目 录

第 1 篇 基础篇.....6
第 1 章 绪 论.....6
1.1 复习笔记.....6
1.2 课后习题详解.....12
第 2 章 关系数据库.....19
2.1 复习笔记.....19
2.2 课后习题详解.....23
第 3 章 关系数据库标准语言 SQL.....28
3.1 复习笔记.....28
3.2 课后习题详解.....37
第 4 章 数据库安全性.....41
4.1 复习笔记.....41
4.2 课后习题详解.....47
第 5 章 数据库完整性.....51
5.1 复习笔记.....51
5.2 课后习题详解.....53
第 2 篇 设计与应用开发篇.....56
第 6 章 关系数据理论.....56
6.1 复习笔记.....56
6.2 课后习题详解.....60
第 7 章 数据库设计.....63
7.1 复习笔记.....63
7.2 课后习题详解.....71
第 8 章 数据库编程.....76
8.1 复习笔记.....76
8.2 课后习题详解.....85
第 3 篇 系统篇.....87
第 9 章 关系查询处理和查询优化.....87
9.1 复习笔记.....87
9.2 课后习题详解.....91
第 10 章 数据库恢复技术.....94
10.1 复习笔记.....94
10.2 课后习题详解.....97
第 11 章 并发控制.....100
11.1 复习笔记.....100
11.2 课后习题详解.....104
第 12 章 数据库管理系统.....111
12.1 复习笔记.....111
12.2 课后习题详解.....115
第 4 篇 新技术篇.....140
第 13 章 数据库技术发展概述.....140
13.1 复习笔记.....140
13.1 课后习题详解.....143
第 14 章 大数据管理.....145
14.1 复习笔记.....145
14.1 课后习题详解.....147
第 15 章 内存数据库系统.....150
15.1 复习笔记.....150

15.2 课后习题详解.....152
第 16 章 数据仓库与联机分析处理技术.....154
16.1 复习笔记.....154
16.2 课后习题详解.....158

第 1 篇　基础篇

第 1 章　绪　论

1.1　复习笔记

一、数据库系统概述

1．数据库的 4 个基本概念

（1）数据（data）

数据是数据库中存储的基本对象，描述事物的符号记录称为数据。描述事物的符号可以是数字，也可以是文字、图形、图像、音频、视频等。

（2）数据库（DataBase，DB）

①定义

数据库是长期储存在计算机内、有组织的、可共享的大量数据的集合。数据库中的数据按一定的数据模型组织、描述和储存，具有较小的冗余度（redundancy）、较高的数据独立性（dam independency）和易扩展性（scalability），并可为各种用户共享。

②特点

a．永久存储；

b．有组织；

c．可共享。

（3）数据库管理系统（DataBase Management system，DBMS）

数据库管理系统是位于用户与操作系统之间的一层数据管理软件。数据库管理系统和操作系统一样是计算机的基础软件，也是一个大型复杂的软件系统。它的主要功能包括以下几个方面：

①数据定义功能

数据库管理系统提供数据定义语言（Data Definition Language，DDL），用户通过它可以方便地对数据库中的数据对象的组成与结构进行定义。

②数据组织、存储和管理

数据组织和存储的基本目标是提高存储空间利用率和方便存取，提供多种存取方法（如索引查找、hash 查找、顺序查找等）来提高存取效率。

③数据操纵功能

数据库管理系统还提供数据操纵语言（DataManipulationLanguage，DML），实现对数据库的基本操作，如查询、插入、删除和修改等。

④数据库的事务管理和运行管理

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

⑤数据库的建立和维护功能

数据库的建立和维护功能包括数据库初始数据的输入、转换功能，数据库的转储、恢复功能，数据库的重组织功能和性能监视、分析功能等。

⑥其他功能

其他功能包括通信功能；数据转换功能；互访和互操作功能等。

（4）数据库系统（DataBase System，DBS）

数据库系统是由数据库、数据库管理系统（及其应用开发工具）、应用程序和数据库管理员（DataBase Administrator，DBA）组成的存储、管理、处理和维护数据的系统。

数据库系统如图 1-1 所示。引入数据库后计算机的层次结构如图 1-2 所示。

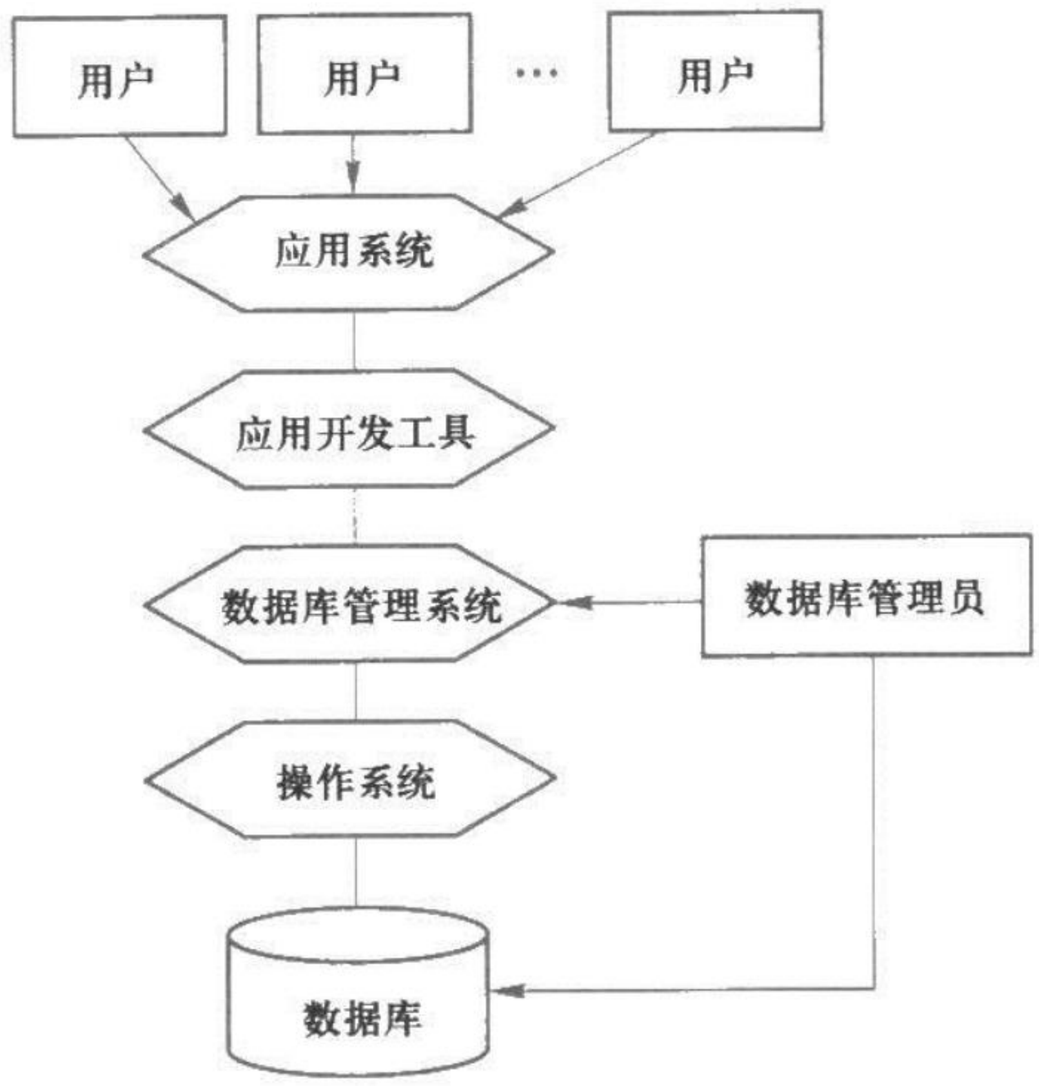


图 1-1　数据库系统

图 1-2　引入数据库后计算机系统的层次结构

2．数据管理技术的产生和发展

在计算机硬件、软件发展的基础上，数据管理技术经历了人工管理、文件系统、数据库系统 3 个阶段。这 3 个阶段的特点及其比较如表 1-1 所示。

表 1-1　数据管理 3 个阶段的比较

		人工管理阶段	文件系统阶段	数据库系统阶段
背景	应用背景	科学计算	科学训算、数据管理	大规模数据管理
	硬件背景	无直接存取存储设备	磁盘、磁鼓	大容量磁盘、磁盘阵列
	软件背景	没有操作系统	有文件系统	有数据库管理系统
	处理方式	批处理	联机实时处理、批处理	联机实时处理、分布处理、批处理
特点	数据的管理者	用户（程序员）	文件系统	数据库管理系统
	数据面向的对象	某一应用程序	某一应用	现实世界（一个部门、企业、跨国组织等）
	数据的共享程	无共享，冗余度极大	共享性差，冗余度大	共享性高，冗余度小

	度			
	数据的独立性	不独立，完全依赖于程序	独立性差	具有高度的物理独立性和一定的逻辑独立性
	数据的结构化	无结构	记录内有结构、整体无结构	整体结构化，用数据模型描述
	数据控制能力	应用程序自己控制	应用程序自己控制	由数据库管理系统提供数据安全性、完整性、并发控制和恢复能力

（1）人工管理阶段

人工管理数据具有如下特点：

①数据不保存。

②应用程序管理数据。

③数据不共享。

④数据不具有独立性。

（2）文件系统阶段

①特点

a．数据可以长期保存；

b．由文件系统管理数据。

②缺点

a．数据共享性差，冗余度大；

b．数据独立性差。

（3）数据库系统阶段

①特点

a．数据结构化；

b．数据的共享性高、冗余度低且易扩充；

c．数据独立性高；

d．数据由数据库管理系统统一管理和控制。

②DBMS 的数据控制功能

a．数据的安全性（secufity）保护；

b．数据的完整性（integrity）检查；

c．并发（concurrency）控制；

d．数据库恢复（recovery）。

二、数据模型

1．定义

数据模型（data model）是对现实世界数据特征的抽象，是用来描述数据、组织数据和对数据进行操作的模型。

2．分类

根据模型应用的不同目的，数据模型分为两类：一类是概念模型；一类是逻辑模型和物理模型。

（1）概念模型

概念模型（conceptual model）也称信息模型，它是按用户的观点来对数据和信息建模，主要用于数据库设计。

（2）逻辑模型和物理模型

逻辑模型主要包括层次模型（hierarchicalmodel）、网状模型（networkmodel）、关系模型（relational model）、面向对象数据模型（object oriented data model）和对象关系数据模型（object relational data model）、半结构化数据模型（semistructured data model）等。它是按计算机系统的观点对数据建模，主要用于 DBMS 的实现；物理模型是对数据最底层的抽象，它描述数据在系统内部的表示方式和存取方法，或在磁盘或磁带上的存储方式和存取方法，是面向计算机系统的。

3．概念模型

（1）特点

①较强的语义表达能力。

②简单、清晰、易于用户理解。

（2）基本概念

①实体（entity）

客观存在并可相互区别的事物称为实体。

②属性（attribute）

实体所具有的某一特性称为属性。

③自马（key）

唯一标识实体的属性集称为码。

④实体型（entity type）

具有相同属性的实体必然具有共同的特征和性质。用实体名及其属性名集合来抽象和刻画同类实体，称为实体型。

⑤实体集（entity set）

同一类型实体的集合称为实体集。

⑥联系（relationship）

在现实世界中，事物内部以及事物之间是有联系的，这些联系在信息世界中反映为实体（型）内部的联系和实体（型）之间的联系。

（3）实体一联系方法

概念模型是对信息世界建模，能够方便、准确地表示出信息世界中的常用概念。概念模型的表示方法很多，最常用的方法用 E-R 图（E．R diagram）来描述现实世界的概念模型，E-R 方法也称为 E-R 模型。

4．组成要素

（1）数据结构

数据结构描述数据库的组成对象以及对象之间的联系。数据结构是所描述的对象类型的集合，是对系统静态特性的描述。

（2）数据操作

数据操作是指对数据库中各种对象（型）的实例（值）允许执行的操作的集合，包括操作及有关的操作规则。数据库主要有查询和更新（包括插入、删除、修改）两大类操作。数据操作是对系统动态特性的描述。

（3）数据的完整性约束条件

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

5．常用的数据模型

（1）层次模型

①概述

层次模型是数据库系统中最早出现的数据模型，层次数据库系统采用层次模型作为数据的组织方式。层次模型用树形结构来表示各类实体以及实体间的联系。

②数据结构

在数据库中定义满足下面两个条件的基本层次联系的集合为层次模型。

a．有且只有一个结点没有双亲结点，这个结点称为根结点；

b．根以外的其他结点有且只有一个双亲结点。

③数据操纵与完整性约束

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

④优缺点

a．优点

数据结构比较简单清晰；层次数据库的查询效率高；提供了良好的完整性支持。

b．缺点

多对多联系不适合用层次模型表示；对插入和删除的限制比较多，应用程序的编写较复杂；查询予女结点必须通过双亲结点；由于结构严密，层次命令趋于程序化。

（2）网状模型

①概述

网状数据库系统采用网状模型作为数据的组织方式。网状数据模型的典型代表是 DBTG 系统，亦称 CODASYL 系统。

②数据结构

网状模型满足以下两个条件：

a．允许一个以上的结点无双亲；

b．一个结点可以有多于一个的双亲。

③数据操纵与完整性约束

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

a．支持记录码的概念，码即唯一标识记录的数据项的集合；

b．保证一个联系中双亲记录和子女记录之间是一一对多的联系；

c．可以支持双亲记录和子女记录之间的某些约束条件。

④优缺点

a．优点

能够更为直接地描述现实世界；具有良好的性能，存取效率较高。

b．缺点

结构比较复杂；DDL、DML 复杂，用户不容易掌握，不容易使用；加重了编写应用程序的负担。

（3）关系模型

①数据结构

关系模型建立在严格的数学概念的基础上。从用户观点看，关系模型由一组关系组成。②术语

a．关系：一个关系对应通常说的一张表；

b．元组（tuple）：表中的一行即为一个元组；

c．属性（attribute）：表中的一列即为一个属性，给每一个属性起一个名称即属性名；

d．码（key）：也称为码键。表中的某个属性组，它可以唯一确定一个元组；

e．域（domain）：域是一组具有相同数据类型的值的集合；

f．分量：元组中的一个属性值；

g．关系模式：对关系的描述，一般表示为关系名（属性 1，属性 2，…，属性 n）。

关系的每一个分量必须是一个不可分的数据项，不允许表中还有表。

可以把关系和现实生活中的表格所使用的术语做一个粗略的对比，如表 1-2 所示。

表 1-2　术语对比

关系术语	一般表格的术语
关系名	表名
关系模式	表头（表格的描述）
关系	（一张）二维表
元组	记录或行
属性	列
属性名	列名
属性值	列值
分量	一条记录中的一个列值
非规范关系	表中有表（大表中嵌有小表）

③数据操纵与完整性约束

关系模型的数据操纵主要包括查询、插入、删除和更新数据。这些操作必须满足关系的完整性约束条件。关系的完整性约束条件包括三大类：实体完整性、参照完整性和用户定义的完整性。

④优缺点

a．优点

建立在严格的数学基础上；关系模型的概念单一；具有更高的数据独立性、更好的安全保密性，也简化了程序员的工作和数据库开发建立的工作。

b．缺点

查询效率往往不如格式化数据模型；增加了开发数据库管理系统的难度。

三、数据库系统的结构

1．数据库系统模式的概念

模式（schema）是数据库中全体数据的逻辑结构和特征的描述，它仅仅涉及型的描述，

不涉及具体的值。模式的一个具体值称为模式的一个实例（instance）。模式是相对稳定的，而实例是相对变动的，因为数据库中的数据是在不断更新的。模式反映的是数据的结构及其联系，而实例反映的是数据库某一时刻的状态。

2．数据库系统的三级模式结构

（1）模式（schema）

模式也称逻辑模式，是数据库中全体数据的逻辑结构和特征的描述，是所有用户的公共数据视图。模式实际上是数据库数据在逻辑级上的视图。一个数据库只有一个模式。

（2）外模式（external schema）

外模式也称子模式（subschema）或用户模式，它是数据库用户（包括应用程序员和最终用户）能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。外模式通常是模式的子集。一个数据库可以有多个外模式。

（3）内模式（internal schema）

内模式也称存储模式（storage schema），一个数据库只有一个内模式。它是数据物理结构和存储方式的描述，是数据在数据库内部的组织方式。

3．数据库的二级映像功能与数据独立性

（1）概述

为了能够在系统内部实现这三个抽象层次的联系和转换，数据库管理系统在这三级模式之间提供了两层映像：外模式 / 模式映像和模式 / 内模式映像。这两层映像保证了数据库系统中的数据能够具有较高的逻辑独立性和物理独立性。

（2）二级映像

①外模式 / 模式映像

模式描述的是数据的全局逻辑结构，外模式描述的是数据的局部逻辑结构。对应于同一个模式可以有任意多个外模式。对于每一个外模式，数据库系统都有一个外模式 / 模式映像，它定义了该外模式与模式之间的对应关系。

②模式 / 内模式映像

数据库中只有一个模式，也只有一个内模式，所以模式 / 内模式映像是唯一的，它定义了数据全局逻辑结构与存储结构之间的对应关系。

（3）数据独立性

当数据库的存储结构改变时，由数据库管理员对模式 / 内模式映像作相应改变，可以使模式保持不变，从而应用程序也不必改变。保证了数据与程序的物理独立性，简称数据的物理独立性。

四、数据库系统的组成

1．硬件平台及数据库

数据库系统对硬件资源的要求：

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

（2）有足够大的磁盘或磁盘阵列等设备存放数据库，有足够大的磁带（或光盘）作数据备份。

（3）要求系统有较高的通道能力，以提高数据传送率。

2．软件

数据库系统的软件主要包括：

（1）数据库管理系统。

（2）支持数据库管理系统运行的操作系统。

（3）具有与数据库接口的高级语言及其编译系统。

（4）以数据库管理系统为核心的应用开发工具。

（5）为特定应用环境开发的数据库应用系统。

3．人员

开发、管理和使用数据库系统的人员主要包括数据库管理员、系统分析员和数据库设计人员、应用程序员和最终用户。不同的人员涉及不同的数据抽象级别，具有不同的数据视图，如图 1-3 所示。

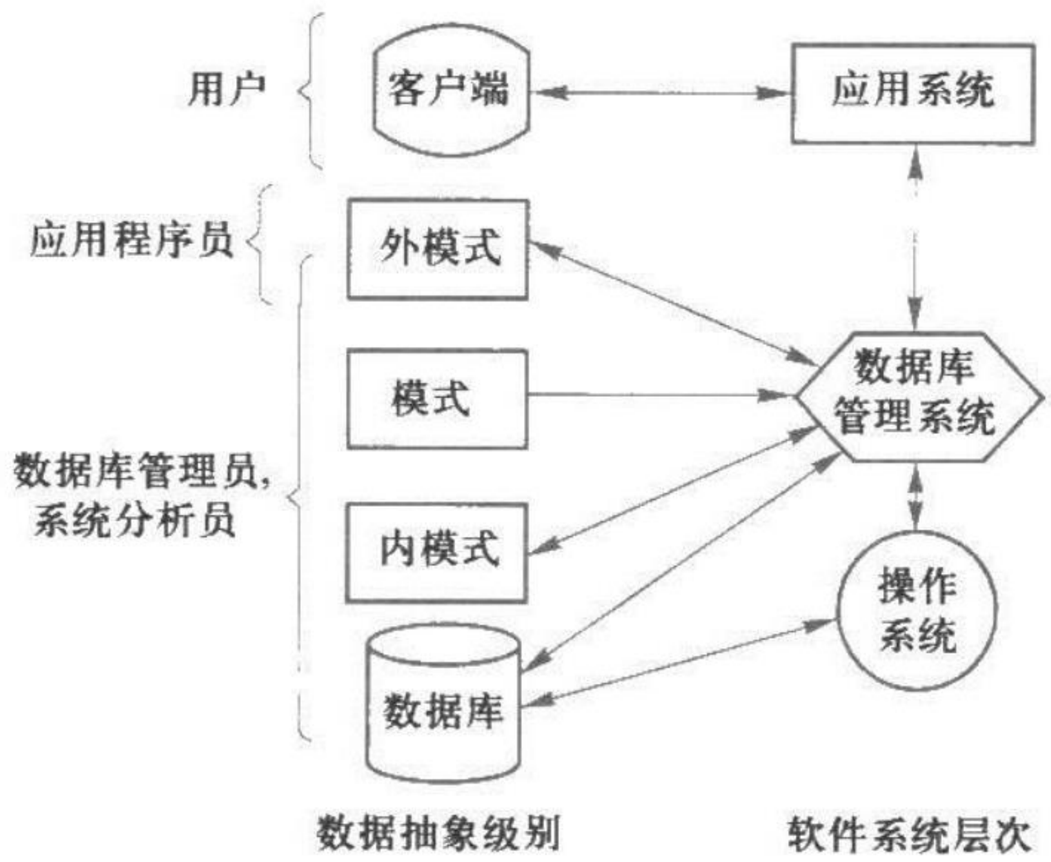


图 1-3　各种人员的数据视图

（1）数据库管理员（Data Base Administrator，DBA）

数据库管理员负责全面管理和控制数据库系统，具体职责包括：

①决定数据库中的信息内容和结构。

②决定数据库的存储结构和存取策略。

③定义数据的安全性要求和完整性约束条件。

④监控数据库的使用和运行。

⑤数据库的改进和重组、重构。

（2）系统分析员和数据库设计人员

系统分析员负责应用系统的需求分析和规范说明，要和用户及数据库管理员相结合，确定系统的硬件软件配置，并参与数据库系统的概要设计。数据库设计人员负责数据库中数据的确定及数据库各级模式的设计。数据库设计人员必须参加用户需求调查和系统分析，然后进行数据库设计。

（3）应用程序员

应用程序员负责设计和编写应用系统的程序模块，并进行调试和安装。

（4）用户

用户是指最终用户（end user）。最终用户通过应用系统的用户接口使用数据库。最终用户三类：

①偶然用户。

②简单用户。

③复杂用户。

1.2　课后习题详解

1．试述数据、数据库、数据库管理系统、数据库系统的概念。

答：（1）数据是数据库中存储的基本对象，是描述事物的符号记录。数据有多种表现形式，它们都可以经过数字化后存入计算机。数据的种类有数字、文字、图形、图像、声音、正文等。

（2）数据库是长期储存在计算机内、有组织的、可共享的大量数据的集合。数据库中的数据按一定的数据模型组织、描述和储存，具有较小的冗余度、较高的数据独立性和易扩展性，并可为各种用户共享。数据库数据具有永久存储、有组织和可共享三个基本特点。

（3）数据库管理系统是位于用户与操作系统之间的一层数据管理软件，用于科学地组织和存储数据、高效地获取和维护数据。

（4）数据库系统是指在计算机系统中引入数据库后的系统，一般由数据库、数据库管理系统（及其开发工具)、应用系统、数据库管理员构成。

2．使用数据库系统有什么好处？

答：使用数据库系统的好处是由数据库管理系统的特点或优点决定的，比如：

（1）可以大大提高应用开发的效率。在数据库系统中，应用程序不必考虑数据的定义、存储和数据存取的具体路径，这些工作都由 DBMS 来完成。开发人员可以专注于应用逻辑的设计，而不必为数据管理的许多复杂的细节操心。

（2）数据库系统提供了数据与程序之间的独立性。当应用逻辑发生改变，数据的逻辑结构需要改变时，DBA 负责修改数据的逻辑结构，开发人员不必修改应用程序，或者只需要修改很少的应用程序，从而既简化了应用程序的编制，又大大减少了应用程序的维护和修改，方便用户的使用。

（3）使用数据库系统可以减轻数据库系统管理人员维护系统的负担。因为 DBMS 在数据库建立、运用和维护时对数据库进行统一的管理和控制，包括数据的完整性、安全性、多用户并发控制、故障恢复等，都由 DBMS 执行。

总之，使用数据库系统的优点很多，既便于数据的集中管理，控制数据冗余，提高数据的利用率和一致性，又有利于应用程序的开发和维护。

3．试述文件系统与数据库系统的区别和联系。

答：（1）文件系统与数据库系统的区别：文件系统面向某一应用程序，共享性差，冗余度大，数据独立性差，记录内有结构，整体无结构，由应用程序自己控制。数据库系统面向现实世界，共享性高，冗余度小，具有较高的物理独立性和一定的逻辑独立性，整体结构化，用数据模型描述，由数据库管理系统提供数据的安全性、完整性、并发控制和恢复能力。

（2）文件系统与数据库系统的联系：文件系统与数据库系统都是计算机系统中管理数据的软件。文件系统是操作系统的重要组成部分；而 DBMS 是独立于操作系统的软件。DBMS 是在操作系统的基础上实现的；数据库中数据的组织和存储是通过操作系统中的文件系统来实现的。

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

答：适合用文件系统而不是数据库系统的应用例子：数据的备份，软件或应用程序使用过程中的临时数据存储一般使用文件系统比较合适。功能比较简单、比较固定的应用系统也适合用文件系统。

适合用数据库系统而非文件系统的应用例子：目前，几乎所有企业或部门的信息系统都以数据库系统为基础，都使用数据库。例如，一个工厂的管理信息系统（其中包括许多子系统，如库存管理系统、物资采购系统、作业调度系统、设备管理系统、人事管理系统等），学校的学生管理系统，人事管理系统，图书馆的图书管理系统等等，都适合用数据库系统。

5．试述数据库系统的特点。

答：数据库系统的主要特点有：

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

（2）数据的共享性高，冗余度低，易扩充。数据库的数据不再面向某个应用而是面向整个系统，因此可以被多个用户、多个应用以多种不同的语言共享使用。由于数据面向整个系统，是有结构的数据，不仅可以被多个应用共享使用，而且容易增加新的应用，这就使得数据库系统弹性大，易于扩充。

（3）数据独立性高。数据独立性包括数据的物理独立性和数据的逻辑独立性。数据库管理系统的模式结构和二级映像功能保证了数据库中的数据具有很高的物理独立性和逻辑独立性。

（4）数据由 DBMS 统一管理和控制。数据库的共享是并发的共享，即多个用户可以同时存取数据库中的数据甚至可以同时存取数据库中同一个数据。为此，DBMS 必须提供统一的数据控制功能，包括数据的安全性保护、数据的完整性检查、并发控制和数据库恢复。

6．数据库管理系统的主要功能有哪些？

答：数据库管理系统的主要功能有：

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

（2）数据组织、存储和管理功能。通过对数据的组织和存储提高存储空间利用率和方便存取，数据库管理系统提供多种存取方法(如索引查找、Hash 查找、顺序查找等)来提高存取效率。

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

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

（5）数据库的建立和维护功能。数据库初始数据的输入、转换功能，数据库的转储、恢复功能，数据库的重组织功能和性能监视、分析功能等。这些功能通常是由一些实用程序或管理工具完成的。

（6）其他功能。例如 DBMS 与网络中其他软件系统的通信功能；一个 DBMS 与另一个 DBMS 或文件系统的数据转换功能；异构数据库之间的互访和互操作功能等。

7．什么是概念模型？试述概念模型的作用。

答：（1）数据模型是对现实世界数据特征的抽象，用来描述数据、组织数据和对数据进行操作。

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

①数据结构：它是所研究的对象类型的集合，是对系统静态特性的描述。

②数据操作：是指对数据库中各种对象（型）的实例（值）允许进行的操作的集合，包括操作及有关的操作规则，是对系统动态特性的描述。

③完整性约束条件：数据的约束条件是一组完整性规则的集合。完整性规则是给定的数据模型中数据及其联系所具有的制约和依存规则，用以限定符合数据模型的数据库状态以及状态的变化，以保证数据的正确、有效、相容。

（2）概念模型的作用：概念模型实际上是现实世界到机器世界的一个中间层次。概念模型用于信息世界的建模，是现实世界到信息世界的第一层抽象，是数据库设计人员进行数据库设计的有力工具，也是数据库设计人员和用户之间进行交流的语言。

8．定义并解释概念模型中以下术语： 实体，实体型，实体集，实体之间的联系。

答：（1）实体：客观存在并可以相互区分的事物。

（2）实体型：具有相同属性的实体具有相同的特征和性质，用实体名及其属性名集合来抽象和刻画同类实体。

（3）实体集：同型实体的集合。

（4）实体联系图（E-R 图）：提供了表示实体型、属性和联系的方法。

①实体型：用矩形表示，矩形框内写明实体名；

②属性：用椭圆形表示，用无向边将其与相应的实体连接起来；

③联系：用菱形表示，菱形框内写明联系名，并用无向边分别与有关实体连接起来，同时在无向边旁标上联系的类型（1:1，1:n 或 m:n）。

9．试述数据模型的概念、数据模型的作用和数据模型的三个要素。

答：（1）数据模型是对现实世界数据特征的抽象，一般来讲，数据模型是严格定义的概念的集合。

（2）数据模型用来描述数据、组织数据和对数据进行操作。这些概念精确描述了系统的静态特性、动态特性和完整性约束条件。

（3）数据模型通常由数据结构、数据操作和完整性约束三部分组成：

①数据结构：它是所研究的对象类型的集合，是对系统静态特性的描述。

②数据操作：是指对数据库中各种对象（型）的实例（值）允许进行的操作的集合，包括操作及有关的操作规则，是对系统动态特性的描述。

③完整性约束条件：数据的约束条件是一组完整性规则的集合。完整性规则是给定的数据模型中数据及其联系所具有的制约和依存规则，用以限定符合数据模型的数据库状态以及状态的变化，以保证数据的正确、有效、相容。

10．试述层次模型的概念，举出三个层次模型的实例。

（1）层次模型满足如下条件：有且只有一个结点没有双亲结点，这个结点称为根结点；根以外的其他结点有且只有一个双亲结点。

（2）三个层次模型的实例：

①教员学生层次数据库模型如图 1-4 所示：

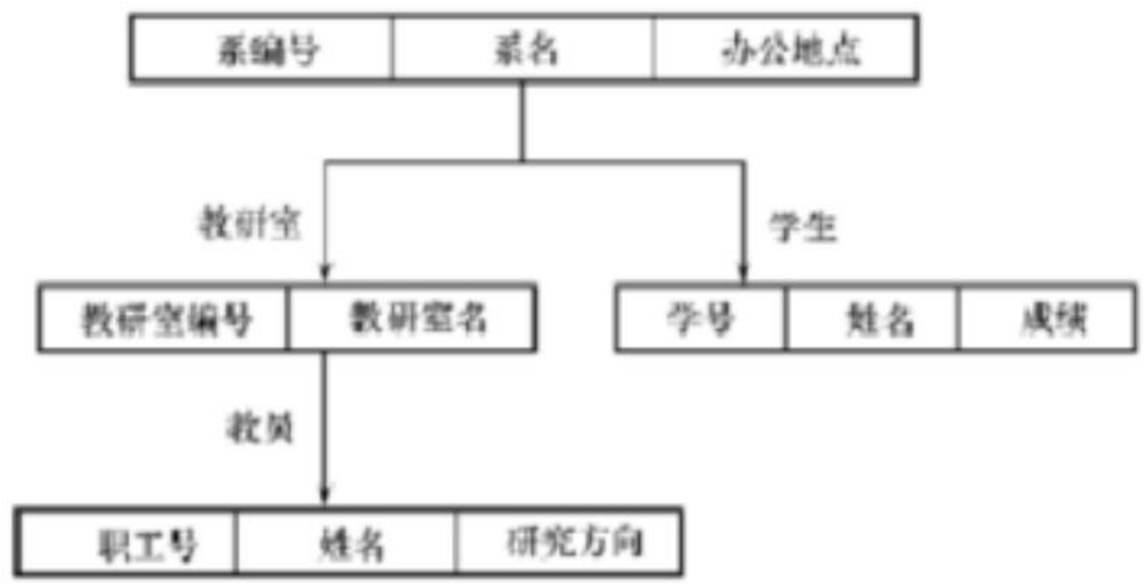


图 1-4

②行政机构层次数据库模型如图 1-5 所示：

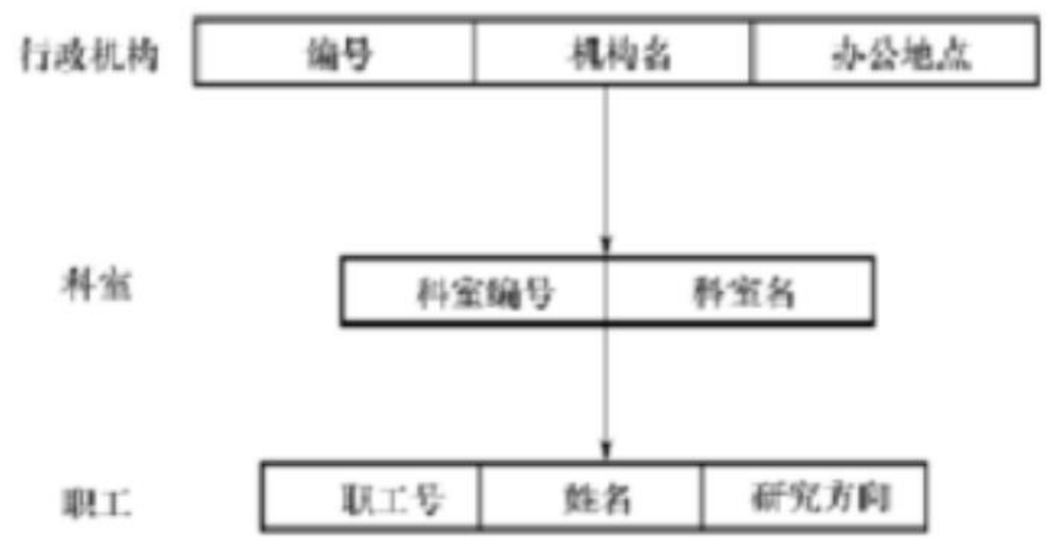


图 1-5

③行政区域层次数据库模型如图 1-6 所示：

图 1-6

11．试述网状模型的概念，举出三个网状　　模型的实例。

答：（1）满足下面两个条件的基本层次联系集合为网状模型。

①允许一个以上的结点无双亲；

②一个结点可以有多于一个的双亲。

（2）三个网状模型的实例；

①实例 1：

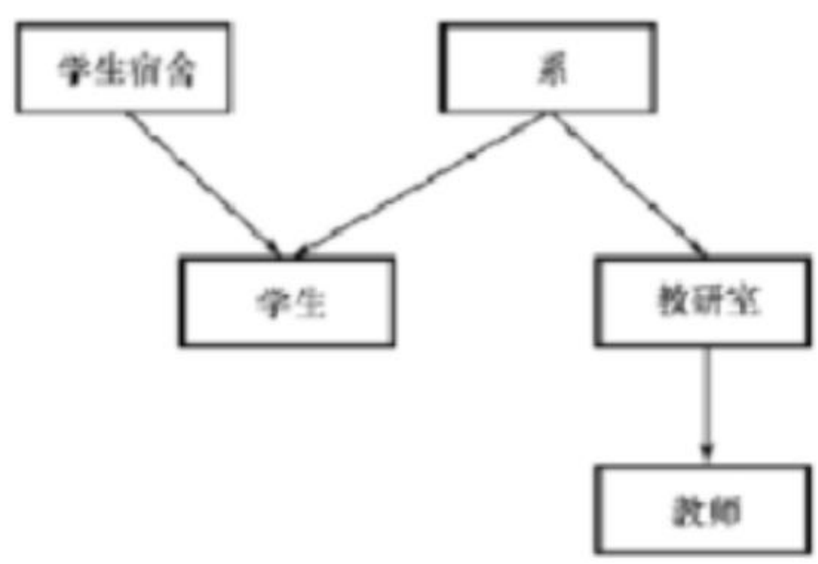


图 1-7

②实例 2：

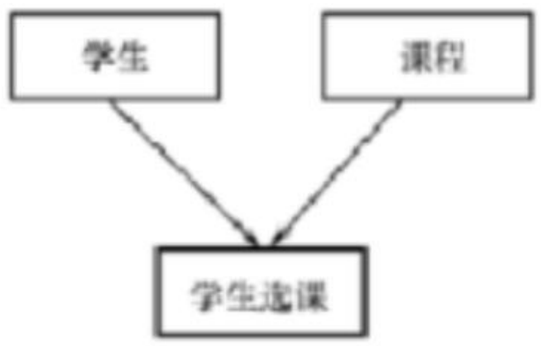


图 1-8

③实例 3：

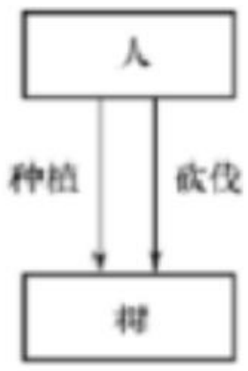


图 1-9

12．试述网状、层次数据库的优缺点。

答：（1）网状数据模型的优缺点：

优点：

①能够更为直接地描述现实世界，如一个结点可以有多个双亲。

②具有良好的性能，存取效率较高。

缺点：

①结构比较复杂，而且随着应用环境的扩大，数据库的结构会变得越来越复杂，不利于最终用户掌握。

②其 DDL、DML 语言复杂，用户不容易使用。网状数据模型记录之间的联系通过存取路径实现，应用程序在访问数据时必须选择适当的存取路径。因此，用户必须了解系统结构的细节，加重了编写应用程序的负担。

（2）层次模型的优缺点：

优点：

①模型简单，对具有一对多层次关系的部门描述非常自然、直观，容易理解。

②用层次模型的应用系统性能好，特别是对于那些实体间联系是固定的且预先定义好的应用，采用层次模型来实现，其性能优于关系模型。

③层次数据模型提供了良好的完整性支持。

缺点：

①现实世界中很多联系是非层次性的，如多对多联系、一个结点具有多个双亲等，层次模型不能自然地表示这类联系，只能通过引入冗余数据或引入虚拟结点来解决。

②对插入和删除操作的限制比较多。

③查询子女结点必须通过双亲结点。

13．试述关系模型的概念，定义并解释以下术语：关系，属性，域，元组，码，分量，关系模式

答：（1）关系模型由关系数据结构、关系操作集合和关系完整性约束三部分组成。在用户观点中，关系模型中数据的逻辑结构是一张二维表，由行和列组成。

（2）术语的定义和解释：

①关系：一个关系对应通常所说的一张表。

②属性：表中的一列即为一个属性。

③域：属性的取值范围。

④元组：表中的一行即为一个元组。

⑤主码：表中的某个属性组，它可以惟一确定一个元组。

⑥分量：元组中的一个属性值。

⑦关系模式：对关系的描述，一般表示为关系名（属性 1，属性 2， …，属性 n）。

14．试述关系数据库的特点。

答：关系数据模型具有下列优点：

（1）关系模型与非关系模型不同，它是建立在严格的数学概念基础上的。

（2）关系模型的概念单一，无论实体还是实体之间的联系都用关系来表示。对数据的检索和更新结果也是关系(即表)。所以其数据结构简单、清晰，用户易懂易用。

（3）关系模型的存取路径对用户透明，从而具有更高的数据独立性、更好的安全保密性，同时也简化了程序员的工作和数据库开发建立的工作。所以关系数据模型诞生以后发展迅速，深受用户的喜爱。

当然，关系数据模型也有缺点，其中最主要的缺点是，由于存取路径对用户透明，查询效率往往不如格式化数据模型。因此为了提高性能，DBMS 必须对用户的查询请求进行优化因此增加了开发 DBMS 的难度。不过，用户不必考虑这些系统内部的优化技术细节。

15．试述数据库系统的三级模式结构，并说明这种结构的优点是什么。

答：（1）数据库系统的三级模式结构由外模式、模式和内模式组成。

①外模式，亦称子模式或用户模式，是数据库用户（包括应用程序员和最终用户）能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。

②模式，亦称逻辑模式，是数据库中全体数据的逻辑结构和特征的描述，是所有用户的公共数据视图。模式描述的是数据的全局逻辑结构。外模式涉及的是数据的局部逻辑结构，通常是模式的子集。

③内模式，亦称存储模式，是数据在数据库系统内部的表示，即对数据的物理结构和存储方式的描述。

（2）数据库系统的三级模式是对数据的三个抽象级别，它把数据的具体组织留给 DBMS 管理，使用户能逻辑抽象地处理数据，而不必关心数据在计算机中的表示和存储。为了能够在内部实现这三个抽象层次的联系和转换，数据库系统在这三级模式之间提供了两层映像：外模式/模式映像和模式/内模式映像。正是这两层映像保证了数据库系统中的数据能够具有较高的逻辑独立性和物理独立性。

16．定义并解释以下术语：模式，外模式，内模式，数据定义语言，数据操纵语言。

答：（1）外模式：亦称子模式或用户模式，数据库用户（包括应用程序员和最终用户）能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。

(2)模式：亦称逻辑模式，是数据库中全体数据的逻辑结构和特征的描述，是所有用户的公共数据视图。模式描述的是数据的全局逻辑结构。外模式涉及的是数据的局部逻辑结构，通常是模式的子集。

（3）内模式：亦称存储模式，是数据在数据库系统内部的表示，即对数据的物理结构和存储方式的描述。

（4）数据定义语言（Data Definition Language，DDL）：用来定义数据库模式、外模式、内模式的语言。

（5）数据操纵语言(Data Manipulation Language，DML)：用来对数据库中的数据进行查询、插入、删除和修改的语句。

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

答：（1）数据与程序的物理独立性：当数据库的存储结构改变，由数据库管理员对模式/内模式映像做相应改变，可以使模式保持不变，从而应用程序也不必改变，保证了数据与程序的物理独立性，简称数据的物理独立性。

（2）数据与程序的逻辑独立性：当模式改变时（例如增加新的关系、新的属性、改变属性的数据类型等），由数据库管理员对各个外模式/模式的映像做相应改变，可以使外模式保持不变。应用程序是依据数据的外模式编写的，从而应用程序不必修改，保证了数据与程序的逻辑独立性，简称数据的逻辑独立性。

（3）数据库管理系统在三级模式之间提供的两层映像保证了数据库系统中的数据能够具有较高的逻辑独立性和物理独立性。

18．试述数据库系统的组成。

答：数据库系统一般由数据库、数据库管理系统（及其开发工具）、应用系统、数据库管理员和用户构成。

（1）硬件平台及数据库。由于数据库系统数据量都很大，加之 DBMS 丰富的功能使得自身的规模也很大，因此整个数据库系统对硬件资源提出了较高的要求：①要有足够大的内存，存放操作系统、DBMS 的核心模块、数据缓冲区和应用程序；②有足够的大的磁盘或磁盘阵列等设备存放数据库，有足够的磁带(或光盘)作数据备份；③要求系统有较高的通道能力，以提高数据传送率。

（2）软件。①DBMS；②支持 DBMS 运行的操作系统；③具有与数据库接口的高级语言及其编译系统；④以 DBMS 为核心的应用开发工具；⑤为特定应用环境开发的数据库应用系统。

（3）人员。开发、管理和使用数据库系统的人员主要是：数据库管理员、系统分析员和数据库设计人员、应用程序员和最终用户。

19．试述数据库管理员、系统分析员、数据库设计人员、应用程序员的职责。

答：（1）数据库管理员：负责全面地管理和控制数据库系统。具体职责包括：

①决定数据库的信息内容和结构。

②决定数据库的存储结构和存取策略。

③定义数据的安全性要求和完整性约束条件。

④监督和控制数据库的使用和运行。

⑤数据库的改进和重组重构。

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

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

（4）应用程序员：应用程序员负责设计和编写应用系统的程序模块，并进行调试和安装。

第 2 章　关系数据库

2.1　复习笔记

一、关系数据结构及形式化定义

关系数据库系统是支持关系模型的数据库系统。按照数据模型的三个要素，关系模型由关系数据结构、关系操作集合和关系完整性约束三部分组成。

1．关系

关系模型的数据结构只包含单一的数据结构-关系，能够描述出现实世界的实体以及实体间的各种联系。在关系模型中，现实世界的实体以及实体间的各种联系均用单一的结构类型，即关系来表示。

（1）域（domain）

域是一组具有相同数据类型的值的集合。

（2）笛卡儿积（cartesian product）

笛卡儿积是域上的一种集合运算。D_1，D_2，…，D_n 的笛卡尔积为 $D_1 \times D_2 \times \cdots \times D_n = \{(d_1, d_2, \cdots, d_n) \mid d_i \in D_i, i=1, 2, \cdots, n\}$，其中每一个元素 $(d_1, d_2, \cdots, d_n)$ 是一个 n 元组(n-tuple)或简称元组(Tuple)。笛卡尔积可表示为一个二维表。表中的每行对应一个元组，表中的每一列的值来自一个域。

（3）关系（relation）

$D_1 \times D_2 \times \ldots \times D_n$ 的子集称作在域 D_1，D_2，…，D_n 上的关系，表示为 R（D_1，D_2，…，D_n）。R 表示关系的名字，n 是关系的目或度(Degree)。关系中的每个元素是关系中的元组，通常用 t 表示。关系是笛卡儿积的有限子集，所以关系也是一张二维表，表的每行对应一个元组，表的每列对应一个域。若关系中的某一属性组的值能唯一地标识一个元组，则称该属性组为候选码（candidate key）。若一个关系有多个候选码，则选定其中一个为主码（primary key）。候选码的诸属性称为主属性（prime attribute）。不包含在任何候选码中的属性称为非主属性（non．prime attribute）或非码属性（non—key attribute）。

基本关系的性质为：

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

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

③列的顺序无所谓，即列的次序可以任意交换。

④任意两个元组的候选码不能取相同的值。

⑤行的顺序无所谓，即行的次序可以任意交换。

⑥分量必须取原子值，即每一个分量都必须是不可分的数据项。

2．关系模式

关系数据库中，关系模式是型，关系是值。关系模式是对关系的描述。关系的描述称为关系模式（relation schema），它可以形式化地表示为 R（U，D，DOM，F），其中 R 为关系名，U 为组成该关系的属性名集合，D 为 U 中属性所来自的域，DOM 为属性向域的映像集合，F 为属性间数据的依赖关系集合。关系是关系模式在某一时刻的状态或内容。关系模式是静态的、稳定的，而关系是动态的、随时间不断变化的。

3．关系数据库

在关系模型中，实体以及实体间的联系都是用关系来表示的。在一个给定的应用领域中，所有关系的集合构成一个关系数据库。关系数据库也有型和值之分。关系数据库的型也称为关系数据库模式，是对关系数据库的描述。关系数据库模式包括若干域的定义，以及在这些域上定义的若干关系模式。关系数据库的值是这些关系模式在某一时刻对应的关系的集合，通常就称为关系数据库。

4．关系模型的存储结构

在关系数据库的物理组织中，有的关系数据库管理系统中一个表对应一个操作系统文件，将物理数据组织交给操作系统完成；有的关系数据库管理系统从操作系统那里申请若干个大的文件，自己划分文件空间，组织表、索引等存储结构，并进行存储管理。

二、关系操作

1．基本的关系操作

（1）查询（query）操作

关系的查询表达能力很强，是关系操作中最主要的部分。查询操作又可以分为选择（select）、投影（project）、连接（join）、除（divide）、并（union）、差（except）、交（intersection）、笛卡儿积(Cartesian Product)等。

（2）插入（insert）、删除（delete）、修改（update）操作。

2．关系数据语言的分类

关系数据语言可以分为三类，如图 2-1 所示。

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

图 2-1　关系数据语言分类

三、关系的完整性

关系模型的完整性规则是对关系的某种约束条件。关系模型中有三类完整性约束：实体完整性（entity integrity）、参照完整性（referential integrity）和用户定义的完整性（user-defined integnty）。

1．实体完整性

（1）定义

若属性（指一个或一组属性）A 是基本关系 R 的主属性，则 A 不能取空值（null value）。所谓空值就是“不知道”或“不存在”或“无意义”的值。如果主码由若干属性组成，则所有这些主属性都不能取空值。

（2）规则说明

①实体完整性规则是针对基本关系而言的。一个基本表通常对应现实世界的一个实体集。

②现实世界中的实体是可区分的，即它们具有某种唯一性标识。

③以主码作为唯一性标识。

④主属性不能取空值。

2．参照完整性

（1）定义

设 F 是基本关系 R 的一个或一组属性，但不是关系 R 的码，K_s 是基本关系 S 的主码。如果 F 与 K_s 相对应，则称 F 是 R 的外码（foreign key），并称基本关系 R 为参照关系（referencing relation），基本关系 S 为被参照关系（referenced relation）或目标关系（target relation）。

（2）参照完整性规则

参照完整性规则就是定义外码与主码之间的引用规则。若属性（或属性组）F 是基本关系 R 的外码，它与基本关系 S 的主码 K_s 相对应（基本关系 R 和 S 不一定是不同的关系），则对于 R 中每个元组在 F 上的值必须取空值（F 的每个属性值均为空值），或者等于 S 中某个元组的主码值。

3．用户定义的完整性

用户定义的完整性是针对某一具体关系数据库的约束条件，它反映某一具体应用所涉及的数据必须满足的语义要求。关系模型应提供定义和检验这类完整性的机制，以便用统一的系统的方法处理它们，而不需由应用程序承担这一功能。

四、关系代数

关系代数是一种抽象的查询语言,它用对关系的运算来表达查询。运算的三大要素为：运算对象、运算符、运算结果。关系代数的运算对象是关系，运算结果亦为关系。关系代数用到的运算符包括两类：集合运算符和专门的关系运算符。

1．传统的集合运算

设关系 R 和关系 S 具有相同的目 n（即两个关系都有 n 个属性），且相应的属性取自同一个域，t 是元组变量，t∈R 表示 t 是 R 的一个元组。

（1）并(Union)

关系 R 与关系 S 的并记作：

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

其结果仍为 n 目关系，由属于 R 或属于 S 的元组组成。

（2）差(Except)

关系 R 与关系 S 的差记作：

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

其结果关系仍为 n 目关系，由属于 R 而不属于 S 的所有元组组成。

（3）交(Intersection)

关系 R 与关系 S 的交记作：

$$\mathbf{R} \cap \mathbf{S} = \{\mathbf{t} \mid \mathbf{t} \in \mathbf{R} \wedge \mathbf{t} \in \mathbf{S}\}$$

其结果关系仍为 n 目关系，由既属于 R 又属于 S 的元组组成。

（4）笛卡尔积(Cartesian Product)

两个分别为 n 目和 m 目的关系 R 和 S 的笛卡尔积是一个 n+m 列的元组的集合。元组的前 n 列是关系 R 的一个元组，后 m 列是关系 S 的一个元组。若 R 有 k_1 个元组，S 有 k_2 个元组，则关系 R 和关系 S 的笛卡尔积有 $K_1 \times K_2$ 个元组。记作：

$$\mathbf{R} \times \mathbf{S} = \{\mathbf{t_r t_s} \mid \mathbf{t_r} \in \mathbf{R} \wedge \mathbf{t_s} \in \mathbf{S}\}$$

2．专门的关系运算

（1）选择（selection）

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

$$\sigma_{\mathbf{F}}(\mathbf{R}) = \{\mathbf{t} \mid \mathbf{t} \in \mathbf{R} \wedge \mathbf{F}(\mathbf{t}) = \text{'真'}\}$$

其中 F 表示选择条件，它是一个逻辑表达式，取逻辑值“真”或“假”。逻辑表达式 F 的基本形式为：

$X_1 \theta Y_1$

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

(2)投影（projection）

关系 R 上的投影是从 R 中选择出若干属性列组成新的关系。记作：

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

其中 A 为 R 中的属性列。投影操作是从列的角度进行的运算。

(3)连接（join）

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

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

其中，A 和 B 分别为 R 和 S 上列数相等且可比的属性组，A 是比较运算符。连接运算从 R 和 S 的笛卡儿积 R×S 中选取 R 关系在 A 属性组上的值与 S 关系在 B 属性组上的值满足比较关系 θ 的元组。θ 为“=”的连接运算称为等值连接。它是从关系 R 与 S 的广义笛卡儿积中选取 A、B 属性值相等的那些元组，即等值连接为：

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

自然连接是一种特殊的等值连接。它要求两个关系中进行比较的分量必须是同名的属性组，并且在结果中把重复的属性列去掉。即若 R 和 S 中具有相同的属性组 B，U 为 R 和 S 的全体属性集合，则自然连接可记作：

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

（4）除运算（division）

给定关系 R（X，Y）和 S（Y，Z），其中 X、Y、Z 为属性组。R 中的 Y 与 S 中的 Y 可以有不同的属性名，但必须出自相同的域集。

R 与 S 的除运算得到一个新的关系 P（X），P 是 R 中满足下列条件的元组在 X 属性列上的投影：元组在 X 上分量值 x 的象集 Y_x 包含 S 在 Y 上投影的集合。记作：

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

其中 Y_x 为 x 在 R 中的象集。

五、关系演算

关系演算是以数理逻辑中的谓词演算为基础的。按谓词变化的不同，关系演算可分为元组关系演算和域关系演算。

1．元组关系演算语言 ALPHA

元组关系演算以元组变量作为谓词变元的基本对象。ALPHA 语言主要有 GET、PUT、HOLD、UPDATE、

DELETE、DROP 6 条语句，语句的基本格式为：

{操作语句 工作空间名（表达式）：操作条件}

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

（1）检索操作

检索操作用 GET 语句实现。检索操作包括：简单检索(即不带条件的检索)，例如：

GET W （SC．Cno）

W 为工作空间名。这里条件为空，表示没有限定条件。

限定的检索（即带条件的检索），例如：

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

还包括带排序的检索；指定返回元组条数的检索；用元组变量的检索；用存在量词的检索；带有多个关系的表达式的检索；用全称量词的检索；用两种量词的检索；用蕴涵的检索；聚集函数等。

（2）更新操作

①修改操作

修改操作用 UPDATE 语句实现。其步骤是：

a．首先用 HOLD 语句将要修改的元组从数据库中读到工作空间中；

b．然后用宿主语言修改工作空间中元组的属性值；

c．最后用 UPDATE 语句将修改后的元组送回数据库中。

②插入操作

插入操作用 PUT 语句实现。其步骤是：

a．首先用宿主语言在工作空间中建立新元组；

b．用 PUT 语句插入该元组。

③删除

删除操作用 DELETE 语句实现。其步骤为：

a．用 HOLD 语句把要删除的元组从数据库中读到工作空间中；

b．用 DELETE 语句删除该元组。

2．元组关系演算

在元组关系演算系统中，称{t｜φ（t）}为元组演算表达式。其中 t 是元组变量，φ（t）为元组关系演算公式，简称公式，它由原子公式和运算符组成。

（1）原子公式的分类

①R（t）

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

②t[i]θu[j]

t 和 u 是元组变量，θ 是算术比较运算符。t[i]θu[j]表示断言元组 t 的第 i 个分量与元组的第 i 个分量满足比较关系 θ。

③t[i]θc 或 cθt[i]

这里 c 是常量，该公式表示 t 的第 i 个分量与常量 c 满足比较关系 θ。

（2）元组运算符的优先次序

①算术比较运算符最高。

②量词次之，且∃的优先级高于∀的优先级。

③逻辑运算符最低，且1的优先级高于八的优先级，∧的优先级高于 V 的优先级。

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

（3）基本运算

①并

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

②差

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

③笛卡儿积

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

这里 $t^{(n+m)}$ 表示 t 的目数是（n+m）。

④投影

$$\Pi_{i_1,\ i_2,\ \cdots,\ i_k}(R)=\{t^{(k)}|(\exists u)(R(u)\wedge t[1]=u[i_1]\wedge\cdots\wedge t[k]=u[i_k])\}$$

⑤选择

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

F'公式 F 用 t[i]代替运算对象 i 得到的等价公式。

3．域关系演算语言 QBE

关系演算的另一种形式是域关系演算。域关系演算以元组变量的分量（即域变量）作为谓词变元的基本对象。QBE 是 QueryBvExample（即通过例子进行查询）的简称，它最突出的特点是操作方式。它是一种高度非过程化的基于屏幕表格的查询语言，用户通过终端屏幕编辑程序，以填写表格的方式构造查询要求，而查询结果也是以表格形式显示，因此非常直观、易学易用。QBE 中用示例元素来表示查询结果可能的情况，示例元素实质上就是域变量。

2.2 课后习题详解

1．试述关系模型的 3 个组成部分。

答：关系模型由关系数据结构、关系操作集合和关系完整性约束三部分组成。

（1）关系数据结构：在关系模型中，现实世界的实体以及实体间的各种联系均用单一的结构类型即关系来表示。

（2）关系操作集合：关系模型中常用的关系操作包括查询操作和插入、删除、修改操作。

（3）关系完整性约束：关系模型中有实体完整性约束、参照完整性约束和用户定义的完整性约束三类约束。

2．简述关系数据语言的特点和分类。

答：（1）这些关系数据库语言的共同特点是：都是非过程化的集合操作语言，具有完备的表达能力，功能强，能够嵌入高级语言中使用。

（2）关系数据语言分为三类：

①关系代数语言。关系代数是用对关系的运算来表达查询要求的。

②关系演算语言。关系演算是用谓词来表达查询要求的。

③SQL。具有丰富的查询功能，而且具有数据定义和数据控制功能，是集查询 DDL、DML 和 DCL 于一体的关系数据语言。

3．定义并理解下列术语，说明它们之间的联系与区别：

（1）域，笛卡儿积，关系，元组，属性；

（2）主码，候选码，外码；

（3）关系模式，关系，关系数据库。

答：（1）域，笛卡尔积，关系，元组，属性

①域：一组具有相同数据类型的值的集合。

②笛卡儿积：两个分别为 n 目和 m 目的关系 R 和 S 的笛卡尔积是一个 n+m 列的元组的集合。

③关系：在域 D_1，D_2，…，D_n 上笛卡尔积 $D_1\times D_2\times\cdots\times D_n$ 的子集称为关系，表示为 R（D_1，D_2，…，D_n）。

④元组：关系中的每个元素是关系中的元组。

⑤属性：关系也是一个二维表，表的每行对应一个元组，表的每列对应一个域。由于域可以相同，为了加以区分，必须对每列起一个名字，称为属性。

（2）主码，候选码，外部码

①候选码：关系中能惟一标识一个元组的某一属性组。

②主码：若一个关系有多个候选码，则选定其中一个为主码。

③外部码：设 F 是基本关系 R 的一个或一组属性，但不是关系 R 的码，如果 F 与基本关系 S 的主码 K_s 相对应，则称 F 是基本关系 R 的外部码，简称外码。

（3）关系模式，关系，关系数据库

①关系模式：关系的描述称为关系模式，它可以形式化地表示为：R（U，D，Dom，F），其中 R 为关系名，U 为组成该关系的属性名集合，D 为属性组 U 中属性所来自的域，Dom 为属性向域的映射集合，F 为属性间数据的依赖关系集合。

②关系：在域 D_1，D_2，…，D_n 上笛卡尔积 $D_1 \times D_2 \times \cdots \times D_n$ 的子集称为关系，表示为 R（D_1，D_2，…，D_n）。关系是关系模式在某一时刻的状态或内容。关系模式是静态的、稳定的，而关系是动态的、随时间不断变化的，因为关系操作在不断地更新着数据库中的数据。

③关系数据库：关系数据库有型和值之分。关系数据库的型也称为关系数据库模式，是对关系数据库的描述，它包括若干域的定义以及在这些域上定义的若干关系模式。关系数据库的值是这些关系模式在某一时刻对应的关系的集合，通常就称为关系数据库。

4．举例说明关系模式和关系的区别。

答：关系模式是静态的，关系是动态的。对于常见的二维表，关系模式通常指的是二维表的表头，即有哪些列构成，每个列的名称、类型、长度等。关系通常指的是一张表的具体内容，因为表经常进行插入、删除、修改等操作，关系可能不一样。

5．试述关系模式的完整性规则。在参照完整件中，什么情况下外码属性的值可以为空值？

答：（1）关系模型的完整性规则是对关系的某种约束条件。关系模型中可以有三类完整性约束：实体完整性、参照完整性和用户定义的完整性。

①实体完整性规则：若属性 A 是基本关系 R 的主属性，则属性 A 不能取空值。

②参照完整性规则：若属性（或属性组）F 是基本关系 R 的外码，它与基本关系 S 的主码 Ks 相对应（基本关系 R 和 S 不一定是不同的关系），则对于 R 中每个元组在 F 上的值必须为取空值（F 的每个属性值均为空值），或者等于 S 中某个元组的主码值。

③用户定义的完整性是针对某一具体关系数据库的约束条件。它反映某一具体应用所涉及 的数据必须满足的语义要求。

（2）在参照完整性中，外码属性值可以为空，它表示该属性的值尚未确定，但前提条件是该外码属性不是其所在参照关系的主属性。

6．设有一个 SPJ 数据库，包括 S、P、J 及 SPJ4 个关系模式：

S(SNO,SNAME,STATUS,CITY)；

P(PNO,PNAME,COLOR,WEIGHT)；

J(JNO,JNAME,CITY)；

SPJ(SNO,PNO,JNO,QTY)。

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

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

工程项目表 J 由工程项目代码（JNO）、工程项目名（JNAME）、工程项目所在城市（CITY）组成。供应情况表 SPJ 由供应商代码（SNO）、零件代码（PNO）、工程项目代码（JNO）、供应数量（QTY）组成，表示某供应商供应某种零件给某工程项目的数量为 QTY。

今有若干数据如下：

S 表

SNO	SNAME	STATUS	CITY
S1	精益	20	天津
S2	盛锡	10	北京
S3	东方红	30	北京
S4	丰泰盛	20	天津
S5	为民	30	上海

P 表

PNO	PNAME	COLOR	WEIOHT
PI	螺母	红	12
P2	螺栓	绿	17
P3	螺丝刀	蓝	14
P4	螺丝刀	红	14
P5	凸轮	蓝	40
P6	齿轮	红	30

J 表

JNO	JNAME	CITY
J1	三建	北京
J2	一汽	长春
J3	弹簧厂	天津
J4	造船厂	天津
J5	机车厂	唐山
J6	无线电厂	常州
J7	半导体厂	南京

SPJ 表

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

试用关系代数、ALPHA 语言、QBE 语言完成如下查询：

（1）求供应工程 J1 零件的供应商号码 SNO；

（2）求供应工程 J1 零件 Pl 的供应商号码 SNO；

（3）求供应 T 程 Jl 零件为红色的供应商号码 SNO；

（4）求没有使用天津供应商生产的红色零件的工程项目代码 JNO；

（5）求至少用了供应商 Sl 所供应的全部零件的工程项目代码 JNO。

答：（1）关系代数、ALPHA 语言、QBE 语言完成如下：

①关系代数： $\pi_{SNO}(\sigma_{JNO='J1'}(SPJ))$

②ALPHA 语言：GET W(SPJ.SNO):SPJ.JNO=’J1’

③QBE 语言：

SPJ	SNO	PNO	JNO	QTY
	P. S1		J1	

（2）关系代数、ALPHA 语言、QBE 语言完成如下：

①关系代数： $\pi_{SNO}(\sigma_{JNO='J1' \wedge PNO='P1'}(SPJ))$

②ALPHA 语言：GET W(SPJ.SNO):SPJ.JNO=’J1’∧SPJ.PNO=’P1’

③QBE 语言：

SPJ	SNO	PNO	JNO	QTY
	P. S1	P1	J1	

（3）关系代数、ALPHA 语言、QBE 语言完成如下：

①关系代数： $\pi_{SNO}(\pi_{SNO,\ PNO}(\sigma_{JNO='J1'}(SPJ)) \bowtie \pi_{PNO}(\sigma_{COLOR='红'}(P)))$

②ALPHA 语言：RANGE P PX

GET W (SPJ.SNO): ∃ PX (PX.PNO=SPJ.PNO ∧ SPJ.JNO='J1'∧ PX .COLOR='红')

③QBE 语言：

SPJ	SNO	PNO	JNO	QTY
	P. S1	P1	J1	

P	PNO	PNAME	COLOR	WEIGHT
	P1		红	

（4）关系代数、ALPHA 语言、QBE 语言完成如下：

①关系代数：

$$\pi_{JNO}(J) - \pi_{JNO}(\pi_{SNO}(\sigma_{CITY='天津'}(S)) \bowtie \pi_{SNO,\ PNO,\ JNO}(SPJ) \bowtie \pi_{PNO}(\sigma_{COLOR='红'}(P)))$$

②ALPHA 语言：

```
RANGE   SPJ      SPJX
         P        PX
         S        SX
GET   W (J.JNO): ¬∃SPJX( SPJX.JNO=J.JNO ∧
                         ∃SX (SX.SNO=SPJX.SNO ∧ SX.CITY='天津') ∧
                         ∃PX(PX .PNO=SPJX.PNO ∧ PX.COLOR='红'))
```

③QBE 语言：

不考虑未使用任何零件的工程。

S	SNO	SNAME	STATUS	CITY
	S1			天津

P	PNO	PNAME	COLOR	WEIGHT
	P1		红	

SPJ	SNO	PNO	JNO	QTY
¬	S1	P1	P、J1	

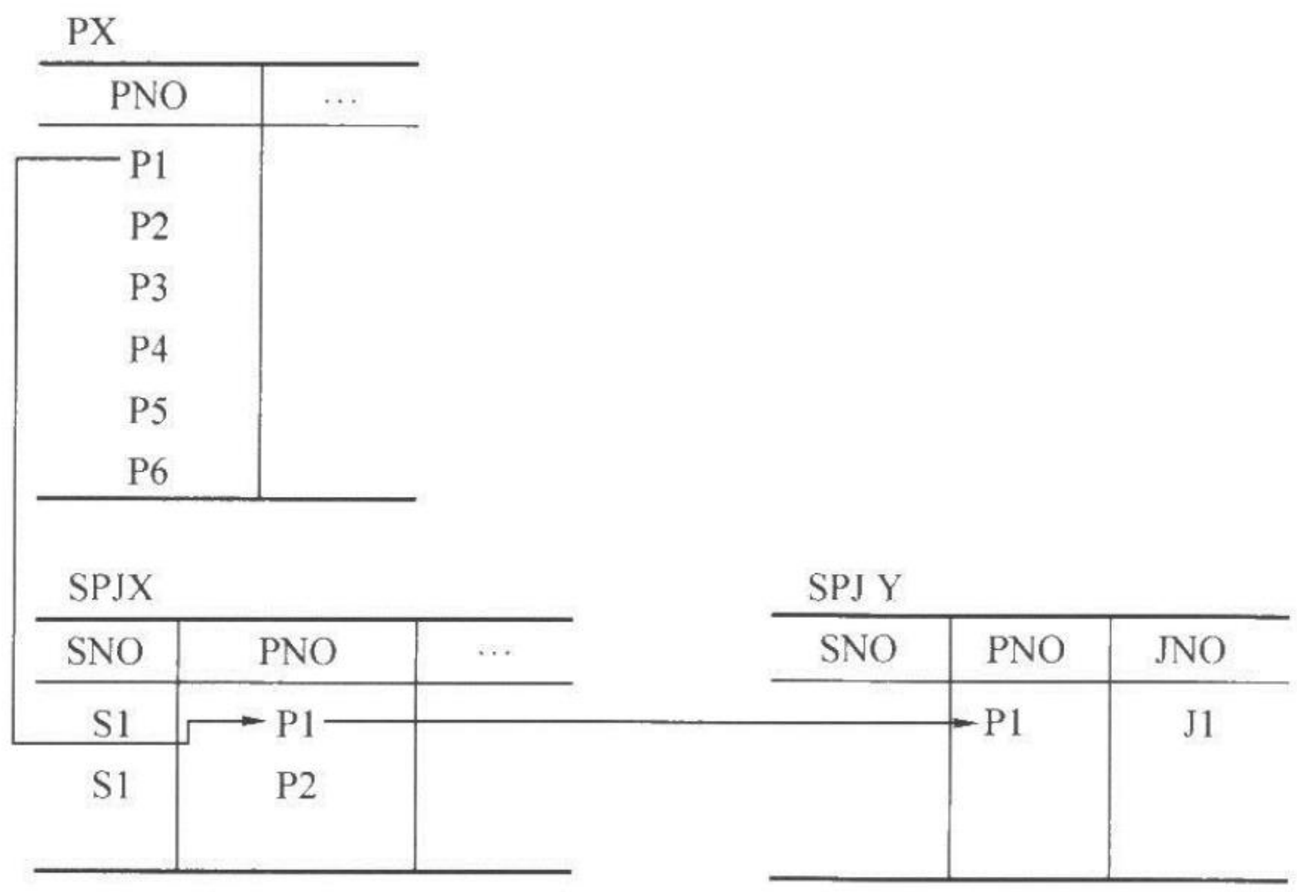
（5）关系代数、ALPHA 语言、QBE 语言完成如下：

①关系代数：$\pi_{JNO,\ PNO}(SPJ) \div \pi_{PNO}(\sigma_{SNO='S1'}(SPJ))$

②ALPHA 语言：

```
RANGE    SPJ   SPJX
         SPJ   SPJY
         P     PX
GET W(J.JNO): ∀PX(∃SPJX(SPJX.PNO=PX.PNO∧SPJX.SNO='S1')
                → ∃SPJY(SPJY.JNO=J.JNO∧SPJY.PNO=PX.PNO))
```

③QBE 语言：



7．试述等值连接与自然连接的区别和联系。

答：连接运算中有两种最为重要也最为常用的连接，一种是等值连接(Equijoin)，另一种是自然连接(Natural join)。θ 为“=”的连接运算称为等值连接。它是从关系 R 与 S 的广义笛卡尔积中选取 A，B 属性值相等的那些元组，即等值连接为

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

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

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

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

8．关系代数的基本运算有哪些？如何用这些基本运算来表示其他运算？

答：关系代数的基本运算包括并、差、笛卡尔积、投影和选择 5 种运算。其他 3 种运算，即交、连接和除，均可以用这 5 种基本运算来表达。

（1）交运算：R∩S=R-(R-S)或 R∩S=S-(S-R)；

（2）连接运算：$R \underset{A\theta B}{\bowtie} S = \sigma_{A\theta B}(R \times S)$；

（3）除运算：$R(X, Y) \div S(Y, Z) = \pi_X(R) - \pi_X(\pi_X(R) \times \pi_Y(S) - R)$。

第 3 章　关系数据库标准语言 SQL

3.1　复习笔记

一、SQL 概述

1．SQL 的产生与发展

SQL 在 1974 年由 **Boyce** 和 **Chamberlin** 提出的，最初叫 **Sequel**，并在 **IBM** 公司研制的关系数据库管理系统原型 **System R** 上实现。**SQL** 简单易学，功能丰富。1986 年 10 月，美国国家标准局（**American National standard Institute AMSI**）的数据厍委员会 **X**3**H**2 批准了 **SQL** 作为关系数据库语言的美国标准，同年公布了 **SQL** 标准文本（简称 **SQL-**86）。1987 年，国际标准化组织（**Intemationaɪ Organization for Standardization**，**ISO**）也通过了这一标准。

2．SQL 的特点

（1）综合统一

数据库系统的主要功能是通过数据库支持的数据语言来实现的。非关系模型（层次模型、网状模型）的数据语言一般都分为：

①模式数据定义语言（**SchemaDataDefinition Language**，模式 **DDL**）。

②外模式数据定义语言（**Subschema Data Definition Language**，外模式 **DDL** 或子模式 **DDL**）。

③数据存储有关的描述语言（**Data StorageDescriptionLanguage**，**DSDL**）。

④数据操纵语言（**Data Manipulation Language**，**DML**）。

（2）高度非过程化

SQL 进行数据操作时，只要提出“做什么”，面无须指明“怎么做”，因此无须了解存取路径。存取路径的选择以及 **SQL** 的操作过程由系统自动完成。

（3）面向集合的操作方式

SQL 采用集合操作方式，不仅操作对象、查找结果可以是元组的集合，而且一次插入、删除、更新操作的对象也可以是元组的集合。

（4）以同一种语法结构提供多种使用方式

SQL 既是独立的语言，又是嵌入式语言。在两种不同的使用方式下，**SQL** 的语法结构基本上是一致的。这种以统一的语法结构提供多种不同使用方式的做法，提供了极大的灵活性与方便性。

（5）语言简洁，易学易用

SQL 功能极强，但由于设计巧妙，语言十分简洁，完成核心功能只用了 9 个动词，如表 3-1 所示。**SQL** 接近英语口语，因此易于学习和使用。

表 3-1　**SOL** 的动词

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

3．SQL 的基本概念

支持 **SQL** 的关系数据库管理系统同样支持关系数据库三级模式结构。如图 3-1 所示。

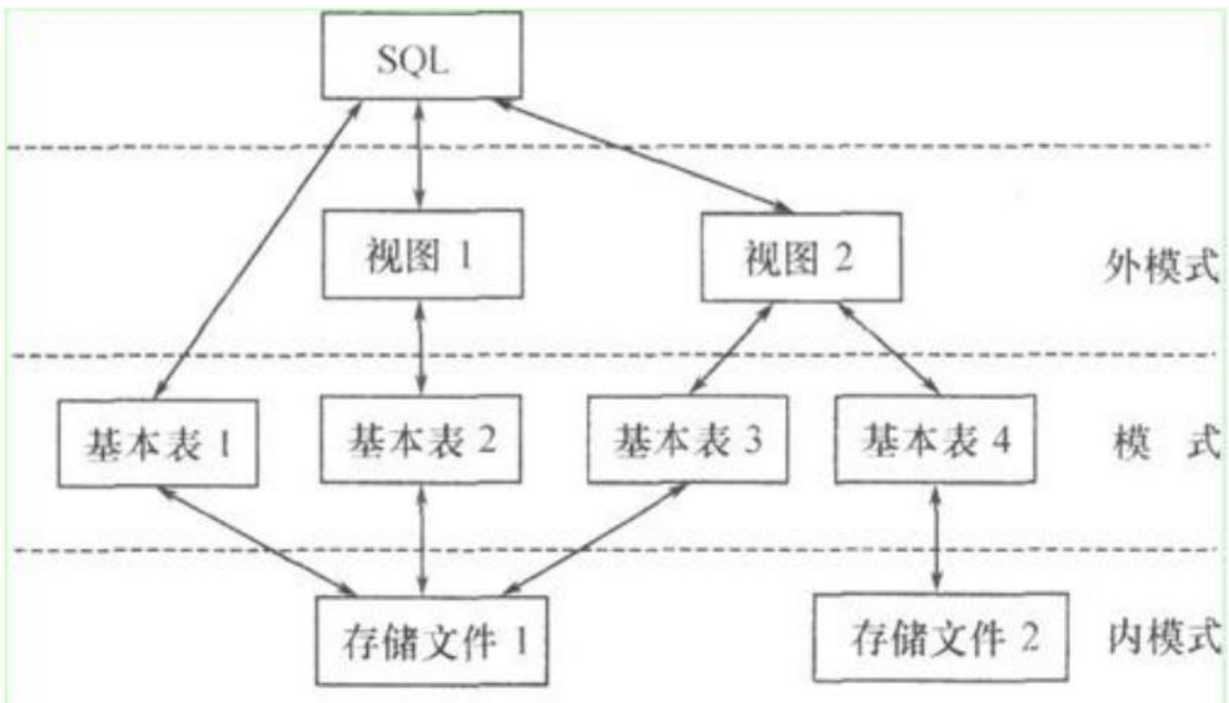


图 3-1　SQL 对关系数据库模式的支持

外模式包括若干视图（view）和部分基本表（base table），数据库模式包括若干基本表，内模式包括若干存储文件（stored file）。

（1）基本表

基本表是本身独立存在的表，在关系数据库管理系统中一个关系就对应一个基本表。一个或多个基本表对应一个存储文件，一个表可以带若干索引，索引也存放在存储文件中。

（2）视图

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

二、数据定义

SQL 的数据定义功能包括模式定义、表定义、视图和索引的定义，如表 3-2 所示。

表 3-2　SQL 的数据定义语句

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

1．模式的定义与删除

（1）定义模式

在 SQL 中，模式定义语句如下：

CREATE SCHEMA<模式名>AUTHORIZATION<用户名>

如果没有指定<模式名>，那么<模式名>隐含为<用户名>。要创建模式，调用该命令的用户必须拥有数据库管理员权限，或者获得了数据库管理员授予的 CREATE SCHEMA 的权限。

定义模式实际上定义了一个命名空间，在这个空间可以进一步定义该模式包含的数据库对象，例如基本表、视图、索引等。

在 CREATE SCHEMA 中可以接受 CREATE TABLE，CREATE VIEW 和 GRANT 子句。也就是说用户可以在创建模式的同时在这个模式定义中进一步创建基本表、视图，定义授权。

模式嵌套定义的语句是：

CREATE SCHEMA<模式名>AUTHORIZATION<用户名>[<表定义子句>|<视图定义子句>|<授权定义了句>]

（2）删除模式

在 SQL 中，删除模式语句如下：

DROP SCHEMA<模式名><CASCADE | RESTRICT>

其中 CASCADE 和 RESTRICT 两者必选其一。选择了 CASCADE（级联），表示在删除模式的同时把该模式中所有的数据库对象全部删除；选择了 RESTRICT（限制），表示如果该模式中已经定义了下属的数据库对象（如表、视图等），则拒绝该删除语句的执行。只有当该模式中没有任何下属的对象时才能执行 DROP SCHEMA 语句。

2．基本表的定义、删除与修改

（1）定义基本表

创建了一个模式就建立了一个数据库的命名空间，一个框架。在这个空间中首先要定义的是该模式包含的数据库基本表。

SQL 语言使用 CREATE TABLE 语句定义基本表，其基本格式如下：

CREATE TABLE<表名>（<列名><数据类型>[列级完整性约束条件]

[，<列名><数据类型>[列级完整性约束条件]]

…

[，<表级完整性约束条件>]）；

建表的同时通常还可以定义与该表有关的完整性约束条件，这些完整性约束条件被存入系统的数据字典中，当用户操作表中数据时由关系数据库管理系统自动检查该操作是否违背这些完整性约束条件。

（2）数据类型

SQL 标准支持多种数据类型，常用数据类型如表 3-3 所示。

表 3-3　数据类型

数据类型	含义
CHAR（n），CHARACTER（n）	长度为 n 的定长字符串
VARCHAR（n），CHARACTERVARYING（n）	最大长度为 n 的变长字符串
CLOB	字符串大对象
BLOB	二进制大对象
INT，INTEGER	长整数（4 字节）
SMALLINT	短整数（2 字节）
BIGINT	大整数（8 字节）
NUMERIC（p,d）	定点数，有 p 位数字（不包括符号、小数点）组成，小数点后面有 d 位数字
DECIMAL（p，d），DEC（p，d）	同 NUMERIC
REAL	取决丁机器粘度的单精度浮点数
DoUBLE PRECISION	取决于机器精度的双精度浮点数
FLOAT（n）	可选精度的浮点数，精度垒少为”位数字
BOOLEAN	逻辑布尔量
DATE	日期，包含年、月、日，格式为 YYYY-MM-DD
TIME	时间，包含一口的时、分、秒，格式为 HH：MM：SS
TIMESTAMP	时间戳类型
INTERVAL	时间间隔类型

（3）模式与表

每一个基本表都属于某一个模式，一个模式包含多个基本表。定义基本表所属模式的三种方法如下所示：

①在表名中明显地给出模式名。

②创建模式语句中同时创建表。

③设置所属的模式，这样在创建表时表名中不必给出模式名。

（4）修改基本表

SQL 语言用 ALTER TABLE 语句修改基本表，其一般格式为：

ALTER TABLE <表名>

[ADD [COLUMN] <新列名><数据类型> [完整性约束]]

[ADD <表级完整性约束>]

[DROP [COLUMN] <列名> [CASCADE| RESTRICT]]

[DROP CONSTRAINT<完整性约束名> [RESTRICT | CASCADE]]

[ALTER COLUMN <列名><数据类型>]；

其中<表名>是要修改的基本表，ADD 子句用于增加新列、新的列级完整性约束条件和新的表级完整性约束条件。DROP COLUMN 子句用于删除表中的列， DROP CONSTRAINT 子句用于删除指定的完整性约束条件，ALTER COLUMN 子句用于修改原有的列定义，包括修改列名和数据类型。

（5）删除基本表

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

DROP TABLE<表名> ERESTRICT ｜ CASCADE;

若选择 RESTRICT，则该表的删除是有限制条件的；若选择 CASCADE，则该表的删除没有限制条件。默认情况是 RESTRICT。基本表定义一旦被删除，不仅表中的数据和此表的定义将被删除，而且此表上建立的索引、触发器等对象一般也都将被删除。

3．索引的建立与删除

建立索引是加快查询速度的有效手段。数据库索引有多种类型，常见索引包括顺序文件上的索引、B+树索引、散列（hash）索引、位图索引等。索引虽然能够加速数据库查询，但需要占用一定的存储空间，当基本表更新时，索引要进行相应的维护，这些都会增加数据库的负担，因此要根据实际应用的需要有选择地创建索引。

（1）建立索引

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

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

其中，<表名>足要建索引的基本表的名字。索引可以建立在该表的一列或多列上，各列名之间用逗号分隔。每个<列名>后面还可以用<次序>指定索引值的排列次序，可选 ASC（升序）或 DESC（降序），默认值为 ASC。

UNIQUE 表明此索引的每一个索引值只对应唯一的数据记录。

CLUSTER 表示要建立的索引是聚簇索引。

（2）删除索引

索引一经建立，就由系统使用和维护它，不需用户干预。

在 SQL 中，删除索引使用 DROP INDEX 语句，其一般格式为“DROP INDEX<索引名>”。删除索引时，系统会同时从数据字典中删去有关该索引的描述。

4．数据字典

数据字典是关系数据库管理系统内部的一组系统表，它记录了数据库中所有的定义信息，包括关系模式定义、视图定义、索引定义、完整性约束定义、各类用户对数据库的操作权限、统计信息等。

三、数据查询

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

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

整个 SELECT 语句的含义是：根据 WHERE 子句的条件表达式，从 FROM 子句指定的基本表或视图中找出满足条件的元组，再按 SELECT 子句中的目标列表达式，选出元组中的属性值形成结果表。

如果有 GROUP BY 子句，则将结果按<列名 l>的值进行分组，该属性列值相等的元组为一个组。通常会在每组中作用聚集函数。如果 GROUP BY 子句带 HAVING 短语，则只有满足指定条件的组才予以输出。如果有 ORDER BY 子句，则结果表还要按<列名 2>的值的升序或降序排序。

SELECT 语句既可以完成简单的单表查询，也可以完成复杂的连接查询和嵌套查询。

1．单表查询

单表查询是指仅涉及一个表的查询。

（1）选择表中的若干列

选择表中的全部或部分列即关系代数的投影运算。

①查询指定列

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

②查询全部列

将表中的所有属性列都选出来有两种方法：

a．在 SELECT 关键字后列出所有列名；

b．如果列的显示顺序与其在基表中的顺序相同，也可以简单地将<目标列表达式>指定为“*”。

③查询经过计算的值

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

用户可以通过指定别名来改变查询结果的列标题，这对于含算术表达式、常量、函数名的目标列表达式尤为有用。

（2）选择表中的若干元组

①消除取值重复的行

两个本来并不完全相同的元组，投影到指定的某些列上后，可能变成相同的行了，可以用 DISTINCT 取消它们。

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

②查询满足条件的元组

查询满足指定条件的元组可以通过 WHERE 子句实现。WHERE 子句常用的查询条件如表 3-4 所示。

表 3-4　常用的查询条件

查询条件	谓词
比较	=，>，<，>=，<=，！=，<>，！>，！<,NOT+上述比较运算符
确定范围	BETWEEN AND，NOT BETWEEN AND
确定集合	IN，NOTIN
字符匹配	LIKE，NOT LIKE
空值	ISNULL，ISNOTNULL
多重条件（逻辑运算）	AND，OR，NOT

a．比较大小

用于进行比较的运算符一般包括：=（等于），>（大于），<（小于），>=（大于等于），<=（小于等于），!=或<>（不等于），!>（不大于），!<（不小于）。

b．确定谓词范围

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

c．确定集合

谓词 IN 可以用来查找属性值属于指定集合的元组；与 IN 相对的谓词是 NOT IN，用于查找属性值不属于指定集合的元组。

d．字符匹配

谓词 LIKE 可以用来进行字符串的匹配。其一般语法格式如下：

[NOT] LIKE <匹配串> [ESCAPE‘<换码字符>’]

其含义是查找指定的属性列值与<匹配串>相匹配的元组。<匹配串>可以是一个完整的字符串，也可以含有通配符％和_。其中：％（百分号）代表任意长度（长度可以为 0）的字符串。（下横线）代表任意单个字符。

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

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

e．涉及多值的查询

f．多重条件查询

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

（3）ORDER BY 子句

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

对于空值，排序时显示的次序由具体系统实现来决定。

（4）聚集函数

为了进一步方便用户，增强检索功能，SQL 提供了许多聚集函数，主要有：

COUNT（*）统计元组个数

COUNT（[D1STINCTIALL]<列名>）统计列中值的个数

SUM（[DISTINCT JALL]<列名>）计算一列值的总和（此列必须是数值型） AVG

（LDlSTINCTIALL]<列名>）计算一列值的平均值（此列必须是数值型） MAX

（[DISTINCTIALL]<列名>）求一列值中的晟大值

MIN（[DlSTINCTIALL]<列名>）求一列值中的最小值

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

当聚集函数遇到空值时，除 COUNT（*）外，都跳过空值而只处理非空值。COUNT（*）是对元组进行计数，某个元组的一个或部分列取空值不影响 COUNT 的统计结果。WHERE 子句中是不能用聚集函数作为条件表达式的。聚集函数只能用于 SELECT 子句和 GROUP BY 中的 HAVING 子句。

（5）GROUP BY 子句

GROUP BY 子句将查询结果按某一列或多列的值分组，值相等的为一组。

对查询结果分组的目的是为了细化聚集函数的作用对象。如果未对查询结果分组，聚集函数将作用于整个查询结果。分组后聚集函数将作用于每一个组，即每一组都有一个函数值。

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

2．连接查询

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

（1）等值与非等值连接查询

连接查询的 WHERE 子句中用来连接两个表的条件称为连接条件或连接谓词，其一般格式为:

[<表名 1>]<列名 1><比较运算符>[<表名 2>]<列名 2>

其中比较运算符主要有：=、>、<、>=、<=、!=（或<>）等。

此外连接谓词还可以使用下面形式：

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

当连接运算符为“=”时，称为等值连接，若在等值连接中把目标列中重复的属性列去掉则为自然连接。使用其他运算符称为非等值连接。

（2）自身连接

连接操作是一个表与其自己进行连接，称为表的自身连接。

（3）外连接

左外连接列出左边关系中所有的元组，右外连接列出右边关系中所有的元组。

（4）多表连接

连接操作除了可以是两表连接、一个表与其自身连接外，还可以是两个以上的表进行连接，后者通常称为多表连接。

关系数据库管理系统在执行多表连接时，通常是先进行两个表的连接操作，再将其连接结果与第三个表进行连接。

3．嵌套查询

在 SQL 语言中，一个 SELECT FROM WHERE 语句称为一个查询块。将一个查询块嵌套在另一个查询块的 WHERE 子句或 HAVING 短语的条件中的查询称为嵌套查询(nested query)。上层的查询块称为外层查询或父查询，下层查询块称为内层查询或子查询。

SQL 语言允许多层嵌套查询。子查询的 SELECT 语句中不能使用 ORDER BY 子句，ORDER BY 子句只能对最终查询结果排序。

（1）带有 IN 谓词的子查询

在嵌套查询中，子查询的结果往往是一个集合，所以谓词 IN 是嵌套查询中最经常使用的谓词。

查询涉及多个关系时，用嵌套查询逐步求解，层次清楚，易于构造，具有结构化程序设计的优点。

如果子查询的查询条件依赖于父查询，这类子查询称为相关子查询(Correlated Subquery)，整个查询语句称为相关嵌套查询(Correlated nested query)语句。求解相关子查询不能像求解不相关子查询那样，一次将子查询求解出来，然后求解父查询。内层查询由于与外层查询有关，因此必须反复求值。

（2）带有比较运算符的子查询

带有比较运算符的子查询是指父查询与子查询之间用比较运算符进行连接。当用户能确切知道内层查询返回的是单值时，可以用>、<、=、>=、<=、!=和<>等比较运算符。

（3）带有 ANY(SOME)或 ALL 谓词的子查询

子查询返回单值时可以用比较运算符，但返回多值时要用 ANY（有的系统用 SOME）或 ALL 谓词修饰符。而使用 ANY 或 ALL 谓词时则必须同时使用比较运算符。其语义为：

>ANY	大于子查询结果中的某个值
>ALL	大于子查询结果中的所有值
<ANY	小于子查询结果中的某个值
<ALL	小于子查询结果中的所有值
>=ANY	大于等于子查询结果中的某个值
>=ALL	大于等于子查询结果中的所有值
<=ANY	小于等于子查询结果中的某个值
<=ALL	小于等于子查询结果中的所有值
=ANY	等于子查询结果中的某个值
=ALL	等于子查询结果中的所有值（通常没有实际意义）
!=（或<>）ANY	不等于子查询结果中的某个值
!=（或<>）ALL	不等于子查询结果中的任何一个值

ANY、ALL 与聚集函数的对应关系如表 3-5 所示。

表 3-5　ANY、ALL 与聚集函数的对应关系

	=	<> 或 !=	<	<=	>	>=
ANY	IN	--	< MAX	<= MAX	> MIN	>= MIN
ALL	--	NOT IN	< MIN	<= MIN	> MAX	>= MAX

（4）带有 EXISTS 谓词的子查询

EXISTS 代表存在量词。带有 EXISTS 谓词的子查询不返回任何数据，只产生逻辑真值“true”或逻辑假值“false”。

可以利用 EXISTS 来判断 X∈S、S∈R、S=R、S∩R 非空等是否成立。

4．集合查询

集合操作主要包括并操作 UNION、交操作 INTERSECT 和差操作 EXCEPT。

参加集合操作的各查询结果的列数必须相同；对应项的数据类型也必须相同。

5．基于派生表的查询

子查询不仅可以出现在 WHERE 子句中，还可以出现在 FROM 子句中，这时子查询生成的临时派生表（derivedtable）成为主查询的查询对象。

如果子查询中没有聚集函数，派生表可以不指定属性列，子查询 SELECT 子句后面的列名为其默认属性。通过 FROM 子句生成派生表时，AS 关键字可以省略，但必须为派生关系指定一个别名。而对于基本表，别名是可选择项。

6．SELECT 语句的一般格式

SELECT 语句的一般格式：

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

（1）目标列表达式的可选格式：

①*。

②<表名>．*。

③COUNT（[DISTINCTIALL]*）。

④[<表名>．]<属性列名表达式>[，[<表名>．]<属性列名表达式>]…。

（2）聚集函数的一般格式为：

$$\left\{\begin{array}{l}\text{COUNT}\\ \text{SUM}\\ \text{AVG}\\ \text{MAX}\\ \text{MIN}\end{array}\right\}(\text{[DISTINCT|ALL]}\ \langle\text{列名}\rangle)$$

（3）WHERE 子句的条件表达式的可选格式：

（1）

$$\langle\text{属性列名}\rangle\theta\left\{\begin{array}{l}\langle\text{属性列名}\rangle\\ \langle\text{常量}\rangle\\ \text{[ANY|ALL]}\ (\text{SELECT 语句})\end{array}\right\}$$

（2）

$$\langle\text{属性列名}\rangle\ \text{[NOT] BETWEEN}\left\{\begin{array}{l}\langle\text{属性列名}\rangle\\ \langle\text{常量}\rangle\\ (\text{SELECT 语句})\end{array}\right\}\text{AND}\left\{\begin{array}{l}\langle\text{属性列名}\rangle\\ \langle\text{常量}\rangle\\ (\text{SELECT 语句})\end{array}\right\}$$

（3）

$$\langle\text{属性列名}\rangle\ \text{[NOT] IN}\left\{\begin{array}{l}(\langle\text{值 1}\rangle\ \text{[,}\ \langle\text{值 2}\rangle\text{]}\cdots)\\ \\ (\text{SELECT 语句})\end{array}\right\}$$

（4）<属性列名> [NOT] LIKE <匹配串>

（5）<属性列名> IS [NOT] NULL

（6）[NOT] EXISTS (SELECT 语句)

（7）

$$\langle\text{条件表达式}\rangle\left\{\begin{array}{l}\text{AND}\\ \\ \text{OR}\end{array}\right\}\langle\text{条件表达式}\rangle\left(\left\{\begin{array}{l}\text{AND}\\ \\ \text{OR}\end{array}\right\}\langle\text{条件表达式}\rangle\cdots\right)$$

四、数据更新

数据更新操作有三种：向表中添加若干行数据、修改表中的数据和删除表中的若干行数据。

1．插入数据

SQL 的数据插入语句 INSERT 通常有两种形式，一种是插入一个元组，另一种是插入子查询结果。后者可以一次插入多个元组。

（1）插入元组

插入元组的 INSERT 语句的格式为：

```
INSERT
INTO <表名> [(<属性列1>[,<属性列2>…)]
VALUES (<常量1> [,<常量2>]…);
```

其功能是将新元组插入指定表中。其中新元组的属性列 1 的值为常量 1，属性列 2 的值为常量 2，……。INTO 子句中没有出现的属性列，新元组在这些列上将取空值。如果 INTO 子句中没有指明任何属性列名，则新插入的元组必须在每个属性列上均有值。

（2）插入子查询结果

插入子查询结果的 INSERT 语句的格式为：

```
INSERT
INTO <表名> [(<属性列1> [,<属性列2>…)]
子查询;
```

2．修改数据

修改操作又称为更新操作，其语句的一般格式为：

```
UPDATE <表名>
SET <列名>=<表达式> [,<列名>=<表达式>] …
[WHERE <条件>];
```

其功能是修改指定表中满足 WHERE 子句条件的元组。其中 SET 子句给出<表达式>的值用于取代相应的属性列值。如果省略 WHERE 子句，则表示要修改表中的所有元组。

3．删除数据

删除语句的一般格式为

```
DELETE
FROM <表名>
[WHERE <条件>];
```

DELETE 语句的功能是从指定表中删除满足 WHERE 子句条件的所有元组。如果省略 WHERE 子句则表示删除表中全部元组，但表的定义仍在字典中。DELETE 语句删除的是表中的数据，而不是关于表的定义。

五、空值的处理

空值就是“不知道”或“不存在”或“无意义”的值。SQL 语言中允许某些元组的某些属性在一定情况下取空值。

1．空值的可能性

（1）该属性应该有一个值，但目前不知道它的具体值。

（2）该属性不应该有值。

（3）由于某种原因不便于填写。

2．空值的处理

（1）空值的产生。

（2）空值的判断。

（3）空值的约束条件。

（4）空值的算术运算、比较运算和逻辑运算。

六、视图

视图是从一个或几个基本表（或视图）导出的表。它与基本表不同。视图就像一个窗口，透过它可以看到数据库中自己感兴趣的数据及其变化，它可以和基本表一样被查询、被删除。也可以在一个视图之上再定义新的视图，但对视图的更新（增、删、改）操作则有一定的限制。

1．定义视图

（1）建立视图

SQL 语言用 CREATE VIEW 命令建立视图，其一般格式为：

```
CREATE VIEW <视图名>[(<列名>[,<列名>]…)]
AS <子查询>
[WITH CHECK OPTION];
```

其中，子查询可以是任意复杂的 SELECT 语句，但通常不允许含有 ORDER BY 子句和 DISTINCT 短语。

WITH CHECK OPTION 表示对视图进行 UPDATE，INSERT 和 DELETE 操作时要保证更新、插入或删除的行满足视图定义中的谓词条件（即子查询中的条件表达式）。

在下列三种情况下必须明确指定组成视图的所有列名：

①某个目标列不是单纯的属性名，而是聚集函数或列表达式。

②多表连接时选出了几个同名列作为视图的字段。

③需要在视图中为某个列启用新的更合适的名字。

RDBMS 执行 CREATE VIEW 语句的结果只是把视图的定义存入数据字典，并不执行其中的 SELECT 语句。只是在对视图查询时，才按视图的定义从基本表中将数据查出。

视图不仅可以建立在一个或多个基本表上，也可以建立在一个或多个已定义好的视图上，或建立在基本表与视图上。

（2）删除视图

删除语句的格式为

DROP VIEW <视图名> [CASCADE];

视图删除后视图的定义将从数据字典中删除。如果该视图上还导出了其他视图，则使用 CASCADE 级联删除语句，把该视图和由它导出的所有视图一起删除。

基本表删除后，由该基本表导出的所有视图(定义)没有被删除，但均已无法使用。删除这些视图(定义)需要显式地使用 DROP VIEW 语句。

2．查询视图

RDBMS 执行对视图的查询时，首先进行有效性检查。检查查询中涉及的表、视图等是否存在。如果存在，则从数据字典中取出视图的定义，把定义中的子查询和用户的查询结合起来，转换成等价的对基本表的查询，然后再执行修正了的查询。

3．更新视图

更新视图是指通过视图来插入(INSERT)、删除(DELETE)和修改(UPDATE)数据。

为防止用户通过视图对数据进行增加、删除、修改时，有意无意地对不属于视图范围内的基本表数据进行操作，可在定义视图时加上 WITH CHECK OPTION 子句。这样在视图上增加、删除、修改数据时，RDBMS 会检查视图定义中的条件，若不满足条件，则拒绝执行该操作。

4．视图的作用

合理使用视图能够带来许多好处：

（1）视图能够简化用户的操作。

（2）视图使用户能以多种角度看待同一数据。

（3）视图对重构数据库提供了一定程度的逻辑独立性。

（4）视图能够对机密数据提供安全保护。

（5）适当的利用视图可以更清晰的表达查询。

3.2 课后习题详解

1．试述 SQL 的特点。

答：SQL 有以下五个特点：

（1）综合统一：SQL 语言集数据定义语言 DDL、数据操纵语言 DML、数据控制语言 DCL 的功能于一体。在关系模型中实体和实体间的联系均用关系表示，这种数据结构的单一性带来了数据操作符的统一性，查找、插入、删除、更新等每一种操作都只需一种操作符，从而克服了非关系系统由于信息表示方式的多样性带来的操作复杂性。

（2）高度非过程化：用 SQL 语言进行数据操作，只要提出“做什么”，而无需指明“怎么做”，因此无需了解存取路径、存取路径的选择以及 SQL 语句的操作过程。这有利于提高数据独立性。

（3）面向集合的操作方式：SQL 语言采用集合操作方式，不仅操作对象和查找结果可以是元组的集合，而且插入、删除、更新操作的对象也可以是元组的集合。

（4）以同一种语法结构提供两种使用方式：SQL 语言既是自含式语言，又是嵌入式语言。作为自含式语言，它能够独立地用于联机交互的使用方式；作为嵌入式语言，它能够嵌入到高级语言程序中，供程序员设计程序时使用。

（5）语言简捷，易学易用。

2．说明在 DROP TABLE 时，RESTRICT 和 CASCADE 的区别。

答：若选择 RESTRICT，则该表的删除是有限制条件的；若选择 CASCADE，则该表的删除没有限制条件。默认情况是 RESTRICT。

3．有两个关系 S（A，B，C，D）和 T（C，D，E，F），写出与下列查询等价的 SQL 表达式：

（1）$\sigma_{A=10}(S)$；（2）$\Pi_{A,B}(S)$；（3）$S \bowtie T$；（4）$S \underset{S.C=T.C}{\bowtie} T$；（5）$S \underset{A<E}{\bowtie} T$；（6）$\Pi_{C,D}(S)\times T$。

答：（1）SELECT * FROM S WHERE A=10；

（2）SELECT A,B FROM S；

（3）SELECT * FROM S INNER JOIN T ON S.C=T.C AND S.D=T.D；
（4）SELECT * FROM S INNER JOIN T ON S.C=T.C；
（5）SELECT * FROM S INNER JOIN T ON S.A<S.E；
（6）SELECT * FROM S,T WHERE S.C=T.C AND T.D=S.D。

4．用 SQL 语句建立第 2 章习题 6 中的 4 个表；针对建立的 4 个表用 SQL 完成第 2 章习题 6 中的查询。
答：（1）建表：
①建立 S 表：S (SNO, SNAME, STATUS, CITY)；
CREATE TABLE S(SNO char(2) UNIQUE，SNAME char(6)，STATUS char(2)，CITY char(4));
②建立 P 表：P (PNO, PNAME, COLOR, WEIGHT)；
CREATE TABLE P(PNO char(2) UNIQUE，PNAME char(6)，COLOR char(2)，WEIGHT INT);
③建立 J 表：J (JNO, JNAME, CITY)；
CREATE TABLE J(JNO char(2) UNIQUE，JNAME char(8)，CITY char(4));
④建立 SPJ 表：SPJ(SNO,PNO,JNO,QTY)；
CREATE TABLE SPJ(SNO char(2)，PNO char(2)，JNO char(2)，QTY INT));
（2）查询：
①供应工程 J1 零件的供应商号码 SN0 对应的 SQL 语句为：
SELECT SNO FROM SPJ WHERE JNO=‘J1’；
②供应工程 J1 零件的供应商号码 SN0 对应的 SQL 语句为：
SELECT SNO FROM SPJ WHERE JNO=‘J1’ AND PNO=‘P1’；
③供应工程 J1 零件为红色的供应商号码 SN0 对应的 SQL 语句为：
SELECT SNO FROM SPJ,P WHERE JNO=‘J1’ AND SPJ.PNO=P .PNO AND COLOR=‘红’；
④没有使用天津供应商生产的红色零件的工程号 JN0 对应的 SQL 语句为：
SELECT JNO FROM SPJ WHERE JNO NOT IN (SELE JNO FROM SPJ,P,S WHERE S.CITY=‘天津’ AND COLOR=‘红’ AND S.SNO=SPJ.SNO AND P.PNO=SPJ.PNO);
⑤由于 VFP 不允许子查询嵌套太深，将查询分为两步：
a．查询 S1 供应商供应的零件号：
SELECT DIST PNO FROM SPJ WHERE SNO=‘S1’；
b．查询哪一工程既使用 P1 零件又使用 P2 零件：
SELECT JNO FROM SPJ WHERE PNO=‘P1’ AND JNO IN (SELECT JNO FROM SPJ WHERE PNO=‘P2’);

5．针对习题 4 中的 4 个表试用 SQL 完成以下各项操作：
（1）找出所有供应商的姓名和所在城市；
（2）找出所有零件的名称、颜色、重量；
（3）找出使用供应商 S1 所供应零件的工程号码；
（4）找出工程项目 J2 使用的各种零件的名称及其数量；
（5）找出上海厂商供应的所有零件号码；
（6）找出使用上海产的零件的工程名称；
（7）找出没有使用天津产的零件的工程号码；
（8）把全部红色零件的颜色改成蓝色；
（9）由 S5 供给 J4 的零件 P6 改为由 S3 供应，请作必要的修改；
（10）从供应商关系中删除 S2 的记录，并从供应情况关系中删除相应的记录；
（11）请将（S2，J6，P4，200）插入供应情况关系。
答：（1）SELECT SNAME，CITY FROM S；
（2）SELECT PNAME，COLOR，WEIGHT FROM P；
（3）SELECT DIST JNO FROM SPJ WHERE SNO=‘S1’；
（4）SELECT PNAME，QTY FROM SPJ，P WHERE P.PNO=SPJ.PNO AND SPJ.JNO=‘J2’；
（5）SELECT PNO FROM SPJ，S WHERE S.SNO=SPJ.SNO AND CITY=‘上海’；
（6）SELECT JNAME FROM SPJ，S，J WHERE S.SNO=SPJ.SNO AND S.CITY=‘上海’ AND J.JNO=SPJ.JNO；
（7）SELECT DISP JNO FROM SPJ WHERE JNO NOT IN (SELECT DIST JNO FROM SPJ，S WHERE

S.SNO=SPJ.SNO AND S.CITY=‘天津’);

（8）UPDATE P SET COLOR=‘蓝’ WHERE COLOR=‘红’;

（9）UPDATE SPJ SET SNO=‘S3’ WHERE SNO=‘S5’ AND JNO=‘J4’ AND PNO=‘P6’;

（10）①DELETE FROM S WHERE SNO=‘S2’;

②DELETE FROM SPJ WHERE SNO=‘S2’;

（11）INSERT INTO SPJ VALUES（‘S2’，‘J6’，‘P4’，200）。

6．什么是基本表?什么是视图?两者的区别和联系是什么?

答：（1）基本表是本身独立存在的表，在 SQL 中一个关系就对应一个表。一个(或多个)基本表对应一个存储文件，一个表可以带若干索引，索引也存放在存储文件中。

（2）视图是从一个或几个基本表导出的表。它本身不独立存储在数据库中，即数据库中只存放视图的定义而不存放视图对应的数据。这些数据仍存放在导出视图的基本表中，因此视图是一个虚表。

（3）基本表与视图的区别和联系：

①区别：视图本身不独立存储在数据库中，是一个虚表。即数据库中只存放视图的定义而不存放视图对应的数据，这些数据仍存放在导出视图的基本表中。

②联系：视图在概念上与基本表等同，用户可以如同基本表那样使用视图，可以在视图上再定义视图。所以基本表中的数据发生变化，从视图中查询出的数据也就随之发生改变。

7．试述视图的优点。

答：视图有以下五个优点：

（1）视图能够简化用户的操作：视图机制使用户可以将注意力集中在所关心的数据上，如果这些数据不是直接来自基本表，则可以通过定义视图，使数据库看起来结构简单、清晰，并且可以简化用户的数据查询操作；

（2）视图使用户能以多种角度看待同一数据：视图机制能使不同的用户以不同的方式看待同一数据，当许多不同种类的用户共享一个数据库时，这种灵活性是非常重要的；

（3）视图对重构数据库提供了一定程度的逻辑独立性：数据的逻辑独立性是指当数据库重构时，用户的应用程序不会受影响；

（4）视图能够对机密数据提供安全保护：有了视图机制，就可以在设计数据库应用系统时，对不同的用户定义不同的视图，使机密数据不出现在不应看到这些数据的用户视图上。这样视图机制就自动提供了对机密数据的安全保护功能。

（5）使用视图可以更清晰的表达查询。

8．哪类视图是可以更新的?哪类视图是不可更新的?各举一例说明。

答：（1）基本表的行列子集视图一般是可更新的，举例如下：

向信息系学生视图 IS—Student 中插入一个新的学生记录，其中学号为 200215129，姓名为赵新，年龄为 20 岁。

```
INSERT
INTO IS_Student
VALUES('200215129','赵新',20);
```

转换为对基本表的更新：

```
INSERT
INTO Student(Sno,Sname,Sage,Sdept)
VALUES('200215129','赵新',20,'IS');
```

这里系统自动将系名'Is'放入 VALUES 子句中。

（2）若视图的属性来自集函数、表达式，则该视图是不可以更新的。例如，前面定义的视图 S_G[例 6]是由学号和平均成绩两个属性列组成的，其中平均绩一项是由 Student 表中对元组分组后计算平均值得来的：

```
CREAT VIEW S_G(Sno,Gavg)
AS
SELECT Sno,AVG(Grade)
FROM SC
GROUP BY Sno;
```

如果想把视图 S_G 中学号为 200215121 的学生的平均成绩改成 90 分，SQL 语句如下：

```
UPDATE S_G
SET Gavg = 90
WHERE Sno ='200215121';
```

但是视图 S_G 的更新无法转换成对基本表 SC 的更新，因为系统无法修改各科成绩，以使平均成绩成为 90。所以 S_G 视图是不可更新的。

9. 请为三建工程项目建立一个供应情况的视图，包括供应商代码（SNO）、零件代码（PNO）、供应数量（QTY）。针对该视图完成下列查询：

（1）找出三建工程项目使用的各种零件代码及其数量；

（2）找出供应商 S1 的供应情况。

答：创建视图：

```
CREATE  VIEW   V_SPJ  AS
SELECT  SNO, PNO, QTY
FROM    SPJ
WHERE   JNO=
        （SELECT  JNO
         FROM  J
         WHERE  JNAME='三建'）;
```

对该视图查询：

（1）查询三建工程项目使用的各种零件代码及其数量的 SQL 语句为：

SELECT DIST PNO，QTY FROM V_SPJ；

（2）查询供应商 S1 的供应情况的 SQL 语句为：

SELECT DIST PNO,QTY FROM V_SPJ WHERE SNO=‘S1’；

第 4 章　数据库安全性

4.1　复习笔记

一、数据库安全性概述

数据库的安全性是指保护数据库以防止不合法使用所造成的数据泄露、更改或破坏。

1．数据库的不安全因素

（1）非授权用户对数据库的恶意存取和破坏；

（2）数据库中重要或敏感的数据被泄露；

（3）安全环境的脆弱性数据库的安全性与计算机系统的安全性。

2．安全标准简介

计算机以及信息安全技术领域最有影响的安全标准是 TCSEC 和 CC 这两个标准。

（1）安全标准发展

①TCSEC 是指 1985 年美国国防部正式颁布的《可信计算机系统评估准则》(Trusted Computer System Evaluation Criteria，TCSEC)。

②CTCPEC、FC、TCSEC 和 ITSEC 的发起了 CC 项目，他们建立了专门的委员会来开发 CC 通用准则。

③TCSEC 又称桔皮书，TDI 将 TCSEC 扩展到数据库管理系统。TDI 中定义了数据库管理系统的设计与实现中需满足和用以进行安全性级别评估的标准。

（2）TCSEC/TDI

TCSEC/TDI 从以下四个方面来描述安全性级别划分的指标：安全策略、责任、保证和文档。每个方面又细分为若干项。

TCSEC/TDI 将系统划分为四组七个等级，依次是 D、C(C1，C2)、B(B1，B2，B3)、A(A1)，按系统可靠或可信程度逐渐增高，如表 4-1 所示。

表 4-1　TCSEC / TDI 安全级别划分

安全级别	定义
A1	验证设计（verified design）
B3	安全域（security domains）
B2	结构化保护（structural protection）
B1	标记安全保护（1abeled security protection）
C2	受控的存取保护（controlled access protection）
C1	自主安全保护（discretionary securityprotection）
D	最小保护（minimai protection）

①D 级是最低级别。

②C1 级只提供了非常初级的自主安全保护，能够实现对用户和数据的分离，进行自主存取控制(DAC)，保护或限制用户权限的传播。

③C2 级实际是安全产品的最低档次，提供受控的存取保护，即将 C1 级的 DAC 进一步细化，以个人身份注册负责，并实施审计和资源隔离。

④B1 级标记安全保护。对系统的数据加以标记，并对标记的主体和客体实施强制存取控制(MAC)以及审计等安全机制。B1 级别的产品才认为是真正意义上的安全产品，满足此级别的产品前一般多冠以“安全”(Security)或“可信的”(Trusted)字样，作为区别于普通产品的安全产品出售。

⑤B2 级结构化保护。建立形式化的安全策略模型并对系统内的所有主体和客体实施 DAC 和 MAC。

⑥B3 级安全域。该级的 TCB 必须满足访问监控器的要求，审计跟踪能力更强，并提供系统恢复过程。

⑦A1 级验证设计，即提供 B3 级保护的同时给出系统的形式化设计说明和验证以确信各安全保护真正实现。

（3）CC 标准

CC 提出了目前国际上公认的表述信息技术安全性的结构，即把对信息产品的安全要求分为安全功能要求和安全保证要求。

安全功能要求用以规范产品和系统的安全行为，安全保证要求解决如何正确有效地实施这些功能。安全功能要求和安全保证要求都以“类-子类-组件”的结构表述，组件是安全要求的最小构件块。

①CC 的文本组成包括三部分：

a．简介和一般模型

介绍 CC 中的有关术语、基本概念和一般模型以及与评估有关的一些框架。

b．安全功能要求

列出了一系列类、子类和组件。由 11 大类、66 个子类和 135 个组件构成。

c．安全保证要求

②评估保证级（Evaluation Assurance Level，EAL）

从 EALl 至 EAL7 共分为 7 级，按保证程度逐渐增高。如表 4-2 所示。

表 4-2　CC 评估保证级（EAL）的划分

评估保证级	定义	TCSEC 安全级别（近似相当）
EAL1	功能测试（functionally tested）	
EAL2	结构测试（structurally tested）CI	
EAL3	系统地测试和检查（methodically tested and checked）	C2
EAL4	系统地设计、测试和复查（methodically designed，tested and reviewed）	B1
EAL5	半形式化设计和测试（semiformally designed and tested）	B2
EAL6	半形式化验证的设计和测试（semiformally verified design and tested）	B3
EAL7	形式化验证的设计和测试（formally verified design and tested）	A1

二、数据库安全性控制

在一般计算机系统中，安全措施是一级一级层层设置的。

在图 4-1 所示的安全模型中，用户要求进入计算机系统时，系统首先根据输入的用户标识进行用户身份鉴定，只有合法的用户才准许进入计算机系统。对已进入系统的用户，DBMS 还要进行存取控制，只允许用户执行合法操作。

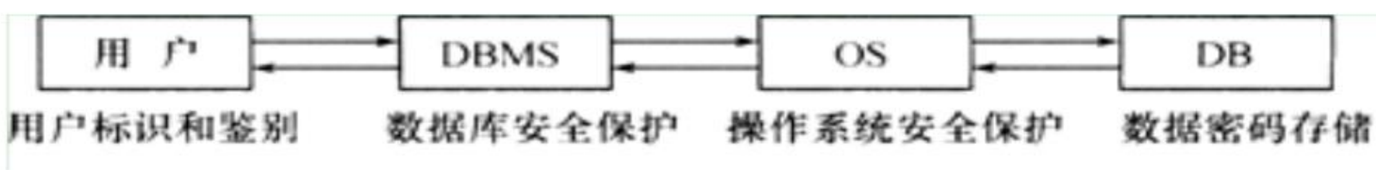


图 4-1　计算机系统的安全模型

1．用户身份鉴别

用户身份鉴别是数据库管理系统提供的最外层安全保护措施。每个用户在系统中都有一个用户标识。每个用户标识由用户名（user name）和用户标识号（UID）两部分组成。UID 在系统的整个生命周期内是唯一的。每次用户要求进入系统时，由系统进行核对，通过鉴定后才提供使用数据库管理系统的权限。

用户身份鉴别常用方法有：。

（1）静态口令鉴别

静态口令一般由用户自己设定，鉴别时只要按要求输入正确的口令，系统将允许用户使用数据库管理系统。

（2）动态口令鉴别

这种方式的口令是动态变化的，每次鉴别时均需使用动态产生的新口令登录数据库管理系统，即采用一次一密的方法。

（3）生物特征鉴别

这种方式通过采用图像处理和模式识别等技术实现了基于生物特征的认证，与传统的口令鉴别相比，无疑产生了质的飞跃，安全性较高。

（4）智能卡鉴别

智能卡是一种不可复制的硬件，内置集成电路的芯片，具有硬件加密功能。智能卡由用户随身携带，登录数据库管理系统时用户将智能卡插入专用的读卡器进行身份验证。

2．存取控制

数据库安全最重要的一点就是确保只授权给有资格的用户访问数据库的权限，同时令所有未被授权的人员无法接近数据，这主要通过数据库系统的存取控制机制实现。

（1）DBMS 的安全子系统

存取控制机制主要包括两部分：

①定义用户权限，并将用户权限登记到数据字典中

用户对某一数据对象的操作权力称为权限。某个用户应该具有何种权限是个管理问题和政策问题而不是技术问题。DBMS 的功能是保证这些决定的执行。

②合法权限检查

每当用户发出存取数据库的操作请求后，DBMS 查找数据字典，根据安全规则进行合法权限检查，若用户的操作请求超出了定义的权限，系统将拒绝执行此操作。

用户权限定义和合法权检查机制一起组成了 DBMS 的安全子系统。

（2）DAC 和 MAC

大型的 DBMS 一般都支持 C2 级中的自主存取控制（简记为 DAC），有些 DBMS 同时还支持 B1 级中的强制存取控制（简记为 MAC）。

这两类方法的简单定义是：

①DAC

在自主存取控制方法中，用户对于不同的数据库对象有不同的存取权限，不同的用户对同一对象也有不同的权限，而且用户还可将其拥有的存取权限转授给其他用户。因此自主存取控制非常灵活。

②MAC

在强制存取控制方法中，每一个数据库对象被标以一定的密级，每一个用户也被授予某一个级别的许可证。对于任意一个对象，只有具有合法许可证的用户才可以存取。强制存取控制因此相对比较严格。

3．自主存取控制方法

SQL 标准对自主存取控制提供支持，主要通过 SQL 的 GRANT 语句和 REVOKE 语句来实现。用户权限是由两个要素组成的：数据库对象和操作类型。在数据库系统中，定义存取权限称为授权（authorization）。

在非关系系统中，用户只能对数据进行操作，存取控制的数据库对象也仅限于数据本身。在关系数据库系统中，存取控制的对象不仅有数据本身（基本表中的数据、属性列上的数据），还有数据库模式（包括数据库、基本表、视图和索引的创建等），表 4-3 列出了主要的存取权限。

表 4-3　关系数据库系统中的存取权限

对象类型	对象操作类型	
数据库模式	模式	CREATE SCHEMA
	基本表	CREATE TABLE，ALTER TABLE
	视图	CREATE VIEW
数据	索引基本表和视图	CREATE INDEXSELECT，INSERT，UPDATE，DELETE，REFERENCES，ALLPRIVILEGES
	属性列	SELECT，INSERT，UPDATE，REFERENCES，ALL PR / VILEGES

4．授权：授予与收回

GRANT 语句向用户授予权限，REVOKE 语句收回授予的权限。

（1）GRANT

GRANT 语句的一般格式为：

```
GRANT <权限> [,<权限>]…
ON <对象类型> <对象名>[,<对象类型> <对象名>]…
TO <用户>[,<用户>]…
[WITH GRANT OPTION];
```

其语义为：将对指定操作对象的指定操作权限授予指定的用户。发出该 GRANT 语句的可以是 DBA，也可以是该数据库对象创建者（即属主 Owner），也可以是已经拥有该权限的用户。接受权限的用户可以是一个或多个具体用户，也可以是 PUBLIC，即全体用户。

如果指定了 WITH GRANT OPTION 子句，则获得某种权限的用户还可以把这种权限再授予其他的用户。如果没有指定 WITH GRANT OPTION 子句，则获得某种权限的用户只能使用该权限，不能传播该权限。

（2）REVOKE

授予用户的权限可以由数据库管理员或其他授权者用 REVOKE 语句收回，REVOKE 语句的一般格式为:

```
REVOKE <权限> [,<权限>]…
ON <对象类型> <对象名> [,<对象类型><对象名>]…
FROM <用户> [,<用户>]…[CASCADE|RESTRICT];
```

数据库管理员拥有对数据库中所有对象的所有权限，并可以根据实际情况将不同的权限授予不同的用户。

用户对自己建立的基本表和视图拥有全部的操作权限，并且可以用 GRANT 语句把其中某些权限授予其他用户。所有授予出去的权力在必要时又都可以用 REVOKE 语句收回。

（3）创建数据库模式的权限

GRANT 和 REVOKE 语句向用户授予或收回对数据的操作权限。对创建数据库模式一类的数据库对象的授权则由数据库管理员在创建用户时实现。

CREATE USER 语句一般格式如下：

CREATE USER <username> [WITH] [DBA | RESOURCE | CONNECT];

对 CREATE USER 语句说明如下：

①只有系统的超级用户才有权创建一个新的数据库用户。

②新创建的数据库用户有三种权限：CONNECT、RESOURCE 和 DBA。

a．CREATE USER 命令中如果没有指定创建的新用户的权限，默认该用户拥有 CONNECT 权限。拥有 CONNECT 权限的用户不能创建新用户，不能创建模式，也不能创建基本表，只能登录数据库。然后由 DBA 或其他用户授予他应有的权限，根据获得的授权情况他可以对数据库对象进行权限范围内的操作。

b．拥有 RESOURCE 权限的用户能创建基本表和视图，成为所创建对象的属主。但是不能创建模式，不能创建新的用户。数据库对象的属主可以使用 GRANT 语句把该对象上的存取权限授予其他用户。

c．拥有 DBA 权限的用户是系统中的超级用户，可以创建新的用户、创建模式、创建基本表和视图等；DBA 拥有对所有数据库对象的存取权限，还可以把这些权限授予一般用户。

以上说明可以用表 4-4 来总结。

表 4-4　权限与可执行的操作对照表

	可否执行的操作			
拥有的权限	CREATEUSER	CREATE SCHEMA	CREATE TABLE	登录数据库，执行数据查询和操纵
DBA	可以	可以	可以	可以
RESOURCE	不可以	不可以	可以	可以
CONNECT	不可以	不可以	不可以	可以，但必须拥有相应权限

CREATE USER 语句不是 SQL 标准，因此不同的关系数据库管理系统的语法和内容相差甚远。

5．数据库角色

数据库角色是被命名的一组与数据库操作相关的权限，角色是权限的集合。在 SQL 中用 CREATE ROLE 语句创建角色，用 GRANT 语句给角色授权，用 REVOKE 语句收回授予角色的权限。

（1）角色创建

创建角色的 SQL 语句格式是“CREATE ROLE<角色名>”。

刚刚创建的角色是空的，没有任何内容。

（2）角色授权

角色授权的 SQL 语句格式是：

```
GRANT <权限>[,<权限>]…
ON <对象类型>对象名
TO <角色>[,<角色>]…
```

DBA 和用户可以利用 GRANT 语句将权限授予某一个或几个角色

（3）将一个角色授予其他的角色或用户

将角色授权给其他角色或用户的 SQL 语句格式是：

```
GRANT <角色1>[,<角色2>]…
TO <角色3>[,<用户1>]…
[WITH ADMIN OPTION]
```

该语句把角色授予某用户，或授予另一个角色。这样，一个角色（角色 3）所拥有的权限就是授予它的全部角色（角色1和角色 2）所包含的权限的总和。

一个角色包含的权限包括直接授予这个角色的全部权限加上其他角色授予这个角色的全部权限。

（4）角色权限的收回

回收角色权限的 SQL 语句格式是：

REVOKE <权限>[,<权限>]…
ON <对象类型> <对象名>
FROM <角色>[,<角色>]…

用户可以回收角色的权限，从而修改角色拥有的权限。

REVOKE 动作的执行者或者是角色的创建者，或者拥有在这个（些）角色上的 ADMIN OPTION。

6．强制存取控制方法

自主存取控制（MAC）能够通过授权机制有效地控制对敏感数据的存取。强制存取控制是指系统为保证更高程度的安全性，按照 TDL / TCSEC 标准中安全策略的要求所采取的强制存取检查手段。

（1）MAC 中的主客体

在 MAC 中，DBMS 所管理的全部实体被分为主体和客体两大类。对于主体和客体，DBMS 为它们每个实例（值）指派一个敏感度标记(Label)。

①主体

主体是系统中的活动实体，既包括 DBMS 所管理的实际用户，也包括代表用户的各进程。主体的敏感度标记称为许可证级别。

②客体

客体是系统中的被动实体，是受主体操纵的，包括文件、基本表、索引、视图等。客体的敏感度标记称为密级。

MAC 机制就是通过对比主体的 Label 和客体的 Label，最终确定主体是否能够存取客体。

（2）存取规则

当某一用户（或某一主体）以标记 Label 注册入系统时，系统要求他对任何客体的存取必须遵循如下规则：

①仅当主体的许可证级别大于或等于客体的密级时，该主体才能读取相应的客体。

②仅当主体的许可证级别等于客体的密级时，该主体才能写相应的客体。

强制存取控制(MAC)是对数据本身进行密级标记，无论数据如何复制，标记与数据是一个不可分的整体，只有符合密级标记要求的用户才可以操纵数据，从而提供了更高级别的安全性。

（3）MAC 与 DAC 的联系

较高安全性级别提供的安全保护要包含较低级别的所有保护，因此在实现 MAC 时要首先实现 DAC，即 DAC 与 MAC 共同构成 DBMS 的安全机制，如图 4-2 所示。

系统首先进行 DAC 检查，对通过 DAC 检查的允许存取的数据库对象再由系统自动进行 MAC 检查，只有通过 MAC 检查的数据库对象方可存取。

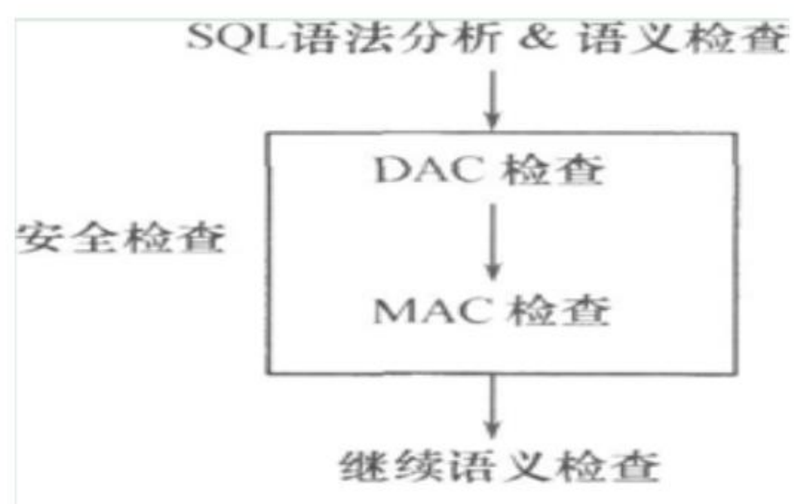


图 4-2　DAC、MAC 安全检查示意图

三、视图机制

根据不同的用户定义不同的视图，把数据对象限制在一定的范围内，通过视图机制把要保密的数据对无权存取的用户隐藏起来，从而自动地对数据提供一定程度的安全保护。

视图机制间接地实现支持存取谓词的用户权限定义。

四、审计

1．定义

审计功能把用户对数据库的所有操作自动记录下来放入审计目志（audit log）中。审计员可以利用审计日志监控数据库中的各种行为，重现导致数据库现有状况的一系列事件，找出非法存取数据的人、时间和内容等。

2．审计事件

（1）服务器事件

审计数据库服务器发生的事件，包含数据库服务器的启动、停止、数据库服务器配置文件的重新加载。

（2）系统权限

对系统拥有的结构或模式对象进行操作的审计，要求该操作的权限是通过系统权限获得的。

（3）语句事件

对 SQL 语句，如 DDL、DML、DQL（Data Query Language，数据查询语言）及 DCL 语句的审计。

（4）模式对象事件

对特定模式对象上进行的 SELECT 或 DML 操作的审计。

3．审计功能

（1）基本功能。

（2）提供多套审计规则。

（3）提供审计分析和报表功能。

（4）审计日志管理功能。

（5）系统提供查询审计设置及审计记录信息的专门视图。

4．AUDIT 语句和 NOAUDIT 语句

AUDIT 语句用来设置审计功能，NOAUDIT 语句则取消审计功能。

5．分类

（1）用户级审计

用户级审计是任何用户可设置的审计，主要是用户针对自己创建的数据库表或视图进行审计，记录所有用户对这些表或视图的一切成功和（或）不成功的访问要求以及各种类型的 SQL 操作。

（2）系统级审计

系统级审计只能由数据库管理员设置，用以监测成功或失败的登录要求、监测授权和收回操作以及其他数据库级权限下的操作。

五、数据加密

1．定义

数据加密是防止数据库数据在存储和传输中失密的有效手段。加密的基本思想是根据一定的算法将原始数据-明文（plain text）变换为不可直接识别的格式-密文（cipher text），从而使得不知道解密算法的人无法获知数据的内容。

2．分类

（1）存储加密

对于存储加密，一般提供透明和非透明两种存储加密方式。透明存储加密是内核级加密保护方式，对用户完全透明；非透明存储加密则是通过多个加密函数实现的。

（2）传输加密

①链路加密

链路加密对传输数据在链路层进行加密，它的传输信息由报头和报文两部分组成，前者是路由选择信息，而后者是传送的数据信息。这种方式对报文和报头均加密。

②端到端加密

端到端加密对传输数据在发送端加密，接收端解密。它只加密报文，不加密报头。

（3）通信加密步骤

①确认通信双方端点的可靠性

②协商加密算法和密钥

③可信数据传输

六、其他安全性保护

1．隐蔽信道

隐蔽信道（covert channal）处理内容是强制存取控制未解决的问题。

2．数据隐私保护

数据隐私是控制不愿被他人知道或他人不便知道的个人数据的能力。数据隐私范围很广，涉及数据管理中的数据收集、数据存储、数据处理和数据发布等各个阶段。

4.2 课后习题详解

1．什么是数据库的安全性?

答：数据库的安全性是指保护数据库以防止不合法的使用所造成的数据泄露、更改或破坏。

2．举例说明对数据库安全性产生威胁的因素。

答：（1）滥用过高权限：当用户（或应用程序）被授予超出了其工作职能所需的数据库访问权限时，这些权限可能会被恶意滥用。例如，一个大学管理员在工作中只需要能够更改学生的联系信息，不过他可能会利用过高的数据库更新权限来更改分数。

（2）滥用合法权限：用户还可能将合法的数据库权限用于未经授权的目的。假设一个恶意的医务人员拥有可以通过自定义 Web 应用程序查看单个患者病历的权限。通常情况下，该 Web 应用程序的结构限制用户只能查看单个患者的病史，即无法同时查看多个患者的病历并且不允许复制电子副本。但是，恶意的医务人员可以通过使用其他客户端（如 MS-Excel）连接到数据库，来规避这些限制。通过使用 MS-Excel 以及合法的登录凭据，该医务人员就可以检索和保存所有患者的病历。

（3）权限提升：攻击者可以利用数据库平台软件的漏洞将普通用户的权限转换为管理员权限。漏洞可以在存储过程、内置函数、协议实现甚至是 SQL 语句中找到。例如，一个金融机构的软件开发人员可以利用有漏洞的函数来获得数据库管理权限。使用管理权限，恶意的开发人员可以禁用审计机制、开设伪造的帐户以及转帐等。

3．试述信息安全标准的发展历史，试述 CC 评估保证级划分的基本内容。

答：（1）信息安全标准的发展历史如下：

①TCSEC 是指 1985 年美国国防部正式颁布的《可信计算机系统评估准则》（简称 TCSEC）。

在 TCSEC 推出后的十年里，不同国家都开始开发建立在 TCSEC 概念上的评估准则，如欧洲的信息技术安全评估准则(ITSEC)、加拿大的可信计算机产品评估准则(CTCPEC)、美国的信息技术安全联邦标准(FC)草案等。

②CTCPEC、FC、TCSEC 和 ITSEC 的发起组织于 1993 年起开始联合行动，解决原标准中概念和技术上的差异，将各自独立的准则集合成一组单一的、能被广泛使用的 IT 安全准则，这一行动被称为 CC 项目。CC V2.1 版于 1999 年被 IS0 采用为国际标准，2001 年被我国采用为国家标准。目前 CC 已经基本取代了 TCSEC，成为评估信息产品安全性的主要标准。

③1991 年 4 月美国 NCSC 颁布了《可信计算机系统评估准则关于可信数据库系统的解释》（简称 TDI），将 TCSEC 扩展到数据库管理系统。TDI 中定义了数据库管理系统的设计与实现中需满足和用以进行安全性级别评估的标准。

④TDI/TCSEC 从四个方面来描述安全性级别划分的指标：安全策略、责任、保证和文档。每个方面又细分为若干项。根据计算机系统对各项指标的支持情况，TDI/TCSEC 将系统划分为四组七个等级，依次是 D、C(C1，C2)、B(B1，B2，B3)、A(A1)，按系统可靠或可信程度逐渐增高。

⑤CC 提出了目前国际上公认的表述信息技术安全性的结构，即把对信息产品的安全要求分为安全功能要求和安全保证要求。安全功能要求用以规范产品和系统的安全行为，安全保证要求解决如何正确有效地实施这些功能。安全功能要求和安全保证要求都以“类-子类-组件”的结构表述，组件是安全要求的最小构件块。

（2）评估保证级是在 CC 第三部分中预先定义的由保证组件组成的保证包，每一保证包描述了一组特定的保证要求，对应着一种评估保证级别。

从 EAL1 至 EAL7 共分为七级，按保证程度逐渐增高，如表 4-5 所示。

表 4-5　CC 评估保证级划分

评估保证级	定 义	TCSEC 安全级别（近似相当）
EAL1	功能测试（functionally tested）	
EAL2	结构测试（structurally tested）	C1
EAL3	系统地测试和检查（methodically tested and checked）	C2
EAL4	系统地设计、测试和复查（methodically designed，tested，and reviewed）	B1
EAL5	半形式化设计和测试（semiformally designed and tested）	B2
EAL6	半形式化验证的设计和测试（semiformally verified design and tested）	B3
EAL7	形式化验证的设计和测试（formally verified design and tested）	A1

4．试述实现数据库安全性控制的常用方法和技术。

答：实现数据库安全性控制的常用方法和技术有：

（1）用户标识和鉴别：该方法由系统提供一定的方式让用户标识自己的名字或身份。每次用户要求进入系统时，由系统进行核对，通过鉴定后才提供系统的使用权。

（2）存取控制：通过用户权限定义和合法权检查确保只有合法权限的用户访问数据库，所有未被授权的人员无法存取数据。

（3）视图机制：为不同的用户定义不同的视图，通过视图机制把要保密的数据对无权存取的用户隐藏起来，从而自动地对数据提供一定程度的安全保护。

（4）审计：建立审计日志，把用户对数据库的所有操作自动记录下来放入审计日志中，DBA 可以利用审计跟踪的信息，重现导致数据库现有状况的一系列事件，找出非法存取数据的人、时间和内容等。

（5）数据加密：对存储和传输的数据进行加密处理，从而使得不知道解密算法的人无法获知数据的内容。

5．什么是数据库中的自主存取控制方法和强制存取控制方法?

答：（1）自主存取控制方法：定义各个用户对不同数据对象的存取权限。当用户对数据库访问时首先检查用户的存取权限，防止不合法用户对数据库的存取。

（2）强制存取控制方法：每一个数据对象被强制地标以一定的密级，每一个用户也被强制地授予某一个级别的许可证。系统规定只有具有某一许可证级别的用户才能存取某一个密级的数据对象。

6．对下列两个关系模式：学生（学号．姓名，年龄，性别，家庭住址，班级号）班级（班级号，班级名，班主任，班长）使用 GRANT 语句完成下列授权功能：

（1）授予用户 U1 对两个表的所有权限，并可给其他用户授权。

（2）授予用户 U2 对学生表具有查看权限，对家庭住址具有更新权限。

（3）将对班级表查看权限授予所有用户。

（4）将对学生表的查询、更新权限授予角色 R1。

（5）将角色 R1 授予用户 U1，并且 U1 可继续授权给其他角色。

答：（1）GRANT ALL PRIVILIGES ON TABLE 学生，班级 TO U1 WITH GRANT OPTION;

（2）GRANT SELECT，UPDATE(家庭地址) ON TABLE 学生 TO U2；

（3）GRANT SELECT ON 班级 TO PUBLIC；

（4）GRANT SELECT ,UPDATA ON TABLE 学生 TO R1；

（5）GRANT R1 TO U1 WITH GRANT OPTION。

7．今有以下两个关系模式：

职工（职工号，姓名，年龄，职务，工资，部门号）部门（部门员，名称，经理名，地址，电话号）

请用 SQL 的 GRANT 和 REVOKE 语句（加上视图机制）完成以下授权定义或存取控制功能：

（1）用户王明对两个表有 SELECT 权限。

（2）用户李勇对两个表有 INSERT 和 DELETE 权限。

（3）每个职工只对自己的记录有 SELECT 权限。

（4）用户刘星对职]二表有 SELECT 权限，对工资字段具有更新权限。

（5）用户张新具有修改这两个表的结构的权限。

（6）用户周平具有对两个表的所有权限（读、插、改、删数据），并具有给其他用户，授权的权限。

（7）用户杨兰具有从每个部门职工中 SELECT 最高工资、最低工资、平均工资的权限，他不能查看每个人的工资。

答：（1）用户王明对两个表有 SELECT 权力的语句为：

GRANT SELECT ON 职工，部门 TO 王明；

（2）用户李勇对两个表有 INSERT 和 DELETE 权力的语句为：

GRANT INSERT，DELETE ON 职工，部门 TO 李勇；

（3）每个职工只对自己的记录有 SELECT 权力的语句为：

GRANT SELECT ON 职工 WHEN USER() = NAME TO ALL；

（4）用户刘星对职工表有 SELECT 权力，对工资字段具有更新权力的语句为：

GRANT SELECT，UPDATE(工资) ON 职工 TO 刘星；

（5）用户张新具有修改这两个表的结构的权力的语句为：

GRANT ALTER TABLE ON 职工，部门 TO 张新；

（6）用户周平具有对两个表所有权力（读，插，改，删数据），并具有给其他用户授权的权力的语句为：

GRANT ALL PRIVILIGES ON 职工，部门 TO 周平 WITH GRANT OPTION；

（7）用户杨兰具有从每个部门职工中 SELECT 最高工资、最低工资、平均工资的权力，他不能查看每个人的工资的语句为：

CREATE VIEW 部门工资 AS SELECT 部门.名称，MAX(工资)，MIN(工资)，AVG(工资) FROM 职工，部门 WHERE 职工.部门号=部门.部门号 GROUP BY 职工.部门号 GRANT SELECT ON 部门工资 TO 杨兰。

8．针对习题 7 中（1）～（7）的每种情况，撤销各用户所授予的权限。

答：（1）撤销用户王明对两个表有 SELECT 权力的语句为：

REVOKE SELECT ON 职工，部门 FROM 王明；

（2）撤销用户李勇对两个表有 INSERT 和 DELETE 权力的语句为：

REVOKE INSERT，DELETE ON 职工，部门 FROM 李勇；

（3）撤销每个职工只对自己的记录有 SELECT 权力的语句为：

REOVKE SELECT ON 职工 WHEN (USER) = NAME FROM ALL；

（4）撤销用户刘星对职工表有 SELECT 权力，对工资字段具有更新权力的语句为：

REVOKE SELECT，UPDATE ON 职工 FROM 刘星；

（5）撤销用户张新具有修改这两个表的结构的权力的语句为：

REVOKE ALTER TABLE ON 职工，部门 FROM 张新；

（6）撤销用户周平具有对两个表所有权力（读，插，改，删数据），并具有给其他用户授权的权力的语句为：

REVOKE ALL PRIVILIGES ON 职工，部门 FROM 周平；

（7）撤销用户杨兰具有从每个部门职工中 SELECT 最高工资、最低工资、平均工资的权力，他不能查看每个人的工资的语句为：

REVOKE SELECT ON 部门工资 FROM 杨兰； DROP VIEW 部门工资；

9．解释强制存取控制机制中主体、客体、敏感度标记的含义。

答：（1）主体是系统中的活动实体，既包括 DBMS 所管理的实际用户，也包括代表用户的各进程。

（2）客体是系统中的被动实体，受主体操纵，包括文件、基表、索引、视图等。

对于主体和客体，DBMS 为它们每个实例（值）指派一个敏感度标记（Label）。

（3）敏感度标记被分成若干级别，例如绝密、机密、可信、公开等。主体的敏感度标记称为许可证级别，客体的敏感度标记称为密级。

10．举例说明强制存取控制机制是如何确定主体能否存取客体的。

答：假设要对关系变量 S 进行 MAC 控制,为简化起见，假设要控制存取的数据单元是元组，则每个元组标以密级。如表 4-6 所示：(4=绝密，3=机密，2=秘密)

表 4-6 元组的密级

S#	SNAME	STATUS	CITY	CLASS
S1	Smith	20	London	2
S2	Jones	10	Paris	3
S3	Clark	20	London	4

假设用户 U1 和 U2 的许可证级别分别为 3 和 2,则根据规则 U1 能查得元组 S1 和 S2，可修改元组 S2；而 U2 只能查得元组 S1，只能修改元组 S1。

11．什么是数据库的审计功能，为什么要提供审计功能?

答：（1）审计功能是指 DBMS 的审计模块在用户对数据库执行操作的同时把所有操作自动记录到系统的审计日志中。审计通常是很费时间和空间的，所以 DBMS 往往都将其作为可选特征，允许 DBA 根据应用对安全性的要求，灵活地打开或关闭审计功能。审计功能一般主要用于安全性要求较高的部门。

（2）提供审计功能的原因：任何系统的安全保护措施都不是完美无缺的，蓄意盗窃破坏数据的人总可能存

在。利用数据库的审计功能，DBA 可以根据审计跟踪的信息，重现导致数据库现有状况的一系列事件，找出非法存取数据的人、时间和内容等。

第 5 章　数据库完整性

5.1　复习笔记

一、概述

数据库的完整性（integrity）是指数据的正确性（correctness）和相容性（compat-abilitv）。数据的完整性是为了防止数据库中存在不符合语义的数据，也就是防止数据库中存在不正确的数据。数据的安全性是保护数据库防止恶意破坏和非法存取。

为维护数据库的完整性，数据库管理系统必须能够实现：

1．提供定义完整性约束条件的机制；

2．提供完整性检查的方法；

3．进行违约处理。

二、实体完整性

1．定义

关系模型的实体完整性在 CREATE TABLE 中用 PRIMARY KEY 定义。对单属性构成的码有两种说明方法，一种是定义为列级约束条件，另一种是定义为表级约束条件。对多个属性构成的码只有一种说明方法，即定义为表级约束条件。

2．检查和违约处理

用 PRIMARY KEY 短语定义了关系的主码后，每当用户程序对基本表插入一条记录或对主码列进行更新操作时，关系数据库管理系统将按照实体完整性规则自动进行检查。

（1）实体完整性规则

①检查主码值是否唯一，如果不唯一则拒绝插入或修改。

②检查主码的各个属性是否为空，只要有一个为空就拒绝插入或修改。

（2）检查方法

检查记录中主码值是否唯一有两种方法：

①全表扫描

依次判断表中每一条记录的主码值与将插入记录上的主码值（或者修改的新主码值）是否相同，这种方法很费时。

②在主码上自动建立一个 B+树索引

通过索引查找基本表中是否已经存在新的主码值，将大大提高效率。

三、参照完整性

1．定义

关系模型的参照完整性在 CREATE TABLE 中用 FOREIGN KEY 短语定义哪些列为外码，用 REFERENCES 短语指明这些外码参照哪些表的主码。

2．参照完整性检查和违约处理

参照完整性将两个表中的相应元组联系起来。因此，对被参照表和参照表进行增、删、改操作时有可能破坏参照完整性，必须进行检查以保证这两个表的相容性。

当上述的不一致发生时，系统可以采用以下策略加以处理:

（1）拒绝（NOATION）执行

不允许该操作执行。该策略一般设置为默认策略。

（2）级联（CASCADE）操作

当删除或修改被参照表（Student）的一个元组导致与参照表（SC）的不一致时，删除或修改参照表中的所有导致不一致的元组。

（3）设置为空值

当删除或修改被参照表的一个元组时造成了不一致，则将参照表中的所有造成不一致的元组的对应属性设置为空值。

四、用户定义的完整性

用户定义的完整性就是针对某一具体应用的数据必须满足的语义要求。

1．属性上的约束条件

（1）定义

在 CREATE TABLE 中定义属性的同时可以根据应用要求，定义属性上的约束条件，即属性值限制，包括：

①列值非空(NOT NULL 短语)。

②列值唯一(UNIQUE 短语)。

③检查列值是否满足一个布尔表达式(CHECK 短语)。

（2）检查和违约处理

当往表中插入元组或修改属性的值时，关系数据库管理系统就检查属性上的约束条件是否被满足，如果不满足则操作被拒绝执行。

2．元组上的约束条件

（1）定义

在 CREATE TABLE 语句中可以用 CHECK 短语定义元组上的约束条件，即元组级的限制。同属性值限制相比，元组级的限制可以设置不同属性之间的取值的相互约束条件。

（2）检查和违约处理

当往表中插入元组或修改属性的值时，关系数据库管理系统将检查元组上的约束条件是否被满足，如果不满足则操作被拒绝执行。

五、完整性约束命名子句

SQL 在 CREAT TABLE 语句中提供了完整性约束命名子句 CONSTRAINT，用来对完整性约束条件命名，这可以灵活地增加、删除一个完整性约束条件。

1．完整性约束命名子句

完整性约束命名子句的格式为：

CONSTRAINT<完整性约束条件名>[PRIMARY KEY 短语|FOREIGN KEY 短语|CHECK 短语]

2．修改表中的完整性限制

使用 ALTER TABLE 语句可以修改表中的完整性限制，先删除原来的约束条件，再增加新的约束条件。

六、域中的完整性限制

SQL 支持域的概念，并可以用 CREATE DOMAIN 语句建立一个域以及该域应该满足的完整性约束条件，然后就可以用域来定义属性。这样定义的优点是，数据库中不同的属性可以来自同一个域，当域上的完整性约束条件改变时只要修改域的定义即可，而不必一一修改域上的各个属性。

七、断言

在 SQL 中可以使用数据定义语言中的 CREATE ASSERTION 语句，通过声明性断言（declarative assertions）来指定更具一般性的约束。可以定义涉及多个表或聚集操作的比较复杂的完整性约束。断言创建以后，任何对断言中所涉及关系的操作都会触发关系数据库管理系统对断言的检查，任何使断言不为真值的操作都会被拒绝执行。

1．创建断言的语句格式

CREATE ASSERTION<断言名><CHECK 子句>

2．删除断言的语句格式

DROP ASSERTION<断言名>

八、触发器

触发器(Trigger)是用户定义在关系表上的一类由事件驱动的特殊过程。一旦定义，任何用户对表的增删改操作均由服务器自动激活相应的触发器，在 DBMS 核心层进行集的完整性控制。触发器类似于约束，但是比约束更加灵活，可以实施比 FOREIGN KEY 束、CHECK 约束更为复杂的检查和操作，具有更精细和更强大的数据控制能力。

1．定义触发器

SQL 使用 CREATE TRIGGER 命令建立触发器，其一般格式为：

```
CREATE TRIGGER <触发器名>                    /*每当触发事件发生时，该触发器被激活*/
{BEFORE | AFTER} <触发事件> ON <表名>
                                             /*指明触发器激活的时间是在执行触发事件前或后*/
REFERENCING NEW|OLD ROW AS<变量>             /*REFERENCING 指出引用的变量*/
FOR EACH{ROW | STATEMENT}                    /*定义触发器的类型，指明动作体执行的频率*/
[WHEN <触发条件>] <触发动作体>               /*仅当触发条件为真时才执行触发动作体*/
```

下面对定义触发器的各部分语法进行详细说明。

（1）权限

表的拥有者即创建表的用户才可以在表上创建触发器，并且一个表上只能创建一定数量的触发器。

（2）触发器名

触发器名可以包含模式名，也可以不包含模式名。同一模式下，触发器名必须是唯一的；并且触发器名和<表名>必须在同一模式下。

（3）表名

当这个表的数据发生变化时，将激活定义在该表上相应<触发事件>的触发器，因此，该表也称为触发器的目标表。

（4）触发事件

触发事件可以是 INSERT、DELETE 或 UPDATE，也可以是这几个事件的组合，如 INSERT OR DELETE 等。UPDATE 后面还可以有 OF<触发列…>，即进一步指明修改哪些列时触发器激活。

（5）触发器类型

触发器按照所触发动作的间隔尺寸可以分为行级触发器和语句级触发器。

（6）触发条件

触发器被激活时，只有当触发条件为真时触发动作体才执行；否则触发动作体不执行。如果省略 WHEN 触发条件，则触发动作体在触发器激活后立即执行。

（7）触发动作体

触发动作体既可以是一个匿名 PL/SQL 过程块，也可以是对已创建存储过程的调用。如果是行级触发器，在两种情况下，用户都可以在过程体中使用 NEW 和 OLD 引用 UPDATE/INSERT 事件之后的新值和 UPDATE/DELETE 事件之前的旧值。如果是语句级触发器，则不能在触发动作体中使用 NEW 或 OLD 进行引用。如果触发动作体执行失败，激活触发器的事件就会终止执行，触发器的目标表或触发器可能影响的其他对象不发生任何变化。

2．激活触发器

触发器的执行是由触发事件激活，并由数据库服务器自动执行的。一个数据表上可能定义了多个触发器，如多个 BEFORE 触发器、多个 AFTER 触发器等，同一个表上的多个触发器激活时遵循如下的执行顺序：

（1）执行该表上的 BEFORE 触发器。

（2）激活触发器的 SQL 语句。

（3）执行该表上的 AFTER 触发器。

对于同一个表上的多个 BEFORE（AFTER）触发器，遵循“谁先创建谁先执行”的原则，即按照触发器创建的时间先后顺序执行。有些关系数据库管理系统是按照触发器名称的字母排序顺序执行触发器。

3．删除触发器

删除触发器的 SQL 语法如下：

```
DROP TRIGGER<触发器名>ON<表名>
```

触发器必须是一个已经创建的触发器，并且只能由具有相应权限的用户删除。

5.2　课后习题详解

1．什么是数据库的完整性？

答：数据库的完整性是指数据的正确性和相容性。数据库的完整性是为了防止数据库中存在不符合语义的数据，也就是防止数据库中存在不正确的数据。

2．数据库的完整性概念与数据库的安全性概念有什么区别和联系？

答：数据的完整性和安全性是两个不同的概念，但是有一定的联系。数据的完整性是为了防止数据库中存在不符合语义的数据，也就是防止数据库中存在不正确的数据。数据的安全性是保护数据库防止恶意的破坏和非法的存取。因此，完整性检查和控制的防范对象是不合语义的、不正确的数据，防止它们进入数据库。安全性控制的防范对象是非法用户和非法操作，防止他们对数据库数据的非法存取。

3．什么是数据库的完整性约束条件？

答：完整性约束条件是指数据库中的数据应该满足的语义约束条件。

完整性约束条件分为六类：静态列级约束、静态元组约束、静态关系约束、动态列级约束、动态元组约束、动态关系约束。

（1）静态列级约束是对一个列的取值域的说明，包括数据类型的约束：数据的类型、长度、单位、精度等；对数据格式的约束；对取值范围或取值集合的约束；空值的约束；其他约束。

（2）静态元组约束就是规定组成一个元组的各个列之间的约束关系，静态元组约束只局限在单个元组上。

（3）静态关系约束是在一个关系的各个元组之间或者若干关系之间常常存在各种联系或约束，常见的静态关系约束有：实体完整性约束，参照完整性约束，函数依赖约束。

（4）动态列级约束是修改列定义或列值时应满足的约束条件，包括下面两方面：修改列定义时的约束，修改列值时的约束。

（5）动态元组约束是指修改某个元组的值时需要参照其旧值，并且新旧值之间需要满足某种约束条件。

（6）动态关系约束是加在关系变化前后状态上的限制条件，例如事务一致性、原子性等约束条件。

4．关系数据库管理系统的完整性控制机制应具有哪三方面的功能？

答：DBMS 的完整性控制机制应具有三个方面的功能：

（1）提供定义完整性约束条件的机制。完整性约束条件也称为完整性规则，是数据库中的数据必须满足的语义约束条件。SQL 标准使用了一系列概念来描述完整性，包括关系模型的实体完整性、参照完整性和用户定义完整性。这些完整性一般由 SQL 的 DDL 语句来实现。它们作为数据库模式的一部分存入数据字典中。

（2）提供完整性检查的方法。DBMS 中检查数据是否满足完整性约束条件的机制称为完整性检查。一般在 INSERT、UPDATE、DELETE 语句执行后开始检查，也可以在事务提交时检查。检查这些操作执行后数据库中的数据是否违背了完整性约束条件。

（3）违约处理。DBMS 若发现用户的操作违背了完整性约束条件，就采取一定的动作，如拒绝(NO ACTION)执行该操作，或级连(CASCADE)执行其他操作，进行违约处理以保证数据的完整性。

5．关系数据库管理系统在实现参照完整性时需要考虑哪些方面？

答：RDBMS 在实现参照完整性时需要考虑以下几个方面：

（1）外码是否可以接受空值。

（2）删除被参照关系元组时的问题，系统可能采取的作法有三种：级联删除、受限删除和置空值删除。

（3）在参照关系中插入元组时的问题，系统可能采取的做法有：受限插入和递归插入。

（4）修改关系中主码的问题。如果需要修改主码值，只能先删除该元组，然后再把具有新主码值的元组插入到关系中。如果允许修改主码，首先要保证主码的惟一性和非空，否则拒绝修改，然后要区分是参照关系还是被参照关系。

6．假设有下面两个关系模式：

职工（职工号，姓名，年龄，职务，工资，部门号），其中职工号为主码；

部门（部门号，名称，经理名，电话），其中部门号为主码。

用 SQL 语言定义这两个关系模式，要求在模式中完成以 F 完整性约束条件的定义：

（1）定义每个模式的主码；

（2）定义参照完整性；

（3）定义职工年龄不得超过 60 岁。

答：

```
CREATE TABLE DEPT
    ( Deptno NUMBER(2),
      Deptname VARCHAR(10),
```

```
        Manager VARCHAR(10),
        PhoneNumber Char(12)
        CONSTRAINT PK_SC PRIMARY KEY(Deptno));
CREATE TABLE EMP
    ( Empno NUMBER(4),
        Ename VARCHAR(10),
        Age NUMBER(2),
        CONSTRAINT C1 CHECK (Age<=60),
        Job VARCHAR(9),
        Sal NUMBER(7,2),
        Deptno NUMBER(2),
        CONSTRAINT FK_DEPTNO
        FOREIGN KEY(Deptno)
        REFFERENCES DEPT(Deptno));
```

7．在关系系统中，当操作违反实体完整性、参照完整性和用户定义的完整性约束条件时，一般是如何分别进行处理的？

答：（1）当违反实体完整性约束条件时，一般采用的方式是拒绝执行，比如拒绝插入或拒绝修改等。

（2）当违反参照完整性约束条件时，并不都是简单地拒绝执行，有时要根据应用语义执行一些附加的操作，以保证数据库的正确性。比如拒绝执行，级连操作，设置为空值等。

（3）当违反用户定义的完整性约束条件时，一般采用的方式是拒绝执行。

8．某单位想举行一个小型的联谊会，关系 Male 记录注册的男宾信息，关系 Female 记录注册的女宾信息。建立一个断言，将来宾的人数限制在 50 人以内。（提示，先创建关系 Female 和关系 Male。）

答：

```
create assertion SUM_MF check
(not exists (select count(*) from (select name from Male
union all
select name from Female)>50))
```

第 2 篇　设计与应用开发篇

第 6 章　关系数据理论

6.1　复习笔记

一、问题的提出

关系要符合一个最基本的条件：每一个分量必须是不可分的数据项。满足了这个条件的关系模式就属于第一范式（1NF）。

1．数据依赖

数据依赖是一个关系内部属性与属性之间的一种约束关系。这种约束关系是通过属性间值是否相等体现出来的数据间的相关联系。它是现实世界属性间相互联系的抽象，是数据内在的性质，是语义的体现。许多种类型的数据依赖中，最重要的是函数依赖(Functional Dependency，FD)和多值依赖(Multivalued Dependency，MVD)。

2．关系模式存在的问题

（1）数据冗余太大；

（2）更新异常；

（3）插入异常；

（4）删除异常。

一个“好”的模式应当不会发生插入异常、删除异常、更新异常，数据冗余应尽可能少。

二、规范化

1．函数依赖

（1）定义

设 R（U）是属性集 u 上的关系模式，X，Y 是 U 的子集。若对于 R（U）的任意一个可能的关系 r，r 中不可能存在两个元组在 X 上的属性值相等，而在 Y 上的属性值不等，则称 X 函数确定 Y 或 Y 函数依赖于 X，记作 X→Y。

（2）完全依赖、部分依赖和传递依赖

①完全依赖

在 R(U)中，如果 X→Y，并且对于 X 的任何一个真子集 X’，都有 X’$\not\rightarrow$Y，则称 Y 对 X 完全函数依赖，记作：

$$X \xrightarrow{F} Y$$

②部分依赖

若 X→Y，但 Y 不完全函数依赖于 X，则称 Y 对 X 部分函数依赖，记作：

$$X \xrightarrow{P} Y$$

③传递依赖

在 R(U)中，如果

$$X \rightarrow Y, (Y \not\subseteq X), Y \not\rightarrow X, Y \rightarrow Z, Z \notin Y$$

则称 Z 对 X 传递函数依赖。

记为：

$$X \xrightarrow{\text{传递}} Z$$

加上条件 Y$\not\rightarrow$X，是因为如果 Y→X，则 X→Y，实际上是$X \xrightarrow{\text{直接}} Z$，是直接函数依赖而不是传递函数依赖。

2．码

（1）候选码

设 K 为 R<U, F>中的属性或属性组合，若$K \xrightarrow{F} U$，则 K 为 R 的候选码（candidate key）。

（2）主码

若候选码多于一个，则选定其中的一个为主码（primary key）。包含在任何一个候选码中的属性称为主属性

（prime attribute）；不包含在任何候选码中的属性称为非主属性（nonprime attribute）或非码属性（non key attribute）。

（3）外码

关系模式 R 中属性或属性组 X 并非 R 的码，但 X 是另一个关系模式的码，则称 X 是 R 的外部码（foreign key），也称外码。

3．范式

关系数据库中的关系是要满足一定要求的，满足不同程度要求的为不同范式。满足最低要求的叫第一范式，简称 1NF；在第一范式中满足进一步要求的为第二范式，其余以此类推。范式是指符合某一种级别的关系模式的集合，R 为第几范式可以写成 R∈xNF。各种范式之间的关系是：5NF⊂4NF⊂BCNF⊂3NF⊂2NF⊂INF，如图 6-1 所示。

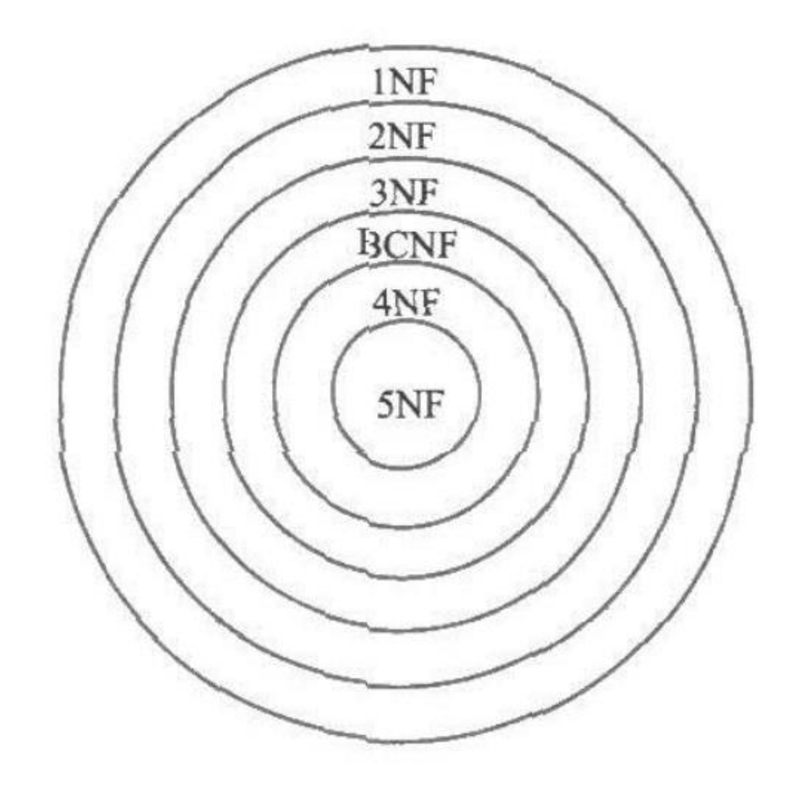


图 6-1　各种范式之间的天系

一个低一级范式的关系模式通过模式分解（schema decomposition）可以转换为若干个高一级范式的关系模式的集合，这种过程就叫规范化（normalization）。

4．2NF

2NF 的定义：

若 R∈INF，且每一个非主属性完全函数依赖于任何一个候选码，则 R∈2NF。

一个关系模式 R 不属于 2NF，就会严生以下几个问题：

（1）插入异常;

（2）删除异常;

（3）修改复杂。

5．3NF

定义：关系模式 R<U，F>中若不存在这样的码 x，属性组 y 及非主属性 z(z∈y)，使得 x→y，y→z 成立，y↛x，则称 R<U，F>∈3NF。

若 R∈3NF，则每一个非主属性既不部分依赖于码也不传递依赖于码。

6．BCNF

（1）定义

关系模式 R<U F>∈INF，若 X→Y 且 $Y \not\subseteq X$ 时 X 必含有码，则 R<U, F>∈BCNF。

即关系模式 R<U，F>中，若每一个决定因素都包含码，则 R<U，F>∈BCNF。

由 BCNF 的定义可以得到结论，一个满足 BCNF 的关系模式有：

①所有非主属性对每一个码都是完全函数依赖。

②所有的主属性对每一个不包含它的码，也是完全函数依赖。

③没有任何属性完全函数依赖于非码的任何一组属性。

7．多值依赖

（1）定义

设 R（U）是属性集 U 上的一个关系模式。X，Y，Z 是 U 的子集，并且 Z=U-X-Y。关系模式 R（U）中多值依赖 X→→Y 成立，当且仅当对 R（U）的任一关系 r，给定的一对（x，z）值，有一组 r 的值，这组值仅仅决定于 x 值而与 z 值无关。

（2）多值依赖的另一个等价的形式化的定义是

在 R（U）的任一关系 r 中，如果存在元组 t、s 使得 t[X]=sIX]，那么就必然存在元组 w、v∈r（w、v 可以与 s、t 相同），使得 w[x]=v[X]=t[X]，而 w[Y]=t[Y]，w[Z]=s[Z]，v[Y]=s[Y]，v[Z]=t[Z]（即交换 s、t 元组的 y 值所得的两个新元组必在 r 中），则 Y 多值依赖于 X, 记为 X→→Y 这里，X、Y 是 U 的子集，Z=U−X−Y。

（3）性质

①对称性。

②传递性。

③函数依赖可以看作是多值依赖的特殊情况。

④若 X→→Y，X→→Z, 则 X→→YZ。

⑤若 X→→Y，X→→Z 则 X→→Y∩Z。

⑥若 X→→Y, X→→Z，则 X→→Y−Z, X→→Z−Y。

（4）多值依赖和函数依赖的区别

①多值依赖的有效性与属性集的范围有关。若 X→→Y 在 U 上成立，则在 W（XY⊆W⊆U）上一定成立；反之则不然，即 X→→Y 在 W（W⊂U）上成立，在 U 上并不一定成立。

②若函数依赖 X→Y 在 R（U）上成立，则对于任何 Y′⊂Y 均有 X→Y′，成立。而多值依赖 X→→Y 若在 R（U）上成立，却不能断言对于任何 Y′⊂Y 有 X→→Y 成立。

8．4NF

（1）定义

关系模式 R<U, F>∈1NF，如果对于 R 的每个非平凡多值依赖 X→→ y（Y∉X）都含有码，则称 R<U, F>∈4NF。

（2）4NF 与 BCNF

4NF 就是限制关系模式的属性之间不允许有非平凡且非函数依赖的多值依赖。如果一个关系模式是 4NF，则必为 BCNF。一个关系模式如果已达到了 BCNF 但不是 4NF，这样的关系模式仍然具有不好的性质。可以用投影分解的方法消去非平凡且非函数依赖的多值依赖。

（3）函数依赖和多值依赖

函数依赖和多值依赖是两种最重要的数据依赖。如果只考虑函数依赖，则属于 BCNF 的关系模式规范化程度已经是最高的了。如果考虑多值依赖，则属于 4NF 的关系模式规范化程度是最高的。

三、数据依赖的公理系统

1．Armstrong 公理系统

（1）定义

对于满足一组函数依赖 F 的关系模式 R<U, F>，其任何一个关系 r，若函数依赖 X→Y，都成立（即 r 中任意两元组 t、s，若 t[X]=s[X]，则 t[Y]=s[Y]）．则称 F 逻辑蕴涵 X→Y.

（2）推理规则

①自反律

若 Y⊆X⊆U，则 X→Y 为 F 所蕴涵。

②增广律

若 X→Y 为 F 所蕴涵，且 Z⊆U，则 XZ→YZ 为 F 所蕴涵。

③传递律

（3）重要定义

①在关系模式 R<U, F>中为 F 所逻辑蕴涵的函数依赖的全体叫作 F 的闭包（closure），记为 F^+。人们把自反律、传递律和增广律称为 Armstrong 公理系统。

②设 F 为属性集 U 上的一组函数依赖，X、Y⊆U，$X_F^+=\{A|X\rightarrow A$ 能由 F 根据 Arrnstrong 公理导出}，X_F^+ 称为属性集 X 关于函数依赖集 F 的闭包。

③如果函数依赖集，满足下列条件，则称 F 为一个极小函数依赖集，亦称为最小依赖集或最小覆盖（minimal cover）。

a．F 中任一函数依赖的右部仅含有一个属性；

b．F 中不存在这样的函数依赖 X→A，使得 F 与 F−{X→A}等价；

c．F 中不存在这样的函数依赖 X→A，X 有真子集 2 使得 F-{X-A）U{Z→A}与 F 等价。

（4）引理

①$X \to A_1A_2 \cdots A_k$ 成立的充分必要条件是 $X \to A_i$ 成立（i=1，2，…，k）。

②设 F 为属性集 u 上的一组函数依赖，Y、Y⊆U，X→Y 能由 F 根据 Armstrong

公理导出的充分必要条件是 $Y \subseteq X_F^+$。。

③$F^+=G^+$的充分必要条件是 $F^+ \subseteq G^+$和 $G \subseteq F^+$。

（5）有效性和完备性

Armstrong 公理系统是有效的、完备的。

①有效性

有效性指的是由 F 出发根据 Armstrong 公理推导出来的每一个函数依赖一定在 F+中。

②完备性

完备性指的是 F^+中的每一个函数依赖，必定可以由 F 出发根据 Armstrong 公理推导出来。

四、模式的分解

关系模式 R<U，F>的一个分解是指: $\rho=\{R_1<U_1,F_1>, R_2<U_2,F_2>,\ldots, R_k<U_k,F_k>\}$。

其中$U=\bigcup_{i=1}^{n} U_i$，并且有$U_i \subseteq U_j, 1 \leqslant i,j \leqslant n$，$F_i$是 F 在 U 上的投影。

1．模式分解的 3 个定义

对于一个模式的分解是多种多样的，但是分解后产生的模式应与原模式等价。

对“等价”的概念有三种不同的定义：

（1）分解具有“无损连接性”;

（2）分解要“保持函数依赖”;

（3）分解既要“保持函数依赖”，又要具有“无损连接性”。

按照不同的分解准则，模式所能达到分离程度各不相同，各种范式就是对分离程度的测度。

2．分解的无损连接性和保持函数依赖性

设 $\rho=\{R_1<U_1, F_1>, \cdots, R_k<U_k, F_k>\}$ 是 R<U，F>的一个分解，r 是 R<U，F>的一个关系。定义

$m_\rho(r)=\bigtimes_{i=1}^{k} \pi_{R_i}(r)$，即 m_ρ（r）是 r 在 ρ 中各关系模式上投影的连接。这里 $\pi_{R_i}(r)=\{t.U_i|t \in r\}$。

（1）引理

①设 R<U，F>是一个关系模式，$\rho=\{R_1<U_1, F_1>, \cdots, R_k<U_k, F_k>\}$ 是 R 的一个分解，r 是 R 的一个关系，$r_i=\pi_{R_i}(r)$，则：

a．$r \subseteq m_\rho$（r）

b．若 $s=m_\rho$（r），则 $\pi_{R_i}(s)=r_i$

c． $m_\rho(m_\rho(r))=m_\rho(r)$

②对于 R<U, F>的一个分解 $\rho=\{R_1<U_1, F_1>, R_2<U_2, F_2>\}$，如果$U_1 \cap U_2 \to U_1-U_2 \in F^+$或

$U_1 \cap U_2 \to U_2-U_1 \in F^+$，则 ρ 具有无损连接性。

3．模式分解的算法

关于模式分解的几个重要事实是：

（1）若要求分解保持函数依赖，那么模式分离总可以达到 3NF，但不一定能达到 BCNF。

（2）若要求分解既保持函数依赖，又具有无损连接性，可以达到 3NF，但不一定能达到 BCNF。

（3）若要求分解具有无损连接性，那一定可达到 4NF。

6.2 课后习题详解

1．理解并给出 F 列术语的定义：

函数依赖、部分函数依赖、完全函数依赖、传递依赖、候选码、超码、主码、外码、伞码（all—key）、1NF、2NF、3NF、BCNF、多值依赖、4NF。

答：（1）函数依赖：设 R(U)是属性集 U 上的关系模式。X，Y 是属性集 U 的子集。若对于 R(U)的任意一个可能的关系 r，r 中不可能存在两个元组在 X 上的属性值相等，而在 Y 上的属性值不等，则称 X 函数确定 Y 或 Y 函数依赖于 X，记作 X→Y。（即只要 X 上的属性值相等，Y 上的值一定相等）。

（2）部分函数依赖：若 X→Y，但 Y 不完全函数依赖于 X，则称 Y 对 X 部分函数依赖，记作：$X \xrightarrow{P} Y$。

（3）完全函数依赖：在 R(U)中，如果 X→Y，并且对于 X 的任何一个真子集 X'，都有 X'→Y，则称 Y 对 X 完全函数依赖，记作 $X \xrightarrow{F} Y$。

（4）传递依赖：在 R(U)中，如果 $X \to Y,(Y \not\subseteq X),Y \not\to X,Y \to Z,Z \notin Y$，则称 Z 对 X 传递函数依赖，记为：$X \xrightarrow{传递} Z$。

（5）候选码：设 K 为 R<U，F>中的属性或属性组合，若 $K \xrightarrow{F} U$，则 K 为 R 的候选码。

（6）主码：若候选码多于一个，则选定其中的一个为主码。

（7）外码：关系模式 R 中属性或属性组 X 并非 R 的码，但 X 是另一个关系模式的码，则称 X 是 R 的外部码，也称外码。

（8）全码：整个属性组是码，称为全码。

（9）1NF：关系模式 R 的每一个分量是不可再分的数据项。

（10）2NF：关系模式 R∈1NF，且每一个非主属性完全函数依赖于码。

（11）3NF：关系模式 R<U，F>中不存在这样的码 X、属性组 Y 及非主属性 Z(Z 不是 Y 的子集)使得 X→Y，Y $\not\to$ X，Y→Z 成立。

（12）BCNF：关系模式 R<U，F>∈1NF，X→Y 且 Y 不是 X 的子集时,X 必含有码。

（13）多值依赖：设 R(U)是属性集 U 上的一个关系模式。X，Y，Z 是 U 的子集，并且 Z=U-X-Y。关系模式 R(U)中多值依赖 x→→y 成立，当且仅当对 R(U)的任一关系 r，给定的一对(x，z)值，有一组 y 的值，这组值仅仅决定于 X 值而与 Z 值无关。

（14）4NF：关系模式 R<U，F>∈1NF，如果对于 R 的每个非平凡多值依赖 X→→Y（Y 不是 X 的子集，Z=U-X-Y 不为空），X 都含有码。

2．建立一个关于系、学生、班级、学会等诸信息的关系数据库。

描述学生的属性有：学号、姓名、出生年月、系名、班号、宿舍区；

描述班级的属性有：班号、专业名、系名、人数、入校年份；

描述系的属性有：系名、系号、系办公室地点、人数：

描述学会的属性有：学会名、成立年份、地点、人数。

有关语义如下：一个系有若干专业，每个专业每年只招一个班，每个班有若干学生。一个系的学生住在同一宿舍区。每个学生可参加若干学会，每个学会有若干学生。学生参加某学会有一个入会年份。

请给出关系模式，写出每个关系模式的极小函数依赖集，指出是否存在传递函数依赖，对于函数依赖左部是多属性的情况，讨论函数依赖是完全函数依赖还是部分函数依赖。指出各关系的候选码、外部码，并说明是否全码存在。

答：（1）关系模式如下：

学生：S(Sno，Sname，Sbirth，Dept，Class，Rno)

班级：C(Class，Pname，Dept，Cnum，Cyear)

系：D(Dept，Dno，Office，Dnum)

学会：M(Mname，Myear，Maddr，Mnum)

（2）每个关系模式的最小函数依赖集如下：

①学生 S(Sno，Sname，Sbirth，Dept，Class，Rno)的最小函数依赖集如下:Sno→Sname，Sno→Sbirth，Sno→Class，Class→Dept，DEPT→Rno。

传递依赖如下：

由于 Sno→Dept，而 Dept→Sno，Dept→Rno（宿舍区），所以 Sno 与 Rno 之间存在着传递函数依赖；由于

Class→Dept，Dept→Class，Dept→Rno，所以 Class 与 Rno 之间存在着传递函数依赖；由于 Sno→Class，Class→Sno，Class→Dept，所以 Sno 与 Dept 之间存在着传递函数依赖。

②班级 C(Class，Pname，Dept，Cnum，Cyear)的最小函数依赖集如下:

Class→Pname，Class→Cnum，Class→Cyear，Pname→Dept。

由于 Class→Pname，Pname→Class，Pname→Dept，所以 Class 与 Dept 之间存在着传递函数依赖。

③系 D(Dept，Dno，Office，Dnum)的最小函数依赖集如下：

Dept→Dno，Dno→Dept，Dno→Office，Dno→Dnum。

Dept 与 Office，Dept 与 Dnum 之间不存在传递依赖。

④学会 M(Mname，Myear，Maddr，Mnum)的最小函数依赖集如下：

Mname→Myear，Mname→Maddr，Mname→Mnum。

该模式不存在传递依赖。

（3）各关系模式的候选码、外部码，全码如下：

①学生 S 候选码：Sno；外部码：Dept、Class；无全码。

②班级 C 候选码：Class；外部码：Dept；无全码。

③系 D 候选码：Dept 或 Dno；无外部码；无全码。

④学会 M 候选码：Mname；无外部码；无全码。

3．试由 Armostrong 公理系统推导出下面三条推理规则。

（1）合并规则：若 X→Z，X→Y 则有 X→YZ。

（2）伪传递规则：由 X→Y, WY→Z 有朋 XW→Z。

（3）分解规则：X→Y．Z⊆Y 有 X→Z。

答：（1）已知 X→Z,由增广律知 XY→YZ，又因 X→Y，可得 XX→XY→YZ，根据传递律得 X→YZ。

（2）已知 X→Y，由增广律知 XW→WY，又因 WY→Z，可得 XW→WY→Z，根据传递律得 XW→Z。

（3）已知 Z⊆Y，由自反律知 Y→Z，又因 X→Y，所以由传递律可得 X→Z。

4．关于多值依赖的另一种定义是：

给定一个关系模式 R（X, Y, Z），其中 X, Y、Z 可以是属性或属性组合。

设 x∈x，y∈Y，z∈Z，xz 在 R 中的像集为

Y_{XZ}={r Y |r．X=x∧r·Z=z∧r∈R}

定义 R（X, Y, Z）当且仅当 $Y_{xz}=Y_{xz'}$ 对于每组（x, z, z′）都成立，则 Y 对 X 多值依赖，记作 X→→Y。这里，允许 Z 为空集，在 Z 为宅集时，称为平凡的多值依赖。请证明这里的定义和 6．27 节中定义 6．9 是等价的。

答：设 $Y_{xz}=Y_{xz'}$ 对于每一组(x，z，z’)都成立，现证其能推出讲义中定义的条件：

设 s,t 是关系 r 中的两个元组，s[X]=t[X]，由新定义的条件可知对于每一个 z 值，都对应相同的一组 y 值。这样一来，对相同的 x 值，交换 y 值后所得到的元组仍然属于关系 r，即讲义中多值依赖的条件成立。

如果讲义定义的条件成立，则对相同的 x 值，交换 y 值后所得的元组仍属于关系 r，由于任意性及其对称性，可知每个 z 值对应相同的一组 y 值，所以 $Y_{xz}=Y_{xz'}$ 对于每一组(x，z，z’)都成立。

综上可知，两者是等价的。

5．试举出三个多值依赖的实例。

答：（1）关系模式 MSC(M,S,C）中，M 表示专业，S 表示学生，C 表示该专业的必修课。假设每个专业有多个学生，有一组必修课。设同专业内所有学生选修的必修课相同，实例关系如下。按照语义对于 M 的每一个值 Mi，S 有一个完整的集合与之对应而不论 C 取何值，所以 M→→S。由于 C 与 S 的完全对称性，必然有 M→→C 成立。

（2）关系模式 ISA(I,S,A）中，I 表示学生兴趣小组，S 表示学生，A 表示某兴趣小组的活动项目。假设每个兴趣小组有多个学生，有若干活动项目。每个学生必须参加所在兴趣小组的所有活动项目，每个活动项目要求该兴趣小组的所有学生参加。按照语义有 I→→S,I→→A 成立。

（3）上课（学号，教师工号，教室），一个学生可由多个教师来教，一个学生可在多教室上课，而且一个教师可在多个教室上课，一个教室可由多个教师上课。所以存在如下多值依赖：学号→→教师工号和学号→→教室。

6．有关系模式 R（A，B，C，D，E），回答下面各个问题：

（1）若 A 是 R 的候选码，具有函数依赖 BC→DE，那么在什么条件 R 是 BCNF?

（2）如果存在函数依赖 A→B，BC→D，DE→A，列出 R 的所有码。

（3）如果存在函数依赖 A→B，BC→D，DE→A，R 属于 3NF 还是 BCNF。

答：略。

7．F 面的结论哪些是正确的?哪些是错误的?对于错误的请给出一个反例说明之。

（1）任何一个二目关系是属于 3NF 的。

（2）任何一个二目关系是属于 BCNF 的。

（3）任何一个二目关系是属于 4NF 的。

（4）当且仅当函数依赖 A→B 在 R 上成立，关系 R（A，B，C）等于其投影 R_1（A，B）和 R_2（A，C）的连接。

（5）若 RA→RB，RB→RC，则 RA→RC。

（6）若 RA→RB，RA→RC，则 R．A→R，（B, C）。

（7）若 RB→RA，RC→RA，则 R，（B, C）→R．A。（8）若 R，（B, C）→R．A，则 RB→RA，RC→RA。

答：（1）正确。因为关系模式中只有两个属性，所以无传递。

（2）正确。按 BCNF 的定义，若 X→Y，且 Y 不是 X 的子集时，每个决定因素都包含码，对于二目关系决定因素必然包含码。

（3）正确。因为只有两个属性，所以无非平凡的多值依赖。

（4）错误。当 A→B 在 R 上成立，关系 R(A,B,C)等于其投影 R_1(A,B)和 R_2(A,C)的连接。反之则不然。正确的是当且仅当函数依赖 A→→B 在 R 上成立，关系 R(A,B,C)等于其投影 R_1(A,B)和 R_2(A,C)的连接。

（5）正确。

（6）正确。

（7）正确。

（8）错误。反例关系模式 SC(S#,C#,G)，(S#,C#)→G，但 S# $\not\rightarrow$ G，C# $\not\rightarrow$ G。

8．证明：

（1）如果 R 是 BCNF 关系模式，则 R 是 3NF 关系模式，反之则不然。

（2）如果 R 是 3NF 关系模式，则 R 一定是 2NF 关系模式。

证明：（1）用反证法：设 R 是一个 BCNF，但不是 3NF，则必存在非主属性 A 和候选码 x 以及属性集 y，使得 x→y，y→A，其中 A→x，A→y，y→x 不在函数依赖中，这就是说 y 不可能包含 R 的码，但 y→A 却成立。根据 BCNF 定义，R 不是 BCNF，与题设矛盾，所以一个 BCNF 范式必是 3NF。

（2）反证法：假设 R 中非主属性 A 部分依赖于关键字 K，则存在 K'是 K 的子集,使得 K'→A，因 K'是 K 的子集有 K→K'，但 K' $\not\rightarrow$ K，于是有 K→K',K' $\not\rightarrow$ K,K'→A，并且 A 不属于 K，因而 A 传递以来于 K,即 R 不属于 3NF,与已知矛盾，所以一个 3NF 一定是 2NF。

第 7 章　数据库设计

7.1　复习笔记

一、数据库设计

数据库设计是指对于一个给定的应用环境，构造（设计）优化的数据库逻辑模式和物理结构，并据此建立数据库及其应用系统，使之能够有效地存储和管理数据，满足各种用户的应用需求。数据库设计的目标是为用户和各种应用系统提供一个信息基础设施和高效的运行环境。

1．数据库设计的特点

（1）数据库建设的基本规律

“三分技术，七分管理，十二分基础数据”是数据库设计的特点之一。

（2）结构（数据）设计和行为（处理）设计相结合

数据库设计应该和应用系统设计相结合。也就是说，整个设计过程中要把数据库结构设计和对数据的处理设计密切结合起来。

2．数据库设计方法

（1）新奥尔良（NewOrleans）方法。

（2）基于 E-R 模型的设计方法。

（3）3NF（第三范式）的设计方法。

（4）面向对象的数据库设计方法。

（5）统一建模语言（Unified Model Language，UML）方法。

3．数据库设计的基本步骤

按照结构化系统设计的方法，考虑数据库及其应用系统开发全过程，将数据库设计分为以下 6 个阶段（如图 7-1 所示）。

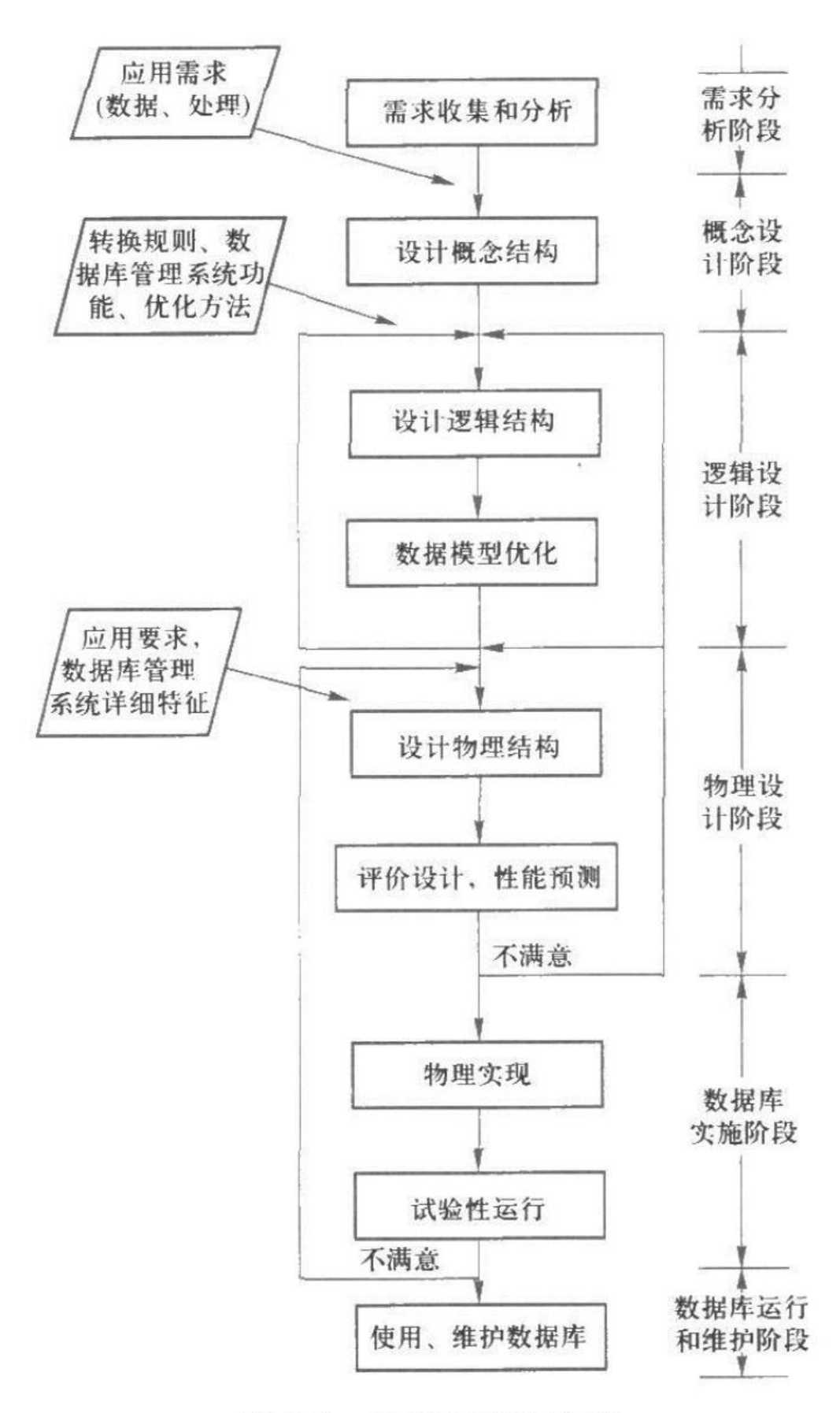


图 7-1　数据库设计步骤

在数据库设计过程中，需求分析和概念结构设计可以独立于任何数据库管理系统进行，逻辑结构设计和物理结构设计与选用的数据库管理系统密切相关。

（1）需求分析阶段

进行数据库设计首先必须准确了解与分析用户需求（包括数据与处理）。需求分析是整个设计过程的基础，是最困难和最耗费时间的一步。

（2）概念结构设计阶段

概念结构设计是整个数据库设计的关键，它通过对用户需求进行综合、归纳与抽象，形成一个独立于具体数据库管理系统的概念模型。

（3）逻辑结构设计阶段

逻辑结构设计是将概念结构转换为某个数据库管理系统所支持的数据模型，并对其进行优化。

（4）物理结构设计阶段

物理结构设计是为逻辑数据模型选取一个最适合应用环境的物理结构（包括存储结构和存取方法）。

（5）数据库实施阶段

在数据库实施阶段，设计人员运用数据库管理系统提供的数据库语言及其宿主语言，根据逻辑设计和物理设计的结果建立数据库，编写与调试应用程序，组织数据入库，并进行试运行。

（6）数据库运行和维护阶段

数据库应用系统经过试运行后即可投入正式运行。在数据库系统运行过程中必须不断地对其进行评估、调整与修改。

这个设计步骤既是数据库设计的过程，也包括了数据库应用系统的设计过程。设计过程各个阶段的设计描述，可用图 7-2 概括地给出。

设计阶段	设计描述
需求分析	数字字典、全系统中数据项、数据结构、数据流、数据存储的描述
概念结构设计	概念模型（E-R 图） 数据字典
逻辑结构设计	某种数据模型 关系　非关系
物理结构设计	存储安排 存取方法选择 存取路径建立 ⋮　分区1 分区2
数据库实施	创建数据库模式 装入数据 数据库试运行 Creat … Load …
数据库运行和维护	性能监测、转储/恢复、数据库重组和重构

图 7-2　数据库设计各个阶段的数据设计描述

4．数据库设计过程中的各级模式

数据库设计的不同阶段形成数据库的各级模式，如图 7-3 所示。

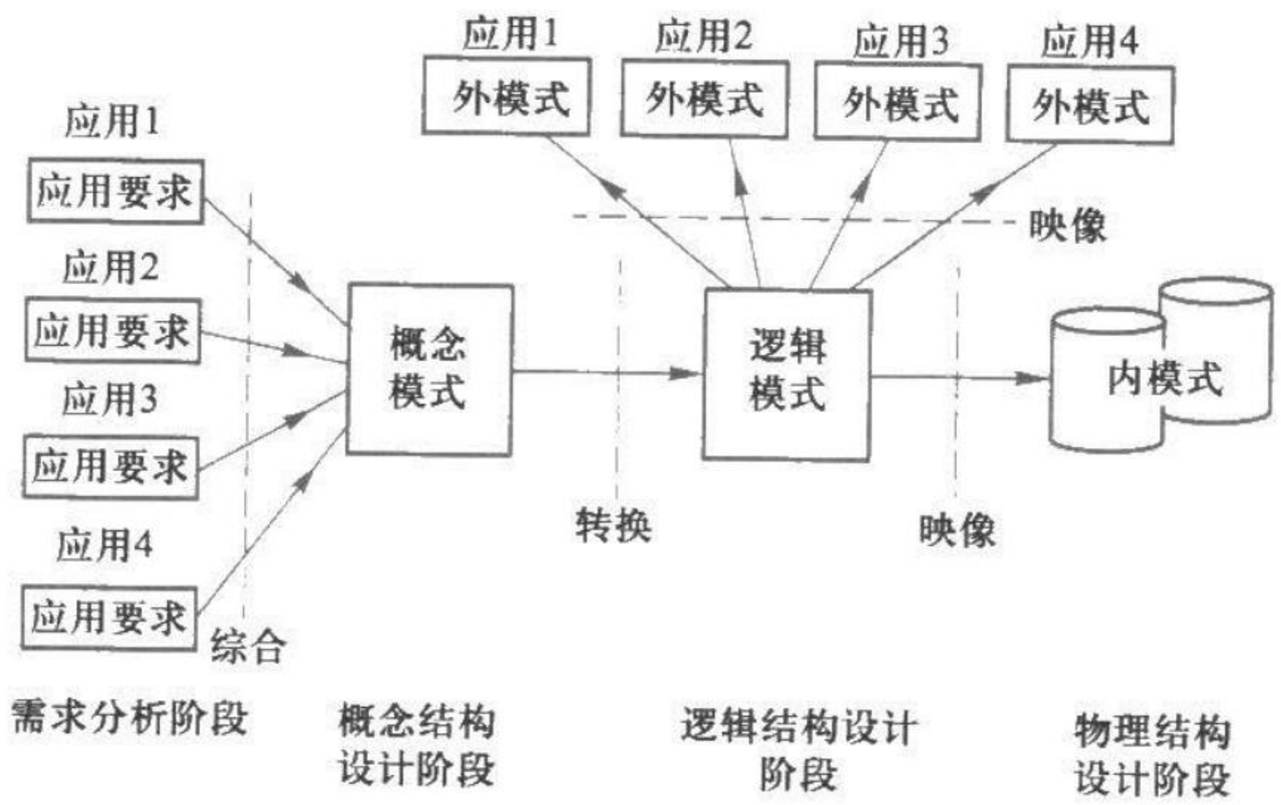


图 7-3　数据库的各级模式

（1）需求分析阶段，综合各个用户的应用需求；

（2）在概念设计阶段形成独立于机器特点，独立于各个 DBMS 产品的概念模式，在本篇中就是 E-R 图；

（3）在逻辑设计阶段将 E-R 图转换成具体的数据库产品支持的数据模型，如关系模型，形成数据库逻辑模式；

（4）根据用户处理的要求、安全性的考虑，在基本表的基础上再建立必要的视图(View)，形成数据的外模式；

（5）在物理设计阶段，根据 DBMS 特点和处理的需要，进行物理存储安排，建立索引，形成数据库内模式。

二、需求分析

需求分析简单地说就是分析用户的要求。需求分析是设计数据库的起点，需求分析结果是否准确反映用，的实际要求将直接影响到后面各阶段的设计，并影响到设计结果是否合理和实用。

1．需求分析的任务

需求分析的任务是通过详细调查现实世界要处理的对象（组织、部门、企业等），充分了解原系统（手工系统或计算机系统）的工作概况，明确用户的各种需求，然后在此基础上确定新系统的功能。新系统必须充分考虑今后可能的扩充和改变，不能仅仅按当前应用需求来设计数据库。

调查的重点是“数据”和“处理”，通过调查、收集与分析，获得用户对数据库的如下要求：（1）信息要求；（2）处理要求。

2．需求分析的方法

进行需求分析首先是调查清楚用户的实际要求，与用户达成共识，然后分析与表达这些需求。

（1）调查步骤

①调查组织机构情况。

②调查各部门的业务活动情况。

③在熟悉了业务活动的基础上，协助用户明确对新系统的各种要求。

④确定新系统的边界。

（2）方法

①跟班作业。

②开调查会。

③请专人介绍。

④询问。

⑤设计调查表请用户填写。

⑥查阅记录。

结构化分析（Structured Analysis，SA）方法是一种简单实用的方法。SA 方法从最上层的系统组织机构入手，采用自顶向下、逐层分解的方式分析系统。

对用户需求进行分析与表达后，需求分析报告必须提交给用户，征得用户的认可。图 7-4 描述了需求分析的过程。

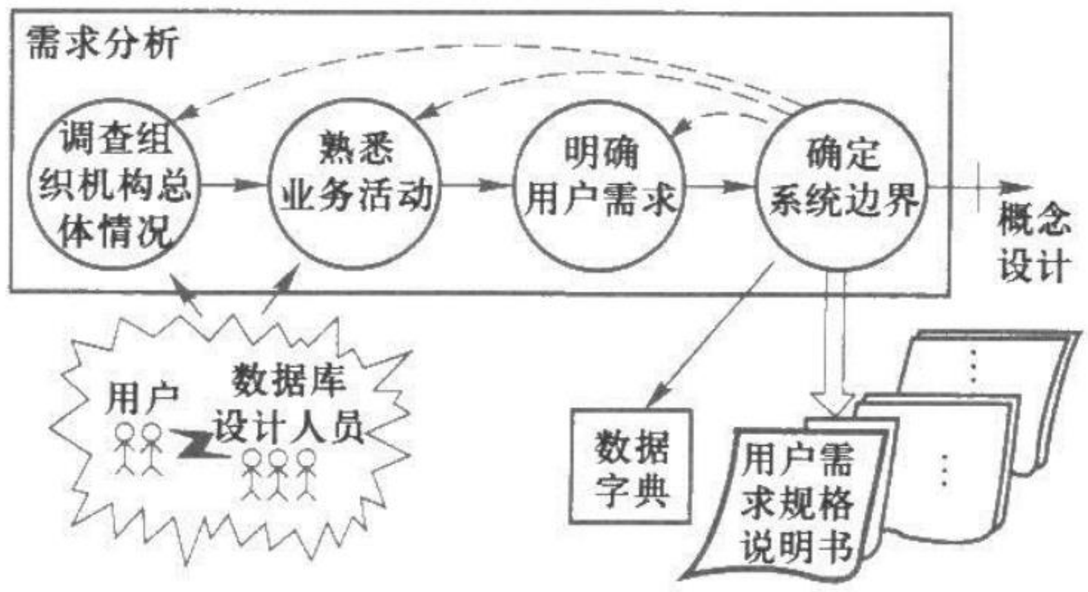


图 7-4　需求分析过程

3．数据字典

数据字典是系统中各类数据描述的集合，是进行详细的数据收集和数据分析所获得的主要成果。数据字典通常包括数据项、数据结构、数据流、数据存储和处理过程 5 个部分。

（1）数据项

①定义

数据项是不可再分的数据单位。

②描述

数据项描述={数据项名，数据项含义说明，别名，数据类型，长度，取值范围，取值含义，与其他数据项的逻辑关系，数据项之间的联系}

其中，“取值范围”、“与其他数据项的逻辑关系”定义了数据的完整性约束条件，是设计数据检验功能的依据。

（2）数据结构

①定义

数据结构反映了数据之间的组合关系。一个数据结构可以由若干个数据项组成，也可以由若干个数据结构组成，或由若干个数据项和数据结构混合组成。

②描述

数据结构描述={数据结构名，含义说明，组成：{数据项或数据结构}}

（3）数据流

①定义

数据流是数据结构在系统内传输的路径。

②描述

数据流描述={数据流名，说明，数据流来源，数据流去向，组成：{数据结构}，平均流量，高峰期流量}

其中，“数据流来源”是说明该数据流来自哪个过程；“数据流去向”是说明该数据流到哪个过程去。

（4）数据存储

①定义

数据存储是数据结构停留或保存的地方，也是数据流的来源和去向之一。

②描述

数据存储描述={数据存储名，说明，编号，输入的数据流，输出的数据流，组成：{数据结构}，数据量，存取频度，存取方式}

其中，“存取频度”指每小时或每天或每周存取几次、每次存取多少数据等信息；“存取方式”包括是批处理还是联机处理、是检索还是更新、是顺序检索还是随机检索等。

（5）处理过程

①定义

处理过程的具体处理逻辑一般用判定表或判定树来描述。数据字典中只需要描述处理过程的说明性信息。

②描述

处理过程描述={处理过程名，说明，输入：{数据流}，输出：{数据流}，处理：{简要说明}}

其中，“简要说明”中主要说明该处理过程的功能及处理要求。功能是指该处理过程用来做什么（而不是怎么做）；处理要求包括处理频度要求，如单位时间里处理多少事务、多少数据量、响应时间要求等。

③注意事项

a．设计人员应充分考虑到可能的扩充和改变，使设计易于更改，系统易于扩充；

b．必须强调用户的参与，这是数据库应用系统设计的特点。

三、概念结构设计

将需求分析得到的用户需求抽象为信息结构（即概念模型）的过程就是概念结构设计。它是整个数据库设计的关键。

1．概念模型

在需求分析阶段所得到的应用需求应该首先抽象为信息世界的结构，然后才能更好、更准确地用某一数据库管理系统实现这些需求。

概念模型的主要特点是：

（1）能真实、充分地反映现实世界，包括事物和事物之间的联系，能满足用户对数据的处理要求，是现实世界的一个真实模型。

（2）易于理解，可以用它和不熟悉计算机的用户交换意见。用户的积极参与是数据库设计成功的关键。

（3）易于更改，当应用环境和应用要求改变时容易对概念模型修改和扩充。

（4）易于向关系、网状、层次等各种数据模型转换。

概念模型是各种数据模型的共同基础，它比数据模型更独立于机器、更抽象，从而更加稳定。描述概念模型的有力工具是 E-R 模型。

2．E-R 模型

（1）实体之间的联系

a．一对一联系（1：1）

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

b．一对多联系（1：n）

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

c．多对多联系（m：n）

如果对于实体集 A 中的每一个实体，实体集 B 中有 n 个实体（n>10）与之联系，反之，对于实体集 B 中的每一个实体，实体集 A 中也有 m 个实体（m>10）与之联系，则称实体集 A 与实体集 B 具有多对多联系，记为 m：n。

（2）E-R 图

E-R 图提供了表示实体型、属性和联系的方法。

①实体型用矩形表示，矩形框内写明实体名。

②属性用椭圆形表示，并用无向边将其与相应的实体型连接起来。

③联系用菱形表示，菱形框内写明联系名，并用无向边分别与有关实体型连接起来，同时在无向边旁标上联系的类型（1：1、I：n 或 m：n）。

3．扩展的 E-R 模型

（1）ISA 联系

ISA 联系的一个重要的性质是子类继承了父类的所有属性，当然子类也可以有自己的属性。ISA 联系描述了对一个实体型中实体的一种分类方法。

（2）不相交约束与可重叠约束

不相交约束描述父类中的一个实体不能同时属于多个子类中的实体集，即一个父类中的实体最多属于一个子类实体集，用 ISA 联系三角形符号内加一个叉号“X”来表示。完备性约束描述父类中的一个实体是否必须是某一个子类中的实体，如果是，则叫做完全特化（total specialization），否则叫做部分特化（partial specialization）。

（3）基数约束

基数约束是对实体之间一对一、一对多和多对多联系的细化。参与联系的每个实体型用基数约束来说明实体型中的任何一个实体可以在联系中出现的最少次数和最多次数。约束用一个数对 min…max 表示，0≤min≤max。

（4）Part-of 联系

Part-of 联系即部分联系，它表明某个实体型是另外一个实体型的一部分。Part-of 联系可以分为两种情况，一种是整体实体如果被破坏，部分实体仍然可以独立存在，称为非独占的 Part-of 联系。非独占的 Part-of 联系可以通过基数约束来表达。与非独占联系相反，还有一种 Part-of 联系是独占联系。即整体实体如果被破坏，部分实体不能存在，在 E-R 图中用弱实体类型和识别联系来表示独占联系。如果一个实体型的存在依赖于其他实体型的存在，则这个实体型叫做弱实体型，否则叫做强实体型。

4．UML

UML 是对象管理组织（Obiect Management Group，OMG）的一个标准，它不是专门针对数据建模的，而是为软件开发的所有阶段提供模型化和可视化支持的规范语言，从需求规格描述到系统完成后的测试和维护都可以用到 UML。

（1）实体型

用类表示，矩形框中实体名放在上部，下面列出属性名。

（2）实体的码

在类图中在属性后面加“PK”（primarykey）来表示码属性。

（3）联系

用类图之间的“关联”来表示。早期的 UML 只能表示二元关联，关联的两个类用无向边相连，在连线上面写关联的名称。

5．概念结构设计

（1）实体与属性的划分原则

①作为属性，不能再具有需要描述的性质，即属性必须是不可分的数据项，不能包含其他属性。

②属性不能与其他实体具有联系，即 E-R 图中所表示的联系是实体之间的联系。凡满足上述两条准则的事物，一般均可作为属性对待。

（2）E-R 图的集成

E-R 图的集成一般需要分两步走，如图 7-5 所示。

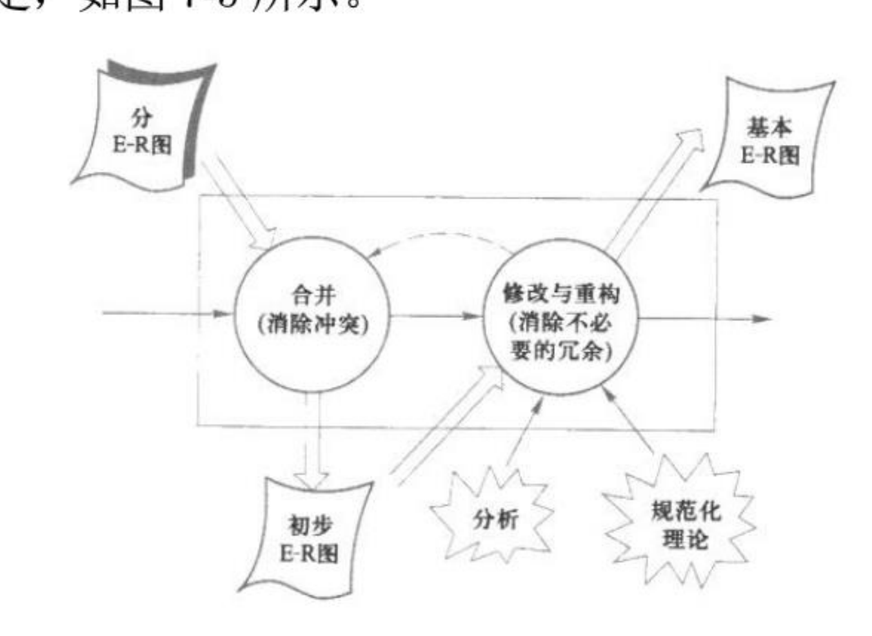


图 7-5　E-R 图集成

①合并 E-R 图，生成初步 E-R 图

合并这些 E-R 图时并不能简单地将各个 E-R 图画到一起，而是必须着力消除各个 E-R 图中的不一致，以形成一个能为全系统中所有用户共同理解和接受的统一的概念模型。合理消除各 E-R 图的冲突是合并 E-R 图的主要工作与关键所在。

各子系统的 E-R 图之间的冲突主要有三类：

a．属性冲突；

b．命名冲突；

c．结构冲突。

②消除不必要的冗余，设计基本 E-R 图

冗余的数据是指可由基本数据导出的数据，冗余的联系是指可由其他联系导出的联系。冗余数据和冗余联系容易破坏数据库的完整性，给数据库维护增加困难，应当予以消除。消除了冗余后的初步 E-R 图称为基本 E-R 图。消除冗余常见的两种方法为：

a．分析方法；

b．规范化理论。

四、逻辑结构设计

概念结构是独立于任何一种数据模型的信息结构，逻辑结构设计的任务就是把概念结构设计阶段设计好的基本 E-R 图转换为与选用数据库管理系统产品所支持的数据模型相符合的逻辑结构。

1．E-R 图向关系模型的转换

E-R 图转换为关系模型遵循如下原则：一个实体型转换为一个关系模式。实体的属性就是关系的属性，实体的码就是关系的码。

对于实体型间的联系则有以下不同的情况：

（1）一个 1:1 联系可以转换为一个独立的关系模式，也可以与任意一端对应的关系模式合并。如果转换为

一个独立的关系模式，则与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，每个实体的码均是该关系的候选码。如果与某一端实体对应的关系模式合并，则需要在该关系模式的属性中加入另一个关系模式的码和联系本身的属性。

（2）一个 1:n 联系可以转换为一个独立的关系模式，也可以与 n 端对应的关系模式合并。如果转换为一个独立的关系模式，则与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，而关系的码为 n 端实体的码。

（3）一个 m:n 联系转换为一个关系模式，与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，各实体的码组成关系的码或关系码的一部分。

（4）3 个或 3 个以上实体间的一个多元联系可以转换为一个关系模式。与该多元联系相连的各实体的码以及联系本身的属性均转换为关系的属性、各实体的码、组成关系的码或关系码的一部分。

（5）具有相同码的关系模式可合并。

2．数据模型的优化

关系数据模型优化通常以规范化理论为指导，方法为：

（1）确定数据依赖；

（2）对于各个关系模式之间的数据依赖进行极小化处理，消除冗余的联系；

（3）按照数据依赖的理论对关系模式逐一进行分析，考察是否存在部分函数依赖、传函数依赖、多值依赖等，确定各关系模式分别属于第几范式；

（4）按照需求分析阶段得到的处理要求，分析对于这样的应用环境这些模式是否合适确定是否要对某些模式进行合并或分解；

（5）对关系模式进行必要的分解，提高数据操作的效率和存储空间的利用率，常用的两种分解方法是水平分解和垂直分解：

①水平分解是把（基本）关系的元组分为若干子集合，定义每个子集合为一个子关系，以提高系统的效率；

②垂直分解是把关系模式 R 的属性分解为若干子集合，形成若干子关系模式。

3．设计用户子模式

定义用户外模式时可以注重考虑用户的习惯与方便。包括：

（1）使用更符合用户习惯的别名。

（2）可以对不同级别的用户定义不同的 View，以保证系统的安全性。

（3）简化用户对系统的使用。

五、物理结构设计

为一个给定的逻辑数据模型选取一个最适合应用要求的物理结构的过程，就是数据库的物理设计。

1．物理设计步骤

数据库的物理设计通常分为两步：

（1）确定数据库的物理结构，在关系数据库中主要指存取方法和存储结构。

（2）对物理结构进行评价，评价的重点是时间和空间效率。

如果评价结果满足原设计要求，则可进入到物理实施阶段，否则，就需要重新设计或修改物理结构，有时甚至要返回逻辑设计阶段修改数据模型。

2．数据库物理设计的内容和方法

希望设计优化的物理数据库结构，使得在数据库上运行的各种事务响应时间小、存储空间利用率高、事务吞吐率大，因此需要：

（1）首先对要运行的事务进行详细分析，获得选择物理数据库设计所需要的参数。

（2）要充分了解所用关系数据库管理系统的内部特征，特别是系统提供的存取方法和存储结构。

3．关系模式存取方法选择

（1）B+树索引存取方法的选择

选择索引存取方法就是根据应用要求确定对关系的哪些属性列建立索引、哪些属性列建立组合索引、哪些索引要设计为唯一索引等。

①如果一个（或一组）属性经常在查询条件中出现，则考虑在这个（或这组）属性上建立索引（或组合索引）。

②如果一个属性经常作为最大值和最小值等聚集函数的参数，则考虑在这个属性上建立索引。

③如果一个（或一组）属性经常在连接操作的连接条件中出现，则考虑在这个（或这组）属性上建立索引。

关系上定义的索引数并不是越多越好，系统为维护索引要付出代价，查找索引也要付出代价。

（2）hash 索引存取方法的选择

选择 hash 存取方法的规则如下：如果一个关系的属性主要出现在等值连接条件中或主要出现在等值比较选择条件中，而且满足下列两个条件之一，则此关系可以选择 hash 存取方法。

①一个关系的大小可预知，而且不变。

②关系的大小动态改变，但数据库管理系统提供了动态 hash 存取方法。

（3）聚簇存取方法的选择

为了提高某个属性（或属性组）的查询速度，把这个或这些属性（称为聚簇码）上具有相同值的元组集所存放在的连续物理块称为聚簇。

聚簇功能可以大大提高按聚簇码进行查询的效率，它不但适用于单个关系，也适用于经常进行连接操作的多个关系。一个数据库可以建立多个聚簇，一个关系只能加入一个聚簇。

①设计候选聚簇的条件

a．对经常在一起进行连接操作的关系可以建立聚簇；

b．如果一个关系的一组属性经常出现在相等比较条件中，则该单个关系可建立聚簇；

c．如果一个关系的一个（或一组）属性上的值重复率很高，则此单个关系可建立聚簇即对应每个聚簇码值的平均元组数不太少。

②检查候选聚簇

a．从聚簇中删除经常进行全表扫描的关系；

b．从聚簇中删除更新操作远多于连接操作的关系；

c．不同的聚簇中可能包含相同的关系，一个关系可以在某一个聚簇中，但不能同时加入多个聚簇。要从这多个聚簇方案（包括不建立聚簇）中选择一个较优的，即在这个聚簇运行各种事务的总代价最小。

4．确定数据库的存储结构

确定数据库物理结构主要指确定数据的存放位置和存储结构，包括确定关系、索引、聚簇、日志、备份等的存储安排和存储结构，确定系统配置等。

确定数据的存放位置和存储结构要综合考虑存取时间、存储空间利用率和维护代价三方面的因素。这三个方面常常是相互矛盾的，因此需要进行权衡，选择一个折中方案。

（1）确定数据的存放位置

为了提高系统性能，应该根据应用情况将数据的易变部分与稳定部分、经常存取部分和存取频率较低部分分开存放。

（2）确定系统配置

关系数据库管理系统产品一般都提供了一些系统配置变量和存储分配参数，供设计人员和数据库管理员对数据库进行物理优化。初始情况下，系统都为这些变量赋予了合理的默认值。但是这些值不一定适合每一种应用环境，在进行物理设计时需要重新对这些变量赋值，以改善系统的性能。

5．评价物理结构

评价物理数据库的方法完全依赖于所选用的关系数据库管理系统，主要是从定量估算各种方案的存储空间、存取时间和维护代价入手，对估算结果进行权衡、比较，选择出一个较优的、合理的物理结构。如果该结构不符合用户需求，则需要修改设计。

六、数据库的实施和维护

完成数据库的物理设计之后，设计人员就要用关系数据库管理系统提供的数据定义语言和其他实用程序将数据库逻辑设计和物理设计结果严格描述出来，成为关系数据库管理系统可以接受的源代码，再经过调试产生目标模式，然后就可以组织数据入库了，这就是数据库实施阶段。

1．数据的载入和应用程序的调试

数据库实施阶段包括两项重要的工作，一项是数据的载入，另一项是应用程序的编码和调试。组织数据录入就要将各源数据从各个局部应用中抽取出来，输入计算机，再分类转换，最后综合成符合新设计的数据库结构的形式，输入数据库。

2．数据库的试运行

在原有系统的数据有一小部分已输入数据库后，就可以开始对数据库系统进行联调试，这又称为数据库的试运行。这一阶段要实际运行数据库应用程序，执行对数据库的各种操作，测试应用程序的功能是否满足设计要求。在数据库试运行时，还要测试系统的性能指标，分析其是否达到设计目标。

3．数据库的运行和维护

在数据库运行阶段，对数据库经常性的维护工作主要是由 DBA 完成的，它包括：

（1）数据库的转储和恢复

DBA 要针对不同的应用要求制定不同的转储计划，以保证一旦发生故障能尽快将数据库恢复到某种一致的状态，并尽可能减少对数据库的破坏。

（2）数据库的安全性、完整性控制

（3）数据库性能的监督、分析和改造

有些 DBMS 产品提供了监测系统性能参数的工具，DBA 可以利用这些工具方便地得到系统运行过程中一系列性能参数的值。

（4）数据库的重组织与重构造

DBA 在重组织的过程中，按原设计要求重新安排存储位置、回收垃圾、减少指针链等，提高系统性能。

数据库的重组织并不修改原设计的逻辑和物理结构，而数据库的重构造则不同，它是指部分修改数据库的模式和内模式。

7.2 课后习题详解

1．试述数据库设计过程。

答：数据库设计过程的六个阶段：需求分析；概念结构设计；逻辑结构设计；数据库物理设计；数据库实施；数据库运行和维护。

（1）需求分析阶段：进行数据库设计首先必须准确了解与分析用户需求（包括数据与处理）。需求分析是整个设计过程的基础，是最困难、最耗费时间的一步。

（2）概念结构设计阶段：概念结构设计是整个数据库设计的关键，它通过对用户需求进行综合、归纳与抽象，形成一个独立于具体 DBMS 的概念模型。

（3）逻辑结构设计阶段：逻辑结构设计是将概念结构转换为某个 DBMS 所支持的数据模型，并对其进行优化。

（4）物理设计阶段：物理设计是为逻辑数据模型选取一个最适合应用环境的物理结构（包括存储结构和存取方法）。

（5）数据库实施阶段：在数据库实施阶段，设计人员运用 DBMS 提供的数据库语言（如 SQL）及其宿主语言，根据逻辑设计和物理设计的结果建立数据库，编制与调试应用程序，组织数据入库，并进行试运行。

（6）数据库运行和维护阶段：数据库应用系统经过试运行后即可投入正式运行。在数据库系统运行过程中必须不断地对其进行评价、调整与修改。

设计一个完善的数据库应用系统往往是上述六个阶段的不断反复。

2．试述数据库设计过程中形成的数据库模式。

答：数据库结构设计的不同阶段形成数据库的各级模式，即：

（1）在概念设计阶段形成独立于机器特点，独立于各个 DBMS 产品的概念模式，在本篇中就是 E-R 图；

（2）在逻辑设计阶段将 E-R 图转换成具体的数据库产品支持的数据模型，如关系模型，形成数据库逻辑模式，然后在基本表的基础上再建立必要的视图，形成数据的外模式；

（3）在物理设计阶段，根据 DBMS 特点和处理的需要，进行物理存储安排，建立索引，形成数据库内模式。

3．需求分析阶段的设计目标是什么？调查的内容是什么？

答：需求分析阶段的设计目标是通过详细调查现实世界要处理的对象（组织、部门、企业等），充分了解原系统（手工系统或计算机系统）工作概况，明确用户的各种需求，然后在此基础上确定新系统的功能。

调查的内容是“数据”和“处理”，即获得用户对数据库的如下要求：

（1）信息要求，指用户需要从数据库中获得信息的内容与性质，由信息要求可以导出数据要求，即在数据库中需要存储哪些数据；

（2）处理要求，指用户要完成什么处理功能，对处理的响应时间有什么要求，处理方式是批处理还是联机处理；

（3）安全性与完整性要求。

4．数据字典的内容和作用是什么？

答：（1）数据字典是系统中各类数据描述的集合。数据字典的内容通常包括数据项、数据结构、数据流、数据存储和处理过程五个部分。数据项是组成数据的最小组成单位，若干个数据项可以组成一个数据结构。数据字典通过对数据项和数据结构的定义来描述数据流和数据存储的逻辑内容。

（2）数据字典的作用：数据字典是关于数据库中数据的描述，在需求分析阶段建立，是下一步进行概念设计的基础，并在数据库设计过程中不断修改、充实、完善。

5．什么是数据库的概念结构？试述其特点和设计策略。

答：（1）在需求分析阶段所得到的应用需求应该首先抽象为信息世界的结构，才能更好地、更准确地用某一DBMS 实现这些需求。所以概念结构是信息世界的结构，即概念模型。

（2）其主要特点是：

①能真实、充分地反映现实世界，包括事物和事物之间的联系，能满足用户对数据的处理要求，是对现实世界的一个真实模型；

②易于理解，从而可以用它和不熟悉计算机的用户交换意见，用户的积极参与是数据库设计成功与否的关键；

③易于更改，当应用环境和应用要求改变时，容易对概念模型修改和扩充；

④易于向关系、网状、层次等各种数据模型转换。

（3）概念结构的设计策略通常有四种：

①自顶向下，即首先定义全局概念结构的框架，然后逐步细化；

②自底向上，即首先定义各局部应用的概念结构，然后将它们集成起来，得到全局概念结构；

③逐步扩张，首先定义最重要的核心概念结构，然后向外扩充，以滚雪球的方式逐步生成其他概念结构，直至总体概念结构；

④混合策略，即将自顶向下和自底向上相结合，用自顶向下策略设计一个全局概念结构的框架，以它为骨架集成由自底向上策略中设计的各局部概念结构。

6．定义并解释概念模型中以下术语：实体，实体型，实体集，属性，码，实体联系图（E-R 图）

答：（1）实体：客观存在并可以相互区分的事物。

（2）实体型：具有相同属性的实体具有相同的特征和性质，用实体名及其属性名集合来抽象和刻画同类实体。

（3）实体集：同型实体的集合。

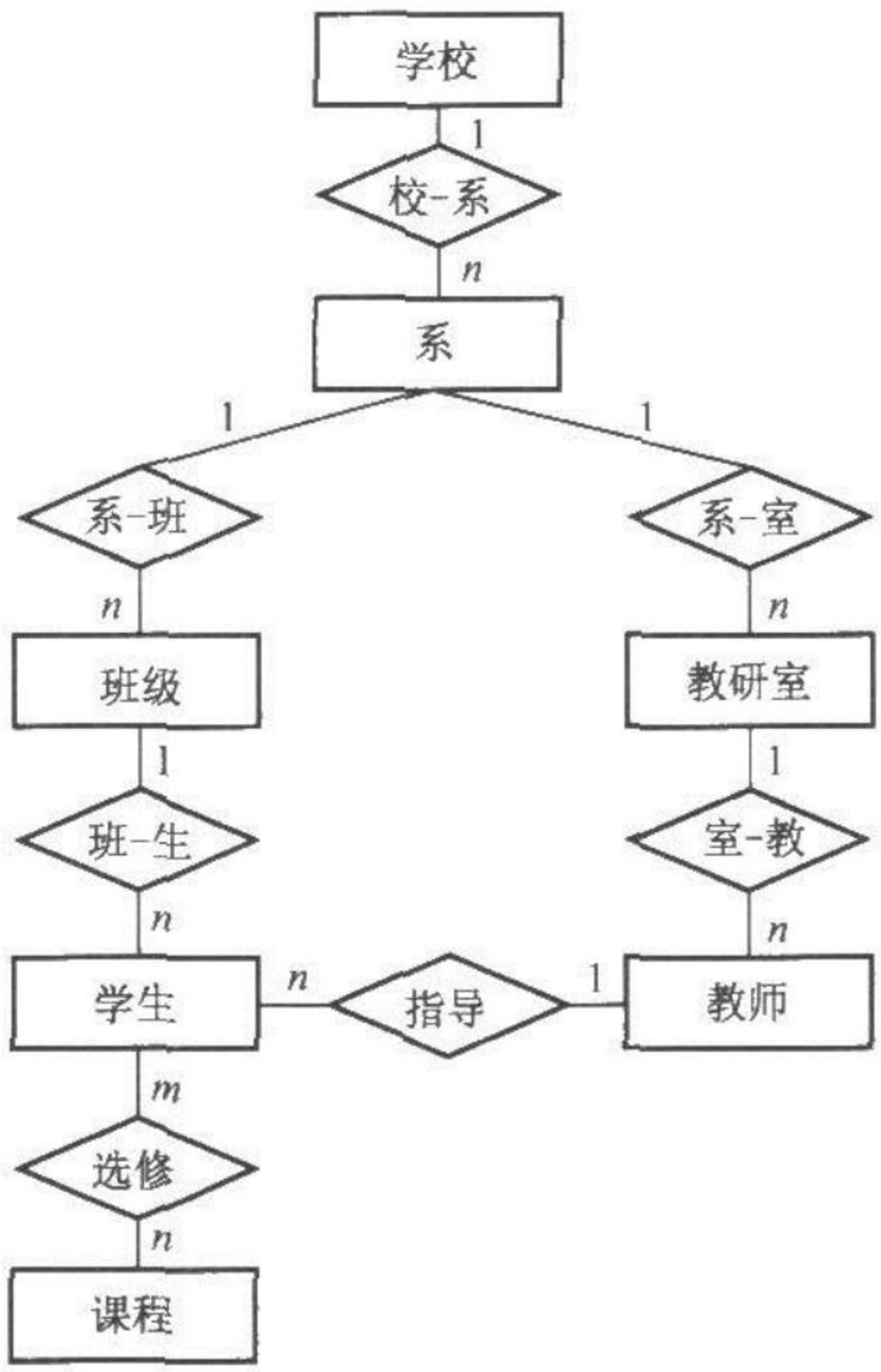
（4）属性：表中的一列即为一个属性。

（5）码：码就是能唯一标识实体的属性，他是整个实体集的性质，而不是单个实体的性质。

（6）实体联系图：提供了表示实体型、属性和联系的方法，用来描述现实世界的概念模型。

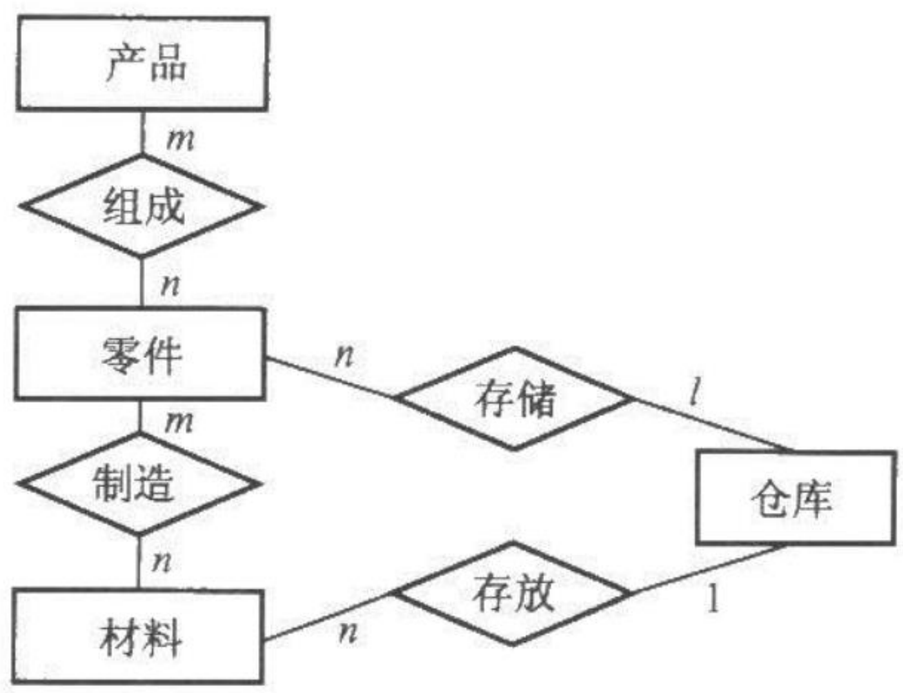
7．学校中有若干系，每个系有若干班级和教研室，每个教研室有若干教员，其中有的教授和副教授每人各带若干硪究生，每个班有若干学生，每个学生选修若干课程，每门课可由若干学生选修。请用 E-R 图画出此学校的概念模型。

答：



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

答：



9．什么是数据库的逻辑结构设计？试述其设计步骤。

答：（1）数据库的逻辑结构设计就是把概念结构设计阶段设计好的基本 E-R 图转换为与选用的 DBMS 产品所支持的数据模型相符合的逻辑结构。

（2）数据库的逻辑结构设计步骤为：

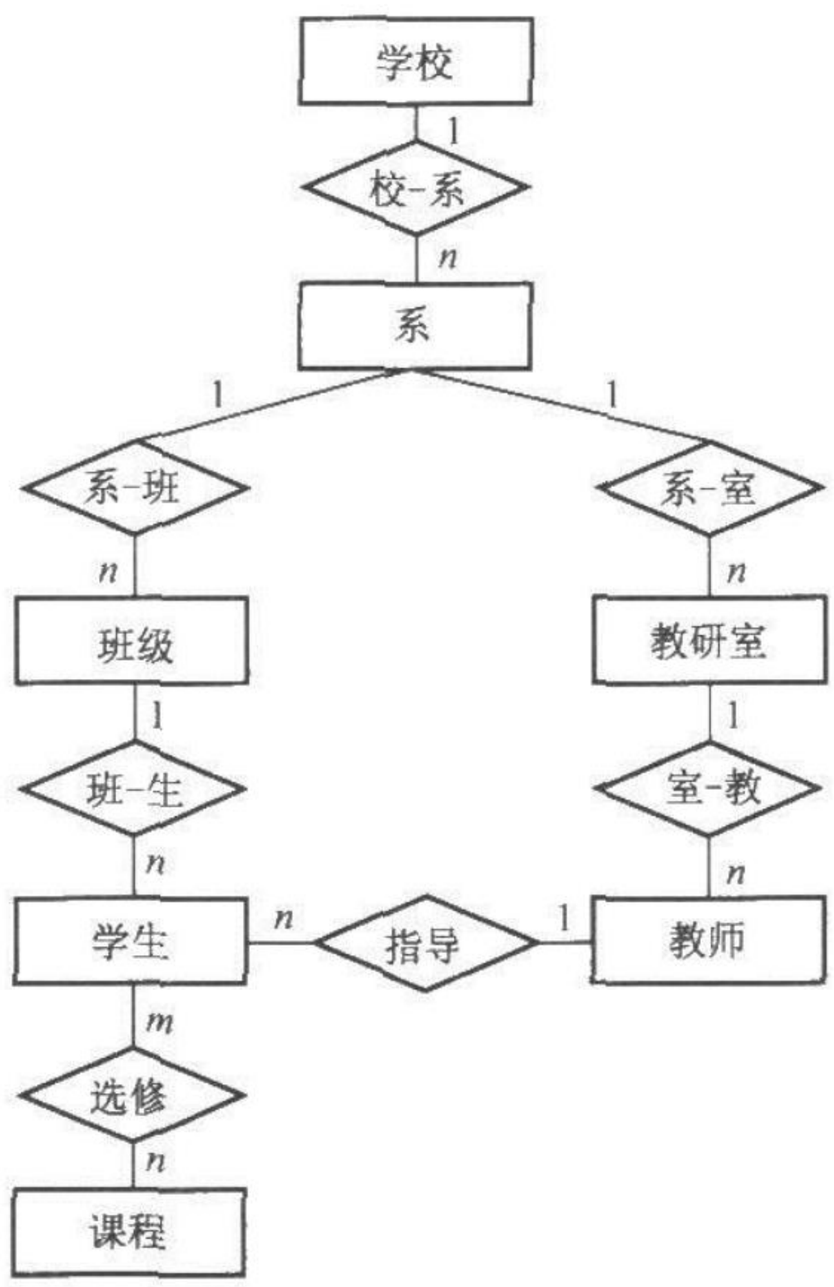
①将概念结构转换为一般的关系、网状、层次模型；

②将转换来的关系、网状、层次模型向特定 DBMS 支持下的数据模型转换；

③对数据模型进行优化。

10．试把习题 7 和习题 8 中的 E-R 图转换为关系模型。

答：（1）习题 7 中的 E-R 图为：



各实体的属性为：

系：系号，系名，学校名；

班级：班级号，班级名，系编号；

教研室：教研室编号，教研室名，系编号；

学生：学号，姓名，学历，班级号，导师职工号；

课程：课程号，课程名；

教员：职工号，姓名，职称，科研室编号；

各联系的属性为：

选修课：成绩。

其关系模型为：

系（系编号，系名，学校名）；

班级（班级号，班级名，系编号）；

教研室（教研室编号，教研室名，系编号）；

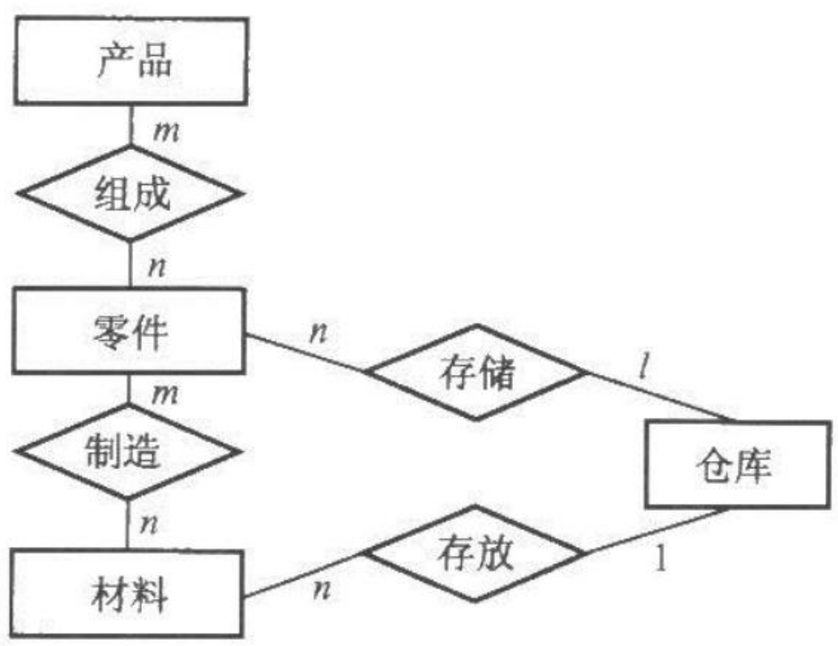
学生（学号，姓名，学历，班级号，导师职工号）；

课程（课程号，课程名）；

教员（职工号，姓名，职称，教研室编号）；

选课（学号，课程号，成绩）。

（2）习题 8 中的 E-R 图为：



对应的关系模型为：

产品（产品号，产品名，仓库号）；

零件（零件号，零件名）；
材料（材料号，材料名，类别，仓库号，存放量）；
仓库（仓库号，仓库名）；
产品组成（产品名，零件号，使用零件量）；
零件储存（零件号，仓库号，存储量）；
零件制造（零件号，材料号，使用材料量）。

11．试用规范化理论中有关范式的概念分析习题 7 设计的关系模型中各个关系模式的候选码，它们属于第几范式？会产生什么更新异常？

答：习题 7 中设计的两个关系数据库的各个关系模式的候选码都用下划线注明，这些关系模式都只有一个码，且都是惟一决定的因素，所以都属于 BCNF，不会产生更新异常现象。

12．规范化理论对数据库设计有什么指导意义？

答：规范化理论为数据库设计人员判断关系模式的优劣提供了理论标准，可用以指导关系数据模型的优化，用来预测模式可能出现的问题，为设计人员提供了自动产生各种模式的算法工具，使数据库设计工作有了严格的理论基础。

13．试述数据库物理设计的内容和步骤。

答：数据库在物理设备上的存储结构与存取方法称为数据库的物理结构，它依赖于选定的数据库管理系统。数据库物理设计的主要内容是为一个给定的逻辑数据模型选取一个最适合应用要求的物理结构。

数据库的物理设计步骤通常分为两步：

（1）确定数据库的物理结构，在关系数据库中主要指存取方法和存储结构；

（2）对物理结构进行评价，评价的重点是时间效率和空间效率。

14．数据输入在实施阶段的重要性是什么？如何保证输入数据的正确性？

答：（1）数据库是用来对数据进行存储、管理与应用的，因此在实施阶段必须将原有系统中的历史数据输入到数据库。数据量一般都很大，而且数据来源于部门中的各个不同的单位。数据的组织方式、结构和格式都与新设计的数据库系统有相当的差距，组织数据录入就要将各类源数据从各个局部应用中抽取出来，分类转换，最后综合成符合新设计的数据库结构的形式，输入数据库。因此这样的数据转换、组织入库的工作是相当费力费时的工作。特别是原系统是手工数据处理系统时，各类数据分散在各种不同的原始表格、凭证、单据之中，数据输入工作量更大。

（2）保证输入数据正确性的方法：为提高数据输入工作的效率和质量，应该针对具体的应用环境设计一个数据录入子系统，由计算机来完成数据入库的任务。在源数据入库之前要采用多种方法对其进行检验，以防止不正确的数据入库。

15．什么是数据库的再组织和重构造？为什么要进行数据库的再组织和重构造？

答：（1）数据库的再组织是指按原设计要求重新安排存储位置、回收垃圾、减少指针链等，以提高系统性能。数据库的重构造则是指部分修改数据库的模式和内模式，即修改原设计的逻辑和物理结构。数据库的再组织是不修改数据库的模式和内模式的。

（2）进行数据库的再组织和重构造的原因：数据库运行一段时间后，由于记录不断增、删、改，会使数据库的物理存储情况变坏，降低了数据的存取效率，数据库性能下降，这时 DBA 就要对数据库进行重组织。DBMS 一般都提供用于数据重组织的实用程序。数据库应用环境常常发生变化，如增加新的应用或新的实体，取消了某些应用，有的实体与实体间的联系也发生了变化等，使原有的数据库设计不能满足新的需求，需要调整数据库的模式和内模式，这就要进行数据库重构造。

第 8 章　数据库编程

8.1　复习笔记

一、嵌入式 SQL

1．嵌入式 SQL 的处理过程

（1）嵌入式 SQL 定义

嵌入式 SQL 是将 SQL 语句嵌入程序设计语言中，被嵌入的程序设计语言，如 C、C++、 Java，称为宿主语言，简称主语言。

（2）嵌入式 SQL 处理过程

对 ESQL，RDBMS 一般采用预编译方法处理，即由 RDBMS 的预处理程序对源程序进行扫描，识别出 ESQL 语句，把它们转换成主语言调用语句，以使主语言编译程序能识别它们，然后由主语言的编译程序将纯的主语言程序编译成目标码，如图 8-1 所示。

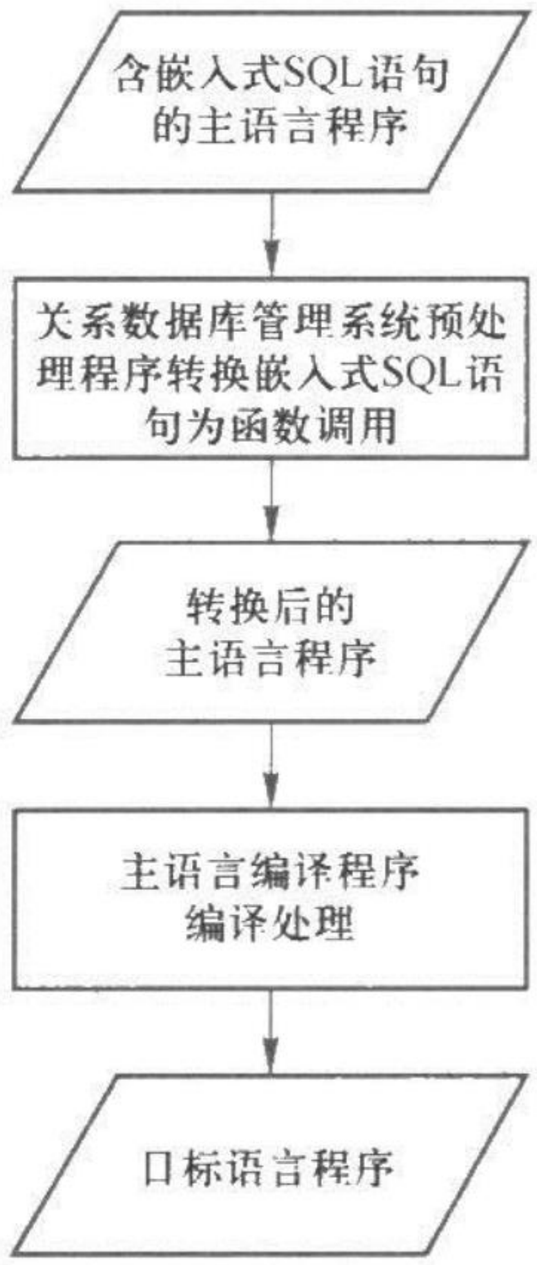


图 8-1　嵌入式 SQL 基本处理过程

（3）ESQL 执行语句

在 ESQL 中，为了能够区分 SQL 语句与主语言语句，所有 SQL 语句都必须加前缀“EXEC SQL”，以“；”结束成为一个程序片断：“EXEC SQL<SQL 语句>；”。

2．嵌入式 SQL 语句与主语言之间的通信

将 SQL 嵌入到高级语言中混合编程，SQL 语句负责操纵数据库，高级语言语句负责控制逻辑流程。这时程序中会含有两种不同计算模型的语句，它们之间应该如何通信呢?

（1）数据库工作单元与源程序工作单元之间的通信内容

①向主语言传递 SQL 语句的执行状态信息，使主语言能够据此信息控制程序流程，主要用 SQL 通信区（SQL CommunicationArea，SQLCA）实现。

②主语言向 SQL 语句提供参数，主要用主变量（host variable）实现。

③将 SQL 语句查询数据库的结果交主语言处理，主要用主变量和游标（cursor）实现。

（2）SQL 通信区

SQL 语句执行后，系统要反馈给应用程序若干信息，这些信息将送到 SQL 通信区中，应用程序从 SQL 通信区中取出这些状态信息，据此决定接下来执行的语句。

SQL 通信区中有一个变量 SQLCODE，用来存放每次执行 SQL 语句后返回的代码。

应用程序每执行完一条 SQL 语句之后都应该测试一下 SQLCODE 的值，以了解该 SQL 语句执行情况并做相

应处理。。

（3）主变量

①定义

SQL 语句中使用的主语言程序变量简称为主变量。

②输入主变量和输出主变量

主变量分为输入主变量和输出主变量。输入主变量由应用程序对其赋值，SQL 语句引用；输出主变量由 SQL 语句对其赋值或设置状态信息，返回给应用程序。

③指示变量

指示变量是一个整型变量，用来“指示”所指主变量的值或条件。指示变量可以指示输入主变量是否为空值，可以检测输出主变量是否空值，值是否被截断。

④执行语句格式

所有主变量和指示变量必须在 SQL 语句 BEGIN DECLARE SECTION 与 END DECLARE SECTION 之间进行说明。

为了与数据库对象名（表名、视图名、列名等）区别，SQL 语句中的主变量名和指示变量前要加冒号“:”作为标志。

（4）游标

游标是系统为用户开设的一个数据缓冲区，存放 SQL 语句的执行结果，每个游标区都有一个名字。用户可以通过游标逐一获取记录，并赋给主变量，交由主语言进一步处理。

（5）建立和关闭数据库连接

嵌入式 SQL 程序要访问数据库必须先连接数据库。RDBMS 根据用户信息对连接请求进行合法性验证，只有通过身份验证，才能建立一个可用的合法连接。

①建立数据库连接

建立连接的 ESQL 语句是：

EXEC SQL CONNECT TO target[AS connection-name][USER user-name]；

a．target 是要连接的数据库服务器，它可以是一个常见的服务器标识串，或者是包含服务器标识的 SQL 串常量，也可以是 DEFAULT。

b．connection-name 是可选的连接名，连接必须是一个有效的标识符，主要用来识别一个程序内同时建立的多个连接，如果在整个程序内只有一个连接也可以不指定连接名。

②关闭数据库连接

当某个连接上的所有数据库操作完成后，应用程序应该主动释放所占用的连接资源。

关闭数据库连接的 ESQL 语句是：

EXEC SQL DISCONNECT[connection-name]；

其中 connection-name 是 EXEC SQL CONNECT 所建立的数据库连接。

3．不用游标的 SQL 语句

有的嵌入式 SQL 语句不需要使用游标。它们是：说明性语句、数据定义语句、数据控制语句、查询结果为单记录的 SELECT 语句、非 CURRENT 形式的增删改语句。

（1）查询结果为单记录的 SELECT 语句

这类语句不需要使用游标，因为查询结果只有一个，只需要用 INTO 子句指定存放查询结果的主变量。

使用单记录的 SELECT 语句需要注意以下几点：

①INTO 子句、WHERE 子句和 HAVING 短语的条件表达式中均可以使用主变量；

②查询结果为空值的处理

查询返回的记录中，可能某些列为空值 NULL。当指示变量值为负值时，不管主变量为何值，均认为主变量值为 NULL，指示变量只能用于 INTO 子句中。

③如果查询结果实际上并不是单条记录，而是多条记录，则程序出错，RDBMS 会在 SQLCA 中返回错误信息。

（2）非 CURRENT 形式的增删改语句

有些增删改语句不需要使用游标，不是 CURRENT 形式的。在 UPDATE 的 SET 子句和 WHERE 子句中可以使用主变量，SET 子句还可以使用指示变量。

4．使用游标的 SQL 语句

必须使用游标的 SQL 语句有查询结果为多条记录的 SELECT 语句、CURRENT 形式的 UPDATE 和 DELETE

语句。

（1）查询结果为多条记录的 SELECT 语句

一般情况下，SELECT 语句查询结果是多条记录，因此需要用游标机制将多条记录一次一条地送主程序处理，从而把对集合的操作转换为对单个记录的处理。

使用游标的步骤为：

①说明游标

用 DECLARE 语句为一条 SELECT 语句定义游标：

EXEC SQL DECLARE <游标名> CURSOR FOR <SELECT 语句>;

定义游标仅仅是一条说明性语句，这时关系数据库管理系统并不执行 SELECT 语句。

②打开游标

用 OPEN 语句将定义的游标打开。

EXEC SQL OPEN <游标名>;

打开游标实际上是执行相应的 SELECT 语句，把查询结果取到缓冲区中。这时游标处于活动状态，指针指向查询结果集中的第一条记录。

③推进游标指针并取当前记录

EXEC SQL FETCH <游标名>
INTO <主变量>[<指示变量>][,<主变量>[<指示变量>]]…;

其中主变量必须与 SELECT 语句中的目标列表达式具有一一对应关系。

用 FETCH 语句把游标指针向前推进一条记录，同时将缓冲区中的当前记录取出来送至主变量供主语言进一步处理。通过循环执行 FETCH 语句逐条取出结果集中的行进行处理。

④关闭游标

用 CLOSE 语句关闭游标，释放结果集占用的缓冲区及其他资源。

EXEC SQL CLOSE <游标名>;

游标被关闭后就不再和原来的查询结果集相联系。但被关闭的游标可以再次被打开，与新的查询结果相联系。

（2）CURRENT 形式的 UPDATE 和 DELETE 语句

UPDATE 语句和 DELETE 语句都是集合操作，UPDATE 语句和 DELETE 语句中要用子句“WHERE CURRENT OF<游标名>”来表示修改或删除的是最近一次取出的记录，即游标指针指向的记录。

5．动态 SQL

某些应用到执行时才能够确定要提交的 SQL 语句、查询的条件，需要使用动态 SQL 来解决。动态 SQL 支持动态组装 SQL 语句和动态参数两种形式。

（1）使用 SQL 语句主变量

程序主变量包含的内容是 SQL 语句的内容，而不是原来保存数据的输入或输出变量，这样的变量称为 SQL 语句主变量。SQL 语句主变量在程序执行期间可以设定不同的 SQL 语句，然后立即执行。

（2）动态参数

动态参数是 SQL 语句中的可变元素，使用参数符号“?”表示该位置的数据在运行时设定。动态参数的输入是通过 prepare 语句准备主变量和执行(execute)时绑定数据或主变量来完成。使用动态参数的步骤有：

①声明 SQL 语句主变量

②准备 SQL 语句(PREPARE)

PREPARE 将分析含主变量的 SQL 语句内容，建立语句中包含的动态参数的内部描述符，并用<语句名>标识它们的整体。

EXEC SQL PREPARE<语句名>FROM<SQL 语句主变量>;

③执行准备好的语句(EXECUTE)

EXECUTE 将 SQL 语句中分析出的动态参数和主变量或数据常量绑定作为语句的输入或输出变量。

EXEC SQL EXECUTE<语句名>[INTO<主变量表>][USING<主变量或常量>];

二、过程化 SQL

1．过程化 SQL 的块结构

过程化 SQL 是对 SQL 的扩展，使其增加了过程化语句功能。过程化 SQL 程序的基本结构是块。所有的过程化 SQL 程序都是由块组成的。这些块之间可以互相嵌套，每个块完成一个逻辑操作。图 8-2 是过程化 SQL 块的基本结构。

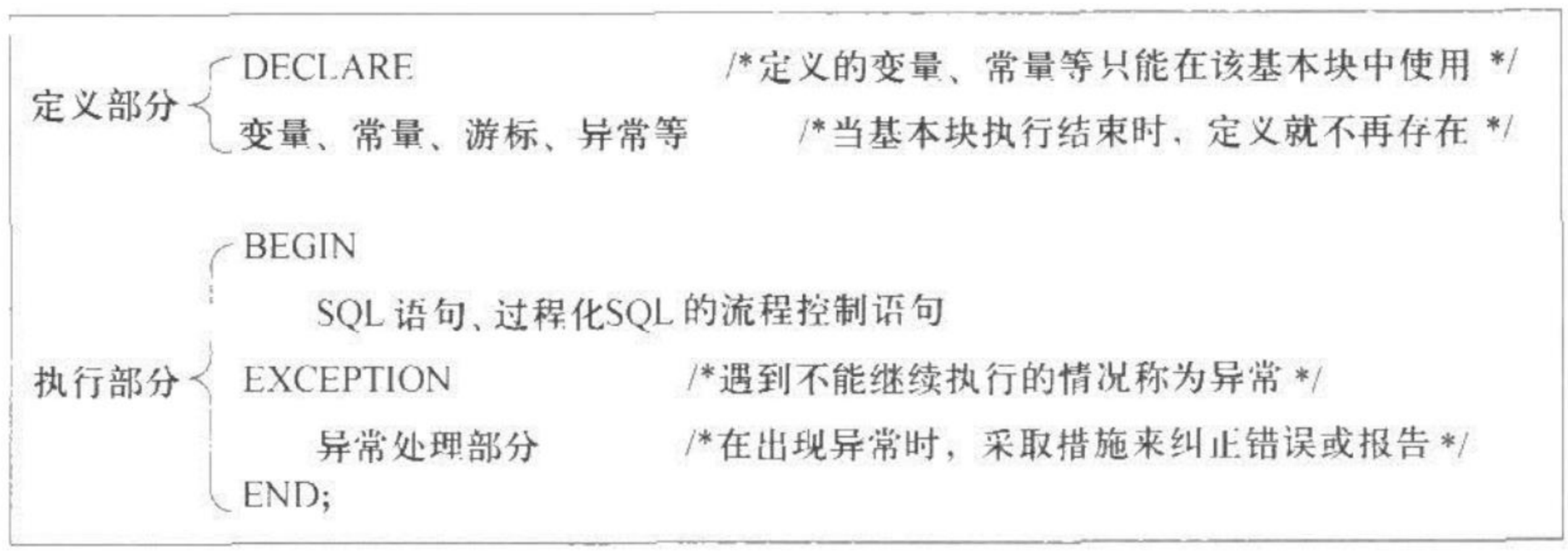


图 8-2　过程化 SQL 块的基本结构

2．变量和常量的定义

（1）变量定义

变量名 数据类型 [[NOT NULL] :=初值表达式] 或

变量名 数据类型 [[NOT NULL] 初值表达式]

（2）常量的定义

常量名数据类型 CONSTANT：=常量表达式

常量必须要给一个值，并且该值在存在期间或常量的作用域内不能改变。如果试图修改它，过程化 SQL 将返回一个异常。

（3）赋值语句

变量名：=表达式

3．流程控制

（1）条件控制语句

①IF 语句

```
IF condition THEN
    Sequence_of_statements; /*条件为真时语句序列才被执行*/
END IF;                     /*条件为假或 NULL 时什么也不做，控制转移至下一个语句*/
```

②IF-THEN 语句

```
IF condition THEN
    Sequence_of_statements1;    /*条件为真时执行语句序列 1*/
ELSE
    Sequence_of_statements2;    /*条件为假或 NULL 时执行语句序列 2*/
END IF;
```

③嵌套的 IF 语句

在 THEN 和 ELSE 子句中还可以再包含 IF 语句，即 IF 语句可以嵌套。

（2）循环控制语句

①最简单的循环语句 LOOP

```
LOOP
    Sequence_of_statements;          /*循环体，一组过程化 SQL 语句*/
END LOOP;
```

多数数据库服务器的过程化 SQL 都提供 EXIT、BREAK 或 LEAVE 等循环结束语句，以保证 LOOP 语句块能够在适当的条件下提前结束。

②WHILE-LOOP 循环语句

```
WHILE condition LOOP
    Sequence_of_statements;          /*条件为真时执行循环体内的语句序列*/
END LOOP;
```

每次执行循环体语句之前首先要对条件进行求值，如果条件为真则执行循环体内的语句序列，如果条件为假则跳过循环并把控制传递给下一个语句。

③FOR-LOOP 循环语句

```
FOR count  IN  [REVERSE] bound1 .. bound2  LOOP
    Sequence_of_statements;
END LOOP;
```

（3）错误处理

如果过程化 SQL 在执行时出现异常，则应该让程序在产生异常的语句处停下来，根据异常的类型去执行异常处理语句。

SQL 标准对数据库服务器提供什么样的异常处理做出了建议，要求过程化 SQL 管理器提供完善的异常处理机制。

三、存储过程和函数

过程化 SQL 块主要有两种类型，即命名块和匿名块。匿名块每次执行时都要进行编译，它不能被存储到数据库中，也不能在其他过程化 SQL 块中调用。过程和函数是命名块，它们被编译后保存在数据库中，称为持久性存储模块（PersistentStored Module，PSM），可以被反复调用，运行速度较快。

1．存储过程

（1）存储过程的优点

①运行效率高。

②降低了客户机和服务器之间的通信量。

③方便实施企业规则。

（2）存储过程的用户接口

用户通过下面的 SQL 语句创建、执行、修改和删除存储过程。

①创建存储过程

```
CREATE OR REPLACE PROCEDURE 过程名（[参数 1,参数 2,...]）    /*存储过程首部*/
AS <过程化 SQL 块>;              /*存储过程体，描述该存储过程的操作*/
```

②执行存储过程

```
CALL / PERFORM PROCEDURE 过程名（[参数 1，参数 2…]）;
```

③修改存储过程

a．使用 ALTER PROCEDURE 重命名一个存储过程

```
ALTER PROCEDURE 过程名 1   RENAME TO 过程名 2;
```

b．使用 ALTER PROCEDURE 重新编译个存储过程

```
ALTER PROCEDURE 过程名 COMPILE;
```

④删除存储过程

```
DROP PROCEDURE 过程名 O;
```

2．函数

（1）函数的定义语句格式

CREATE OR REPLACE FUNCTION 函数名（[参数 1, 参数 2, ...]）RETURNS <类型>
AS <过程化 SQL 块>;

（2）函数的执行语句格式

CALL/SELECT 函数名([参数 1,参数 2,...]);

（3）修改函数

①使用 ALTERFUNCTION 重命名一个自定义函数

ALTER FUNCTION 过程名 1 RENAME TO 过样名 2;

②使用 ALTER FUNCTION 重新编译一个函数

ALTER FUNCTION 函数名 COMPILE;

3．过程化 SQL 中的游标

在过程化 SQL 中如果 SELECT 语句只返回一条记录，可以将该结果存放到变量中。当查询返回多条记录时，就要使用游标对结果集进行处理。一个游标与一个 SQL 语句相关联。

四、ODBC 编程

1．ODBC 概述

ODBC 是微软公司开放服务体系（Windows Open ServicesArchitecture，WOSA）中有关数据库的一个组成部分，它建立了一组规范，并提供一组访问数据库的应用程序编程接口（ApplicationProgrammingInterface，API）。ODBC 具有两重功效或约束力：一方面规范应用开发，另一方面规范关系数据库管理系统应用接口。

2．ODBC 工作原理概述

使用 0DBC 开发应用系统，体系结构如图 8-3 所示，它由四部分构成：用户应用程序、驱动程序管理器(ODBC Driver Manager)、数据库驱动程序(ODBC Driver)、数据源(如 RDBM 和数据库)。

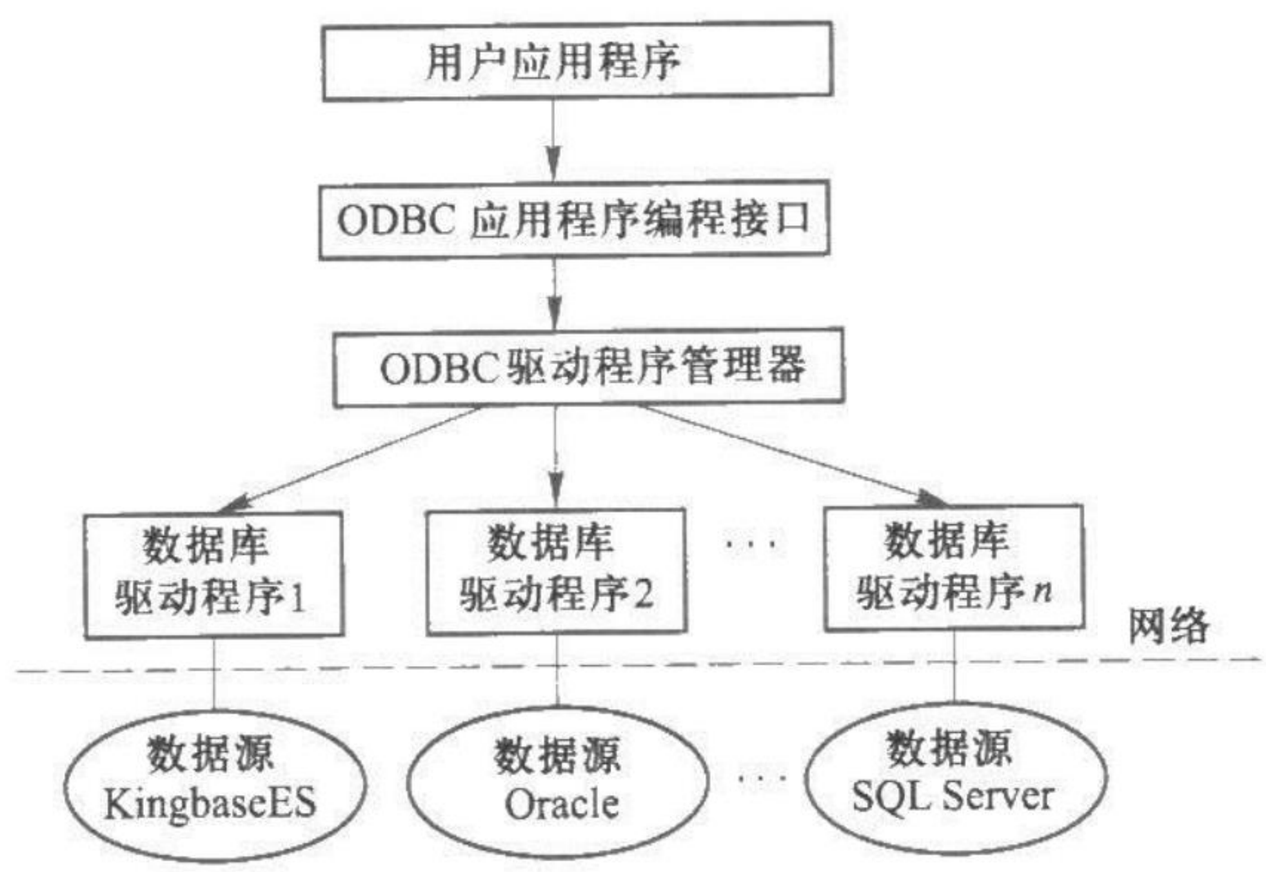


图 8-3　ODBC 应用系统的体系结构

（1）应用程序

应用程序提供用户界面、应用逻辑和事务逻辑。使用 ODBC 来开发应用系统的程序简称为 ODBC 应用程序，包括的内容有：

①请求连接数据库。

②向数据源发送 SQL 语句。

③为 SQL 语句执行结果分配存储空间，定义所读取的数据格式。

④获取数据库操作结果，或处理错误。

⑤进行数据处理并向用户提交处理结果。

⑥请求事务的提交和回滚操作。

⑦断开与数据源的连接。

（2）驱动程序管理器

驱动程序管理器是管理应用程序和驱动程序之间的通信。

驱动程序管理器的主要功能包括；

①装载 ODBC 驱动程序选择和连接正确的驱动程序。

②管理数据源。

③检查 ODBC 调用参数的合法性及记录 ODBC 函数的调用。

④当应用层需要时返回驱动程序的有关信息。

ODBC 驱动程序管理器可以建立、配置或删除数据源，并查看系统当前所安装的数据库 ODBC 驱动程序。

（3）数据库驱动程序

①作用

ODBC 通过驱动程序来提供应用系统与数据库平台的独立性。ODBC 的各种操作请求由驱动程序管理器提交给某个 RDBMS 的 ODBC 驱动程序，通过调用驱动程序所支持的函数来存取数据库。数据库的操作结果也通过驱动程序返回给应用程序。

②分类

ODBC 驱动程序主要有单束和多束两类。

a．单束驱动程序

单束驱动程序一般是数据源和应用程序在同一台机器上，驱动程序直接完成对数据文件的 I/O 操作。

b．多束驱动程序

多束驱动程序由驱动程序完成数据库访问请求的提交和结果集接收，应用程序使用驱动程序提供的结果集管理接口操纵执行后的结果数据。

（4）ODBC 数据源管理

数据源是最终用户需要访问的数据，包含了数据库位置和数据库类型等信息，是一种数据连接的抽象。

ODBC 给每个被访问的数据源指定唯一的数据源名（简称 DSN），在连接中，用数据源名来代表用户名、服务器名、所连接的数据库名等。最终用户无需知道 DBMS 或其他数据管理软件、网络以及有关 ODBC 驱动程序的细节。

3．ODBC API 基础

（1）一致性

各个数据库厂商的 ODBC 应用程序接口（简称 ODBC API）都要符合两方面的一致性：

①API 一致性

API 一致性级别有核心级、扩展 1 级、扩展 2 级；

②语法一致性

语法一致性级别有最低限度 SQL 语法级、核心 SQL 语法级、扩展 SQL 语法级。

（2）函数概述

ODBC 3.0 标准提供了 76 个函数接口，大致可以分为：

①分配和释放环境句柄、连接句柄、语句句柄；

②连接函数(SQLDriverconnect 等)；

③与信息相关的函数(如获取描述信息函数 SQLGetinfo、SQLGetFunction)；

④事务处理函数(如 SQLEndTran)；

⑤执行相关函数(SQLExecdirect、SQLExecute 等)；

⑥编目函数。

（3）句柄及其属性

句柄是 32 位整数值，代表一个指针。ODBC 3.0 中句柄可以分为环境句柄、连接句柄、语句句柄或描述符句柄四类，对于每种句柄不同的驱动程序有不同的数据结构，这四种句柄的关系如图 8-4 所示。

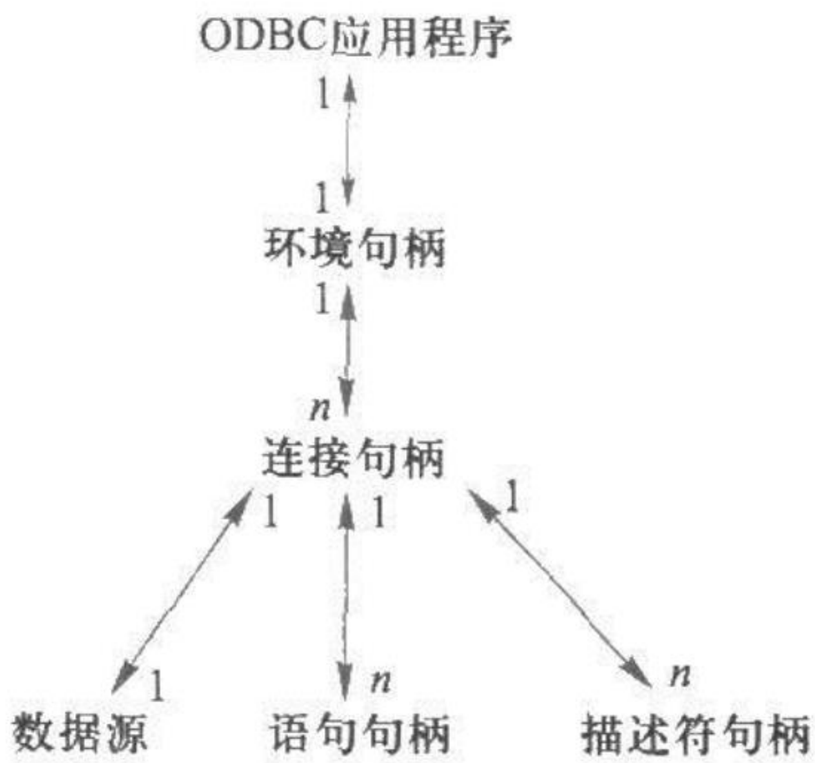


图 8-4　应用程序句柄之间的关系

①环境句柄

每个 ODBC 应用程序需要建立一个 ODBC 环境，分配一个环境句柄，存取数据的全局性背景如环境状态、当前环境状态诊断、当前在环境上分配的连接句柄等；

②连接句柄

一个环境句柄可以建立多个连接句柄，每一个连接句柄实现与一个数据源之间的连接；

③语句句柄

在一个连接中可以建立多个语句句柄，它不只是一个 SQL 语句，还包括 SQL 语句产生的结果集以及相关的信息等；

④描述句柄

在 ODBC 3.0 中提出了描述符句柄的概念，它是描述 SQL 语句的参数、结果集列的元数据集合。

（4）数据类型

ODBC 定义了两套数据类型，即 SQL 数据类型和 C 数据类型。SQL 数据类型用于数据源，而 C 数据类型用于应用程序的 C 代码。它们之间的转换情况如表 8-1 所示。

表 8-1　SQL 数据类型和 C 数据类型之间的转换规则

	SQL 数据类型	C 数据类型
SQL 数据类型	数据源之间转换	应用程序变量传送到语句参数（SQLBindparameter）
C 数据类型	从结果集列中返回剑应用程序变量（SQLBindcol）	应用程序变量之间转换

4．ODBC 的工作流程

使用 ODBC 的应用系统大致的工作流程如图 8-5 所示。

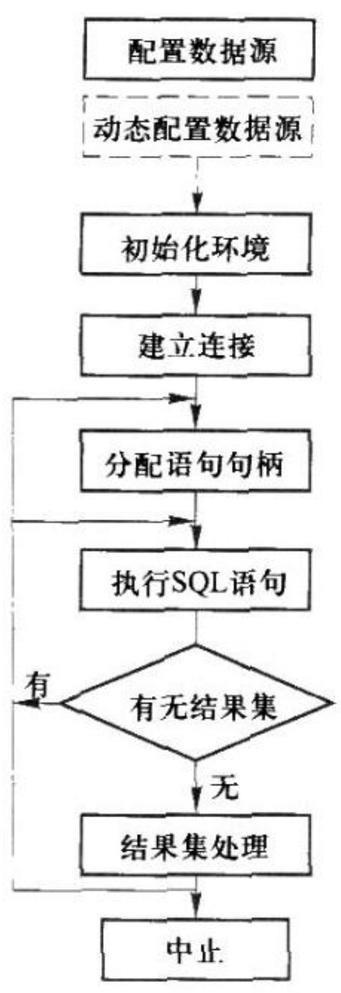


图 8-5　ODBC 的工作流程

（1）配置数据源

配置数据源有两种方法：

①运行数据源管理工具来进行配置；

②使用 Driver Manager 提供的 ConfigDsn 函数来增加、修改或删除数据源。这种方法特别适用于在应用程序中创建的临时使用的数据源。

（2）初始化环境

应用程序通过调用连接数和某个数据源进行连接后，Driver Manager 调用所连的驱动程序中的 SQLAllocHandle，分配环境句柄的数据结构。

（3）建立连接

应用程序调用 SQLAllocHandle 分配连接句柄，通过 SQLConnect、SQLDriverconnect、SQLBrowseconnect 与数据源连接。

（4）分配语句句柄

语句句柄含有具体的 SQL 语句以及输出的结果集等信息。在后面的执行函数中，语句句柄都是必要的输入参数。应用程序还可以通过 SQL GetStmtAttr 来设置语句属性(也可以使用默认值)。

（5）执行 SQL 语句

应用程序处理 SQL 语句的方式有两种：预处理(SQLPrepare、SQLExecute 适用于语句的多次执行)或直接执行(SQLExecdirect)。

（6）结果集处理

ODBC 中使用游标来处理结果集数据。

①游标分类

a．forward-only 游标

只能在结果集中向前滚动，它是 0DBC 的默认游标类型。

b．可滚动游标

又可以分为静态(static)、动态(dynamic)，码集驱动和混合型(mixed)四种。

②游标打开方式

当结果集刚刚生成时，游标指向第一行数据之前,应用程序通过 SQLBindCol，把查询结果绑定到应用程序缓冲区中，通过 SQLFetch 或是 SQLFetchScroll 来移动游标获取结果集中的每一行数据。最后通过 SQLCloseCursor 来关闭游标。

（7）中止处理

处理结束后，应用程序将首先释放语句句柄，然后释放数据库连接，并与数据库服务器断开，最后释放 ODBC 环境。

五、OLE DB

1．定义

OLE DB 是基于组件对象模型（Component ObjectModel，COM）来访问各种数据源的 ActiveX 的通用接口，它提供访问数据的一种统一手段，而不管存储数据时使用的方法如何。OLE DB 支持的数据源可以是数据库，也可以是文本文件、Excel 表格、ISAM 等各种不同格式的数据存储。

2．结构

图 8-6 是一个基于 OLE DB 体系结构设“程序的编程模型。OLE DB 体系结构中包含消费者（consumer）和提供者（provider）两部分。

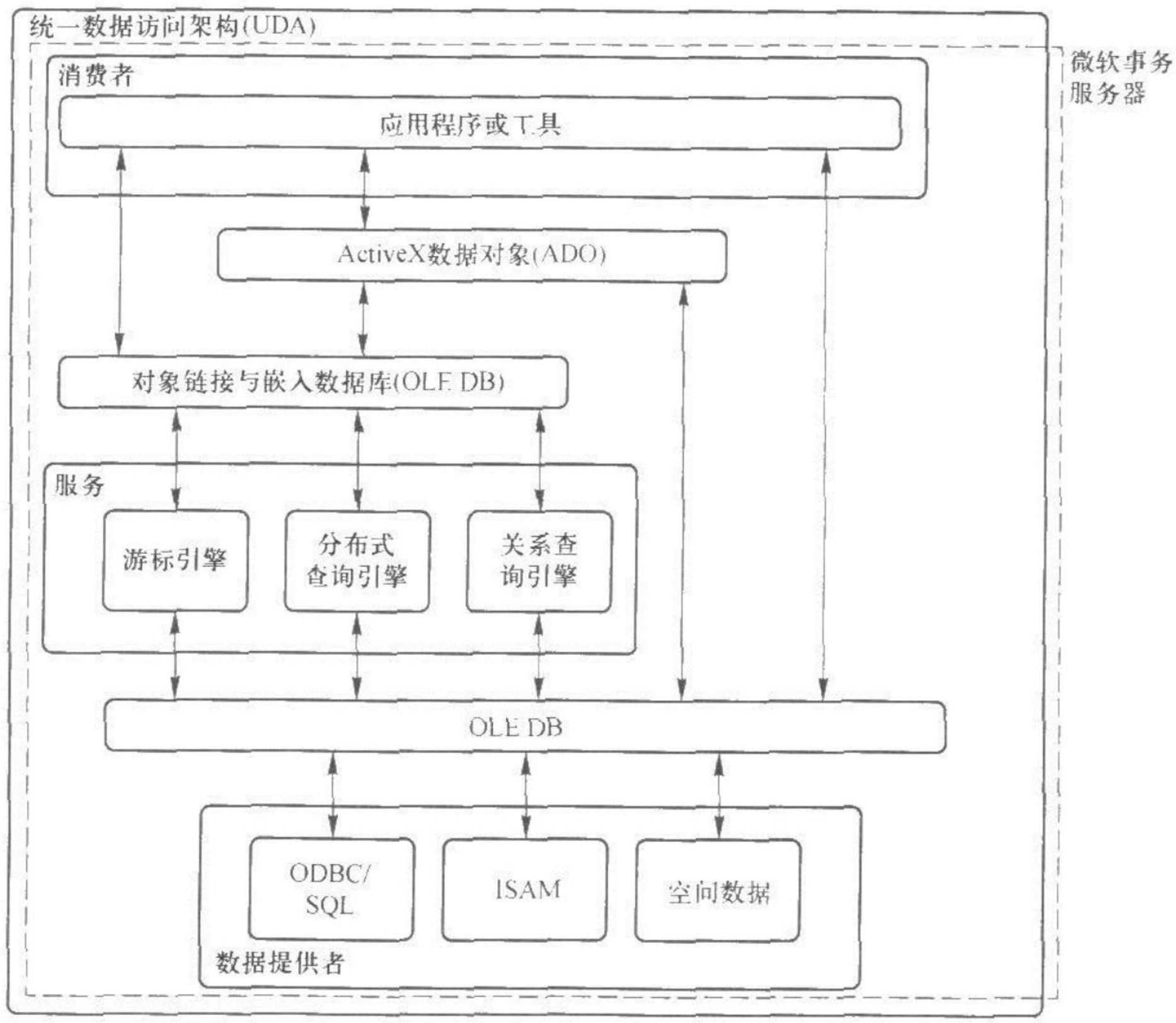


图 8-6　OLE DB 基本体系结构

（1）消费者

OLE DB 的消费者利用 OLE DB 提供者提供的接口访问数据源数据的客户端应用程序或其他工具。在 OLE DB 实现中，OLE DB 组件本身也可能作为消费者存在。

（2）提供者

OLE DB 的提供者是一个由 COM 组件构成的数据访问中介，位于数据源和消费者应用程序之间，向消费者提供访问数据源数据的各种接口。提供者主要有服务提供者和数据提供者。

①服务提供者。这类提供者自身没有数据，它通过 OLE DB 接口封装服务，从下层获取数据并向上层提供数据，具有提供者和消费者双重身份。一个服务提供者还可以和其他服务提供者或组件组合定义新的服务组件。

②数据提供者。数据提供者自己拥有数据并通过接口形成表格形式的数据。它不依赖于其他服务类或数据类的提供者，直接向消费者提供数据。

3．编程模型

OLE DB 基于 COM 对象技术形成一个支持数据访问的通用编程模型：数据管理任务必须由消费者访问数据，由提供者发布（deliver）数据。OLE DB 编程模型有两种：RowSet 模型和 Binder 模型。

六、JDBC 编程

JDBC 是 Java 的开发者 Sun 制定的 Java 数据库连接技术的简称，为 DBMS 提供支持无缝连接应用的技术。JDBC 是 Java 实现数据库访问的应用程序编程接口。

8.2　课后习题详解

1．使用嵌入式 SQL 对学生-课程数据库中的表完成下述功能：

（1）查询某一门课程的信息。要查询的课程由用户在程序运行过程中指定，放在主变量中。

（2）查询选修某一门课程的选课信息，要查询的课程号由用户在程序运行过程中指定，放在主变量中，然后根据用户的要求修改其中某些记录的成绩字段。

答：略。

2．对学生-课程数据库编写存储过程，完成下述功能：

（1）统计离散数学的成绩分布情况，即按照各分数段统计人数。

（2）统计任意一门课的平均成绩。

（3）将学生选课成绩从百分制改为等级制（即 A、B、C、D、E）。

答：略。

3．使用 ODBC 编写应用程序来对异构数据库进行各种数据操作。

配置两个不同的数据源，编写程序连接两个不同关系数据库管理系统的数据源，对异构数据库进行操作。例如，将 KingbaseES 数据库的某个表中的数据转移到 SQL Server 数据库的表中。

答：略。

第 3 篇　系统篇

第 9 章　关系查询处理和查询优化

9.1　复习笔记

一、概述

本章介绍关系数据库的查询处理（query processing）和查询优化（query optimization）技术。查询优化一般可分为代数优化（也称为逻辑优化）和物理优化（也称为非代数优化）。代数优化是指关系代数表达式的优化，物理优化则是指通过存取路径和底层操作算法的选择进行的优化。

二、关系数据库系统的查询处理

查询处理是关系数据库管理系统执行查询语句的过程，其任务是把用户提交给关系数据库管理系统的查询语句转换为高效的查询执行计划。

1．查询处理步骤

关系数据库管理系统查询处理可以分为 4 个阶段：查询分析、查询检查、查询优化和查询执行，如图 9-1 所示。

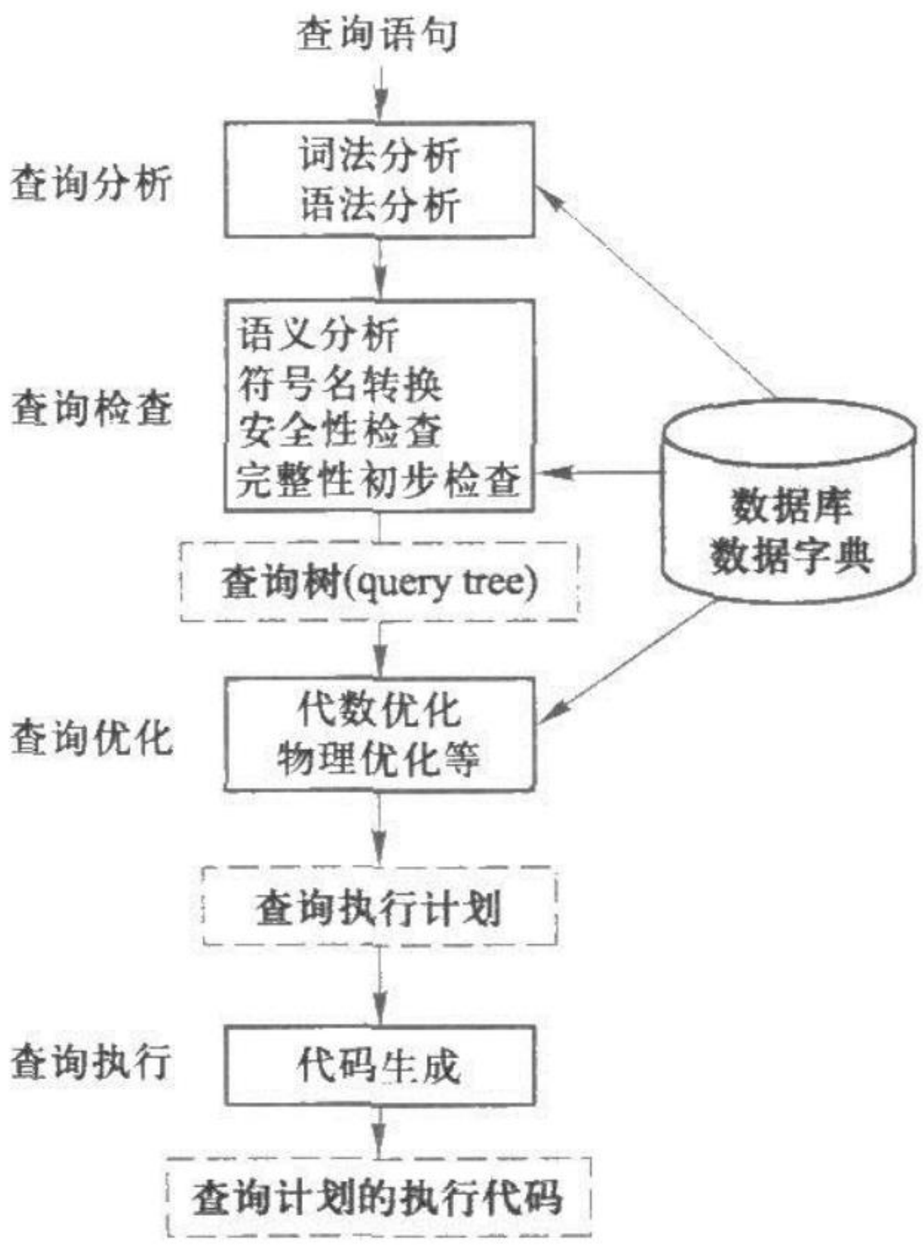


图 9-1　查询处理步骤

（1）查询分析

首先对查询语句进行扫描、词法分析和语法分析。从查询语句中识别出语言符号，如 SQL 关键字、属性名和关系名等，进行语法检查和语法分析，即判断查询语句是否符合 SQL 语法规则。

（2）查询检查

根据数据字典中有关的模式定义检查语句中的数据库对象，如关系名、属性名是否存在和有效。如果是对视图的操作，则要用视图消解方法把对视图的操作转换成对基本表的操作。还要根据数据字典中的用户权限和完整性约束。对用户的存取权限进行检查时，如果该用户没有相应的访问权限或违反了完整性约束，就拒绝执行该查询。检查通过后便把 SQL 查询语句转换成内部表示，即等价的关系代数表达式。这个过程中要把数据库对象的外部名称转换为内部表示。关系数据库管理系统一般都用查询树（query tree），也称为语法分析树（syntax tree）来表示扩展的关系代数表达式。

（3）查询优化

查询优化就是选择一个高效执行的查询处理策略。查询优化有多种方法。按照优化的层次一般可将查询优化分为代数优化和物理优化。

①代数优化

代数优化是指关系代数表达式的优化，即按照一定的规则，通过对关系代数表达式进行等价变换，改变代数表达式中操作的次序和组合，使查询执行更高效。

②物理优化

物理优化则是指存取路径和底层操作算法的选择。

（4）查询执行

依据优化器得到的执行策略生成查询执行计划，由代码生成器（code generator）生成执行这个查询计划的代码。

2．实现查询操作的算法

（1）选择操作的实现

①简单的全表扫描方法

对查询的基本表顺序扫描，逐一检查每个元组是否满足选择条件，把满足条件的元组作为结果输出。对于小表，这种方法简单有效。对于大表顺序扫描十分费时，效率很低。

②索引（或散列）扫描方法

如果选择条件中的属性上有索引(例如 B+树索引或 Hash 索引)，可以用索引扫描方法。通过索引先找到满足条件的元组主码或元组指针，再通过元组指针直接在查询的基本表中找到元组。

（2）连接操作的实现

连接操作是查询处理中最耗时的操作之一。等值连接（或自然连接）最常用的实现算法有：

①嵌套循环方法(nested loop)

这是最简单可行的算法。

②排序-合并方法

适合连接的诸表已经排好序的情况。

③索引连接(index join)方法

按索引进行连接。

④Hash Join 方法

把连接属性作为 hash 码，用同一个 hash 函数把 R 和 S 中的元组散列到同一个 hash 文件中。

三、关系数据库系统的查询优化

关系系统的查询优化既是 RDBMS 实现的关键技术又是关系系统的优点所在。

1．查询优化的优点

（1）它减轻了用户选择存取路径的负担。用户不必考虑如何最好地表达查询以获得较好的效率。

（2）系统可以比用户程序的“优化”做得更好。

2．查询优化的特点

（1）优化器可以从数据字典中获取许多统计信息，根据这些信息做出正确的估算，选择高效的执行计划，而用户程序则难以获得这些信息。

（2）如果数据库的物理统计信息改变了，系统可以自动对查询进行重新优化以选择相适应的执行计划。在非关系系统中必须重写程序，而重写程序在实际应用中往往是不太可能的。

（3）优化器可以考虑数百种不同的执行计划，而程序员一般只能考虑有限的几种可能性。

（4）优化器中包括了很多复杂的优化技术，这些优化技术往往只有最好的程序员才能掌握。系统的自动优化相当于使得所有人都拥有这些优化技术。

3．目标

查询优化的总目标是：选择有效的策略，求得给定关系表达式的值，使得查询代价最小。

四、代数优化

1．关系代数表达式等价变换规则

（1）连接、笛卡儿积的交换律

设 E_1 和 E_2 是关系代数表达式，F 是连接运算的条件，则有

$$E_1 \times E_2 \equiv E_2 \times E_1$$
$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$$
$$E_1 \underset{F}{\bowtie} E_2 \equiv E_2 \underset{F}{\bowtie} E_1$$

（2）连接、笛卡儿积的结合律

设 E_1、E_2、E_3是关系代数表达式，F_1和 F_2是连接运算的条件，则有

$$(E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3)$$
$$(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$$
$$(E_1 \underset{F_1}{\bowtie} E_2) \underset{F_2}{\bowtie} E_3 \equiv E_1 \underset{F_1}{\bowtie} (E_2 \underset{F_2}{\bowtie} E_3)$$

（3）投影的串接定律

$$\Pi_{A_1,A_2,\cdots,A_n}(\Pi_{B_1,B_2,\cdots,B_m}(E)) \equiv \Pi_{A_1,A_2,\cdots,A_n}(E)$$

这里，E 是关系代数表达式，A_i（i=1，2，…，n），B_i（i=1，2，…，m）是属性名，且{A_1， A_2，…，A_n}构成{B_1，B_2，…，B_m}的子集。

（4）选择的串接定律

$$\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_1 \wedge F_2}(E)$$

这里，E 是关系代数表达式，F_1、F_2是选择条件。选择的串接律说明选择条件可以合并，这样一次就可检查全部条件。

（5）选择与投影操作的交换律

$$\sigma_F(\Pi_{A_1,A_2,\cdots,A_n}(E)) \equiv \Pi_{A_1,A_2,\cdots,A_n}(\sigma_F(E))$$

这里，选择条件 F 只涉及属性 A_1，…，A_n。

若 F 中有不属于 A_1，…，A_n的属性 B_1，…，B_m，则有更一般的规则：

$$\Pi_{A_1,A_2,\cdots,A_n}(\sigma_F(E)) \equiv \Pi_{A_1,A_2,\cdots,A_n}(\sigma_F(\Pi_{A_1,A_2,\cdots,A_n,B_1,B_2,\cdots,B_m}(E)))$$

（6）选择与笛卡儿积的交换律

如果 F 中涉及的属性都是 E_1中的属性，则

$$\sigma_F(E_1 \times E_2) \equiv \sigma_F(E_1) \times E_2$$

如果 $F=F_1 \wedge F_2$，并且 F_1只涉及 E_1中的属性，F_2只涉及 E_2中的属性，则由上面的等价变换规则可推出：

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F_1}(E_1) \times \sigma_{F_2}(E_2)$$

若 F_1只涉及 E_1中的属性，F_2涉及 E_1和 E_2两者的属性，则仍有

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F_2}(\sigma_{F_1}(E_1) \times E_2)$$

它使部分选择在笛卡儿积前先做。

（7）选择与并的分配律

设 $E=E_1 \cup E_2$，E_1、E_2有相同的属性名，则

$$\sigma_F(E_1 \cup E_2) \equiv \sigma_F(E_1) \cup \sigma_F(E_2)$$

（8）选择与差运算的分配律

若 E_1与 E_2有相同的属性名，则

$$\sigma_F(E_1 - E_2) \equiv \sigma_F(E_1) - \sigma_F(E_2)$$

（9）选择对自然连接的分配律

$$\sigma_F(E_1 \bowtie E_2) \equiv \sigma_F(E_1) \bowtie \sigma_F(E_2)$$

F 只涉及 E_1与 E_2的公共属性。

（10）投影与笛卡儿积的分配律

设 E_1 和 E_2 是两个关系表达式，A_1，…，A_n 是 E_1 的属性，B_1，…，B_m 是 E_2 的属性，则

$$\Pi_{A_1,A_2,\cdots,A_n,B_1,B_2,\cdots,B_m}(E_1 \times E_2) \equiv \Pi_{A_1,A_2,\cdots,A_n}(E_1) \times \Pi_{B_1,B_2,\cdots,B_m}(E_2)$$

（11）投影与并的分配律

设 E_1 和 E_2 有相同的属性名，则

$$\Pi_{A_1,A_2,\cdots,A_n}(E_1 \cup E_2) \equiv \Pi_{A_1,A_2,\cdots,A_n}(E_1) \cup \Pi_{A_1,A_2,\cdots,A_n}(E_2)$$

2．查询树的启发式优化

典型的启发式规则有：

（1）选择运算应尽可能先做。

（2）把投影运算和选择运算同时进行。

（3）把投影同其前或其后的双目运算结合起来。

（4）把某些选择同在它前面要执行的笛卡尔积结合起来成为一个连接运算。

（5）找出公共子表达式。

五、物理优化

物理优化是要选择高效合理的操作算法或存取路径，求得优化的查询计划，达到查询优化的目标。选择的方法可以是：基于规则的启发式优化；基于代价估算的优化；两者结合的优化方法。

1．基于启发式规则的存取路径选择优化

（1）选择操作的启发式规则

①对于小关系，使用全表顺序扫描，即使选择列上有索引。

②对于大关系：

a．对于选择条件是主码=值的查询，查询结果最多是一个元组，可以选择主码索引；

b．对于选择条件是非主属性＝值的查询，并且选择列上有索引，则要估算查询结果的元组数目，比例较小(<10％)可以使用索引扫描方法，否则使用全表顺序扫描；

c．对于选择条件是属性上的非等值查询或者范围查询，并且选择列上有索引，同样要估算查询结果的元组数目，比例较小(<10％)可以使用索引扫描方法，否则使用全表顺序扫描；

d．对于用 AND 连接的合取选择条件，如果涉及这些属性的组合索引，则优先采用组合索引扫描方法；如某些属性上有一般的索引，使用索引扫描方法，否则使用全表顺序扫描；

e．对于用 OR 连接的析取选择条件，一般使用全表顺序扫描。

（2）连接操作的启发式规则

①如果两个表都已经按照连接属性排序，则选用排序-合并方法。

②如果一个表在连接属性上有索引，则可以选用索引连接方法。

③如果上面两个规则都不适用，其中一个表较小，则可以选用 Hash Join 方法。

④最后可以选用嵌套循环方法，并选择其中较小的表，即占用的块数较少的表作为外表。

2．基于代价估算的优化

在编译执行的系统中，一次编译优化，多次执行，查询优化和查询执行是分开的，适合采用精细复杂的基于代价的优化方法。

（1）统计信息

基于代价的优化方法要计算各种操作算法的执行代价，它与数据库的状态密切相关。为此，在数据字典中存储了优化器需要的统计信息(database statistics)，主要包括如下几个方面：

①对每个基本表，该表的元组总数(N)、元组长度(I)、占用的块数(B)、占用的溢出块数(BO)；

②对基表的每个列，该列不同值的个数、选择率、该列最大值、最小值，该列上是否已经建立了索引，是哪种索引（B+树索引、Hash 索引、聚集索引）；

③对索引，例如 B+树索引，该索引的层数(L)、不同索引值的个数、索引的选择基数、索引的叶结点数(Y)。

（2）代价估算

①全表扫描算法的代价估算公式。如果基本表大小为 B 块，全表扫描算法的代价 cost=B；如果选择条件是码=值，那么平均搜索代价 cost=B/2。

②索引扫描算法的代价估算公式

a．如果选择条件是码=值，则采用该表的主索引，若为 B+树，层数为 L，所以 cost=L+1；

b．如果选择条件涉及非码属性，若为 B+树索引，选择条件是相等比较，S 是索引的选择基数，所以(最坏的情况)cost=L+S；

c．如果比较条件是>，>=，<，<=操作，假设有一半的元组满足条件，那么就要存取一半的叶结点，并通过索引访问一半的表存储块。所以 cost=L+Y/2+B/2。如果可以获得更准确的选择基数，可以进一步修正 Y/2 与 B/2。

③嵌套循环连接算法的代价估算公式

把连接结果写回磁盘，则 cost=Br+Br Bs/(K-1)+($Frs^n r=Nr^n s$)/Mrs。

④排序一合并连接算法的代价估算公式

a．如果连接表已经按照连接属性排好序，则 cost=Br 十 Bs+($Frs^n rs^n s$)/Mrs；

b．如果必须对文件排序，那么还需要在代价函数中加上排序的代价。对于包含 B 个块的文件排序的代价大约是(2*B)+(2*l*B*$l0g^2B$)。

六、查询计划的执行

查询优化完成后，关系数据库管理系统为用户查询生成了一个查询计划。该查询计划的执行可以分为自顶向下和自底向上两种执行方法。

1．自顶向下

在自顶向下的执行方式中，系统反复向查询计划顶端的操作符发出需要查询结果元组的请求，操作符收到请求后，就试图计算下一个（几个）元组并返回这些元组。

2．自底向上

在自底向上的执行方式中，查询计划从叶结点开始执行，叶结点操作符不断地产生元组并将它们放入其输出缓冲区中，直到缓冲区填满为止，这时它必须等待其父操作符将元组从该缓冲区中取走才能继续执行。

9.2　课后习题详解

1．试述查询优化在关系数据库系统中的重要性和可能性。

答：（1）查询优化在关系数据库系统中的重要性：

关系系统的查询优化既是 RDBMS 实现的关键技术，又是关系系统的优点所在。它减轻了用户选择存取路径的负担。用户只要提出“干什么”，不必考虑如何最好地表达查询以获取较好的效率，而且系统可以比用户程序的“优化”做得更好。

（2）查询优化在关系数据库系统中的可能性：

①优化器可以从数据字典中获取许多统计信息，例如关系中的元组数、关系中每个属性的分布情况、这些属性上是否有索引（B+树索引、HASH 索引、唯一索引或组合索引）等。优化器可以根据这些信息选择有效的执行计划，而用户程序则难以获得这些信息。

②如果数据库的物理统计信息改变了，系统可以自动对查询进行重新优化以选择相适应的执行计划。在非关系系统中必须重写程序，而重写程序在实际应用中往往是不太可能的。

③优化器可以考虑数十甚至数百种不同的执行计划，从中选出较优的一个，而程序员一般只能考虑有限的几种可能性。

④优化器中包括了很多复杂的技术，这些优化技术往往只有最好的程序员才能掌握。系统的自动优化相当于使得所有人都拥有这些优化技术。

2．假设关系 R（A，B）和 S（B，C，D）情况如下：R 有 20000 个元组，S 有 1200 个元组，一个块能装 40 个 R 的元组，能装 30 个 S 的元组，估算下列操作需要多少次磁盘块读写。

（1）R 上没有索引，select*from R：

（2）R 中 A 为主码，A 有 3 层 B+树索引，select*from R where A=10；

（3）嵌套循环连接$R \bowtie S$；

（4）排序合并连接$R \bowtie S$，区分 R 与 S 在 B 属性上已经有序和无序两种情况。

答：略。

3．对学生课程数据库，查询信息系学生选修了的所有课程名称。

SELECT Cname

FROM Student, Course, SC

WHERE Student.Sno=SC.Sno AND SC.Cno=Course.Cno AND Student.Sdept='IS'；

试画出用关系代数表示的语法树，并用关系代数表达式优化算法对原始的语法树进行优化处理，画出优化后的标准语法树。

答：（1）学生-课程数据库用关系代数表示的语法树如图 9-2 所示：

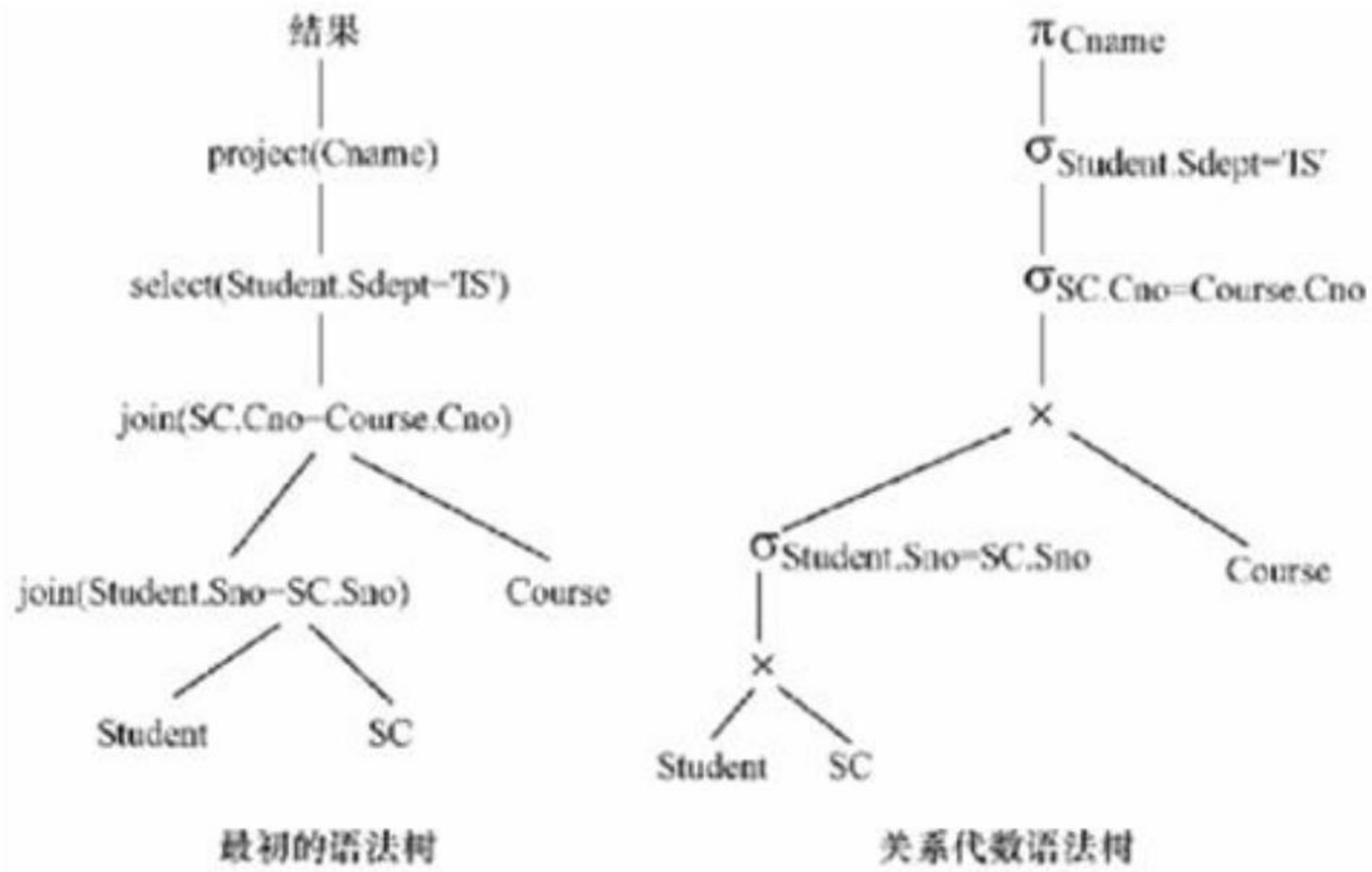


图 9-2　关系代数语法树

（2）优化后的标准语法树如图 9-3 所示：

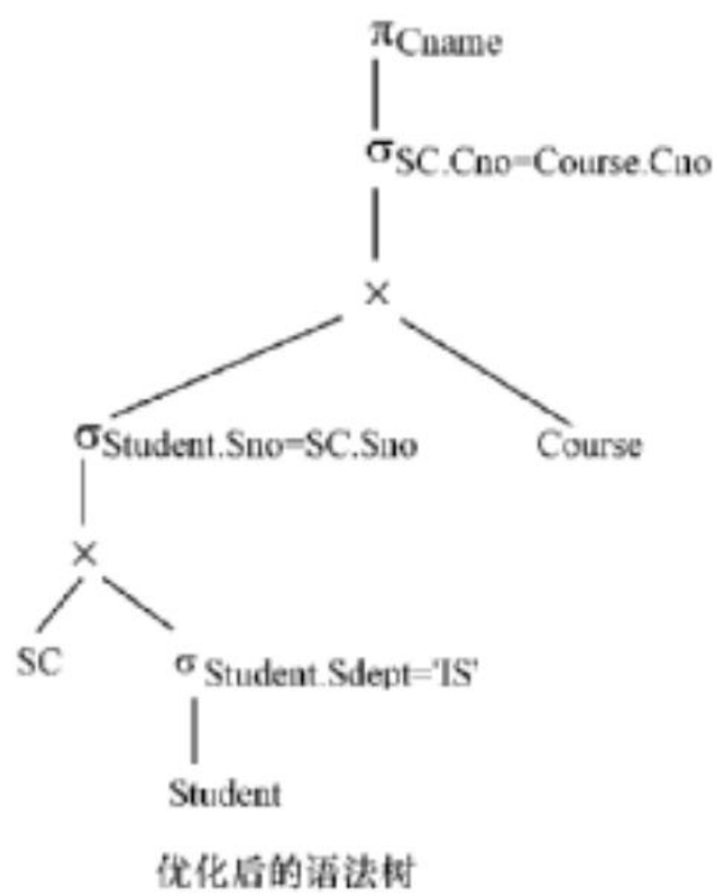


图 9-3　优化后的语法树

4．对于下面的数据库模式

Teacher（Tno，Tname，Tage，Tsex）；Department（Dno，Dname，Trio）；Work（Tno,Dn0，Year，Salary）

假设 Teacher 的 Tno 属性、Department 的 Dno 属性以及 Work 的 Year 属性上有 B+树索引，说明下列查询语句的一种较优的处理方法。

（1）select * from teacher where Tsex = '女'

（2）select * from department where Dno < 301

（3）select * from work where Year <> 2000

（4）select * from work where year > 2000 and salary < 5000

（5）select * from work where year < 2000 or salary < 5000

答：略。

5．对于题 4 中的数据库模式，有如下的查询：

select Tname

from teacher, department, work

where teacher.tno = work.tno and department.dno = work.dno and

department.dname = ‘计算机系’ and salary > 5000

画出语法树以及用关系代数表示的语法树，并对关系代数语法树进行优化，画出优化后的语法树。

答：略。

6．试述关系数据库管理系统查询优化的一般准则。

答：下面的优化策略一般能提高查询效率：

（1）选择运算应尽可能先做。

（2）投影运算和选择运算同时进行。

（3）投影同其前或其后的双目运算结合起来。

（4）某些选择同在它前面要执行的笛卡尔积结合起来成为一个连接运算。

（5）找出公共子表达式。

（6）选取合适的连接算法。

7．试述关系数据库管理系统查询优化的一般步骤。

答：各个关系系统的优化方法不尽相同，大致的步骤可以归纳如下：

（1）把查询转换成某种内部表示，通常用的内部表示是语法树。

（2）把语法树转换成标准（优化）形式，即利用优化算法，把原始的语法树转换成优化的形式。

（3）选择低层的存取路径。

（4）生成查询计划，选择代价最小的。

第 10 章　数据库恢复技术

10.1　复习笔记

一、事务的基本概念

1．事务

事务是用户定义的一个数据库操作序列，这些操作要么全做，要么全不做，是一个不可分割的工作单位。

事务和程序是两个概念。一般地讲，一个程序中包含多个事务。

事务的开始与结束可以由用户显式控制。如果用户没有显式地定义事务，则由数据库管理系统按默认规定自动划分事务。在 SQL 中，定义事务的语句一般有三条：

```
BEGIN TRANSACTION;
COMMIT;
ROLLBACK;
```

2．事务的 ACID 特性

（1）原子性

事务是数据库的逻辑工作单位，事务中包括的诸操作要么都做，要么都不做。

（2）一致性

事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态。

（3）隔离性

一个事务的执行不能被其他事务干扰。即一个事务的内部操作及使用的数据对其他并发事务是隔离的，并发执行的各个事务之间不能互相干扰。

（4）持续性

持续性也称永久性（Permanence），指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其执行结果有任何影响。

3．影响 ACID 特性的因素

（1）多个事务并行运行时，不同事务的操作交叉执行。

（2）事务在运行过程中被强行停止。

二、数据库恢复概述

计算机系统中硬件的故障、软件的错误、操作员的失误以及恶意的破坏仍是不可避免的，这些故障轻则造成运行事务非正常中断，影响数据库中数据的正确性，重则破坏数据库，使数据库中全部或部分数据丢失。因此数据库管理系统必须具有把数据库从错误状态恢复到某一已知的正确状态（亦称为一致状态或完整状态）的功能，这就是数据库的恢复。

三、故障的种类

1．事务内部的故障

事务内部的故障有的是可以通过事务程序本身发现的，有的是非预期的，不能由事务程序处理。

事务故障意味着事务没有达到预期的终点（COMMIT 或者显式的 ROLLBACK），因此，数据库可能处于不正确状态。恢复程序要在不影响其他事务运行的情况下，强行回滚该事务，即撤销该事务已经作出的任何对数据库的修改，使得该事务好像根本没有启动一样。这类恢复操作称为事务撤销（UNDO）。

2．系统故障

系统故障是指造成系统停止运转的任何事件，使得系统要重新启动。发生系统故障时，一些尚未完成的事务的结果可能已送入物理数据库，从而造成数据库可能处于不正确的状态。为保证数据一致性，需要清除这些事务对数据库的所有修改。

3．介质故障

系统故障常称为软故障（soft crash），介质故障称为硬故障（hard crash）。硬故障指外存故障，如磁盘损坏、磁头碰撞，瞬时强磁场干扰等。这类故障将破坏数据库或部分数据库，并影响正在存取这部分数据的所有事务。这类故障比前两类故障发生的可能性小得多，但破坏性最大。

4．计算机病毒

计算机病毒是一种人为的故障或破坏，是一些恶作剧者研制的一种计算机程序。这种程序与其他程序不同，它像微生物学所称的病毒一样可以繁殖和传播，并造成对计算机系统包括数据库的危害。

四、恢复的实现技术

1．概述

恢复机制涉及的两个关键问题是：

（1）如何建立冗余数据；

（2）如何利用这些冗余数据实施数据库恢复。

2．数据转储

（1）定义

转储即数据库管理员定期地将整个数据库复制到磁带、磁盘或其他存储介质上保存起来的过程。这些备用的数据称为后备副本（backup）或后援副本。

（2）方法

当数据库遭到破坏后可以将后备副本重新装入，但重装后备副本只能将数据库恢复到转储时的状态，要想恢复到故障发生时的状态，必须重新运行自转储以后的所有更新事务。

转储是十分耗费时间和资源的，不能频繁进行。数据库管理员应该根据数据库使用情况确定一个适当的转储周期。

（3）分类

转储按状态可分为静态转储和动态转储，按方式可分为海量转储和增量转储，数据分类方式如表 10-1 所示。

表 10-1　数据转储分类

转储方式	转储状态	
	动态转储	静态转储
海量转储	动态海量转储	静态海量转储
增量转储	动态增量转储	静态增量转储

①静态转储

静态转储是在系统中无运行事务时进行的转储操作。静态转储简单，通过静态转储得到的一定是一个数据一致性的副本。但转储必须等待正运行的用户事务结束才能进行，新的事务必须等待转储结束才能执行，这会降低数据库的可用性。

②动态转储

动态转储是指转储期间允许对数据库进行存取或修改，即转储和用户事务可以并发执行。动态转储可以克服静态转储的缺点，它不用等待正在运行的用户事务结束，也不会影响新事务的运行。但转储结束时后援副本上的数据并不能保证正确有效。必须把转储期间各事务对数据库的修改活动登记下来，建立日志文件(log file)。

③海量转储

海量转储是指每次转储全部数据库。

④增量转储

增量转储指每次只转储上一次转储后更新过的数据。

从恢复角度看，使用海量转储得到的后备副本进行恢复更方便些；但如果数据库很大，事务处理又十分频繁，而增量转储方式更实用更有效。

3．登记日志文件

（1）日志文件的格式和内容

日志文件是用来记录事务对数据库的更新操作的文件。日志文件主要有以下两种格式：

①以记录为单位的日志文件

对于以记录为单位的日志文件，日志文件中需要登记的内容包括：各个事务的开始标记；各个事务的结束标记；各个事务的所有更新操作。

每个日志记录的内容主要包括：事务标识；操作的类型；操作对象；更新前数据的旧值；更新后数据的新值。

②以数据块为单位的日志文件

日志记录的内容包括事务标识和被更新的数据块。

（2）日志文件的作用

①事务故障恢复和系统故障恢复必须用日志文件。

②在动态转储方式中必须建立日志文件，后备副本和日志文件结合起来才能有效地恢复发生故障时的数据

库。

③在静态转储方式中，也可以建立日志文件。当数据库毁坏后可重新装入后援副本把数据库恢复到转储结束时刻的正确状态，然后利用日志文件，把已完成的事务进行重做处理，对故障发生时尚未完成的事务进行撤销处理。这样不必重新运行那些已完成的事务程序就可把数据库恢复到故障前某一时刻的正确状态。

（3）登记日志文件

为保证数据库是可恢复的，登记日志文件时必须遵循两条原则：

①登记的次序严格按并发事务执行的时间次序。

②必须先写日志文件，后写数据库。

五、恢复策略

1．事务故障的恢复

事务故障是指事务在运行至正常终止点前被终止，这时恢复子系统应利用日志文件撤销（UNDO）此事务已对数据库进行的修改。系统的恢复步骤是：

（1）反向扫描日志文件（即从最后向前扫描日志文件），查找该事务的更新操作；

（2）对该事务的更新操作执行逆操作，即将日志记录中“更新前的值”写入数据库；

（3）继续反向扫描日志文件，查找该事务的其他更新操作，并做同样处理；

（4）如此处理下去，直至读到此事务的开始标记，事务故障恢复就完成了。

2．系统故障的恢复

系统故障的恢复是由系统在重新启动时自动完成的，不需要用户干预。系统的恢复步骤是：

（1）正向扫描日志文件（即从头扫描日志文件），找出在故障发生前已经提交的事务，将其事务标识记入重做队列（RED0-LIST）。同时找出故障发生时尚未完成的事务，将其事务标识记入撤销队列（UNDO．LIST）；

（2）对撤销队列中的各个事务进行撤销（UNDO）处理；

（3）对重做队列中的各个事务进行重做处理。

3．介质故障的恢复

发生介质故障后，磁盘上的物理数据和日志文件被破坏，恢复方法是重装数据库，然后重做已完成的事务。

（1）装入最新的数据库后备副本（离故障发生时刻最近的转储副本），使数据库恢复到最近一次转储时的一致性状态；

（2）装入相应的日志文件副本（转储结束时刻的日志文件副本），重做已完成的事务。

六、具有检查点的恢复技术

具有检查点的恢复技术在日志文件中增加一类新的记录——检查点(checkpoint)记录，增加一个重新开始文件，并让恢复子系统在登录日志文件期间动态地维护日志。

1．检查点记录的内容

（1）建立检查点时刻所有正在执行的事务清单；

（2）这些事务最近一个日志记录的地址。

重新开始文件用来记录各个检查点记录在日志文件中的地址。

2．动态维护日志文件的方法

动态维护日志文件的方法是，周期性地执行如下操作：建立检查点，保存数据库状态。具体步骤是：

（1）将当前日志缓冲区中的所有日志记录写入磁盘的日志文件上；

（2）在日志文件中写入一个检查点记录；

（3）将当前数据缓冲区的所有数据记录写入磁盘的数据库中；

（4）把检查点记录在日志文件中的地址写入一个重新开始文件。

恢复子系统可以定期或不定期地建立检查点保存数据库状态。检查点可以按照预定的一个时间间隔建立，也可以按照某种规则建立检查点。

3．恢复步骤

系统使用检查点方法进行恢复的步骤是：

（1）从重新开始文件中找到最后一个检查点记录在日志文件中的地址，由该地址在日志文件中找到最后一个检查点记录；

（2）由该检查点记录得到检查点建立时刻所有正在执行的事务清单；

（3）从检查点开始正向扫描日志文件；

（4）对 UNDO-LIST 中的每个事务执行 UNDO 操作，对 REDO-LIST 中的每个事务执行 REDO 操作。

七、数据库镜像

1．定义

数据库镜像是指根据 DBA 的要求，自动把整个数据库或其中的关键数据复制到另一个磁盘上。每当主数据库更新时，DBMS 自动把更新后的数据复制过去，即 DBMS 自动保证镜像数据与主数据库的一致性。

2．用处

（1）用于数据库恢复

当出现介质故障时，可由镜像磁盘继续提供使用，同时 DBMS 自动利用镜像磁盘数据进行数据库的恢复，不需要关闭系统和重装数据库副本。

（2）提高数据库的可用性

在没有出现故障时，当一个用户对某个数据加排它锁进行修改时，其他用户可以读镜像数据库上的数据，而不必等待该用户释放锁。

10.2　课后习题详解

1．试述事务的概念及事务的 4 个特性。恢复技术能保证事务的哪些特性?

答：（1）事务是用户定义的一个数据库操作序列，是一个不可分割的工作单位。事务具有 4 个特性：原子性、一致性、隔离性和持续性。这 4 个特性也简称为 ACID 特性。

①原子性：事务是数据库的逻辑工作单位，事务中的操作要么都做，要么都不做。

②一致性：事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态。当数据库只包含成功事务提交的结果时，就说数据库处于一致性状态。

③隔离性：一个事务的执行不能被其他事务干扰，即一个事务内部的操作及使用的数据对其他并发事务是隔离的，并发执行的各个事务之间不能互相干扰。

④持续性：持续性指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的，接下来的其他操作或故障不应该对其执行结果有任何影响。

（2）恢复技术保证了事务的原子性、一致性和持续性。

2．为什么事务非正常结束时会影响数据库数据的正确性?请举例说明之。

答：（1）事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态。如果数据库系统运行中发生故障，有些事务尚未完成就被迫中断，这些未完成事务对数据库所做的修改有一部分已写入物理数据库，这时数据库就处于一种不正确的状态，或者说是不一致的状态。

（2）例如某工厂的库存管理系统中，要把数量为 Q 的某种零件从仓库 1 移到仓库 2 存放。则可以定义一个事务 T,T 包括两个操作：Q1=Q1-Q,Q2=Q2＋Q。如果 T 非正常终止时只做了第一个操作，则数据库就处于不一致性状态，库存量无缘无故少了 Q。

3．登记日志文件时为什么必须先写日志文件，后写数据库?

答：把对数据的修改写到数据库中和把表示这个修改的日志记录写到日志文件中是两个不同的操作。有可能在这两个操作之间发生故障，即这两个写操作只完成了一个。

如果先写了数据库修改，而在运行记录中没有登记这个修改，则以后就无法恢复这个修改了。如果先写日志，但没有修改数据库，在恢复时只不过是多执行一次 UNDO 操作，并不会影响数据库的正确性。所以一定要先写日志文件，即首先把日志记录写到日志文件中，然后进行数据库的修改。

4．考虑下图所示的日志记录：

序号	日志
1	T_1：开始
2	T_1：写 A=10
3	T_2：开始
4	T_2：写 B，B=9

5	T_1：写 C，C=11
6	T_1：提交
7	T_2：写 C，C=13
8	T_3；开始
9	T_3：写 A，A=8
10	T_2：回滚
11	T_3：写 B，B=7
12	T_4：开始
13	T_3：提交
14	T_4：写 C，C=12

（1）如果系统故障发生在 14 之后，说明哪些事务需要重做，哪些事务需要回滚。

（2）如果系统故障发生在 10 之后，说明哪些事务需要重做，哪些事务需要回滚。

（3）如果系统故障发生在 9 之后，说明哪些事务需要重做，哪些事务需要回滚。

（4）如果系统故障发生在 7 之后，说明哪些事务需要重做，哪些事务需要回滚。

答：（1）需要重做的事务有：T_1 、T_2、T_3，需要回滚的事务有： T_4。

（2）需要重做的事务有：T_1 、T_2，需要回滚的事务有： T_3。

（3）需要重做的事务有：T_1，需要回滚的事务有：T_2、T_3。

（4）需要重做的事务有：T_1，需要回滚的事务有：T_2。

5．考虑题 4 所示的日志记录，假设开始时 A、B、C 的值都是 0：

（1）如果系统故障发生在 14 之后，写出系统恢复后 A、B、C 的值；

（2）如果系统故障发生在 12 之后，写出系统恢复后 A、B、C 的值：

（3）如果系统故障发生在 10 之后，写出系统恢复后 A、B、C 的值；

（4）如果系统故障发生在 9 之后，写出系统恢复后 A、B、C 的值；

（5）如果系统故障发生在 7 之后，写出系统恢复后 A、B、C 的值；

（6）如果系统故障发生在 5 之后，写出系统恢复后 A、B、C 的值。

答：（1）A=8，B=7，C=11。

（2）如果系统故障发生在 12 之后，需要重做的事务有：T_1 、T_2，需要回滚的事务有： T_3 、T_4。系统恢复后 A=8，B=9，C=11。

（3）A=10，B=9，C=11。

（4）A=10，B=9，C=13。

（5）A=10，B=9，C=11。

（6）如果系统故障发生在 5 之后，T_1 、T_2、T_3 、T_4均需要回滚。系统恢复后 A=10，B=9，C=0。

6．针对不同的故障，试给出恢复的策略和方法。（即如何进行事务故障的恢复，如何进行系统故障的恢复，以及如何进行介质故障的恢复。）

答：（1）事务故障的恢复是由 DBMS 执行的。恢复步骤是自动完成的，对用户是透明的。具体过程是：

①反向扫描文件日志（即从后向前扫描日志文件），查找该事务的更新操作；

②对该事务的更新操作执行逆操作，即将日志记录中“更新前的值”写入数据库；

③继续反向扫描日志文件，做同样处理；

④如此处理下去，直至读到此事务的开始标记，该事务故障的恢复就完成了。

（2）系统故障的恢复操作是指撤销（UNDO）故障发生时未完成的事务，重做（REDO）已完成的事务。系统的恢复步骤是：

①正向扫描日志文件，找出在故障发生前已经提交的事务队列（REDO 队列）和未完成的事务队列（UNDO 队列）；

②对撤销队列中的各个事务进行 UNDO 处理；

③对重做队列中的各个事务进行 REDO 处理。

（3）介质故障是最严重的一种故障。恢复方法是重装数据库，重做已完成的事务。具体过程是：

①DBA 装入最新的数据库后备副本（离故障发生时刻最近的转储副本）,使数据库恢复到转储时的一致性状

态；

②DBA 装入转储结束时刻的日志文件副本；

③DBA 启动系统恢复命令，由 DBMS 完成恢复功能，即重做已完成的事务。

7．什么是检查点记录?检查点记录包括哪些内容?

答：（1）检查点记录是一类新的日志记录。

（2）检查点记录的内容包括：

①建立检查点时刻所有正在执行的事务清单；

②这些事务的最近一个日志记录的地址。

8．具有检查点的恢复技术有什么优点?试举一个具体例子加以说明。

答：（1）在采用检查点技术之前，利用日志技术进行数据库的恢复时需要从头扫描日志文件，而利用检查点技术只需要从检查点所处时间点起开始扫描日志，这就缩短了扫描日志的时间，改善恢复效率。

（2）例如当事务 T 在一个检查点之前提交，T 对数据库所做的修改已经写入数据库，那么在进行恢复处理时，没有必要对事务 T 执行 REDO 操作。

9．试述使用检查点方法进行恢复的步骤。

答：系统使用检查点方法进行恢复的步骤是：

（1）从重新开始文件中找到最后一个检查点记录在日志文件中的地址，由该地址在日志文件中找到最后一个检查点记录。

（2）由该检查点记录得到检查点建立时刻所有正在执行的事务清单 ACTIVE-LIST。

这里需要建立以下两个事务队列：

①UNDO-LIST:需要执行 UNDO 操作的事务集合。

②REDO-LIST:需要执行 REDO 操作的事务集合。

把 ACTIVE-LIST 暂时放入 UNDO-LIST 队列，REDO 队列暂时为空。

（3）从检查点开始正向扫描日志文件。

①如果有新开始的事务，则将其暂时放入 UNDO-LIST 队列。

②如果有已提交的事务，则将其从 UNDO-LIST 队列移至 REDO-LIST 队列，直到日志文件结束。

（4）对 UNDO-LIST 中的每个事务执行 UNDO 操作，对 REDO-LIST 中的每个事务执行 REDO 操作。

10．什么是数据库镜像?它有什么用途?

答：（1）数据库镜像即根据 DBA 的要求，自动把整个数据库或者其中的部分关键数据复制到另一个磁盘上。每当主数据库更新时，DBMS 自动把更新后的数据复制过去，即 DBMS 自动保证镜像数据与主数据的一致性。

（2）数据库镜像的用途有两点：

①用于数据库恢复，当出现介质故障时，可由镜像磁盘继续提供使用，同时 DBMS 自动利用镜像磁盘数据进行数据库的恢复，不需要关闭系统和重装数据库副本。

②提高数据库的可用性，在没有出现故障时，当一个用户对某个数据加排它锁进行修改时，其他用户可以读镜像数据库上的数据，而不必等待该用户释放排它锁。

第 11 章　并发控制

11.1　复习笔记

一、并发控制概述

1．并发控制的责任

事务是并发控制的基本单位，保证事务 ACID 特性是事务处理的重要任务，而事务 ACID 特性可能遭到破坏的原因之一是多个事务对数据库的并发操作造成的。为了保证事务的隔离性和一致性，DBMS 需要对并发操作进行正确调度。这些就是数据库管理系统中并发控制机制的责任。

2．数据不一致及其原因

并发操作带来的数据不一致性主要包括丢失修改、不可重复读和读“脏”数据。产生三类数据不一致性的主要原因是并发操作破坏了事务的隔离性。并发控制就是要用正确的方式调度并发操作，使一个用户事务的执行不受其他事务的干扰，从而避免造成数据的不一致性。

并发控制的主要技术有封锁(Locking)、时间戳(Timestamp)和乐观控制法，商用的 DBMS 一般都采用封锁方法。

二、封锁

1．定义

封锁是事务 T 在对某个数据对象例如表、记录等操作之前，先向系统发出请求，对其加锁。加锁后事务 T 就对该数据对象有了一定的控制，在事务 T 释放它的锁之前，其他事务不能更新此数据对象。

2．分类

基本的封锁类型有两种：排他锁（exclusive locks，简称 X 锁）和共享锁（sharelocks，简称 S 锁）。

（1）排他锁

排他锁又称为写锁。若事务 T 对数据对象 A 加上 X 锁，则只允许 T 读取和修改 A，其他任何事务都不能再对 A 加任何类型的锁，直到 T 释放 A 上的锁为止。

（2）共享锁

共享锁又称为读锁。若事务 T 对数据对象 A 加上 S 锁，则事务 T 可以读 A 但不能修改 A，其他事务只能再对 A 加 S 锁，而不能加 X 锁，直到 T 释放 A 上的 S 锁为止。

排他锁与共享锁的控制方式可以用图 11-1 所示的相容矩阵（compatibility matrix）来表示。

T_1 \ T_2	X	S	–
X	N	N	Y
S	N	Y	Y
–	Y	Y	Y

Y=Yes，相容的请求

N=No，不相容的请求

图 11-1　封锁类型的相容矩阵

三、封锁协议

1．一级封锁协议

一级封锁协议是指事务 T 在修改数据 R 之前必须先对其加 X 锁，直到事务结束才释放。事务结束包括正常结束（COMMIT）和非正常结束（ROLLBACK）。一级封锁协议可防止丢失修改，并保证事务 T 是可恢复的。

2．二级封锁协议

二级封锁协议是指在一级封锁协议基础上增加事务 T 在读取数据 R 之前必须先对其加 S 锁，读完后即可释放 S 锁。二级封锁协议除防止了丢失修改，还可进一步防止读“脏”数据。

3．三级封锁协议

三级封锁协议是指在一级封锁协议的基础上增加事务 T 在读取数据 R 之前必须先对其加 S 锁，直到事务结束才释放。三级封锁协议除了防止丢失修改和读“脏”数据外，还进一步防止了不可重复读。

三级封锁协议可以总结为表 11-1。

表 11-1　不同级别的封锁协议和一致性保证

	X 锁		S 锁		一致性保证		
	操作结束释放	事务结束释放	操作结束释放	事务结束释放	不丢失修改	不读“脏”数据	可重复读
一级封锁协议		√			√		
二级封锁协议		√	√		√	√	
三级封锁协议		√		√	√	√	√

四、活锁和死锁

1．活锁

（1）定义

如果事务 T_1 封锁了数据 R，事务 T_2 又请求封锁 R，于是 T_2 等待。T_3 也请求封锁 R，当 T_1 释放了 R 上的封锁之后系统首先批准了 T_3 的请求，T_2 仍然等待。然后 T_4 又请求封锁 R，当 T_3 释放了 R 上的封锁之后系统又批准了 T_4 的请求……T_2 有可能永远等待，这就是活锁的情形，如图 11-2 所示。

T_1	T_2	T_3	T_4
lock R	⋮		
	Lock R	⋮	⋮
⋮	等待	Lock R	
	等待		Lock R
Unlock R	等待	⋮	等待
	等待	Lock R	等待
	等待	⋮	等待
⋮	等待	Unlock	
	等待		Lock R
	等待	⋮	⋮

(a) 活锁

T_1	T_2
Lock R_1	⋮
⋮	Lock R_2
Lock R_2	⋮
等待	
等待	
等待	Lock R_1
等待	等待
等待	等待
	⋮

(b) 死锁

图 11-2　死锁与活锁示例

（2）避免的方法

避免活锁的简单方法是采用先来先服务的策略。当多个事务请求封锁同一数据对象时，封锁子系统按请求封锁的先后次序对事务排队，数据对象上的锁一旦释放就批准申请队列中第一个事务获得锁。

2．死锁

（1）定义

如果事务 T_1 封锁了数据 R_1，T_2 封锁了数据 R_2，然后 T_1 又请求封锁 R_2，因 T_2 已封锁了 R_2，于是 T_1 等待 T_2 释放 R_2 上的锁；接着 T_2 又申请封锁 R_1，因 T_1 已封锁了 R_1，T_2 也只能等待 T_1 释放 R_1 上的锁。

（2）避免的方法

在数据库中解决死锁问题主要有两类方法，一类方法是采取一定措施来预防死锁的发生，另一类方法是允许发生死锁，采用一定手段定期诊断系统中有无死锁，若有则解除之。

①死锁的预防

在数据库中，产生死锁的原因是两个或多个事务都已封锁了一些数据对象，然后又都

请求对已被其他事务封锁的数据对象加锁，从而出现死等待。防止死锁的发生其实就是要破坏产生死锁的条件。预防死锁通常有以下两种方法。

a．一次封锁法

一次封锁法要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行。一次封锁法虽然可以有效地防止死锁的发生，但也存在问题。

第一，一次就将以后要用到的全部数据加锁，势必扩大了封锁的范围，从而降低了系统的并发度；

第二，数据库中数据是不断变化的，原来不要求封锁的数据在执行过程中可能会变成封锁对象，所以很难事先精确地确定每个事务所要封锁的数据对象，为此只能扩大封锁范围，将事务在执行过程中可能要封锁的数据对象全部加锁，这就进一步降低了并发度。

b．顺序封锁法

顺序封锁法是预先对数据对象规定一个封锁顺序，所有事务都按这个顺序实行封锁。顺序封锁法可以有效地防止死锁，但也同样存在问题。

第一，数据库系统中封锁的数据对象极多，并且随数据的插入、删除等操作而不断地变化，要维护这样的资源的封锁顺序非常困难，成本很高。

第二，事务的封锁请求可以随着事务的执行而动态地决定，很难事先确定每一个事务要封锁哪些对象，因此也就很难按规定的顺序去施加封锁。

②死锁的诊断与解除

数据库系统中诊断死锁的方法与操作系统类似，一般使用超时法或事务等待图法。

a．超时法

如果一个事务的等待时间超过了规定的时限，就认为发生了死锁。超时法实现简单，但其不足也很明显。

第一，有可能误判死锁，事务因为其他原因使等待时间超过时限，系统会误认为发生了死锁。

第二，时限若设置得太长，死锁发生后不能及时发现。

b．等待图法

事务等待图是一个有向图 G=(T，U)。T 为结点的集合，每个结点表示正运行的事务；U 为边的集合，每条边表示事务等待的情况。若 T_1 等待 T_2，则 T_1，T_2 之间划一条有向边，从 T_1 指向 T_2。如图 11-3 所示。

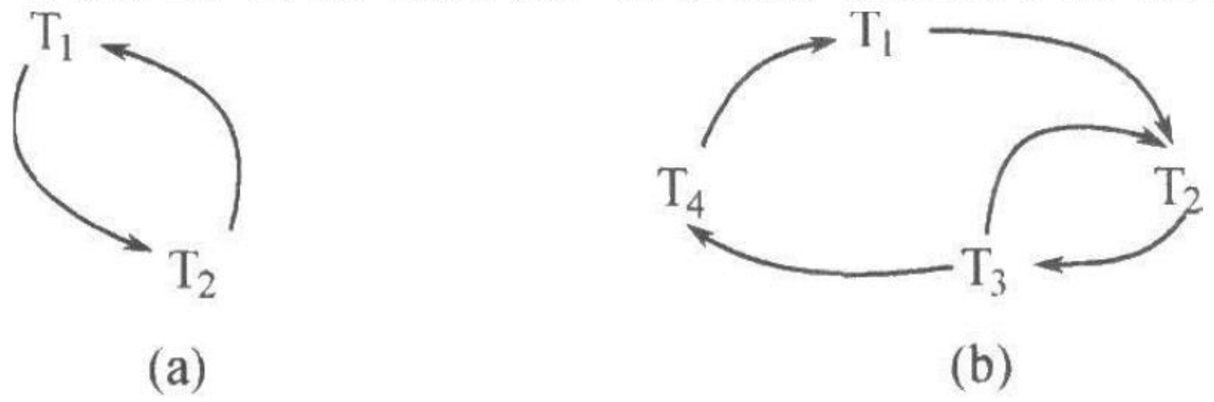


图 11-3　事务等待图

事务等待图动态地反映了所有事务的等待情况。并发控制子系统周期性地生成事务等待图，并进行检测。如果发现图中存在回路，则表示系统中出现了死锁。

五、并发调度的可串行性

1．可串行化调度

多个事务的并发执行是正确的，当且仅当其结果与按某一次序串行地执行这些事务时的结果相同，称这种调度策略为可串行化（serializable）调度。可串行性（serializability）是并发事务正确调度的准则。

2．冲突可串行化调度

（1）冲突操作

冲突操作是指不同的事务对同一个数据的读写操作和写写操作，其他操作是不冲突操作。不同事务的冲突操作和同一事务的两个操作是不能交换（swap）的。

（2）冲突可串行调度

一个调度 Sc 在保证冲突操作的次序不变的情况 F，通过交换两个事务不冲突操作的次序得到另一个调度 Sc′，如果 Sc′ 是串行的，称调度 Sc 为冲突可串行化的调度。若一个调度是冲突可串行化，则一定是可串行化的调度。

六、两段锁协议

为了保证并发调度的正确性，DBMS 的并发控制机制必须提供一定的手段来保证调度是可串行化的。目前 DBMS 普遍采用两段锁(Two Phase Locking，简称 2PL)协议的方法实现并发调度的可串行性，从而保证调度的正确性。

1．定义

两段锁协议是指所有事务必须分两个阶段对数据项加锁和解锁：在对任何数据进行读、写操作之前，首先要申请并获得对该数据的封锁；在释放一个封锁之后，事务不再申请和获得任何其他封锁。“两段”锁的含义是，事务分为两个阶段。第一阶段是获得封锁；第二阶段是释放封锁。

2．两段锁协议和一次封锁法的异同点

（1）相同点

一次封锁法要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行，因此一次封锁法遵守两段锁协议。

（2）不同点

两段锁协议并不要求事务必须一次将所有要使用的数据全部加锁，遵守两段锁协议的事务可能发生死锁。

七、封锁的粒度

1．封锁粒度

封锁对象的大小称为封锁粒度(Granularity)。封锁对象可以是逻辑单元，也可以是物理单元。

（1）封锁粒度与系统的关系

封锁粒度与系统的并发度和并发控制的开销密切相关。

①封锁的粒度越大，数据库所能够封锁的数据单元就越少，并发度就越小，系统开销也越小；

②封锁的粒度越小，并发度较高，但系统开销也就越大。

（2）粒度的选择

在一个系统中同时支持多种封锁粒度供不同的事务选择的封锁方法称为多粒度封锁。选择封锁粒度时应该同时考虑封锁开销和并发度两个因素，适当选择封锁粒度以求得最优的效果。

①需要处理大量元组的事务可以以关系为封锁粒度；

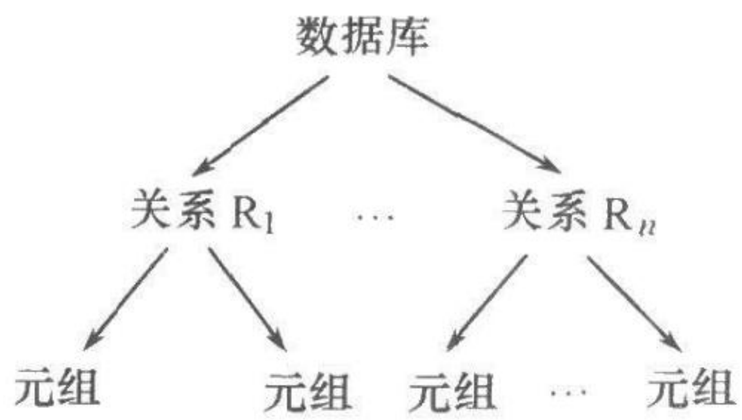
②需要处理多个关系的大量元组的事务可以以数据库为封锁粒度；

③对于处理少量元组的用户事务，以元组为封锁粒度比较合适。

2．多粒度封锁

（1）结构

多粒度树的根结点是整个数据库，表示最大的数据粒度。叶结点表示最小的数据粒度。如图 11-4 所示定义了一个三级多粒度树。



11-4　三级粒度树

（2）显式封锁和隐式封锁

多粒度封锁协议允许多粒度树中的每个结点被独立地加锁。对一个结点加锁意味着这个结点的所有后裔结点也被加以同样类型的锁。因此，在多粒度封锁中一个数据对象可能以两种方式封锁，显式封锁和隐式封锁。

①显式封锁

显式封锁是应事务的要求直接加到数据对象上的封锁。

②隐式封锁

隐式封锁是该数据对象没有独立加锁，是由于其上级结点加锁而使该数据对象加上了锁。

（3）检查封锁方法

系统检查封锁冲突时不仅要检查显式封锁还要检查隐式封锁。

①对某个数据对象加锁，系统要检查该数据对象上有无显式封锁与之冲突；

②检查其所有上级结点，看本事务的显式封锁是否与该数据对象上的隐式封锁冲突；

③检查其所有下级结点，看上面的显式封锁是否与本事务的隐式封锁冲突。

3．意向锁

如果对一个结点加意向锁，则说明该结点的下层结点正在被加锁；对一结点加锁时，必须先对它的上层结点加意向锁。三种常用的意向锁包括意向共享锁（Intent Share Lock，简称 IS 锁）；意向排它锁（Intent Exclusive Lock，简称 IX 锁）；共享意向排它锁（Share Intent Exclusive Lock，简称 SIX 锁）。

（1）IS 锁

如果对一个数据对象加 IS 锁，表示它的后裔结点拟（意向）加 S 锁。

（2）IX 锁

如果对一个数据对象加 IX 锁，表示它的后裔结点拟（意向）加 X 锁。

（3）SIX 锁

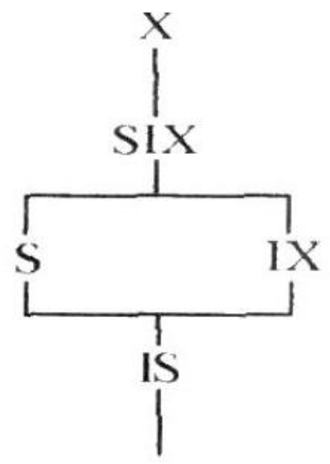
如果对一个数据对象加 SIX 锁，表示对它加 S 锁，再加 IX 锁，即 SIX=S+IX。

如图 11-5 所示给出了这些锁的相容矩阵，从中可以发现这 5 种锁的强度如图 11-5 所示的偏序关系。锁的强度是指它对其他锁的排斥程度。

T_1 \ T_2	S	X	IS	IX	SIX	-
S	Y	N	Y	N	N	Y
X	N	N	N	N	N	Y
IS	Y	N	Y	Y	Y	Y
IX	N	N	Y	Y	N	Y
SIX	N	N	Y	N	N	Y
-	Y	Y	Y	Y	Y	Y

Y=Yes，表示相容的请求　　N=No，表示不相容的请求

(a) 数据锁的相容矩阵



(b) 锁的强度的偏序关系

图 11-5　加上意向锁之后锁的相容矩阵与偏序关系

申请封锁时应该按自上而下的次序进行，释放封锁时则应该按自下而上的次序进行。具有意向锁的多粒度封锁方法提高了系统的并发度，减少了加锁和解锁的开销。

八、其他并发控制机制

1．时间戳方法

时间戳方法给每一个事务盖上一个时标，即事务开始执行的时间。每个事务具有唯一的时间戳，并按照这个时间戳来解决事务的冲突操作。如果发生冲突操作，就回滚具有较早时间戳的事务，以保证其他事务的正常执行，被回滚的事务被赋予新的时间戳并从头开始执行。

2．乐观控制法

乐观控制法认为事务执行时很少发生冲突，因此不对事务进行特殊的管制，而是让它自由执行，事务提交前再进行正确性检查。如果检查后发现该事务执行中出现过冲突并影响了可串行性，则拒绝提交并回滚该事务。乐观控制法又被称为验证方法（certifier）。

3．多版本控制

多版本并发控制（MultiVersion Concurrency Control，MVCC）是指在数据库中通过维护数据对象的多个版本信息来实现高效并发控制的种策略。

（1）多版本并发控制

版本（version）是指数据库中数据对象的一个快照，记录了数据对象某个时刻的状态。随着计算机系统存储设备价格的不断降低，可以考虑为数据库系统的数据对象保留多个版本，以提高系统的并发操作程度。

①多版本协议描述

假设版本 Q_k 具有小于或等于 TS（T）的最大时间戳。

a．若事务 T 发出 read（Q），则返回版本 Q_k 的内容。

b．若事务 T 发出 write（Q），则：当 TS（T）<R-timestamp（Q_k）1 对，回滚 T；当 TS（T1=w-timestamp（Q_k）时，覆盖 Q_k 的内容；否则，创建 Q 的新版本。

②优点

a．消除了数据库中数据对象读和写操作的冲突；

b．提高了系统的性能；

c．提高事务的并发度。

（2）改进的多版本并发控制

多版本协议可以进一步改进。区分事务的类型为只读事务和更新事务。对于只读事务，发生冲突的可能性很小，可以采用多版本时间戳。对于更新事务，采用较保守的两阶段封锁（2PL）协议。这样的混合协议称为 MV2PL。

11.2　课后习题详解

1．在数据库中为什么要并发控制？并发控制技术能保证事务的哪些特性？

答：（1）数据库是共享资源，可以供多个用户使用，所以通常有许多个事务同时在运行。当多个事务并发地存取数据库时就会产生同时读取或修改同一数据的情况。若对并发操作不加以控制就可能会存取和存储不正确的数据，破坏数据库的一致性。所以数据库管理系统必须提供并发控制机制。

（2）并发控制可以保证事务的一致性和隔离性，保证数据库的一致性。

2．并发操作可能会产生哪几类数据不一致？用什么方法能避免各种小致的情况？

答：（1）并发操作带来的数据不一致性包括三类：丢失修改、不可重复读和读“脏”数据。

①丢失修改。两个事务 T_1 和 T_2 读入同一数据并修改，T_2 提交的结果破坏了 T_1 提交的结果，导致 T_1 的修改被丢失。

②不可重复读。不可重复读是指事务 T_1 读取数据后，事务 T_2 执行更新操作，使 T_1 无法再现前一次读取结果。

③读“脏”数据。读“脏”数据是指事务 T_1 修改某一数据，并将其写回磁盘，事务 T_2 读取同一数据后，T_1 由于某种原因被撤销，这时 T_1 已修改过的数据恢复原值，T_2 读到的数据就与数据库中的数据不一致，则 T_2 读到的数据就为“脏”数据，即不正确的数据。

（2）产生上述三类数据不一致性的主要原因是并发操作破坏了事务的隔离性。并发控制就是要用正确的方式调度并发操作，使一个用户事务的执行不受其他事务的干扰，从而避免造成数据的不一致性。

3．什么是封锁？基本的封锁类型有几种？试述它们的含义。

答：（1）封锁是指事务 T 在对某个数据对象（例如表、记录等）进行操作之前，先向系统发出请求，对其加锁。加锁后，事务 T 就对该数据对象有控制权，在事务 T 释放锁之前，其他事务不能更新此数据对象。

（2）基本的封锁类型有两种：排它锁（简称 X 锁）和共享锁（简称 S 锁）。

①排它锁又称为写锁。若事务 T 对数据对象 A 加上 X 锁，则只允许 T 来读取和修改 A，其他任何事务都不能再对 A 加任何类型的锁，直到 T 释放 A 上的锁。这就保证了其他事务在 T 释放 A 上的锁之前不能再读取和修改 A。

②共享锁又称为读锁。若事务 T 对数据对象 A 加上 S 锁，则事务 T 可以读 A 但不能修改 A，其他事务只能再对 A 加 S 锁，而不能加 X 锁，直到事务 T 释放 A 上的 S 锁。这就保证了其他事务可以读 A，但在事务 T 释放 A 上的 S 锁之前不能对 A 做任何修改。

4．如何用封锁机制保证数据的一致性？

答：DBMS 在对数据进行读、写操作之前首先对该数据执行封锁操作，例如图 11-6 中事务 T_1 在对 A 进行修改之前先对 A 执行 Xlock A，即对 A 加 X 锁。这样，当 T_2 请求对 A 加 X 锁时就被拒绝，T_2 只能等待 T_1 释放 A 上的锁后才能获得对 A 的 X 锁，这时它读到的 A 是 T_1 更新后的值，再按此新的 A 值进行运算。这样就不会丢失 T_1 的更新。DBMS 按照一定的封锁协议，对并发操作进行控制，使得多个并发操作有序地执行，避免了丢失修改、不可重复读和读“脏”数据等数据不一致性。

T_1	T_2
① Xlock (A)	
获得控制权	
② 读 A=16	
	Xlock (A)
	等待
③ A←A-1	等待
写回 A=15	等待
Commit	等待
Unlock (A)	
④	获得 Xlock (A)
	读 A=15
	A←A-1
⑤	写回 A=14
	Commit
	Unlock (A)

图 11-6

5．什么是活锁？试述活锁的产生原因和解决方法。

答：（1）如果事务 T_1 封锁了数据 R，事务 T_2 又请求封锁 R，于是 T_2 等待。T_3 也请求封锁 R，当 T_1 释放了 R

上的封锁之后系统首先批准了 T_3 的请求，T_2 仍然等待。然后 T_4 请求封锁 R，当 T_3 释放了 R 上的封锁之后系统批准了 T_4 的请求……T_2 有可能永远等待，这就是活锁，如图 11-7 所示。

T_1	T_2	T_3	T_4
Lock R	⋮	⋮	⋮
⋮	Lock R		
	等待	Lock R	
Unlock	等待	⋮	Lock R
	等待	Lock R	等待
	等待	⋮	等待
⋮	等待	Unlock	等待
	等待	⋮	Lock R
	等待		⋮

图 11-7

（2）活锁产生的原因：当一系列封锁不能按照其先后顺序执行时，可能导致一些事务无限期地等待某个封锁，从而导致活锁。

（3）避免活锁的解决方法是采用先来先服务的策略。当多个事务请求封锁同一数据对象时，封锁子系统按请求封锁的先后次序对事务排队，数据对象上的锁一旦释放，就批准申请队列中第一个事务获得锁。

6．什么是死锁？请给出预防死锁的若干方法。

答：（1）如果事务 T_1 封锁了数据 R_1，T_2 封锁了数据 R_2，T_1 又请求封锁 R_2，因 T_2 已封锁了 R_2，于是 T_1 等待 T_2 释放 R_2 上的锁。T_2 又申请封锁 R_1，因 T_1 已封锁了 R_1，T_2 也只能等待 T_1 释放 R_1 上的锁。这样就出现了 T_1 在等待 T_2，而 T_2 又在等待 T_1 的局面，T_1 和 T_2 两个事务永远不能结束，形成死锁。

（2）防止死锁发生其实是要破坏产生死锁的条件。预防死锁通常可以有两种方法：

①一次封锁法

要求每个事务必须一次将所用的所有数据全部加锁，否则就不能执行。

②顺序封锁法

预先对数据对象规定一个封锁顺序，所有事务都按照这个顺序实行封锁。但是，预防死锁的策略不大适合数据库系统的特点。

7．请给出检测死锁发生的一种方法，当发生死锁后如何解除死锁？

答：（1）数据库系统一般采用允许死锁发生，DBMS 检测到死锁后加以解除的方法。DBMS 中诊断死锁的方法与操作系统类似，一般使用超时法或事务等待图法。

①超时法是如果一个事务的等待时间超过了规定的时限，就认为发生了死锁。超时法实现简单，但有可能误判死锁，事务因其他原因长时间等待超过时限时，系统会误认为发生了死锁。若时限设置得太长，又不能及时发现死锁。

②事务等待图是一个有向图 G=(T，U)。T 为结点的集合，每个结点表示正运行的事务；U 为边的集合，每条边表示事务等待的情况。若 T_1 等待 T_2，则 T_1，T_2 之间划一条有向边，从 T_1 指向 T_2。事务等待图动态地反映了所有事务的等待情况。并发控制子系统周期性地生成事务等待图，并进行检测。如果发现图中存在回路，则表示系统中出现了死锁。

（2）DBMS 并发控制子系统检测到死锁后，就要设法解除。通常采用的方法是选择一个处理死锁代价最小的事务，将其撤消，释放此事务持有的所有锁，使其他事务得以继续运行。对撤销的事务所执行的数据修改操作必须加以恢复。

8．什么样的并发调度是正确的调度？

答：可串行化的调度是正确的调度。

可串行化的调度是指多个事务的并发执行是正确的，当且仅当其结果与按某一次序串行执行它们时的结果相同的调度策略。

9．设 T_1、T_2、T_3 是如下的三个事务，设 A 的初值为 0。

T_1：A：=A+2：

T_2：A：=A*2；

T_3：A：=A**2；（即 $A \leftarrow A^2$）

（1）若这三个事务允许并发执行，则有多少种可能的正确结果？请一列举出来。

（2）请给出个可串行化的调度，并给出执行结果。

（3）请给出一个非串行化的调度，并给出执行结果。

（4）若这三个事务都遵守两段锁协议，请给出一个不产生死锁的可串行化调度。

（5）若这三个事务都遵守两段锁协议，请给出一个产生死锁的调度。

答：（1）A 的最终结果可能有 2、4、8、16，因为串行执行次序有 $T_1T_2T_3$、$T_1T_3T_2$、$T_2T_1T_3$、 $T_2T_3T_1$、$T_3T_1T_2$、$T_3T_2T_1$，对应的执行结果分别是 16、8、4、2、4、2。

（2）可串行化的调度为：

T1	T2	T3
slock A		
Y=A=0		
Unlock A		
Xlock A		
	Slock A	
A=Y+2	等待	
写回 A(=2)	等待	
Unlock A	等待	
	Y=A=2	
	Unlock A	
	Xlock A	
		Slock A
	A=Y*2	等待
	写回 A(=4)	等待
	Unlock A	等待
		Y=A=4
		Unlock A
		Xlock A
		写回 A(=16)
		Unlock A

最后结果 A 为 16，是可串行化的调度。

（3）非串行化的调度为：

T1	T2	T3
Slock A		
Y=A=0		
Unlock A		
	Slock A	
	Y=A=0	
Xlock A		
等待	Unlock A	
A=Y+2		
写回 A(=2)		Slock A
Unlock A		等待
		Y=A=2
		Unlock A
		Xlock A
	Xlock A	
	等待	A=Y**2
	等待	写回 A(=4)
	等待	Unlock A
	A=Y*2	
	写回 A(=0)	
	Unlock A	

最后结果 A 为 0，是非串行化的调度。

（4）若这 3 个事务都遵守两段锁协议，不产生死锁的可串行化调度为：

T_1	T_2	T_3
Slock A		
Y = A = 0		
Xlock A		
A = Y + 2	Slock A	
写回 A(= 2)	等待	
Unlock A	等待	
	Y = A = 2	
	Xlock A	
Unlock A	等待	Slock A
	A = Y * 2	等待
	写回 A(= 4)	等待
	Unlock A	等待
		Y = A = 4
	Unlock A	
		Xlock A
		A = Y * *2
		写回 A(= 16)
		Unlock A
		Unlock A

（5）若这 3 个事务都遵守两段锁协议，产生死锁的调度为：

T_1	T_2	T_3
Slock A		
Y = A = 0		
	Slock A	
	Y = A = 0	
Xlock A		
等待		
	Xlock A	
	等待	
		Slock A
		Y = A = 0
		Xlock A
		等待

10．今有三个事务的一个调度 r_3（B）r_1（A）W_3（B）r_2（B）r_2（A）W_2（B）r_1（B）W_1（A），该调度足冲突可串行化的调度吗？为什么？

答：该调度是冲突可串行化的调度。

Sc1=$r_3(B)r_1(A)w_3(B)r_2(B)r_2(A)w_2(B)r_1(B)w_1(A)$，

交换 $r_1(A)$和 $w_3(B)$得到 $r_3(B)w_3(B)r_1(A)r_2(B)r_2(A)w_2(B)r_1(B)w_1(A)$，

再交换 $r_1(A)$和 $r_2(B)r_2(A)w_2(B)$，得到：Sc2=$r_3(B)w_3(B)r_2(B)r_2(A)w_2(B)r_1(A)r_1(B)w_1(A)$，

因为 Sc2 是一个串行调度。因此 $r_3(B)r_1(A)w_3(B)r_2(B)r_2(A)w_2(B)r_1(B)w_1(A)$是一个冲突可串行化调度。

11．试证明若并发事务遵守两段锁协议，则对这些事务的并发调度是可串行化的。

答：首先以两个并发事务 T_1和 T_2为例，存在多个并发事务的情形可以类推。

根据可串行化定义可知，事务不可串行化只可能发生在下列两种情况：

事务 T_1写某个数据对象 A，T_2读或写 A；

事务 T_1读或写某个数据对象 A，T_2写 A。

下面称 A 为潜在冲突对象。

设 T_1 和 T_2 访问的潜在冲突的公共对象为$\{A_1,A_2,\cdots,A_n\}$。不失一般性，假设这组潜在冲突对象中 $X=\{A_1,A_2,\cdots,A_i\}$ 均符合情况（1）；$Y=\{A_{i+1},\cdots,A_n\}$ 符合情况（2）。

对于所有 $x\in X$,T_1需要 XLock x；①

T_2需要 SLock x 或 XLock x。②

（1）如果操作①先执行，则事务 T_1 获得锁，T_2 等待；

由于遵守两段锁协议，事务 T_1 在成功获得 X 和 Y 中全部对象及非潜在冲突对象的锁后，才会释放锁。这时如果存在 w∈X 或 Y，T_2 已获得 w 的锁，则会出现死锁；否则，T_1 在对 X、Y 中对象全部处理完毕后，T_2 才能执行。这相当于按 T_1、T_2 的顺序串行执行。根据可串行化定义，T_1 和 T_2 的调度是可串行化的。

（2）操作②先执行的情况与（1）对称。

因此，若并发事务遵守两段锁协议，则在不发生死锁的情况下，对这些事务的并发调度一定是可串行化的。

12．举例说明对并发事务的某个调度是可串行化的，而这些并发事务不一定遵守两段锁协议。

答：举例如下：

T_1	T_2
Slock B	
读 B = 2	
Y = B	
Unlock B	
Xlock A	
	Slock A
A = Y + 1	等待
写回 A = 3	等待
Unlock A	等待
	等待
	Slock A
	读 A = 3
	X = A
	Unlock A
	Xlock B
	B = X + 1
	写回 B = 4
	Unlock B

13．考虑如下的调度，说明这些调度集合之间的包含关系。

（1）正确的调度。

（2）可串行化的调度。

（3）遵循两阶段封锁（2PL）的调度。

（4）串行调度。

答：遵循两阶段封锁（2PL）的调度⊂可串行化得调度⊂串行调度⊂正确的调度。

14．考虑 T_1 和 T_2 两个事务。

T_1：R（A）；R（B）；B=A+B；W（B）

T_2：R（B）；R（A）；A=A+B；W（A）

（1）改写 T_1 和 T_2，增加加锁操作和解锁操作，并要求遵循两阶段封锁协议。

（2）说明 T_1 和 T_2 的执行是否会引起死锁，给出 T_1 和 T_2 的一个调度并说明之。

答：增加加锁和解锁操作后的 T_1 和 T_2 事务可串行执行，不会引起死锁。调度如下表所示：

T1	T2
SLOCK A	
XLOCK B	
R(A)	SLOCK B
R(B)	等待
B=A+B	等待
W(B)	等待
UNLOCK A	等待
UNLOCK B	XLOCK A

	R(B)
	R(A)
	A=A+B
	UNLOCK B
	UNLOCK A

15．为什么要引进意向锁？意向锁的含义是什么？

答：（1）引进意向锁是为了提高封锁子系统的效率，封锁子系统支持多种封锁粒度。原因是在多粒度封锁方法中一个数据对象可能以两种方式加锁——显式封锁和隐式封锁。因此系统在对某一数据对象加锁时不仅要检查该数据对象上有无（显式和隐式）封锁与之冲突，还要检查其所有上级结点和所有下级结点，看申请的封锁是否与这些结点上的（显式和隐式）封锁冲突，这样的检查方法效率很低，为此引进了意向锁。

（2）意向锁的含义是：对任一结点加锁时，必须先对它的上层结点加意向锁。

引进意向锁后，系统对某一数据对象加锁时，不必逐个检查与下一级结点的封锁冲突。

16．试述常用的意向锁：IS 锁、IX 锁、SIX 锁，给出这些锁的相容矩阵。

答：（1）如果对一个数据对象加 IS 锁，表示它的后裔结点拟加 S 锁。例如，要对某个元组加 S 锁，则要首先对关系和数据库加 IS 锁。

（2）如果对一个数据对象加 IX 锁，表示它的后裔结点拟加 X 锁。例如，要对某个元组加 X 锁，则要首先对关系和数据库加 IX 锁。

（3）如果对一个数据对象加 SIX 锁，表示对它加 S 锁，再加 IX 锁，即 SIX=S＋IX。

这些锁的相容矩阵如图 11-8 所示：

T_1 \ T_2	S	X	IS	IX	SIX	–
S	Y	N	Y	N	N	Y
X	N	N	N	N	N	Y
IS	Y	N	Y	Y	Y	Y
IX	N	N	Y	Y	N	Y
SIX	N	N	Y	N	N	Y
–	Y	Y	Y	Y	Y	Y

图 11-8

第 12 章　数据库管理系统

12.1　复习笔记

一、数据库管理系统（DBMS）的基本功能

DBMS 具有如下基本功能：

1．数据库定义和创建

创建数据库主要是用数据定义语言 DDL 定义和创建数据库模式、外模式、内模式等数据库对象。在关系数据库中就是建立数据库(或 Schema)、表、视图、索引等。还有创建用户、安全保密定义(如用户口令、级别、角色、存取权限)、数据库的完整性定义。

2．数据组织、存储和管理

DBMS 要分类组织、存储和管理各种数据，包括数据字典、用户数据、存取路径等；要确定以何种文件结构和存取方式在存储器上组织这些数据，如何实现数据之间的联系。数据组织和存储的基本目标是提高存储空间利用率和方便存取，提供多种存取方法(如索引查找、Hash 查找、顺序查找等)提高存取效率。

3．数据存取

提供用户对数据的操作功能，实现对数据库数据的检索、插入、修改和删除。

4．数据库事务管理和运行管理

这是指 DBMS 运行控制和管理功能。

5．数据库的建立和维护

包括数据库的初始建立、数据的转换、数据库的转储和恢复、数据库的重组织和重构造以及性能监测分析等功能。

6．其他功能

包括 DBMS 与网络中其他软件系统的通信功能；一个 DBMS 与另一个 DBMS 或文件系统的数据转换功能；异构数据库之间的互访和互操作功能等。

二、数据库管理系统的系统结构

1．层次结构

图 12-1 给出一个关系数据库管理系统的层次结构示例。这个层次结构是按照处理对象的不同，依最高级到最低级的次序来划分的，具有普遍性。

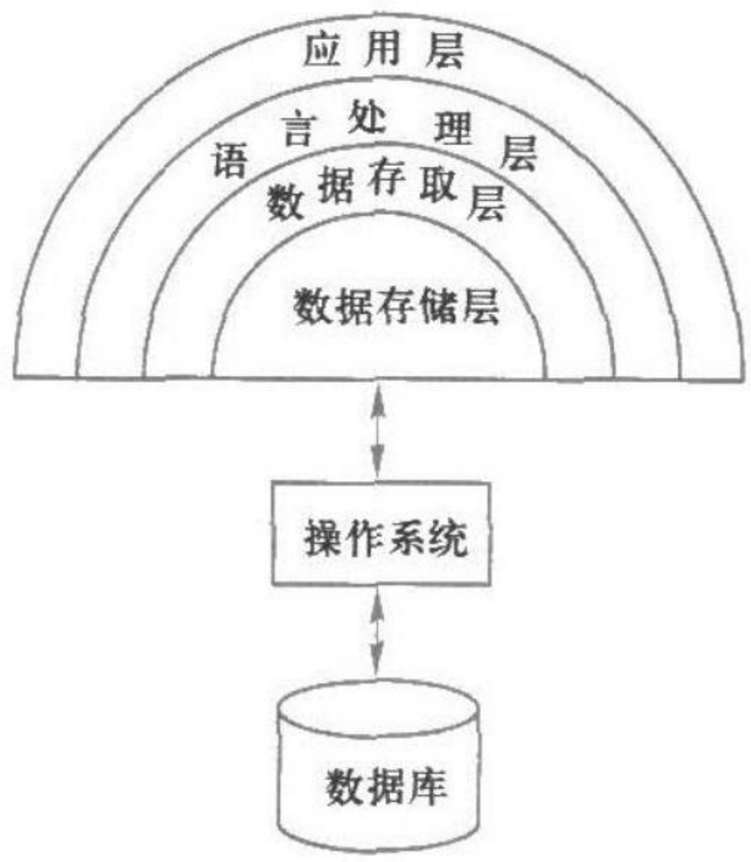


图 12-1　关系数据库管理系统的层次结构

（1）应用层

应用层位于关系数据库管理系统的核心之外。应用层处理的对象是各种各样的数据库应用，该层是关系数据库管理系统与用户 / 应用程序的界面层。

（2）语言处理层

语言处理层处理的对象是数据库语言。该层的功能是对数据库语言的各类语句进行语法分析、视图转换、安全性检查、完整性检查、查询优化等；通过对下层基本模块的调用，生成可执行代码。

（3）数据存取层

数据存取层处理的对象是单个元组。其功能是把上层的集合操作转换为单记录操作，执行扫描排序、元组的查找、插入、修改、删除、封锁等基本操作，完成数据记录的存取、存取路径维护、事务管理、并发控制和恢复等工作。

（4）数据存储层

数据存储层处理的对象是数据页和系统缓冲区。其功能是执行文件的逻辑打开、关闭、读页、写页、缓冲区读和写、页面淘汰等操作，完成缓冲区管理、内外存交换、外存的数据管理等功能。

（5）操作系统

操作系统是 RDBMS 的基础。操作系统处理的对象是数据文件的物理块。其功能是执行物理文件的读写操作，保证 RDBMS 对数据逻辑上的读写真实地映射到物理文件上。操作系统提供的存取原语和基本的存取方法通常作为和 RDBMS 存储层的接口。

2．关系数据库管理系统的运行过程

RDBMS 是一个复杂而有序的整体，应该用动态的观点看待 RDBMS 各个功能模块。如图 12-2 所示是 RDBMS 读取数据库中数据的过程。整个 RDBMS 各层模块互相配合、互相依赖共同完成对数据库的操纵。

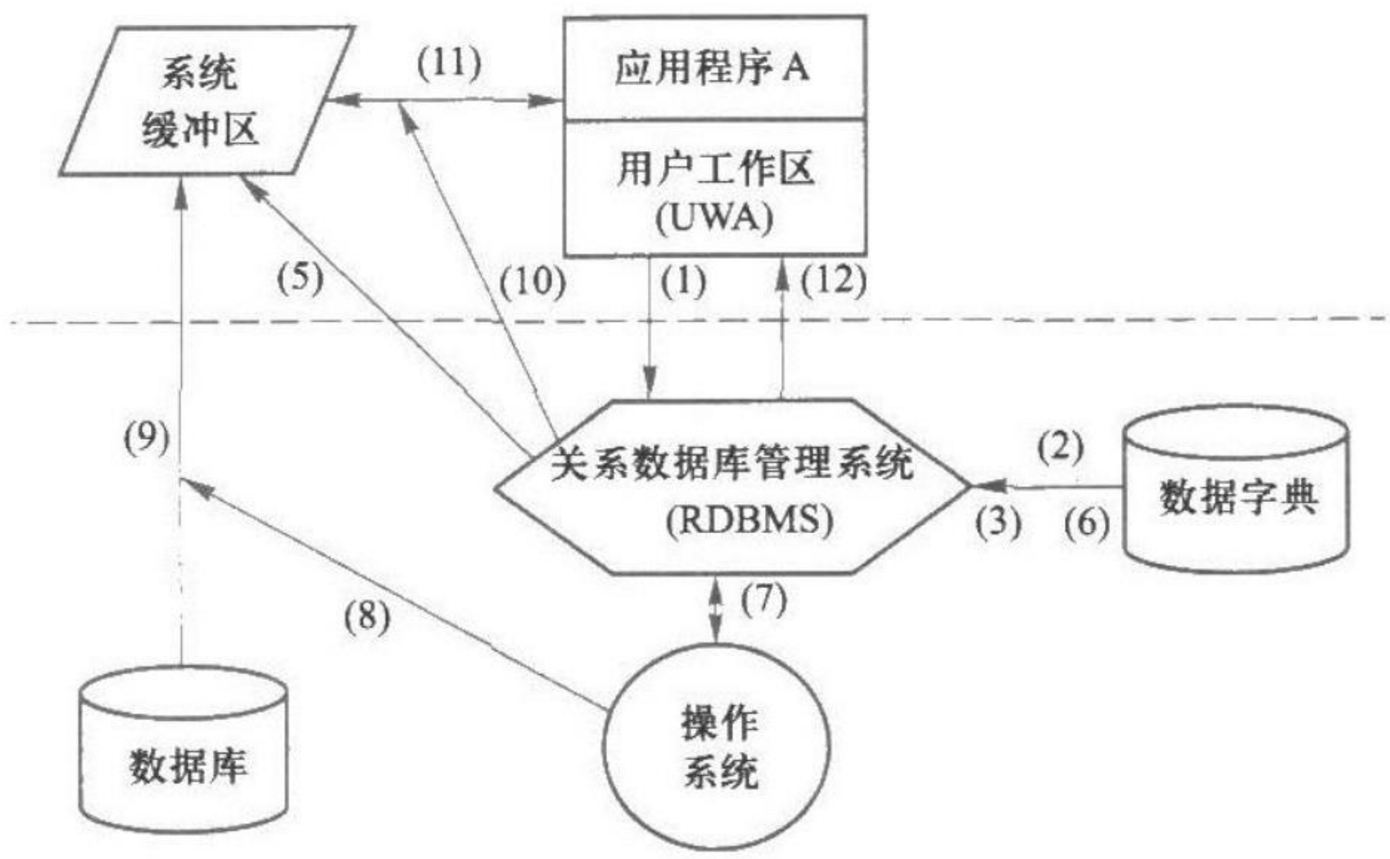


图 12-2　关系数据库管理系统的运行过程

三、语言处理层

1．任务和工作步骤

（1）任务

语言处理层的任务就是把用户在各种方式下提交给关系数据库管理系统的数据库语句转换成对关系数据库管理系统内层可执行的基本存取模块的调用序列。

（2）数据库语言

数据库语言通常包括数据定义语言、数据操纵语言和数据控制语言三部分。数据定义语句的处理相对独立和简单，数据操纵语句和数据控制语句则较为复杂。

（3）数据操纵语句处理

对数据操纵语句，语言处理层要做的工作比较多，图 12-3 给出了关系数据库管理系统中数据操纵语句处理过程的示意。

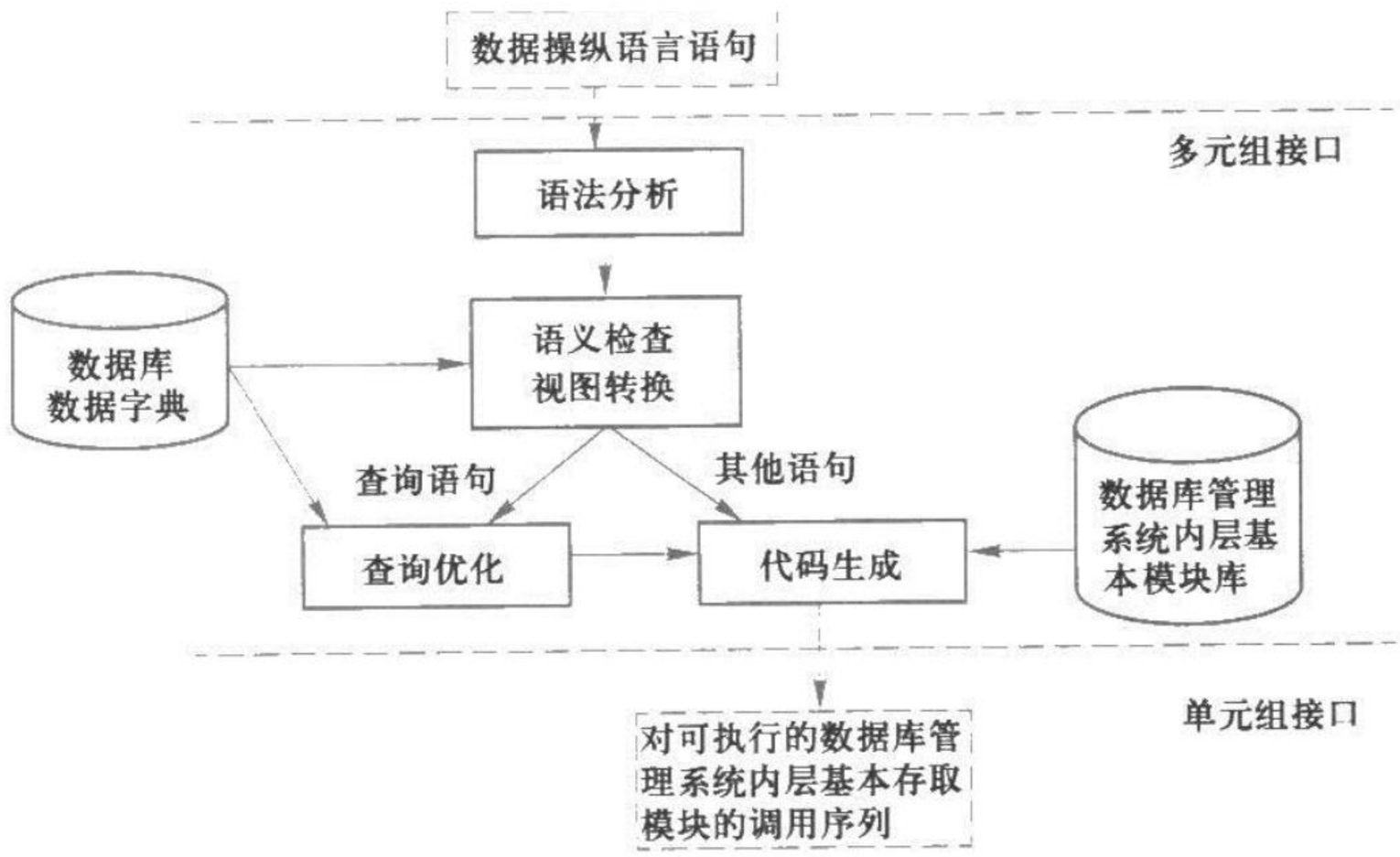


图 12-3　束缚过程

2．解释方法

（1）内容

解释执行方法的具体做法是直到执行前，数据库操纵语句都以原始字符串的形式保存；当执行到该语句时，才利用解释程序去完成全部过程，同时予以执行。这种方法通过尽量推迟束缚过程来赢得数据独立性。

（2）优点

解释方法具有灵活、应变性强的优点，甚至能适应在解释过程中发生的数据结构、存储结构等的变化，因此能保持较高的数据独立性。

（3）缺点

由于每次执行一个 DML 语句时都要执行所有步骤，尤其当这样的语句位于一个循环体内时，就要多次重复解释一个 DML 语句，开销会很大，因此效率比较低。

3．编译方法

（1）内容

预编译方法的基本思想是：在用户提交 DML 语句之后，在运行之前对它进行翻译处理，保存产生好的可执行代码。当需要运行时，取出保存的可执行代码加以执行。

（2）优点

自动重编译技术使得预编译方法既拥有了编译时进行束缚所带来的高效率，又具备了执行时束缚带来的数据独立性。

（3）缺点

使用预编译方法会遇到这样的问题：在束缚过程中进行优化所依据的条件可能在运行前已不存在，或者数据库结构已被修改，因而导致已作出的应用规划在执行时不再有效。

四、数据存取层

数据存取层介于语言处理层和数据存储层之间。它向上提供单元组接口，即导航式的一次一个元组的存取操作；向下则以系统缓冲区的存储器接口作为实现基础。

1．任务

存取层的任务主要包括：

（1）提供一次一个元组的查找、插入、删除、修改等基本操作。

（2）提供元组查找所循的存取路径以及对存取路径的维护操作，如对索引记录的查找、插入、删除、修改。若索引是采用 B+树，则应提供 B+树的建立、查找、插入、删除、修改等功能。

（3）对记录和存取路径的封锁、解锁操作。

（4）日志文件的登记和读取操作。

（5）其他辅助操作，如扫描、合并/排序，其操作对象有关系、有序表、索引等。

为了完成上述功能，通常把数据存取层又划分为若干功能子系统加以实现。

2．数据存取层的系统结构

（1）记录存取、事务管理子系统；

（2）控制信息管理模块；

（3）排序/合并子系统；

（4）存取路径维护子系统；

（5）封锁子系统，执行并发控制；

（6）日志登记子系统，用以执行恢复任务。

3．数据存取层的功能子系统

（1）记录存取、事务管理子系统

①记录存取子系统提供按某个属性值直接取一个元组和顺序取一个元组的存取原语。这种存取运算是按已选定的某个逻辑存取路径进行的，如某个数据文件或某个索引。

②事务管理子系统提供定义和控制事务的操作。

（2）日志登记子系统

日志登记子系统和事务管理子系统紧密配合，完成 RDBMS 对事务和数据库的恢复任务，它把事务开始、回滚、提交；对元组的插入、删除、修改；对索引记录的插入、删除、修改等。每一个操作作为一个日志记录存入日志文件中。当事务或系统软、硬件发生故障时利用日志文件执行恢复。

（3）控制信息管理模块

利用专门的数据区(内存中)登记不同记录类型以及不同存取路径的说明信息(取自数据字典)和控制信息。这些信息是存取元组和管理事务的依据。它和事务管理、记录存取子系统一起保证事务的正常运行。该模块提供对数据字典中说明信息的读取、增加、删除和修改操作。

（4）排序/合并子系统

排序/合并子系统实现输出有序结果，删去重复值，动态建立索引结构，减少数据块的存取次数等功能。

（5）存取路径维护子系统

对数据执行插入、删除、修改操作的同时要对相应的存取路径进行维护。

（6）封锁子系统

封锁子系统完成并发控制功能。

五、缓冲区管理

1．设立缓冲区的原因

（1）隔离外存设备与存储层以上各系统，保证 DBMS 具有设备独立性。

（2）减少内外存交换的次数，提高存取效率。

2．缓冲区构成

缓冲区由控制信息和若干定长页面组成。

（1）缓冲区管理模块向上层提供的操作是缓冲区的读、写。

（2）缓冲区内部的管理操作有：查找页、申请页、淘汰页。

（3）缓冲区管理调用操作系统的操作有：读、写。

3．主要算法

缓冲区管理中主要算法是淘汰算法和查找算法。查找算法用来确定所请求的页是否在内存，可采用顺序扫描、折半查找、Hash 查找算法等。

六、数据库的物理组织

1．数据库组织标准

衡量数据库组织是否适宜的标准包括两方面：

（1）存储效率高，节省存储空间；

（2）存取速度快，代价小。

2．数据库系统与文件系统的区别和联系

（1）数据库系统是文件系统的发展；

（2）文件系统中每个文件存储同质实体的数据，各文件是孤立的，没有体现实体之间的联系；

（3）数据库系统中数据的物理组织必须体现实体之间的联系，支持数据库的逻辑结构。

3．数据字典的组织

有关数据的描述存储在数据库的数据字典中。数据字典的特点是数据量比较小、使用频繁，因为任何数据库操作都要参照数据字典的内容。数据字典按不同的内容在逻辑上组织为若干张表，在物理上就对应若干文件而不

是一个文件。由于每个文件中存放数据量不大，可简单地用顺序文件来组织。

4．数据和数据联系的组织

（1）文件结构类型

操作系统提供的常用文件结构有：顺序文件、索引文件、索引顺序文件、Hash 文件（杂凑文件）和 B 树类文件等。

（2）关系数据库组织方式

在关系数据库中，实体及实体之间的联系都用一种数据结构—“表”来表示。在数据库的物理组织中，每一个表通常对应一种文件结构。因此数据和数据之间的联系两者组织方式相同。

5．存取路径的组织

（1）在网状和层次数据库中，存取路径是用数据之间的联系来表示的，因此已与数据结合并固定下来。

（2）关系数据库中，存取路径和数据是分离的，对用户是隐蔽的。存取路径可以动态建立、删除。关系数据库中存取路径的建立是十分灵活的。

12.2　课后习题详解

1．试述数据库管理系统的基本功能。

答：DBMS 具有如下基本功能：

（1）数据库定义和创建

创建数据库主要是用数据定义语言 DDL 定义和创建数据库模式、外模式、内模式等数据库对象。

（2）数据组织、存储和管理

DBMS 要分类组织、存储和管理各种数据，包括数据字典、用户数据、存取路径等。

（3）数据存取

提供用户对数据的操作功能，实现对数据库数据的检索、插入、修改和删除。

（4）数据库事务管理和运行管理

DBMS 运行控制和管理功能。这些功能保证了数据库系统的正常运行，保证了事务的 ACID 特性。

（5）数据库的建立和维护

数据库的建立和维护包括数据库的初始建立、数据的转换、数据库的转储和恢复、数据库的重组织和重构造以及性能监测分析等功能。

（6）其他功能

包括 DBMS 与网络中其他软件系统的通信功能；一个 DBMS 与另一个 DBMS 或文件系统的数据转换功能；异构数据库之间的互访和互操作功能等。

2．关系数据库管理系统的工作过程是什么？给出数据库管理系统插入一个记录的活动过程，画出活动过程示意图。

答：DBMS 是一个复杂而有序的整体，图 12-4 是 DBMS 的运行过程示例。结合图 12-4，给出 DBMS 插入一条记录的活动过程。

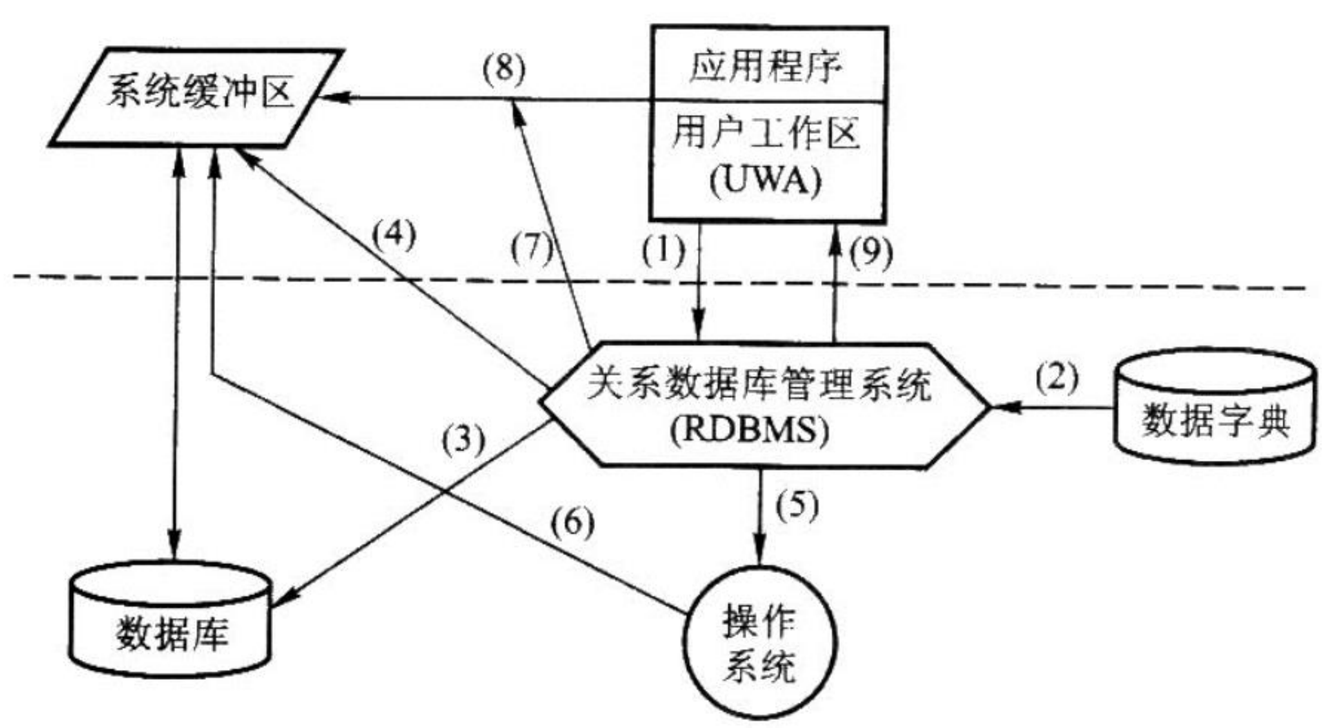


图 12-4　数据库管理系统的运行过程

（1）用户通过应用程序向 DBMS（以 RDBMS 为例）发出调用数据库数据的 INSERT 命令。在命令中给出一个关系名和所插入的元组值。

（2）DBMS 先对命令进行语法检查、语义检查和用户存取权限检查。语义检查的具体做法是，DBMS 读取数据字典，检查是否存在该关系及相应的字段，值的数据类型是否正确。检查该用户是否具有该关系上执行 INSERT 操作的权限。若未能通过检查就拒绝执行 INSERT 命令，返回有关的错误提示信息。

（3）DBMS 查看存储模式，找到新记录所应插入的位置和相应的页面 P。

（4）DBMS 在系统缓冲区中找到一个空页。

（5）DBMS 根据步骤（3）的执行结果，向操作系统发出读取物理页面 P 的命令。

（6）操作系统执行读操作。将数据页 P 读入系统缓冲区中的空白页处。

（7）DBMS 根据插入命令和数据字典的内容将数据转化成内部记录的格式。

（8）DBMS 将数据记录写入系统缓冲区的页面 P 中。

（9）DBMS 将执行事务的提交。把状态信息（如成功或不成功的指示）、例外状态信息等返回给应用程序。（这里未考虑多用户并发控制的问题。）

3．关系数据库管理系统的语言处理层是如何处理一个数据定义语言语句的？

答：语言处理层首先要对 DDL 语句进行语法检查、语义检查和用户权限检查。语义检查的具体做法是，DBMS 读取数据字典，检查是否存在与该语句中的表、视图或索引等要创建的对象相同的对象名，检查该用户是否具有创建数据库对象的权限。

然后，把 DDL 语句翻译成内部表示，将其存储在系统的数据字典中。例如建立一个新表，就要把关系名、建立者、属性个数、记录长度等信息记入数据字典中。

4．试述关系数据库管理系统的语言处理层处理一个数据操纵语言语句的大致过程。

答：DML 语句处理的过程包括：

（1）先对 DML 语句进行词法分析和语法分析，并把外部关系名、属性名转换为内部名，通过词法和语法分析后生成语法分析树；

（2）根据数据字典中的内容进行查询检查，包括审核用户的存取权限和完整性检查；

（3）对查询进行优化，包括代数优化和存取路径优化；

（4）把选中的查询执行方案描述出来。

5．什么是处理数据操纵语言语句的解释方法和预编译方法？试述二者的区别、联系，比较各自的优缺点。

答：（1）解释方法

解释执行方法的具体做法是直到执行前，数据库 DML 语句都以原始字符串的形式保存；当执行到该语句时，才利用解释程序去完成全部过程，同时予以执行。这种方法通过尽量推迟束缚过程来赢得数据独立性。

①优点：灵活、应变性强，甚至能适应在解释过程中发生的数据结构、存储结构等的变化，因此能保持较高的数据独立性。

②缺点：每次执行一个 DML 语句时都要执行所有步骤，开销很大，因此效率比较低。

（2）预编译方法

预编译方法的基本思想是：在用户提交 DML 语句之后，在运行之前对它进行翻译处理，保存产生好的可执行代码。当需要运行时，取出保存的可执行代码加以执行。

①优点：自动重编译技术使得预编译方法既拥有了编译时进行束缚所带来的高效率，又具备了执行时束缚带来的数据独立性。

②缺点：在束缚过程中进行优化所依据的条件可能在运行前已不存在，或者数据库结构已被修改，因而导致已作出的应用规划在执行时不再有效。

6．试述数据存取层主要的了系统及其功能。

答：数据库存取层主要包括以下几个子系统：

（1）记录存取、事务管理子系统

记录存取子系统提供按某个属性值直接取一个元组和顺序取一个元组的存取原语，事务管理子系统提供定义和控制事务的操作。

（2）封锁子系统，执行并发控制。

（3）恢复子系统

主要是指日志登记子系统对事物执行开始、回滚、提交；对元组进行插入、删除、修改；对索引记录进行插

入、删除、修改等等，每一个操作作为一个日志记录存入日志文件中。对不同的故障恢复策略执行相应的恢复。

（4）控制信息管理模块

该模块利用专门的数据区（内存中）登记不同记录类型以及不同存取路径的说明信息（取自数据字典）和控制信息。该信息是存取元组和管理事务的依据。它和事务管理、记录存取子系统一起保证事务的正常运行。该模块提供对数据字典中说明信息的读取、增加、删除和修改操作。

（5）存取路径维护子系统

该模块在对数据进行修改操作时要对该表上已建立的所有索引进行动态维护。

（6）排序/合并子系统

该模块主要功能包括输出有序结果，删去重复值降低开销，先对两个关系按连接属性值排序再进行连接的连接操作以提高连接运算速度，建立索引结构减少数据块的存取次数等。

7．在操作系统中也有并发控制问题，为什么数据库管理系统还要并发控制机制？

答：操作系统提供的封锁机制和 DBMS 的封锁机制在封锁对象、封锁对象的状态、封锁的粒度、及封锁的类型上存在很大的差别，操作系统的封锁机制不能直接应用在 DBMS 中，DBMS 必须重新设计，来满足复杂的封锁需求。

8．试比较数据库管理系统与操作系统的封锁技术。

答：DBMS 封锁技术比操作系统封锁内容更加丰富,技术更加复杂，同时其实现手段依赖于操作系统提供的环境。

（1）操作系统封锁对象（即系统资源）单一，封锁对象状态确定，封锁力度不能改变，排他锁类型单一。

（2）DBMS 封锁对象多样，包括用户数据、索引、数据字典等，封锁对象动态改变，封锁力度可变，封锁类型多样。

两者的具体区别如表 12-1 所示：

表 12-1　操作系统和数据库管理系统封锁技术的比较

	操作系统	数据库管理系统
封锁对象	单一,系统资源(包括 CPU、设备、表格等)	多样,数据库中各种数据对象(包括用户数据、索引(存取路径)、数据字典等)
封锁对象的状态	静态、确定、各种封锁对象在封锁表中占有一项。封锁对象数是不变的	动态,不确定。封锁对象动态改变着、常常在执行前不能确定。一个封锁对象只有当封锁时才在封锁表中占据一项
封锁的粒度	不变,由于封锁对象单一、固定,封锁粒度不会改变	可变,封锁可加到或大或小的数据单位上,封锁粒度可以是整个数据库、记录或字段
封锁的类型	单一,排它锁	多样,一般有共享锁(S Lock)、排它锁(X Lock)或其他类型的封锁,随系统而异

9．数据库管理系统中为什么要设置系统缓冲区？

答：设立系统缓冲区的原因有以下两点：

（1）隔离外存设备与存储层以上各系统，保证 DBMS 具有设备独立性。

（2）提高存取效率。利用缓冲区滞留数据，只有数据不在缓冲区时才从外存读入页面，写入数据先在页面作标记，当事务结束时或缓冲区满需调入新页时才写入外存。

10．数据库中要存储和管理的数据内容包括哪些方面？

答：数据库系统中数据的物理组织必须体现实体之间的联系，支持数据库的逻辑结构——各种数据模型。因此数据库中要存储 4 个方面的数据：

（1）数据描述，即数据的外模式、模式、内模式；

（2）数据本身；

（3）数据之间的联系；

（4）存取路径。

*11 请给出缓冲区管理中的一个淘汰算法，并上机实现（提示：首先需要设计缓冲区的数据结构，然后写出算法）。

答：借助队列实现 LRU 页面替换算法作为缓冲区管理的淘汰算法。

定义缓冲区的数据结构如下：

```
typedef struct LRU
{
	int data;
	int time;//计次数
} LRU;

typedef struct Queue
{
	LRU *pBase;//结构数组
	int front;//队列头
	int rear;//队列尾
}QUEUE;
```

完整的参考代码如下：

```
#include <stdio.h>
#include <string.h>
#include <malloc．h>
int len;

typedef struct LRU
{
	int data;
	int time;//计次数
} LRU;

typedef struct Queue
{
	LRU *pBase;//结构数组
	int front;//队列头
	int rear;//队列尾
}QUEUE;

void init(QUEUE *pQ)
{
	int N = len+1;
	pQ->pBase = (LRU*)malloc(sizeof(LRU ) * N);
	pQ->front = pQ->rear = 0;	//初始化为 0
}

int full_queue(QUEUE *pQ)
{
	int N = len+1;
	if((pQ->rear +1)%N == pQ->front)//循环队列
		return 1;
	else
		return 0;
```

```
}

int en_queue(QUEUE *pQ, int val)//入队前判断队列是否已满
{
	int N = len+1;
	if( full_queue(pQ) )
	{
		return 0;
	}
	else
	{
		pQ->pBase[pQ->rear].data = val;//压栈在队尾
		pQ->pBase[pQ->rear].time = 0;//初始化次数为 0
		pQ->rear = (pQ->rear+1) % N;
		return 1;
	}
}

int empty_queue(QUEUE *pQ)//1-->空	0-->非空
{
	int N = len+1;
	if(pQ->front == pQ->rear)
			return 1;
	else
			return 0;
}

int out_queue(QUEUE *pQ, int *pVal)//出队前判断队列是否为空
{
	int N = len+1;
	if(empty_queue(pQ))
	{
		return 0;
	}
	else
	{
		*pVal = pQ->pBase[pQ->front].data;//把出队的元素保存起来
		pQ->front = (pQ->front+1)%N;
		return 1;
	}
}

void add_time(QUEUE *pQ)
{
	int N = len+1;
	int i = pQ->front;
	while( i != pQ->rear)
	{
		pQ->pBase[i].time ++;
		i = (i + 1) % N;
```

```
            //printf("%d    %d", pQ->pBase[i].time, i);
        }
}

void Set_time_shot(QUEUE *pQ, int x)//若待入队元素与队中元素相同，将次数置为 0
{
        int N = len + 1;
        int i = pQ->front;
        while( i != pQ->rear)
        {
            if( pQ->pBase[i].data == x)
            {
                pQ->pBase[i].time = 0;
            }
            i = (i+1) % N;
        }
}

int Find_big_time(QUEUE *pQ)
{
        int N = len + 1;
        int i = pQ->front;
        int max_i = i;
        int max_time = pQ->pBase[pQ->front].time;
        while( i != pQ->rear)
        {
            if( pQ->pBase[i].time > max_time)
            {
                max_i = i; max_time = pQ->pBase[i].time;
            }
            i = (i+1) % N;
        }
        return max_i;
}

void Replace_big_time(QUEUE *pQ, int x)//若待入队元素与队中元素不相同，替换元素，并将次数置为 0
{
        int max_time = Find_big_time(pQ);
        printf("%d ", pQ->pBase[max_time].data);
        pQ->pBase[max_time].data = x;
        pQ->pBase[max_time].time = 0;
}

int same_queue(QUEUE *pQ, int x)//判断待入队元素是否与队中元素相同
{
        int N = len+1;
        int i = pQ->front;
        while( i != pQ->rear)
        {
            if( pQ->pBase[i].data == x)
```

```
                return 1;
            i = (i+1) % N;
        }
        return 0;
}

int main(void)
{
        char str[100];
        int val, data;
        int i, cnt = 0;
        scanf("%d", &len);
        scanf("%s", str);

        QUEUE Q;
        init(&Q);
        for(i=0; str[i] != '\0'; i++)
        {
            val = str[i] - '0';
            if ( empty_queue( &Q ) )//如果队列为空
            {
                en_queue(&Q, val);
            }
            else
            {
                add_time(&Q);
                if(full_queue(&Q))//如果队列已满
                {
                    if( !same_queue(&Q, val))
                    {
                        Replace_big_time(&Q, val);
                    }
                    else
                    {
                        Set_time_shot(&Q, val);
                        cnt++;
                    }
                }
                else//如果队列没满也不为空
                {
                    if( !same_queue(&Q, val))
                    {
                        en_queue(&Q, val);
                    }
                    else
                    {
                        Set_time_shot(&Q, val);
                        cnt++;
                    }
                }
```

```
            }
        }
        printf("\n%d/%d", cnt, strlen(str));
        return 0;
    }
```

*12．请写出对一个文件按某个属性的排序算法（设该文件的记录是定长的），并上机实现。若要按多个属性排序，能否写出改进的算法？

答：（1）使用败者树实现多路归并的外部排序算法，对文件按某个属性进行排序。

参考代码：

```
#include <stdio.h>
#include <stdlib．h>
#include <string.h>

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define MINKEY -1
#define MAXKEY 100

/* Status 是函数的类型,其值是函数结果状态代码，如 OK 等 */
typedef int Status;

/* Boolean 是布尔类型,其值是 TRUE 或 FALSE */
typedef int Boolean;
/* 一个用作示例的小顺序表的最大长度 */
#define MAXSIZE 20

typedef int KeyType;
/* k 路归并 */
#define k 3

/* 设输出 M 个数据换行 */
#define M 10

/* k+1 个文件指针(fp[k]为大文件指针)，全局变量 */
FILE *fp[k + 1];

/* 败者树是完全二叉树且不含叶子，可采用顺序存储结构 */
typedef int LoserTree[k];

typedef KeyType ExNode, External[k+1];

/* 全局变量 */
External b;

/* 从第 i 个文件(第 i 个归并段)读入该段当前第 1 个记录的关键字到外结点 */
int input(int i, KeyType *a){
```

```
    int j = fscanf(fp[i], "%d ", a);
    if(j > 0){
        printf("%d\n", *a);
        return 1;
    }else{
        return 0;
    }
}

/* 将第 i 个文件(第 i 个归并段)中当前的记录写至输出归并段 */
void output(int i){
    fprintf(fp[k], "%d ", b[i]);
}

/* 沿从叶子结点 b[s]到根结点 ls[0]的路径调整败者树。*/
void Adjust(LoserTree ls, int s){
    int i, t;

    /* ls[t]是 b[s]的双亲结点 */
    t = (s + k) / 2;
    while(t > 0){
        /* s 指示新的胜者 */
        if(b[s] > b[ls[t]]){
            i = s;
            s = ls[t];
            ls[t] = i;
        }
        t = t / 2;
    }
    ls[0] = s;
}

/**
* 已知 b[0]到 b[k-1]为完全二叉树 ls 的叶子结点，存有 k 个关键字，沿从叶子
* 到根的 k 条路径将 ls 调整成为败者树。
*/
void CreateLoserTree(LoserTree ls){
    int i;
    b[k] = MINKEY;

    /* 设置 ls 中“败者”的初值 */
    for(i = 0; i < k; ++i){
        ls[i] = k;
    }

    /* 依次从 b[k-1]，b[k-2]，…，b[0]出发调整败者 */
    for(i = k - 1; i >= 0; --i){
        Adjust(ls, i);
    }
}
```

```
/**
* 利用败者树 ls 将编号从 0 到 k-1 的 k 个输入归并段中的记录归并到输出归并段。
* b[0]至 b[k-1]为败者树上的 k 个叶子结点，分别存放 k 个输入归并段中当前记录的关键字。
*/
void K_Merge(LoserTree ls, External b){
    int i, q;

    /* 分别从 k 个输入归并段读人该段当前第一个记录的关键字到外结点 */
    for(i = 0; i < k; ++i) {
        input(i, &b[i]);
    }

    /* 建败者树 ls，选得最小关键字为 b[ls[0]].key */
    CreateLoserTree(ls);

    while(b[ls[0]] != MAXKEY){
        /* q 指示当前最小关键字所在归并段 */
        q = ls[0];

        /* 将编号为 q 的归并段中当前（关键字为 b[q].key）的记录写至输出归并段 */
        output(q);

        /* 从编号为 q 的输入归并段中读人下一个记录的关键字 */
        if(input(q, &b[q]) > 0){
            /* 调整败者树，选择新的最小关键字 */
            Adjust(ls,q);
        }
    }

    /* 将含最大关键字 MAXKEY 的记录写至输出归并段 */
    output(ls[0]);
}

void show(KeyType t) {
    printf("(%d)", t);
}

int main(){
    KeyType r;
    int i, j;
    char fname[k][4], fout[5] = "out", s[3];
    LoserTree ls;

    /* 依次打开 f0,f1,f2,…,k 个文件 */
    for(i = 0; i < k; i++){
        /* 生成 k 个文件名 f0,f1,f2,… */
        itoa(i, s, 10);
        strcpy(fname[i], "f");
        strcat(fname[i], s);
```

```
/* 以读的方式打开文件 f0,f1,… */
fp[i] = fopen(fname[i], "r");
printf("有序子文件 f%d 的记录为:\n",i);

/* 依次将 f0,f1,…的数据读入 r */
do{
    j = fscanf(fp[i], "%d ", &r);
    /* 输出 r 的内容 */
    if(j == 1){
        show(r);
    }
    }while(j == 1);
    printf("\n");

    /* 使 fp[i]的指针重新返回 f0,f1,…的起始位置，以便重新读入内存 */
    rewind(fp[i]);
}

/* 以写的方式打开大文件 fout */
fp[k] = fopen(fout, "w");

/* 利用败者树 ls 将 k 个输入归并段中的记录归并到输出归并段，即大文件 fout */
K_Merge(ls, b);

/* 关闭文件 f0,f1,…和文件 fout */
for(i = 0; i <= k; i++){
    fclose(fp[i]);
}

/* 以读的方式重新打开大文件 fout 验证排序 */
fp[k] = fopen(fout, "r");
printf("排序后的大文件的记录为:\n");

i = 1;
do{
    /* 将 fout 的数据读入 r */
    j = fscanf(fp[k], "%d ", &r);

    /* 输出 r 的内容 */
    if(j == 1){
        show(r);
    }

    /* 换行 */
    if(i++ % M == 0){
        printf("\n");
    }
}while(j == 1);
printf("\n");
```

```
    /* 关闭大文件 fout */
    fclose(fp[k]);
    return 0;
}
```

（2）若要按多个属性排序，可以采用基数排序算法。

实现方法有：①最高位优先(Most Significant Digit first)法，简称 MSD 法。先按 k1 排序分组，同一组中记录，关键码 k1 相等，再对各组按 k2 排序分成子组，之后，对后面的关键码继续这样的排序分组，直到按最次位关键码 kd 对各子组排序后。再将各组连接起来，便得到一个有序序列。②最低位优先(Least Significant Digit first)法，简称 LSD 法：先从 kd 开始排序，再对 kd-1 进行排序，依次重复，直到对 k1 排序后便得到一个有序序列。

*13．请给出 B+树文件的创建和维护（增、删、改）算法并上机实现（提示：设 B+树的叶结点上仅存放索引项（码值，TID），首先要设计索引项，B+树叶页和非叶页的数据结构，然后写出算法）。

答：

参考代码如下：

```
#include <stdlib．h>
#include <stdio.h>
#include <string.h>
#include <malloc．h>

#define MAX_KEY_LEN 15
#define MAX_NUM_VALUE 50

//定义 B 树结点的数据结构
typedef struct node {
    void **pointers;
    char **keys;
    struct node *parent;
    int num_keys;
    bool is_leaf;//为 true 表示叶页，为 false 表示非叶页
    struct node *next;    // used for printing
} node;

typedef struct record {
    int value;
} record;

int size = 5;    // number of pointers of each node

void print_tree(node *root);
record *find(node *root, char *key);
node *find_leaf(node *root, char *key);

// 插入函数声明 Insertion
record *make_new_record(int value);
node *make_new_node();
node *make_new_leaf();
node *make_new_tree(char *key, int value);
node *make_new_root(node *left, node *right, char *key);
node *insert(node *root, char *key, int value);
```

```
node *insert_into_parent(node *root, node *left, node *right, char *key);
void insert_into_node(node *nd, node *right, int index, char *key);
node *insert_into_node_after_splitting(node *root, node *nd, node *right,
                                        int index, char *key);
node *insert_into_leaf_after_splitting(node *root, node *leaf, int index,
                                        char *key, record *rec);
void insert_into_leaf(node *leaf, int index, char *key, record *rec);

// 删除函数声明 Deletion
void destroy_node(node *nd);
void *remove_entry(node *nd, int index);
node *delete(node *root, char *key);
node *delete_entry(node *root, node *nd, int index);
node *adjust_root(node *root);
int get_node_index(node *nd);
node *coalesce_nodes(node *root, node *nd, node *neighbor, int nd_index);
void distribute_nodes(node *nd, node *neighbor, int nd_index);

// Bulk Loading -- build B+-tree after sorting
node *bulk_load(char (*keys)[MAX_KEY_LEN], int *values, int n);
int cmp(const void *, const void *);

//打印树结点
void print_tree(node *root)
{
    node *p, *p_down;
    int i;
    if (root == NULL){
        printf("Empty tree!\n");
        return;
    }
    p = root;
    p_down = root;
    while (!p->is_leaf){
        for (i = 0; i < p->num_keys; i++)
            printf("%s ", p->keys[i]);
            // printf("%d ", p->keys[i][0]);    // for test
        printf("| ");
        p = p->next;
        if (!p){
            p_down = p_down->pointers[0];    // next level
            p = p_down;
            printf("\n");
        }
    }

    while (p){
        for (i = 0; i < p->num_keys; i++)
            printf("%s ", p->keys[i]);
            // printf("%d ", p->keys[i][0]);    // for test
```

```
            printf(" | ");
            p = p->pointers[size-1];
        }
        printf("\n");
}

//查找
record *find(node *root, char *key)
{
        node *leaf;
        int i;
        leaf = find_leaf(root, key);
        if (leaf == NULL)
            return NULL;
        for (i = 0; i < leaf->num_keys && strcmp(leaf->keys[i], key) != 0; i++)
            ;
        if (i == leaf->num_keys)
            return NULL;
        return (record *)leaf->pointers[i];
}

node *find_leaf(node *root, char *key)
{
        node *nd;
        int i;
        if (root == NULL)
            return root;
        nd = root;
        while (!nd->is_leaf){
            for (i = 0; i < nd->num_keys && strcmp(nd->keys[i], key) <= 0; i++)
                ;
            nd = (node *)nd->pointers[i];
        }
        return nd;
}

record *make_new_record(int value)
{
        record *rec;
        rec = (record *)malloc(sizeof(record));
        rec->value = value;
        return rec;
}

node *make_new_node()
{
        node *nd;
        nd = (node *)malloc(sizeof(node));
        nd->pointers = malloc(size * sizeof(void *));
        nd->keys = malloc((size - 1) * sizeof(char *));
```

```
        nd->parent = NULL;
        nd->num_keys = 0;
        nd->is_leaf = false;
        nd->next = NULL;
        return nd;
}

node *make_new_leaf()
{
        node *leaf;
        leaf = make_new_node();
        leaf->is_leaf = true;
        return leaf;
}

node *make_new_tree(char *key, int value)
{
        node *root;
        record *rec;
        root = make_new_leaf();
        rec = make_new_record(value);
        root->pointers[0] = rec;
        root->keys[0] = malloc(MAX_KEY_LEN);
        strcpy(root->keys[0], key);
        root->pointers[size-1] = NULL;
        root->num_keys++;
        return root;
}

node *make_new_root(node *left, node *right, char *key)
{
        node *root;
        root = make_new_node();
        root->pointers[0] = left;
        root->pointers[1] = right;
        root->keys[0] = malloc(MAX_KEY_LEN);
        strcpy(root->keys[0], key);
        root->num_keys++;
        left->parent = root;
        right->parent = root;
        return root;
}

node *insert(node *root, char *key, int value)
{
        record *rec;
        node *leaf;
        int index, cond;
        leaf = find_leaf(root, key);
        if (!leaf){    // cannot find the leaf, the tree is empty
```

```
        return make_new_tree(key, value);
    }
    for (index = 0; index < leaf->num_keys && (cond = strcmp(leaf->keys[index], key)) < 0; index++)
        ;
    if (cond == 0)    // ignore duplicates
        return root;
    rec = make_new_record(value);
    if (leaf->num_keys < size - 1){
        insert_into_leaf(leaf, index, key, rec);
        return root;    // the root remains unchanged
    }
    return insert_into_leaf_after_splitting(root, leaf, index, key, rec);
}

node *insert_into_parent(node *root, node *left, node *right, char *key)
{
    node *parent;
    int index, i;
    parent = left->parent;

    if (parent == NULL){
        return make_new_root(left, right, key);
    }

    for (index = 0; index < parent->num_keys && parent->pointers[index] != left; index++);
        ;
    if (parent->num_keys < size - 1){
        insert_into_node(parent, right, index, key);
        return root;    // the root remains unchanged
    }
    return insert_into_node_after_splitting(root, parent, right, index, key);
}

void insert_into_node(node *nd, node *right, int index, char *key)
{
    int i;
    for (i = nd->num_keys; i > index; i--){
        nd->keys[i] = nd->keys[i-1];
        nd->pointers[i+1] = nd->pointers[i];
    }
    nd->keys[index] = malloc(MAX_KEY_LEN);
    strcpy(nd->keys[index], key);
    nd->pointers[index+1] = right;
    nd->num_keys++;
}

node *insert_into_node_after_splitting(node *root, node *nd, node *right, int index, char *key)
{
    int i, split;
    node **temp_ps, *new_nd, *child;
```

```
char **temp_ks, *new_key;
temp_ps = malloc((size + 1) * sizeof(node *));
temp_ks = malloc(size * sizeof(char *));

for (i = 0; i < size + 1; i++){
	if (i == index + 1)
		temp_ps[i] = right;
	else if (i < index + 1)
		temp_ps[i] = nd->pointers[i];
	else
		temp_ps[i] = nd->pointers[i-1];
}
for (i = 0; i < size; i++){
	if (i == index){
		temp_ks[i] = malloc(MAX_KEY_LEN);
		strcpy(temp_ks[i], key);
	}
	else if (i < index)
		temp_ks[i] = nd->keys[i];
	else
		temp_ks[i] = nd->keys[i-1];
}


split = size % 2 ? size / 2 + 1 : size / 2;	// split is #pointers
nd->num_keys = split - 1;
for (i = 0; i < split - 1; i++){
	nd->pointers[i] = temp_ps[i];
	nd->keys[i] = temp_ks[i];
}
nd->pointers[i] = temp_ps[i];	// i == split - 1
new_key = temp_ks[split - 1];

new_nd = make_new_node();
new_nd->num_keys = size - split;
for (++i; i < size; i++){
	new_nd->pointers[i - split] = temp_ps[i];
	new_nd->keys[i - split] = temp_ks[i];
}
new_nd->pointers[i - split] = temp_ps[i];
new_nd->parent = nd->parent;
for (i = 0; i <= new_nd->num_keys; i++){	//	#pointers == num_keys + 1
	child = (node *)(new_nd->pointers[i]);
	child->parent = new_nd;
}
new_nd->next = nd->next;
nd->next = new_nd;

free(temp_ps);
free(temp_ks);
```

```
        return insert_into_parent(root, nd, new_nd, new_key);
}

void insert_into_leaf(node *leaf, int index, char *key, record *rec)
{
        int i;
        for (i = leaf->num_keys; i > index; i--){
                leaf->keys[i] = leaf->keys[i-1];
                leaf->pointers[i] = leaf->pointers[i-1];
        }
        leaf->keys[index] = malloc(MAX_KEY_LEN);
        strcpy(leaf->keys[index], key);
        leaf->pointers[index] = rec;
        leaf->num_keys++;
}

node *insert_into_leaf_after_splitting(node *root, node *leaf, int index, char *key, record *rec)
{
        node *new_leaf;
        record **temp_ps;
        char **temp_ks, *new_key;
        int i, split;

        temp_ps = malloc(size * sizeof(record *));
        temp_ks = malloc(size * sizeof(char *));
        for (i = 0; i < size; i++){
                if (i == index){
                        temp_ps[i] = rec;
                        temp_ks[i] = malloc(MAX_KEY_LEN);
                        strcpy(temp_ks[i], key);
                }
                else if (i < index){
                        temp_ps[i] = leaf->pointers[i];
                        temp_ks[i] = leaf->keys[i];
                }
                else{
                        temp_ps[i] = leaf->pointers[i-1];
                        temp_ks[i] = leaf->keys[i-1];
                }
        }

        split = size / 2;
        leaf->num_keys = split;
        for (i = 0; i < split; i++){
                leaf->pointers[i] = temp_ps[i];
                leaf->keys[i] = temp_ks[i];
        }

        new_leaf = make_new_leaf();
        new_leaf->num_keys = size - split;
```

```
    for (; i < size; i++){
        new_leaf->pointers[i - split] = temp_ps[i];
        new_leaf->keys[i - split] = temp_ks[i];
    }

    new_leaf->parent = leaf->parent;
    new_leaf->pointers[size - 1] = leaf->pointers[size - 1];
    leaf->pointers[size - 1] = new_leaf;
    free(temp_ps);
    free(temp_ks);
    new_key = new_leaf->keys[0];
    return insert_into_parent(root, leaf, new_leaf, new_key);
}

node *delete(node *root, char *key)
{
    node *leaf;
    record *rec;
    int i;
    leaf = find_leaf(root, key);
    if (leaf == NULL)
        return root;
    for (i = 0; i < leaf->num_keys && strcmp(leaf->keys[i], key) != 0; i++)
        ;
    if (i == leaf->num_keys)    // no such key
        return root;
    rec = (record *)leaf->pointers[i];
    root = delete_entry(root, leaf, i);
    return root;
}

node *delete_entry(node *root, node *nd, int index)
{
    int min_keys, cap, nd_index;
    node *neighbor;

    remove_entry(nd, index);
    if (nd == root)
        return adjust_root(nd);
    min_keys = nd->is_leaf ? size / 2 : (size - 1) / 2;
    if (nd->num_keys >= min_keys) {
        return root;
    }

    nd_index = get_node_index(nd);
    if (nd_index == 0)
        neighbor = nd->parent->pointers[1];    // right neighbor
    else
        neighbor = nd->parent->pointers[nd_index - 1]; // left neighbor
```

```
        cap = nd->is_leaf ? size - 1 : size - 2;
        if (neighbor->num_keys + nd->num_keys <= cap)
                return coalesce_nodes(root, nd, neighbor, nd_index);

        distribute_nodes(nd, neighbor, nd_index);
        return root;
}

void distribute_nodes(node *nd, node *neighbor, int nd_index)
{
        int i;
        node *tmp;
        if (nd_index != 0) {
                if (!nd->is_leaf)
                        nd->pointers[nd->num_keys + 1] = nd->pointers[nd->num_keys];
                for (i = nd->num_keys; i > 0; i--){    // shift to right by 1
                        nd->keys[i] = nd->keys[i - 1];
                        nd->pointers[i] = nd->pointers[i - 1];
                }
                if (!nd->is_leaf){
                        nd->keys[0] = nd->parent->keys[nd_index - 1];

                        nd->pointers[0] = neighbor->pointers[neighbor->num_keys];
                        tmp = (node *)nd->pointers[0];
                        tmp->parent = nd;
                        neighbor->pointers[neighbor->num_keys] = NULL;

                        nd->parent->keys[nd_index - 1] = neighbor->keys[neighbor->num_keys - 1];
                        neighbor->keys[neighbor->num_keys - 1] = NULL;
                }
                else {
                        nd->keys[0] = neighbor->keys[neighbor->num_keys - 1];
                        neighbor->keys[neighbor->num_keys - 1] = NULL;

                        nd->pointers[0] = neighbor->pointers[neighbor->num_keys - 1];
                        neighbor->pointers[neighbor->num_keys - 1] = NULL;

                        // nd->parent->keys[nd_index - 1] = nd->keys[0];    //    share the same key with child !!
                        strcpy(nd->parent->keys[nd_index - 1], nd->keys[0]);
                }
        }
        else {
                if (!nd->is_leaf){
                        nd->keys[nd->num_keys] = nd->parent->keys[0];    // link to father's key
                        nd->pointers[nd->num_keys + 1] = neighbor->pointers[0];
                        tmp = (node *)nd->pointers[nd->num_keys + 1];
                        tmp->parent = nd;
                        nd->parent->keys[0] = neighbor->keys[0];    //
                }
                else {
```

```
        nd->keys[nd->num_keys] = neighbor->keys[0];
        nd->pointers[nd->num_keys] = neighbor->pointers[0];
        // nd->parent->keys[0] = neighbor->keys[1];    // share the same key with chid !!
        strcpy(nd->parent->keys[0], neighbor->keys[1]);
    }
    for (i = 0; i < neighbor->num_keys - 1; i++){
        neighbor->keys[i] = neighbor->keys[i + 1];
        neighbor->pointers[i] = neighbor->pointers[i + 1];
    }
    neighbor->keys[i] = NULL;
    if (!nd->is_leaf)
        neighbor->pointers[i] = neighbor->pointers[i + 1];
        else
            neighbor->pointers[i] = NULL;
    }

    neighbor->num_keys--;
    nd->num_keys++;

}

node *coalesce_nodes(node *root, node *nd, node *neighbor, int nd_index)
{
    int i, j, start, end;
    char *k_prime;
    node *tmp, *parent;

    if (nd_index == 0) {    // make sure neighbor is on the left
        tmp = nd;
        nd = neighbor;
        neighbor = tmp;
        nd_index = 1;
    }
    parent = nd->parent;

    start = neighbor->num_keys;
    if (nd->is_leaf){
        for (i = start, j = 0; j < nd->num_keys; i++, j++){
            neighbor->keys[i] = nd->keys[j];
            neighbor->pointers[i] = nd->pointers[j];
            nd->keys[j] = NULL;
            nd->pointers[j] = NULL;
        }
        neighbor->num_keys += nd->num_keys;
        neighbor->pointers[size - 1] = nd->pointers[size - 1];
    }
    else {
        neighbor->keys[start] = malloc(MAX_KEY_LEN);
        strcpy(neighbor->keys[start], parent->keys[nd_index - 1]);
        // neighbor->keys[start] = parent->keys[nd_index - 1];
```

```
        for (i = start + 1, j = 0; j < nd->num_keys; i++, j++){
                neighbor->keys[i] = nd->keys[j];
                neighbor->pointers[i] = nd->pointers[j];
        }
        neighbor->pointers[i] = nd->pointers[j];
        neighbor->num_keys += nd->num_keys + 1;
        neighbor->next = nd->next;

        for (i = 0; i <= neighbor->num_keys; i++){
                tmp = (node *)neighbor->pointers[i];
                tmp->parent = neighbor;
        }
        }
        destroy_node(nd);
        return delete_entry(root, parent, nd_index);
}

int get_node_index(node *nd)
{
        node *parent;
        int i;
        parent = nd->parent;
        for (i = 0; i < parent->num_keys && parent->pointers[i] != nd; i++)
                ;
        return i;
}

void destroy_node(node *nd)
{
        free(nd->keys);
        free(nd->pointers);
        free(nd);
}

node *adjust_root(node *root)
{
        node *new_root;
        if (root->num_keys > 0)    // at least two childs
                return root;
        if (!root->is_leaf){    // root has only one child
                new_root = root->pointers[0];
                new_root->parent = NULL;
        }
        else
                new_root = NULL;
        destroy_node(root);
        return new_root;
}

void *remove_entry(node *nd, int index)
```

```
{
    int i, index_k;

    if (nd->is_leaf){
        free(nd->keys[index]);
        free(nd->pointers[index]);    // destroy the record
        for (i = index; i < nd->num_keys - 1; i++){
            nd->keys[i] = nd->keys[i + 1];
            nd->pointers[i] = nd->pointers[i + 1];
        }
        nd->keys[i] = NULL;
        nd->pointers[i] = NULL;
    }
    else{
        index_k = index - 1;    // index_p == index
        free(nd->keys[index_k]);
        for (i = index_k; i < nd->num_keys - 1; i++){
            nd->keys[i] = nd->keys[i + 1];
            nd->pointers[i + 1] = nd->pointers[i + 2];
        }
        nd->keys[i] = NULL;
        nd->pointers[i + 1] = NULL;
    }
    nd->num_keys--;
}

node *bulk_load(char (*keys)[MAX_KEY_LEN], int *values, int n)
{
    node *root, *p;
    record *rec;
    int i;
    qsort(keys, n, MAX_KEY_LEN, cmp);
    p = NULL;
    root = make_new_tree(keys[0], values[0]);
    for (i = 1; i < n; i++) {
        if (strcmp(keys[i], keys[i-1]) == 0)    // ignore duplicates (key)
            continue;
        p = root;
        while (!p->is_leaf) {
            p = p->pointers[p->num_keys];    // right most child
        } // p is the right most child
        rec = make_new_record(values[i]);
        if (p->num_keys < size - 1)
            insert_into_leaf(p, p->num_keys, keys[i], rec);
        else
            root = insert_into_leaf_after_splitting(root, p, p->num_keys, keys[i], rec);
    }
    return root;
}
```

```
int cmp(const void *p, const void *q)
{
	int cond;
	if ((cond = strcmp((char *)p, (char *)q)) < 0)
		return -1;
	else if (cond > 0)
		return 1;
	return 0;
}

// for test
void test_find(node *root)
{
	char *key;
	record *r;
	key = malloc(MAX_KEY_LEN);
	while (1) {
		scanf("%s", key);
		if (strcmp(key, "exit") == 0)
			break;
		r = find(root, key);
		if (r == NULL) {
			printf("Not found!!\n");
			continue;
		}
		printf("Record of %s: %d\n", key, r->value);
	}
}

node *test_delete(node *root)
{
	char *key;
	key = malloc(MAX_KEY_LEN);
	while (1) {
		scanf("%s", key);
		if (strcmp(key, "exit") == 0)
			break;
		root = delete(root, key);
		print_tree(root);
	}
	return root;
}
// end of test

main(int argc, char *argv[])
{
	node *root = NULL;
	char keys[MAX_NUM_VALUE][MAX_KEY_LEN];
	int values[MAX_NUM_VALUE];
	int i, n;
```

```
    FILE *fp;
    if (argc > 1) {
        fp = fopen(argv[1], "r");
        for (n = 0; n < MAX_NUM_VALUE && fscanf(fp, "%s%d", keys[n], &values[n]) != EOF; n++)
            ;
        fclose(fp);
    }
    else {
        for (n = 0; n < MAX_NUM_VALUE && scanf("%s%d", keys[n], &values[n]) != EOF; n++)
            ;
    }
    root = bulk_load(keys, values, n);
    print_tree(root);

    root = NULL;
    while (n--){
        root = insert(root, keys[n], values[n]);
    }
    print_tree(root);

    // test_find(root);
    test_delete(root);
}
```

注：习题 11、12、13 可作为学生数据库管理系统课程的实习课题。这些模块是数据库管理系统中必不可少的基本模块。

第 4 篇　新技术篇

第 13 章　数据库技术发展概述

13.1　复习笔记

一、数据库技术发展历史

数据库技术产生于 20 世纪 60 年代中期，至今仅仅 50 年的历史，已经历了三代演变，发展了以数据建模和数据库管理系统核心技术为主，内容丰富的一门学科。

图 13-1 通过一个三维视图从数据模型、相关技术、应用领域三个方面描述了数据库系统的发展历史、特点和相互关系。

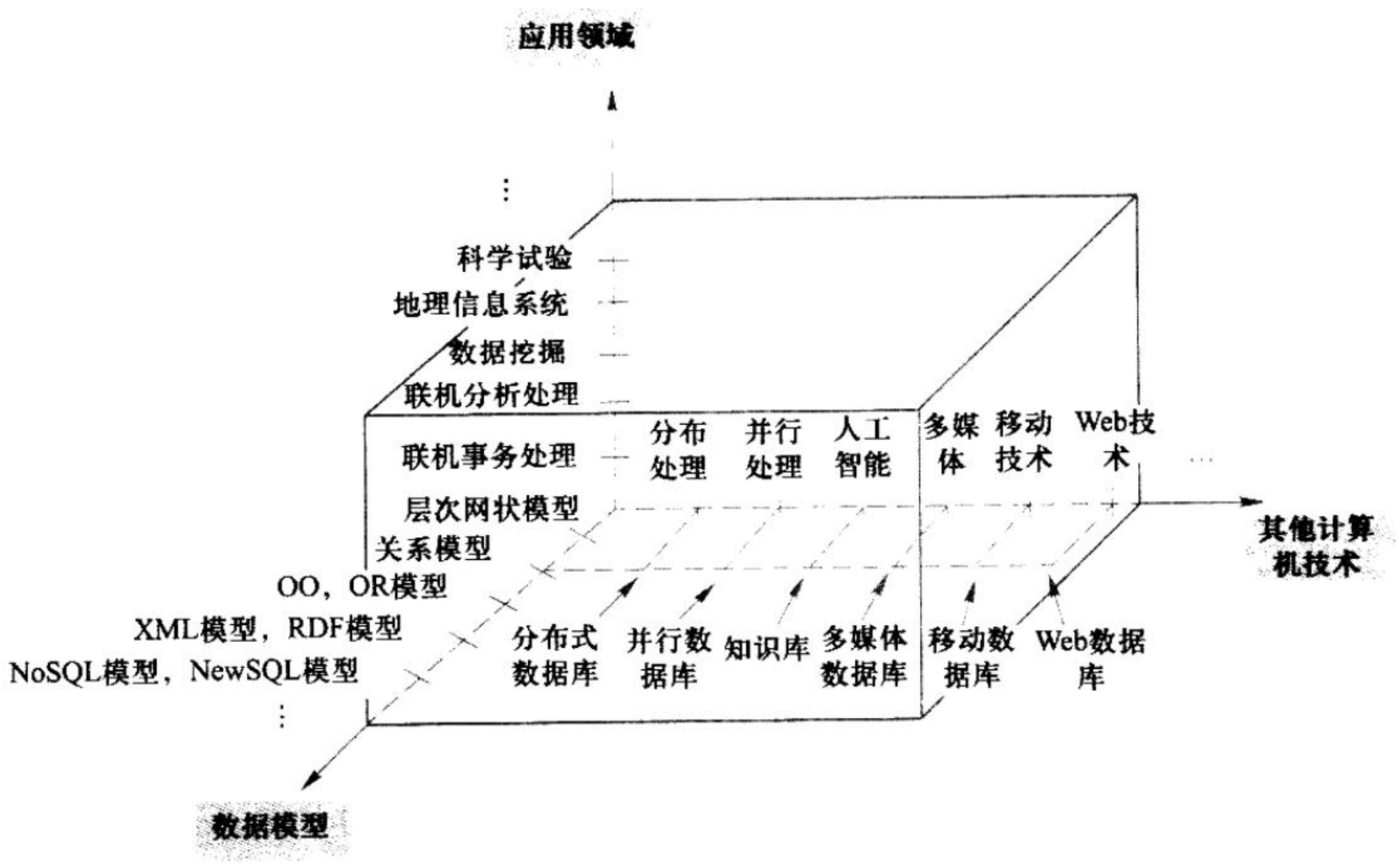


图 13-1　数据库系统的发展和相互关系示意图

二、数据库发展的三个阶段

1．第一代数据库系统

层次模型和网状模型都是格式化模型。它们从体系结构、数据库语言到数据存储管理均具有共同特征，是第一代数据库系统。

（1）代表

①1969 年由 IBM 公司研制的层次模型数据库管理系统 IMS。

②美国数据库系统语言研究会（CODASYL）下属的数据库任务组（DBTG）对数据库方法进行了系统的研究和探讨，于 20 世纪 60 年代末 70 年代初提出了若干报告，称为 DBTG 报告。

（2）共同特点

①支持三级模式（外模式、模式、内模式）的体系结构。模式之间具有转换（或称为映射）功能；

②用存取路径来表示数据之间的联系；

③独立的数据定义语言。

2．第二代数据库系统

支持关系数据模型的关系数据库系统是第二代数据库系统。

（1）主要成果

①奠定了关系模型的理论基础，给出了人们一致接受的关系模型的规范说明；

②研究了关系数据语言，包括关系代数、关系演算、SQL 及 QBE 等。确立了 SQC 为关系数据库语言标准。

（2）优点

①模型简单清晰；

②理论基础好；

③数据独立性强；

④数据库语言非过程化和标准化。

3．新一代数据库系统

三个基本特征：

（1）支持数据管理、对象管理和知识管理；

（2）保持或继承第二代数据库系统的技术；

（3）必须对其他系统开放。

三、数据库系统发展的特点

1．数据模型的发展

数据库的发展集中表现在数据模型的发展上。从最初的层次、网状数据模型发展到关系数据模型，数据库技术产生了巨大的飞跃。

（1）面向对象数据模型

一系列面向对象核心概念构成了面向对象数据模型（Object Oriented Data Model，OO 模型）的基础，主要包括以下一些概念：

①现实世界中的任何事物都被建模为对象。每个对象具有一个唯一的对象标识（OID）。

②对象是其状态和行为的封装，其中状态是对象属性值的集合，行为是变更对象状态的方法集合。

③具有相同属性和方法的对象的全体构成了类，类中的对象称为类的实例。

④类的属性的定义域也可以是类，从而构成了类的复合。类具有继承性，一个类可以继承另一个类的属性与方法，被继承类和继承类也称为超类和子类。类与类之间的复合与继承关系形成了一个有向无环图，称为类层次。

⑤对象是被封装起来的，它的状态和行为在对象外部不可见，从外部只能通过对象显式定义的消息传递对对象进行操作。

（2）XML 数据模型

XML 数据模型由表示 XML 文档的结点标记树、结点标记树之上的操作和语义约束组成。XML 结点标记树中包括不同类型的结点。XML 数据管理的实现方式可以采用纯 XML 数据库系统的方式。纯 XML 数据库基于 XML 结点树模型，能够较自然地支持 XML 数据的管理。

（3）RDF 数据模型

RDF 是一种用于描述 Web 资源的标记语言，其结构就是由（主语，谓词，宾语）构成的三元组。RDF 也是一种数据模型，并被广泛作为语义网、知识库的基础数据模型。谓词在 RDF 模型中具有特殊的地位，其语义是由谓词符号本身决定的。

2．数据库技术与相关技术相结合

数据库技术与其他计算机技术相结合，是数据库技术的显著特征。

（1）分布式数据库系统

分布式数据库系统是在集中式数据库系统和计算机网络的基础上发展起来的，它是分布式数据处理的关键技术之一。分布式数据库由一组数据组成，这组数据分布在计算机网络的不同计算机上，网络中的每个结点具有独立处理的能力（称为场地自治），可以执行局部应用。同时，每个结点也能通过网络通信系统执行全局应用。

（2）并行数据库系统

①定义

并行数据库系统是在并行机上运行的具有并行处理能力的数据库系统。并行数据库系统能充分发挥多处理和 I / O 并行性，是数据库技术与并行计算技术相结合的产物。

②研究内容

a．实现数据库查询并行化的数据流方法；

b．并行数据库的物理组织；

c．新的并行数据操作算法；

d．查询优化。

3．面向应用领域的数据库新技术

数据库技术被应用到特定的领域中，出现了数据仓库、工程数据库、统计数据库、空间数据库、科学数据库等多种数据库（如图 13-2 所示），使数据库领域的应用范围不断扩大。

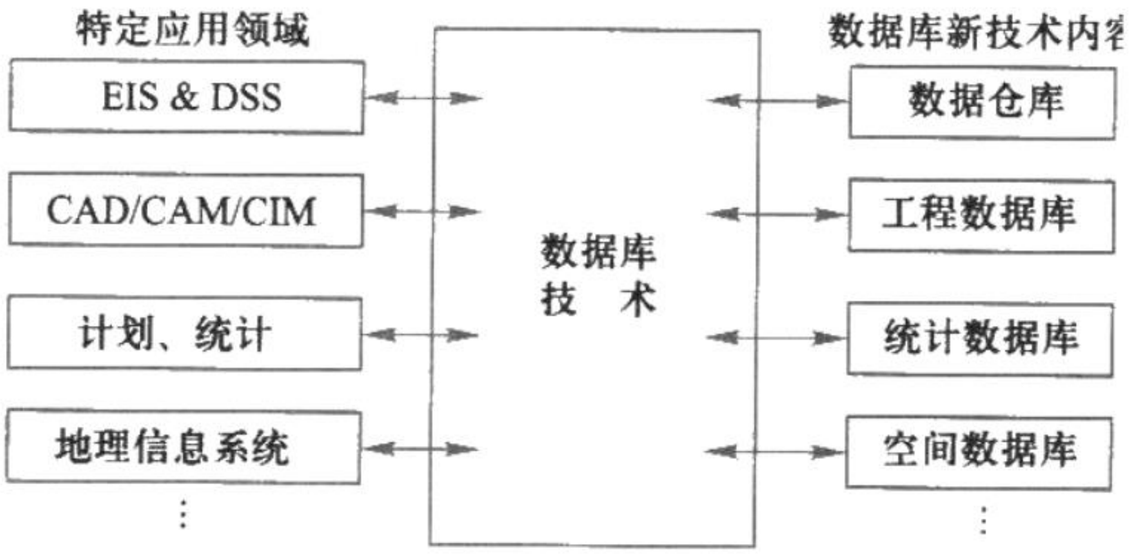


图 13-2　特定应用领域中的数据库技术

（1）工程数据库

①定义

工程数据库（Engineering Data Base，EDB）是一种能存储和管理各种工程设计图形和工程设计文档，并能为工程设计提供各种服务的数据库。

②功能

a．支持复杂对象（如图形数据、工程设计文档）的表示和处理；

b．可扩展的数据类型；

c．支持复杂多样的工程数据的存储和集成管理；

d．支持变长结构数据实体的处理；

e．支持工程长事务和嵌套事务的并发控制和恢复；

f．支持设计过程中多个不同数据版本的存储和管理；

g．支持模式的动态修改和扩展；

h．支持多种工程应用程序。

③实现方式

a．在关系数据库系统的基础上加以扩充或改进；

b．开发支持新数据模型的数据库管理系统。

（2）空间数据库

①用途

空间数据是用于表示空间物体的位置、形状、大小和分布特征等诸方面信息的数据，适用于描述所有二维、三维和多维分布的关于区域的现象。

②特点

a．包括物体本身的空间位置及状态信息；

b．表示物体的空间关系（即拓扑关系）的信息。

③主要内容

a．空间数据模型：描述空间实体和空间实体关系的数据模型，一般传统的数据模型加以扩充和修改来实现，有的用面向对象的数据模型来实现。

b．空间数据查询：包括位置查询、空间关系查询和属性查询。前两种查询是空间数据库特有的查询方式。

c．空间数据库系统：大多数空间数据库系统是以现有的数据库管理系统为基础建立的。d．查询语言：大多是以 SQL 语言为基础，增加相应的函数实现对空间对象和空间关系的查询。

四、数据管理技术的发展趋势

1．数据管理技术面临的挑战

随着数据获取手段的自动化、多样化与智能化，数据量越来越巨大，对于海量数据的存储和管理，要求系统具有高度的可扩展性和可伸缩性，以满足数据量不断增长的需要。传统的分布式数据库和并行数据库在可扩展性和可伸缩性方面明显不足。

2．数据管理技术的发展与展望

大数据给数据管理、数据处理和数据分析提出了全面挑战。传统的关系数据库在系统的伸缩性、容错性和可扩展性等方面难以满足海量数据的柔性管理需求，NoSQL 技术顺应大数据发展的需要，蓬勃发展。NoSQL 是指非关系型的、分布式的、不保证满足 ACID 特性的一类数据管理系统。

NoSQL 技术有如下特点：

（1）对数据进行划分（partitioning），通过大量节点的并行处理获得高性能，采用的是横向扩展的方式（scale out）。

（2）放松对数据的 ACID 一致性约束，允许数据暂时出现不一致情况，接受最终一致性（eventual consistency）。

（3）对各个数据分区进行备份（一般是三份），应对节点可能的失败，提高系统可用性等。

13.1 课后习题详解

1．请阅读本章参考文献。

答：本章参考文献如下：

[1]王珊数据库与信息系统：研究与挑战（1988--2003 研究报告）北京：高等教育出版社，2005.（自 1988 年起，每隔几年，国际上一些资深的数据库专家就会聚集一堂，探讨数据库的研究现状、存在的问题和未来需要关注的新的技术焦点。迄今为止，这样的会议已经举办，6 次。分析这 6 次会议的总结报告能够帮助我们清晰地把握数据库技术进展的脉络，从中也体会到数据库专家们当年的远见卓识。为了让更多的人分享国际资深专家的工作成果，我们将 6 次会议的总结报告做了编译和整理成为本书。本书分为上、下两篇，上篇是对会议报告的编译整理，下篇是会议报告的英语原文，供读者参考。）

[2]BERNSTEIN P，DAYAL U，DEWITT D J，et al．Future Directions in DBMS Research——The Laguna Beach Participants SIGMOD Record，1989，18（I）：17-26.

[3]SILBERSCHATZ A，STONEBRAKER M，ULLMAN J D Database Systems：Achievements and Opportunities CACM，1991，34（10）：110—120

[4]SILBERscHATz A，STONEBRAKER M，ULLMAN J D Database Research，Achievements andOpportunitiesintothe 21 stCentury SIGMODRecord，1996，25（1）：52-63

[5]SILBERSCHATZ A，ZDONIK S B Strategic Directions in Database Systems-Breaking out of the Box ACM Computing Surveys，1996，28（4）：764-778.

[6]BERNSTEIN P，BRODIE M L，CERI S，et al．The Asilomar Report on Database Research SIGMOD Record，l998，27（4）：74—80

[7]ABITEBOUL S，AGRAWAL R，BERNSTEIN P，P，et, al The Lowell Database Research Self-Assessment Meeting Lowell Massachusetts，2003 http：//research microsoft com/～9ray/Iowell

[8]CRA Conference on“Grand Research Challenges”in Computer Science and Engineerin9．http：//www eva ore,/Activities/grand challenges/

[9]DEWlTT D J，GRAY J Parallel Database Systems：the Future of High Performance Database Systems Commun ACM，1992，35（6）：85-98.

[10]李昭原数据库技术新进展．北京：清华大学出版社，1997

[11]李建巾并行关系数据库管理系统引论数据库丛书之一北京：科学出版社，1998

[12]何新贵，唐常杰，李霖．特种数据库技术数据库丛书之一，北京：科学出版社，l999（文献）

[12]主要包括时态数据库技术、移动数据库技术、主动数据庠技术和模糊数据库技术等，每部分都讲解了各领域的基本理论和处理技术，以及国内外有关学者们在相应领域巾的最新研究成果等。

[13]刘惟一，田雯．数据模型数据库技术之一，北京：科学出版社，2001

[14]SHANMUGASUNDARAM J，TuFTE K，ZHANG CHUN，et al Relational Databases for Querying XMLl3 ocuments：Limitations andOpportunities VLDB，1999，302．314

[15]LIU LING，OZSU M T Encyclopedia of Database Systems Springer，2009.

[16]徐俊刚，邵佩英．分布式数据库系统及其应用．3 版北京：科学出版社，2012.

[l7]张效祥，徐家福计算机科学技术百科全书．3 版．北京：清华大学出版社，2014.

[18]覃雄派，王会举，李芙蓉，等数据管理技术的新格局．软件学报，2013，24（2）：175-197

[19]申德荣，于戈，王习特，等支持大数据管理的 NoSQL 系统研究综述．软件学报，2013，24（8）：1786-1803.

[20]A Commullity White Paper Developed by Leading Researchers Across the United States Challenges and Opportunities with Big Data，2012.

2．试述数据库技术的发展过程。数据库技术发展的特点是什么？

答：数据库技术的特点是：

（1）面向对象的方法和技术对数据库发展的影响最为深远

数据库研究人员借鉴和吸收了面向对象的方法和技术，提出了面向对象数据模型（简称对象模型）。该模型克服了传统数据模型的局限性，促进了数据库技术在一个新的技术基础上继续发展。

（2）数据库技术与多学科技术的有机结合

计算机领域中其他新兴技术的发展对数据库技术产生了重大影响。传统的数据库技术和其他计算机技术，如网络通信技术、人工智能技术、面向对象程序设计技术、并行计算技术、移动计算技术等的互相结合、互相渗透，使数据库中新的技术内容层出不穷。

（3）面向应用领域的数据库技术的研究

在传统数据库系统基础上，结合各个应用领域的特点，研究适合该应用领域的数据库技术，如数据仓库、工程数据库、统计数据库、科学数据库、空间数据库、地理数据库等，这是当前数据库技术发展的又一重要特征。

3．试述数据模型在数据库系统发展中的作用和地位。

答：（1）数据模型是数据库系统的核心和基础。

（2）数据库的发展集中表现在数据模型的发展。

4．诸用实例阐述数据库技术与其他计算机技术相结合的成果。

答：数据库技术与其他学科的内容相结合，是新一代数据库技术的一个显著特征，涌现出各种新型的数据库系统（如图 13-3 所示）。例如：

（1）数据库技术与分布处理技术相结合，出现了分布式数据库系统；

（2）数据库技术与并行处理技术相结合，出现了并行数据库系统；

（3）数据库技术与人工智能技术相结合，出现了知识库系统和主动数据库系统；

（4）数据库技术与多媒体技术相结合，出现了多媒体数据库系统；

（5）数据库技术与模糊技术相结合，出现了模糊数据库系统等等。

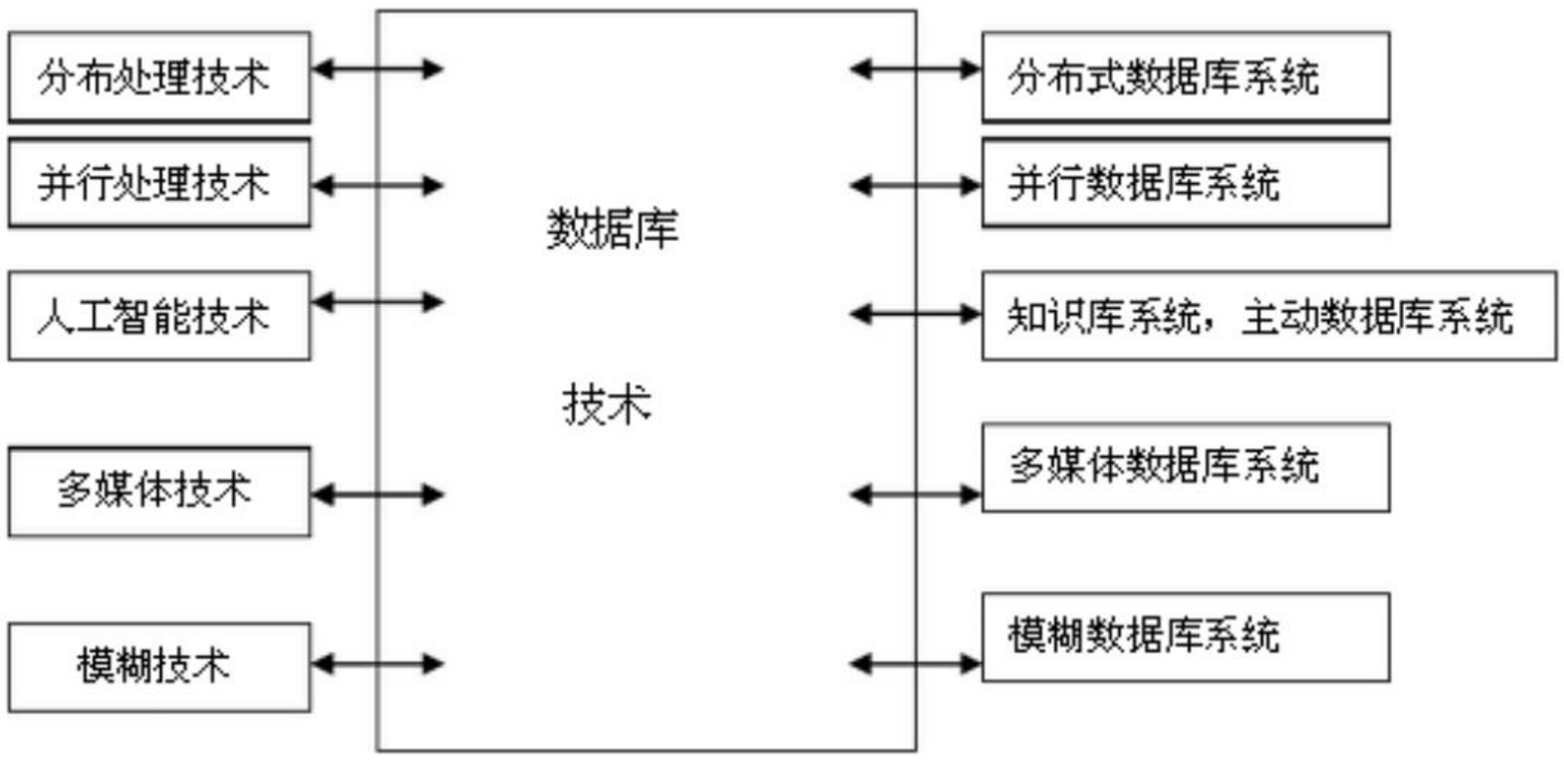


图 13-3　新型数据库系统

第 14 章　大数据管理

14.1　复习笔记

一、大数据概述

1．大数据

（1）定义

大数据是指无法在可容忍的时间内用现有 IT 技术和软硬件工具对其进行感知、获取、管理、处理和服务的数据集合。

专家给出的定义：大数据通常被认为是 PB（10^3TB）或 EB（1EB=1^6TB）或更高数量级的数据，包括结构化的、半结构化的和非结构化的数据。其规模或复杂程度超出了传统数据库和软件技术所能管理和处理的数据集范围。

（2）分类

①海量交易数据（企业 OLTP 应用）

海量交易数据的应用特点是数据海量、读写操作比较简单、访问和更新频繁、一次交易的数据量大大，但要求支持事务 ACID 特性。对数据的完整性及安全性要求高，必须保证强一致性。

②海量交互数据（社交网、传感器、全球定位系统、Web 信息）

海量交互数据的应用特点是实时交互性强，但不要求支持事务特性。其数据的典型特点是类型多样异构、不完备、噪音大、数据增长快，不要求具有强一致性。

③海量处理数据（企业 OLAP 应用）

海量处理数据的应用特点是面向海量数据分析，计算复杂，往往涉及多次迭代完成，追求数据分析的高效率，但不要求支持事务特性。

2．大数据的特征

（1）巨量

大数据的首要特征是数据量巨大，而且在持续、急剧地膨胀。

大规模数据的几个主要来源如下：

①科学研究（天文学、生物学、高能物理等）、计算机仿真领域；

②互联网应用、电子商务领域；

③传感器数据（sensor data）；

④网站点击流数据（click stream data）；

⑤移动设备数据（mobile device data）；

⑥无线射频识别数据（RFID Data）；

⑦传统的数据库和数据仓库所管理的结构化数据也在急速增大。

（2）多样

大数据的多样性通常是指异构的数据类型、不同的数据表示和语义解释。

（3）快变

大数据的快变性也称为实时性，一方面指数据到达的速度很快，另一方而指能够进行处理的时间很短，或者要求响应速度很快，即实时响应。

（4）价值

大数据的价值是潜在的、巨大的。大数据不仅具有经济价值和产业价值，还具有科学价值。这是大数据最重要的特点，也是大数据的魅力所在。

二、大数据的应用

1．感知现在预测未来—互联网文本大数据管理与挖掘

互联网文本大数据管理的特点如下：

（1）互联网文本大数据蕴含着丰富的社会信息，可以看作是对真实社会的网络映射。

（2）实时、深入分析互联网文本大数据，帮助人们在海量数据中获取有价值的信息，发现蕴含的规律，可以更好地感知现在、预测未来，体现了第四范式数据密集型科学发现的研究方式和思维方式。

（3）互联网文本大数据管理对大数据系统和技术的挑战是全面的、跨学科跨领域的，需要创新，也要继承传统数据管理技术和数据仓库分析技术的精华。

2．数据服务实时推荐—基于大数据分析的用户建模

这一类大数据应用的特点如下：

（1）模型的建立来自对大数据的分析结果，通俗地讲是“用数据说话”。建模的过程是动态的，随着实际对象的变化，模型也在变化。

（2）数据处理既有对历史数据的离线分析和挖掘，又有对实时流数据的在线采集和分析，体现了大数据上不同层次的分析：流分析、SQL 分析、深度分析的需求。

（3）用户模型本身也是大数据，维度高，信息稀疏，用户模型的存储、管理是数据服务的重要任务，要满足大规模应用需要的高并发数据更新与读取。

三、大数据管理系统

1．NoSQL 数据管理系统

NoSQL 是以互联网大数据应用为背景发展起来的分布式数据管理系统，它有两种解释：一种是 Non-Relational，即非关系数据库；另一种是 Not Only SQL，即数据管理技术不仅仅是 SQL。NoSQL 系统支持的数据模型通常分为：Key-Value 模型、BigTable 模型、文档（document）。

NoSQL 系统为了提高存储能力和并发读写能力采用了极其简单的数据模型，支持简单的查询操作，而将复杂操作留给应用层实现。该系统对数据进行划分，对各个数据分区进行备份，以应对结点可能的失败，提高系统可用性；通过大量结点的并行处理获得高性能，采用的是横向扩展的方式（scale out）。

2．NewSQL 数据库系统

NewSQL 系统是融合了 NoSQL 系统和传统数据库事务管理功能的新型数据库系统。NewSQL 将 SQL 和 NoSQL 的优势结合起来，充分利用计算机硬件的新技术、新结构，研究与开发了若干创新的实现技术。表 14-1 给出了 SQL 系统、NoSQL 系统与 NewSQL 系统的比较。

表 14-1　SQL 系统、NoSQL 系统与 NewSQL 系统的比较

系统名称	易用性	对事务的支持	扩展性	数据量	成本	代表系统
	操作方式	一致件，并发控制等				
经典关系数据库系统 SQL 系统	易用 SQL	ACID 强一致性	<1000 结点	TB	高	Oracle，DB2，Greenplum 等
NoSQL 系统	Get/Put 等存取原语	弱一致性最终一致性	>10000 结点	PB	低	BigTable，PNUTS，Cloudera 等
NewSQL 系统	SQL	ACID	>10000 结点	PB	低	VoltDB，Spanner 等

3．MapReduce 技术

MapReduce 技术主要应用于大规模廉价集群上的大数据并行处理，是以 key / value 的分布式存储系统为基础，通过元数据集中存储、数据以 chunk 为单位分布存储和数据 chunk 冗余复制来保证其高可用性。MapReduce 是一种并行编程模型。其处理模式以离线式批量处理为主。

MapReduce 存在如下不足：

（1）基于 MapReduce 的应用软件较少，许多数据分析功能需要用户自行开发，从而导致使用成本增加；

（2）程序与数据缺乏独立性；

（3）在同等硬件条件下，MapReduce 的性能远低于并行数据库；

（4）MapReduce 处理连接的性能尤其不尽如人意。

因此，近年来大量研究着手将并行数据库和 MapReduce 两者结合起来，设计兼具两者优点的大数据分析平台。这种架构又可以分为并行数据库主导型、MapReduce 主导型、并行数据库和 MapReduce 集成型，表 14-2 对三种架构进行了对比。

表 14-2　数据库与 MapReduce 的借鉴融合

解决方案	着眼点	代表系统	缺陷
并行数据库主导型	利用 MapReduce 技术来增强开放性，以实现存储和处理能力的可扩展性	Greenplum，Aster Data	规模扩展性有待提高
MapReduce 主导型	学习关系数据库的 SQL 接口及模式支持等，改善易用性	Hive，Pig Latin	性能需要优化
并行数据库和 ManReduce 集成型	集成两者，使两者各自做自己擅长的工作	HadoopDB，Vertica，Teradata	各自的某些优点在集成后有所损耗

4．大数据管理系统的新格局

关系数据管理技术针对自身的局限性，不断借鉴 MapReduce 的优秀思想加以改造和创新，提高管理海量数据的能力。而以 MapReduce 为代表的非关系数据管理技术阵营，从关系数据管理技术所积累的宝贵财富中挖掘可以借鉴的技术和方法，不断解决其性能问题、易用性问题，并提供事务管理能力。

（1）面向操作型应用的关系数据库技术

基于行存储的关系数据库系统、并行数据库系统、面向实时计算的内存数据库系统等，它们具有高度的数据一致性、高精确度、系统的可恢复性等关键特性，同时扩展性和性能也在不断提高，仍然是众多事务处理系统的核心引擎。

（2）面向分析型应用的关系数据库技术

在数据仓库领域，面向 OLAP 分析的关系数据库系统采用了 Shared Nothing 的并行体系架构，支持较高的扩展性。同时，数据库工作者研究了面向分析型应用的列存储数据库和内存数据库。列存储数据库以其高效的压缩、更高的 UO 效率等特点，在分析型应用领域获得了比行存储数据库高得多的性能。

（3）面向操作型应用的 NoSQL 技术

NoSQL 数据库系统相对于关系数据库系统具有两个明显的优势：

①数据模型灵活，支持多样的数据类型（包括图数据）；

②高度的扩展性，很少有一个关系数据库系统部署到超过 1000 个结点的集群上，而 NoSQL 在大规模集群上获得了极高的性能。

（4）面向分析型应用的 MapReduce 技术

系统的高扩展性是大数据分析最重要的需求。MapReduce 并行计算模型框架简单，具有高度的扩展性和容错性，适合于海量数据的聚集计算，成为面向分析型应用的 NoSQL 技术的代表。但是 MapReduce 支持的分析功能有限，具有一定的局限性。

14.1　课后习题详解

1．请阅读本章参考文献。

答：本章参考文献如下：

[1]Sanjay Ghemawat，Howard Gobioff,Shun—TakLeung The Google File System SOSP2003：29-43

[2]Jeffrey Dean，Sanjay Ghemawat MapReduce：Simplified Data Processing on Large Clusters OSDI，2004：137-150

[3]Fay Chang，Jeffrey Dean，Sanjay Ghemawat，et al Big table：A Distributed Storage System for Structured Data OSDI，2006：205—218

[4]覃雄派，王会举，杜小勇，等大数据分析——关系数据库和 MapReduce 技术的竞争、交融和共生软件学报，2011：960—970

[5]王珊，王会举，覃雄派，等，架构大数据：挑战、现状与展望、计算机学报，2011，34（10）：1741-1752

[6]A Community White Paper Developed by Leading Researchers Across the united states．Challenges and Opportunities with Big Data，2012．

（文献[6]是美国数据管理领域 20 余名著名专家研究编写的白皮书《challenges and Oppommities with Big Data》，2012 年 7 月发布。译文：大数据的挑战和机遇，中国人民大学李翠平、王敏峰翻译，发表在科研信息化技术与应用，2013，4（1）：12．18。）

[7]李国杰，大数据研究的科学价值中国计算机学会通讯，2012，8（9）：8-15

[8]周晓方，陆嘉恒，李翠平，等，从数据管理视角看大数据挑战，计算机学会通讯，2012，8（9）：16-20
[9]覃雄派，王会举，李芙蓉，等，数据管理技术的新格局，软件学报，2013，24（2）：175．197．
[10]申德荣，于戈，王习特，等支持大数据管理的 NoSQL 系统研究综述软件学报，2013，24（8）：1786-1803．
[11]张俊，周新，于素华，等，NoSQL 数据管理技术，科研信息化技术与应用，2013，4（1）：3-11．
[12]张延松，探索以 MapReduce 为应用与开发平台的数据库新技术科研信息化技术与应用，2013，4（1）：19-29．
[13]阳振坤，杨传辉，李震海量结构化数据存储管理系统 Ocean Base，科研信息化技术与应用，2013，4（1）：41-48
[141 程学旗，王元卓，靳小龙，网络大数据计算技术与应用综述，科研信息化技术与应用，2013，4（6）：3-14．
[15]Advancing Discovery in Science and Engineering．Computing Community Consortium，2011·
[16]Advancing Personalized Education．Computing Community Consortium，Spring2011．
[17]Smart Health and Well being Computing Community Consortium，Spring2011．
[18]A Sustainable Future．Computing Community Consortium，Summer2011
[19]Drowning in Numbers-Digital Data Will Flood the Planet and Help Us Understand it Better.The Economist,Nov18,2011
http://www economist com/blogs/dailychart/2011/11/big-data-0·
[20]FLOODM，JAGADISHHV，KYLEA，et al Using Data for Systemic Financial Risk Management．Proc．Fifth Biennial Conf．Innovative Data Systems Research，2011
[21]Puttem-Based Strategy：Getting Value from Big Data Gartner Group Press Release，July2011Availableat
http：//www gartner com/it/page jsp？id=1731916．
[22]LAZERD，PENTLANDA，ADAMICL，et al Computational Social Science．Science6 February，2009，323（5915）：721-723．
[23]MANYIKAJ，CHUIM，BROWNB，et al．Big Data：The Next Frontier for Innovation，Competition，and Productivity McKinsey Global Institute，May，2011．
[24]Yuki Noguchi．Following the Bread crumb sto Big Data Gold．National Public Radio，Nov29，2011．
http：//www npr org/2011/11/29/
142521910the-digital-breadcrumbs-that-lead-to-big-data
[25]Yuki Noguchi．The Search for Analysts to Make Sense of Big Data．National Public Radio，Nov·30,2011．
http：//www npr org/2011/11/30/142893065/
the-search-for-analysts-to-make-sense-of-big-data．
[26]Steve Lohr．New York Times，The Age of Big Data．Feb11，2012
http：//www·ny Umescorn/2012/02/12/sun day．review/
big-datas-impact-in-the-world html
[27]Designing a Digital Future：Federally Funded Research and Development in Networking and Information Technology PCAST Report，Dec 2010 Availableat
http：//www white house gov/sites/default/files/microsites/ostp/pcast-nitrd-report-2010 pdf
[28]陆嘉恒．分布式系统及云计算概论，北京：清华大学出版社，2011．

2．什么是大数据，试述大数据的基本特征。

答：（1）大数据是指无法在可容忍的时间内用现有 IT 技术和软硬件工具对其进行感知、获取、管理、处理和服务的数据集合。

（2）大数据的基本特征如下：

①大数据的首要特征是数据量巨大，而且在持续、急剧地膨胀。

②大数据异构的数据类型、不同的数据表示和语义解释多样。

③大数据具有快变性也称为实时性，一方面指数据到达的速度很快，另一方而指能够进行处理的时间很短，或者要求响应速度很快，即实时响应。

④大数据的价值是潜在的、巨大的，大数据不仅具有经济价值和产业价值，还具有科学价值。这是大数据最重要的特点，也是大数据的魅力所在。

3．分析传统 RDBMS 在大数据时代的局限性。

答：关系数据库在大数据时代丧失了互联网搜索这个机会，其主要原因是关系数据库管理系统（并行数据库）的扩展性遇到了前所未有的障碍，不能胜任大数据分析的需求，关系数据管理模型追求的是高度的一致性和正确性，面向超大数据的分析需求。

4．分析传统 RDBMS 的哪些技术应该在非关系数据管理系统中继承和发展。

答：传统 RDBMS 的一致性和 ACID 特性在非关系数据管理系统中继续和发展。

5．什么是 NoSQL，试述 NoSQL 系统在人数据库发展中的作用。

答：（1）NoSQL 是以互联网大数据应用为背景发展起来的分布式数据管理系统，它有两种解释：一种是 Non-Relational，即非关系数据库；另一种是 Not Only SQL，即数据管理技术不仅仅是 SQL。NoSQL 系统支持的数据模型通常分为：Key-Value 模型、BigTable 模型、文档（document）。

（2）NoSQL 系统为了提高存储能力和并发读写能力采用了极其简单的数据模型，支持简单的查询操作，而将复杂操作留给应用层实现。该系统对数据进行划分，对各个数据分区进行备份，以应对结点可能的失败，提高系统可用性；通过大量结点的并行处理获得高性能，采用的是横向扩展的方式（scale out）。

6．什么是 NewSQL，查询相关资料，分析 NewSQL 足如何融合 NoSQL 和 RDBMS 两者的优势的。

答：（1）NewSQL 系统是融合了 NoSQL 系统和传统数据库事务管理功能的新型数据库系统。

（2）NewSQL 将 SQL 和 NoSQL 的优势结合起来，充分利用计算机硬件的新技术、新结构，研究与开发了若干创新的实现技术。

7．描述 MapReduce 的计算过程。分析 MapReduce 技术作为大数据分析平台的优势和不足。

答：（1）MapReduce 技术主要应用于大规模廉价集群上的大数据并行处理，是以 key / value 的分布式存储系统为基础，通过元数据集中存储、数据以 chunk 为单位分布存储和数据 chunk 冗余复制来保证其高可用性。

（2）优势：MapReduce 是一种并行编程模型。其处理模式以离线式批量处理为主。

（3）MapReduce 存在如下不足：

①基于 MapReduce 的应用软件较少，许多数据分析功能需要用户自行开发，从而导致使用成本增加；

②程序与数据缺乏独立性；

③在同等硬件条件下，MapReduce 的性能远低于并行数据库；

④MapReduce 处理连接的性能尤其不尽如人意。

第 15 章　内存数据库系统

15.1　复习笔记

一、概述

1．内存数据库

内存数据库（Main Memory Data Base，MMDB）系统是指将数据库的全部或大部分数据放在内存中的数据库系统。内存数据库有时也称主存数据库，In Memory DataBase 等。

2．磁盘数据库

磁盘数据库（Disk Resident DataBase，DRDB）是使用磁盘作为常规数据存储设备，使用内存作为工作数据缓冲区的数据库系统。

内存数据库与磁盘数据库的区别如图 15-1 所示。

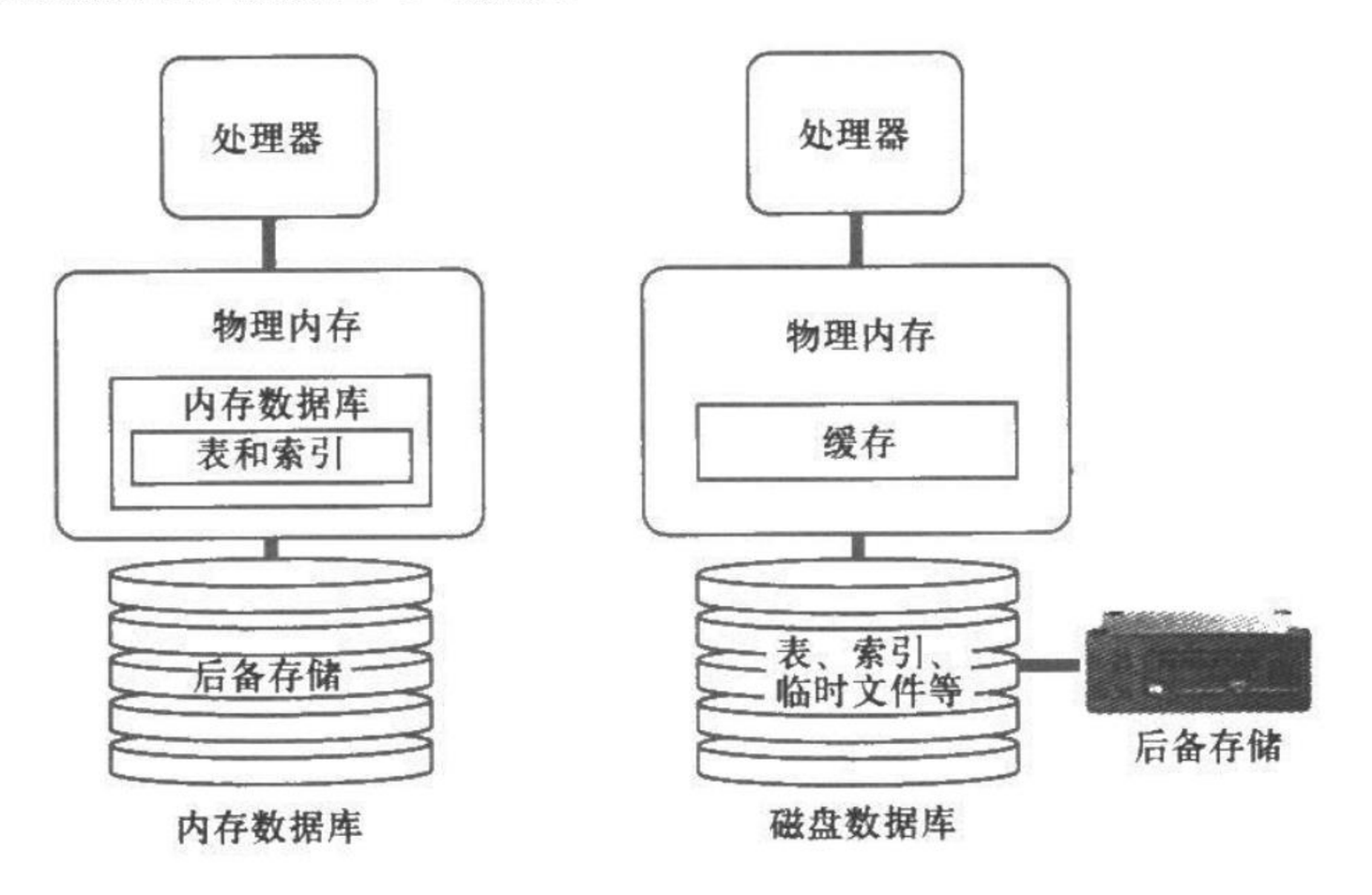


图 15-1　内存数据库和磁盘数据库对比示意图

二、内存数据库的发展历程

1．内存数据库的雏形期

1969 年，IBM 公司研制了国际上最早的层次数据库管理系统 IMS。IMS 在一个系统中提供了两种数据管理方法，一种是采用内存存储的 Fast Path，另一种是支持磁盘存储的 IMS。Fast Path 支持内存驻留数据，是内存数据库的雏形。

2．内存数据库的研究发展期

1984 年，D J．De Witt 等人发表了“内存数据库系统的实现技术”一文，第一次提出了 Main Memory Data Base 的概念；1985 年，IBM 推出了在 IBM 370 上运行的 0BE 内存数据库；1986 年，R B Hagman 提出了使用检查点技术实现内存数据库的恢复机制；1987 年，ACM SIGMOD 会议中有论文提出了以堆文件（heap file）作为内存数据库的数据存储结构；1988 年，普林斯顿大学设计出 TPK 内存数据库；1990 年，普林斯顿大学又设计出 System M 内存数据库。

3．内存数据库的产品成长期

1994 年，美国 0SE 公司推出了第一个商业化的、开始实际应用的内存数据库产品 P01yhedra。

三、内存数据库的特性

内存是计算机存储体系结构中能够被程序可控访问（相对于硬件控制的 cache）的最高层次，是能够提供大量数据存储的最快的存储层。内存数据库具有几个重要特性：

1．高吞吐率和低访问延迟；

2．并行处理能力；

3．硬件相关性。

四、内存数据库的关键技术

通用的内存数据库管理系统要为用户提供 SQL 接口，具有内存存储管理、面向内存的查询处理和优化等基本模块，还应提供多用户的并发控制、事务管理和访问控制，能够保证数据库的完整性和安全性，在内存数据库出现故障时能够对系统进行恢复。

1．数据存储

（1）行存储

在行存储模型中元组是连续存放的，适合事务处理中一次更新多个属性的操作，能够保证对多个属性的操作产生最小的内存访问；但对于只涉及表中相对较少属性的分析处理时，即使该查询只涉及元组的某个或某些属性，其他属性也会被同时从内存读入到缓存，降低了缓存利用率。

（2）列存储

列存储模型将关系按列进行垂直划分，相同属性的数据连续存储。当访问特定属性时只读入所需要的属性所在的分片，所以节省内存带宽，并且具有较高的数据访问局部性，可减少缓存失效，提高数据访问效率；同时列存储将相同类型的数据集中存储，能够更好地对数据进行压缩以减少内存带宽消耗，利用 SIMD（单指令多数据流）技术提高并行处理效率，通过列存储的数据定长化处理支持对数据按偏移位置的访问。

（3）混合存储

该模型把同一元组的所有属性值存储在一页内，在页内对元组进行垂直划分。根据关系的属性个数 m，将每一页划分为 m 个 Mini Page，每个 Mini Page 对应一个属性，连续存放每一页中所有元组的该属性的值。由于元组在页内进行垂直划分，所以该模型具有较好的数据空间局部性，可优化缓存性能；同时，同一元组的值存储在同一页内，所以元组的重构代价比较少。

2．查询处理及优化

内存数据库的查询处理性能主要由两个因素决定：内存数据访问性能和内存数据处理性能。内存数据访问性能由内存带宽和内存访问延迟决定。

（1）面向 cache 特性的查询处理与优化技术

①内存数据库和磁盘数据库的比较

内存数据库的基础假设是数据库的工作数据集常驻于内存中（memory resident），从而消除了传统磁盘数据库的 I / O 代价，内存数据库的性能较磁盘数据库有数十倍甚至数百倍的提升。

磁盘数据库使用内存缓冲区（buffer）来优化 I / O 代价，现代 CPU 使用硬件级的多级 cache 机制优化内存访问，内存数据库的内存访问优化由硬件级的 cache 机制来完成，采用类 LRU（最近最少访问）替换算法实现 cache 中的数据管理。

②cache 失效的分类

a．强制（compulsory）失效。强制失效是数据首次访问时在 cache 中所产生的失效，是内存数据访问不可避免的。

b．容量（capacity）失效。容量失效是由于工作数据集超过 cache 容量大小而导致的数据访问时在 cache 中的失效。

c．冲突（conflict）失效。冲突失效则是在 cache 容量充足时由于大量弱局部性数据（一次性访问数据或复用周期很长的数据）将强局部性数据（频繁使用的数据集）驱逐出 cache 而在对强局部性数据重复访问时产生的 cache 失效。

③cache 优化技术的分类

a．cache．conscious 优化技术；

b．cache．oblivious 优化技术；

c．page—coloring 优化技术。

④数据存放技术

采用聚类（clustering）和染色（coloring）存放技术，优化了 cache 的性能。

（2）索引技术

索引是数据库中提高查询性能的有效方法，在磁盘数据库中广泛使用的 hash 索引、B+树索引等不适合内存数据库的需求，所以一些针对内存数据库特性的索引得到了广泛的研究。

（3）面向多核的查询处理技术

在多核平台上，查询算法需要改写为多核并行算法，将串行操作符并行化。在多核并行优化时需要解决的关键技术包括并行处理时的共享 cache 优化，数据分区优化等技术。常见的三种多核并行 hash 连接技术为：

①无分区（no partitioning）hash 连接算法；

②基于分区（partitioned）的 hash 连接算法；

③radix hash 连接算法。

（4）面向众核的查询处理技术

多核处理器逐渐进入众核时代。内存数据库面临越来越多的计算核心，查询算法需要进化为高可扩展并行算法，以充分利用先进众核处理器提供的强大并行计算性能。数据库采用以存储为中心的设计思想，内存数据库的优化技术也一直以内存数据访问优化为核心。

3．并发与恢复

（1）并发控制

①减少锁的开销的方法

a．采用较大的封锁粒度（如表级锁）；

b．采用乐观加锁方式；

c．减少锁的类型；

d．将锁信息存储在数据本身。

②串行的并发控制协议特点

写事务在整个数据库上施加互斥锁（mutex），通过时间戳和互斥锁在事务的提交记录没有到达磁盘之前允许新事务开始，并且保证任何提交的读事务不会读到未提交的数据。

（2）恢复机制

由于内存的脆弱性和易失性，内存数据库中数据容易被破坏和丢失，所以内存数据库数据需要在磁盘等非易失性存储介质中进行备份，并且在对数据更新时将日志写到非易失性存储介质中。

①TimesTen 记录日志的方式

a．将日志记在内存的一个区域中；

b．可以将日志记在磁盘文件上。

②RAMCloud 记录日志的方式

RAMCloud 采用了一种主从式内存日志机制，即当数据被修改时将更新日志写入两个或更多的内存后备服务器。日志首先存储于后备服务器的内存中，再采用异步方式批量写入磁盘。

③恢复数据的步骤

a．首先恢复热点数据，即执行事务所必须的数据；

b．在后台恢复其他非热点数据。

15.2　课后习题详解

1．内存数据库和磁盘数据库有什么区别?

答：内存数据库与磁盘数据库的区别如图 15-2 所示。

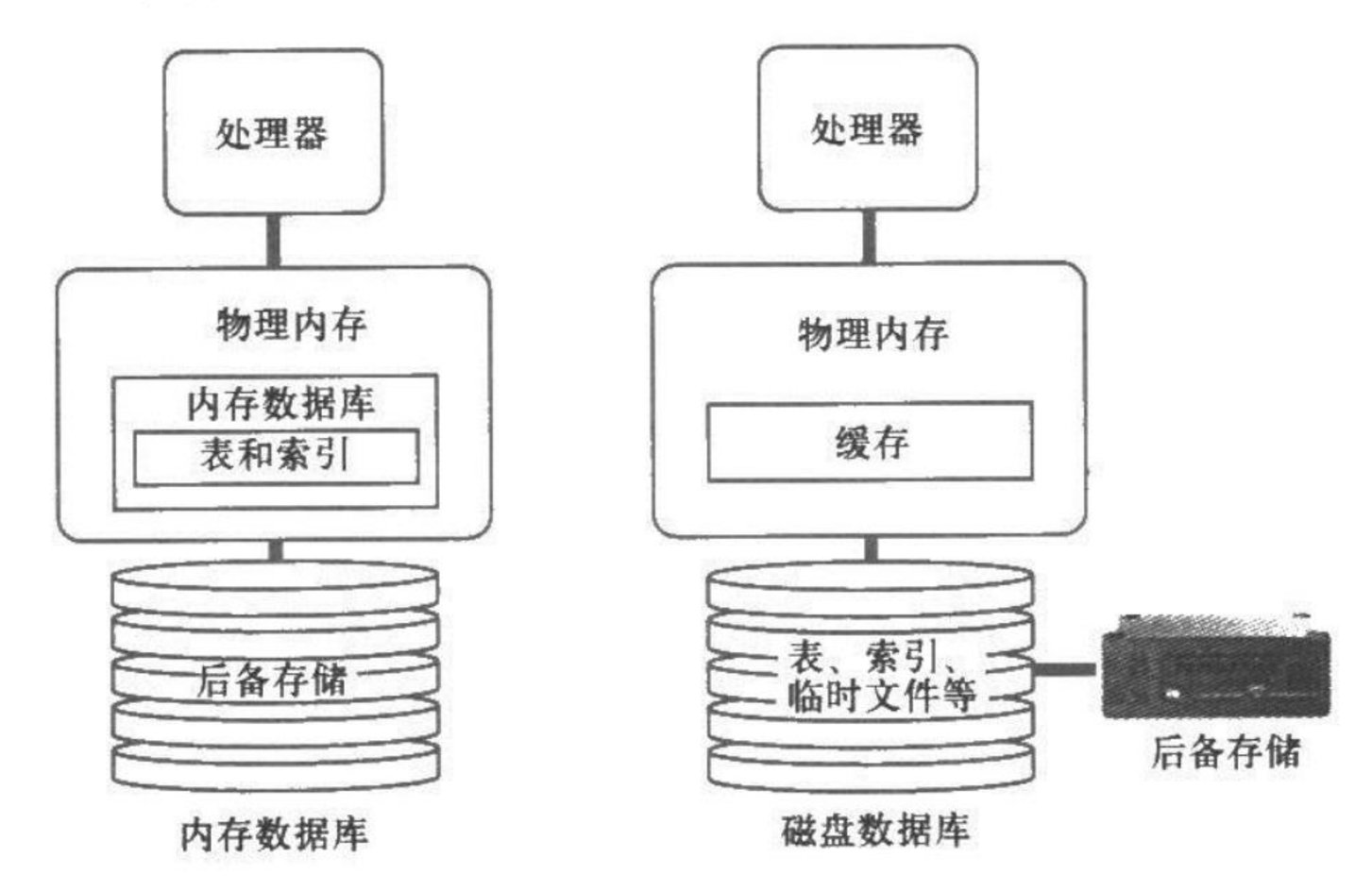


图 15-2　内存数据库和磁盘数据库对比示意图

2．内存数据库的特点有哪些?

答：内存是计算机存储体系结构中能够被程序可控访问（相对于硬件控制的 cache）的最高层次，是能够提供大量数据存储的最快的存储层。内存数据库具有几个重要特性：

（1）高吞吐率和低访问延迟；

（2）并行处理能力；

（3）硬件相关性。

3．试述内存数据库和硬件的相关性。哪些硬件技术影响内存数据库的性能?

答：（1）内存数据库和硬件的相关性：内存数据库的性能受硬件特性的直接影响。计算机硬件技术的发展主要体现在高端计算设备和存储设备上，如多核处理器、众核协处理器（Many Integrated Core，MIC）、通用 GPU、PCM 存储（Phase Change Memory，相变存储）、固态硬盘（solid State Disk，SSD）存储等。这些计算能力和存储性能的提升有助于内存吞吐率需求的提升（众核技术）、提高内存持久存储能力（PCM 技术）或为内存提供二级存储（SSD 技术）。硬件技术在多核及众核处理器、高性能存储和高速网络等方面的发展为内存数据库提供了高并行处理、高性能存储访问以及高速连通的硬件平台。内存数据库的设计应该充分考虑并有效利用由新硬件技术带来的功能扩展和性能提高。

（2）大容量内存、flash、PCM 存储、多核 CPU、众核处理器、高性能网络传输等硬件技术的发展为内存数据库提供了良好的平台，直接影响到内存数据库的性能。

4．大数据时代对内存数据库提出了哪些挑战?

答：大数据的特点有：数据量大（Volume）、类型繁多（Variety）、价值密度低（Value）、速度快时效高（Velocity）。随着大数据时代的到来，未来众核协处理器、通用计算图形处理器（General Purpose Graphic Unit，GPGPU）等新的高性能计算平台进入数据库领域，同时也对内存数据库提出了更多挑战。

（1）查询处理与优化，主要针对大数据数据量大和类型繁多的特点。page—coloring 优化技术对于数据持久驻留内存的内存数据库来说，较大的弱局部性数据集往往需要预先分配较大的内存地址范围，而较少的 page color 对应的地址范围较小，难以满足大数据集存储的要求。

（2）实时分析处理性能，主要针对大数据速度快时效高的特点。内存数据库摆脱了 I / O 延迟之后，内存访问速度得到极大的提升，在新兴的非易失性内存，如 PCM 等技术支持下，内存计算和更新的速度进一步提升。事务型内存数据库的一个技术发展趋势是将事务串行化，简化并发控制机制，提高内存数据库代码执行效率，使串行处理性能能够满足高吞吐性能需求。分析型内存数据库则将计算最大化并行，以提高多核处理器的并行计算效率，提高应对内存大数据实时分析处理的性能需求。

（3）并发与恢复技术，主要针对大数据数据量大和价值密度低的特点。通过对大数据表的共享扫描减少并发查询时独立大表扫描所产生的 cache 缺失，主要通过查询分组以及查询操作符批处理技术实现共享扫描基础上的高并发查询处理。

第 16 章　数据仓库与联机分析处理技术

16.1　复习笔记

一、数据仓库技术

1．操作型与分析型数据区别

操作型数据与分析型数据之间的区别如表 16-1 所示。

表 16-1　操作型数据和分析型数据的区别

操作型数据	分析型数据
细节的	综合的，或提炼的
在存取瞬间是准确的	代表过去的数据
可更新	不可更新
操作需求事先可知道	操作需求事先不知道
生命周期符台软件开发生命周期（SDLC）	完全不同的生命周期
对性能要求高	对性能要求宽松
个时刻操作一个元组	一个时刻操作一个集合
事务驱动	分析驱动
面向应用	面向分析
一次操作数据量小	一次操作数据量大
支持日常操作	支持管理决策需求

数据仓库是一个用以更好地支持企业（或组织）决策分析处理的、面向主题的、集成的、不可更新的、随时间不断变化的数据集合。数据仓库本质上和数据库一样，是长期储存在计算机内的、有组织、可共享的数据集合。

2．数据仓库的基本特征

（1）主题与面向主题

主题是一个抽象的概念，是在较高层次上将企业信息系统中的数据综合、归类并进行分析利用的抽象。面向主题的数据组织方式是根据分析要求将数据组织成一个完备的分析领域，即主题域。

主题域应该具有以下两个特点：

①独立性；

②完备性。

（2）数据仓库是集成的

加工和集成数据的步骤包括：

①统一原始数据中所有矛盾之处；

②将原始数据结构作一个从面向应用到面向主题的大转变；

③进行数据综合和计算。

数据综合工作可以在抽取数据时生成，也可以在进入数据仓库以后进行综合时生成。

（3）数据仓库是不可更新的

OLTP 数据库中的数据经过抽取、清洗、转换和装载存放到数据仓库中（简称 ECTL）。一旦数据存放到数据仓库中，数据就不再更新了。

（4）数据仓库是随时间变化的

数据仓库的数据是随时间的变化不断变化的，这一特征表现在以下三方面：

①数据仓库随时间变化不断增加新的数据内容；

②数据仓库随时间变化不断删去旧的数据内容；

③数据仓库数据的码键都包含时间项，以标明数据的历史时期。

3．数据仓库中的数据组织

数据仓库中的数据分为多个级别：早期细节级、当前细节级、轻度综合级、高度综合级。数据仓库的数据组织结构如图 16-1 所示。

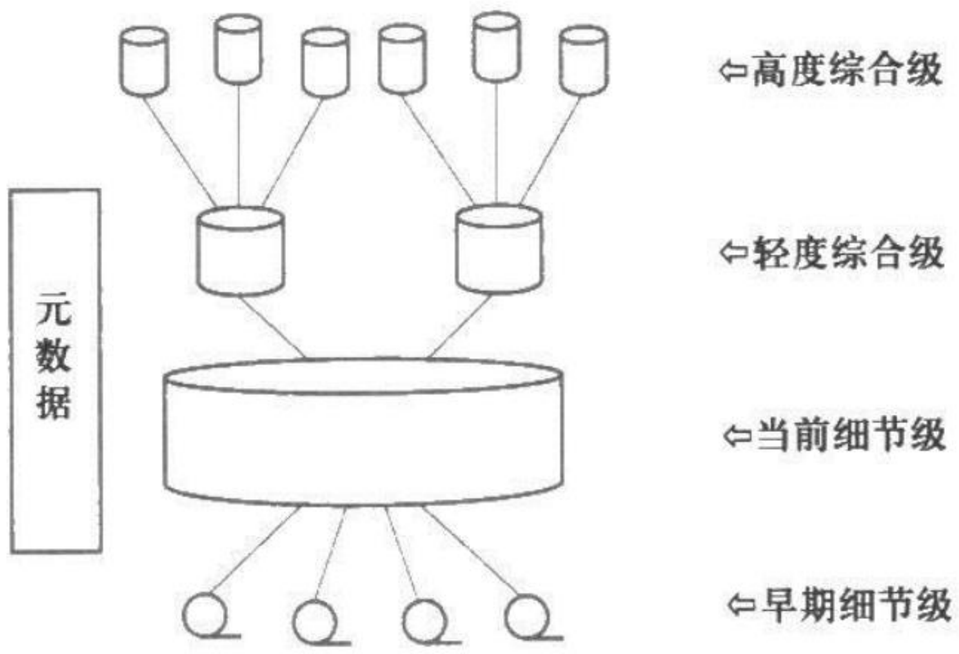
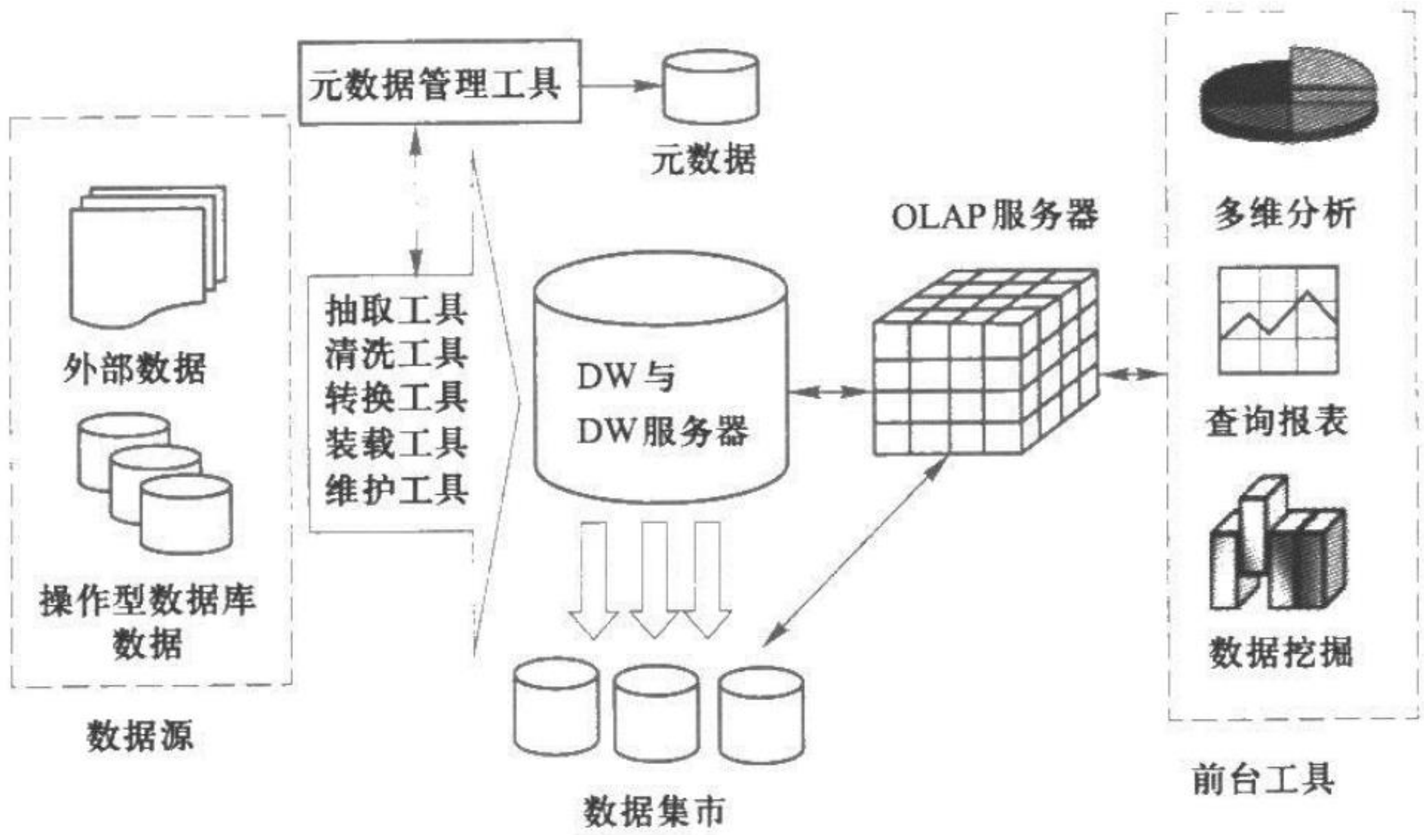


图 16-1　数据仓库的数据组织结构

粒度是指数据仓库中数据具有的不同综合级别。粒度越大，表示细节程度越低，综合程度越高。多重粒度的数据组织可以大大提高联机分析的效率。不同粒度的数据可以存储在不同级别的存储设备上。

4．数据仓库系统的体系结构

数据仓库系统架构由数据仓库的后台工具、数据仓库服务器、OLAP 服务器和前台工具组成，如图 16-2 所示。



16-2　数据仓库系统的体系结构图

（1）后台工具

后台工具包括数据抽取、清洗、转换、装载和维护工具。

（2）数据仓库服务器

数据仓库服务器相当于数据库系统中的 DBMS，它负责管理数据仓库中数据的存储管理和数据存取，并给 OLAP 服务器和前台工具提供存取接口。

（3）OLAP 服务器

OLAP 服务器透明地为前台工具和用户提供多维数据视图。OLAP 服务器则必须考虑物理上这些分析数据的存储问题。

（4）前台工具

前台工具包括查询报表工具、多维分析工具、数据挖掘工具和分析结果可视化工具等。

二、联机分析处理技术

联机分析处理是以海量数据为基础的复杂分析技术。

1．多维数据模型

（1）定义

多维数据模型是数据分析时用户的数据视图，是面向分析的数据模型，用于给分析人员提供多种观察的视角和面向分析的操作。

（2）表示

多维数据模型的数据结构可以用这样的一个多维数组来表示：(维 1，维 2，…，维 n，度量值)。多维数组还

可以用多维立方体 CUBE 来表示。

2．多维分析操作

常用的 OLAP 多维分析操作有切片、切块、旋转、向上综合、向下钻取等。

3．OLAP 的实现方式

OLAP 服务器透明地为分析软件和用户提供多维数据视图，实现对多维数据的存储、索引、查询和优化等等。按照多维数据模型的不同实现方式，有 MOLAP 结构、ROLAP 结构、HOLAP 结构等多种结构。

（1）MOLAP 结构

MOLAP 结构直接以多维立方体 CUBE 来组织数据，以多维数组来存储数据，支持直接对多维数据的各种操作。

（2）ROLAP 结构

ROLAP 结构用 RDBMS 或扩展的 RDBMS 来管理多维数据，用关系的表来组织和存储多维数据。同时，它将多维立方体上的操作映射为标准的关系操作。

①事实表和维表

ROLAP 将多维立方体结构划分为两类表，一类是事实(fact)表，另一类是维表。

a．事实表用来描述和存储多维立方体的度量值及各个维的码值；

b．维表用来描述维信息。

②表示形式

ROLAP 用关系数据库的二维表来表示事实表和维表，即 ROLAP 用“星形模式”和“雪花模式”来表示多维数据模型。

（3）星形模式

星形模式通常由一个中心表（事实表）和一组维表组成。

事实表一般很大，维表一般较小。

（4）雪花模式

雪花模式是对维表按层次进一步细化后形成的。在“星形”维表的角上出现了分支，这样变形的星形模式被称为“雪片模式”。

三、数据挖掘技术

1．概述

（1）定义

数据挖掘是从大量数据中发现并提取隐藏在内的、人们事先不知道的但又可能有用的信息和知识的一种新技术。

（2）目的

数据挖掘的目的是帮助决策者寻找数据间潜在的关联，发现经营者被忽略的要素，而这些要素对预测趋势、决策行为也许是十分有用的信息。

（3）涉及学科

数据挖掘技术涉及数据库技术、人工智能技术、机器学习、统计分析等多种技术，它使决策支持系统(DSS)跨入了一个新阶段。

2．数据挖掘和传统分析方法的区别

（1）传统的 DSS 系统通常是在某个假设的前提下通过数据查询和分析来验证或否定这个假设。

（2）数据挖掘是在没有明确假设的前提下去挖掘信息，发现知识。

（3）数据挖掘技术是基于大量的来自实际应用的数据，进行自动分析、归纳推理，从中发掘出数据间潜在的模式。

数据挖掘所得到的信息应具有事先未知、有效和实用 3 个特征。

3．数据挖掘的数据源

（1）来源

数据挖掘的数据主要有两种来源：

①从数据仓库中来；

②直接从数据库中来。

（2）数据来源于数据仓库的好处

数据仓库的数据已经过了预处理，许多数据不一致的问题都较好地解决了，在数据挖掘时大大减少了清理数

据的工作量。

（3）数据来源于数据仓库的坏处

需要建立一个巨大的数据仓库，要解决所有的数据冲突问题，花费巨资。

4．数据挖掘的功能

数据挖掘的功能主要有以下六种：

（1）概念描述

指归纳总结出数据的某些特征。

（2）关联分析

若两个或多个变量的取值之间存在某种规律性，就称为关联。

（3）分类和预测

找到一定的函数或者模型来描述和区分数据类之间的区别，用这些函数和模型对未来进行预测。分类的结果表示为决策树、分类规则或神经网络。

（4）聚类

将数据分为多个类，使得类内部数据之间的差异最小，而类之间数据的差异最大。聚类技术主要包括传统的模式识别方法和数学分类学等。

（5）孤立点的检测

孤立点是指数据中的整体表现行为不一致的那些数据集合。

（6）趋势和演变分析

描述行为随着时间变化的对象所遵循的规律或趋势。

一个典型的数据挖掘系统的体系结构如图 16-3 所示。

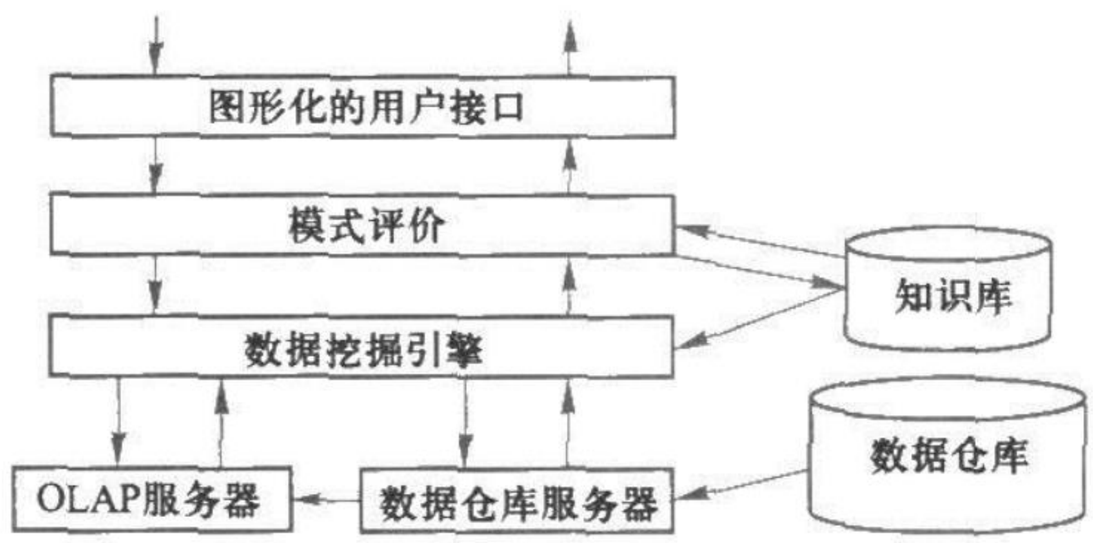


图 16-3　典型的数据挖掘系统的体系结构

四、大数据时代的新型数据仓库

1．系统需求的变化

和传统的数据仓库相比，大数据时代 BI 系统面临的需求发生了如下变化：

（1）数据量急剧增长；

（2）数据类型多样；

（3）决策分析复杂；

（4）底层硬件环境变化。

2．传统数据仓库所面临的问题

（1）数据移动代价过高；

（2）不能快速适应变化。

3．大数据时代的新型数据仓库

为了应对大数据时代系统在数据量、数据类型、决策分析复杂度和底层硬件环境等方面的变化，以较低的成本高效地支持大数据分析，新型的数据仓库解决方案需具备表 16-2 所示的几个重要特性。

表 16-2　大数据分析平台需具备的特性

特性	简要说明
高度可扩展	横向大规模可扩展，大规模并行处理
高性能	快速响应复杂查询与分析
高度容错性	查询失败时，只需重做部分工作
支持异构环境	对硬件平台一致性要求不高，适应能力强
较低的分析延迟	业务需求变化时，能快速反应
易用且开放接口	既能方便查询，又能处理复杂分析
较低成本	较高的性价比
向下兼容性	支持传统的 BI 工具

16.2　课后习题详解

1．数据仓库的 4 个基本特征是什么？

答：数据仓库的 4 个基本特征如下：

（1）数据仓库的数据是面向主题的。

主题是一个抽象的概念，是在较高层次上将企业信息系统中的数据综合、归类并进行分析利用的抽象。面向主题的数据组织方式是根据分析要求将数据组织成一个完备的分析领域，即主题域。

（2）数据仓库的数据是集成的。

操作型数据与分析型数据之间差别甚大，数据仓库的数据是从原有分散的数据库数据中抽取来的，因此数据在进入数据仓库之前，必然要经过加工与集成，统一与综合。

（3）数据仓库的数据是不可更新的。

OLTP 数据库中的数据经过抽取、清洗、转换和装载存放到数据仓库中（简称 ECTL）。一旦数据存放到数据仓库中，数据就不再更新了。

（4）数据仓库的数据是随时间变化的。

数据仓库的数据是随时间的变化不断变化的，这一特征表现在以下三方面：

①数据仓库随时间变化不断增加新的数据内容；

②数据仓库随时间变化不断删去旧的数据内容；

③数据仓库数据的码键都包含时间项，以标明数据的历史时期。

2．操作型数据和分析型数据的主要区别是什么？

答：操作型数据和分析型数据的主要区别包括以下几个方面：

（1）操作型数据是细节的，而分析型数据是综合的或提炼的；

（2）操作型数据在存取瞬间是准确的，而分析型数据代表过去的数据；

（3）操作型数据可更新，分析型数据不可更新；

（4）操作型数据操作需求事先可知道，而分析型数据操作需求事先不知道；

（5）操作型数据生命周期符合 SDLC，而分析型数据生命周期完全不同；

（6）操作型数据对性能要求高，而分析型数据对性能要求宽松；

（7）操作型数据一个时刻操作一元组，而分析型数据一个时刻操作一集合；

（8）操作型数据是事物驱动的，而分析型数据是分析驱动的；

（9）操作型数据是面向应用的，而分析型数据是面向分析的；

（10）操作型数据一次操作数据量小，而分析型数据一次操作数据量大；

（11）操作型数据支持日常操作，分析型数据支持管理决策需求。

3．在基于关系数据库的联机分析处理实现中，举例说明如何利用关系数据库的二维表来表达多维概念。

答：如图 16-4 所示的星形模式的中心是销售事实表，其周围的维表有时间维表、顾客维表、销售员维表、制造商维表和产品维表。

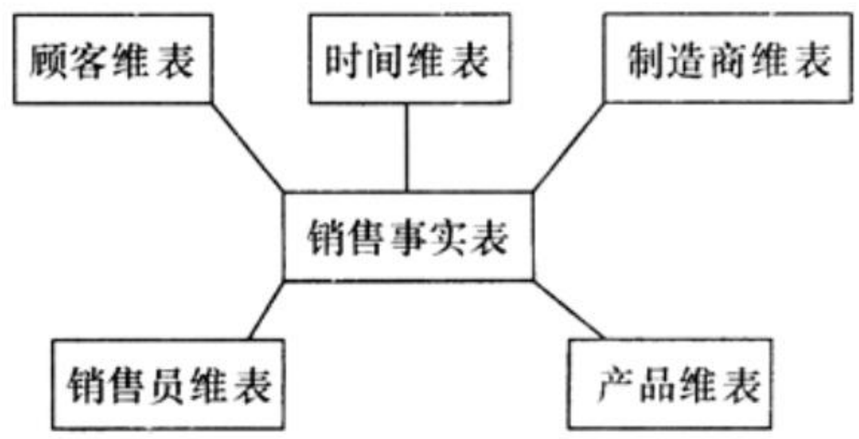


图 16-4　星形模式表

如果对上图星状模式中的维表按照层次进一步细化：顾客维表可以按照所在地区的位置分类聚集；时间维表可以有两类层次——日、月，日、星期；制造商维表可以按照工厂且工厂按照所在地区分层。这样就形成了如图 16-5 所示的雪花模式。

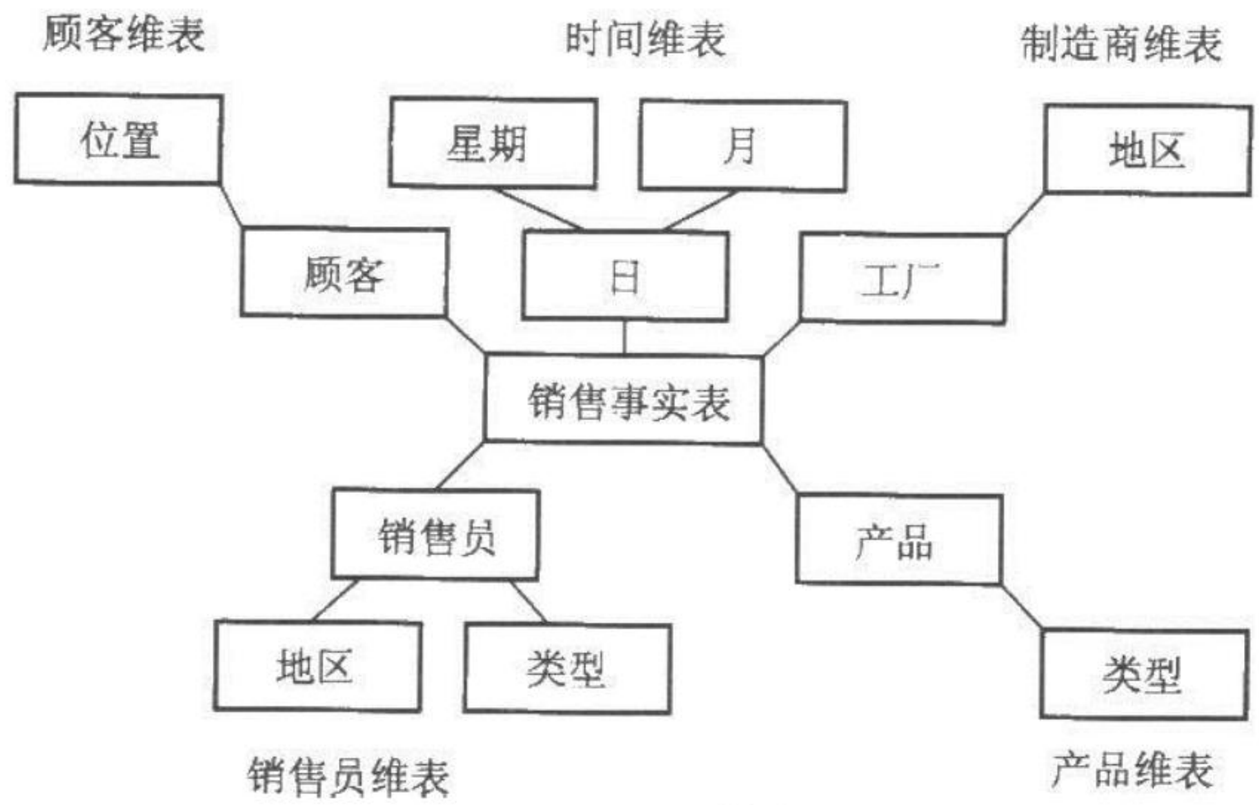


图 16-5　雪花模式表

4．数据挖掘和传统的分析方法的主要不同是什么？

答：（1）数据挖掘技术是从大量数据中发现并提取隐藏的、人们事先未知的但又可能有用的信息和知识的一种新技术，是在没有明确假设的前提下去挖掘信息，发现知识。

（2）传统的 DSS 系统通常是在某个假设的前提下，通过数据查询和分析来验证或否定这个假设。

5．大数据时代传统的数据仓库系统面临哪些问题?如何应对这些挑战?

答：（1）传统数据仓库所面临的问题：

①数据移动代价过高；

②不能快速适应变化。

（2）为了应对这些挑战，以较低的成本高效地支持大数据分析，新型的数据仓库解决方案需具备表 16-3 所示的几个重要特性。

表 16-3　新型数据仓库需具备的特性

特性	简要说明
高度可扩展	横向大规模可扩展，大规模并行处理
高性能	快速响应复杂查询与分析
高度容错性	查询失败时，只需重做部分工作
支持异构环境	对硬件平台一致性要求不高，适应能力强
较低的分析延迟	业务需求变化时，能快速反应
易用且开放接口	既能方便查询，又能处理复杂分析
较低成本	较高的性价比
向下兼容性	支持传统的 BI 工具