发控制与故障恢复技术.....	92
5.1 事务处理.....	92
5.1.1 事务.....	92
5.1.2 事务的性质.....	93
5.1.3 事务活动.....	93
5.1.4 有关事务的语句.....	93
5.2 并发控制技术.....	94
5.2.1 事务的并发执行.....	94
5.2.2 封锁.....	97
5.2.3 封锁协议.....	98
5.2.4 两阶段封锁协议.....	99
5.2.5 封锁粒度.....	100
5.2.6 活锁与死锁.....	101
5.3 数据库恢复技术.....	101
5.3.1 概述.....	101
5.3.2 数据库故障分类.....	102
5.3.3 数据库故障恢复三大技术.....	102
5.3.4 恢复策略.....	103

习题五.....	104
第五章复习指导.....	105
第六章 数据库中的数据交换.....	106
6.1 概述.....	106
6.1.1 数据交换模型.....	106
6.1.2 数据交换的五种方式.....	106
6.2 数据交换的管理.....	108
6.2.1 会话管理.....	108
6.2.2 连接管理.....	110
6.2.3 游标管理.....	110
6.2.4 诊断管理.....	111
6.2.5 动态 SQL.....	111
6.3 数据交换的流程.....	112
6.4 数据交换的四种方式.....	113
6.4.1 嵌入式 SQL.....	113
6.4.2 自含式 SQL.....	115
6.4.3 调用层接口.....	118
6.4.4 Web 方式.....	119
习题六.....	120
第六章复习指导.....	121
*第七章 数据库的物理组织.....	122
7.1 概论.....	122
7.2 数据库的物理存储介质.....	122
7.3 磁盘存储器及其结构.....	123
7.4 文件组织.....	125
7.4.1 文件记录与磁盘块.....	125
7.4.2 文件的定长记录与变长记录.....	126
7.5 文件记录组织.....	126
7.6 索引技术与散列技术.....	127
7.7.1 索引技术.....	128
7.7.2 索引技术中的 B ⁺ 树.....	131
7.7.3 散列技术.....	133
7.7 数据库与文件.....	134
7.7.1 数据库中数据分类.....	134
7.7.2 数据库存储空间组织.....	135
习题七.....	136
第七章复习指导.....	137
第八章 关系数据库规范化理论.....	138
8.1 概述.....	138

8.2	规范化理论.....	140
8.2.1	函数依赖.....	140
8.2.2	与函数依赖有关的范式.....	143
8.2.3	多值依赖与第四范式.....	147
8.2.4	小结.....	150
8.3	规范化所引起的一些问题.....	150
8.4	关系数据库规范化的非形式化判别法.....	151
	习题八.....	151
	第八章复习指导.....	153
第九章	数据库设计.....	154
9.1	数据库设计概述.....	154
9.2	数据库设计的需求分析.....	155
9.2.1	需求调查.....	155
9.2.2	需求分析.....	155
9.2.3	数据需求分析说明书.....	157
9.3	数据库的概念设计.....	159
9.3.1	数据库概念设计概述.....	159
9.3.2	数据库概念设计的过程.....	159
9.3.3	数据库概念设计说明书.....	163
9.4	数据库的逻辑设计.....	164
9.4.1	数据库逻辑设计基本方法.....	164
9.4.2	关系视图设计.....	166
9.4.3	数据库逻辑设计说明书.....	167
9.5	数据库的物理设计.....	167
9.5.1	存取方法设计.....	168
9.5.2	存贮结构设计.....	169
9.5.3	数据库物理设计说明书.....	169
	习题九.....	170
	第九章复习指导.....	172
第十章	数据库管理.....	174
10.1	数据库管理概述.....	174
10.2	数据库管理的内容.....	174
10.3	数据库管理员 DBA.....	178
	习题十.....	179
	第十章复习指导.....	180
	参 考 文 献.....	181

第一章 数据库系统概述

本章主要介绍数据库系统的基本概况，包括基本概念、特点与基本内容，本章对全书内容具有提纲契领的作用。

1.1 基本概念

计算机科学与技术的发展，计算机应用的深入与拓展，使得数据库在计算机领域中的地位与作用日益重要，它在商业中、事务处理中占有主导地位，近年来在工程领域、多媒体领域等非事务处理领域中以及分析领域中的地位与作用也变得十分重要。随着网络应用的普及，它在网络中的应用也日渐重要，因此，数据库已成为构成一个计算机应用系统的重要支撑。本书以数据库为核心对其基本原理、应用及新技术作全面的介绍。

首先，本节将对与数据有关的六个基本概念作介绍，它们是数据、数据库、数据库管理系统、数据库管理员、数据库系统及数据库应用系统。

1. 数据 (data)

1) 什么是数据

数据是现实世界中客体在计算机中的抽象表示，具体的说，它是一种存储于计算机内的符号串。

2) 数据的特性

数据有下面五个特性：

(1) 数据表现形式的多样性

从表现形式看，数据表现的形式很多，除常用的数字、文字、时间等表示形式外，还包括图像、图形、语言、视频等多媒体数据以及表示知识、规则、数学符号及推理等抽象数据，数据表现形式多样性为数据的应用提供了有力的基础。

(2) 数据的可构造性

从结构看，数据分为结构化数据 (structured data)、半结构化数据 (semistructred data) 与非结构化数据 (non-structured data)。所谓非结构化数据即表示符号串是不规则结构形式，所谓半结构化数据即表示符号串呈半规则结构形式，如文件中之流式文件，如互联网中的 Web 结构等均属非结构化及半结构化形式。而在软件中的数据大多是有结构的，它们称结构化数据。首先，结构化数据有型 (type) 与值 (value) 之分，数据的型给出了数据表示的类型如整型、实型、字符型等，而数据的值给出了符合给定型的数值。随着应用需求的扩大，数据的型有了进一步的扩大，它包括了将多种相关数据以一定结构方式组合构成特定的数据框架称为数据结构 (data structure)，具有统一结构形式的数据结构的具体描述可称为数据模式 (data schema)。

(3) 数据的挥发性与持久性

从存储时间看，数据一般分为两部分，其中一部分与程序仅有短时间的交互关系，随着程序的结束而消亡，它们称为临时性数据或挥发性 (transient) 数据。这类数据一般存放于计算机内存中；而另一部分数据则对系统起着长期持久的作用，它们称为持久性 (persistent) 数据，这类数据一般存放于计算机中的次级存储器内 (如磁盘)。

(4) 数据的私有性与共享性

从其使用对象看，数据可分为私有性与共享性两种。为特定应用（程序）服务的数据称私有（private）数据，而为多个应用（程序）服务的数据则称为共享（share）数据。

（5）数据的海量性

从其存储数量看，数据可分为小量、大量及海量三种。数据的量是衡量与区别数据的重要标志，这主要是由于数据“量”的变化可能会引起数据“质”的变化。数据量由小变大后，数据就需要进行管理，需要保护与控制。目前数据以海量数据为多见，因此一般数据均需管理、保护与控制。

随着技术的进步与应用的扩大，数据的特性都在发生变化，这些变化主要表现为：

- （1）数据的量由小量到大量进而到海量；
- （2）数据的组织由非结构化到结构化；
- （3）数据的服务范围由私有到共享；
- （4）数据的存储周期由挥发到持久。

数据的这些变化使得现代数据具有以海量的、结构化的、持久的和共享的特点，本书如不作特别说明，所提数据即具此四种特性。

下面我们讨论数据与其它几个重要概念间的关系，它们是数据与软件及数据与信息间的关系。

3）数据与软件（software）

首先讨论软件，我们说软件是计算机科学与技术的一大门类，它是建立在计算机硬件之上的一种运行（或处理）实体。软件一般由程序与数据两部分组成，其中程序给出了运行的过程而数据则给出了运行的对象与结果。

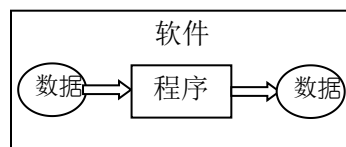


图 1.1 程序与数据关系示意图

在软件中数据（主要指其结构）是其最稳定部分，而程序则是可变部分，因此数据称为软件中的不动点（fixed point），它在软件中起着基础性的作用。

软件发展至今，程序与数据间的不同关系形成了目前流行的两种结构方式：

（1）以程序为中心的结构：在此种软件结构中以程序为中心以数据为辅助，即每个程序有若干个数据为其支撑，它们构成了如图 1.2（a）所示结构。

（2）以数据为中心的结构：在此种软件结构中以数据为中心以程序为辅助，即以一个数据集合为中心围绕它有若干个程序对数据作处理，它们构成了如图 1.2（b）所示结构。

在目前，大多数软件结构采用以数据为中心的结构。

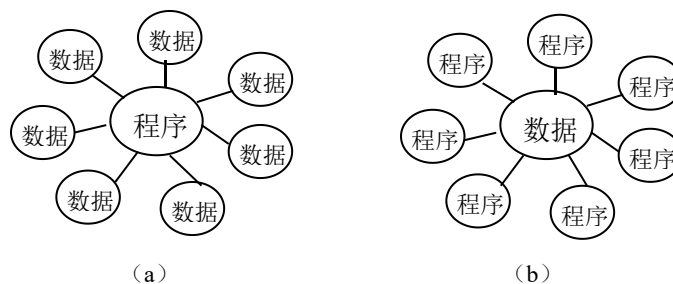


图 1.2 软件两种结构图

在过去，软件是以程序为中心，而数据则以私有形式从属于程序。在此种系统中，数据是分散、凌乱的，它造成了数据管理的混乱，如数据冗余大、一致性差、结构复杂等多种弊病，但经过若干年的发展，数据在软件中的地位和作用发生了本质的变化，在软件中它已占主体地位，而程序则已退居附属地位，它们构成了以数据为中心的结构。在此种结构中，需要对数据作集中、统一的管理，并使其为多个应用程序共享，它们构成了如图 1.3 所示的结构图，这种结构方式为数据库系统的产生与发展奠定了基础。

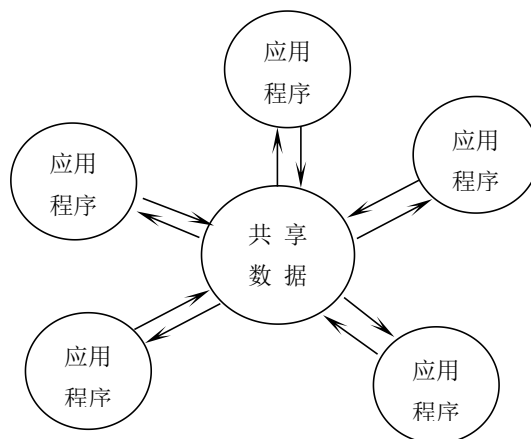


图 1.3 以数据为主体的软件系统

4) 数据与信息 (information)

数据与信息是两种不同的概念，在实际应用中往往引起混淆，我们说数据是指客体在计算机中表示的形式而信息则是客体的语义表现。数据与信息表示了一个事物的两个不同方面的理解。

2. 数据库 (database, 简称 DB)

数据库是**数据的集合**，它具有统一的结构形式，存放于统一的存贮介质内，并由统一机构管理，它由多种应用数据集成，并可被应用所共享。

数据库存放数据，数据按所提供的数据模式存放，它能构造复杂的数据结构以建立数据间内在联系与复杂关系，从而构成数据的全局结构模式。

数据库中的数据具有“集成”、“共享”的特点，亦即是数据库集中了各种应用的数据，并对其进行统一的构造与存储，而数据可为不同应用服务与使用。

3. 数据库管理系统 (database management system, 简称 DBMS)

数据库管理系统是统一**管理数据库的一种软件** (属系统软件)，它负责：

- 数据库中的数据组织。
- 数据库中的数据操纵。
- 数据库中的数据维护。
- 控制及保护数据不受破坏。
- 数据库中的数据交换。
- 数据库中的数据服务。
- 数据字典

数据库中的数据是具有海量级的数据，且其结构复杂，因此是需要管理的，它主要有如下几方面功能：

(1) 数据模式定义

数据库管理系统负责为数据库构造模式，亦即是它为数据库构造其统一数据框架。

(2) 数据存取的物理构造

数据库管理系统负责为数据模式的物理存取构造有效的存取方法与手段，如构造索引(index)、集簇(cluster)及分区(partition)等。

(3) 数据操纵

数据库管理系统为用户使用数据提供方便，它一般提供数据查询、插入、修改以及删除的功能，此外，它自身还具有一定的运算、转换及统计的能力。它还可以有一定的过程调用能力。

(4) 数据的完整性、安全性定义与检查

数据库中的数据具有内在语义上的关联性与一致性，它们构成了数据的完整性。数据的完整性是保证数据库中数据正确性的必要条件，因此必须经常检查以维护数据的正确。

数据库中的数据具有共享性，而数据共享可能会引发数据的非法使用，因此必须要对数据正确使用作出必要的规定，并在使用时作检查，这就是数据的安全性。

数据库的完整性与安全性的维护是数据库管理系统的基本职能。

(5) 数据的并发控制与故障恢复

数据库是一个集成、共享的数据集合体，它能为多个应用服务，因此就存在着多个应用对数据库的并发操作，而在并发操作中由于多个应用间的相互干扰会对数据库中的数据造成破坏，因此，数据库管理系统必须对多个应用的并发操作作必要的控制以保证数据不受破坏，这就是数据的并发控制。

数据库中的数据是重要的信息财富，数据库管理系统有责任保护它并在它遭受破坏后有能及时恢复，这就是数据的故障恢复。

(6) 数据交换

数据库中的数据需要与外界数据主体作数据交换，这种外界数据主体可以是操作员，应用程序也可以是另一种数据体，作为一种扩充，数据库管理系统提供数据交换管理功能。

(7) 数据的服务

数据库管理系统提供对数据库中数据的多种服务功能，如数学函数、输入/出函数、数据转换函数、日期函数等，此外还提供如数据拷贝、转储、重组、性能监测、分析等服务功能。

(8) 数据字典

数据字典是一组关于数据的数据又称元数据(metadata)，它存放数据库管理系统中的数据模式结构，数据完整性规则、安全性要求等数据。数据字典是数据库管理系统中的一个专门的系统数据库，它具有固定的模式结构，称信息模式(information schema)，用户可用查询语言对其作操作，以获取数据库的结构性信息。

为完成以上八个功能，数据库管理系统一般提供相应的数据语言(data language)：

(1) 数据定义语言(data definition language, 简称 DDL)。该语言负责数据的模式定义与数据的物理存取构造。

(2) 数据操纵语言 (data manipulation language, 简称 DML)。该语言负责数据的操纵, 包括查询及增、删、改等操作, 此外还包括数据交换等操作。

(3) 数据控制语言 (data control language, 简称 DCL)。该语言负责数据完整性、安全性的定义与检查以及并发控制、故障恢复等功能。

以上三种语言都是非过程性语言, 它们可以有多种表示形式。

此外, 数据库管理系统还包括为用户提供服务的服务 (utility) 软件, 如提供程序包或函数库、类库等。

4. 数据库管理员 (database administrator, 简称 DBA)

由于数据库的共享性, 因此对数据库的规划、设计、维护、监视需要有专人管理, 它们称为数据库管理员, 其主要工作如下:

(1) 数据库的建立与调整

DBA 的主要任务之一是在数据库设计基础上进行数据模式的建立, 同时进行数据加载, 此外在数据库运行过程中还需对数据库进行调整、重组及重构以保证其运行的效率。

(2) 数据库维护

DBA 必须对数据库中数据的安全性、完整性、并发控制及系统恢复进行实施与维护。

(3) 改善系统性能, 提高系统效率

DBA 必须随时监视数据库运行状态, 不断调整内部结构, 使系统保持最佳状态与最高效率。

5. 数据库系统 (database system, 简称 DBS)

数据库系统由五个部分组成:

- 数据库 (数据)
- 数据库管理系统 (软件)
- 数据库管理员 (人员)
- 系统平台之一——硬件平台 (硬件)
- 系统平台之二——软件平台 (软件)

这五个部分包括数据、软件、硬件及人员等, 它们构成了一个以数据库为核心的完整的运行实体, 称为数据库系统。

在数据库系统中其硬件平台包括以下两类:

(1) **计算机**。它是系统中硬件的基础平台, 且前常用的有微型机、小型机、中型机、大型机及巨型机。

(2) **网络**。过去数据库系统一般建立在单机上, 但是近年来它较多地建立在网络上, 包括局域网、广域网及互联网, 而其结构形式又以客户/服务器 (C/S) 方式与浏览器/服务器 (B/S) 方式为主, 并逐步走向三层结构体系。

在数据库系统中其软件平台包括三类:

(1) **操作系统**。它是系统的基础软件平台, 目前常用的有 Windows 与 Unix (包括 Linux) 两种。

(2) **数据库系统开发工具**。为开发数据库应用所提供的工具, 它包括过程性程序设计语言, 如 C、C++ 等, 也包括可视化开发工具 VB、PB、Delphi 等, 它还包括近期与互联网有关的 ASP、JSP、PHP、HTML 及 XML 等以及一些专用开发工具。

(3) 中间件。在网络环境下数据库系统中数据库与应用间需要有一个提供标准接口与服务的统一平台,它们称为中间件(middle ware),目前使用较普遍的中间件有微软的.NET、ODMG 的 CORBA 以及 SUN 公司的 J2EE 等。它们为支持数据库应用开发、为方便用户使用提供了基础性的服务。

6. 数据库应用系统(database application system, 简称 DBAS)

利用数据库系统作应用开发可构成一个数据库应用系统,数据库应用系统是由数据库系统加上应用软件、应用界面以及用户这四部分组成,具体内容包括:

- 数据库
- 数据库管理系统
- 数据库管理员
- 系统平台之一——硬件平台
- 系统平台之二——软件平台
- 应用软件
- 应用界面
- 用户

其中应用软件是由数据库系统所提供的数据库管理系统(软件)及数据库系统开发工具书写而成,应用界面大都由相关的可视化工具开发而成,而用户则是应用系统使用者。这三部分与应用关系密切。

数据库系统的五部分与数据库应用系统的三个部分以一定的逻辑层次结构方式组成一个以数据库系统为核心的应用实体,其层次结构示意图如图 1.4 所示。

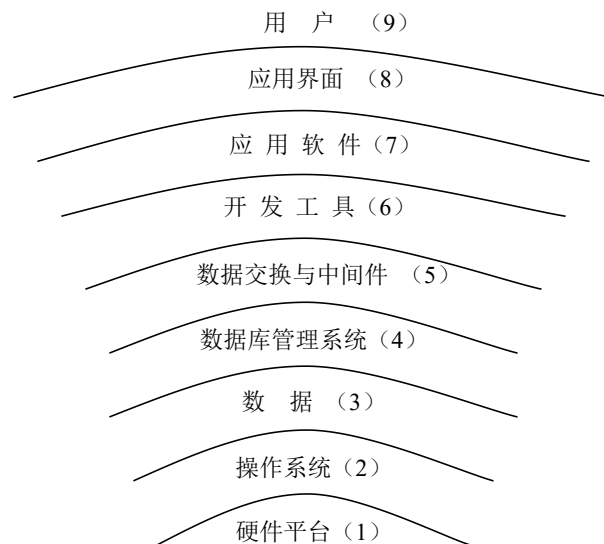


图 1.4 数据库应用系统层次结构示意图

目前大量的流行的应用系统即属此种系统,它一般也可称信息系统(Information System)。其典型例如管理信息系统(MIS)、企业资源规划(ERP)、办公自动化系统(OA)、情报检索系统(IRS)、客户关系管理(CRM)、财务信息系统(FIS)等事务处理系统。

1.2 数据库系统的发展及当前主流

数据库系统的发展至今已近 50 年，随着计算机应用领域不断扩大，它的功能、适应范围也愈来愈广，到目前已成为计算机系统的基本及主要的支撑软件，它的发展经历了若干个阶段，其主要发展历史可用图 1.5 表示。

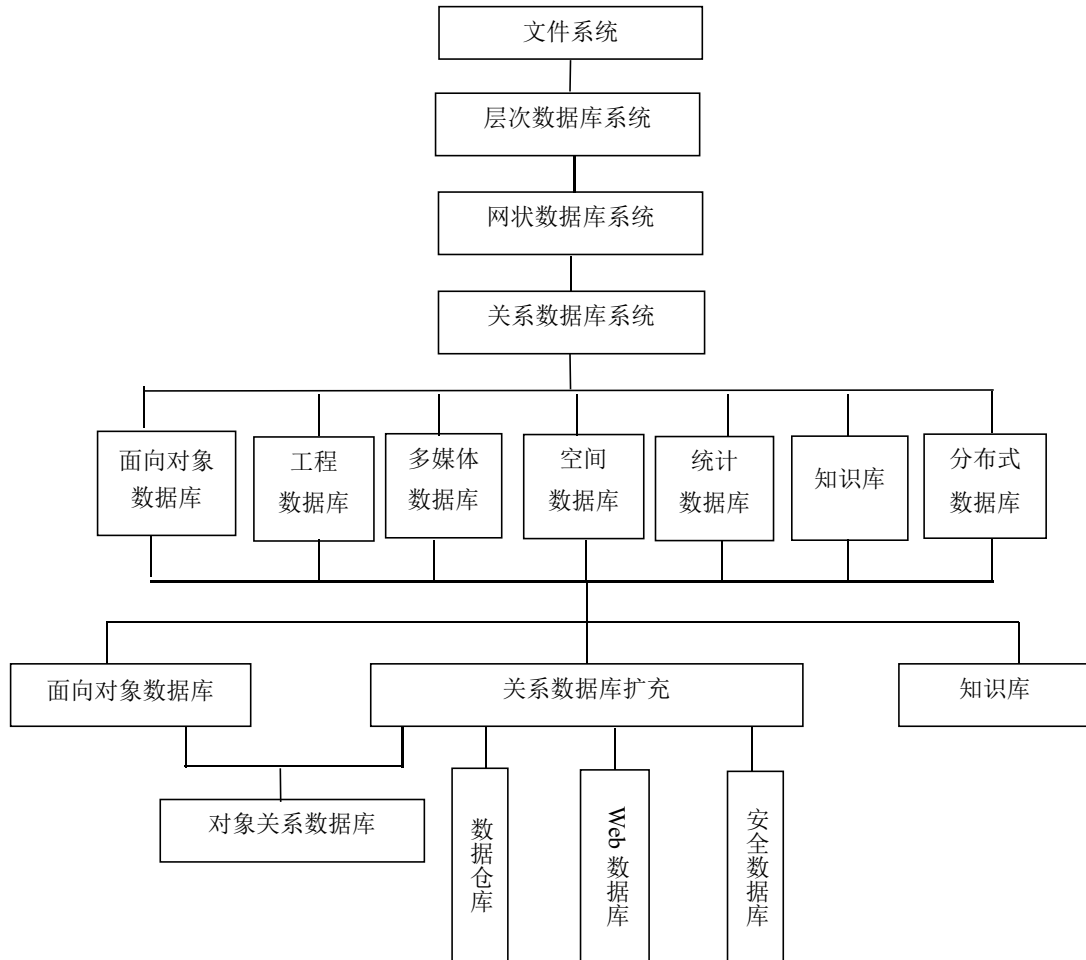


图 1.5 数据库系统发展简图

下面就数据库系统发展的几个阶段作介绍。

1. 文件系统阶段

文件系统是数据库系统发展的初级阶段，它出现于 20 世纪 50 年代末，它提供了简单的数据共享与数据管理能力，但是它无法提供完整统一的管理与数据共享能力。由于它的功能简单，因此它只能附属于操作系统而不能成为独立的软件，目前一般将其看成仅是数据库系统的雏形，而不是真正的数据库系统。

2. 层次数据库与网状数据库系统阶段

20 世纪 60 年代末，真正的数据库系统：层次数据库与网状数据库开始发展，它们为统一管理与共享数据提供了有力支撑，这个时期数据库系统蓬勃发展形成了有名的“数据库时代”。但是这两种系统也存在不足，主要是它们脱胎于文件系统，受文件的物理影响较大，给数据库使用带来诸多不便，同时，此类系统的数据模式构造繁琐难以推广使用。

3. 关系数据库系统阶段

关系数据库系统出现于 20 世纪 70 年代，并在 80 年代得到蓬勃发展，并逐渐取代前两种系统。关系数据库系统结构简单、使用方便、逻辑性强、物理性少，因此在 80 年代以后一直占据数据库领域的主导地位。但是由于此系统来源于商业应用，适合于事务处理领域而在非事务处理领域的应用受到限制，于是在 80 年代末兴起专用数据库系统如下。

- (1) 工程数据库系统：是数据库与工程领域的结合。
- (2) 多媒体数据库系统：是数据库与多媒体应用的结合。
- (3) 空间数据库系统：是数据库与 CAD/CAM 应用的结合。
- (4) 统计数据库系统：是数据库与统计应用的结合。
- (5) 知识库系统：是数据库与人工智能应用领域的结合。
- (6) 分布式数据库系统：是数据库与网络应用的结合。
- (7) 面向对象数据库系统：是数据库与面向对象方法的结合。

这些专用数据库的兴起都是为弥补关系数据库在非事务处理中的不足而产生的，它们的出现带来了数据库发展的百花齐放和又一个高潮的出现。

在 90 年代初、中期面对多种专用数据库的出现产生了一系列问题，最主要的是这些数据库系统专用性有余而通用性不足，因此严重影响其推广使用，人们又一次进行反思，认为发展通用性数据库系统是数据库发展的必然之路，因此将研究与发展重点集中于具有通用性的三个数据库系统，它们是：

(1) 面向对象数据库系统 用面向对象方法构筑面向对象数据模型使其具有比关系数据库系统更为通用的能力。

(2) 知识库系统 用人工智能中的方法特别是用谓词逻辑知识表示方法构筑数据模型，使其模型具有特别通用的能力。

(3) 关系数据库系统扩充 利用关系数据库作进一步扩展，使其在模型的表达能力与功能上有进一步的加强。

关系数据库系统在 20 余年间经历了“合久必分，分久必合”的复杂过程。

4. 新一代数据库系统阶段

进入 21 世纪，数据库系统的发展集中于对关系数据库系统的进一步扩充与改造，由于面向对象数据库系统与知识库系统在发展过程中遇到了算法与应用上的困难，因此，进一步改造关系数据库系统并扩充其功能成为近年来的主要方向，它主要表现在如下几个方面：

(1) 对象关系数据库系统

在关系数据库系统之上增加面向对象的若干功能，使系统在保留关系模型的基础上又具有一定面向对象功能，这是目前关系数据库发展的主要方向。目前主要商用数据库产品如 Oracle, Sybase, DB2 等都具有一定面向对象的功能，此种系统称为对象关系数据库系统。

(2) 数据仓库

关系数据库系统的进一步发展由传统事务性处理向决策、统计型发展从而出现了数据仓库，这是关系数据库系统功能上的进一步扩充。

(3) Web 数据库

由于互联网的发展，关系数据库系统与互联网结合成为重要方向，因此出现了 Web 数据库等基于网络的数据库系统。

(4) 安全数据库

数据库与网络的紧密结合以及网上黑客的横行使得数据库的安全变得特别重要，因此出现了安全数据库，它已成为网络时代信息安全重要内容之一。

以上四个方向构成了新一代数据库发展的主流。在本书中将介绍图 1.5 所列的所有数据库系统（个别除外）。

1.3 数据库系统的特点

数据库系统有很多特点，下面就几个基本特点作介绍。

1. 数据的集成性

数据库系统的数据集成性主要表现在如下几个方面：

(1) 在数据库系统中采用统一的数据结构方式，如在关系数据库中采用二维表统一结构方式。

(2) 在数据库系统中按照多个应用的需要组织全局的统一的数据结构，称为数据模式。数据模式不仅可以建立全局的数据结构，还可以建立数据间的语义联系，从而构成一个内在紧密联系的数据整体。

(3) 数据库系统中的数据模式是多个应用共同的、全局的数据结构，而每个应用的数据则是全局结构中的一部分，称为局部结构，这种全局与局部的结构模式构成了数据库系统数据集成性的主要特征。

2. 数据的高共享性与低冗余性

由于数据的集成性使得数据可为多个应用所共享，而数据的共享又可极大地减少数据的冗余性，它不仅可以减少不必要的存储空间，更为重要的是可以避免数据的不一致性。

所谓数据的一致性即是在系统中同一数据的不同出现应保持相同的值；而数据的不一致性指的是同一数据在系统的不同拷贝处有不同的值。数据的不一致性会造成系统的混乱，因此，减少冗余性避免数据的不同出现是保证系统一致性的基础。

数据的共享不仅可以为多个应用服务，还可以为不断出现的新的应用提供服务，特别是在网络发达的今天，数据库与网络的结合扩大了数据关系的范围，使数据信息这种财富可以发挥更大的作用。

3. 数据独立性

数据独立性即是数据库中数据独立于应用程序而不依赖于应用程序，也就是说数据的逻辑结构、存储结构与存取方式的改变不影响应用程序。

数据独立性一般分为物理独立性与逻辑独立性两级。

(1) 物理独立性是指数据的物理结构（包括存储结构、存取方式等）的改变，如存储设备的更换、物理存储的更换、存取方式的改变等都不影响数据库的逻辑结构，从而不致引起应用程序的变化。

(2) 逻辑独立性是指数据库总体逻辑结构的改变, 如修改数据模式、增加新的数据类型、改变数据间联系等, 不需要相应修改应用程序, 这就是数据的逻辑独立性。遗憾的是到目前为止数据逻辑独立性还无法作到完全的实现。

总之, 数据独立性即是数据与程序间的互不依赖性。一个具有数据独立性的系统可称为以数据为中心的系统或称为面向数据的系统。

4. 数据统一管理与控制

数据库系统不仅为数据提供高度集成环境, 同时还为数据提供统一管理的手段, 这主要表现为:

- (1) 为数据查询及增、删、改提供统一的服务。
- (2) 数据的完整性检查: 对数据库中数据的正确性作检查以保证数据的正确。
- (3) 数据的安全性保护: 对数据库访问者作检查以防止非法访问。
- (4) 并发控制: 对多个应用并发访问所产生的相互干扰作控制以保证其正确性。
- (5) 数据库故障恢复: 使遭受破坏的数据具有恢复能力, 使数据库具有抗破坏性。

1.4 数据库内部结构体系

数据库在构造时其内部具有三级模式及二级映射, 它们分别是概念模式、内模式与外模式, 其映射则分别是概念到内模式的映射以及外模式到概念模式的映射, 这种三级模式与二级映射构成了数据库内部的抽象结构体系, 如图 1.6 所示。

1.4.1 数据库三级模式

数据模式是数据库中数据结构的具体表示与描述, 它具有不同的层次与结构方式。数据库三级模式结构最早是在 1971 年由 DBTG 给出, 1975 年列入美国 ANSI/X 3/SPARC 标准, 它是一种数据库内部抽象结构体系并具有对构造系统的理论指导价值, 这三级模式结构如下。

1. 概念模式 (conceptual schema)

概念模式是数据库中全局数据逻辑结构的描述, 是全体用户 (应用) 公共数据视图, 此种描述是一种抽象的描述, 它不涉及具体的硬件环境与平台, 也与具体的软件环境无关。

概念模式主要描述数据的概念记录类型以及它们间的关系, 它还包括一些数据间的语义约束, 对它的描述可用 DBMS 中的 DDL 语言定义。

2. 外模式 (external schema)

外模式也称子模式 (subschema) 或称用户模式 (user's schema), 它是用户的数据视图, 亦即是用户所见到的模式, 它由概念模式推导而出, 概念模式给出了系统全局的数据描述而外模式则给出每个用户的局部描述。一个概念模式可以有若干个外模式, 每个用户只关心与它有关的模式, 这样可以屏蔽大量无关信息且有利于数据保护, 因此对用户极为有利。在一般的 DBMS 中都提供有相关的外模式描述语言 (外模式 DDL)。

3. 内模式 (internal schema)

内模式又称物理模式 (physical schema), 它给出了数据库物理存储结构与物理存取方法, 如数据存储的文件结构、索引、集簇及 hash 等存取方式与存取路径, 内模式的物理性主要体现在操作系统及文件级上, 它还不深入到设备级上 (如磁盘及磁盘操作), 但近年来

有向设备级发展的趋势（如原始磁盘、磁盘分块技术等），DBMS 一般提供相关的内模式描述语言（内模式 DDL）。

数据模式给出了数据库的数据框架结构，而数据库中的数据才是真正的实体，但这些数据必须按框架描述的结构组织，以概念模式为框架组成的数据库叫概念数据库（conceptual database），以外模式为框架所组成的数据库叫用户数据库（user's database），以内模式为框架组成的数据库叫物理数据库（physical database），这三种数据库中只有物理数据库是真实存在于计算机外存中，其他两种数据库并不真正存在于计算机中，而是通过两种映射由物理数据库映射而成。

模式的三个级别层次反映了模式的三个不同环境以及它们的不同要求，其中内模式处于最低层，它反映了数据在计算机物理结构中的实际存储形式；概念模式处于中层，它反映了设计者的数据全局逻辑要求；而外模式处于最外层，它反映了用户对数据的要求。

1.4.2 数据库两级映射

数据库三级模式是对数据的三个级别抽象，它把数据的具体物理实现留给物理模式，使用户与全局设计者能不必关心数据库的具体实现与物理背景，同时，它通过两级映射建立三级模式间的联系与转换，使得概念模式与外模式虽然并不物理存在，但是也能通过映射而获得其存在的实体，同时两级映射也保证了数据库系统中数据的独立性，亦即数据的物理组织改变与逻辑概念级改变，并不影响用户外模式的改变，它只要调整映射方式而不必改变用户模式。

1. 从概念模式到内模式的映射

该映射给出了概念模式中数据的全局逻辑结构到数据的物理存储结构间的对应关系，此种映射一般由 DBMS 实现。

2. 从外模式到概念模式的映射

概念模式是一个全局模式，而外模式则是用户的局部模式，一个概念模式中可以定义多个外模式，而每个外模式是概念模式的一个基本视图。外模式到概念模式的映射给出了外模式与概念模式的对应关系，这种映射一般由 DBMS 实现。

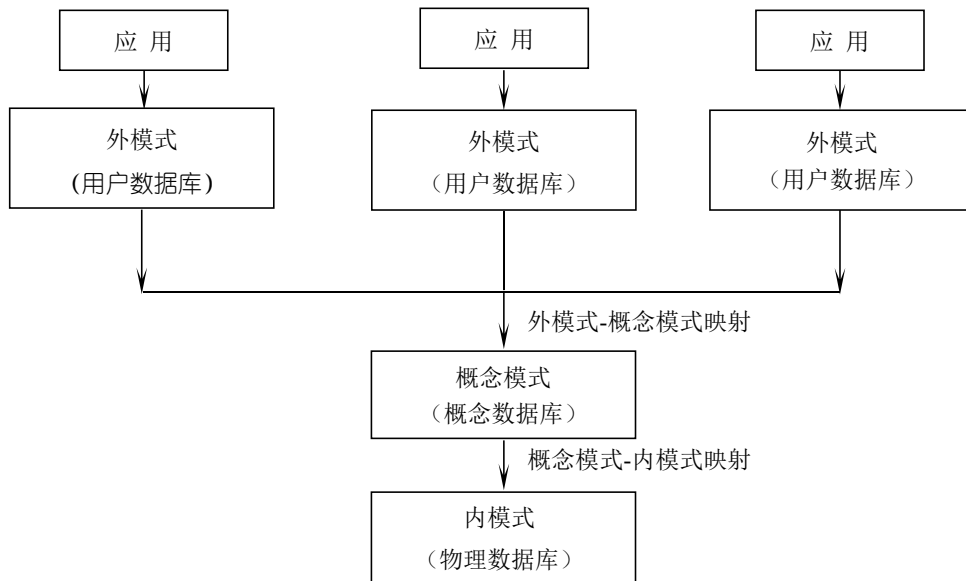


图 1.6 三级模式两种映射关系图

习题一

1.1 试解释下列术语：

(1) 数据库；(2) 数据库管理系统；(3) 数据库系统；(4) 数据库应用系统

1.2 试述数据库系统及数据库应用系统的组成内容及其差别。

1.3 试述数据库系统中数据的特性。

1.4 什么叫数据库管理员？他的主要工作是什么？

1.5 试述数据库系统发展的几个阶段，以及当前发展的方向。

1.6 试述数据库系统的特点。

1.7 试述数据库的三级模式与二级映射。

第一章复习指导

本章是数据库系统的全局性介绍,对全书具有提纲契领的作用。对有关数据库系统的全局性概念、特点、结构原理及发展历史等四个方面都作了介绍,读者在学习该章后能对数据库系统有一个全面的了解。

1. 基本概念

本章介绍了数据库系统具全局性的六个基本概念。

- 数据 (data)
- 数据库 DB (databse)
- 数据库管理系统 DBMS (databse management system)
- 数据库管理员 DBA (databse adalistrator)
- 数据库系统 DBS (databse system)
- 数据库应用系统 DBAS (databse application system)

对上述六个概念须有深刻了解,从两个方面入手:

- (1) 每个概念的基本内容
- (2) 概念间的关系

概念间有相关关联,必需了解,如:

- 数据与数据库间关系;
- DB 与 DBMS 间的关系;
- DBMS 与 DBA 间的关系;
- DBMS 与 DBS 间的关系;
- DBS 与 DBAS 间的关系;

2. 数据库系统特点

- 数据的集成性
- 数据的共享性
- 数据的独立性
- 数据的统一管理与控制

3. 数据库基本结构原理

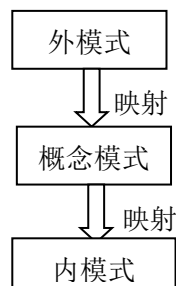
数据库基本结构遵从数据库三级模式与二级映射的结构原理。

(1) 三级模式结构

- 概念模式
- 外模式
- 内模式

(2) 二级映射

- 概念模式到内模式映射
- 外模式到概念模式映射



4. 数据库系统的发展与目前主流

(1) 数据库系统发展的四个阶段:

- 文件系统
- 层次及网状数据库系统
- 关系数据库系统
- 新一代数据库系统

(2) 目前主流数据库系统

- 关系数据库扩充, 包括 Web 数据库、数据仓库、移动式及嵌入式数据库、安全数据库等。
- 对象关系数据库系统

5. 本章的重点内容

- 基本概念
- 基本结构

第二章 数据模型

本章主要讨论数据模型。数据模型是数据库系统的基础与核心，本章将对数据模型作系统的介绍，其中涉及到数据模型的基本概念与基本内容。

2.1 数据模型的基本概念

数据是现实世界中客体的符号抽象，而数据模型（data model）则是数据基本特征的抽象。数据模型是数据库系统的核心与基础。数据模型描述数据的结构、定义在结构上的操作以及约束条件。它从抽象层次上描述了系统的静态特征、动态行为和约束条件，为数据库系统的表示和操作提供一个框架。

数据模型按不同的应用层次分成三种类型，它们是概念数据模型（conceptual data model）、逻辑数据模型（logic data model）及物理数据模型（physical data model）。

概念数据模型又称概念模型，它是一种面向客观世界、面向用户的模型，它与具体的数据库管理系统无关，与具体的计算机平台无关。概念模型着重于对客观世界复杂事物的结构描述及它们间的内在联系的刻画，而将与 DBMS、计算机有关的物理的、细节的描述留给其他种类的模型。因此，概念模型是整个数据模型的基础。目前，较为有名的概念模型有 E-R 模型、扩充的 E-R 模型、面向对象模型及谓词模型等。

逻辑数据模型又称逻辑模型，它是一种面向数据库系统的模型，该模型着重于在数据库系统一级的实现。它是客观世界到计算机间的中介模型，具有承上启下的功能。概念模型只有在转换成逻辑模型后才能在数据库中得以表示。目前，逻辑模型很多，较为成熟并被人们大量使用的有：层次模型、网状模型、关系模型、面向对象模型、谓词模型以及对象关系模型等。

物理数据模型又称物理模型，它是一种面向计算机物理表示的模型，此模型给出了数据模型在计算机上物理结构的表示。

数据模型所描述的内容有三个部分，它们是数据结构、数据操作与数据约束。

(1) 数据结构。数据模型中的数据结构主要描述基础数据项的类型、性质以及数据项间的关联，且在数据库系统中具有统一的结构形式，它也称数据模式。数据结构是数据模型的基础，数据操作与约束均建立在数据结构上。不同数据结构有不同的操作与约束。因此，一般数据模型均依据数据结构的分类。

(2) 数据操作。数据模型中的数据操作主要描述在相应数据结构上的操作类型与操作方式。

(3) 数据约束。数据模型中的数据约束主要描述数据结构内数据间的语法、语义联系，它们间的制约与依存关系，以及数据动态变化的规则以保证数据的正确、有效与相容。

2.2 数据模型的四个世界

数据库中的数据模型可以将复杂的现实世界要求反映到计算机数据库中的物理世界，这种反映是一个逐步转化的过程，它分为四个阶段，被称为四个世界。由现实世界开始，经历概念世界、信息世界而至计算机世界，从而完成整个转化。由现实世界开始每到达一个新的世界都是一次新的飞跃和提高。

(1) 现实世界 (real world)。用户为了某种需要, 需将现实世界中的部分需求用数据库实现。此时, 它设定了需求及边界条件, 这为整个转换提供了客观基础和初始启动环境。此时, 人们所见到的是客观世界中划定边界的一个部分环境, 它称为现实世界。

(2) 概念世界 (conceptual world)。以现实世界为基础作进一步的抽象形成概念模型, 这是一次新的飞跃与提高, 它将现实世界中错综复杂的关系作分析, 去粗取精, 去伪存真, 最后形成一些基本概念与基本关系, 它们可以用概念模型提供的术语和方法统一表示, 从而构成了一个新的世界——概念世界。在概念世界中所表示的模型都是较为抽象的, 它们与具体数据库、具体计算机平台无关, 这样做的目的是为了集中精力构作数据的框架性而不是拘泥于细节性的修饰。

(3) 信息世界 (information world)。在概念世界的基础上进一步着重于在数据库级上的刻划, 而构成的逻辑模型叫信息世界。信息世界与数据库的具体模型有关, 如层次、网状、关系模型等。

(4) 计算机世界 (computer world)。在信息世界的基础上致力于在计算机物理结构上的实现, 从而形成的物理模型叫计算机世界。现实世界的要求只有在计算机世界中才得到真正的物理实现, 而这种实现是通过概念世界、信息世界逐步转化得到的。

上面所述的四个世界中, 现实世界是客观存在, 而其他三个世界则是人们加工而得到的, 这种加工转化的过程是一种逐步精化的层次过程, 如图 2.1 所示。它符合人类认识客观事物的规律。

下面分三节对概念世界、信息世界与计算机世界作较为详细的介绍。

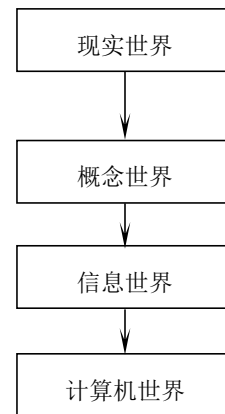


图 2.1 四个世界的转化示意图

2.3 概念世界与概念模型

概念世界是一个较为抽象、概念化的世界, 它给出了数据的概念化结构。概念世界一般用概念模型表示。概念模型目前常用的有若干种, 本书选用 E-R 模型、扩充 E-R 模型、面向对象模型和谓词模型等四种。

2.3.1 E-R 模型

E-R 模型 (entity-relationship model) 又称实体联系模型, 它于 1976 年由 Peter Chen 首先提出, 这是一种概念化的模型, 它将现实世界的要求转化成实体、联系、属性等几个基本概念以及它们间的两种基本关系, 并且用一种较为简单的图表示叫 E-R 图 (entity-relationship diagram), 该图简单明了, 易于使用, 因此很受欢迎, 长期以来作为一种主要的概念模型被广泛应用。

1. E-R 模型的基本概念

E-R 模型有如下三个基本概念。

(1) 实体 (entity): 现实世界中的事物可以抽象成为实体, 实体是概念世界中的基本单位, 它们是客观存在的且又能相互区别的事物。凡是有共性的实体可组成一个集合称为实体集 (entity set)。如学生张三、李四是实体, 而他们也均是学生, 从而组成一个实体集。

(2) 属性 (attribute): 现实世界中事物均有一些特性, 这些特性可以用属性这个概念表示。属性刻划了实体的特征。属性一般由属性名、属性型和属性值组成。其中属性名是

属性标识，而属性的型与值则给出了属性的类型与取值，属性取值有一定范围称属性域（domain）一个实体往往可以有若干个属性，如实体张三的属性可以有姓名、性别、年龄等。

(3) 联系（relationship）：现实世界中事物间的关联称为联系。在概念世界中联系反映了实体集间的一定关系，如医生与病人这两个实体集间的医治关系，官、兵间的上下级管理关系，旅客与列车间的乘坐关系。

实体集间的联系，就实体集的个数而言可分为以下几种。

(1) 两个实体集间的联系。两个实体集间联系是一种最为常见的联系，前面举的例子均属两个实体集间的联系。

(2) 多个实体集间的联系。这种联系包括三个实体集间的联系以及三个以上实体集间的联系。如工厂、产品、用户这三个实体集间存在着工厂提供产品为用户服务的联系。

(3) 一个实体集内部的联系。一个实体集内有若干个实体，它们间的联系称实体集内部联系。如某公司职工这个实体集内部可以有上下级联系。往往某人（如科长）既可以是一些人的下级（如处长），也可以是另一些人的上级（如本科内科员）。

实体集间联系的个数可以是单个也可以是多个。如官、兵之间既有上下级联系，也有同志间联系，还可以有兴趣爱好的联系等。

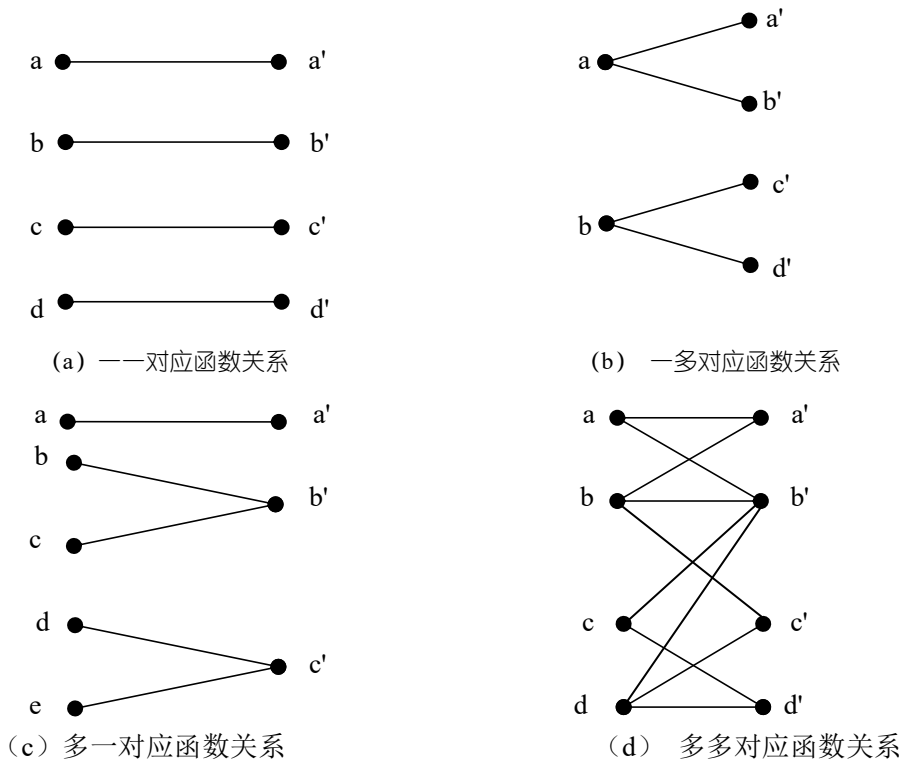


图 2.2 四种函数关系表示图

两个实体集间的联系实际上是实体集间的函数关系，这种函数关系可以有下面几种。

(1) 一一对应（one to one）的函数关系：这种函数关系是常见的函数关系之一，它可以记为 1:1。如学校与校长间的联系，一个学校与一个校长间相互一一对应。

(2) 一对多对应 (one to many) 或多一对应 (many to one) 函数关系: 这两种函数关系实际上是同一种类型, 它们可以记为 1:m 或 m:1。如学生与其宿舍房间的联系是多一对应函数关系 (反之, 则为一多对应函数关系), 即多个学生对应一个房间。

(3) 多多对应 (many to many) 函数关系: 这是一种较为复杂的函数关系, 可记为 m:n。如教师与学生这两个实体集间的教与学的联系是多多对应函数关系。因为一个教师可以教授多个学生, 而一个学生又可以受教于多个教师。

以上四种函数关系可用图 2.2 表示。

E-R 模型由以上三个基本概念组成。这三个基本概念之间的关系如下。

2. E-R 模型三个基本概念之间的联接关系

(1) 实体集 (联系) 与属性间的联接关系

实体是概念世界中的基本单位, 属性附属于实体, 它本身并不构成独立单位。一个实体可以有若干个属性, 实体以及它的所有属性构成了实体的一个完整描述。因此实体与属性间有一定联接关系。如在人事档案中每个人 (实体) 可以有编号、姓名、性别、年龄、籍贯、政治面貌等若干属性, 它们组成了一个有关人 (实体) 的完整描述。

实体有型与值之别, 一个实体的所有属性构成了这个实体的型, (如表 2.1 中人事档案中的实体, 它的型是编号、姓名、性别、年龄、籍贯、政治面貌等), 而实体中属性值的集合 (如表 2.1 中 138, 徐英健, 女, 18, 浙江, 团员) 则构成了这个实体的值。

相同型的实体构成了实体集。实体集由实体集名、实体型和实体三部分组成。如表 2.1 中的每一行是一个实体, 它们均有相同的型, 因此表内诸实体构成了一个实体集。

表 2.1 人事档案简表

编号	姓名	性别	年龄	籍贯	政治面貌
138	徐英健	女	18	浙江	团员
139	赵文虎	男	23	江苏	党员
140	沈亦奇	男	20	上海	群众
141	王 宾	男	21	江苏	群众
142	李红梅	女	19	安徽	团员

联系也可以附有属性, 联系和它的所有属性构成了联系的一个完整描述, 因此, 联系与属性间也有联接关系。如有教师与学生两个实体集间的教与学的联系, 该联系尚可附有属性——教室号。

(2) 实体 (集) 与联系间的联接关系

实体集间可通过联系建立联接关系, 一般而言, 实体集间无法建立直接关系, 它只能通过联系才能建立起联接关系。如教师与学生之间无法直接建立关系, 只有通过“教与学”的联系才能在相互之间建立关系。

表 2.2 实体 (集)、属性、联系三者的联接关系表

	实 体 (集)	属 性	联 系
实 体 (集)	×	单 向	双 向
属 性	单 向	×	单 向
联 系	双向	单 向	×

上面所述的两个联接关系建立了实体 (集)、属性、联系三者的关系, 用表 2.2 表示。

E-R 模型中有三个基本概念以及它们间的两种基本关系。它们将现实世界中错综复杂的现象抽象成简单明了的几个概念及关系，具有极强的概括性，因此，E-R 模型目前已成为表示概念世界的有力工具。

3. E-R 模型的图示法

E-R 模型另一个很大的优点是它可以用一种非常直观的图的形式表示，这种图称为 E-R 图。在 E-R 图中我们分别用不同的几何图形表示 E-R 模型中的三个概念与两个联接关系。

(1) 实体集表示法。在 E-R 图中用矩形表示实体集，在矩形内写上该实体集之名。如实体集学生 (student)、课程 (course) 可用图 2.3 表示。



图 2.3 实体集表示法

(2) 属性表示法。在 E-R 图中用椭圆形表示属性，在椭圆形内写上该属性名。如学生有属性学号 (sno)、姓名 (sn) 及年龄 (sa)，可以用图 2.4 表示。

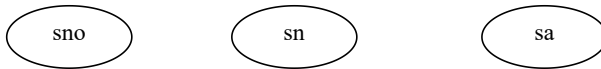


图 2.4 属性表示法

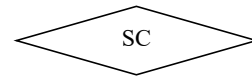


图 2.5 联系表示法

(3) 联系表示法。在 E-R 图中用菱形表示联系，在菱形内写上该联系名。如学生与课程间联系 SC，用图 2.5 表示。

三个基本概念分别用三种几何图形表示，它们间的联接关系也可用图形表示。

(4) 实体集 (联系) 与属性间的联接关系。属性依附于实体集，因此，它们之间有联接关系。在 E-R 图中这种关系可用联接这两个图形间的无向线段表示 (一般情况下可用直线)。如实体集 student 有属性 sno (学号)、sn (学生姓名) 及 sa (学生年龄)；实体集 course 有属性 cno (课程号)、cn (课程名) 及 pno (预修课号)，此时它们可用图 2.6 联接。



图 2.6 实体集的属性间的联接

属性也依附于联系，它们间也有联接关系，因此也可用无向线段表示。如联系 SC 可与学生的课程成绩属性 g 建立联接，用图 2.7 表示。

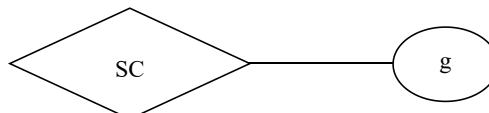


图 2.7 联系与属性间的联接

(5) 实体集与联系间的联接关系。在 E-R 图中实体集与联系间的联接关系可用联接这两个图形间的无向线段表示。如实体集 student 与联系 SC 间有联接关系，实体集 course 与联系 SC 间也有联接关系，因此它们间可用无向线段相联，如图 2.8 所示。

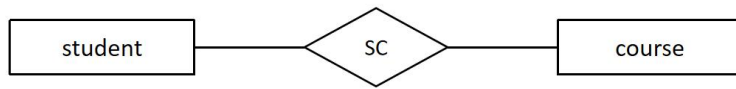


图 2.8 实体集与联系间的联接关系

有时为了进一步刻划实体间的函数关系，还可在线段边上注明其对应的函数关系，如 1:1, 1:n, n:m 等。如 student 与 course 间有多多函数对应关系，此时可以用图 2.9 表示。

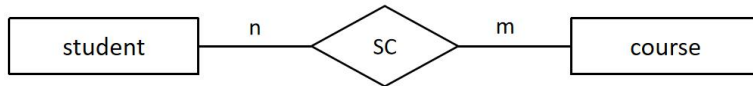


图 2.9 实体集间的函数关系表示图

实体集与联系间的联接可以有多种，上面所举例子均是两个实体集间联系叫二元联系，也可以是多个实体集间联系，叫多元联系。如工厂、产品与用户间的联系 FPU 是一种三元联系，可用图 2.10 表示。

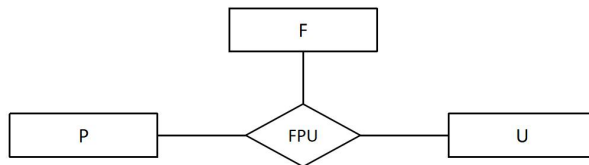


图 2.10 多个实体集间联系的联接方法

一个实体集内部可以有联系。如某公司职工 (employee) 与上下级管理 (manage) 间的联系，可用图 2.11 (a) 表示。

实体集间可有多种联系。如教师 (T) 与学生 (S) 之间可以有教学 (E) 联系也可有同志 (C) 间的联系，可用图 2.11 (b) 表示。

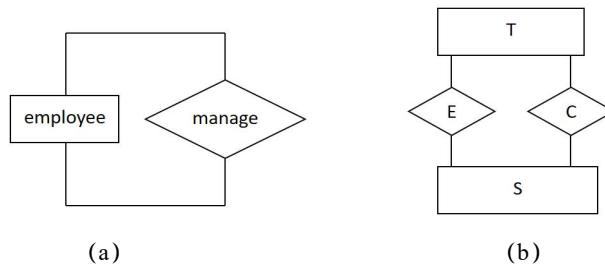


图 2.11 实体集间多种联系

矩形、椭圆形、菱形以及按一定要求相互间相联接的线段构成了一个完整的 E-R 图。

例 2.1 由前面所述的实体集 student、course 及附属于它们的属性和它们间联系 SC 以及附属于 SC 的属性 g，构成了一个有关学生、课程以及他们的成绩和他们间的联系的概念模型。用 E-R 图表示如图 2.12 所示。

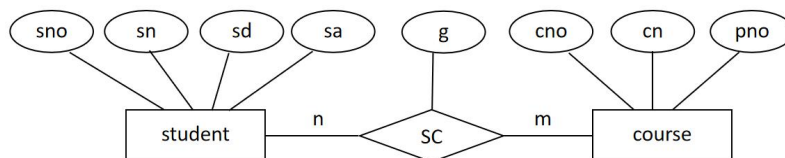


图 2.12 E-R 图的一个实例

例 2.2 图 2.13 给出了一个工厂的物资管理 E-R 图，它由职工 (employee)，仓库 (warehouse)，项目 (project)，零件 (part)，供应商 (supplier) 等五个实体集以及供应、库存、领导、工作等四个联系所组成。

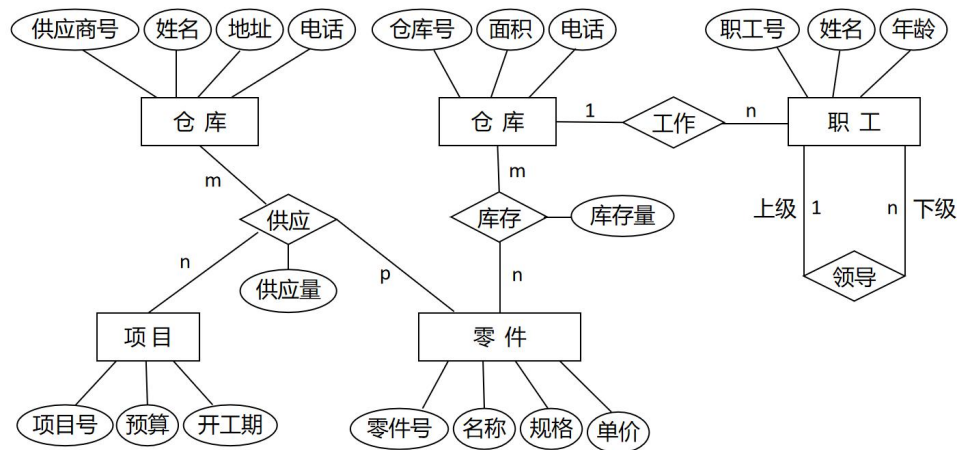


图 2.13 某工厂物资管理 E-R 图

在概念上，E-R 模型中的实体、属性与联系是三个有明显区别的不同概念，但是在分析客观世界的具体事物时，往往会产生混淆甚至区别不清。这是构造 E-R 模型最困难的地方，这主要靠经验与积累，当然也有一些规则可循，但关键取决于应用的背景以及设计人员的理解。

2.3.2 扩充的 E-R 模型——EE-R 模型

E-R 模型在表示概念世界中使用较为普遍，但是，它在表示上尚有不足，最主要的是联系的语义不够丰富。因此，不少人对 E-R 模型作了改正与扩充，Teorey 等人提出的扩充 E-R 模型称为 EE-R 模型 (extend entity-relationship model)，它在 E-R 模型的基础上适当作了扩充，它保持了 E-R 模型简明、清晰的特点，同时又弥补了 E-R 模型的一些不足。

EE-R 模型是 E-R 模型的扩充，其主要扩充了如下：

在 EE-R 模型中增加了一种特殊的联系叫 IS-a 联系。IS-a 联系建立了两个实体集间的继承 (inheritance) 关系，亦即是说如有实体集 A，B，而 B 是 A 的一个子集且具有比 A 更多的属性，此时，通过 IS-a 联系，B 中实体可以“继承”A 中所有属性（因为 B 是 A 的子集，因此 B 中实体必出现在 A 中），同时 B 还可有自己的属性，此时 A 称为 B 的超（实体）集 (super-entity set)，而 B 称为 A 的子（实体）集 (sub-entity set)。

例 2.3 设有学生 (student)、研究生 (graduate student) 实体集。学生实体集有属性学号 (sno)、姓名 (sn)、系别 (sd) 及年龄 (sa)，而研究生也是学生，它具有学生的所有属性，同时还有属性导师姓名 (adviser-name) 以及研究方向 (research-field)。此时，“学生”与“研究生”间可建立起 IS-a 联系，该联系一旦建立，“研究生”即可继承“学生”的所有属性，亦即是说，此时“研究生”可视为有 sno, sn, sd, sa 及 adviser-name, research-field 等六个属性。

在 E-R 模型中有 E-R 图，而在 EE-R 模型中则有 EE-R 图，它基本与 E-R 图一样，也仅在上面所述的扩充上增加了新的表示。在 EE-R 图中两实体集间 IS-a 联系可用有向线段相连并用圈 O 表示联系，而线段的方向由子（实体）集指向超（实体）集。

例 2.4 例 2.3 所示的 EE-R 图可用图 2.14 表示。

EE-R 模型比 E-R 模型有更丰富的语义和更强的表达力，同时仍保持 E-R 简明性，此外，EE-R 图基本上继承了 E-R 图直观、明了的特点。因此，这种模型在表示概念世界时比 E-R 模型更为有效。在本书中后面凡涉及概念世界的描述时均采用此种模型。

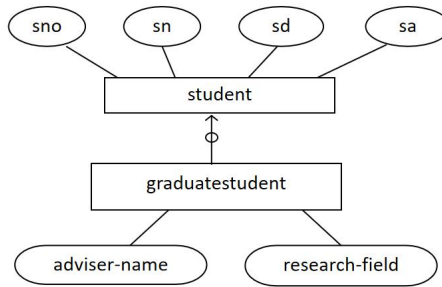


图 2.14 例 2.3 的 EE-R 图

最后我们用一个较为复杂的例子结束对 EE-R 模型的讨论。

例 2.5 可用图 2.15 所示的 EE-R 图表示大学、校长、职工间的概念模型。

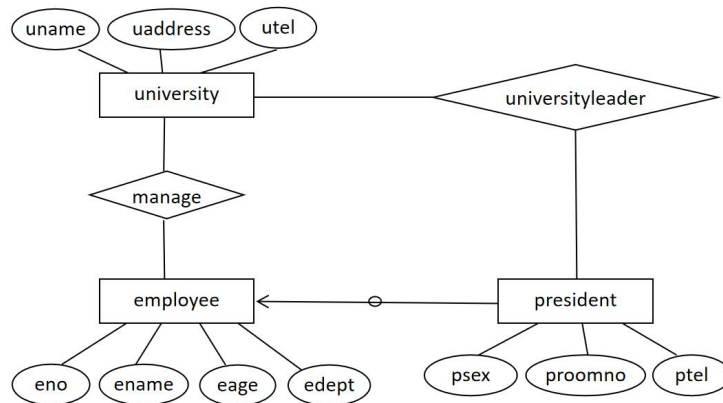


图 2.15 一个 EE-R 图

2.3.3 面向对象模型

面向对象模型 (object-oriented model) 是近几年来迅速崛起并得到发展的一种模型，该模型是在吸收了以前各种概念模型优点的基础上并借鉴了面向对象方法而建立的模型。这种模型具有更强的表示能力，特别是在表示非传统的数据领域如 CAD、工程领域、多媒体领域等复杂数据关系领域具有极强的表达力。

在面向对象模型中的最基本的概念是对象 (object)，它是客观世界中概念化的基本实体，它可以是简单的实体如一个螺丝帽、一片树叶，也可以是复杂的实体如地球、宇宙飞船。它可以是具体的实体，也可以是抽象的实体。总之，相互能区别的事物均可视为对象，在数据库系统中，对象是一个基本数据单位，如数据元组即为对象。

1. 对象的组成

(1) 对象标识符: 每个对象均有一个能相互区别的名字叫对象标识符 (object Identifier, 简称 OID)。

(2) 对象的静态特性: 对象的静态特性即是对对象的属性刻划, 每个对象都有一些属性以刻划该对象的特性, 在数据库系统中对象的属性类似于 E-R 模型中的属性。

(3) 对象的动态特性: 对象的动态特性即是可以对对象施行操作的一种方法, 它称为方法 (method) 或称操作 (operation)。一个对象可以有若干方法, 在数据库系统中, 方法可视为基于数据上的一种操作。

2. 对象的特点

(1) 对象的封装 (encapsulate): 对象的属性与方法封装在一起的, 亦即是说, 操作与属性是相互依存的。

(2) 对象标识符的独立性: 每个对象均有一个独立于属性以外的名字, 亦即是说, 对象的属性值与对象名是有区别的。如两个相同品牌的茶杯, 它们的属性值相等, 但它们是不同的对象, 因此有不同的对象名, 这就是对象标识符的独立性。在 E-R 模型中每个实体的名是由一个属性值集 (即主码) 表示的, 这与面向对象模型有完全的不同。

(3) 对象属性值的多值性: 一个对象的属性, 可以是单值也可以是多值。如一学生, 他的姓名、年龄、性别等均为单值, 但如有学生修读课程这一属性时, 此属性可为多值, 因为学生一般修读课程可超过一门, 这就是对象属性值的多值性, 该多值性比较合理地反映了客观的实际。

3. 类与类的特性

“人以群分, 物以类聚”, 为便于概念上认识的方便, 我们一般将具有相同属性、方法的对象集合一起称为类 (class), 而此时类中对象称为实例 (instance)。在数据库系统中, 类是一种基本处理单位。

类与类之间存在着某些联系, 在面向对象模型中, 一般分两种:

(1) 子类 (sub-class) 与超类 (super-class): 类的子集也可以是一个类, 称为该类的子类, 而原来的类, 则称为子类的超类。子类继承 (inheritance) 超类的属性与方法, 并具有自己的属性与方法。这两种类间的联系称为 IS-a 联系。从概念上讲, 从下向上是一个普遍化、抽象化的过程叫普化 (generalization)。反之, 由上向下是一个特殊、具体化的过程叫特化 (specialization)。图 2.16 给出了子类、超类的一个示意图。

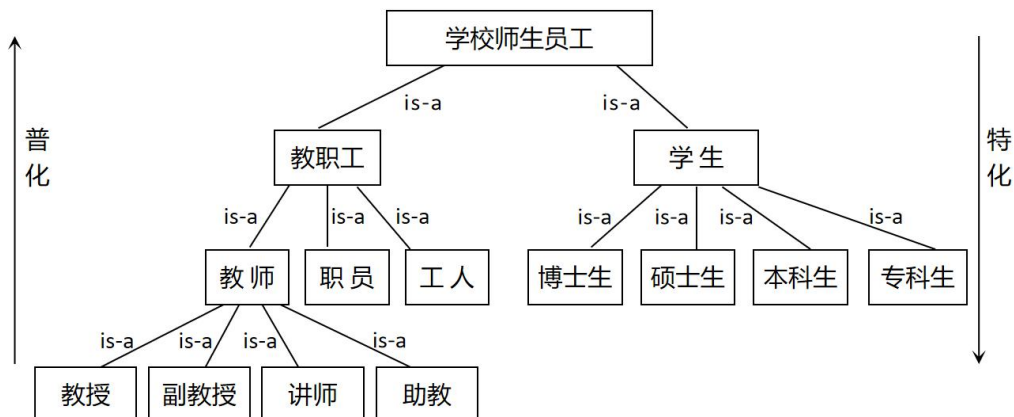


图 2.16 子类、超类示意图

(2) 类的聚合 (composition) 与分解 (decomposition) : 在现实世界中往往存在着由简单的对象组合成复杂的对象的现象, 在面向对象模型中即是类的聚合, 类的聚合即表示若干个简单类可以聚合成一个复杂类。类之间的这种聚合联系构成了类与类的 Is-part-of 关系。反之, 则是类的分解, 可以由复杂类分解成若干层次的简单类。

类的聚合现象很普遍, 如一台机器由若干部件组成, 而部件则由零件构成。图 2.17 给出了类的聚合与分解的示意图。

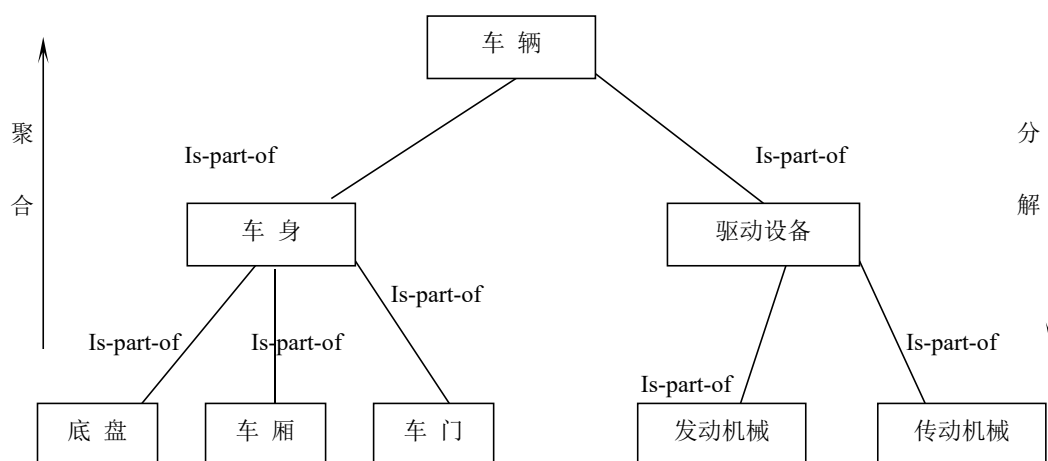


图 2.17 类的聚合、分解表示

在数据库系统中, 以类为处理单位, 以类间继承、聚合为关联所构成的模型称为面向对象模型。

面向对象模型能描述复杂的现实世界, 具有较强灵活性、可扩充性与可重用性。它的动态特性描述、对象标识符、类的普化/特化、类的聚合/分解, 以及消息功能等都比前面所介绍的概念模式要好。此模型既是一种概念模型, 同时还可以依据它直接构造数据逻辑模型, 称为面向对象逻辑模型。

*2.3.4 谓词模型

谓词模型 (predicate model) 又称谓词逻辑模型, 它用一阶谓词逻辑的理论和方法构造模型, 下面我们对其作介绍。

1. 谓词

一阶谓词逻辑的基础是谓词, 谓词可以用 $P(x_1, x_2, \dots, x_n)$ 表示, 其中 P 为谓词符号,

$x_i (i = 1, 2, \dots, n)$ 是个体变量, 给定一指派 (a_1, a_2, \dots, a_n) 代入 $P(x_1, x_2, \dots, x_n)$ 后它可有真:

T (true) 或假: F (false) 两个值。

谓词可以表示客观世界事物的性质与关系, 一般讲, 一元谓词可以刻划事物特性, 如 P (x) 即表示 x 具有性质 P, 而多元谓词可以表示事物间的关系, 如 $P(x_1, x_2, \dots, x_n)$ 即表

示 x_1, x_2, \dots, x_n 间有关系 P , 下面的例 2.6 给出了谓词所表示的具体例子。

例 2.6 下面的谓词给出了所刻划的事物性质与关系:

- (1) $\text{Red}(x)$ —— 该一元谓词刻划了具有红色特色的事物的集合。
- (2) $\text{Parent}(x, y)$ —— 该二元谓词刻划了具有双亲子女关系的人物对的集合。
- (3) $\text{Plus}(x, y, z)$ —— 该三元谓词刻划了 $x+y=z$ 的加法运算关系。

在实际应用中, 有些固定的谓词特别有用, 我们称为常谓词, 在数据库系统中下面的常谓词是经常用到的:

- 小于谓词: $<(x, y)$
- 小于等于谓词: $\leq(x, y)$
- 相等谓词: $=(x, y)$
- 不等谓词: $\neq(x, y)$
- 字符串型谓词: $\text{ch}(n)(x)$
- 布尔型谓词: $\text{bool}(x)$
- 整型谓词: $\text{int}(x)$
- 实型谓词: $\text{real}(x)$
- 汉字型谓词: $\text{chines}(n)(x)$
- 空值谓词: $\text{null}(x)$

2. 谓词公式

(1) 谓词公式定义

谓词公式 (简称公式) 可用下面的方式定义:

- (1) 谓词是公式;
- (2) 如果 φ_1, φ_2 是公式则 $\varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2, \varphi_1 \rightarrow \varphi_2$ 以及 $\neg \varphi_1$ 均是公式;
- (3) 如果 φ 是公式则 $\forall x \varphi$ 及 $\exists x \varphi$ 是公式;
- (4) 公式由且仅由上面三种方式通过有限次迭代实现。

但在数据库的数据模型中一般不用此种复杂形式表示公式, 而用 Horn 子句或 Datalog 子句形式表示。

3. Horn 子句

Horn 子句的一般形式是:

$$A_n :- A_1, A_2, \dots, A_{n-1}$$

它是: $A_1 \wedge A_2 \wedge \dots \wedge A_{n-1} \rightarrow A_n$ 的另一种表示方法, 其中 $A_i (i=1, 2, \dots, n)$ 均是谓词,

其中 A_n 称为子句的头而 A_1, A_2, \dots, A_{n-1} 则称为子句的体。

Horn 子句还可以有下面几种特殊表示:

(1) 断言:

当 Horn 子句中 $n=1$ 时此子句称断言, 并可表示为:

A_n

(2) 假设:

当 Horn 子句中 A_n 不出现时此子句称假设, 并可表示为:

$$:- A_1, A_2, \dots, A_{n-1}$$

(3) 空子句:

当 Horn 子句中 $n=0$ 时此子句称空子句, 并可表示为:



4. DATALOG 子句

DATALOG 子句是专门为数据库中数据模型设计的一种语言, 它的基本形式与 Horn 子句一样, 但是需加一些限制:

(1) DATALOG 中的变量必须受限, 即 DATALOG 中的变量不能在无限域中变化, 必须限制在一个有限域内, 这主要是由于数据库中数据是有限的特性所决定, 而这种限制一般均在 DATALOG 中增加一些限制性谓词而实现。

(2) 在 DATALOG 中一般只出现自由变量以及存在量词形式出现的约束变量, 在子句的头中所出现的变量均为自由变量, 而在子句的体中所出现的变量 (除头中变量外) 均为约束变量。

下面给出几个例子:

例 2.7 $G(x, y) :- P(x, z), P(z, y), ch(20)(x), ch(20)(y)$

在这个 DATALOG 子句中给出了祖-孙关系的定义, 即 x 与 z 是父子关系, z 与 y 是父子关系则 x 与 y 必是祖孙关系, 而在子句中 x, y 是自由变量, 而 z 则是约束变量, 同时自由变量 x, y 需加限制性谓词 $Ch(20)(x)$ 及 $Ch(20)(y)$ 。

应用 DATALOG 子句可以刻画和表示数据模型中的数据结构、数据操纵与数据约束, 下面我们以 E-R 模型为背景用 DATALOG 刻画之。

1. 实体集

可以用谓词 $P(x_1, x_2, \dots, x_n)$ 表示实体集 $R(x_1, x_2, \dots, x_n)$, 其中 R 为实体集名,

$x_i (i = 1, 2, \dots, n)$ 为其属性, 实体集内的元组使 $P(x_1, x_2, \dots, x_n)$ 为真, 而非实体集内的元组

则使 $P(x_1, x_2, \dots, x_n)$ 为假:

$$P(x_1, x_2, \dots, x_n) = \begin{cases} T, & \text{当 } (a_1, a_2, \dots, a_n) \in R \\ F, & \text{当 } (a_1, a_2, \dots, a_n) \notin R \end{cases}$$

例 2.8 如图 2.18 所示的实体集 $S(sno, sn, sa, sd)$ 与 $C(cno, cn, pno)$ 可

用谓词 **S** (sno, sn, sa, sd) 及 **C** (cno, cn, pno) 的谓词序列列表 (表 2.3) 表示:

S:																				
<table border="1" style="display: inline-table; border-collapse: collapse;"> <thead> <tr><th>sno</th><th>sn</th><th>sa</th><th>sd</th></tr> </thead> <tbody> <tr><td>001</td><td>Marry</td><td>18</td><td>CS</td></tr> <tr><td>002</td><td>Bob</td><td>19</td><td>CS</td></tr> <tr><td>003</td><td>Aries</td><td>20</td><td>MA</td></tr> <tr><td>004</td><td>Jhon</td><td>18</td><td>MA</td></tr> </tbody> </table>	sno	sn	sa	sd	001	Marry	18	CS	002	Bob	19	CS	003	Aries	20	MA	004	Jhon	18	MA
sno	sn	sa	sd																	
001	Marry	18	CS																	
002	Bob	19	CS																	
003	Aries	20	MA																	
004	Jhon	18	MA																	

C:												
<table border="1" style="display: inline-table; border-collapse: collapse;"> <thead> <tr><th>cno</th><th>cn</th><th>pno</th></tr> </thead> <tbody> <tr><td>C01</td><td>database</td><td>C02</td></tr> <tr><td>C02</td><td>OS</td><td>C03</td></tr> <tr><td>C03</td><td>C++</td><td>C04</td></tr> </tbody> </table>	cno	cn	pno	C01	database	C02	C02	OS	C03	C03	C++	C04
cno	cn	pno										
C01	database	C02										
C02	OS	C03										
C03	C++	C04										

图 2.18 实体集示例图

表 2.3 实体集谓词表

- S** (001, Marry, 18, CS)
- S** (002, Bob, 19, CS)
- S** (003, Aries, 20, MA)
- S** (003, Jhon, 18, MA)
- C** (C01, database, C02)
- C** (C02, OS, C03)
- C** (C03, C++, C04)

2. 属性

可以用 $P(x_1, x_2, \dots, x_n)$ 中的个体变量 x_i ($i=1, 2, \dots, n$) 表示属性, 而属性的域可用谓词表示, 如属性 x_3 为整型则可用 $\text{Int}(x_3)$ 表示。一般, 对每个谓词中的个体变量均有一定的约束, 可用统一的约束谓词 $C_i(x)$ ($i=1, 2, \dots, n$) 表示, 因此一个包含属性的实体集往往可用下面的谓词公式表示:

$$P(x_1, x_2, \dots, x_n) \wedge C_1(x_1) \wedge C_2(x_2) \wedge \dots \wedge C_n(x_n)$$

3. 联系

可以用谓词表示联系, 其原理与谓词表示实体集类似, 下面用一例说明之。

例 2.9 如在例 2.8 中实体集 **S** 与 **C** 间有联系 **SC** (sno, cno, g), (见图 2.19), 此时它可用谓词 **SC** (sno, cno, g) 刻划之, 并可用下面的谓词序列列表 (表 2.4) 表示。

SC

sno	cno	g
001	C01	优
001	C02	良
002	C03	良
003	C02	优
004	C02	良

图 2.19 联系示例图

表 2.4 联系谓词表

SC (001, C01, 优)
SC (001, C02, 良)
SC (002, C03, 良)
SC (003, C02, 优)
SC (004, C02, 良)

谓词模型除了可刻画 E-R 模型外, 还可以表示数据模式、数据操纵及数据约束等内容, 下面我们对其讨论之。

4. 模式

可以用 DATALOG 表示数据模式, 一个完整的数据模式可以由数据体结构、属性类型等组成, 它们可用 DATALOG 表示如下:

$$MS(a_1, a_2, \dots, a_n) :- DS(a_1, a_2, \dots, a_n), A_1(a_1), A_2(a_2), \dots, A_n(a_n) \quad (2.1)$$

其中 $MS(a_1, a_2, \dots, a_n)$ 表示模式, $DS(a_1, a_2, \dots, a_n)$ 是数据体结构谓词, 而 $A_i(a_i)$ 则是属性数据类型谓词, 而 a_i 则是属性 $i=(1, 2, \dots, n)$ 。

下面用一个例子介绍上面所建立的学生数据库的模式表达式。

例 2.10 上面所示的学生数据库的数据模式可以用谓词模型表示, 实体集 student、course 分别可用谓词 **S** (sno, sn, sd, sa) 与 **C** (cno, cn, pno) 表示, 联系 SC 可用谓词 **SC** (sno, cno, g) 表示, 而属性 sno, sn, sd, sa, cno, cn, pno, g 的域约束分别可用常谓词表示。最后整个模式 **MS** (sno, sn, sd, sa, cno, cn, pno, g) 可用下面的 DATALOG 表示:

$$\begin{aligned}
 MS(sno, sn, sd, sa, cno, cn, pno, g) :- & S(sno, sn, sd, sa), C(cno, cn, \\
 pno), SC(sno, cno, g), ch(5)(sno), & chines(4)(sn), ch(2)(sd), ch(2)(sa), \\
 ch(4)(cno), chines(10)(cn), ch(4)(pno), & Ch(1)(g), notnull(sno), notnull(cno).
 \end{aligned} \quad (2.2)$$

5. 操作

可以用 DATALOG 表示操作。如对上例中之模式可作如下查询操作:

- 查询所有学生学号与姓名——(其中谓词 Q 表示查询结果谓词)

$$Q(sno, sn) :- S(sno, sn, sd, sa) \quad (2.3)$$

- 查询年龄大于 20 岁的学生姓名

$$Q(sn) :- S(sno, sn, sd, sa), sa > '20' \quad (2.4)$$

- 查询修读课程名为 DTABASE 的所有学生姓名

$$Q(sn) :- S(sno, sn, sd, sa), C(cno, cn, pno), SC(sno, cno, g), cn = 'databse' \quad (2.5)$$

6. 约束

可以用 DATALOG 表示约束。如 $P(x_1, x_2, \dots, x_n)$ 中 x_i 与 x_j 间应满足一定约束关系, 此时可用一个完整性约束谓词 $I(x_i, x_j)$ 表示。又如学生数据库中 $sa \geq 18$ 且 $sa \leq 35$, 则可用如下 DATALOG 表达:

$$I(sa) :- S(sno, sn, sd, sa), sa \leq 35, 18 \leq sa \quad (2.6)$$

根据上述介绍, 可以用 DATALOG 表示关系数据模式以及相关操作与约束等。此外,

用 DATALOG 还可以表示很多更复杂的模型中的语义。

2.4 信息世界与逻辑模型

2.4.1 概述

信息世界是数据库的世界，该世界着重于数据库系统的构造与操作。信息世界由逻辑模型描述。

由于数据库系统不同的实现手段与方法，因此逻辑模型的种类很多，目前常用的有层次模型、网状模型、关系模型、对象-关系模型、面向对象模型及谓词模型等。其中层次模型发展最早并盛行于 20 世纪的 60~70 年代。网状模型稍后且具有比层次模型更为优越的性能，它盛行于 70~80 年代。关系模型的概念虽然出现很早（1970 年 IBM 公司的 Codd 在一篇文章中已提出了有关关系模型的论述），但由于在实现技术上的困难，直到 70 年代后期才出现真正的实用系统。基于关系模型的数据库系统在 80 年代开始流行，而到 90 年代直至今在仍占数据库市场的主导地位。面向对象模型出现于 20 世纪的 80 年代，在 90 年代开始流行。谓词逻辑模型出现于 70 年代末期，它表示力强，表示形式简单，目前它是演绎数据库及知识库的主要模型。面向对象模型与谓词逻辑模型既是概念模型又可作为逻辑模型。对象-关系模型是一种以关系模型为基础再适当增加面向对象模型内容的一种模型，它对应的概念模型是 EE-R 模型，这种模型目前是关系模型的一种扩充。

上面的六种逻辑模型分别与前面的四种概念模型相对应，它们的对应关系见表 2.5。

表 2.5 逻辑模型与概念模型的对应关系

概念模型	E-R 模型			EE-R 模型	面向对象模型	谓词模型
逻辑模型	层次模型	网状模型	关系模型	对象关系模型	面向对象模型	谓词模型

本章介绍关系模型、面向对象模型及谓词模型等三种逻辑模型，由于对象-关系模型是关系模型与面向对象模型的结合，本章不作介绍，但在第 11 章中将会作详细介绍，而层次模型、网状模型由于历史原因，目前使用者甚少，在我国使用者更为寥寥，故本章也不作介绍。

2.4.2 关系模型与关系模型数据库管理系统

关系模型（relational model）的基本数据结构是二维表，简称表（table）。大家知道，表格方式在日常生活中应用很广，在商业系统中，如金融、财务处理中无不以表格形式表示数据框架，这给了我们一个启发，用表格作为一种数据结构有着广泛的应用基础，关系模型即是以此思想为基础建立起来的。

关系模型中的操纵与约束也是建立在二维表上的，它包括对一张表及多张表间的查询、删除、插入及修改操作，以及相应于表的约束。

关系模型的思想是 IBM 公司的 E.F.Codd 于 1970 年在一篇论文中提出的，他在该年 6 月所发表的论文《大型共享数据库的关系模型》（a relational model for large shared data banks）中提出了关系模型与关系模型数据库的概念与理论，并用数学理论作为该模型的基础支撑。由于关系模型有很多诱人的优点，因此，从那时起就有很多人转向此方面的研究，并在算法与实现技术上取得了突破，在 1976 年以后出现了商用的关系模型数据库管理系

统，如 IBM 公司在 IBM-370 机上的 system-R 系统，美国加州大学在 DEC 的 PDP-11 机上基于 UNIX 的 Ingres 系统，Codd 也因他所提出的关系模型与关系理论的开创性工作而荣获了 1981 年计算机领域的最高奖——图灵（Turing）奖。

关系模型数据库由于其结构简单、使用方便、理论成熟而引来了众多的用户，上世纪 80 年代以后已成为数据库系统中的主流模型，很多著名的系统纷纷出现并占领了数据库应用的主要市场。目前，主要产品有 ORACLE、SQL Server、DB2、Sybase 等。关系模型数据库管理系统的数据库子语言也由多种形式而逐渐统一成一种标准化形式，即是 SQL 语言。

我国数据库应用起始于上世纪的 80 年代，当时大多采用 dBASE-II，dBASE-III 等初级形式的 RDBMS，90 年代初逐步进入关系模型数据库时代，ORACLE、Sybase 等数据库管理系统已逐渐替代低级系统，标准的 SQL 语言已取代非标准语言。

2.4.2.1 关系模型

关系模型由关系数据结构、关系操纵及关系中的数据约束三部分组成。

2.4.2.1.1 关系数据结构

1. 表结构

关系模型统一采用二维表结构，它也可简称表。二维表由表框架（frame）及表元组（tuple）组成。表框架由 n 个命名的属性（attribute）组成， n 称为属性元数（arity），每个属性有一个取值范围称为值域（domain）。

在表框架中按行可以存放数据，每行数据称为元组（tuple），实际上，一个元组是由 n 个元组分量所组成，每个元组分量是表框架中每个属性的投影值。一个表框架可以存放 m 个元组， m 称为表的基数（cardinality）。

一个 n 元表框架及框架内 m 个元组构成了一个完整的二维表，表 2.6 给出了二维表的一个实例。这是一个有关学生（S）的二维表。

表 2.6 二维表的一个实例

Sno	Sn	Sd	Sa
98001	张曼英	CS	18
98002	丁一明	CS	20
98003	王爱国	CS	18
98004	李 强	CS	21

二维表一般满足下面七个性质：

- (1) 二维表中元组个数是有限的——元组个数有限性。
- (2) 二维表中元组均不相同——元组的惟一性。
- (3) 二维表中元组的次序可以任意交换——元组的次序无关性。
- (4) 二维表中元组的分量是不可分割的基本数据项——元组分量的原子性。
- (5) 二维表中属性名各不相同——属性名惟一性。
- (6) 二维表中属性与次序无关——属性的次序无关性，（但属性次序一经确定则不能更改）。
- (7) 二维表中属性列中分量具有与该属性相同值域——分量值域的同—性。

满足七个性质的二维表称为关系（relation），因此以二维表为基本结构所建立的模型称为关系模型。

2. 关系

关系是二维表的一种抽象，关系是关系模型的基本数据单位，具有 n 个属性的关系称 n 元关系， $n=0$ 时称空关系，每个关系有一个名称为关系名，关系名及关系中的属性构成了关系框架，设关系的名为 R ，其属性为 a_1, a_2, \dots, a_n ，则该关系的框架是： $R(a_1, a_2, \dots, a_n)$

如表 2.6 所示的关系框架可以表示成： $S(sno, sn, sd, sa)$

每个关系有 m 个元组，设关系的框架为 $R(a_1, a_2, \dots, a_n)$ ，则其元组必具下面的形式：

$$\begin{aligned} & (a_{11}, a_{12}, \dots, a_{1n}) \\ & (a_{21}, a_{22}, \dots, a_{2n}) \\ & \vdots \\ & (a_{m1}, a_{m2}, \dots, a_{mn}) \end{aligned}$$

其中 a_{ij} ($i \in \{1, 2, \dots, m\}, j \in \{1, 2, \dots, n\}$) 为元组分量。

关系框架与关系元组构成了一个关系。一个语义相关的关系集合构成一个关系数据库 (relational database)。而语义相关的关系框架集合则构成了关系数据库模式 (relational database schema)，简称关系模式 (relational schema)。

关系模式支持子模式，关系子模式是关系数据库模式中用户所见到的那部分数据描述，关系子模式也是二维表结构，关系子模式对应用户数据库称视图 (view)。

3. 键

键是关系模型中的一个重要概念，它具有标识元组、建立元组间联系等重要作用。

- (1) 键 (key)：在二维表中凡能惟一最小标识元组的属性集称为该表的键。
- (2) 候选键 (candidate key)：二维表中可能有若干个键，它们称为该表的候选键。
- (3) 主键 (primary key)：从二维表的所有候选键中选取一个作为用户使用的键称为主键。一般主键也简称键。
- (4) 外键 (foreign key)：表 A 中的某属性集是某表 B 的键则称该属性集为 A 的外键。表中一定有键，因为如果表中所有属性子集均不是键则至少表中属性全集必为键，因此也一定有主键。

4. 关系与 E-R 模型

关系的结构简单，但它的表示范围广，E-R 模型中的属性、实体 (集) 及联系均可用它表示，表 2.7 给出了 E-R 模型与关系间的比较。

表 2.7 E-R 模型与关系间的比较表

E-R 模型	关 系
属 性	属 性

实 体	元 组
实 体 集	关 系
联 系	关 系

在关系模型中关系既能表示实体集又能表示联系。表 2.8 给出了某公司职工间上下级联系的关系表示。

表 2.8 上下级联系的关系表示

上 级	下 级
王 雷	杨 光 明
杨 光 明	吴 爱 珍
杨 光 明	徐 晴
吴 爱 珍	钱 华
吴 爱 珍	李 光 西

2.4.2.1.2 关系操纵

关系模型的数据操纵即是建立在关系上的数据操纵，一般有查询、删除、插入及修改等四种操作。

1. 数据查询

用户可以查询关系数据库中的数据，它包括一个关系内的查询以及多个关系间的查询。

(1) 对一个关系内查询的基本单位是元组分量，其基本过程是先定位后操作，所谓定位包括纵向定位与横向定位，纵向定位即是指定关系中的一些属性（称列指定），横向定位即是选择满足某些逻辑条件的元组（称行选择）。通过纵向与横向定位后一个关系中的元组分量即可确定了。在定位后即可进行查询操作，即将定位的数据从关系数据库中取出并放入至指定内存。

(2) 对多个关系间的数据查询则可分为 3 步进行，第 1 步将多个关系合并成一个关系，第 2 步为对合并后的一个关系作定位，最后第 3 步为查询操作。其中第 2 步与第 3 步为对一个关系的查询故我们只介绍第一步。对多个关系的合并可分解成两个关系的逐步合并，如有 3 个关系 R_1 ， R_2 与 R_3 ，它合并过程是先将 R_1 与 R_2 合并成 R_4 ，然后再将 R_4 与 R_3 合并成最终结果 R_5 。

因此，对关系数据库的查询可以分解成三个基本定位操作与一个查询操作：

- 一个关系内的属性指定
- 一个关系内的元组选择
- 两个关系的合并
- 查询操作

2. 数据删除

数据删除的基本单位是元组，它的功能是将指定关系内的指定元组删除，它也分为定位与操作两部分，其中定位部分只需要横向定位而无需纵向定位，定位后即是执行删除操作，因此数据删除可以分解为两个基本操作：一个关系内的元组选择；关系中元组删除操作。

3. 数据插入

数据插入仅对一个关系而言，在指定关系中插入一个或多个元组，在数据插入中不需定位，仅需作关系中元组插入操作。因此数据插入只有一个基本操作：关系中元组插入操作。

4. 数据修改

数据修改是在一个关系中修改指定的元组与属性值。

数据修改不是一个基本操作，它可以分解为两个更基本的操作：先删除需修改的元组，然后插入修改后的元组。

5. 关系操作小结

以上四种操作的对象都是关系，而操作结果也是关系，因此都是建立在关系上的操作。其次这四种操作可以分解成六种基本操作，这样，关系模型的数据操纵可以总结如下：

(1) 关系模型数据操作的对象是关系，而操作结果也是关系。

(2) 关系模型基本操作有如下六种：（其中三种为定位操作，三种为查询、插入及删除操作）

- 关系的属性指定
- 关系的元组选择
- 两个关系合并
- 一个关系的查询操作
- 关系中元组的插入操作
- 关系中元组的删除操作

6. 空值处理

在关系元组的分量中允许出现空值（null value）以表示信息的空缺，空值的含义如下：

- 未知的值
- 不可能出现的值

在出现空值的元组分量中一般可用 NULL 表示。目前一般关系数据库系统中都支持空值，但是它们都具有如下两个限制。

(1) 关系的主键中不允许出现空值

关系中主键不能为空值，这是因为主键是关系元组的标识，如主键为空值则失去了其标识的作用。

(2) 需要定义有关空值的运算

在算术运算中如出现有空值则其结果为空值，在比较运算中如出现有空值则其结果为 F（假），此外在作统计时，如 SUM, AVG, MAX, MIN 中有空值输入时其结果也为空值，而在作 COUNT 时如有空值输入则其值为 0。

2.4.2.1.3 关系中的数据约束

关系模型允许定义三类数据约束，它们是实体完整性约束、参照完整性约束以及用户定义的完整性约束，其中前两种完整性约束由关系系统自动支持，而用户定义的完整性约束则是用系统提供的完整性约束语言，由用户给出约束条件（它一般包括属性约束、关系约束以及关系间约束三种），并由系统自动检验，有关具体的说明可见第 4 章。

此外，有关关系的安全性约束与多用户的并发控制实际上也是一种数据约束，其具体说明可见第 4 章、第 5 章。

2.4.2.2 关系模型数据库管理系统

建立在关系模型基础上的数据库管理系统称为关系模型数据库管理系统或简称关系数

数据库管理系统 RDBMS (relational database management system), 关系数据库管理系统是一种基于二维表的系统, 它的主要内容与功能如下:

1. 二维表定义功能

关系数据库管理系统具有能定义二维表结构的能力, 它包括属性定义、表定义、主码定义以及相应的视图定义功能, 关系数据库管理系统提供模式定义语言以实现此类功能。

2. 二维表的操纵功能

关系数据库管理系统具有对一表或多表作查询的能力, 以及对表中元组作增、删、改操作的能力, 关系数据库管理系统提供数据操纵语言以实现此类功能。

3. 完整性约束功能

关系数据库管理系统具有能对表中数据作三类完整性约束的能力, 它包括实体完整性约束, 参照完整性约束以及用户自定义完整性约束, 而所定义的约束可以用关系数据库管理系统所提供的数据库控制语言中的完整性约束语句定义, 而所有完整性约束检验则由系统自动完成。

4. 安全功能

安全是一种约束, 它为多个用户访问数据提供必要的限制, 此种限制由数据库管理系统所提供的数据库控制语言中的安全性语句定义, 并由系统自动检验。

5. 并发控制功能

并发控制也是一种约束, 它为多个用户间并发运行提供必要的限制, 此种限制由关系数据库管理系统所提供的数据库控制语言中的并发控制语句定义, 并由系统自动作控制。

此外, 有关数据受破坏后的恢复, 也可用数据库控制语句定义并由系统自动控制恢复。

6. 数据交换

关系数据库中的数据可以与外界数据主体进行数据交换, 以达到数据应用的目的。

7. 数据服务功能

关系数据库管理系统提供多种服务功能, 如:

- (1) 为数据录入提供的数据库加载功能;
- (2) 为数据库管理提供的拷贝、转储及重组的功能;
- (3) 为 DBA 提供的数据库监视、性能监测以及数据库调优的功能;
- (4) 为数据库操纵提供多种函数与过程。

有关关系模型我们将会在第 3、4、5、6、7 章中作专门介绍。

2.4.3 面向对象模型与面向对象数据库管理系统

在面向对象模型中其逻辑模型与概念模型是一致的, 这主要由于在逻辑一级上存在着面向对象数据库管理系统, 它能有效、无缝的将面向对象概念模型转换成数据库系统一级。面向对象数据库管理系统即是以面向对象模型为核心的数据库管理系统, 简称 OODBMS (object oriented database management system), 在此系统中, 数据基本单位是对象, 而数据的处理单位是类, 类包括数据、相应操作及约束, 它们构成一个完整的封装体。在系统中有很多类, 类间存在着多种继承、聚合的关联, 它们构成了面向对象的数据模型。该模型将数据与数据间、数据与操作/约束间有机统一成一体, 有效地解决了关系模型中数据间关联语义简单及数据与操作/约束间不统一的缺点。OODBMS 的内容与功能如下:

- (1) 类管理

类管理主要是对数据模式及其操作的管理，它包括类定义、类继承与聚合的定义、类删除、类模式演化（即类模式的改变）等内容，用类管理可以定义类结构并且还可以改变类结构。

(2) 对象管理

对象管理主要是对类中数据的管理，它包括类中数据的查询以及相应的增、删、改的管理等内容。

(3) 对象控制管理

对象控制管理主要对类中数据对象作约束控制，其内容包括完整性约束控制、安全性约束控制以及并发控制等。

(4) 数据服务

与关系数据库管理系统的服务功能类似，面向对象数据库管理系统也提供相应的多种数据服务功能。

目前常用的面向对象数据库语言有 ODMG 所推荐的 OQL 语言及 ISO 的 SQL-3 语言。

目前常用的面向对象数据库管理系统有：Object store, Ontos 及 O₂ 等。有关面向对象模型我们将会在第 11 章中作专门介绍。

*2.4.4 谓词模型及知识库系统

在谓词模型中其逻辑模型与概念模型是一致的，这主要由于在逻辑一级上存在着知识库系统，它能有效、无缝的将谓词模型转换成数据库系统一级，而知识库系统即是以谓词模型为核心的数据库系统简称 KBS (knowledgebase system)。

在知识库系统中数据模式定义、数据操纵及数据约束都用谓词逻辑公式中的 DATALOG 表示，其中数据模式及约束均是系统中的已知部分，此外，用谓词表示的数据也是已知部分，而数据操纵则是系统中所需求证的部分，因此，在知识库系统中所需完成的功能即是由系统的已知部分出发，通过一种推理机制控制以获得系统中的求证部分，这种推理机制称为推理引擎 (inference engine)，下面的图 2.20 给出了这种示意图。

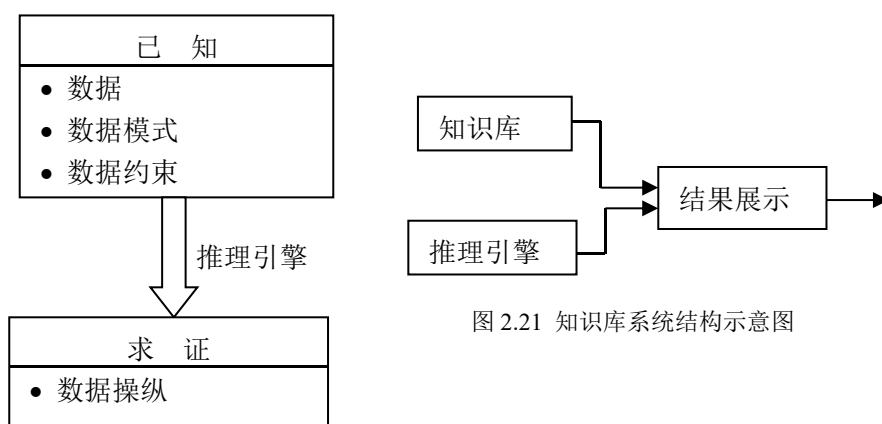


图 2.20 推理引擎示意图

图 2.21 知识库系统结构示意图

在知识库系统中，由谓词及谓词公式所表示的已知部分构成了知识库，再通过推理引

引擎可以获得求证结果（即数据操纵结果）。这样，有关知识库系统可由知识库及基于知识库的推理引擎以及结果表示这三部分所组成，它们构成了如图 2.21 所示的知识库系统结构图。

下面用一个例表示知识库系统。

例 2.11 在 2.3.4 中表示的学生数据库中可构造如下图 2.22 所示的知识库系统：

下面简单介绍知识库系统的功能与内容：

（1）知识库的建立

知识库由数据、数据模式及数据约束三部分组成，它们都用谓词公式中的 DATALOG 表示，由于谓词公式在人工智能中可称为知识，因此该存贮谓词公式的场所就称为知识库。在知识库管理系统中首要的任务是建立有关知识库，这个知识库按一定规则以持久性形式存贮知识并便于查询及增、删、改。

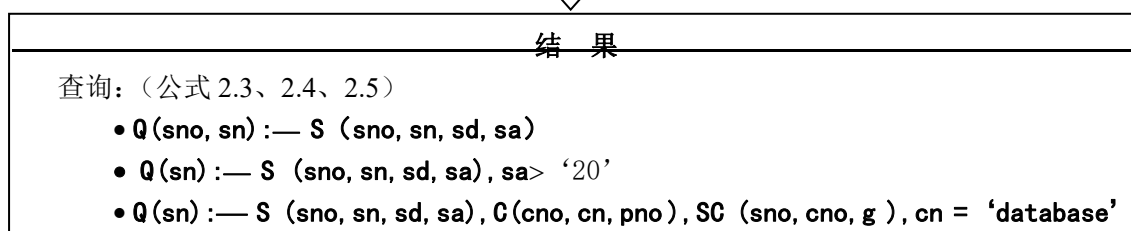
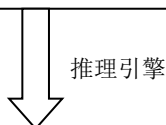
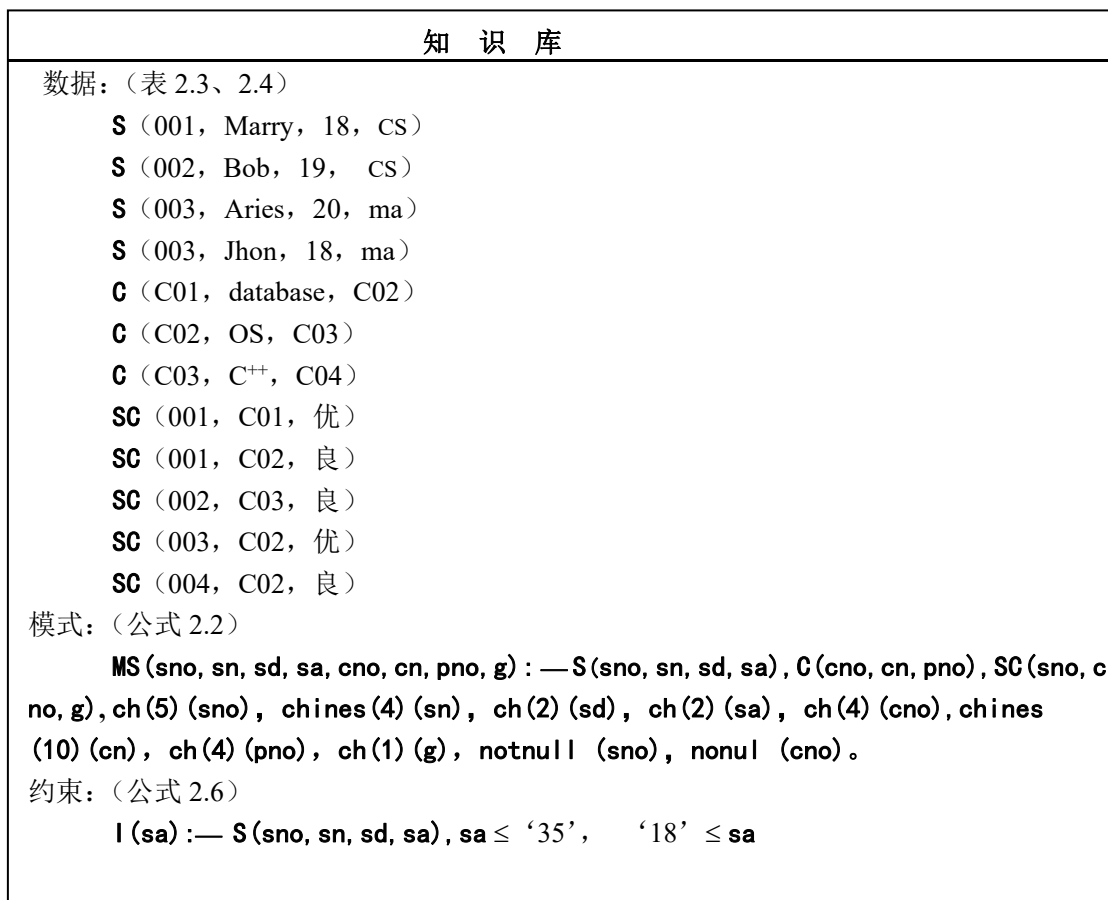
（2）推理引擎

目前有多种方法可以对知识库中的知识作推理并最终形成结果，常用的如基于归结原理的推理方法，其具体介绍可见后面的知识库这一章。

（3）结果展示

知识库中知识推理后可得到结果，常见的如查询结果，它们可以用多种可视化的形式表示，如报表、图示、文字等等形式，也可以通过网络发布。

有关谓词模型我们将会在第 11 章中作专门介绍。



计算机世界是计算机系统与相应的操作系统的总称，在概念世界与信息世界所表示的概念、方法以及数据结构及数据操纵最终均用计算机世界所提供的手段和方法实现。计算机世界一般用物理模型表示。物理模型主要是指操作系统的文件级，以及计算机系统的磁盘级。

2.5.1 计算机中的磁盘

在现在的数据库管理系统中，往往采用原始磁盘 (raw disk) 方法，即数据库管理系统可以直接管理磁盘，磁盘的数据存/取单位如下：

(1) 块 (block)。内/外存交换数据的基本单位，它又称物理块或磁盘块，它的大小有 512 字节、1024 字节、2048 字节等。

(2) 卷 (volume)。磁盘设备的一个盘组称一个卷。

在计算机所提供的磁盘设备基础上，经操作系统包装可以提供若干原语与语句供用户操作使用，如对磁盘的 Get (取)、Put (存) 操作，这是一种简单的存取操作，其中“取”操作的功能是将磁盘中的数据以块为单位取出后放入指定的内存缓冲区，而“存”操作功能则相反。

2.5.2 文件系统

1. 文件系统的组成

文件系统是实现数据库系统的直接物理支持，文件系统的基本结构由项、记录、文件及文件集合等四个层次组成。

(1) 项 (item)。项是文件系统中最小基本单位，项内符号是不能继续分割的，否则，就没有任何逻辑含义了。如城市名南京、上海均是项，我们无法将上海进一步分割成“上”与“海”，因为这样将失去完整的逻辑含义。

(2) 记录 (record)。记录由若干项组成，记录内的各项间是有内在语义联系的，如一张电影票内的所有有关项构成了一个有关电影票的完整记录信息，任何单独或部分项均无法得到关于电影票的完整语义，它只是一些不得要领的知识。

记录有型与值的区别，如电影票这一记录，其型是影院名称、映出日期、时间、座位号，而其值则是解放电影院、1999 年 5 月 20 日 21 时 30 分、楼下第 13 排第 3 号，可以用表 2.9 表示。

(3) 文件 (file)。文件是记录的集合。一般讲，一个文件所包含的记录都是同型的。每个文件都有文件名。

表 2.9 记录的型与值

影院名称	映出日期			时间		座位号		
	年	月	日	时	分	楼上/下	排号	位号
解放	1999	5	20	21	30	下	13	3

(4) 文件集 (file set) 若干个文件构成了文件集。

2. 文件的操作

文件有若干操作，一般的操作有如下五种：

- 打开文件
- 关闭文件
- 读记录
- 写记录
- 删除记录

3. 提高文件读写操作效率的方法

(1) 索引 (index)。索引起源于日常生活中的目录，如图书目录、产品目录，它在提高查阅速度方面有很大作用。在文件系统中为提高查询速度需要在文件的记录与其物理地

址（即磁盘块）间建立一张对应关系表以便于快速查找，这就是索引。

索引一般也是一个文件。当文件中记录数很大时，索引本身也还需要建立索引，这叫二级索引。

文件中的索引对提高文件的记录查找速度有很大作用，目前它被广泛使用。

由于数据库系统的直接物理基础是文件系统，因此在数据库系统中也广泛地使用索引以提高数据查找速度。

(2) hash 法。hash 法是一种函数转换法，其主要思想是通过一个 hash 函数将要查找的记录转换成该记录所在的物理地址，这是另一种提高查找速度的方法，它目前在文件系统以及在数据库系统中使用得也较为普遍。

(3) 集簇 (cluster)。在记录查找中往往需要按某项的项值查找，为提高查找速度，有必要将具有相同或相邻值的项值聚集在相同块内或圆柱体内以减少调盘次数，这就称为集簇。集簇在数据库系统中应用比较广泛，特别是在关系数据库中尤为如此。

2.5.3 逻辑模型的物理存储结构

信息世界中的各类模型均可通过计算机世界得到实现，我们现以关系模型为例，构造它在计算机世界中的物理存储结构。如表 2.10 所示。

表 2.10 关系模型与文件系统

关系模型	文件系统
属 性	项
元 组	记 录
关系模式	记录型
关 系	文 件
关系数据库	文件集

表 2.10 给出了关系模型与文件系统间的大致关系，在具体处理时还可能会有很大变化。有关数据库的物理模型我们还将将在第七章‘数据库物理组织’中作详细介绍。

习题二

- 2.1 什么叫数据模型，它分哪几种类型？
- 2.2 试区别数据模型与数据模式。
- 2.3 试述数据模型 4 个世界的基本内容。
- 2.4 试介绍 E-R 模型、EE-R 模型、面向对象模型及谓词模型，并各举例说明。
- 2.5 试比较面向对象模型、谓词模型及关系模型之优缺点。
- 2.6 试说明关系模型的基本结构与操作。
- 2.7 目前流行的 RDBMS 有哪些你比较熟悉，试介绍其特点。
- 2.8 请你画出某图书馆阅览部门的书刊、读者及借阅三者间的 E-R 模型。其中书刊属性为书刊号、书刊名、出版单位，而读者属性为读者名及读者姓名，其中一读者可借阅多种书刊而一种书刊可以被多个读者借阅。
- 2.9 设有一图书出版销售系统，其中的数据有：图书的书号、书名及作者姓名；出版社名称、地

址及电话；书店名称、地址及其经销图书的销售数量。

其中图书、出版社及书店间满足如下条件：

每种图书只能由一家出版社出版；

每种图书可由多家书店销售；

每家书店可以经销多种图书。

(1) 请画出该数据库的 E-R 图

(2) 在该 E-R 图中必须标明联系间的函数关系

2.10 设有一车辆管理系统，其中的数据有：

车辆号码、名称、型号；

驾驶员身份证号、姓名、地址、电话；

驾驶证号、发证单位。

其中车辆、驾驶员及驾驶证间满足如下条件：

一辆车可以由多个驾驶员驾驶；

每个驾驶员可以驾驶多辆车；

每个驾驶员可以有多个驾驶证；

每个驾驶证只能供一个驾驶员使用。

请设计该数据库的 E-R 图，并给出联系间的函数关系。

2.11 在上例中驾驶员又可分为小车司机与卡车司机，它们均拥有驾驶员的属性，同时小车司机还有属性：驾龄，卡车司机有属性：所属单位，在增加了上述情况后请画出车辆管理系统的 EE-R 图。

2.12 在习题 2.9 的图书出版销售系统中请构造谓词模型。

2.13 一人事档案中有干部、职工，干部又有高级干部与一般干部，其中职工有属性：职工号、职工姓名、籍贯、出生年月，而干部又增加属性：职务，高级干部又增加：参加工作年份，用 EE-R 图画出它们间的关系。

2.14 试说明数据模型的四个世界间的转化关系。

2.15 试比较 E-R 模型与 EE-R 模型间的关系，再比较 EE-R 模型与面向对象模型间的关系。

第二章复习指导

本章讨论数据模型，它是数据库系统的核心，读者学习后对数据库系统的本质内容应有了解。

1. 数据模型基本概念

- 数据模型是数据库特征的抽象。
- 数据模型描述数据结构、定义其上操作及约束条件。
- 数据模型分三个层次：概念模型、逻辑模型与物理模型。
- 数据模型的二维结构图。

	数据结构	操纵	约束
概念层			
逻辑层			
物理层			

2. 概念模型

(1) 四种模型

- E-R 模型
- EE-R 模型
- 面向对象模型
- 谓词模型

(2) 四种模型间的关系

- E-R、EE-R 与面向对象模型间具有包含关系，谓词模型具有最大的包含力。

3. 逻辑模型

(1) 四种逻辑模型

- 关系模型
- 面向对象模型
- 对象-关系模型
- 谓词模型

(2) 四种逻辑模型间的关系

4. 物理模型

5. 概念模型、逻辑模型与数据库管理系统

- E-R 模型——关系模型——关系数据库管理系统。
- 面向对象模型——面向对象模型——面向对象数据库管理系统。
- EE-R 模型——对象-关系模型——对象-关系数据库管理系统。
- 谓词模型——谓词模型——知识库管理系统。

6. 本章的重点内容

- 模型基本概念
- E-R 方法与 E-R 图
- 关系模型

第三章 关系数据库系统

本章主要讨论关系模型数据库系统，其中包括它的基本原理和理论以及标准语言 SQL 等。

3.1 关系数据库系统概述

关系数据库系统是基于关系模型的数据库系统，它由 E.F.Codd 于 1970 年提出，1976 年以后相继出现了实验性及商品化系统如 System-R、Ingres、QBE 等。70 年代末以后所问世的数据库产品大多为关系模型，并逐渐替代网状、层次模型数据库系统而成为主流数据库系统。关系数据库系统的崛起并迅速占领市场与它明显的优越性有关。一般认为，关系数据库系统具有以下优点：

(1) 数据结构简单。关系数据库系统中采用统一的二维表作为数据结构，不存在复杂的内部连接关系，具有高度简洁性与方便性。

(2) 用户使用方便。关系数据库系统数据结构简单，它的使用不涉及系统内部物理结构，用户不必了解，更不需干预系统内部组织，所用数据语言均为非过程性语言，因此操作、使用均很方便。

(3) 功能强。关系数据库系统能直接构造复杂的数据模型，特别是多联系间的构模能力，它可以一次获取一组元组，它可以修改数据间联系，同时也可以有一定程度修改数据模式的能力。此外，路径选择的灵活性，存储结构的简单性都是它的优点。

(4) 数据独立性高。关系数据库系统的组织、使用由于不涉及物理存贮因素，不涉及过程性因素，因此数据的物理独立性很高，数据的逻辑独立性也有一定的改善。

(5) 理论基础深。Codd 在提出关系模型时即以“关系理论”形式出现，在经过若干年理论探索后才出现产品，因此，关系数据库系统的特点之一是以理论“引导”产品。目前的关系数据库系统一般建立在逻辑与代数基础上，由于有理论工具的支撑使得对关系数据库系统的进一步研究与提高有了可靠的保证，如查询优化、知识库系统的研究即是以关系数据库的理论为基础的。

当然，关系数据库系统也有一些不足之处，如它对事务处理领域应用效果较好，但对非事务性应用及分析领域应用尚嫌不足等。

目前关系数据库系统已经成熟，其产品全方位的向纵深发展，主要表现如下几个方面：

(1) 可移植性。目前，大量的产品能同时适应多个机种与多个操作系统，如 Oracle 能适应 70 多种操作系统。

(2) 标准化。数据库语言的标准化工作经过多年的努力后，目前以 SQL 代表的结构化查询语言已陆续被美国标准化组织 ANSI、国际标准化组织 ISO 以及我国标准化组织确定为关系数据库使用的标准化语言，从而完成了它的使用的统一性，这被称为是一次关系数据库领域的革命。而其中 SQL'92 又被认为是典型的关系数据库系统语言。

(3) 开发工具。由于数据库在应用中大量使用，用户对它的直接操作的需要，要求不仅有数据定义、操纵与控制等作用，还需要有大量的用户界面生成以及开发的工具软件以利用户开发应用。因此，自 20 世纪 80 年代以来，关系数据库所提供的软件还包括有大量的用户界面生成软件以及开发工具。如 Oracle Developer-2000，微软的 VB 以及

PowerBuilder, Delphi 等。

(4) 分布式功能。由于数据库在计算机网络上的大量应用以及数据共享的要求, 数据库的分布式功能已在应用中成为急需, 因此目前多数关系数据库系统都提供有此类功能, 它们的方式有数据库远程访问、客户-服务器方式、浏览器-服务器方式。

(5) 开放性。现代关系数据库系统大都具有较好的开放性, 能与不同的数据库、不同的应用接口并能不断的扩充与发展, 一般关系数据库系统都具有通用的 ODBC 与 JDBC 接口以及快速的专用接口。

3.2 关系数据库系统的衡量准则

关系数据库系统自诞生以来, 为区分关系系统出现过很多衡量标准, 其中以 Codd 分别于 1974 年及 1985 年提出的准则最为著名, 它也是目前衡量关系数据库的重要标准。Codd 于 1974 年在 IFIP 会议上提出了关系数据库的 6 个准则, 在 11 年以后, Codd 于 1985 年在 “Computer World” 杂志上提出了完全关系型的 12 条严格标准:

准则 0 基础准则。一个关系数据库管理系统 (RBMS) 必须能完全通过它的关系能力来管理数据库, 这是关系数据库管理系统的最基本的准则, 任何一个 DBMS 必须满足准则 0。

准则 1 信息准则。关系数据库的所有信息都应能在逻辑一级惟一地用表中的值显示表示。同时, 数据库的结构描述也应在逻辑上组织成关系形式。这一条表示关系数据库的数据模式具有关系表形式。

准则 2 确保访问准则。关系数据库管理系统应能保证用逻辑方式依靠表名、关键字值与列名的组合访问数据库每一个数据的原始值。这一条给出了访问关系数据库的基本原则。

准则 3 空值的关系处理准则。在数据库中经常会出现 “无意义” 或 “当前未知” 的值, 如外星人的性别, 如外星人的性别, 如未婚者的配偶值, 这些值统称为空值 (Null value), 空值的出现导致了处理的复杂化, 因此在处理中要考虑到支持系统处理空值的能力。

准则 4 基于关系模型的动态联机数据字典。数据库中应有数据字典并应与一般的数据采用同样的访问方式。

准则 5 统一的数据子语言准则。关系数据库可以支持多种语言和终端使用方式, 但必须至少有一种语言能全面支持下列功能。

1) 数据定义功能: 具有能定义数据描述中每个关系, 包括属性、域、主关键字等的的能力。

2) 视图定义功能: 具有能定义视图的能力。视图是一种虚拟的关系, 它可由用户通过数据库中的基本关系导出, 它给出了用户对数据库的需求范围。

3) 数据操纵: 具有能支持交互使用方式与程序使用方式 (自含的或嵌入的) 数据操纵能力。它包括完备的关系操作, 这种操作可以与关系代数方式等价。

4) 完整性约束能力: 具有能定义与检验数据完整性的能力。

5) 授权机制: 具有能定义一定安全机制的能力。

6) 事务处理能力: 具有一定的并发控制能力。

准则 6 视图更新准则。用户能对视图作查询操作。此外, 它还能对视图作一定能力更新操作, 如在视图上的增、删、改操作。

准则 7 高级插入、修改及删除操作。

准则 8 物理数据独立性。

准则 9 逻辑数据独立性。

准则 10 数据完整性准则。应具有能支持三类数据完整性，它包括实体完整性、关联完整性以及针对具体数据库所定义的用户完整性约束。

准则 11 分布独立性。当数据由集中存储变为分布存储时或数据重新分布时，应用程序或用户终端的工作应保持不受影响。

准则 12 无损害原则。如果某关系数据库管理系统允许使用每次处理一个元组的低级语言，则此种使用不能损害数据完整性准则。

根据上面十二条准则，我们把目前市场上所出现的关系数据库产品分成为下面三种：

(1) 半关系型系统。这些系统大都采用关系作为基本数据结构，但不提供完备数据子语言，数据独立性差，无视图功能与空值概念，它只满足准则中的少量的原则，早期的关系数据库产品如 dBASE-III 及 Foxbase 等均属此类产品。

(2) 基本关系型系统。这些系统均采用关系作为基本数据结构，有完备的数据子语言，有一定的数据独立性，并有一定空值处理能力，有视图功能，它满足准则之大部分条件，但不满足准则之所有条件。目前，大多数关系数据库产品均属此类，如 DB2, Oracle, Sybase 等。

(3) 完全关系型系统。凡严格符合 12 条准则的关系型系统称完全关系型系统，这是一种理想化的系统，到目前为止还尚未有完全关系型系统出现。

3.3 关系模型数学理论——关系代数

关系数据库系统是建立在数学理论基础之上的，目前很多数学理论可以表示关系模型，其中最为著名的是关系代数 (relational algebra)。关系代数是使用代数方法表示关系模型。此外还有关系演算 (relational calculus)，它是用谓词逻辑方法表示关系模型，在本节中主要介绍关系代数，而关系演算的主要思想可以参阅 2.3.4 中谓词模型及第 12 章中“知识库系统”。

3.3.1 关系的表示

关系是由若干个不同元组所组成，因此关系可视为元组的集合。一个 n 元关系的元组可视为一个 n 元有序组，因此，我们说 n 元关系是一个 (n 元) 有序组的集合。

设有一 n 元关系 R ，它有 n 个域，分别是 D_1, D_2, \dots, D_n 。此时，它的笛卡尔乘积是：

$$D_1 \times D_2 \times \dots \times D_n$$

这个笛卡尔乘积给出了这样一个集合，它的每个元素都是具有如下形式的 n 元有序组：

$$(d_1, d_2, \dots, d_n) \quad d_i \in D_i (i = 1, 2, \dots, n)$$

它与 n 元关系 R 有如下之联系： $R \subseteq D_1 \times D_2 \times \dots \times D_n$

因此我们说， n 元关系 R 是 n 元有序组的集合，是它的域的笛卡尔乘积的子集。

例 3.1 表 3.1 所示的关系 S 可用 5 个 4 元有序组所组成的集合表示，它是一个四元关系。

$$S = \{ (13761, \text{王诚}, \text{MA}, 21), (13762, \text{徐一飞}, \text{CS}, 22), (13763, \text{李峰}, \text{CS}, 20), (13764, \text{赵建平}, \text{CS}, 18), (13765, \text{申桂花}, \text{MA}, 21) \}$$

表 3.1 关系 S

sno	sn	sd	sa
13761	王 诚	MA	21
13762	徐一飞	CS	22
13763	李 峰	CS	20
13764	赵建平	CS	18
13765	申桂花	MA	21

3.3.2 关系操纵的表示

1. 关系模型的基本操作

前面已经讲过，关系模型有四种基本操纵，它可以分解成六种基本操作。

(1) 关系的属性指定：指定一个关系内的某些属性，用它确定关系这个二维表中的列，它主要用于检索或定位。

(2) 关系元组选择：用一个逻辑表达式给出关系中满足此表达式的元组，用它以确定关系这个二维表的行，它主要用于检索或定位。要注意的是，这种表示方法是一种隐式表示方法。

用上述两种操作即可确定一张二维表内满足一定行、列要求的数据。

(3) 两个关系的合并：将两个关系合并成一个关系。用此操作可以不断合并从而可以将若干个关系合并成一个关系，以建立多个关系间的检索与定位。

用上述三个操作可以进行多个关系的定位。

(4) 关系的查询：在一个关系或多个关系间作查询，查询的结果也是关系。

(5) 关系中元组的插入：在关系中增添一些元组，用以完成插入。

(6) 关系中元组的删除：在关系中删除一些元组，用以完成删除与修改。

2. 关系模型的运算

分析这些基本操作后可以发现：这些操作的基本对象都是一个或两个关系；一个或两个关系经操作后所得的结果仍是一个关系。可以将这些操作看成是对关系的运算。而关系是 n 元有序组的集合，因此，可以将操作看成是集合的运算。

下面讨论这些基本操作是些什么运算。

(1) 插入。设有关系 R 插入若干元组，这些元组组成关系 R' ，由传统集合论可知，可用集合并运算表示，即可写为： $R \cup R'$

(2) 删除。设有关系 R 删除一些元组，这些元组组成关系 R' ，由传统集合论可知，可用集合差运算表示，即可写为： $R - R'$

(3) 修改。修改关系 R 内的元组内容可用下面方法实现：

设需修改之元组构成关系 R' ，则先作删除，得： $R - R'$

设需修改后的元组构成关系 R'' ，此时我们将其插入，从而得到结果：

$$(R - R') \cup R''$$

(4) 查询。无法用传统的集合论方法去表示用于查询的三个操作，从而要引入一些新的运算。

1) 投影 (projection) 运算。对于关系内的域指定可引入新的运算，称为投影运算。投影运算是一个一元运算，一个关系通过投影运算 (并由该运算给出所指定的属性) 后仍为一个关系 R' 。 R' 是这样一关系，它是 R 中投影运算所指出的那些域的列所组成的关系。设 R 有 n 个域: A_1, A_2, \dots, A_n ，则在 R 上对域 $A_{i_1}, A_{i_2}, \dots, A_{i_m}$ ($A_{i_j} \in \{A_1, A_2, \dots, A_n\}$) 的投影可表示成为下面的一元运算:

$$\pi_{A_{i_1}, A_{i_2}, \dots, A_{i_m}}(R)$$

例 3.2 对表 3.1 所示的关系 S 有:

$$\pi_{sn,sa}(S) = \{(王诚, 21), (徐一飞, 22), (李峰, 20), (赵建平, 18), (申桂花, 21)\}$$

$$\pi_{sn}(S) = \{(王诚), (徐一飞), (李峰), (赵建平), (申桂花)\}$$

2) 选择 (selection) 运算。对关系内有序组的选择可引入另一新的运算——选择运算。选择运算也是一个一元运算，关系 R 通过选择运算 (并由该运算给出所选择的逻辑条件) 后仍为一个关系。这个关系是由 R 中那些满足逻辑条件的有序组所组成。设关系的逻辑条件为 F ，则 R 满足 F 的选择运算可写成为: $\sigma_F(R)$

逻辑条件 F 是一个逻辑表达式，它可以具有 $\alpha \theta \beta$ 的形式，其中 α, β 是域 (变量) 或常量，但 α, β 又不能同为常量， θ 是比较运算符，它可以是 $<, >, \leq, \geq, =$ 及 \neq 。 $\alpha \theta \beta$ 叫基本逻辑条件；也可由若干个基本逻辑条件经逻辑运算 \wedge (并且) 和 \vee (或者) 构成，称为复合逻辑条件。

例 3.3 表 3.1 所示的关系 S 中找出年龄大于 20 岁的所有元组，可以写成为:

$$\sigma_{sa>20}(S) = \{(13761, 王诚, MA, 21), (13762, 徐一飞, CS, 22), \\ (13765, 申桂花, MA, 21)\}$$

例 3.4 在关系 S 中找出年龄大于 20 岁且在数学系 (MA) 学习的学生，可以写成为:

$$\sigma_{(sa>20) \wedge (sd=MA)}(S) = \{(13761, 王诚, MA, 21), (13765, 申桂花, MA, 21)\}$$

有了上述两个运算后，对一个关系内的任意行、列的数据都可以方便地找到，下面给出一例，见例 3.5。

例 3.5 在关系 S 中查出所有年龄大于 20 岁的学生姓名，它可以表示如下:

$$\pi_{sn} \sigma_{sa>20}(S) = \{(王诚), (徐一飞), (申桂花)\}$$

3) 笛卡尔乘积 (cartesian product) 运算。对于两个关系的合并操作可以用笛卡尔乘积表示。设有关系 R 和 S , 它们分别为 n, m 元关系, 并分别有 p, q 个元组, 此时, 关系 R 与 S 经笛卡尔乘积所得的关系 T 是一个 $n+m$ 元关系, 它的有序组个数是 $p \times q$, T 的有序组是由 R 与 S 的有序组组合而成。关系 R 与 S 的笛卡尔乘积可写为:

$R \times S$

设有如表 3.2 所示的两个关系 R, S , 则 R 与 S 的笛卡尔乘积 $T=R \times S$ 可用表 3.3 表示。

表 3.2 关系 R, S

R:												
<table border="1" style="display: inline-table; border-collapse: collapse; text-align: center;"> <tr><td>R_1</td><td>R_2</td><td>R_3</td></tr> <tr><td>a</td><td>b</td><td>c</td></tr> <tr><td>d</td><td>e</td><td>f</td></tr> <tr><td>g</td><td>h</td><td>i</td></tr> </table>	R_1	R_2	R_3	a	b	c	d	e	f	g	h	i
R_1	R_2	R_3										
a	b	c										
d	e	f										
g	h	i										

S:												
<table border="1" style="display: inline-table; border-collapse: collapse; text-align: center;"> <tr><td>S_1</td><td>S_2</td><td>S_3</td></tr> <tr><td>j</td><td>k</td><td>l</td></tr> <tr><td>m</td><td>n</td><td>o</td></tr> <tr><td>p</td><td>q</td><td>r</td></tr> </table>	S_1	S_2	S_3	j	k	l	m	n	o	p	q	r
S_1	S_2	S_3										
j	k	l										
m	n	o										
p	q	r										

表 3.3 $T=R \times S$

R_1	R_2	R_3	S_1	S_2	S_3
a	b	c	j	k	L
a	b	c	m	n	o
a	b	c	p	q	r
d	e	f	j	k	l
d	e	f	m	n	o
d	e	f	p	q	r
g	h	i	j	k	l
g	h	i	m	n	o
g	h	i	p	q	r

3.3.3 关系模型与关系代数

到此为止, 已经知道关系是一个 n 元有序组的集合, 而关系操纵则是集合上的一些运算。

(1) 投影: 一元运算, 可用 $\pi_{A_1, A_2, \dots, A_m}(R)$ 表示。

(2) 选择: 一元运算, 可用 $\sigma_F(R)$ 表示。

(3) 笛卡尔乘积: 二元运算, 可用 $R \times S$ 表示。

(4) 并: 二元运算, 可用 $R \cup S$ 表示。

(5) 差: 二元运算, 可用 $S - R$ 表示。

这样在集合 A 上的二个一元运算及三个二元运算构成了一个代数系统:

$(A, \pi, \sigma, \times, \cup, -)$

这个代数系统称之为关系代数。

3.3.4 关系代数中的扩充运算

关系代数中上述五个运算是最基本的运算，但为操纵方便，还需增添一些运算，这些运算均可由基本运算导出，常用的扩充运算有交、除、联结及自然联结等四种，现分别介绍如下。

1. 交 (Intersection) 运算

交运算是一个传统集合运算，关系 R 与 S 经此运算后所得到的关系是由那些既在 R 内又在 S 内的有序组所组成。

关系 R 与 S 的交运算可写成：

$$R \cap S$$

例 3.6 表 3.4 (a), (b) 所示的关系 R 与 S 经交运算后可以得到如表 3.5 所示的关系。

表 3.4 关系 R, S

R:	S:																																
<table style="width: 100%; border-collapse: collapse; border: none;"> <thead> <tr> <th style="border: none; padding: 2px 10px;">A</th> <th style="border: none; padding: 2px 10px;">B</th> <th style="border: none; padding: 2px 10px;">C</th> <th style="border: none; padding: 2px 10px;">D</th> </tr> </thead> <tbody> <tr> <td style="border: none; padding: 2px 10px;">1</td> <td style="border: none; padding: 2px 10px;">2</td> <td style="border: none; padding: 2px 10px;">3</td> <td style="border: none; padding: 2px 10px;">4</td> </tr> <tr> <td style="border: none; padding: 2px 10px;">2</td> <td style="border: none; padding: 2px 10px;">2</td> <td style="border: none; padding: 2px 10px;">5</td> <td style="border: none; padding: 2px 10px;">7</td> </tr> <tr> <td style="border: none; padding: 2px 10px;">9</td> <td style="border: none; padding: 2px 10px;">0</td> <td style="border: none; padding: 2px 10px;">3</td> <td style="border: none; padding: 2px 10px;">8</td> </tr> </tbody> </table>	A	B	C	D	1	2	3	4	2	2	5	7	9	0	3	8	<table style="width: 100%; border-collapse: collapse; border: none;"> <thead> <tr> <th style="border: none; padding: 2px 10px;">A</th> <th style="border: none; padding: 2px 10px;">B</th> <th style="border: none; padding: 2px 10px;">C</th> <th style="border: none; padding: 2px 10px;">D</th> </tr> </thead> <tbody> <tr> <td style="border: none; padding: 2px 10px;">2</td> <td style="border: none; padding: 2px 10px;">2</td> <td style="border: none; padding: 2px 10px;">3</td> <td style="border: none; padding: 2px 10px;">8</td> </tr> <tr> <td style="border: none; padding: 2px 10px;">1</td> <td style="border: none; padding: 2px 10px;">2</td> <td style="border: none; padding: 2px 10px;">3</td> <td style="border: none; padding: 2px 10px;">4</td> </tr> <tr> <td style="border: none; padding: 2px 10px;">9</td> <td style="border: none; padding: 2px 10px;">1</td> <td style="border: none; padding: 2px 10px;">2</td> <td style="border: none; padding: 2px 10px;">3</td> </tr> </tbody> </table>	A	B	C	D	2	2	3	8	1	2	3	4	9	1	2	3
A	B	C	D																														
1	2	3	4																														
2	2	5	7																														
9	0	3	8																														
A	B	C	D																														
2	2	3	8																														
1	2	3	4																														
9	1	2	3																														

(a)

(b)

表 3.5 $R \cap S$

A	B	C	D
1	2	3	4

交运算并不是基本运算，它可由基本运算推导而得：

$$R \cap S = R - (R - S)$$

2. 除 (Division) 运算

除运算是一个非传统的集合运算，如果认为笛卡尔乘积运算为乘运算的话，那么这个运算就是它的逆运算，因此称为除运算。

$T = R \times S$ 则可将除运算写成 $T \div R = S$ 或 $T/R = S$ ，S 称为 R 除以 T 的商 (quotient)。由于除是采用的逆运算，因此除运算的执行是需要满足一定条件的。

设有关系 T、R，T 能被 R 除的充分必要条件是：

- (1) T 中的域包含 R 中的所有属性。
- (2) T 中有一些域不出现在 R 中。

在除运算中 S 的域由 T 中那些不出现在 R 中的域所组成，其有序组则由 R 中出现的所有元组在 T 中所对应的相同值所组成。

例 3.7 建立一组除法如表 3.6 所示，从这组除法可以清楚地看出除法的含义及商的内容。

表 3.6 3 个除法

T:

A	B	C	D
1	2	3	4
7	8	5	6
7	8	3	4
1	2	5	6
1	2	4	2

R:

C	D
3	4
5	6

C	D
3	4

C	D
3	4
5	6
4	2

S:

A	B
1	2
7	8

A	B
1	2
7	8

A	B
1	2

例 3.8 设关系 R (表 3.7 (a)) 给出了学生修读课程情况, 关系 S (表 3.7 (b)) 给出了所有课程号, 试找出修读所有课程的学生号。

表 3.7 学生修读课程的除法运算

R:	<table border="1" style="border-collapse: collapse;"> <thead> <tr> <th>sno</th> <th>cno</th> </tr> </thead> <tbody> <tr><td>S₁</td><td>C₁</td></tr> <tr><td>S₁</td><td>C₂</td></tr> <tr><td>S₂</td><td>C₁</td></tr> <tr><td>S₂</td><td>C₂</td></tr> <tr><td>S₂</td><td>C₃</td></tr> <tr><td>S₃</td><td>C₂</td></tr> </tbody> </table>	sno	cno	S ₁	C ₁	S ₁	C ₂	S ₂	C ₁	S ₂	C ₂	S ₂	C ₃	S ₃	C ₂	S:	<table border="1" style="border-collapse: collapse;"> <thead> <tr> <th>cno</th> </tr> </thead> <tbody> <tr><td>C₁</td></tr> <tr><td>C₂</td></tr> <tr><td>C₃</td></tr> </tbody> </table>	cno	C ₁	C ₂	C ₃	T:	<table border="1" style="border-collapse: collapse;"> <thead> <tr> <th>sno</th> </tr> </thead> <tbody> <tr><td>S₂</td></tr> </tbody> </table>	sno	S ₂
sno	cno																								
S ₁	C ₁																								
S ₁	C ₂																								
S ₂	C ₁																								
S ₂	C ₂																								
S ₂	C ₃																								
S ₃	C ₂																								
cno																									
C ₁																									
C ₂																									
C ₃																									
sno																									
S ₂																									
	(a)		(b)		(c)																				

此问题可以很方便地用除法解决, 修读所有课程的学生号可用 $T=R/S$ 表示。它的结果由表 3.7 (c) 表示。

除法运算不是基本运算, 它可以由基本运算推导而出。设关系 R 有域 A_1, A_2, \dots, A_n , 关系 S 有域 $A_{n-s+1}, A_{n-s+2}, \dots, A_n$, 此时有:

$$R \div S = \pi_{A_1, A_2, \dots, A_{n-s}}(R) - \pi_{A_1, A_2, \dots, A_{n-s}}((\pi_{A_1, A_2, \dots, A_{n-s}}(R) \times S) - R)$$

3. 联接 (join) 与自然联接 (natural join) 运算

用笛卡尔乘积可以建立两个关系间的联接, 但此种方法并不是一种好的方法, 因为这样所建立的关系是一个较为庞大的关系, 而且也并不符合实际操作的需要。在实际应用中一般两个相互联接的关系往往须满足一些条件, 所得到的结果也较为简单。因此, 对笛卡尔乘积作适当的限制, 以适应实际应用的需要。这样就引入了联接运算与自然联接运算。

联接运算又可称为 θ -联接运算, 这是一种二元运算, 通过它可以两个关系合并成一个大关系。设有关系 R, S 以及比较式 $i \theta j$, 其中 i 为 R 中域, j 为 S 中域, θ 含义同前。

此时可以将 R, S 在域 i, j 上的 θ -联接记为: $R \underset{i \theta j}{\infty} S$

它的含义可用下式定义:

$$R \underset{i \theta j}{\infty} S = \sigma_{i \theta j} (R \times S)$$

亦即是说, R 与 S 的 θ -联接是 R 与 S 的笛卡尔乘积再加上限制 $i \theta j$ 而成。显然, $R \underset{i \theta j}{\infty} S$

的有序组数远远少于 $R \times S$ 的有序组数。 $T = R \underset{D > E}{\infty} S$

所要注意的是, 在 θ -联接中, i 与 j 需具有相同域, 否则无法作比较。

θ -联接中如 θ 为 “=”, 此时称为等值联接, 否则称为不等值联接。如 θ 为 “<” 时称为小于联接, 如 θ 为 “>” 时称为大于联接。

例 3.9 设有关系 R 和 S 分别如表 3.8 (a), (b) 所示, 则 $T = R \underset{D > E}{\infty} S$ 为表 3.8 (c) 所示的关系, 而 $T' = R \underset{D = E}{\infty} S$ 为如表 3.8 (d) 所示的关系。

表 3.8 R 与 S 的 θ -联接实例

R:	<table style="display: inline-table; border-collapse: collapse;"> <tr><th>A</th><th>B</th><th>C</th><th>D</th></tr> <tr><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>3</td><td>2</td><td>1</td><td>8</td></tr> <tr><td>7</td><td>3</td><td>2</td><td>1</td></tr> </table>	A	B	C	D	1	2	3	4	3	2	1	8	7	3	2	1	S:	<table style="display: inline-table; border-collapse: collapse;"> <tr><th>E</th><th>F</th></tr> <tr><td>1</td><td>8</td></tr> <tr><td>7</td><td>9</td></tr> <tr><td>5</td><td>2</td></tr> </table>	E	F	1	8	7	9	5	2
A	B	C	D																								
1	2	3	4																								
3	2	1	8																								
7	3	2	1																								
E	F																										
1	8																										
7	9																										
5	2																										
	(a)		(b)																								

$T = R \underset{D > E}{\infty} S$	<table style="display: inline-table; border-collapse: collapse;"> <tr><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th><th>F</th></tr> <tr><td>1</td><td>2</td><td>3</td><td>4</td><td>1</td><td>8</td></tr> <tr><td>3</td><td>2</td><td>1</td><td>8</td><td>1</td><td>8</td></tr> <tr><td>3</td><td>2</td><td>1</td><td>8</td><td>7</td><td>9</td></tr> <tr><td>3</td><td>2</td><td>1</td><td>8</td><td>5</td><td>2</td></tr> </table>	A	B	C	D	E	F	1	2	3	4	1	8	3	2	1	8	1	8	3	2	1	8	7	9	3	2	1	8	5	2	$T' = R \underset{D = E}{\infty} S$	<table style="display: inline-table; border-collapse: collapse;"> <tr><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th><th>F</th></tr> <tr><td>7</td><td>3</td><td>2</td><td>1</td><td>1</td><td>8</td></tr> </table>	A	B	C	D	E	F	7	3	2	1	1	8
A	B	C	D	E	F																																								
1	2	3	4	1	8																																								
3	2	1	8	1	8																																								
3	2	1	8	7	9																																								
3	2	1	8	5	2																																								
A	B	C	D	E	F																																								
7	3	2	1	1	8																																								
	(c)		(d)																																										

但是, 在实际应用中最为常用的联接是 θ -联接的一个特例叫自然联接, 这主要是因为常用的两关系间联接大都满足条件:

- 两关系间有公共域

- 通过公共域的相等值进行联接

根据这两个条件我们建立自然联接运算。

设有关系 R, S, 关系 R 有域 A_1, A_2, \dots, A_n , 关系 S 有域 B_1, B_2, \dots, B_m , 它们间 $A_{i1}, A_{i2}, \dots, A_{ij}$

与 B_1, B_2, \dots, B_j 分别为相同域, 此时它们自然联接可记为: $R \bowtie S$

自然联接的含义可用下式表示:

$$R \bowtie S = \pi_{A_1, A_2, \dots, A_n, B_{j+1}, B_{j+2}, \dots, B_m} (\sigma_{A_{i1}=B_1 \wedge A_{i2}=B_2 \wedge \dots \wedge A_{ij}=B_j} (R \times S))$$

例 3.10 设关系 R 和 S 如表 3.9 (a), (b) 所示, 此时 $T = R \bowtie S$ 则为如表 3.9 (c) 所示。

表 3.9 R 与 S 的自然联接实例

R:	A	B	C	D	S:	D	E	T:	A	B	C	D	E
	1	2	3	4		5	1		2	4	2	6	4
	1	5	8	3		6	4		2	4	2	6	8
	2	4	2	6		7	3		1	1	4	7	3
	1	1	4	7		6	8						

至此, 引入了五种基本运算与四种扩充运算, 一共九种, 其中最常用的是五种, 它们是投影运算、选择运算、自然联接运算、并运算、差运算。一般讲有这五种运算就够了, 但需注意的是, 自然联接运算是实现两个关系合并的‘常用运算’但不是‘基本运算’。

3.3.5 关系代数实例

用关系代数表达式可以表示检索、插入、删除及修改等操作, 下面用几个例子来说明。在此之前先建立一个关系数据库, 称为学生数据库, 它由三个关系组成。它们的模式是:

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

其中 sno, cno, sn, sd, sa, cn, pno, g 是属性, 分别表示学号、课程号、学生姓名、学生系别、学生年龄、课程名、预修课程号、成绩; 而 S, C, SC 则为关系, 分别表示学生、课程、学生与课程联系。下面用关系代数表达式以表示在该数据库上的操作。

例 3.11 检索学生所有情况:

S

例 3.12 检索学生年龄大于等于 20 岁的学生姓名:

$$\pi_{sn} (\sigma_{sa \geq 20} (S))$$

例 3.13 检索预修课号为 C₂ 的课程的课程号:

$$\pi_{cno} (\sigma_{pno=C_2} (C))$$

例 3.14 检索课程号为 C, 且成绩为 A 的所有学生姓名:

$$\pi_{sn}(\sigma_{cno=C \wedge g=A}(S \bowtie SC))$$

注意 这是一个涉及到两个关系的检索, 此时需用联接运算。

例 3.15 检索 s₁ 所修读的所有课程名及其预修课号:

$$\pi_{cn,pno}(\sigma_{sno=s_1}(C \bowtie SC))$$

例 3.16 检索年龄为 23 岁的学生所修读的课程名:

$$\pi_{cn}(\sigma_{sa=23}(S \bowtie SC \bowtie C))$$

注意: 这是涉及到三个关系的检索。

例 3.17 检索至少修读为 S₅ 所修读的一门课的学生姓名。

这个例子比较复杂, 需作一些分析, 将问题分 3 步解决:

第 1 步, 取得 S₅ 修读的课程号, 它可以表示为:

$$R = \pi_{cno}(\sigma_{sno=S_5}(SC))$$

第 2 步, 取得至少修读为 S₅ 修读的一门课的学号:

$$W = \pi_{sno}(SC \bowtie R)$$

第 3 步, 最后得到结果为:

$$\pi_{sn}(S \bowtie W)$$

分别将 R, W 代入后即得检索要求的表达式:

$$\pi_{sn}(S \bowtie \pi_{sno}(SC \bowtie \pi_{cno}(\sigma_{sno=S_5}(SC))))$$

例 3.18 检索修读 S₄ 所修读的所有课程的学生姓名。

这个检索要求也可分为以下三步解决。

第 1 步, 取得 S₄ 所修读的课程号, 可表示为:

$$R = \pi_{cno}(\sigma_{sno=S_4}(SC))$$

第 2 步, 取得修读 S₄ 所修读的所有课程的学号, 可以表示为:

$$W = \pi_{sno,cno}(SC) \div R$$

第 3 步, 最后得到结果为:

$$\pi_{sn}(S \bowtie W)$$

将 R, W 作代人后即得检索要求的表达式:

$$\pi_{sn} (S \cap (\pi_{sno,cno} (SC) \div \pi_{cno} (\sigma_{sno=S_4} (SC))))$$

例 3.19 检索修读所有课程的学生学号:

$$\pi_{sno,cno} (SC) \div \pi_{cno} (C)$$

例 3.20 检索不修读任何课程的学生学号:

$$\pi_{sno} (S) - \pi_{sno} (SC)$$

例 3.21 在关系 C 中增添一门新课程:

$$(C13, ML, C3)$$

令此新课程元组所构成的关系为 R, 即有:

$$R = \{(C13, ML, C3)\}$$

此时有结果:

$$C \cup R$$

例 3.22 学号为 S₁₇ 的学生因故退学, 请在 S 及 SC 中将其除名:

$$SC - \sigma_{sno=S_{17}} (SC)$$

$$S - \sigma_{sno=S_{17}} (S)$$

例 3.23 将关系 S 中学生 S₆ 的年龄改为 22 岁:

$$(S - \sigma_{sno=S_6} (S)) \cup W$$

W 为修改后的学生有序组所组成的关系。

例 3.24 将关系 S 中的年龄均增加 1 岁:

$$S(sno, sn, sd, sa + 1)$$

3.4 关系数据库语言 SQL'92

3.4.1 SQL 概貌

关系数据库系统的数据语言有多种, 但在经过 10 余年的使用、竞争、淘汰与更新后, SQL 语言以其独特风格, 独树一帜, 成为国际标准化组织所确认的关系数据库系统标准语言。目前, SQL 语言已成为关系数据库系统所使用的惟一数据语言, 一般而言, 用该语言所书写的程序大致可以在任何关系数据库系统上运行。

SQL 语言又称结构化查询语言 (structured query language) 是 1974 年由 Boyce 和 Chamberlin 提出的并在 IBM 公司 San Jose 研究实验室所研制的关系数据库管理系统 System R 上实现了这种语言, 最初称为 SEQUEL, 接着 IBM 公司又实现了商用系统 SQL/DS 与 DB2, 其中 SQL/DS 是在 IBM 公司中型机环境下实现的, 而 DB2 则主要用于大型机环境。

SQL 语言在 1986 年被美国标准化组织 ANSI 批准为国家标准，1987 年又被国际标准化组织 ISO 批准为国际标准，并经修改后于 1989 年正式公布，称之为 SQL'89。此标准也于 1993 年被我国批准为中国国家标准。此后 ISO 陆续发布了 SQL'92、SQL'99 及 SQL'03 等版本。其中 SQL'92 又称 SQL-2，而 SQL'99 又称 SQL-3。目前，国际上所有关系数据库管理系统均采用 SQL 语言，它包括 DB2 以及 Oracle、SQL Servers、Sybase、Ingres、Informix 等关系数据库管理系统。

在 SQL 中 SQL'92 是一种典型的关系数据库语言，但是自 SQL'99 以后就具有某种面向对象的特色了，同时还有某些多媒体及工程方面的功能扩充，因此就关系数据库语言而言，我们一般指的是 SQL'92。

SQL 称为结构化查询语言，但是它实际上包括查询在内的多种功能，它包括数据定义、数据操纵（包括查询）和数据控制等三个方面。近年来还包括数据交换功能，SQL 所操作的对象称为基表（base table），它即是关系。SQL 是一种统一的语言，它们具有 DDL、DML 及 DCL 的功能，即首先它给出了的数据定义功能即是对模式、基表以及视图定义的功能。而 DML 给出了数据操纵与数据控制功能，其中 SQL 的数据操纵功能即是在基表上的查询、删除、插入、修改等功能以及数据交换功能。SQL 的数据控制功能即是基于基表上的完整性、安全性及并发控制等功能，而数据交换功能则是数据库与外界作数据交互的功能。

SQL 语言有多种使用方式：一种是联机交互使用方式，在此此种方式下，SQL 可以通过操作员与数据库进行独立交互；另一种是与应用程序连接使用方式，其最著名的是嵌入式使用方式，在此种方式下，它可以用某些高级程序设计语言（如 COBOL，C 等）为主语言，而 SQL 则被嵌入其中依附于主语言（称为嵌入式语言）。此外还有自含式使用方式，在此种方式下，SQL 自身包含有主语言的成份，因此可以自行组织应用程序（包含 DML），近年来发展的调用层接口方式可以在网络上将应用程序与数据库通过接口函数调用方式实现数据交换。在 Web 发展的今天更可以通过 Web 方式实现 XML 与数据库间的数据交换。

在本书中，本章主要介绍 SQL'92 的数据定义与数据操纵功能，在下一章中介绍 SQL 的安全性及完整性功能，而在第五章中则介绍 SQL 中的并发控制与故障恢复功能，最后在第六章中介绍 SQL 的多种数据交换方式。

本书主要以介绍 SQL'92 为主，SQL'92 内容分三级，它们是初级、中级与完全级，目前大部分主流数据库产品都与 SQL'92 的初级标准符合，因此本书也以介绍 SQL'92 初级标准为主，同时也吸收部分 SQL'99 及 SQL'03 的功能以及其它 SQL 产品功能。在本章中此后凡提及 SQL 者都可理解为 SQL'92。

SQL 的功能有如下五个方面：

（1）SQL 的数据定义功能。

SQL 的数据定义主要有如下几种功能：

- 1) 模式的定义与取消；
- 2) 基表的定义与取消；
- 3) 视图的定义与取消；
- 4) 索引、集簇的建立与删除。

（2）SQL 的数据操纵功能。

SQL 的数据操纵主要有如下几种功能：

- 1) 数据查询功能;
 - 2) 数据删除功能;
 - 3) 数据插入功能;
 - 4) 数据修改功能;
 - 5) 数据的简单计算及统计功能。
- (3) SQL 的数据控制功能。

SQL 的数据控制主要有如下几种功能:

- 1) 数据的完整性约束功能;
 - 2) 数据的安全性及存取授权功能;
 - 3) 数据的触发功能;
 - 4) 数据的并发控制功能及故障恢复功能。
- (4) SQL 的数据交换功能

SQL 的数据交换主要有如下几种功能:

- 1) 会话功能;
 - 2) 连接功能
 - 3) 游标功能
 - 4) 诊断功能
 - 5) 动态 SQL 功能
- (5) SQL 的扩展功能

SQL 的扩展功能包括四种对外交换方式:

- 1) SQL/BD——嵌入式方式;
- 2) SQL/PSM——持久存贮模块方式 (又称自含式方式);
- 3) SQL/CLI——调用层接口方式;
- 4) SQL/XML——Web 方式。

在本章中主要介绍上述的功能 (1) 和 (2), 而功能 (3)、(4)、(5) 将在第四、五、六等三章中介绍。

3.4.2 SQL 数据定义功能

SQL 数据定义功能包括基本数据类型、模式定义、基表定义、索引定义及视图定义等五部分, 本节介绍前面四个部分, 视图部分放到 3.5 节介绍。

3.4.2.1 SQL 基本数据类型

SQL'92 支持的数据类型包括表 3.10 所示的类型。

表 3.10 数据类型

序 列	符 号	数据类型	备 注
1	INT	整数	
2	SMALLINT	短整数	
3	DEC(m, n)	十进制数	m 表示小数点前位数, n 表示小数点后位数
4	FLOAT	浮点数	
5	CHAR(n)	定长字符串	n 表示字符串位数
6	VARCHAR(n)	变长字符串	n 表示最大变长数
7	NATIONAL CHAR	民族字符串	用于表示汉字

8	BIT(n)	位串	n 为位串长度
9	BIT VARYING(n)	变长位串	n 为最大变长数
10	DATE	日期	
11	TIME	时间	
12	TIMESTAMP	时间戳	
13	INTERVAL	时间间隔	

3.4.2.2 模式定义与删除

模式给出了一些表的集合，它包括基表、视图及索引等，它们称模式元素。模式一般由 SQL 语句中的创建模式（CREATE SCHEMA）及删除模式（DROP SCHEMA）表示。

1. 模式定义

模式定义由 CREATE SCHEMA 完成，其形式为：

```
CREATE SCHEMA <模式名> AUTHORIZATION <用户名>
```

该语句共有两个参数，它们是模式名及用户名。

模式一旦定义后，该模式后所定义的模式元素即归属于此模式，如学生数据库的模式可定义如下：

```
CREATE SCHEMA STUDENT AUTHORIZATION LIN
```

2. 模式删除

模式删除可由 DROP SCHEMA 完成，其形式为：

```
DROP SCHEMA <模式名> <删除方式>
```

在参数“删除方式”中一共有两种，一种是连锁式：CASCADE，另一种是受限制：RESTRICT，其中 CASCADE 表示删除与模式所关联的模式元素，而 RESTRICT 则表示只有在模式中无任何关联模式元素时才能删除。

如学生数据库模式可删除如下：

```
DROP SCHEMA STUDENT CASCADE
```

该语句执行后则删除模式及与其关联的所有模式元素。

3.4.2.3 基表的定义、删除与修改

1. 基表的定义

可以通过创建表（CREATE TABLE）语句以定义一个基表的框架，其形式为：

```
CREATE TABLE<基表名> (<列定义>[<列定义>]...) [其他参数]
```

其中列定义有如下形式：

```
<列名> <数据类型>[NOT NULL]
```

其中任选项[其他参数]是与物理存储有关的参数，它随具体系统而有所不同。

任选项[NOT NULL]表示指定列不允许出现有空值，它可由用户任选。通过基表定义可以建立一个关系框架。

可以将前面所定义的学生数据库模式用 CREATE TABLE 定义如下：

```
CREATE TABLE S ( sno CHAR(5) NOT NULL,
                 sn CHAR(20),
                 sd CHAR(2),
                 sa SMALLINT );
CREATE TABLE C ( cno CHAR(4) NOT NULL,
```

```
cn CHAR(30),
pno CHAR(4);
CREATE TABLE SC ( sno CHAR(5) NOT NULL,
cno CHAR(4) NOT NULL,
g SMALLINT);
```

2. 基表的更改

可以通过更改表（ALTER TABLE）语句以扩充或删除基表的列，从而构成一个新的基表框架，其中增加列的形式为：

```
ALTER TABLE <基表名> ADD <列名> <数据类型>
```

如可以在 S 中添加一个新的域 SEX，并可用如下形式表示：

```
ALTER TABLE S ADD sex SMALLINT
```

所要注意的是在扩充列时不允许出现 NOT NULL。

而删除列的形式为：

```
ALTER TABLE <基表名> DROP <列名> <数据类型>
```

如在 S 中删除列 sa，可用如下形式表示：

```
ALTER TABLE S DROP sa
```

3. 基表的删除

可以通过删除表（DROP TABLE）语句以删除一个基表，包括基表的结构连同该基表的数据、索引以及由该基表所导出的视图并释放相应空间。删除表的形式为：

```
DROP TABLE <基表名>
```

如果删除关系 S，可用如下的形式：

```
DROP TABLE S
```

3.4.2.4 索引的建立与删除

在 SQL 中可以对基表建立索引。索引的建立一般由 DBA 或建表人负责。

索引的建立可以通过建立索引（CREATE INDEX）语句实现，该语句可以按指定基表、指定列以及指定顺序（升序或降序）建立索引。其形式如下：

```
CREATE [UNIQUE] [CLUSTER] INDEX <索引名>
```

```
ON <基表名> ( <列名> [<顺序>] [, <列名> [<顺序>], ... ) [其他参数]
```

语句中 UNIQUE 为任选项，在建立索引中若出现 UNIQUE 则表示不允许两个元组在给定的索引中有相同的值。CLUSTER 表示所建立的索引是集簇索引。所谓集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。在最经常查询的列上可建立集簇索引以提高查询效率，一般讲一张表只能建一个集簇索引，而建立此索引代价极大，因此对经常更新的列不宜建立集簇索引。语句中顺序可按升序（ASC）或降序（DESC）给出，顺序可以默认，默认时为升序。语句中其他参数为任选项，它与物理存储有关，在此不予细述。现举三例如下：

```
CREATE UNIQUE INDEX XSNO ON S (sno)
```

此例表示在 S (sno) 上建立一个按升序的惟一性的索引 XSNO。

```
CREATE INDEX XSC ON SC (sno, cno)
```

此例表示在 SC 上建立一个按 (sno, cno) 升序排列名为 XSC 的索引。

```
CREATE CLUSTER INDEX STUSN ON S(sn)
```

此例表示在 S (sn) 上建立集簇索引 STUSN, 且 S 表上记录将按 sn 值的升序存放。

在 SQL 中可以用删除索引 (DROP INDEX) 语句以删除一个已建立的索引:

```
DROP INDEX <索引名>
```

如下面的例子表示将已建立的名为 XSNO 的索引删除:

```
DROP INDEX XSNO
```

3.4.3 SQL 数据操纵功能

SQL 的数据操纵具有数据查询、删除、插入及修改等功能。此外还具有如下的一些附属功能:

(1) 赋值功能。在数据操纵过程中所产生的一些中间结果以及需作永久保留的结果, 必须以新的关系形式存储于数据库内, 因此对这些新关系须予以命名并赋值, 经赋值后的关系今后在数据库中即可供用户使用。

(2) 计算功能。SQL 作为自含式语言需要有一些计算功能:

1) 简单的四则运算。它包括在查询过程中可以出现有加、减、乘、除等简单计算。

2) 统计功能。由于数据库在统计中有极大的应用, 因此 SQL 有常用的统计功能, 它们为求和、求平均值、求总数、求最大值、求最小值等。

3) 分类功能。

(3) 输入/出功能。SQL 提供标准的数据录入与输出功能。

SQL 语言的数据操纵功能是一种基于关系代数的操纵形式。我们知道, 对查询的基本要求关系代数中可以用下式表示:

$$\pi_{A_1, A_2, \dots, A_m} \sigma_F (R_1 \times R_2 \times \dots \times R_n)$$

在这个公式中有三种基本参数, 它们是:

1) 查询的目标属性: A_1, A_2, \dots, A_m

2) 查询所涉及的关系: R_1, R_2, \dots, R_n

3) 查询的逻辑条件: F

这三种参数可以用 SQL 中的基本语句——SELECT 语句的三个子句分别表示。SELECT 语句由 SELECT, FROM 及 WHERE 等三个子句组成, 其中 SELECT 子句给出查询的目标属性, FROM 子句给出查询所涉及的关系, 而 WHERE 子句则给出查询的逻辑条件, 它们可以用下面形式表示:

```
SELECT <列名> [, <列名>, ...]  
FROM <基表名> [, <基表名>, ...]  
WHERE <逻辑条件>
```

这种 SELECT 语句在数据查询中表达力很强, 不仅如此, 为了表示简洁与方便, SELECT 语句中的 WHERE 子句还具有更多的表示能力:

(1) SELECT 语句在 WHERE 子句中具有嵌套能力。

(2) WHERE 子句中的逻辑条件具有集合表达能力的一阶谓词公式形式。

此外, SQL 还具有数据删除、插入、修改的能力, 其语句形式与 SELECT 语句类似。

3.4.3.1 SQL 的基本查询功能

SQL 的查询功能基本上是用 SELECT 语句实现的,下面用若干例子说明 SELECT 语句的使用,这些例子仍以过去所述的学生数据库 STUDENT 为背景:

S (sno, sn, sa, sd)

C (cno, cn, pno)

SC (sno, cno, g)

1. 单表简单查询

单表的简单查询包括:

- 单表全列查询
- 单表的列查询
- 单表的行查询
- 单表的行与列查询

它们可用下面四个例子表示。

例 3.25 查询 S 的所有情况:

```
SELECT *  
FROM S
```

其中*表示所有列。

例 3.26 查询全体学生名单。

```
SELECT sn  
FROM S
```

此例为选择表中列的查询。

例 3.27 查询学号为 990137 的学生学号与姓名。

```
SELECT sno, sn  
FROM S  
WHERE sno='990137'
```

此例为选择表中行的查询。

选择表中行的查询中须使用比较符,常用的比较符包括如下一些:

=, <, >, >=, <=, <>。

它们构成 A θ B 之形式,其中 A, B 为列名、列表式或列的值。A θ B 称比较谓词,它是一个判断结果仅为真(true)或假(false)的谓词。

例 3.28 查询所有年龄大于 20 岁的学生姓名与学号:

```
SELECT sn, sno  
FROM S  
WHERE sa > 20
```

此例为选择表中行与列的查询。

2. 常用谓词

除比较谓词外,SELECT 语句中还有若干谓词,谓词可以增强语句表达能力,所谓谓词即是数理逻辑中谓词,它的值仅是 T/F,在这里我们介绍几个常见的谓词,它们是:

- DISTINCT

-
- BETWEEN...AND...
 - LIKE
 - NULL

一般的谓词常用于 WHERE 子句中，但是 DISTINCT 则仅用于 SELECT 子句中。

例 3.29 查询所有选修了课程的学生学号：

```
SELECT DISTINCT sno
FROM SC
```

SELECT 后的 DISTINCT 表示在结果中去掉重复 sno。

例 3.30 查询年龄在 18 至 21 岁（包括 18 与 21 岁）的学生姓名与年龄：

```
SELECT sn, sa
FROM S
WHERE sa BETWEEN 18 AND 21
```

例 3.31 查询年龄不在 18 至 21 岁的学生姓名与年龄：

```
SELECT sn, sa
FROM S
WHERE sa NOT BETWEEN 18 AND 21
```

上两例给出了 WHERE 子句中 BETWEEN 的使用方法。

例 3.32 查询姓名以 A 打头的学生姓名与所在系：

```
SELECT sn, sd
FROM S
WHERE sn LIKE 'A%'
```

此例给出了 WHERE 中 LIKE 的使用方法，LIKE 的一般形式是：

```
<列名> [NOT] LIKE <字符串常量>
```

其中列名类型必须为字符串，字符串常量的设置方式是：

字符%表示可以与任意长的字符相配，字符_（下横线）表示可以与任意单个字符相配；其他字符代表本身。

例 3.33 查询姓名以 A 打头，且第三个字符必为 P 的学生姓名与系别：

```
SELECT sn, sd
FROM S
WHERE sn LIKE 'A_P%'
```

例 3.34 查询无课程分数的学号与课程号：

```
SELECT sno, cno
FROM SC
WHERE g IS NULL
```

此例给出了 NULL 的使用方法，NULL 的一般形式是：

```
<列名> IS [NOT] NULL
```

3. 布尔表达式

在 WHERE 子句中经常需要使用逻辑表达式，它一般由谓词通过 NOT、AND 与 OR

三个联结词构成，称为布尔表达式。下面举几个例子。

例 3.35 查询计算机系年龄小于等于 20 岁的学生姓名：

```
SELECT sn
FROM S
WHERE sd='CS' AND sa <= 20
```

例 3.36 查询非计算机系或年龄不为 18 岁的学生姓名：

```
SELECT sn
FROM S
WHERE NOT sd='CS' OR NOT sa=18
```

在三个联结词中结合强度依次为 NOT、AND、及 OR，在表达式中若同时出现有若干个联结词且有时不按结合强度要求则需加括号。

4. 简单连接

在多表查询中涉及到表间连接，其中简单的连接方式是表间等值连接，它可用 where 子句设置两表不同属性间的相等谓词，下面用两个例子以说明之。

例 3.37 查询修读课程号为 c₁ 的所有学生的姓名。

这是一个涉及到两张表的查询，它可以写为：

```
SELECT S.Sn
FROM S, SC
WHERE SC.sno=S.sno AND SC.cno='c1'
```

S.sn, S.sno 及 SC.sno, SC.cno 分别表示表 S 中的属性 sn, sno 以及表 SC 中的属性 sno, cno。一般而言，在涉及多张表查询时须在属性前标明该属性所属的表，但是凡查询中能区分的属性则其前面的表名可省略。

例 3.38 查询修读课程名为 DATABASE 的所有学生姓名。

这是一个涉及到三张表的查询，它可以写为：

```
SELECT S.sn
FROM S, SC, C
WHERE S.sno=SC.sno AND SC.cno=C.cno AND C.cn='DATABASE'
```

5. 自连接

有时在查询中需要对相同的表进行联结，为区别两张相同的表，须对一表用两个别名，现以一例说明。

例 3.39 查询至少修读 S₅ 所修读的一门课的学生学号。

```
SELECT FIRST_SC.sno
FROM SC FIRST_SC, SC SECOND_SC
WHERE FIRST_SC.cno=SECOND_SC.cno AND
      SECOND_SC.sno='S5'
```

它可以用下面的图 3.1 表示之。

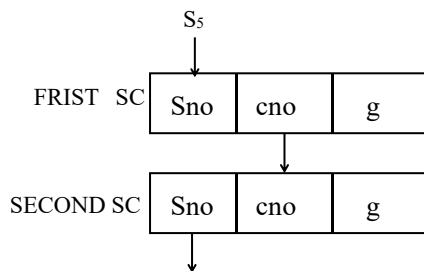


图 3.1 连接图

6. 结果排序

有时, 希望查询结果能按某种顺序显示, 此时须在语句后加一个排序子句 ORDER BY, 该子句具有下面的形式:

```
ORDER BY <列名> [ASC | DESC]
```

其中<列名>给出了所需排序的列的列名, 而 ASC 或 DESC 则分别给出了排序的升序与降序, 有时为方便起见, ASC 可以省略。

例 3.40 查询计算机系所有学生名单并按学号顺序升序显示。

```
SELECT sno, sn
FROM S
WHERE sd='CS'
ORDER BY sno ASC
```

例 3.41 查询全体学生情况, 结果按学生年龄降序排列。

```
SELECT *
FROM S
ORDER BY sa DESC
```

3.4.3.2 分层结构查询与集合谓词使用

SQL 是分层结构的, 在 SELECT 语句的 WHERE 子句中嵌套使用 SELECT 语句。而一个 SELECT 语句的结果是一张新表, 它是元组的集合, 因此嵌套的 SELECT 语句是一个集合, 因而在 WHERE 子句中除可以用比较谓词及常用谓词外还可以用集合间的关系表示复杂的谓词。这些关系可以包括下列几个方面 (其中, S 和 S' 表示集合, X 表示集合中的元素, θ 是比较运算符, \exists 和 \forall 表示量词):

(1) 元素与集合间的属于关系:

$$X \in S, X \notin S$$

(2) 集合间的包含、相等关系:

$$S \supset S', S \subset S', S = S'$$

(3) 集合的存在关系:

$$\exists S$$

(4) 元素与集合元素间的量化比较关系:

$$X \theta \exists S, X \theta \forall S$$

在 SQL 语言中, 提供了如下的一些谓词来表示上述的关系 (除了集合与集合之间的包含关系):

- 元素与集合之间的属于关系: IN
- 集合的存在量词: EXISTS
- 用于量化比较的量词: 存在量词 SOME 和 ANY, 全程量词 ALL

1. 谓词 IN 的使用

元素与集合间的属于关系可以用 IN 表示。

例 3.43 查询在计算机系、数学系及物理系的学生姓名。

```
SELECT sn
FROM S
WHERE sd IN {'MA', 'CS', 'PY'}
```

在 WHERE 中使用“IN”还可能会出现 SELECT 语句的嵌套使用。由于 IN 中必须出现有集合, 而 SELECT 语句的结果为元组集合, 因此在 IN 的使用中允许出现有 SELECT 语句, 这样就出现了 SELECT 语句的嵌套使用。

例 3.44 查询修读课程号为 C₁ 的所有学生姓名。

```
SELECT S.sn
FROM S
WHERE S.sno IN
      (SELECT SC.sno
       FROM SC
       WHERE SC.cno='C1')
```

在此例子中 WHERE 子句具有 $X \in S$ 之形式, 其中 S.sno 为元素, IN 为“属于”(∈), 而嵌套 SELECT 语句:

```
SELECT SC.sno
FROM SC
WHERE SC.cno='C1'
```

为集合, 它被称为是一个子查询。

这个嵌套可以用下面的图 3.2 表示。

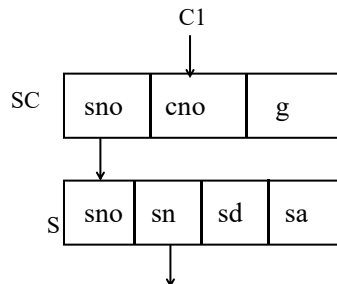


图 3.2 嵌套图

例 3.45 查询所有成绩都及格的学生姓名。

```
SELECT sn
FROM S
WHERE sno NOT IN
    ( SELECT sno
      FROM SC
      WHERE g < 60 )
```

通过 WHERE 子句中的 IN 可以实现嵌套，这种嵌套可以有多重，下面的例子即是三重嵌套，这是一个如图 3.3 所示的嵌套形式。

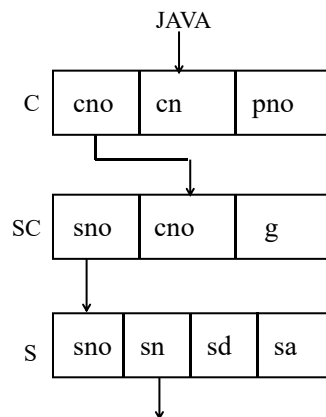


图 3.3 三重嵌套图

例 3.46 查询修读课程名为 JAVA 的所有学生姓名。

```
SELECT sn
FROM S
WHERE sno IN
    ( SELECT sno
      FROM SC
      WHERE cno IN
          ( SELECT cno
            FROM C
            WHERE cn='JAVA' ) )
```

2. 限定比较谓词的使用

在元素与集合的关系中尚有更为复杂的情况，它们是元素与集合中元素的比较关系，它可用带有比较符的 ANY 或 ALL 表示。其中谓词 ANY 表示子查询结果集中某个值，而谓词 ALL 则表示子查询结果集中的所有值。这样，‘>ANY’表示大于子查询结果集中的某个值，‘>ALL’表示大于子查询结果集中的所有值。其它如 ‘>=ANY’、‘>=ALL’、‘=ANY’、‘=ALL’、‘<ANY’、‘<ALL’、‘<=ANY’、‘<=ALL’、‘<>ANY’、‘<>ALL’等。注意，现在经常用 ‘SOME’ 代替 ‘ANY’ 这是允许的，它们具相同效果。下面用两个例子说明使用量化比较谓词的查询。

例 3.47 查询有学生成绩大于 C₁ 课程号中所有学生成绩的学生学号。

```
SELECT sno
FROM SC
WHERE g > ALL
      (SELECT g
       FROM SC
       WHERE cno='C1')
```

例 3.48 查询有学生成绩大于等于 C₁ 课程号中的任何一位学生成绩的学生学号。

```
SELECT sno
FROM SC
WHERE g >= ANY
      (SELECT g
       FROM SC
       WHERE cno='C1')
```

3. 量词 EXISTS 的使用

在 WHERE 子句中所出现的集合关系允许出现有量词，一般量词有两种：存在量词与全称量词。在 SQL 中存在量词可用谓词[NOT] EXISTS 表示。由于有：

$$\forall xQ(x) = \neg\exists x(\neg Q(x))$$

其中， \neg 表示逻辑非。故 SQL 中无需全称量词表示式，而可用 NOT EXISTS 通过转换表示全称量词。

用存在量词可以以另一种形式表示集合间的关系。

例 3.50 查询修读课程号为 C₁ 的所有学生姓名（修读过 C₁ 这门课程）

```
SELECT sn
FROM S
WHERE EXISTS (SELECT *
              FROM SC
              WHERE S.sno=SC.sno AND SC.cno='C1')
```

例 3.51 查询修读课程号不为 C₁ 的所有学生姓名（没有修读过 C₁ 这门课程）

```
SELECT sn
FROM S
WHERE NOT EXISTS (SELECT *
                  FROM SC
                  WHERE S.sno=SC.sno AND SC.cno='C1')
```

3.4.3.3 SELECT 语句间的运算

在 SQL 中 SELECT 语句是集合的一种表示，而 SELECT 语句间是可以作运算的，这种运算是集合运算，它们有 UNION、INTERSECT 及 EXCEPT 三个分别表示集合的并、交、差运算。所要注意的是这些集合运算其 SELECT 语句中的 SELECT 子句须具相同结构形式。

例 3.52 查询计算机科学系的学生以及年龄小于 20 岁的学生。

```
(SELECT *
FROM S
WHERE sd='CS')
UNION
(SELECT *
FROM S
WHERE sa<20)
```

3.4.3.4 SQL 计算、统计、分类的功能

可在 SQL 的查询语句中插入计算、统计、分类的功能以增强数据查询能力。

1. 统计功能

SQL 的查询中可以插入一些常用统计功能，它们能对集合中的元素作下列计算。

- (1) COUNT: 集合元素个数统计。
- (2) SUM: 集合元素的和 (仅当元素为数值型)。
- (3) AVG: 集合元素平均值 (仅当元素为数值型)。
- (4) MAX: 集合中最大元素 (仅当元素为数值型或字符型)。
- (5) MIN: 集合中最小元素 (仅当元素为数值型或字符型)。

以上五个函数叫总计函数 (aggregate function)，这种函数是以集合为其变域以数值为其值域，可用图 3.4 表示。

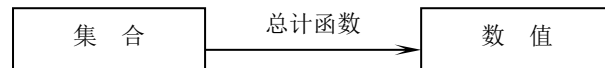


图 3.4 总计函数的功能

例 3.53 给出全体学生数。

```
SELECT COUNT (*)
FROM S
```

例 3.54 给出学生 S₁ 修读的课程数。

```
SELECT COUNT (*)
FROM SC
WHERE sno='S1'
```

例 3.55 给出学生 S₇ 所修读课程的平均成绩。

```
SELECT AVG (g)
FROM SC
WHERE sno='S7'
```

2. 计算功能

SQL 查询中可以插入简单的算术表达式如四则运算功能，下面举几个例子说明之。

例 3.56 给出修读课程为 C₇ 的所有学生的学分级 (即学分数*3)。

```
SELECT sno, cno, g*3
FROM S
WHERE cno='c7'
```

例 3.57 给出计算机系下一年度学生的年龄。

```
SELECT sn, sa+1
FROM S
WHERE sd='CS'
```

3. 分类功能

SQL 语句中允许增加两个子句。

```
GROUP BY
HAVING
```

此两子句可以对 SELECT 语句所得到的集合元组分组（用 GROUP BY 子句），并还可设置逻辑条件（用 HAVING 子句），下面举几个例子说明之。

例 3.58 给出每个学生的平均成绩。

```
SELECT sno, AVG(g)
FROM SC
GROUP BY sno
```

例 3.59 给出每个学生修读课程的门数。

```
SELECT sno, COUNT(cno)
FROM SC
GROUP BY sno
```

例 3.60 给出所有超过五个学生所修读课程的学生数。

```
SELECT cno, COUNT(sno)
FROM SC
GROUP BY cno
HAVING COUNT(*) > 5
```

例 3.61 按总平均值降序给出所有课程都及格但不包括 C₈ 的所有学生总平均成绩。

```
SELECT sno, AVG(g)
FROM SC
WHERE cno <> 'C8'
GROUP BY sno
HAVING MIN(g) >= 60
ORDER BY AVG(g) DESC
```

3.4.3.5 SELECT 语句使用的一般规则

在这节中我们对 SELECT 语句的使用作一个小结。

1. SELECT 语句中的子句及使用规则

SELECT 语句有六个子句，其中 SELECT 及 FROM 子句是必须的，它们按下面的次序排列：

```
SELECT  目标子句
FROM    范围子句
[ WHERE 条件子句 ]
[ GROUP BY ...
[ HAVING ...      ] ]
[ ORDER BY ...    ]
```

这七个子句中主要使用的是 SELECT、FROM 及 WHERE 三个子句。

(1) 目标子句 SELECT

目标子句 SELECT 给出查询的结果目标，主要表示形式如下：

```
[DISTINCT] [<表名>.<列表表达式>, [<表名>.[<列表表达式>], ...
```

其中列表表达式包括：列名、算术表达式及总计函数等。目标子句中还允许加入谓词 DISTINCT。

(2) 范围子句 FROM

范围子句 FROM 给出查询所涉及的表，包括表名以及表的别名。

(3) 条件子句 WHERE

条件子句 WHERE 给出查询所须满足的条件，即每个查询目标必须满足 WHERE 子句中的条件。也可以说，包括每个查询目标值是那些使 WHERE 子句中的条件为 T 的值。由此可以看出 WHERE 子句中的条件是一个逻辑条件，它具有真 (T)、假 (F) 之别，它是一种逻辑表达式，称布尔表达式。

布尔表达式可由下面三个层次组成：

1) 操作数：

- ① 标量个体：个体常量与个体变量。
- ② 集合量：集合常量与集合变量。

2) 谓词：

谓词是谓词公式中基本单元，它有 T/F 之分，它由操作数组成。在 WHERE 子句中可以使用的谓词可分为三种：

① 标量谓词

- 比较谓词：它是一种具 $A\theta B$ 型的谓词，其中 A, B 为个体， θ 为比较符。
- 其它谓词：DISTINCT、BETWEEN、LIKE 及 NULL 等。

② 集合量谓词

IN 及 EXISTS 谓词，它们可以表示集合与个体间关系： $a \in S$ 或 $\exists S$ 。

③ 标量—集合量谓词： θ ANY、 θ SOME、 θ ALL

3) 布尔表达式

由谓词通过联结符 AND, OR 及 NOT 所组成的公式称布尔表达式。

2. SELECT 语句间的运算

SELECT 语句间存在集合运算，可以使用 UNION、INTERSECT 及 EXCEPT 将 SELECT 语句连接起来构成一个具有相同结构的 SQL 查询。

3.4.4 SQL 的更新功能

SQL 的更新功能包括删除、插入及修改等三种操作。

1. SQL 的删除功能

SQL 的删除语句一般形式为：

```
DELETE
FROM   <基表名>
WHERE  <逻辑条件>
```

其中 DELETE 指明该语句为删除语句，FROM 与 WHERE 的含义与 SELECT 语句中相同。

例 3.62 删除学生 WANG 的记录。

```
DELETE
FROM   S
WHERE  sn='WANG'
```

例 3.63 删除计算机系全体学生的选课记录。

```
DELETE
FROM   SC
WHERE  'CS' IN (SELECT sd
                FROM   S
                WHERE  S.sno =SC.sno)
```

2. SQL 的插入功能

SQL 插入语句的一般形式为：

```
INSERT INTO <基表名>[( <列名> [, <列名>, ... ])]
VALUES  (<常量> [, <常量>, ...])
```

该语句的含义是执行一个插入操作，将 VALUES 所给出的值插入 INTO 所指定的表中。

插入语句还可以将某个查询结果插入至指定表中，其形式为：

```
INSERT INTO <基表名>[( <列名> [, <列名>, ... ])]
<子查询语句>
```

例 3.64 插入一个选课记录 (S₁₀, C₁₅, 5)。

```
INSERT
INTO  SC
VALUES ('S10', 'C15', 5)
```

例 3.65 将 SC 中成绩及格的记录插入到 SCI 中。

```
INSERT
INTO  SCI
SELECT *
FROM  SC
WHERE g>60
```

3. SQL 的修改功能

SQL 修改语句的一般形式为:

```
UPDATE <表名>
SET <列名>=表达式 [, <列名>=表达式 ,...]
WHERE <逻辑条件>
```

该语句的含义是修改 (UPDATE) 指定基表中满足 (WHERE) 逻辑条件的元组, 并把这些元组按 SET 子句中的表达式修改相应列上的值。

例 3.66 将学号为 S₁₆ 的学生系别改为 CS。

```
UPDATE S
SET sd='CS'
WHERE sno='S16'
```

例 3.67 将数学系学生的年龄均加 1 岁。

```
UPDATE S
SET sa=sa+1
WHERE sd='MA'
```

例 3.68 将计算机系学生的成绩全置零。

```
UPDATE SC
SET g=0
WHERE 'CS' IN (SELECT sd
                FROM S
                WHERE S.sno=SC.sno)
```

3.4.5 视图

SQL 提供视图 (View) 功能, 关系数据库管理系统中的视图与传统数据库中的视图略有不同。传统数据库的视图纯属概念数据库的一部分, 而关系数据库管理系统中的视图则由概念数据库改造而成, 它是由若干基表经 SELECT 语句而构筑成的表, 称为导出表 (drived table)。这种表本身并不实际存在于数据库内, 而仅保留其构造 (即 SELECT 语句)。只有在实际操作时, 才将它与操作语句结合转化成对基表的操作, 因此这种表也称为虚表 (virtual table)。

1. 视图定义

SQL 的视图可用创建视图语句定义, 其一般形式如下:

```
CREATE VIEW <视图名> [( <列名> [, <列名>, ... ] )]
AS <SELECT 语句>
[WITH CHECK OPTION]
```

其中 WITH CHECK OPTION 子句是可选的, 它表示对视图作增、删、改操作时要保证增、删、改的行要满足视图定义中的逻辑条件。

例 3. 69 定义一个计算机系学生的视图。

```
CREATE VIEW CS_S(SNO, SN, SD, SA)
AS (SELECT *
     FROM S
     WHERE Sd='cs')
```

例 3. 70 定义学生姓名和他修读的课程名及其成绩的视图。

```
CREATE VIEW S_C_G (SN, CN, G)
AS (SELECT S.sn, C.cn, SC.g
     FROM S, SC, C
     WHERE S.sno =SC.sno AND SC.cno=C.cno)
```

例 3. 71 定义学生姓名及其平均成绩的视图。

```
CREATE VIEW S_G (SN, AVG)
AS ( SELECT sn, AVG (g)
     FROM S, SC
     WHERE S.sno=SC.sno
     GROUP BY sn)
```

SQL 的视图可以用取消视图语句将其删除，其形式如下：

```
DROP VIEW <视图名>
```

例 3. 72 删除已建立的视图 S_G。

```
DROP VIEW S_G
```

视图的取消表示不仅取消该视图而且还包括由该视图所导出的其他视图。

2. 视图操作

对视图可以作查询操作，但对更新操作则受一定限制。一般在创建视图后可像基表一样对视图作查询。

例 3. 73 用已定义视图 CS—S 作查询，查询计算机系中年龄大于 20 岁的学生。

```
SELECT *
FROM CS_S
WHERE sa > 20
```

对此查询在实际操作时需将该查询转换成为对基表的查询，即用视图 CS—S 的定义将此查询转换成为：

```
SELECT *
FROM S
WHERE sd='cs' AND sa > 20
```

对视图作删除、插入及修改等更新操作是有一定困难的。这主要是因为视图仅是一种虚构的表，而并非实际存在于数据库中，而作更新操作时必然会涉及到数据库中数据的变动，因此就出现了困难，故对视图作更新操作一般不能进行，只在下面特殊情况才可以进行：

-
- (1) 视图的每一行必须对应基表的惟一一行。
 - (2) 视图的每一列必须对应基表的惟一列。

下面举一个能做更新操作的视图。

例 3.74 定义年龄大于 18 岁的学生视图。

```
CREATE VIEW SS (SNO, SN, SD, SA)
AS (SELECT *
    FROM S
    WHERE sa > 18)
```

这个视图满足上面两个条件，对此视图可作更新操作。

例 3.75 插入元组：27188，沈佩华，CS，20 至视图 SS。

```
INSERT
INTO SS
VALUES ('27188', '沈佩华', 'CS', 20)
```

有了视图后，数据独立性大为提高，不管基表是扩充还是分解均不影响对概念数据库的认识。只需重新定义视图内容，而不改变面向用户的视图形式，因而保持了关系数据库逻辑上的独立性。同时视图也简化了用户观点，用户不必了解整个模式，仅需将注意力集中于它所关注的领域，大大方便了使用。此外，视图还提供了自动的安全保护功能。

习题三

- 3.1 试述关系数据库系统的优点。
- 3.2 试述关系型的 12 条标准，并以你所熟悉的一个 RDBMS 为例用此标准衡量之。
- 3.3 关系代数是一种代数系统吗？请说明之。
- 3.4 试比较关系代数与关系数据库能力是否相同？请说明之。
- 3.5 请你对关系数据库系统的数学理论作出评价。
- 3.6 今有如下商品供应关系数据库。
供应商： S (SNO, SNAME, STATUS, CITY)
零件： P (PNO, PNAME, COLOR, WEIGHT)
工程： J (JNO, JNAME, CITY)
供应关系： SPJ (SNO, PNO, JNO, QTY) (注：QTY 表示供应数量)
试画出其 E-R 图并用 SQL 写出下面之查询公式：
 - (1) 求供应工程 J1 零件单位号码。
 - (2) 求没有使用天津单位生产的红色零件的工程号。
 - (3) 求供应工程 J1 零件 P1 的供应商号码。
 - (4) 求供应工程 J1 零件为红色的单位号码。
 - (5) 求至少用了单位 S1 所供应的全部零件的工程号。
 - (6) 求供应商与工程在同一城市能供应的零件数量。
- 3.7 试述 SQL 的特点与功能。
- 3.8 什么是基表，什么是视图？两者有何关系与区别？
- 3.9 SQL 查询中的谓词与数理逻辑中的谓词是同一个概念吗？请回答并说明理由。

3.10 在 SQL 中有哪些方法可作表间连接, 请说明之, 并各举一例。

3.11 有如图 3.2 所示结构的医院组织。请用关系代数及 SQL 作如下查询:

- (1) 找出外科病房所有医生姓名;
- (2) 找出管辖 13 号病房的医生姓名;
- (3) 找出管理病员李维德的医生姓名;
- (4) 给出内科病房患食道癌病人总数。

病房:	编号	名称	所在位置	主任姓名
医生:	编号	姓名	职称	管辖病房号
病人:	编号	姓名	患何种病	病房号

图 3.2 某医院组织结构

3.12 在本章所定义的学生数据库中用 SQL 作如下操作:

- (1) 查询系别为计算机的学生学号与姓名;
- (2) 查询计算机系所开课程之课程号与课程名;
- (3) 查询至少修读一门 OS 的学生姓名;
- (4) 查询每个学生已选课程门数和总平均成绩;
- (5) 查询所有课程的成绩都在 80 分以上的学生姓名、学号并按学号顺序排列;
- (6) 删除在 S, SC 中所有 Sno 以 91 开头的元组。

3.13 设有图书管理数据库, 其数据库模式如下:

图书 (书号、书名、作者姓名、出版社名称、单价)

作者 (姓名、性别、籍贯)

出版社 (出版社名称、所在城市名、电话号码)

请同时使用关系代数与 SQL 语言表示下述查询:

- (1) 由‘科学出版社’出版发行的所有图书书号。
- (2) 由籍贯是‘江苏省’的作者所编号的图书书名。
- (3) 图书‘软件工程基础’的作者的籍贯及其出版社所在城市名称。

3.14 设有车辆管理数据库的数据模式如下:

车辆 (车号、车牌名、车颜色、生产厂名)

工厂 (厂名、厂长姓名、所在城市名)

城市 (城市名、人口、市长姓名)

请用 SQL 语言写出如下查询:

- (1) 查询车牌名为红旗牌轿车的所有车号。
- (2) 查询红旗牌轿车的生产厂家及厂长姓名。
- (3) 查询跃进牌货车的生产厂家及所在城市的市长姓名。
- (4) 查询第一汽车制造厂所生产车辆的颜色。
- (5) 查询武汉生产哪些牌子的车。

3.15 设有一课程设置数据库, 其数据模式如下:

课程 C (课程号 Cno、课程名 Cname、学分数 Score、系别 Dept)

学生 S (学号 Cno、姓名 name、年龄 Age、系别 Dept)

课程设置 SEC (编号 Secid、课程编号 Cno、年 year、学期 Sem)

成 绩 GRADE (编号 Secid、学号 Sno、成绩 G)

其中成绩 G 采用五级记分法, 即分为 1, 2, 3, 4, 5 五级。

(1) 请用关系代数表示下列查询

1) 查询‘计算机’系的所有课程的课程名和学分数。

2) 查询学号为‘993701’的学生在 2002 年所修课程的课程名和成绩。

(2) 请用 SQL 作下列查询

1) 查询‘计算机’系的所有课程。

2) 查询‘计算机’在 2003 年开设课程的总数。

2) 查询‘计算机’每个学生的学号及总学分并按总学分从高到低顺序输出。

3.16 设有一职工管理数据库, 其关系模式如下:

职工 (职工编号、姓名、住址、工资、部门名称)

部门 (部门名称、地址、电话、部门负责人、职工编号)

请用 SQL 写出如下查询:

(1) 查询所有部门负责人的姓名和住址。

(2) 查询每个部门名称及职工人数。

3.17 设有一产品销售数据库, 其数据模式如下:

产品 P (pno、pn、price)

客户 C (cno、cn、ctel、ccity)

产品销售 S (cno、pno、year、month、date、num)

其中 pno、pn、price 分别表示产品编号、产品名及单价; cno、cn、ctel、ccity 分别表示客户编号、客户名、电话及所在城市; 而 year、month、date、num 则分别表示销售年、月、日及销售数量。

(1) 在上面的基表上定义一个产品销售视图 S-V, 它包括产品编号 Pno, 名称 Pname, 购买产品的客户所在城市 City 以及该产品在该城市的销售总数 P-C-total 和销售总金额 P-C-money, 请写出该视图定义语句。

(2) 请用 SQL 表示下列查询:

1) 购买‘熊猫电视机’的客户所在的城市名称。

2) ‘春兰空调’在‘南京市’的月销售情况: 月份及当月销售总数量。

3) 查询每种产品编号、名称及累计销售数量最高的城市名称 (提示: 可用本题 (1) 中建立的视图)。

3.18 设有一关系模式如下:

顾客 Customers (cid, cname, city, discent)

供应商 Agent (aid, aname, city, persent)

商品 Product (pid, pname, city, quantity, price)

订单 Orders (oid, month, cid, aid, pid, qty, dollars)

(1) 请用关系代数表示下面的操作:

1) 查询既购买过‘P01’号商品又购买过‘P07’号商品的顾客编号 (cid)。

-
- 2) 查询销售过 ‘C004’ 号顾客购买过的所有商品的供应商编号 (cid)。
- (2) 请用 SQL 作如下查询:
- 1) 查询购买过 ‘P02’ 号商品的顾客所在的城市 (city) 以及销售过 ‘P02’ 号商品的供应商所在城市 (city)。
- 2) 查询仅通过 ‘a03’ 号供应商来购买商品的顾客编号 (cid)。
- 3.19** 在学生数据库中建立计算机系的视图 (包括 S, SC, C)。
- 3.20** 利用建立的计算机系视图查询修读 Database 的学生姓名。
- 3.21** 在学生数据库中请修改 S 的模式为 S' (Sno, Sname, Ssex, Sdept)。
- 3.22** 在上题中用 SQL 作如下删、改操作:
- (1) 删除顾客号为 ‘C01’ 的元组。
- (2) 将供应商号为 ‘a07’ 的 city 改为武汉。
- 3.23** 将学生数据库中 S 的年龄全部增加 1 岁并按学号重新排序后, 赋值给新表 S'(S#, sname, sdept, sage) (注意: S' 须重新定义)。

第三章复习指导

本章介绍关系数据库系统。考虑到关系数据库系统在整个数据库领域中的作用与重要性，因此本章是全书的重点，本章对关系数据库系统作全面介绍，它包括基本原理、基本理论以及标准语言。

1. 基本原理

关系数据库系统基本原理包括：

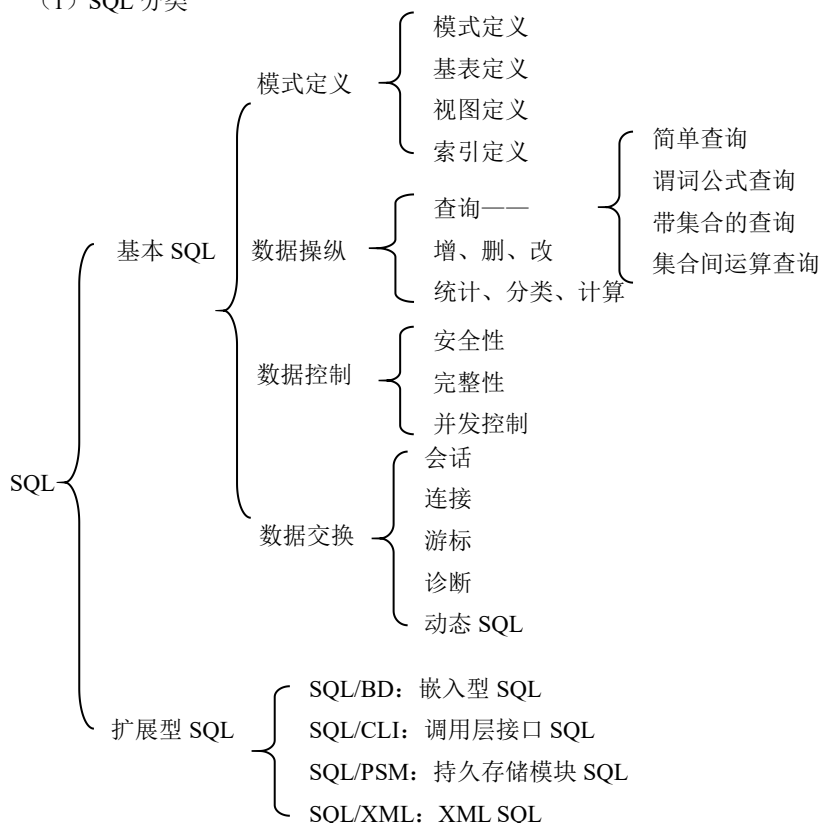
- (1) 十二条衡量准则
- (2) 关系模型

2. 基本理论

- (1) 关系代数——用代数方法所建立的关系模型
- (2) 关系代数与关系模型间的一致性

3. 标准语言——SQL

(1) SQL 分类



(2) SQLSELECT 语句的使用

1) SELECT 语句主要成份

SELECT 目标子句
FROM 范围子句
WHERE 条件子句

2) WHERE 子句的使用

WHERE 子句中的条件是一个逻辑值，它具有 T（真）与 F（假）之别，它的结构称布尔表达式：

- 操作数：标量个体及集合量。
 - 谓词：标量谓词：比较谓词，DISTINCT，BETWEEN，LIKE，NULL。
集合谓词：IN，CONTAINS，EXIST。
标量—集合量谓词：ANY，ALL。
 - 布尔表达式：由谓词及其联结符 AND，OR，NOT 所组成的公式。
- 3) SELECT 语句间的集合运算：UNION，INTERSECT，EXCEPT。

4. 本章重点内容

- 十二条衡量准则。
- 基本 SQL 中的模式定义、数据操纵。
- SQL 语句的使用。

第四章 数据库的安全性与完整性保护

数据库是计算机系统中大量数据集中存放的场所,它保存着长期积累的信息资源。它们都是经过长期不懈艰苦努力所获得的,因此是一种宝贵的信息财富。如何保护这些财富使之不受来自外部的破坏与非法盗用是数据库管理系统的重要任务。在计算机网络发达的今天,数据库一方面承担着网上开放向用户提供数据支撑服务,以达到数据资源共享的目的;而另一方面又要承担着开放所带来的负面影响,即防止非法与恶意使用数据库而引起的灾难性后果。因此保护数据库中数据不受外部的破坏与非法盗用是当今网络时代中的重要责任。当然,有关数据库中的数据保护是多方面的,它包括计算机系统外部的保护:

- (1) 环境的保护,如加强警戒、防火、防盗等。
- (2) 社会的保护,如建立各种法规、制度,进行安全教育等。
- (3) 设备保护,如及时进行设备检查、维修,部件更新等。

它也包括计算机系统内部的保护:

- (1) 计算机系统硬件、软件的保护
- (2) 网络中数据传输时数据保护。
- (3) 计算机系统中的数据保护。
- (4) 操作系统中的数据保护。
- (5) 数据库系统中的数据保护。
- (6) 应用系统中的数据保护等。

而在这些保护中,数据库系统中的数据保护是至关重要的。在本章中将主要讨论数据库中的数据保护,这就是数据库的安全性(security)与完整性(integrity)保护。

在本章中还将介绍用 SQL 以实现数据安全与完整的数据控制功能。由于 SQL'92 以及包括 SQL 所有版本在内对数据安全及完整的支持力度不足,因此本章不仅介绍 SQL'92,还根据需要介绍 SQL'99 以及 SQL 标准外的控制操作,如 ORACLE 的相关操作。

4.1 数据库的安全性保护

4.1.1 数据库的安全与安全数据库

所谓数据库的安全(database security)即是保证对数据库的正确访问与防止对数据库的非法访问。数据库中的数据是共享资源,必须在数据库管理系统中建立一套完整的使用规则。使用者凡按照规则访问数据库必能获得访问权限,而不按规则访问数据库者必无法获得访问权限,而最终无法访问数据库。这就是数据管理系统中数据库安全的问题。

访问数据库的规则有多种,不同的规则适用于不同的应用。有的规则较为宽松,有的规则较为严厉。在过去单机方式下的数据库由于共享面窄,因此规则较为宽松,而在网络方式下特别是在因特网方式下,由于数据共享面广,规则较为严厉。因此,根据应用的不同需求,数据库的安全分不同级别。而那些能适应网络环境下安全要求级别数据库称为安全数据库(secure database)或称为可信数据库(trusted database)。

目前我国所使用的数据库管理系统基本上都达不到安全数据库的要求,在网上使用均存在安全隐患,因此大力推广使用安全数据库是当务之急。

4.1.2 数据库安全的基本概念与内容

数据库安全的基本概念与内容如下：

(1) 可信计算基 (trusted computing base)

可信计算基简称 TCB，它是为实现数据库安全的所有实施策略与机制的集合，它是实施、检查、监督数据库安全的机构，这是数据库安全中的一个基本概念，在下面将多次引用这个概念。

(2) 主体 (subject)、客体 (object) 与主客体分离

在讨论数据库安全时，将数据库中与安全有关的实体一一列出，它们是数据库中数据及其载体，包括数据表、视图、快照、存储过程以及数据文件等，还包括数据库中数据访问者，如数据库用户、DBA、进程、线程等，然后将实体抽象成客体与主体两个部分。所谓客体即是数据库中数据及其载体。所谓主体即是数据库中数据访问者，进程、线程等，在数据库安全中有关的实体是独立的并且只能被标识成为一种类型（客体或主体），因此可以将数据库中的有关实体集分解成为两个子集：客体（子）集与主体（子）集。这两个子集互不相交且覆盖整个实体集，构成了实体集的一个划分，同时这两个子集存在着单向访问的特性，即主体子集中的实体可以在一定条件下访问客体子集，此种关系可用图 4.1 表示。

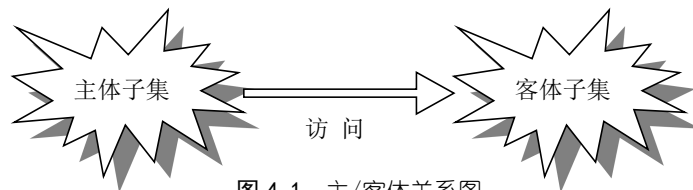


图 4.1 主/客体关系图

数据库安全中有关实体的主/客体划分以及主/客体间的访问关系构成了数据库安全的基础。

(3) 身份标识与鉴别 (identification and authentication)

在数据库安全中主体访问客体时需进行一定的安全控制与检查，目前存在三种控制方式，它们是身份标识与鉴别、自主访问控制与强制访问控制。

在数据库安全中每个主体必须有一个标志自己身份的标识符，用以区别不同的主体，当主体访问客体时可信计算基鉴别其身份并阻止非法访问。

目前常用的标识与鉴别的方法有用户名、口令等，也可用计算过程与函数，最新的也可用密码学中的身份鉴别技术等手段。

身份标识与鉴别是主体访问客体的最简单也是最基本的安全控制方式。

(4) 自主访问控制 (discretionary access control)

自主访问控制简称 DAC，它是主体访问客体的一种常用的安全控制方式，它较为适合于单机方式下的安全控制。

DAC 的安全控制机制是一种基于存取矩阵的模型，此种模型起源于 1971 年，由 Lampson 创立，1973 年经 Galgham 与 Denning 改进，到 1976 年由 Harrison 最后完成，此模型由 3 种元素组成，它们是主体、客体与存/取操作，它们构成了一个矩阵，矩阵的列表示主体，而矩阵的行则表示客体，而矩阵中的元素则是存/取操作（如读、写、删、改等），在这个模型中，指定主体（行）与客体（列）后可根据矩阵得到指定的操作，其示意图如

图 4.2 所示。

主体 \ 客体	主体 1	主体 2	主体 3	...	主体 n
客体 1	Read	Write	Write	...	Read
客体 2	Delete	Read/Write	Read	...	Read/Write
...
客体 m	Read	Updata	Read/Write	...	Read/Write

图 4.2 存/取矩阵模型示意图

在自主访问控制中主体按存取矩阵模型要求访问客体,凡不符合存取矩阵要求的访问均属非法访问。在自主访问控制中,其访问控制的实施由 TCB 完成。

在自主访问控制中的存取矩阵的元素是可以经常改变的,主体可以通过授权的形式变更某些操作权限,因此在自主访问控制中访问控制受主体主观随意性的影响较大,其安全力度尚嫌不足。

(5) 强制访问控制 (mandatory access control)

强制访问控制简称 MAC,它是主体访问客体的一种强制性的安全控制方式,强制访问控制方式主要用于网络环境,对网络中的数据库安全实体作统一的、强制性的访问管理,为实现此目标首先需对主/客体作标记 (label),标记分两种:一种是安全级别标记 (label of security level),另一种是安全范围标记 (label of security category)。安全级别标记是一个数字,它规定了主、客体的安全级别,在访问时只有主体级别与客体级别满足一定比较关系时,访问才能允许。安全范围标记是一个集合,它规定了主体访问的范围,在访问时只有主体的范围标记与客体的范围标记满足一定包含关系时,访问才能允许。综上所述,可以看出,强制控制访问的具体实施步骤如下:

1) 对每个主体、客体作安全级别标记与安全范围标记。

2) 主体在访问客体时由 TCB 检查各自标记,只有在主、客体两种标记都符合允许访问条件时访问才能进行。

强制访问控制的实施机制是一种叫 Bell-La padula 模型,在此模型中任一主体/客体均有一个统一标记,它是一个二元组,其中一个为整数(叫分层密级),而另一个则为集合(叫非分层范畴级)它可以用: $(n, \{A, B, C, \dots\})$ 表示,当主体访问客体时必须满足如下条件:

1) 仅当主体分层密级大于或等于客体分层密级,且主体非分层范畴集合包含或等于客体非分层范畴集合时,主体才能读客体。

2) 仅当主体分层密级小于或等于客体分层密级,且主体非分层范畴集合被包含或等于客体非分层范畴集合时,主体才能写客体。

设主体的标记为 (n, s) , 客体的标记为 (m, s') 则主体读客体的条件为:

$$n > m \quad \text{且} \quad S \supseteq S'$$

主体写客体的条件是:

$$n \leq m \quad \text{且} \quad S \subseteq S'$$

在强制访问控制中主体按 Bell-La padula 模型要求访问客体,凡不符合模型要求的访问均属非法访问,在强制访问控制中,其访问控制的实施由 TCB 完成。

强制访问控制中的主、客体标记由专门的安全管理员设置,任何主体均无权设置与授

权，它体现了在网上对数据库安全的强制性与统一性。

(6) 数据完整性 (data integrity)

数据完整性保护数据库中数据不受破坏，亦即是防止非法使用删除 (delete)，修改 (update) 等影响数据完整的操作。

(7) 隐蔽通道 (hidding cannel)

在主体访问客体时一般均通过正常路径访问，因此有可能使用 TCB 在访问路径中作检查，但是，实际上在实施时往往存在着多种非正常访问路径，它们构成了非法访问渠道，这种渠道往往比较隐蔽且逃过了 TCB 的检查，称为隐蔽通道，在数据库安全中，一定要寻找与防止隐蔽通道的出现，一旦出现要采取措施以堵塞此类通道。

(8) 数据库安全的形式化模型 (formulization of database security)

由于数据库安全在整个系统中的重要性，因此需要建立一套有效的形式化体系，用于保证自身的正确性，发现并填补安全漏洞，防止隐蔽通道，并可为数据安全进一步研究提供理论基础，因此数据库安全的形式化已成为高级别数据库安全的必要条件。目前，对此方面的研究很多，有基于代数的格理论形式化研究，有基于逻辑的形式化研究，但是，均未达到理想要求，从而对数据库安全级别的提高产生直接的影响。因此，对数据库安全形式化的研究是当前数据库安全领域的重要课题。

(9) 审计 (audit)

在数据库安全中除了采取有效手段对主体访问客体作检查外，还采用辅助的跟踪、审计手段，随时记录主体对客体访问的轨迹，并作出分析供参考，同时在一旦发生非法访问后能即时提供初始记录供进一步处理，这就是数据库安全中的审计。

审计的主要功能是对主体访问客体作即时的记录，记录内容包括：访问时间、访问类型、访问客体名、是否成功等。为提高审计效能，还可设置审计事件发生积累机制，当超过一定阈值时能发出报警，以提示采取措施。

(10) 访问监控器 (access monitor)

上述功能还需要有一个独立的抗篡改的复杂度足够小的系统实体以实现数据安全，这就是访问监控器。访问监控器在功能上仲裁主体对客体的全部访问，具有扩充的审计功能，提供系统恢复机制，它是一个独立的物理机构，由一定的软件与硬件联合组成。

4.1.3 数据库的安全标准

目前，国际上及我国均颁布有关数据库安全的等级标准，最早的标准是美国国防部 (DOD) 1985 年所颁布的“可信计算机系统评估标准” (trusted computer system evaluation criteria)，简称 TCSEC。1991 年美国国家计算机安全中心 (NCSC) 颁布了“可信计算机系统评估标准——关于数据库系统解释” (trusted database interpretation)，简称 TDI。1996 年国际标准化组织 ISO 又颁布了“信息技术安全技术——信息安全性评估准则” (information technology security techniques—evaluation criteria for IT security)，简称 CC 标准。我国政府于 1999 年颁布了“计算机信息系统安全保护等级划分准则”。这是一种以 TCSEC 为蓝本的标准，2001 年我国又颁布了以 CC 为蓝本的标准：“信息技术安全技术——信息安全评估准则”。在 2005 年又颁布了以公安部所制订的行业标准为蓝本的有关数据库的安全国家标准。目前国际上广泛采用的是美国标准 TCSEC (TDI)，在此标准中将数据安全划分为四组七级，我国标准则划分为五个级别。下面分别讨论这些分级标准。

1. TCSEC (TDI) 标准

TCSEC (TDI) 标准是目前常用的标准, 现介绍如下。

- (1) D 级标准。为无安全保护的系统。
- (2) C1 级标准。满足该级别的系统必须具有如下功能:
 - 1) 主体、客体及主、客分离;
 - 2) 身份标识与鉴别;
 - 3) 数据完整性;
 - 4) 自主访问控制。

其核心是自主访问控制。C1 级安全适合于单机工作方式, 目前国内使用的系统大都符合此标准。

- (3) C2 级标准。满足该级别的系统必须具有如下功能:
 - 1) 满足 C1 级标准的全部功能;
 - 2) 审计。

C2 级安全的核心是审计, C2 级适合于单机工作方式, 目前国内使用的系统一部分符合此种标准。

- (4) B1 级标准。满足该级别的系统必须具有如下功能:
 - 1) 满足 C2 级标准全部功能;
 - 2) 强制访问控制。

B1 级安全的核心是强制访问控制, B1 级适合于网络工作方式, 目前国内使用的系统基本不符合此种标准, 而在国际上有部分系统符合此种标准。

一个数据库系统凡符合 B1 级标准者称为安全数据库系统 (secure DB system) 或可信数据库系统 (trusted DB system)。因此可以说我国国内所使用的系统基本不是安全数据库系统。

- (5) B2 级标准。满足该级别的系统必须具有如下功能:
 - 1) 满足 B1 级标准全部功能;
 - 2) 隐蔽通道;
 - 3) 数据库安全的形式化。

B2 级安全核心是隐蔽通道与形式化, 它适合于网络工作方式, 目前国内外均尚无符合此类标准的系统, 其主要的难点是数据库安全的形式化表示困难。

- (6) B3 级标准。满足该级别的系统必须具有如下功能:
 - 1) 满足 B2 级标准的全部功能;
 - 2) 访问监控器。

B₃ 级安全核心是访问监控器, 它适合于网络工作方式, 目前国内外均尚无符合此类标准的系统。

- (7) A 级标准。满足该级别的系统必须具有如下功能:
 - 1) 满足 B3 级标准的全部功能;
 - 2) 较高的形式化要求。

此级为安全之最高等级, 应具有完善之形式化要求, 目前尚无法实现, 仅是一种理想化的等级。

2. 我国国家标准

我国国家标准于 1999 年颁布, 为与国际接轨其基本结构与 TCSEC 相似。我国标准分 5 级, 从第 1 级到第五级基本上与 TCSEC 标准的 C 级 (C1, C2) 及 B 级 (B1, B2, B3) 一致, 现将我国标准与 TCSEC 标准比较如表 4.1 所示。

表 4.1 TCSEC 标准与国标的比较

TCSEC 标准	我国标准
D 级标准	无
C1 级标准	第 1 级: 用户自主保护级
C2 级标准	第 2 级: 系统审计保护级
B1 级标准	第 3 级: 安全标记保护级
B2 级标准	第 4 级: 结构化保护级
B3 级标准	第 5 级: 访问验证保护级
A 级标准	无

4.1.3 SQL 对数据库安全的支持

在 SQL'92 中提供了 C₁ 级数据库安全的支持, 它们是:

(1) 主体、客体及主/客体分离

在 SQL'92 中设置了主体与客体, 它包括用户与 DBA 作为主体以及表、视图作为客体, 此外还包括主/客体分离的原则。

(2) 身份标识与鉴别

在 SQL'92 中通过用户口令作身份标识与鉴别。

(3) 数据完整性

在 SQL'92 中提供了数据完整性功能, 它包括实体完整性、参照完整性及用户定义完整性规则 (详见 4.2 节)。

(4) 自主访问控制与授权功能

在 SQL'92 中提供了自主访问控制权的功能, 它包括了操作、数据域与用户三部分。

1) 操作。SQL'92 提供了六种操作权限。

- SELECT 权: 即是查询权;
- INSERT 权: 即是插入权;
- DELETE 权: 即是删除权;
- UPDATE 权: 即是修改权;
- REFERENCE 权: 即是定义新表时允许使用它表的关键字作为其外关键字;
- USAGE 权: 允许用户使用已定义的属性。

2) 数据域。数据域即是用户访问的数据对象的粒度, SQL 包含三种数据域。

- 表: 即以基表作为访问对象;
- 视图: 即以视图作为访问对象;
- 属性: 即以基表中属性作为访问对象。

3) 用户。即是数据库中所登录的用户。

4) 授权语句。SQL'92 提供了授权语句, 授权语句的功能是将指定数据域的指定操作授予指定的用户, 其语句形式如下:

```
GRANT<操作表>ON<数据域>TO<用户名表>[WITH GRANT OPTION]
```

其中 WITH GRANT OPTION 表示获得权限的用户还能获得传递权限, 即能将获得的

权限传授给其它用户。

例 4.1 GRANT SELECT, UPDATE ON S TO XU LIN WITH GRANT OPTION
表示将表 S 上的查询与修改权授予用户徐林 (XU LIN), 同时也表示用户徐林可以将此权限传授给其它用户。

5) 回收语句。用户 A 将某权限授予用户 B, 则用户 A 也可以在其认为必要时将权限从 B 中收回, 收回权限的语句称为回收语句, 其具体形式如下:

REVOKE<操作表>ON<数据域>FROM<用户名表>[RESTRICT/CASCADE]

语句中带有 CASCADE 表示回收权限时要引起连锁回收, 而 RESTRICT 则表示不存在连锁回收时才能回收权限, 否则拒绝回收。

例 4.2 REVOKE SELECT, UPDATE ON S FROM XU LIN CASCADE
并且还表示从用户徐林中收回表 S 上的查询与修改权, 并且是连锁收回。

在 SQL 标准中并不设置审计、强制访问控制等功能, 因此该标准达到 C₁ 级安全要求。

6) 自 SQL'99 以后 SQL 提供了角色 (role) 功能。角色是一组操作权限的集合, 目的是为简化操作权限管理, 一般提供 3 种角色, 它们是 CONNECT、RESOURCE 和 DBA, 其中每个角色拥有一定的操作权限。

① CONNECT 权限。该权限是用户的最基本权限, 每个登录用户至少拥有 CONNECT 权限, CONNECT 权限包括如下内容。

- 可以查询或更新数据库中的数据
- 可以创建视图或定义表的别名。

② RESOURCE 权限。该权限是建立在 CONNECT 基础上的一种权限, 它除拥有 CONNECT 的操作权外, 还具有创建表及表上索引以及在该表上所有操作的权限和对该表所有操作作授权与回收的权限。

③ DBA 权限。DBA 拥有最高操作权限, 它除拥有 CONNECT 与 RESOURCE 权限外, 能对所有表的数据作操纵, 并具有控制权限与数据库管理权限。

DBA 通过角色授权语句将用户及相应角色登录, 此语句形式如下:

GRANT<角色名>TO<用户名表>

此语句执行后, 相应的用户名及其角色均进入数据库中数据字典, 此后相应用户即拥有其指定的角色。

同样, DBA 可用 REVOKE 语句取消用户的角色。

例 4.3 GRANT CONNECT TO XU LIN
此语句表示将 CONNECT 权授予用户徐林。

例 4.4 REVOKE CONNECT FROM XU LIN
此语句表示从用户徐林处收回 CONNECT 权限。

(5) 审计

SQL 不支持审计, 但 ORACLE 设置审计, 我们用 ORACLE 审计功能以介绍审计内容:

1) 审计事件。在 ORACLE 中设置两种审计, 一种是用户审计, 另一种是系统审计。用户审计由用户设置, 用以审计该用户所创建的基表与视图的所有访问操作, 系统审计则由 DBA 设置用以审计用户登录, GRANT 及 REVOKE 操作以及其它管理型操作, 所有一

切引起审计的操作称为审计事件。

2) 审计内容。一旦审计事件产生, ORACLE 中的审计模块即行启动, 它自动记录事件的有关内容:

- 操作类型 (查询/修改/删除/插入……)。
- 事件发生时间。
- 操作终端标识与操作者标志。
- 所操作的数据域 (基表、视图、属性等)。

3) 审计操作。ORACLE 提供两条审计语句: AUDIT 与 NOAUDIT。其中 AUDIT 语句用于选择审计事件, 审计数据域以及是否审计, 而 NOAUDIT 语句则是取消审计, 它们的语句形式如下:

AUDIT<操作表>ON<数据域名>

NOAUDIT<操作表>ON<数据域名>

例 4.5 AUDIT UPDATE, DELETE ON S

此语句表示对基表 S 上设置对修改与删除的审计。

例 4.6 NOAUDIT DELETE ON S

此语句表示取消对基表 S 的删除的审计。

4.2 数据库的完整性保护

数据库的完整性保护指的是数据库中数据正确性的维护, 任何一个数据库都会由于某些自然因素而受到局部或全局的破坏。它们有的是无法避免的, 因此如何及时发现并采取措施防止错误扩散并及时恢复, 这是完整性保护的主要目的。

4.2.1 数据库完整性保护的功能

在数据库中为实现完整性保护须有三个基本功能, 它们是:

1. 设置功能: 须设置完整性约束条件 (又称完整性规则), 这是一种语义约束条件, 它由系统或用户设置, 它给出了系统及用户对数据库完整性的基本要求。
2. 检查功能: 数据库完整性保护必须有能检查数据库中数据是否有违反约束条件的现象出现。
3. 处理功能: 在出现有违反约束条件的现象时须有即时处理的能力。

4.2.2 完整性规则的三个内容

数据库完整性规则由如下三部分内容组成。

1. 实体完整性规则(entity integrity rule)

这条规则要求基表上的主键中属性值不能为空值, 这是数据库完整性的最基本要求, 因为主键是惟一决定元组的, 如为空值则其惟一性就成为不可能的了。

2. 参照完整性规则 (reference integrity rule)

这条规则也是完整性中的为基本规则, 它不允许引用不存在的元组: 亦即是说在基表中的外键要么为空值, 要么其关联表中必存在的元组。如在基表 S (sno, sn, sd, sa) 与 SC (sno, cno, g) 中, SC 的主键为 (sno, cno), 而 S 的外键为 sno, SC 与 S 通过 sno 相关联, 而参照完整性规则要求 SC 中的 sno 的值必在 S 中有相应元组值, 如有 SC (S₁₃, C₈, 70)

则必在 S 中存在 S (S₁₃,..., ..., ...)。

这条规则给出了表之间相关联的基本要求。

上述两种规则是关系数据库所必需遵守的规则，因此任何一个 DBMS 必须支持。

3. 用户定义的完整性规则 (userdefined integrity rule)

这是针对具体数据环境与应用环境由用户具体设置的规则，它反映了具体应用中数据的语义要求。

在下节中我们主要讨论完整性约束的设置、检查与处理。

4.2.3 完整性约束的设置、检查与处理

用户定义的完整性规则一般由如下三部分组成。

1. 完整性约束条件设置

用户一般可以用完整性约束语句定义数据库应用中的语义约束条件,它们可以是:

(1) 域约束: 可以约束数据库中数据域的范围与条件, 该约束可定义一个约束名以及一个约束条件, 在 SQL'92 中其形式如下:

```
CONSTRAINT<约束名>
```

```
CHECK: <约束条件>
```

域约束往往定义在 SQL'92 中创建域语句的后面。

例 4.7 定义一个域: COLOR, 其约束名为: VALID-COLORS, 约束条件为: 'Red'、'Yellow'、'Blue'、'Green', 此时可用下面语句表示:

```
CONSTRAINT VALID-COLORS
```

```
CHECK (VALUE IN('Red'、'Yellow'、'Blue'、'Green'))
```

(2) 表约束: 可以约束表的范围与条件, 它包括三部分内容: 候选键定义, 外键定义及检查约束。

- 候选键约束, 还包括主键约束。可用: UNIQUE<列名序列>及 PRIMARY KEY<列名序列>, 分别表示基表中的候选键及主键, 它们一般定义在创建基表语句的后面。

- 外键定义: 可用 SQL'92 中的下面的形式表示外键定义:

```
FOREIGN KEY<列名序列>
```

```
REFERENCE<参照表><列名序列>
```

```
[ON DELETE<参照动作>]
```

```
[ON UPDATE<参照动作>]
```

其中第一个<列名序列>是外键而第二个<列名序列>则是参照表中的主键或候选键, 而参照动作有五个, 它们是: NO ACTION(默认值), CASCADE、RESTRICT、SET NULL 及 SET DEFAULT 分别表示无动作、动作受牵连、动作受限、置空及置默认值等, 其中动作受牵连表示在删除元组(或修改)时相应表中的相关元组一起删除(或修改), 而动作受限则表示在删除(或修改)时仅限于指定的表的元组。它们一般也定义在创建基表语句的后面。

- 检查约束: 用于对基表内的属性间设置语义约束, 所使用的语句与域约束中的相同, 它一般也定义在创建基表的语句后面。

例 4.8

```
CREATE TABLE Student
```

```
( Sno    NUMBER (8),  
  Sname  VARCHAR (20),  
  Sage   NUMBER (20),  
  PRIMARY KEY (Sno));
```

此中用创建表语句定义 Student，在最后由 PRIMARYKEY (Sno) 确定主键为 Sno。

例 4.9

```
CREATE TABLE EMP  
  (Empno    NUMBER (4),  
  Ename     VARCHAR (10),  
  Job       VARCHAR (9),  
  Mgr       NUMBER (4),  
  Sal       NUMBER (7,2),  
  Deptno    NUMBER (2),  
  PRIMARYKEY(Empno)  
  FOREIGN KEY (Deptno)  
  REFERENCES DEPT (Deptno)  
  ON DELETE CASCADE);
```

在此例中用创建表语句定义 EMP，在定义最后确定有关实体完整性及参照完整性的参数，其中：

- 1) PRIMARY KEY (Empno) 确定主键为 Empno;
- 2) FOREIGN KEY(deptno) 确定外键为 Deptno;
- 3) REFERENCES DEPT (deptno)确定其外键所对应的另一表为 DEPT，而所对应的主键为 deptno;

4) ON DELETE CASCADE 表示当要删除 DEPT 表中某一元组时，系统也要检查 EMP 表，若找到相应元组则将它们也随之删除。

(3) 断言：当完整性约束涉及到多个基表时，此时可用断言 (assertion) 以建立多表间属性的约束条件。在 SQL'92 中，可用创建断言与撤消断言建立与撤消约束条件，它们是：

```
CREATE ASSERTION<断言名>CHECK<约束条件>  
DROP ASSERTION<断言名>
```

其中它的约束条件一般可有三种形式：

- 非空属性 (NOT NULL);
- 惟一属性 (UNIQUE);
- 用布尔表达式表示的属性或属性间关系。

例 4.10

```
CREATE ASSERTION student-constraint  
CHECK (Sno BETWEEN '90000'AND '99999')  
CHECK(sn is NOT NULL)  
CHECK (Sage<'29')
```

在此例中是建立了三个完整性约束条件：

- (1) 学号必须在 90000~99999 之间。
- (2) 学生姓名不能为空值。
- (3) 学生年龄必须小于 29 岁。

2. 完整性约束条件的检查

一般在数据库管理系统内部设置专门软件，对完整性约束语句所设置的完整性约束条件作随时的检查，以保证完整性约束条件的设置能得到随时的监督与实施。

3. 完整性约束条件的处理

在数据库管理系统内部同样也设置有专门软件，对一旦出现违反完整性约束条件的现象时能及时进行处理，以保证系统内部数据的完整性，处理的方法有简单与复杂之分，简单的处理方式是拒绝执行并报警或报错，复杂的方式是调用相应的过程进行处理。

4.2.4 触发器

触发器（trigger）是近年来在数据库中使用的较多的一种功能，它起因于用作完整性保护，但目前其使用已远远超出此范围，由于它体现了数据库的主动功能，因此大量用于主动性领域。

触发器一般由触发事件与结果动作两部分组成，其中触发事件给出了触发条件，当触发条件一旦出现，触发器则立刻调用对应的结果动作对触发事件进行处理，整个触发器过程可用图 4.3 简单地表示。

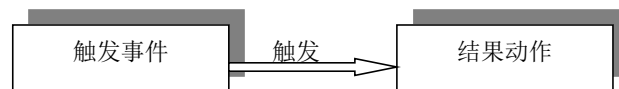


图 4.3 触发器

目前在一般数据库管理系统中触发事件往往仅限于增、删、改操作。

触发器在数据库完整性保护中起着很大的作用，一般可用触发器完成很多数据库完整性保护的功能，其中触发事件即是完整性约束条件，而完整性检查即是触发器的结果动作，最后结果过程的调用即是完整性检查的处理。

在数据库系统管理中一般都有创建触发器的功能，在 SQL 中其语句形式为：

```
CREATE TRIGGER <触发器名><触发时间><触发事件>  
ON<表名>REFERENCES<新、旧元组别名>  
<触发动作>
```

其中<触发时间>有两种：

- AFTER 表事件执行后触发器才被激活；
- BEFORE 表事件执行前触发器就被激活。

<表名>给出了所触发的表。

<触发事件>一般有三种，它们是 DELETE、UPDATA 及 INSERT。在 UPDATA 后允许跟 OF<属性表>。

<REFERENCES>给出触发前后的元组变量别名，一般触发前用 OLD AS<别名>，触发后用 NEW AS<列名>表示。

<触发动作>则给出了触发器执行的动作，它一般是一个过程，它由三部分组成：

- 触发间隔尺寸：用 FOR EACH 子句定义，常用有 FOR EACH ROW 及 FOR EACH STATEMENT 两种，它们分别称‘元组级触发器’及‘语句级触发器’。
- 动作时间条件：用 WHEN 子句定义。
- 动作体：是一个过程，它给出了触发器动作的实体。

一旦创建了触发器后，系统即能随时检查触发事件，当该事件成立时，即调用相应过程以处理该事件。

下面的例 4.11 给出了触发器的一个实例。

例 4.11

```
CREATE TRIGGER update-sal
AFTER INSERT, UPDATE OF Sal Pos
ON Teacher
REFERENCNG OLD AS OT NEW AS NT
FOR EACH ROW
WHEN (: NT • new • pos ='教授') /*某教师晋级为教授*/
BEGIN
    IF: OT • new • sal<5000 THEN : NT • new • sal: =5000;
    END IF;
END;
```

此例表示创建一个名为 update-sal 的触发器，该触发器的触发事件是“修改或增添教师职务工资，如教师晋升为教授”，其调用的过程是“则若教授工资低于 5000 元时，则自动转为 8000 元”。

习题四

- 4.1 什么叫安全数据库？试作说明。
- 4.2 试述 TCSEC (TDI) 中的四类七级标准的基本内容以及与我国五级标准间的关系。
- 4.3 试述自主访问控制的主要内容。
- 4.4 为什么强制访问控制的安全保护能力强于自主访问控制，请说明理由。
- 4.5 什么叫强制访问控制与 Bell-Lapadula 模型？试作说明。
- 4.6 审计的作用是什么？试作说明。
- 4.7 试解释触发器与完整性约束间的关系。
- 4.8 试说明完整性规则的 3 个组成内容。
- 4.9 试解释网络环境下数据安全的重要性。
- 4.10 在下面的学生数据库中：

S (sno , sn , sd , sa)

SC (sno , cno , g)

C (cno , cn , pno)

请用 SQL'92 中的 GRANT 及 REVOKE 语句以完成如下的授权控制：

- (1) 用户张军对三个表的 SELECT 权；

-
- (2) 用户李林对三个表的 INSERT 及 DELETE 权;
 - (3) 用户王星对表 SC 有查询权, 对 S 及 C 有更改权;
 - (4) 用户徐立功具有对三个表的所有权限;
 - (5) 撤销张军、李林的所授予的权限;
- 4.11** 对学生数据库设置如下审计:
- (1) 对 S 设置 DELETE 及 UPDATA 审计;
 - (2) 对 SC 设置 UPDATA、INSERT 及 DELETE 审计;
 - (3) 撤销对 SC 的 DELETE 审计;
- 4.12** 什么叫实体完整性与参照完整性规则, 试解释之;
- 4.13** 在 SQL'92 中提供什么语句以支持定义完整性规则;
- 4.14** 在学生数据库中用 SQL'92 中语句以定义下列完整性规则:
- (1) 定义 S, SC 及 C 的主码;
 - (2) 定义 SC 中外码 pno;
 - (3) 定义 S 中学生年龄不得超过 50 岁。
- 4.15** 什么是数据库的完整性? 试说明之。
- 4.16** 什么是数据库的安全性? 试说明之。
- 4.17** 数据库的完整性约束条件分哪几类? 试说明之。
- 4.18** 数据库的安全性与完整性概念有什么区别与联系?
- 4.19** 试介绍一个你所熟悉的 DBMS 产品的安全性与完整性功能。
- 4.20** 试说明数据库安全性在整个计算机系统中的地位与作用。

第四章复习指导

本章主要介绍数据库的安全性与完整性保护,在网络发达的今天,数据库已基本上在网络环境下使用,数据安全已成为网上应用的重要条件,读者学习本章后能对数据安全与完整性须有深入了解。

1. 数据安全性包含:

(1) 数据库安全性保护——防止非法使用数据库。

(2) 数据安全性十大基本概念:

- 可信计算基;
- 主体/客体及主/客体分离原则;
- 身份标识与鉴别;
- 自主访问控制;
- 强制访问控制;
- 数据完整性;
- 隐蔽通道、
- 形式化模型;
- 审计;
- 访问监控器。

(3) 数据安全性的等级:

- D级——无安全要求;
- C₁级(第一级)——主体/客体及主/客体分离、身份标识与鉴别、数据完整性及自主访问控制;
- C₂级(第二级)——C₁级+审计;
- B₁级(第三级)——C₂级+强制访问控制;
- B₂级(第四级)——B₁级+隐蔽通道+形式化模型;
- B₃级(第五级)——B₂级+访问监控器;
- A级——B₃级+更高的形式化模型;

2. 数据完整性保护

(1) 数据完整性保护——数据库中数据的正确性维护。

(2) 完整性的三个内容:

- 实体完整性;
 - 参照完整性;
 - 用户定义完整性
- (3) 触发器
- 域约束;
 - 表约束;
 - 断言。

3. 本章重点内容

- 数据库安全等级
- 强制访问控制

第五章 事务处理、并发控制与故障恢复技术

本章将事务处理、并发控制与数据库故障恢复技术合并在一起讨论，这主要是由于这三者有着紧密的联系与概念上的一致性，而且它们均以事务处理为核心。因此，在本章中首先讨论事务处理，紧接着讨论与事务处理紧密相关的并发控制技术与故障恢复技术。

5.1 事务处理

5.1.1 事务

数据库是一个共享的数据实体，多个用户可以在其中做多种操作(包括读操作与写操作)，为了保持数据库中数据的一致性，每个用户对数据库的操作必须具有一定操作连贯性，为说明此问题我们从一例说起，设某银行有 A, B 两个账户，它们分别存有 20000 元与 10000 元人民币，现有一笔转账业务须从 A 将 5000 元钱转至 B，此应用的操作可描述如下：

应用 T1：

```
Read (A)
A: = A - 5000
Write (A)
Read (B)
B: = B + 5000
Write (B)
```

在执行 T1 前 $A = 20000$ ， $B = 10000$ ，其银行总存款为 $A + B = 20000 + 10000 = 30000$ 元，在此六个步骤完成后，A 与 B 分别为：

```
A = 15000
B = 15000
```

其银行的总存款数为 $A + B = 15000 + 15000 = 30000$ ，此时仍保持其总款数不变，亦即是说，在数据库中从原有的一致性在经过操作 T1 后保持了新的一致性。但是在此六个步骤操作中必须作为整体一次完成，其中间是不应允许被其他应用所打断的，否则其一致性就受到破坏。如在第三步结束后中间被另一个应用 T2 打断，此时 T2 所面对的数据库是一个不一致的数据库，即原有银行总存款数为 $A + B = 30000$ ，但是在此时 T1 执行并没有完成，呈现在 T2 面前的总存款数为 $A + B = 15000 + 10000 = 25000$ ，此时出现了不一致性，而 T2 在此不一致性的数据库面前是无法正确执行操作的。为解决此问题，必须保证 T1 执行操作的连贯性，即 T1 整个六步操作必须连贯完成，其中间不允许被其他应用所打断，这就引出了事务的概念。

事务 (transaction) 是数据库应用程序的基本逻辑工作单位，在事务中集中了若干个数据库操作，它们构成了一个操作序列，它们要么全做，要么全不做，是一个不可分割的工作单位。

一般而言，一个数据库应用程序是由若干事务组成，每个事务构成数据库的一个状态，它形成了某种一致性，而整个应用程序的操作过程则是通过不同事务使数据库由某种一致性不断转换到新的一致性的过程。

5.1.2 事务的性质

事务具有四个特性，它们是事务的原子性(atomicity)、一致性 (consistency)、隔离性 (isolation) 以及持久性 (durability)，简称为事务的 ACID 性质。

1. 原子性

事务中所有的数据库操作是一个不可分割的操作序列，这些操作要么全执行，要么全不执行。

2. 一致性

事务执行的结果将使数据库由一种一致性到达了另一种新的一致性。

3. 隔离性

在多个事务并发执行时事务不必关心其他事务的执行，如同在单用户环境下执行一样。

4. 持久性

一个事务一旦完成其全部操作后，它对数据库的所有更新永久地反映在数据库中，即使以后发生故障也应保留这个事务执行的结果。

事务及其 ACID 性质保证了并发控制与故障恢复的顺利执行，因此在下面的并发控制与故障恢复的讨论中均以事务为基本执行单位。

5.1.3 事务活动

一个事务一般由 SET TRANSACTION 开始至 COMMIT 或 BOLLBACK 结束。在事务开始执行后，它不断做 Read 或 Write 操作，但是，此时所做的 Write 操作，仅将数据写入磁盘缓冲区，而并非真正写入磁盘内。在事务执行过程中可能会产生两种状况：其一是顺利执行——此时事务继续正常执行；其二是产生故障等原因而中止执行，对此种情况称事务夭折 (abort)，此时根据原子性性质，事务需返回开始处重新执行，此时称事务回滚 (rollback)。在一般情况下，事务正常执行直至全部操作执行完成，以后再执行事务提交 (commit) 后整个事务结束，所谓提交即是将所有在事务执行过程中写在磁盘缓冲区的数据，真正、物理地写入磁盘内，从而完成整个事务。因此，事务的整个活动过程可以用图 5.1 表示。

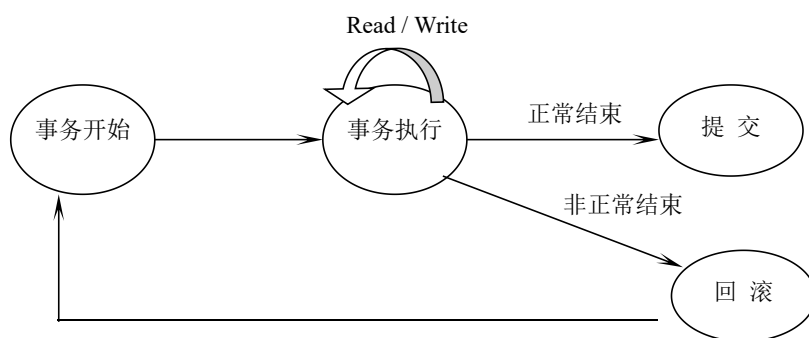


图 5.1 事务活动过程图

5.1.4 有关事务的语句

一个应用由若干个事务组成，事务一般嵌入在应用中，在 SQL'92 中应用所嵌入的事务由三个语句组成，它们是一个置事务语句与两个事务结束语句。

1. 置事务语句 SET TRANSACTION

此语句表示事务从此句开始执行，此语句也是事务回滚的标志点，在大多数情况下，可以不用此语句，对每个事务结束后的数据库的操作都包含着一个新事务的开始。

2. 事务提交语句 COMMIT

当前事务正常结束，用此语句通知系统，此时系统将所有磁盘缓冲区中数据写入磁盘内，在不用“置事务”语句时，同时表示开始一个新的事务。

3. 事务回滚语句 ROLLBACK

当前事务非正常结束，用此语句通知系统，此时系统将事务回滚至事务开始处并重新开始执行。

有时对事务还可以作分类，它们是按事务的读/写类型分类。一般事务是由“读”与“写”语句组成，而当事务仅由“读”语句组成时则此事务的最终提交变得十分简单。因此，有时可以将事务分成读写型与只读型两种，以方便事务的不同处理。

(1) 只读型 (read only): 事务对数据库的操作只能是“读”语句。定义此类型后表示随后的事务均为只读型，直到新的类型定义出现为止。

(2) 读/写型 (read/write): 事务对数据库可作“读”与“写”操作，定义此类型后表示随后的事务均为读/写型，直到新的类型定义出现为止，此类型定义有时可以缺省。

上述两种类型可以用置事务语句带参数形式表示如下：

```
SET TRANSACTION READ ONLY
SET TRANSACTION READ WRITE
```

5.2 并发控制技术

在数据库中多个应用是以事务为单位执行的，此种方法虽然能保证执行的正确性但效率较低，在本节中将要讨论通过并发控制技术以提高执行效率的方法。

5.2.1 事务的并发执行

在多个应用、多个事务执行中有几种不同的执行方法。

(1) 串行执行：以事务为单位，多个事务依次顺序执行，此种执行称为串行执行，串行执行能保证事务的正确执行。

(2) 并发执行：多个事务按一定调度策略同时执行，此种执行称为并发执行。

(3) 并发执行的可串行化：事务的并发执行并不能保证事务正确性，因此需要采用一定的技术，使得在并发执行时像串行执行时一样（正确），此种执行称为并发事务的可串行化 (serializability)。而所采用的技术则称为并发控制 (concurrent control) 技术。

下面的例子分别给出串行执行、并发执行（不正确）以及并发执行可串行化（正确）的实际执行过程。

例 5.1 再次以银行转账为例，事务 T1 从账号 A（初值为 20000 元）转 10000 元至账号 B（初值为 20000 元），事务 T2 从账号 A 转 5000 元的款项至账号 B，其具体的程序如下：

<p>T₁:</p> <p>Read (A)</p> <p>A: = A-10000</p> <p>Write (A)</p> <p>Read (B)</p> <p>B: = B+10000</p> <p>Write (B)</p>	<p>T₂:</p> <p>Read (A)</p> <p>A: = A-5000</p> <p>Write (A)</p> <p>Read (B)</p> <p>B: = B+5000</p> <p>Write (B)</p>
---------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------

次序	T ₁	T ₂	A	B	A+B
1	Read (A)		20000	20000	40000
2	A:=A-10000		20000	20000	40000
3	Write (A)		10000	20000	30000
4	Read (B)		10000	20000	30000
5	B:=B+10000		10000	20000	30000
6	Write (B)		10000	30000	40000
7		Read (A)	10000	30000	40000
8		A:=A-5000	10000	30000	40000
9		Write (A)	5000	30000	35000
10		Read (B)	5000	30000	35000
11		B:=B+5000	5000	30000	35000
12		Write (B)	5000	35000	40000

图 5.2 串行执行之一

在图 5.2 中，两个事务 T₁与 T₂串行执行，不管先后次序其执行结果都是正确的，亦即是说两事务执行前与执行后的结果是一致的，执行前 A+B=4000 而执行后仍为 A+B=4000，达到了新的一致。

在图 5.3 与图 5.4 并发执行中，图 5.3 所示的并发执行所执行结果帐号 A 与 B 总和为 A+B=10000+25000=35000 与执行前的 A+B=40000 元产生了不一致，因此此种并发执行是错误的。而图 5.4 所示的并发执行，其执行后结果与执行前的结果是一致的，(A+B=40000)，它与图 5.2 的执行结果一样，称为并发执行可串行化。由此可见，事务的并发执行有时是正确的，有时是错误的，它需要进行一定的控制，否则就会产生错误。

次序	T ₁	T ₂	A	B	A+B
1	Read (A)		20000	20000	40000
2	A:=A-10000		20000	20000	40000
3		Read (A)	20000	20000	40000
4		A:=A-5000	20000	20000	40000
5		Write (A)	15000	20000	35000
6		Read (B)	15000	20000	35000
7	Write (A)		10000	20000	30000
8	Read (B)		10000	20000	30000
9	B:=B+10000		10000	20000	30000
10	Write (B)		10000	30000	40000
11		B:=B+5000	10000	30000	40000
12		Write (B)	10000	25000	35000

图 5.3 并发执行（不正确）

次序	T ₁	T ₂	A	B	A+B
1	Read (A)		20000	20000	40000
2	A:=A-10000		20000	20000	40000
3	Write (A)		10000	20000	30000
4		Read (A)	10000	20000	30000
5	Read (B)	A:=A-5000	10000	20000	30000
6	B:=B+10000	Write (A)	5000	20000	25000
7	Write (B)		5000	30000	35000
8		Read (B)	5000	30000	35000
9		B:=B+50000	5000	30000	35000
10		Write (B)	5000	35000	40000

图 5.4 并发执行可串行化（正确）

下面我们讨论事务并发执行的几种错误，以及防止其错误产生的控制策略。

1. 丢失修改（lost update）

事务 T₁ 与 T₂ 进入同一数据并修改，T₂ 提交的结果破坏了 T₁ 提交的结果，导致 T₁ 的修改被丢失。其具体执行可见下面的例 5.2。

例 5.2 设有 A, B 两个民航售票点，它们按下面的次序进行订票操作：

(1) A 售票点执行事务 T1 通过网络在数据库中读出某航班的机票余额为 y , 设 $y=5$ 。
 (2) B 售票点执行事务 T2 通过网络在数据库中读出某航班的机票余额也为 y , 设 $y=5$ 。
 (3) A 售票点事务 T1 卖出一张机票修改后余额为 $y:=y-1$, 此时 $y=4$, 将 4 写回数据库。
 (4) B 售票点事务 T2 卖出一张机票修改后余额为 $y:=y-1$, 此时 $y=4$, 将 4 写回数据库。
 最后形成的结果是卖出两张机票, 但在数据库中仅减去了一张, 从而造成了错误, 这就是有名的民航订票问题, 其具体执行过程如图 5.5 所示。

这是一个典型的并发执行所引起的不一致例子, 其主要原因就是“丢失修改”引起的。

2. 不可重复读 (non-repeatable read)

不可重复读表示事务 T1 读取数据后, 事务 T2 作更新操作, 使 T1 无法再读前一次读取的结果, 具体如图 5.6 所示, 在图中 T1 对 A 与 B 求和为 160, 此后 T2 将 B 更新为 200, 而此时 T1 要对 A+B 作验算, 发现结果为 260, 从而造成验算不正确。

3. 脏读 (dirty read)

脏读指的是事务 T1 将数据 a 修改成数据 b 后将其写回磁盘, 此后事务 T2 读取该数据(数据 b), 接下来 T1 因故被撤销, 而它修改过的数据仍为 b, 从而与数据库中数据产生了一致, 而 T2 所读得的数据称为“脏”数据, 此种操作称为“脏读”, 详细例子如图 5.7 所示。

上述 3 种错误的产生主要是由于违反了事务 ACID 中的四项原则, 特别是隔离性原则, 为保证事务并发执行执行的正确, 必须要有一定的控制手段以保障事务并发执行中一事务执行时不受它事务的影响, 目前我们一般采用封锁 (locking) 的办法以解决此类问题的产生。

	T1	T2		T1	T2		T1	T2
1	Read:y=5		1	Read:A=60		1	Read:c=100	
2		Read:y=5		Read:B=100			$C \leftarrow C * 2$	
3	$y \leftarrow y - 1$			A+B=160			Write:C=200	
	Write:y=4		2		Read:B=100	2		Read:c=200
4		$y \leftarrow y - 1$			$B \leftarrow B * 2$	3	ROLLBACK	
		Write:y=4			Write:B=200		C 恢复为 100	

图 5.5 丢失修改

图 5.6 不可重复读

图 5.7 脏读

5.2.2 封锁

封锁是事务并发执行的一种调度和控制手段, 它可以保证并发执行的事务间相互隔离、互不干扰, 从而保证并发事务的正确执行。

所谓封锁就是事务对某些数据对象的操作实行某种专有的控制, 在一事务 T 需要对某些数据对象操作 (读/写) 时, 它必须向系统提出申请, 对其加锁, 在获得加锁成功后, 即具有对此类数据的一定操作权限与控制权限。此时, 其他事务不能对加锁的数据随意操作, 当事务 T 操作完成后即释放锁, 此后该数据即可为其他事务操作服务。

目前一般常用的有两种锁: 排它锁和共享锁。

(1) 排它锁 (exclusive lock) 又称写锁, 或称 X 锁。一事务 T 对数据 A 加 X 锁后, T 可以对 A 作读、写, 而其他事务对 A 不能作任何操作 (包括读、写)。排它锁保证了事务对数据的独占性, 排除了其他事务对它的干扰。

(2) 共享锁 (sharing lock) 又称读锁, 或称 S 锁, 一事务 T 对数据 A 加 S 锁后, T

可以读 A 但不能写 A，而其他事务可以对 A 加 S 锁但不能加 X 锁。共享锁保证了多个事务都可以读 A，但它们都不能在 T 释放 A 上的 S 锁前写 A。

排它锁与共享锁的控制方式可用图 5.8 表示。

$T_1 \backslash T_2$	X 锁	S 锁	—
X 锁	N	N	Y
S 锁	N	Y	Y
—	Y	Y	Y

图 5.8 封锁的相容矩阵

(Y 表示相容, N 表示不相容)

一事务在作读、写操作前必须申请加锁 (X 锁或 S 锁)。如此时 A 正被他事务加锁, 如果 A 加锁不成功, 则必须等待, 直至他事务将锁释放后, 才能获得加锁成功并执行操作, 在操作完成后 A 必须释放锁, 此种事务称为合适 (well formed) 事务。合适事务是为保证正确的并发执行所必须的基本条件。

5.2.3 封锁协议

利用封锁的办法可以使并发执行的事务正确执行, 但是这仅是一个原则性方法, 真正做到正确执行, 还需要有多种具体的考虑, 包括:

- (1) 事务申请何种锁 (X 锁/S 锁)。
- (2) 事务持锁时间。
- (3) 事务何时释放锁。

对不同封锁方式考虑可组成不同封锁规则, 称为封锁协议 (locking protocol), 不同的协议可以防止不同的并发错误。

目前一般将封锁协议分为三级。

1. 一级封锁协议

事务 T 在对数据 A 作写前, 必须对 A 加 X 锁, 直到事务结束 (包括 COMMIT 与 ROLLBACK) 才释放加在 A 上的 X 锁。

2. 二级封锁协议

事务 T 在对数据 A 作读前必须先对 A 加 S 锁, 在读完后即释放加在 A 上的 S 锁。此种封锁方式与一级封锁协议联合构成了二级封锁协议。

3. 三级封锁协议

事务 T 在对数据 A 作读前必须先对 A 作 S 锁, 直到事务结束才能释放加在 A 上的 S 锁。此种封锁方式与一级封锁协议联合构成了三级封锁协议。

上述三种封锁协议可以分别防止不同的并发错误。

首先, 一级封锁协议可以防止丢失修改, 这主要是采用一级封锁协议后, 事务在对数据 A 作写操作时必须申请 X 锁, 以保证其他事务对 A 不能作任何操作, 直至事务结束, 此 X 锁才能释放。以图 5.5 为例, 对它作一级封锁后即可防止丢失修改, 可用图 5.9 表示。

其次，二级封锁协议包含了一级封锁协议内容，因此可以防止丢失修改，同时它还可以防止脏读。这主要是在事务对数据 A 作读、写时用 X 锁，作读时用 S 锁，从而防止了脏读。以图 5.6 为例，对它作二级封锁，即可防止脏读，它可用图 5.10 表示。

	T1	T2
1	Xlock y	
2	Read: y=5	
3	y←y-1 Write: y=4 Commit Ulock y	Xlock y Wait Wait Wait
4		获得 Xlock y Read: y=4 y←y-1 Write: y=3 Commit Ulock y

图 5.9 使用一级封锁协议后防止丢失修改

	T1	T2
1	Xlock C Read: C=100 C←C*2 Write: C=200	
2		Slock C
3	ROLLBACK (C 恢复为 100) Unlock C	Wait Wait Wait
4		获得 Slock C Read: C=100 Commit Unlock C

图 5.10 使用二级封锁协议后可防止脏读

第三，三级封锁协议除可以防止丢失修改、脏读外，还可防止不可重复读。这主要是在对数据 A 作读、写时申请 X 锁，直到事务结束后才释放。同样，对 A 作读时申请 S 锁，也是事务结束后才释放，这样可以防止不可重复读。以图 5.7 为例，对它作三级封锁，即可防止不可重复读，它可用图 5.11 表示。

5.2.4 两阶段封锁协议

由前文所述可知，三级封锁协议可以保证事务并发执行的正确性。但是，按三级封锁协议规定，所有在事务中申请的锁必须在事务结束后才能释放。这就意味着，在一个事务

执行中必须把锁的申请与释放分为两个阶段，其中第一个阶段是申请并获得封锁。在此阶段中，事务可以申请其整个执行过程中所需操作数据的锁，此阶段也可称为扩展阶段。第二阶段是释放所有原申请并获得的锁，此阶段也可称为收缩阶段。此种设置封锁的方法称为两阶段封锁协议（two-phase locking protocol）。

	T1	T2
1	Slock A Slock B Read: A=60 Read: B=100 A+B=160	
2		Xlock B
3	Read: A=60 Read: B=100 A+B=160 Commit Unlock A	Wait Wait Wait Wait Wait
4	Unlock B	获得 Xlock B Read: B=100 B ← B*2 Write: B=200 Commit Unlock B

图 5.11 使用三级封锁协议后可防止不可重复读

图 5.12 所示的是遵守两阶段封锁协议的封锁序列，而图 5.13 所示的是不遵守两个阶段封锁协议的封锁序列。

T: Slock A Slock B ...Slock C Unlock B Unlock A Unlock C
 |← 扩展阶段 →| |← 收缩阶段 →|

图 5.12 遵守两个阶段封锁协议的封锁序列

T: Slock A Unlock A Slock B Xlock C Unlock C Unlock B

图 5.13 不遵从两阶段封锁协议的封锁序列

在并发执行中，当一个事务遵守两阶段协议进行封锁时，那它一定能正确执行。也就是说，此时事务并发执行与事务串行执行具有相同效果，即事务在并发处理中如按两阶段封锁协议执行，此时是可串行化事务。

5.2.5 封锁粒度

封锁粒度(grenularity)即是事务封锁的数据目标的大小，在关系数据库中封锁粒度一般有如下几种。

- 属性（值）及属性（值）集
- 元组
- 表
- 物理页面（或物理块）
- 索引
- 关系数据库

从上面七种不同粒度中可以看出，事务封锁粒度有大有小，一般而言，封锁粒度小则并发性高但开销大，封锁粒度大则并发性低但开销小，综合平衡照顾不同需求以合理选取封锁粒度是很重要的，常用的封锁粒度有表和属性。

5.2.6 活锁与死锁

采用封锁的方法可以有效地解决事务并发执行中的错误出现，保证并发事务的可串行化。但是封锁本身带来了一些麻烦，最主要的就是由封锁引起的活锁(live lock)与死锁(dead lock)。所谓活锁即是某些事务永远处于等待状态，得不到解锁机会；而所谓死锁即是事务间对锁的循环等待。亦即是说，多个事务申请不同锁，而申请者均拥有一部分锁，而它又在等待另外事务所拥有的锁，这样相互等待，而造成它们都无法继续执行。一个典型的死锁例子如图 5.15 所示。在例中事务 T1 占有 X 锁 A，而申请 X 锁 B，事务 T2 占有 X 锁 B 而申请 X 锁 A，这样就出现了无休止的相互等待的局面。

	T1	T2
1	Xlock A	
2		Xlock B
3	Read: A	
4		Read: B
5	Xlock B	
6	Wait	
7	Wait	Xlock A
8	Wait	Wait
9	Wait	Wait

图 5.12 死锁实例

活锁和死锁都有办法可以解决。解决活锁的最有效的办法是采用“先来先执行”的控制策略，也就是采用简单的排队方式，而解决死锁的办法目前常用的如下几种：

1. 预防法

预防法即是预先采用一定的操作模式以避免死锁的出现。

(1) 顺序申请法。即将封锁的对象按顺序编号，事务在申请封锁时按编号顺序（从小到大或反之）申请，这样能避免死锁发生。

(2) 一次申请法。事务在执行开始时将它需要的所有锁一次申请完成，并在操作完成后一次性归还所有的锁。

2. 死锁解除法

死锁解除法允许产生死锁，在产生后通过一定手段予以解决，常用的方法如下：

(1) 定时法。对每个锁设置一个时限，当事务等待此锁超过时限后即认为已产生死锁，此时调用解锁程序，以解除死锁。

(2) 死锁检测法。在系统内设置一个死锁检测程序，该程序定时启动检查系统中是否产生死锁，一旦发现产生死锁则调用解锁程序以解除死锁。

5.3 数据库恢复技术

5.3.1 概述

尽管对数据库采取多种严格的防护措施，但是数据库遭受破坏仍是不可避免的，因此，一个数据库管理系统除了要有较好的完整性、安全性保护措施以及并发控制能力外，还需

要有数据库恢复的能力。数据库恢复技术是一种被动的方法，而数据库完整性、安全性保护及并发控制技术则是主动的保护方法，这两种方法的有机结合可以使数据库得到有效的保护。

数据库故障恢复技术所采用的主要手段是冗余与事务。所谓数据冗余即是采取数据备用副本和日志，所谓事务即是利用事务作为操作单位进行恢复。

5.3.2 数据库故障分类

为讨论数据库恢复,我们首先须对数据库故障作分类,数据库故障大致可以分为小型故障、中型故障与大型故障等三种类型共六个部分。

1. 小型故障

- **事务内部故障：**此类故障是事务内部执行对所产生的逻辑错误与系统错误,如数据输入错误、数据溢出、资源不足（以上属逻辑错误）以及死锁、事务执行失败（以上属系统错误）等，此类故障属小型故障。

所谓小型故障即是其故障影响范围在一个事务之内。

2. 中型故障

- **系统故障：**此类故障是由于系统硬件（如 CPU）故障、操作系统、DBMS 以及应用程序代码错误所造成的故障，此类故障可以造成整个系统停止工作，内存破坏，正在工作的事务全部非正常中止，但是磁盘数据不受影响，数据库不遭破坏，此类故障属中型故障。

- **外部影响：**此类故障主要是由于外部原因（如停电等）所引起的，它也造成系统停止工作，内存破坏，正在工作的事务全部非正常中止，但数据库不受破坏，此类故障属中型故障。

中型故障的影响范围是事务级的，即某些事务要重做，某些事务要撤销，但是它不需要对整个数据库作全面修复。

3. 大型故障

- **磁盘故障：**此类故障包括磁盘表面受损,磁头损坏等,此时整个磁盘受到破坏,数据库严重受影响。

- **计算机病毒：**计算机病毒是目前破坏数据库系统的主要根源之一，它不但对计算机主机产生破坏（包括内存）也对磁盘文件产生破坏。

- **黑客入侵：**黑客入侵可以造成主机、内存及磁盘数据的严重破坏。

以上三种故障造成内存、磁盘及主机严重破坏，同时也使整个数据库受到破坏，因此此类故障属系统及故障。

5.3.3 数据库故障恢复三大技术

为恢复数据库中的数据一般采用下面三大技术。

1. 数据转储

所谓数据转储即是定期将数据库中的内容复制到另一个存储设备中去，这些存储的拷贝称为后援副本或备份。

转储可分为静态转储与动态转储。静态转储指的是转储过程中不允许对数据库有任何操作（包括存取与修改操作），即转储事务与应用事务不可并发执行。动态转储指的是转储过程中允许对数据库作操作，即转储事务与应用事务可并发执行。

静态转储执行比较简单，但转储事务必须等到应用事务全部结束后才能进行，因此带

来一些麻烦。动态转储可随时进行，但是转储事务与应用事务并发执行，容易带来动态过程中的数据不一致性，因此技术上要求较高。

数据转储还可以分为海量转储与增量转储，海量转储指的是每次转储数据库的全部数据，而增量转储则是每次只转储数据库中自上次转储以来所产生变化的那些数据。由于海量转储数据量大，不易进行，因此增量转储往往是一种有效的办法。

2. 日誌 (logging)

所谓日誌即是系统建立的一个文件，该文件用于系统记录数据库中修改型操作的数据更改情况，其内容有：

- (1) 事务开始 (SET TRANSACTION) 标志。
- (2) 事务结束 (COMMIT 或 ROLLBACK) 标志。
- (3) 事务的所有更新操作。

具体的内容有：事务标志、操作时间、操作类型 (增、删、或改操作)、操作目标数据、更改前数据旧值、更改后数据新值。

日誌以事务为单位按执行的时间次序，且遵循先写日誌后修改数据库的原则进行。

3. 事务撤销与重做

数据库故障恢复的基本单位是事务，因此在数据恢复时主要使用事务撤销 (UNDO) 与事务重做 (REDO) 两种操作。

• 事务撤销操作

在一事务执行中产生故障，为进行恢复，首先必须撤销该事务，使事务恢复到开始处，其具体过程如下：

- (1) 反向扫描日誌文件，查找应该撤销的事务。
- (2) 找到该事务更新的操作。
- (3) 对更新操作做逆操作，即如是插入操作则做删除操作，如是删除操作则用更改前数据旧值作插入，如是修改操作则用修改前值替代修改后值。
- (4) 如此反向扫描一直反复做更新操作的逆操作，直到事务开始标志出现为止，此时事务撤销结束。

• 事务重做操作

当一事务已执行完成，它的更改数据也已写入数据库，但是由于数据库遭受破坏，为恢复数据需要重做，所谓事务重做实际上是仅对其更改操作重做，重做的过程如下。

- (1) 正向扫描日誌文件,查找重做事务。
- (2) 找到该查找事务的更新操作。
- (3) 对更新操作重做，如是插入操作则将更改后新值插入至数据库，如是删除操作，则将更改前旧值删除，如是修改则将更改前旧值修改成更新后前值。
- (4) 如此正向扫描反复做更新操作，直到事务结束标志出现为止，此时事务重做操作结束。

5.3.4 恢复策略

利用后备副本、日誌以及事务的 UNDO 与 REDO 可以对不同的数据库进行恢复，其具体恢复策略如下。

1. 小型故障的恢复

小型故障属事务内部故障，其恢复方法是利用事务的 UNDO 操作，将事务在非正常中止时利用 UNDO 恢复到事务起点。

2. 中型故障的恢复

中型故障所需要恢复的事务有两种：

(1) 事务非正常中止。

(2) 已完成提交的事务，但其更新操作还留在内存缓冲区尚未来得及写入，由于故障使内存缓冲区数据丢失。

对第一种事务采用 UNDO 操作，使其恢复至事务起点，对第二种事务用 REDO 操作进行重做。

3. 大型故障的恢复

大型故障是那些整个磁盘、内存及系统都遭受破坏的故障，因此对它的恢复就较为复杂，它大致分为下列步骤：

(1) 将后备复本拷贝至磁盘。

(2) 做事务恢复第一步——检查日志文件，将拷贝后所有执行完成的事务做 REDO。

(3) 做事务恢复第二步——检查日志文件，将未执行完成（即事务非正常中止）的事务做 UNDO。

经过这三步处理后可以较好完成数据库中数据的恢复。数据库中恢复一般由 DBA 执行。数据库恢复功能是数据库的重要功能，每个数据库管理系统都有此种功能。

习题五

- 5.1 什么叫事务？它有哪些性质？试说明。
- 5.2 什么叫并发事务可串行技术？它能否保证并发事务的正确执行？试说明。
- 5.3 试述三级封锁协议以及如何防止并发错误的发生。
- 5.4 试叙述 UNDO、REDO 操作以及它们在故障恢复中的作用。
- 5.5 什么叫日志？它在故障恢复中起什么作用？试说明。
- 5.6 试解释事务、并发控制及故障恢复间的关系。
- 5.7 试给出故障分类以及这些类中如何进行恢复的方法。
- 5.8 什么叫数据转储？如何实现转储？以及转储在恢复中的作用？请说明之。
- 5.9 设有数据库系统，每晚 10 时作拷贝，它在某日 13 时发生大型故障，请给出其恢复策略。
- 5.10 请给出一个并发执行但未加控制而造成错误的例子，并作分析说明并发控制之必要性。

第五章复习指导

本章介绍以事务为核心的三种技术，它们是事务处理、并发控制及故障恢复技术。读者在学习本章后对上述三种技术有一个基本的了解。

1. 事务处理

- (1) 事务的概念
- (2) 事务的四大特性 ACID
- (3) 事务的活动：事务开始——事务执行——事务结束（分正常结束：提交及非正常结束：回滚）
- (4) 有关事务的三个语句：
 - SET TRANSACTION
 - COMMIT
 - ROLLBACK

2. 并发控制技术

(1) 并发控制的概念
为提高效率，事务需并发执行，并发执行会发生错误，因此必需对并发执行的事务作一定的控制，此种技术称为并发执行技术，它既可并发执行以提高效率又可保证执行的正确性。

- (2) 常见的三种并发执行错误
 - 丢失修改
 - 不可重复读
 - 脏读
- (3) 并发控制的主要手段——封锁技术
 - 封锁的基本概念
 - 三级封锁与二阶段封锁协议
 - 死锁与活锁

3. 数据库恢复技术

- (1) 数据库的三种故障：小型、中型及大型故障
- (2) 数据库恢复的三大技术
 - 数据转储
 - 日志
 - 事务撤消与重做

4. 事务、并发控制与故障恢复间的关系

- (1) 事务与并发控制关系——并发控制以事务为单位
- (2) 事务与故障恢复关系——故障恢复以事务为单位
- (3) 事务、并发控制与故障恢复间的关系

5. 本章的重点内容

- 事务基本概念
- 封锁技术
- 三大恢复技术

第六章 数据库中的数据交换

数据库中的数据交换（data exchange）是数据库与其使用者间的数据交互过程，而数据交换是需要管理的，数据交换的管理是对数据交换的方式、操作流程及操作规范的控制与监督。数据库中的数据交换是数据库开拓应用的基本前提与保证，它对数据库应用的开发极为重要。本章主要介绍数据库中数据交换的基本原理、数据交换的管理以及目前常用的四种交换方式。

6.1 概述

数据库是一种共享机构，它为用户使用提供了基本共享保证，但是如何适应使用环境、扩大使用范围、保证使用方便，这是扩大数据库使用所要解决的问题，这就是所谓数据库中的数据交换（下面简称数据交换）。在这章中我们主要讨论 SQL 中的数据交换模型、数据交换管理以及四种数据交换方式。考虑到数据交换与应用环境、应用需求紧密相关，因此数据交换虽然具有其共同的规律与操作流程，但是要实现标准化难度较大，因此在本章中我们主要介绍数据交换的一般性规则与方法，而并不拘泥于标准的使用束缚，当然我们也会介绍一些重要的标准（以 SQL'99 为主）以及它们与目前流行方法的区别。

6.1.1 数据交换模型

数据交换是数据主体与数据客体间数据的交互过程。所谓数据客体即是数据库，它是数据提供者，而数据主体是数据的使用者，也是数据接收者，它可以是操作员（人）、应用程序，也可以是另一种数据体。具体说来，即是首先由使用者通过 SQL 语言向数据库提出数据请求，接下来数据库响应此项请求进行数据操纵并返回执行结果，执行结果有两项：返回的数据值以及执行结果代码，（它给出了执行结果正确与否，出错信息以及其它辅助性质）它可用下面图 6.1 所示的数据交换模型表示之。

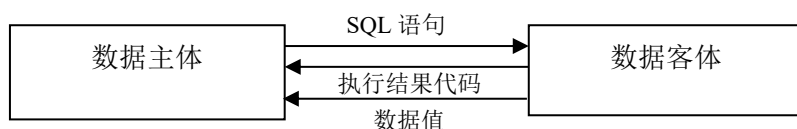


图 6.1 数据交换模型图

6.1.2 数据交换的五种方式

从数据库诞生起即有数据交换存在，但由于其交换方式与交换管理都很简单，因此并未出现有数据交换的概念。真正出现数据交换并对其作规范化管理的是 SQL'92 而在 SQL'99 中则对其作了进一步规范，并明确划分了数据交换的四种方式。在最近公布的 SQL'03 中将原有四种交换方式扩充到八种，并将数据交换管理明确规定为五种，此外在 20 余年来众多机构与相关单位也纷纷推出多种数据交换的规范与产品，有的已成为业内的事实标准。

从发展历史看，数据交换已随着应用环境变化与应用范围变化而经历了三个阶段，而每个阶段已有着一种或多种方式，因此出现了三个阶段的六种方式的多种局面，下面对其

作简单介绍。

1. 初级阶段与人机对话方式

在数据库发展的初期，数据库主体仅为数据库操作员（即为人），因此数据交换仅是一种简单的人机交互关系，因而称为人机对话方式或称交互方式。在此种方式下 SQL 具有即席（adhoc）交互特点。

在此种阶段中应用环境是一种单机、集中式工作环境，管理简单，这反映了当时数据库应用范围与规模的简单性。

2. 中级阶段

随着数据库应用的规模与范围扩大，数据库环境的发展数据进入了新的阶段，此阶段称为中级阶段，其主要特点是数据主体已由人变成为应用程序。由于数据库管理系统自身的局限性，要开发一个完整的应用系统，它还需要与程序设计语言配合。在此阶段中数据交换是一种应用程序与数据库间的数据交互过程，而此时的数据交换管理则形成为两种不同类型语言（算法程序设计语言与数据库语言）间的数据交换。

在此阶段中按不同应用环境可分为三种不同交换方式，它们是：

（1）嵌入式方式

在中级阶段的初期，数据应用发展迅猛而数据库自身又缺乏有效的程序语言支持而显得力不从心，因此迫切需要寻找一些应用开发语言，将两者紧密结合以构成一种有效的应用开发的工具。最为立竿见影的办法是将数据库语言直接嵌入至某些常用的算法程序设计语言中，两者直接捆绑而快速形成开发力量，这就是数据交换的嵌入式方式。在此种方式中算法程序设计语言称为主语言（host language）而嵌入主语言的 SQL 则称为子语言（sub language）。这种状态反映了数据库发展初期在应用系统开发中的幼稚与从属的地位。

嵌入式方式出现于单机、集中式环境，在网络时代它存在于数据服务器中，而由于嵌入方式中两种语言的不一致性而引起的阻抗不匹配，因此要将这两者捆绑在一起需要大量的交换管理工作，同时捆绑还会引起大量的辅助开销，因此出现编程难度大、执行效率低的情况，这也反映了这种方式是一种不成熟、临时、凑合的方式。

嵌入式方式的 SQL 是出现最早的应用程序与数据库间的数据交换方式，在 SQL 出现不久即有此种方式存在，在 SQL'89 中即列入其中，而与其捆绑的语言也由原先的三种至 SQL'03 已增至八种，它们是：C, PASCAL, FORTRAN, COBOL, ADA, PL/1, MUMPS, JAVA 等，此种方式在 SQL 标准中称为 SQL/BD。嵌入式数据交换方式由于其所存在多种不足，目前使用者已极为寥寥，但它在数据交换历史上则发挥过重要作用，而由它所开创的数据交换管理技术也为此后的多种数据交换方式提供了基础。

（2）自含方式

随着数据库管理系统的成熟以及数据库厂商势力的增强，从而出现了由数据库管理系统自身包含程序设计语言的主要语句成份，因而将 SQL 与程序设计语言统一于 DBMS 之内，这就称为自含（contains self）方式。

自含方式的出现改变了嵌入方式的诸多不便，使用也极为方便，自此以后自含方式已逐步取代嵌入方式。

自含方式出现于单机、集中式时代，在网络环境中，它存在于数据服务器中。在目前商用数据库产品中，自含方式的 SQL 有 ORACLE V.7.0 中的 PL/SQL, Sybase Adaptive Server

中的 T-SQL 以及微软的 SQL Server2000 中的 T-SQL。在 SQL 标准中自 SQL'92 起就有此类方式出现，称 SQL / PSM。

(3) 调用层接口 (call level interface) 方式

自数据库应用进入网络时代后，数据库结构出现了 C/S 结构模型，在此模型中数据库应用系统功能分布为两个部分，其中应用程序分布于客户端 C，而数据库则分布服务器端 S。在此模型中，应用程序与数据库间的数据交换变成为客户端应用程序。通过调用函数方式以实现从服务器调用数据的数据交换方式，其具体方法是对网络中不同数据源设置一组统一的数据交换函数以实现数据交换，而客户端对数据库的数据请求的 SQL 语句，以某些函数的参数出现，连同函数本身一起传递至服务器执行。

此种方法目前也可以应用于 B/S 结构模型中，由于目前数据库应用环境多采用 C/S 方式与 B/S 方式，因此调用层接口方式已被广泛采用。在 SQL 标准中 SQL'97 中开始出现调用接口层的接口方式 SQL/CLI，在企业中也出现有微软的 ODBC 标准与 SUN 公司的 JDBC 标准，而根据后两者所开发的产品则由于使用广泛，目前已成为事实上的标准。

3. 近期阶段与 Web 方式

在近期，互联网的出现以及 Web 的发展使得数据交换方式又出现了新的形式，其主要的特点是 XML 与传统数据库间的数据交换方式。由于传统数据库是一种严格的格式化数据，而 XML 则是一种松散的半格式化数据，由于两者数据结构形式的严重差异，因此须进行数据交换，而此种方式称 Web 数据交换方式，它在 Web 环境下应用广泛。

在 SQL 标准的 SQL'03 中出现有此种方式称 SQL/XML，此外，微软与 SUN 公司中也有此类方式的产品出现，如 ASP、JSP 等。

上面所介绍的五种方式反映了数据库应用发展过程中不同阶段、不同环境的数据交换需求，而目前最为常用的是：

- 在服务器中的自含方式。
- 在 C/S 结构中的调用接口层方式。
- 在 Web 环境中的 Web 方式。

6.2 数据交换的管理

数据交换是需要管理的，特别是在应用环境日益复杂的今天，数据交换管理尤为重要。数据交换管理一般由下面几部分内容所组成：

- 会话管理
- 连接管理
- 游标管理
- 诊断管理
- 动态 SQL

下面我们分别介绍之。

6.2.1 会话管理

数据交换是两个数据体之间的会话过程，而会话是在一定环境下进行的，在当今复杂的应用中，会话的进行须预先作环境的设定，这就是会话管理，会话管理须设定的环境参

数如下：

- 会话的数据客体模式设定；
- 会话的语言模式设定；
- 会话的时间模式设定；
- 会话的标识符设定。

下面对其作介绍。

1. 设置会话的数据模式

在会话开始前首先须明确数据主体与之会话的数据客体模式，在网络环境中可有多个数据客体模式，目前大致可以分为三个层次，它们是网络环境，目录层与模式层，其结构可见图 6.2。

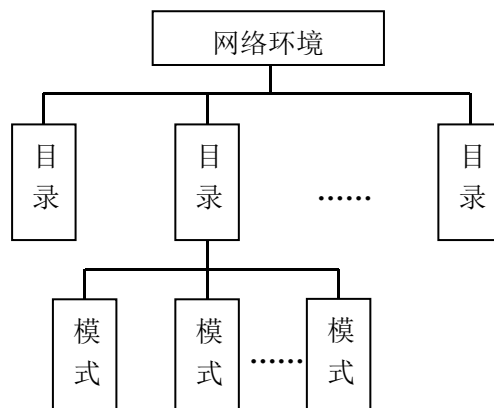


图 6.2 数据模式层次图

在网络环境中，需选定与之会话的目录以及目录下的模式，它们可用下面的语句设置：

- 设置目录语句：
SET CATALOG <目录名>
- 设置模式语句：
SET SCHEMA <模式名>

2. 设置会话的字符集

在现代数据交换中涉及到所使用的多种语言文字，包括英语、汉语、日语、少数民族语言等多种语言文字，它们以不同字符集形式表示，因此须设置会话字符集如下：

- 设置名语句：
SET NAMES <字符集名>

3. 设置会话局部时区

在互联网环境中，两个进行数据交换的数据体间存在着时区差异的可能性，因此需要对两者时区差异作设置，其设置语句如下：

- 设置局部时区语句：
SET TIME ZONE <设置时区值>

4. 设置会话的授权标识符

对设置了上述参数的一个固定会话环境须给出一个名，称为会话授权标识符，它可用下面的语句设置如下：

- 设置会话授权标识符语句:

SET SESSION AUTHORIZATION <标识符>

在会话管理中常用的设置为:

- 会话授权标识符设置
- 模式设置

6.2.2 连接管理

连接管理负责数据主、客体间建立实质性的关联,包括服务器指定、内存区域分配等,同时也可以断开两者间的实质性关联,它一般可用两条语句实现。

- 连接语句

```
CONNECT TO <连接目标>
<连接目标>::=<服务器名>
[AS<连接名>][USER<用户名>]
```

连接语句有时还可用包括:数据模式名、表名以及使用 SQL 语句名以及相应的内存分配。

- 断开连接语句

```
DISCONNECT <断开对象>
<断开对象>::=<连接名>|ALL|CURRENT
```

其中<连接名>表示指定的连接, ALL 表所有连接而 CURRENT 则表示当前的连接。

6.2.3 游标管理

在数据交换中,数据库 SQL 中的变量是集合型的而应用程序的程序设计语言中的变量则是标量型,因此数据库中 SQL 变量不能直接供程序设计语言使用,而需要有一种机制将 SQL 变量中的集合量逐个取出后送入应用程序变量内供其使用,而提供此种机制方法是增加游标(cursor)语句。

游标语句一共有四条:

- (1) 定义游标。为某 SELECT 语句的结果集合定义一个命名游标,其形式为:

```
DECLARE <游标名>CURSOR FOR <SELECT 语句>
```

- (2) 打开游标。在游标定义后当使用数据时需打开游标,此时游标处于活动状态并指向集合的第一个记录,打开游标语句形式为:

```
OPEN<游标名>
```

- (3) 推进游标。此语句功能是将游标定位于集合中指定的记录,并从该记录取值,送入程序变量中。

```
FETCH<定位取向>FROM <游标名>INTO<程序变量列表>
```

```
<定位取向>:::=NEXT|PRIOR|FIRST|LAST|±n|ABSOLUTE ±n|RELATIVE ±n
```

在此语句的定位取向中给出了游标移动方位:

- 从当前位置向前推进一行: NEXT
- 从当前位置向后推进一行: PRIOR
- 推向游标第一行: FIRST
- 推向游标最后一行: LAST
- 从当前位置向后推进 n 行: ABSOLUTE-n

-
- 从当前位置向前推进 n 行: **ABSOLUTE +n**
 - 推向游标第 n 行: **RELATIVE+n**
 - 推向游标倒数第 n 行: **RELATIVE-n**
- (4) 关闭游标。游标使用完后需关闭, 其语句形式为:
- CLOSE <游标名>**

6.2.4 诊断管理

在进行数据交换时数据主体发出数据交换请求后, 数据客体返回两种信息, 一种返回所请求的数据值, 另一种是返回执行的状态值, 而这种状态值称为诊断值, 而生成、获取诊断值的管理称诊断管理。

诊断管理由两部分组成, 它们是诊断区域及诊断操作。

1. 诊断区域

诊断区域是存放诊断值的内存区域, 它包括执行完成信息以及异常条件信息。诊断区域由两部分组成, 它们是标题字段与状态字段。其中标题字段给出诊断的类型(如 **NUMBER** 表执行结果的数值表示), 而状态字段则给出该诊断类型执行结果的编码, 它们是: 语句执行成功为 0, 不成功给出错误代码。

2. 诊断操作

诊断操作有两种:

- (1) **DBMS** 在执行 **SQL** 语句后将执行状态自动存放于诊断区域内。
- (2) 使用者用获取诊断语句以取得语句执行的状态, 获取诊断语句的形式如下:

GET DIAGNOSTICS<标题名>=<变量名>

该语句的执行结果是将诊断区域指定标题的状态信息取出。

6.2.5 动态 SQL

在数据交换时, 有时数据主体所发出的 **SQL** 语句请求常常不能预先确定, 而需根据情况在程序运行时动态指定, 这就是动态 **SQL**。动态 **SQL** 在目前实现中分为不同类型。

1. **SQL** 语句正文动态可变

允许在运行时临时输入完整 **SQL** 语句。

2. 变量个数动态可变

在 **SQL** 语句的子句中允许变量个数在运行时动态增/减。

3. 类型动态可变

SQL 语句中的变量类型可在运行时动态调整。

4. **SQL** 语句引用对象动态可变

SQL 语句 **SELECT** 及 **GROUPBY** 中的列名, **From** 子句中的表名, **WHERE** 及 **HAVING** 子句中的条件在运行时动态调整。

为实现动态 **SQL**, 应用程序与数据库需进行信息交互, 即应用向数据库提供动态参数, 而数据库则需向应用提供查询结果, 因此需在内存开辟一个区域供交互之用, 该区域称为描述符区 (**descriptor area**), 而描述数据的数据称描述符 (**descriptor**), 描述符共有两种, 它们是:

1. 参数描述符: 用于应用程序向数据库提供动态参数;
2. 行描述符: 用于数据库向应用程序提供查询结果, 这些结果以表中行为单位, 因

此称行描述符。

动态 SQL 的操作一般有下面两种语句：

1. 有关描述符区的操作语句：

- 分配描述符语句：ALLOCATION DESCRIPTOR——分配一个 SQL 描述符区。
- 解除分配描述符语句：DEALLOCATION DESCRIPTOR——解除所分配的 SQL 描述符区。
- 设置描述符语句：SET DESCRIPTOR——在 SQL 的描述符区设置信息。
- 取描述符语句：GET DESCRIPTOR——从 SQL 的描述符区取得信息。

2. 有关动态 SQL 使用语句：

- 准备语句：PREPARE<动态 SQL 语句名>FROM<动态 SQL 语句>，动态 SQL 在使用前必须有一个准备阶段，可用此语句作准备，语句中<动态 SQL 语句>可以用一个变量定义，也可以是 SQL 字符串。

- 执行语句：EXECUTE<动态 SQL 语句名>，此语句表示执行动态 SQL 语句。

- 立即执行语句：EXECUTE IMMEDIATE<动态 SQL 语句>，此语句表示动态准备并立即执行一个动态 SQL 语句。当 PREPARE 语句所组成的动态 SQL 语句只需执行一次时，此时可将 PREPARE 与 EXECUTE 合并成此语句执行。

对执行语句尚可作一些补充，即当 PREPARE 语句所组成的动态 SQL 语句的条件值尚缺时，可在执行语句中增设 USING 子句补上，此时 EXECUTE 语句是：

EXECUTE<动态 SQL 语句名>USING<变量>

此外还有关于游标及增、删、改等的动态 SQL 语句。

6.3 数据交换的流程

数据交换是一个按一定步骤进行的过程，利用数据交换管理可以实现数据交换过程，其全部流程如下：

1. 数据交换准备

应用会话管理设置数据交换的各项环境参数，包括设置数据库的数据模式，设置会话授权标识符以及设置字符集与局部时区。

会话环境设置是面向固定应用的，它一经设定后一般不会改变，因此它是某个应用的数据交换前提。

2. 数据连接

在设置环境参数后接下来的重要步骤是建立两个数据体间的物理连接，包括连接通路的建立，内存区域的分配等，数据连接一般建立在两个数据体处于网络不同结点的情况下。

3. 数据交换

在经过数据连接后数据交换即可进行，在数据交换中主要是数据主体应用 SQL 语句以获取数据库中的数据，在取得后放入指定区域，同时返回执行的状态信息，此时最关键的是需要不断使用游标语句与诊断语句，并在必要时使用动态 SQL。

4. 断开连接

在数据交换结束后即可以断开两个数据体间的连接，包括断开连接的通路以及取回所

分配的内存区域。

在一个数据交换结束后由会话管理所设置的以会话授权标识符为代表的各项环境参数仍可保留，此后可进入下一轮数据交换，（即 2、3、4 三个阶段）如此不断循环而构成数据交换的完整过程。下面的图 6.3 给出数据交换的流程图。

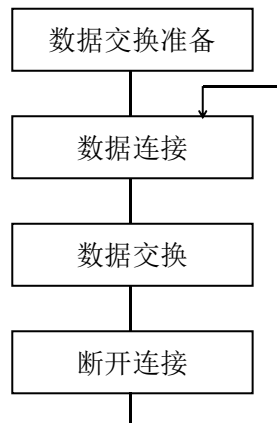


图 6.3 数据交换过程的流程图

在数据交换的五种方式中基本上均按上述流程图进行数据交换，但是其流程的严格性有所不同，如人机对话方式则较为简单，而调用层接口及 Web 方式则较为严格。

6.4 数据交换的四种方式

数据交换的五种方式中由于人机对话方式较为简单，因此在本节中我们不作介绍，下面我们较为详细的介绍其余的四种方式。

6.4.1 嵌入式 SQL

6.4.1.1 嵌入式 SQL 的特点

SQL 语言在其应用的初始阶段是联机终端用户在交互式环境下使用的，称为交互式 SQL (interactive SQL)，简称 ISQL。但是作为完整开发应用系统，SQL 尚须得到某些算法程序设计语言的支撑，因此 SQL 还可以作为一种数据子语言嵌入于某些主语言中，从而构成一种完整的应用开发工具。SQL 可以嵌入于 COBOL, FORTRAN, C 等语言中使用，称为嵌入式 SQL (embedded SQL)，简称 ESQL。

在此种方式下，一个完整的应用程序是由主语言语句与 SQL 语句两部分构成，由于这两者使用方式的差异，因此存在着操作上的不同。下面，探讨其差异与关联，主要有四个方面：

- (1) 在嵌入方式下，程序中既有主语言的语句又有 SQL 语句，如何区别之？
- (2) 在嵌入方式下，程序中主语言程序与 ESQL 间如何通讯？
- (3) 在嵌入方式下，程序中如何区分主语言变量与 SQL 的列变量？
- (4) 在嵌入方式下，主语言变量一般均为标量，而 SQL 中的列变量一般均为集合量，如何建立此两种变量间的交互（特别是由集合量到标量的转换）？

下面我们讨论在嵌入方式下这四个问题的解决方法。

1. 主语言语句与 ESQL 语句的区别

在嵌入方式下,SQL 语句在嵌入主语言的程序时其前缀加 EXEC SQL,其结束处用 END—EXEC 或用分号 (;), 结束标志因主语言的不同而不同。

2. 主语言程序与 ESQL 程序间的通讯

在程序中所使用的 SQL 中的表,包括基表与视图都要用 EXEC SQL DECLARE 语句加以说明。SQL 语句执行后, 需要进行诊断, 其诊断值送入 SQL 诊断区。

3. 主语言变量与 SQL 变量的区别

在嵌入方式下, 程序中的 SQL 语句段内可使用主语言的变量(称为主变量), 在这些变量前加冒号(:)以示区别, 在主程序中, SQL 变量一般不出现。

所有 SQL 语句中用到的主变量及指示变量都必须加以说明, 说明的开头行与结尾行分别为:

```
EXEC SQL BEGIN DECLARE SECTION
EXEC SQL END DECLARE SECTION
```

4. 游标语句的使用

在嵌入方式下, SQL 的变量是集合型的, 而主变量则是标量型的, 因此需要使用游标语句, 其使用办法在前面 6.2.3 中说明。

6.4.1.2 嵌入式 SQL 程序的编制

嵌入式 SQL 程序编制一般分下面几个步骤:

- (1) 定义主变量说明;
- (2) 定义游标
- (3) 使用游标与诊断语句进行主程序与 ESQL 间的数据交互。

下面给出一个具体例子。

例 6.2 查询 DEPT 变量中给出的某个系的全体学生信息。

```
EXEC SQL BEGIN DECLARE SECTION
..... /*说明主变量: DEPT, SNO, SNAME, SAGE*/
EXEC SQL END DECLARE SECTION
EXEC SQL DECLARE SX CURSOR FOR /*定义游标*/
    SELECT sno, sn, sa
    FROM S
    WHERE sd=: DEPT; /*由主程序赋值输入主变量 DEPT, 它给出指定的系名*/

EXEC SQL OPEN SX; /*打开游标*/
WHILE /*用循环语句逐条处理集合中的数据 */
{
    EXEC SQL FETCH NEXT FROM SX; /*取一个游标值送入输出主变量*/
    INTO : SNO, : SNAME, : SAGE
    GET DIAGNOSTICS NUMBER=Sqlstate /*取得诊断状态值 Sqlstate*/
    IF(sqlstate<>o) /*若所有查询结束或出现错误, 则退出循环 */
```

```

BREAK
..... /*由主语言语句作进一步处理 */
};
EXEC SQL CLOSE SX; /*关闭游标 */

```

嵌入式 SQL 起源于单机、集中式方式，后又仅限于数据服务器内使用，因此其主体与客体均在同一机器内故连接过程较为简单，在本节中将不予介绍。

6.4.1.3 嵌入式 SQL 的编译

在主语言中嵌入 SQL 后所形成的两种语言组成的程序在编译时是这样处理的，首先要对此程序作予编译以建立两个编译（或解释）系统的关联，其次再实施各语言的编译（或解释），通过此两个步骤以完成整个编译，因此总体看来，嵌入式 SQL 的编译是一个较为复杂的过程。

6.4.2 自含式 SQL

在数据交换中自含式 SQL 是嵌入式 SQL 的一种发展，目前它一般定位于服务器内，用于在服务器内的数据交换，因此它基本不需连接。本节重点介绍数据交换中两种语言的无缝结合上，目前自含式 SQL 主要用于服务器中的应用程序编制，如触发器中的过程及存储过程的编制。

6.4.2.1 自含式 SQL 的内容

自含式 SQL 即是在 SQL 中包含有一般程序设计语言的主要成份，一个完整的自含式 SQL 大致包含有如下内容：

- 传统的 SQL。
- 传统程序设计语言的主要成份，包括流程控制及循环语句、输出语句、调用语句以及服务性的函数库、类库等。
- SQL 中的数据交换，包括游标、诊断及动态 SQL。

下面对其作介绍：

1. 传统 SQL 内容

传统 SQL 内容包括模式定义、数据操纵及数据控制等 SQL 语句。

2. 程序设计语言中的流程控制及循环语句

它们包括：

(1) 复合语句

```

BEGIN
.....
END

```

将多条语句封装形成语句块。

(2) 赋值语句

对 SQL 变量、参数、程序设计语言变量、参数进行赋值。

(3) CASE 语句

提供基于<搜索条件>或操作数等式的选择性执行。

(4) IF 语句

提供基于条件真、假值的选择性执行。

(5) BREAK 语句

终止循环执行。

(6) LEAVE 语句

退出一个已标号的语句块继续执行。

(7) LOOP 语句

重复语句的执行。

(8) WHILE 语句

当指定条件为真时，重复语句的执行。

(9) REPEAT 语句

重复语句的执行。

(10) FOR 语句

对一个表的每一行执行一条语句。

3. 输出语句

(1) PRINT 语句

用于打印输出的语句。

(2) RAISERROR 语句

当出错时输出用户定义的出错信息。

(3) READTEXT

用于文本输出的语句。

4. 返回语句

(1) RETURN 语句

用于调用返回的语句。

5. 游标语句与诊断语句

用游标语句与诊断语句以进行集合量与标量间的数据交换，它们包括：

(1) DECLARE 语句

用于定义游标的语句。

(2) OPEN 语句

打开游标语句。

(3) FETCH 语句

用于推进游标并读取的语句。

(4) CLOSE 语句

关闭游标语句。

(5) GET DIAGNOSTICS 语句

获取诊断信息的语句。

6. 动态 SQL

内含式 SQL 中一般还包括部分动态 SQL。

7. 函数库或类库

内含式 SQL 中一般还包括一些函数库或类库，如类型转换、日期计算、字符串操作，数学计算以及文本、图像操作等函数或类库。

6.4.2.2 自含式 SQL 的编程

自含式 SQL 将 SQL 与程序设计融于一体两者构成无缝接口，因此它比嵌入 SQL 高出了一个层次，其优势是明显的，主要表现为：

1. 在嵌入 SQL 中由于涉及到两种语言的联编，因此它的整个编译过程较为复杂，而在自含式 SQL 则是一个统一的语言，因此编译过程就极为简单了。

2. 基于同样的理由，在嵌入式 SQL 的程序中需区分主语言程序与 SQL 子语言程序；需区分主变量与 SQL 变量，而在自含式 SQL 中则无须区别。

3. 在嵌入式 SQL 与自含式 SQL 中，在数据交换中均需使用游标语句与诊断语句。

从上面三点比较中可以看出，自含式 SQL 书写简单、编译方便、执行效率高，因此自含式 SQL 问世后嵌入式 SQL 已逐渐衰落，当然嵌入式 SQL 也还有其一定的优势，这主要是嵌入式 SQL 中的某些主语言内容的丰富、使用的广泛（如 C、JAVA 等），因此它还有一定的应用市场。

自含式 SQL 编程的结构单位是块，一个块一般包括三部分内容，它们是：

1. 声明部分：它给出程序的局部变量声明与游标声明，它以 DECLARE 开始。

2. 主体部分：它是程序的可执行部分，以 BEGIN 开始。

3. 例外部分：它给出程序的例外处理，包括出错处理等，它以 EXCEPTION 开始。

块的结构因语言而异，这里所示的结构是以 ORACLE 的 PL/SQL 为例。

下面我们举几个关于自含式 SQL 编程的例子。

例 6.3 查询 DEPT 变量中给出的某个系的全体学生信息。

这个例子即是嵌入式 SQL 中的例 6.2，在自含式 SQL 的编程中就变得简单多了。

```
DECLARE
    SNO CHAR (5)
    SNAME CHAR (20)
    SAGE SMALLINT
    DEPT CHAR (2)
    DECLARE SX CURSOR FOR      /*定义游标*/
        SELECT sno, sn, sa
        FROM S
        WHERE sd:=DEPT      /*由变量 DEPT 赋值 sd, 给出指定的系名*/
BEGIN
    OPEN SX                  /*打开游标*/
    WHILE                    /*用循环语句逐条处理集合中数据*/
    {
        FETCH NEXT FROM SX    /*取一个游标值至标量中*/
        INTO SNO, SNAME, SAGE
        GET DIAGNOSTICS       /*取得诊断状态值 sqlca . sqlstate*/
        IF sqlca . sqlstate<>o /*若查询结束或出现错误则退出循环*/
        BREAK
        .....                /*应用程序作进一步处理*/
```

```
};  
CLOSE SX          /*关闭游标*/  
END
```

6.4.3 调用层接口

自数据库应用进入网络环境后，出现了 C/S 结构方式，C/S 方式的结构特点是在应用系统中应用程序与数据库分离（此种方式在以后的 B/S 方式中也如此），亦即是说，在 C/S 方式下应用程序功能分布于客户端 C，而数据库则功能分布于服务器端 S，而它们间则有网络相联。在此种情况下，应用程序与数据库间的数据交换实际上成为网上两个结点间的数据通讯，它们称为调用层接口，即通过调用以接口方式实现数据交换。目前有关的标准及产品有三种，它们是 SQL'99 中的 SQL/CLI 以及 ODBC、JDBC 两种产品，由于 SQL 标准在调用层接口中的重要性，因此在本节中我们先介绍 SQL/CLI，并将在第十三章中介绍 ODBC 与 JDBC。

SQL/CLI 由 47 个函数组成，数据交换遵守 6.3 所规定的数据库交换流程，因此我们将主要函数按流程的四个部分划分介绍如下：

1. 连接阶段

有关连接阶段的函数包括关于分配 SQL 环境、SQL 资源、SQL 语句、SQL 连接以及最后是创建连接。

有关连接阶段的函数有：

- AlloEnv 分配 SQL 环境
- AlloStmt 分配 SQL 语句
- AlloHandle 分配 SQL 资源
- AlloConnect 分配 SQL 连接
- Connect 创建连接

2. 数据交换阶段

在数据交换阶段共需用到四种类型的函数，它们是：

(1) 有关游标的函数

- SetCursorName 设置游标名
- GetCursorName 得到游标名
- Fetch 推进游标并读取数据
- ScrollFetch 在指定的行定位游标并读取数据
- CloseCursor 关闭游标

(2) 有关诊断的函数

- GetDiagField 从诊断区得到信息

(3) 有关动态 SQL 的函数

由于在 SQL/CLI 中 SQL 语句经常以动态 SQL 形式出现，因此须有动态 SQL 的函数，它们是：

- SetDescField 在描述符区设置一个字段
- GetDescField 从描述符区取得一个字段
- CopyDesc 复制描述符

(4) 有关数据获取的函数

它包括两种数据获取的函数，它们是直接执行函数以及先作准备，再执行的两条函数，共计三条函数。

- ExeDirect 直接执行语句
- Prepare 准备语句
- Execute 执行准备的语句

在数据获取函数中，SQL 语句以函数中的参数形式出现。

此外还包括执行事务的函数，一般讲，一个数据获取函数的执行即表示一个新的事务开始，而用函数 EndTran 表示事务的结束。

- EndTran 终止一个 SQL 事务

3. 断开连接阶段

有关断开连接阶段的函数有关于取消分配的 SQL 语句、SQL 环境以及 SQL 连接，释放相关的资源，并最终断开连接。

- FreeStmt 取消分配的 SQL 语句
- FreeHandle 释放 SQL 资源
- FreeEnv 取消分配的 SQL 环境
- FreeConnect 取消分配的 SQL 连接
- Disconnect 断开连接

6.4.4 Web 方式

在 Web 环境中的数据交换主要是 XML 与数据库间的交换，它称为 Web 交换方式或简称 Web 方式，目前常用的有三种形式。

1. 通过 ASP 实现数据交换

在微软公司的 Windows 环境下基于 Web 的开发工具 ASP 可以实现数据交换，ASP 具有脚本语言 VBScript，JavaScript 以及 ADO 控件，其中脚本语言能插入 XML，而 ADO 也是一种调用层接口，因此可以通过 ASP 将 XML 与数据库相接口，从而实现 Web 下的数据交换，其示意图如图 6.4 所示。

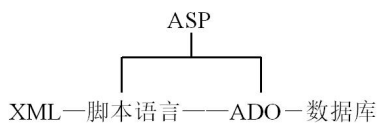


图 6.4 基于 ASP 的数据交换方式

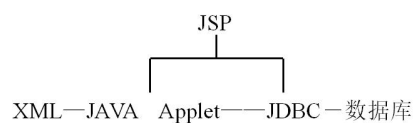


图 6.5 基于 JSP 的数据交换方式

2. 通过 JSP 实现数据交换

在 SUN 公司的 UNIX 环境下基于 Web 的开发工具 JSP 可以实现数据交换。JSP 的 JAVA Applet 具有能插入 XML 中的功能，同时能通过调用 JDBC 而实现与数据库接口，因此可以通过 JSP 将 XML 与数据库接口，从而实现 Web 下的数据交换，其示意图 6.5 所示。

3. SQL/XML 标准

根据 SQL'03 标准中的 PART9: SQL/XML 给出了 XML 与数据库的交换接口标准，在

该标准中可以将 XML 看成是数据库中的一种新的数据类型,因此 XML 也可看成是一种数据库的结构,这样它们间的数据交换实际上变成为数据库内部数据交换了。

有关数据交换中 Web 方式的详细介绍可见第十三章“分布式数据库与 Web 数据库”,在这里仅作概念性的介绍。

习题六

- 6.1 什么叫数据交换?数据交换起什么作用?试说明。
- 6.2 试给出数据交换的模型并给出说明。
- 6.3 请给出数据交换的五种方式以及相应的环境。
- 6.4 请给出数据交换的五种管理并作出说明。
- 6.5 请给出数据交换的流程并作出说明。
- 6.6 请给出嵌入式编程过程,并给出一实例。
- 6.7 自含式 SQL 包括那些内容,请说明之。
- 6.8 调用层接口 SQL/CLI 由那些主要函数组成?并作出说明。
- 6.9 请说明在 Web 方式下如何通过 ASP 以实现数据交换。
- 6.10 请指出 SQLserver 或 ORACLE 中有那些数据交换功能。

第六章复习指导

本章主要是对数据库中数据交换作介绍。数据交换是数据库系统近年来发展的一个重要部分，由于应用环境的变化而引发出数据库与外界进行数据交换的多种形式。数据交换是数据库应用的重要前提与保证，学习本章后读者应对数据交换的原理与使用有基本的了解。

1. 数据交换的基本概念：数据主体与数据客体间的交互过程。
2. 数据交换的模型



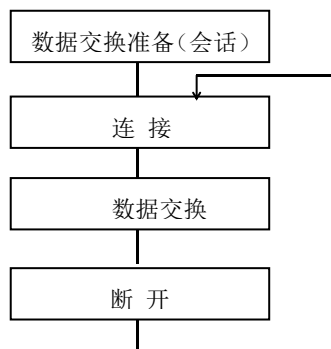
3. 数据交换的五种方式

- 人机对话方式——单机，集中式，数据主体为人；
- 嵌入式方式——单机，集中式（或 C/S 中的 S），数据主体为应用程序；
- 自含式方式——单机，集中式（或 C/S 中的 S），数据主体为应用程序；
- 调用层接口方式——C/S 方式，数据主体为应用程序；
- Web 方式——B/S 方式，数据主体为数据（或应用程序）。

4. 数据交换管理

- 会话管理
- 连接管理
- 游标
- 诊断
- 动态 SQL

5. 数据交换流程



*第七章 数据库的物理组织

数据库的物理组织包括计算机的磁盘组织以及基于磁盘的文件组织。本章主要介绍磁盘、文件组织以及基于这两者之上的数据库中数据以及它们的分类。

7.1 概论

数据库是一个庞大的数据集合，它不仅存储大量数据还有多种辅助信息及复杂数据结构，如何将它们以最合理方式组织起来，使之达到存储效率高及存取速度快之目的是本章要讨论的问题。

数据库中数据一般存储于磁盘中，而磁盘又按文件形式组织，因此本章讨论的另一个问题是如何组织文件以及讨论磁盘结构。

本章讨论的第三个问题是索引技术与散列技术，它是提高文件查找速度的主要途径，在庞大的数据库中查找特定的数据犹如“大海捞针”般困难，如何在“大海”中迅速定位找到指定的“针”即是索引技术与散列技术的主要任务。

数据库的物理组织基本上可用图 7.1 所示的三个层次组成，它的最低层是磁盘层，存放真正的数据。在磁盘层上是文件层，它以文件形式组织，文件由磁盘块按一定规则组成。在文件层上的是数据库层，它按一定数据结构（如关系中的表结构）组织，本章主要讨论这三层的物理组织与结构。

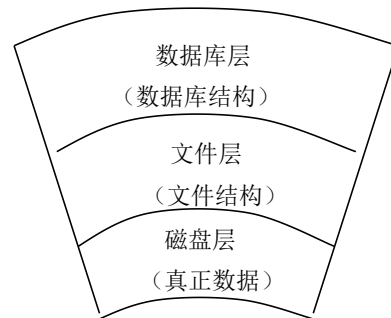


图 7.1 数据库物理组织的三个层次

7.2 数据库的物理存储介质

与数据库有关的物理存储介质以磁盘存储器为主，共有以下三类：

1. 主存储器 (main memory)

主存储器又称内存或主存，它是计算机机器指令执行操作的地方。由于其存储量较小，成本高、存储时间短，因此它在数据库中仅是数据存储的辅助实体，如作为工作区 (work area) (数据加工区)、缓冲区 (buffer area) (磁盘与主存的交换区) 等。

2. 磁盘存储器 (magnetic-disk storage)

磁盘存储器又称二级存储器或次级存储器。由于它存储量较大，能长期保存又有一定的存取速度且价格合理，因此成为目前数据库真正存放数据的物理实体。

3. 磁带存储器 (tape storage)

磁带是一种顺序存取存储器，它具有极大的存储容量，价格便宜可以脱机存放，因此可以用于存储磁盘或主存中的拷贝数据，它是一种辅助存储设备，也称为三级存储器。

磁盘能存储数据，但不能对它的的数据直接“操作”，只有将其数据通过缓冲区进入内存才能对数据作操作 (在工作区内)，因此磁盘与内存的有效配合构成了数据库物理结构的主要内容，而加上磁带存储的辅助性配合从而构成了一个数据库物理存储的完整实体，图 5.2 给出了其示意图。

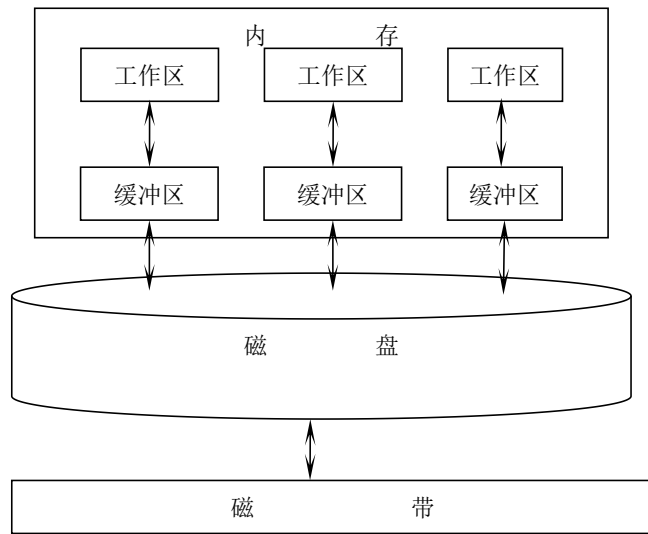


图 7.2 磁带、磁盘与内存的有效结合

7.3 磁盘存储器及其结构

由于磁盘是数据库数据的主要物理存储实体，因此本节主要介绍磁盘及其结构。

磁盘存储器是一种大容量、直接存取的外部存储设备，所谓大容量指的是其存储容量极大，大约在几个 GB 到几十个 GB，甚至几百个 GB 之间，所谓直接存取指的是可以随机到达磁盘上任何一个部位存取数据。磁盘存储器由磁盘盘片与磁盘驱动器两部分所组成。

1. 磁盘盘片

磁盘盘片是一种表面上下两层涂以磁性材料的铝片，盘片是一种圆形物体，分上、下两面，以圆心为主轴，将盘片划分成若干个磁道 (track)，每个磁道是一个半径不同的同心圆，每个磁道又分为若干个扇区 (sector)，它又称磁盘块 (block)，磁盘块是磁盘交换信息的基本单位，通常可以存放 32~4096B，而常用为 512B，每个磁道一般可有 4~32 个磁盘块，每个盘面上一般有 20~1500 个磁道，图 7.3 给出了磁盘盘面的结构。

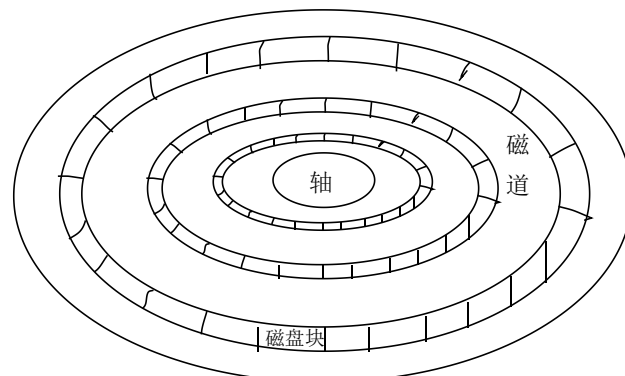


图 7.3 磁盘盘面结构

一个磁盘存储器往往由若干个盘片（6~11片）组成一个盘片组，固定在一个主轴上，以每个盘片磁道为 viewpoint 可以构成一个无形的同心圆柱体，从内到外层层相套。每个圆柱体从上到下有若干个磁道围绕其上，构成如图 7.4 所示的结构。

2. 磁盘驱动器

磁盘驱动器由活动臂、读写头等组成，每个盘片有两个臂，分别对应上、下两面，每个臂的尽头是一个读/写头（或称磁头），用它可以读取（或写出）盘片中的数据。一个由 n 个磁盘片所组成的盘片组对应有 $2n$ 个（每个盘片分两面）活动臂，它们组合在一起构成臂组合件，这种组合件可以自由伸缩活动，它以磁道为单位往前推进或向后退缩，用它可以对磁道定位，由于它是组合方式以全体活动臂为单位作进退，因此它的推进或后退实际上是对圆柱体定位。

3. 磁盘存储器

一个磁盘存储器是由盘片组以及磁盘驱动器所组成，其中盘片组以轴为核心作不间断的旋转，速度以每秒 60、90、120 或 150 转不等，而活动臂组合件则以圆柱体为单位做前进或后退操作。

这样，一个磁盘存储器上的任何一个磁盘块都可由下面三个部分定位。

(1) 圆柱体号：确定圆柱体（由活动臂移动定位）。

(2) 读/写头号：确定圆柱体中磁道（由选择组合件中活动臂定位）。

(3) 磁盘块号：确定磁道中的盘块号（由盘片组旋转定位）。

整个磁盘存储器结构可见图 7.5。

4. 磁盘存储器的 I/O 操作

为进行有效管理，系统对磁盘作统一编址，编址按圆柱体号、磁道号及盘块号编码，编码规则如下。

(1) 圆柱体号：设有 n 个圆柱体则编号自柱面的外层至内层，从 $0 \sim n-1$ 。

(2) 磁道号：设一个圆柱体有 m 个磁道，则磁道号统一编码从上到下顺序编号，从 0 号圆柱体至 $m-1$ 号圆柱体共为 $0 \sim (n \times m) - 1$ 个。

(3) 磁盘块号：设一个磁道有 r 个盘块，则磁盘块号也是统一编码，从 $0 \sim (n \times m \times r) - 1$ 个。

磁盘在投入使用前都要进行格式化，其目的是在各盘块的头部加注该块地址，包括该块所在的圆柱体号，读/写头号及盘块号以及某些

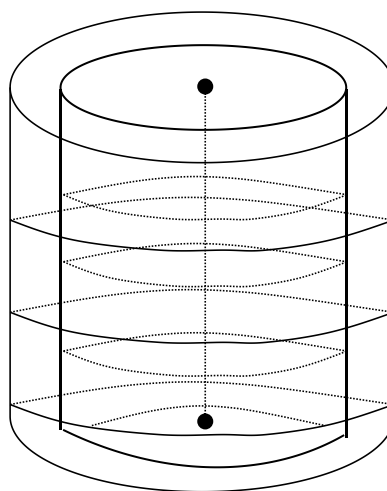


图 7.4 圆柱体示意图

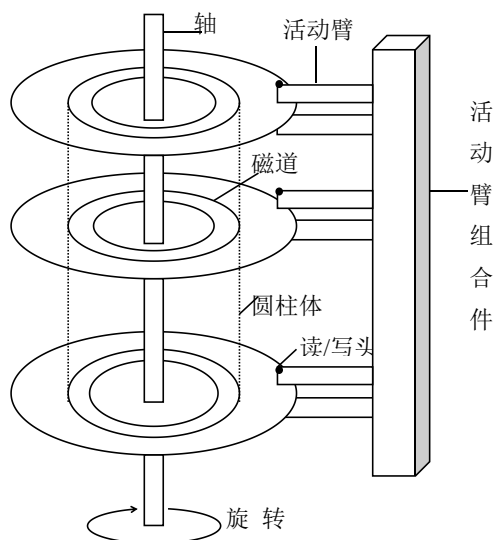


图 7.5 磁盘存储器示意图

状态标志。在具体操作时用户给出磁盘地址，此时活动臂组合件作机械运动并定位于指定圆柱体，同时系统选择指定的读/写头以确定磁道，最终读/写头跟踪旋转的磁道，并读出旋转时每磁盘块的地址。当用户给出的地址与磁盘地址一致时则表示此地址已找到，此时系统就将该地址中的数据读入内存中的磁盘缓冲区（或从磁盘缓冲区将数据写入指定磁盘地址），这就完成了一次磁盘读/写操作或称 I/O 操作。

7.4 文件组织

在磁盘中数据库中数据以文件形式组织，一般磁盘空间可由若干个文件组成，而文件则由记录（record）组成。记录有型与值之分，一个记录型有若干个项（item），项中规定域，它给出了项的取值范围。记录型给出了记录的结构类型，而记录值（亦可简称记录）则给出了符合记录型要求的数据，一个文件可以由多个记录值组成。

文件一般由操作系统管理。但是在现代数据库管理系统中，为管理上方便起见，一般由 DBMS 在初始化时向操作系统一次性申请数据库中全部磁盘空间，然后再由 DBMS 统一分配管理数据库的磁盘空间、组织文件。这种方法一般称为原始磁盘（raw disk）法。

7.4.1 文件记录与磁盘块

文件记录是一个逻辑单位，在具体实现时记录必须分配到具体磁盘块中去，但是记录的大小往往与磁盘块大小不一致，因此在具体分配时可涉及到多种方式，一般讲有如下几种。

- (1) 单块单记录 即一个记录占用一个磁盘块，具体表示如图 7.6 所示。

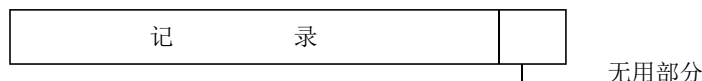


图 7.6 单块单记录

- (2) 单块多记录 即一个磁盘块包含多个记录，具体表示如图 7.7 所示。

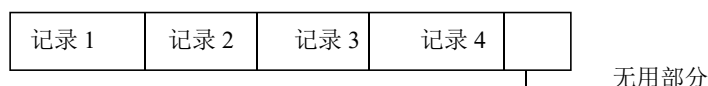


图 7.7 单块多记录

- (3) 多块单记录 即一个记录占用多个块，此种记录是一种跨块记录，具体如图 7.8 所示。



图 7.8 多块单记

- (4) 多块多记录 即 n 个记录占用 m 个磁盘块，此种方式也存在跨块现象，具体表示如图 7.9 所示，这表示一个 2 块 5 记录。

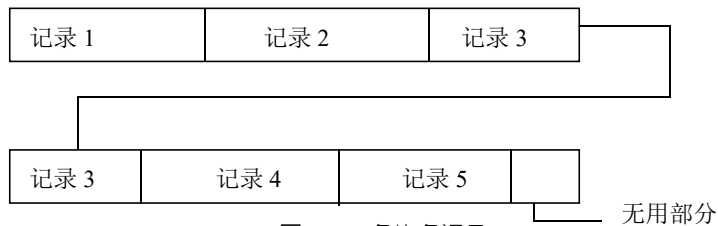


图 7.9 多块多记录

7.4.2 文件的定长记录与变长记录

在数据库中文件一般按定长及变长两种方式组织。

1. 文件的定长记录

一般在文件中的记录都是有固定长度的，这种长度一般都由文件记录型确定。

2. 文件的变长记录

在文件中的记录有时其长度是可以变化的，如一个文件中允许存在不同记录型的记录，或允许记录型记录可以是变长的，以及记录中某些项是变长的等。

变长记录的表示有两种形式：

(1) 变长记录的字节串表示形式。这种形式是将每个记录看成连续字节串，然后在每个记录尾部附加特殊的标志符叫“记录尾标识符”，用以区别不同的记录。

(2) 变长记录的定长表示形式。在文件中往往使用一个或多个定长记录来表示变长记录的方法。

变长记录在具体实现时有两种技术，预留空间及指针技术。所谓预留空间的办法是取变长记录中最长一个记录长度作为存储空间的记录长度用以存储变长记录每个变长记录，用此定长记录存取，其尾部空余部分填上特定之空值或记录尾标标志。所谓指针形式即是若干个定长记录通过指针构成一个变长记录，在此形式中，每个定长记录后加一个指针字段，该字段指出是否有指针以及如果有指针则给出下一定长记录的地址，此种方法即是用若干个定长记录拼成一个变长记录。

7.5 文件记录组织

文件记录在磁盘中如何存放和如何安排，这是文件中的记录组织问题。文件的记录组织对文件的存取方法及效率会产生很大影响，一般文件的记录组织有如下几种方式。

1. 堆文件组织 (heap file)

在这种组织中，记录可以放在文件的任何位置上，一般以记录录入时间先后为序。在此种组织中记录的存储顺序与它的键值无关。

2. 顺序文件组织 (sequential file)

在这种组织中记录是按其项值的升序或降序存储的。在这种文件中每个记录增加一个指针字段，根据项值的大小用指针把记录链接起来。

在文件初始建立时存储记录尽可能使物理顺序与项值大小顺序一致，以便访问数据时减少磁盘块访问次数。

图 7.10 给出了此种文件的组织方式，在图 7.10 中，关系 S 的码为 Sno，按 Sno 的键值

从小到大用指针构造顺序文件。

顺序文件可以很方便地按码值大小顺序读出所有记录。

Sno	Sn	Sd	Sa	
S ₁	LU	CS	18	
S ₂	LI	CS	17	
S ₃	XU	MA	18	
S ₄	LO	CS	18	
S ₅	LIN	PH	19	
S ₆	WANG	CS	17	
S ₇	SEN	MA	17	
S ₈	EHEN	PH	18	




图 7.10 顺序文件例子

3. 散列文件组织 (hashing file)

在这种组织中是根据记录中的某个项值通过散列函数求得的值作为记录的存储地址 (即块号)。

4. 聚集文件组织 (clustering file)

在关系数据库中一般一个文件存储一个关系, 而在这种组织中一个文件可以存储多个关系的记录, 不同关系中有联系的记录存储在一起可以提高查找速度。如将教学数据库中关系 S 和 SC 组织在同一个文件中, 那么在作联结操作时可以较快得到结果, 如下面的查询:

```
SELECT S • sno, S • sn, SC • cno, SC • g
FROM S, SC
WHERE S • sno = SC • sno
```

在关系 S 与 SC 数量很大时, 要做联接的查询速度是很慢的。但是如果把 S 和 SC 放在一个聚集文件内, 那么在读学生信息时能够同时把学生的课程及成绩一起读入, 下面图 7.11 给出具体的例子。

7.6 索引技术与散列技术

在数据库中数据查找的速度是至关重要的, 但是当数据库中数据量增大时。数据查找速度就会受到影响; 当数据量极大时, 查找速度将会受到严重影响。为解决此问题需引入索引技术与散列技术, 本节将介绍这两种技术。

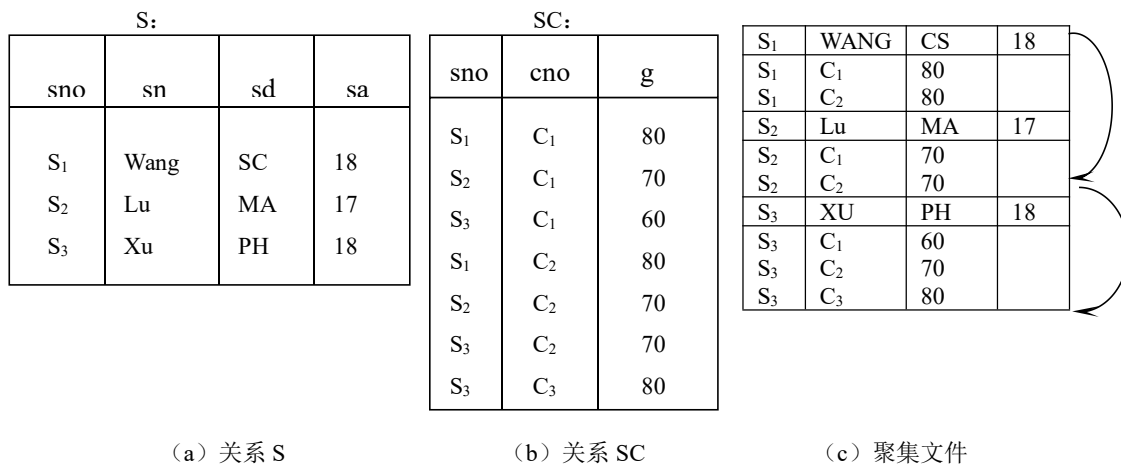


图 7.11 聚集文件例子

7.6.1 索引技术

数据库的存储实体是文件，而索引技术是提高文件查找速度的一种方法。在索引技术中对文件（一般用顺序文件）的查找采用类似书刊中目录的方法，即对文件中记录的指定项（称索引项）的项值给出其记录的地址，从而构成若干索引记录，它们称索引，索引一般也用文件表示，其结构如图 7.12 所示。

索引项值	对应记录地址

图 7.12 索引结构

因此，在索引技术中一个索引文件一般由主文件与索引两部分组成，其中主文件存放数据而索引则存放数据地址。

索引一般分主索引和辅助索引两种，其中主索引指的是索引中索引项值的大小顺序与主文件顺序一致，否则则称为辅助索引。

1. 主索引

主索引是索引中的最常用的一种，它一般可有下面三种实现方法：

(1) 稠密索引。所谓稠密索引（dense Index）是指对主文件索引项的每个“项值”建立一个索引项值，即索引记录包括索引项中的所有项值以及指向该值的记录链表中第一个记录的指针，图 7.13 给出了稠密索引的一个例子。

(2) 稀疏索引。稀疏索引（Sparse Index）是指对主文件索引项的部分项值建立索引记录，这种部分项值的选择具有一定的稀疏特征，即每隔一定距离选择一个，图 7.14 给出了稀疏索引的一个例子。

稀疏索引适用于索引本身数量很大时为减少索引记录而采用的一种手段。



图 7.13 稠密索引例图

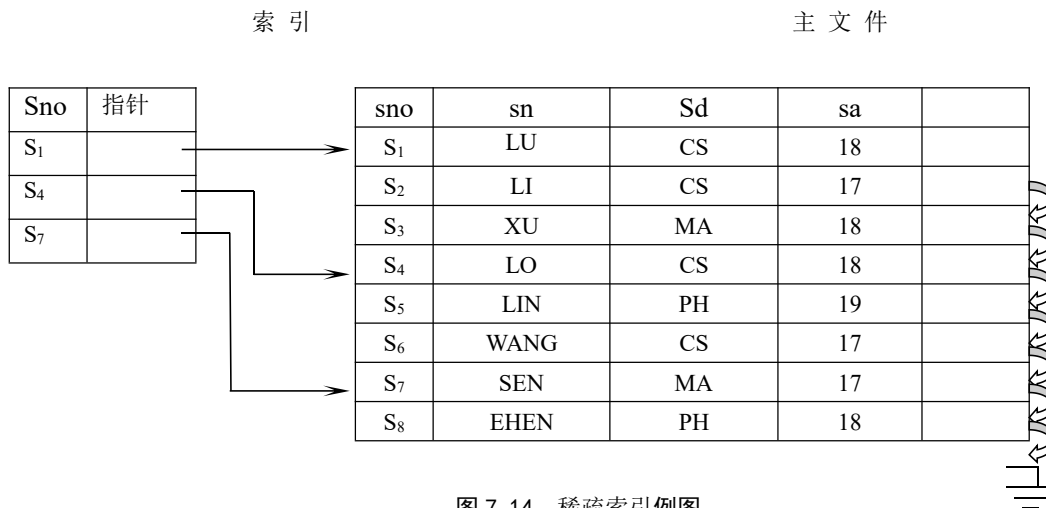


图 7.14 稀疏索引例图

(3) 多级索引。当索引本身数量很大时还有一种办法是采用多级索引，即对索引本身采用索引。图 7.15 给出了二级索引的例子。

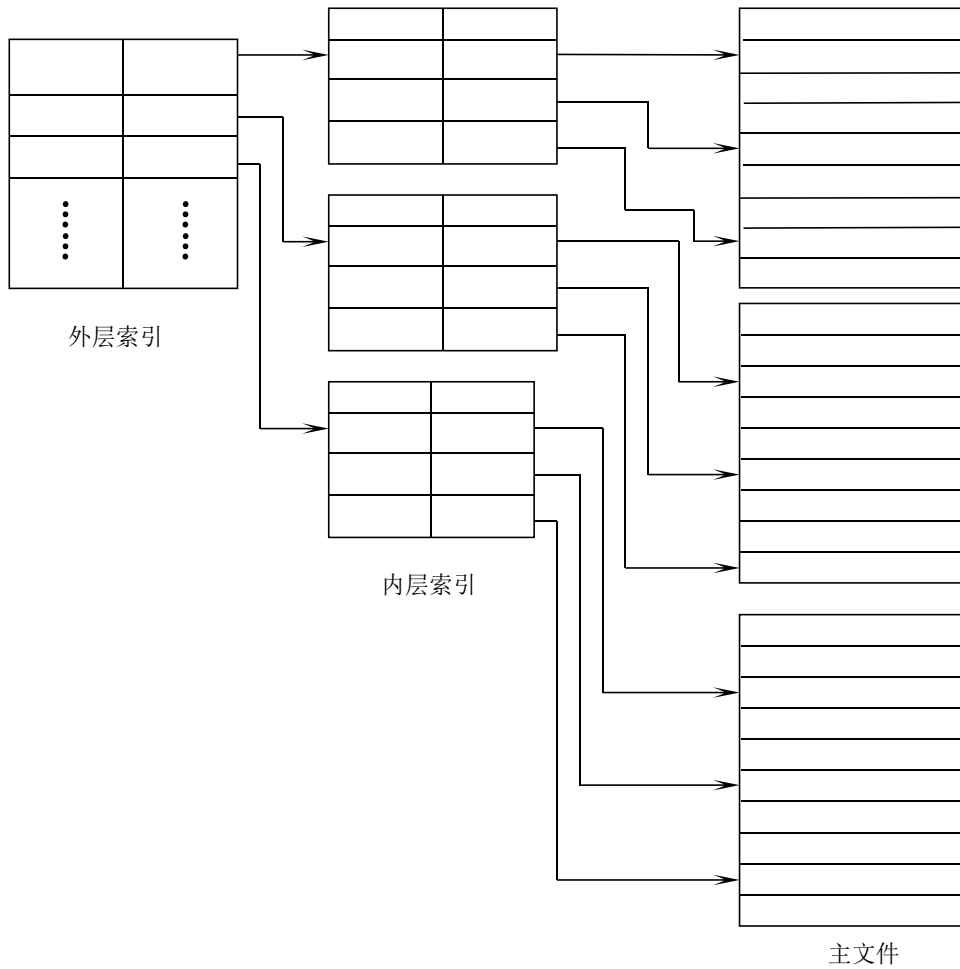


图 7.15 二级索引例图

2. 辅助索引

由于主索引中具有相同索引项值的记录在同一地址或相邻地址中因而查找速度快，有时还需使用辅助索引。采用辅助索引查找速度较快，一般采用下面的方法。即仍采用索引记录（索引项值与对应的指针）不过此时指针指向的不是主文件记录地址而是指向一个桶（bucket），桶内存放指向具有同一索引项值的指针。如在前稠密索引例中，建立以年龄 S_a 为索引项的辅助索引。如图 7.16 所示，在这种结构中，以辅助索引为中介，可以通过二层指针很方便查到对应的主文件地址。

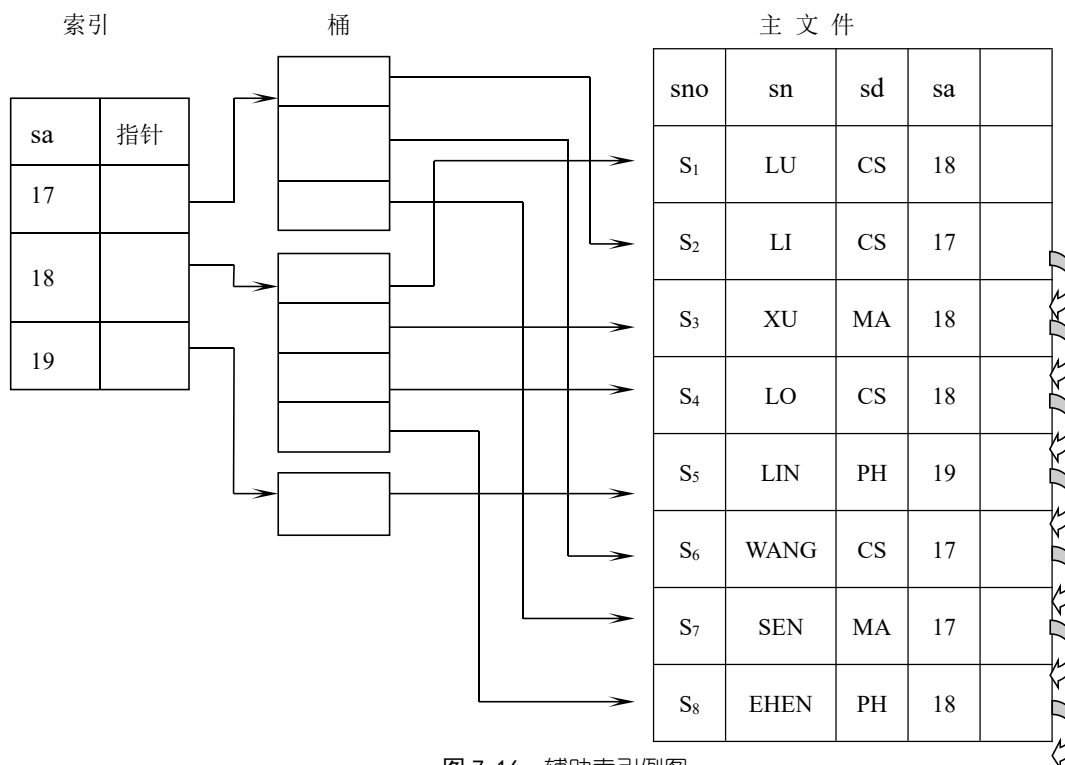


图 7.16 辅助索引例图

7.6.2 索引技术中的 B⁺树

在一般数据库中由于主文件数据量往往很大，因此最有效的办法是采用多级索引，而目前最流行的多级索引技术是 B⁺树索引。

B⁺树的索引结构是树的形式，最上一级索引是树的根结点，最下一级索引是树的叶结点。该级索引指针直接指向主文件的记录地址，而非叶的其他结点（包括根结点）的索引则指向其下一结点的地址。

非叶结点的索引一般采用稀疏索引，而叶结点索引则采用稠密索引，在 B⁺树中索引的结构如图 7.17 所示。

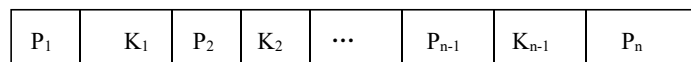


图 7.17 B⁺树的索引结构

其中 P_i 为指针，K_i 为索引项值 (i=1,2, ⋯,n)。对于非叶结点第 i 个指针 P_i，P_i 指向此图结构为 B_i 子树中所有索引项值均小于 K_i，而大于或等于 K_{i-1}。P₁ 所指向的子树中所有索引项值均小于 K₁，P_n 所指向的子树中所有索引项值均大于 K_{n-1}。而对于叶结点，第 i 个指针 P_i 则直接指向主文件中对应索引项值 K_i 的记录地址，而 P_n 则指向右边的叶结点地址。

图 7.18 给出了一个 B⁺树的例子，这是一个三级树，为举例方便起见仅给出 18 个记录，在例中索引项为 sno，从 S₁ 到 S₁₈，是一种主索引的形式。

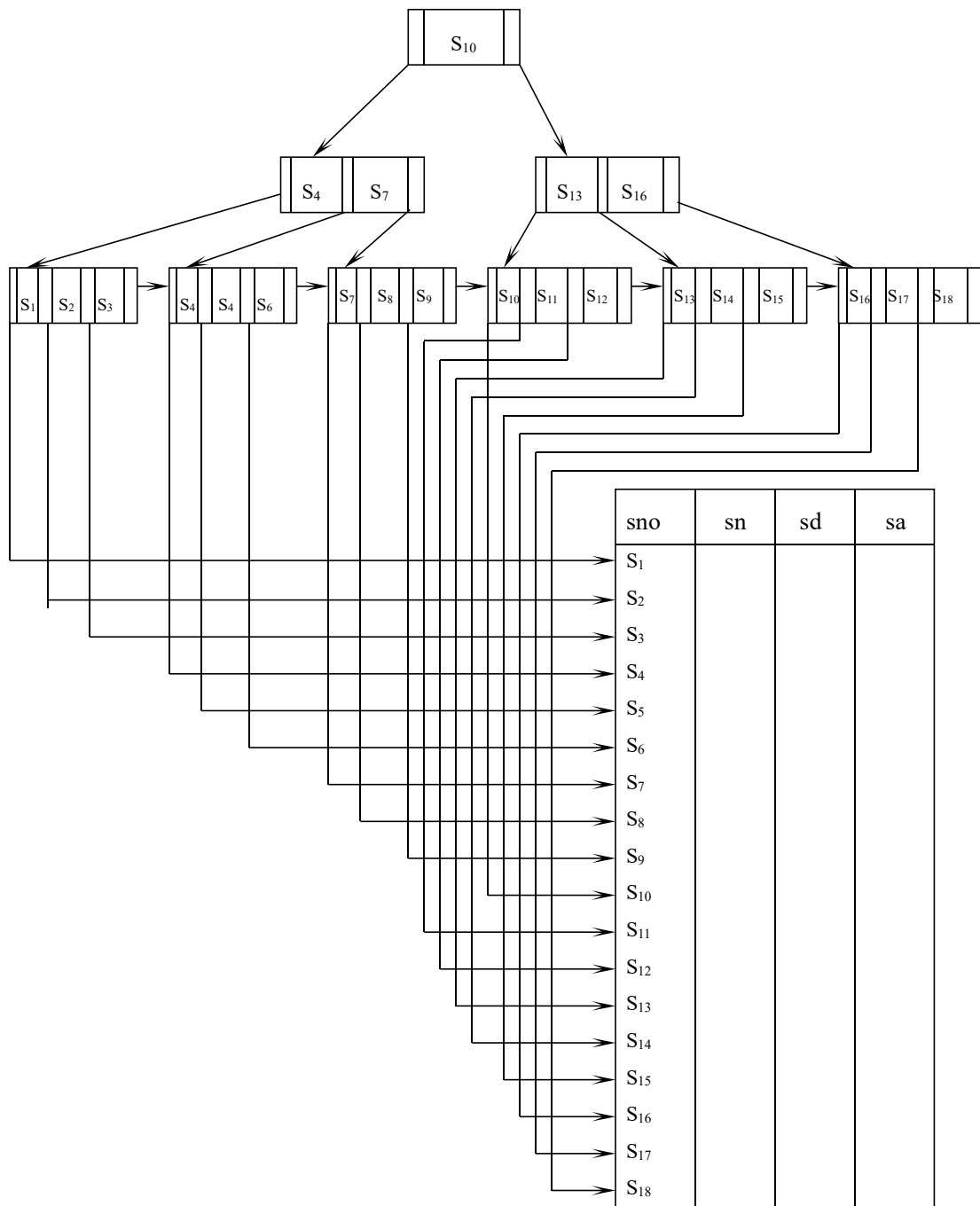


图 7.18 B⁺ 树例图

对于 B⁺ 树的查询可用如下办法（以查索引项值 K 为例）：

- (1) 在根结点中找大于 K 的最小索引项值，设为 K_i，由 K_i 的左边指针 P_i 开始到达第 2 层结点。如无比 K 小的索引项值应用右指针。
- (2) 在第 2 层结点中用类似办法找到相应指针并达到第 3 层结点。
- (3) 如此重复到达叶结点为止，并从叶结点中找到直接指向主文件的记录地址。

(4) 根据此地址找到该记录。

例 7.1 以图 7.18 为例查找 S_{15} 的记录。

- (1) 首先从根结点中找到 S_{10} 右边指针，由该指针到达第 2 层结点。
- (2) 在第 2 层结点中找到 S_{16} 左边指针，由该指针到达第 3 层结点。
- (3) 在第 3 层结点中找到 S_{15} 左边指针，由该指针直接找到 S_{15} 的记录地址。
- (4) 最终由叶结点的指针找到并取出指定 (S_{15}) 的记录。

7.6.3 散列技术

提高数据库查找速度的另一种办法是采用散列 (Hash) 技术。散列技术是一种有效、简单的提高查询速度的方法。它的基本原理是利用一种散列函数 $h(K_i)$ 建立起主文件指定项值与磁盘物理块间的映射关系，这样只要给出指定项的值 K_i 立即可通过 $h(K_i)$ 得到其对应的存储物理块地址。散列技术的示意图如图 7.19 所示。

散列技术的具体实现方法如下：

- (1) 建立主文件的指定项 K 以及该项值的集合 $\{K_1, K_2, \dots, K_n\}$ 。
- (2) 建立磁盘物理存储单位桶 (Bucket) 以及桶地址的集合 $\{b_1, b_2, \dots, b_m\}$ 。桶可以存放多个记录，桶可以是磁盘物理块，也可以是比块还大的物理空间。
- (3) 建立散列函数 $h(K_i)$ ，它建立主文件指定项的值与桶间的对应关系，对应一个 K_i 必通过 $h(K_i)$ 找到唯一的一个桶地址。

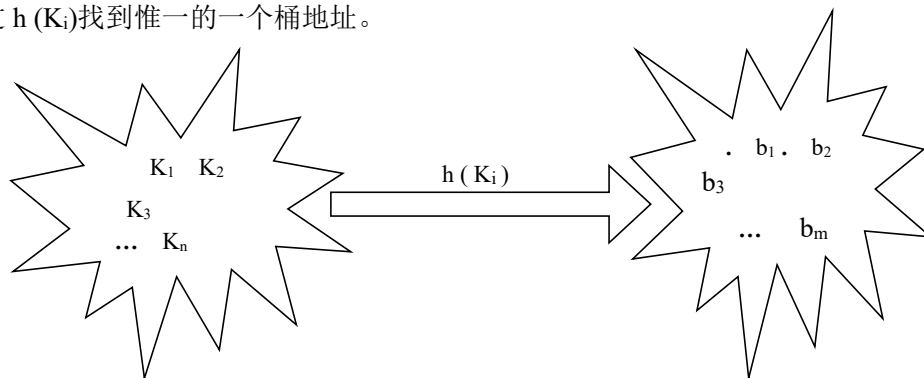


图 7.19 散列技术示意图

散列技术中的桶空间是固定的，但是在实际应用中往往其负担不均衡造成桶空间溢出。此时需要通过链接的方法将一些特制的“溢出桶”与其相联，以达到扩大桶空间的目的。还可以将散列与索引相结合形成“散列索引”，从而使其效果更为有效。散列索引的具体构造方法是：首先将主文件中每个索引项值建立一个索引记录，然后将这些索引记录组织成散列结构，这样将两者有机结合构成了一种散列索引结构。

例 7.2 用图 7.13 中的主文件建散列索引。以 sno 为指定项，首先建立索引记录（见图 7.13），然后建 $\{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8\}$ 与桶 $\{0, 1, 2\}$ 间的散列函数 $h(K_i) = i \pmod{3}$ ，其中 $i \pmod{3}$ 是以 3 为模的剩余，如 $2 \pmod{3} = 2, 3 \pmod{3} = 0$ 这样就建立起如图 7.20 所示的散列。

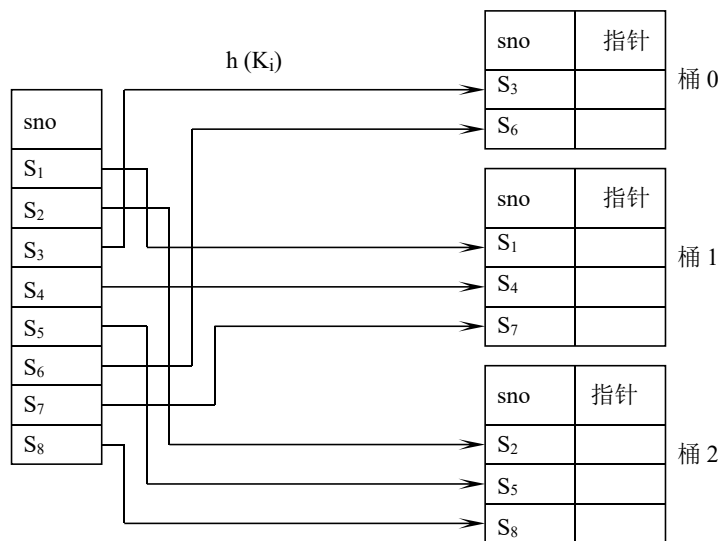


图 7.20 散列图示

将两者结合最终可得到一个散列索引如图 7.21 所示。

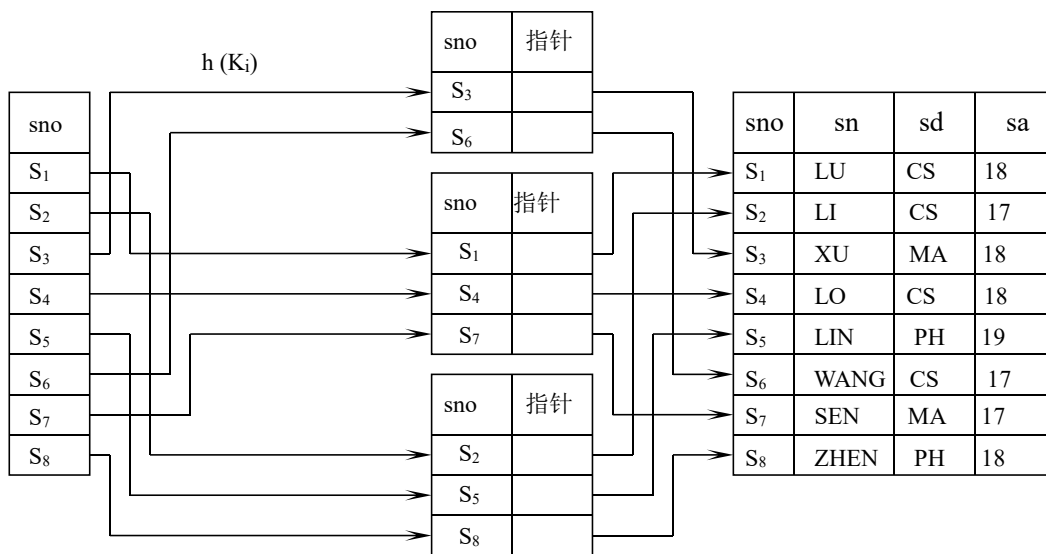


图 7.21 散列索引例图

7.7 数据库与文件

7.7.1 数据库中数据分类

存储于数据库中的数据除了数据主体外还需要很多相应的辅助信息，它们的整体构成了完整的数据库数据的全体。

1. 数据主体 (main data)

数据库中数据主体分数据体自身及辅助数据，其中数据体自身即是存储数据的本身，

如关系数据库中的数据元组，而辅助数据即是相应的控制信息如数据长度、相应物理地址等。

2. 数据字典 (data dictionary)

有关数据的描述作为系统信息存储于数据字典内，数据字典一般存放数据模式结构信息、视图信息以及有关物理模式结构信息，此外，还存放有关完整性、数据安全的信息。数据字典信息量小但使用频率高，是一种特殊的信息。

3. 数据间联系的信息

数据主体内部存在着数据间的联系，需要用一定的“数据”表示，用链接或邻接方法实现，如用指针方法或 MSAM 层次顺序方法等，而在关系数据库中数据主体内在联系也用关系表示并且融入主体中。

4. 数据存取路径信息

在关系数据库中数据存取路径都是在数据查询要求时临时动态建立，它们通过索引及散列实现，而索引与散列的有关数据，如索引目录及散列的桶信息均需存储并在数据操纵时调用。

5. 与数据主体有关的其他信息

(1) 日志信息：日志用于记录对数据库作“更新”操作的有关信息，以对付数据库遭受破坏时恢复之用。

(2) 用户信息：有关数据库用户登录信息以及相应的用户权限信息。

(3) 审计信息：用于跟踪用户是否正确使用数据库的审计信息。

7.7.2 数据库存储空间组织

在数据库数据存储空间组织统一由 DBMS 管理，它包括系统区和数据区，其中系统区有数据字典、日志、用户信息及审计信息等，而数据区则由数据主体及相应信息组成。

数据库的存储空间组织在逻辑上一般由若干分区组成。其中系统区有若干个分区：如

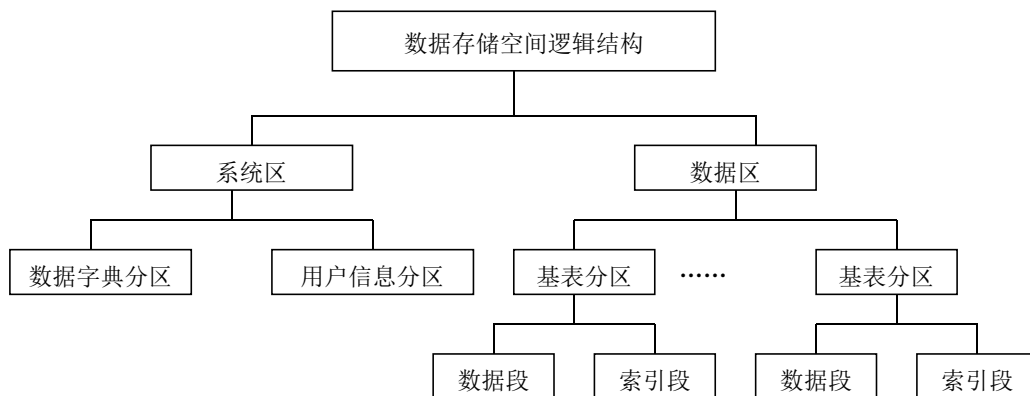


图 7.22 数据库存储空间逻辑结构图

数据字典分区、用户信息分区等，数据区也有若干个分区，每个分区包括一至多个数据库基表，它们只属于有关分区，不能跨分区存放。在数据分区中又自动分为数据段与索引段，其中数据段存放数据元组及相应控制信息，而索引段则存放相应索引信息。图 7.22 给出了

数据库存储空间组织的逻辑结构。

习题七

- 7.1 试述数据库中的 3 种物理存储介质以及它们间的关系。
- 7.2 试说明磁盘存储器结构。
- 7.3 为什么在数据库物理模型中需要引入索引与散列技术？它们在提高存取效率中起什么作用？试解释之。
- 7.4 试分析索引与散列在提高数据库存取效率中的不同之处。
- 7.5 文件有哪几种组织形式？试说明之。
- 7.6 试述文件与数据库间的关系。
- 7.7 试述 B⁺树的结构与特点。
- 7.8 什么叫散列索引？试述它的特点。
- 7.9 试述数据库存储空间的组织。
- 7.10 请你评价数据库物理组织在数据库系统中的地位与重要性。

第七章复习指导

本章主要讨论数据库的底层结构——物理组织，它是数据库中的数据最终的归宿地。数据库的存储空间有效利用及存/取效率的高低均取决于此，这是全书讨论数据库物理结构的唯一的一章，其重要性是可想而知的。学习本章后读者对数据库的内部结构须有一个基本了解。

1. 数据库物理组织的三个层次

- 磁盘层
- 文件层
- 数据库层

2. 磁盘层

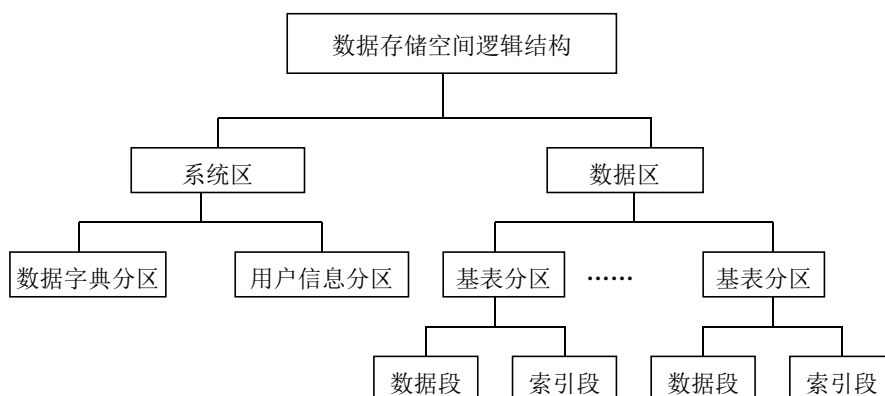
- 磁盘结构
- 磁盘工作原理

3. 文件层

- 文件组织——文件、记录与项
- 文件记录组织——堆文件组织、顺序文件组织、散列文件组织与聚集文件组织
- 索引与散列（特别是 B^+ 树索引）

4. 数据库层

- 数据库中数据分类——数据主体、数据字典、数据间联系信息、存取路径信息及其它信息。



图：数据存储空间逻辑结构图

5. 本章内容重点

- 三个层次结构
- 磁盘结构
- B^+ 树索引

第八章 关系数据库规范化理论

前面已经讨论了关系数据库的一般原理，但是其中有一个不易察觉但又很重要的问题尚未提及，这就是在关系数据库中如何构造合适的数据模式。这是数据库中的一个基本问题，它还涉及到一系列的理论。由于关系模型有较为严格的数学工具作支撑，故一般都以关系模型作为讨论的环境，从而形成了关系数据库的一种理论。由于这种合适的数据模式要符合一定的规范化要求，因而又可称为关系数据库的规范化理论。本章主要就是讨论这个问题。

8.1 概述

为讨论关系数据库规范化理论我们从例子着手，设需设计一个学生数据库 D，它有属性 sno, sn, sd, sa, cno, g, cn, pno。第一个问题是，如何将这八个属性构造造成一些合适的关系模式，从而构成一个关系数据库。构造的方法有很多，表 8.1 仅给出了两种不同的构造方法（当然，还可以构造出更多不同的关系模式来）。

表 8.1 两种关系模式的构造方法

(a)								
SCG:	sno	sn	sd	sa	cno	cn	pno	g

(b)												
S:	sno	sn	sd	sa	SC:	sno	cno	g	C:	cno	cn	pno

第二个问题是，是否随便构造一种关系模式的方案在关系数据库中使用都是一样呢？还以这个例子来说明这个问题，先看用表 8.1(a)所构造的模式建立的一个简单的关系数据库，它可以用表 8.2 表示。

从这个数据库中可以看出它有如下缺点。

(1) 冗余度大：在这个数据库中，一个学生如修读 n 门课，则他的有关信息就要重复 n 遍。如王剑飞这个学生修读 5 门课，在这个数据库中有关他的所有信息就要重复 5 次，这就造成了数据的极大冗余。类似的情况也出现在有关课程的信息中。

(2) 插入异常：在这个数据库中，如果要插入一门课程的信息，但此课程本学期不开设，因此无学生选读，故很难将其存入这个数据库内。这就使这个数据库在功能上产生了极不正常的现象，同时也给用户使用带来极大的不便，这种现象就叫插入异常。

(3) 删除异常：类似有相反情况出现，如在这个数据库中 0003 的方世觉因病退学，因而有关他的元组在数据库中就被删除，但遗憾的是在方世觉的有关情况删除时，连课程 BHD 的有关信息也同时被删除了，并且在整个数据库中再也找不到有关课程 BHD 的有关

信息了。这叫做“城门失火，殃及池鱼”。这也是数据库中的一种极其不正常的现象，同时也会给用户带来极大的不便，这种现象就叫删除异常。

表 8.2 一个关系数据库实例

sno	sn	Sd	sa	cno	cn	pno	g
0001	王剑飞	CS	17	101	ABC	102	5
0001	王剑飞	CS	17	102	ACD	105	5
0001	王剑飞	CS	17	103	BBC	105	4
0001	王剑飞	CS	17	105	AEF	107	3
0001	王剑飞	CS	17	110	BCF	111	4
0002	陈 瑛	MA	19	103	BDE	105	3
0002	陈 瑛	MA	19	105	APC	107	3
0003	方世觉	CS	17	107	BHD	110	4

表 8.3 另一个关系数据库实例

Sno	sn	sd	sa
0001	王剑飞	CS	17
0002	陈瑛	MA	19
0003	方世觉	CS	17

sno	cno	g
0001	101	5
0001	102	5
0001	103	4
0001	105	3
0001	110	4
0002	103	3
0002	105	3
0003	107	4

cno	cn	pno
101	ABC	102
102	ACD	105
103	BBC	105
105	AEF	107
107	BHP	110
110	BCF	111

但是，在表 8.1 (b) 所示的关系模式所构成的关系数据库中，情况完全不同了。表 8.3 给出了它的一个数据库，它存放的数据内容与表 8.2 相同。将这个数据库与前面的相比较就会发现其不同之处：

(1) 冗余度：这个数据库的冗余度大大小于前一个，它仅有小量的冗余。这些冗余都保持在一个合理的水平。

(2) 插入异常：由于将课程、学生及他们所修课程的分数均分离成不同的关系，因此不会产生插入异常的现象。如前面所述的要插入一门课程的信息，只要在关系 C 中增加一个元组即成，而且并不牵涉到学生是否选读的问题。

(3) 删除异常：由于分离成三个关系，故也不会产生删除异常的现象，如前例中由于删除学生信息而引起的将课程信息也一并删除的现象也不会出现了。

从上面这个例子中可以看出，在具有相同数据属性下所构造的不同数据模式方案是有“好”“坏”之分的，有的构造方案既能具有合理的冗余度又能做到无异常现象出现，有的构造方案则冗余度偏大且易产生异常现象。因此，在关系数据库设计中，其关系模式的设计是极其讲究的，必须予以重视。

是什么原因引起异常现象与大量冗余的出现，从而导致关系模式构造弊病的产生呢？这个问题要从语义上着手分析。要构造的数据库中的各属性间是相互关联的，它们互相依赖、互相制约，构成一个结构严密的整体。因此，在构造关系模式方案时，必须从语义上摸清这些关联，将互相依赖密切的属性构成单独的模式，切忌将依赖关系并不紧密的属性“拉郎配”式地硬凑在一起，从而会引起很多“排它”性的反常现象出现。如例子中学生信息的四个属性，它们关系紧密，互相都依赖于 sno，从而构成一个独立的完整结构体系，

而课程及分数等也均有类似现象。这样分解成三个关系模式后一切极不正常现象均会自动消失，这主要就是因为掌握了属性间的内在依赖关系，根据这种内在关系按客观规律办事，因此消除了隐患，从而达到了较为合理的设计方案。

由上面分析知道，要设计一个好的关系模式方案，其根本办法是要掌握属性间的内在语义联系。就目前所知，一般有两种依赖联系，函数依赖与多值依赖。关于这两种依赖的详细情况在后面将会一一介绍。

最后，由前面讨论已经看出，在关系数据库中并不是随便一种关系模式设计方案均是可行的。也不是任何一种关系模式都无所谓的，实际情况是一个关系数据库中的每个关系模式的属性间一定要满足某种内在联系，而这种联系又可对关系的不同要求分为若干个等级，这就叫做关系的规范化(normalization)。

到此为止，可以对前面讨论的内容小结如下：

(1) 在关系数据库设计中，关系模式设计方案是可以有多个的。

(2) 多个关系模式设计方案是有“好”、“坏”之分的，因此，需要重视关系模式的设计，使得所设计的方案是好的或较好的。

(3) 要设计一个好的关系模式方案，关键是要摸清属性间的内在语义联系，特别是依赖联系，依赖联系主要有函数依赖与多值依赖。因此，研究这些依赖关系以及由此而产生的一整套有关理论是关系数据库设计中的重要理论研究。

(4) 一个好的或较好的关系数据库模式设计方案，它的每个关系中属性一定满足某种内在语义条件也就是说要按一定的规范构造关系，这就是关系的规范化。规范化可根据不同的要求而分成若干个级别。因此，一个关系数据库，它的每个关系一定得按规范化要求构造，这样才能得出一个好的数据库。

这就是这章所要讨论的主要问题。其中函数依赖、多值依赖理论以及规范化理论是本章讨论的重点。

8.2 规范化理论

关系数据库中关系规范化问题在 1970 年 Godd 提出关系模型时就同时被提出来，关系规范化可按属性间不同的依赖程度分为第一范式、第二范式、第三范式、Boyce-Codd 范式以及第四范式。人们对规范化的认识是有一个过程的，在 1970 年时已发现属性间的函数依赖关系，从而定义了与函数依赖关系有关的第一、第二、第三，及 Boyce-Codd 范式。在 1976~1978 年间，Fagin, Delobe 以及 Zanjolo 发现了多值依赖关系，从而定义了与多值依赖有关的第四范式。

范式的定义与属性间的依赖关系的发现有密切关系，本节介绍函数依赖与多值依赖这两个概念，并在此基础上定义第一范式、第二范式、第三范式、Boyce-Codd 范式以及第四范式。

8.2.1 函数依赖

函数依赖(functional dependency)是关系模式内属性间最常见的一种依赖关系，例如在关系模式 S 中，sno 与 sd 间有一种依赖关系。即 sno 的值一经确定后 sd 的值也随之惟一地确定了，此时即称 sno 函数决定 sd 或称 sd 函数依赖于 sno，它可用下面符号表示：

sno \rightarrow sd

同样，还可以有：

$$sno \rightarrow sa$$

$$sno \rightarrow sn$$

但是关系模式 SC 中的 sno 与 g 间则没有函数依赖关系，因为一个确定的学号 sno 可以允许有多个成绩(它们分别对应于不同的课程)，因此成绩 g 并不能惟一地确定，但是(sno, cno)与 g 间则存在着函数依赖关系，即有：

$$(sno, cno) \rightarrow g$$

函数依赖这个概念是属语义范畴的，只能根据语义确定属性间是否存在这种依赖，此外别无他法可循。

定义 8.1 设有属性集 U 上的关系模式 R(U)，X, Y 是 U 的子集，若对于任一个关系 R 中的任一元组在 X 中的属性值确定后则在 Y 中的属性值必确定，则称 Y 函数依赖于 X 或称 X 函数决定 Y，并记作 $X \rightarrow Y$ ，而其中 X 称为决定因素，Y 称为依赖因素。

对于函数依赖，一般总是使用一种叫非平凡的函数依赖，在本章中如无特殊声明，凡提到函数依赖时总认为指的是非平凡的函数依赖。下面对非平凡函数依赖下一个定义。

定义 8.2 一个函数依赖关系 $X \rightarrow Y$ 如满足 $Y \not\subseteq X$ ，则称此函数依赖是非平凡的函数依赖。

为了对函数依赖作深入研究，也为了规范化的需要，还得引入几种不同类型的函数依赖。

首先，引入一种完全函数依赖的概念，这个概念为真正的函数依赖打下基础。例如在 S 中我们有 $sno \rightarrow sd$ ，因而同样也会有：

$$(sno, sn) \rightarrow sd$$

$$(sno, sa) \rightarrow sd$$

比较这三种函数依赖后我们会发现，实际上真正起作用的函数依赖是：

$$sno \rightarrow sd$$

而其他两种函数依赖都是由它派生而成的，即是说在函数依赖中真正起作用的是 sno，而不是 sn 或 sa 等。这样，在研究函数依赖时要区别这两种不同类型的函数依赖，前一种叫完全函数依赖，而后一种叫不完全函数依赖。

定义 8.3 R(U) 中如有 X, $Y \subseteq U$ ，满足 $X \rightarrow Y$ ，且对任何 X 的真子集 X' 都有 $X' \not\rightarrow Y$ ，则称 Y 完全函数依赖于 X 并记作： $X \xrightarrow{f} Y$

根据定义 8.3 可知，如果 Y 不函数依赖于 X 的任何真子集，此时的 $X \rightarrow Y$ 才被称为‘完全函数依赖’。

定义 8.4 在 R(U) 中如有 X, $Y \subseteq U$ 且满足 $X \rightarrow Y$ ，但 Y 不完全函数依赖于 X，则称 Y 部分依赖于 X，并记作： $X \xrightarrow{p} Y$

由上所述可知，sd 完全函数依赖于 sno，但 sd 不完全函数依赖于(sno, sn)，亦即有：

$$sno \xrightarrow{f} sd$$

$$(sno, sn) \xrightarrow{p} sd$$

$$(sno, sa) \xrightarrow{p} sd$$

在函数依赖中还要区别直接函数依赖与间接函数依赖这两个不同的概念，例如：

$sno \rightarrow sd$ 中 sd 是直接函数依赖于 sno ，但如果在属性中尚有系的电话号码 dt (假如每个系有惟一的一个电话号码)，则有 $sd \rightarrow dt$ ，从而由 $sno \rightarrow sd$ 及 $sd \rightarrow dt$ 可得到：

$sno \rightarrow dt$

在这个函数依赖中， dt 并不直接函数依赖于 sno ，而是经过中间属性 sd 传递而依赖于 sno ，亦即是 dt 直接依赖于 sd ，而 sd 又直接依赖于 sno ，从而构成了 dt 依赖于 sno 。这种函数依赖关系，是一种间接依赖关系，或叫传递依赖关系。可以对它定义如下。

定义 8.5 在 $R(U)$ 中如有 $X, Y, Z \subseteq U$ 且满足：

$X \rightarrow Y, (Y \not\subseteq X) \ Y \not\rightarrow X, Y \rightarrow Z$

则称 Z 传递函数依赖于 X ；否则，称为非传递函数依赖。

注意 在这里传递函数依赖与非传递函数依赖仅作概念上区别，在形式表示上不作任何区别，即 Z 传递函数依赖于 X 或 Z 非传递函数依赖于 X 都用 $X \rightarrow Z$ 表示，这样做的目的也是为了从全局考虑使得表示尽量简单与方便。

定义了几种不同的函数依赖关系后，在此基础上继续定义一些十分重要的基本概念，即有关键(key)的一些概念。

定义 8.6 在 $R(U)$ 中如有 $K \subseteq U$ 且满足 $K \xrightarrow{f} U$ ，则称 K 为 R 的键。

一个关系模式可以有若干个键，我们在使用中选取其中的一个就够了，这个被选中的键叫做这个关系模式的主键(primary key)，而一般的关键字叫候选键(candidate key)。

在关系模式 S, C, SC 中， S 的键是 sno ， C 的键是 cno ，而 SC 的键是 (sno, cno) ，因为有：

$sno \xrightarrow{f} (sno, sn, sd, sa)$

$cno \xrightarrow{f} (cno, cn, pno)$

$(sno, cno) \xrightarrow{f} (sno, cno, g)$

而 S 中， $(sno, sn), (sno, sd)$ 等均不是键，因为有：

$(sno, sn) \xrightarrow{p} (sno, sn, sd, sa)$

$(sno, sd) \xrightarrow{p} (sno, sn, sd, sa)$

在一个关系模式中，所有键中的属性构成一个集合，而所有其余的属性则构成另一个集合，这两个集合分别叫做这个关系模式的主属性集与非主属性集。主属性集中的属性叫做主属性(prime attribute)，非主属性集中的属性则叫非主属性(non-prime attribute)。例如：

在关系模式 $S(sno, sn, sd, sa)$ 中，主属性集为 $\{sno\}$ ，而非主属性集为 $\{sn, sd, sa\}$

在 $SC(sno, cno, g)$ 中，主属性集为 $\{sno, cno\}$ ，而非主属性为 $\{g\}$

下面我们给出它们的定义。

定义 8.7 $R(U)$ 中所有键中的属性构成的集合 P 称为 $R(U)$ 的主属性集。

定义 8.8 在 $R(U)$ 中所有非键中的属性构成的集合 N 称为 $R(U)$ 的非主属性集。

以上建立了一系列与函数依赖有关的概念，有了它们后就可以讨论与函数依赖有关的范式，它们是第一范式、第二范式、第三范式以及 Boyce-Codd 范式（实际上第一范式与所有依赖均无关，但为叙述方便起见，可视为与函数依赖有关）。至于函数依赖的有关理论的探讨，将在本章稍后部分再作详细介绍。

8.2.2 与函数依赖有关的范式

在本节中讨论 4 种范式，它们是第一范式、第二范式、第三范式以及 Boyce-Codd 范式。先介绍第一范式。第一范式是关系模式所要遵循的基本条件，即关系中的每个属性值均必须是一个不可分割的数据量。如一个关系模式满足此条件则称它属于第一范式(first normal form, 或简写成 1NF)，一个关系模式 R 如满足第一范式，则可记为 $R \in 1NF$ 。

第一范式规定了一个关系中的属性值必须是一个不可分割的数据，它排斥了属性值为元组、数组或某种复合数据等的可能性，使关系数据库中所有关系的属性值均是最简单的，这样可以做到结构简单、讨论方便。一般说来，每个关系模式均要满足第一范式，因为这是对每个关系的最基本要求。

下面开始讨论真正与函数依赖有关的三个范式。为了讨论这几个范式，一般对一个关系模式除了要确定其属性外，还要根据它的语义确定在这个模式上的所有函数依赖。设有关系模式 R ，它有属性集 U ，而在它上的函数依赖集是 F ，则此时一个关系模式可由 R, U, F 确定，它可以写成为：

$R(U, F)$

例如，前面所提到的学生关系模式 S ，它可表示为：

$S(\{sno, sn, sd, sa\}, \{sno \rightarrow sn, sno \rightarrow sd, sno \rightarrow sa\})$

又如，有一个关系模式叫 SCG，它由属性 sno, sn, sd, ss, cno, g 组成，其中 ss 表示学生所学专业，其他含义同前。在这个关系模式中有如下一些语义信息：

- (1) 每个学生均只属一个系与一个专业。
- (2) 每个学生修读之每门课有且仅有一个成绩。
- (3) 各系无相同专业。

根据上述语义信息以及其他的一些基本常识，可以将它们用函数依赖形式表示出来：

$sno \rightarrow sn$

$sno \rightarrow sd$

$sno \rightarrow ss$

$ss \rightarrow sd$

$(sno, cno) \rightarrow g$

因此，这个关系模式的有关信息可写成：

$SCG(\{sno, sn, sd, ss, cno, g\}, \{sno \rightarrow sn, sno \rightarrow sd, sno \rightarrow ss, ss \rightarrow sd, (sno, cno) \rightarrow g\})$

关系模式有了函数依赖后就可以讨论规范化的问题了。关系中的每一级范式均提出了关系模式所要遵循的约束条件，目的是为了使得关系模式具有较少异常性与较小的冗余度，即是说使关系模式更“好”一些。

下面讨论第二范式。

定义 8.9 设有 $R \in 1NF$ 且其每个非主属性完全函数依赖于键，则称 R 满足第二范式(可简称为 2NF)并记作 $R \in 2NF$ 。

实际上并不是每个满足第一范式的关系模式必满足第二式，如前面例子中的关系模式 SCG 即不满足第二范式。这是因在 SCG 中，它的键是 (sno, cno) ，而它的非主属性集是 $\{sd, g, sn, ss\}$ ，虽然有 $(sno, cno) \xrightarrow{f} g$ ，但是 sn, sd, ss 均并不完全依赖 (sno, cno) ，因此不满足第二范式的条件。

一个关系模式若满足第二范式，则它具有较少异常与较小冗余度。因此，一个关系模式若仅满足第一范式还不够，它必须满足第二范式，否则，就要采用分解手段将一个关系模式分解成几个关系模式，使分解后的模式能满足第二范式。

如关系模式 SCG 可分解成两个关系模式：

SCG1 $(\{sno, cno, g\}, \{(sno, cno) \rightarrow g\})$

SCG2 $(\{sno, sn, sd, ss\}, \{sno \rightarrow sn, sno \rightarrow sd, sno \rightarrow ss, ss \rightarrow sd\})$

这两个模式及 SCG 均可用图 8.1 表示。

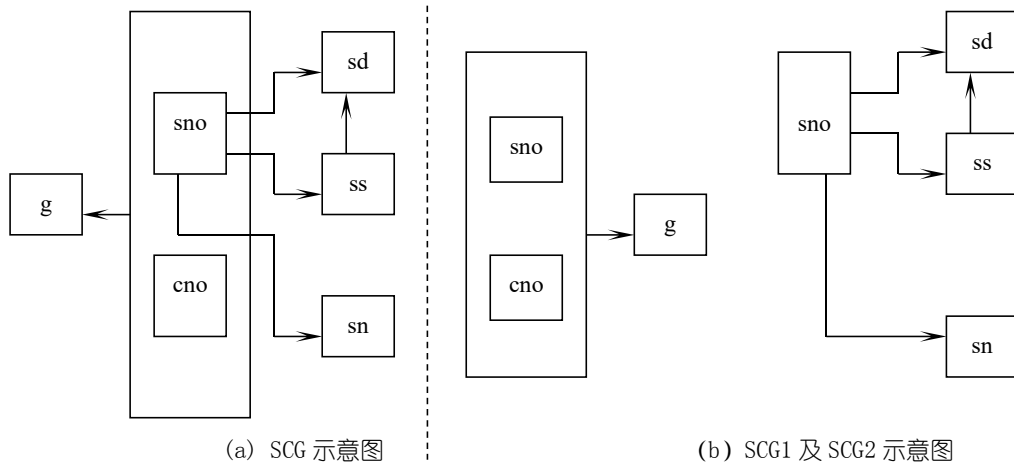


图 8.1 三个关系模式函数依赖示意图

模式 SCG1 与 SCG2 均满足第二范式，它们均有较少异常与较小冗余度，而 SCG1 还可以做到无插入与删除异常的出现，而 SCG 由于不满足第二范式，因此插入异常、删除异常均有存在，且数据冗余度也很大。关于这方面的验证请读者自己去做。

但是，第二范式还不能完全避免异常现象的出现，如 SCG2 虽满足第二范式，但仍会出现插入异常与删除异常。如在 SCG2 中，它有如表 8.4 所示的模式。在这个模式中，如果要登记一个尚未招生的系的专业设置情况，要插入这个情况在模式中是较为困难的。这样，如果要删除一些学生，有可能会将有关系的专业设置情况一起删除。究其原因，不外是因为 sd 既函数依赖于 sno ，又函数依赖于 ss ，同时 ss 又函数依赖于 sno ，并且由此引起了传递函数依赖的出现。因此，看来要消除异常现象，必须使关系模式中无传递函数依赖现象出现，这样就产生了第三范式。

表 8.4 SCG2 的关系模式

SCG2:	sno	sn	sd	ss

第三范式要求关系模式首先得满足第二范式，同时每个非主属性都非传递依赖于键。由此可以看出，如满足第三范式则每个非主属性既不部分依赖也不传递依赖于键。

定义 8.10 若关系模式 R 的每个非主属性都不部分依赖也不传递依赖于键，则称 R 满足第三范式(可简称为 3NF)，并记作 $R \in 3NF$ 。

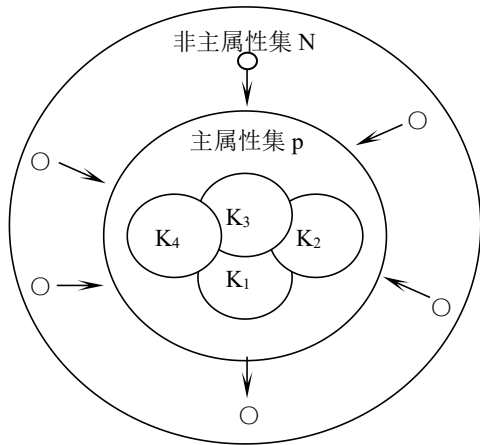


图 8.2 第三范式的“原子”模型

第三范式将关系模式中的属性分成为两类，一类是非主属性集，另一类是主属性集，而非主属性集的每个属性均完全、不传递依赖于主属性集中的键，从而做到在关系模式中理顺了复杂的依赖关系，使依赖单一化与标准化，进而力求达到避免异常性的出现，其示意图如图 8.2 所示，在图中可将关系模式比拟成一个原子，其中主属性集是这个原子的原子核，而非主属性集中的属性则是这个原子中的电子，它们紧紧依赖于主属性集构成一个紧密整体。

一个关系模式如果不满足第三范式，可以通过模式分解使其分解成若干个模式，使分解后的模式能满足第三范式。例如关系模式 SCG2

中，SCG2 满足第二范式，但不满足第三范式，此时可将其分解成下面两个模式：

SCG21(sno, sn, ss)

SCG22(ss, sd)

其依赖示意图如图 8.3 所示。

在 SCG 中经过几次分解后，得到三个关系模式：

SCG1, SCG21, SCG22

这三个模式均满足第三范式且没有异常现象出现，同时冗余度小。

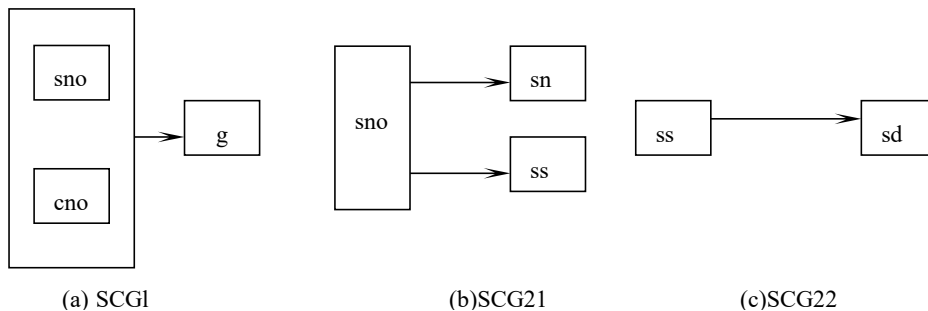


图 8.3 模式分解图

1972 年 Boyce 和 Codd 人等从另一个角度研究了范式，发现了函数依赖中的决定因素与键间的联系与范式有关，从而创立了另一种第三范式，称为 Boyce-Codd 范式。

Boyce-Codd 范式的大概意思是，如果关系模式中，每个决定因素都是键，则满足

Boyce-Codd 范式。一般而言，每个函数依赖中的决定因素不一定是键，因此只有当 R 中决定因素都是键时才能认为满足 Boyce-Codd 范式。

定义 8.11 如 R 中 $X, Y \subseteq U$ ，假定满足 $R \in 1NF$ ，且若 $X \rightarrow Y (Y \subseteq X)$ 时 X 必含键，则称 R 满足 Boyce-Codd 范式（可简记 BCNF），并记作 $R \in BCNF$ 。

下面一个问题需要研究 BCNF 与 3NF 间究竟有什么关系？经过仔细研究后，人们发现 BCNF 比 3NF 更为严格。下面的定理给出了这个回答。

定理 8.1 关系模式 R 若满足 BCNF，则必定满足 3NF。

这个定理的证明请读者设法自行证明（注：可以用 BCNF 及 3NF 的定义证得）。

这个定理告诉我们，一关系模式满足 BCNF 必满足 3NF。但是，一关系模式满足 3NF 是否满足 BCNF 呢？即是说，定理 8.1 的充分条件是否成立呢？回答是否定的，即必存在 R 满足 3NF，但不满足 BCNF，这只要用一例即可说明。

例 8.1 设有关系模式 $R(sn, cn, tn)$ ，其中 sn, cn 表示教师与课程同前，tn 表示教师，R 有下列语义信息：

- (1) 每个教师仅上一门课。
- (2) 学生与课程确定后，教师即唯一确定。

这样，R 就有如下函数依赖关系：

$(sn, cn) \rightarrow tn$

$tn \rightarrow cn$

这个关系模式满足 3NF，因为它的主属性集为 $\{sn, cn, tn\}$ ，非主属性集为 $\{\}$ （空集），不存在非主属性对关键字的部分依赖和传递依赖。但这个关系模式不满足 BCNF，因为在函数依赖 $tn \rightarrow cn$ 中，tn 是决定因素但 tn 不是键，不符合 BCNF 的定义。这个模式的示意图如图 8.4 所示。

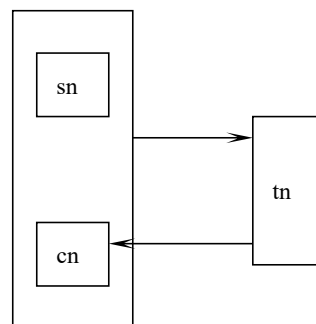


图 8.4 例 8.1 示意图

从这个例子中也可以看出，实际上第三范式也避免不了异常性，如某课程本学期不开设，因此就无学生选读，此时有关教师固定开设某课程的信息就无法表示。因此，要避免此种异常性，还需要进一步将关系模式分解成 BCNF。如在此例中可将 R 进一步分解成：

$R_1 (sn, tn)$

$R_2 (tn, cn)$

其示意图如图 8.5 所示。而 R_1, R_2 则为 BCNF，这两个模式均不会产生异常现象。



图 8.5 R 分解成两个符合 BCNF 的关系

从上面所述可以看出，BCNF 比 3NF 更为严格，它将关系模式中的属性分成两类，一类是决定因素集，另一类是非决定因素集。非决定因素集中的属性均完全、不传递地依赖于决定因素集中的每个决定因素。关于这种比喻的一个示意图如图 8.6 所示。

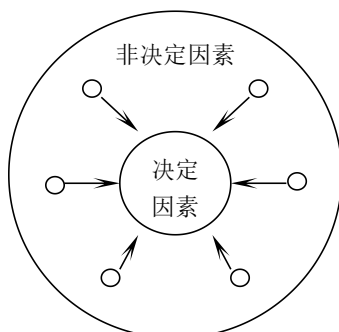


图8.6 BCNF的原子模型

到此为止，由函数依赖所引起的异常现象，只要分解成 BCNF 即可获得解决。在 BCNF 中每个关系模式内部的函数依赖均比较单一和有规则，它们紧密依赖而构成一个整体，从而可以避免异常现象以及冗余度过大的现象出现。

8.2.3 多值依赖与第四范式

前面研究了函数依赖及与它有关的几种范式，但是否关系模式内属性间的依赖关系除函数依赖外就没有其他依赖关系了呢？事实并不如此，函数依赖关系是一种较为常见的、明显的依赖关系，但是随着人们对关系模式了解越来越深刻后，发现尚有另外的一些依赖关系存在，多值依赖就是其中的一种。先举一个例子，以说明多值依赖的存在。

例 8.2 设有一个课程关系 R，它可用表 8.5 表示。此表表示高等数学这门课的任课教师可以有三个，它的参考书可以有两本；普通物理这门课的任课教师也可以有三个，它的参考书可以有三本。如用关系的形式表示，如表 8.6 所示。

表 8.5 关系 R 的示意图

课程名 C	教师名 T	选用参考书 L
高等数学	李华民	高等数学
	王天华	高等数学教程
	林 静	
普通物理	吴铁钢	物理学
	谢晓芳	普通物理
	徐秋芳	普通物理基础

从这个关系中可以看出：

- (1) 这个关系的数据冗余很大。
- (2) 这个关系的属性间有一种有别于函数依赖的依赖关系存在。

仔细分析这种特殊依赖关系后发现它有两个特点：

(1) 设如 $R(U)$ 中 X 与 Y 有这种依赖关系，则当 X 的值一经确定后可以有一组 Y 值与之对应。如确定 C 为高等数学，则有一组 T 的值：李华民、王天华、林静与之对应。同样 C 与 L 也有类似的依赖。

(2) 当 X 的值一经确定后，其所对应的一组 Y 值与 $U-X-Y$ 无关。如在 C 中，对应高等数学课的一组教师与此课程的参考书毫无关系，这就表示 C 与 T 有这种依赖，则 T 的值

的确定与 $U-C-T=L$ 无关。

上述这种依赖显然不是函数依赖，我们称之为多值依赖(multi-valued dependency)，如 Y 多值依赖于 X ，则可记为 $X \twoheadrightarrow Y$ 。

表 8.6 关系 R

C	T	L
高等数学	李华民	高等数学
高等数学	李华民	高等数学教程
高等数学	王天华	高等数学
高等数学	王天华	高等数学教程
高等数学	林 静	高等数学
高等数学	林 静	高等数学教程
普通物理	吴铁钢	物理学
普通物理	吴铁钢	普通物理
普通物理	吴铁钢	普通物理基础
普通物理	谢晓芳	物理学
普通物理	谢晓芳	普通物理
普通物理	谢晓芳	普通物理基础
普通物理	徐秋芳	物理学
普通物理	徐秋芳	普通物理
普通物理	徐秋芳	普通物理基础

从上面所描述多值依赖 $X \twoheadrightarrow Y$ 的特点看，其第一个特点表示 X 与 Y 的对应关系是很随便的， X 的一个值所对应的 Y 值的个数可不作任何强制性规定，即 Y 的值可以从 0 到任意多个，其主要起强制性约束的是第二个条件，即 X 所对应的 Y 取值与 $U-X-Y$ 无关，说得确切些，如有 R 且如存在 $X \twoheadrightarrow Y$ ，则对 R 的任何一个关系 R ，如有元组 $s, t \in R$ ，且有 $s[X]=t[X]$ (表示 s 与 t 在 X 的投影相等)，如将它们在 $U-X-Y$ 的投影(记为 $s[U-X-Y]$ ， $t[U-X-Y]$)，交换后所得元组称为 u, v ，则必有 $u, v \in R$ 。关于这个情况可以用表 8.7 表示。

表 8.7 多值依赖示意图

	X	Y	U-X-Y	
R:	s	s [X]	s [Y]	s[U-X-Y]
	t	t [X]	t [Y]	t[U-X-Y]
	u	s [X]	s [Y]	t[U-X-Y]
	v	t [X]	t [Y]	s[U-X-Y]
.....	

对多值依赖有了充分了解后，可对它定义如下：

定义 8.12 设 R 中有 $X, Y \subseteq U$ ，若对 R 的任何一个关系，对 X 的一个确定值，存在 Y 的一组值与之对应，且 Y 的这组值又与 $Z=U-X-Y$ 中的属性值不相关，此时称 Y 多值依赖于 X ，并记为 $X \twoheadrightarrow Y$ 。

在多值依赖中若 $X \twoheadrightarrow Y$ 且 $Z = U - X - Y \neq \emptyset$, 则称 $X \twoheadrightarrow Y$ 为非平凡多值依赖, 否则称为平凡多值依赖。

多值依赖有下面的一些性质:

- (1) 在 R 中如有 $X \twoheadrightarrow Y$, 则必有 $X \twoheadrightarrow U - X - Y$ 。
- (2) 在 R 中如有 $X \twoheadrightarrow Y$, 则必有 $X \twoheadrightarrow Y$ 。

请注意, 我们在 R 中讨论多值依赖时并不意味着 R 中已不需要讨论函数依赖了, 恰恰相反, 我们一般不仅要在 R 找出所有多值依赖关系来, 而且还要找出所有的函数依赖关系来。因此, 一个完整的 R 应该包含一个函数依赖集 F 以及一个多值依赖集 F' , 它可以用 $R(U, F, F')$ 表示。

前面已经讲过, 具有多值依赖的关系, 它们的数据冗余度特别大, 如何设法减少数据冗余呢? 从例 8.2 中的关系 R 中可以看出, 如果将 $R(C, T, L)$ 分解成两个关系 $R_1(C, T), R_2(C, L)$ 后, 它们的冗余度会明显下降。 R_1, R_2 这两个关系可用表 8.8 表示。

表 8.8 关系 R 分解成关系 R_1 和 R_2

C	T	C	L
高等数学	李华民	高等数学	高等数学
高等数学	王天华	高等数学	高等数学教程
高等数学	林 静	普通物理	物理学
普通物理	吴铁钢	普通物理	普通物理
普通物理	谢晓芳	普通物理	普通物理基础
普通物理	徐秋芳	普通物理	

(a) 关系 R_1

(b) 关系 R_2

从表 8.8 可以看到, 数据冗余的减少是极其明显的。

从多值依赖的观点看, 在 R_1, R_2 中各对应一个多值依赖 $C \twoheadrightarrow T$ 与 $C \twoheadrightarrow L$, 它们都是平凡多值依赖。因此, 在多值依赖时, 减少数据冗余的方法是使关系分解为仅有平凡多值依赖。

这样, 就可以规定一个比 BCNF 更高的范式, 它叫第四范式, 可简记为 4NF。这个范式的特点是, 在关系模式中它必须满足:

- (1) 只允许出现平凡多值依赖(不允许出现非平凡多值依赖)。
- (2) 函数依赖要满足 BCNF。

由于函数依赖是多值依赖的特例, 因此可统一用多值依赖概念定义第四范式。

定义 8.13 R 中如果 $X \twoheadrightarrow Y$ 是非平凡多值依赖, 则 X 必含有键, 此时称 R 满足第四范式, 并记作 $R \in 4NF$ 。

由这个第四范式定义可以看出, 前面所定义的关系 R , 它虽是 BCNF, 但不是 4NF, 因为在 $R(C, T, L)$ 中有:

$$C \twoheadrightarrow T$$

$$C \twoheadrightarrow L$$

而它的键是 (C, T, L) 。

虽然 $R \in BCNF$, 但 C 不是键, 所以 $R \notin 4NF$ 。对它作分解后所产生的 R_1 及 R_2 显然因为 $R_1(C, T)$ 有 $C \twoheadrightarrow T$, 故不存在非平凡多值依赖, 因此有 $R_1 \in 4NF$, 同理有 $R_2 \in 4NF$ 。

8.2.4 小结

在规范化讨论中定义了 5 个范式，对这些范式的认识是逐步深入的。总的说来，可以总结成下面几点。

- (1) 规范化的目的：解决插入、删除及修改异常以及数据冗余度高的问题。
- (2) 规范化的方法：从模式中各属性间的依赖关系(函数依依赖及多值依赖)入手，尽量做到每个模式表示客观世界中的一个“事件”。
- (3) 规范化的实现手段：用模式分解的方法。

实际上从第一范式到第四范式的过程是一个不断消除一些依赖关系中弊病的过程。图 8.7 给出了这个过程。

所要注意的是规范化是一种理论，它研究如何通过规范以解决异常与冗余现象，在实际数据库设计中构造关系模式时需要考虑到这个因素。但是，客观世界是复杂的，在构造模式时尚需考虑到其他的多种因素，如模式分解过多，势必在数据查询时要用到较多的联接运算，这样就会影响查询速度。因此，在实际构造模式中，需要综合多种正反因素，统一权衡利弊得失，最后构做出一个较为适合实际的模式来。

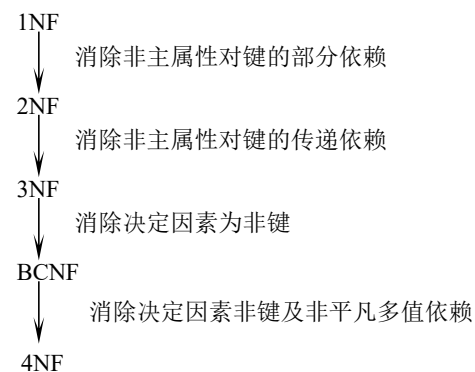


图 8.7 规范化的过程

8.3 规范化所引起的一些问题

由规范化而引起了对一些问题的进一步研究。

1. 函数依赖理论的研究

属性间的函数依赖与多值依赖是规范化的基本依据，因此有必要对它们作进一步研究，这些研究包括：

(1) 可由关系模式上的一些函数依赖通过公理系统（称 **Armstrong 公理**）而获得关系模式上的所有函数依赖。由此可知，一个关系模式上的所有函数依赖可由两部分组成：基础部分是直接由语义获取，其他部分可由公理系统推演而得。

(2) 引入了函数依赖集的等价概念与最小函数依赖集，即如果两函数依赖集能推演出相同的集来，则称它们是等价的，而等价的函数依赖集之最小者称为最小函数依赖集。

这些研究为规范化提供了更多的基础信息。

2. 模式分解的研究

规范化的实施主要依靠不断地进行模式分解。在模式分解中需要研究下列问题：

(1) 分解后关系中的数据是否会丢失？这叫无损连接 (lossless join)，亦即是说分解后的关系再经连接后能恢复到原来的关系。

(2) 分解后关系中的函数依赖是否会丢失？这叫依赖保持 (preserve dependency)。

(3) 在满足无损联接性与依赖保持性下可分解到第几范式

经过研究，可以得到下面几个事实：

(1) 若要求满足无损联接性，则模式分解一定可以达到 BCNF。

(2) 若要求满足依赖保持性，则模式分解一定可以达到 3NF，但不一定能达到 BCNF。

(3) 若既要求满足无损联接性又要求满足依赖保持性，则模式分解一定可以达到 3NF，但不一定能达到 BCNF。

上述三点均可通过三个算法获得实现。

一般在做模式分解时尽量要求能保持函数依赖与无损连接。

由于规范化所引起的这两个问题的研究的详细探讨均比较复杂，故本书中不拟详述，仅将结果陈述于上，供读者参考。

8.4 关系数据库规范化的非形式化判别法

关系数据库规范化在数据库设计及数据库应用中有重要的作用，规范化理论从理论上对关系数据库给予严格的规范与界定这是极端必要的，但是在实际应用中由于理论的抽象性与复杂性，因此很难具体界定，为方便应用，在本节中给出常用范式的非形式化判别方法以供参考。

一般而言，一个关系模式至少需满足 3NF，因此 3NF 成为鉴别关系模式是否合理的最基本条件，在本节中我们介绍判别 3NF 的非形式化方法，这个方法称为“一事一地” (“one foct one place”) 原则。即一件事放一张表而不同事则放不同表的原则。前面的学生数据库中 学生 (S)、课程 (C) 与学生课程 (SC) 是不相干的三件事，因此必需放在三张不同表中，这样所构成的模式必满足 3NF，而任何其中两张表的组合必不满足 3NF。

“一事一地”原则是判别关系模式满足 3NF 的有效方法，此种方法既非形式化又较为简单，因此在数据库设计中被经常使用，所唯一要注意的是，此种方法要求对所关注的数据体的语义要清楚了解，具体的说即要对数据体中的不同“事”能严格区分，这样才能将其放入不同“表”中。

习题八

8.1 请给出下列术语的含义：

- (1) 函数依赖
- (2) 完全函数依赖
- (3) 传递函数依赖
- (4) 键
- (5) 主属性集
- (6) 决定因素
- (7) 多值依赖
- (8) 1NF

- (9) 2NF
- (10) 3NF
- (11) BCNF
- (12) 4NF。

8.2 在关系 SC (sno, cno, g) 中 $sno \twoheadrightarrow cno$ 正确吗? 请说明其理由。

8.3 是不是规范化最佳的模式结构是最好的结构? 为什么?

8.4 试证明若 $R \in BCNF$, 则必有 $R \in 3NF$ 。

8.5 如何用非形式化判别 3NF, 并请用一例说明之。

8.6 试问下列关系模式最高属第几范式, 并解释其原因。

- (1) $R(A, B, C, D)$, $F: \{B \rightarrow D, AB \rightarrow C\}$;
- (2) $R(A, B, C)$, $F: \{A \rightarrow B, B \rightarrow A, A \rightarrow C\}$;
- (3) $R(A, B, C, D)$, $F: \{A \rightarrow C, D \rightarrow B\}$;
- (4) $R(A, B, C, D)$, $F: \{A \rightarrow C, CD \rightarrow B\}$ 。

8.7 设有一关系模式 $R(A, B, C, D, E, F)$ 其函数依赖为: $\{A \rightarrow C, (A, B) \rightarrow D, C \rightarrow E, D \rightarrow (B, F)\}$, 请按下述每题要求进行模式分解, 并给出每次分解后的结果关系模式上存在的函数依赖。

- (1) 给出模式 R 上的候选关键字;
- (2) 将 R 分解到满足 2NF;
- (3) 将上小题的结果分解到满足 3NF;
- (4) 将上小题的结果分解到满足 BCNF。

8.8 设有关系模式 $R(A, B, C, D, E, F)$, 其函数依赖为: $\{(A, B) \rightarrow C, C \rightarrow D, (C, E) \rightarrow F\}$

- (1) 给出模式 R 的候选键;
- (2) 将 R 分解到满足 3NF;

8.9 设有关系模式 $R(A, B, C, D, E)$, 其函数依赖为: $\{A \rightarrow B, B \rightarrow E, (A, C) \rightarrow D\}$

- (1) 给出 R 的候选键;
- (2) 将 R 分解到 2NF;
- (3) 进一步将其分解到 3NF。

8.10 有一项目管理系统, 它包括职工的工作证号, 姓名, 身份证号; 项目的项目编号, 名称, 实施地点; 职工参与某项目的工作时间。其中, 职工的工作证号及身份证号; 项目的项目编号具有唯一性; 每个项目有若干实施地点, 在每个实施地点上又可以同时开展多个项目; 每个职工可以参与多个项目, 每个项目有多个职工参加; 在职工与项目确定后, 则职工参与项目的实施地点也就唯一确定, 但每个职工在每个实施地点上只能参加其中一个项目。由工作证号、姓名、身份证号、项目编号、项目名称、实施地点和工作时间等七个属性构成关系模式 R。

- 1. 给出 R 的函数依赖集合及其候选键;
- 2. 将 R 分解到满足 3NF。

8.11 什么叫模式分解? 它与规范化有什么关系? 请说明之。

8.12 什么叫模式分解中关系的无损连接与依赖保持? 请说明之。

第八章复习指导

本章讨论关系表的规范化结构，它既有理论意义又有实际价值，它是数据库设计的基础，又是数据库理论的一部分。读者学习此章后对关系规范化的理论与实际均应有所了解。

1. 关系模式规范化讨论的三个层次

关系模式规范化讨论的三个层次，它们是：

(1) 语义层：从模式中属性间的语义建立两种语义关系：

- 函数依赖关系
- 多值依赖关系

(2) 规范层：按语义分成五种范式

(3) 实现层

2. 语义层

(1) 函数依赖

- 函数依赖基本概念
- 二种函数依赖——完全函数依赖、传递函数依赖
- 键
- 决定因素

(2) 多值依赖

- 多值依赖基本概念
- 多值依赖性质

3. 规范层

按语义分成五种范式：

- 1NF——基本范式
- 2NF——与完全函数依赖有关范式
- 3NF——与传递函数依赖有关范式
- BCNF——与决定因素有关范式
- 4NF——与多值依赖有关范式

4. 实现层

- 模式分解
- 非形式化判别法

5. 本章内容重点

- 函数依赖与多值依赖
- 范式

第九章 数据库设计

本章主要讨论数据库的设计，设计的重点是数据库的需求分析、概念设计、逻辑设计及物理设计等四个阶段。数据库设计是数据库应用的一个重要环节，本章内容对开发数据库的应用十分重要。

9.1 数据库设计概述

在数据库应用系统中的一个核心问题就是设计一个符合环境要求又能满足用户需求、性能良好的数据库，这就是数据库设计（database design）的主要任务。

数据库设计又称数据库分析与设计，它的基本依据是用户对象的数据需求、处理需求和数据库的支持环境（包括硬件、操作系统与 DBMS）。所谓数据需求是指用户对象的数据及其结构，它反映了数据库的静态要求；所谓处理需求表示用户对象的数据处理过程和方式，反映了数据库的动态要求；以此两者为基础作设计，其最终的结果是数据模式（包括概念模式、逻辑模式与物理模式）。数据库设计中有一定的制约条件，它们即是系统平台，包括系统软件，工具软件以及设备、网络等软、硬件平台。因此数据库设计即是在一定平台制约下，根据数据需求与处理需求设计出性能良好的数据模式。

数据库设计是一个“工程”问题，它是软件工程（software engineering）的一个部分。在软件工程中将软件开发过程称为软件生存周期（life cycle），它一般分为需求分析、设计、编码、测试、运行和维护等阶段。在软件工程中这六个阶段一般讲是顺序执行的，这种方式称瀑布（water falling）模型，而每个阶段执行结束均有一个标志性结果称里程碑（mile stone）。软件工程中数据部分的分析、设计与开发称数据工程（data engineering），它也采用软件工程中生存周期方法，它将整个数据库应用开发分解成目标独立的若干阶段：

- (1) 需求分析阶段。
- (2) 概念设计阶段。
- (3) 逻辑设计阶段。
- (4) 物理设计阶段。
- (5) 编码阶段。
- (6) 测试阶段。
- (7) 运行阶段。
- (8) 维护阶段。

而数据库设计则是数据工程的前期设计部分，即采用上面八个阶段中的前四个阶段，并且重点以数据结构与模式的设计为主线，它可用图 9.1 表示。

在这四个阶段中每个阶段结束都有一个里程碑，它们分别是需求分析说明书、概念设计说明书、逻辑设计说明书以及物理设计说明书，而在逻辑设计中需附加 DBMS 模型限制，在物理设计中则需附加网络、硬件及系统软件平台的限制。

本章按这四个阶段分别作介绍，并主要介绍关系模式的设计。

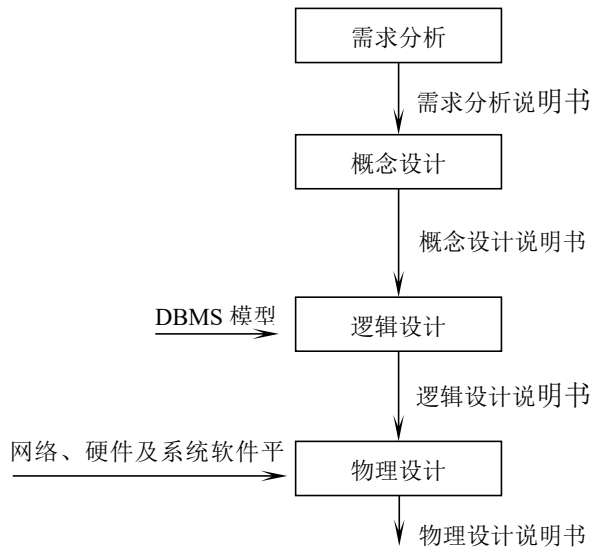


图 9.1 数据库设计的四个阶段

9.2 数据库设计的需求分析

在数据库设计的整个过程中需求分析是基础，需求分析的好坏直接影响到最终数据模式，需求分析从调查用户着手，深入了解用户单位数据流程，数据使用情况，数据的数量、流量、流向、数据性质，并作出分析，最终按一定规范要求以文档形式写出数据的需求说明书，其大致结构可用图 9.2 表示。

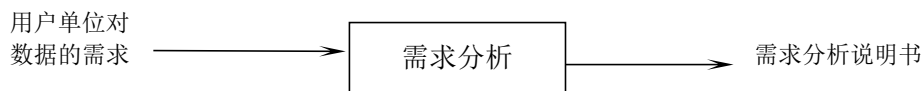


图 9.2 需求分析结构图

9.2.1 需求调查

需求调查是需求分析的第一步，在此步中调查者须收集用户单位的有关资料，这些资料包括报表、台账、单据、文档、档案、发票、收据等原始资料，此外还包括组织机构及业务活动。其次，还需召开座谈会，了解有关数据需求的情况，在特殊情况下需作个别调查与专题调查，并作出记录。

9.2.2 需求分析

在需求调查基础上对所有资料作分析，并着重从“数据”与“处理”两方面入手，而重点以数据为核心作分析。

1. 数据边界的确定

确定整个需求的数据范围，了解系统所需要的数据范围以及不属系统考虑的数据范围，用此以建立整个系统的数据边界。

数据边界确立了整个系统所注视的目标与对象，建立了整个数据领域所考虑范围。

2. 绘制数据流图

在数据边界确定后首先是要分析用户活动，在此基础上着重分析其数据流（包括数据

流向及加工), 进而绘制出数据流图 FDF (dataflow diagram), DFD 有四个基本成份, 它们是:

- 数据流向: \longrightarrow (箭头)
- 数据处理: \bigcirc (圆形)
- 数据存储: \equiv (双线段)
- 数据端点 (包括源点与终点): \square (矩形)

数据流图是由“数据源端点”到“数据终端点”的一种单向流程图, 它由“数据流向”引导并以“数据处理”与“数据存储”为加工结点所构成的一种图, 该图反映了用户单位抽象的数据动态模型。

下面的图 9.3 是一个简单的学生考试成绩批改与发送 DFD 示例图。

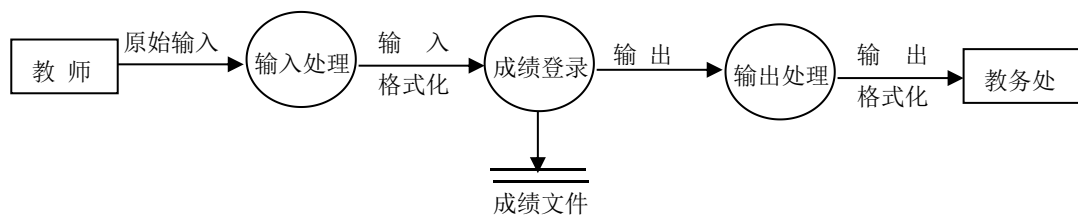


图 9.3 一个 DFD 示例图

在图 9.3 中试卷由教师批改后经格式化处理将成绩登录入成绩文件, 以后统一输出, 并经格式化处理后发送至教务处。

3. 数据字典

在 DFD 基础上构作数据字典, 数据字典给出了用户单位的数据基本需求。数据字典包括数据项、数据结构、数据流、数据存储及数据处理等五个部分, 它们分别是:

(1) 数据项

数据项是数据基本单位, 它包括如下内容:

- 数据项名;
- 数据项说明;
- 数据类型;
- 长度;
- 取值范围;
- 语义约束——说明其语义上的限制条件包括完整性、安全性限制条件;
- 与其它项的关联。

(2) 数据结构

数据结构由数据项组成, 它给出了数据基本结构单位, 如数据记录即是一种数据结构, 它包括如下内容:

-
- 数据结构名;
 - 数据结构说明;
 - 数据结构组成: {数据项/数据结构};
 - 数据结构约束: 从结构角度说明语义上的限制, 包括完整性及安全性限制条件。

(3) 数据流

数据流是数据结构在系统内流通过程, 它包括如下内容:

- 数据流名;
- 数据流说明;
- 数据流来源;
- 数据流去向;
- 组成: {数据结构};
- 平均流量;
- 高峰流量。

(4) 数据存贮

数据存贮是数据结构保存或停留之处, 也是数据来源与去向之一, 它包括如下内容:

- 数据存储名;
- 数据存储说明;
- 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。

(5) 数据处理

数据处理给出处理的说明信息, 它包括如下内容:

- 数据处理名;
- 数据处理说明;
- 输入数据: {数据结构};
- 输出数据: {数据结构};
- 处理: {简要说明}。

9.2.3 数据需求分析说明书

在调查与分析的基础上依据一定的规范要求编写数据需求分析说明书。数据需求分析说明书一般是依据一定规范要求编写的, 目前我国有国家标准与部委标准, 也有各企业标准, 其制定的目的是为了规范说明书的编写, 规范需求分析的内容, 同时也为了统一编写形式。

数据需求分析说明书的主要内容包括需求调查内容与分析内容。下面的图 9.4 给出需求分析说明书的一个模板, 供编写时参考。在这个模板中有几处需要说明:

- (1) 在“编写目的”中主要说明编写此报告的意图与用意, 并指明此文档的预期读者;

- (2) 在“背景”中包括如下内容：
- 所开发的项目名称
 - 项目任务提出者、开发者、用户及实现该项目的单位。
- (3) 在名词“定义”中需对报告中所用到的专门术语予以介绍，必要时还须注明其英语原文。
- (4) 在“参考资料”中需列出与本报告有关的参考资料名称、出版单位及出版时间等。

×××项目需求分析说明书	
1. 前言	
1.1 编写目的	
1.2 背景	
1.3 名词定义	
1.4 参考资料	
2. 数据边界分析	
2.1 数据范围	
2.2 数据内部关系分析	
2.3 数据边界分析	
2.4 数据环境分析	
3. 数据流	
3.1 数据流图之 1	
.....	
3. m.....m	
4. 数据字典	
4.1 数据项分析	
4.2 数据结构分析	
4.3 数据流分析	
4.4 数据存贮分析	
4.5 数据处理分析	
5. 原始资料汇编	
5.1 原始资料汇编之 1	
.....	
5. nn	
编写人员： _____	审核人员： _____
审批人员： _____	日 期： _____

图 9.4 需求分析说明书模板

9.3 数据库的概念设计

9.3.1 数据库概念设计概述

数据库概念设计是在数据需求分析基础上进行的，其目的是分析数据间的内在语义关联，在此基础上建立一个数据的抽象模型。数据库概念设计所使用的方法很多，有 E-R 法、EE-R 法以及面向对象方法等，目前常用的方法是 EE-R 方法。在本节中采用 EE-R 方法，它有四种基本概念，必须将其区分出来，它们是属性、实体集、联系与继承。属性与实体是基本对象，而联系与继承则是属性与实体间的语义关联。

一个部门或单位有大有小，有简单有复杂，其内在逻辑关系与语义关联可以非常复杂，如何在需求调查基础上，设计出一个数据概念模型一般有两种方法。

1. 集中式模式设计法

这是一种统一的模式设计方法，它根据需要由一个统一机构设计一个综合的全局模式，这种方法设计简单方便，它强调统一与一致，它适用于小型或不复杂的单位或部门。

2. 视图集成设计法

这种方法是将一个单位分解成若干个部分，先对每个部分作局部模式设计，建立各个部分的视图(此中所指的视图不是第三章中所指的那种导出表，而表示局部模式的结果图)，然后以各视图为基础进行集成，在集成过程中可能会出现一些冲突，这是由于视图设计的分散性所成的，因此需对视图作修正，最终形成全局模式。视图集成设计法是一种由分散到集中的方法，它的设计过程复杂但它能较好的反映需求，适合于大型与复杂的单位，避免设计的粗糙与不周到。在本章中我们主要介绍此种方法。

9.3.2 数据库概念设计的过程

在本节中采用 EE-R 方法与视图集成法进行设计，其具体步骤如下。

1. 分解

首先对用户单位作分解，分解成若干个具有一定独立逻辑功能的用户组，并针对该用户组的需求分析作视图设计。用户组不宜太大，其实体保持在“7±2”个为原则。

2. 视图设计

视图设计一般有三种设计次序：

(1) 自顶向下：先从抽象级别高且普遍性强的对象开始逐步细化、具体化与特殊化，如学生这个视图可先从一般学生开始，再分成大学生、研究生等，进一步再由大学生细化为大学本科与专科，研究生细化为硕士生与博士生等，还可以再细化成学生姓名、年龄、专业等细节。

(2) 由底向上：先从具体的对象开始，逐步抽象，普遍化与一般化，最后形成一个完整的视图设计。

(3) 由内向外：先从最基本与最明显的对象着手逐步扩充至非基本、不明显的其他对象，如学生视图可从最基本的学生开始逐步扩展至学生所读的课程，上课的教室与任课的教师等其他对象。

上面三种方法为构作视图设计提供了具体的操作方式，它们可单独使用也可混合使用。在视图设计中须作三方面的设计。

(1) 实体与属性设计:

1) 如何区分实体与属性: 实体与属性是视图中的基本单位, 它们间无明确区分标准, 一般讲可以有下面的一些原则作分析时参考。

- 描述信息原则: 一般讲实体需有进一步的性质描述, 而属性则无。
- 依赖性原则: 一般讲属性仅单向依赖于某个实体, 且此种依赖是包含性依赖, 如学生实体中的学号、学生姓名等均单向依赖于学生。
- 一致性原则: 一实体由若干个属性组成, 这些属性间有内在的关联性与一致性, 如学生实体有学号、学生姓名、年龄、专业等属性, 它们分别独立表示实体的某种独特个性, 并在总体上协调一致, 互相配合, 构成了一个完整的整体。

上述三个原则仅供分析时参考, 在具体操作时还要灵活掌握使用。

2) 实体与属性的描述: 在确定了实体与属性后需对下述几个问题作详细描述。

- 实体与属性名: 实体与属性的命名须有一定原则, 它们应清晰明了便于记忆, 并尽可能采用用户所熟悉的名字, 名字要有特点, 减少冲突, 方便使用, 并要遵守缩写规则。
- 确定实体标识: 实体标识即是该实体的主键, 首先要列出实体的所有候选键, 在此基础上选择一个作为主键。
- 非空值原则: 在属性中可能会出现空值, 这并不奇怪, 重要的是在主键中不允许出现有空值。

(2) 联系与继承设计

1) 区分联系与继承。联系与继承建立了视图中属性与实体间的语义关联, 从理论上讲两者语义是很清楚的。

- 继承: 实体间的分类关系与包含关系;
- 联系: 实体间的一种广泛语义联系, 它反映了实体间的内在逻辑关联。

2) 联系的详细描述。

• 联系的种类很多, 大致有三种, 存在性联系, 如学校有教师、教师有学生等; 功能性联系, 如教师授课, 教师参与管理学生等; 事件联系, 如学生借书, 学生打网球等。用上面三种可以检查需求中联系是否有遗漏。

- 实体间联系的对应关系有 1:1, 1:n, n:m 等三种。
- 实体间联系的元数: 实体间联系常用的是两个实体间的联系称二元联系, 偶尔也会用到三个及三个以上联系, 称多元联系, 特殊情况是一个实体内部的联系称一元联系。

3) 继承的详细描述。

• 部分继承与全继承: 如果子实体继承超实体全部属性, 此种继承称全继承, 而有时子实体仅继承超实体的部分属性, 此种继承称部分继承。

• 单继承与多继承: 一个实体仅有一个超实体, 此种继承称单继承, 而如果一个实体有多于一个超实体, 此种继承称多继承。

根据上述一些原则, 可以对用户组作视图设计, 下面给出几个视图设计的例子。

例 9.1 学校教务处关于学生的视图可用图 9.5 表示。

例 9.2 学校研究生院关于研究生的视图可用图 9.6 表示。

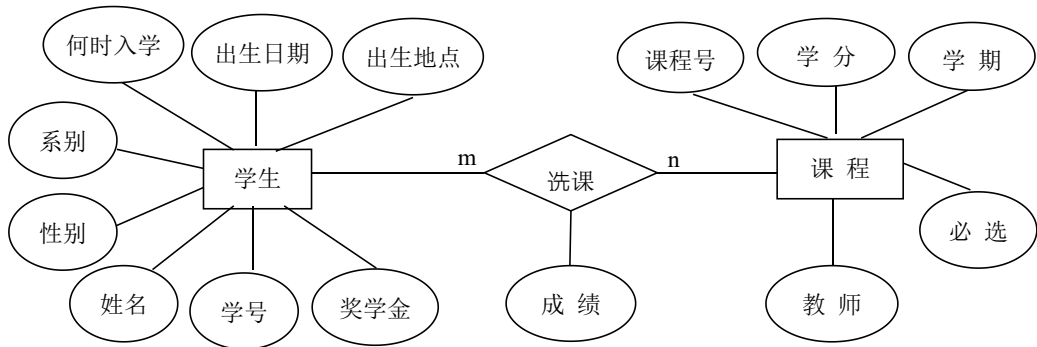


图 9.5 教务处关于学生的视图

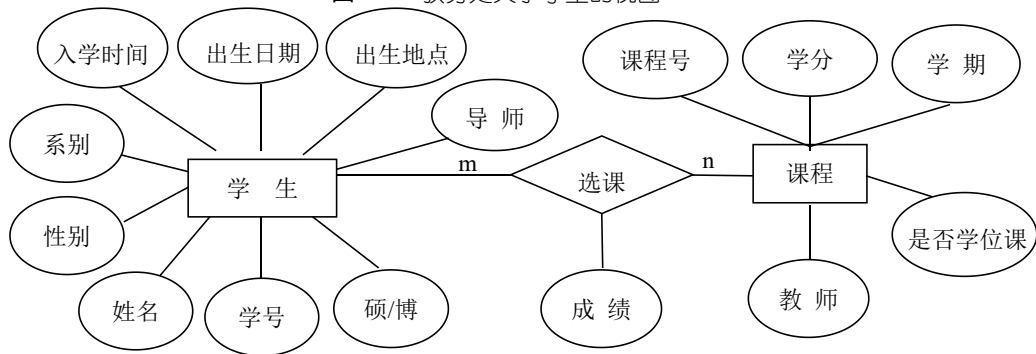


图 9.6 研究生院关于研究生的视图

例 9.3 学校教职工视图可用图 9.7 表示。

例 9.4 计算机生产厂家视图可用图 9.8 表示。

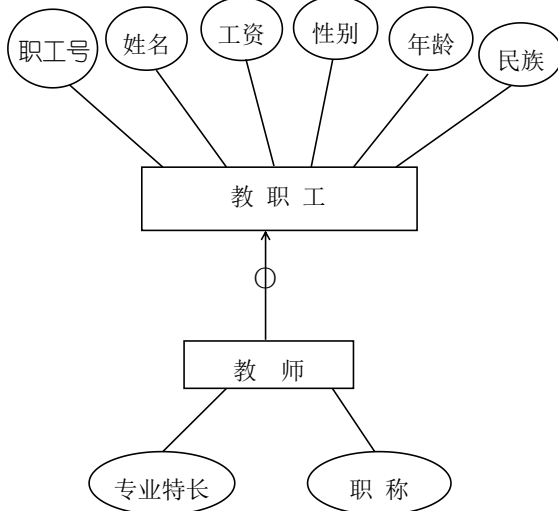


图 9.7 学校教职工视图

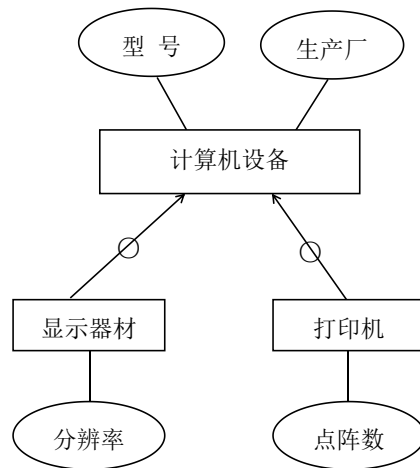


图 9.8 计算机生产厂家视图

3. 视图集成

(1) 原理与策略

视图集成的实质是所有局部视图统一与合并成一个完整的数据模式。在此过程中主要

使用三种集成方法，它们是等同(identity)、聚合(aggregation)与抽取(generalization)。

1) 等同：等同是指两个或多个数据对象有相同的语义，它包括简单的属性等同、实体等同以及语义关联的等同，等同的对象其语法形式表示可能不一致，如某单位职工按身份证号编号，属性“职工编号”与属性“职工身份证号”有相同语义。等同具有同义同名或同义异名两种含义。

2) 聚合：聚合表示数据对象间的一种组成关系，如实体“学生”可由学号、姓名、性别等聚合而成，通过聚合可将不同实体聚合成一体或将它们连接起来。

3) 抽取：抽取即是不同实体中的相同属性提取成一个新的实体并构成具有继承关系的结构。

图 9.9 给出了聚合与抽取的示例。综合运用上述三种方法可以有效地进行视图集成

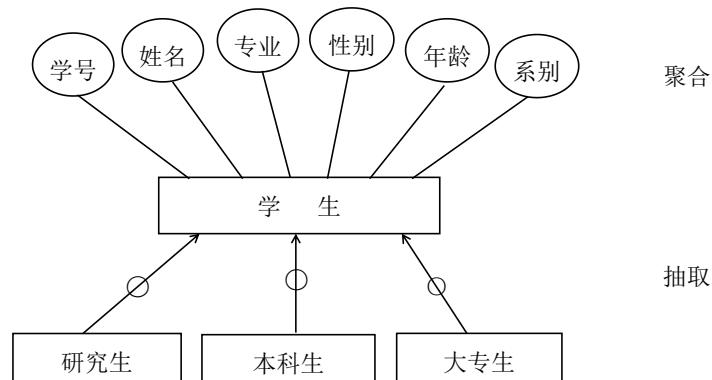


图 9.9 聚合与抽取

(2) 视图集成步骤

视图集成一般分为两步：预集成步骤与最终集成步骤。

1) 预集成步骤的主要任务：

- 确定总的集成策略，包括集成优先次序，一次集成视图数及初始集成序列等。
- 检查集成过程需要用到信息是否齐全。
- 揭示和解决冲突，为下阶段视图归并奠定基础。

2) 最终集成步骤的主要任务：

- 完整性和正确性。全局视图必需是每个局部视图正确全面的反映。
- 最小化原则。原则上现实同一概念只在一个地方表示。
- 可理解性。即应选择最为用户理解的模式结构。

(3) 冲突和解决

在集成过程中由于每个局部视图在设计时的一致性，因而会产生矛盾，引起冲突，常见冲突有下列几种。

1) 命名冲突：命名冲突有同名异义和同义异名两种，在图 9.5 及图 9.6 中“学生”一词属同名异义，在图 9.5 中表示大学生而在图 9.6 中表示研究生，同样，图 9.5 中的属性“何时入学”在图 9.6 中为“入学时间”，它们属同义异名。

2) 概念冲突：同一概念在一处为实体而在另一处为属性或联系。

3) 域冲突：相同的属性在不同视图中有不同的域，如学号在某视图中的域为字符串而在另一个视图中可为整数，有些属性采用不同度量单位也属域冲突。

4) 约束冲突: 不同视图可能有不同约束, 例如对“选课”这个联系, 大学生与研究生的最少与最多的数可能不一样。

上述冲突一般在集成时需要作统一, 形成一致的表示, 其办法即是对视图作适当修改, 如将两个视图中(图 9.5 及图 9.6)的“学生”, 一个改成“大学生”另一个改成“研究生”, 又如将“何时入学”, 与“入学时间”统一成“入学时间”从而将不一致修改成一致。图 9.5 与图 9.6 所示的两个视图经集成后形成如图 9.10 的视图。

在此视图集成中使用了等同、聚合与抽取, 并对命名冲突作了一致性处理。

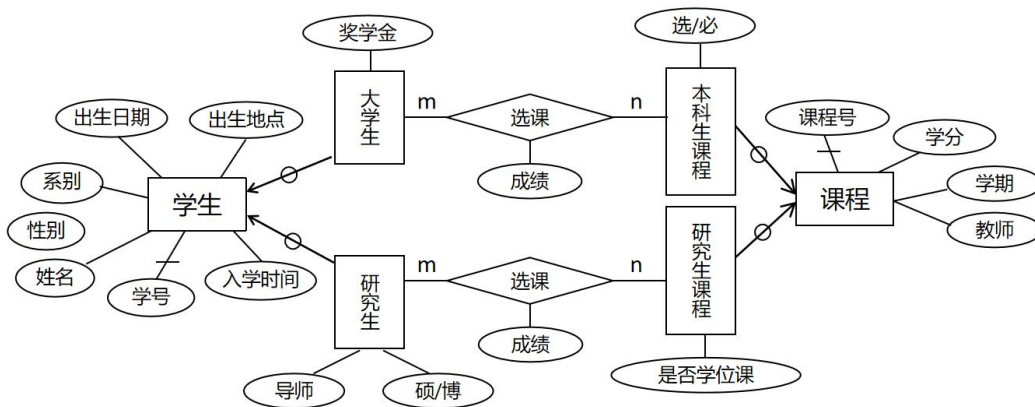


图9.10 两个视图集成

9.3.3 数据库概念设计说明书

数据库概念设计说明书主要给出概念设计中的几个基本要素—属性、实体及关联, 同时并给出相应的关联图。在这里我们给出数据库概念设计说明书模板(可见图 9.11)。

×××项目概念设计说明书	
1. 前言	<ul style="list-style-type: none"> 1.1 编写目的 1.2 背景 1.3 名词定义 1.4 参考资料
2. 属性设计	<ul style="list-style-type: none"> 2.1 属性一览 2.2 属性说明
3. 实体设计	<ul style="list-style-type: none"> 3.1 实体一览 3.2 实体说明 3.3 候选键及主键 3.4 实体与属性关系表

4. 关联设计	
4.1 联系一览	
4.2 联系说明	
4.3 继承一览	
4.4 继承说明	
5. EE-R 图	
编写人员: _____	审核人员: _____
审批人员: _____	日期: _____

图 9.11 概念设计说明书模板

9.4 数据库的逻辑设计

数据库逻辑设计包括基本设计与视图设计两部分，下面对其将作介绍。

9.4.1 数据库逻辑设计基本方法

数据库的逻辑设计的基本方法是将 EE-R 图转换成指定 RDBMS 中的关系模式，此外还包括关系的规范化以及性能调整，最后是约束条件设置。首先，从 E-R 图到关系模式的转换是比较直接的，实体与联系都可以表示成关系表，E-R 图中属性也可以转换成关系表的属性。在 EE-R 图中继承也可通过一定转换变成为关系模式中的关系表或关系视图，下面讨论由 EE-R 图转换成关系模式的一些转换问题。

1. 命名与属性域的处理

关系模式中的命名可以用 EE-R 图中原有命名，也可另行命名，但是应尽量避免重名，RDBMS 一般只支持有限种数据类型而 EE-R 中的属性域则不受此限制，如出现有 RDBMS 不支持的数据类型时则要进行类型转换。

2. 非原子属性处理

EE-R 图中允许出现非原子属性，但在关系模式中应符合第一范式故不允许出现非原子属性，非原子属性主要有集合型和元组型。如出现此种情况时可以进行转换，其转换办法是集合属性纵向展开，而元组属性则横向展开。

例 9.5 学生实体有学号、学生姓名及选读课程三个属性，其前两个为原子属性而一个为非原子属性，因为一学生可选读若干课程，设有学生 S1307, 王承志, 他修读 Database, OS 及 Network 三门课，此时可将其纵向展开用关系表形式如表 9.1 所示。

表 9.1 学生实体

学号	学生姓名	选读课程
S1307	王承志	Database
S1307	王承志	OS
S1307	王承志	Network

例 9.6 设有表示圆的实体，它有三个属性：圆标识符、圆心与半径，而圆心是由坐标 X 轴、Y 轴的位置所组成的二元组表示，在此情况下可通过横向展开将三个属性转换成四个属性，即圆标识符、圆心 X 轴位置、圆心 Y 轴位置以及半径。

3. 实体集的处理

原则上讲，一个实体集可用一个关系表示。

4. 联系的转换

在一般情况下联系可用关系表示，但是在有些情况下联系可归并到相关联的实体的关系中。具体的说来即是对 $n:m$ 联系可用单独的关系表示，而对 $1:1$ 及 $1:n$ 联系可将其归并到相关联的实体的关系中。

(1) 在 $1:1$ 联系中，该联系可以归并到相关联的实体的关系中，如图 9.12 所示。有实体集 E_1 、 E_2 及 $1:1$ 联系，其中 E_1 有主键 k ，属性 a ； E_2 有主键 h ，属性 b ；而联系 r 有属性 s ，此时，可以将 r 归并至 E_1 处，而用关系表 $R_1(k, a, h, s)$ 表示，同时将 E_2 用关系表 $R_2(h, b)$ 表示。

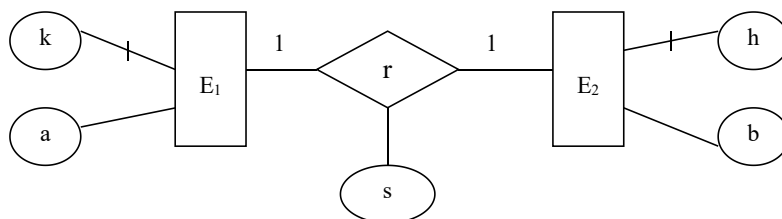


图 9.12 1:1 联系

(2) 在 $1:n$ 联系中也可将联系归并至相关联为 n 处的实体的关系表中，如图 9.13 所示，有实体集 E_1 、 E_2 及 $1:n$ 联系 r ，其中 E_1 有主码 k ，属性 a ； E_2 有主码 h ，属性 b ，而 r 有属性 s ，此时，可以将 E_1 用关系 $R_1(k, a)$ 表示，而将 E_2 及联系 r 用 $R_2(h, b, k, s)$ 表示。

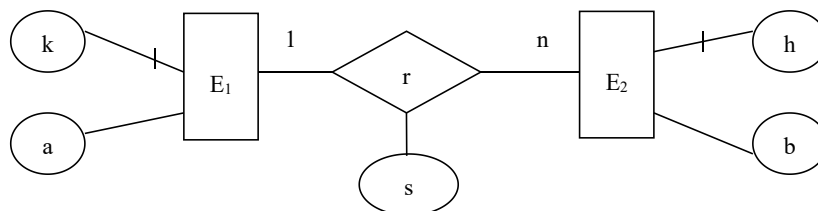


图 9.13 1:n 联系

5. 继承的转换

在 EE-R 图中继承可以有多种转换方式，一般常用的方法称扁平化处理方法，对图 9.14 所示的继承图中，超实体 S 的属性为 $\{k, a_1, a_2, \dots, a_n\}$ ，子实体 S_1 属性为 $\{a'_1, a'_2, \dots, a'_m\}$ ， S_2 属性为 $\{a''_1, a''_2, \dots, a''_p\}$ ，此时图 9.14 所示的 EE-R 图可有下面三种表示方法：

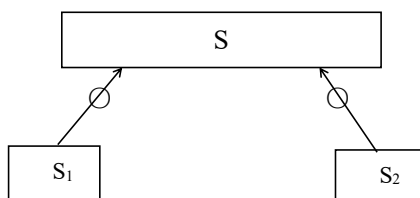


图 9.14 一个继承示例

(1) 第一种方法:

$R(k, a_1, a_2, \dots, a_n)$

$R_1(k, a'_1, a'_2, \dots, a'_m)$

$R_2(k, a''_1, a''_2, \dots, a''_p)$

(2) 第二种方法:

$R_1(k, a_1, a_2, \dots, a_n, a'_1, a'_2, \dots, a'_m)$

$R_2(k, a_1, a_2, \dots, a_n, a''_1, a''_2, \dots, a''_p)$

(3) 第三种方法:

$R(k, a_1, a_2, \dots, a_n, a'_1, a'_2, \dots, a'_m, a''_1, a''_2, \dots, a''_p)$

上述几种方案在使用时有一定限制。用第一种方法时为得到每个实体需做一次连接，即 $R \bowtie R_1$ 和 $R \bowtie R_2$ ，它们构成两个关系视图。对第二种方法一般讲仅限于子实体间不相交或子实体全覆盖，如子实体相交则一个元组可能会属于多个子实体，此时其超实体能继承的属性值在多个子实体中存储，这种冗余会产生修改异常，如果不是子实体全覆盖超实体则存在一些元素因不属于任何子实体而造成丢失。第三种方法可能会有很多 NULL，如果子实体中其特殊属性不多则可用此种方法。

在将 EE-R 图转换成关系表后，接下来是作规范化、性能调整等工作。

6. 规范化

在逻辑设计中初步形成关系表后还需对关系作规范化验证，使每个关系表至少满足第三范式，并确定主键与外键。

7. RDBMS 性能调整

满足 RDBMS 的性能、存储空间等要求的调整以及适应 RDBMS 限制条件的修改，包括如下内容：

- (1) 调整性能以减少连接运算。
- (2) 调整关系表大小，使每个关系表数量保持在合理水平，从而可以提高存取效率。
- (3) 尽量采用快照(snapshot)，因在应用中经常仅需某固定时刻的值，此时可用快照将某时刻值固定成快照，并定期更换，此种方式可以显著提高查询速度。

8. 约束条件设置

经调整后最后所生成的表尚需对其设置一定约束条件，包括表内属性及属性间的约束条件及表间属性的约束条件。这些约束条件可以是完整性约束、安全性约束，它也可以包括数据类型约束及数据量的约束等，此外，还要重新设置每个表的候选键、主键及外键。

9.4.2 关系视图设计

逻辑设计的另一个重要内容是关系视图的设计。它是在关系模式基础上所设计的直接面向操作用户的视图，它可以根据用户需求随时构作。

关系视图一般由同一模式下的表或视图组成，它由视图名、视图列名以及视图定义和视图说明等几部分组成，其作用大致有如下几点。

(1) 提供数据的逻辑独立性：数据的逻辑模式会随着应用的发展而不断地变化，一般说来，逻辑模式的变化必会影响到应用程序的变化，这就会产生极为麻烦的维护工作。而

关系视图则起了逻辑模式与应用程序之间的隔离墙作用。有了关系视图后建立在其上的应用程序就不会随逻辑模式的修改而产生变化，此时变动的仅是关系视图的定义，因此，关系视图提供了一种逻辑数据独立性，应用程序不受逻辑模式变化的影响。

(2) 能适应用户对数据的不同需求：每个数据库有一个非常庞大的结构，而每个数据库用户则只需知道他们自己所关心的那部分结构，不必知道数据的全局结构，以减轻用户在此方面的负担，而此时可用关系视图屏蔽用户所不需要的模式，仅将用户感兴趣部分呈现给用户。

(3) 有一定数据保密功能：关系视图为每个用户划定了访问数据的范围，从而在各用户间起了一定的保密隔离作用。

9.4.3 数据库逻辑设计说明书

数据库逻辑设计说明书主要是设计关系数据库中的表、视图、属性以及相应的约束设计作说明，在本节中我们给出该说明书的模板作参考（可见图 9.15）。

9.5 数据库的物理设计

数据库物理设计是在逻辑设计基础上进行的，其主要目标是对数据库内部物理结构作调整并选择合理的存取路径，以提高数据库访问速度及有效利用存储空间。在现代关系数据库中已大量屏蔽了内部物理结构，因此留给用户参与物理设计的余地并不多，一般的 RDBMS 中留给用户参与物理设计的内容大致有如下几种：

1. 存取方法的设计：

- 索引设计
- 集簇设计
- HASH 设计

2. 存贮结构设计

- 确定数据存放位置
- 确定系统配置参数

现就这两个方面的设计作介绍。

×××项目逻辑设计说明书	
<p>1. 前言</p> <p>1.1 编写目的</p> <p>1.2 背景</p> <p>1.3 名词定义</p> <p>1.4 参考资料</p> <p>2. 表设计</p> <p>2.1 表一览</p> <p>2.2 表结构</p> <p>2.3 候选键、主键与外键</p> <p>2.4 表说明</p> <p>3. 属性设计</p> <p>3.1 属性一览</p> <p>3.2 属性说明</p> <p>3.3 属性与表的关系</p> <p>4. 数据约束设计</p> <p>4.1 数据完整性设计</p> <p>4.2 数据安全性设计</p> <p>5. 视图设计</p> <p>5.1 视图一览</p> <p>5.2 视图说明</p> <p>5.3 视图定义</p>	<p>编写人员：_____ 审核人员：_____</p> <p>审批人员：_____ 日期：_____</p>

图 9.15 逻辑设计说明书模板

9.5.1 存取方法设计

1. 索引设计

索引设计是数据库物理设计的基本内容之一，有效的索引机制对提高数据库访问效率有很大作用。

索引一般建立在关系的属性上，它主要用于常用的或重要的查询中，下面给出符合建立索引的条件：

(1) 主键及外键上一般都建立索引，以加快实体间连接速度，有助于引用完整性检查以及惟一性检查。

(2) 以读为主的关系表尽可能多的建立索引。

(3) 对等值查询如满足条件的元组量小可考虑建立索引。

(4) 有些查询可从索引直接得到结果，不必访问数据块，此种查询可建索引，如查询

某属性的 MIN, MAX, AVG, SUM, COUNT 等函数值可沿该属性索引的顺序集扫描直接求得结果。

2. 集簇设计

集簇即是有关的数据元组集中存放于一个物理块内或相邻物理块或同一柱面内以提高查询效率,在目前的 RDBMS 中均有此功能,集簇一般至少定义在一个属性之上,也可以定义在多个属性之上。

集簇对某些特定应用特别有效,它可以明显提高查询效率,但是对于与集簇属性无关的访问则效果不佳。建立集簇开销很大,只有在特殊情况可考虑建立集簇:

(1) 通过集簇访问是对应表的主要应用时可考虑建立集簇。

(2) 集簇属性的对应数据量不能太少也不宜过大,太少效益不明显,而太大则要对盘区采用多种连接方式,对提高效率会产生负面影响。

(3) 集簇属性的值应相对稳定以减少修改集簇所引起的维护开销。

3. HASH 设计

有些 DBMS 提供了 HASH 存取方法,它主要在某些情况下可以使用。如表中属性在等连接条件中或在相等比较选择条件中,以及表的大小可预知测时可用 HASH 方法。

9.5.2 存贮结构设计

1. 数据存放位置设计(又称分区设计)

数据库中的数据一般存放于磁盘内,由于数据量的增大,往往需要用到多个磁盘驱动器或磁盘阵列,因此就产生了数据在多个盘组上的分配问题,这就是所谓磁盘分区设计,它是数据库物理设计内容之一,其一般指导性原则如下:

(1) 减少访盘冲突,提高 I/O 并行性。多个事务并发访问同一磁盘组时会产生访盘冲突而引发等待,如果事务访问数据能均匀分布于不同磁盘组上则可并发执行 I/O,从而提高数据库访问速度。

(2) 分散热点数据,均衡 I/O 负担。在数据库中数据被访问的频率是不均匀的,有些经常被访问的数据称热点数据(hot spot data),此类数据宜分散存放于各磁盘组上以均衡各盘组负荷,充分发挥多磁盘组并行操作优势。

(3) 保证关键数据快速访问,缓解系统瓶颈。在数据库中有些数据如数据字典、数据目录,其访问频率很高,为了保证对它的访问直接影响整个系统的效率,在此种情况可以用某一固定盘组专供其使用以保证其快速访问。

2. 系统参数配置

物理设计的一个重要内容是为数据库管理系统产品设置与调整系统参数配置,如数据库用户数、同时打开数据库数、内存分配参数、缓冲区分配参数、存储分配参数、时间片大小、数据库大小、锁的数目等。

9.5.3 数据库物理设计说明书

数据库物理设计说明书是对数据库的存贮结构及存取方法的设计作说明,在本节中我们给出该说明书的模板供参考,可见图 9.16。

×××项目物理设计说明书	
1. 前言	<ul style="list-style-type: none"> 1.1 编写目的 1.2 背景 1.3 名词定义 1.4 参考文献
2. 存取方法选择	<ul style="list-style-type: none"> 2.1 索引设计 <ul style="list-style-type: none"> 2.1.1 索引一览 2.1.2 索引说明 2.1.3 索引定义 2.2 集簇设计 <ul style="list-style-type: none"> 2.2.1 集簇一览 2.2.2 集簇说明 2.2.3 集簇定义 2.3 HASH 设计 <ul style="list-style-type: none"> 2.3.1 HASH 一览 2.3.2 HASH 说明 2.3.3 HASH 定义
3. 存储结构设计	<ul style="list-style-type: none"> 3.1 分区设计 <ul style="list-style-type: none"> 3.1.1 分区一览 3.1.2 分区说明 3.1.3 分区定义 3.2 系统参数配置 <ul style="list-style-type: none"> 3.2.1 系统参数配置一览 3.2.2 系统参数配置说明 3.2.3 系统参数配置定义
编写人员： _____	审核人员： _____
审批人员： _____	日期： _____

图 9.16 物理设计说明书模板

习题九

- 9.1 试用 EE-R 模型为一个大学数据库作概念设计并最终画出全局模式的 EE-R 图。
- 9.2 试用上题所画的 EE-R 图转换成关系模型。
- 9.3 试用上题转换成的关系模型用 SQL 中的 DDL 语言定义学生数据库中的表。
- 9.4 对上题所定义的表作索引设计。

-
- 9.5 数据库逻辑设计有哪些基本内容？请叙述。
- 9.6 数据库物理设计包括哪些内容？请说明。
- 9.7 什么叫软件工程？什么叫数据工程？它们间有什么区别，请说明之。
- 9.8 试说明数据工程与数据库设计间的关系。
- 9.9 什么叫需求分析及需求分析说明书？试说明。
- 9.10 在概念设计中为何采用 EE-R 方法，它有何优点？请说明。
- 9.11 试说明将 EE-R 图转换成关系模型的规则并用一例说明之。
- 9.12 试述数据库设计的全过程以及所产生的里程碑。

第九章复习指导

本章介绍数据库设计，它是数据库应用开发的重要环节，对数据库应用十分重要。读者学习此章后能了解数据库应用中的设计过程并会具体使用。

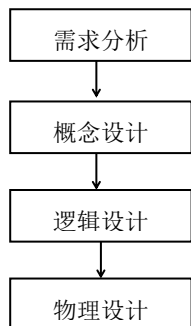
1. 与本章有关的内容

本章内容与下面一些内容相关：

- 与软件工程有关，它是软件工程的有关特殊部分；
- 与数据库规范化理论有关；
- 与数据库的概念模型、逻辑模型及物理模型有关。

数据库设计是在上述知识支持下所构成的数据库应用开发流程之一。

2. 设计流程



3. 需求分析

- (1) 需求调查
- (2) 需求分析
- (3) 绘制数据流程
- (4) 数据字典
- (5) 需求分析说明书

4. 概念设计

采用 EE-R 方法

- (1) 分解——对数据范围作分解
- (2) 视图设计
 - 1) 视图设计方法
 - 自顶向下
 - 由底向上
 - 从内向外
 - 2) 视图设计原则
 - 描述信息原则
 - 依赖性原则
 - 一致性原则
 - 3) 视图设计过程
 - 属性与实体

- 联系

- 继承

(3) 视图集成

1) 原理

- 等同

- 聚合

- 抽象

2) 步骤

5. 逻辑设计

(1) 基本原理：将 EE-R 图转换成关系表及视图

(2) 转换方法：

- 属性 \Rightarrow 属性

- 实体集 \Rightarrow 表

- 联系 $\begin{cases} 1:1 \text{ 及 } 1:n \Rightarrow \text{吸收} \\ n:n \Rightarrow \text{表} \end{cases}$

- 继承 $\xrightarrow{\text{扁平化}}$ 表

(3) 表的规范化

(4) 物理性能调整

(5) 完整性、安全性设置

(6) 在关系表之上设计视图

6. 物理设计

(1) 物理设计的两个内容

- 存取方法选择

- 存取结构设计

(2) 存取方法选择

- 索引设计

- 集簇设计

- HASH 设计

(3) 存取结构设计

- 确定系统参数配置

- 确定数据存放位置

7. 本章内容重点

- EE-R 图的构作

- EE-R 图到关系表的转换

第十章 数据库管理

数据库是需要管理的，对数据库的有效管理可以充分发挥数据资源的作用，为数据应用服务。本章主要介绍数据库管理的有关方法与技术。

10.1 数据库管理概述

数据库是一种共享的复杂的数据体，在数据库设计完成后，经过数据库建立、数据加载之后投入实际运行，并在运行过程中进行监控和维护。上述的这些管理维护工作被称为数据库管理(database administration)，而实施此项管理的人则被称为数据库管理员(database administrator，简称 DBA)。数据库管理可为充分发挥数据资源作用及为数据应用服务提供有效的及优质的管理。

在数据库系统对数据库的管理有两种方式，它们是用软件对数据库管理的 DBMS 以及用 DBA 对数据库管理的数据库管理，而后者是一种高级别的管理，它是 DBMS 所无法完成的那些管理工作，它在 DBMS 及其它工具软件的协助下完成如下的一些工作：

- 数据库的建立
- 数据库的调整
- 数据库的重组
- 数据库的重构
- 数据库的安全性控制与完整性控制
- 数据库的并发控制
- 数据库的故障恢复
- 数据库的监控

在本章的下面我们将对这些数据库管理内容作讨论。

10.2 数据库管理的内容

1. 数据库的建立

在完成数据库设计之后，要得到可以实际运行的数据库，还需要经过一个数据库的建立过程。数据库的建立包括三部分的工作：数据库运行环境的设置，数据模式的建立及数据加载。

(1) 数据库运行环境的设置

在建立数据库之前，应首先了解所选用的 DBMS 及其对运行环境的要求，熟悉 DBMS 的各种运行参数的含义、缺省值及其对系统的影响，确定新系统中各个参数的取值。不同的 DBMS 有不同的参数，一般有下列三种类型：有关内外存分配的参数，如数据文件的大小、数据块的大小、最大文件数、缓冲区的大小等；有关 DBMS 运行参数的设置，如可同时连接的用户数、可同时打开的文件和游标数量、日志缓冲区的大小等；有关数据库的故障恢复和审计跟踪的参数，如审计功能的开启/关闭参数、系统日志的设置等。

(2) 数据模式建立

数据模式由 DBA 负责建立。DBA 利用 DBMS 中的 DDL 语言来定义数据库的数据建模脚本，然后通过 DBMS 所提供的建模工具或交互式的 DDL 执行界面运行建模脚本，从而完成数据模式的建立任务。

在建模脚本中，可以使用 DDL 语言来建立数据模式定义数据库名，定义分区并申请初始存储空间和各种系统资源，定义模式表及相应属性，定义表中的主键、外键及其它数据完整性约束，定义索引、集簇等性能优化措施，定义各个授权用户在表上的访问权限，此外还可以通过定义视图来定义用户外模式，为进行数据交换还需设置会话环境参数。

(3) 会话环境的建立

在网络上运行的数据库在建立模式后尚需建立会话环境，如会话语言、时间的设定、会话数据客体模式设定以及最终会话标识符的设定。

(4) 数据加载

在数据模式建立后接下来的工作即是在模式上加加载数据，数据加载必须先做准备工作，这包含数据采集、整理及校验等工作，此后即可加载，一般 DBMS 都提供加载程序以供数据加载之用。此外，有些数据可以从其它数据体（如其它文件、数据库或网络其它结点）获取，此时必须选用相应的转换程序以实现加载。

2. 数据库调整

在数据库建立后并经一段时间运行往往会产生一些不适应的情况，有时是最初建立的数据模式不能完全满足用户应用的数据需求，也可能因为增加了新的应用而带来新的数据需求，此时需要对其作调整。同时，随着数据库中数据的不断增加，系统的性能也有可能下降而无法满应用程序在性能方面的需求，这也需要对数据库进行调整。数据库的调整一般由 DBA 完成，调整包括下面一些内容。

(1) 调整关系模式与视图使之更能适应用户的数据需求

如果是因数据项的缺失而引起的数据库调整，那么可以通过 ALTER TABLE 命令修改关系表的属性定义。如果是为改善某些应用的性能，可以采取逆规范化、关系分割以及建立快照等措施。如果是用户的数据需求发生了变化，那么就需要修改外模式，定义新的视图或修改原来的视图。

逆规范化是将若干个子关系模式合并为一个关系模式，减少应用访问过程中的关系连接次数，从而改善应用的执行性能。关系分割又分为水平分割和垂直分割，水平分割是将一个关系中的元组集合划分为互不相交的若干个子集，为每个子集建立一个独立的子模式。垂直分割是将一个关系中的属性划分为若干个属性子集，每个属性子集单独构成一个子模式，垂直分割需要满足无损联接性。通过关系的分割可以使用户的应用只需要访问某个子关系中的属性和元组，从而减少被访问关系中的数据量，达到提高数据访问性能的目的。

建立快照则是通过查询结果的实例化存储来避免对于查询结果的重复使用而带来的查询开销。

(2) 调整索引与集簇使数据库性能与效率更佳

集簇和索引的设计是数据库物理设计的主要内容，也是数据库调整的任务之一。集簇和索引的设计通常是针对用户的核心应用及其数据访问方式来进行的，随着数据库中数据量的变化以及各个用户应用的重要性程度的变化，可能需要调整原来的集簇和索引的设计方案，撤消原来的一些集簇或索引，建立一些新的集簇和索引。

(3) 调整分区、调整数据库缓冲区大小以及调整并发度使数据库物理性能更好

通过上述数据库运行环境或参数的调整以达到提高系统性能的目的。如调整数据在不同磁盘驱动器上的分布情况，调整数据库缓冲区的大小和系统的最大并发用户数等。

3. 数据库重组

数据库在经过一定时间运行后，其性能会逐步下降，下降的原因主要是由于不断的修改、删除与插入所造成的，由于不断的删除而造成盘区内废块的增多而影响 I/O 速度，由于不断的删除与插入而造成集簇的性能下降，同时也造成了存储空间分配的零散化，使得一个完整表的空间分散，从而造成存取效率下降，基于这些原因需要对数据库进行重新整理，重新调整存贮空间，此种工作叫数据库重组，一般数据库重组需花大量时间，并作大量的数据搬迁工作，往往是先作数据卸载 (unload)，然后再重新加载 (reload)，即将数据库的数据先行卸载到其它存储区域中，然后按照模式的定义重新加载到指定空间，从而达到数据重组的目的。目前一般 RDBMS 都提供一定手段，以实现数据重组功能。

重组是对数据库结构的全面调整，很费时间，也影响数据库正常服务，一般是利用周末、节假日或夜间进行，但有些数据库系统不允许中断运行，此时可采用边重组边使用的办法。但是在重组期间数据库性能将会降低，其优点是可以不中断数据库的服务。

数据库重组可提高系统性能但它要付出代价，这是一对矛盾，因此重组周期的选择要权衡利弊使之保持在合理的水平。即在重组时要进行慎重研究，选择有效的重组代价模型，在经过模型计算后才最终确定是否有重组的必要。

4. 数据库重构

数据库重构即是数据库模式的重新构造。一般而言在数据库系统中，数据模式是稳定的，但是在使用过程中由于应用环境改变而需求改变时，要求对数据库的模式作重大的变动，这称数据库重构。

数据库的重构与大量的应用程序紧密相关，这主要是数据库中结构、命名、规则的改变而影响应用程序的调用，因此在重构时最大的原则是尽量减少应用程序的改动，一种比较有效的办法是采用视图的方法，将模式修改部分尽可能对原有应用程序屏蔽，现在的 DBMS 中一般均提供数据重构的功能。

5. 数据库的安全性控制

数据库是一种高度共享的资源，它的安全性是极端重要的，DBA 应采取措施保证数据不受非法盗用与数据不受破坏。数据的安全性控制包括外部环境的安全，数据库管理系统内部的安全以及其它计算机系统内部的安全措施。

外部环境的安全主要是通过行政手段，并建立一定的规章制度以确保数据库运行环境的安全。其它计算机系统内部的安全主要是指操作系统的安全以及病毒防护安全。通过操作系统的安全保护措施是为了防止黑客或非法用户通过操作系统途径来直接访问数据库中的数据，病毒防护是指通过防病毒软件来防止病毒入侵并能及时消毒，以提高系统的可用性。

在数据库管理系统内部，数据库安全的基础是用户身份的确认（即身份鉴别）和存取控制，并辅以审计跟踪功能以达到数据库安全保护的目的。

(1) 身份鉴别

身份鉴别功能包括用户标识和用户鉴别两个方面。用户标识是指为每个进入 DBMS 的

用户分配一个用户名及一个用户标识，并在 DBMS 的整个生命周期实现用户标识的唯一性以及用户名与标识间的一致性。用户鉴别是指按要求进行用户身份鉴别，用户在登录使用 DBMS 时必须首先给出用户名及其标识，并提供如口令等身份鉴别信息，只有检验合格后才能进入使用 DBMS。DBA 可以通过对用户帐号及其口令进行控制，如用户帐号的锁定与激活、允许连续注册失败次数，口令字的构成方式、有效期等进行限制，以达到用户管理的目的，保护合法用户对数据库的访问权，防止非法用户的盗用行为。

(2) 自主访问控制

存取控制的方式上包括自主访问控制和强制访问控制，目前常用的是自主访问控制。自主访问控制是指当一个主体（用户、用户组、角色）访问某个客体时，系统应能确认主体对客体的访问操作是否合法，从而限制每个合法用户在登录进行数据库后可以访问的数据范围。DBA 或用户可以通过权限的授予（GRANT）与撤消（REVOKE）命令合理地为用户定义所需要的访问权限。

在关系 DBMS 中，可以授予用户的访问权限有多种类型。一般的 DBMS 都提供了下面的几种权限类型：系统权限，数据库对象权限，表/视图级权限，程序和过程权限。

系统权限一般是指数据库管理命令或运行参数设置命令的执行权限，数据库对象权限则是指 DDL 语句的执行权限。这两类权限的授予应该小心，一般只能由 DBA 保留或仅仅授予少数的可信用户。表/视图级权限是指表中数据的访问权限，可以根据应用的访问需要授予不同的用户，建议不要直接在表一级进行授权，而是在视图级进行用户授权，充分利用视图实现数据的安全保护。通过授予用户程序和过程的 EXECUTE 权限，可以屏蔽需要访问的表或视图，更好地维护产品数据的一致性。

(3) 审计

由于审计需要额外的执行开销，DBA 应该选择针对合适的审计事件、审计对象或审计用户进行审计，记录相关的数据库访问操作。并通过对审计数据的查阅和分析来监控数据库的运行安全。

在一些对安全要求较高的应用领域，可能需要设置更多的安全保护措施，如：强制访问控制，安全审计，数据库推理控制，数据加密等。DBA 应该根据 DBMS 所提供的上述安全保护措施进行安全性控制。

6. 数据完整性控制

数据库质量的保证首先在于其数据的正确性。国外有一句名言，即：“garbage in garbage out”，即进去的是垃圾（不正确数据）则出来的还是垃圾（无用数据），因此保证数据正确性对数据库管理系统而言是极端重要的。

为保证数据的正确性需作完整性控制，使进入数据库内的数据均能保持正确。数据库的完整性控制主要包括如下内容：

(1) 通过完整性约束检查的功能以保证数据的正确性。

首先，系统具有自动检验实体完整性与参照完整性能力，其次，须设置域约束、表约束及断言以建立数据的语义约束，最后可以通过设置触发器以建立较为复杂、有效的完整性约束机制，这三种完整性约束的相互配合使用可以保证数据库中数据的正确性。

(2) 建立必要的规章制度进行数据的及时、正确的采集及校验。

数据库中数据大都来自外界，对这些采集来的数据的正确性保证是必须建立必要的规

章制度及效验制度，从而将错误消灭在数据录入前。

7. 数据库的并发控制

为提高数据库的性能可采取数据库的并发控制技术，但是在执行中经常会出现性能不够理想以及错误，它们是：

(1) 性能的调整

当并发控制执行时，由于环境与应用的改变经常会出现并发控制性能的不稳定，此时须即时调整并发度以取得较高的执行效率。

(2) 死锁、活锁的解除

并发控制执行时经常会产生活锁与死锁现象，此时须采取相应措施调整事务的执行以解除事务的活锁与死锁，使并发控制得以继续进行。

8. 数据库的故障恢复

数据库中数据是经常会遭受破坏的，这种破坏可来自外部和内部两个方面，数据在一旦遭受破坏后能及时进行恢复是数据库管理的基本任务。DBMS 一般都提供故障恢复的有关手段，并由 DBA 负责执行故障恢复功能。一般讲，为实现故障恢复首先要进行定期的转储，常用的方法是拷贝。其次，在数据库运行时要启动日誌工作，最后，DBA 利用日誌、拷贝及 REDO、UNDO 等手段，根据不同的故障类型（小型、中型、大型）及类别进行故障恢复，此外对重要的数据库需要制订灾难恢复计划，这是一种预防性的应急计划，其目的是减轻灾难事件所带来的破坏并使应用系统尽快恢复工作，数据损失达到最小。

9. 数据库的监控

DBA 使用监控工具随时观察数据库的动态变化，这种行为称为数据库监控。当 DBA 在监控过程中发现数据错误、故障或产生不适应情况时随时可产生报警以便并即时采取措施。如数据库死锁、对数据库的误操作等，同时监控工具还可监视数据库的性能变化，以为数据库重组提供动态信息使 DBA 在必要时能对数据库作调整。

数据库的监控是数据管理的基本任务，也是 DBA 的日常工作之一。

10.3 数据库管理员 DBA

DBA 是管理数据库的核心人物，他一般由若干个人组成，他是数据库的监护人，也是数据库与用户间的联系人。

DBA 具有最高级别的特权，他对数据库系统应有足够的了解与熟悉，一个数据库能否正常、成功的运行，DBA 是关键。一般讲 DBA 除了完成数据库管理的工作外，它还需要完成相关的行政管理工作以及参与数据库设计的部分工作，其具体任务如下：

- (1) 参与数据库设计的各个阶段的工作，对数据库有足够的了解。
- (2) 负责数据库的建立、调整、重组与重构。
- (3) 维护数据库中数据的安全性。
- (4) 维护数据库中数据的完整性以及对并发控制作管理。
- (5) 负责数据库的故障恢复及制订灾难恢复计划。
- (6) 对数据库作监控，及时处理数据库运行中的突发事件并对其性能作调整。

在上面六个任务中，DBA 承担着数据库管理的有关技术性工作，特别要提醒的是第三

个任务即有关安全性维护，在网络环境中数据库的用户已是 DBA 所无法完全控制的了，因此加强安全管理是数据库管理的重要任务，在此情况下，将有关安全控制的设置与管理以及审计控制与管理的职能从 DBA 中单独分开，专门设置安全管理员（security administrator）及审计员（auditor），这种设置方式有利于对数据库安全的管理，而这种管理模式称为三权分列式。

下面的两个任务则是 DBA 的行政性管理任务。

(7) 帮助与指导数据库用户

与用户保持联系，了解用户需求，倾听用户反映，帮助他们解决有关技术问题，编写技术文档，指导用户正确使用数据库。

(8) 制定必要的规章制度，并组织实施。

为便于使用管理数据库，需要制订必要的规章制度，如数据库使用规定，数据库安全操作规定，数据库值班记录要求等，同时还要组织、检查及实施这些规定。

习题十

- 10.1 数据库管理有哪几件工作？请说明之。
- 10.2 DBA 的具体任务是什么？试说明之。
- 10.3 试说明 DBA 与安全管理员及审计员三者职能的区别。
- 10.4 试说明 DBMS 与 DBA 的管理工作有什么不同？

第十章复习指导

数据库管理是对数据库中的数据作高层次管理的一种手段与技术，学习数据库管理这章后须对数据库管理有一个全面的了解，并初步掌握管理的方法。

1. 数据管理的两个层次：
 - 低层次——用软件 DBMS 对数据作管理 (management)。
 - 高层次——DBA 与软件相结合对数据作管理 (administration)。
2. 数据库管理与 DBA
 - 数据库管理限技术层面。
 - DBA 负责数据库管理以及行政管理。
3. 数据库管理任务：
 - (1) 数据库的建立
 - (2) 数据库的调整
 - (3) 数据库的重组
 - (4) 数据库的重构
 - (5) 数据库的安全性控制与完整性控制
 - (6) 数据库的并发控制
 - (7) 数据库的故障恢复
 - (8) 数据库的监控
4. DBA 任务：
 - (1) 参与数据库设计的各个阶段的工作，对数据库有足够的了解。
 - (2) 负责数据库的建立、调整、重组与重构。
 - (3) 维护数据库中数据的安全性。
 - (4) 完整性以及对并发控制作管理。
 - (5) 负责数据库的故障恢复及制订灾难恢复计划。
 - (6) 对数据库作监控，及时处理数据库运行中的突发事件并对其性能作调整。
 - (7) 帮助与指导数据库用户。
 - (8) 制定必要的规章制度，并组织实施。
5. 三权分列模式
 - (1) DBA
 - (2) 安全管理员
 - (3) 审计员
6. 本章重点：

数据库管理的九大任务

参 考 文 献

- 1 徐洁磐, 王银根. 数据库系统原理. 重庆: 科学技术文献出版社重庆分社, 1989
- 2 施伯乐. 数据库理论及新领域. 北京: 高等教育出版社, 1990
- 3 萨师煊, 王珊. 数据库系统概论. 北京: 高等教育出版社, 1991
- 4 王能斌, 董逸生. 数据库设计与实现. 武汉: 华中理工大学出版社, 1991
- 5 王珊, 冯念真. ORACLE RDBMS 分析. 北京: 中国人民大学出版社, 1991
- 6 徐洁磐, 王银根. 数据库系统导论. 北京: 科学技术文献出版社, 1991
- 7 李昭原. 数据库系统原理与技术. 北京: 北京航空航天大学出版社, 1992
- 8 汪成为等. 面向对象分析、设计与应用. 北京: 国防工业出版社, 1992
- 9 张少润, 陈自安. SQL/DS 数据库系统. 厦门: 厦门大学出版社, 1992
- 10 冯玉才. 数据库基础. 武汉: 华中工学院出版社, 1992
- 11 唐常杰. 数据库管理系统设计和实现. 北京: 电子工业出版社, 1993
- 12 贾 焰. 知识库系统原理. 北京: 国防工业出版社, 1993
- 13 石树刚, 郑振楣. 关系数据库. 北京: 清华大学出版社, 1993
- 14 王珊. SYBASE 关系数据库系统原理和指南. 合肥: 中国科学技术大学出版社, 1993
- 15 马玉书. 面向对象程序设计语言. 北京: 石油工业出版社, 1994
- 16 马玉书主编. 数据库技术名词解释. 北京: 石油工业出版社, 1994
- 17 何守才主编. 数据库综合大辞典. 上海: 上海科学技术文献出版社, 1994
- 18 王能斌. 数据库系统. 北京: 电子工业出版社, 1995
- 19 何炎祥, 郑振楣, 石树刚. 面向对象数据库. 武汉: 武汉大学出版社, 1995
- 20 陈建新等. 数据库技术. 北京: 石油工业出版社, 1995
- 21 徐洁磐, 王银根. 数据库系统引论. 南京: 南京大学出版社, 1996
- 22 李昭原主编. 数据库技术新进展. 北京: 清华大学出版社, 1997
- 23 周傲英, 汪卫, 刘宏亮. DB2 应用开发指南. 北京: 电子工业出版社, 1998
- 24 徐洁磐. 数据库系统原理. 上海: 上海科技文献出版社, 1999
- 25 施伯乐等. 数据库系统教程. 北京: 高等教育出版社, 1999
- 26 刘启源. 数据库与信息系统安全. 北京: 科学出版社, 1999
- 27 徐洁磐. 知识库系统导论. 北京: 科学出版社, 1999
- 28 王能斌. 数据库系统原理. 北京: 电子工业出版社, 2000
- 29 萨师煊, 王珊. 数据库系统概论 (第 3 版). 北京: 高等教育出版社, 2000
- 30 徐洁磐. 现代数据库系统教程. 北京: 北京希望电子出版社, 2002
- 31 施伯乐, 丁宝康. 数据库技术. 北京: 科学出版社, 2002
- 32 王能斌. 数据库系统教程. 北京: 电子工业出版社, 2002
- 33 徐洁磐. 面向对象数据库系统及其应用. 北京: 科学出版社, 2003
- 34 李建中, 王珊. 数据库系统原理 (第二版). 北京: 电子工业出版社, 2004
- 35 冯建华, 周立柱. 数据库系统设计与原理. 北京: 清华大学出版社, 2004
- 36 徐洁磐. 数据仓库与决策支持系统. 北京: 科学出版社, 2005

-
- 37 许龙飞等. Web 数据库技术与应用. 北京: 科学出版社, 2005
 - 38 邵佩英. 分布式数据库系统及其应用 (第二版). 北京: 科学出版社, 2005
 - 39 Martin J . Principles of Database Management . Computer science press, 1978
 - 40 Ullman J.D . Principles of Database Systems . Computer Science press, 1982
 - 41 Ullman J.D . Principles of Database and Knowledge Base Systems . Computer science press, 1989
 - 42 Chao-chinyang . Relational Database . Prentice-Hall, 1989
 - 43 Kim W . Introduction to Object – Oriented Database System . Computer science press, 1993
 - 44 Kemper A . Object – Oriented Database Management . Prentice-Hall, Inc., 1994
 - 45 Chon H.T . Database Design and its Application . McGraw-Hill press, 1995
 - 46 R. Karts . Knowledge Base System . Addison-Weslag press, 1996
 - 47 Immon W.H . Building the Data Warehouse . 2nd ed. John Wiley & Sons, Inc.,1996
 - 48 Grant J . Logical introduction to database . HBJ press, 1996
 - 49 Date C.J . Database primer . Computer science press, 1997
 - 50 Elmasri R . Fundamentals of Database systems . McGraw-Hill press, 1997
 - 51 Samet H . The Design and Analysis of Object – Oriented Database . Addison-Weslag press, 1998
 - 52 Silbersdaatg A . Database System Concepts . McGraw-Hill Companies ,Inc.,1999
 - 53 Date C.J . An Introduction to Database System (7’Edition) . Addison-Weslay, 2000
 - 54 Han J . Data Mining Concepts and Techniques . Academic press, 2001
 - 55 Stephens R. K. Database Design . McGraw-Hill Companies, Inc.,2001
 - 56 Kroenke D.M.Database Processing: Fundamentals, Design and Implementation , (8’Edition), Prentice Hall, 2002
 - 57 Lewis, P.H. Databse and Transaction Processing – An Application-Oriented Approach, Addison-Weslay, 2002