

数据库系统原理

软件开发环境国家重点实验室

郎波

Tel: 82317656

E-Mail: langbo@buaa.edu.cn

数据库系统示例



The screenshot displays a web browser window with the title '教务管理系统' (Academic Management System). The page features a navigation bar with tabs for '教学任务' (Teaching Tasks) and '成绩管理' (Grade Management). The '教学任务' tab is active, showing a sub-menu with '教学任务查询' (Teaching Task Query), '课表查询' (Class Schedule Query), '课表查询 (英文)' (Class Schedule Query (English)), and '任课考试信息查询' (Instructor Exam Information Query).

The main content area shows the current location: '当前位置: 教学任务 >> 教学任务查询'. Below this, there is a dropdown menu for '学年学期' (Academic Year/Quarter) set to '2013-2014-1' and a '查询' (Query) button.

The table below lists the courses:

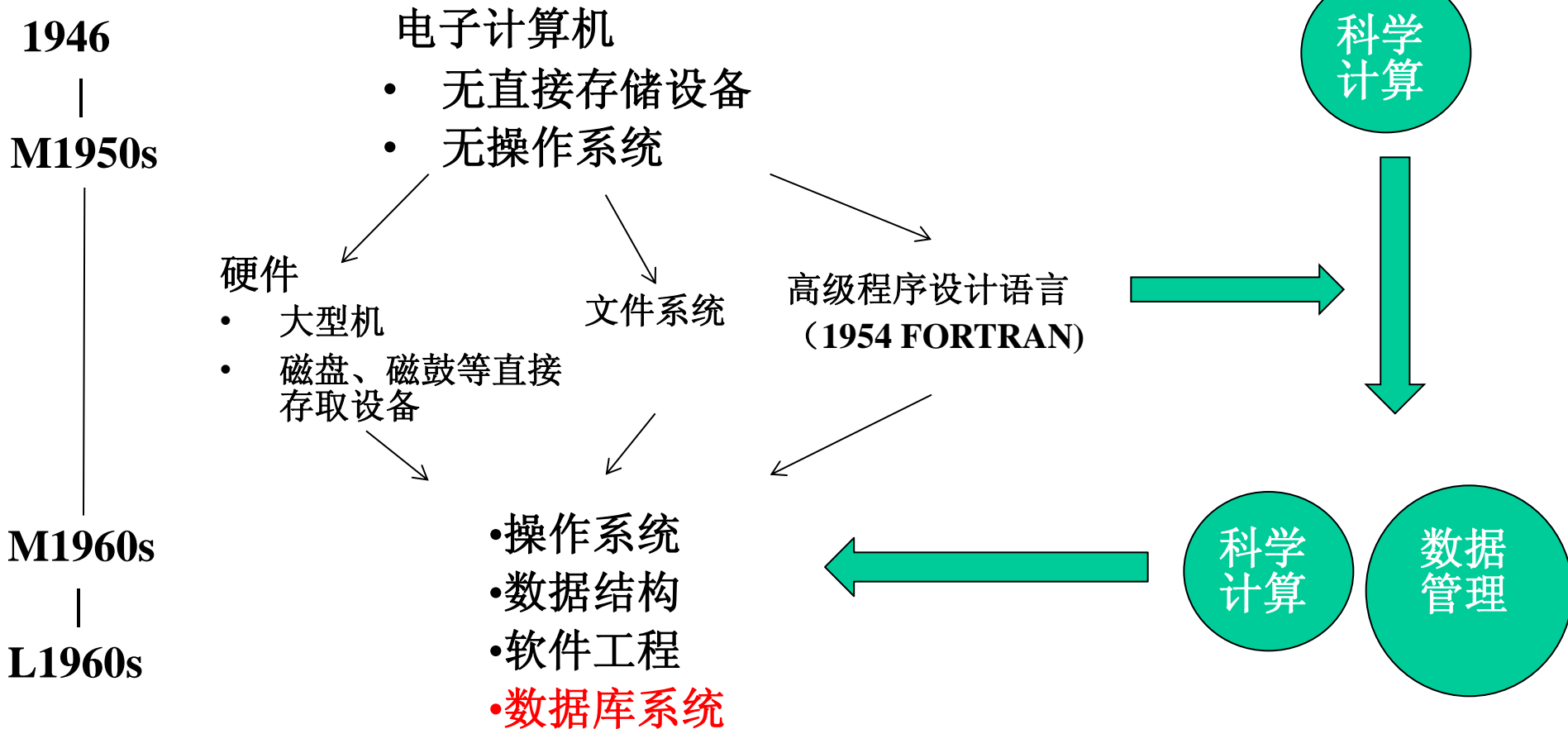
课程代码	课程名称	学分	总学时	理论学时	实验学时	上机学时	上课时间	教师	职称	班级	选课人数	课程性质	下载点名册
80083130	数据库系统原理	3	48	48	0	0	周一: 3, 4节, 周五: 3, 4节	阮波		110611 110612 110613	105	必修	下载点名册
80083130	数据库系统原理	3	48	48	0	0	周一: 3, 4节, 周五: 3, 4节	阮波		110641 110642	19	选修	下载点名册
90083820	数据库原理课程设计	1	36	0	0	36		阮波		110611 110612 110613	104	必修	下载点名册
90083820	数据库原理课程设计	1	36	0	0	36		阮波		110641 110642	18	选修	下载点名册

At the bottom of the page, there is a pagination bar showing '共 4 条 每页 10 条 当前第 1 页 共 1 页 转到 1 页 GO 每页 10 条'.

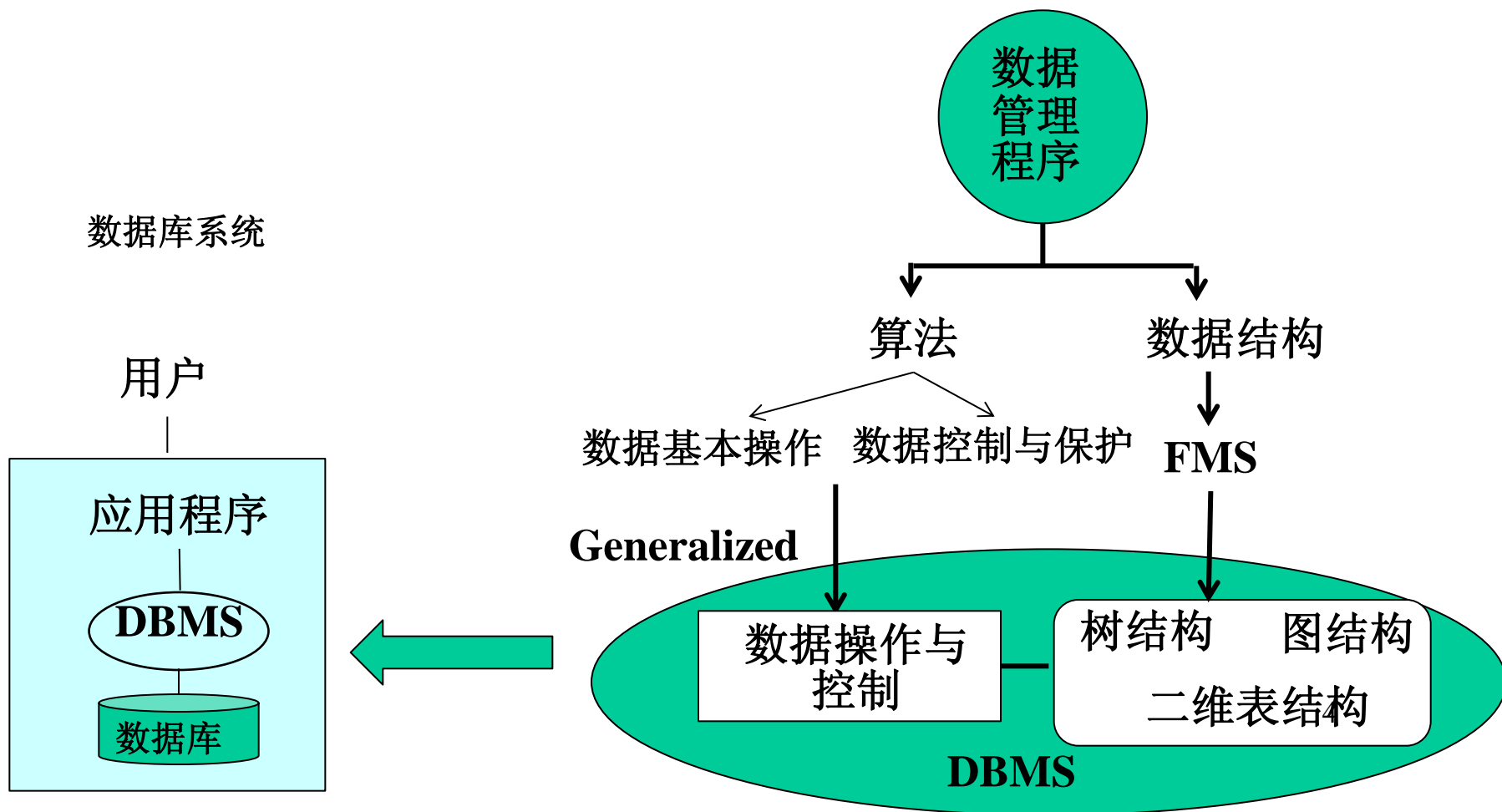
数据库技术的产生背景

计算机科学与技术的发展

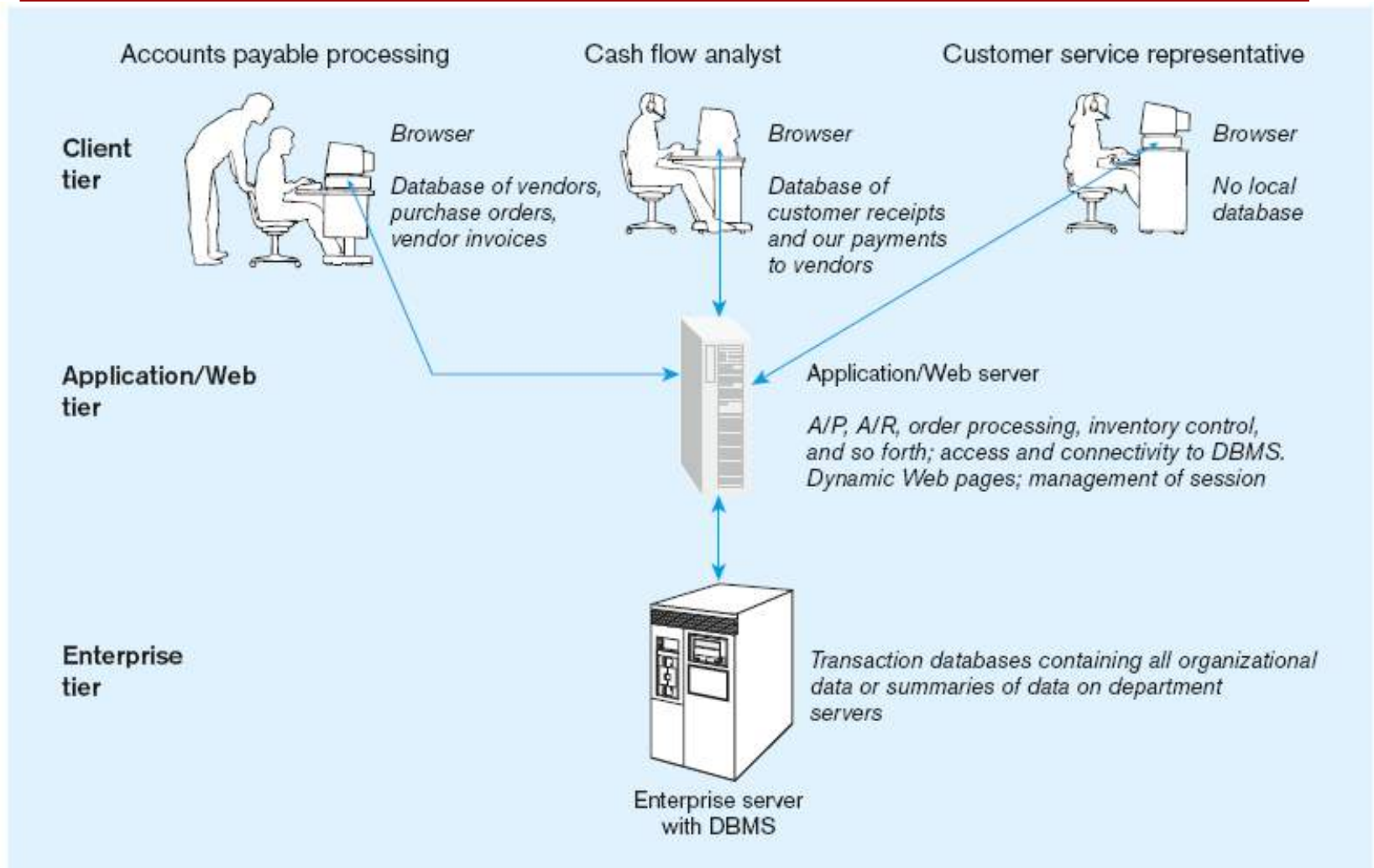
应用的发展



什么是数据库系统



数据库系统示例(3)



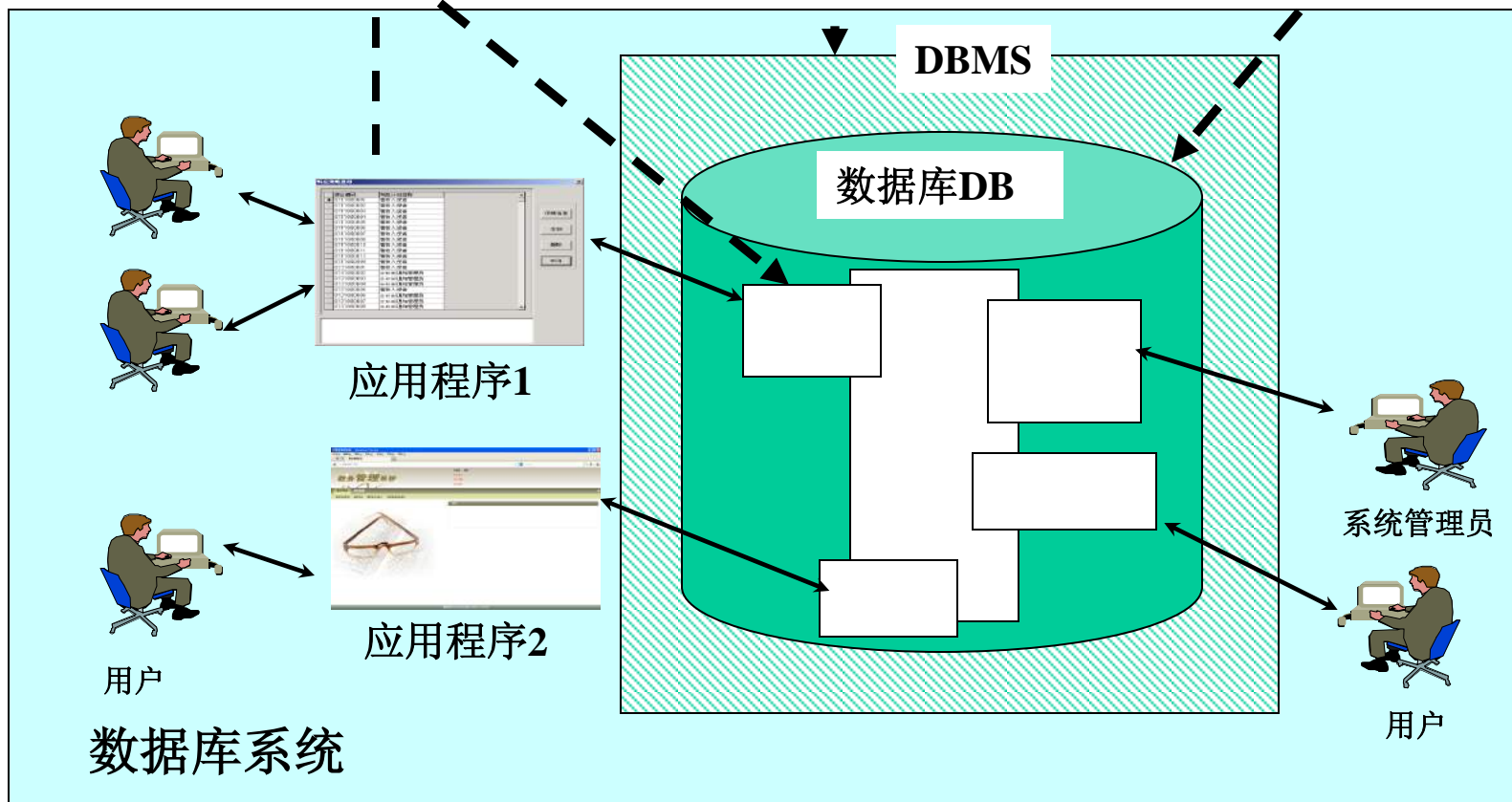
数据库系统原理

数据库系统原理

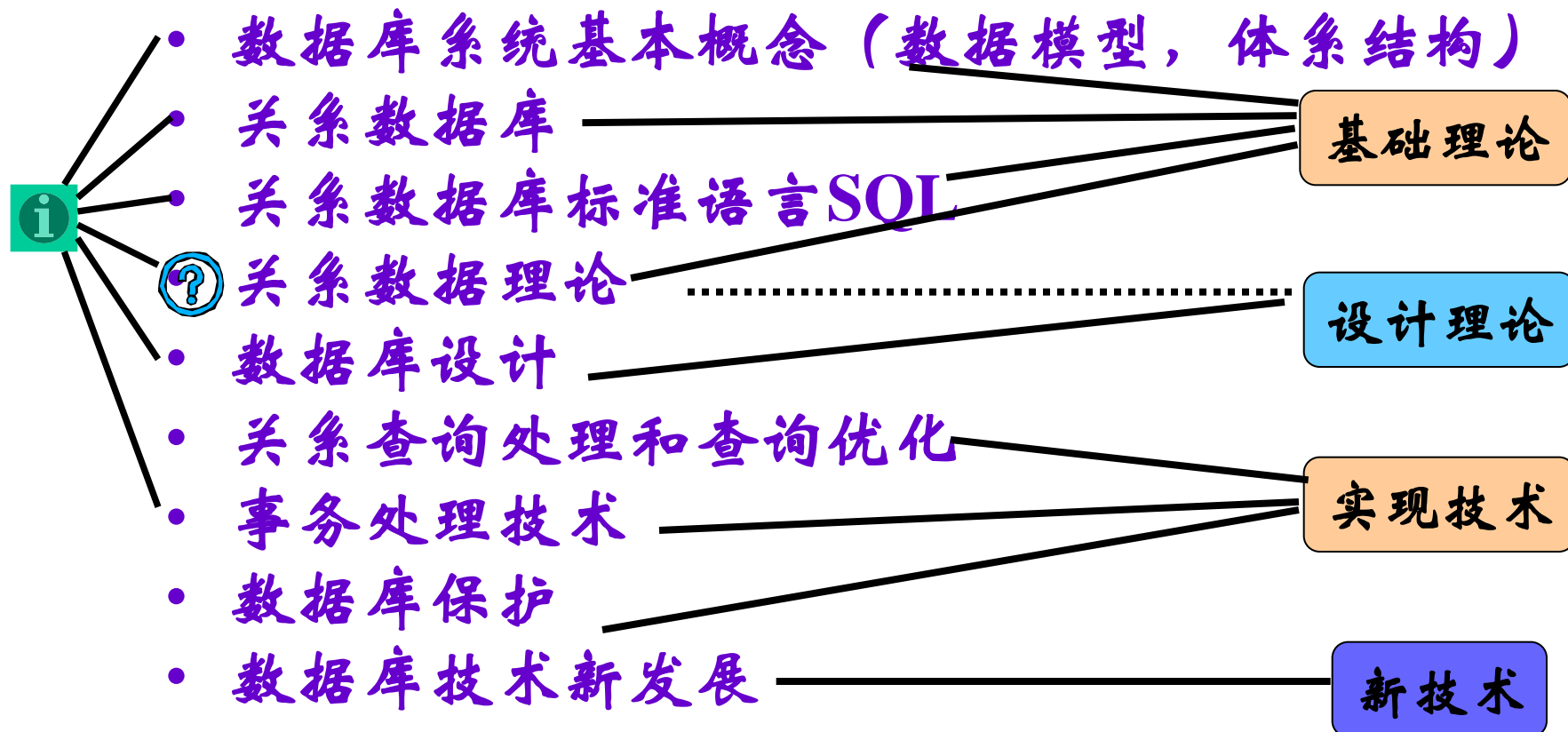
关系数据理论、
数据库设计方法

底层实现技术、事务处理
数据库安全、完整性控制

数据模型



课程内容



课程目标

- 课程目标：
 - 理解与掌握数据库系统的基本理论
 - 掌握标准数据操纵语言与数据库系统的设计方法
 - 了解数据库管理系统的实现技术

教材及考核方式

- 教材

- 萨师煊,王珊编著, 数据库系统概论, 第三版/第四版, 高等教育出版社

- 考核方式

- 作业 10%
- 课堂参与(测验,提问) 10%
- 期末考试 80%

课程设计

- 36学时
- 与课程同步进行
- 单独考核，五级评分

课程管理

- 数据库教学平台
(<http://sammi.nlsde.buaa.edu.cn>)
- 课程助教



第一章 概述

- 什么是数据库系统
- 数据模型
- 数据库系统结构
- 数据库技术的发展

什么是数据库系统

- 数据管理技术的产生
- 数据管理技术的发展
- 数据库系统的组成



数据与信息

- 数据
 - 数据是描述现实世界中的各种事物的可以识别的符号；
- 信息
 - 信息是一种已经被加工为特定形式的数据，这些数据对现在与将来的决策有明显价值；
- 数据与信息
 - 数据是信息的载体，是信息的具体表现形式；信息是各种数据所包括的意义，数据与信息是密切关联的。

数据处理与数据管理

- 数据处理

- 目的：从大量原始数据中抽取和推导出有价值的信息。
- 过程：数据收集、组织、存储、加工、分类、检索、输出、传输等操作。

- 数据管理

- 数据处理一般性的基本操作，如数据收集、组织、存储、分类、检索、传输等称为数据管理，并研究专门的技术——数据管理技术。



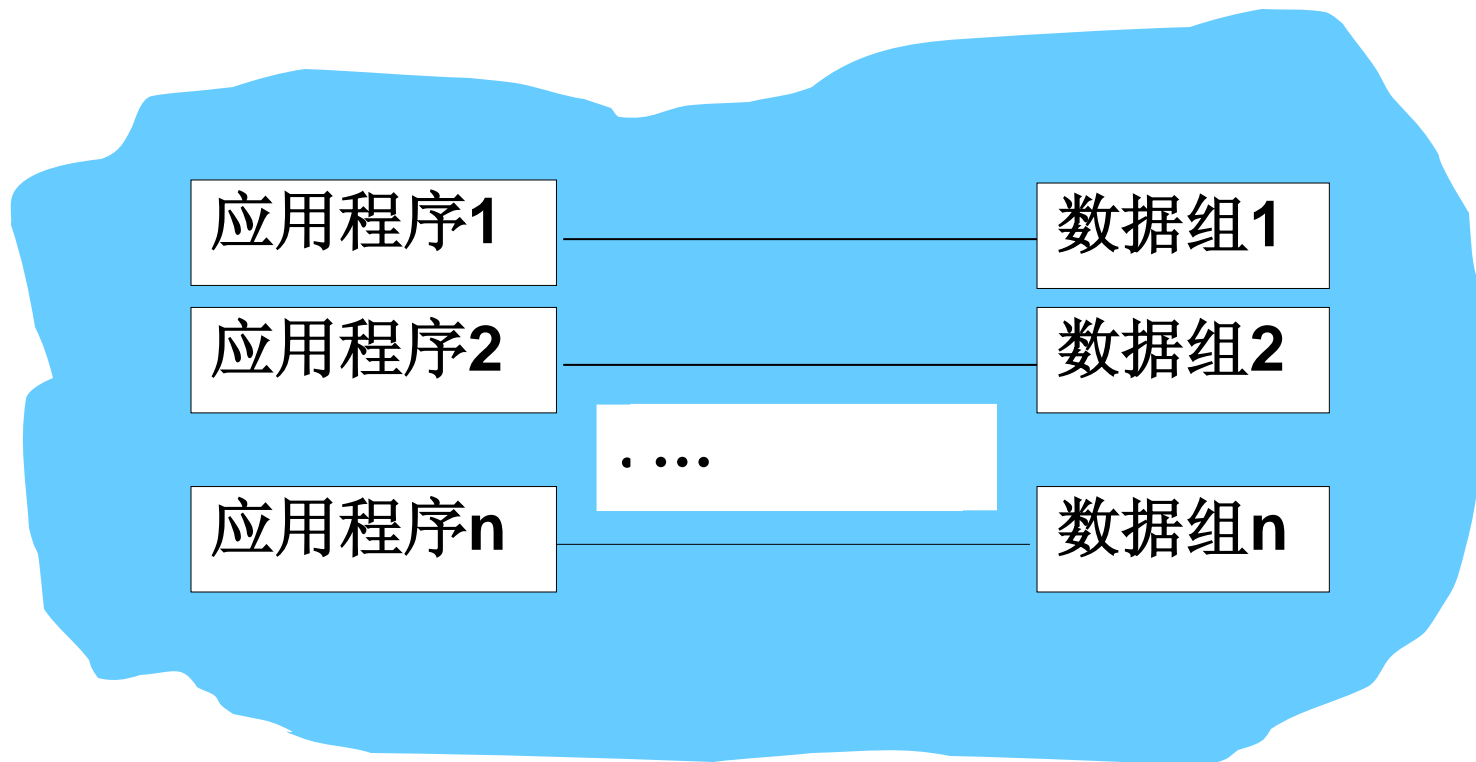
数据管理技术的发展

- 人工管理阶段
- 文件系统阶段
- 数据库系统阶段

人工管理阶段

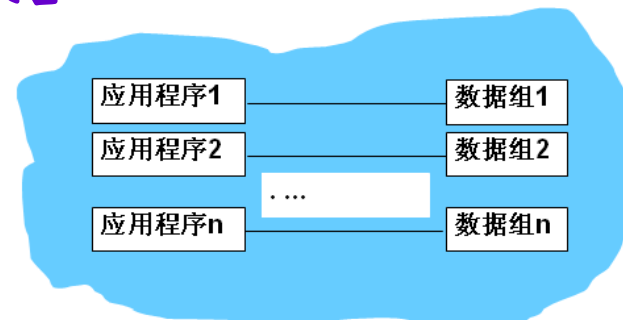
- 时间：20世纪50年代中期以前
- 背景：
 - 外存只有磁带、卡片、纸带等，没有磁盘等直接存取设备。
 - 没有操作系统，没有数据管理软件（用户用机器指令编码）。
 - 计算机主要用于科学计算。

人工管理阶段



人工管理阶段特点

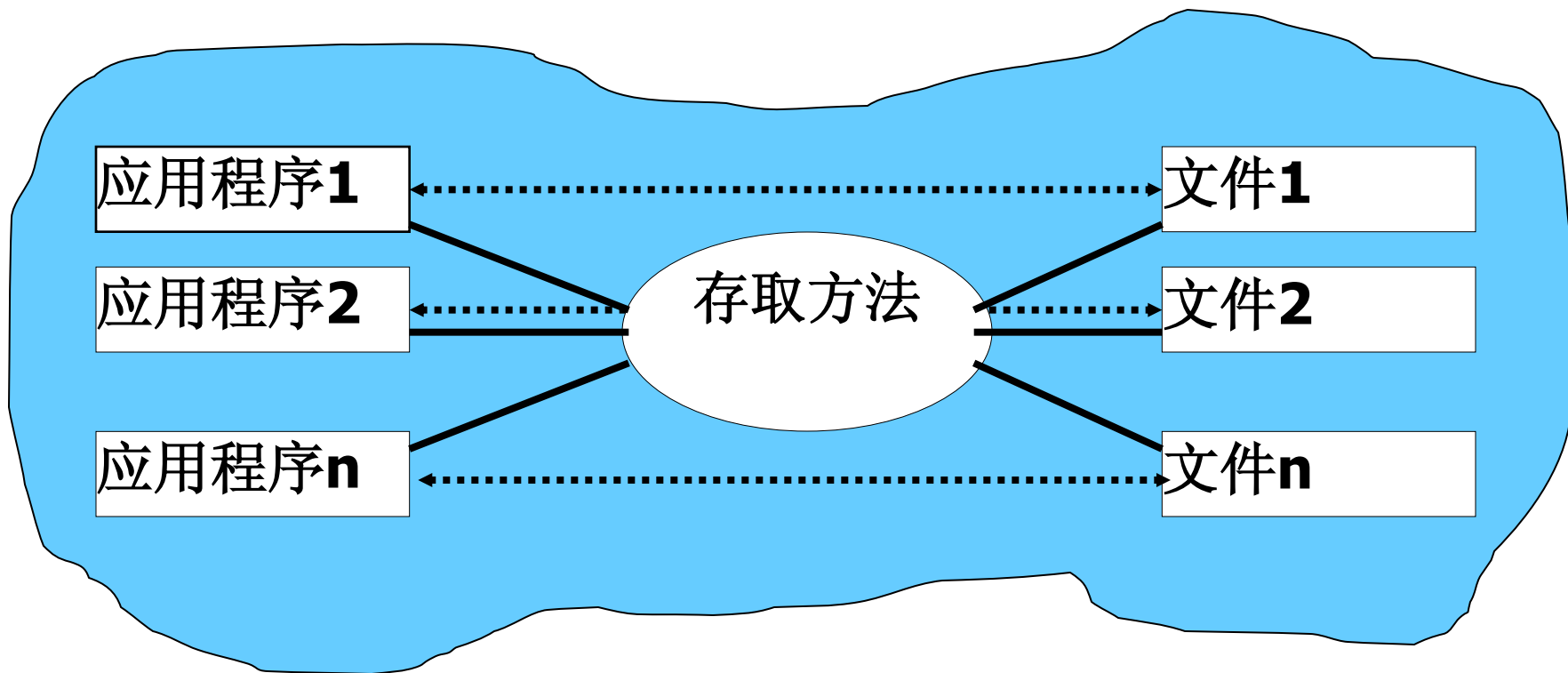
- 数据不在计算机上保存；
- 没有软件系统对数据进行管理。程序规定数据的逻辑结构与物理结构。数据与程序不具有独立性；
- 基本没有文件概念，数据组织方式必须由程序员自行设计；
- 一组数据对应一个程序，数据是面向应用的，程序间不能共享数据。



文件系统阶段

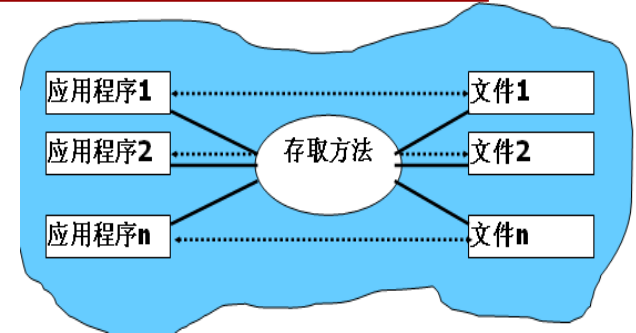
- 时间：20世纪50年代后期到60年代中期
- 背景：
 - 外存有了磁盘、磁鼓等直接存取设备；
 - 有了专门管理数据的软件，一般称为文件系统，包括在操作系统中；
 - 计算机不但用于科学计算，还用于管理；

文件系统阶段



文件系统阶段特点

- 数据以文件形式保留在外存上；
- 程序和数据有一定的独立性；
- 文件多样化；
- 数据的存取基本上以记录为单位；
- 缺点：
 - 数据冗余度大：浪费空间并易造成数据的不一致性。
 - 数据和程序缺乏独立性（逻辑独立性）

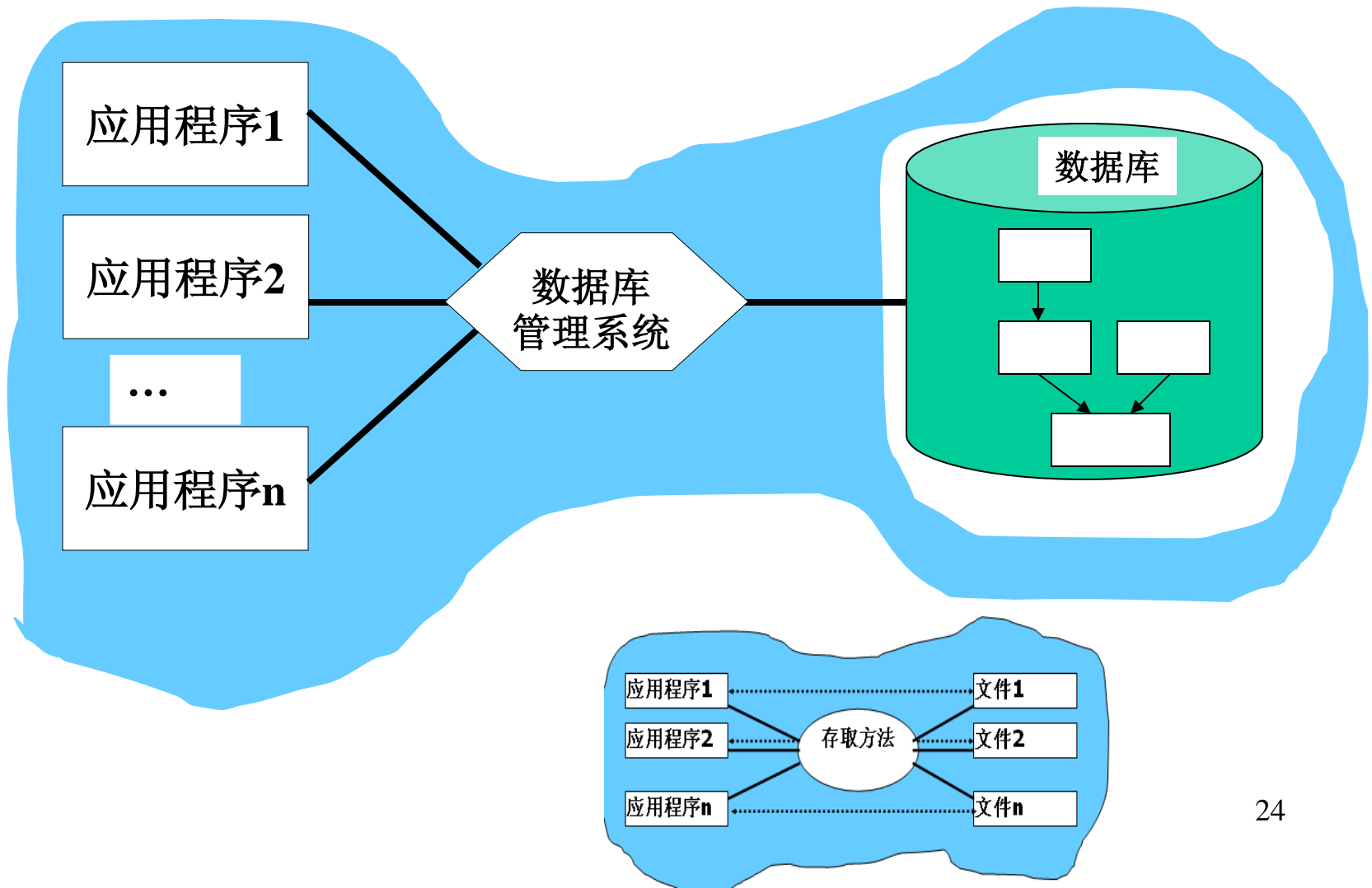


文件系统是不具弹性的无结构的数据集合，数据之间是孤立的，不能反映现实世界事物之间的内在联系。

数据库系统阶段

- 时间：20世纪60年代后期开始
- 背景：
 - 外存有了大容量磁盘，光盘；
 - 软件价格上升，硬件价格下降，编制和维护软件及应用程序成本相对增加，其中维护的成本更高；
 - 计算机管理的数据量大，关系复杂，共享性要求强（多种应用、不同语言共享数据）。

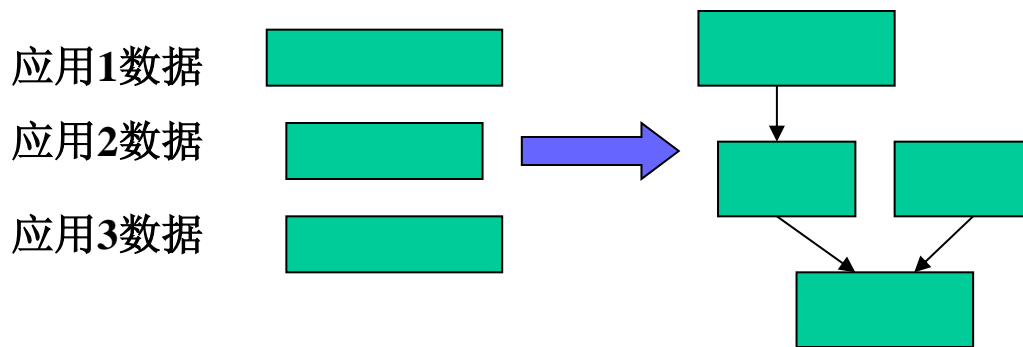
数据库系统阶段



数据库系统数据管理特点 (1)

- 面向全组织的复杂的数据结构

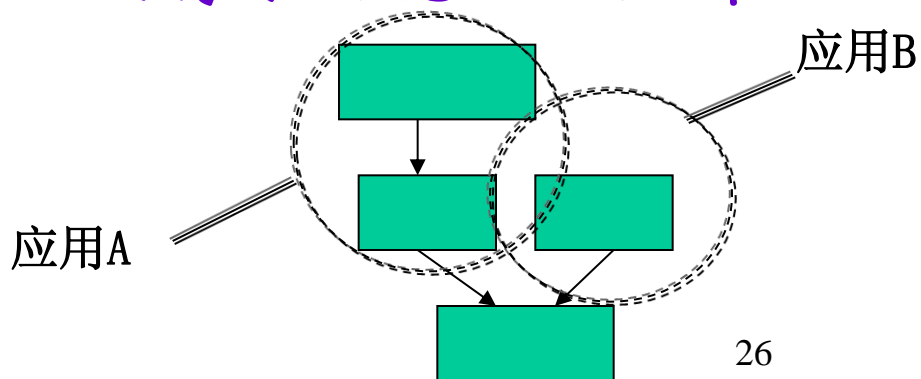
- 在描述数据时，不仅描述数据本身，还要描述数据之间的联系，使整个组织的数据结构化；



- 数据结构化是数据库主要特征之一，是数据库与文件系统的根本区别。

数据库系统数据管理特点 (2)

- 数据冗余度小，易扩充
 - 数据库从整体观点描述数据。数据不再面向某个应用，而是面向整个系统，从而大大减小数据的冗余度；
 - 数据库数据的应用可以有很灵活的方式，可以取整体数据的各种合理子集用于不同的应用系统，并可以根据应用需求的变化，重新选取不同子集。



数据库系统数据管理特点 (3)

- 具有较高的数据和程序的独立性

- 数据独立性

- 数据的物理独立性：数据的存储结构（物理结构）改变时，数据的逻辑结构可以不变，从而应用程序也不必改变；
 - 数据的逻辑独立性：数据的逻辑结构改变时，应用程序可以不变；

- 数据库系统提供了两方面的映象（转换）功能：

- 数据的存储结构与逻辑结构之间——实现数据的物理独立性
 - 数据的总体逻辑结构与某类应用所涉及的局部逻辑结构之间——实现数据的逻辑独立性

数据库系统数据管理特点 (4)

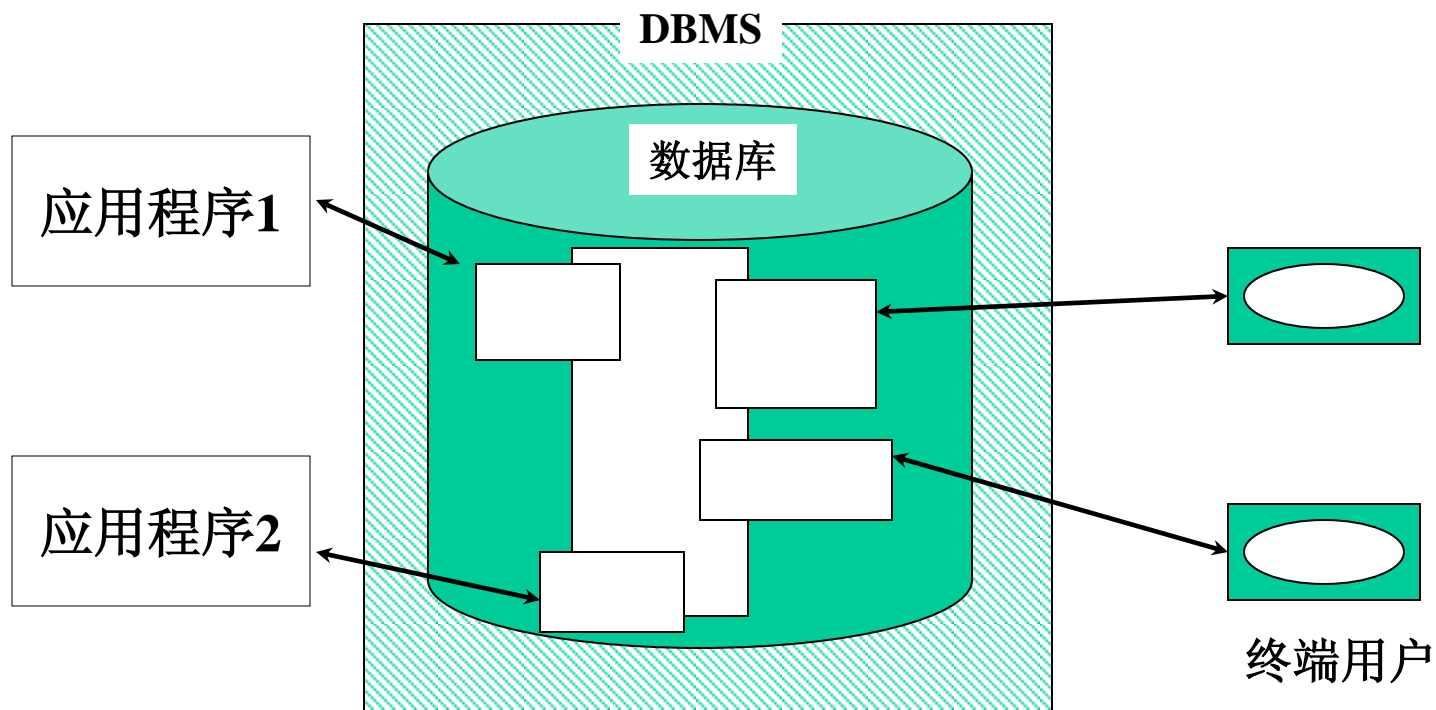
- 统一的数据控制功能
 - 数据的安全性控制
 - 保护数据以防止不合法的使用所造成数据的泄密和破坏。
 - 数据的完整性控制
 - 指数据的正确性与相容性。
 - 并发控制
 - 对多用户的并发操作进行控制、协调，保护数据的完整性。
 - 数据库恢复
 - 将数据库从错误状态恢复到某一已知的正确状态

数据库系统数据管理特点 (5)

- 数据的最小存取单位是数据项
 - 既可以存取一个或一组记录，也可以数据库中某个或一组数据项。



数据库系统的组成



数据库系统包括四个主要部分：数据库、用户、软件、硬件。

数据库系统的组成 (1)

- 数据库——数据库系统中集中存储的一批数据的集合。数据库中存储的数据是“集成的”和“共享的”。
 - “集成的”是指把特定应用环境中各种应用相关的数据及数据之间的联系全部地、集中地并按照一定结构形式进行存储。
 - “共享的”是指数据库中的数据可为多个不同的用户所共享。

数据库系统的组成 (2)

- 用户——指存储、维护和检索数据的各类请求。主要有三类用户：终端用户、应用程序员和数据库管理员。
 - 终端用户：使用终端命令语言，通过交互式对话方式存取数据；或通过应用系统的界面使用数据库
 - 应用程序员：通过设计、编写使用及维护数据库的应用程序存取数据；
 - 数据库管理员：全面负责数据库系统的管理、维护和正常使用的人员。

数据库系统的组成 (3)

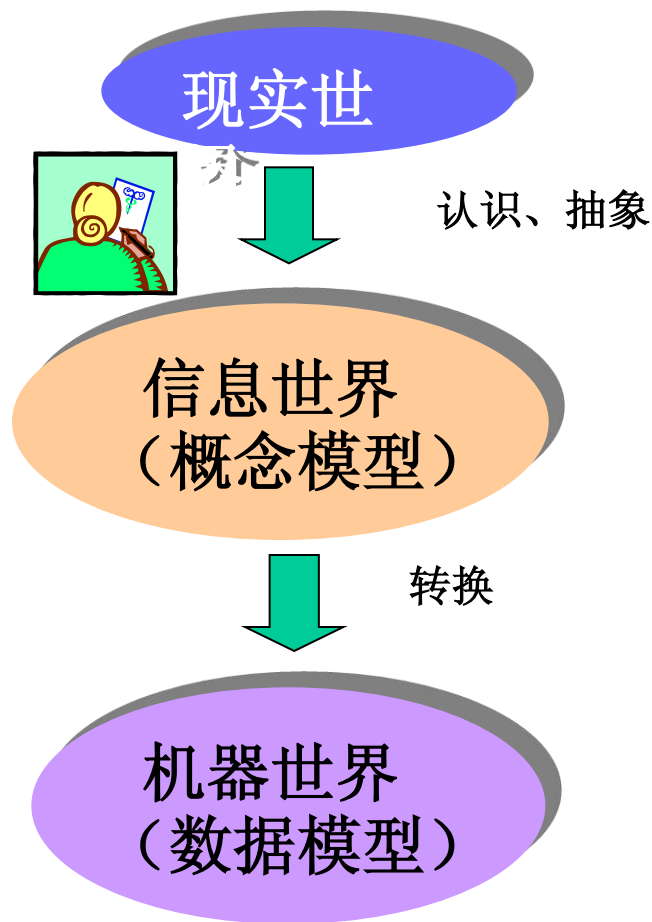
- 软件——指负责数据库存取、维护和管理软件系统。包括数据库管理系统DBMS，各种应用系统。DBMS是数据库系统的核心软件，在操作系统支持下工作；
- 硬件——指存储数据库和运行DBMS的硬件资源，包括计算机的内存和外存等。



数据模型

- 数据模型用来抽象和表示现实世界中的数据 and 信息。
- 在数据库中，根据模型应用的不同目的，将模型分为两类或两个层次：
 - 概念模型（也称信息模型）
 - 数据模型（如层次、网状、关系模型）。

数据模型的层次



- 概念模型用于信息世界建模，是现实世界到信息世界的抽象，是用户和数据库设计人员进行交流的语言。
- 数据模型用于机器世界，按计算机系统的观点对数据建模。

数据模型

- 概念模型
- 数据模型
- 数据模型的分类



概念模型

- 概念模型基于信息世界的主要概念，具有较强的语义表达能力，能够方便、直接表达应用中的各种语义。
- 概念模型应该简单、清晰、易于理解。
- 概念模型的最常用的表示方法：
 实体—联系方法（E-R）法。

概念模型中的基本概念

- 实体 (Entity) : 客观存在并可相互区分的事物;
- 属性 (Attribute) : 实体所具有的某一特性;
- 码(Key): 唯一标识实体的属性集;
- 域 (Domain) : 某个 (些) 属性的取值范围;
- 实体型 (Entity Type) : 用实体名及其属性名集合来抽象、刻画同类实体;
- 联系 (Relation) : 实体型之间的联系。
 - 一对一联系 (1: 1)
 - 一对多联系 (1: n)
 - 多对多联系 (m: n)



数据库中的数据模型

- 数据模型是严格定义的概念的集合。这些概念精确地描述系统中数据的静态特性、动态特性和完整性约束。
- 数据模型的三要素：
 - 数据结构
 - 数据操作
 - 完整性约束

数据模型三要素——数据结构

- 数据结构由描述数据对象以及对象之间联系的一组概念组成。包括：
 - 描述对象的类型、内容、性质的概念，如关系模型中的域、属性等；
 - 描述对象之间联系的概念，如关系模型中的关系。
- 是数据模型静态特性的描述。
- 数据结构是刻画数据模型最重要的方面。通常按照数据结构的类型来命名数据模型。

数据模型三要素——数据操作

- 是对数据库中各种数据对象（型）的实例（值）允许执行的操作集合，包括操作及操作规则。数据模型要定义操作的确切含义、操作符号、操作规则及操作语言。
- 是数据模型的动态特性的描述。
- 数据库主要有检索和更新（插、删、改）两大类操作。

数据模型三要素——数据的约束条件

- 是完整性规则的集合。完整性规则是给定的数据模型中数据及其联系所有的制约和依存规则，用以保证数据的正确、相容。
- 完整约束条件包括：
 - 符合这种数据模型所必须遵守的基本的通用的完整性约束条件；
 - 针对具体数据的特定语义约束条件。

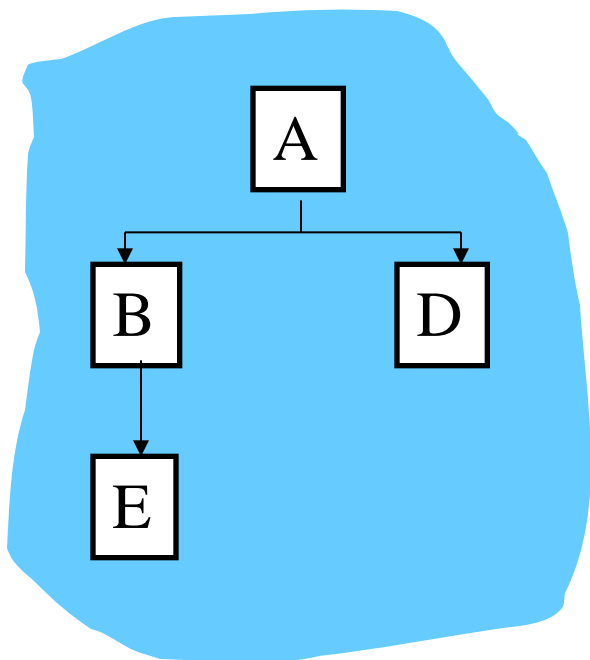


数据模型的分类

- 当前公认的基本数据模型有三类：层次模型、网状模型、关系模型。
- 各种数据类型之间的根本区别在于数据之间联系的表示方式不同：
 - 层次模型：用“树结构”表示数据之间的联系；
 - 网状模型：用“图结构”表示数据之间的联系
 - 关系模型：用“二维表”表示数据之间的联系

层次模型

- 是数据库系统中最早出现的数据模型。
- 数据结构是有向树。节点代表实体型，连线表示两实体型间的一对多联系。



特征：

- 有且仅有一个结点没有双亲；
- 其它结点有且仅有一个双亲。

层次模型

- 优点：

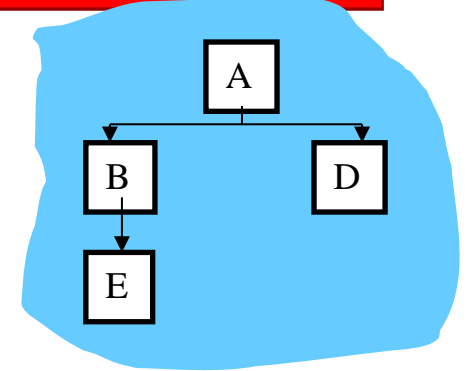
- 结构简单，易于实现。

- 缺点：

- 支持的联系种类太少，只支持二元一对多联系。
- 数据操纵不方便，子结点的存取只能通过父结点来进行。

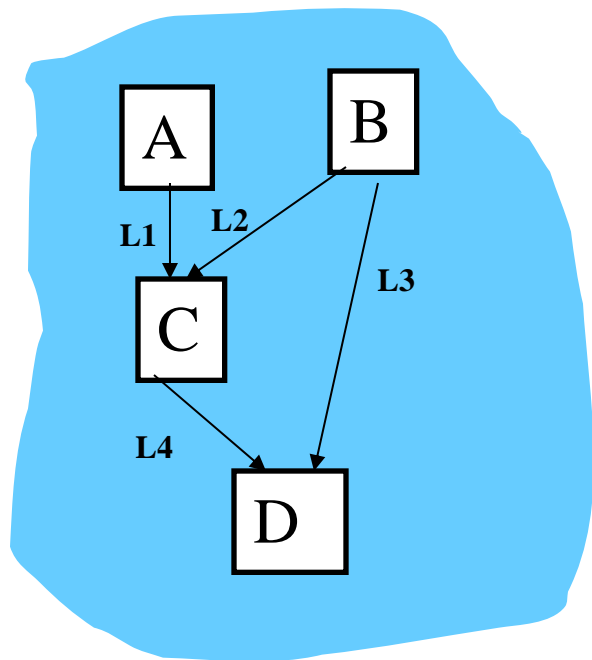
- 代表产品：

- IBM的IMS数据库，1968年研制成功。



网状模型

- 数据结构是有向图。节点代表实体型，连线表示两实体型间的一对多联系。



特征：

- 可有一个以上的结点没有双亲；
- 至少有一个结点有多于一个的双亲。

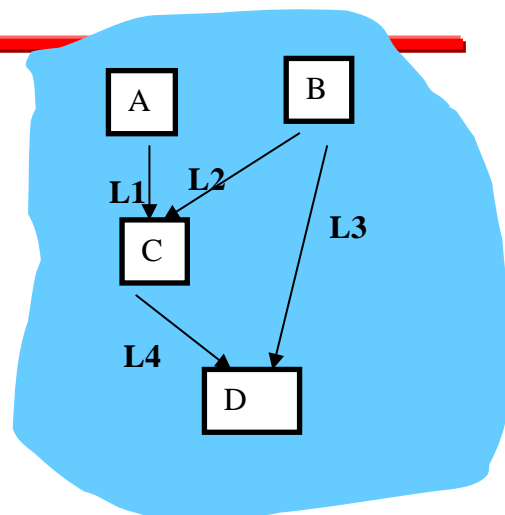
网状模型

- 特点：

- 表达的联糸种类丰富。
- 结构复杂。

- DBTG报告：

- 1969年，由美国CODASYL（Conference On Data System Language，数据系统语言协商会）下属的DBTG（Data Base Task Group）组提出，确立了网状数据库系统的概念、方法、技术。



关系模型

- 用二维表格结构（关系）表示实体及实体之间的联系。

学生

学号	姓名	年龄	系别
s1	A	18	CS
s2	B	18	CS
s3	C	18	MA

课程

课号	课名	先行课号
c1	aaa	
c2	bbb	c1
c3	ccc	

学生选课

学号	课号	成绩
s1	c1	80
s1	c2	90
s2	c1	95

关系模型

- 特点

- 用关系描述实体及实体间的联系。这种描述一致性使数据结构大大简化，概念简单。
- 可直接表示多对多联系。
- 关系必须是规范化关系。即每个分量是不可分的数据项，或不许表中套表。
- 关系模型是建立在数学概念基础上，有较强的理论基础。

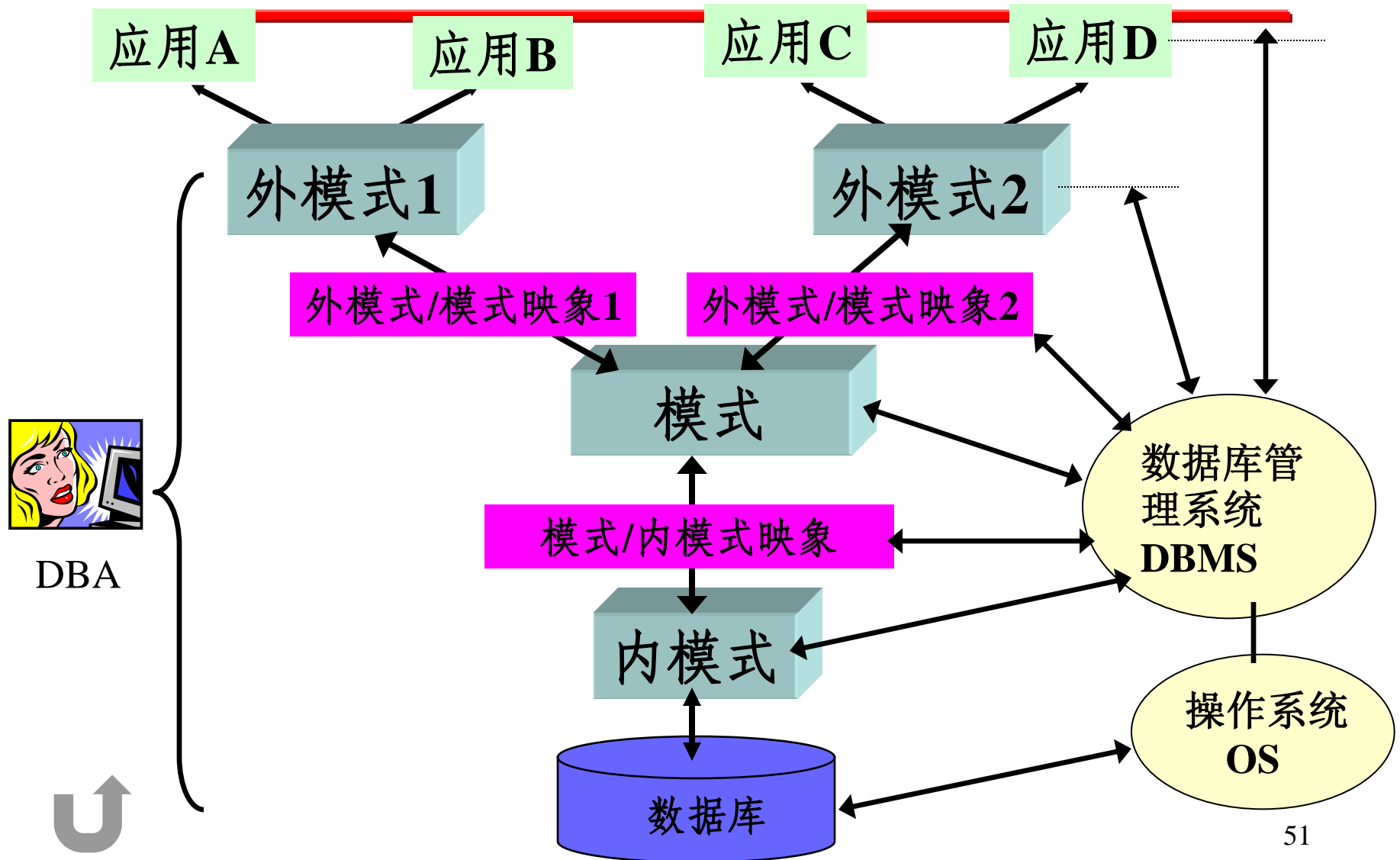
- 关系数据库产品：Oracle, SQL Server, MySQL等。

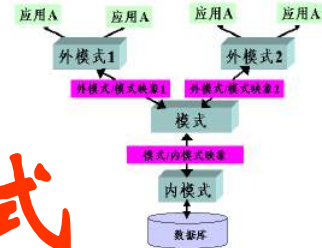


数据库系统的结构

- 数据库系统的体系结构上具有三级模式的结构特征。
 - 模式
 - 外模式
 - 内模式
 - 两级映象
- DBMS和DBA

数据库系统的结构



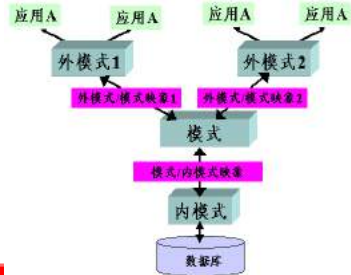


数据库系统三级模式结构——模式

- 也称为逻辑模式、概念模式，是数据库中全体数据的逻辑结构和特性的描述。是所有用户的公共数据视图。
- 是三级模式的核心。不涉及数据物理存储细节，与具体的应用程序与编程语言无关。
- 具体定义数据的逻辑结构（数据记录结构、数据之间的联系）、数据安全性、完整性要求。
- 数据库系统提供模式描述语言（模式DDL，Data Description Language）进行模式定义。
用模式DDL写出的一个数据库逻辑定义的全部语句，称为某个数据库的模式。

数据库系统三级模式结构

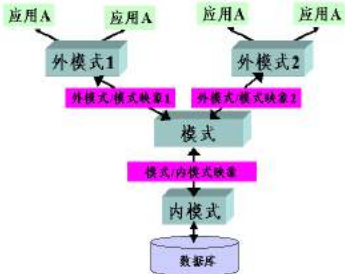
——外模式



- 也称为子模式或用户模式。是个别用户的数据视图，即与某一应用有关的数据的逻辑表示。
- 通常是模式的子集。不同应用的外模式可以相互覆盖，一个应用只能启用一个外模式。
- 数据库系统提供外模式描述语言（外模式DDL）定义外模式。外模式DDL和用户选用的程序设计语言具有相容的语法。

数据库系统三级模式结构

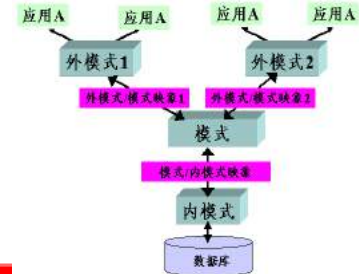
——内模式



- 也称为存储模式，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。
- 内模式通常用内模式数据描述语言（内模式DDL）来描述和定义。

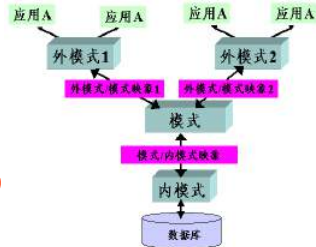
数据库系统三级模式结构

——两级映像



- 外模式/模式映像定义某个外模式与模式之间的对应关系。当模式改变时，外模式/模式映像做相应改变，可以保证外模式不变。——数据的逻辑独立性
- 模式/内模式映像定义数据逻辑结构与存储结构之间的对应关系。当内模式改变时，模式/内模式映像做相应修改，使得模式保持不变。——数据的物理独立性

数据库系统三级模式结构优点

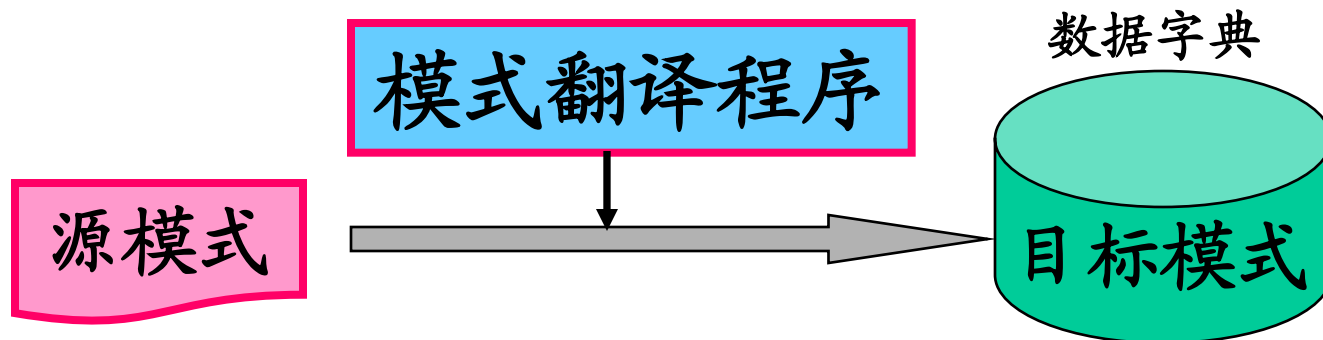


- 保证数据的独立性
 - 模式与内模式分开——数据物理独立性
 - 外模式与模式分开——数据逻辑独立性
- 简化用户接口，方便用户使用
 - 用户只按照外模式操作，无需了解数据库的总体逻辑结构与物理存储结构。
- 有利于数据共享
 - 从模式产生不同的外模式，外模式间可相互覆盖。
- 有利于数据的安全保密
 - 应用程序只能操作其对应的外模式

DBMS主要功能 (1)

- 数据库定义功能

- 提供DDL语言描述外模式、模式、内模式（源模式）。
- 模式翻译程序把源模式翻译成目标模式，存入数据字典中。



DBMS主要功能 (2)

- 数据存取功能
 - 提供DML语言 (Data manipulation language) 对数据库进行检索、插入、修改、删除。
- 数据库运行管理
 - 并发控制、存取控制、完整性约束条件检查和执行，日志组织和管理，事务管理和自动恢复。

DBMS主要功能 (3)

- 数据组织、存储和管理
 - 用户数据、索引、数据字典的组织、存储和管理，包括文件结构、存取方式、数据之间联系的实现等。
- 数据库的建立和维护功能
 - 数据的装入、转换，数据库的转储、恢复、性能监视和分析等。

DBMS的组成

- 语言编译处理程序
- 系统运行控制程序
 - 包括系统总控、存取控制、并发控制、完整性控制、保密性控制、数据存取和更新、通信控制等程序。
- 系统建立和维护程序
 - 数据装入、数据库系统恢复、性能监督、工作日志等程序。
- 数据字典
 - 也称为数据目录或系统目录，由一系列表组成，存储着数据库中有关信息的当前描述，包括数据库的三级模式、用户名表、用户权限等信息。

DBA

- 建库方面：

- 确定模式、外模式、存储结构、存取策略、负责数据的整理和装入。

- 用库方面：

- 定义完整性约束条件，规定数据的保密级别、用户权限，监督和控制数据库的运行情况，制定后援和恢复策略，负责故障恢复。

- 改进方面：

- 监督分析系统的性能（空间利用率，处理效率）；
- 数据库重组织，物理上重组织，以提高性能；
- 数据库重构造，设计上较大改动，模式和内模式修改。



数据库技术的发展 (1)

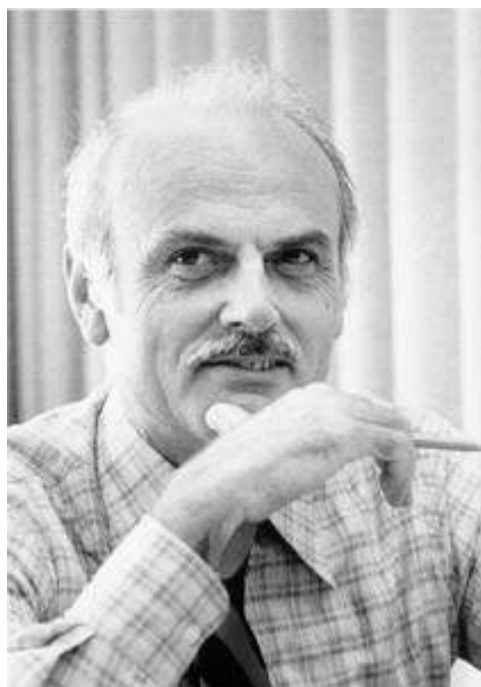
- 第一代数据库系统
 - 70年代的层次和网状数据库系统。
 - 代表：1968年IBM研制的IMS。
 - 60年代末，70年代初CODASYL给出的DBTG报告。
- 第二代数据库系统
 - 1970年IBM San Jose实验室研究员E.F.Codd提出关系模型
 - 代表：IBM开发的System R和加州大学Berkley分校开发的INGRES。

数据库技术的发展 (2)

- 第三代数据库系统
 - 面向对象数据库
 - 数据库技术与其他学科的内容相互结合
 - 面向应用领域的数据库技术研究

第二章 关系数据库

E.F.Codd于70年代初提出关系数据理论，因此获得1981年的ACM图灵奖。



Edgar Frank Codd, 1923—2003

Information Retrieval

P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and need-responsive programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on n -ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, structure at run, structure at data, structure, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity.

CR CATEGORIES: 370, 373, 375, 420, 422, 429

1. Relational Model and Normal Form

1.1. INTRODUCTION

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levin and Maron [2] provide numerous references to work in this area.

In contrast, the problems treated here are those of data independence—the independence of application programs and terminal activities from growth in data types and changes in data representation—and certain kinds of data inconsistency which are expected to become troublesome even in nondeductive systems.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for non-inferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Amazingly, it provides a basis for a high-level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. DATA DEPENDENCIES IN PRESENT SYSTEMS

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed without logically impairing some application programs is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is closely associated with the hardware-determined ordering of addresses. For example, the records of a file concerning parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a subordering of) the

-
- 关系理论是建立在集合代数理论基础上的，有着坚实的数学基础。
 - 早期代表系统
 - System R：由IBM研制。
 - INGRES：由加州Berkeley分校研制。
 - 目前主流的商业数据库系统
 - Oracle, Informix, DB2 , SQL Server, Sybase
 - Access, FoxPro, Foxbase

主要内容

- 关系模型的基本概念
- 关系代数
- 元组关系演算与域关系演算
- 三类关系运算的安全约束及等价性
- 关系数据语言概述

关系的数学定义

- 域 (Domain)

- 定义：一组具有相同的数据类型的值的集合。例如，整数，实数， $\{0, 1\}$ 等。

- 元组和分量

- 定义：给定一组域 D_1, D_2, \dots, D_n ，这些域中可以有相同的。

D_1, D_2, \dots, D_n 的笛卡尔积 (Cartesian Product) 为：

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

笛卡尔积的每个元素 (d_1, d_2, \dots, d_n) 称作一个n元组 (n-tuple) 或简称元组。元组的每一个值 d_i 叫做一个分量 (component)。

若 D_i ($i=1, \dots, n$) 为有限集，其基数 (Cardinal number) 为 m_i ，则笛卡尔积的基数为

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

关系的数学定义

- 笛卡儿积可以表达为二维表：

域		D1	D2	...	Dn
		d1	d2	...	dn
元组

分量

$$D1 \times D2 \times \dots \times Dn = \{(d1, d2, \dots, dn) \mid di \in Di, i=1, \dots, n\}$$

笛卡儿积示例

例：有三个域

$D_1 = \text{MAN} = \{M_1, M_2, M_3\}$

$D_2 = \text{WOMAN} = \{W_1, W_2\}$

$D_3 = \text{CHILD} = \{C_1, C_2, C_3\}$

则 $D_1 \times D_2 \times D_3 =$

$\{ (M_1, W_1, C_1), (M_1, W_1, C_2),$
 $(M_1, W_1, C_3), \dots \}$

二维表的表示：

共18个

D_1	D_2	D_3
M_1	W_1	C_1
M_1	W_1	C_2
M_1	W_1	C_3
...

关系的定义

- 关系的定义：笛卡尔积 $D_1 \times D_2 \times \dots \times D_n$ 的子集叫做在域 D_1, D_2, \dots, D_n 上的关系，用 $R(D_1, D_2, \dots, D_n)$ 表示。 R 是关系的名字， n 是关系的度或目(Degree)。

当 $n=1$ 时称为单元关系，

当 $n=2$ 时称为二元关系，依此类推。

- 关系可以表示为二维表：
 - 表的框架由 D_i ($i=1,2,\dots,n$) 构成
 - 表的每一行对应一个元组
 - 表的每一列对应一个域
 - 每个列附加一个名称，则每个列称为一个属性(Attribute)。

属性的名字是唯一的。属性 A_i 的取值范围 D_i ($i=1,2,\dots,n$) 称为值域。 n 元关系必有 n 个属性。

A_1	A_2	...	A_n
a_1	a_2	...	a_n
...
...

关系示例

以MAN、WOMAN、CHILD三个域的笛卡儿积的子集构造FAMILY关系：

FAMILY (MAN, WOMAN, CHILD)

其中MAN—丈夫，WOMAN—妻子，CHILD—这对夫妻的子女。假设M1、W1、C1，M2、W2、C2，C3是两个家庭，则FAMILY关系为：

MAN	WOMAN	CHILD
M ₁	W ₁	C ₁
M ₂	W ₂	C ₂
M ₂	W ₂	C ₃

属性

元组

关系的性质

- 列是同质的 (Homogeneous) 即每一列中的分量来自同一域, 是同一类型的数据;
- 不同的列可出自同一域, 每列必须有不同的属性名;
- 列的顺序无所谓, 即列次序可以互换;
- 任意两个元组不能完全相同;
- 行的顺序无所谓, 即行次序可以互换;
- 每一分量必须是不可再分的数据。满足这一条件的关系称作满足第一范式 (1NF) 的。

关系数据模型

- 关系模型的数据结构
- 关系模型的语义约束
- 关系模型的数据操作概述



关系模型的数据结构 (1)

- 关系模型的数据结构——关系
 - 实体及实体之间的联系均用单一的数据结构——“关系”来表示。
- 几个基本概念
 - 关系、域、n目关系、元组、属性。
 - 码 (Key, 键)
 - 候选码 (Candidate key)：关系中的某一属性组，若它的值唯一地标识了一个元组，并具有最小性，则称该属性组为候选码。
 - 主码 (Primary key, 首码, 码)：若一个关系有多个候选码，则选定其中一个为主码。
 - 主属性与非主属性
 - 码中的诸属性称为主属性，不包含在任何候选码中的属性称为非主属性。

关系数据库结构(2)

U为组成该关系的属性名集合

D为属性集U所来自的域

dom为属性向域的映象集合

F为属性间的数据依赖关系集合

I为完整性约束集合

模式

关系的描述称作关系模式。它可以形式化地表示为：

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

- 关系模式通常可以简记作 $R(A_1, A_2, \dots, A_n)$ ，R为关系名， A_1, A_2, \dots, A_n 为属性名，域名及属性向域的映象直接说明为属性的类型、长度。
- 关系是关系模式在某一时刻的状态或内容。关系模式是相对稳定的。而关系是动态的，是随时间不断变化的。

—关系数据库（关系数据库的型和值的概念）

- 关系模式的集合构成关系数据库模式——关系数据库的型；
- 关系的集合则构成具体的关系数据库——关系数据库的值。



关系模型的语义约束

- 实体完整性
 - 参照完整性
 - 用户定义完整性
- 用户针对具体的应用环境定义的完整性约束条件。
- 关系模型必须支持的约束条件

关系模型的语义约束

- 实体完整性 (Entity Integrity)
 - 要有属性或属性组合作为主码，主码值不可为空或部分为空。或定义为若属性A是关系R的主属性，则属性A不能取空值。
 - 空值的含义是：不知道或不存在的值。

学生

学号	姓名	年龄	系别
s1	A	18	CS
s2	B	18	CS
s3	C	18	MA

关系模型的语义约束

- 参照完整性

- 外部码

- 设F是基本关系R的一个或一组属性，但不是R的码。如果F与基本关系S的主码 K_s 相对应，则称F是关系R的外部码 (Foreign Key)，并称R为参照关系 (Referencing Relation)，S为被参照关系 (Referenced Relation) 或目标关系 (Target Relation)。R和S不一定是不同的关系。
 - 目标关系S的主码 K_s 和参照关系的外部码F必须定义在一个域上。

- 参照完整性

- 如果关系R的外部码 F_k 与关系S的主码 P_k 相对应，则R中的每一个元组的 F_k 值或者等于S中某个元组的 P_k 值，或者为空值。

关系模型的语义约束

- 参照完整性示例

职工关系EMP (ENO, ENAME, DNO) 和部门关系DEPT (DNO, DNAME) 是两个基本关系。

DNO是EMP的外码。

EMP中每个元组在DNO上的取值允许有两种：

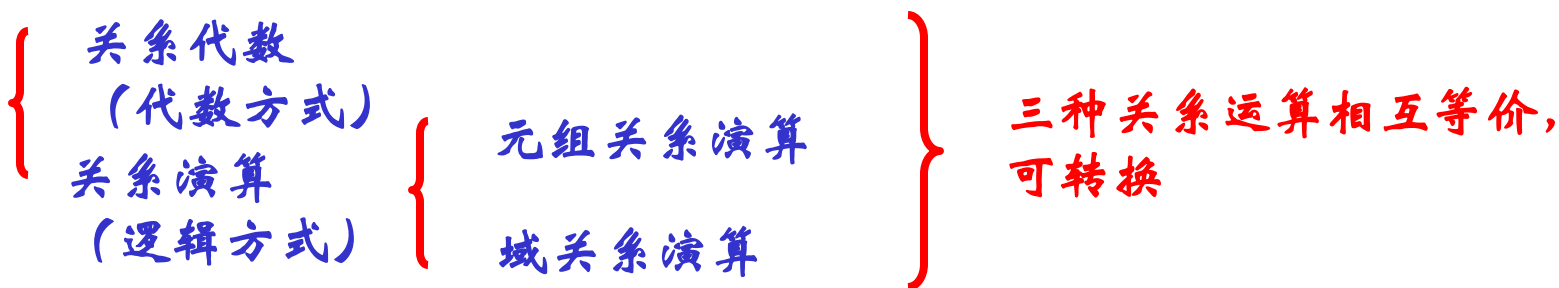
(1) 取空值；

(2) 非空，则DNO的值必须是DEPT中某个元组的DNO值。



关系模型的数据操作

- 关系数据操作方式的特点是**集合操作**，“一次一集合”方式。操作的对象与结果都是集合。
- 关系数据操作的基础是“**关系运算**”。关系运算方式有两种：代数方式，逻辑方式。



关系代数简介

- 关系代数是三种关系运算中的基础方法，有9种：
 - 常规集合运算：并、差、交、广义笛卡儿积（乘）；
 - 特有的关系运算：选择、投影、连接、自然连接、求商。

关系演算简介

- 元组演算表达式的形式:

$$\{t / \Phi(t)\},$$

其中 t 为元组变量, $\Phi(t)$ 是元组关系演算公式, 由原子公式和运算符组成;

- 域关系演算表达式的形式:

$$\{(x_1, x_2, \dots, x_k) / \Phi(x_1, x_2, \dots, x_k)\}$$

其中 x_i 代表域变量, Φ 为域关系演算公式, 由原子公式和运算符组成。



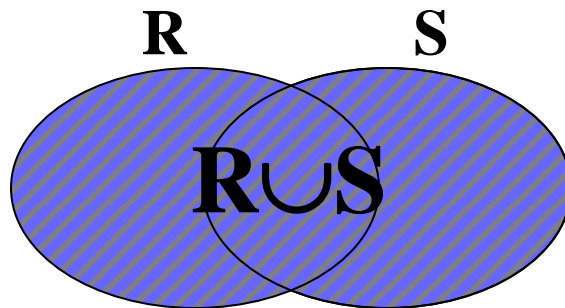
关系代数——传统集合运算

- 传统的集合运算是二目运算。除笛卡儿积外，要求参加运算的两个关系必须是同类关系，即两个关系具有相同的度“ n ”，且相对应的属性值必须取自同一个域。
- 设关系 R 和 S 是同类关系，可定义关系的集合运算如下：

关系代数——传统集合运算

- 并 (Union)

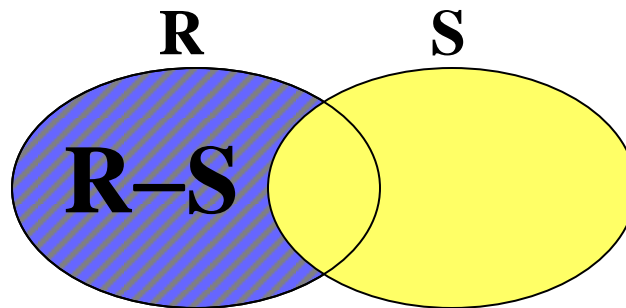
- 关系R和关系S的并记为 $R \cup S$ ，结果仍为n元关系，由属于R或属于S的元组构成。
- $R \cup S$ 表达为 $R \cup S = \{ t \mid t \in R \vee t \in S \}$



关系代数——传统集合运算

- 差 (Difference)

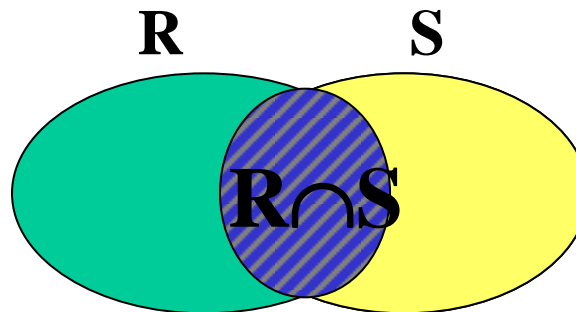
- 关系R和关系S的差记为R-S。结果仍为n元关系，由属于R而不属于S的元组构成。
- R-S表达为 $R-S = \{t \mid t \in R \wedge t \notin S\}$



关系代数——传统集合运算

- 交 (Intersection)

- 关系R和关系S的交记为 $R \cap S$ 。结果仍为n元关系，由既属于R又属于S的元组构成。
- $R \cap S$ 表达为 $R \cap S = \{ t \mid t \in R \wedge t \in S \}$



关系代数——传统集合运算

- 广义笛卡尔积

- 两个关系R, S, 其度分别为n, m, 则它们的笛卡尔积是所有这样的元组集合: 元组的前n个分量是R中的一个元组, 后m个分量是S中的一个元组。 $R \times S$ 的度为 $(n+m)$ 。

若R有 k_1 个元组, S有 k_2 个元组, 则 $R \times S$ 有 $k_1 \times k_2$ 个元组。

- $R \times S$ 表达为 $R \times S = \{ t \mid t = \langle r, s \rangle \wedge r \in R \wedge s \in S \}$

关系代数——传统集合运算

- 示例

R

A	B	C
a1	b1	c1
a1	b2	c2
a2	b2	c1

S

A	B	C
a1	b2	c2
a1	b3	c2
a2	b2	c1

求: $R \cup S$

$R - S$

$R \cap S$

$R \times S$

关系代数——专门的关系运算

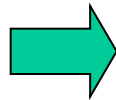
- 选取或限制 (Selection or Restriction)

- 定义：在关系R中选择满足给定条件的元组。记作：

$$\sigma_F(R) = \{t \mid t \in R, F(t) = \text{'真'}\}$$

- F是选择的条件，取逻辑值“真”或“假”。
 - F由运算对象（属性名、常数、简单函数），运算符，包括算术比较符（ $>$, \geq , $<$, \leq , $=$, \neq ）和逻辑运算符（ \wedge , \vee , \neg ）连接起来的表达式组成。

S#	C#	G
95001	C1	92
95001	C2	85
95001	C3	88
95002	C2	90
95002	C3	80



S#	C#	G
95001	C1	92
95001	C2	85
95001	C3	88

示例

学生表 S

S#	SN	SA	SD
95001	李勇	20	CS
95002	刘晨	19	IS
95003	王敏	18	MA
95004	张立	19	IS

学生选课表SC

S#	C#	G
95001	C1	92
95001	C2	85
95001	C3	88
95002	C2	90
95002	C3	80

课程表C

C#	CN	PC#
C1	数据库	C5
C2	数学	
C3	信息系统	C1
C4	操作系统	C6
C5	数据结构	C7
C6	数据处理	
C7	Pascal语言	C6

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 例1：检索计算机科学系（CS）学生的信息

$\sigma_{SD='CS'}(S)$

- 例2：检索C1课程的选修情况

$\sigma_{C\#='C1'}(SC)$

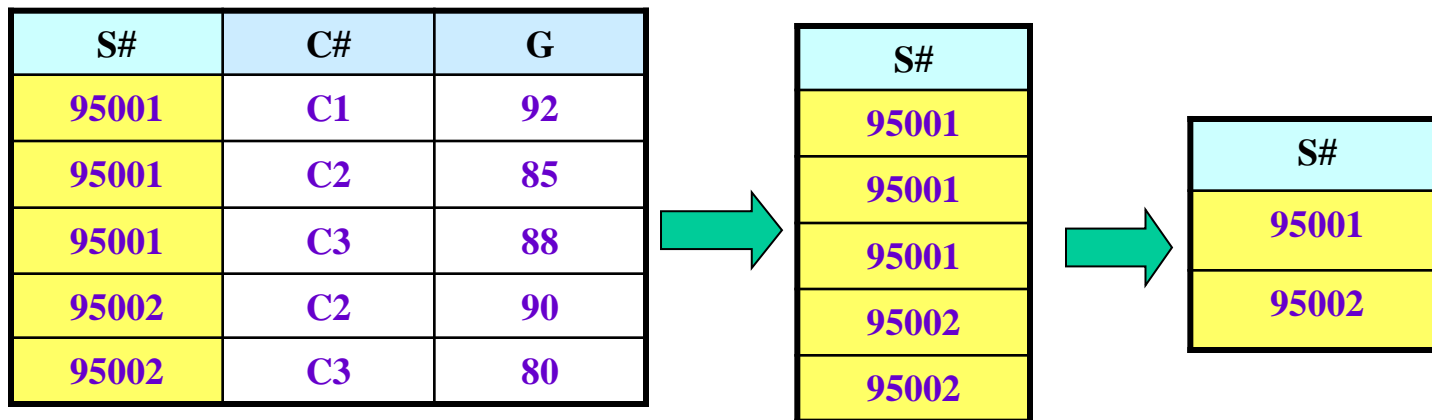
关系代数——专门的关系运算

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 投影 (Projection)

— 定义：从关系R(U)中取若干属性列并删去重复行，组成新的关系。记作：

$$\Pi_A(R) = \{ t[A] \mid t \in R, A \subseteq U \}$$



-
- 例1: 查询学生的姓名和所在系

$\Pi_{SN, SD}(S)$

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

关系代数——专门的关系运算

- 连接(Join)

- 定义：关系R和S在属性X和Y上的连接（X、Y是连接属性，即X、Y包含同等数量的属性，且相应的属性均具有共同的域），是从两个关系的广义笛卡儿积 $R \times S$ 中选取给定属性间满足 θ 比较条件的元组。记作：

$$R \bowtie_{X \theta Y} S = \{ t \mid t = \langle r, s \rangle \wedge r \in R \wedge s \in S \wedge r[X] \theta s[Y] \}$$

其中： \bowtie 是连接运算符；

θ 是算术比较运算符，

该连接也称为 θ 连接

当 θ 为 “=” 时，称为等值连接；

当 θ 为 “<” 时，称为小于连接；

当 θ 为 “>” 时，称为大于连接；

S				CS		
S#	SN	SA	SD	S#	C#	G
95001	李勇	20	CS	95001	C1	92
95002	刘晨	19	IS	95001	C2	85
95003	王敏	18	MA	95001	C3	88
95004	张立	19	IS	95002	C2	90
				95002	C3	80

CSS					
S#	SN	SD	C#	G	
95001	李勇	CS	C1	92	
95001	李勇	CS	C2	85	
95001	李勇	CS	C3	88	
95002	刘晨	IS	C2	90	
95002	刘晨	IS	C3	80	

关系代数——专门的关系运算

$S(S\#,SN,SA,SD);$
 $C(C\#,CN,PC\#);$
 $SC(S\#,C\#,G)$

- 自然连接 (Natural Join)

- 定义：关系R与关系S的自然连接，是从两个关系的广义笛卡儿积 $R \times S$ 中选取在相同属性列上取值相等的元组，并去掉重复的属性列。记作： $R \bowtie S$ ，或 $R * S$

$$R \bowtie S = \{ (Z,X,W) \mid (Z,X) \in R \wedge (W,X) \in S \wedge r[X] = s[X] \}$$

- 连接与自然连接：在等值（ θ 取“=”）连接情况下，连接属性X和Y是相同属性时，R与S的连接称为自然连接。自然连接的结果要在上述R与S的等值连接结果基础上再进行投影运算，去掉重复的属性列。

示例

- 连接与自然连接示例

R

A	B	C
a1	b1	5
a2	b3	8

S

B	E
b1	3
b3	7
b3	2

则 $R \bowtie_{C < E} S$

A	R.B	C	S.B	E
a1	b1	5	b3	7

$R \bowtie S$

A	B	C	E
a1	b1	5	3
a2	b3	8	7
a2	b3	8	2

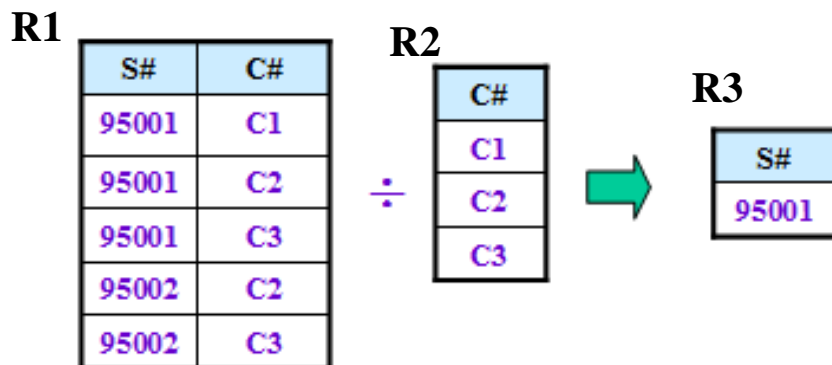
关系代数——专门的关系运算

- 除法

- 定义：设关系 $R(X, Y)$ 与关系 $S(Z)$ ，其中 Y 和 Z 具有相同的属性数，且对应属性出自相同域。关系 R 除以关系 S 所得的商关系是关系 R 在属性 X 上投影的一个子集，该子集和 $S(Z)$ 的笛卡儿积必须包含在 $R(X, Y)$ 中。

记为：

$$R \div S = \{t \mid t \in \Pi_x(R) \wedge s \in S \wedge \langle t, s \rangle \in R\}$$



示例

• 例：

R

A	B	C
b	2	d
b	3	b
c	2	d
d	4	b

S

B	C
2	d
3	b

则： $R \div S$

A
b

关系代数综合示例

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 例1：求选修了C2课程的学生学号。

$$\Pi_{S\#}(\sigma_{C\#='C2'}(SC))$$

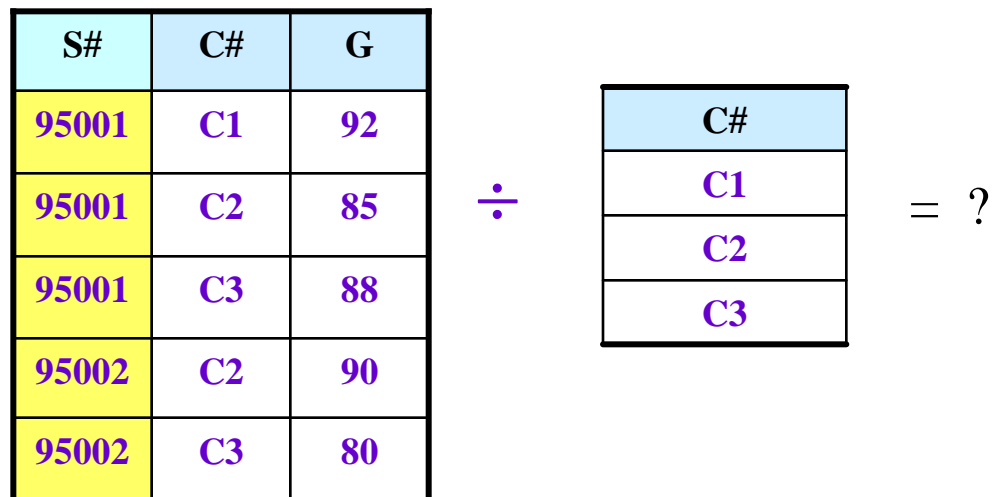
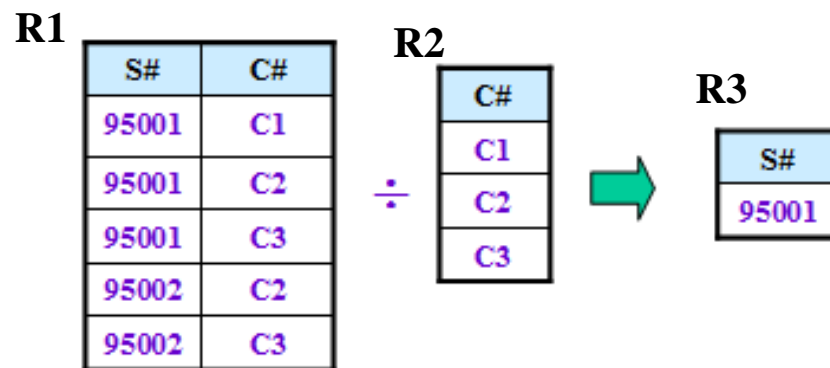
- 例2：求至少选修了这样一门课程的学生姓名，这门课的直接先行课为C2。

$$\Pi_{SN}(\sigma_{PC\#='C2'}(C) \bowtie SC \bowtie \Pi_{S\#,SN}(S))$$

- 例3：求选修了全部课程的学生学号和姓名。

$$\Pi_{S\#,C\#}(SC) \div \Pi_{C\#}(C) \bowtie \Pi_{S\#,SN}(S)$$

- 思考



关系代数小结

- 从数学角度，基本关系代数运算有5种：
并、差、乘、选择、投影
- 从数据库角度，核心的关系代数运算为：
选择、投影、连接（或自然连接）

选择

S#	C#	G
95001	C1	92
95001	C2	85
95001	C3	88
95002	C2	90
95002	C3	80



S#	C#	G
95001	C1	92
95001	C2	85
95001	C3	88

投影



S#	C#	G
95001	C1	92
95001	C2	85
95001	C3	88
95002	C2	90
95002	C3	80



S#
95001
95001
95001
95002
95002
95002



S#
95001
95002

S

S#	SN	SA	SD
95001	李勇	20	CS
95002	刘晨	19	IS
95003	王敏	18	MA
95004	张立	19	IS

CS

S#	C#	G
95001	C1	92
95001	C2	85
95001	C3	88
95002	C2	90
95002	C3	80



CSS

S#	SN	SD	C#	G
95001	李勇	CS	C1	92
95001	李勇	CS	C2	85
95001	李勇	CS	C3	88
95002	刘晨	IS	C2	90
95002	刘晨	IS	C3	80

连接

元组关系演算与域关系演算

- 把谓词演算应用到关系运算中就是关系演算。
 - 以元组为变量，简称元组演算；
 - 以域为变量，简称域演算。

元组关系演算

- 元组关系演算的基本结构是元组演算表达式。元组关系表达式的形式定义： $\{t / \Phi(t)\}$ ，表示了所有使 Φ 为真的元组的集合，即表示了一个关系。

其中：

- t 为元组变量。如果元组变量前有“全称”(\forall)或“存在”(\exists)量词，则称其为约束元组变量，否则称为自由元组变量。
- $\Phi(t)$ 是元组关系演算公式，由原子公式和运算符组成，简称公式。

元组关系演算公式的递归定义 (1)

(1) 原子命题函数是公式，称为原子公式。原子公式有三类：

- $R(t)$ 。 t 是关系 R 中的一个元组。
- $t[i] \theta u[j]$ 。 $t[i]$ 与 $u[j]$ 分别为 t 的第 i 个分量和 u 的第 j 个分量，它们之间满足比较关系 θ 。
- $t[i] \theta c$ 或 $c \theta t[i]$ 。分量 $t[i]$ 与常量 c 之间满足比较关系 θ 。

(2) 如果 Φ_1, Φ_2 是公式，则 $\Phi_1 \wedge \Phi_2, \Phi_1 \vee \Phi_2, \neg \Phi$ 也是公式。

(3) 如果 Φ 是公式，则 $\exists t(\Phi)$ 和 $\forall t(\Phi)$ 也是公式。

元组关系演算公式的递归定义 (2)

(4)在元组演算公式中，各种运算符的优先次序为：

- 算术比较运算符最高。
 - 量词次之，且 \exists 的优先级高于 \forall 。
 - 逻辑运算符最低，且 \neg 优先级高于 \wedge ， \wedge 高于 \vee 。
 - 如果有括号，则括号中的运算优先级最高。
- 按照上述4个规则对元组公式进行有限次复合，可以得到元组演算的所有公式。

示例

- 例：设关系R, S, W, 求如下元组表达式的结果：

R	A1	A2	A3	S	A1	A2	A3	W	B1	B2
	a	e	8		a	e	8		4	x
	c	f	6		b	c	5		5	d
	d	b	4		d	b	4			
	d	f	3		d	f	6			

(1) $R1 = \{t | R(t) \wedge t[3] \geq 4\};$

(2) $R2 = \{t \mid (\exists u)(R(t) \wedge W(u) \wedge t[3] < u[1])\};$

(3) $R3 = \{t | R(t) \wedge S(t)\};$

结果： R1

A1	A2	A3
a	e	8
c	f	6
d	b	4

R2

A1	A2	A3
d	b	4
d	f	3

元组演算与关系代数的等价性

用关系演算表达五种基本运算：

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

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

— 笛卡儿积：

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

— 投影：

$\prod_{i_1, i_2, \dots, i_k} (R) = \{ t^{(k)} \mid (\exists u)(R(u) \wedge t[1] = u[i_1] \wedge \dots \wedge t[k] = u[i_k]) \}$

— 选取： $\sigma_F(R) = \{ t \mid R(t) \wedge F' \}$ ， F' 是 F 用 $t[i]$ 代替原运算对象 i 得到的等价公式。

示例

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

用关系代数与元组关系演算表达下列查询：

- 例1：查询计算机系（CS）的全体学生

$$\sigma_{SD='CS'}(S); \quad \{t \mid S(t) \wedge t[4]='CS'\}$$

- 例2：查询年龄小于20岁的学生

$$\sigma_{SA < 20}(S); \quad \{t \mid S(t) \wedge t[3] < 20\}$$

- 例3：查询学生的姓名和所在的系

$$\Pi_{SN, SD}(SD);$$
$$\{t^{(2)} \mid (\exists u)(S(u) \wedge t[1] = u[2] \wedge t[2] = u[4])\}$$

域关系演算

- 域关系演算类似于元组演算。不同的是公式中的变量是对应元组各个分量的域变量。
- 域演算表达式的形式定义：

$\{ (x_1 x_2 \dots x_k) \mid \Phi (x_1, x_2, \dots, x_k) \}$ ，表示所有使 Φ 为真的那些 (x_1, x_2, \dots, x_k) 组成的元组集合，即表达了一个关系。

其中：

- x_i 代表域变量，如果域变量前有“全称” (\forall) 或“存在” (\exists) 量词，则称其为约束域变量，否则称为自由域变量。
- Φ 为域关系演算公式，由原子公式和运算符组成。

域演算公式的递归定义

- 域演算有三种形式的原子命题函数或称原子公式：
 - $R(x_1, x_2, \dots, x_k)$ 。 (x_1, x_2, \dots, x_k) 是 R 的一个元组， x_i 是域变量或常量。
 - $x_i \theta y_j$ 。域变量 x_i 与 y_j 之间满足比较关系 θ 。
 - $x_i \theta c$ 或 $c \theta x_i$ 。域变量 x_i 与常量 c 之间满足比较关系 θ 。
- 域演算与元组关系演算具有相同的运算符，相同的公式递归定义。

示例

- 例：设关系R，S分别如下图所示，现给出域演算公式，求结果关系。

R

A1	A2	A3
d	ce	5
d	bd	2
g	ef	7
d	cd	9

S

A1	A2	A3
c	bd	7
c	he	3
b	cf	6
d	cd	9

$$(1) R1 = \{xyz | R(xyz) \wedge z < 8 \wedge x = d\}$$

$$(2) R2 = \{xyz | (R(xyz) \vee S(xyz)) \wedge x \neq c \wedge y \neq cd\}$$

结果： R1

A1	A2	A3
d	ce	5
d	bd	2

R2

A1	A2	A3
d	ce	5
d	bd	2
g	ef	7
b	cf	6



关系运算的安全约束

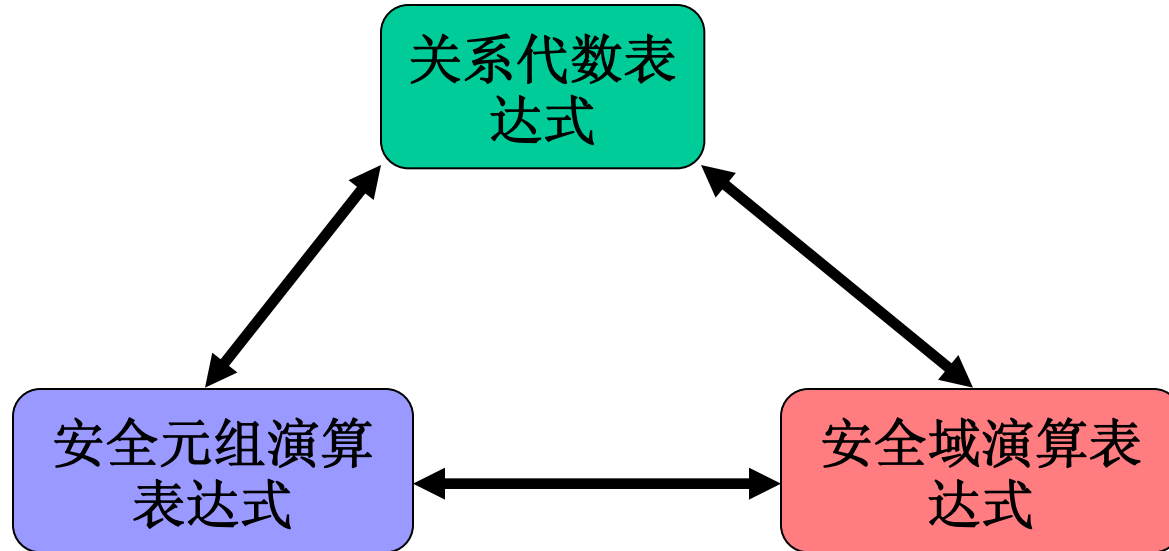
- 关系运算中把不产生无限关系和无穷验证的运算称为安全运算；其运算表达式称为安全表达式；对其所采取的限制称为安全约束。
- 关系代数是安全运算，关系演算则不一定是。所以对关系演算要进行安全约束。

关系运算的安全约束

- 在关系演算中，通常采用的安全约束方法是对 Φ 定义一个有限的符号集 $DOM(\Phi)$ ，使 Φ 的运算结果及中间结果所产生的关系及其元组的各个分量都必须属于 $DOM(\Phi)$ 。
- 设 Φ 是一个元组关系演算公式，为 $DOM(\Phi)$ 是由如下两类符号构成的集合：
 - Φ 中的所有常量
 - Φ 中出现的所有元组的所有分量值

三类关系运算的等价性

- 经过安全约束后的三类关系运算的表达能力是等价的，可以相互转换。



- 每一个关系代数表达式有一个等价的安全元组演算表达式；
- 每一个安全元组演算表达式有一个等价的安全域演算表达式；
- 每一个安全域演算表达式有一个等价的关系代数表达式。



关系数据库语言概述

- 数据库数据语言
- 关系数据库语言特点
- 关系数据库语言分类

数据库数据语言

- 数据库数据语言从功能上一般分为以下几种：
 - 数据定义（描述）语言（Data definition or description language），包括模式DDL，外模式DDL，内模式DDL；
 - 数据操纵语言（Data Manipulation Language）DML
 - 数据库有四种基本操作：检索、插入、修改、删除
 - DML有联机交互方式和宿主语言方式。
 - 联机交互方式下的DML称为自含式语言，可独立使用，适用于终端直接查询；
 - 宿主语言方式下的DML称为嵌入式语言，依附于宿主语言，是嵌入高级语言的程序中，以实现数据操作。
 - 数据控制语言（Data Control Language）DCL,完成数据库的安全性控制、完整性控制、并发控制等。

关系数据语言的特点 (1)

- 一体化
 - 将数据的定义、查询、更新、控制等功能融为一体，只给用户提供一种称之为“查询语言”的语言。便于用户学习与使用。
- 非过程化
 - 用户只需提出“干什么”，而“怎样干”由DBMS解决。所以语言操作简单，易学、易用。
- 面向集合的存取方式
 - 操作对象是一个或多个关系，操作的结果也是一个新关系。
- 既可独立使用又可与主语言嵌套使用

关系数据语言的特点 (2)

- 关系数据语言优越性的根源：
 - 关系模型采用了最简单、最规范化的数据结构，这使DML大大简化；
 - 关系数据语言是建立在关系运算的数学基础上，可实现关系的垂直方向和水平方向的任意分割和组装操作，使得关系语言可随机地构造出用户需要的各种各样的新关系。

关系数据语言的分类 (1)

- 关系数据语言的核心是查询，所以又称为**查询语言**。而查询往往表示成一个关系运算表达式，因此关系运算是设计关系数据语言的基础，**关系运算的分类也决定了关系语言的分类**。
- 关系数据语言目前常用分类：

采用的关系运算		语言名称	
关系代数		ISBL	SQL
关系演算	元组关系演算	ALPHA; QUEL	
	域关系演算	QBE	

关系数据语言的分类 (2)

- ISBL(Information System Base Language), 关系代数语言的代表。英格兰底特律的IBM英国科学中心研究。应用于实验系统PRTV上。
- QUEL (Query Language) , 基于元组演算的语言。美国加利福尼亚大学研制。用于Ingres数据库。
- QBE(Query By Example), 基于域演算的表格显示语言。用于QBE数据库。
- SQL(Structured Query Language), 介于关系代数与关系演算之间的语言, 标准的关系数据语言。

第三章 关系数据库标准语言SQL

- 概述
- SQL数据查询功能
- SQL数据定义功能
- SQL视图操作
- SQL数据更新
- SQL数据控制
- 嵌入式SQL

SQL概述

- SQL的产生与发展
 - 1974年，由Boyce和Chamber提出，称为SEQUEL (Structured English Query Language)；
 - 1975-1979年，在IBM的San Jose研究室研制的System R上实现；
 - 1981年，IBM在推出SQL/DS关系数据库时，将其命名为SQL (Structured Query Language)；
 - 随着SQL语言应用的日益广泛，ANSI和ISO先后制定了SQL-86、SQL-89、SQL-92、SQL-99、SQL-2003等多个SQL标准。

SQL特点

- 综合统一
- 高度非过程化
- 面向集合的操作方式
- 以同一种语法结构提供两种使用方式
- 语言简捷，易学易用

SQL功能	操作符
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

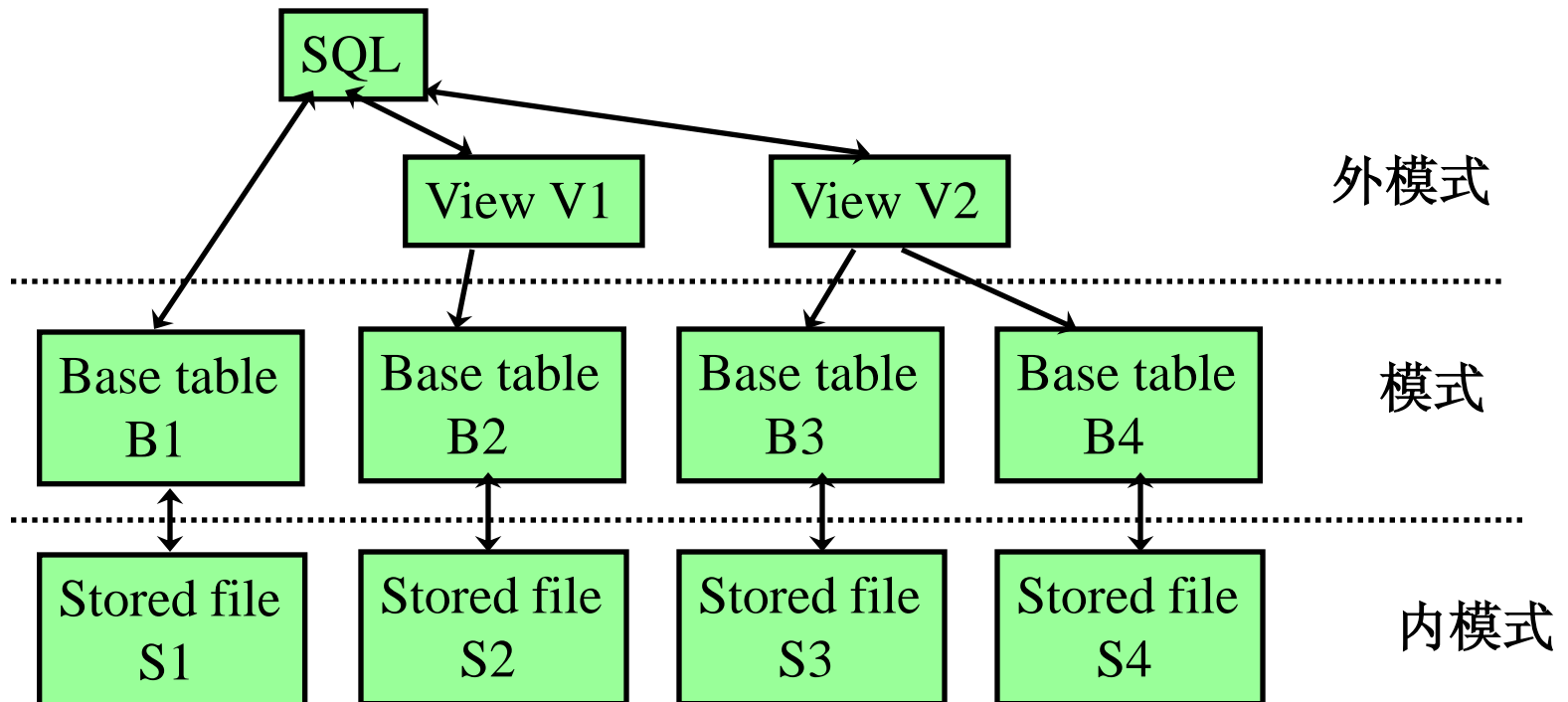
SQL语言的基本概念

- 基本表与导出表

- 基本表：是实际存在的，每个表在存储中可用一个存储文件来表示。
- 导出表：是从基本表导出的表，有视图（View）和快照(Snapshot)。
 - 视图是一个**虚表**。即视图所对应的数据不实际存储在数据库中，只在数据库的数据字典中存储视图的定义。
 - 视图一经定义就可以和基本表一样进行查询等操作，也可以用来定义新的视图。

SQL语言的基本概念

- 关系数据库的三级模式结构



SQL数据查询功能

- 查询的基本结构是 **SELECT-FROM-WHERE** 组成的查询块。一般形式：

SELECT 目标列

要检索的数据项

FROM 基本表（或视图）

要操作的关系名，
1个或多个

WHERE 检索条件

查询结果应满
足的条件表达
式

- 查询块的结果仍是一个表；
- 查询块执行的过程是在表的水平方向上按“检索条件”选取元组，又在垂直方向上按SELECT指定的列进行投影；
- 查询块可进行关系代数中投影、选取、连接等操作的组合。

示例用表

学生表 S

S#	SN	SA	SD
s1	李勇	20	CS
s2	刘晨	19	IS
s3	王敏	18	MA
s4	张立	19	IS

学生选课表SC

S#	C#	G
s1	C1	92
s1	C2	85
s1	C3	88
s2	C2	90
s2	C3	80

课程表C

C#	CN	PC#
C1	数据库	C5
C2	数学	
C3	信息系统	C1
C4	操作系统	C6
C5	数据结构	C7
C6	数据处理	
C7	Pascal语言	C6

投影检索

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- **SELECT-FROM-WHERE** 查询块中，没有 **WHERE** 子句，是单纯的投影操作。

例1：检索学生的姓名，年龄

```
SELECT SN, SA  
FROM S ;
```

- 采用 **DISTINCT** 消去 **SELECT** 结果中的重复行。

例2：检索学生所选修课程的课程号

```
SELECT DISTINCT C#  
FROM SC ;
```


选取检索

```
S(S#,SN,SA,SD);  
C(C#,CN,PC#);  
SC(S#,C#,G)
```

- 由 WHERE 子句指出查询条件。
- 检索条件可以包括如下运算符：
 - 比较运算符：=, <> (!=), >, >=, <, <=
 - 布尔运算符：AND, OR, NOT
 - ()

例1：检索选修C2课程的所有学生的学号和成绩。

```
SELECT S#, G  
FROM SC  
WHERE C#='C2';
```

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

例2：检索选修C1或C2且成绩高于70分的学生学号、课程号和成绩。

```
SELECT S#, C#, G  
FROM SC  
WHERE (C#='C1' OR C#='C2') AND G>=70;
```

例3：检索成绩在70至85分之间的学生学号、课程号和成绩。

```
SELECT S#, C#, G  
FROM SC  
WHERE G BETWEEN 70 AND 85;
```

排序检索

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 在SELECT-FROM-WHERE查询块后接**ORDER BY**子句，将结果按指定列排序。

格式：**ORDER BY 列名 ASC 或DESC**

- ASC为升序；DESC为降序，缺省为升序。
- 可以是单列排序或多列排序
- 该子句在SELECT语句中作为最后一个子句出现。

例1：检索全体学生信息，并按系号升序，同一个系按年龄降序排列。

```
SELECT *  
FROM S  
ORDER BY SD , SA DESC;
```

连表检索 (1)

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 将多个**相互关联**的表按照一定条件连接起来，实现多表数据检索。
- **SELECT-FROM-WHERE**语句块结构：
 - SELECT**——指明选取的列名（来自多个表）
 - FROM**——指明要进行连接的表名
 - WHERE**——指明**连接条件**（连接谓词）与**选取条件**。
 - 连接条件一般格式为：

[<表名>.]<列名><比较运算符> [<表名>.]<列名>

其中，<列名>称为连接字段；<比较运算符>主要有：=，>，<，>=，<=，!=

连表检索 (2)

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

例1：检索学生张华所学课程的成绩。

```
SELECT SN, C#, G  
FROM S, SC  
WHERE S.S#=SC.S# AND SN='张华' ;
```

连表条件

选取条件

注：如果连接的表中有属性名相同，要用表名作前缀加以区分。

连表检索 (3)

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 表自身的连接
 - 通过定义别名，将一个表看成两个表，进行连接。

例2：检索所有比李勇年龄大的学生姓名、年龄。

```
SELECT X.SN, X.SA  
FROM S X, S Y  
WHERE X.SA>Y.SA AND Y.SN='李勇';
```

连表检索 (4)

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 外连接

- 在连接谓词某一边加 (*或+), 则逻辑上为 *所在边的表增加了一个空行。它可以与另一个表中所有不满足连接条件的元组进行连接, 使这些元组能够输出。

例3: 检索所有学生的全部信息。

```
SELECT *  
FROM S, SC  
WHERE S.S#=SC.S#(*);
```

子查询嵌套检索 (1)

- WHERE子句中可以包含另一个查询块，该查询块称为子查询或嵌套查询，包含子查询的语句称为外部查询。
- 外部查询利用子查询来获取检索条件的条件值，检索条件根据子查询的结果来确定外部查询的结果数据。
- 子查询按照与外部查询的联系不同，分为普通子查询和相关子查询。
 - 普通子查询：与外部查询无关，可单独执行得一组值。
 - 相关子查询：把外查询的列值作为检索条件的条件值。

子查询嵌套检索 (2)

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

• 涉及同一个表的子查询

例1: 检索与李勇同岁的学生姓名。

```
SELECT SN  
FROM S  
WHERE S.SA=  
      (SELECT SA FROM S  
       WHERE SN='李勇' );
```

- 如果子查询返回单值，可以直接用比较运算符 =, <>, >, >=, <, <= 等连接子查询。
- 如果子查询返回一组值，则必须在比较运算符和子查询之间插入 ANY 和 ALL 等操作符。

子查询嵌套检索 (3)

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

例2：检索选修C2课程的学生姓名。

```
SELECT SN  
FROM S  
WHERE S#=ANY  
      (SELECT S# FROM SC  
       WHERE C#='C2');
```

例3：检索选修C2课程的成绩最高的学生学号。

```
SELECT S#  
FROM SC  
WHERE C#='C2' AND G>=ALL  
      (SELECT G FROM SC  
       WHERE C#='C2');
```

子查询嵌套检索 (4)

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 用IN检索
 - IN在嵌套子查询中最常使用。可代替“=ANY”，是集合运算中的“ \in ”运算。

例4：检索选修C2课程的学生姓名。

```
SELECT SN
FROM S
WHERE S# IN
      (SELECT S# FROM SC
       WHERE C#='C2');
```

- 用NOT IN检索
 - NOT IN表示不在集合中，与 \neq ALL相同。

子查询嵌套检索 (5)

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 用EXISTS检索
 - EXISTS表示存在量词“ \exists ”。
 - 表达式 **EXISTS (子查询)**
当且仅当子查询结果为非空时为真。

例5：检索选修C2课程的学生姓名。

```
SELECT SN
FROM S
WHERE EXISTS
      (SELECT * FROM SC
       WHERE S#=S.S# AND C#='C2');
```

- 用NOT EXISTS 检索
 - 表示“不存在”。
 - 表达式 **NOT EXISTS (子查询)** 在子查询结果为空时为真。

子查询嵌套检索 (6)

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 用NOT EXISTS表达全称量词 (\forall)
 - 任何一个带有全称量词的谓词总可以转换为等价的带存在量词的谓词:
 - $(\forall x)P \equiv \neg (\exists x (\neg P))$

例6: 检索选修所有课程的学生姓名。

本题等价于“检索这样的学生的姓名, 不存在他不选修的课程”。

```
SELECT SN  
FROM S  
WHERE NOT EXISTS
```

```
(SELECT * FROM C  
WHERE NOT EXISTS  
检索S.S#选修  
C.C#的记录 { (SELECT * FROM SC  
WHERE S#=S.S# AND C#=C.C#));
```

检索s.s# 没有选修的课程 {

子查询嵌套检索 (7)

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 用NOT EXISTS 表达蕴涵

$$- p \rightarrow q \equiv \neg p \vee q$$

$$(\forall y) p \rightarrow q \equiv \neg (\exists y (\neg (p \rightarrow q))) \equiv \neg (\exists y (\neg (\neg p \vee q))) \equiv \neg \exists y (p \wedge \neg q)$$

例7: 检索至少选修了学生S2选修的全部课程的学生学号。

本题用蕴含表达:

“检索这样学生的学号, 对于任意课程, 只要S2选修, 他就选修。”

本题等价于“检索这样学生的学号, 不存在S2选修而他没选修的课程”。

SELECT DISTINCT S#

FROM SC SCX

WHERE NOT EXISTS

(SELECT * FROM SC SCY

WHERE SCY.S#='S2' AND NOT EXISTS

(SELECT * FROM SC SCZ

WHERE S#=SCX.S# AND C#=SCY.C#));

检索S2选修而SCX.S#
没有选修的课程

检索SCX.S#选修SCY.C#的记录

并、差、交检索 (1)

- 并、差、交的SQL运算符：
 - 并：UNION
 - 差：MINUS
 - 交：INTERSECT
- 并、差、交检索的操作对象必须是相容的，即必须有相同数量的属性列，且相应属性列的域也必须相同。

并、差、交检索 (2)

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

例1：检索选修了C1或C2课程的学生学号。

```
SELECT S#    FROM SC WHERE C#='C1'  
UNION  
SELECT S#    FROM SC WHERE C#='C2';
```

例2：检索无人选修的课程号和名称。

```
SELECT C#, CN FROM C WHERE C# IN  
(SELECT C# FROM C  
MINUS  
SELECT DISTINCT C# FROM SC) ;
```


库函数检索 (1)

- 库（集）函数
 - COUNT() 按列值计个数，COUNT(*)对行计数。
 - SUM() 对数值列求总和
 - AVG() 求数值列的平均值
 - MAX() 在列中找出最大值
 - MIN() 在列中找出最小值

库函数检索 (2)

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

例1: 检索学生总数。

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

例2: 检索选修了课程的学生人数。

```
SELECT COUNT (DISTINCT S#) FROM SC ;
```

例3: 求学号为S4的学生的总分和平均分。

```
SELECT SUM (G) , AVG (G)  
FROM SC  
WHERE S#='S4';
```

例4: 检索选修了C1课程的学生最高分。

```
SELECT MAX (G) FROM SC WHERE C#='C1';
```

分组检索 (1)

- 按属性列（列组）将关系的元组分组，每组在这些分组属性列（列组）上具有相同值，对每一组可执行SELECT操作。
- 分组子句：

GROUP BY 列名

[HAVING 条件表达式] —— 分组条件

- **WHERE子句与HAVING子句**
 - WHERE子句是针对“行”进行，用于去掉不符合条件的若干行；HAVING子句针对“分组”进行，必须和GROUP BY连用，用于去掉不符合条件的若干分组。
 - 在查询块中出现的顺序：WHERE — GROUP BY — HAVING

分组检索 (2)

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

例1：检索至少选修三门课程的学生学号和选课门数。

```
SELECT S#, COUNT (*)  
FROM SC  
GROUP BY S#  
HAVING COUNT (*) >=3;
```

例2：求选修四门以上课程的学生学号和总成绩（不统计不及格的课程）。最后按降序列出总成绩排序名单。

```
SELECT S#, SUM (G)  
FROM SC  
WHERE G>=60  
GROUP BY S#  
HAVING COUNT (*) >=4  
ORDER BY SUM(G) DESC;
```

算术表达式值的检索

- SELECT子句中，可包括由属性列、常数、库函数、算术运算符+、-、*、/等组成的算术表达式。
- 检索结果数据项名可用表达式表示或用“别名”来表示。

例1：有职工表EMP (EMP#, EMPN, JOB, SALARY, BONUS, DEPT)，要求检索所有PROGRAMMER的奖金大于工资25%的职工姓名和一年的总收入，并按奖金与工资之比的降序排列。

```
SELECT EMPN, BONUS/SALARY BS ,  
        12* (SALARY+BONUS) TOTAL  
FROM EMP  
WHERE JOB='PROGRAMMER'  
      AND BONUS>0.25*SALARY  
ORDER BY BONUS/SALARY DESC ;
```

部分匹配查询

- 使用谓词LIKE 或NOT LIKE，一般形式：
<列名> LIKE/NOT LIKE <字符串常量>
 - “列名” 必须为字符型或变长字符型。
 - “字符串常量” 可包含两个特殊符号 % 与 _
 - %：代表任意序列的0个或多个字符；
 - _：代表任意单个字符

例1：检索所有姓刘的学生的学号、姓名。

```
SELECT S#, SN
FROM S
WHERE SN LIKE '刘%';
```



SQL数据定义功能

- 定义、删除、修改基本表
- 定义、删除索引
- 定义、删除视图

操作对象	操作方式		
	创建	删除	修改
表	Create Table	Drop Table	Alter Table
视图	Create View	Drop View	
索引	Create Index	Drop Index	

定义基本表

- 定义基本表

Create Table <表名>

(<列名><数据类型>[<列级完整性约束>]

[[,<列名><数据类型>[<列级完整性约束>]]]

[[,<表级完整性约束>]]);

- 完整性约束

- NULL/NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK

- SQL92的数据类型

- char (n) : 固定长度的字符串。
- varchar (n) : 可变长字符串。
- int: 整数。
- smallint: 小整数类型。
- numeric (p, q) : 定点数共p位, 小数点右边q位。
- Real, double precision : 浮点数与双精度浮点数, 精度与机器有关。
- Float(n): n位的精度浮点数。
- date: 日期 (年、月、日)。
- time: 时间 (小时、分、秒)。
- interval: 两个date或time类型数据之间的差

示例

- **Create table S**
(S# char(5) not null unique,
SN char(20) not null,
SA int,
SD char(15),
Primary key(S#),
Check (SA >=18 and SA <=45));
- **Create table SC**
(S# char(5) not null,
C# char(5) not null,
G number(4,2),
Primary key (S#, C#),
Foreign key (S#) references S (S#),
Foreign key (C#) references C (C#));

修改、删除基本表定义

- 修改基本表

- 语法: **Alter Table <表名>**
[Add <新列名><数据类型>[<完整性约束>]]
[Drop <完整性约束名>]
[Modify <列名><数据类型>]

- 示例

- 例1: S表增加“入学时间”属性

Alter Table S Add Scome Date ;

- 例2: 将SA的数据类型改为半字长整数

Alter Table S Modify SA Smallint

- 删除基本表

- 语法: **Drop Table <表名>**
- 示例: Drop Table S;

定义、删除索引

- 定义索引

- 语法: **Create [Unique][Cluster] Index <索引名>**
On <表名> (<列名>[次序][, <列名>[次序]] ...);
- 示例: **Create Unique Index Scno**
On SC(S# ASC, C# DESC)

- 删除索引

- 语法: **Drop Index <索引名>**
- 示例: **Drop Index Stusno;**



定义视图

- 定义视图

- 语法: **Create View <视图名>**
[(**<列名>**[,**<列名>**] ...)]
As <子查询>
[With Check Option]

- 示例: 建立计算机系的学生视图

```
Create View CS_Student  
As  
SELECT S#, SN, SA FROM S  
WHERE SD = 'CS'
```

```
S(S#,SN,SA,SD);  
C(C#,CN,PC#);  
SC(S#,C#,G)
```

- 删除视图

Drop View <视图名>

查询视图

- 视图消解 (View Resolution)

DBMS执行对视图的查询时，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**，然后再执行修正的查询。这一转换过程称为**视图消解**。

视图查询示例

- 在计算机系学生的视图中找出年龄小于20岁的学生。

Create View CS_Student

As

SELECT S#, SN, SA FROM S

WHERE SD = 'CS'

Select S#, SA From CS_Student

Where SA < 20;

- 转换后为：

Select S#, SA From S

Where SD='CS' And SA <20;

视图的作用

- 能够简化用户操作

— 例：

```
Select S#, SA From IS_Student
```

```
Where SA < 20;
```

```
Select S#, SA From S
```

```
Where SD='IS' And SA <20;
```

- 使用户能够以多种角度看待同一数据
- 提供了一定程度的逻辑独立性
- 能够对机密数据提供安全保护



SQL数据更新

- 插入数据——Insert语句
- 修改数据——Update语句
- 删除数据——Delete语句

插入数据

- 插入单个元组

Insert Into <表名>[(<属性列>[{,<属性列>}])]
Values(<值>[{,<值>}])

- 插入子查询结果

Insert Into <表名>[(<属性列>[{,<属性列>}])]
<子查询>

示例

```
S(S#,SN,SA,SD);  
C(C#,CN,PC#);  
SC(S#,C#,G)
```

例1: Insert Into S

Values ('S10', '陈冬' , 'IS', 18);

例2: Insert Into SC (S#, C#)

Values('S20', 'C1');

例3: Insert Into Dept_Age (Sdept, Avgage)

Select SD, AVG(SA) From S

Group By SD;

修改数据

- 语法： Update <表名>
Set <列名>=<表达式>[{, <列名>=<表达式>}]
[Where <条件>]
- 示例
 - 例1：将学生S1的年龄改为22岁
Update S
Set SA = 22
Where S#='S1'

示例

- 例2：将所有学生的年龄增加1岁

Update S

Set SA=SA+1

- 例3：将计算机系全体学生的成绩置零。

Update SC

Set G = 0

Where 'CS' = (Select SD From S

Where S.S# = SC.S#)

删除数据

- 语法： **Delete From** <表名> [Where <条件>]
- 示例：
 - 例1：删除学号为S19的学生的记录
Delete From S Where S#='S19'
 - 例2：删除所有学生的选课记录
Delete From SC
 - 例3：删除计算机系所有学生的选课记录
Delete From SC
Where 'CS' = (Select SD From S
Where S .S# = SC.S#)



SQL数据控制功能

- 定义完整性约束条件
- 支持事务操作
- 提供安全控制功能

— 授权

- **GRANT** <权限> [**ON** <对象类型> <对象名>]
TO <用户>

— 收回权限

- **REVOKE** <权限> [**ON** <对象类型> <对象名>]
FROM <用户>



嵌入式SQL

- 嵌入式SQL的意义
 - 嵌入式SQL把SQL的最佳特性与程序设计语言的最佳特性(如过程处理能力) 结合起来, 使SQL功能更强, 灵活性更强。
 - 实际的应用系统是非常复杂的, 数据库访问只是其中一个部件。有些动作如与用户交互、图形化显示数据等只能用高级语言实现。

嵌入式SQL程序示例

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL BEGIN DECLARE SECTION;
int var_c1;
char var_c2[21] = { 0x00 };
EXEC SQL END DECLARE SECTION;

int main(int argc, char* argv[])
{
    /* 步骤1: 建立连接 */
    EXEC SQL WHENEVER SQLERROR SQLPRINT; /* 声明异常发生时的处理动作统一为打印消息 */
    EXEC SQL CONNECT TO postgres@localhost:5432 USER postgres/xxxx; /* 指定连接的目标数据库, 用户名, 密码 */

    /* 步骤2: 利用游标执行查询 */
    EXEC SQL DECLARE foo_bar CURSOR FOR SELECT c1, c2 FROM tb1; /* 声明一个游标使之用于执行SELECT文 */
    EXEC SQL OPEN foo_bar; /* 打开游标从而使SELECT文被执行 */
    EXEC SQL FETCH foo_bar INTO :var_c1, :var_c2; /* 获取一行数据 */

    printf("C1: %d, C2: %s\n", var_c1, var_c2);

    EXEC SQL CLOSE foo_bar; /* 关闭所打开的游标 */

    /* 步骤3: 关闭连接 */
    EXEC SQL DISCONNECT CURRENT;

    return 0;
}
```

连接PostgreSQL数据库, 宿主语言是C

-
- 对于嵌入式SQL，DBMS多采用预编译方法进行处理。
 - 把嵌入在程序中的SQL语句翻译为高级语言（主语言）源码，然后按主语言的通常方式进行编译、连接形成可执行代码。

-
- 把SQL嵌入主语言使用时必须解决的问题：
 - 区分SQL语句与主语言语句。
 - 在SQL语句前加前缀 EXEC SQL解决。
 - 数据库工作单元和程序工作单元之间的通信。
 - 通过定义SQL通信区向主语言传递SQL语句执行状态信息；
 - 通过主变量（主语言程序变量）交换参数与数据。
 - SQL与主语言之间操作方式的协调。SQL是一次一集合。主语言是一次一记录（一组主变量一次只能存放一条记录）。
 - 通过游标解决。游标是系统开辟的存放SQL执行结果的缓冲区，由游标指针执行一组记录。程序通过游标逐一获取记录，并赋给主变量。

动态SQL

- 如果在预编译时SQL语句中的主变量、数据库对象等信息不能确定，就必须使用动态SQL技术。
- 动态SQL允许在程序运行过程中“临时”组装SQL语句，主要有三种形式：
 - 语句可变
 - 条件可变
 - 数据库对象，查询条件均可变

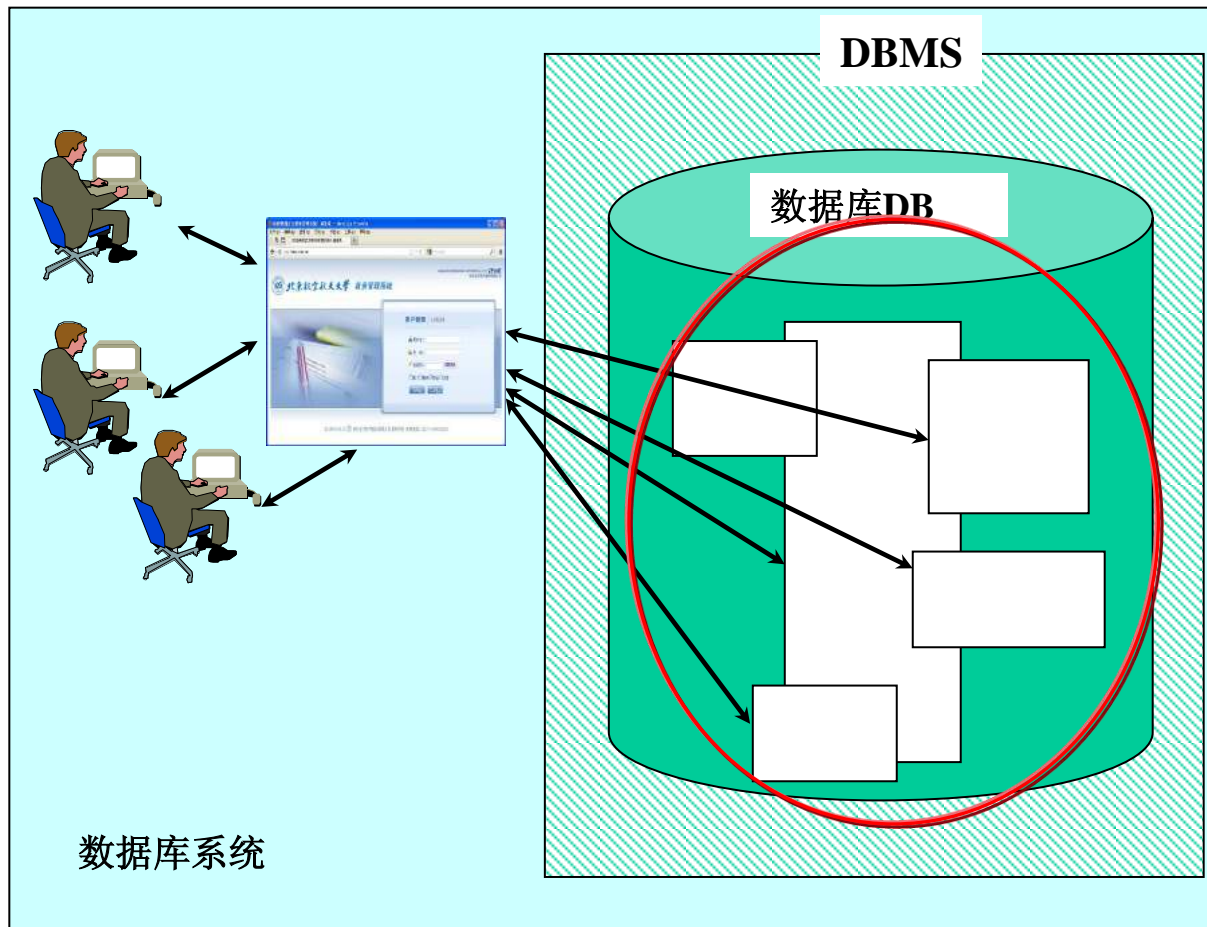
小结

- 概念
 - 基本表与视图；关系数据库的三级模式结构
 - SQL的特点
- SQL的数据定义、查询、更新、控制功能
- 视图的作用及SQL的视图操作

第四章 数据库设计

- 数据库设计概述
- 需求分析
- 概念结构设计
- 逻辑结构设计
- 物理结构设计

数据库设计概述



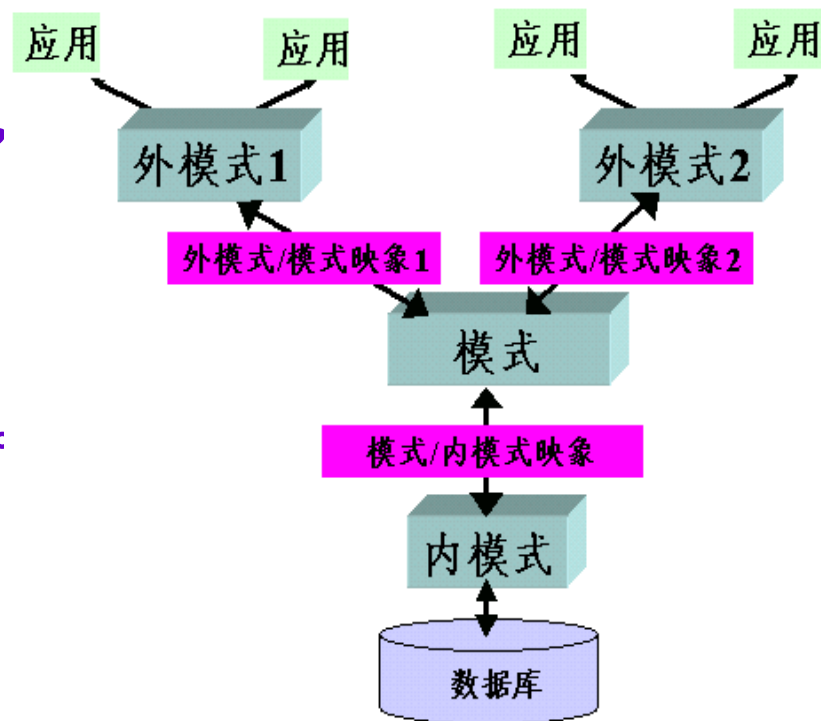
数据库设计概述

- 什么是数据库设计

- 是指对于一个给定的应用环境，设计优化的数据库逻辑和物理结构，建立数据库，使之能够有效地存储数据，为开发满足用户需求的应用系统奠定基础。

- 数据库设计的特点

- 要把数据设计和处理设计密切结合起来；
- 与硬件、软件和管理紧密相关。



数据库系统体系结构

数据库设计概述

- 数据库设计的方法 (1)

- **手工试凑法** (直接设计法) : 根据应用的数据要求与处理要求, 直接设计数据库的结构。缺点:

- 数据间关系复杂, 使用要求各式各样, 很难仅凭经验进行设计;
 - 把数据的逻辑结构, 物理结构、处理要求等一起考虑, 很难对模式进行评价和优化。用户需求一旦发生变化, 数据结构很难随之发生变化;
 - 数据库设计与具体的DBMS紧密结合, 移植困难;
 - 缺乏文档资料, 难于与用户交流, 对设计难于评审;
 - 难以由多个人合作进行设计。

数据库设计概述

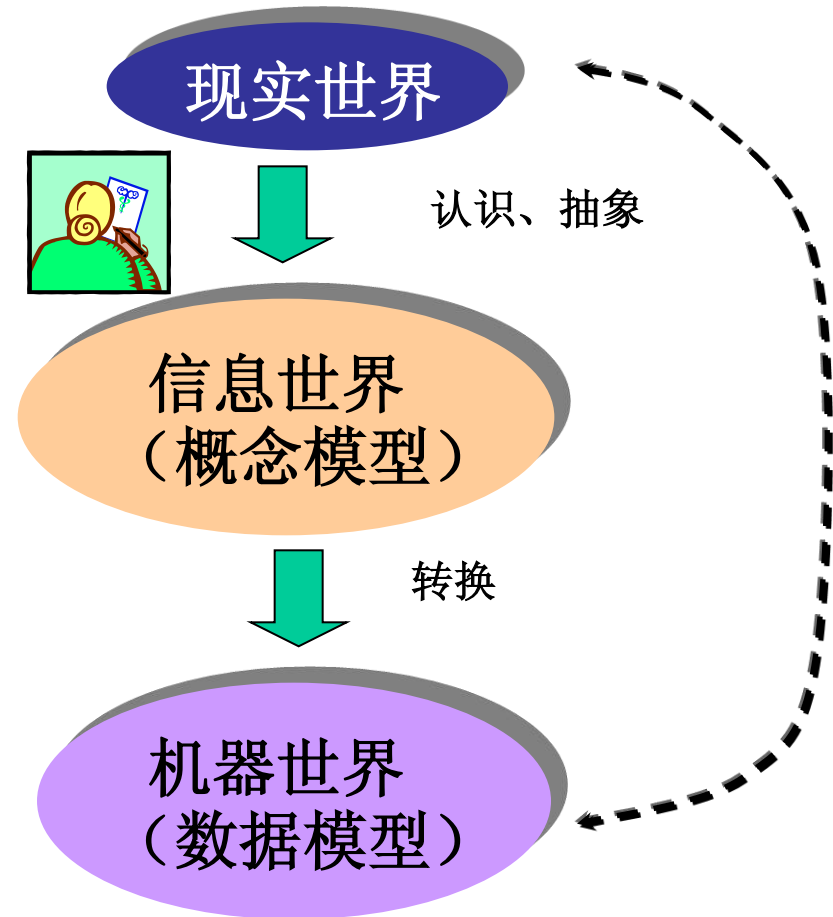
- 数据库设计的方法 (2)

- 规范设计法：运用软件工程的思想和方法，把整个设计过程划分为若干阶段，把复杂的大问题分为若干相对简单的小问题，每个阶段只解决整个设计中的部分问题。
- 规范设计法是迭代和逐步求精的过程，每一过程完成时要进行设计分析，产生各种设计文档，并组织评审和用户交流，如不满足要求则进行修改。

数据库设计的基本步骤

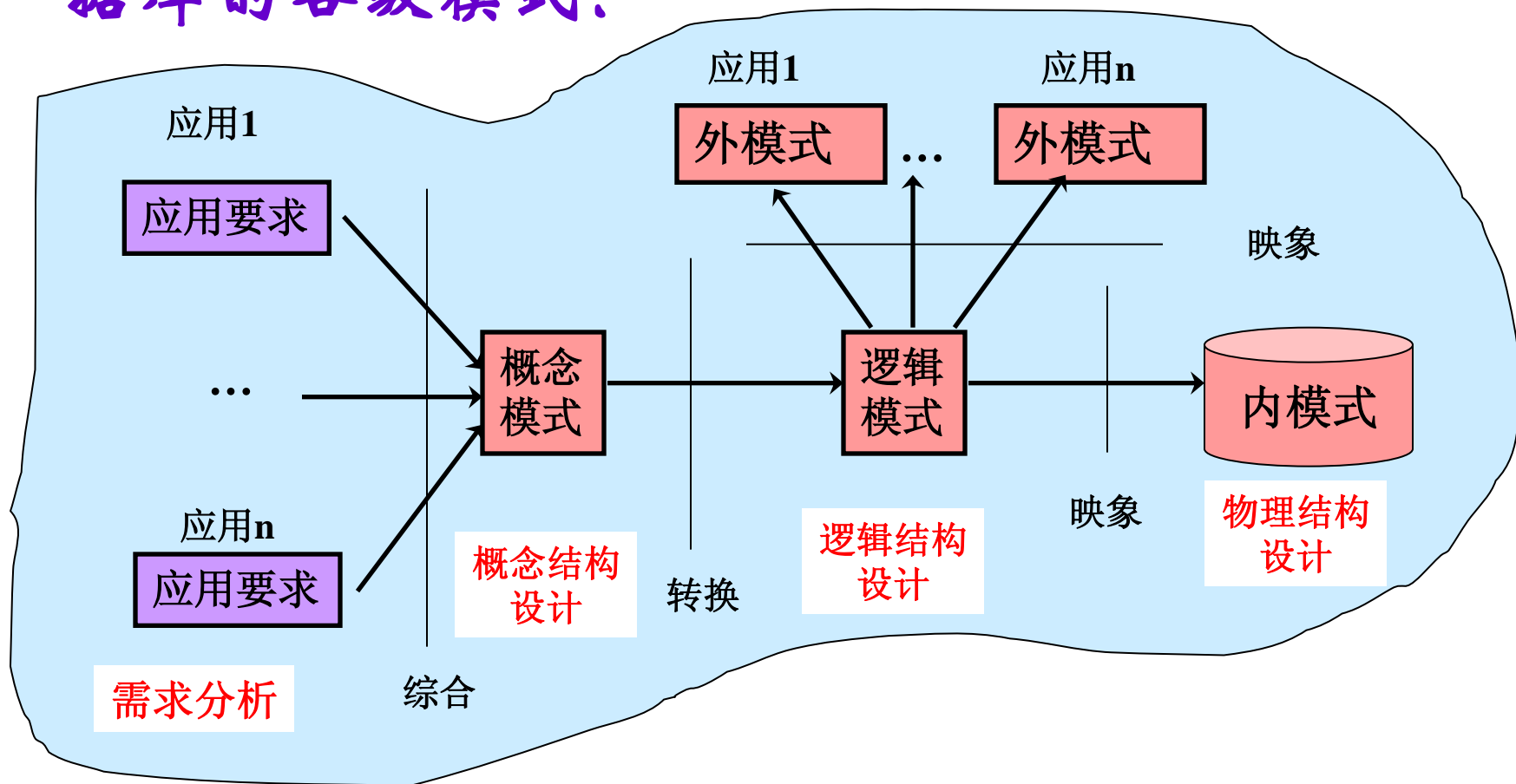
- 按照规范设计方法，数据库设计分为以下**六个阶段**：

- 需求分析；
- 概念结构设计；
- 逻辑结构设计；
- 物理结构设计；
- 数据库实施；
- 数据库运行和维护。



数据库设计的基本步骤

- 在六阶段数据库设计的不同阶段形成了数据库的各级模式：



数据库设计阶段概述 (1)

- 需求分析
 - 对应用环境进行详细调查，收集支持系统目标的基础数据及其处理；
- 数据库概念结构设计
 - 通过对用户需求进行综合、归纳与抽象，形成独立于数据库逻辑结构与具体DBMS的概念模型，可以用E-R图等表示；
- 数据库逻辑结构设计
 - 将概念结构转换为某个DBMS所支持的数据模型，并进行优化。再将得到的逻辑结构转换成特定的DBMS能处理的模式、子模式；

数据库设计阶段概述 (2)

- 数据库物理结构设计

- 设计数据库在物理设备上的**存储结构和存取方法**。一般分为两步：一是确定数据库的内模式；二是对物理结构进行时间与空间效率的评价；

- 数据库实施

- 是**建立数据库**的过程。用DBMS的DDL描述三级模式，并调试产生目标模式。开发应用程序，组织数据入库并试运行；

- 数据库运行和维护

- 在数据库正式运行后，由DBA执行**对数据库经常性的维护工作**，包括数据库转储与恢复、数据库控制、数据库性能监控、数据库的重组与重构。



需求分析的目标

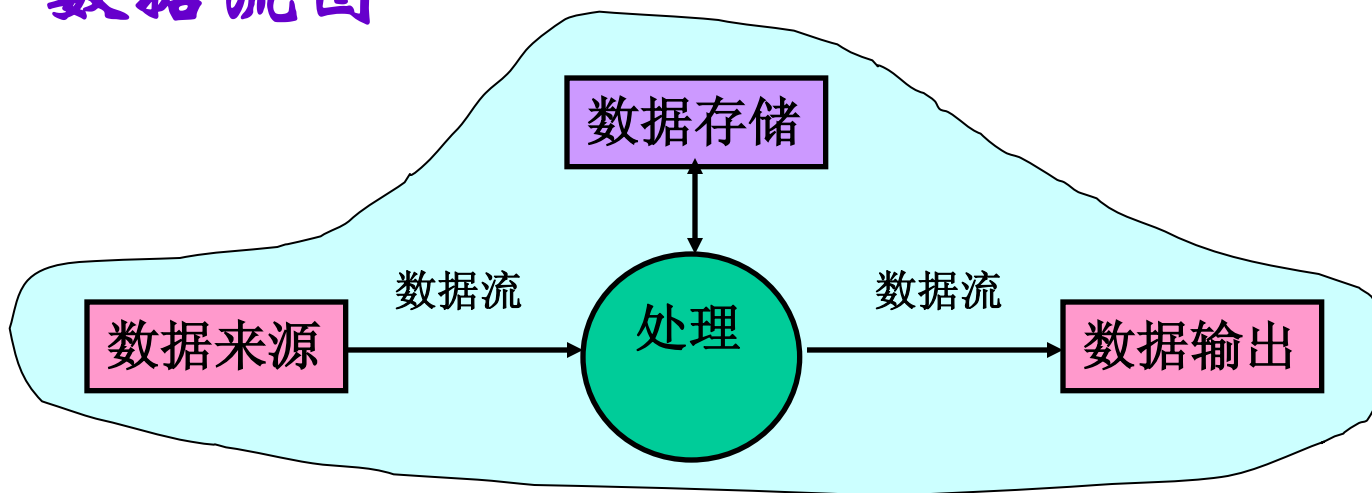
- 需求分析阶段的目标是收集支持系统应用目标的基础数据及其处理。调查的重点是“数据”和“处理”，包括：
 - 信息要求：指系统中所涉及的数据及数据之间的联系。具体收集数据的名称、类型、长度等，确定数据之间联系的类型。
 - 处理要求：指用户要完成什么处理功能，对处理的响应时间和处理方式的要求。
 - 安全性与完整性的要求

需求分析的方法

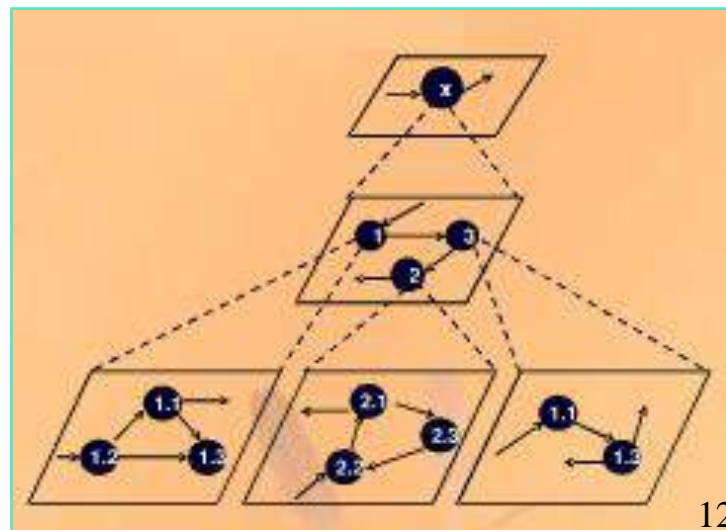
- 需求分析分为两个阶段：
 - 调查用户实际要求，与用户达成共识
 - 分析、表达用户的需求
 - 用数据流图表达数据和处理之间的关系；
 - 用数据字典描述系统中各类数据。

需求的表达 (1)

- 数据流图



- 采用结构化分析方法SA从最上层的系统组织机构入手，自顶向下逐步分解处理功能以及它们所用的数据，形成若干层次的数据流图。



需求的表达 (2)

- 数据字典

- 是对数据的集中的系列说明。包含每一个数据元素的名字、意义等各方面的描述。
 - 从数据流图中提取出所有原子数据项；
 - 把有联系的数据项组合起来形成数据组；
 - 以数据组为单位，写出数据项的如下定义：
 - 语义定义：名字，实际意义等
 - 类型定义：数据类型、数据宽度、小数位数等
 - 完整性约束定义：值约束、空值约束以及其他比较复杂的完整性约束。
 - 根据用户和实际领域的信息模型需要补充其他数据项及其定义。

需求的表达 (3)

数据组名:				
特 征	数据项名字	数据项名字	数据项名字	数据项名字
	编号:	编号:	编号:	编号:
数据类型				
数据宽度				
小数位数				
单 位				
值 约 束				
允许空值否				
值 个 数				

需求的表达 (4)

- 数据字典还可以包括数据在系统内的传输路径、数据存储位置和处理过程信息等。
- 需求分析阶段的数据字典，可看成是数据元素表。

在数据库实施阶段建立起的数据字典，是数据库系统重要组成部分。



概念结构设计

- 概念结构设计的工具—E-R法
- 概念结构设计的方法与步骤
- 基于E-R法的概念结构设计



E-R法概述

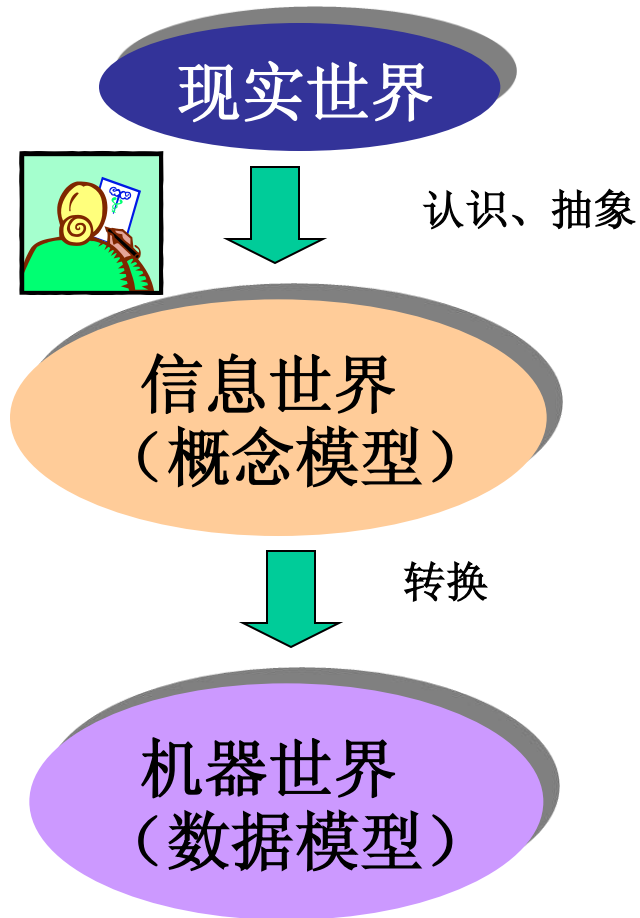
- Peter Chen在1976年提出E-R法（Entity-Relation Approach，实体联系方法）也称为E-R模型；



Dr. Peter Chen (陳品山) at
Louisiana State University (LSU)

- E-R法的思想是用**E-R图**描述现实世界的信息，这种信息结构称为**概念结构**，然后根据具体系统的要求将概念结构转换成特定系统所能接受的逻辑结构（层次、网状、关系）；
- E-R法由**两部分组成**：
 - 用E-R图描述现实世界；
 - 将E-R图转换成相应的数据模型。

概念模型的重要性



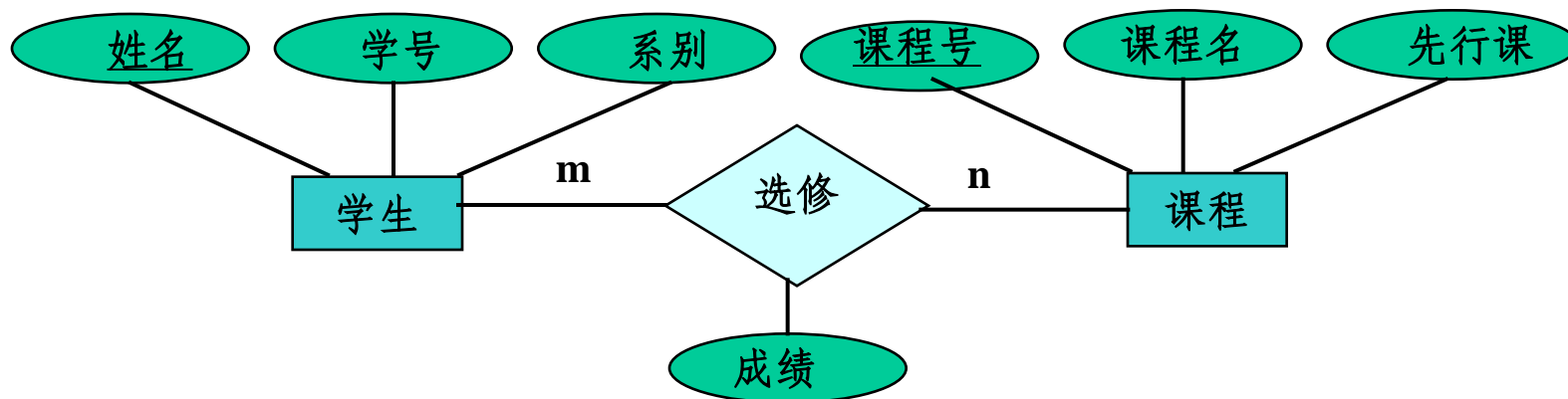
- 概念模型用于信息世界建模，是现实世界到信息世界的抽象，是用户和数据库设计人员进行交流的语言。

E-R图 (1)

- E-R图的组成：实体、联系、属性

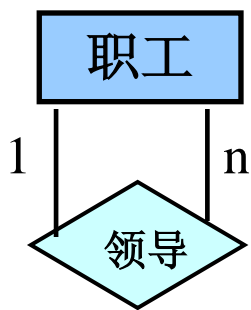
- 实体：用长方形表示实体型，在框内写上实体名。
- 属性：用椭圆形表示实体的属性，并用无向边把实体与其属性连接起来。
- 联系：用菱形表示实体间的联系，菱形框内写上联系名。用无向边把菱形分别与有关实体相连，在无向边旁标上联系的类型。若联系也具有属性，则属性和菱形也用无向边连接上。

例：

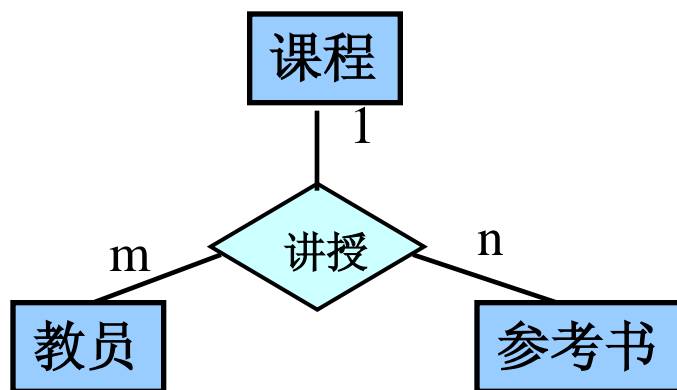


E-R图 (2)

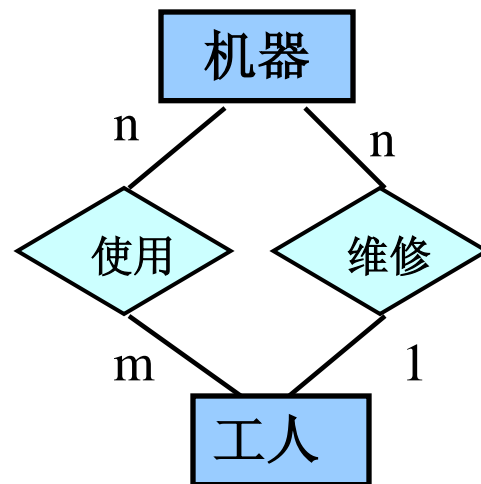
- 同一个实体集内部各实体之间也可以存在一对一、一对多、多对多的联系。(a)
- 三个或多个实体型间可能具有联系。(b)
- 两个实体型之间可具有多种联系。(c)



(a)



(b)

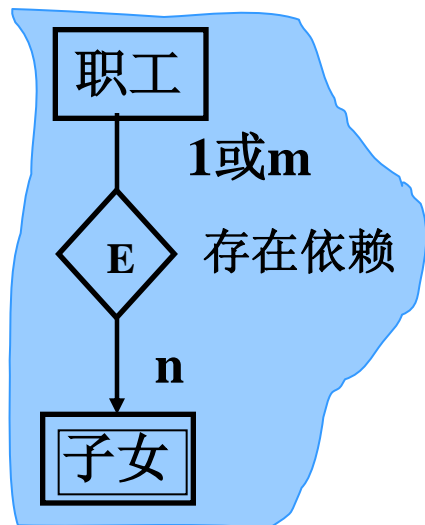


(c)

E-R图 (3)

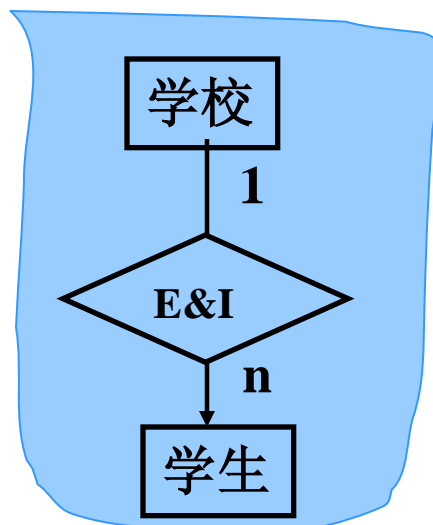
• E-R图实体之间联系的语义扩充

(1) 存在依赖



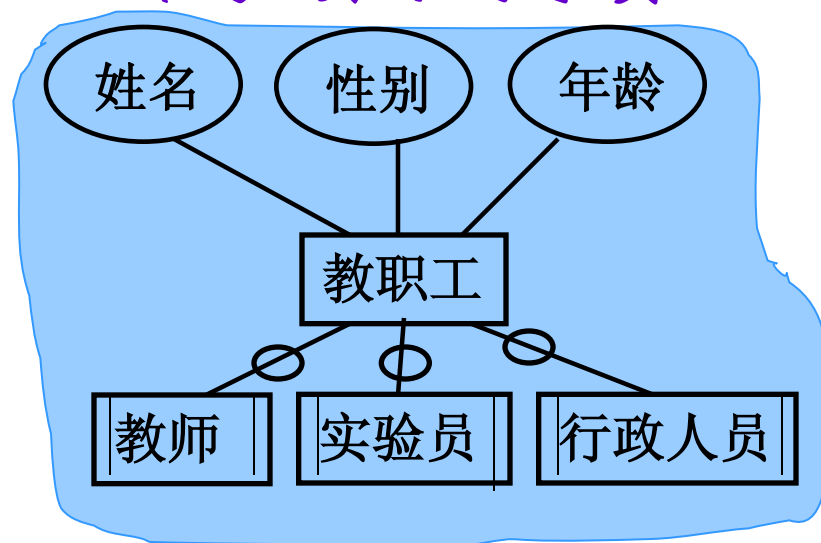
子女实体存在依赖于职工实体的存在，则称子女实体是弱实体。

(2) 标识依赖



如果实体不能由它自己的属性来唯一标识，而必须通过与它相联系的另一实体一起来标识，那么称该实体标识依赖于另一个实体。

(3) 实体的子类



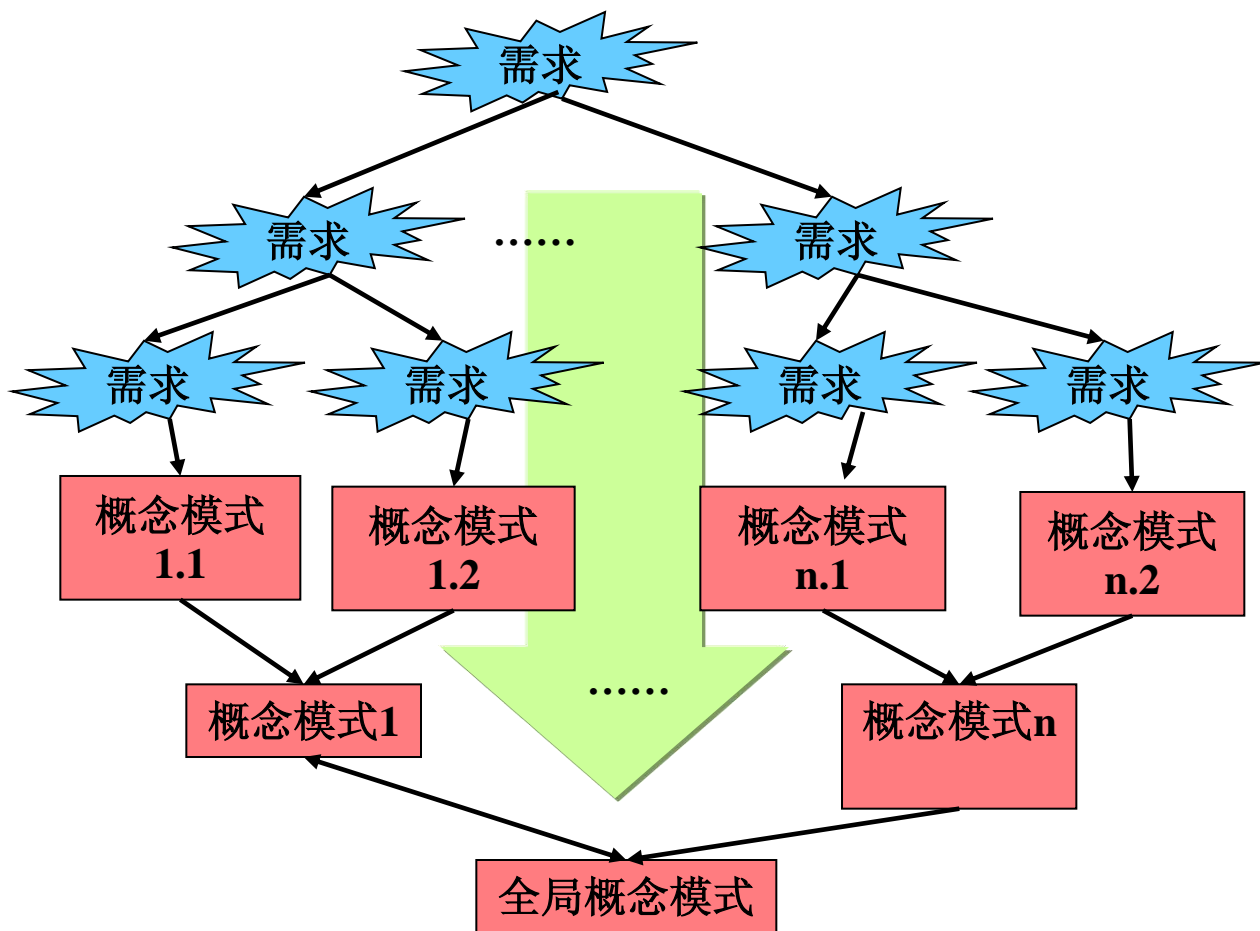
- 子类可继承父类的属性，子类也可以附加某些属性；
- 子类之间的交不一定为空。

概念结构设计方法

- 设计概念结构通常有四类方法：
 - **自顶向下**：首先定义全局概念结构的框架，然后逐步细化。
 - **自底向上**：首先定义各局部应用的概念结构，然后将其集成起来。
 - **逐步扩张**：首先定义核心概念结构，然后向外扩充。
 - **混合策略**：自顶向下设计全局概念框架，自底向上设计各局部概念结构。
- 最常用的方法是**自底向上**方法

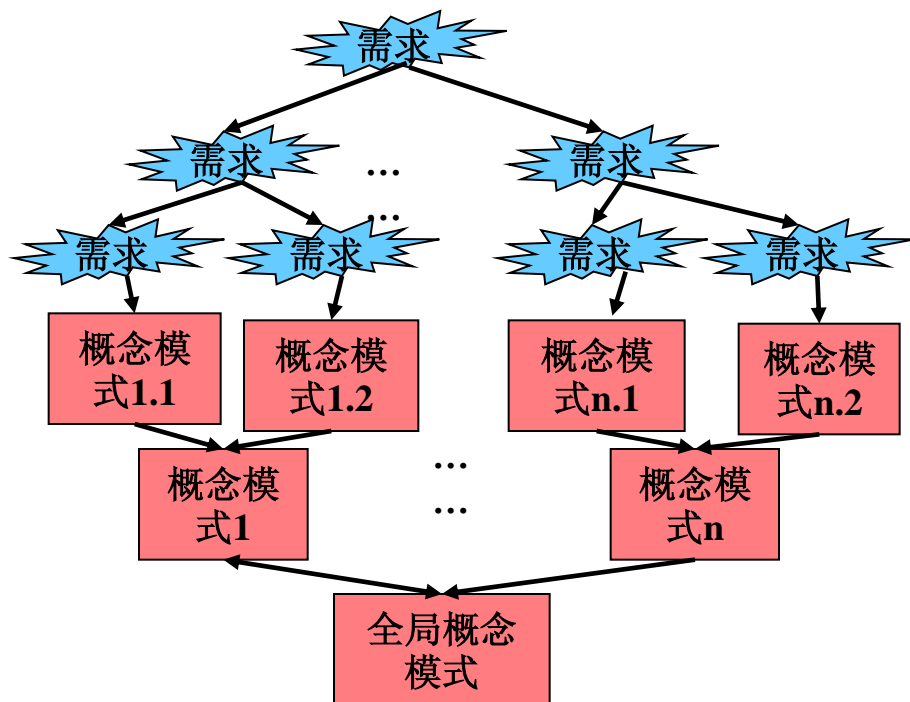
概念结构设计方法

- 自底向上方法
 - 自顶向下
进行需求分析;
 - 自底向上
设计概念结构。



基于E-R法的概念模型设计

- 基于E-R法的概念模式自底向上的设计方法分为两个阶段：
 - 数据抽象与局部E-R图设计
 - 综合局部E-R图形成总E-R图



数据抽象

- 局部E-R图设计的基本思想是利用数据抽象机制对需求分析阶段收集的数据进行分类、组织，形成实体模型并确定实体之间的联系类型，设计分E-R图（局部E-R图）。
- 数据抽象机制包括：
 - 分类
 - 定义某一概念作为现实世界中一组对象的类型。这些对象具有某些共同的特性和行为。如实体型。
 - 聚集
 - 定义某一类型的组成成分。如属性的聚集组成了实体型。
 - 概括
 - 定义类型之间的一种子集联系。

局部E-R图设计 (1)

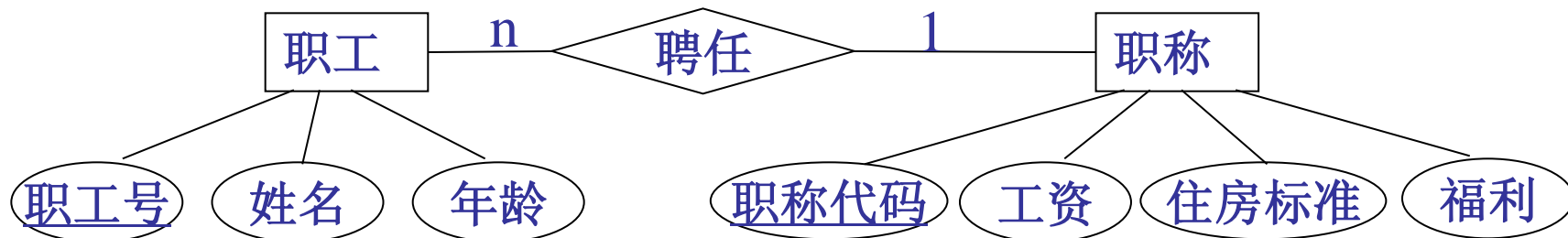
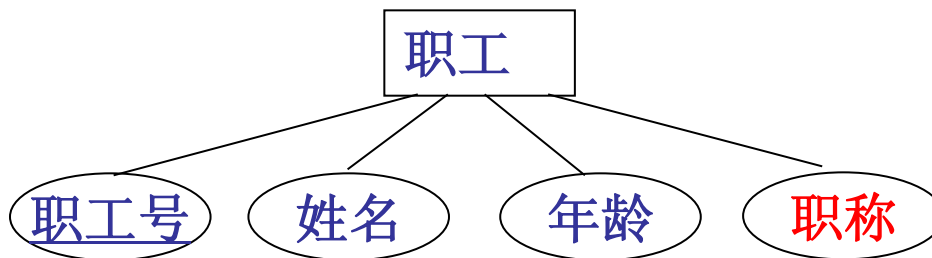
- 局部E-R图具体设计方法：
 - 选择局部应用；
 - 以需求分析中得到的数据元素表为基础，建立实体模型；
 - 确定实体之间的联系类型。用E-R图表示这些实体与实体之间的联系，形成分E-R图。

局部E-R图设计 (2)

- 建立实体模型的关键是确定实体及其属性
- 实体模型的建立方法：
 - 对数据字典进行初步抽象，得到实体和属性，然后再按原则进行必要调整

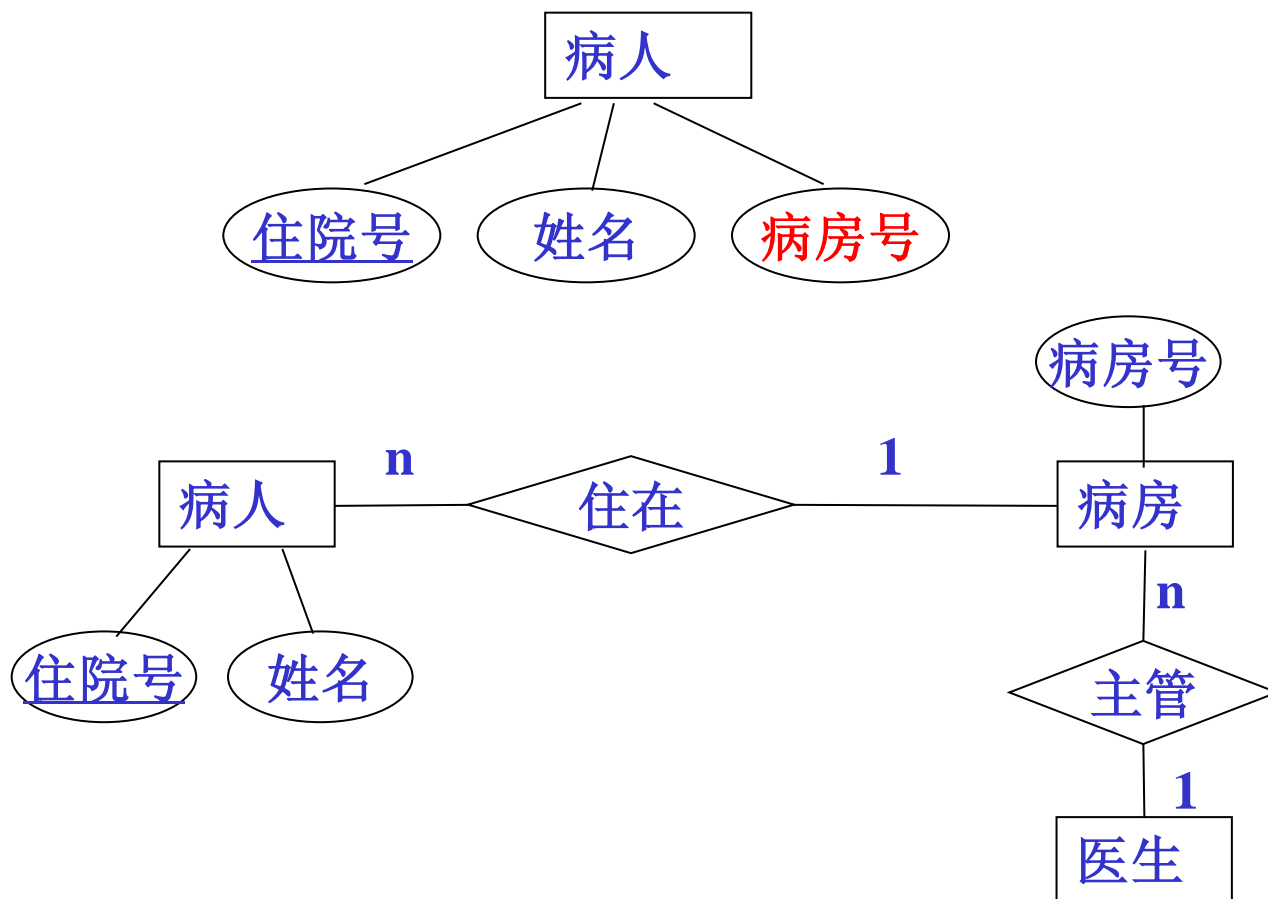
实体模型的调整原则 (1)

- 作为属性，不能再具有需要描述的性质。属性必须是不可分的数据项，不能是另一些属性的聚集。



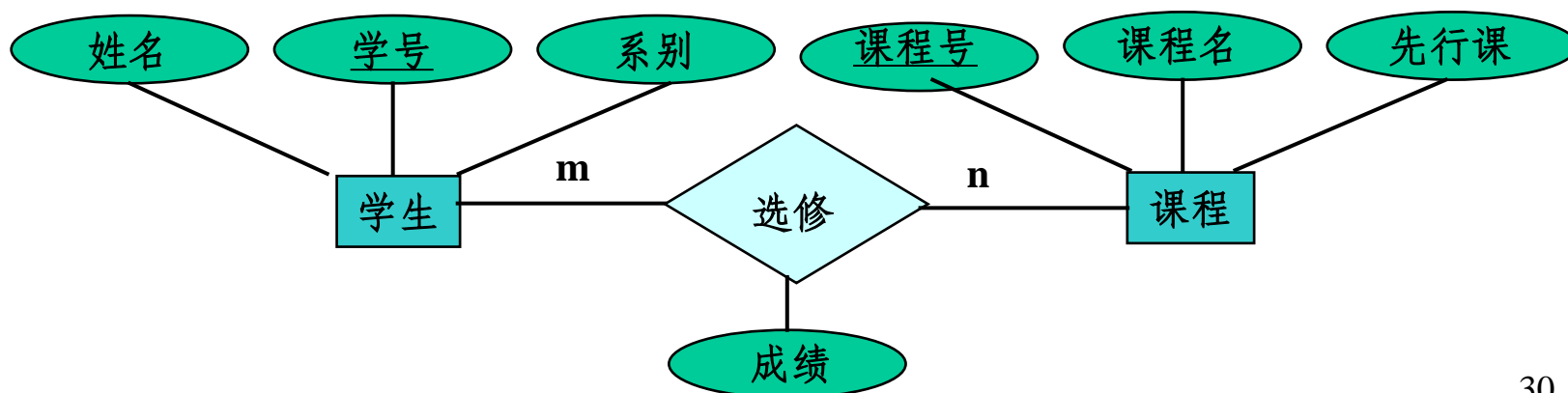
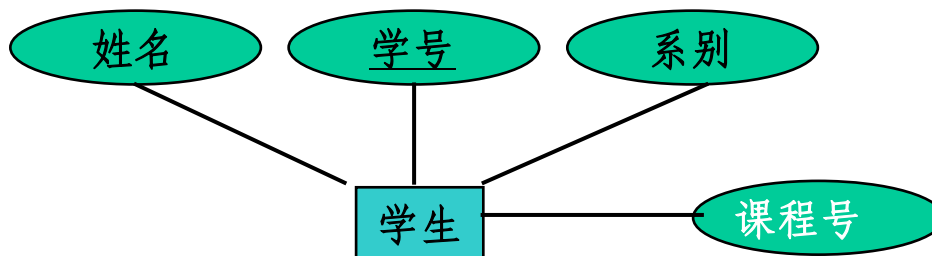
实体模型的调整原则 (2)

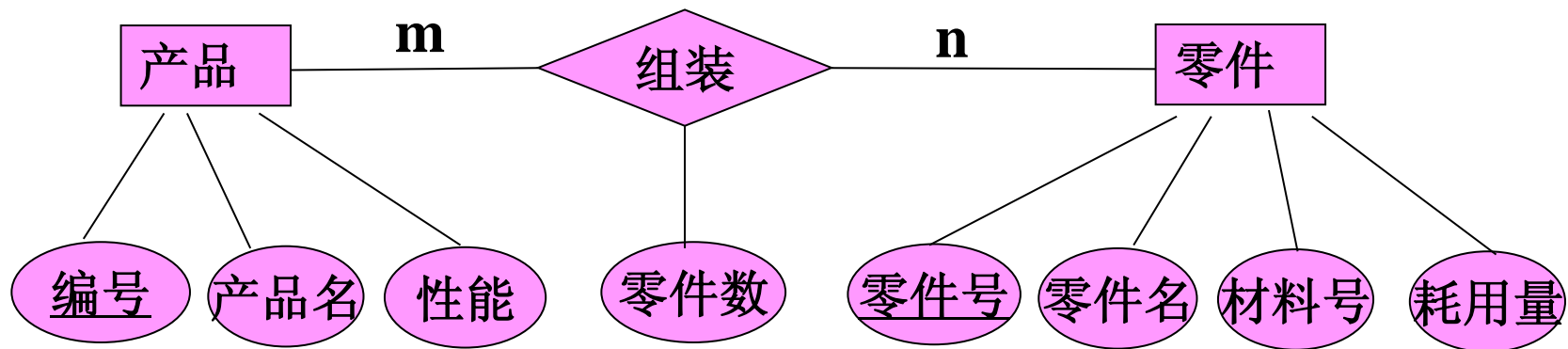
- 属性不能与其他实体具有联系，即ER图中所表示的联系是实体之间的联系



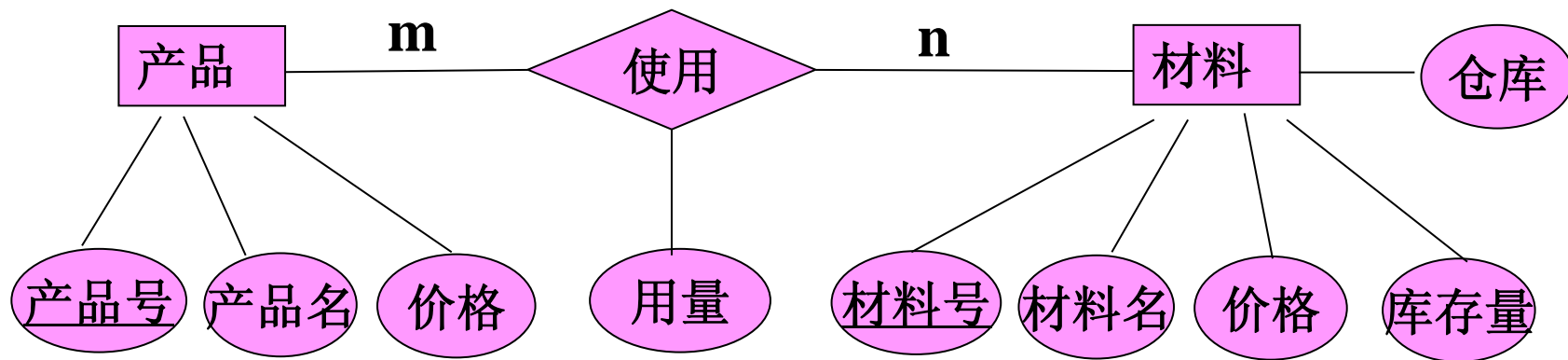
实体模型的调整原则 (3)

- 实体和描述它的属性之间保持1:1或n:1的联系。对于1:n或n:m联系, 要进行调整, 一般可将该属性上升为实体





生产部门的局部E-R图

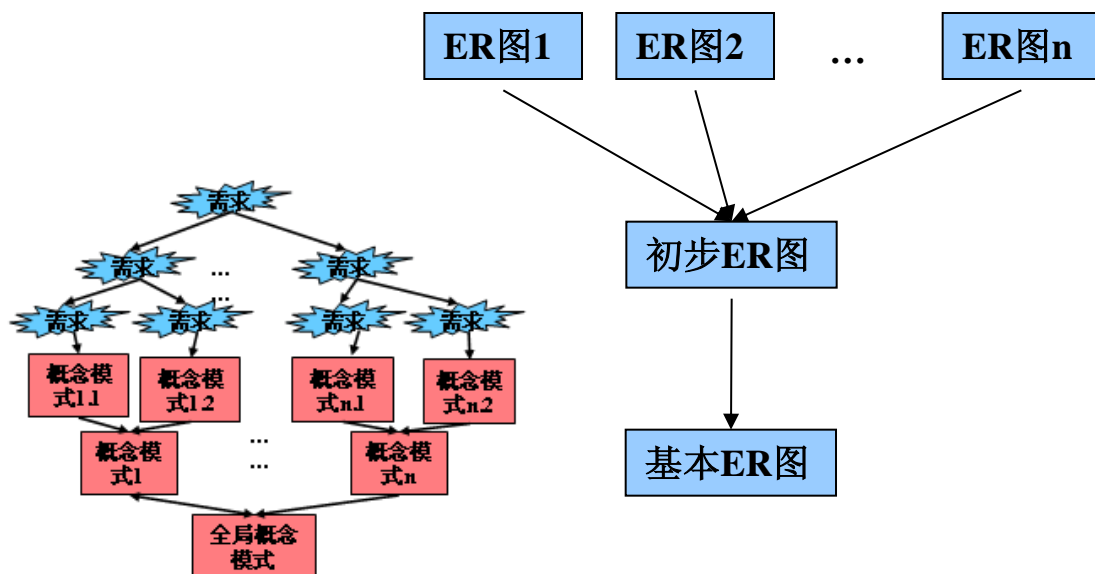


供应部门的局部E-R图

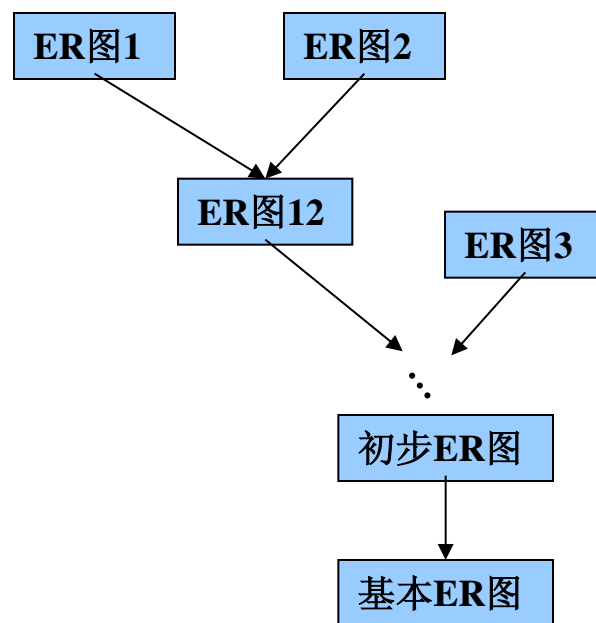


综合分E-R图形成总E-R图

- 设计总E-R图（全局概念模式）可以有两种方法：
 - 多个分E-R图一次集成（a）。
 - 逐步集成，用累加的方式一次集成两个分E-R图（b）。



(a)



(b)

集成局部E-R图的步骤

- 合并
 - 解决各分E-R图之间的冲突，将各分E-R图合并起来生成初步E-R图。
- 修改和重构
 - 消除不必要的冗余，生成基本E-R图。

局部E-R图的合并 (1)

- 消除冲突，合并分E-R图，生成初步E-R图。
- 冲突主要包括：属性冲突、命名冲突和结构冲突

(1) 属性冲突：属性的类型、取值范围或取值集合不同，或属性取值单位冲突。

解决——讨论协商解决。

(2) 命名冲突：包括属性名、实体名、联系名之间的同名异义，异名同义。

- 同名异义：不同意义的对象具有相同的名字。
- 异名同义：相同意义的对象具有不同名字。

解决——建立命名表，统一命名，异名同义的名字可标为别名。

局部E-R图的合并 (2)

(3) 结构冲突:

- 同一对象在不同应用中有不同抽象。如在一应用中为实体，在另一应用中为属性。

解决——遵守实体与属性的划分原则，把属性变为实体或实体变为属性，使同一对象具有相同的抽象。

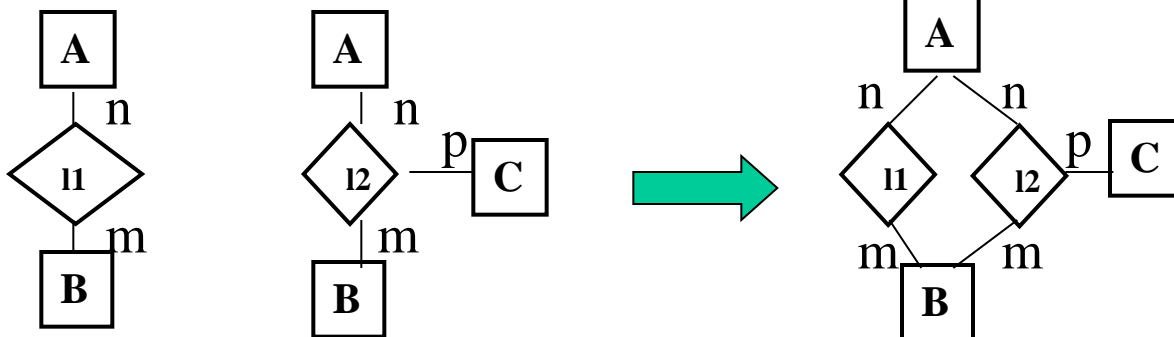
- 同一实体在不同分E-R图中属性个数、次序不同。

解决——同一实体的属性通常取分E-R图中属性的并，再适当调整次序。

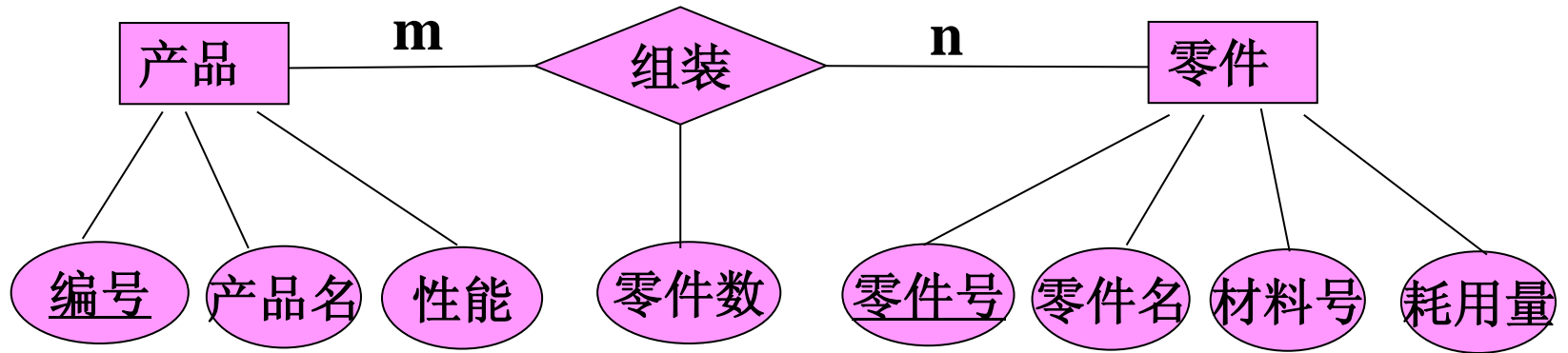
- 实体之间的联系在不同分E-R图中呈现不同类型。

解决——根据语义加以综合或调整。

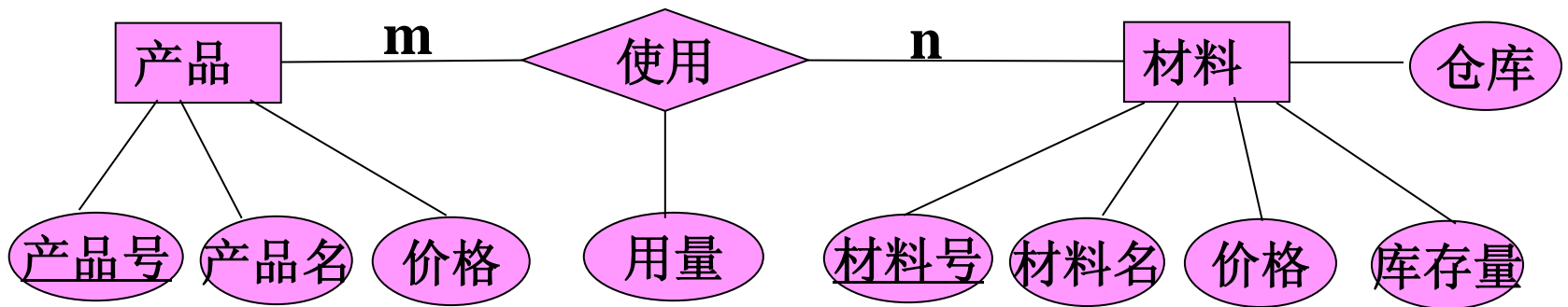
例：



示例

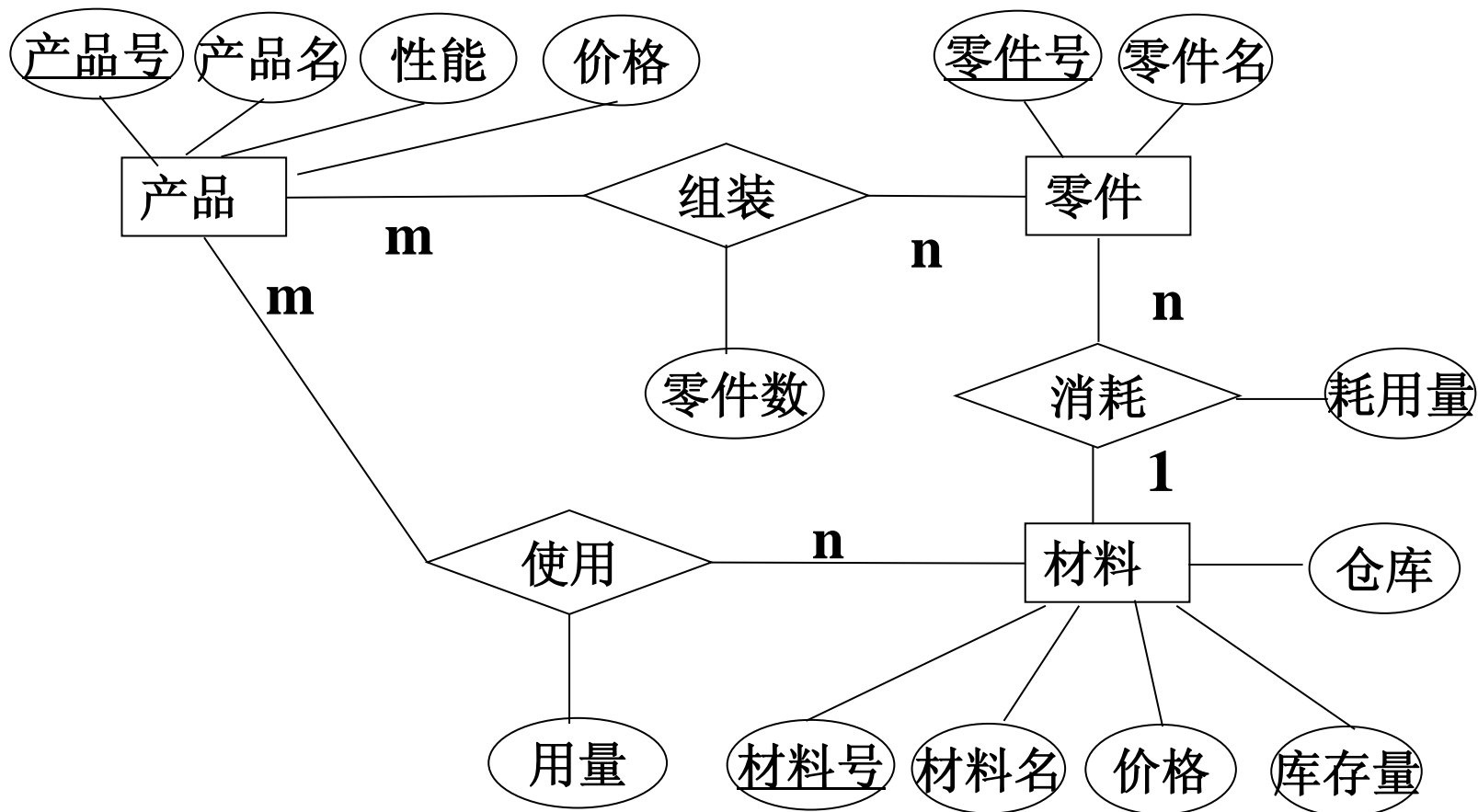


生产部门的局部E-R图



供应部门的局部E-R图

局部E-R图的合并 (3)



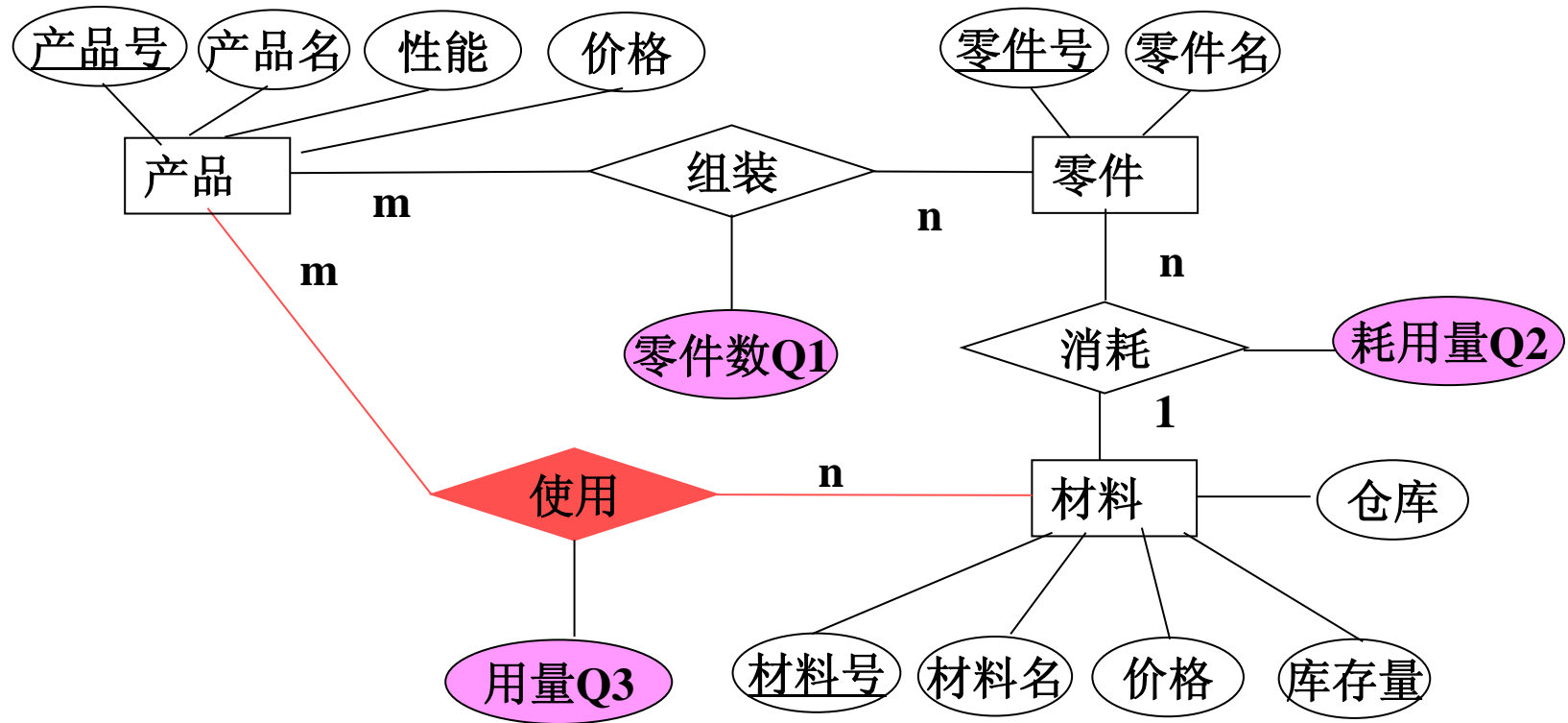
初步E-R图

修改初步E-R图设计基本E-R图(1)

- 消除不必要的冗余，设计基本E-R图
 - 初步E-R图中可能存在冗余的数据和冗余的联系。
 - 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它联系导出的联系。
 - 消除冗余有两种方法：
 - 分析法
 - 规范化方法

修改初步E-R图设计基本E-R图(2)

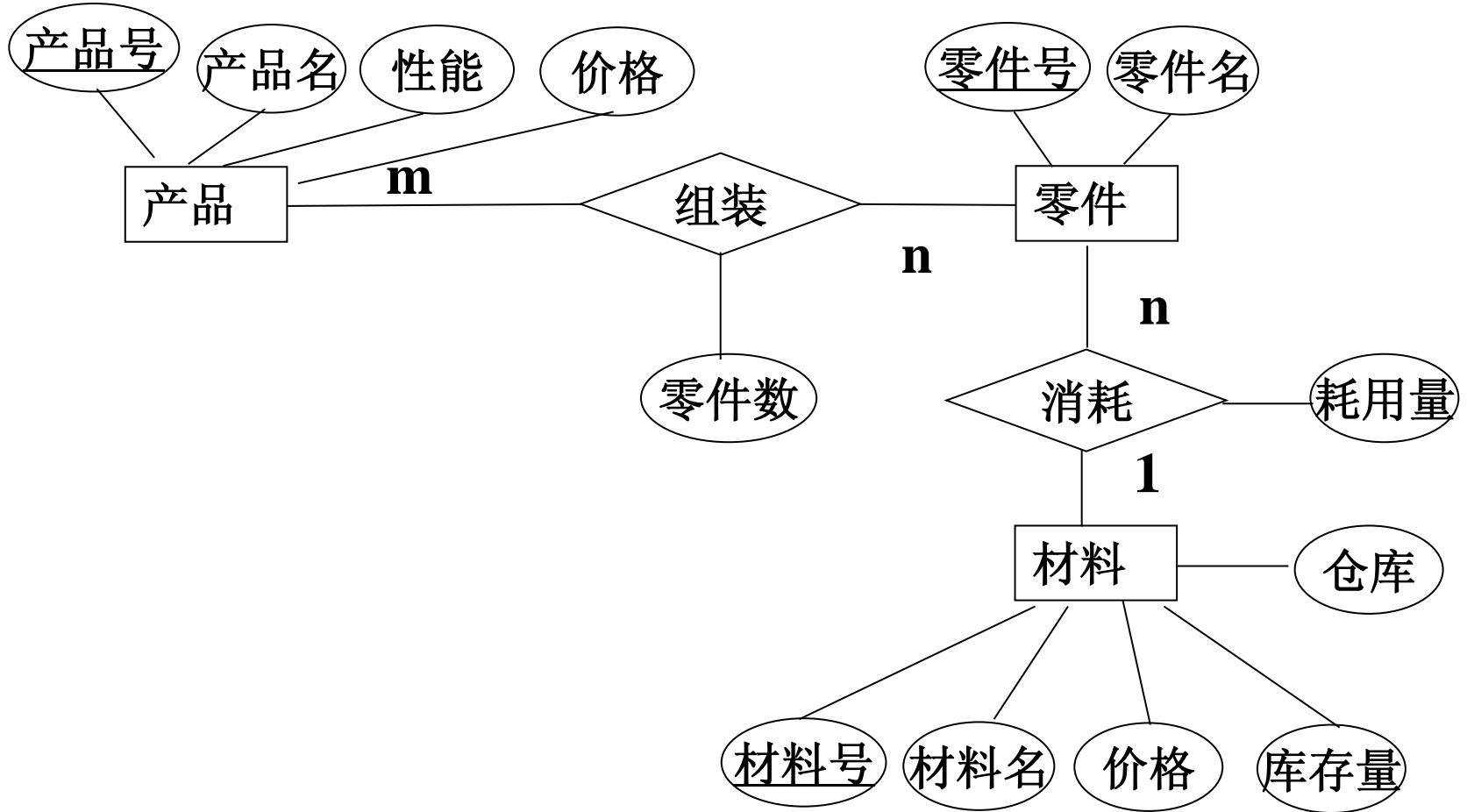
• 分析法消除冗余



• $Q3 = Q1 * Q2$, 所以 $Q3$ 是冗余的数据;

• 产品和材料之间的联系“使用”可以从产品与零件、零件与材料的联系导出, 所以是冗余的联系也可消去。

修改初步E-R图设计基本E-R图(3)



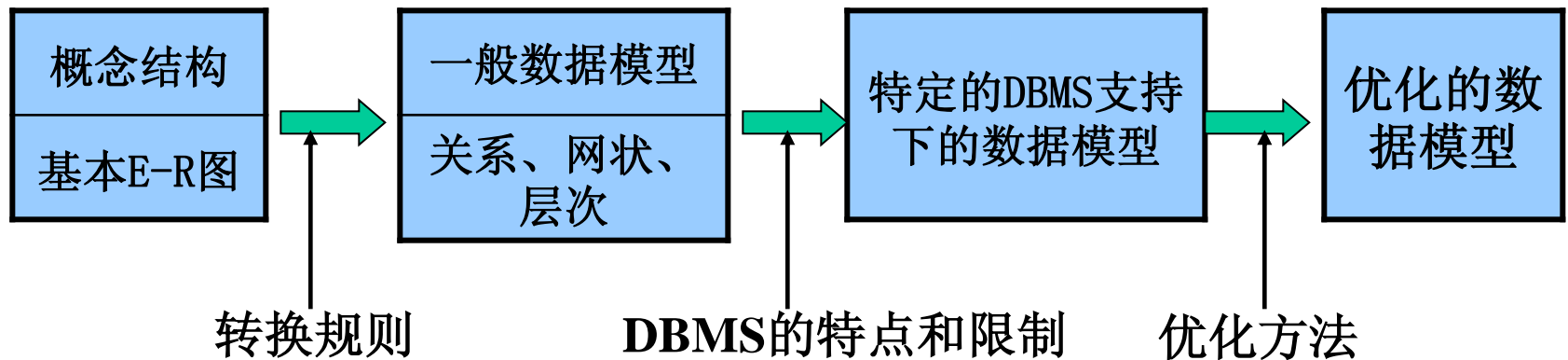
修改初步E-R图设计基本E-R图(4)

- 规范化方法消除冗余联系
 - 把E-R图中实体用符号表示;
 - 对每一对n:1、1:1或n:m联系表示为实体码之间的函数依赖表达式 $X \rightarrow Y$;
 - 利用函数依赖集的最小覆盖算法进行极小化处理。设原函数依赖表达式集合为F, 最小覆盖集为G, 则 $D=F-G$;
 - 考察D中每一个函数依赖表达式, 确定是否冗余联系;
 - 去掉冗余联系后形成基本E-R图。



数据库逻辑结构设计

- 逻辑结构设计的任务就是把概念结构转换为选用的DBMS所支持的数据模型的过程。
- 逻辑结构设计步骤如下：



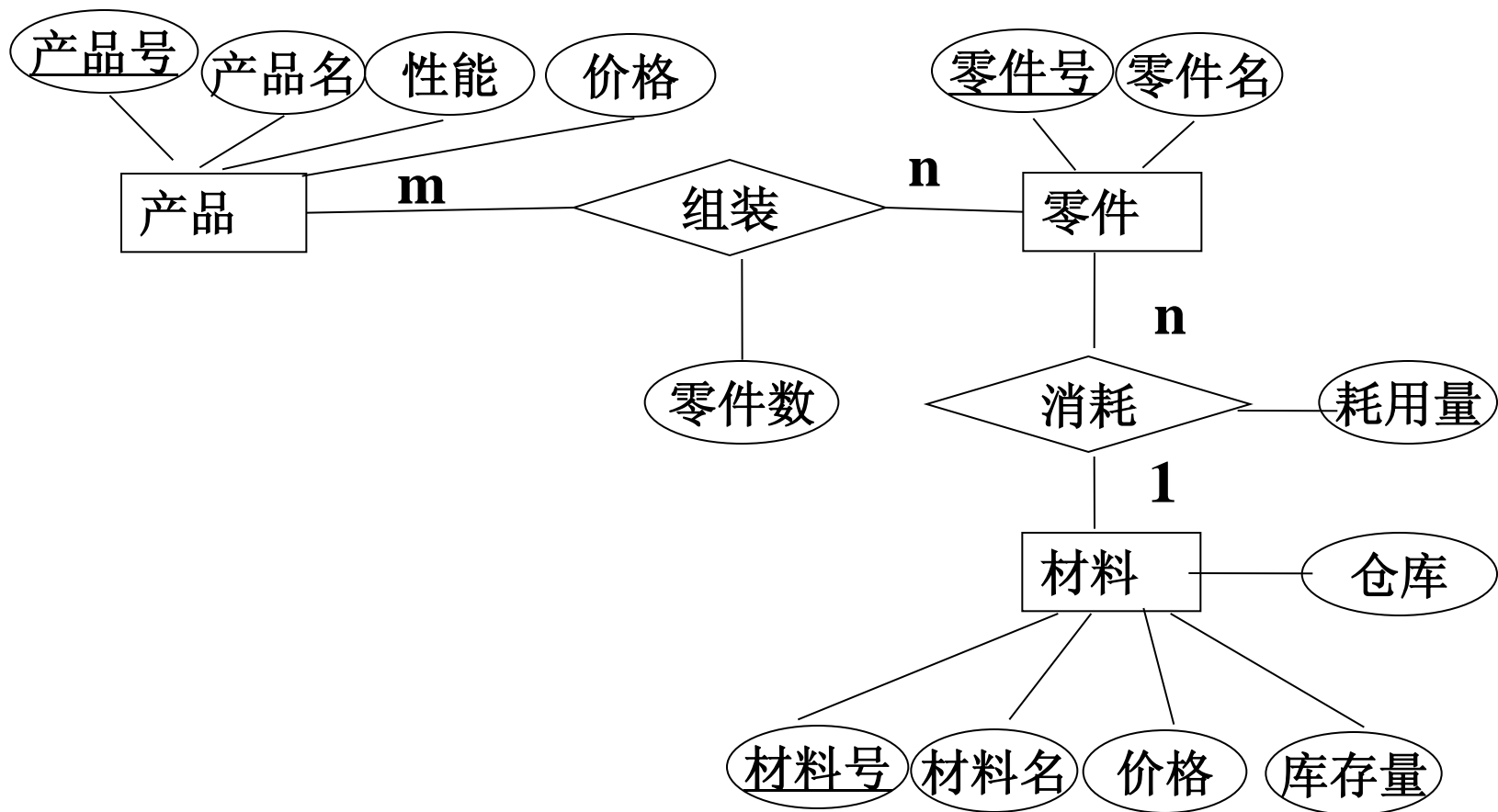
数据库逻辑结构设计

- 关系数据库逻辑结构设计的任务具体包括：
 - 形成初始关系数据库模式
 - 关系模式规范化
 - 关系模式优化
 - 子模式定义

E-R图向关系模型的转换规则

- 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。
- 一个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的属性转换为关系的属性：
 - 若联系为1: 1，则每个关系的码均是该关系的候选码；
 - 若联系为1: n，则该关系的码是n端实体的码；
 - 若联系为n: m，则该关系的码是诸实体码的组合。
- 三个或三个以上实体间的多元联系，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的属性转换为关系的属性，而关系的码为各实体码的组合。
- 具有相同码的关系可以合并。

E-R图向关系模型示例

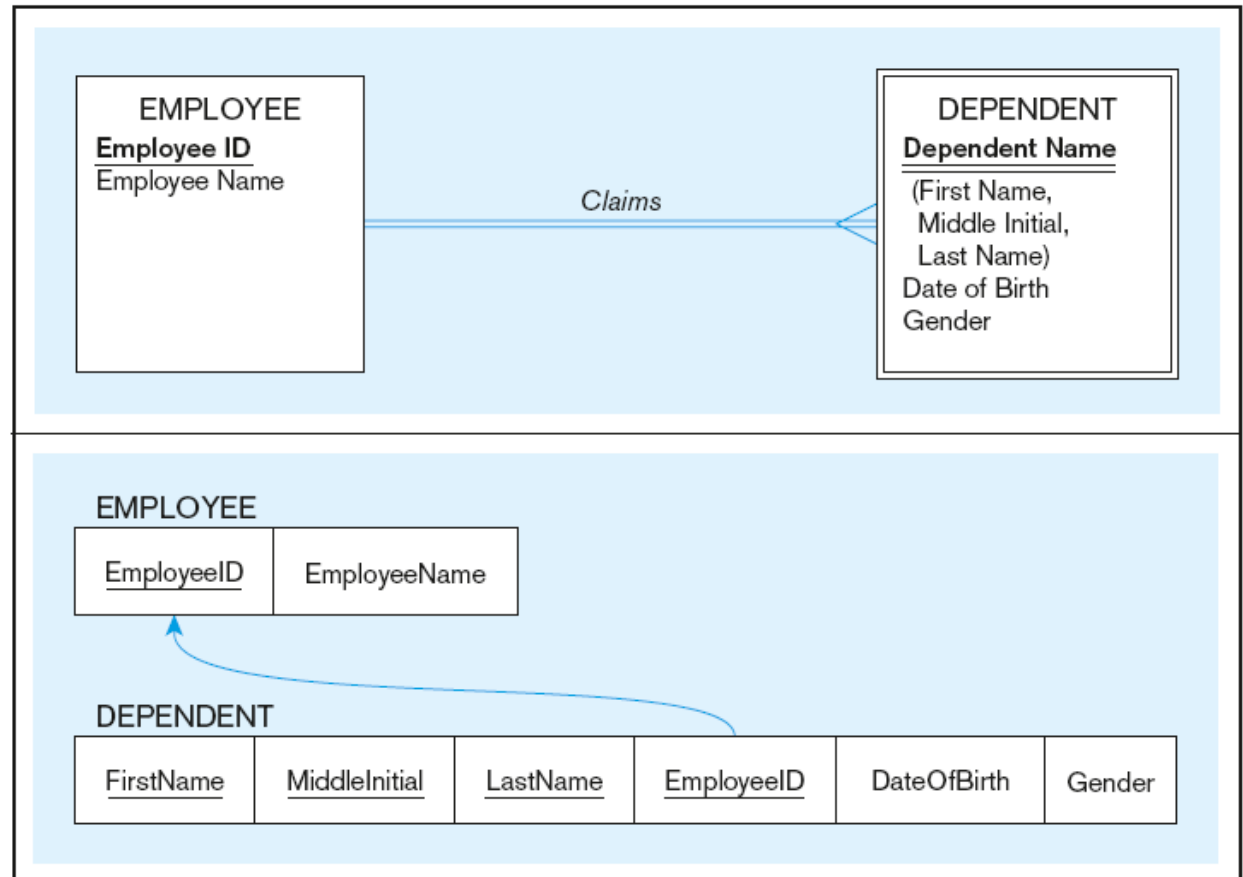


E-R图向关系模型的转换规则

- 弱实体类型的转换
 - 对于每个弱实体类型，创建一个新的关系，该关系中包含所有弱实体类型的属性。
 - 把标识关系的主码添加到新关系中，并将其作为新关系的外码。
 - 新关系的主码是标识关系的主码和弱实体类型的部分标识符的组合。

弱实体类型的转换示例

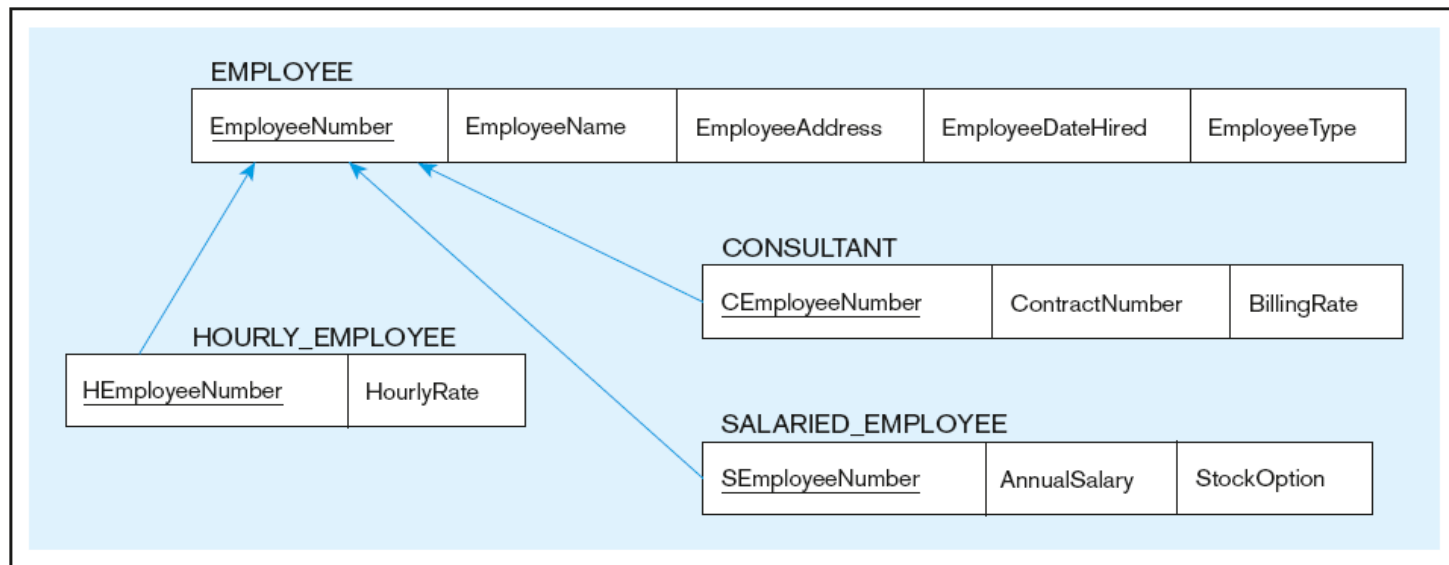
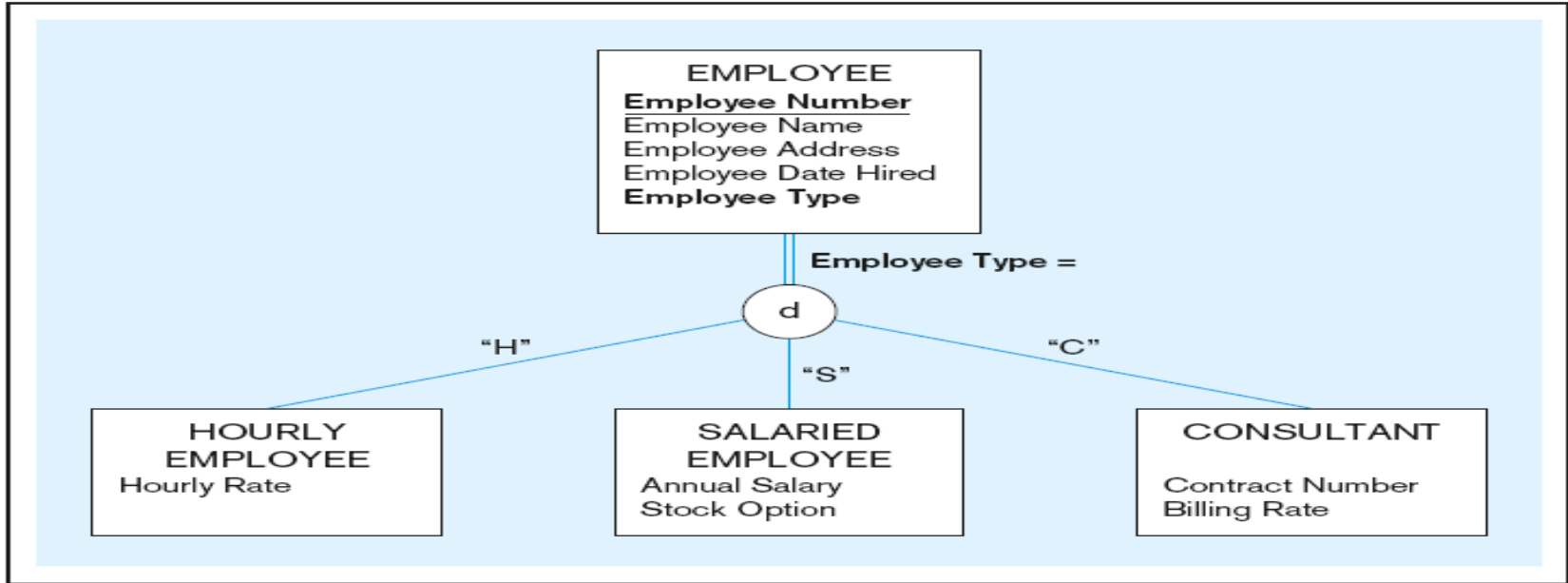
FIGURE 4-11 Example of mapping a weak entity
(a) Weak entity DEPENDENT



E-R图向关系模型的转换规则

- 超类/子类联系的转换
 - 为超类和每个子类创建单独的关系。
 - 在为超类所创建的关系中，包含所有子类成员都共有的属性，包括主码。
 - 在为每个子类所创建的关系中，包含超类的主码以及子类特有的属性。
 - 在超类中包含一个（或多个）属性作为子类判定符。

超类/子类联系转换的示例



关系模型的规范化与优化 (1)

- 按照数据依赖的理论，逐一分析转换所得关系模式，判断是否存在部分函数依赖、传递函数依赖、多值依赖等，确定它们的范式等级；
- 按应用系统的处理要求，确定是否进行模式合并或分解；
- 为了提高存取效率和存储空间的利用率，可以对关系模式进行必要的分解。
 - 水平分解
 - 垂直分解

关系模型的规范化与优化 (2)

- 水平分解

- 是把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，以提高系统效率。

- 80/20原则

- 可以把经常使用的那一部分数据分解出来作为一个关系，其他数据作为另一个关系。

- 数据分片

- 如果关系R上具有n个事务，而且多数事务存取的数据不相交，则R可以分解为少于或等于n个子关系。

关系模型的规范化与优化 (3)

- 垂直分解

- 是把关系模式R的属性分解为若干子集合，形成若干子关系模式。

- 垂直分解的原则是，经常在一起使用的属性从R中分解出来形成一个子关系模式。
 - 垂直分解必须确保无损连接性和保持函数依赖。

设计用户子模式

- 根据局部应用的需求，结合具体DBMS的特点，设计用户外模式（视图）
 - 使用更符合用户习惯的别名；
 - 可以对不同级别的用户定义不同的视图，以保证系统的安全性；
 - 通过定义视图降低复杂查询的难度，简化用户对系统的使用。



数据库的物理设计

- 确定数据库的存储结构
- 选择关系的存取方法
 - 存取方法是使事务能够快速存取数据库中数据的技术。

关系的存取方法

- 常用的存取方法：
 - 索引方法
 - 聚集方法
 - HASH方法

索引方法

- 基本概念

- 索引记录/索引项，索引文件的记录，包括两个域：

- 索引域：存储数据文件中一个或一组域的一个值 (K)
 - 指针：指向索引域值为K的记录所在磁盘块的地址。

- 分类

- 按照索引文件结构

- 稀疏索引：把所有数据记录按索引域值分成许多组，每组设立一个索引项；
 - 稠密索引：为每个记录设立一个索引项，记录存放是任意的，但索引是有序的。

多级索引

- 二叉树索引

- 每个节点只有一个关键字，有两个指针，分别指向关键字值小于或大于当前节点值的节点；

- 多枝树索引

- 每个节点有 D ($D \geq 2$)个关键字值，则有 $D+1$ 个指针，每两个指针之间对应相邻关键字之间取值的区间；

多级索引

- B树（平衡树）索引

- 是附加限制条件的索引树。限制了每个节点放置关键字与指针的最小和最大个数，并且所有的叶节点都在同一层上。B树的关键字是散布在各层上。

- B⁺树索引

- 是B树的改进。把树中所有关键字都按递增次序从左到右安排在叶节点上，并且链接起来。B⁺树能同时进行随机查找和顺序查找。

索引存取方法的选择

- 选择索引域原则：
 - 经常在查询条件中出现的属性
 - 经常作为最大值和最小值库函数的参数
 - 经常作为连接属性
- 索引并非越多越好

聚集方法

- 基本概念

- 把某个属性/组(聚集键)值相同的关系的记录集中存放在连续的物理块, 称为聚集。能够提高该属性的查询速度;
- 把经常进行连接操作的关系的记录以连接属性为中心进行存储, 连接属性为聚集键。提高连接操作的效率。

- 一个关系只能参加一个聚集。

聚集存取方法的选择

- 一般原则

- 经常进行连接操作的关系可建立聚集
- 单个关系的某组属性经常进行相等比较
- 关系的某个属性组值重复率高

- 注意问题

- 建立与维护聚集系统开销很大，对于更新操作远远多于连接操作的关系不应使用聚集方法。

HASH文件

- 一种支持快速存取的文件存储方法
- 基本概念

通过HASH函数将记录关键字转换成地址。

设F是一个文件，A是F的HASH域，H是定义在A的值域上的HASH函数。对于记录r（其A的值为a）的存储地址由 $H(a)$ 确定。

HASH存取方法的选择

- 如果关系的属性主要出现在等连接条件中，或出现在相等比较条件中，而且满足下列条件之一，可以选择该方法：
 - 关系的大小可预知，而且不变；
 - 如果关系大小动态改变，则须DBMS提供动态HASH存取方法。

确定数据库的存储结构

- 确定存放位置
 - 经常存取部分分开存放
 - 数据和日志备份放于磁带上
- 确定系统配置
 - 确定系统配置变量、存储分配参数，进行物理优化

第五章 关系数据理论

问题的提出 (1)

例：为学校设计一个关系数据库，管理的信息包括学生学号、选修课程名称、成绩、所在系以及系主任名。

现实世界的语义：

- 一个系有若干个学生，但一个学生只属于一个系。
- 一个系只有一名系主任。
- 一个学生可以选修多门课程，每门课程可有若干学生选修。
- 每个学生学习每门课程有一个成绩

设计名称为UN的关系模式：

UN (S#, CN, G, SDN, MN) ,

其中：S#—学号， CN—课程名， G—成绩，
SDN—系名， MN—系负责人

问题的提出 (2)

- 对UN进行操作时的问题:

UN (S#, CN, G, SDN, MN)

插入异常!

删除异常!

数据冗余大!

- UN是“不好”的关系模式!

问题的提出 (3)

将学生管理数据库进行如下设计：

$$\left\{ \begin{array}{l} \text{SD} (\underline{\text{S\#}}, \text{SDN}) \\ \text{SG} (\underline{\text{S\#}}, \underline{\text{CN}}, \text{G}) \\ \text{DM} (\underline{\text{SDN}}, \text{MN}) \end{array} \right.$$

其中：S# — 学号，SDN — 系名，CN — 课程名，
G — 成绩，MN — 系负责人

SD、SG、DM 不会发生插入异常和删除异常，冗余最少。
SD、SG、DM 是“好的”数据库模式！

问题的分析

UN (S#, CN, G, SDN, MN)

不好的关系模式

属性之间关系:

$S\# \rightarrow SDN$;

$SDN \rightarrow MN$;

$S\# \rightarrow MN$;

$(S\#, CN) \rightarrow G$;

$(S\#, CN) \rightarrow SDN$;

$(S\#, CN) \rightarrow MN$

?

因素

因素: 关系内部属性之间的约束关系

好的关系模式

SD (S#, SDN)

SG (S#, CN, G)

DM (SDN, MN)

属性之间关系:

$S\# \rightarrow SDN$;

$(S\#, CN) \rightarrow G$;

$SDN \rightarrow MN$;

数据依赖的概念

- 数据依赖：属性值之间相互依赖又相互制约的关系。
- 数据依赖有许多种类型，其中最重要的有两种：
 - 函数依赖(Functional Dependency);
 - 多值依赖(Multivalued Dependency)。

关系数据理论与数据依赖

不好的关系模式

如：UN (S#, CN, G, SDN, MN)

关系的
规范化

规范化方法
与算法

范式
理论

数据依赖的
概念与公理

- 函数依赖
- 多值依赖

关系数据理论

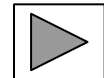
好的关系模式

如：

SD (S#, SDN)

SG (S#, CN, G)

DM (SDN, MN)



关系数据理论

- 函数依赖
- 规范化
- 多值依赖与第四范式
- 模式分解的理论
- 候选码的求解理论和算法
- 在数据库设计中的应用

函数依赖

- 函数依赖的定义
- 三种函数依赖
- 关系键的形式定义
- 函数依赖的逻辑蕴涵
- Armstrong公理系统



函数依赖

- 函数依赖的定义

设 $R(U)$ 是属性集 U 上的关系模式。 X 、 Y 是 U 的子集。 r 是 R 的任意一个具体关系， t, s 是 r 中任意两个元组。如果 $t[X] = s[X]$ ，则 $t[Y] = s[Y]$ ，则称“ X 函数确定 Y ”或“ Y 函数依赖于 X ”，记作： $X \rightarrow Y$ 。

- 函数依赖 $X \rightarrow Y$ 也可定义为：

对于 X 的每个具体值， Y 有唯一的值与之对应，则称“ X 函数确定 Y ”或“ Y 函数依赖于 X ”。

函数依赖示例

例：

S#	CN	G	SDN	MN
S601	数据库	90	CS	张明
S602	数据库	87	CS	张明
S601	编译	85	CS	张明
S602	编译	86	CS	张明
S801	C++	78	IS	李立
S802	C++	80	IS	李立

$S\# \rightarrow SDN;$

$SDN \rightarrow MN;$

$(S\#, CN) \rightarrow G;$

函数依赖相关术语

- 平凡与非平凡的函数依赖

- 对于函数依赖 $X \rightarrow Y$ ，若 $Y \subseteq X$ ，则称 $X \rightarrow Y$ 是平凡的函数依赖；若 $Y \not\subseteq X$ ，则称 $X \rightarrow Y$ 是非平凡的函数依赖。

- 决定因素

- 对于函数依赖 $X \rightarrow Y$ ，则 X 叫做决定因素。

函数依赖的进一步说明

- 函数依赖是不随时间而变的
 - 若关系R具有函数依赖 $X \rightarrow Y$ ，那么虽然关系R随时间而变化，但 $X \rightarrow Y$ 不变
- 函数依赖是语义范畴的概念
 - 只能根据语义来确定一个函数依赖，而不能形式化证明一个函数依赖成立

函数依赖与属性间的联系类型

- 1:1 (一对一) 关系

如：学生的学号与身份证号

– 若X与Y是1:1, 则

$$X \rightarrow Y, Y \rightarrow X;$$

- 1:m (一对多) 关系

如：学生所在系的系名与学号

– X与Y是1:m, 则

$$\text{只存在 } Y \rightarrow X$$

函数依赖与属性间的联系类型

- **$n:m$ (多对多) 关系**

如：学号与课程名

– 若 X 与 Y 是 $n:m$ ，则

X 与 Y 之间不存在函数依赖。



三种函数依赖

- 完全函数依赖与部分函数依赖

- 定义：在 $R(U)$ 中，如果 $X \rightarrow Y$ ，且对于任意 X 的真子集 X' ，都有 $X' \not\rightarrow Y$ ，则称 Y 对 X 完全函数依赖，记作 $X \xrightarrow{f} Y$ ，否则称为 Y 对 X 部分函数依赖，记作 $X \xrightarrow{p} Y$ 。

- 传递函数依赖

- 定义：在 $R(U)$ 中，如果 $X \rightarrow Y$ ， $Y \rightarrow Z$ ，且 $Y \not\rightarrow X$ ，则称 Z 对 X 传递函数依赖。

三种函数依赖示例

例：UN (S#, CN, G, SDN, MN)

属性之间函数依赖关系：

$S\# \xrightarrow{f} SDN;$

$SDN \xrightarrow{f} MN;$

$S\# \xrightarrow{t} MN;$

$(S\#, CN) \xrightarrow{f} G;$

$(S\#, CN) \xrightarrow{p} SDN;$

现实世界的语义：

- 一个系有若干个学生，但一个学生只属于一个系。
- 一个系只有一名系主任。
- 一个学生可以选修多门课程，每门课程可有若干学生选修。
- 每个学生学习每门课程有一个成绩



关系键的形式定义 (1)

- 候选码 (键) 与主码 (键)

- 定义: 设 K 为 $R \langle U, F \rangle$ 中的属性或属性组合, 若 $K \xrightarrow{f} U$, 则称 K 为 R 的**候选码**。若候选码多于一个, 则选定其中的一个作为**主码**。

- 主码的性质:

- **唯一性**: 唯一地标识关系中的元组。

- **最小性**: 若抽去主码中的任意一属性, 则主码将失去标识的唯一性。

- 主属性与非主属性:

- 包含在任何一个候选码中的属性, 叫**主属性**。

- 不包含在任何码中的属性称为**非主属性**。

关系键的形式定义 (2)

- 外部码

- 定义：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的外码。

- 例：

SD (S#, SDN)

SG (S#, CN, G)

DM (SDN, MN)



函数依赖的逻辑蕴涵

- 定义：关系模式 $R\langle U, F \rangle$ 中， X 、 Y 是 R 的属性集合，如果从 F 中的函数依赖能够推出 $X \rightarrow Y$ ，则称 F 逻辑蕴涵 $X \rightarrow Y$ 。
- 函数依赖集 F 的闭包
 - 定义：在关系模式 $R\langle U, F \rangle$ 中，为 F 所逻辑蕴涵的函数依赖的全体称作 F 的闭包，记作 F^+ 。



Armstrong公理系统

- Armstrong公理及推论
- 属性集的闭包
- Armstrong公理系统的有效性与完备性
- 闭包的计算
- 函数依赖集等价与覆盖
- 函数依赖集的最小依赖集



Armstrong公理系统

- Armstrong公理系统

对于 $R\langle U, F \rangle$ ，有如下规则：

- **A1自反律**：若 $Y \subseteq X \subseteq U$ ，则 $X \rightarrow Y$ 为 F 所蕴含。
- **A2增广律**：若 $X \rightarrow Y$ 为 F 所蕴含，且 $Z \subseteq U$ 则 $XZ \rightarrow YZ$ 为 F 所蕴含。
- **A3传递律**：若 $X \rightarrow Y$ ， $Y \rightarrow Z$ 为 F 所蕴含，则 $X \rightarrow Z$ 为 F 所蕴含。

Armstrong公理系统的正确性

对 $R\langle U, F \rangle$ 的任一关系 r 中任意两个元组 t, s :

• 证明自反律: 即若 $Y \subseteq X$, 则 $X \rightarrow Y$ 。

$$\left. \begin{array}{l} \text{若 } \underline{t[X] = s[X]} \\ \text{且 } Y \subseteq X \end{array} \right\} \longrightarrow \underline{t[Y] = s[Y]} \longrightarrow X \rightarrow Y$$

• 证明增广律: 即若 $X \rightarrow Y$, 则 $XZ \rightarrow YZ$ 。

$$\begin{aligned} \text{若 } \underline{t[XZ] = s[XZ]} &\longrightarrow \left\{ \begin{array}{l} \underline{t[Z] = s[Z]} \\ t[X] = s[X] \\ \text{又因 } X \rightarrow Y \end{array} \right\} \longrightarrow \underline{t[Y] = s[Y]} \\ &\longrightarrow \underline{t[YZ] = s[YZ]} \longrightarrow XZ \rightarrow YZ \end{aligned}$$

Armstrong公理系统的正确性

对 $R\langle U, F \rangle$ 的任一关系 r 中任意两个元组 t, s :

- **证明传递律**: 即若 $X \rightarrow Y$, $Y \rightarrow Z$, 则 $X \rightarrow Z$ 。

$$\left. \begin{array}{l} \text{若 } \underline{t[X] = s[X]} \\ \text{由于 } X \rightarrow Y \end{array} \right\} \longrightarrow \left. \begin{array}{l} t[Y] = s[Y] \\ \text{又因 } Y \rightarrow Z \end{array} \right\} \longrightarrow \underline{t[Z] = s[Z]}$$
$$\longrightarrow X \rightarrow Z$$

Armstrong公理的推论

- 由Armstrong公理系统得到的三条推理规则
 - 合并规则：由 $X \rightarrow Y$, $X \rightarrow Z$, 有 $X \rightarrow YZ$ 。
 - 伪传递规则：由 $X \rightarrow Y$, $WY \rightarrow Z$, 有 $XW \rightarrow Z$ 。
 - 分解规则：由 $X \rightarrow Y$ 及 $Z \subseteq Y$, 有 $X \rightarrow Z$ 。

- 从合并规则和分解规则得出如下定理：

定理1:

$X \rightarrow A_1 A_2 \dots A_k$ 成立 $\Leftrightarrow X \rightarrow A_i$ 成立 ($i=1, 2, \dots, k$)



属性集的闭包

- 属性集 X 关于函数依赖集 F 的闭包

- 定义：在 $R\langle U, F \rangle$ 中， $X \subseteq U$,

- $X_F^+ = \{A \mid X \rightarrow A \text{ 能由 } F \text{ 根据 Armstrong 公理导出}\}$

- 称 X_F^+ 为属性集 X 关于函数依赖集 F 的闭包。

- 定理：

- $X \rightarrow Y$ 能够由 F 根据Armstrong公理导出

- $\Leftrightarrow Y \subseteq X_F^+$



Armstrong公理系统的有效性与完备性

- Armstrong公理系统是有效的，完备的。
 - **有效性**：指由F出发根据Armstrong公理推导出来的每个函数依赖一定在F所蕴含的函数依赖的全体之中。
 - **完备性**：F所蕴含的函数依赖的全体中的每一个函数依赖，必定可以由F根据Armstrong公理导出。
- Armstrong公理系统的有效性由Armstrong公理系统的正确性得到证明，需要进一步证明Armstrong公理系统的完备性。

Armstrong公理完备性证明 (1)

- Armstrong公理完备性的证明

- 公理的完备性：F所蕴含的函数依赖的全体中的每一个函数依赖，必定可以由F根据Armstrong公理导出
- 证明逆否命题，证明若 $X \rightarrow Y$ 不能用Armstrong公理从F中导出，那么它必然不被F逻辑蕴涵。

或者说，存在一个具体关系r，F中所有的函数依赖都满足r，而不能用公理推出的 $X \rightarrow Y$ 不满足r，即 $X \rightarrow Y$ 不被F逻辑蕴涵。

- 设 $X \rightarrow Y$ 不能用Armstrong公理导出，并建立关系r。

	X_F^+	$U - X_F^+$
t	11...1	00 ...0
s	11...1	11...1

公理完备性需证明：

(1) 在r中F的所有函数依赖都成立；

(2) 在r中， $X \rightarrow Y$ 不能成立。

Armstrong公理完备性 证明 (2)

X_F^+	$U - X_F^+$
t 11...1	00 ...0
s 11...1	11...1

(1) 设 $V \rightarrow W$ 是 F 中任一函数依赖，则有下列两种情况：

- a) $V \subseteq X_F^+$ 。因为 $V \subseteq X_F^+$ ，所以有 $X \rightarrow V$ ；于是 $X \rightarrow W$ 成立，所以 $W \subseteq X_F^+$ 。因为 r 中 X_F^+ 的值全相等，所以 $V \rightarrow W$ 在 r 上成立。
- b) $V \not\subseteq X_F^+$ 。如果 V 不完全属于 X_F^+ ，则 V 在两元组 t 和 s 上的属性值必不相等，则 $V \rightarrow W$ 在 r 上成立。

因此，在关系 r 中， F 的任一函数依赖都成立。

Armstrong公理完备性 证明 (3)

X_F^+	$U - X_F^+$
t 11...1	00 ...0
s 11...1	11...1

(2) 因为 $X \rightarrow Y$ 不能用公理从 F 推出, 则 $Y \not\subseteq X_F^+$,
而 $X \subseteq X_F^+$, 那么 r 中元组 t, s 在 X 上的值相等,
而在 Y 上的值不等, 则 $X \rightarrow Y$ 在 r 上不成立。

结论: 凡不能用公理推出的函数依赖都不被 F 逻辑蕴涵, 即凡是 F 逻辑蕴涵的函数依赖都能用Armstrong公理从 F 导出。

—Armstrong公理是完备的。



闭包的计算

- 求 X_F^+ 的算法

Input: X, F

Output: X_F^+

$X_F^+ := X;$

do

for any $A \subseteq X_F^+$ **do**

if 在 F 中存在函数依赖 $A \rightarrow B$

then $X_F^+ = X_F^+ \cup B$

while (X_F^+ 发生变化且 $X_F^+ \neq U$)

示例 (1)

- 求 X_F^+ 的示例1:

$R < U, F >$, $U = (A, B, C, D, E)$, $F = \{AB \rightarrow C, B \rightarrow D, C \rightarrow E, CE \rightarrow B, AC \rightarrow B\}$, 计算 $(AB)_F^+$ 。

所用依赖

$(AB)_F^+$

AB

AB \rightarrow C

ABC

B \rightarrow D

ABCD

C \rightarrow E

ABCDE

$(AB)_F^+ = ABCDE$

示例 (2)

- 求 X_F^+ 的示例2:

$R \langle U, F \rangle$, $U = (A, B, C, G, H, I)$, $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$, 计算 $(AG)_F^+$ 。

所用依赖	$(AG)_F^+$
	AG
$A \rightarrow B$	AGB
$A \rightarrow C$	AGBC
$CG \rightarrow H$	AGBCH
$CG \rightarrow I$	AGBCH I
$(AG)_F^+ = \text{AGBCH I}$	

示例 (3)

- 示例3

$R \langle U, F \rangle$, $U = (A, B, C, D, E, G)$, $F = \{A \rightarrow E, BE \rightarrow AG, CE \rightarrow A, G \rightarrow D\}$, 计算 $(AB)_F^+$ 。

所用依赖

$(AB)_F^+$

AB

$A \rightarrow E$

ABE

$BE \rightarrow AG$

ABEG

$G \rightarrow D$

ABEGD

$(AB)_F^+ = ABEGD$



函数依赖集等价与覆盖

- 函数依赖集等价

- 函数依赖集 F , G , 若 $F^+ = G^+$, 则称 F 与 G 等价。
- 如果 F 和 G 等价, 则称 F 覆盖 G , 同时 G 也覆盖 F 。
- $F^+ = G^+ \Leftrightarrow F \subseteq G^+, G \subseteq F^+$



函数依赖集的最小依赖集

- 最小依赖集

- 定义：若函数依赖集F满足下列条件，则称F为一个极小函数依赖集，也称为最小依赖集或最小覆盖：

- F中任一函数依赖 $X \rightarrow A$ ，A必是单属性。

- (右部单属性化)

- F中不存在这样的函数依赖 $X \rightarrow A$ ，使得F与 $F - \{X \rightarrow A\}$ 等价。(没有多余的FD)

- F中不存在这样的函数依赖 $X \rightarrow A$ ，在X中有真子集Z，使得F与 $F - \{X \rightarrow A\} \cup \{Z \rightarrow A\}$ 等价。(每个FD左部没有多余属性)

函数依赖集F的极小化处理

- 函数依赖集F的极小化处理

- **定理**：每个函数依赖集F均等价于一个极小函数依赖集 F_m ，此 F_m 为F的最小依赖集。

- **F的极小化算法**：

- 逐个检查F中各函数依赖 $FD_i: X \rightarrow Y$ ，若 $Y = A_1 A_2 \dots A_k$ ， $k \geq 2$ ，则用 $\{X \rightarrow A_i \mid i = 1, 2, \dots, k\}$ 代替 $X \rightarrow Y$ 。
 - 逐个检查F中各函数依赖 $X \rightarrow A$ ，设 $X = B_1 \dots B_m$ ，逐个考查 B_i ，若 $A \in (X - B_i)_F^+$ ，则以 $(X - B_i)$ 取代 X 。直到F不再改变。
 - 逐个检查F中各函数依赖 $X \rightarrow A$ ，令 $G = F - \{X \rightarrow A\}$ ，若 $A \in (X)_G^+$ ，则从F中去掉该函数依赖，直到F不再改变。

示例

- 示例1

$F = \{A \rightarrow B, B \rightarrow A, A \rightarrow C, B \rightarrow C\}$, 求 F_m 。

$F_m = \{A \rightarrow B, B \rightarrow A, B \rightarrow C\}$

或者

$F_m = \{A \rightarrow B, B \rightarrow A, A \rightarrow C\}$

- 示例2

$F = \{C \rightarrow A, A \rightarrow G, CG \rightarrow B, B \rightarrow A\}$, 求 F_m 。

$F_m = \{A \rightarrow G, C \rightarrow B, B \rightarrow A\}$



范式的概念 (1)

- 如果一个关系满足某个指定的约束集，则称它属于某种特定的**范式 (Normal Form)**；
- 满足最低要求约束的称为**第一范式**，简称**1NF**，
当一个关系只包含**原子值**这一约束时，称为**1NF**。
原子值即为二维表的每一行和列的交叉位置上总是精确地存在一个值，而不是值集。也就是不能“表中有表”；
- 满足“原子值”这一约束条件的关系称为规范化关系，简称**范式**。在关系数据库中，都是规范化的关系。

范式的概念 (1)

- 范式理论的发展过程：
 - 1971- 1972 CODD系统提出1NF, 2NF, 3NF的概念, 讨论了进一步规范化的问题。
 - 1974- CODD 和BOYCE提出BCNF。
 - 1976- FAGIN 提出4NF, 后来又提出了“投影-连接范式” PJNF, 也称5NF。
- 各级范式间的联系：
$$1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF$$
- 一个低一级范式的关系模式, 通过**模式分解**可以转换为若干个高级范式的关系模式的集合, 这一过程称作**规范化**。

2NF (1)

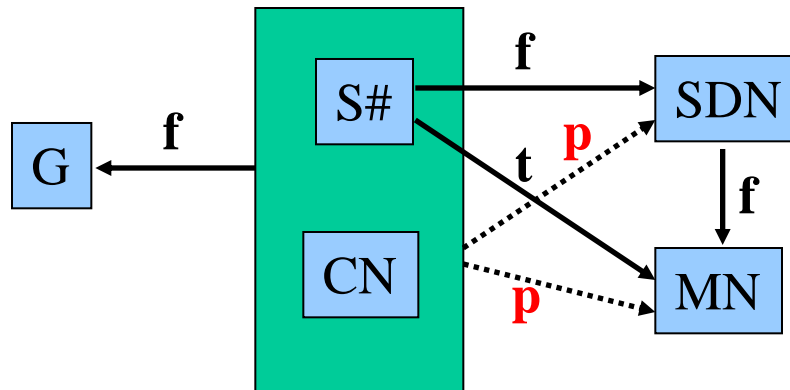
- 定义：若 $R \in 1NF$ ，且每个非主属性完全依赖于码，则称 $R \in 2NF$ ；
- 注意：
 - 如果关系 R 的全体属性都是 R 的主属性，那么 $R \in 2NF$ ；
 - 从 $1NF$ 中消除非主属性对码的部分函数依赖，则可获得 $2NF$ 关系；
 - 在 $2NF$ 中，允许主属性部分函数依赖于码。

2NF (2)

- 2NF的规范化

- 把1NF关系模式规范提高到2NF关系模式的集合。

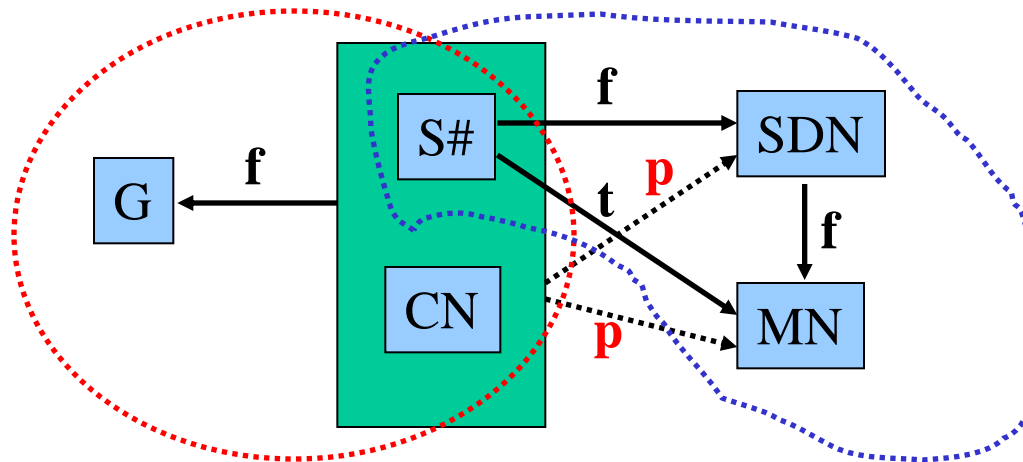
例：关系 UN (S#, CN, G, SDN, MN) \in 1NF,
其属性之间的函数依赖关系，用函数依赖图表示：



所以 $UN \notin 2NF$ 。

2NF (3)

- 采取**投影分解**方法，消除UN中的非主属性对码的部分函数依赖。



$$\text{UN} \longrightarrow \begin{cases} \text{SG} = \text{UN}[\underline{\text{S\#}}, \text{CN}, \text{G}] \in 2\text{NF} \\ \text{SDM} = \text{UN}[\underline{\text{S\#}}, \text{SDN}, \text{MN}] \in 2\text{NF} \end{cases}$$

2NF (4)

- 2NF存在的弊病

$SG = UN[\underline{S\#}, \underline{CN}, G] \in 2NF$

$SDM = UN[\underline{S\#}, SDN, MN] \in 2NF$

- **插入异常**有所改善，但还是存在：如果系中没有学生，则有关系的信息就无法插入。
- **删除异常**：如果删除学生的信息，所在系的信息也随之删除了。
- **数据冗余**得到一定改善：每个学生都存储了所在系的系主任的信息。

3NF (1)

- 定义：关系模式 $R\langle U, F \rangle$ 中，若不存在这样的码 X ，属性组 Y 及非主属性 $Z (Z \notin Y)$ ，使得下式成立， $X \rightarrow Y, Y \rightarrow Z, Y \not\rightarrow X$ 则称 $R \in 3NF$ 。

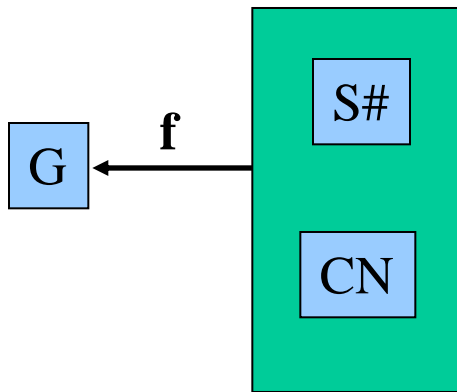
- 或定义为：

若关系模式 $R \in 2NF$ ，且每个非主属性都不传递依赖于 R 的任何码，则 $R \in 3NF$ 。

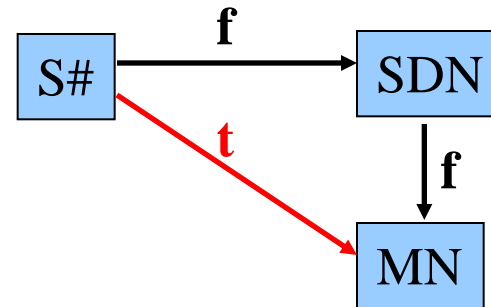
3NF (2)

- 3NF 规范化

UN \longrightarrow $\begin{cases} \text{SG}=\text{UN}[\underline{\text{S\#}}, \text{CN}, \text{G}] \in 2\text{NF} \\ \text{SDM}=\text{UN}[\underline{\text{S\#}}, \text{SDN}, \text{MN}] \in 2\text{NF} \end{cases}$



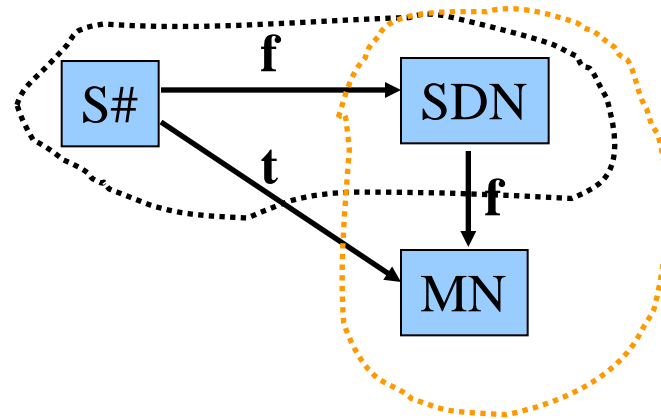
SG $\in 3\text{NF}$



SDM $\notin 3\text{NF}$

3NF (3)

- 采用投影分解的方法，将SDM规范到3NF。



SDM \rightarrow $\left\{ \begin{array}{l} \text{SD} = \text{SDM}[\underline{\text{S\#}}, \text{SDN}] \in 3\text{NF} \\ \text{SDM} = \text{SDM}[\underline{\text{SDN}}, \text{MN}] \in 3\text{NF} \end{array} \right.$

3NF (4)

- 所以有如下结果：

$$\text{UN} \longrightarrow \left\{ \begin{array}{l} \text{SG (S\#, CN, G)} \in 3\text{NF} \\ \text{SD (S\#, SDN)} \in 3\text{NF} \\ \text{SDM (SDN, MN)} \in 3\text{NF} \end{array} \right.$$

- SG, SD, DM均是单个关系表示单个实体，
所以，所有的“异常”、“毛病”都消失了。

BCNF (1)

- 3NF的不完善

- 3NF没有限制主属性对码的部分与传递函数依赖。如果发生这些依赖，仍可能存在插入异常、删除异常、修改异常。

- 例：

- STJ(S, T, J), S表示学生, T表示教师, J表示课程。

每位老师只教授一门课,每门课由若干教师教,某一学生选定某门课就确定了一个固定的教师,因此具有以下函数依赖:

$$T \rightarrow J, \quad (S, J) \rightarrow T$$

(S, T), (S, J) 为候选码。

STJ \in 3NF。

BCNF (2)

• STJ中存在的弊病

STJ(S, T, J)

- **插入异常**：如果没有学生选修某位老师的任课，则该老师担任课程的信息就无法插入。
- **删除异常**：删除学生选课信息，会删除掉老师的任课信息。
- **数据冗余**：每位学生都存储了有关老师所教授的课程的信息。
- **更新异常**：如果老师所教授的课程有所改动，则所有选修该老师课程的学生元组都要做改动。

BCNF (3)

- BCNF的定义:

- 若关系模式 $R<U, F>\in 1NF$, 如果对于R的每个函数依赖 $X\rightarrow Y$, 且 $Y\nsubseteq X$ 时, X必含有码, 则 $R<U, F>\in BCNF$ 。

- 由BCNF的定义可以看到, 每个BCNF的关系模式都具有如下三个性质:

- 所有非主属性都完全函数依赖于每个候选码。
 - 所有主属性都完全函数依赖于每个不包含它的候选码。
 - 没有任何属性完全函数依赖于非码的任何一组属性。

BCNF (4)

- BCNF的规范化

- STJ (S, T, J) , 因为 $T \rightarrow J$, 而T不是码。
所以, $STJ \notin BCNF$ 。
- 将S分解为ST (S, T) , TJ (T, J) 。



多值依赖与第四范式

- 属性之间的函数依赖反映了现实世界一些特性之间的相互约束。
- 现实世界一些特性之间的还有其他类型的约束：
 - 多值依赖(Multivalued Dependency)
 - 连接依赖(Join Dependency)
 - 分层依赖(Hierarchical Dependency)
 - 相互依赖(Mutual Dependency)。

多值依赖的定义

- 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 $X \twoheadrightarrow Y$ 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。
- 形式化定义：在 $R(U)$ 的任一关系 r 中，如果存在元组 t, s 使得 $t[X] = s[X]$ ，那么就必然存在元组 $w, v \in r$ ，（ w, v 可以与 s, t 相同），使得：

$$w[X] = s[X] = v[X] = t[X]$$

$$w[Y] = t[Y], \quad v[Y] = s[Y]$$

$$w[Z] = s[Z], \quad v[Z] = t[Z]$$

则称 Y 多值依赖于 X ，记作 $X \twoheadrightarrow Y$ 。

多值依赖与函数依赖的比较

- 有效性范围

- $X \rightarrow Y$ 的有效性仅决定于 X 、 Y 属性集上的值，它在任何属性集 W ($XY \subseteq W \subseteq U$) 上都成立；
- $X \twoheadrightarrow Y$ 在 U 上成立，则 $X \twoheadrightarrow Y$ 在属性集 W ($XY \subseteq W \subseteq U$) 上成立；
- $X \twoheadrightarrow Y$ 在属性集 W ($XY \subseteq W \subseteq U$) 上成立，但在 U 上不一定成立；
- 若 $X \rightarrow Y$ 在 $R(U)$ 上成立，则对于任何 $Y' \subseteq Y$ ，均有 $X \rightarrow Y'$ 成立；
- 若 $X \twoheadrightarrow Y$ 在 $R(U)$ 上成立， $Y' \subseteq Y$ ，则不一定有 $X \twoheadrightarrow Y'$ 成立。

示例 (1)

- 关系模式TEACH (C, T, B)，一门课程由多个教师担任，每个教师可以讲授多门课程；一门课程使用相同的一套参考书，每种参考书可被多门课程使用。
它的码是 (C, T, B)，所以属于BCNF。

C	T	B
物理	{张明, 张平}	{普通物理学, 光学原理}
化学	{李勇, 王微}	{无机化学, 有机化学}

C	T	B
物理	张明	普通物理学
物理	张明	光学原理
物理	张平	普通物理学
物理	张平	光学原理
化学	李勇	无机化学
化学	李勇	有机化学
化学	王微	无机化学
化学	王微	有机化学

T多值依赖于C，记作
 $C \twoheadrightarrow T$ ，
同样有 $C \twoheadrightarrow B$

多值依赖的性质

- 多值依赖有**对称性**
 - 若 $X \twoheadrightarrow Y$, 则 $X \twoheadrightarrow Z$, 其中 $Z=U-X-Y$
- 若 $X \rightarrow Y$, 则 $X \twoheadrightarrow Y$ 。即函数依赖可以看作多值依赖的特殊情况。
- 若 $X \twoheadrightarrow Y$, 而 $Z = \emptyset$, 则称 $X \twoheadrightarrow Y$ 为**平凡的多值依赖**, 否则,
若 $X \twoheadrightarrow Y$, 而 $Z \neq \emptyset$, 则称 $X \twoheadrightarrow Y$ 为**非平凡的多值依赖**。
- 若 $X \twoheadrightarrow Y$, $X \twoheadrightarrow Z$, 则 $X \twoheadrightarrow YZ$ 。
- 若 $X \twoheadrightarrow Y$, $X \twoheadrightarrow Z$, 则 $X \twoheadrightarrow Y \cap Z$ 。
- 若 $X \twoheadrightarrow Y$, $X \twoheadrightarrow Z$, 则 $X \twoheadrightarrow Y-Z$, $X \twoheadrightarrow Z-Y$ 。

4NF的定义

- 定义

- 关系模式 $R \langle U, F \rangle \in 1NF$ ，如果对于R的每个非平凡的多值依赖 $X \twoheadrightarrow Y$ ($Y \not\subseteq X$)，X都含有码，则称 $R \in 4NF$ 。
- 4NF所允许的非平凡的多值依赖实际上是函数依赖（左部含有码的）。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。

- 或定义

- 关系模式 $R \in BCNF$ ，且不存在非平凡的非函数依赖的多值依赖，则 $R \in 4NF$ 。
- 含义：若 $R \in BCNF$ ，当R中只存在函数依赖，则 $R \in 4NF$ ；或当R中存在平凡的多值依赖时， $R \in 4NF$ 。

4NF的规范化

C	T	B
物理	张明	普通物理学
物理	张明	光学原理
物理	张平	普通物理学
物理	张平	光学原理
化学	李勇	无机化学
化学	李勇	有机化学
化学	王微	无机化学
化学	王微	有机化学

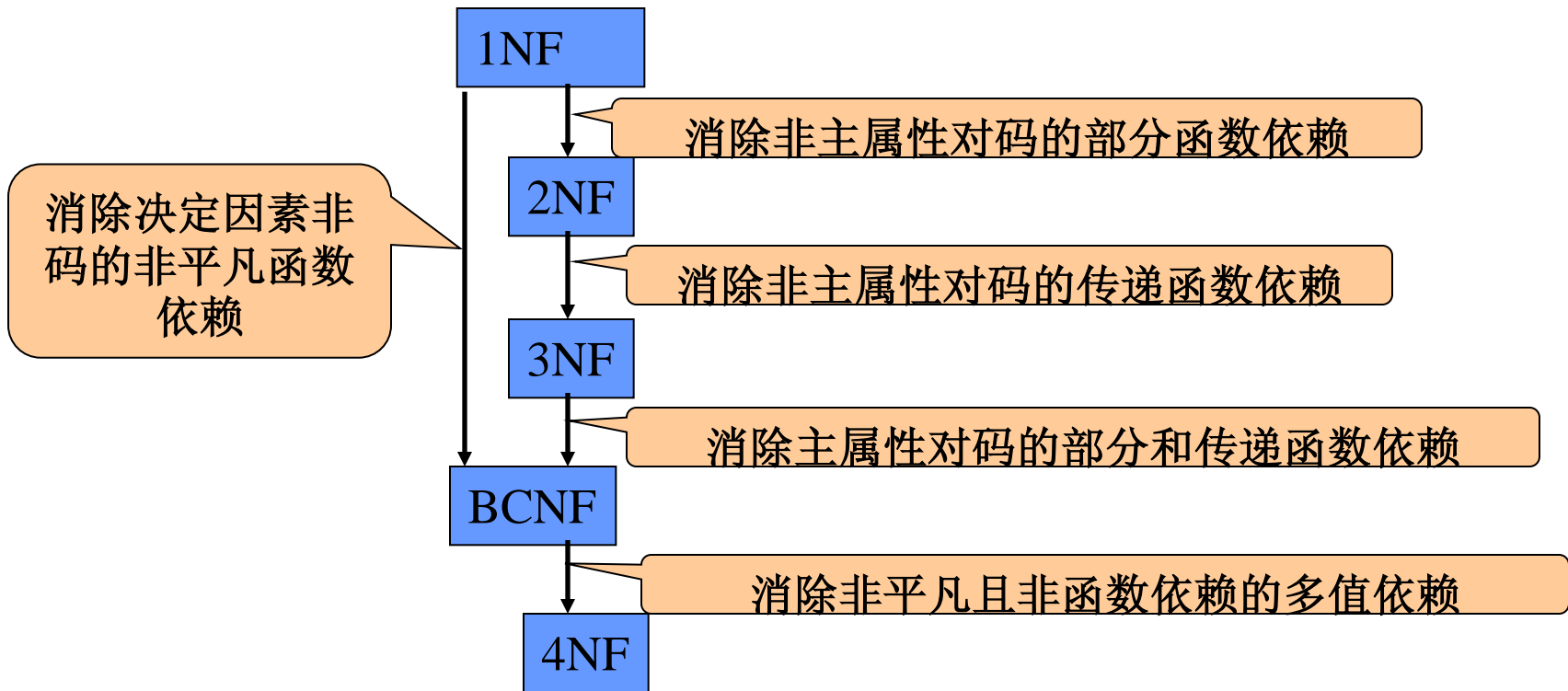
- 非4NF的关系存在的弊病是数据冗余太大
 - 例：TEACH (C, T, B)，由于 $C \twoheadrightarrow T$ ， $C \twoheadrightarrow B$ ，码为 (C, T, B)。所以TEACH \notin 4NF。
 - 如果一门课有m个教师，n本参考书，则同一门课将有 $m \times n$ 个元组。
- 采用模式分解的方法消去非平凡且非函数依赖的多值依赖
 - 例：将CTB分解为CT (C, T)，CB (C, B)。在CT、CB中，CT \in 4NF，CB \in 4NF。
 - 如果一门课有m个教师，n本参考书，则同一门课将有 $m+n$ 个元组。

规范化目的与基本思想

- 在关系数据库中，对关系的最基本要求是满足第一范式。这些关系常有一些异常或冗余等弊病。规范化的目的就是要消除这些弊病。
- 规范化的基本思想是逐步消除数据依赖中不合适的部分，使数据库模式中各关系模式达到某种程度的“分离”，使一个关系只描述一个实体或者实体间的一种联系。即“一事一地”的设计原则。规范化的实质是概念的单一化。

规范化的过程

- 规范化的过程概括如下：



范式之间的关系 (1)

- $3NF \subset 2NF$

反证：若 $R \in 3NF$ ，但 $R \notin 2NF$ ，则按 $2NF$ 定义，一定有非主属性部分依赖于码，

设 X 为 R 的码，则存在 X 的真子集 X' ，以及非主属性 Z ($Z \not\subset X'$)，使得 $X' \rightarrow Z$ 。

于是在 R 中存在码 X ，属性组 X' ，以及非主属性 Z ($Z \not\subset X'$)，使得 $X \rightarrow X'$ ， $X' \rightarrow Z$ ， $X' \rightarrow X$ 成立，这与 $R \in 3NF$ 矛盾。所以 $R \in 2NF$ 。

范式之间的关系 (2)

- $BCNF \subset 3NF$

反证：若 $R \in BCNF$ ，但 $R \notin 3NF$ ，则按 $3NF$ 定义，一定有非主属性对码的传递依赖，于是存在：

R 的码 X ，属性组 Y ，以及非主属性 Z ($Z \not\subset Y$)，使得 $X \rightarrow Y$ ， $Y \rightarrow Z$ ， $Y \not\rightarrow X$ 成立。

由 $Y \rightarrow Z$ ，按 $BCNF$ 定义， Y 含有码，于是 $Y \rightarrow X$ 成立，这与 $Y \not\rightarrow X$ 矛盾。所以 $R \in 3NF$ 。

- $4NF \subset BCNF$



模式分解理论

- 模式分解的定义
- 分解的无损连接性
- 分解的保持函数依赖性
- 模式分解的原则
- 模式分解的算法

模式分解的定义

- 关系模式分解

- 函数依赖集合 $F_i = \{X \rightarrow Y \mid X \rightarrow Y \in F^+ \wedge XY \subseteq U_i\}$, 称 F_i 为 F 在 U_i 上的投影。

- 关系模式 $R\langle U, F \rangle$ 的一个分解 ρ 是指

$$\rho = \{R_1\langle U_1, F_1 \rangle, R_2\langle U_2, F_2 \rangle, \dots, R_n\langle U_n, F_n \rangle\}$$

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

分解的无损连接性

- 设 $\rho = \{R_1\langle U_1, F_1 \rangle, R_2\langle U_2, F_2 \rangle, \dots, R_k\langle U_k, F_k \rangle\}$ 是 $R\langle U, F \rangle$ 的一个分解, r 是 $R\langle U, F \rangle$ 的一个关系, 定义 $m_\rho(r) = \bigotimes_{i=1}^k \Pi_{R_i}(r)$, 即 $m_\rho(r)$ 是 r 在 ρ 中各关系模式投影上的连接。这里 $\Pi_{R_i}(r) = \{t.U_i | t \in r\}$ 。若对于 $R\langle U, F \rangle$ 的任何一个关系 r , 都有 $r = m_\rho(r)$, 则称分解 ρ 具有无损连接性, 简称 ρ 为无损分解。

无损分解的判定算法 (1)

- **算法：**（判别一个分解的无损连接性）

$\rho = \{R_1 \langle U_1, F_1 \rangle, R_2 \langle U_2, F_2 \rangle, \dots, R_k \langle U_k, F_k \rangle\}$,
 $U = \{A_1, A_2, \dots, A_n\}, F = \{FD_1, FD_2, \dots, FD_p\}$, FD_i 为
 $X_i \rightarrow A_i, X_i \subseteq U$ 。

(1) 建立n列k行的表TB:

- 每一列对应一个属性 A_i ;
- 每一行对应分解中的一个关系模式 R_i 。
- 分量的取值: $C_{ij} = \begin{cases} a_j, A_j \in U_i \\ b_{ij}, A_j \notin U_i \end{cases}$

无损分解的判定算法 (2)

(2) 对 FD_i 中每一个函数依赖 $X \rightarrow Y$, 若 TB 中存在元组 t_1, t_2 , 使得 $t_1[X] = t_2[X]$, 则对每一个 $A_i \in Y$:

① 若 $t_1[A_i], t_2[A_i]$ 中有一个等于 a_i , 则另一个也改为 a_i ;

② 若 ① 不成立, 则取 $t_1[A_i] = t_2[A_i]$ (t_1 的行号小于 t_2)。

(3) 反复执行 (2), 直至:

① TB 中出现一行为 a_1, a_2, \dots, a_n 的一行。

② TB 不再发生变化, 且没有一行为 a_1, \dots, a_n 。

在 ① 情况下, ρ 为无损分解, 否则为有损分解。

无损分解的判定算法 (3)

- 例： $U=\{A,B,C,D,E\}$, $F=\{AB\rightarrow C, C\rightarrow D, D\rightarrow E\}$
 $\rho = \{(A, B, C), (C, D), (D, E)\}$

A	B	C	D	E
a_1	a_2	a_3	b_{14}	b_{15}
b_{21}	b_{22}	a_3	a_4	b_{25}
b_{31}	b_{32}	b_{33}	a_4	a_5

$AB\rightarrow C$		$C\rightarrow D$		
A	B	C	D	E
a_1	a_2	a_3	b_{14}	b_{15}
b_{21}	b_{22}	a_3	a_4	b_{25}
b_{31}	b_{32}	b_{33}	a_4	a_5

A	B	C	D	E
a_1	a_2	a_3	a_4	b_{15}
b_{21}	b_{22}	a_3	a_4	b_{25}
b_{31}	b_{32}	b_{33}	a_4	a_5

$D\rightarrow E$				
A	B	C	D	E
a_1	a_2	a_3	a_4	a_5
b_{21}	b_{22}	a_3	a_4	a_5
b_{31}	b_{32}	b_{33}	a_4	a_5

无损分解的判定准则

- **定理：** $R \langle U, F \rangle$ 的一个分解 $\rho = \{R_1 \langle U_1, F_1 \rangle, R_2 \langle U_2, F_2 \rangle\}$ 具有无损连接性的充分必要条件是 $U_1 \cap U_2 \rightarrow U_1 - U_2 \in F^+$ 或 $U_1 \cap U_2 \rightarrow U_2 - U_1 \in F^+$ 。
 - 即 R_1, R_2 的共同属性至少构成 R_1, R_2 二者之一的候选码。

分解的保持函数依赖性

- 定义：若 $F^+ = (\bigcup_{i=1}^n F_i)^+$ ，则称 $R < U, F >$ 的分解 $\rho = \{R_1 < U_1, F_1 >, \dots, R_n < U_n, F_n >\}$ 保持函数依赖。

- 保持函数依赖性的判定方法

设 $G = (\bigcup_{i=1}^n F_i)$ ，则

$$F^+ = G^+ \Leftrightarrow F \subseteq G^+, \text{ 且 } G \subseteq F^+$$

要判定 $F \subseteq G^+$ ，只需逐一对 F 中函数依赖 $X \rightarrow Y$ ，考察 Y 是否属于 $X_{G^+}^+$ 。若有 F 中的函数依赖不满足该条件，则 $F^+ \neq G^+$ ， ρ 未保持函数依赖。

- R 中的每个函数依赖都能够从 $R_1 \dots R_n$ 函数依赖的并集中逻辑导出。

示例

- 例： $R = \langle ABC, \{A \rightarrow B, B \rightarrow C\} \rangle$,
 $\rho = \{ \langle AB, \{A \rightarrow B\} \rangle, \langle AC, \{A \rightarrow C\} \rangle \}$

则 $G = \{A \rightarrow B, A \rightarrow C\}$, $G^+ = \{A \rightarrow B, A \rightarrow C, A \rightarrow BC\}$,
B 关于 G^+ 的闭包为 (B) , 因为对于 F 中 $B \rightarrow C$, $C \notin B$
关于 G^+ 的闭包, 所以 $F \not\subseteq G^+$, $F^+ \neq G^+$, ρ 未保持函
数依赖。

Rissanen 判别条件

- 判别是否具有无损连接性和保持函数依赖

关系R投影分解为R1和R2，如果

(1) R中的每个函数依赖都能够从R₁ 或 R₂逻辑导出，则分解保持函数依赖；

(2) R₁, R₂的共同属性至少构成 R₁, R₂ 二者之一的候选码，则分解具有无损连接性。

模式分解的原则

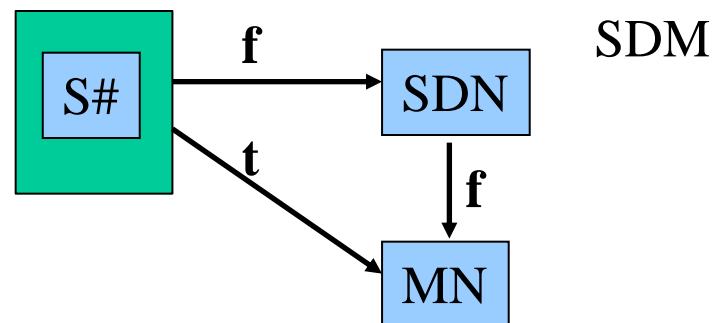
• 规范化中的问题

- 规范化通过投影分解来完成。投影分解不是唯一的。并且结果大不相同。
- 例如，对于SDM ($S\#$, SDN, MN) 的投影分解：

(1) SD ($S\#$, SDN) SM ($S\#$, MN)	(2) SM ($S\#$, MN) DM (SDN, MN)	(3) SD ($S\#$, SDN) DM (SDN, MN)
— 具有无损连接性; — 未保持函数依赖	— 不具有无损连接性; — 未保持函数依赖	— 具有无损连接性; — 保持函数依赖

• 投影分解中应遵循的原则：

- 具有无损连接性
- 保持函数依赖



模式分解能够达到的范式等级

- 模式分解能够达到的范式等级：
 - 若要求分解保持函数依赖，那么模式分解总可以达到3NF，但不一定能达到BCNF；
 - 若要求分解具有无损连接性，那一定可以达到4NF或更高；
 - 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。

关系模式的分解算法

- 关系模式的分解算法
 - 达到3NF且保持函数依赖的分解算法
 - 达到3NF且同时保持无损连接与函数依赖的分解算法
 - 达到BCNF无损连接分解算法

达到3NF的等价模式分解 (1)

- 达到3NF且保持函数依赖的分解算法：
 - 1.对F进行极小化处理，仍记为F。
 - 2.找出不在F中出现的属性，将它们构成一个关系模式，并从U中去掉它们(剩余属性仍记为U)。
 - 3.若有 $X \rightarrow A \in F$ ，且 $XA=U$ ，则 $\rho = \{R\}$ ，算法终止。
 - 4.对F按具有相同左部的原则进行分组（设为k组），每一组函数依赖所涉及的属性全体为 U_i ，若有 $U_i \subseteq U_j$ ($i \neq j$),则去掉 U_i 。令 F_i 为F在 U_i 上的投影，则 $\rho = \{R_1 \langle U_1, F_1 \rangle, \dots, R_k \langle U_k, F_k \rangle\}$ 是 $R \langle U, F \rangle$ 的一个保持函数依赖的分解，并且每个 $R_i \langle U_i, F_i \rangle \in 3NF$ 。

示例

— 示例1:

$U = \{S\#, SD, MN, C\#, G\}$

$F = \{S\# \rightarrow SD, S\# \rightarrow MN, SD \rightarrow MN, (S\#, C\#) \rightarrow G\}$

1. $F_m = \{S\# \rightarrow SD, SD \rightarrow MN, (S\#, C\#) \rightarrow G\}$

2. 保持函数依赖的分解: $\rho = \{$

$\{(S\#, SD), S\# \rightarrow SD\}$

$\{(SD, MN), SD \rightarrow MN\}$

$\{(S\#, C\#, G), (S\#, C\#) \rightarrow G\} \}$

分解具有无损连接性。

示例

– 示例2: $R(ABC; A \rightarrow C, B \rightarrow C)$

保持函数依赖分解:

$\rho = \{ \{AC; A \rightarrow C\}, \{BC; B \rightarrow C\} \}$ 。

分解是有损的。

达到3NF的等价模式分解 (2)

- 达到3NF且同时保持无损连接与函数依赖的分解

- 算法：设 $\rho = \{R_1 \langle U_1, F_1 \rangle, \dots, R_k \langle U_k, F_k \rangle\}$ 是 $R \langle U, F \rangle$ 的一个保持函数依赖的3NF分解。

设 X 为 $R \langle U, F \rangle$ 的码，

(1) 若有某个 U_i , $X \subseteq U_i$, 则 ρ 即为所求，

(2) 否则令 $\tau = \rho \cup \{R^* \langle X, F_X \rangle\}$, τ 即为所求。

示例

例1：求R (ABC; $A \rightarrow C$, $B \rightarrow C$) 的保持无损连接和函数依赖的3NF分解。

(1) 按保持函数依赖分解

进行分组, $\rho = \{\{AC; A \rightarrow C\}, \{BC; B \rightarrow C\}\}$ 。

(2) 码为AB

$$\tau = \rho \cup \{AB\}$$

最后的分解为：

$$\{\{AC; A \rightarrow C\}, \{BC; B \rightarrow C\}, \{AB\}\}$$

达到BCNF无损连接分解算法

- 算法:

给定关系模式 $R\langle U, F \rangle$,

(1) 令 $\rho = \{R\langle U, F \rangle\}$

(2) 检查 ρ 中各关系模式是否属于BCNF, 若是, 则算法终止。

(3) 设 ρ 中 $R_i\langle U_i, F_i \rangle$ 不属于BCNF,

则存在函数依赖 $X \rightarrow A \in F_i^+ (A \notin X)$, 且 X 不是 R_i 的码, 则 XA 是 U_i 的真子集, 将 R_i 分解为 $\sigma = \{S_1, S_2\}$,

其中 $U_{S_1} = XA$, $U_{S_2} = U_i - \{A\}$

以 σ 代替 R_i , 返回到 (2)。

示例

例1：有R $\langle U, F \rangle$ ，其中 $U=\{S\#, SD, MN, C\#, G\}$ ， $F=\{S\#\rightarrow SD, S\#\rightarrow MN, SD\rightarrow MN, (S\#,C\#)\rightarrow G\}$ ，将R无损分解到BCNF。

(1) $U_1=\{S\#, SD\}$, $F_1=\{S\#\rightarrow SD\}$

$U_2=\{S\#, MN, C\#, G\}$, $F_2=\{S\#\rightarrow MN, (S\#,C\#)\rightarrow G\}$

(2) $U_1=\{S\#, SD\}$, $F_1=\{S\#\rightarrow SD\}$

$U_2=\{S\#, MN\}$, $F_2=\{S\#\rightarrow MN\}$

$U_3=\{S\#, C\#, G\}$, $F_3=\{(S\#,C\#)\rightarrow G\}$

$\rho=\{R_1 \langle U_1, F_1 \rangle, R_2 \langle U_2, F_2 \rangle, R_3 \langle U_3, F_3 \rangle\}$ ，
且 R_1, R_2, R_3 均属于BCNF。



候选码的求解理论和算法

- 快速求解候选码的充分条件
- 左边为单属性的函数依赖集候选码成员的图论判定方法
- 多属性依赖集候选码求解法

候选码的求解理论和算法

对于关系 $R\langle U, F \rangle$ ，其属性可分为4类：

- **L**类：仅出现在 F 的函数依赖**左部**的属性。
- **R**类：仅出现在 F 的函数依赖**右部**的属性。
- **N**类：在 F 的函数依赖左右两边**均未出现**的属性。
- **LR**类：在 F 的函数依赖左右两边**均出现**的属性。

快速求解候选码

- **定理1:** 对于关系 $R\langle U, F \rangle$, 若 X ($X \subseteq U$) 是**L类属性**, 则 X 必为 R 的任一候选码成员。
 - **推论1.1:** 对于关系 $R\langle U, F \rangle$, 若 X ($X \subseteq U$) 是**L类属性**, 且 $X_F^+ = U$, 则 X 必为 R 的**唯一候选码**。
- **定理2:** 对于关系 $R\langle U, F \rangle$, 若 X ($X \subseteq U$) 是**R类属性**, 则 X 不在 R 的任何候选码中。
- **定理3:** 对于关系 $R\langle U, F \rangle$, 若 X ($X \subseteq U$) 是**N类属性**, 则 X 必包含在 R 的任一候选码中。
 - **推论3.1:** 对于关系 $R\langle U, F \rangle$, 若 X ($X \subseteq U$) 是**N类和L类组成的属性集**, 且 $X_F^+ = U$, 则 X 为 R 的**唯一候选码**。

示例

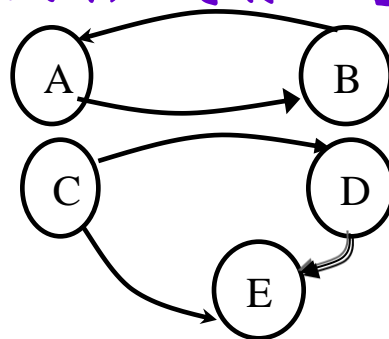
- 例1：设有关系模式R (A, B, C, D) , 其函数依赖集 $F=\{D \rightarrow B, B \rightarrow D, AD \rightarrow B, AC \rightarrow D\}$,求R的所有候选码。
- 例2：设有关系模式R (A, B, C, D, E, P) , 其函数依赖集 $F=\{A \rightarrow D, E \rightarrow D, D \rightarrow B, BC \rightarrow D, DC \rightarrow A\}$, 求R的所有候选码。

候选码的图论判定方法 (1)

- 定义1: 函数依赖关系图 G 是一个有序二元组 (U, F) , 记作 $G = (U, F)$, 其中:
 - (1) $U = (A_1, A_2, \dots, A_n)$ 是有限非空集, $A_i (i = 1, 2, \dots, n)$ 是 G 的结点, 它们表示对应关系模式 $R (A_1, A_2, \dots, A_n)$ 的属性。
 - (2) F 是 G 的边集, 其元素是 G 的一条有向边, 每条边 (A_i, A_j) 表示一个函数依赖 $A_i \rightarrow A_j$, 则 F 是 R 的单属性最小依赖集。

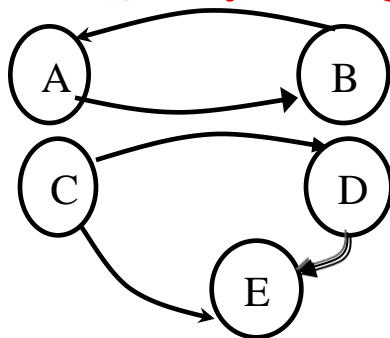
候选码的图论判定方法 (2)

- 定义2: 在一个函数依赖图中有如下术语:
 - 原始点: 只有引出线而无引入线的结点, 表示L类属性;
 - 终结点: 只有引入线而无引出线的结点, 表示R类属性;
 - 途中点: 既有引出线又有引入线的结点, 表示LR类属性;
 - 孤立点: 既无引出线又无引入线的结点, 表示N类属性;
 - 关键点: 原始点和孤立点统称为关键点;
 - 关键属性: 关键点对应的属性;
 - 独立回路: 不能由其他结点到达的回路。
- 回路: 有向图 $G=(V,E)$ 中,若边序列 $P=(e_{i1}, e_{i2}, \dots, e_{iq})$, 如果 e_{iq} 的终点也是 e_{i1} 的始点, 则称 P 是 G 的一条有向回路。



候选码的图论判定方法 (3)

- **定理4:** 关系模式R的函数依赖图G中若存在关键点, 则关键点对应的属性必在R的任何候选码中, 而所有终结点必不在R的任何候选码中。
- **定理5:** 属性集X是R的唯一候选码的充要条件是X能到达G中的任一结点。
 - **推论5.1:** 在单属性情况下, R具有唯一候选码的充要条件是G中不存在独立的回路。



候选码的图论判定方法 (4)

- **定理6:** 设 Y 是途中点, 则 Y 必在某个候选码中的充要条件是 Y 为某一独立回路中的结点。
- **定理7:** 设 F 是单属性依赖集, X 是 R 的关键属性集, G 中存在 K ($K \geq 1$) 个独立回路 r_1, r_2, \dots, r_k , 则:
 - (1) R 的候选码必不唯一;
 - (2) R 的候选码均由两部分构成:
 - 关键属性集 X (X 可为空);
 - K 个独立回路结点集的笛卡儿积的任一元素;
 - (3) 候选码的个数等于 k 个独立回路中结点个数的乘积。
 - (4) 每个候选码所含属性个数是一个常数, 等于关键属性个数加上独立回路个数 k 。

候选码的图论判定方法 (5)

- 算法1：单属性依赖集图论求解法。

- 输入：关系模式R，R的单属性函数依赖集F。
- 输出：R的所有候选码。

算法：

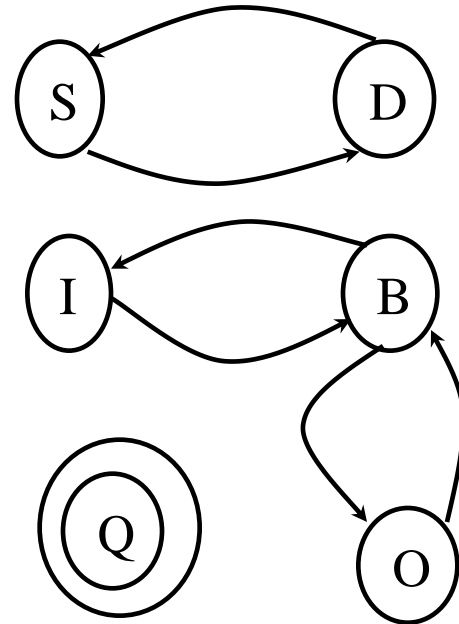
- (1) 求F的最小依赖集 F_m ；
- (2) 构造函数依赖图G；
- (3) 从G中找出关键属性集X（X可为空）；
- (4) 查看G中是否有独立回路，若无则输出X即为R的唯一候选码，结束；若有则继续（5）；
- (5) 从各独立回路中各取一结点对应的属性与X组成一个候选码，并重复这一过程，直至取尽所有可能组合，即为R的全部候选码。结束。

示例

- 例：设 $R(O, B, I, S, Q, D)$ ， $F=\{S \rightarrow D, D \rightarrow S, I \rightarrow B, B \rightarrow I, B \rightarrow O, O \rightarrow B\}$ ，求 R 的所有候选码。

解：

- $F_m = F$;
- 构造函数依赖图;
- 关键属性集 $\{Q\}$;
- 有四条回路，两条独立回路，
所以共有6个候选码，每个
候选码有3个属性;
- R 的所有候选码为 $QSI, QSB, QSO, QDI, QDB, QDO$ 。



多属性依赖集候选码求解法

- 算法2：多属性依赖集候选码求解法。

输入：关系模式 $R\langle U, F \rangle$;

输出：R的所有候选码。

算法：

(1) 将R的所有属性分为L、R、N和LR四类，并令X代表L、N两类，Y代表LR类。

(2) 求 X_F^+ ，若 $X_F^+ = U$ ，则X即为R的唯一候选码，结束。否则，继续。

(3) 对于Y中的任一属性A，求 $(XA)_F^+$ ，若 $(XA)_F^+ = U$ ，则XA为一候选码；否则在Y中依次取两个、三个、...，求其属性闭包，直到其闭包包含R的全部属性。



修改初步E-R图设计基本E-R图

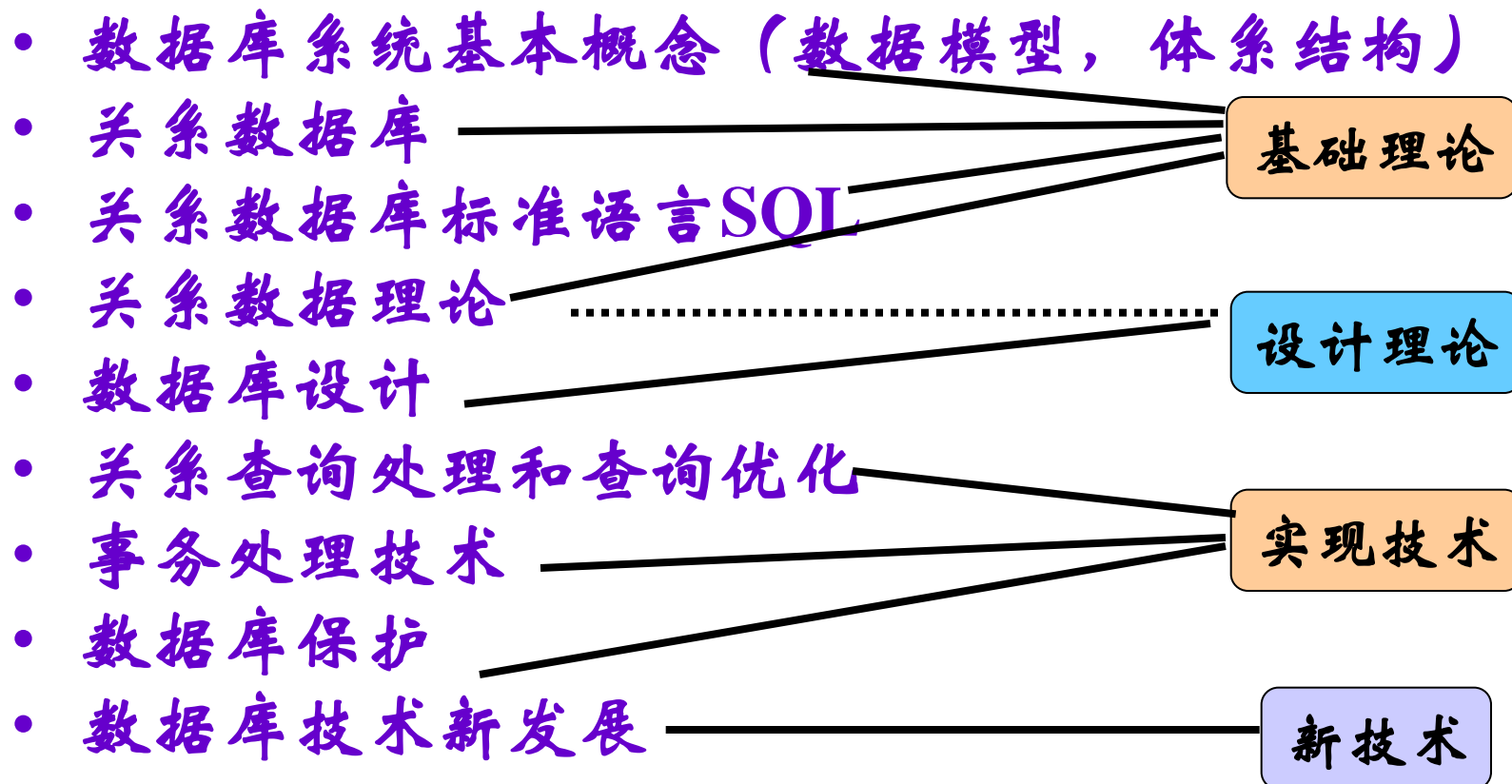
- 通过求最小依赖集消除冗余联系
 - 建立函数依赖集F;
 - 根据需求分析的结果
 - 根据E-R图
 - 把E-R图中实体用符号表示。
 - 对每一对n:1、1:1或n:m联系表示为实体码之间的函数依赖表达式 $X \rightarrow Y$ 。
 - 利用函数依赖集的最小覆盖算法进行极小化处理。设原函数依赖表达式集合为F，最小覆盖集为G，则 $D=F-G$
 - 考察D中每一个函数依赖表达式，确定是否冗余联系。
 - 去掉冗余联系后形成基本E-R图。

关系模型的规范化与优化

- 按照数据依赖的理论，逐一分析转换所得关系模式，判断是否存在部分函数依赖、传递函数依赖、多值依赖等，确定它们的范式等级。
- 按应用系统的处理要求，确定是否进行模式合并或分解。
- 一般情况下3NF已足以满足要求。



课程内容



第六章 关系查询处理与查询优化

- 关系查询处理步骤
- 查询操作的实现算法
- 查询优化
 - 查询的代数优化
 - 查询的物理优化
 - 查询优化的一般步骤

DBMS体系结构 (1)

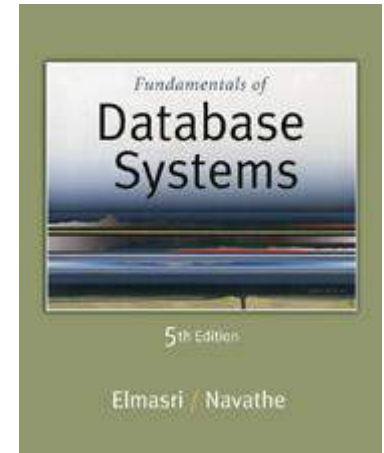
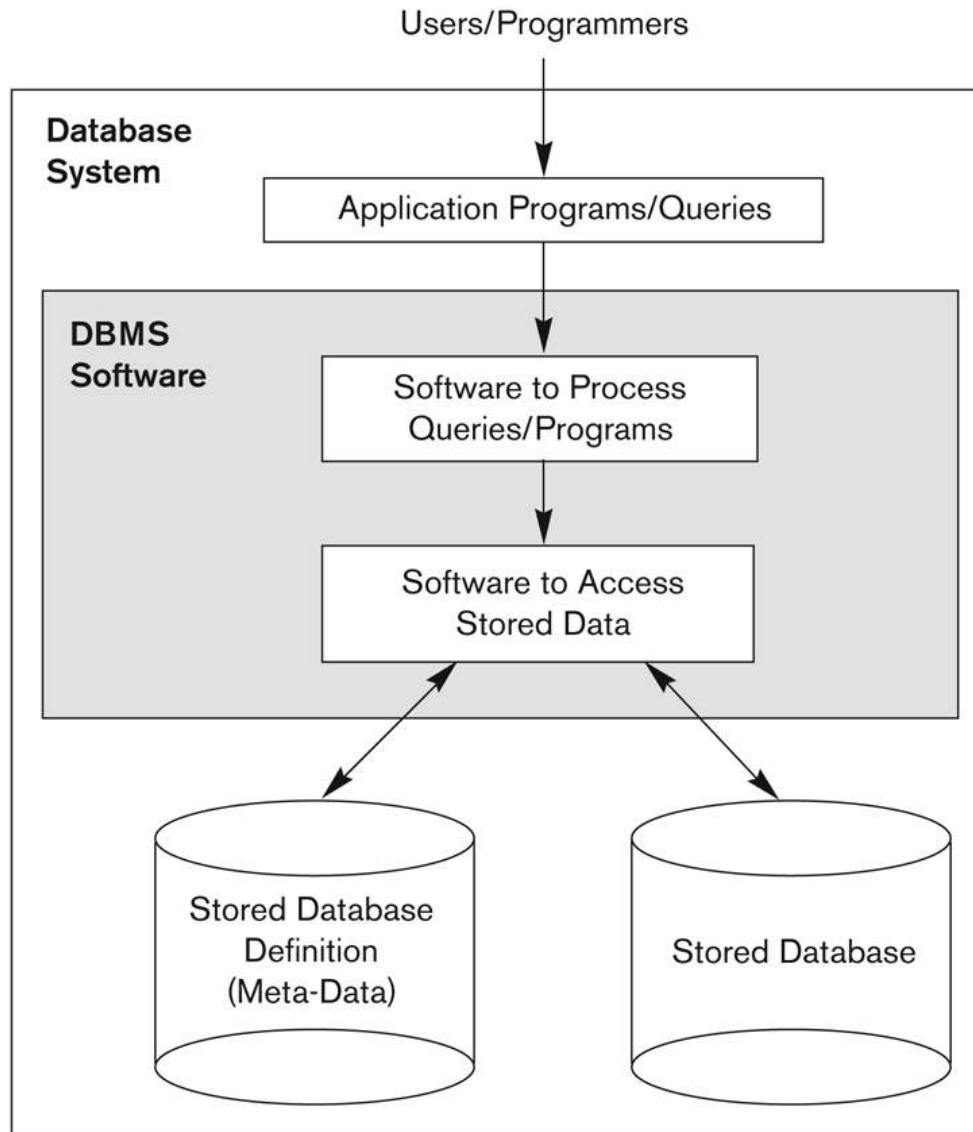


Figure 1.1
A simplified database
system environment.

DBMS体系结构 (2)

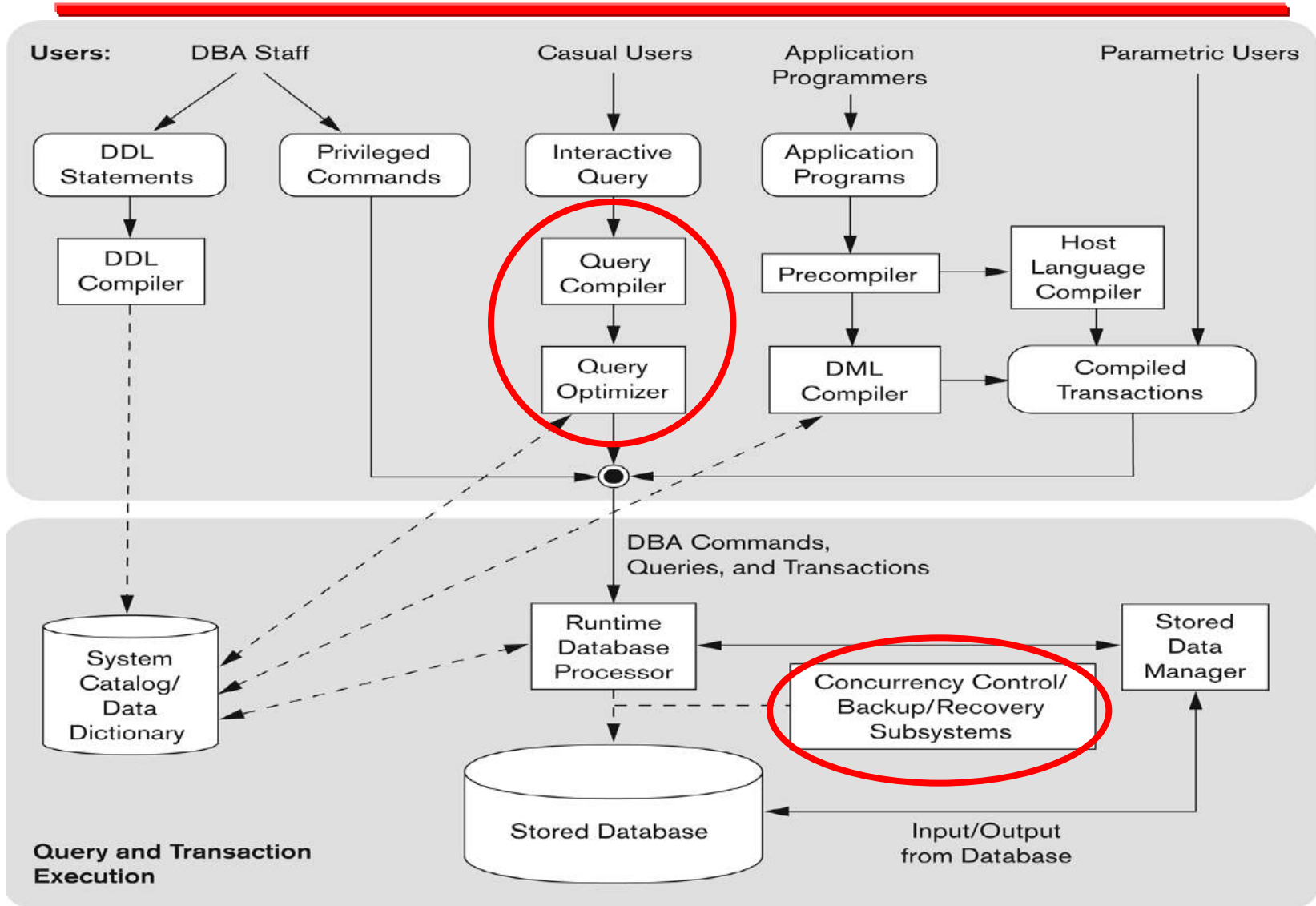


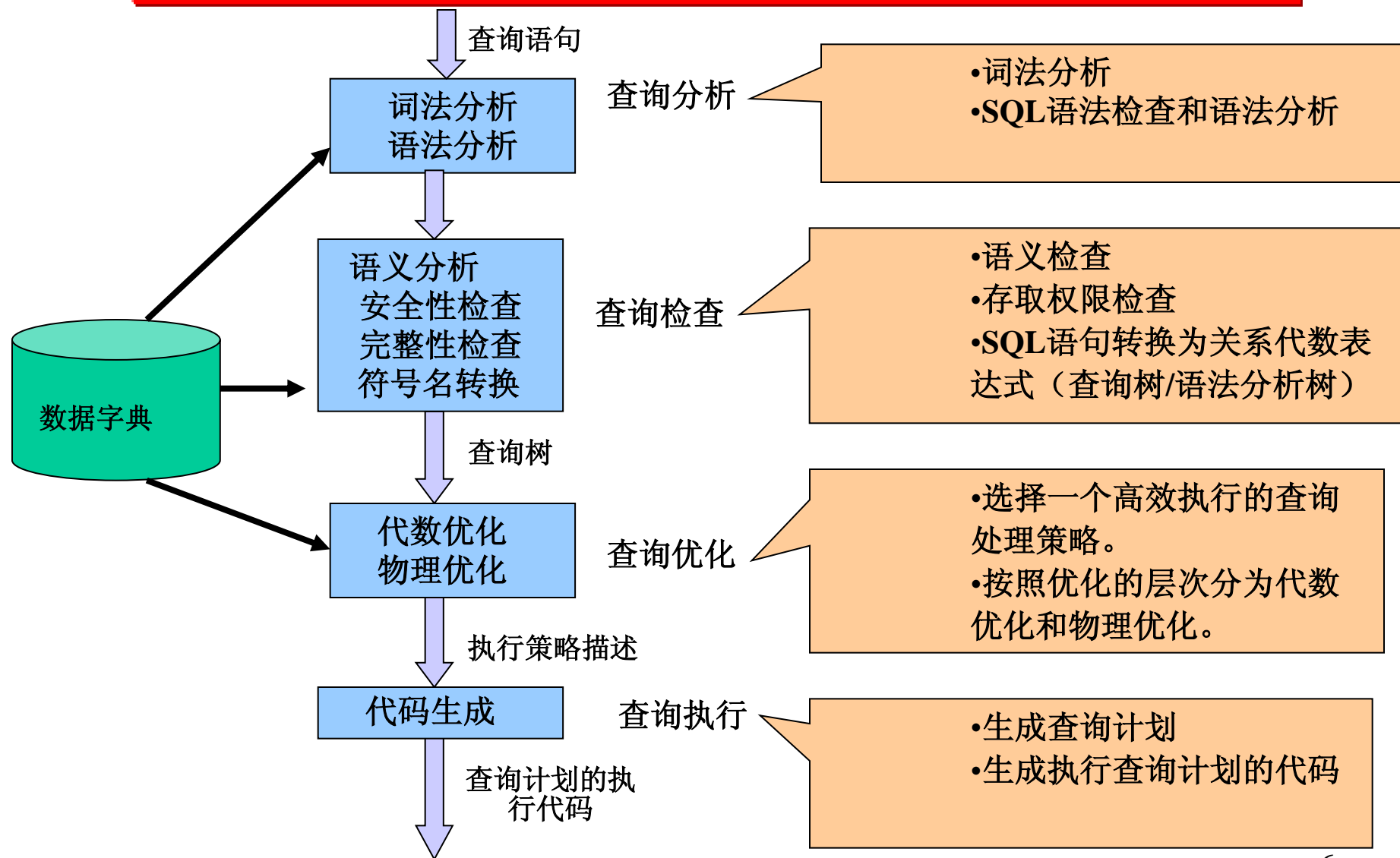
Figure 2.3

Component modules of a DBMS and their interactions.

关系查询处理步骤

- 关系查询处理的四个阶段：
 - 查询分析
 - 查询检查
 - 查询优化
 - 查询执行

关系查询处理步骤



查询优化

- 选择一个高效执行的查询处理策略。
- 按照优化的层次分为代数优化和物理优化。
 - 代数优化：关系代数表达式的优化，即按照一定的规则，**改变代数表达式中操作的次序和组合**
 - 物理优化：**存取路径和底层操作算法的选择**。选择依据——基于规则的、基于代价的、基于语义的

代数优化 (1)

- 关系代数表达式等价变换规则

- 连接、笛卡尔积交换律

$$E_1 \times E_2 \equiv E_2 \times E_1$$

$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$$

$$E_1 \bowtie_F E_2 \equiv E_2 \bowtie_F E_1$$

- 连接、笛卡尔积的结合律

$$(E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3)$$

$$(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$$

$$(E_1 \bowtie_{F_1} E_2) \bowtie_{F_2} E_3 \equiv E_1 \bowtie_{F_1} (E_2 \bowtie_{F_2} E_3)$$

代数优化 (1)

– 投影的串接定律

$$\pi_{A_1, A_2, \dots, A_n}(\pi_{B_1, B_2, \dots, B_m}(E)) \equiv \pi_{A_1, A_2, \dots, A_n}(E)$$

其中 $\{A_1, A_2, \dots, A_n\}$ 是 $\{B_1, B_2, \dots, B_m\}$ 的子集

– 选择的串接定律

$$\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_1 \wedge F_2}(E)$$

– 选择与投影的交换律

$$\sigma_F(\pi_{A_1, A_2, \dots, A_n}(E)) \equiv \pi_{A_1, A_2, \dots, A_n}(\sigma_F(E))$$

– 选择与笛卡尔积交换律

• F中只有E1的属性: $\sigma_F(E_1 \times E_2) \equiv \sigma_F(E_1) \times E_2$

• $F = F_1 \wedge F_2$, 且F1只有E1的属性、F2中只有E2的属性:

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F_1}(E_1) \times \sigma_{F_2}(E_2)$$

• F1只有E1的属性, F2有E1和E2的属性:

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F_2}(\sigma_{F_1}(E_1) \times E_2)$$

代数优化 (1)

— 选择与并的分配律

$$\sigma_F(E_1 \cup E_2) \equiv \sigma_F(E_1) \cup \sigma_F(E_2)$$

— 选择与差的分配律

$$\sigma_F(E_1 - E_2) \equiv \sigma_F(E_1) - \sigma_F(E_2)$$

— 选择对自然连接的分配律

$$\sigma_F(E_1 \bowtie E_2) \equiv \sigma_F(E_1) \bowtie \sigma_F(E_2)$$

— 投影与笛卡尔积的分配律

$$\pi_{A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m}(E_1 \times E_2) \equiv \pi_{A_1, A_2, \dots, A_n}(E_1) \times \pi_{B_1, B_2, \dots, B_m}(E_2)$$

— 投影与并的分配律

$$\pi_{A_1, A_2, \dots, A_n}(E_1 \cup E_2) \equiv \pi_{A_1, A_2, \dots, A_n}(E_1) \cup \pi_{A_1, A_2, \dots, A_n}(E_2)$$

代数优化 (2)

- 查询树的启发式优化（优化的一般准则）
 - 选择运算尽可能先做。是优化策略中最重要、最基本的一条。
 - 把投影运算和选择运算同时进行。
 - 把投影同其前或其后的双目运算结合起来。
 - 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。
 - 找出公共子表达式，把公共子表达式的结果写入中间文件，重复使用。

物理优化

- 选择高效合理的操作算法或存取路径，得到优化的查询计划。
- 常用方法
 - 基于规则的启发式优化方法
 - 基于代价估算的优化方法
 - 两者结合的优化方法

实现查询操作的算法

- 选择(选取)操作的实现算法示例
 - 全表扫描法
 - 索引扫描法
 - 如果在选择条件的属性上有索引，先通过索引找到目标索引项，再通过索引项找到元组

实现查询操作的算法

- 连接操作的实现算法示例

- 嵌套循环法

- 两个连接的表，第一个表为外循环，另一个为内循环

- 排序-合并法

- 两个表都按照连接属性排序，取第一个表元组的连接属性与第二个表元组的比较

- 索引连接法

- 第二个表按照连接属性建索引，取第一个表元组的连接属性与第二个表元组的连接属性比较

- Hash join 法

- 连接属性作为hash码，用同一个hash函数把两个连接表的元组散列到各自的hash文件

基于启发式规则的存取路径选择优化

- 选择操作的启发式规则

- 对于小关系，使用全表顺序扫描。
- 对于大关系：可以采用索引扫描法（如结果的元组数目较小），全表顺序扫描。

- 连接操作的启发式规则

- 如果两个表都已经按照连接属性排序——排序-合并法；
- 如果一个表在连接属性上有索引——索引连接法；
- 如果连接属性上未排序且未建索引，且其中一个表较小——Hash join法；
- 最后可选用嵌套循环法，并选择较小的表为外循环表。

查询优化的一般步骤

- 把查询转换成语法树，如关系代数语法树。
- 把语法树利用代数优化转换成优化后的标准形式。
- 利用基于启发式规则的物理优化，选择底层的存取路径。
- 生成查询计划，利用基于代价的物理优化，选择代价最小的。

第七章 事务处理技术

- 事务的概念
- 数据库恢复技术
- 并发控制技术

DBMS体系结构

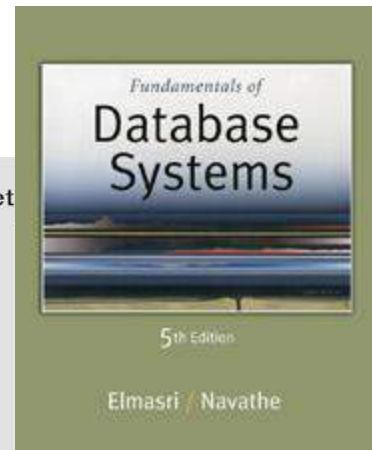
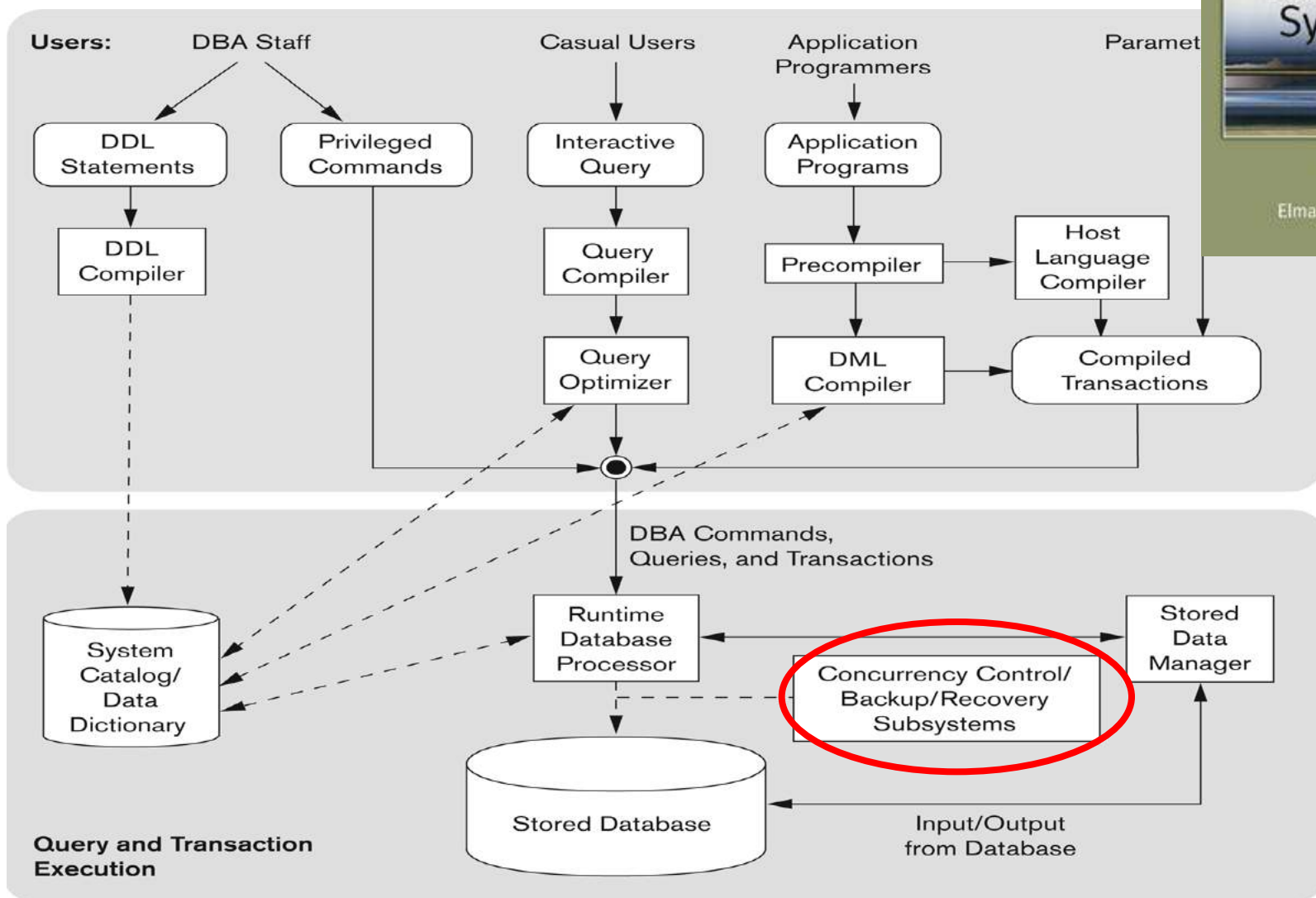


Figure 2.3

Component modules of a DBMS and their interactions.

事务的定义

- **事务(Transaction)**是用户定义的数据库操作序列，这些操作要么都做，要么都不做，是一个不可分割的工作单位。
 - **事务与应用程序**是两个概念，一般来说，一个应用程序可以包含多个事务。
 - **事务的开始与结束可以由用户显式控制**。如果用户没有显式定义事务，则由DBMS按缺省规定自动划分事务。

示例

例：银行转帐：事务T从A帐户过户50元到B帐户。

read(X)：从数据库传送数据项X到事务的工作区中。

write(X)：从事务的工作区中将数据项X写回数据库。

```
T:      read(A);  
  
        A := A - 50;  
  
        write(A);  
  
        read(B);  
  
        B := B + 50;  
  
        write(B);
```

事务的特性 (1)

- 原子性(Atomicity)
 - 事务中包括的所有操作要么都做，要么都不做
- 一致性(Consistency)
 - 事务执行的结果必须是使数据库从一个一致性状态，变到另一个一致性的状态

事务的特性 (2)

- 隔离性(Isolation)

- 一个事务的执行不能被其它事务干扰。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

- 持久性(Durability)

- 一个事务一旦提交之后，它对数据库的影响必须是永久的。其它操作或故障不应该对其执行结果有任何影响

事务的特性 (3)

- 事务的**ACID**特性对于数据库数据的正确、有效具有重要意义。但事务的特性有可能遭破坏，主要有两种情况：
 - 多个事务并行运行时，不同事务的操作交叉进行；
 - 事务在运行过程中被强行停止。

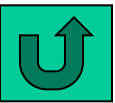
事务的特性 (4)

- 利用数据库并发控制机制以及数据库恢复机制保证事务的特性不被破坏，从而保证数据库数据的正确、有效
 - 原子性由恢复机制实现
 - 一致性是由事务的原子性保证的
 - 隔离性通过并发控制机制实现
 - 持久性通过恢复机制实现
- 事务是数据库恢复和并发控制的基本单位

SQL中事务的定义

- 在SQL中定义事务的语句有三条：

事务开始	BEGIN TRANSACTION	
事务结束	提交事务，正常结束。	COMMIT
	撤消全部更新，回滚到事务开始时状态。非正常结束。	ROLLBACK



数据库恢复技术

- 什么是数据库恢复
- 故障的种类
- 恢复的实现技术
- 恢复的策略
- 具有检查点的恢复技术
- 数据库镜像

数据库恢复技术

- 什么是数据库恢复技术
 - 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正确状态的功能，这就是数据库的恢复。
 - 数据库恢复是通过数据库管理系统的恢复子系统完成的。
- 数据库恢复子系统的意义
 - 保证事务的原子性。实现事务非正常终止时的回滚。
 - 当系统发生故障以后，数据库能够恢复到正确状态。
 - 事务内部的故障
 - 系统故障
 - 介质故障
 - 计算机病毒



数据库系统中故障的种类 (1)

- 事务内部的故障

- 可预期的：事务根据内部的测试条件，确定是否回滚。
- 不可预期的：指不能由应用程序处理的事务故障，如死锁，运算溢出，违反完整性规则等。

- 系统故障

- 是指造成系统停止运行的任何事情，使得系统要重新启动。如硬件错误，操作系统故障，停电等。

这类故障打断所有正在运行的事务，使事务都异常中止，但不会破坏数据库。

数据库系统中故障的种类 (2)

- 介质故障

- 介质故障指外存故障，如磁盘损坏，瞬时强磁场干扰等。

- 这类故障将破坏全部或部分数据库，并影响正在存取这部分数据的所有事务。

- 计算机病毒

- 计算机病毒是一种人为的破坏或故障，已成为数据库系统的主要威胁之一。

- 多数病毒对数据进行非法修改。

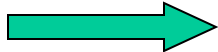


恢复的实现技术

- 故障对数据库系统的两种影响
 - 数据库本身被破坏，如介质故障
 - 数据库没有破坏，但数据可能不正确。
 - 如事务故障、系统故障，由于事务的非法终止造成；
 - 计算机病毒
- 数据库恢复的原理
 - 数据库恢复的基本原理为冗余。即利用存储在系统别处的冗余数据来重建或恢复修正数据库。

恢复的实现技术

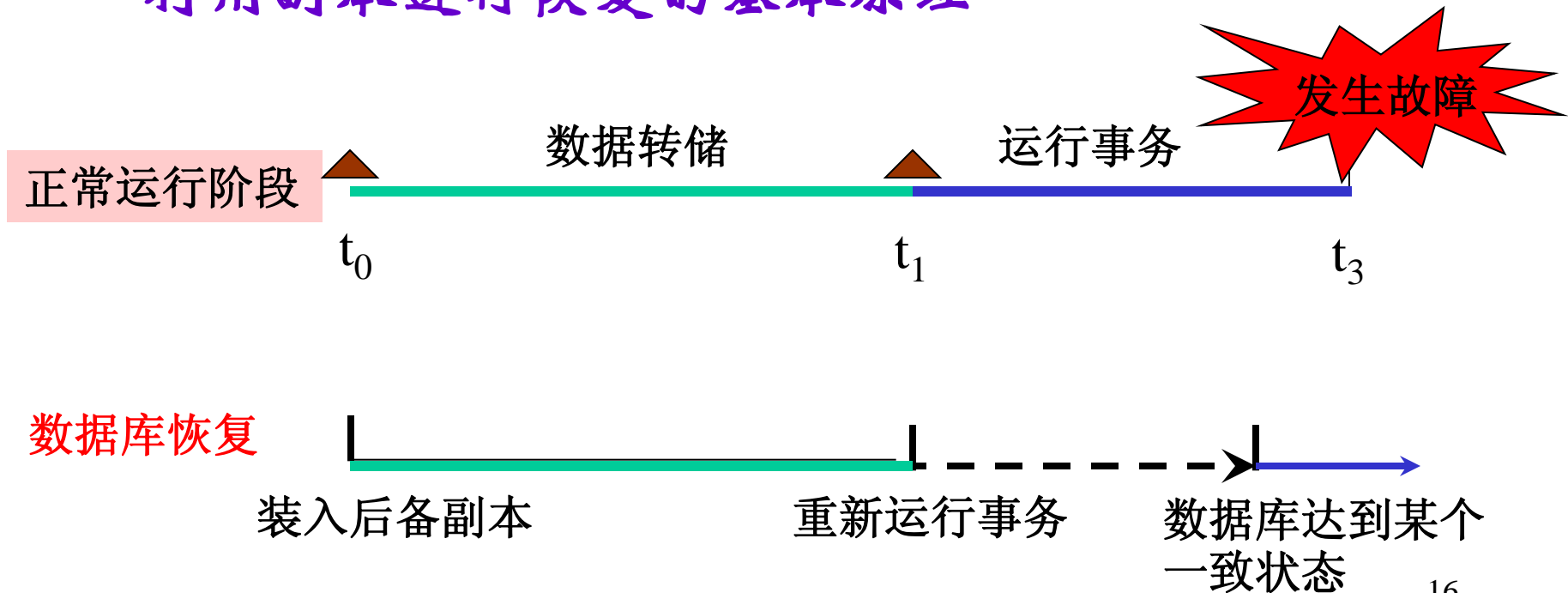
- 数据库恢复的关键问题

- 如何建立冗余  数据转储与登录日志文件

- 如何利用冗余实施数据库恢复

数据转储 (1)

- 是DBA定期地将整个数据库复制到磁带或另一个磁盘上保存起来的过程。这些备用的数据文本称为**后备副本**或**后援副本**。
- 利用副本进行恢复的基本原理



数据转储 (2)

- 数据转储分为两种转储状态
 - **静态转储**：系统中无事务运行时进行的转储操作。并且转储过程中，不允许对数据库进行任何存取、修改。
 - 优点：保证副本的数据一致性；
 - 缺点：由于转储必须等待正在运行的事务结束才能开始，而新的事务必须等待转储结束才能执行，降低了数据库的可用性。
 - **动态转储**：转储期间允许对数据库进行存取或修改。
 - 优点：不影响数据库的可用性
 - 缺点：不能保证副本上的数据正确、有效。还必须把转储期间各事务对数据库的修改记录下来，建立日志文件。**后援副本加上日志文件**就能把数据库恢复到某一时刻的正确状态。

- 数据转储可有两种转储方式

- 海量转储

- 海量转储指每次转储全部数据库。

- 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。

日志文件的建立与使用

- 日志文件格式与内容

日志是用来记录事务对数据库更新操作的文件。日志文件主要有两种格式：以记录为单位的日志文件和以数据块为单位的日志文件。

- 记录为单位的日志文件

- 记载的内容

- 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

- 每个日志记录中包含的信息项

- 事务标识（标明是哪个事务）
 - 操作的类型（插入、删除或修改）
 - 操作对象（记录的内部标识）
 - 更新前数据的旧值（对插入操作，此项为空）
 - 更新后数据的新值（对删除操作，此项为空）

- 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以及更新前和更新后的数据块。

- 日志文件的作用

- 事务故障和系统故障恢复必须使用日志文件。
- 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综合起来才能有效地恢复数据库。
- 在静态转储方式中，用日志文件恢复转储结束时刻到故障点间的事务。

- 日志文件的写入规则

- 登记的次序严格按并发事务执行的时间顺序；
- 必须先写日志文件，后写数据库。



故障的恢复策略

- 事务故障的恢复—UNDO，即撤消事务在不影响其它事务的情况下，强行回滚，撤消已做的修改。具体步骤：
 - 反向扫描日志文件，查找该事务的更新操作。
 - 对该事务的更新操作（插入、删除、修改）执行逆操作，即将日志记录中的“更新前的值”写入数据库。
 - 如此处理下去，直到读到该事务的开始标志。

• 系统故障——UNDO+REDO

- 系统故障造成数据库不一致状态的原因有两个：
 - 一是未完成的事务对数据库的更新可能已经写入数据库；
 - 二是已提交事务对数据库的更新可能还留在缓冲区未写入数据库。
- 因此恢复操作就是要撤销(UNDO)故障发生时未完成的事务，重做(REDO)已完成的事务。

- 系统故障恢复具体步骤:

- 正向扫描日志文件, 找出故障发生前已经提交的事务, 将其事务标识记入**重做 (REDO) 队列**。同时找出故障发生时尚未完成的事务, 将其事务标识记入**撤销 (UNDO) 队列**。
- 对撤销队列中的各个事务进行UNDO处理。
- 对重做队列中的各个事务进行REDO处理。

- 介质故障的恢复

- 恢复的方法是：

- 装入最新的数据库后备副本，使数据库恢复到最近一次转储时的一致状态。对于动态转储的副本，还需要装入转储开始时刻的日志文件副本，将数据库恢复到一致状态；
 - 装入转储以后的日志文件副本，重做已经完成的事务。



具有检查点的恢复技术

- 利用日志技术进行恢复时，恢复子系统通常**需要检查大量日志记录**，存在的问题是：
 - 搜索日志耗费大量时间
 - 不必要重做某些事务
- **检查点技术**可以改善效率，使得在检查点之前提交的事务，在数据库恢复处理时不必重做。

检查点技术 (1)

- 检查点技术在日志文件中增加一类新的记录—检查点 (checkpoint) 记录
- 检查点记录的内容包括：
 - 建立检查点时刻所有正在执行的事务清单
 - 这些事务最近一个日志记录的地址
- 系统中增加一个重新开始文件，用来记录各个检查点记录在日志文件中的地址
- 恢复子系统在登录日志文件期间动态地维护日志

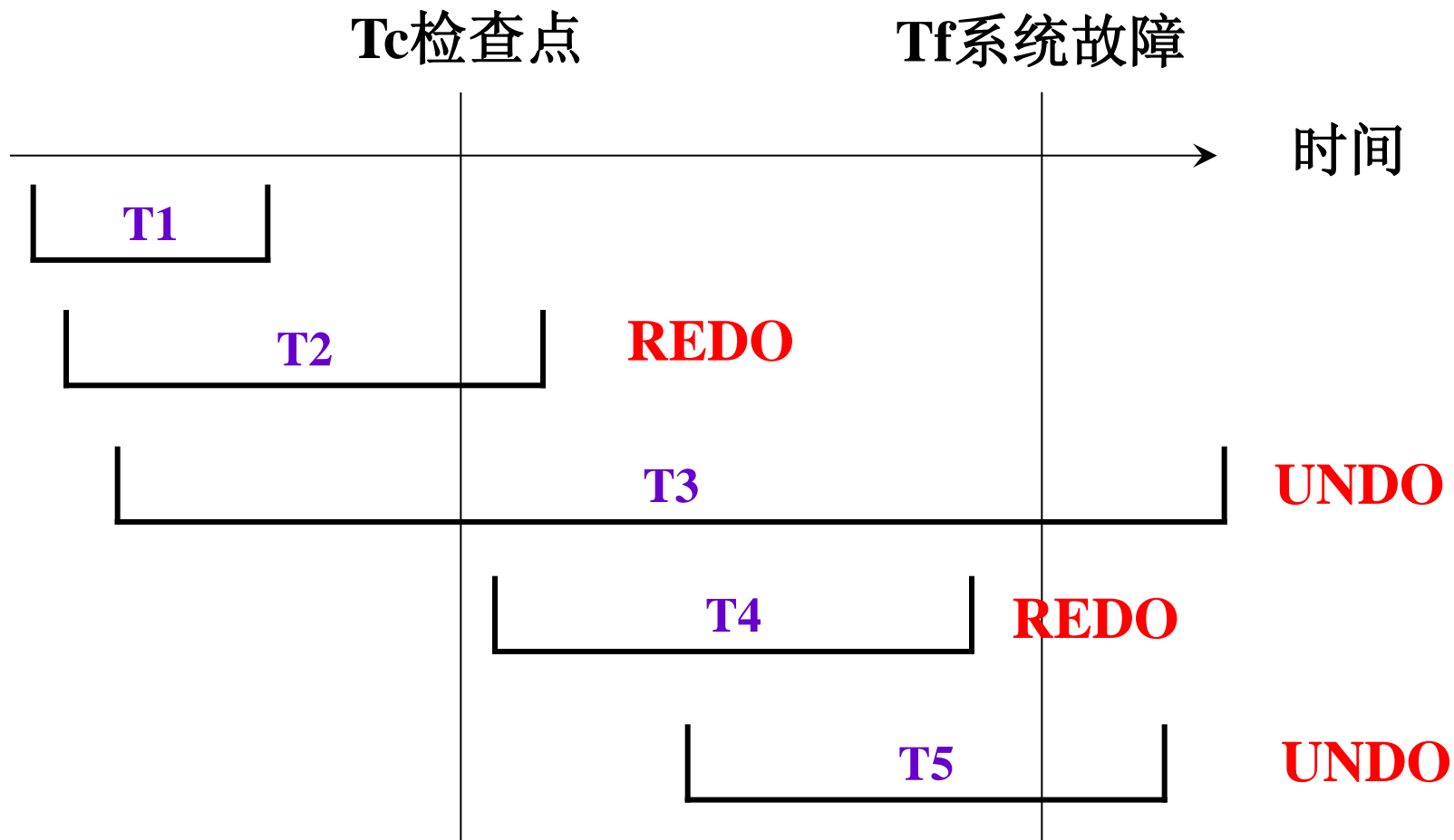
检查点技术 (2)

- 动态维护日志文件的方法是周期性地执行如下操作：
 - 将当前日志缓存中的所有日志记录写入磁盘的日志文件上。
 - 在日志文件上写入一个检查点记录。
 - 将当前数据缓存的所有数据记录写入磁盘的数据库中。
 - 把检查点记录在日志文件中地址写入重新开始文件。

利用检查点技术进行恢复

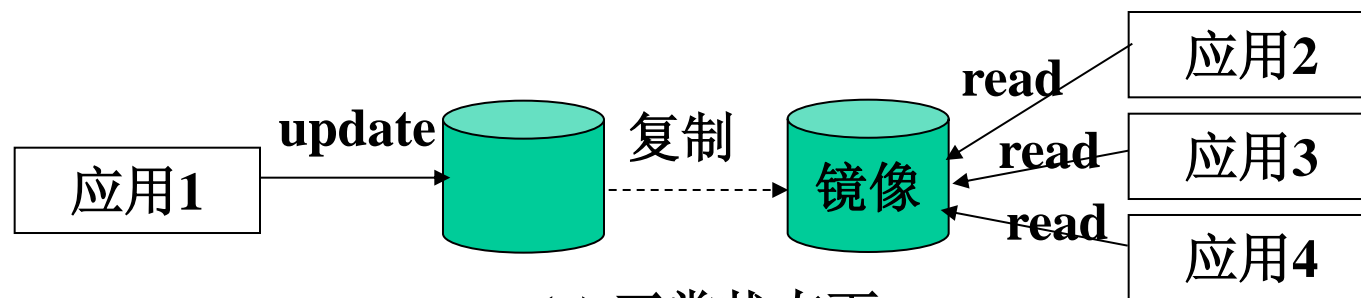
- **定位最近检查点记录：**在重新开始文件中找到最后一个检查点记录在日志文件中的地址。
- **找到检查点时刻运行事务清单：**由该检查点记录得到检查点建立时刻所有正在运行的事务清单ACTIVE-LIST，把ACTIVE-LIST暂时放入UNDO-LIST。
- **确定需要撤消和重做的事务：**从检查点开始正向扫描日志文件，做如下处理，直到文件结束。
 - 如果有新开始的事务 T_i ，把 T_i 暂时放入UNDO-LIST；
 - 如果有提交的事务 T_j ，把 T_j 从UNDO-LIST队列移入到REDO-LIST队列；
- **执行撤消或重做动作：**对UNDO-LIST中的每一个事务执行UNDO操作，对REDO-LIST中的每个事务执行REDO操作。

示例

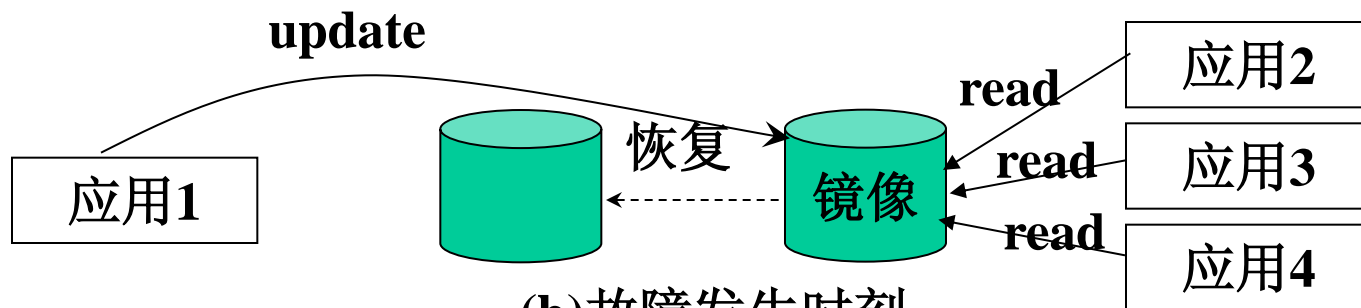


数据库镜像

- 根据DBA的要求，自动把整个DB或其中的关键数据复制到另一个磁盘上，由DBMS自动保证镜像数据库与主数据库的一致性。



(a) 正常状态下



(b) 故障发生时刻



并发控制

- 事务并行执行的优点
 - 一个事务由不同的步骤组成，所涉及的系统资源也不同。这些步骤可以并发执行，以提高系统的吞吐量。
 - 系统中存在着周期不等的各种事务，串行会导致难于预测的时延。如果各个事务所涉及的是数据库的不同部分，采用并发会减少平均响应时间。
- 事务并行执行带来的问题
 - 多个事务同时存取同一数据时，如不加控制就可能会读取或存储不正确的数据，破坏数据库的一致性。

并发控制的必要性

- 例：假设存款余额 $X=1000$ 元，事务甲取走300元，事务乙取走200元，存款余额最终更新后应该是 $X=500$ 元。

时间	事务甲	事务乙
...
t_1	读 X (1000)	
...	...	
t_2		读 X (1000)
...		...
t_3	更新： $X=X-300$ (700)	...
...	...	更新： $X=X-200$ (800)

!错误

并发操作导致的数据不一致性

- 丢失更新(Lost Update)

- 两个事务 T_1 和 T_2 读入同一数据并修改， T_2 提交的结果破坏了 T_1 提交的结果，导致 T_1 的修改被丢失。

T_1	T_2
READ(A) //A=16	
A=A-1	READ(A) //A=16
WRITE(A) //A=15	
	A=A-1
	WRITE(A) //A=15

- “脏”数据的读出(Dirty Read)

- 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为原值， T_2 读到的数据就与数据库中的不一致，则 T_2 读到的数据就为“脏”数据。

T_1	T_2
READ(C) //C=100 C=C*2 WRITE (C)	
ROLLBACK //C=100	READ(C) //C=200

• 不能重复读(Non-Repeatable Read)

- 事务 T_1 读取数据后，事务 T_2 执行更新（修改、插入、删除）操作，使 T_1 无法再现前一次读取的结果。

T_1	T_2
READ(A) READ(B) Sum=A+B	
	READ(B) B=B*2 WRITE(B)
READ(A) READ(B) Sum=A+B	

(a)

T_1	T_2
select * from SC where CNO = C01 and SNO = S01	
	Delete From SC Where CNO =C01 and SNO =S01
select * from SC where CNO = C01 and SNO = S01	

(b)

事务并发操作中可能存在的问题

- 丢失更新(Lost Update)
- “脏”数据的读出(Dirty Read)
- 不能重复读(Non-Repeatable Read)

并发控制基本思想

- 并发控制就是要合理调度并发事务，避免并发事务之间的互相干扰造成数据的不一致性。
- 并发控制的主要方法是采用封锁机制。

并发控制的基本手段——封锁

- **封锁**就是事务T在对某个数据对象如表、记录等操作之前，先向系统发出请求，对其**加锁**，从而对该数据对象有了一定的控制权。
- 基本的封锁有两种类型：
 - 排它锁 (**X锁**，e**X**clusive lock)
 - 共享锁 (**S锁**，Share lock)

-
- **排它锁 (X锁)**：事务T对数据对象R加上X锁，则只允许T读取和修改R，其它事务对R的任何封锁请求都不能成功（因而不能读取和修改R），直至T释放R上的X锁。
 - **共享锁 (S锁)**：事务T对数据对象R加上S锁，则事务T可以读取但不能修改R，其它事务只能对R加S锁（因而可以读取R），而不能对R加X锁（因而不能修改R），直到T释放R上的S锁。

-
- 以相容矩阵表示两种基本锁的控制：

$T_1 \backslash T_2$	X	S
X	N	N
S	N	Y

-
- 运用X锁和S锁这两种基本封锁，可以建立不同的约定，形成不同级别的封锁协议，以保证事务并发执行过程中的数据的一致性。
 - 一级封锁协议——防止丢失修改
 - 二级封锁协议——防止读“脏”数据
 - 三级封锁协议——保证数据可重复读

• 一级封锁协议

- 协议内容：事务T在修改数据R之前必须对其加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)。

T ₁	T ₂
X LOCK A	
...	
A=A-1	X LOCK A
...	等待
COMMIT	.
UNLOCK A	.
	获得

一级封锁协议可以防止丢失修改，并保证事务T是可恢复的。但在一级封锁协议中，不要求读数据操作对数据加锁，因此它不能保证可重复读和不读“脏”数据。

• 二级封锁协议

- 协议内容：一级封锁协议加上事务T在读取数据R之前必须先对其加S锁，读完后即可释放S锁。

T ₁	T ₂
X LOCK C READ(C) //C=100 C=C*2 ... ROLLBACK UNLOCK C ...	S LOCK C 等待 ... READ(C) //C=100 UNLOCK(C) ... COMMIT

二级封锁除了防止丢失更新，还可以进一步防止读“脏”数据。但由于读完后即可释放S锁，所以不能保证可重复读。

• 三级封锁协议

- 协议内容：一级封锁协议加上事务T在读取R之前必须对其加S锁，直到事务结束才释放。

T ₁	T ₂
S LOCK A	
S LOCK B	
...	
READ(A) //A= 50	X LOCK B
READ(B)// B= 100	等待
Sum=A+B //Sum=150	.
...	.

T ₁	T ₂
READ(A)	等待
READ(B)	.
Sum=A+B //Sum=150	.
COMMIT	.
UNLOCK(A)	.
UNLOCK(B)	.
	获得
	WRITE(B)

- 三级封锁协议除了防止丢失修改和读“脏”数据以外，还进一步防止了不可重复读。

封锁协议小结

- 三级协议的主要区别在于何时加锁以及何时放锁。

	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		√		√	√	√	√

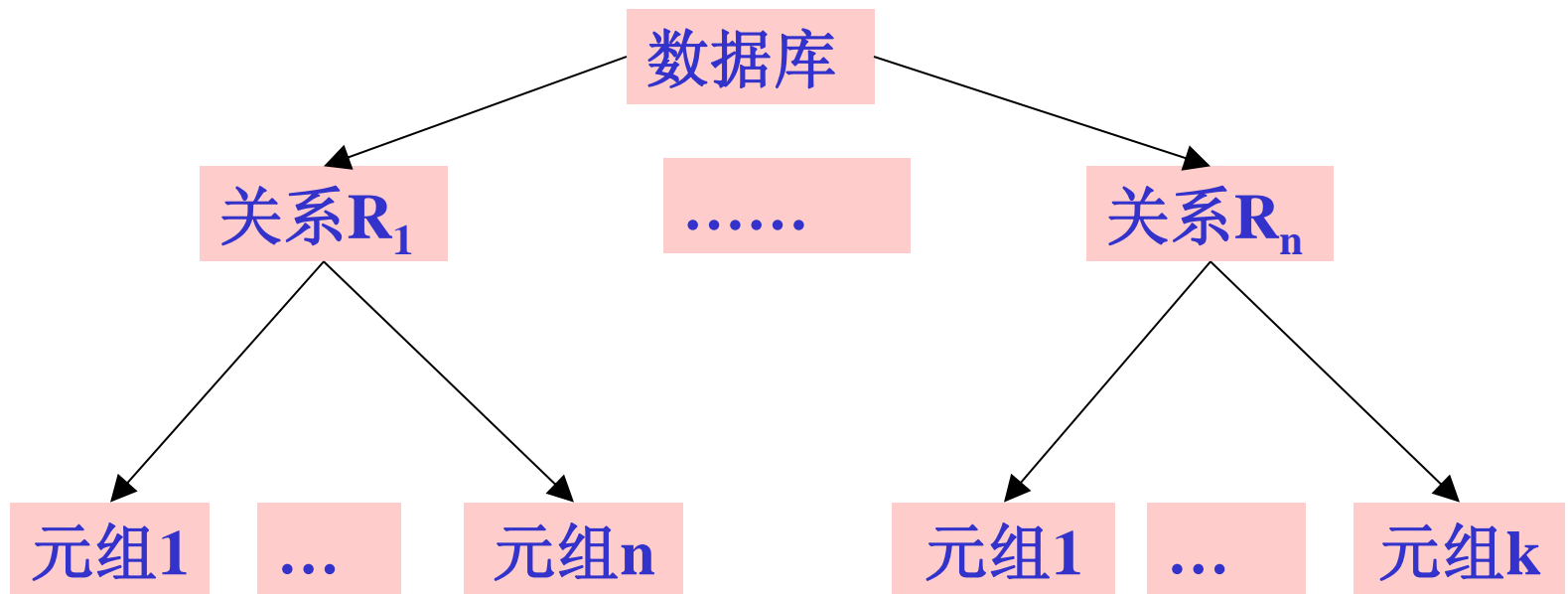
封锁的粒度

- 封锁对象的大小称为封锁粒度。
- 封锁对象：属性值、属性值集合、元组、关系、某索引项、整个索引、整个数据库、物理页、块等。
- 封锁粒度大，则并发度低，封锁机构简单，开销小。
封锁粒度小，则并发度高，封锁机构复杂，开销高。
- 多粒度封锁：在一个系统中同时支持多种封锁粒度供不同的事务选择。选择封锁粒度时应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达到最优效果。

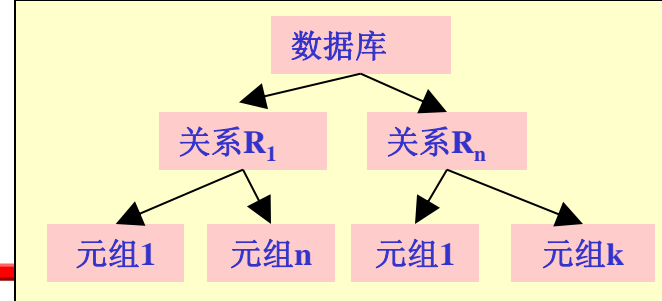
多粒度封锁 (1)

- 多粒度树

- 多粒度树的根结点是整个数据库，表示最大的粒度。叶结点表示最小的粒度。



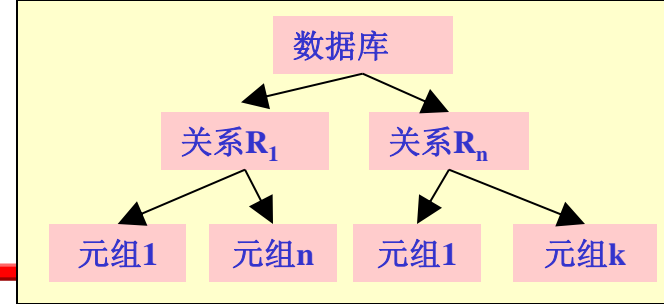
多粒度封锁 (2)



- 多粒度封锁协议

- 多粒度封锁协议允许多粒度树中的每个结点被独立地加锁。对一个结点加锁意味着这个结点的所有后裔结点也被加以同样类型的锁。因此，在多粒度封锁中一个数据对象可能以两种方式封锁，即：
 - 显式封锁是应事务的要求直接加到数据对象上的封锁。
 - 隐式封锁是该数据对象没有独立加锁，是由于其上级结点加锁而使该数据对象加上了锁。
- 在多粒度封锁方法中，显式封锁与隐式封锁的效果是一样的。

多粒度封锁 (3)

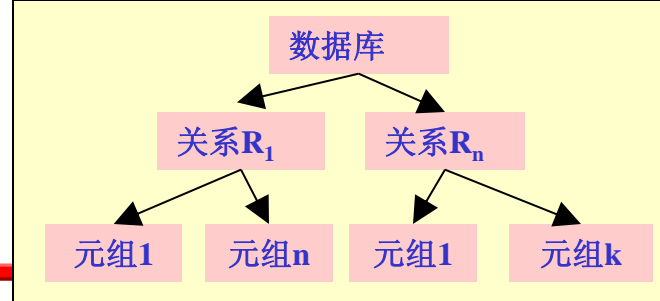


- 多粒度封锁中存在的问题

- 在多粒度封锁方法中，一般对某个数据对象加锁，系统要做如下检查：

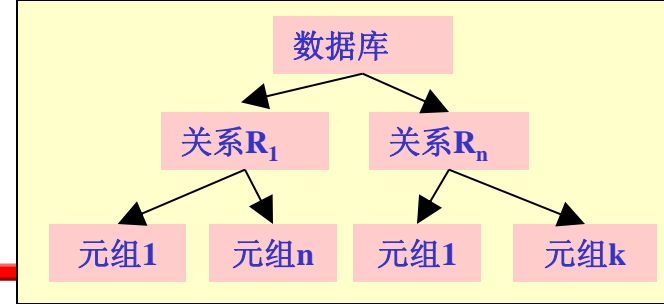
- 是否与该数据对象上的显式封锁冲突（检查对象本身）；
- 是否与该数据对象上的隐式封锁冲突（检查对象的所有上级结点）；
- 是否与该数据对象下级的显式封锁冲突（检查其所有下级结点）。

意向锁 (1)



- **意向锁**的含义是该结点的下层结点正在被加锁
- 意向锁的好处是：在对象加锁时，不再检查下级结点的封锁，只需检查对象和它的上级结点

意向锁 (2)



- 三种常用的意向锁

- 意向共享锁 (Intent Share Lock, 简称IS锁)

- 如果要对一个数据对象加IS锁, 表示它的后裔结点拟 (意向) 加S锁。

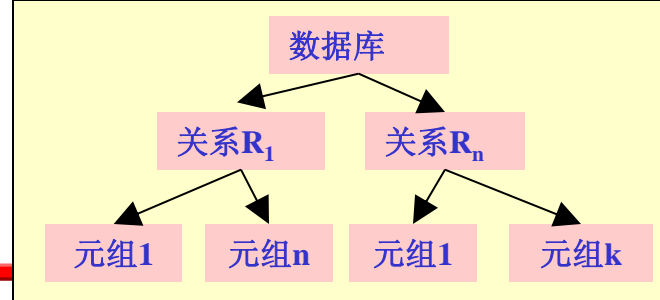
- 意向排它锁 (Intent Exclusive Lock, 简称IX锁)

- 如果要对一个数据对象加IX锁, 表示它的后裔结点拟 (意向) 加X锁。

- 意向共享排它锁 (Share Intent Exclusive Lock, 简称SIX锁)

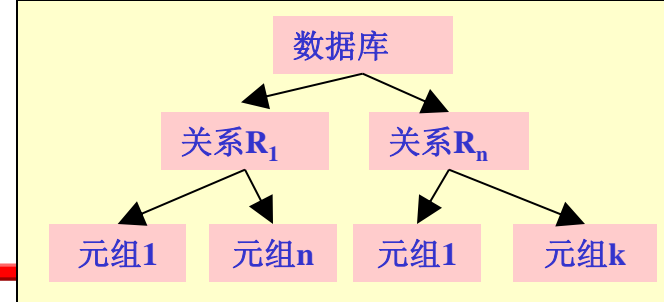
- 如果要对一个数据对象加SIX锁, 表示对它加S锁, 再加IX锁, 即 $SIX=S+IX$ 。

意向锁 (3)



- 具有意向锁的多粒度封锁方法中，任意事务T要对一个数据对象加锁，要先对它的上级对象结点加意向锁，申请封锁按自上而下的次序进行；释放封锁时，应按自下而上的次序进行。

意向锁 (4)

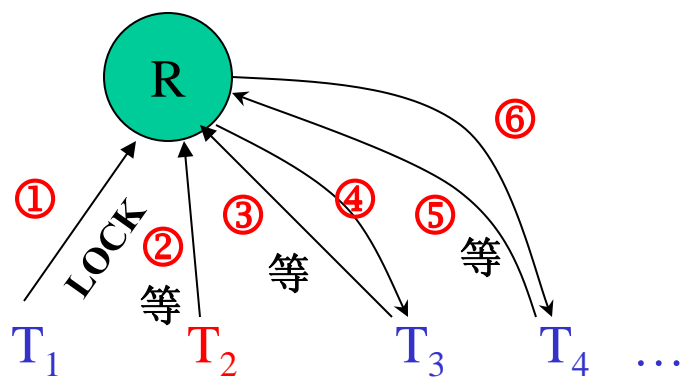


- 意向锁的相容矩阵

$T_1 \backslash T_2$	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

活锁与死锁 (1)

- 活锁



避免活锁的简单方法是采用先来先服务的策略。

T_2 有可能永远等待，这就是活锁

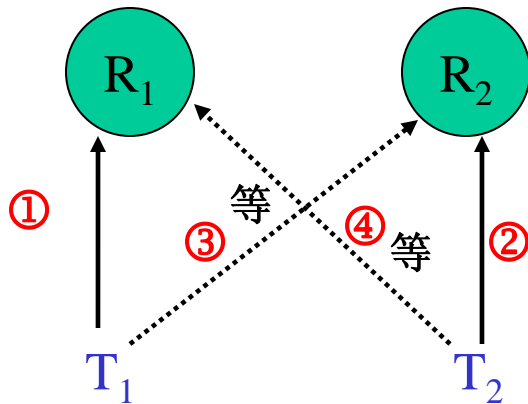
活锁与死锁 (2)

- 死锁

解决死锁的方法

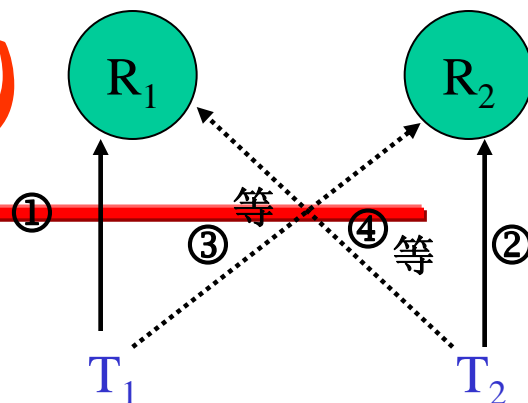
- 预防死锁

- 死锁检测和解除



T_1 , T_2 永远不能结束,
形成死锁。

活锁与死锁 (3)



• 预防死锁

— 一次封锁法

- 要求每个事务必须一次将其所有要使用的数据全部加锁，否则就不能执行。
- 可以有效地防止死锁的发生，但由于需要扩大加锁的范围，因此降低了系统的并发度。

— 顺序封锁法

- 预先对数据对象规定一个封锁顺序，所有的事务都要按照这个顺序实行封锁。
- 可以有效地防止死锁，但由于数据库中数据的不断变化和事务封锁要求的动态提出而实现难度大。

活锁与死锁 (4)

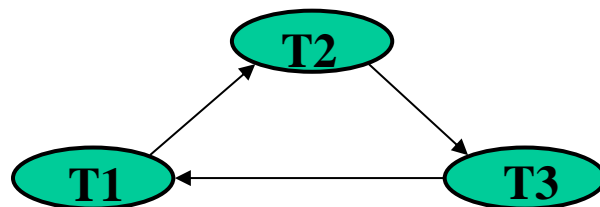
- 死锁的检测

- 超时法

- 如果一个事务的等待时间超过了规定的期限，就认为发生了死锁。

- 等待图法

- 事务等待图是一个有向回路 $G=(T, U)$,其中 T 为结点集合,每个结点表示正在运行的事务, U 为边集,每条边表示事务的等待情况。并发控制子系统周期性的检测事务等待图,如果发现图中存在回路,则表示系统出现死锁。



- 死锁恢复

- 通常采用的方法是选择一个处理死锁代价最小的事务,将其撤销,释放此事务持有的所有锁,使其他事务得以继续运行下去。对于所撤销的事务所作的操作必须加以恢复。

事务的调度

- **N个事务的一个调度S**是N个事务所有操作的一个序列，表示这些操作的执行顺序。并且这个序列满足：对于每个事务T，如果操作i在事务T中先于操作k执行，则在S中操作i也必须先于操作k执行。

串行调度与并行调度

T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2
Slock B			Slock A	Slock B		Slock B	
$Y = R(B) = 2$			$X = R(A) = 2$	$Y = R(B) = 2$		$Y = R(B) = 2$	
Unlock B			Unlock A		Slock A	Unlock B	
Xlock A			Xlock B		$X = R(A) = 2$	Xlock A	
$A = Y + 1 = 3$			$B = X + 1 = 3$	Unlock B			Slock A
$W(A)$			$W(B)$	Xlock A	Unlock A	$A = Y + 1 = 3$	等待
Unlock A			Unlock B	$A = Y + 1 = 3$		$W(A)$	等待
	Slock A	Slock B		$W(A)$		Unlock A	等待
	$X = R(A) = 3$	$Y = R(B) = 3$			Xlock B		$X = R(A) = 3$
	Unlock A	Unlock B			$B = X + 1 = 3$		Unlock A
	Xlock B	Xlock A			$W(B)$		Xlock B
	$B = X + 1 = 4$	$A = Y + 1 = 4$					$B = X + 1 = 4$
	$W(B)$	$W(A)$		Unlock A			$W(B)$
	Unlock B	Unlock A			Unlock B		Unlock B

(a) 串行调度

(b) 串行调度

(c) 不可串行化的调度

(d) 可串行化的调度

图 11.7 并发事务的不同调度

事务并发调度的正确性

- 对并发事务中操作的调度是随机的，而不同的调度可能产生不同的结果，但调度要保证事务执行的正确性
- 事务执行正确性的含义
 - 一个事务正常的或者预想的结果是没有其它并行事务干扰时得到的结果。因此一组事务的串行调度策略一定是正确的调度策略。
 - 虽然不同的串行顺序的结果会不同，由于各种结果都将保持数据库数据的一致性，所以都是正确的。

并发调度的可串行性

- 可串行化

- 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，我们称这种调度策略为可串行化调度。

- 可串行性是并行事务正确性的准则

- 一个给定的并发调度，当且仅当它是可串行化的，才认为是正确调度。

可串行化调度的判定

- 可串行化调度判定的充分条件
 - 一个调度 Sc 在保证冲突操作次序不变的情况下, 可以通过交换两个事务不冲突操作的次序, 得到另一个串行调度 Sc' , 则调度 Sc 为冲突可串行化调度。
 - 冲突操作

冲突操作是不同事务对同一数据的读-写操作以及写-写操作。表示为:

事务 T_i 读 x , T_j 写 x —— $R_i(x)$ 与 $W_j(x)$

事务 T_i 写 x , T_j 写 x —— $W_i(x)$ 与 $W_j(x)$
 - 不同事务的冲突操作与同一个事务的两个操作不能交换。
 - 一个冲突可串行化调度, 一定是可串行化调度。

可串行化调度的判定

- 示例

调度 $Sc1 = R1(A)W1(A)R2(A)W2(A)R1(B)W1(B)R2(B)W2(B)$, 经交换操作得到 $Sc2 = R1(A)W1(A)R1(B)W1(B)R2(A)W2(A)R2(B)W2(B)$
 $= T1, T2$

所以 $Sc1$ 是可串行化调度。

并发调度的可串行性

- 两段锁协议(Two-phase Locking)可保证并行事务的可串行性
 - 两段锁协议的内容
 - ①在对任何数据进行读、写操作之前，事务首先要获得对该数据的封锁。
 - ②在释放一个封锁之后，事务不再获得任何其它封锁。
 - 两段锁协议的含义
 - 事务分为两个阶段，第一个阶段是获得封锁，也称为扩展阶段；第二个阶段是释放封锁，也称为收缩阶段。
- 如：T₁的封锁顺序是：
- S LOCK A ... S LOCK B...X LOCK C...UNLOCK A...UNLOCK C
- T₂的封锁顺序是：
- S LOCK A ... UNLOCK A... S LOCK B... X LOCK C... UNLOCK C... UNLOCK B

并发调度的可串行性

- **定理：**若所有事务均遵从两段锁协议，则这些事务的所有并发调度都是可串行化的。按照这个定理，所有遵守两段锁协议的事务，其并行执行的结果一定是正确的。
- **注意的问题**
 - 事务遵守两段锁协议是可串行化调度的充分条件而不是必要条件。
 - 两段锁协议并不要求事务在执行任何数据库读、写操作之前就一次申请全部封锁，因此遵守两段锁的事务仍可能发生死锁。



第八章 数据库保护

- 数据库安全性控制
- 数据库完整性控制

数据库安全性控制

- 数据库安全性含义
- 数据库安全性控制
 - 用户标识与鉴别
 - 自主存取控制
 - 强制存取方法
 - 其它方法
- 安全数据库体系结构示例
- 可信计算机系统评测标准

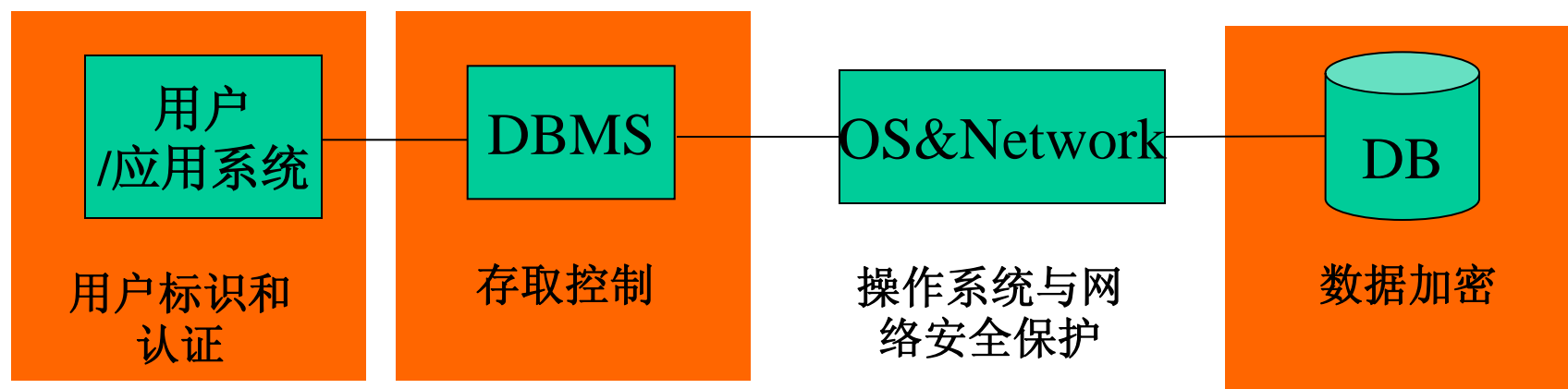
数据库系统的安全性

- 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漏、更改和破坏。它包括两个方面的含义：
 - 向授权用户提供可靠的信息服务。
 - 拒绝非授权的对数据的存取访问请求，保证数据库管理下的数据的可用性、完整性和一致性，进而保护数据库所有者和使用者的合法权益。



数据库安全性控制

- 包含数据库系统的计算机系统安全模型:



- 用户标识与认证
- 访问控制（存取控制）
- 加密技术
- 审计
- 推理的控制
- 隐通道分析技术

用户标识与鉴别

- 用户标识和认证是系统提供的最外层安全保护措施。
 - 标识是指系统采用一定的方式标识其用户或应用程序的名字或身份。
 - 认证是指系统在用户或应用程序登录时判断其是否为合法的授权用户。
 - 常用的方法是采用用户名和口令。

存取控制

- **存取控制**确保合法用户按照指定的权限使用DBMS和访问数据，而非法用户或不具有相关权限的用户则不能。
- 存取控制机制主要包括两个部分：
 - **用户权限定义**：将用户权限记录到数据字典中，形成安全规则或授权规则。
 - **合法权限检查**，每当用户发出数据库操作请求后，DBMS根据数据字典中的安全规则进行合法权限检查，决定是否接受用户的操作请求。
 - **用户权限定义和合法权限检查机制一起组成了DBMS的安全子系统。**

存取控制方法分类

- **自主存取控制**(discretionary access control, 简称DAC)
 - 用户对于不同的数据对象拥有不同的存取权限, 不同的用户对同一对象也有不同的权限, 而且用户还可以将其拥有的权限转授给其他用户。
- **强制存取控制**(mandatory access control, 简称MAC)
 - 每一个数据对象被标以一定的密级, 每一个用户也被授予某一个级别的许可证。对于任一个对象, 只有具有合法许可证的用户才可以存取。

自主存取控制

- 系统根据预先定义好的用户权限进行存取控制。用户权限是指不同的用户对于不同的数据对象允许执行的操作权限，由数据对象和操作类型两个要素组成。

	数据对象	操作类型
模式	模式	建立、修改、检索
	外模式	建立、修改、检索
	内模式	建立、修改、检索
数据	表	查找、插入、删除、修改
	属性列	查找、插入、删除、修改

- 对于用户存取权限的定义称为授权。在授权中应指明：用户名，数据对象名，允许的操作类型。

SQL的数据安全性控制

- 在SQL中可以授予用户两类权限：
 - 用户级权限
 - 是数据库管理员为每个用户授予的特定权限。这种权限与整个数据库相关，与数据库中具体的关系无关。这种权限是对用户使用整个数据库的权限的限定。
 - 关系级权限
 - 是数据库管理员或数据库对象的拥有者为用户授予的与关系或视图有关的权限。这种权限是对用户使用关系和视图的权限的限定。

角色与用户组

- 为了管理数据库特权的方便，数据库还支持角色和用户组的概念。
 - 角色是一组权限的集合，可以把它授予用户或其他角色。当把某个角色授予用户（或角色）或从用户（或角色）处收回时，就同时授予或收回了该角色代表的全部权限。
 - 用户组是一组具有相同特性用户的集合。在授权或收回权限时，可以以用户组为单位进行。

用户级权限与角色的授予与收回

- 在SQL语言中，通过**Grant语句**授予用户用户级权限或角色，其语法格式

Grant <用户级权限>|<角色> [{,<用户级权限>|<角色>}]
To <用户名>|<角色>|public [{,<用户名>|<角色>}]
[With Grant Option]

数据库中的
全部用户

允许被授权的用户将指
定的用户级权限或角色
授予其他用户

- 当要取消一个用户或角色的权限时，可以使用**REVOKE语句**将其收回：

Revoke <用户级权限>|<角色> [{,<用户级权限>|<角色>}]
From <用户名>|<角色>|public [{,<用户名>|<角色>}]

示例

- 为用户授予用户级权限
 - Grant Create Session to SCOTT;
- 将权限授予角色
 - Grant Create table to Student_role;
- 取消用户SCOTT的Create Table权限。
 - Revoke Create Table From SCOTT;

关系级权限的授予与收回

- DBA和数据库对象所有者将这些数据库对象上的部分或全部权限授予其他用户。语法格式为：

Grant ALL|<权限> [{,<权限>}]

On <表名>|<视图名> [{,<表名>|<视图名>}]

To {<用户> [{,<用户>}]|public}

[With Grant Option]

- 回收权限

- Revoke ALL|<表级权限> [{,<表级权限>}]

- On <表名>|<视图名> [{,<表名>|<视图名>}]

- From {<用户> [{,<用户>}]|PUBLIC}

- 收回权限时，若该用户已将权限授予其它用户，则也一并收回。

示例

- 授予用户Liming在Student表上的Select和Insert权限。
 - Grant Select , Update On Student To Liming With Grant Option;
- 将Student表上的全部权限授予全体用户。
 - Grant ALL On Student To PUBLIC;
- 收回Liming对Student表的全部权限
 - Revoke ALL On Student From Liming;

强制存取方法

- 在MAC中，DBMS所管理的**全部实体**被分为**主体**和**客体**两类。
 - **主体**是系统中的活动实体，既包括DBMS所管理的**实际用户**，也包括**代表用户的各进程**。
 - **客体**是系统中的被动实体，是受主体操纵的，包括**文件、基本表、索引、视图等**。
- 对于主体和客体，DBMS为他们每个实例指定一个**敏感度标记(Label)**。敏感度标记被分为若干级别，如**绝密、机密、秘密、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**。

强制存取方法

- MAC机制通过对比主体的Label和客体的Label，最终确定主体是否能够存取客体。
- 当某一主体以某一许可证级别注册入系统时，系统要求他对任何客体的存取必须遵循如下规则：
 - 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；
 - 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。

数据安全性控制的其它方法 (1)

- 视图机制

- 为不同的用户定义不同的视图，可以将用户对数据的访问限制在一定的范围内。
- 例：限制王平只能检索Student表中计算机系学生的学号和姓名。

Create View CS_Student

As Select Sno, Sname From Student

Where Sdept = 'CS';

Grant Select On CS_Student To Wangping;

数据安全性控制的其它方法 (2)

- 审计

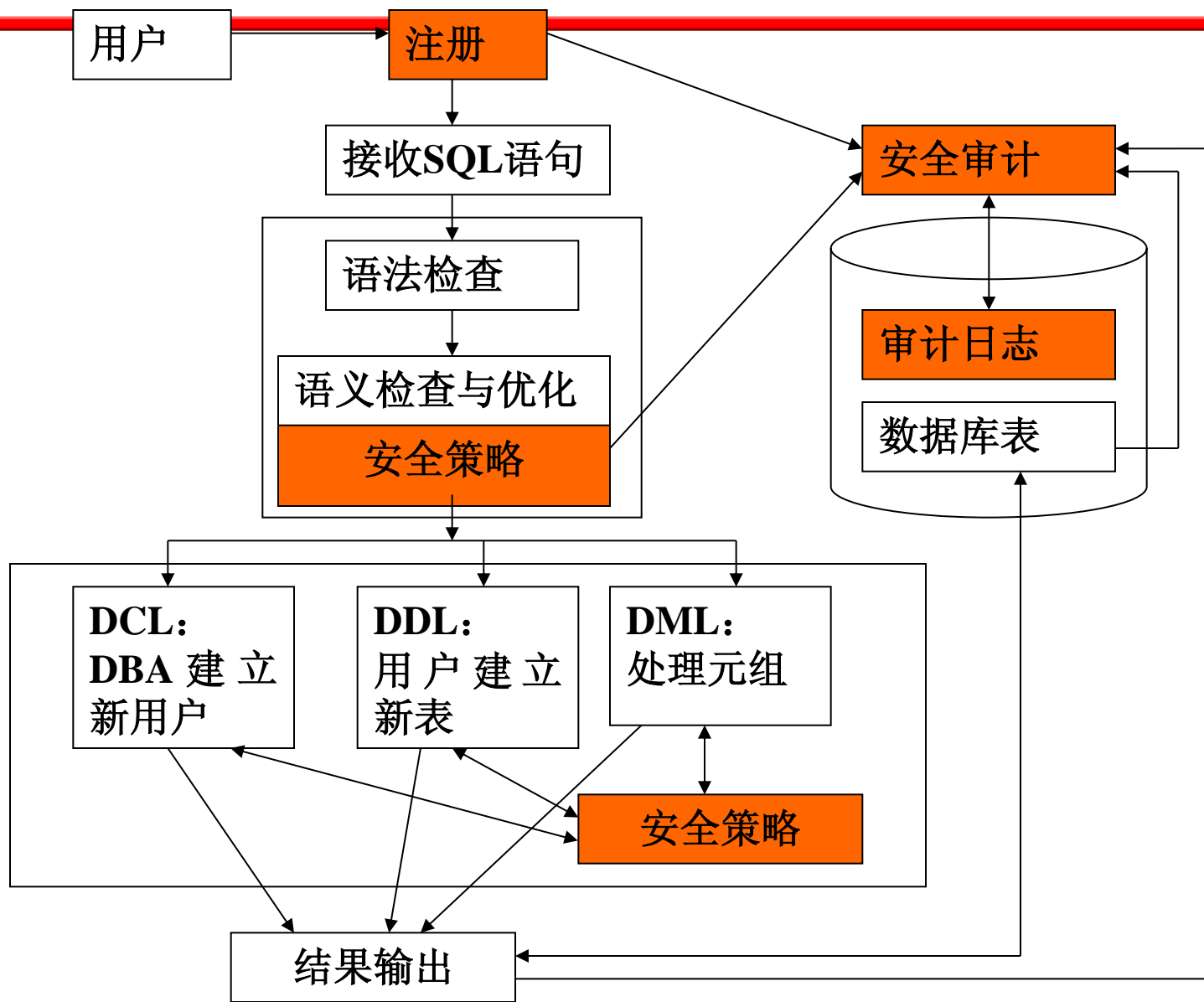
- 把用户对数据库的所有操作都自动记录下来放入审计日志中。DBA可以利用审计跟踪的信息，重现导致数据库现有状况的一系列事件，找出非法存取数据的人、时间和内容等；

- 数据加密

- 防止数据库中数据在存储和传输中失密。加密的基本思想是根据一定的算法将原始数据（明文）变换为不可识别的格式（密文），从而使不知道解密算法的人无法获知数据的内容。



安全数据库体系结构示例



可信计算机系统评测标准

- TCSEC (Trusted Computer System Evaluation Criteria)
 - 1985年, 美国国防部制定了可信计算机评估标准
TCSEC
- TDI/TCSEC
 - 1991年4月, 美国国家计算机安全中心NCSC发布《可信计算机系统评估标准关于数据库系统的解释TDI (Trusted Database Interpretation) 》, 将TCSEC扩展到数据库管理系统;
- TDI与TCSEC从安全策略、责任、保证、文档四个方面描述了安全级别划分的指标。

可信计算机系统评测标准

安全性等级					主要特征
1	D	最低保护等级	D		非安全保护
2	C	自主保护等级	C1	自主安全保护	自主存取控制、审计功能
			C2	可控存取保护	比C1级更强的 自主存取控制、审计功能
3	B	强制保护等级	B1	标记安全保护	强制存取控制，敏感度标记
			B2	可结构化保护	形式化模型，隐蔽通道约束
			B3	安全区域保护	安全内核，高抗渗透能力
4	A	验证保护等级	A1	可验证保护	形式化安全验证，隐蔽通道分析
			超A1		



数据完整性控制

- 数据完整性含义
- 完整性约束条件
- 完整性控制

数据完整性控制

- 数据完整性是指数据的正确性和相容性。
 - 正确性是指数据应具有合法的类型，并在有效的取值范围之内。
 - 相容性是指表示同一个事实的两个数据应该相同。
- 数据库能否保持完整性关系到数据库系统是否能够真实的反映现实世界，因此维护数据库的完整性十分重要。

数据完整性与数据安全性

- **数据完整性控制**是为了防止数据库中存在不符合语义的数据，防止错误信息的输入和输出；
- **数据安全性控制**是保护数据库防止恶意的破坏和非法存取；
- 安全性防范的是非法用户和非法操作，完整性措施的防范对象是不合语义的数据。



完整性约束条件

- 施加在数据库数据之上的语义约束条件称为数据库完整性约束条件。数据库系统依据完整性约束条件进行完整性检查。
- 完整性约束条件作用的对象可以是列、元组、关系三种。
 - 列约束主要是列的类型、取值范围、精度等约束条件；
 - 元组约束是元组中各个字段间联系的约束；
 - 关系约束是若干元组间、关系之间的联系的约束。

完整性约束条件分类

- 完整性约束可分为静态约束和动态约束。
 - 静态约束是指数据库在每一确定状态数据对象所应满足的约束条件，它是反映数据库状态合理性的约束。
 - 动态约束是指数据库从一种状态转变为另一种状态时，新、旧值之间所应满足的约束条件，它是反映数据库状态变迁的约束。

静态约束

- **静态列级约束**是对一个列的取值域の説明，包括对数据类型（包括数据类型、长度、单位、精度等）、数据格式、取值范围或取值集合、空值等的约束。
- **静态元组约束**规定了组成一个元组的各个列之间的约束关系。
- **静态关系约束**规定了一个关系的若干元组或者若干关系之间常常存在的各种联系或约束。包括：实体完整性约束、参照完整性约束、函数依赖、统计约束等。

动态约束

- **动态列级约束**是修改列定义或列值时应满足的约束条件;
- **动态元组约束**指修改元组值时元组中各个字段间需要满足的约束;
- **动态关系约束**是加在关系变化前后状态上的限制条件。



完整性控制

- 数据库完整性控制应包括三个方面的功能：
 - **定义功能**，提供定义完整性约束条件的机制。
 - **检查功能**，检查用户发出的操作请求是否违背了完整性约束条件。
 - **违约响应**，若违背了完整性约束条件，则采取一定措施来保证数据的完整性。

完整性检查的时机

- 完整性约束条件按照完整性检查的时机分为立即执行约束和延迟执行约束。
 - 立即执行约束是指在执行用户事务的过程中，在一条语句执行完后立即进行完整性约束的检查。若违背了完整性约束，系统将拒绝该操作。
 - 延迟执行约束是指在整个用户事务执行完毕后，再进行完整性约束的检查，结果正确方能提交。否则系统将拒绝整个事务。

完整性规则的表示

- 一条完整性规则可以用一个五元组 (D, O, A, C, P) 来描述，其中：
 - D (Data) 约束所作用的数据对象
 - O (Operation) 触发完整性检查的数据库操作。
 - A (Assertion) 数据对象必须满足的断言或语义约束。
 - C (Condition) 选择 A 作用的数据对象值的谓词。
 - P (Procedure) 违反完整性规则时触发的过程。

示例

- 若有表TEACHER(编号, 姓名, 职称, 工资, ...), 表达: 教授的工资不得低于5000元。
 - D : 工资
 - O : 插入或修改
 - A : 工资 \geq 5000
 - C : 职称='教授'
 - P : 拒绝执行该操作

触发器

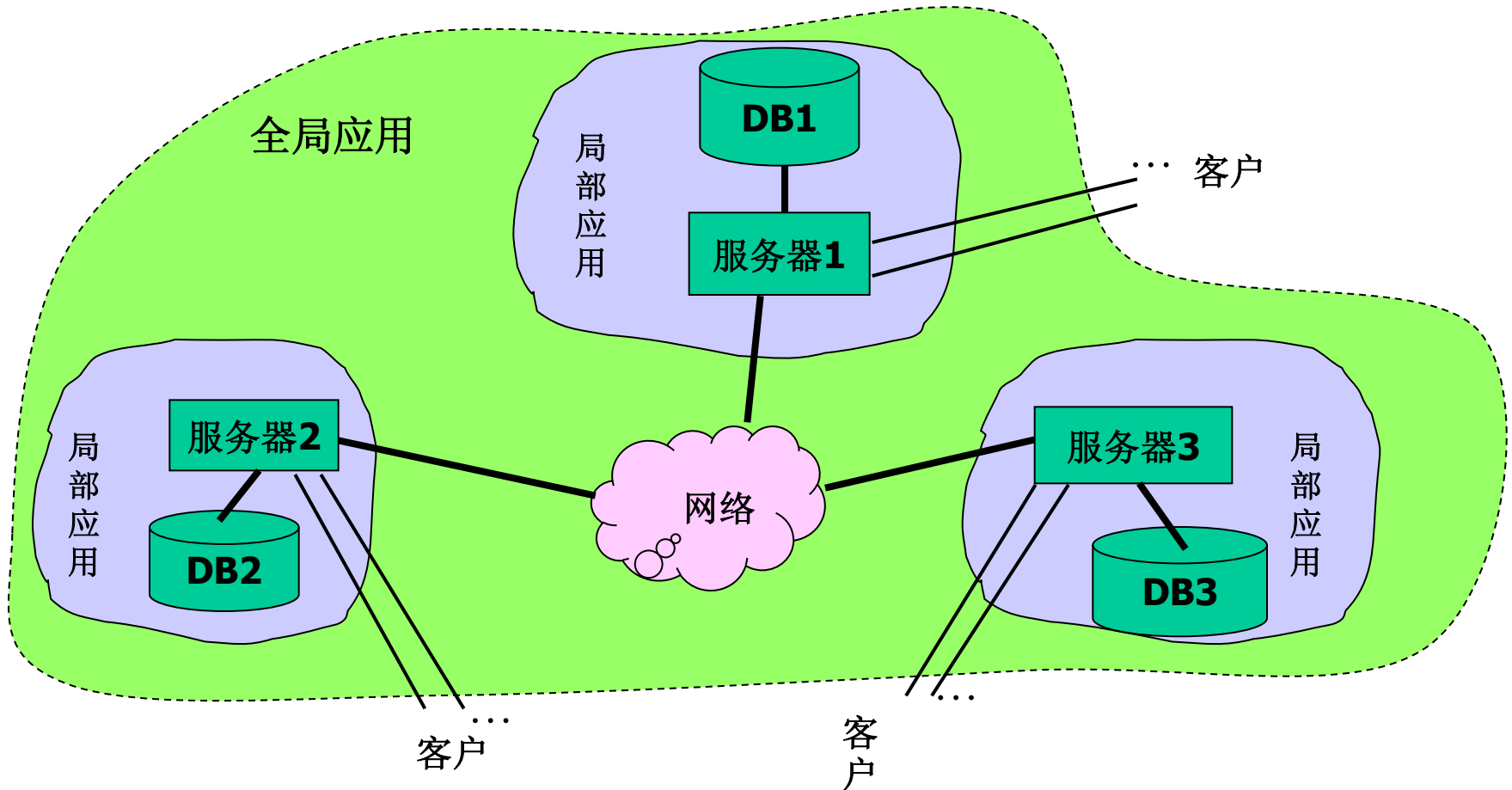
- 触发器 (Trigger) 是用户定义在关系上的一类由事件驱动的特殊过程
- 对于用户对表的更新操作，系统自动激活相应触发器，执行完整性控制
- 定义触发器
 - CREATE TRIGGER <触发器名称>
|BEFORE|AFTER| <触发器事件> ON <表名>
|FOR EACH |ROW|STATEMENT|
[WHEN <触发条件>]
<触发动作体>



第九章 分布式数据库系统

- 分布式数据库系统基本概念
 - 什么是分布式数据库系统
 - 分布式数据库系统的特点
- 分布式数据库系统体系结构
- 分布式数据库系统主要技术

什么是分布式数据库系统

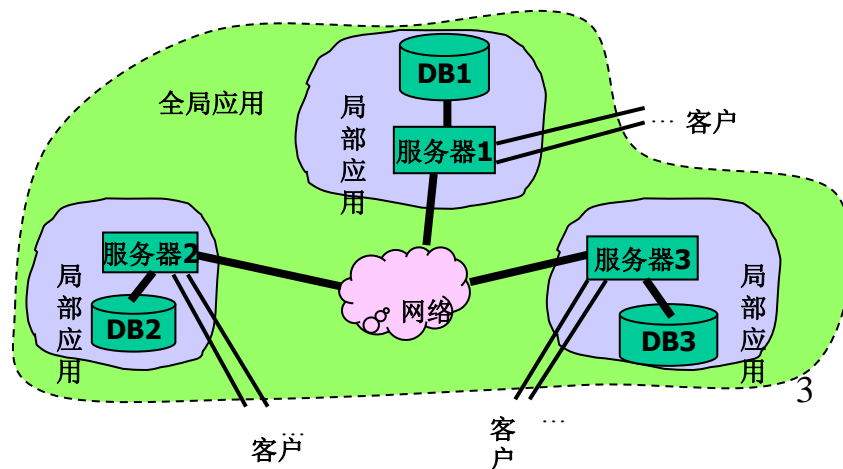


- **分布性**—数据分布存储在网络的各个节点上。
- **逻辑上的整体性**—数据被一种机制联系在一起，构成一个有机整体。

什么是分布式数据库系统

- 分布式数据库定义

- 分布式数据库是由一组分布在计算机网络的不同结点上的数据组成，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支持全局应用。

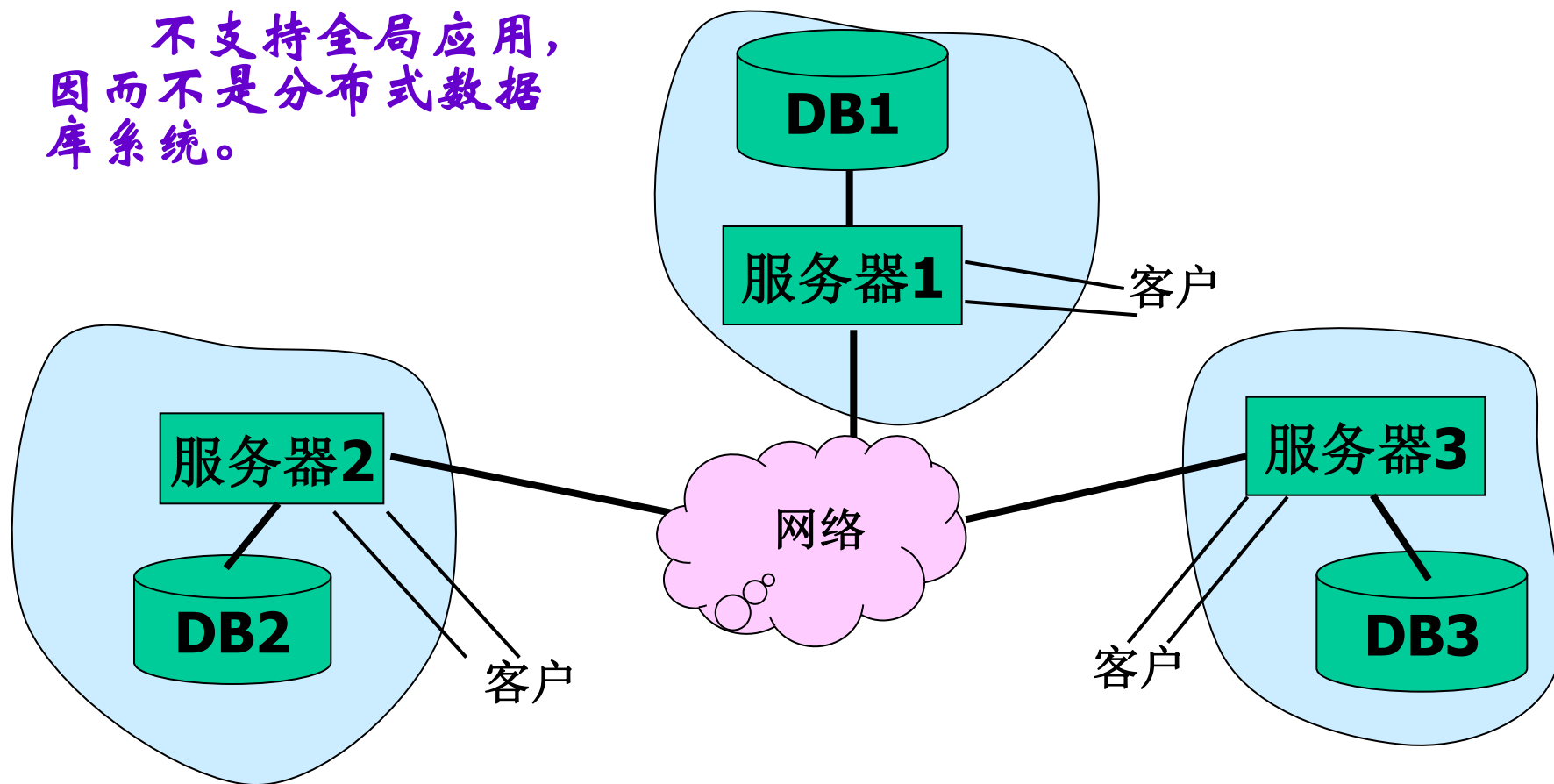


什么是分布式数据库系统

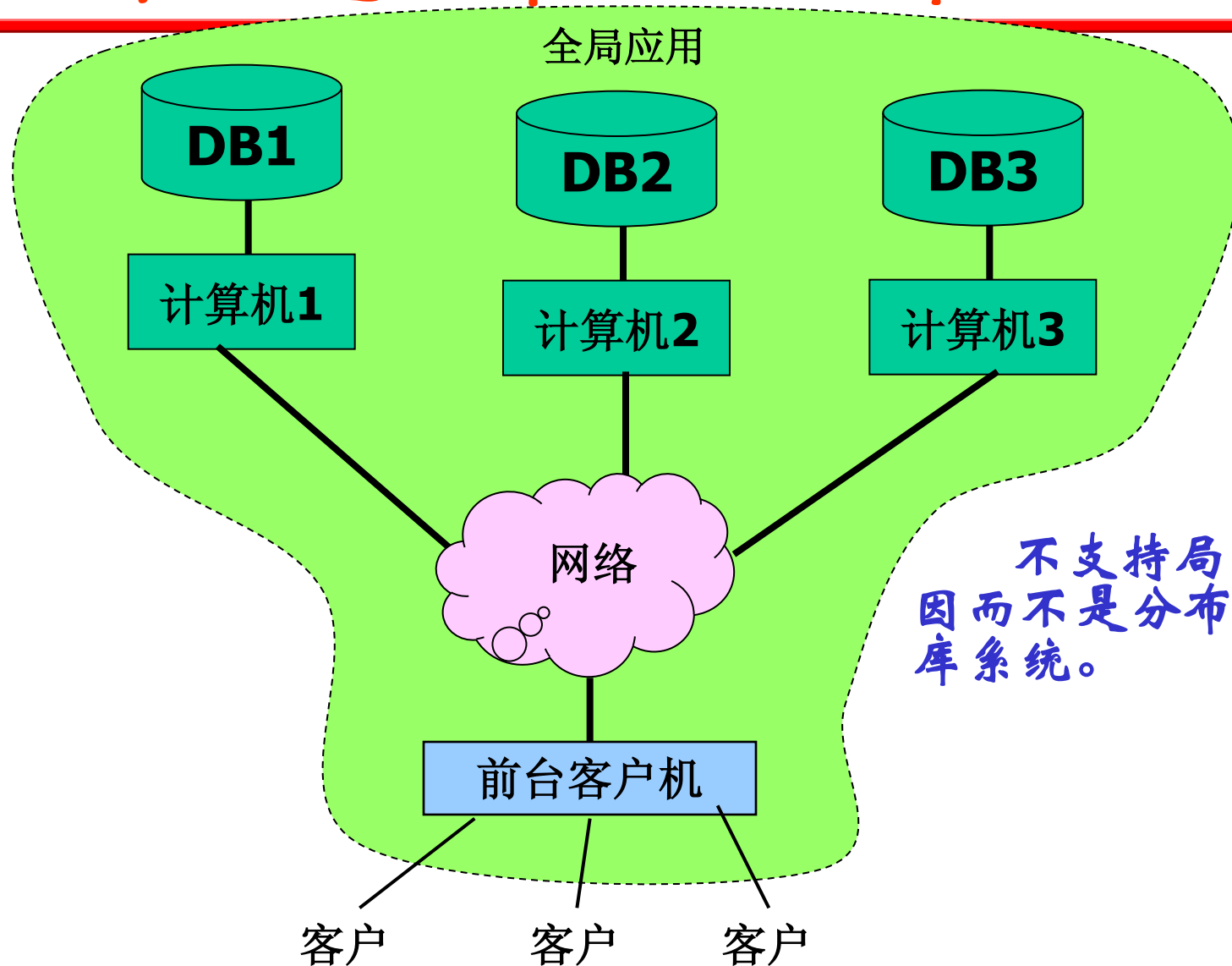
- 分布式数据库以“数据分布”为前提，强调场地自治性（局部应用）以及自治场地之间的协作性（全局应用），两者缺一不可。
 - 场地自治性：每个场地有自己的数据库、一组终端、运行局部DBMS，是独立的DBS，具有高度自治性。
 - 自治场地之间的协作性：各结点组成整体。整体性的含义是，从用户角度看，分布式数据库系统逻辑上如同一个集中式数据库一样，用户可以在任何场地执行全局应用。

什么是分布式数据库系统

不支持全局应用，
因而不是分布式数据库系统。



什么是分布式数据库系统

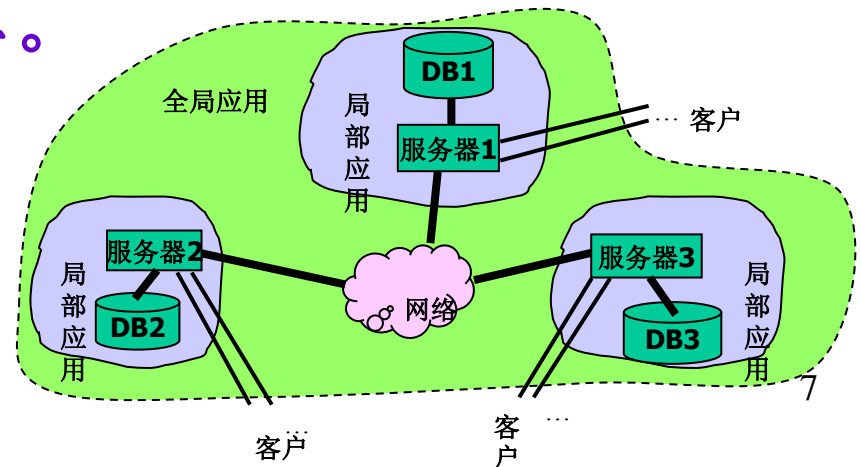


分布式数据库系统的特点 (1)

- 分布式数据库系统是在集中式数据库系统技术的基础上发展起来的，它具有自己独特的特征和性质。

(1) 数据独立性

- 数据的逻辑独立性和物理独立性
- 数据的**分布独立性**（也称**分布透明性**）：数据的逻辑分片、数据物理位置分布的细节、重复副本（冗余数据）一致性问题、局部结点上的数据模型等与用户程序无关。



分布式数据库系统的特点 (2)

(2) 集中与自治相结合的控制机构

- 数据的共享有两个层次：
 - 一是**局部共享**，即在局部数据库中存储局部结点各用户的共享数据；
 - 二是**全局共享**。即在分布式数据库系统的各个结点也存储供其他结点的用户共享的数据，支持系统的全局应用。
- 分布式数据库系统常常采用集中和自治相结合的控制机构。
 - 各局部的DBMS可以独立的管理局部的数据库，具有**自治功能**。
 - 系统又设有**集中控制机构**，协调各局部DBMS的工作，执行全局应用。

分布式数据库系统的特点 (3)

(3) 适当增加数据冗余

- 在分布式数据库系统中适当的增加了冗余数据，在不同的结点存储同一数据的多个副本：
 - 提高系统的可靠性、可用性，当某一结点出现故障时，系统可以对另一结点的相同副本进行操作，不会因为一处故障而造成整个系统的瘫痪；
 - 提高系统性能，系统可以选择用户最近的数据副本来进行操作，减少通信代价，改善整个系统的性能。
- 不利于更新，增加了系统维护代价。

分布式数据库系统的特点 (4)

(4) 全局的一致性、可串行性和可恢复性

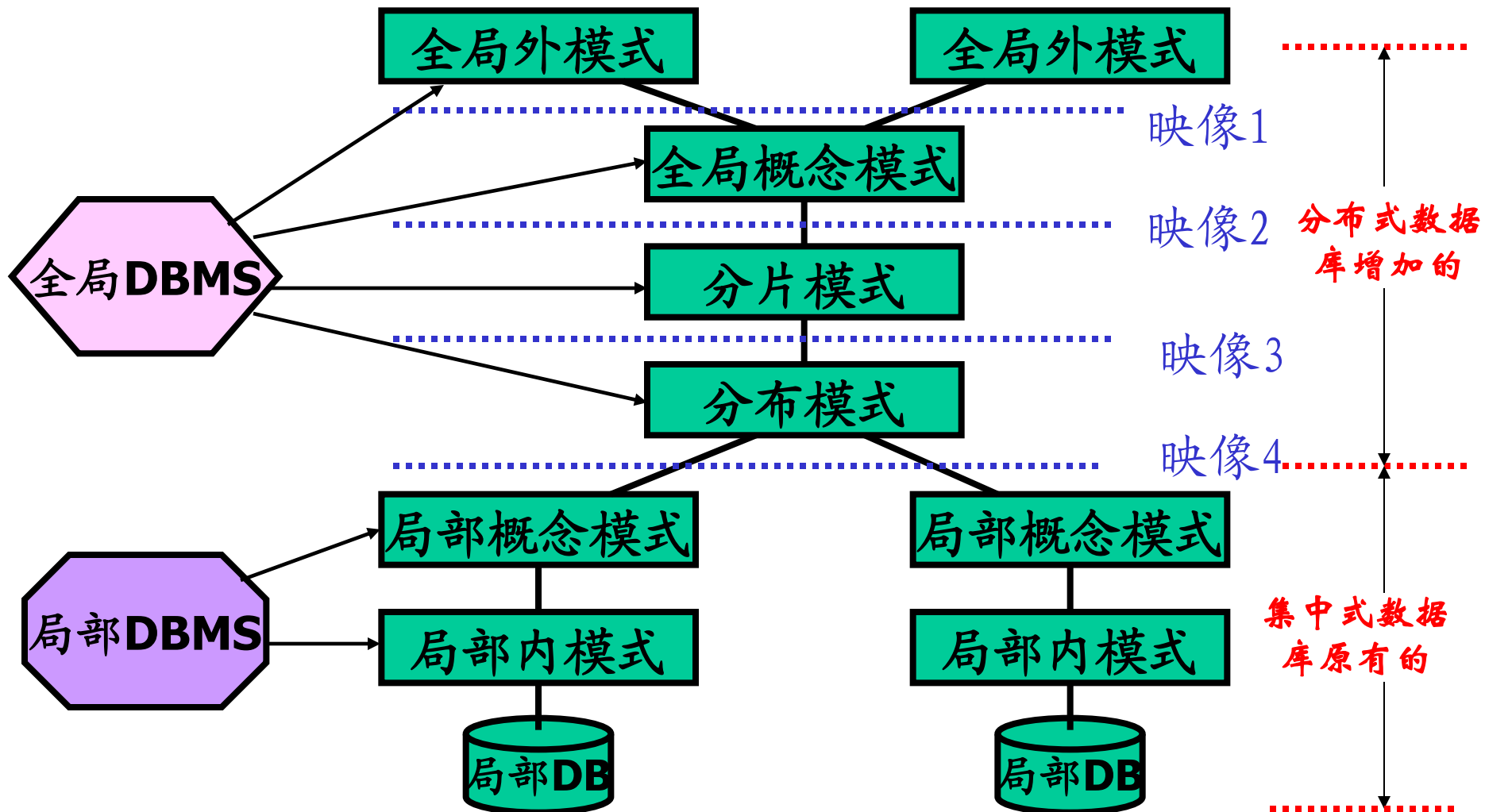
- 分布式数据库系统除了各局部数据库应满足集中式数据库的一致性、可串行性和可恢复性以外，还应保证数据库的**全局一致性、并行操作的可串行性和系统的全局可恢复性**。



分布式数据库系统的体系结构

- 分布式数据库系统的模式结构
- 数据分片
- 分布透明性
- 分布式数据库管理系统DDBMS

分布式数据库系统的模式结构



分布式数据库系统的模式结构

- 全局外模式

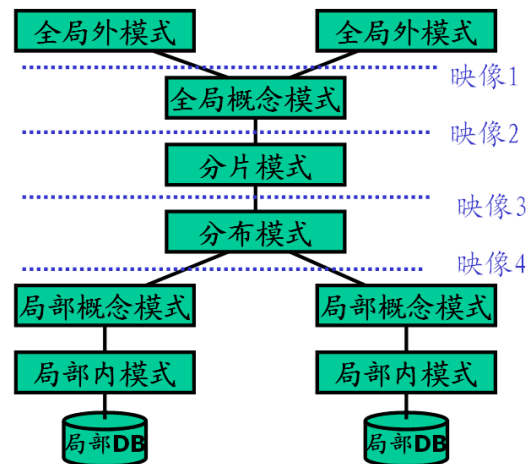
全局应用的用户视图，是全局概念模式的子集。

- 全局概念模式

定义分布式数据库中数据的整体逻辑结构，使得数据如同没有分布一样。

- 分片模式及全局概念模式/分片模式映像（映像2）

每一个全局关系可以分为若干互不相交的部分，每一部分称为一个片段。分片模式及映像2定义片段以及全局关系到片段的映像。这种映像是一对多的。



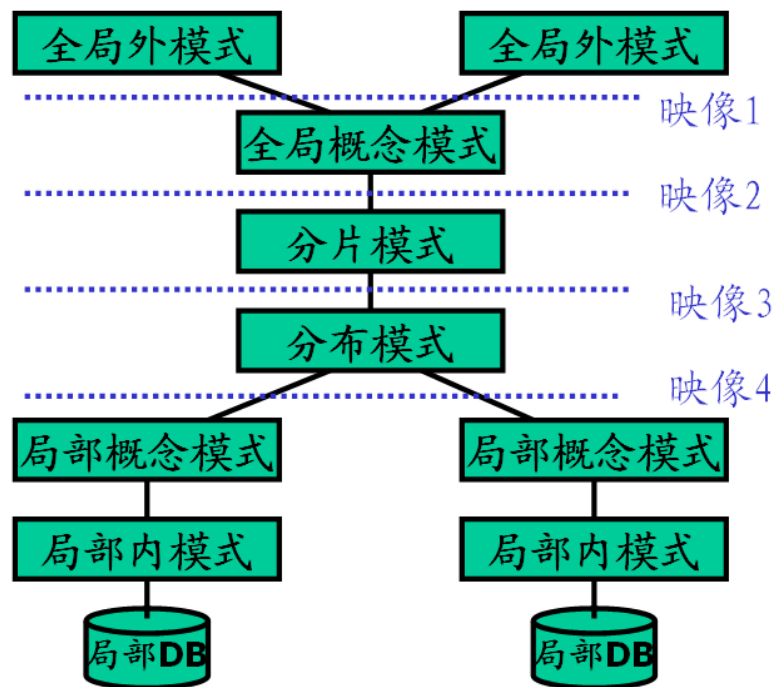
分布式数据库系统的模式结构

• 分布模式及分片模式/分布模式映象 (映象3)

定义片段的存放结点及片段到节点的映象。分布模式的映象类型确定了分布式数据库是冗余的还是非冗余的。

• 分布模式/局部数据库概念模式映象 (映象4)

该映象把存储在局部场地的全局关系或全局关系的片段映象为各局部概念模式。



数据分片

- 分片的方式有多种：
 - 水平分片，垂直分片——两种基本的分片方式
 - 混合分片，导出分片——较复杂的分片方式

数据分片方式

- 水平分片

- 将关系依照一定条件按行分为不相交的若干子集，每个子集称为一个水平片段。

- 垂直分片

- 将关系按列分为若干属性子集，每个子集称为一个垂直片段。垂直分片的片段通过连接的方法恢复原关系。因此垂直分片的诸片段通常都包含关系的码。

- 导出分片

- 导出水平分片，分片的条件不是关系本身属性条件，而是其它关系的属性条件。如SC (SNO, CNO, G) 按S关系中学生所在的系分片。

- 混合分片

- 指按上述三种分片方式得到的片段，继续按另一种方式分片。

数据分片的约束

无论哪种分片方式，都应满足以下条件：

- 完全性

- 一个全局关系中的数据必须完全划分为若干片段，不允许某些数据属于全局关系但不属于任何片段。

- 不相交性

- 不允许一个全局关系的某些数据既属于该全局关系的某一个片段，又属于另一个片段（垂直分片的码属性除外）。

- 可重构性

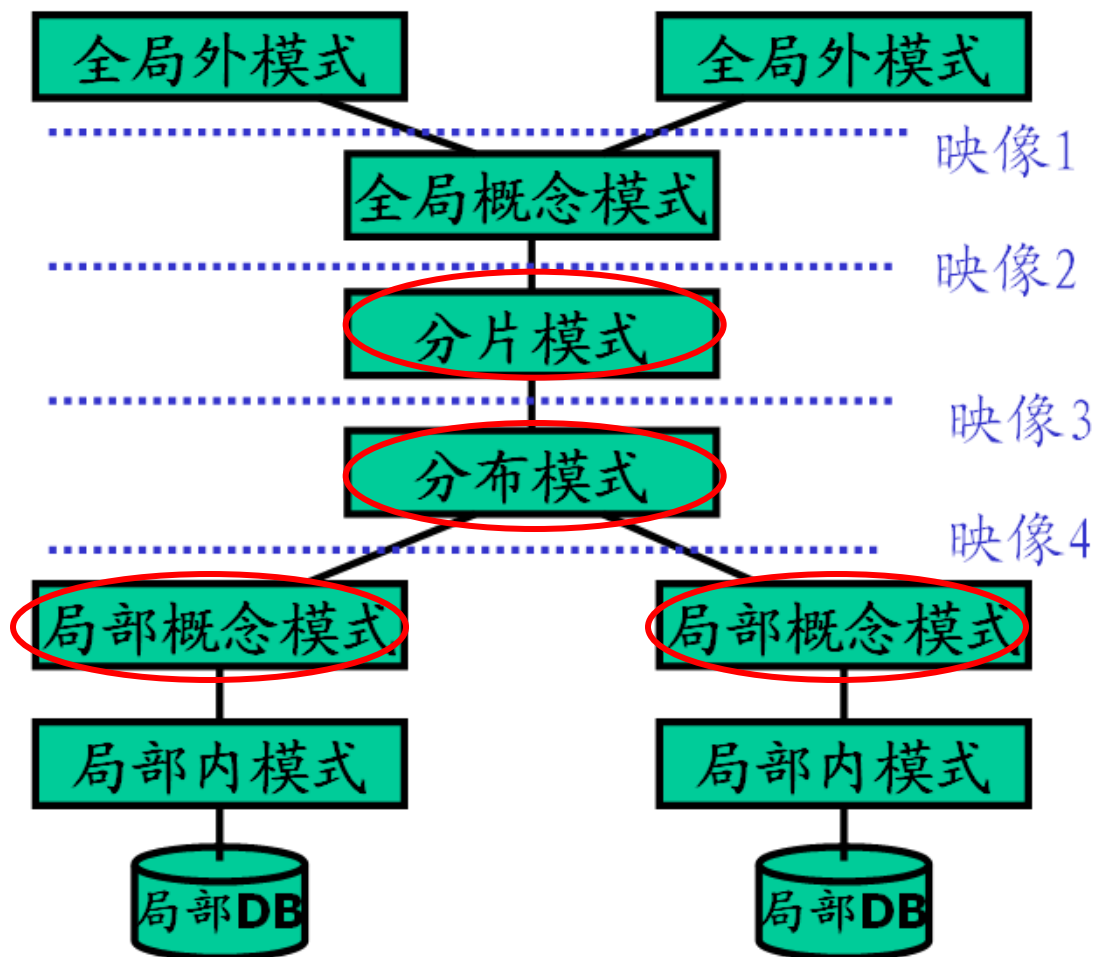
- 可以由片段重构全局关系

- 垂直分片用连接操作重构，例R的码为A，R1，R2，R3为全局关系R的垂直分片，且都包含A，则 $R = R1 \bowtie R2 \bowtie R3$
- 水平分片用并操作重构，例SC_A, SC_B为SC的两个水平分片，则 $SC = SC_A \cup SC_B$



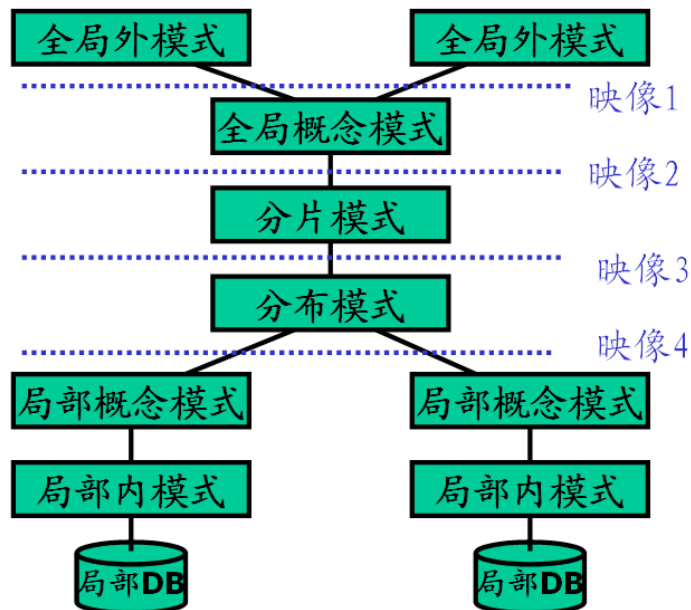
分布透明性 (1)

分布透明性包括：分片透明性，位置透明性，局部数据模型透明性



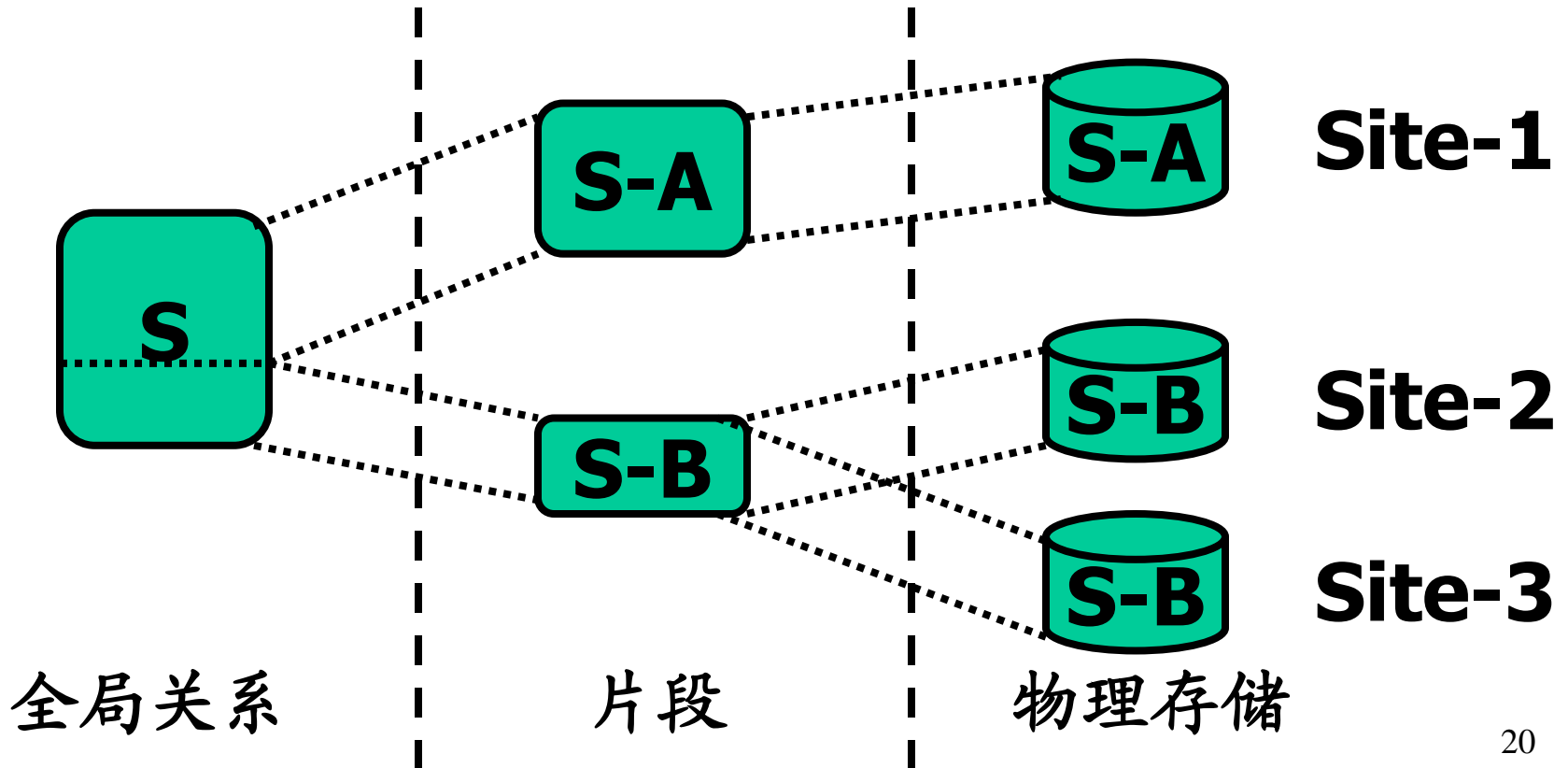
分布透明性 (2)

- **分片透明性**——用户或应用程序只对全局关系进行操作而不必考虑关系的分片。它是分布透明性的最高层次。
- **位置透明性**——用户或应用程序不必了解片段的存储场地也不必关心各数据副本的一致性。
- **局部数据模型透明性**——用户或应用程序不必了解局部场地上使用的是哪种数据模型。模型的转换以及查询语言等的转换均由分布模式/局部概念模式（映象4）完成。

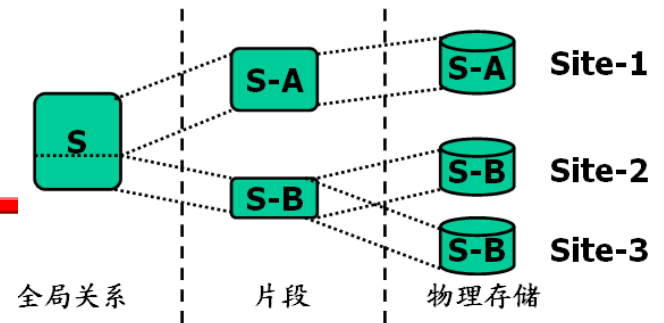


分布透明性示例

- 示例：设有全局关系 S ($S\#, SN, SD, SA$)，它被划分为两个片段 $S-A$ (本科生)， $S-B$ (研究生)， $S-B$ 有两个副本。



分布透明性示例



执行查询操作：检索所有学生的学号和年龄。

- 若系统具有分片透明性，则执行操作：

```
SELECT S#,SA  
FROM S
```

- 若系统具有位置透明性，不具有分片透明性，则可执行操作：

```
SELECT S#, SA  
FROM S_A  
UNION  
SELECT S#, SA  
FROM S_B
```

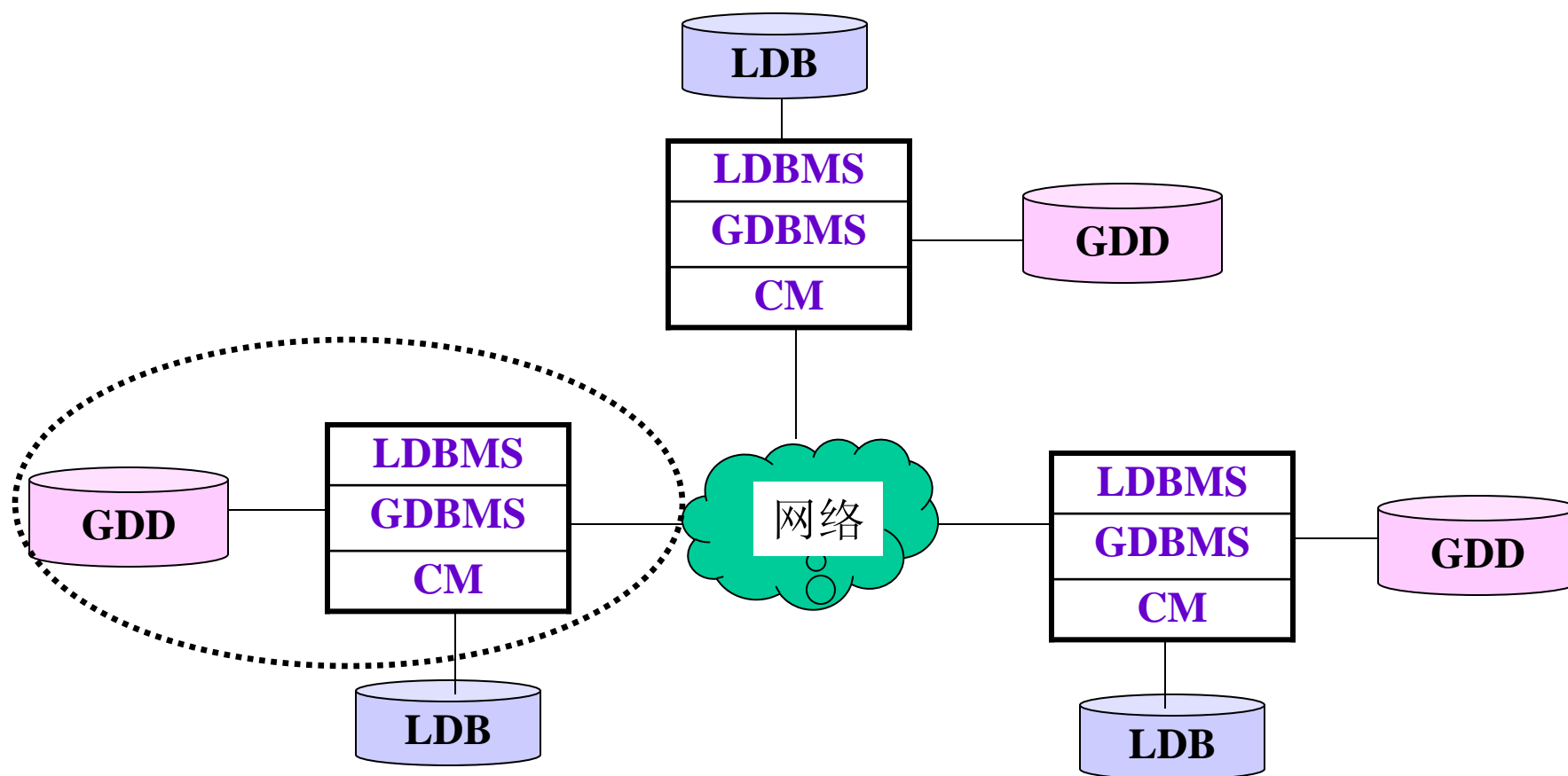
- 若系统只具有局部数据模型的透明性，不具有位置透明性与分片透明性，在执行操作：

```
SELECT S#, SA  
FROM S_A AT SITE-1  
UNION  
SELECT S#, SA  
FROM S_B AT SITE-3
```

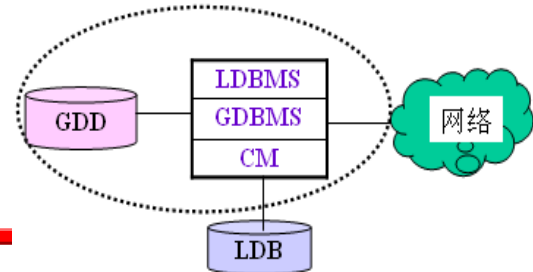


分布式数据库管理系统DDBMS

- DDBMS的组成



DDBMS的组成



— LDBMS(局部场地上的数据库管理系统)

- 功能：建立和管理局部数据库，提供场地自治能力，执行局部应用及全局查询的子查询。

— GDBMS (全局数据库管理系统)

- 功能：提供分布透明性，协调全局事务的执行，协调各局部DBMS以完成全局应用，并保证数据库的全局一致性，执行并发控制，实现更新同步，提供全局恢复功能。

— GDD (全局数据字典)

- 存放全局概念模式、分片模式、分布模式的定义以及各模式之间映象的定义。
- 存放有关用户存取权限的定义，以保证全局用户的合法权限和数据库的安全性。
- 存放数据完整性约束条件的定义。

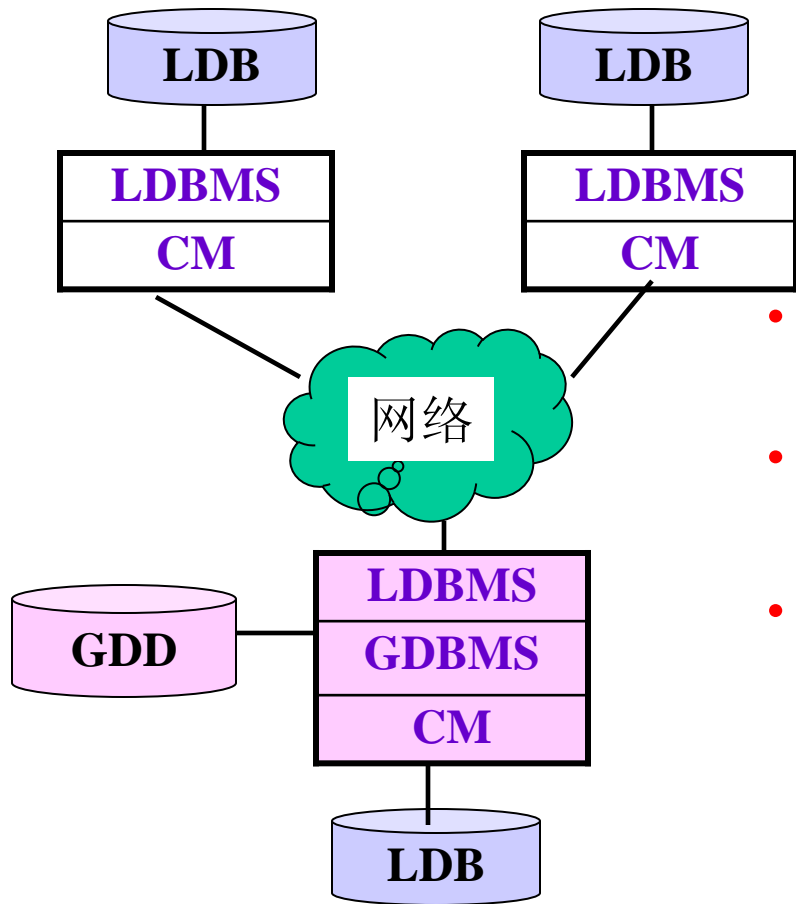
— CM (通信管理)

- 在分布式数据库各场地之间传递消息和数据，完成通信功能。

DDBMS 的分类 (1)

- 按全局控制方式分类

- 全局控制集中的DDBMS



- 特点：**GDBMS集中在某一结点上，GDD只有一个，也放在该结点上。

- 优点：**

- 控制简单，容易设计实现

- 缺点：**

- 易形成瓶颈，并且一旦该结点出现故障，整个系统将瘫痪。

DDBMS 的分类 (2)

— 全局控制分散的DDBMS

- **特点：** GDBMS分散在每一个结点上。GDD 也在每个结点上有一份。这类结构称为**完全分布的DDBMS**。
- **优点：** 结点独立，自治性强，单个结点出现问题不会使系统瘫痪；
- **缺点：** 全局控制的协调机制和一致性维护都比较复杂。

— 全局控制部分分散的DDBMS

- 根据应用的需要将全局数据库管理器和全局数据字典分散在某些结点上。介于上述两者之间的体系结构。

DDBMS 的分类 (3)

- 按局部DBMS的类型分类
 - 同构型DDBMS
 - 每个结点的局部数据库具有相同的DBMS，即使硬件与操作系统不相同。
 - 异构型DDBMS
 - 各结点的局部数据库具有不同的DBMS



分布式数据库系统的主要技术

- 分布式查询处理和优化
- 分布事务管理

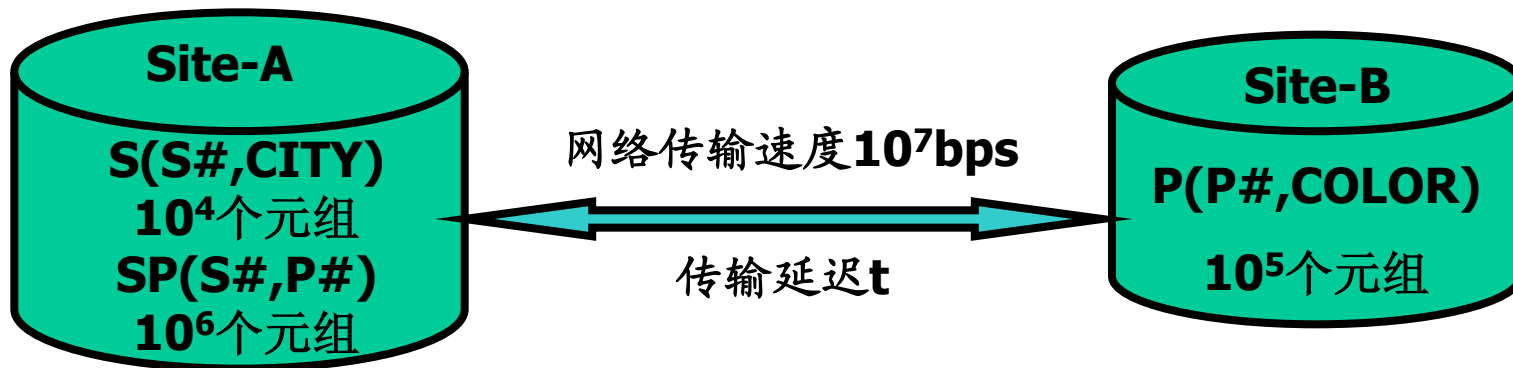
分布式查询处理和优化

分布式数据库系统中查询处理较集中式数据库复杂，查询优化较集中数据库更重要，效果更显著。

- 分布式查询中的突出矛盾
- 查询处理和优化要解决的问题
- 查询优化的目标
- 连接查询的优化

分布式查询中的突出矛盾

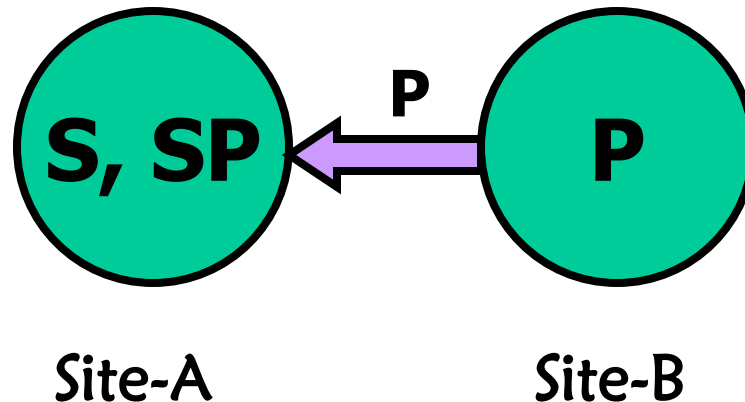
• 例



- 传送时间 $T = \text{总传输延迟} + \text{总数据量} / \text{传输速度}$
- 数据库：
 - Site-A: 供应商库S, 供应商装运单库SP; Site-B: 产品库P;
 - 每个关系的元组均为100byte长;
- 查询: 求供应红色零件的北京的供应商号。
*select S.S# from S, P, SP
where S.S# = SP.S# and SP.P# = P.P#
and COLOR = '红色' and CITY = '北京';*
- 估算值: 红色零件数=10; 北京供应商的装运单数=10⁵;

6种可能的查询策略

1. 把关系P从B站传送到A站，在A站进行查询

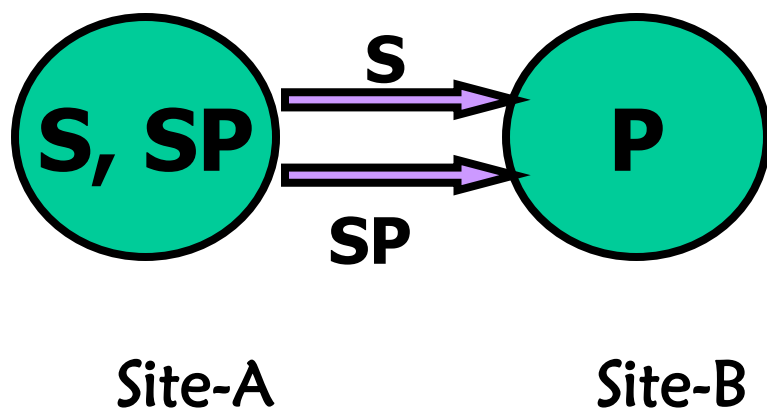


S(S#,CITY)	10 ⁴ 个元组
P(P#,COLOR)	10 ⁵ 个元组
SP(S#,P#)	10 ⁶ 个元组

红色零件数=10;
北京供应商的装运单数=10⁵

$$\text{传送时间 } T = 1t + 10^5 * 100 / 10^7 = (t+1)s$$

2.把关系S, SP从A站传送到B站, 在B站进行查询

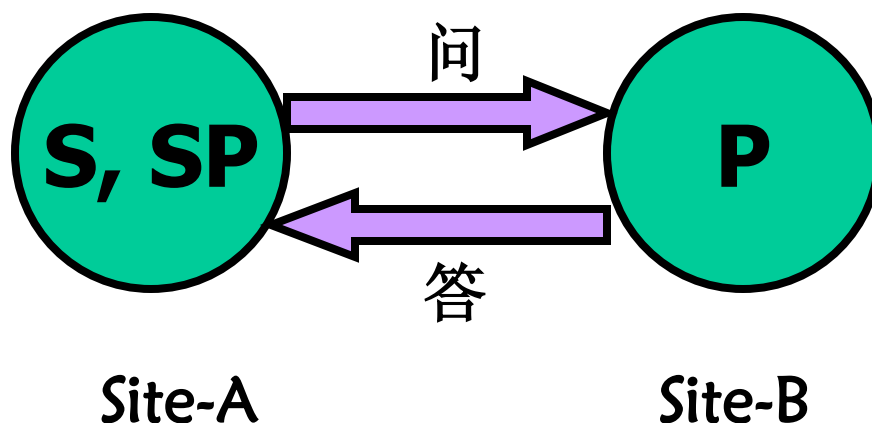


S(S#,CITY)	10⁴个元组
P(P#,COLOR)	10⁵个元组
SP(S#,P#)	10⁶个元组

红色零件数=10;
北京供应商的装运单数=10⁵

$$\text{传送时间 } T = 2t + (10^4 + 10^6) * 100 / 10^7 \approx (2t + 10)s$$

3. 在A站连接S与SP，选出城市为北京的元组（有 10^5 个），然后对其中每个元组的P#，询问B站，看其是否为红色。

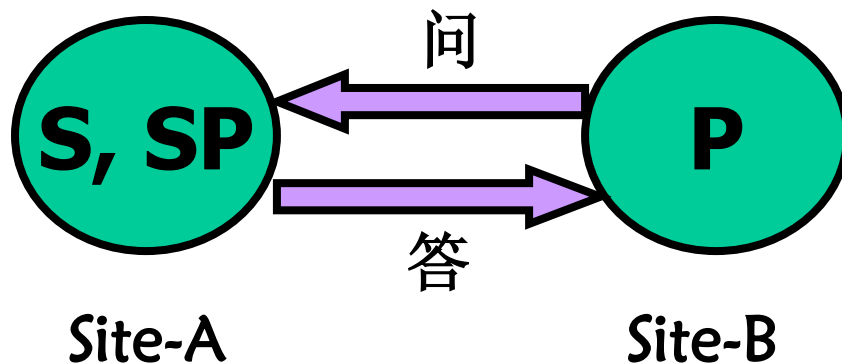


S(S#,CITY)	10 ⁴ 个元组
P(P#,COLOR)	10 ⁵ 个元组
SP(S#,P#)	10 ⁶ 个元组

红色零件数=10;
北京供应商的装运单数=10⁵

传送时间 $T = 2t * 10^5s$

4. 在B站选出红色零件（有10个），然后对每个元组询问A站，看北京的供应商是否供应此零件。

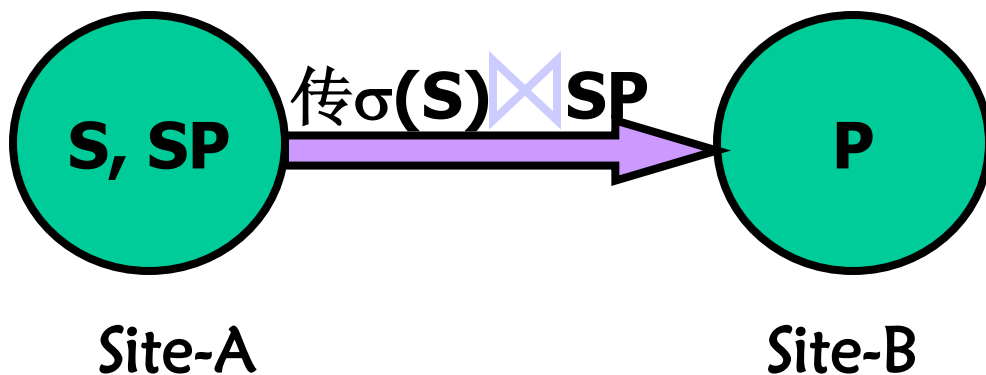


S(S#,CITY)	10⁴个元组
P(P#,COLOR)	10⁵个元组
SP(S#,P#)	10⁶个元组

红色零件数=10;
北京供应商的装运单数=10⁵

$$\text{传送时间 } T = 2 t * 10 = 20t_s$$

5.在A站选出北京的供应商的装运单 (10^5 个), 传送到B站, 在B站完成查询。

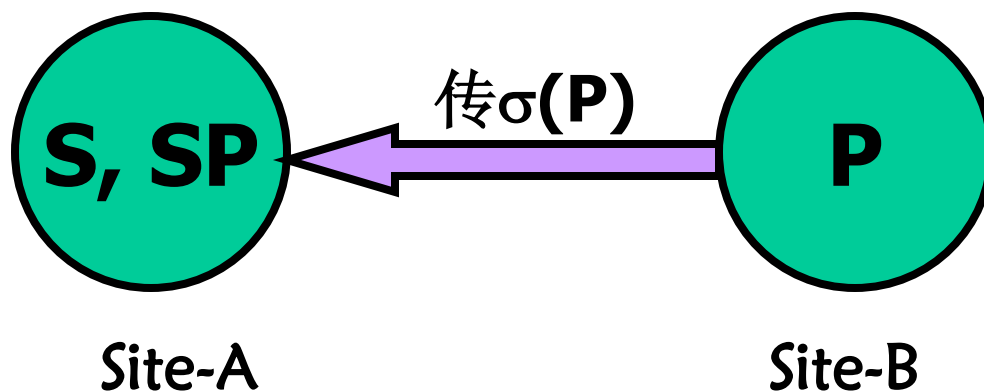


S(S#,CITY)	10⁴个元组
P(P#,COLOR)	10⁵个元组
SP(S#,P#)	10⁶个元组

红色零件数=10;
北京供应商的装运单数=10⁵

$$\text{传送时间 } T = 1t + 10^5 * 100 / 10^7 = (t+1)s$$

6. 在B站选出红色零件（10个），把结果传送到A站，在A站完成查询。



S(S#,CITY)	10⁴个元组
P(P#,COLOR)	10⁵个元组
SP(S#,P#)	10⁶个元组

红色零件数=10;
北京供应商的装运单数=10⁵

$$\text{传送时间 } T = 1t + 10 * 100 / 10^7 = 1t$$

序号	示意图	$T(1s, 10kbps)$	$T(t, 10Mbps) (t=10^{-3}s)$
1		$10^3 s$ (16.7分)	$(t+1)$ (1s)
2		$10^4 s$ (28小时)	$(2t+10)$ (10s)
3		$2 * 10^5 s$ (2.3天)	$2t * 10^5$ (200s)
4		20s	20t (0.02s)
5		$10^3 s$ (16.7分)	$(t+1)$ (1s)
6		1.1s	1t ($10^{-3}s$)

-
- 不同的查询策略通信时间相差很大，必须进行优化；
 - 有些策略中数据传输时间和传输延迟都要考虑，而另一些策略主要考虑传输延迟（如问答方式），还有一些策略数据传输量大，主要考虑传输时间。

查询处理和优化要解决的问题

- 在分布式数据库系统中，查询可分为三类：局部查询，远程查询，全局查询。
 - 局部查询和远程查询只涉及单个结点的数据（本地的或远程的），可以采用集中式数据库的处理技术；
 - 全局查询涉及到多个结点的数据，十分复杂。
- 为了执行全局查询和确定一个好的查询策略，要做许多判断、计算工作。
 - 查询分解
 - 把全局查询分解为若干子查询，每个子查询只涉及某一个结点的数据，可由局部DBMS处理。必须选择查询开销最省的那些结点（物理片段）。
 - 选择操作执行的次序
 - 主要是确定涉及不同结点上关系的连接和并操作的次序。
 - 选择执行操作的方法
 - 包括选择存取路径、选择某种操作的算法以及连接的执行方法。

查询优化的目标

- 查询处理策略的选择是以执行查询的预期代价为依据的。
 - 查询执行的开销：I/O代价+CPU代价+通信代价
 - 分布式查询可分为存取策略的分布优化和局部优化。其中存取策略的分布优化更重要。
- 通信代价可计算，通常是数据传输量的函数：
$$TC(X) = C_0 + X * C_1$$

X 为数据传输量， C_0 为两结点初始化一次传输所花费的开销， C_1 为传输率，即单位数据传输所花费的时间。
- 分布式数据库查询优化中，将“通信代价”作为首要因素进行研究。因此，查询优化的首要目标是：使查询执行时通信代价最省。
- 不同结点之间的连接操作和并操作是数据传输的主要原因，因此连接查询的优化是优化中研究的重要问题。

连接查询的优化

- **半连接**：使用半连接来缩减关系（或片段）进而节省传输开销。
- 半连接法定义： $R \bowtie_{A=B} S = R \bowtie_{A=B} (\Pi_B(S))$

例：

R1

A	B
a1	b1
a2	b1
a2	b3
a2	b4
a3	b3

\bowtie

R2

B	C
b1	c1
b2	c2
b5	c1
b5	c2
b6	c4
b7	c2
b8	c3

=

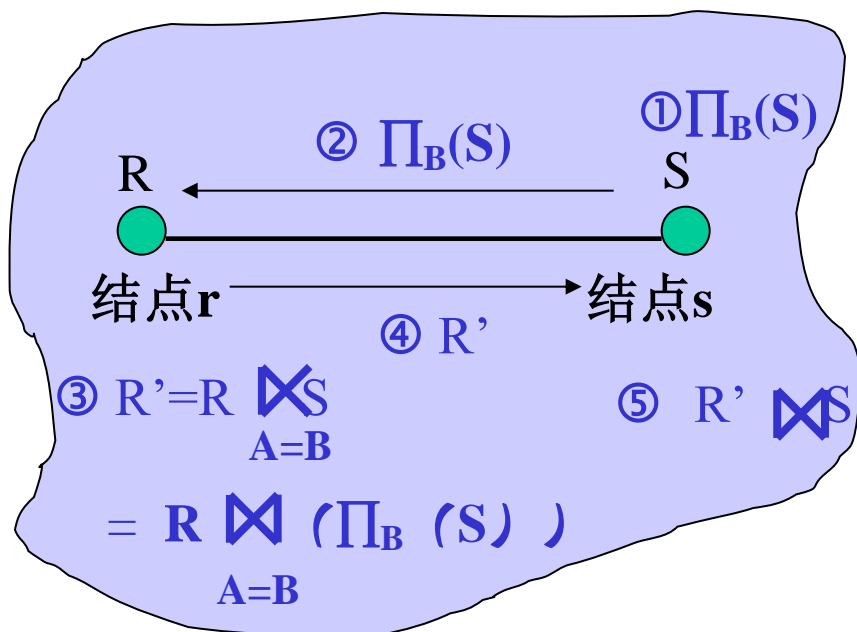
$R1 \bowtie R2$

A	B
a1	b1
a2	b1

- 根据半连接定义，有

$$R \bowtie_{A=B} S = (R \bowtie_{A=B} S) \bowtie_{A=B} S = (R \bowtie_{A=B} \Pi_B(S)) \bowtie_{A=B} S$$

- 半连接在查询优化中的应用



—采用半连接的通讯代价：

$$C_{SJ} = 2C_0 + C_1 *$$

$$(\text{size}(B) * \text{card}(\Pi_B(S)) + \text{size}(R) * \text{card}(R'))$$

—不采用半连接的通讯代价：

$$C_{JN} = C_0 + C_1 * \text{size}(R) * \text{card}(R)$$

—当R中参与连接的元组足够少时
采用半连接策略是有利的。 41

分布事务处理

- 分布事务的原子性与可串行性
 - 在分布式数据库系统中，一个全局事务被划分为在许多结点上的子事务。分布事务的原子性是：组成该事务的所有子事务要么一致地全部提交，要么一致地全部回滚。
 - 在多用户系统中，还必须保证分布式事务的可串行性。
- 分布事务管理主要包括：事务的恢复和并发控制。

分布事务的恢复

- 每个场地都有一个局部事务管理器，负责管理局部子事务的执行。同时，各局部事务管理器之间必须相互协调，保证分布事务的原子性：各子事务要么都提交，要么都回滚。
- 对局部事务管理器进行协调，保证分布事务原子性最常用的技术——两段提交协议 (2- Phase-Commitment Protocol)

两段提交协议 (1)

- 两段提交协议把一个分布事务的所有局部事务管理分为两类：**协调者 (一个)**，**参与者**。
 - **协调者**：负责作出该事务是提交还是撤消的最后决定。
 - **参与者**：负责管理相应子事务的执行以及在各自局部数据库上执行写操作。

两段提交协议 (2)

- 两段提交协议内容

- 第一阶段：协调者征求意见作决定

- 协调者向所有参与者发出“准备提交”信息，并记入日志；参与者准备提交就回答“就绪”，否则回答“撤消”，并记入日志。
 - 如果在规定时间内，协调者收到所有参与者的“就绪”信息，则作出“提交”决定，否则将作出“撤消”决定。

- 第二阶段：参与者执行决定

- 协调者将有关决定写入日志，然后把这个决定发送给所有的参与者。
 - 所有参与者收到命令后，首先在日志中记入“收到提交/撤消决定”的信息，并向协调者发送应答消息，最后执行相应决定。
 - 协调者收到所有参与者的应答消息后，一个事务的执行到此结束。有关日志信息可以脱机保存。

- 采用两段提交协议后，当系统发生故障时，各场地利用各自有关的日志进行事务恢复。

并发控制 (1)

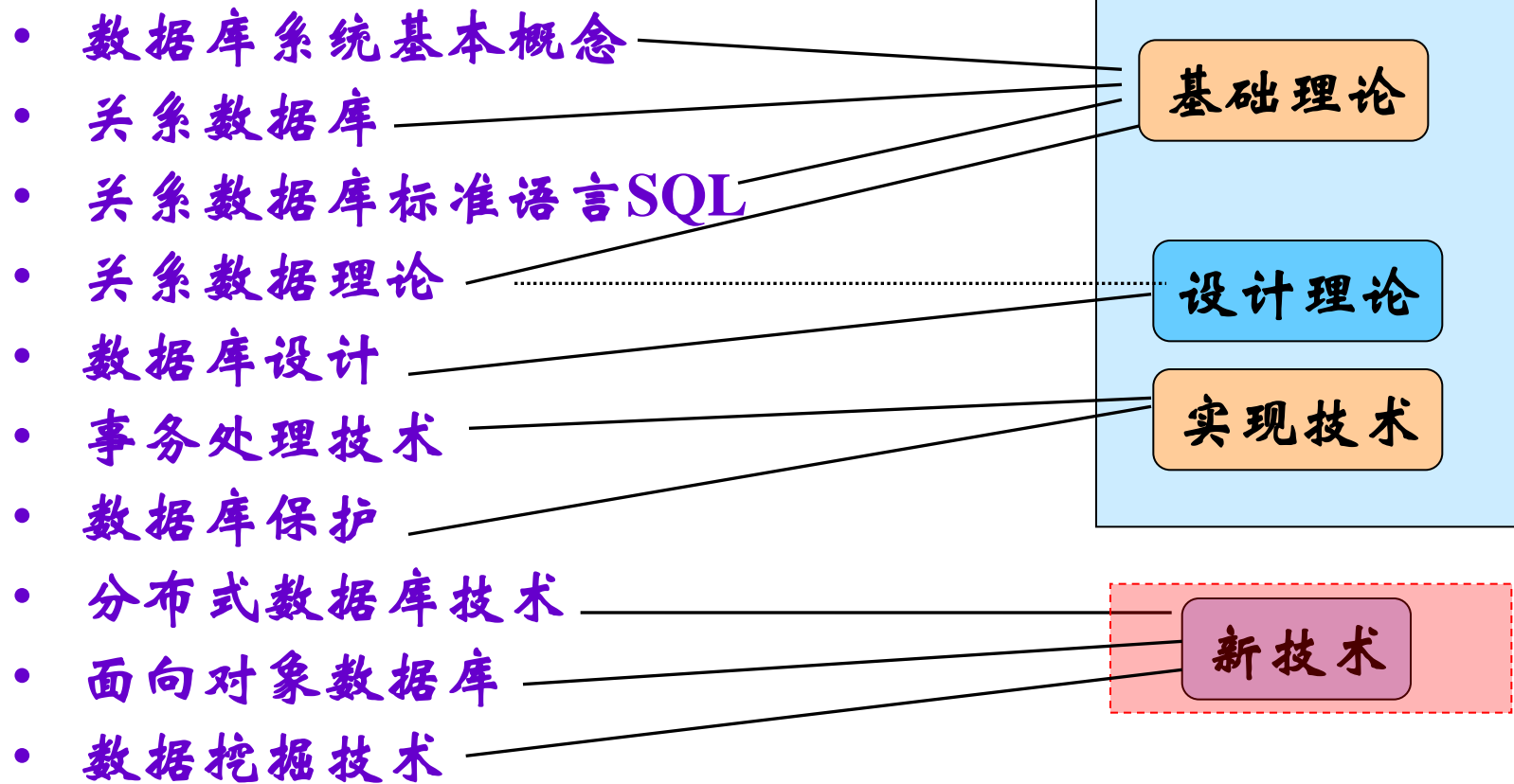
- 分布式数据库系统中的并发控制也可以采用封锁技术，但并发控制更加复杂：
 - 分布式数据库系统支持多副本；
 - 由于事务的分布执行，封锁的方法会引起全局死锁。

并发控制 (2)

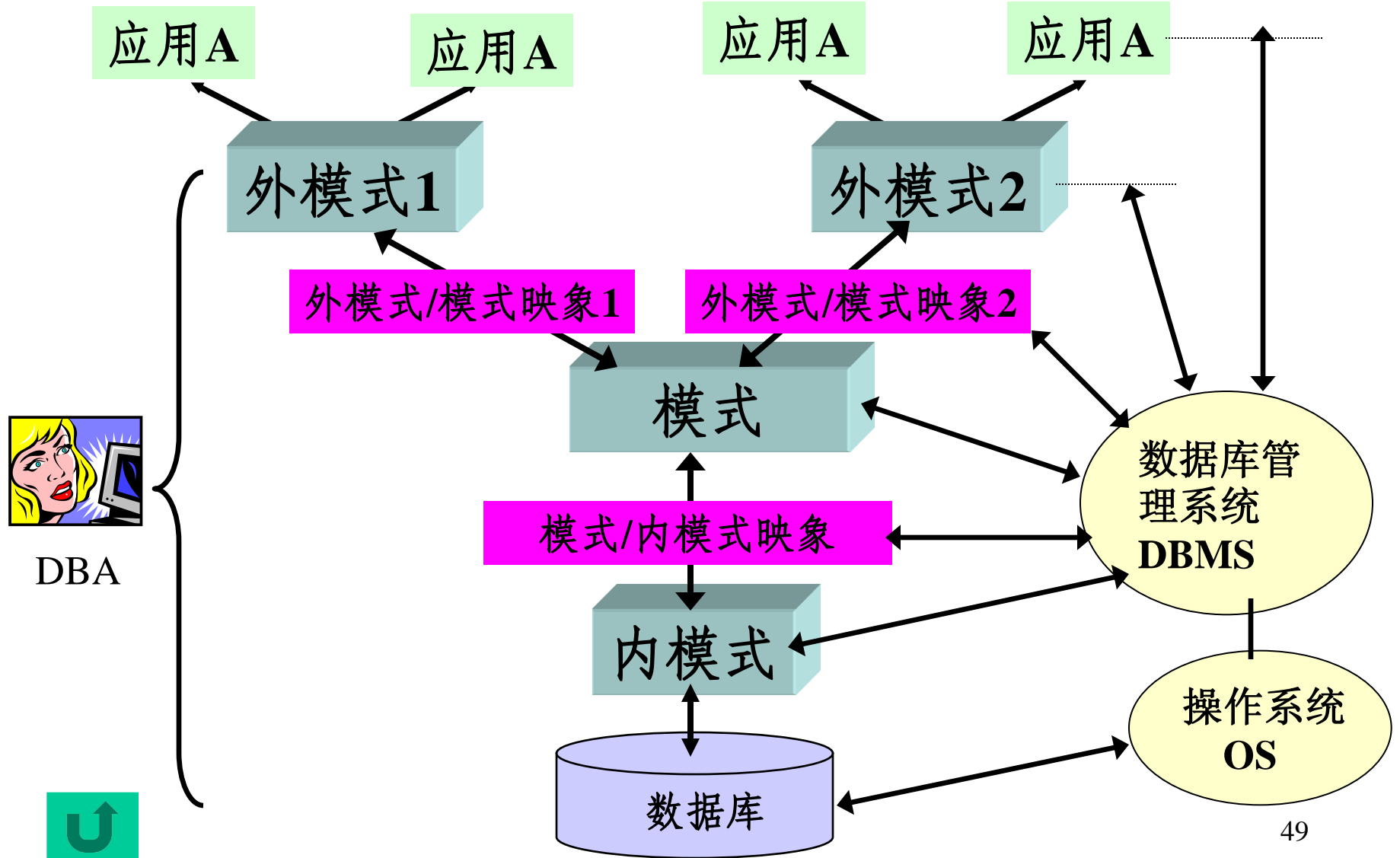
- 分布式数据库系统并发控制进一步的策略：
 - 处理多副本封锁的几种可行方法：
 - 对写操作，要申请所有副本的X锁；对读操作，只要申请某个副本的S锁。
 - 无论是写操作还是读操作都要对大多数副本申请X锁或S锁。
 - 规定某个场地上的副本为主副本，所有的读、写操作均申请对主副本的封锁。
 - 解决全局死锁（两个以上场地上发生死锁）
 - 死锁检测及解除方式
 - 死锁预防，如对事务按某一标准进行排序，只允许事务按这一次序单向等待。



课程知识点回顾



数据库系统的结构



第十章 数据管理技术新进展

- 面向对象数据库
- 数据仓库
- 非结构化数据管理
- 大数据处理

面向对象数据库产生背景

- 对于数据结构异常复杂的某些应用领域，关系数据库显示出局限性。
- 要求DBS能够处理更复杂的数据模型，能适度地演化，便于管理、构造与维护大量的持久数据，并能与大型复杂程序紧密结合。
- 面向对象数据库是面向对象程序设计技术与数据库技术结合的产物。

面向对象数据库发展现状

- 面向对象数据库系统现阶段特征：
 - 缺乏通用数据模型
 - 缺乏坚实的形式化理论基础
 - 具有较强的实践性

面向对象的基本概念

- 对象 (Object) 的概念

- 是由实体所包含的一组数据和施加于这些数据上的操作组成。对象是现实世界概念实体的模型，可以是有形的/抽象的、简单的/复杂的。

- 对象的基本特征

- 有一个状态，可由其他对象的状态构成，并且只能被自身的行为改变；
 - 有一组操作（行为），包括自操作和它操作；
 - 有唯一标识；
 - 对象之间可以互通消息。

面向对象的基本概念

- 对象（Object）的概念

- 封装性

- 基本数据和对数据进行操作的过程和函数的结合称为**封装**。
 - 基本数据的内部结构的不可访问性称为**数据隐藏**。

- 对象的结构

- **属性集合**

- 对象数据的集合。

- **消息集合**

- 对象对外提供的界面，每个消息都能够由该对象接收和响应。

- **方法集合**

- 对象的行为。方法是实现消息的程序代码。

面向对象的基本概念

- 类 (class) 的概念

- 将类似的对象组合在一起，形成一个对象类。

- 对象之间的联系

- 继承：形成类的层次结构。

- 单继承：每个子类只有一个直接超类

- 多继承：每个子类可有多直接超类

- 嵌套：一个对象的属性是另一个对象，对象间形成嵌套层次结构

面向对象数据模型

- 面向对象的数据模型是由一组类组成，这些类构成了有限的层次结构，称为类层次。
- 一个面向对象数据库模式可能包含多个类层次。在一个类层次中，一个类继承其所有超类的全部变量、方法和消息。

OODBS 的特性

- 必备特性
 - 判定OODBS的充要条件
- 可选特性
 - 使系统更完善的添加特性
- 开放特性
 - 系统设计者自由选择部分

OODBS的必备特性

- OODBS必须满足两个标准：
 - 是一个DBS
 - 持久性、存储管理、并发性、恢复和即席查询
 - 是一个面向对象的系统
 - 复杂对象、对象标识、封装性、类、继承、结合
滞后联编的重载、可扩充性、计算完备性

OODBS的必备特性

- 复杂对象

- 如果一个对象的属性是一个对象，则该对象称为复杂对象。

- 对象标识

- 每个对象都有一个系统赋予的唯一不变的标识称为对象标识OID。使对象独立于其值而存在。

OODBS的必备特性

- 重载和滞后联编

- 重写(Overriding)和重载(Overloading)
- 滞后联编: OODBS在编译时不能把重载方法的方法名联编到方法代码上, 必须在运行时把方法名联编到方法代码上, 这种推迟的联编称为滞后联编。

- 可扩充性

- 系统定义和用户定义的类在应用中没有区别, 即新定义的类和原有的类具有同等的地位。

- 计算完备性

- 由DBS的DML可表达任何可计算的功能。

OODBS的必备特性

- 永久性

- 如果一个对象在应用程序结束后仍继续存在，则该对象具有永久性（持久性）。
- 是数据库和程序设计语言的一个重大区别。

- 存储管理

- 包括索引管理、数据聚集、数据缓存、存储路径选择、查询优化等。

- 即席查询

- 通过查询语句、界面进行查询。应具有高层次、高效率和与应用无关的特性。

OODBS的可选特性

- 多重继承
- 类型检查和类型推理
- 分布
- 设计事务处理
 - 长事务或嵌套事务
- 版本管理

OODBS的可选特性

- 模式演进

- 面向对象数据库模式，随着需求的变化而变化，称为模式演进。
- 主要模式演进操作
 - 增加新类
 - 删除类
 - 修改类定义
 - 修改类之间的关系
- 模式一致性：模式内部不能出现矛盾和错误
- 模式演进实现的难点是保证模式的一致性。

OODBMS介绍

- **ORION/ITASCA**
 - MCC (Microelectronics and Computer Technology Corporation)85年开发。
- **GEMSTONE**
 - Servio公司开发。最早的商品化OODBMS之一。基于Smalltalk。
- **ONTOS**
 - Ontos公司开发，基于C++。
- **ObjectStore**
 - Object Design公司产品，基于C++。
- **Versant**
 - Versant Object Technology 公司开发。基于C++，支持C。



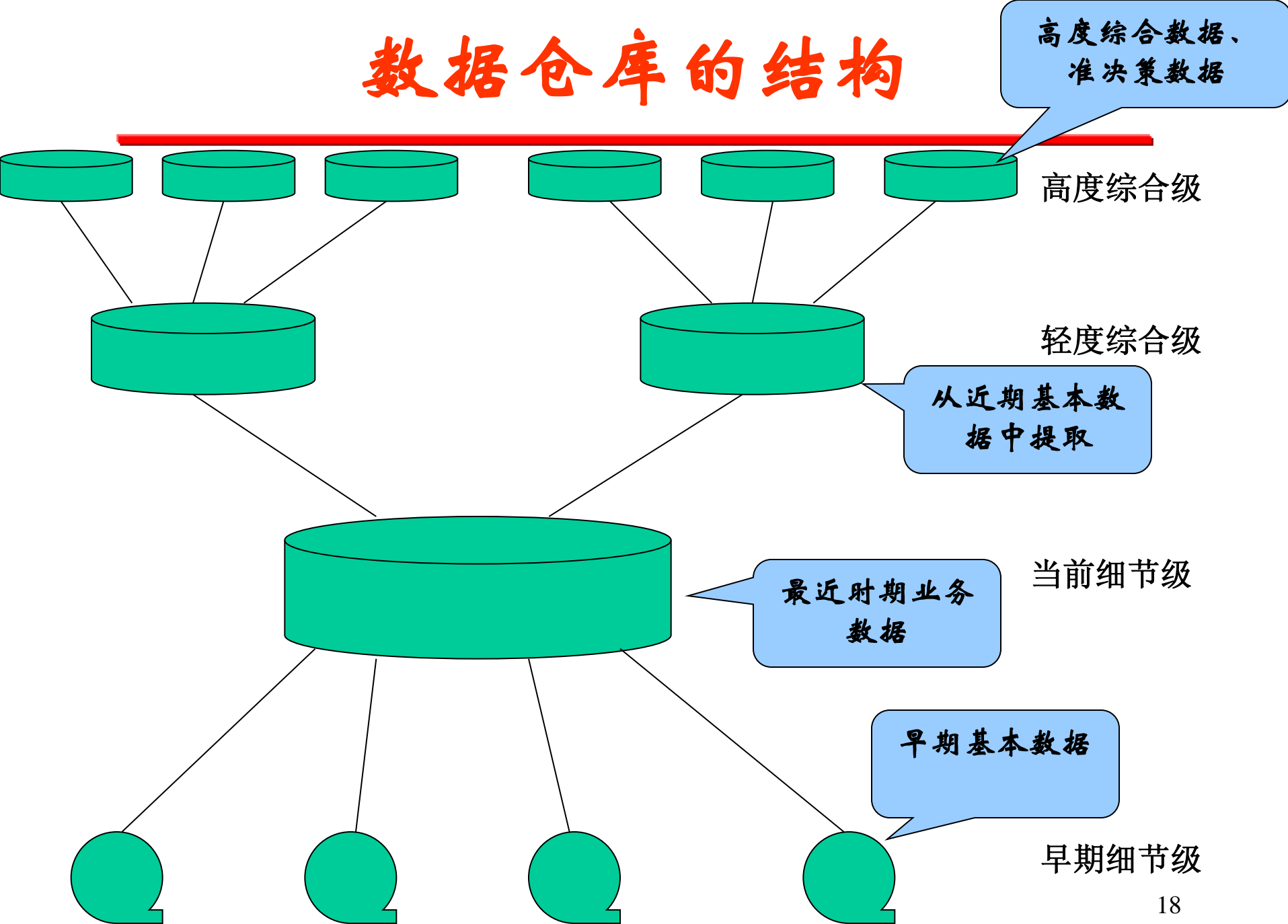
什么是数据仓库

- 数据仓库是支持管理决策过程的、面向主题的、集成的、随时间而增长的持久数据集合。
- 数据仓库中的业务
 - 数据仓库上的业务处理称作OLAP(On-Line Analytical Processing), 即联机分析处理。
 - 数据库上的业务处理称作OLTP(On-Line Transaction Processing), 即联机事务处理

数据仓库主要特征

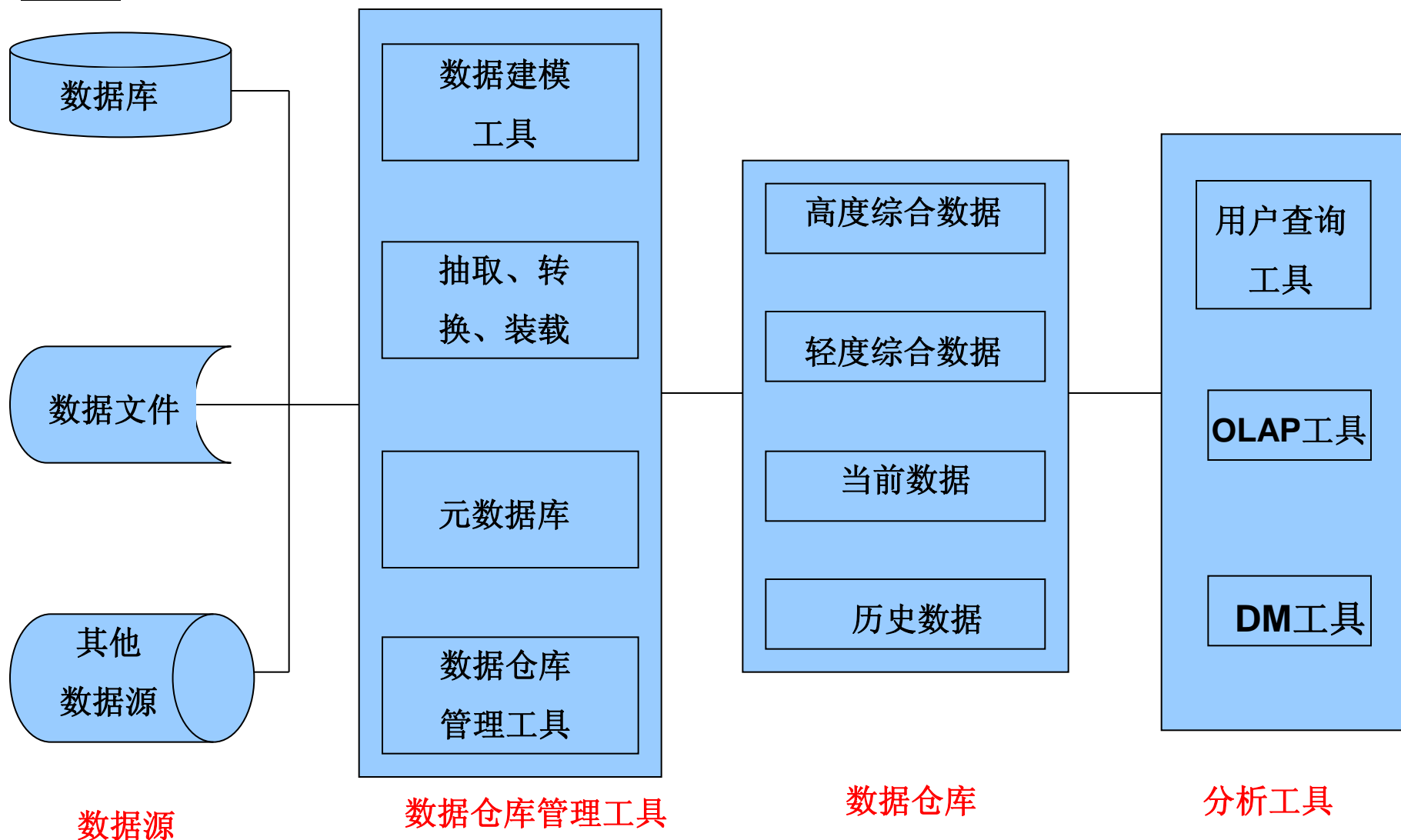
- 面向主题
 - 可以根据最终用户的观点组织和提供数据
- 管理大量信息
 - 数据仓库含有大量历史数据
- 信息存储在多个存储介质上
- 从多种数据来源中将信息集成

数据仓库的结构



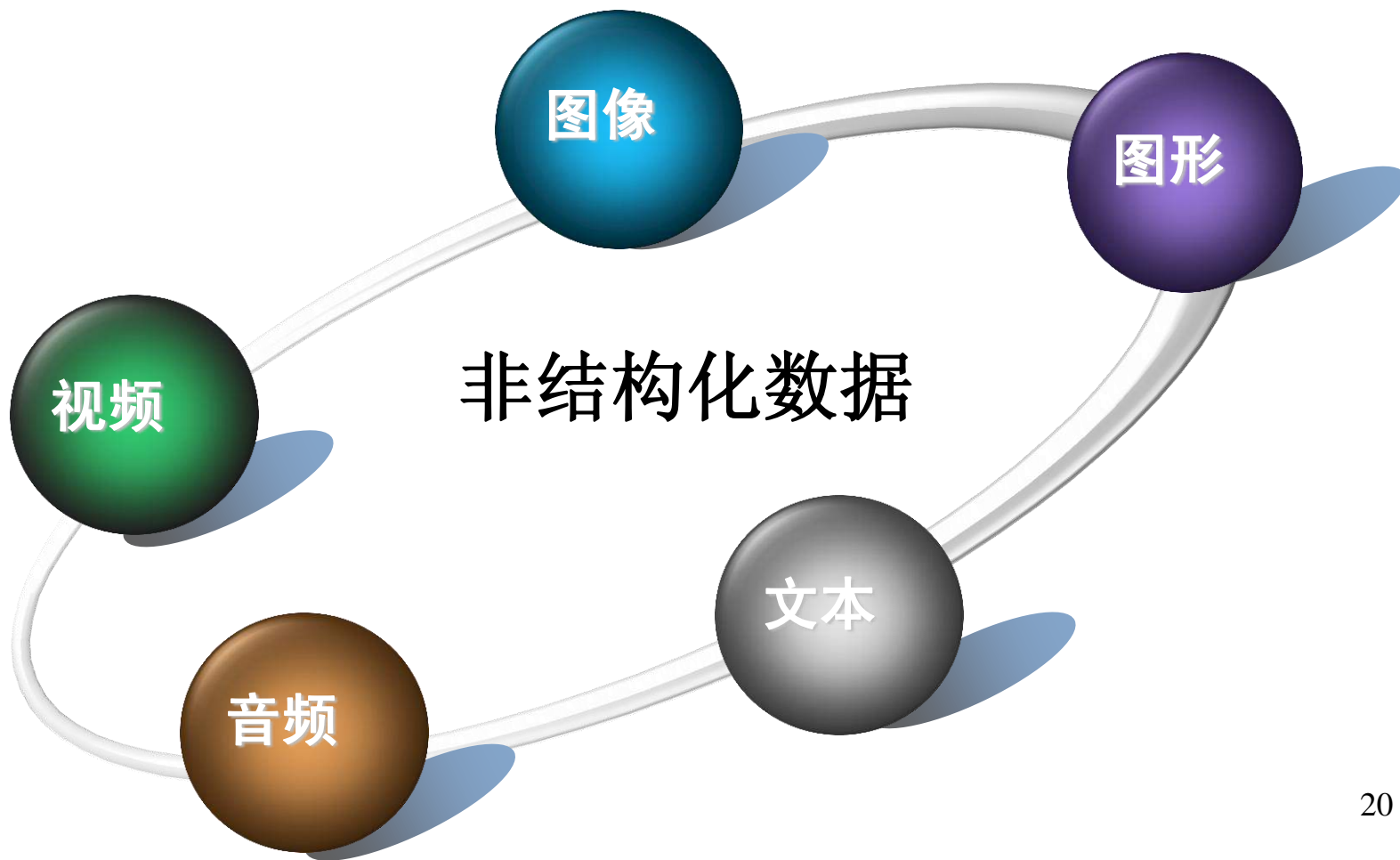


数据仓库系统的组成



非结构化数据

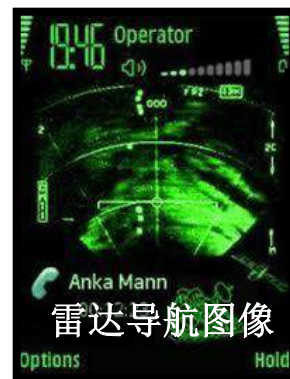
- 非结构化数据与传统数据库使用的常规数据结构不同，缺少甚至没有计算机软件解析数据所需的形式语义



非结构化数据的现状



个人私有数据



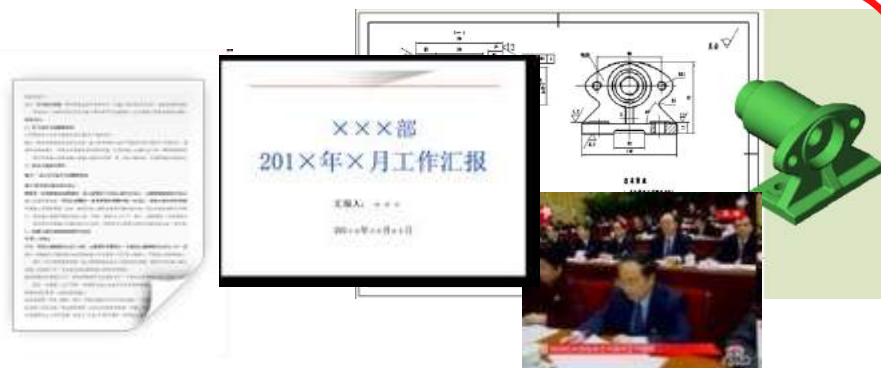
目视导航视频

雷达导航图像



天文导航图像

特定领域数据



超过**80%**的数据为非结构化数据，包括声音、影像、工程图、文件、记录资料、演示文稿等。

企业数据



到**2011年8月**，**FaceBook**有上传图片突破**100亿张**，**Flickr**上有上传图片**60亿张**

Web数据

非结构化数据的使用需求

✈ 原始数据检索

- ◆ 按内容检索
- ◆ 查准、查全、快速

✈ 数据关联检索

- ◆ 同类媒体之间的关联
- ◆ 跨媒体关联

✈ 智能检索

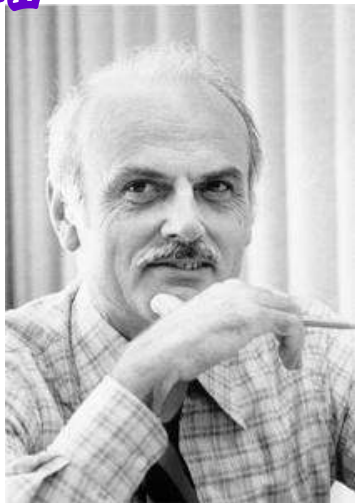
- ◆ 自动分类、聚类
- ◆ 数据分析
- ◆ 知识发现

非结构化数据管理的现状

- 原始数据查询准确性不能令人满意
 - 基于文本检索——受限于语义描述
 - 基于内容——受限于底层特征处理技术
- 缺乏关联检索、模式匹配、多维数据分析和数据挖掘等智能服务
- 目前缺少成熟的非结构化数据管理系统，国内外处于同一起跑线上

非结构化数据管理的突破口

- 数据模型是数据管理的核心，定义了数据描述结构、数据操作方法以及数据完整性约束
- E.F.Codd于70年代初成功提出关系数据模型与数据理论，奠定了几十年来关系数据库应用蓬勃发展的基础



Edgar Frank Codd, 1923—2003

Information Retrieval

P. SAXENIA, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A grouping of records which appears as such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in queries, updates, and report forms and natural growth in the types of stored information.

Scaling nonrelational, formatted data systems provide users with non-structured data or highly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on many relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's mode.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, networks or data, networks or data, relations, redundancy, consistency, security, integrity, file, relational language, products, tables, security, data integrity.

CR CATEGORIES: 3.79, 3.79, 3.79, 4.20, 4.22, 4.29

1. Relational Model and Normal Form

1.1. Introduction

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question answering systems. Levin and Maron [2] provide numerous references to work in this area.

In contrast, the problems treated here are those of data independence—the independence of application programs and terminal activities from growth in data type and changes in data representation—and certain kinds of data inconsistency which are expected to become troublesome even in nonrelational systems.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for non-inferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high-level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

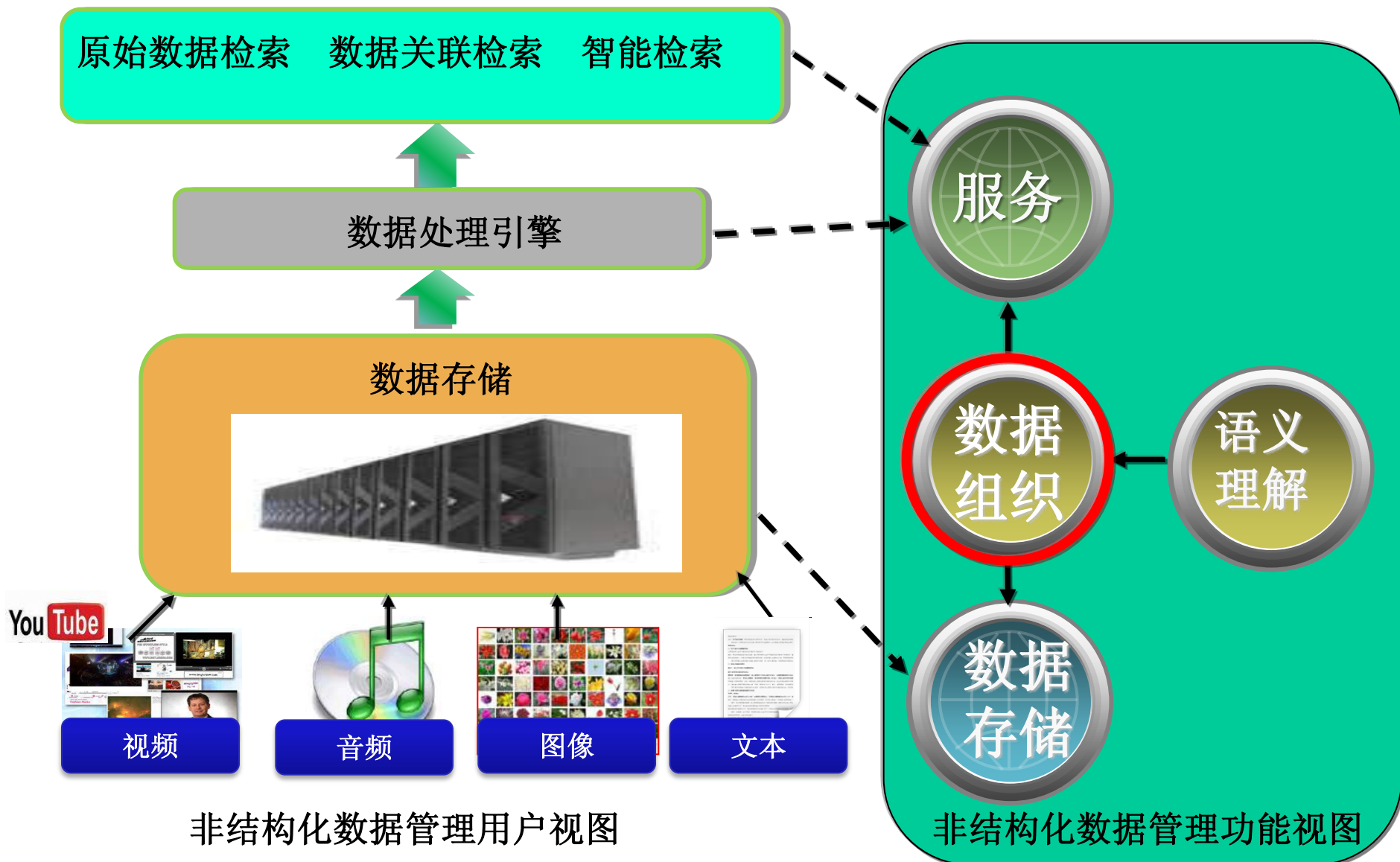
A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusion, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap"). Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. Data Dependence and Inference Systems

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed without logically requiring some application programs to be still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. Ordering Dependence. Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those ordering systems which either require or permit data elements to be stored in at least one total ordering which is closely associated with the heuristic (determined) ordering of addresses. For example, the records of a file concerning parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a subordering of) the

非结构化数据管理的突破口



非结构化数据的构成

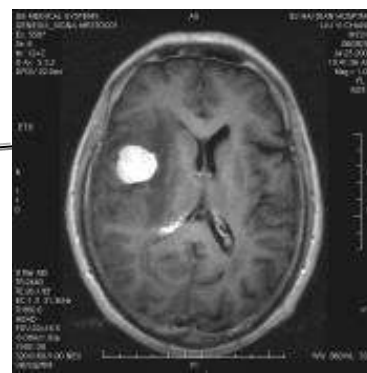
- 每种数据组成复杂

- 多模态

- 关联性强

图像语义描述

语义项	描述
图像质量	预定义的质量类型, 如很差、差、一般、清晰、高清
拍摄者	图像的拍摄者
拍摄时间	图像的拍摄时间
拍摄地点	图像的拍摄地点
感兴趣区域	图像中用户感兴趣区域的描述
内容描述	图像的内容描述
类别	预定义的图像类别, 如动物、植物、运动等
用户评论	用户对图像的评论
宽度	图像的宽度
高度	图像的高度
关键字	反映图像内容的关键字
标题	图像内容的标题
文件后缀	图像文件的后缀类型, 如jpeg, png, gif



图像底层特征

可伸缩颜色描述符(64维): "scalablecolor"; 0; -164 40 18 58 9

颜色空间分布描述符(120维): "descriptorColorLayout"; 33

边缘直方图描述符(80维): "edgehistogram"; 0 0 0 0 0 0

色彩边缘方向性描述符(综合颜色和纹理特征, 144维): "cedd"; 144

模糊颜色纹理直方图(综合颜色和纹理特征, 192维): "fcth" 192

颜色直方图描述符(256维): "RGB" 256 73 13 20 18 20 16

纹理特征(粗糙、对比度、方向性, 18维): "tamura" 18

非结构化数据的构成特点

基本属性 (Basic Attribute)

- 所有数据都具有的一般属性，包括名称、类型、创建者等

语义特征(Semantic Feature)

- 以文字表达的非结构化数据特有的语义属性，包括作者创作意图、数据主题说明、底层特征含义等语义要素

底层特征(Low-level Feature)

- 通过各种专用处理技术（如图像、语音、视频等）获得的非结构化数据特性，如颜色、纹理、形状等

原始数据(Data)

- 非结构化数据的原生态文件



分析、抽象



Notes



XML



图像



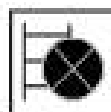
邮件



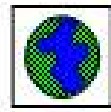
Word



音频



视频



互联网



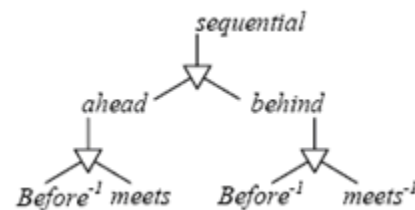
文档管理
系统



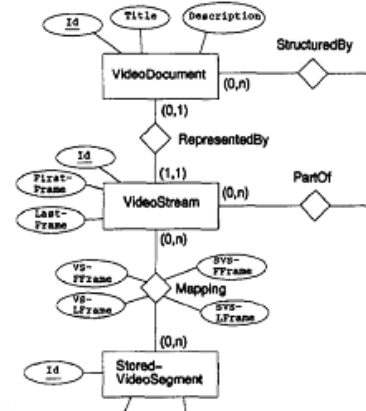
数据库

D1	D2	...	Dn
d1	d2	...	dn
...
...
...

关系模型

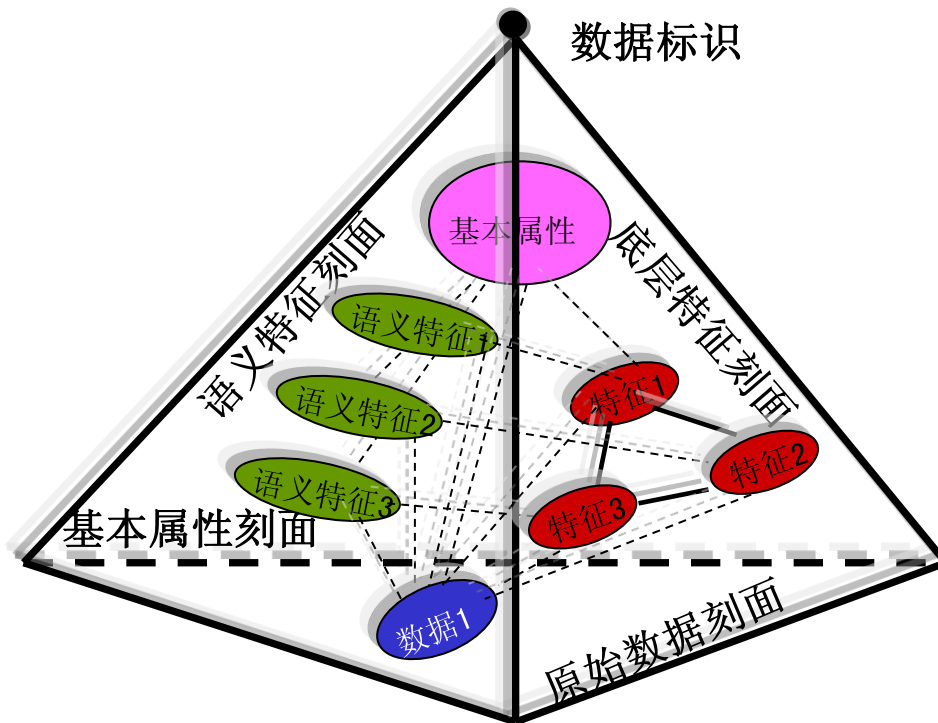


面向对象模型



E-R模型

一种非结构化数据的数据模型——四面体模型



- 四面体模型=顶点+刻面+连线
 - 顶点：数据的标识
 - 刻面1：基本属性刻面
 - 点集：基本属性
 - 刻面2：语义特征刻面
 - 点集：各种语义特征描述
 - 刻面3：底层特征刻面
 - 点集：各种底层特征
 - 刻面4：原始数据刻面
 - 点集：各种原始数据
 - 连线：四个刻面上点之间的关联

$Tetrahedron = (V, BA_FACET, SF_FACET, LF_FACET, RD_FACET, CONUS)$

$BA_FACET = \{Basic_Attribute\}$

$SF_FACET = \{Semantic_Feature_j \mid j \in [1, m]\}$

$LF_FACET = \{Lowlevel_Feature_k \mid k \in [1, m]\}$

$RD_FACET = \{Data_l \mid l \in [1, p]\}$

$CONUS = \{BA_FACET \times SF_FACET \cup BA_FACET \times LF_FACET \cup BA_FACET \times RD_FACET$
 $\cup SF_FACET \times LF_FACET \cup SF_FACET \times RD_FACET \cup LF_FACET \times RD_FACET\}$

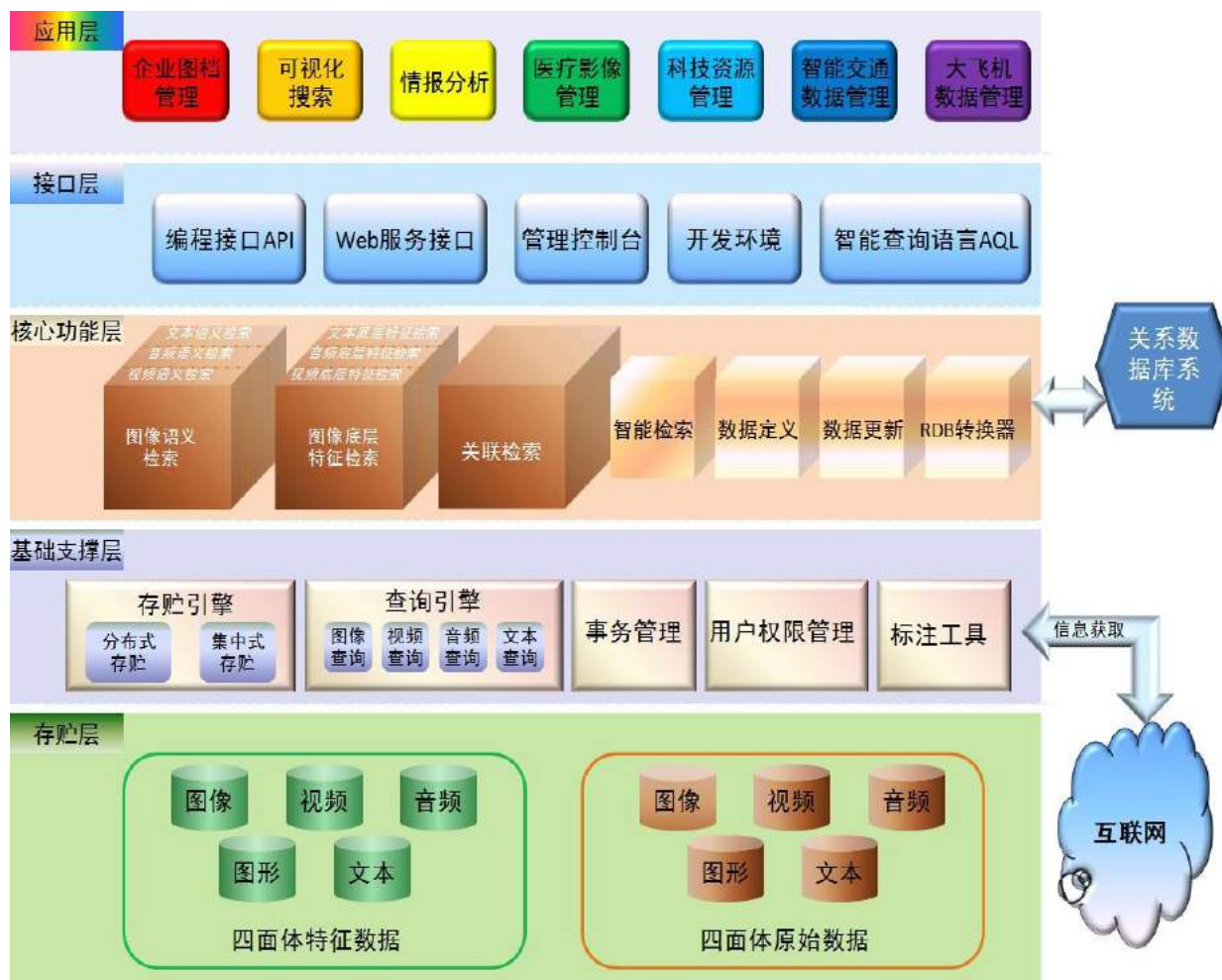
海量非结构化数据管理系统AUDR

— 核心功能层

- 图像、视频、音频、文本的语义检索
- 图像、视频、音频、文本的内容检索
- 图像、视频、音频、文本的关联检索
- 图像、视频、音频、文本的数据定义与更新

— 基础支撑层

- 存储引擎
- 图像、视频、音频、文本的查询引擎
- 图像、视频、音频、文本的标注工具
- 存储管理工具
- 事务管理工具
- 安全管理工具

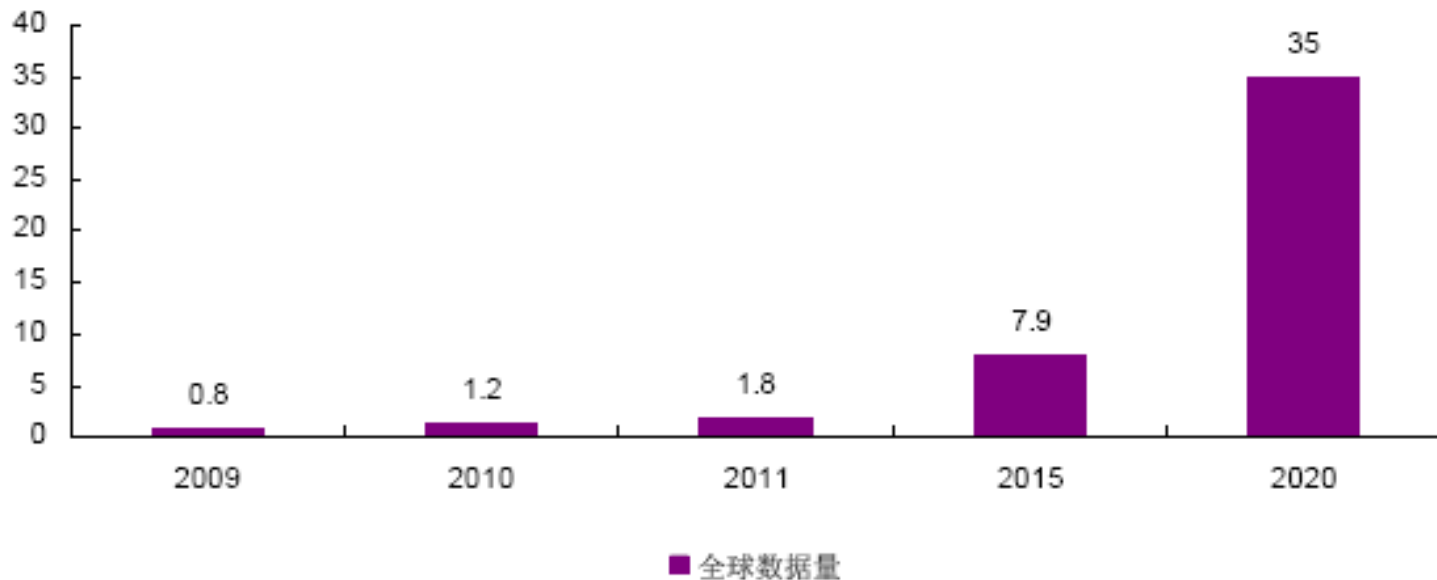


大数据时代

- 2010 年正式进入ZB 时代,根据IDC 监测, 全球数据量大约每两年翻一番
- 预计到2020 年, 全球将总共拥有35ZB 的数据量

IDC全球数据量预测

单位: ZB

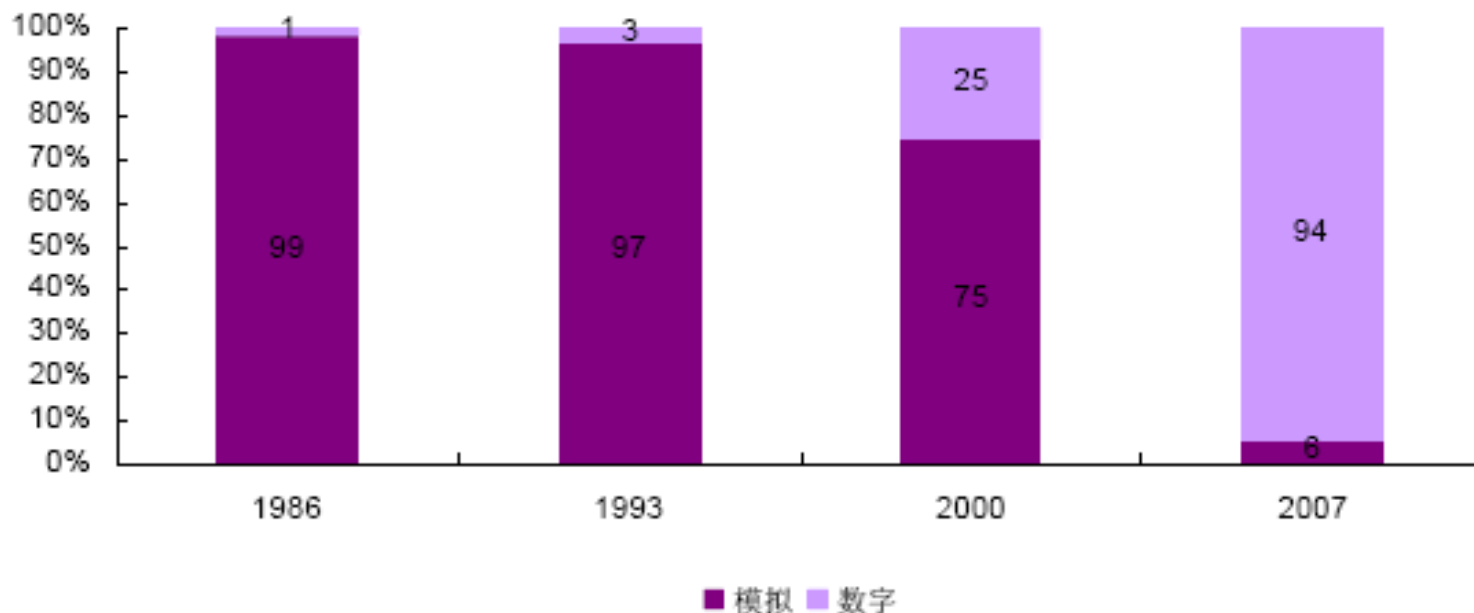


数据单位: Byte、KB、MB、GB、TB、PB、EB、ZB、... ...³⁰

什么是大数据

- “Big data” refers to datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze.
——无法用一般数据库软件工具进行获取、管理和分析处理的数据集合（麦肯锡全球研究所）
- 大数据就是一个量特别大（10TB以上），数据类别特别多的数据集

大数据的产生



资料来源：Hilbert and López, “The world’s technological capacity to store, communicate, and compute information,” Science, 2011

- 超过80%比例的数据是非结构化数据，已经超越了传统数据库的管理能力

大数据的产生

	视频	图像	音频	文本/数字	产生频率
银行	中	中	中	高	
保险	低	低	低	高	
证券投资服务	低	低	低	高	
离散制造业	中	中	低	高	
流程制造业	中	中	低	高	
零售业	中	低	低	高	
批发行业	低	低	低	高	
专业服务业	中	中	中	高	
消费休闲	中	低	中	中	
医疗机构	低	高	低	高	
交通运输	中	中	低	高	
传媒	高	中	高	高	
公用事业	中	中	低	高	
建筑	低	高	低	中	
资源行业	中	中	低	高	
政府	高	中	高	高	
教育	高	中	高	中	

大数据技术

- 是在**成本可承受** (*economically*) 的条件下, 通过**非常快速** (*velocity*) 的采集、发现和分析, 从**大量** (*volumes*)、**多类别** (*variety*) 的数据中**提取价值** (*value*) 的技术, 将是IT领域新一代的技术与架构
- 有很多方法和技术能够对大数据进行聚合、分析等操作, 这些方法和技术来自统计学、应用数学、计算机科学、经济学等多个领域, 一部分是用来处理传统数据的, 通过扩展来处理大数据; 另一部分是最近专门用来处理大数据的

大数据相关研究

- 大数据研究围绕以下方面
 - 数据生成与获取
 - 数据存储
 - 数据处理
 - 数据应用（搜索、分析、挖掘等）

研究热点技术

- 基础软件层面

- 非结构化数据的分布式存储与并行处理
 - Hadoop
 - MapReduce
- 非结构化数据管理系统 (NoSQL)
- 云计算架构

- 应用软件层面

- 数据分析与挖掘
- 数据仓库
- 商业智能