

本章要求：

- 1、了解数据管理的发展过程
- 2、掌握数据库系统的基本概念和主要特点
- 3、掌握数据库系统的三级模式结构和数据库系统的组成
- 4、掌握实体、记录等有关概念和三种数据模型

本章内容：

§ 1 数据库系统概述

§ 2 数据模型

§ 3 DBS的结构

§ 4 数据库系统的组成

请选择内容

返回

§ 1 数据库系统概述

一、基本概念

- 1、**数据**：描述事务的符号记录。可用文字、图形等多种形式表示，经数字化处理后可存入计算机。
- 2、**数据库（DB）**：按一定的数据模型组织、描述和存储在计算机内的、有组织的、可共享的数据集合。
- 3、**数据库管理系统（DBMS）**：位于用户和操作系统之间的一层数据管理软件。主要功能包括：
 - 数据定义功能**：DBMS提供DDL，用户通过它定义数据对象。
 - 数据操纵功能**：DBMS提供DML，用户通过它实现对数据库的查询、插入、删除和修改等操作。

数据库的运行管理：DBMS对数据库的建立、运用和维护进行统一管理、统一控制，以保证数据的安全性、完整性、并发控制及故障恢复。

数据库的建立和维护功能：数据库初始数据的输入、转换，数据库的转储、恢复、重新组织及性能监视与分析等。

4、**数据库系统（DBS）：**计算机中引入数据库后的系统，包括

数据库DB

数据库管理系统DBMS

应用系统

数据库管理员DBA和用户

二、数据管理与数据处理

1、数据管理：

对数据收集、整理、组织、存储、维护、检索、传送等

对象

操作

目标：在适当的时候以适当的形式给适当的人提供适当的数据。

2、数据处理：对数据进行加工、计算、提炼，
从而产生新的有效数据的过程

数据

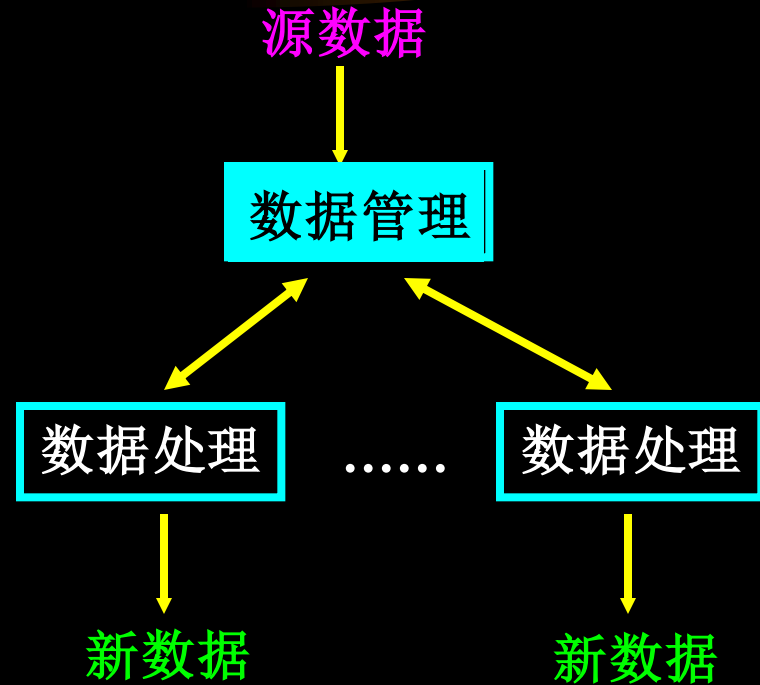


信息

3、管理与处理的关系：

{ 管理是处理的基础
处理为管理服务

管理和处理又可看成一个问题的两个阶段，故可以统一起来，其中心是管理



4、数据管理的任务：两个方面

描述对象：给出数据的定义、组织、存储的描述

描述操作：维护、检索、传送的技术实现

确保在这种描述下
空间利用率高、
操作效率高

确保操作的
正确、安全、高效

三、数据管理的发展阶段

✧ 人工管理阶段（50年代中期以前）

✧ 文件系统阶段（50年代中期至60年代后期）

✧ 数据库系统阶段（60年代后期以后）

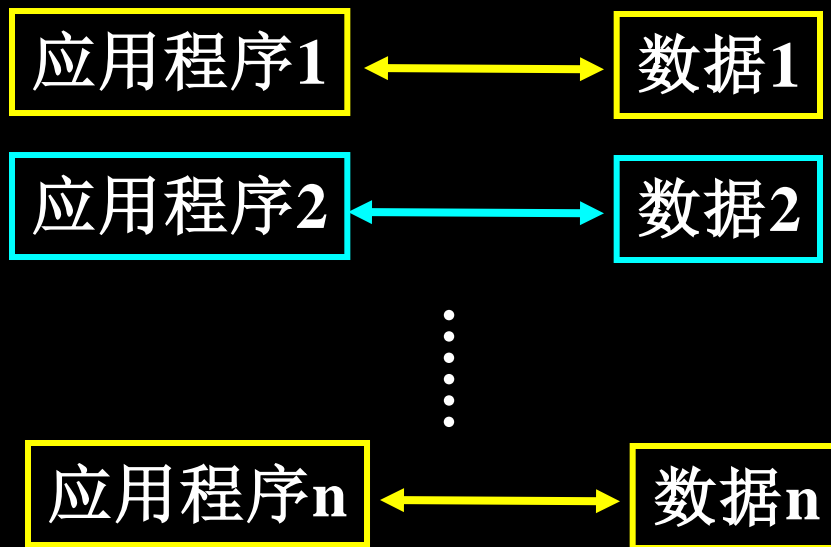
1、人工管理阶段（程序员管理阶段）

特点：🕒 数据不保存

🕒 程序员负责数据管理的一切工作

🕒 数据和程序一一对应，没有独立性和共享性

数据和程序的关系：



2、文件系统阶段

基础 { **硬件**: 有了大容量直接存储外存设备, 如磁盘、磁鼓等
软件: 有了专门的数据管理软件--文件系统
处理方式: 有批处理、联机实时处理等

又可分为两个阶段

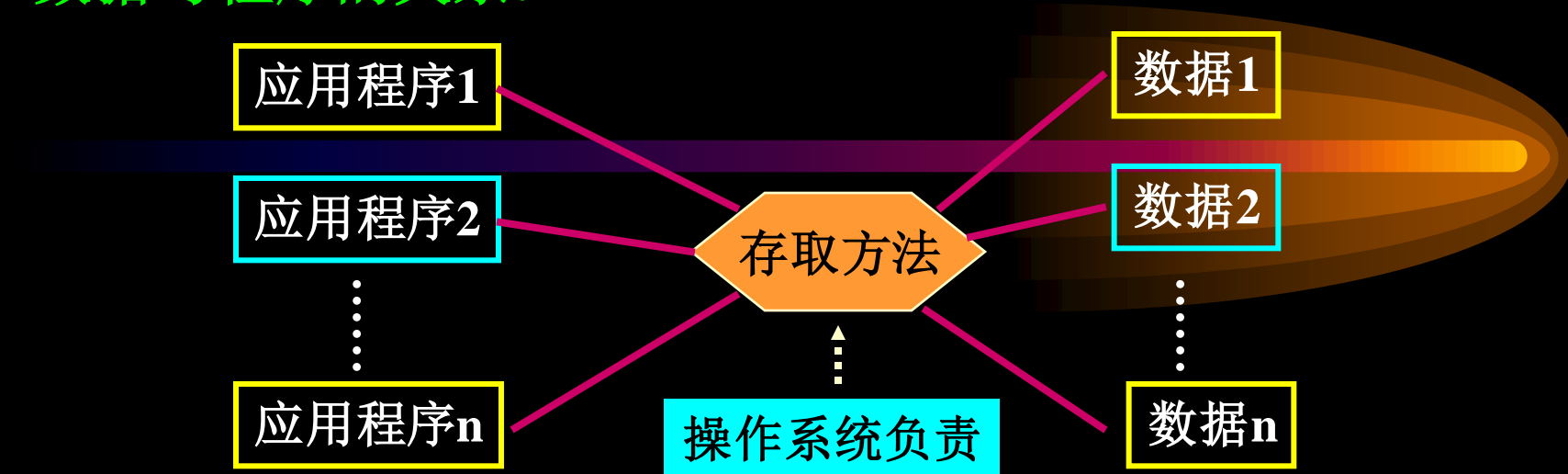
(1) 60年代初期出现了初等的文件系统

主要特点: ☐ **组织方式**: 顺序文件
☐ **数据结构**: 物理结构 = 逻辑结构
☐ **软件功能**: 仅有简单I/O操作

(2) 60年代中期出现了成熟的文件系统

主要特点: ☒ **组织方式**: 顺序和随机存取并用
☒ **数据结构**: 物理结构和逻辑结构有了简单的变换
☒ **软件功能**: 软件系统提供了存取方法

数据与程序的关系:



三个主要缺点:

- ❑ **数据高度冗余:** 数据基本上还是面向应用或特定用户的。
- ❑ **数据共享困难:** 文件基本上是私有的, 只能提供很弱的文件级共享
- ❑ **数据和程序缺乏独立性:** 只有一定的物理独立性, 完全没有逻辑独立性。

3、数据库系统阶段

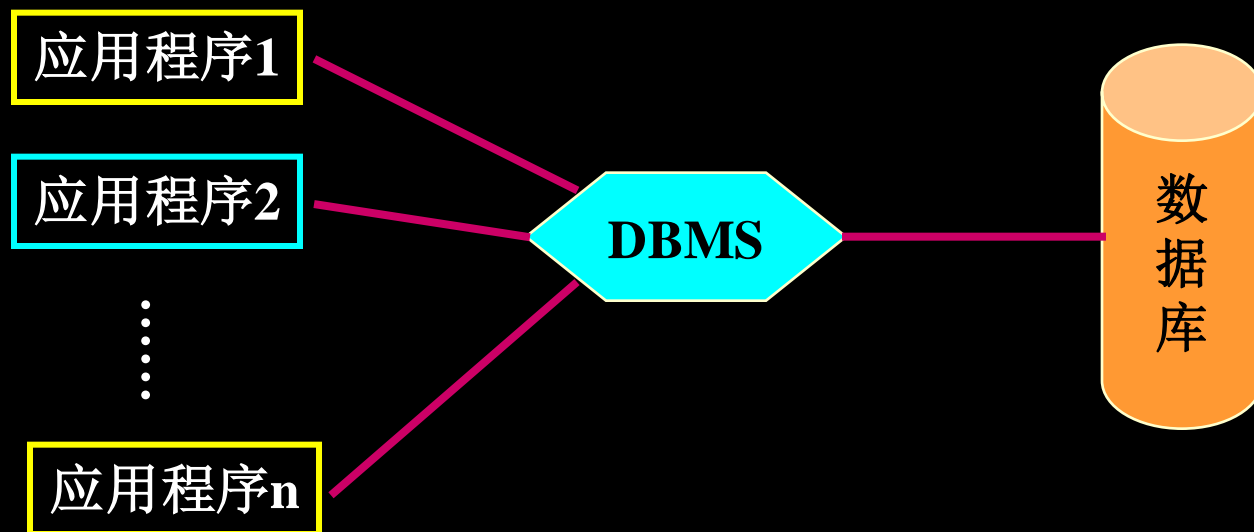
文件系统不能适应大数据量、多应用共享数据的根本原因：

数据没有集中管理

数据库方法的基本出发点：

把数据统一管理、控制，共享使用

数据与程序的关系：



第一章 绪论

- (1) 数据高度结构化集成，面向全组织
- (2) 数据共享性好。可为多个不同的用户共同使用
- (3) 数据冗余少，易扩充
- (4) 数据和程序的独立性高

主要优点

物理独立性：存储结构变，逻辑结构可以不变，从而应用程序也不必改变。

逻辑独立性：总体逻辑结构变，局部逻辑结构可以不变，从而应用程序也不必改变。

好处：简化应用程序的编写和维护

- (5) 数据控制统一

安全性控制：防止泄密和破坏

完整性控制：正确、有效、相容

并发控制：多用户并发操作的协调控制

故障恢复：发生故障时，将数据库恢复到正确状态

对数据库的进一步理解

数据库是以一定的数据结构形式存储在一起的、相互有关的、具有“一少三性”特点的数据集合。

“一少”是指数据冗余少；

“三性”是指：

数据的共享性：库中数据能为多个用户服务

数据的独立性：用户的应用程序及数据的逻辑组织和数据的物理存储方式无关

数据的完整性：数据在新陈代谢活动中始终保持正确性

上述定义是从数据库的外部特征来描述的

第一章 绪论

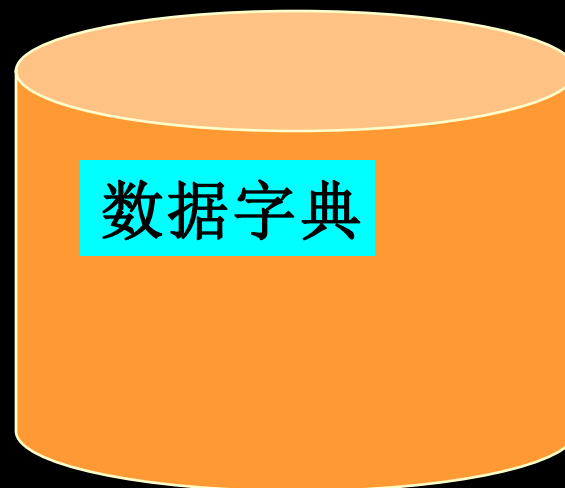
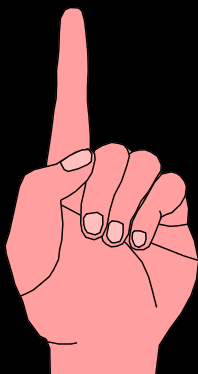
从内部特征来看，可从下述几个方面来描述数据库：

(1) **数据库是自描述的**：它除了包含用户的数据外，

还包含关于它本身结构的描述

称作数据字典，或数据目录，或元数据

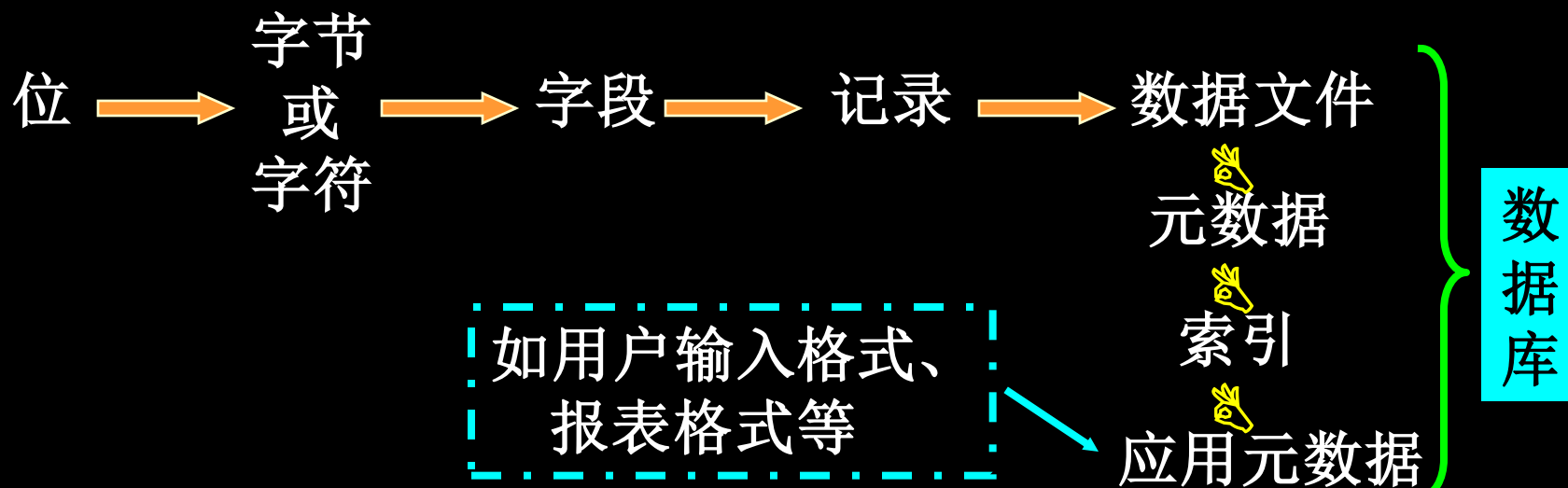
直接的好处：提高了程序和数据独立性



(2) 数据库是集成记录的集合
文件的数据元层次:



数据库的数据元层次:



4、各个阶段的比较：

从四个方面

	人工管理	文件系统	数据库系统
谁管理数据	程序员	操作系统提供存取方法	系统集中管理
面向谁	特定应用	基本上是特定用户	面向系统
共享性	不能	共享很弱	充分共享
数据独立性	没有	一定的物理独立性	较高的独立性

文件系统和数据库系统的本质区别：

内部：数据库的数据是结构化的，有联系的
文件系统的各记录无联系

外部：数据库系统是共享的

文件系统基本上是面向特定用户的

四、数据库技术的发展

三个事件为标志

- 1、1969 IBM IMS 层次模型
- 2、1970年前后, CODASYL DBTG报告 网状模型
- 3、1970 IBM E. F. Codd研究员发表了一系列论文, 奠定了关系数据库的基础

Codd为1981年ACM (Association for Computing Machinery) Turing奖得主。

70年代称为数据库时代

80年代开始, 几乎成为关系数据库的一统天下

数据库的分代:

第一代: 层次、网状数据库

第二代: 关系数据库 →

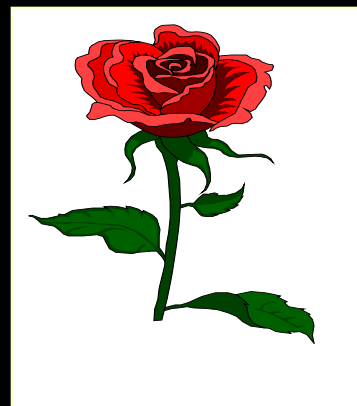
DBASE, FoxBASE, FoxPro
ORACLE, SYBASE, Informix,
Access, DB2, SQLServer

第三代：以面向对象模型为主要特征的新一代数据库系统
技术尚未成熟。

应具备的三个基本特征：

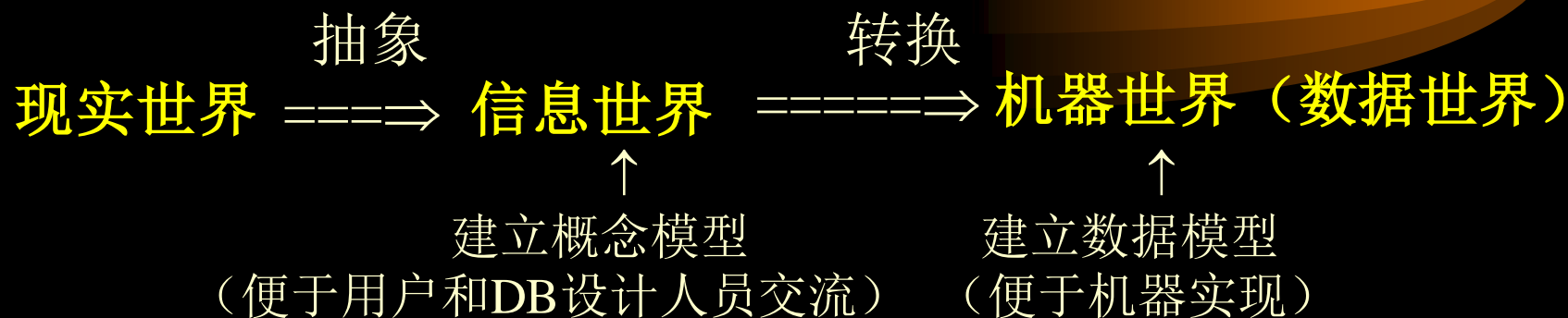
- ☆ 支持数据管理、对象管理、知识管理
- 🕒 保持或继承第二代数据库系统的技术
- 🕒 必须对其它系统开放：支持数据库语言标准，在网络上支持标准网络协议。系统具有良好的可移植性、可连接性、可扩展性和互操作性

Jasmine: 号称是完全面向对象的，
是第一个和唯一专门为充分利用
Internet优势而创造的开发环境。



§ 2 数据模型

数据处理的抽象过程（涉及三个领域）



一、概念模型(信息模型)

把现实世界中的客观对象抽象成的某种信息结构，主要用于数据库设计。

☆ 独立于具体的计算机系统

⌚ 独立于具体的DBMS支持的数据模型

1、实体与记录

信息世界

实体：客观存在并可相互区分的事物。

实体集：性质相同的同类实体的集合。

属性：实体具有的某一特性。

实体标识符：能将一个实体与其它实体区分开来的一个或一组属性。

数据世界

记录 \longleftrightarrow 实体（抽象表示）

文件 \longleftrightarrow 实体集

字段或数据项 \longleftrightarrow 属性

关键字 \longleftrightarrow 实体标识符。唯一地标识一个记录。
又称**码**、**键**。

2、型与值

在DBS中，每一个对象广义上讲都有型与值之分：

型是对象的结构或特性描述，

值是一个具体的对象实例。

类似于程序设计语言中**数据类型**与**数据值**的概念。

(1) 实体型：对实体固有特性或结构的描述。

用实体名及其属性名集合来抽象和刻画。

如 汽车（车牌号，车型，车主）

实体值：实体型的一个实例，即一个具体的实体。

如 （豫A00001，丰田，张三）

(2) 记录型：记录格式。

记录值：一个具体的记录。

如：

车牌号	名称	车主
-----	----	----

豫A00001	丰田	张三
---------	----	----

(3) 几点说明

- 区分型与值的实质
- DBS中讨论的重点是型
- 通常只说实体、记录，含义根据上下文自明

3、实体间的联系

♣ 实体内部的联系（属性间的联系）：

反映在数据上就是记录内部数据项间的联系

♣ 实体之间的联系：

反映在数据上就是记录之间的联系

实体之间的联系可归结为三类：

(1) **1对1联系 (1: 1)**：两个实体集中的每一个实体至多和另一个实体集中的一个实体有联系。

如 国家 —— 总统
学员队 —— 队长

(2) **1对多联系 (1: n)**：若实体集A中的每个实体与实体集B中0个或多个实体有联系，而B中每个实体至多与A中的一个实体有联系，则称从A到B为1对多的联系。

如 国家 —— 部长
学员队 —— 学员

(3) **多对多联系 (m: n)**：两个实体集中的每一个实体都和另一个实体集中0个或多个实体有联系。

如 学员 —— 课程

DBS的核心问题之一：

如何表示和处理实体及实体间的联系。

4、概念模型的表示方法之一：

实体—联系方法（Entity-Relationship Approach）

用E—R图（Entity-Relationship Diagram）描述：

⌚ 实体型：用长方形表示

联系：用菱形表示

属性：用椭圆形表示

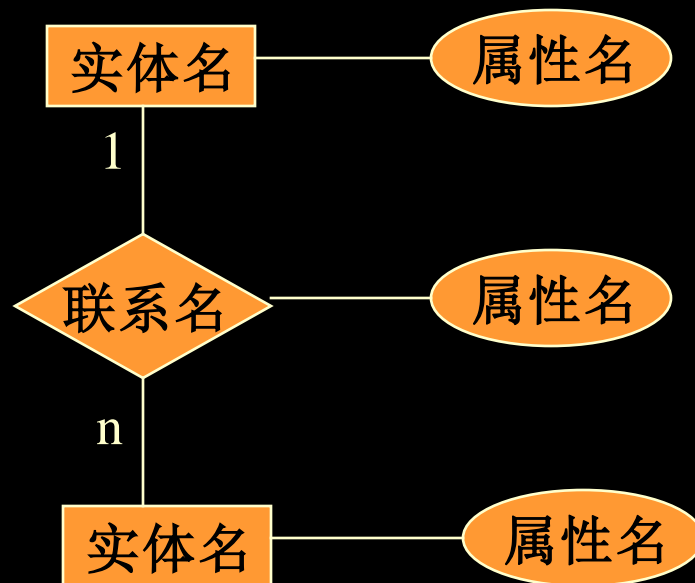
⌚ 框内写上相应的名称

⌚ 用无向边连接：

实体与其属性

联系与其属性

联系与有关实体，并标上联系类型



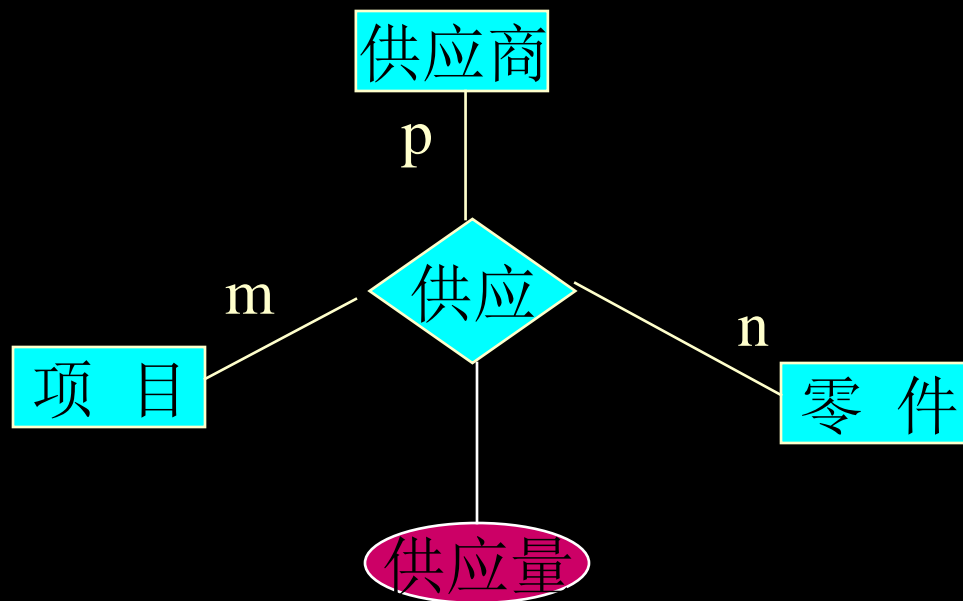
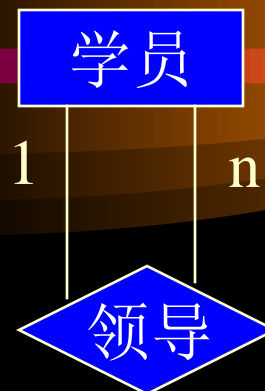
说明:

➤ 联系也必须命名

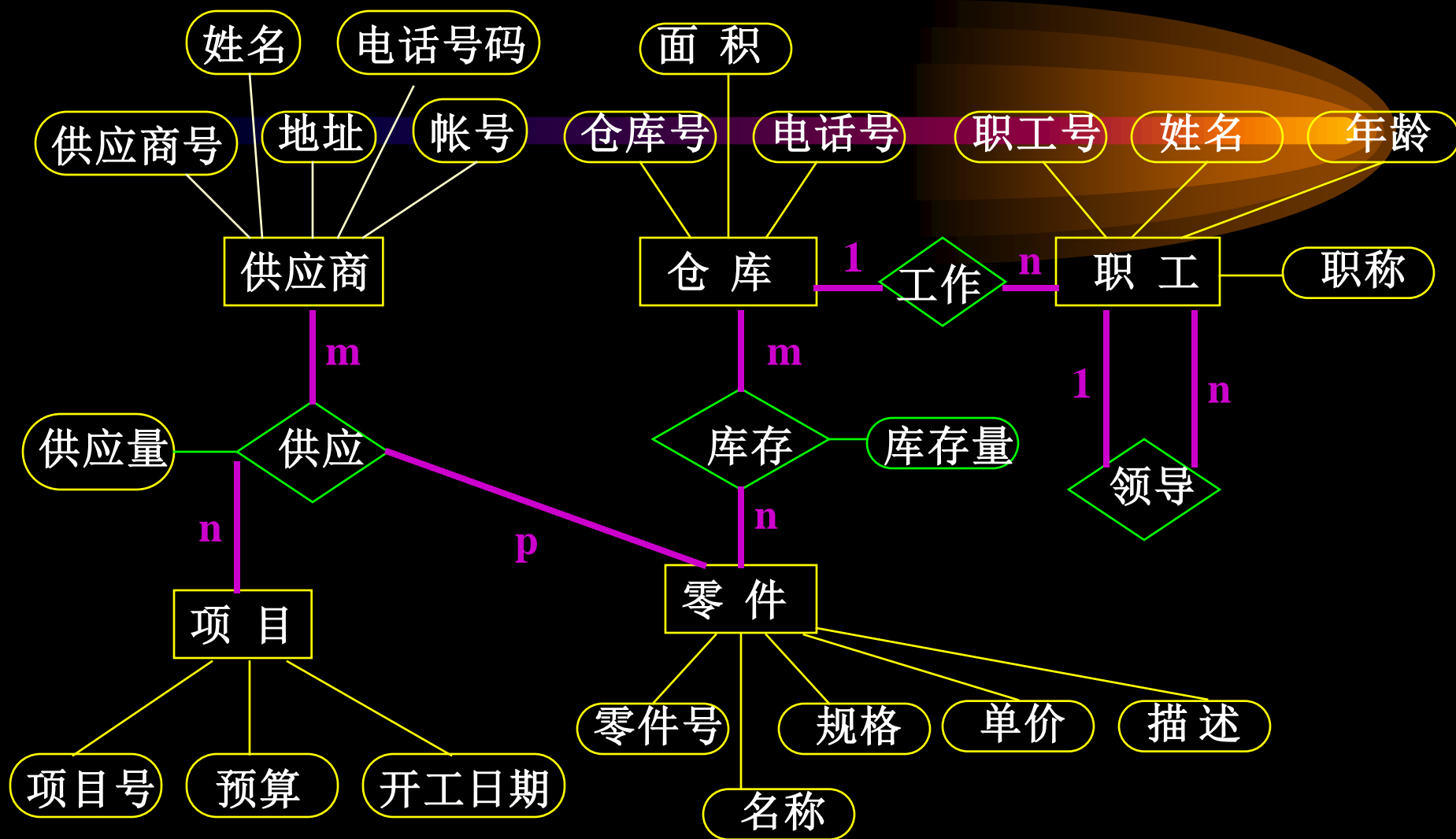
✿ 多个实体之间也可以有联系

单个实体之间也可以有联系

✿ 联系也可以有属性



例：某工厂物资管理E--R图（P20）



二、数据模型

是对现实世界进行抽象的工具，它按计算机系统的观点对数据建模，用于提供数据库系统中信息表示和操作手段的形式框架，主要用于DBMS的实现，是数据库系统的核心和基础。

1、常用的数据模型

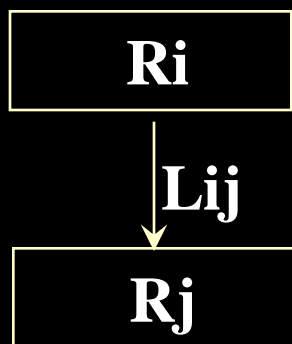
层次模型

网状模型

关系模型

面向对象模型

称作非关系模型，是下列基本层次联系的集合



R_i, R_j 是实体型（记录型）

L_{ij} 是从 R_i 到 R_j 的1: 1或1: n 联系

2、数据模型 的

形式化描述数据、
数据之间的联系
以及数据操作
和有关的语义
约束规则的方法

数据结构（静态）

数据操作（动态）

完整性约束

如何表示
实体及联系

（难点是表示联系）

如何实现
查、增、删、改

如何保证数据的
约束条件得到满足

根据现实世界实体间联系的特征
用四种不同的方法进行抽象

层次模型

网状模型

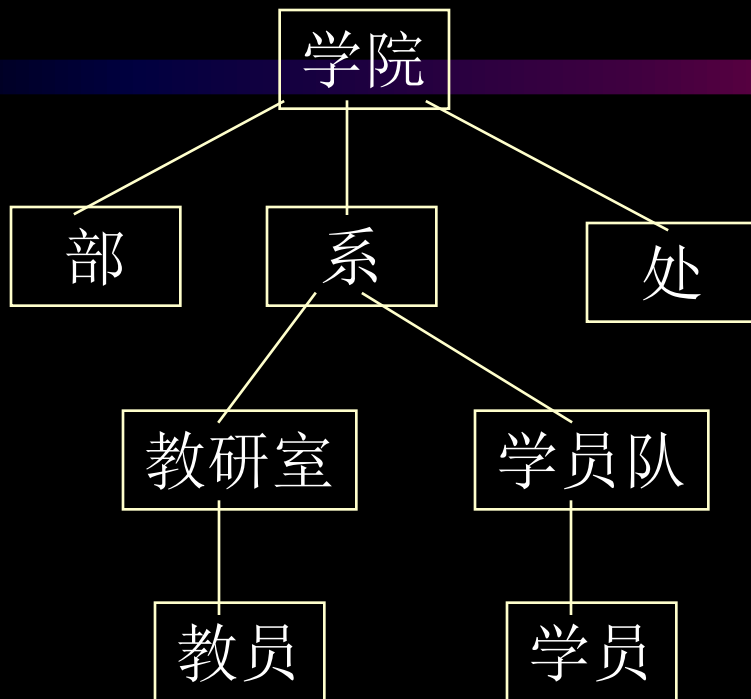
关系模型

面向对象模型

（因此，是按照数据结构的
类型来命名数据模型）

3、层次模型

根据一个单位的组织结构直观地得出



方框表示一个实体型

(结点)

线表示联系

(边)

(1) 定义：用树形结构来表示实体以及实体间联系的模型。

其特征是：(a) 有且仅有一个结点无双亲（根结点）；

(b) 其它结点有且仅有一个双亲。

(2) 说明:

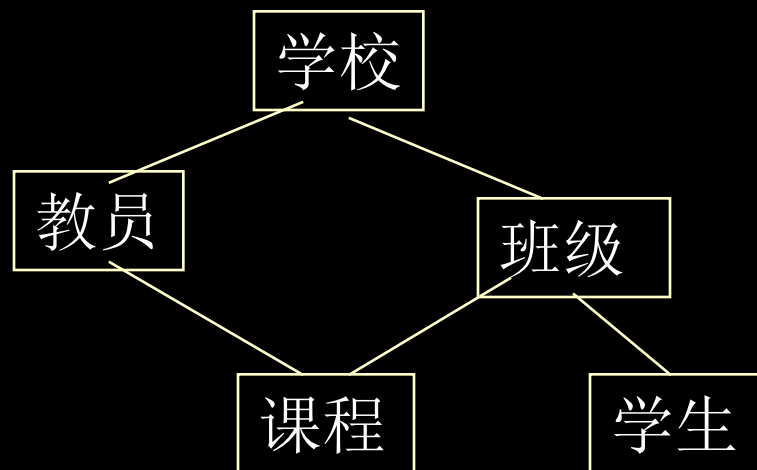
- (a) 树中实体间联系只能是从父到子的1:1或1:n联系, 对m:n联系, 须使用辅助手段转换成多个1:n联系, 但不易掌握
- (b) 简单直观, 结构清晰, 运行效率高, 但编程复杂

4、网状模型

(1) 定义: 用图结构来表示实体以及实体间联系的模型。

其特征是: 任一结点都可以无双亲或有一个以上的双亲。

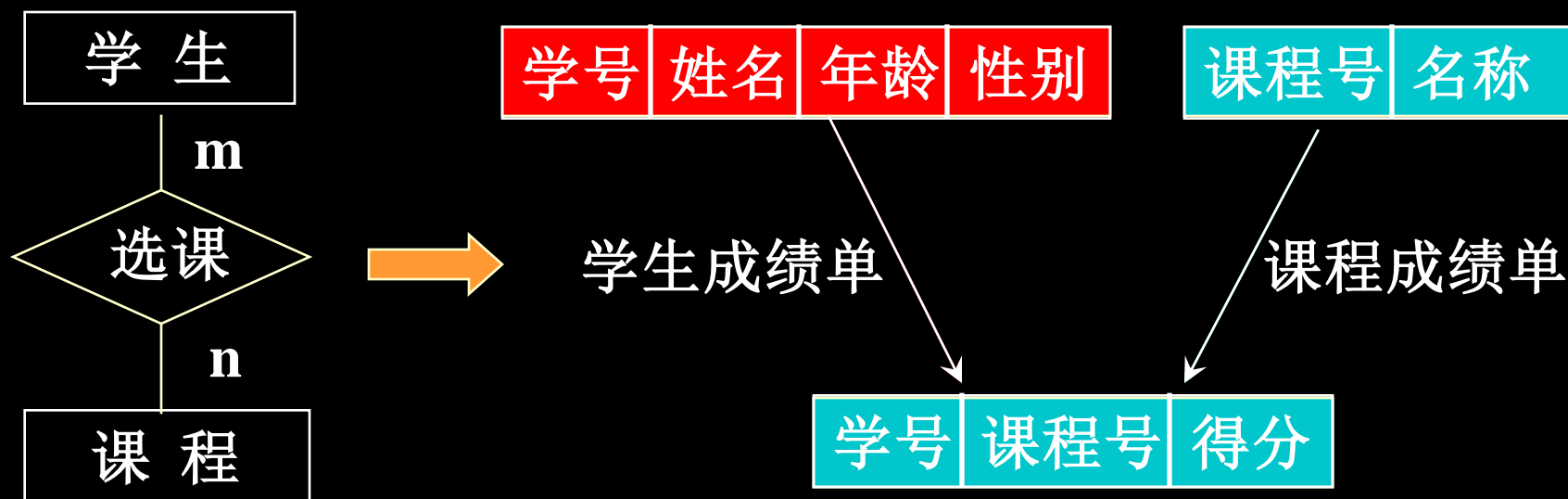
例



- (2) 优：可表示 $m:n$ 的联系，运行效率高
缺：过于复杂，实现困难

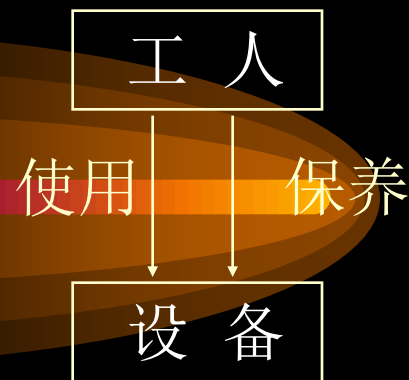
(3) 说明

(a) 即使对网状模型，具体在计算机上实现时， $m:n$ 的联系仍需分解成若干个 $1:n$ 的联系。（因此，网状模型的图结构实质上是有向图），如



(b) 网状模型中允许两结点间有多条边，

层次模型则不允许



5、关系模型

层次、网状模型基本上是面向专业人员的，使用极不方便

问题：寻找一种能面向一般用户的数据模型？

(1) 定义：用二维表（关系）来描述实体及实体间联系的模式。

(2) 示例



供应商S

S#	SNAME	SADDR
S1	张三	北京
S2	李四	郑州
...

零件P

P#	PNAME	PRICE
P1	电机	2000
P2	螺丝	2
...

(联系) 供应SP

S#	P#	QTY
S1	P1	200
S1	P3	22
...

关系： 对应一张表，
每表起一个名称即关系名

元组： 表中的一行

属性： 表中一列，
每列起一个名称即属性名

主码： 唯一确定一个元组的属性组

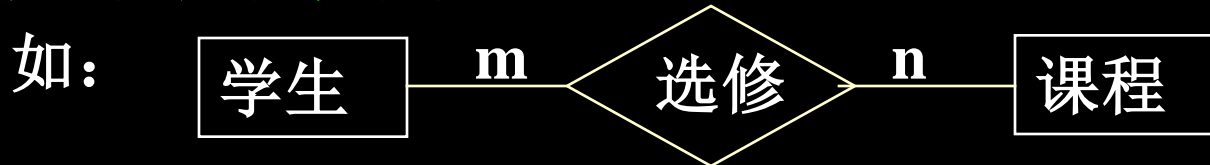
域： 属性的取值范围

(3) 关系模式：对关系的描述，一般表示为：

关系名（属性1，属性2，...，属性n）

(4) 优点：

♠ 无论实体还是实体之间的联系都用统一的数据结构（二维表、关系）来表示，可方便地表示m:n联系，因此概念简单，用户易懂易用



可表示为：

学生（学号，姓名，性别，系和年级）

课程（课程号，课程名，学分）

选修（学号，课程号，成绩）

♠ 表格中行、列次序无关

♠ 有坚实的理论基础（关系理论）

♠ 存取路径对用户透明，用户只需指出“做什么”，不需说明“怎么做”，因此数据独立性更高

缺点：由于存取路径对用户透明，查询效率不够高，必须对查询请求进行优化。

说明：

关系必须规范化，关系的每个分量必须是一个不可分的数据项，不允许表中套表。规范化理论将在后续章节讲解。

(5) 关系模型与非关系模型的比较

	关系模型	非关系模型
实体及实体间联系采用的数据结构	统一 均为关系	不统一
操作方式	一次一集合	一次一记录
存取路径	对用户透明	对用户不透明

§ 3 DBS的结构

三级模式（外模式、模式、内模式）

两级映象（外模式/模式，模式/内模式映象）

一、DBS的三级模式结构

1、模式（Schema）：又称**逻辑模式**。**DB**的全局逻辑结构。

即DB中全体数据的逻辑结构和特征的描述。

说明

- ① 模式只涉及到型的描述，不涉及具体的值（实例），反映的是数据的结构及其联系
- ② 模式不涉及物理存储细节和硬件环境，也与应用程序无关
- ③ 模式承上启下，是DB设计的关键
- ④ DBS提供模式DDL（Data Definition Language）来定义模式（描述DB结构）

⑤ 模式定义的任务（概念模型 \uparrow 模式）

- 定义全局逻辑结构（构成记录的属性名、类型、宽度等）
- 定义有关的安全性、完整性要求
- 定义记录间的联系

⑥ 一个数据库只有一个模式

2、外模式：又称子模式或用户模式。DB的局部逻辑结构。

即与某一应用有关的数据的一个逻辑表示。

说明：

⇒ 外模式是某个用户的数据视图，

模式是所有用户的公共数据视图；

☒ 一个DB只能有一个模式，但可以有多个外模式；

☒ 外模式通常是模式的子集，但可以在结构、类型、长度等方面有差异；

☒ DBS提供外模式DDL。

3、内模式：又称**存储模式**。**数据的物理结构和存储方式的描述**。
即DB中数据的内部表示方式。

说明：

☆ 一个数据库只有一个内模式

🕒 DBS提供内模式DDL；

🌸 内模式定义的任务

——— { 记录存储格式，
索引组织方式，
数据是否压缩、是否加密等。

4、两级映象及其作用

(1) 外模式/模式映象：定义外模式和模式间的对应关系。对应同一个模式可以有多个外模式，对每个外模式都有一个外模式/模式映象。

作用：模式变，可修改映象使外模式保持不变，从而应用程序不必修改，保证了程序和数据的逻辑独立性。

(2) 模式/内模式映象：定义DB全局逻辑结构和存储结构间的对应关系。一个数据库只有一个模式，也只有一个内模式，因此模式/内模式的映象也是唯一的。

作用：存储结构变，可修改映象使逻辑结构（模式）保持不变，从而应用程序不必修改，保证了数据与程序的物理独立性。

§ 4 数据库系统的组成

一、数据库系统（DataBase System，DBS）的组成

广义上讲，DBS就是计算机系统中引进数据库后的构成。

有下面四部分：

1、数据库：一个或多个数据库

数据库的四要素：用户数据、元数据、索引和应用元数据

2、软件

☺ 操作系统：支持DBMS的运行

☺ 数据库管理系统DBMS（DataBase Management System）：

操纵和管理数据库的大型软件系统，是数据库系统的核心

☺ 数据库应用开发工具等辅助软件

☺ 具有数据库接口的高级语言与编译系统，如C、C++等

☺ 某个数据库应用系统

3、人员

用户	应用程序员	数据库管理员DBA
(使用)	(开发)	(管理)

DBA (Data Base administrator) 的职责:

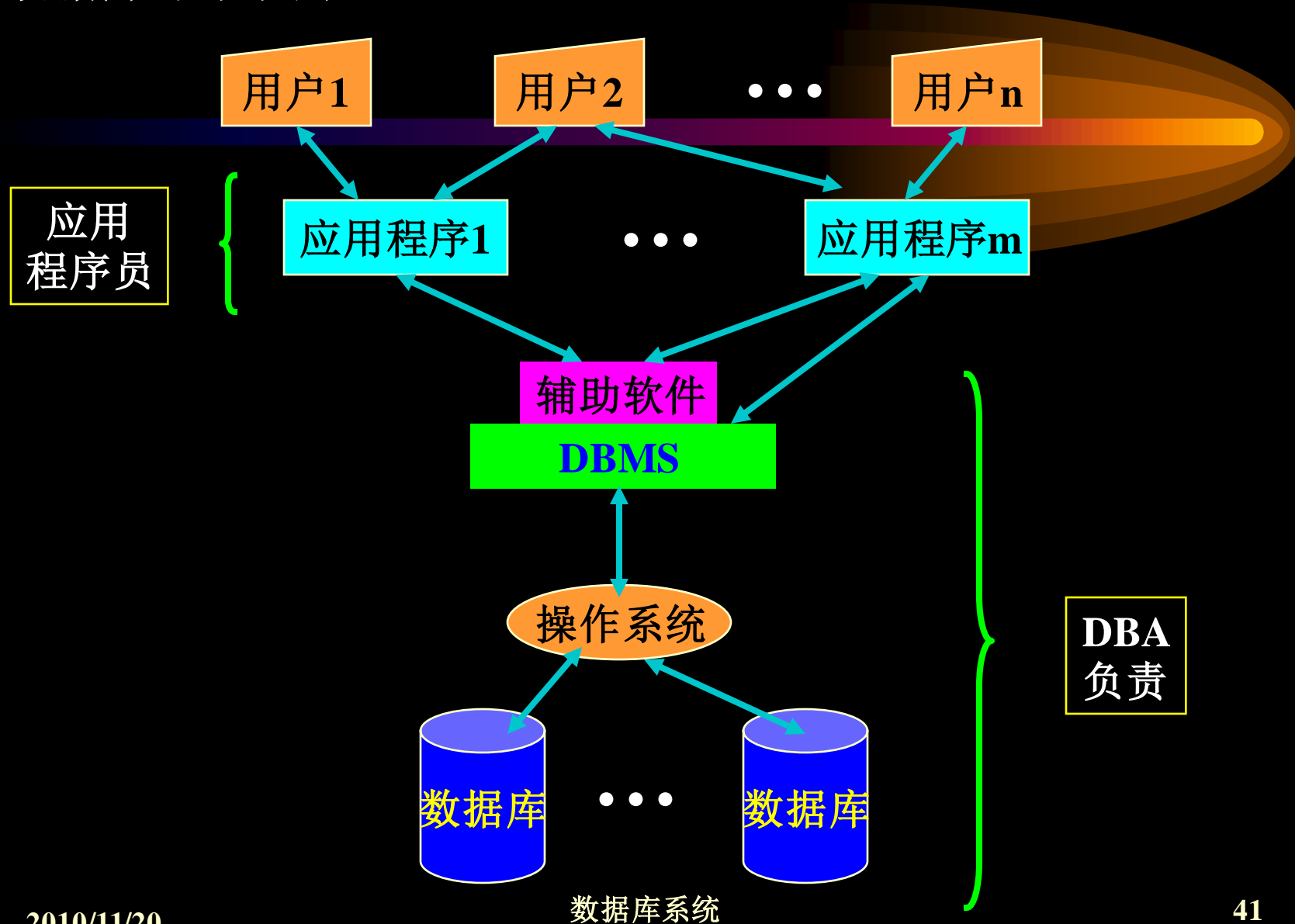
- ① 决定数据库的内容和逻辑结构、存储结构
- ② 确定数据的安全性要求和完整性约束条件
- ③ 监控数据库的使用和运行, 维护数据库
- ④ 决定数据库的存储结构和存储策略
- ⑤ 负责数据库的改进和重组重构

4、硬件

计算机及有关设备, 要求有足够大的内、外存储容量及较高的处理速度。

第一章 绪论

数据库系统图示:



二、数据库系统研究的对象

如何高效巧妙地进行数据管理，而又**花费最少**

三个主要研究领域：

DBMS及其辅助软件

数据库设计

数据库理论

如：占用空间少
查询快
维护方便等

作业：3, 5, 7, 12, 13, 20, 22

本章要求：

- 1、掌握关系、关系模式、关系数据库等基本概念
- 2、掌握关系的三类完整性的含义
- 3、掌握关系代数运算

本章内容：

§ 1 关系模型的基本概念

§ 2 RDBS的数据操纵语言：关系代数

§ 3 RDBS的数据操纵语言：关系演算语言

请选择内容

返回

§ 1 关系模型的基本概念

层次、网状数据库是面向专业人员的，使用很不方便。程序员必须经过良好的培训，对所使用的系统有深入的了解才能用好系统。

关系数据库就是要解决这一问题，使它成为面向用户的系统。

关系数据库是应用数学方法来处理数据的。它具有结构简单、理论基础坚实、数据独立性高以及提供非过程性语言等优点。

一、关系的数学定义

1、**域 (Domain)**：值的集合。它们具有相同的数据类型，语义上通常指某一对象的取值范围。

例如：全体整数，

0到100之间的整数，

长度不超过10的字符串集合

2、**笛卡尔积 (Cartesian Product)**：设 D_1 、 D_2 、...、 D_n 是 n 个域,则它们的笛卡尔积为

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

其中每一个元素称为一个 n 元组(n -tuple),简称**元组**;

元组中的每个值 d_i 称为一个**分量**(component).

笛卡尔积可以写成一个二维表

例如：

设 $D1=\{\text{张三}, \text{李四}\}$,

$D2=\{\text{数学}, \text{语文}\}$,

$D3=\{\text{优}, \text{良}\}$

则 $D1 \times D2 \times D3$ 可用二维表表示为：

张三	数学	优
张三	数学	良
张三	语文	优
张三	语文	良
李四	数学	优
李四	数学	良
李四	语文	优
李四	语文	良

3、关系 (Relation)

笛卡尔积 $D1 \times D2 \times \dots \times Dn$ 的子集合，

记作 $R (D1, D2, \dots, Dn)$

关系名

n 为关系的目或度

4、说明

- ◎ 关系是一个二维表。
- ◎ 每行对应一个元组。
- ◎ 每列可起一个名字，称为属性。属性的取值范围为一个域，元组中的一个属性值是一个分量。

5、关系的性质

- ☆ 列是同质的，即每列中的数据必须来自同一个域
- 🕒 每一列必须是不可再分的数据项（不允许表中套表，即满足第一范式）
- 🕒 不能有相同的行
- 🕒 行、列次序无关

二、关系模型

三部分：关系数据结构、关系操作集合、关系的完整性

(一) 数据结构

1、单一的数据结构：关系（二维表）

不论是实体还是实体间的联系都用关系表示。

实体值 → 关系的元组，在关系数据库中通常称为记录

属性值 → 元组的分量，在关系数据库中通常称为字段

关键字（码）：唯一标识一个元组的属性组

关键字可以有多个，统称候选关键字。在使用时，通常选定一个作为主关键字。主关键字的诸属性称为主属性，其它为非主属性。

第二章 关系数据库

2、关系模式：关系的描述。

包括关系名、诸属性名、属性向域的映象、属性间的依赖。

关系的型

属性的类型、长度等

一个元组为关系的一个值

表示： $R(U, D, dom, F)$

关系数据库模式：对关系数据库的描述，包括域的定义及在域上定义的所有关系模式。

型

关系数据库：所有实体及实体间联系的关系的集合。是某时刻所有关系模式对应的关系的集合。

值

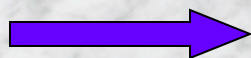
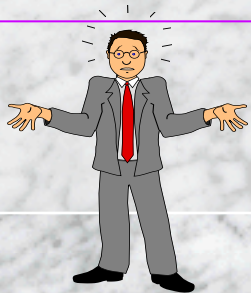
3、关系的三种类型

基本关系：客观存在的基本表

查询表：由基本表按一定条件检索得到的结果

视图（View）：从一个或多个基本关系上导出的关系。它不对应实际的存储数据，是一个虚关系，然而可永久存在。相当于关系模型的外模式。

由于二维表的存储策略非常简单，关于数据库的物理存储完全由DBMS自动完成。因此，在关系模型中不需要与内模式相应的概念。



关系简单吗？

(二) 关系操作

1、种类：选择、投影、连接、除、并、交、差
增加、删除、修改

维护操作

2、特点：

⌚ 集合操作，一次操作
可存取多个元组

一次一集合（关系型）
一次一记录（非关系型）

⌚ 非过程化语言：用户只需告诉做什么（What）
不需告诉怎么做（How）

⌚ 数据定义、数据操纵、数据控制语言集成在一起

DDL

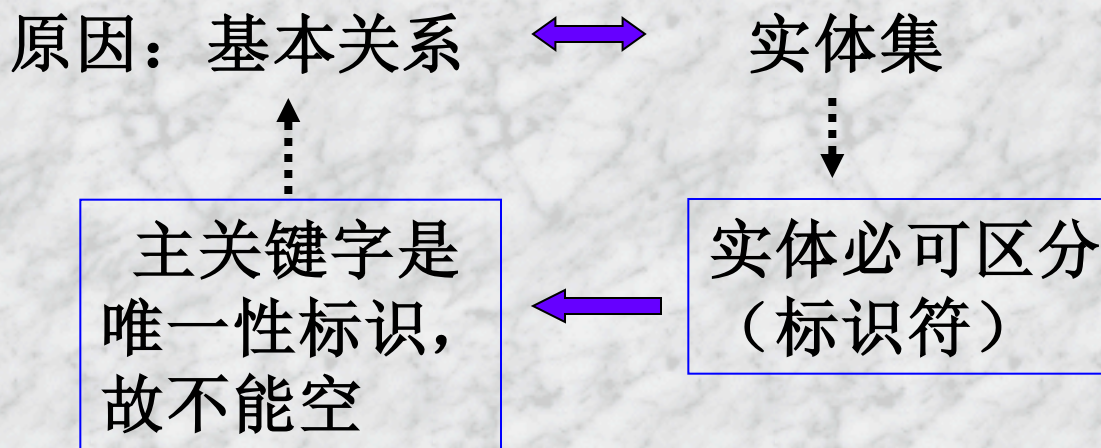
DML

DCL：权限控制、完整性控制等

(三) 关系模型的三类完整性

1、实体完整性 (Entity Integrity)

基本关系的所有主属性不能取空值



2、参照完整性 (Referential Integrity)，也叫引用完整性

若基本关系R含有与另一个基本关系S的主关键字相对应的属性组F (F称为R的**外键**或**外部码**)，则R中每个元组在F上的值或为空值，或等于S中某个元组的主关键字值

第二章 关系数据库

例：职工关系 EMP (ENO, ENAME, **DNO**)

部门关系 DEPT (**DNO**, DNAME)

DEPT的主键

EMP的外键，只能取空值或DEPT中某关键字的值

又如：学生关系 (**SNO**, SNAME, AGE, SEX)

课程关系 (**CNO**, CNAME)

选课关系 (**SNO**, **CNO**, G)

3、用户定义的完整性

用户定义的某一属性值必须满足的语义要求。

一经定义，DBMS会自动检查，从而不必在应用程序中作检查。

本节开头

本章开头

下一节

§ 2 RDBS的数据操纵语言：关系代数

关系代数的运算对象是关系，运算结果也为关系。其运算按运算符的不同可分为两类。

一、传统的集合运算

1、并（Union）： $R \cup S = \{ t \mid t \in R \vee t \in S \}$

2、交（Intersection）： $R \cap S = \{ t \mid t \in R \wedge t \in S \}$

3、差（Difference）： $R - S = \{ t \mid t \in R \wedge t \notin S \}$

4、笛卡尔积（广义）： $R \times S = \{ \widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \}$

二、专门的关系运算

从行的角度的运算

1、选择 (Selection)，又称限制 (Restriction)

$\sigma_F(R)$: 在关系 **R** 中选出满足条件 **F** 的诸元组形成一个新关系。

条件表达式

2、投影 (Projection)

从列的角度的运算

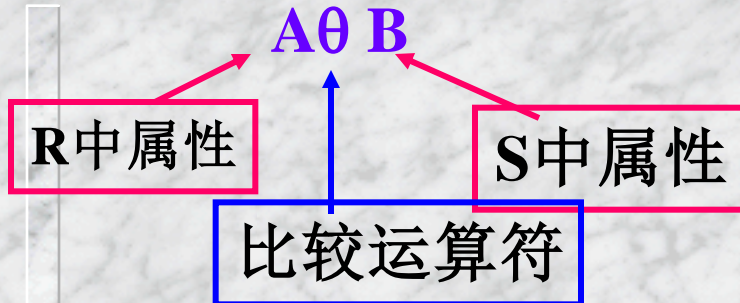
$\pi_A(R)$: 在 **R** 中选出若干属性列组成一个新关系。

属性组

投影后若有重复行，则自动保留一个

3、连接 (Join)

$R \bowtie S$: 从两个关系的笛卡尔积中选取属性间满足条件 $A \theta B$ 的元组。



连接是同时处理
多个关系的
重要运算

说明:

- ◆ $R \bowtie_{A \theta B} S = \sigma_{A \theta B} (R \times S)$

- ◆ 当 θ 为等号且 A 、 B 两属性相同时, 称为自然连接, 记作 $R \bowtie S$

- ◆ 自然连接将去掉重复属性

若仅有 θ 为等号的条件, 称为等值连接

4、除 (Division)

$R(X, Y) \div S(Y, Z)$: 把R按X的值分组, 若某一组中属性组Y的值包含S在Y上投影的全部元组, 则该X的值作为商关系的一个元组

例: 求至少选修C1、C3课程的学生号码

设一临时关系K:

C#
C1
C3

关系代数表达式

$$\pi_{S\#, C\#}(SC) \div K = \{S1, \dots\}$$

SC:

S#	C#	G
S1	C1	A
S1	C2	A
S1	C3	A
S1	C5	B
S2	C1	B
S2	C2	C
S2	C4	C
S3	C2	B
S3	C3	C
S3	C4	B
S4	C1	B
S4	C3	A

4、除 (Division)

$R(X, Y) \div S(Y, Z)$: 把R按X的值分组, 若某一组中属性组Y的值包含S在Y上投影的全部元组, 则该X的值作为商关系的一个元组

属性组

例: 求至少选修C1、C3课程的学生号码

设一临时关系K:

C#
C1
C3

关系代数表达式

$$\pi_{S\#, C\#}(SC) \div K = \{S1, \dots\}$$

SC:

S#	C#	G
S1	C1	A
S1	C2	A
S1	C3	A
S1	C5	B
S2	C1	B
S2	C2	C
S2	C4	C
S3	C2	B
S3	C3	C
S3	C4	B
S4	C1	B
S4	C3	A

按S2分组



S#	C#	G
S1	C1	A
S1	C2	A
S1	C3	A
S1	C5	B
S2	C1	B
S2	C2	C
S2	C4	C
S3	C2	B
S3	C3	C
S3	C4	B
S4	C1	B
S4	C3	A

4、除 (Division)

$R(X, Y) \div S(Y, Z)$: 把R按X的值分组, 若某一组中属性组Y的值包含S在Y上投影的全部元组, 则该X的值作为商关系的一个元组

属性组

例: 求至少选修C1、C3课程的学生号码

设一临时关系K:

C#
C1
C3

关系代数表达式

$$\pi_{S\#, C\#}(SC) \div K = \{S1, \dots\}$$

SC: S# C# G

S1	C1	A
S1	C2	A
S1	C3	A
S1	C5	B
S2	C1	B
S2	C2	C
S2	C4	C
S3	C2	B
S3	C3	C
S3	C4	B
S4	C1	B
S4	C3	A

按S3分组



4、除 (Division)

$R(X, Y) \div S(Y, Z)$: 把R按X的值分组, 若某一组中属性组Y的值包含S在Y上投影的全部元组, 则该X的值作为商关系的一个元组

属性组

例: 求至少选修C1、C3课程的学生号码

设一临时关系K:

C#
C1
C3

关系代数表达式

$$\pi_{S\#, C\#}(SC) \div K = \{S1, S4\}$$

SC: S# C# G

S1	C1	A
S1	C2	A
S1	C3	A
S1	C5	B
S2	C1	B
S2	C2	C
S2	C4	C
S3	C2	B
S3	C3	C
S3	C4	B
S4	C1	B
S4	C3	A

按S4分组

第二章 关系数据库

三、关系代数运算举例

求至少选修这样一

门课的学生姓名，这门
课的直接先行课是C2

① 先找出先行课为C2的课程号：

$\sigma_{PC\#='C2'}(C)$ ，记为PC

② 找选修该类课程的学生学号：

$PC \bowtie \pi_{S\#, C\#}(SC)$

记为PCS

③ 找出学生姓名：

$\pi_{SN} (PCS \bowtie \pi_{S\#, SN}(S))$

S: S# SN SD SA

S1	A	CS	20
S2	B	CS	21
S3	C	MA	19
S4	D	CI	19
S5	E	MA	20
S6	F	CS	22

C: C# CN PC#

C1	G	
C2	H	C1
C3	I	C2
C4	J	C2
C5	K	C4

SC: S# C# G

S1	C1	A
S1	C2	A
S1	C3	A
S1	C5	B
S2	C1	B
S2	C2	C
S2	C4	C
S3	C2	B
S3	C3	C
S3	C4	B
S4	C2	B
S4	C5	D
S5	C2	C
S5	C3	B
S5	C5	B
S6	C1	A
S6	C5	A



最终的关系代数表达式:

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

说明:

PC

PCS

用关系代数表示查询时, 若查询涉及多个关系, 需用连接操作实现; 若查询诸如“选修了全部课程”的学生、“使用了全部零件”的工程等, 需用除法操作实现。

作业: 1, 4, 5, 6

本节开头

本章开头

下一节

§ 3 RDBS的数据操纵语言：关系演算语言

关系演算：基于谓词演算

按谓词变量
的特征划分

面向元组：谓词变量的获得值是关系中的元组
(元组变量)

面向域：谓词变量的获得值是关系中某属性的值
(域变量)

一、元组关系演算

1、元组关系演算表达式：

t为元组变量

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

公式

运算的结果
还是一个关系

2、原子公式

- ✉ $R(t)$: 表示 t 是关系 R 中的一个元组
- ✉ $t[i]\theta u[j]$: 表示 t 的第 i 个分量和 u 的第 j 个分量满足比较关系 θ
- ✉ $t[i]\theta C$ 或 $C\theta t[i]$: 含义同上,只不过 C 为常量

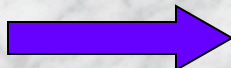
3、公式的递归定义

- (1)每个原子公式是一个公式;
- (2)设 ϕ_1 、 ϕ_2 是公式, 则 $\neg \phi_1$ 、 $\phi_1 \wedge \phi_2$ 、 $\phi_1 \vee \phi_2$ 也是公式;
- (3)设 ϕ 是公式, t 是元组变量, 则 $(\exists t)\phi$ 、 $(\forall t)\phi$ 也是公式;
- (4)除此之外没有其它形式的公式。

4、关系代数运算均可用关系演算来表示，反之亦然
见教材P71。

5、用关系演算来表达查询

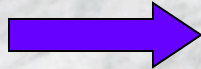
例1，求年龄大于或等于20的学生：

$$S_{20} = \{ t \mid S(t) \wedge t[4] \geq 20 \}$$


S: S# SN SD SA

S1	A	CS	20
S2	B	CS	21
S3	C	MA	19
S4	D	CI	19
S5	E	MA	20
S6	F	CS	22

例2，求学生姓名及所在的系：

$$S1 = \{ t^{(2)} \mid (\exists u)(S(u) \wedge t[1]=u[2] \wedge t[2]=u[3]) \}$$


S1	A	CS	20
S2	B	CS	21
S3	C	MA	19
S4	D	CI	19
S5	E	MA	20
S6	F	CS	22

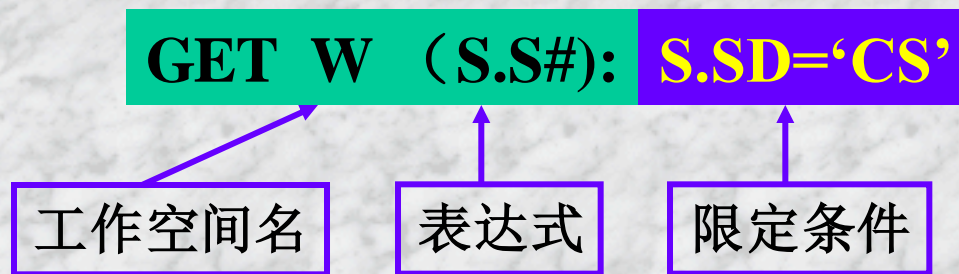
二、未实现的元组关系演算语言——ALPHA

E. F. Codd提出，但并未实现。

1、检索操作（GET）

（1）不设元组变量

例：取出计算机系学生的学号：



二、未实现的元组关系演算语言——ALPHA

E. F. Codd提出，但并未实现。

1、检索操作（GET）

(1) 不设元组变量 (事实上关系名起到元组变量的作用)

例：取出计算机系学生的学号：

GET W (S.S#): S.SD='CS'

↑
相当于投影

↑
相当于原子公式 $t[i]\theta C$

取出一个计算机系学生的学号

GET W (1) (S.S#): S.SD='CS'

↑
定额

(2) 使用元组变量

应用场合

用较短的名字代替较长的关系名
使用量词时

变量范围说明

关系名

元组变量

例 查找不选C1课程的学生姓名

RANGE SC X

GET W (S.SN): $\forall X(X.S\# \neq S.S\# \vee X.C\# \neq 'C1')$

查找选修全部课程的学生姓名

RANGE C CX

RANGE SC SCX

GET W (S.SN): $\forall CX \exists SCX(SCX.S\# = S.S\# \wedge SCX.C\# = CX.C\#)$

2、存储操作

(1) 修改: **UPDATE**

(2) 插入: **PUT**

(3) 删除: **DELETE**

参阅教材P67--P69。

关键字不能修改，
只能先删除、再插入

三、域关系演算

与元组关系演算类似，只不过这里的变量取值范围是属性值，其谓词变元称作欲变量，关系的属性名可视为欲变量。

关系代数、元组关系演算、域关系演算的表达能力是等价的。域关系演算语言——QBE

QBE是Query By Example 的缩写，1978年在IBM370上实现。

1、特点

- 用户通过表格形式提出查询，查询结果也通过表格显示出来
- 用户容易掌握，易学易用

2、使用方法

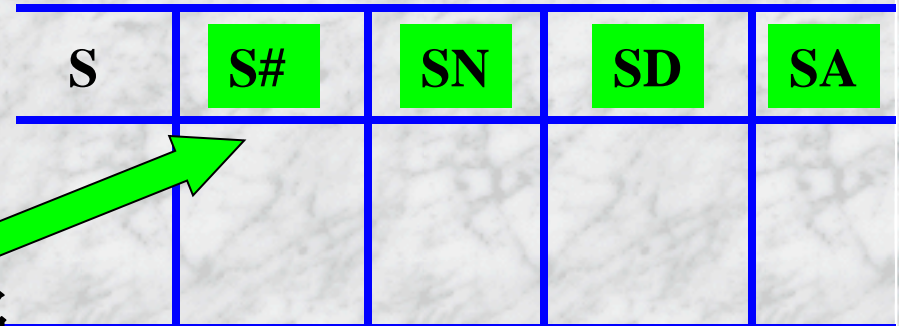
(1) 用户提出使用要求 (如键入某一命令)

(2) 机器显示空白表格

(3) 用户输入关系名

如 学生关系 S

(4) 机器自动显示属性名



S	S#	SN	SD	SA

2、使用方法

(1) 用户提出使用要求 (如键入某一命令)

(2) 机器显示空白表格

(3) 用户输入关系名

如 学生关系 S

(4) 机器自动显示属性名

(5) 提出查询要求

如 查询计算机系
的学生姓名和年龄

查询条件SD='CS'

S	S#	SN	SD	SA
		P.张三	CS	P.30

P.是操作符

示例元素

(任选一个可能的值)

3、其他例子:

(1) 查询操作

例1: 查计算机系年龄大于19的学生姓名

两个条件写两行,
示例元素相同,表示
条件之间是“与”的关系

例2: 查计算机系或年龄
大于19的学生姓名

示例元素不同,表示
条件之间是“或”的关系

S	S#	SN	SD	SA
		P.张三	CS	>19

S	S#	SN	SD	SA
		P.张三 P.张三	CS	>19

S	S#	SN	SD	SA
		P.张三 P.李四	CS	>19

第二章 关系数据库

例3：查选修C2的学生名字

（涉及两个关系，需要连接操作）

SC	S#	C#	G
	<u>S1</u>	C2	

不同关系中的
两个示例元素相同，
表示了连接操作。

S	S#	SN	SD	SA
	<u>S1</u>	P.张三		

(2) 修改操作

修改操作符为“U.”，不允许修改主码，若要修改主码，需先删除元组，再插入。

例1：把学号为S1的学生转入计算机系。

修改操作不包含表达式，可有两种表示方法。

例2：将计算机系所有学生的年龄增加1岁。

S	S#	SN	SD	SA
	S1		U. CS	

S	S#	SN	SD	SA
U.	S1		CS	

S	S#	SN	SD	SA
	<u>S1</u>		CS	<u>19</u>
U.	<u>S1</u>			<u>19+1</u>

(3) 插入操作

操作符为“**I.**”，新元组必须包含码，其他属性值可为空。

例：

S	S#	SN	SD	SA
I.	S8	美丽	CS	19

(4) 删除操作

操作符为“**D.**”。

例：删除计算机系的学生。

S	S#	SN	SD	SA
D.			CS	

第三章 关系数据库标准语言——SQL

本章要求：

- 1、掌握SQL定义基本表和建立索引的方法
- 2、掌握SQL中各种查询方法和数据更新方法
- 3、掌握SQL中视图的定义方法和用法
- 4、掌握SQL的授权机制
- 5、了解嵌入式SQL的基本使用方法

本章内容：

- § 1 SQL概述
- § 2 SQL数据定义功能
- § 3 SQL数据操纵功能
- § 4 视图
- § 5 SQL数据控制功能
- § 6 嵌入式SQL

请选择内容

返回

§ 1 SQL概述

一、SQL的发展

SQL是 **S**tructured **Q**uery **L**anguage的缩写

(ANSI解释为**S**tandard **Q**uery **L**anguage)

74年 Boyce &Chambarlin提出，在IBM的System R上
首先实现

79年 Oracle

82年 IBM的DB2

84年 Sybase

} 采用SQL作为数据库语言

第三章 关系数据库标准语言——SQL

86年10月 成为美国国家标准

87年 国际标准化组织（ISO）采纳为国际标准

89年 ISO推出SQL89

92年 ISO推出SQL2

目前正制定SQL3标准

二、SQL的主要特点

1、一体化：两方面

- 集DDL、DML、DCL为一体
- 实体和联系都是关系，因此每种操作只需一种操作符

第三章 关系数据库标准语言——SQL

- 2、高度非过程化语言（**WHAT** ✓ **HOW** ×）
- 3、面向集合的操作方式（一次一集合）
- 4、交互式 and 嵌入式两种使用方式，统一的语法结构
- 5、语言简洁，易学易用

完成核心功能只有9个动词：

数据查询： **SELECT**

数据定义： **CREATE, DROP, ALTER**

数据操纵： **INSERT, DELETE, UPDATE**

数据控制： **GRANT, REVOKE**

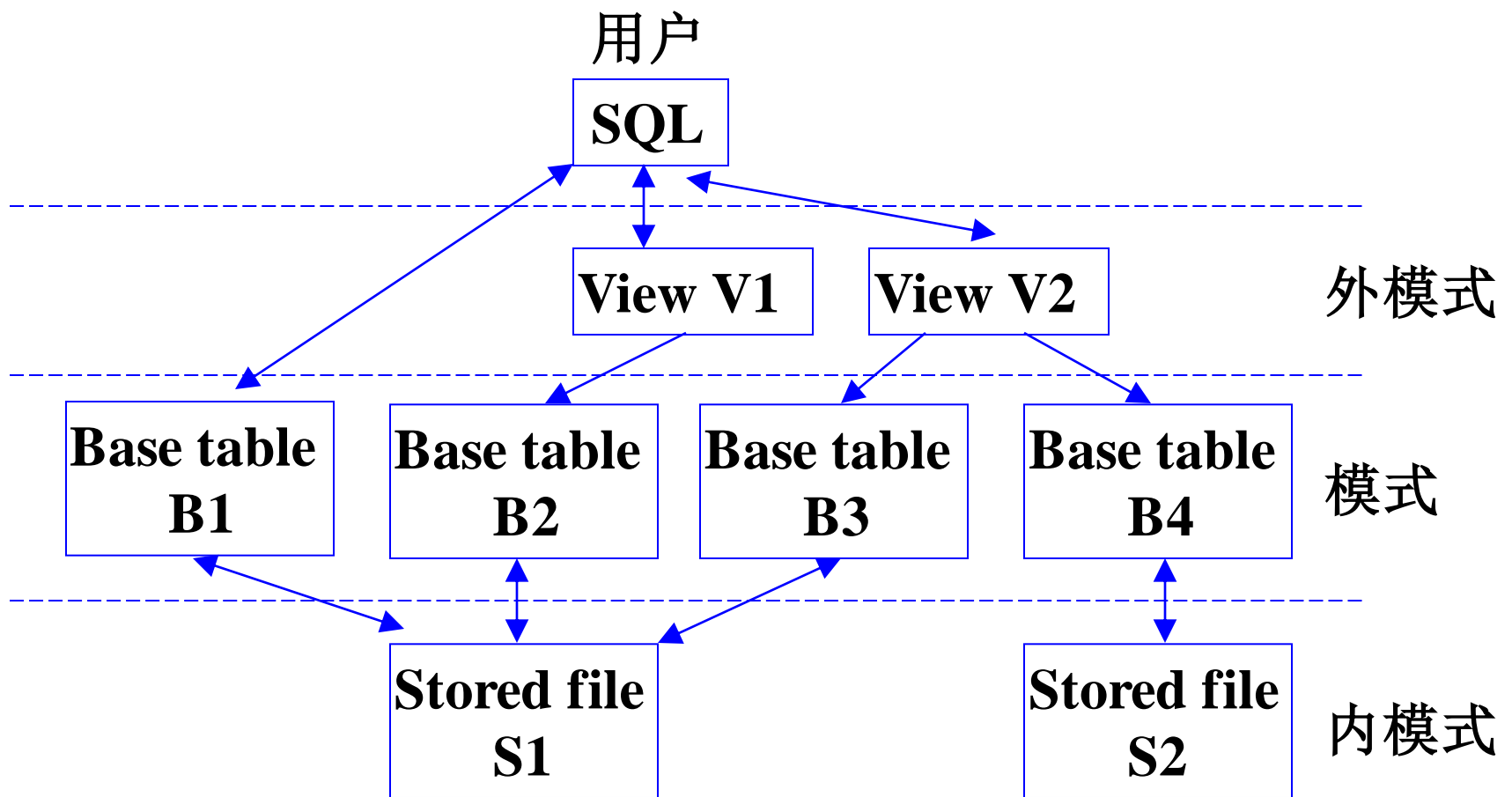
6、支持三级模式结构

视图 ↔ 外模式

基本表（的集合） ↔ 模式

存储文件和索引 ↔ 内模式

第三章 关系数据库标准语言——SQL



SQL支持的三级模式结构

第三章 关系数据库标准语言——SQL

说明：

↓ 基本表是独立存在的表。一个关系对应一个表。一个（或多个）表对应一个存储文件，每个表可有若干索引，这些索引也可放在存储文件中。

↓ 视图是从一个或几个基本表中导出的表，概念上同基本表。但它并不真正存储数据，也不独立存在，它依赖于导出它的基本表，数据也存放在原来的基本表中。

↓ 对内模式，只需定义索引，其余的一切均有DBMS自动完成

本节开头

本章开头

下一节

§ 2 SQL数据定义功能

三部分：

- ☆ 定义和修改基本表（定义模式）：**CREATE TABLE**
DROP TABLE
ALTER TABLE
- 🕒 定义视图（定义外模式）：**CREATE VIEW**
DROP VIEW
- 🕒 定义索引（定义内模式）：**CREATE INDEX**
DROP INDEX

说明：视图是从基本表导出的虚表，索引依赖于基本表，SQL没有修改视图和索引的操作，可通过先删除，再创建达此目的。

第三章 关系数据库标准语言——SQL

一、基本表的定义和修改

1、定义：基本格式为

CREATE TABLE 表名 (列名1 类型 [列级完整性约束]
[, 列名2 类型 [列级完整性约束]...);

示例

CREATE TABLE S (S# CHAR (3) NOT NULL UNIQUE
,

SN CHAR (15) ,

SD CHAR (15) ,

取值唯一

不允许取空值

说明:

↓ 注意SQL语句的书写格式 (SMALLINT);

↓ SQL支持空值的概念。允许空值的列未输入数据时系统自动置为空值。

↓ SQL支持的数据类型随系统不同而有所差异。

第三章 关系数据库标准语言——SQL

2、修改基本表

不支持NOT NULL选择

(1) 增加列 如 **ALTER TABLE S ADD SD INT;**

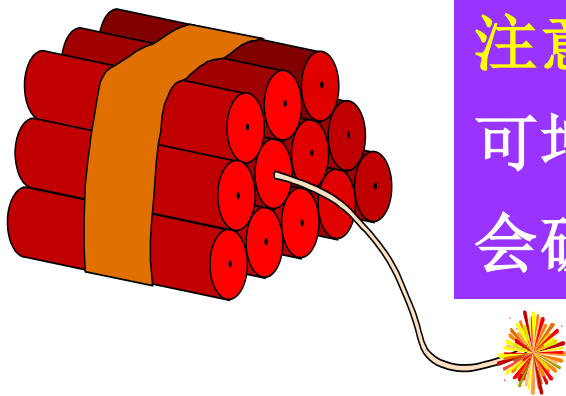
ALTER TABLE 表名 **ADD** 列名 类型 [完整性约束];

(2) 修改列 如 **ALTER TABLE S MODIFY SD CHAR (20**

ALTER TABLE 表名 **MODIFY** 列名 类型;

(3) 删除完整性约束

ALTER TABLE 表名 **DROP** 完整性约束名;



注意：不能删除列，新增列的值一律为空值，可增加列宽，但一般不能减小列宽，修改可能会破坏已有数据。

在定义基本表时要考虑充分

第三章 关系数据库标准语言——SQL

3、删除：

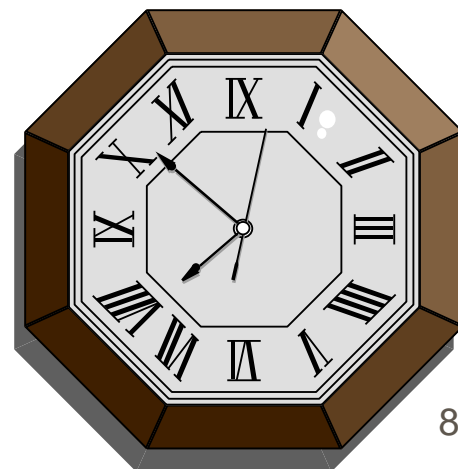
DROP TABLE 表名；

注意：删除基本表时，表中的数据、建立在表上的索引和视图将一并被删除，因此应格外小心。

二、索引的建立和删除

由DBA或表的属主进行，存取数据时由系统自动选取合适的索引作为存取路径，用户不必也不能选择索引。

。



第三章 关系数据库标准语言——SQL

1、建立

一个索引项值仅对应唯一的数据记录

CREATE [UNIQUE] [CLUSTER] INDEX 索引名

ON 表名 (列名 $\begin{pmatrix} \text{ASC} \\ \text{DESC} \end{pmatrix}$ [, 列名 $\begin{pmatrix} \text{ASC} \\ \text{DESC} \end{pmatrix}$]...) ;

改变记录的物理顺序使之与索引项值的排列顺序相同，称为聚簇索引。显然一个表只能建立一个聚簇索引。可通过在经常查询而改动小的表上建立这种索引来提高查询效率。

升序或降序
缺省为升序

如 **CREATE UNIQUE INDEX XSNO ON S (S#) ;**
CREATE UNIQUE INDEX SCNO
ON SC (SNO ASC, CNO DESC) ;

2、删除

DROP INDEX 索引名;

第三章 关系数据库标准语言——SQL

§ 3 SQL数据操纵——查询

查询是数据库的核心操作。SQL仅提供了唯一的语句**SELECT**，其使用方式灵活，功能非常丰富。



第三章 关系数据库标准语言——SQL

2、简单查询

例1：求选修了课程的学生学号

SELECT DISTINCT S#
FROM SC;

从结果中去掉重复的元组

例2：SELECT的后面可以是表达式。

如求计算机系学生的学号和出生年份：

SELECT S#, 'Birthday: ', 2000-SA
FROM S
WHERE SD='CS';

例3：连续范围查询，使用BETWEEN (NOT BETWEEN)

SELECT S#, SA
FROM S
WHERE SA BETWEEN 20 AND 22;

即求20到22岁之间的
学生学号和年龄

第三章 关系数据库标准语言——SQL

例4：离散范围查询，使用 **IN** (**NOT IN**)

SELECT *

星号表示无投影

FROM S

WHERE SD IN ('MA', 'CS');

相当于若干
'OR'的缩写

**SD='MA' OR
SD='CS'**

第三章 关系数据库标准语言——SQL

2、简单查询

例1：求选修了课程的学生学号

SELECT DISTINCT S#
FROM SC;

从结果中去掉重复的元组

例2：SELECT的后面可以是表达式。

如求计算机系学生的学号和出生年份：

SELECT S#, 'Birthday: ', 1998-SA
FROM S
WHERE SD='CS';

例3：连续范围查询，使用BETWEEN (NOT BETWEEN)

SELECT S#, SA
FROM S
WHERE SA BETWEEN 20 AND 22;

SA >= 20 AND
SA <= 22

相当于若干
'AND' 的缩写

第三章 关系数据库标准语言——SQL

例4: 离散范围查询, 使用 **IN** (**NOT IN**)

```
SELECT *  
FROM S  
WHERE SD IN ( 'MA', 'CS' );
```

相当与若干
'OR'的缩写

例5: 模糊查询, 使用 **LIKE** (**NOT LIKE**)

```
SELECT *  
FROM S  
WHERE SN LIKE '%清%';
```

查姓名中有‘清’字的学生

DB2中, 下划线 ‘_’表示匹配任何单个字符 其它语言中, 常用 ‘?’
百分号 ‘%’表示匹配任何字符串 常用 ‘*’

例6: 涉及空值的查询, **IS NULL** (**IS NOT NULL**)

```
SELECT S#, C# FROM SC  
WHERE G IS NULL;
```

第三章 关系数据库标准语言——SQL

3、连接查询：涉及至少两个表的查询

SQL中没有专门的JOIN命令，而是靠SELECT语句中的**WHERE子句**来达到连接运算的目的，因此更加灵活、简便。用来连接两个表的条件称为**连接条件**或**连接谓词**。

连接条件的一般格式为：

[表名1.]列名1 比较运算符 [表名2.]列名2

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

比较运算符主要有：=、>、<、>=、<=、!=。

等值连接：运算符为“=”时。

非等值连接：运算符不是“=”时。

自然连接：等值连接且目标列不含重复属性。

连接操作的实现过程：见教材P102中部。

第三章 关系数据库标准语言——SQL

例1: 简单的连接查询

求选修C1课程的学生学号、
姓名和成绩

```
SELECT S.S#, 连接字段  
FROM S, SC  
WHERE S.S#=SC.S#  
AND SC.C#='C1';
```

表名前缀 (字段名
唯一时可省略)

S : S#	SN	SD	SA	SC:S#	C#	G
01	A	MA	20	01	C1	A
02	B	CS	19	01	C2	A
03	C	IS	21	02	C2	B
04	D	MA	19	02	C3	C
05	E	MA	20	03	C3	B
				04	C1	B
				04	C4	A

连接条件
或称连接谓词

第三章 关系数据库标准语言——SQL

例1: 简单的连接查询

求选修C1课程的学生学号、姓名和成绩

```
SELECT S.S#, SN, G
FROM S, SC
WHERE S.S#=SC.S#
AND SC.C#='C1';
```

查询结果: S# SN G
01 A A

S : S#	SN	SD	SA	SC:S#	C#	G
01	A	MA	20	01	C1	A
02	B	CS	19	01	C2	A
03	C	IS	21	02	C2	B
04	D	MA	19	02	C3	C
05	E	MA	20	03	C3	B
				04	C1	B
				04	C4	A

条件不满足

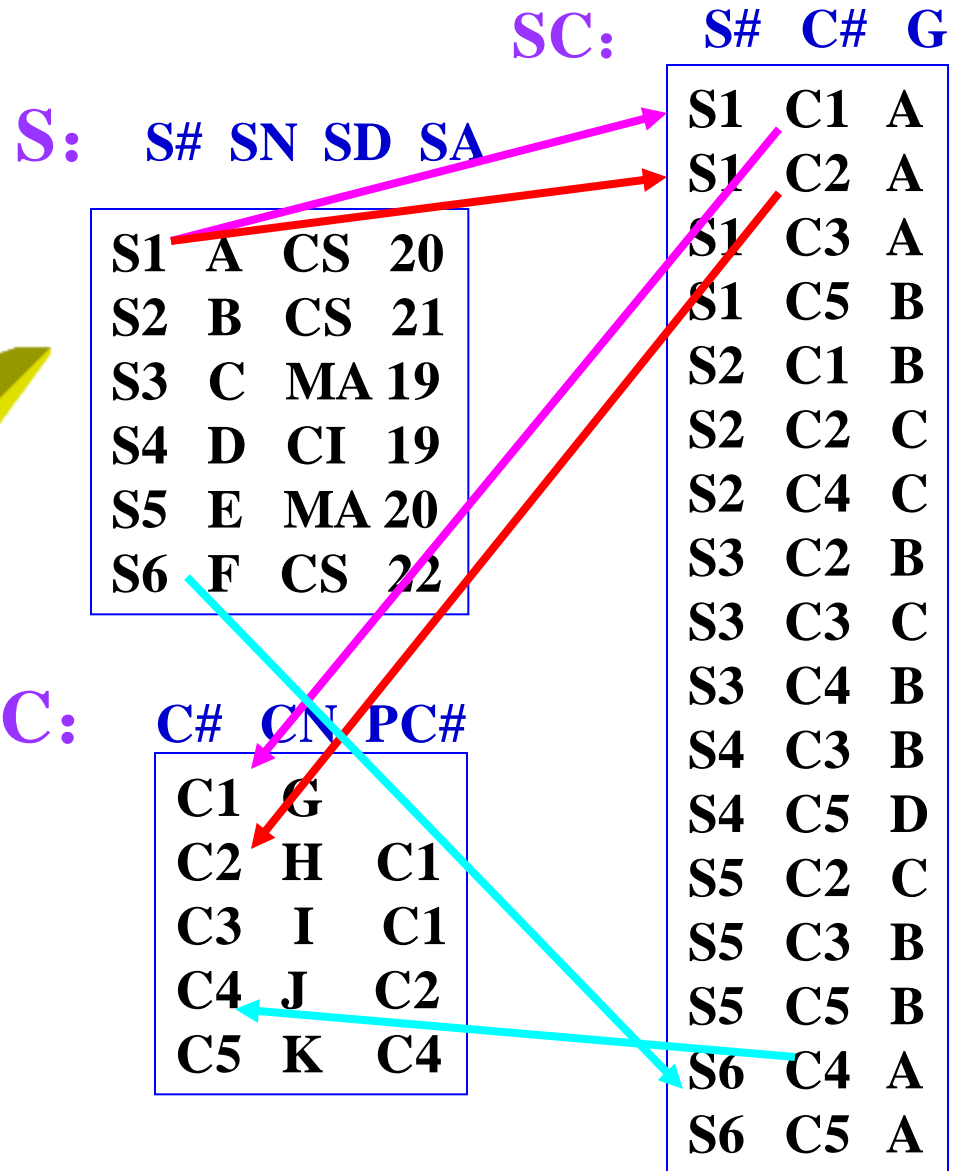
在多个表中出现的列名，必须用表名限定，仅在一个表中出现的属性，可省略表名。

第三章 关系数据库标准语言——SQL

例2: 多表连接

求学生学号、姓名、
选修课程名、成绩

直观的查找过程



第三章 关系数据库标准语言——SQL

例2: 多表连接

求学生学号、姓名、

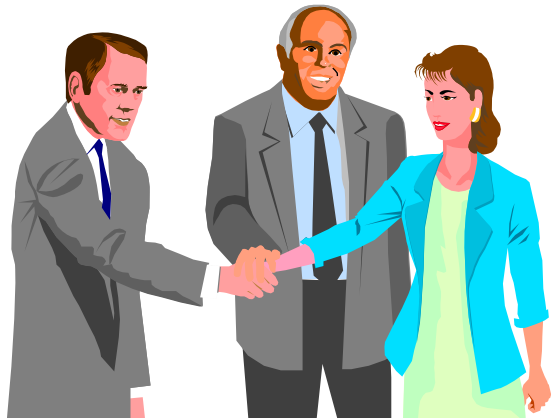
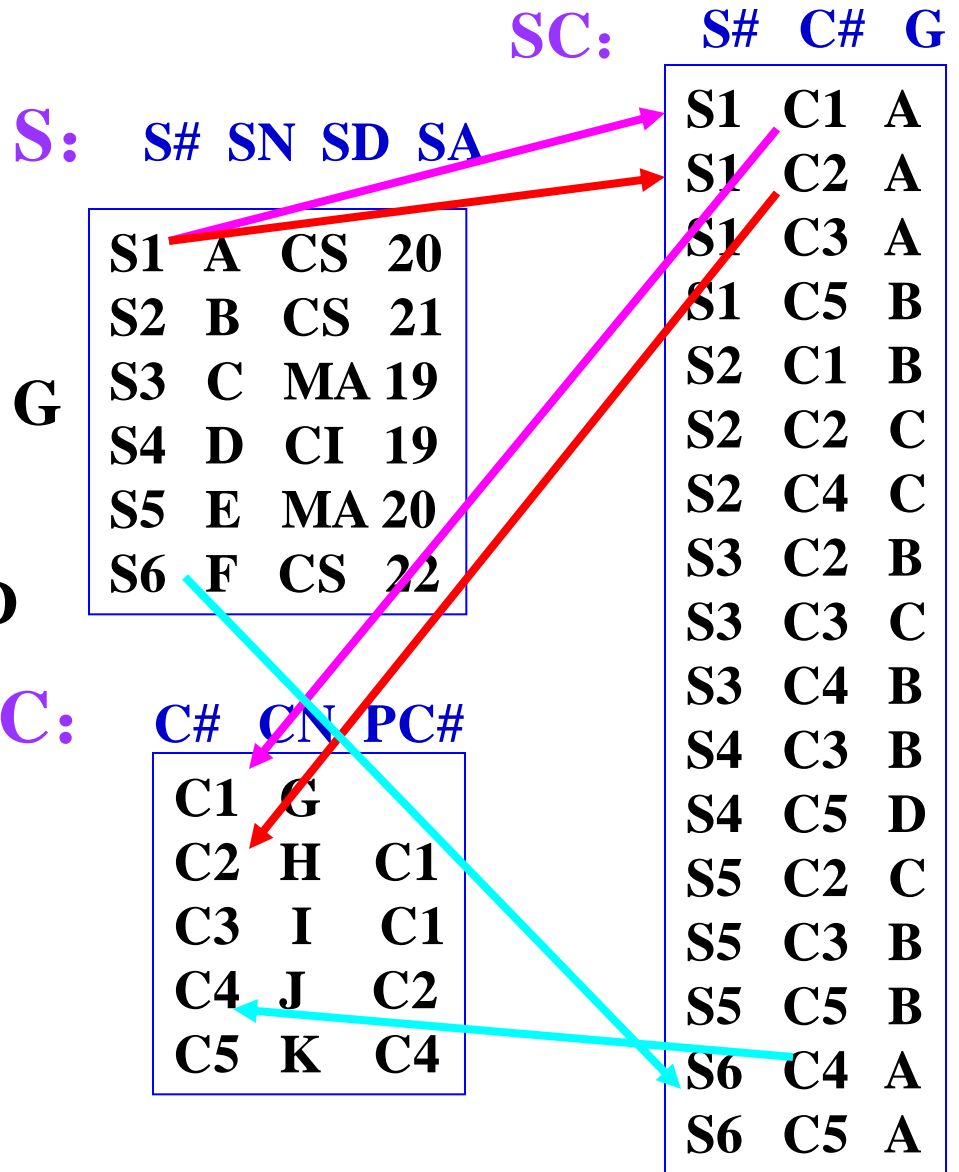
选修课程名、成绩

SELECT S.S#, SN, CN, G

FROM S, C, SC

WHERE S.S#=SC.S# AND

SC.S#=C.C#;

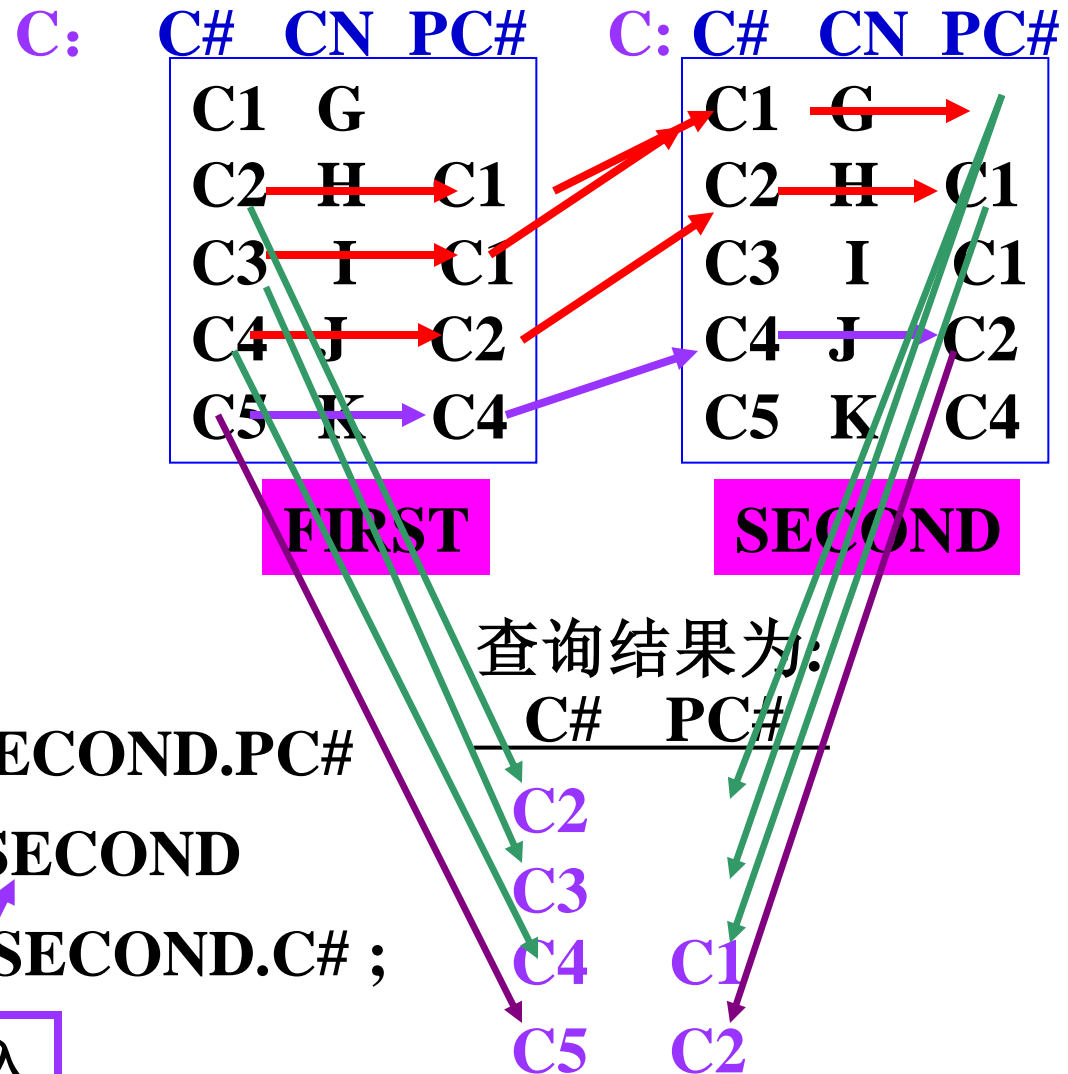


第三章 关系数据库标准语言——SQL

例3: 单表连接

求每一门课程的
间接先行课

直观的查找过程



SQL语句为:

```
SELECT FIRST.C#, SECOND.PC#  
FROM C FIRST, C SECOND  
WHERE FIRST.PC# = SECOND.C# ;
```

别名引入

第三章 关系数据库标准语言——SQL

例4：外连接查询
求每个学生选修的
课程及成绩。

使用外连接时

SQL语句为:

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

有些数据库系统用+
有些将*放在=前后, *=、=*


S:S#	SN	SD	SA	SC:S#	C#	G
S1	A	CS	20	S1	C1	A
S2	B	CS	21	S1	C2	A
S3	C	MA	19	S2	C1	B
S4	D	CI	19	S2	C2	C
				S2	C4	C

查询结果为:

S#	SN	C#	G
S1	A	C1	A
S1	A	C2	A
S2	B	C1	B
S2	B	C2	C
S2	B	C4	C
S3	C		
S4	D		

第三章 关系数据库标准语言——SQL

4、嵌套查询（子查询）

SELECT-FROM-WHERE

查询块嵌入另一个查询块中

例1：求选修了课程名为‘J’

分析：的查询逻辑复杂，但

最后结果只包含S中的字段，

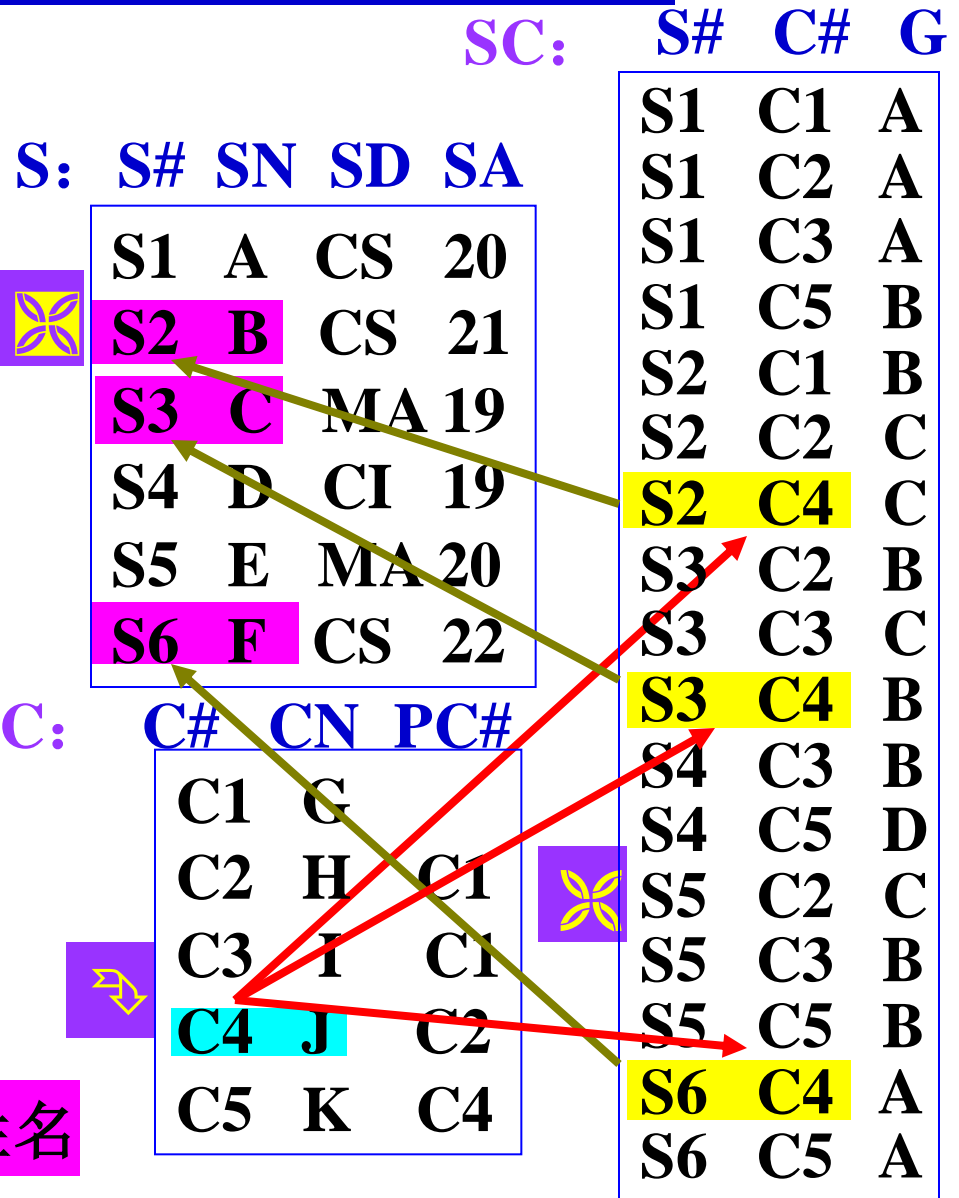
应该考虑更为有效、直观

的方法：
WHERE S.S#=SC.S#

在C中找课程‘J’的编号

在SC中找选修该课的学号

在S中找选修该课的学生姓名



第三章 关系数据库标准语言——SQL

在C中找课程 ‘J’的编号

```
SELECT C#
FROM C
WHERE CN='J';
```

S: S# SN SD SA

S1	A	CS	20
S2	B	CS	21
S3	C	MA	19
S4	D	CI	19
S5	E	MA	20
S6	F	CS	22

在SC中找选修该课的学号

```
SELECT S#
FROM SC
WHERE C# IN ( 'C4' );
```

C: C# CN PC#

C1	G	
C2	H	C1
C3	I	C1
C4	J	C2
C5	K	C4

在S中找选修该课的学生姓名

```
SELECT S#, SN
FROM S
WHERE S# IN ('S2','S3','S6');
```

SC: S# C# G

S1	C1	A
S1	C2	A
S1	C3	A
S1	C5	B
S2	C1	B
S2	C2	C
S2	C4	C
S3	C2	B
S3	C3	C
S3	C4	B
S4	C3	B
S4	C5	D
S5	C2	C
S5	C3	B
S5	C5	B
S6	C4	A
S6	C5	A

第三章 关系数据库标准语言——SQL

在C中找课程 ‘J’的编号

最后的查询语句:

在SC中找选修该课的学号

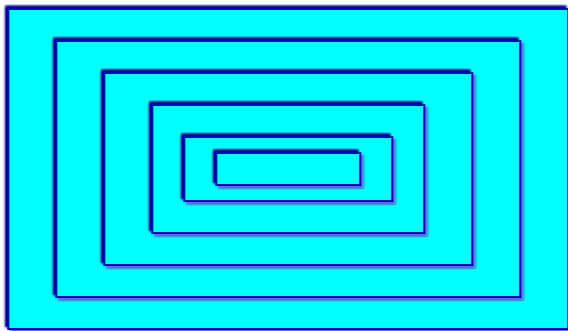
在S中找选修该课的学生姓名

```
SELECT S#, SN
FROM S
WHERE S# IN
(
    SELECT S#
    FROM SC
    WHERE C# IN
    (
        SELECT C#
        FROM C
        WHERE CN='J'
    )
);
```

第三章 关系数据库标准语言——SQL

说明:

- (1) 嵌套查询由内向外处理
- (2) SQL允许多层嵌套
- (3) 嵌套查询中最常用的谓词是IN
- (4) 嵌套查询层次分明、容易理解



最后的查询语句:

```
SELECT S#, SN
FROM S
WHERE S# IN
(
    SELECT S#
    FROM SC
    WHERE C# IN
    (
        SELECT C#
        FROM C
        WHERE CN='J'
    )
);
```


第三章 关系数据库标准语言——SQL

带有比较运算符的子查询

当用户确切知道内层查询的结果是单值（只有一个元组，且该元组只有一个字段）时，可将外层查询的某字段与内层查询的结果用>、<、=、>=、<=、!=等比较运算符进行比较。

例2：求“张三”选修的课程名称及成绩（**设无同名学生**）

```
SELECT C.CN, SC.G
FROM C, SC
WHERE C.C# = SC.C# AND
      SC.S# =
      (SELECT S#
       FROM S
       WHERE SN="张三" ) ;
```

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

子查询必须出现在比较符之后。

第三章 关系数据库标准语言——SQL

使用存在量词 **EXISTS** 和 **NOT EXISTS** 的查询

例3：求至少一门不及格的学生姓名

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

若内层查询结果非空，则为真
否则为假

等价于：

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

第三章 关系数据库标准语言——SQL

例4：查询没有选修“C1”课程的学生姓名。

```
SELECT SN  
FROM S  
WHERE NOT EXISTS  
  (SELECT *  
   FROM SC  
   WHERE S# = S.S#  
    AND C#="C1");
```



```
SELECT SN  
FROM S  
WHERE S# NOT IN  
  (SELECT S#  
   FROM SC  
   WHERE C#="C1");
```

哪种形式效率高

?

第三章 关系数据库标准语言——SQL

例5: 对全称量词的处理

想一想：用关系代数如何实现？

求选修了全部课程的学生姓名

用除法！

带全称量词的谓词 \longrightarrow 带存在量词的谓词

$$\begin{array}{ccc} \downarrow & & \downarrow \\ (\forall \mathbf{x}) \mathbf{P}(\mathbf{x}) & \longrightarrow & \neg (\exists \mathbf{x} (\neg \mathbf{P}(\mathbf{x}))) \end{array}$$

本例变为：选这样的学生姓名，没有一门课程是他不选修的

分析：任给一学号S#，若不存在他不选修的课，则显示他的姓名

$$\neg (\exists C\# (\text{学号为} S\# \text{的学生没修课程} C\#))$$

$$\neg (\exists C\# (\text{在SC中不存在选课单}(S\#, C\#, \dots)))$$

第三章 关系数据库标准语言——SQL

```
SELECT SN
FROM S
WHERE NOT EXISTS
(
  SELECT *
  FROM C
  WHERE NOT EXISTS
  (
    SELECT *
    FROM SC
    WHERE S#=S.S# AND
          C#=C.C#
  )
);
```

为真



查询结果为空



都为假



查询结果都非空



即所有的课程
都选修

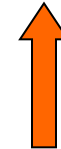
第三章 关系数据库标准语言——SQL

```
SELECT SN
FROM S
WHERE NOT EXISTS
(
  SELECT *
  FROM C
  WHERE NOT EXISTS
  (
    SELECT *
    FROM SC
    WHERE S#=S.S# AND
          C#=C.C#
  )
);
```

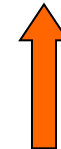
选修全部课程



$\neg (\exists C\# \text{ (学号为S\#的学生没修课程C\#)})$



在SC中不存在选课单 (S#, C#,...)?



在SC中查选课单 (S#, C#,...)

第三章 关系数据库标准语言——SQL

例6 对“蕴涵”的处理

$$P \rightarrow Q \equiv \neg P \vee Q$$

如: 求这样的学生学号, 该生至少选修了学生S2所修的全部课程

即找这样的学号S_x,

对所有的课程C_y, 若S2选修了C_y, 则S_x也选修了C_y

P表示谓词“学生S2选修课程C_y”

Q表示谓词“学生S_x选修课程C_y”

查询条件为:

$$\begin{aligned} & (\forall C_y) (P \rightarrow Q) \\ & \equiv \neg \exists C_y (\neg (P \rightarrow Q)) \\ & \equiv \neg \exists C_y (\neg (\neg P \vee Q)) \\ & \equiv \neg \exists C_y (P \wedge \neg Q) \end{aligned}$$

第三章 关系数据库标准语言——SQL

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 SCZ.S# = SCX.S# AND

SCZ.C# = SCY.C#));

条件 $\neg \exists Cy (P \wedge \neg Q)$

请思考：
直观的查找过程？

第三章 关系数据库标准语言——SQL

问题: 上例可否改为下面的SQL语句?

```
SELECT S#  
FROM S  
WHERE NOT EXISTS  
  ( SELECT *  
    FROM SC SCY  
    WHERE SCY.S# = 'S2' AND  
          NOT EXISTS  
            ( SELECT *  
              FROM SC SCZ  
              WHERE SCZ.S# = S.S# AND  
                    SCZ.C# = SCY.C# ) );
```

选这样的学号 S_x , 不存在S2选修的一门课程 S_x 未选修



选这样的学号 S_x , S2选修的课程 S_x 都选修了

第三章 关系数据库标准语言——SQL

例7：使用UNION的查询
求C1和C4课程成绩为
A的学生的学号及姓名

```
SELECT S.S#, S.SN
FROM S, SC
WHERE S.S#=SC.S# AND
      SC.C#='C1' AND
      SC.G='A'

UNION

SELECT S.S#, S.SN
FROM S, SC
WHERE S.S#=SC.S# AND
      SC.C#='C4' AND
      SC.G='A';
```

S:	S#	SN	SD	SA	SC:	S#	C#	G
	S1	A	CS	17		S1	C1	A
	S2	B	CS	21		S1	C3	A
	S3	C	MA	19		S1	C5	B
	S4	D	CI	17		S2	C1	A
	S5	E	MA	20		S2	C2	C
	S6	F	CS	20		S2	C4	A
	S9	Z	IS	18		S3	C3	
						S3	C4	
						S4	C3	B
						S4	C5	D
						S5	C2	
						S5	C3	
						S5	C5	
						S6	C4	A

参加UNION操
作的各结果表的
列数、对应
项的数据类型
必须相同

S6 F

第三章 关系数据库标准语言——SQL

例7：使用UNION的查询
求C1和C4课程成绩为A
的学生的学号及姓名

等价于：

```
SELECT DISTINCT S.S#,S.SN
FROM S, SC
WHERE (S.S#=SC.S# AND
       SC.C#='C1' AND
       SC.G='A')
OR
      (S.S#=SC.S# AND
       SC.C#='C4' AND
       SC.G='A');
```

S:

S#	SN	SD	SA
S1	A	CS	17
S2	B	CS	21
S3	C	MA	19
S4	D	CI	17
S5	E	MA	20
S6	F	CS	20
S9	Z	IS	18

SC:

S#	C#	G
S1	C1	A
S1	C3	A
S1	C5	B
S2	C1	B
S2	C2	C
S2	C4	C
S3	C3	0
S3	C4	0
S4	C3	B
S4	C5	D
S5	C2	0
S5	C3	0
S5	C5	0
S6	C4	A

查询结果

S#	SN
S1	A
S2	B
S2	B
S6	F

有重复的行

第三章 关系数据库标准语言——SQL

例7：使用UNION的查询
求C1和C4课程成绩为A
的学生的学号及姓名

等价于：

```
SELECT DISTINCT S.S#,S.SN
FROM S, SC
```

```
WHERE S.S#=SC.S# AND
      SC.G='A' AND
      SC.C# IN ('C1','C4');
```

S:	S#	SN	SD	SA
	S1	A	CS	17
	S2	B	CS	21
	S3	C	MA	19
	S4	D	CI	17
	S5	E	MA	20
	S6	F	CS	20
	S9	Z	IS	18

SC:	S#	C#	G
	S1	C1	A
	S1	C3	A
	S1	C5	B
	S2	C1	B
	S2	C2	C
	S2	C4	C
	S3	C3	0
	S3	C4	0
	S4	C3	B
	S4	C5	D
	S5	C2	0
	S5	C3	0
	S5	C5	0
	S6	C4	A

查询结果

S#	SN
S1	A
S2	B
S6	F

第三章 关系数据库标准语言——SQL

说明:

(1) 用UNION查询时, 每个子查询选择的目标列必须相同, 但所基于的表可以不同

(2) UNION查询实际上是修C5课程的学生姓名的并集

(3) UNION查询时自动去掉重复的行

SELECT S.S#, S.SN
FROM S
WHERE S.D#='C5'
UNION
SELECT S.S#, S.SN
FROM S, SC
WHERE S.S#=SC.S# AND SC.C#='C5';

第三章 关系数据库标准语言——SQL

例8：求选修课程C1的学生
与选修C2的学生的交集。
即求同时选修了课程C1和C2
的学生学号。

```
SELECT S#  
FROM SC  
WHERE C#='C1'  
AND S# IN  
  (SELECT S#  
   FROM SC  
   WHERE C#='C2') ;
```

例9：求计算机系学生与年龄小
于20岁的学生的差集。
即求计算机系年龄不小于20岁
的学生。

```
SELECT S#, SN  
FROM S  
WHERE SD='CS'AND  
      SA>=20;
```

第三章 关系数据库标准语言——SQL

5、库函数（聚集函数）

COUNT 统计一列中的（**NOT NULL**）值的个数

COUNT (*) 计算记录个数

SUM 对一列求和

AVG 对一列求平均值

MAX 对一列求最大值

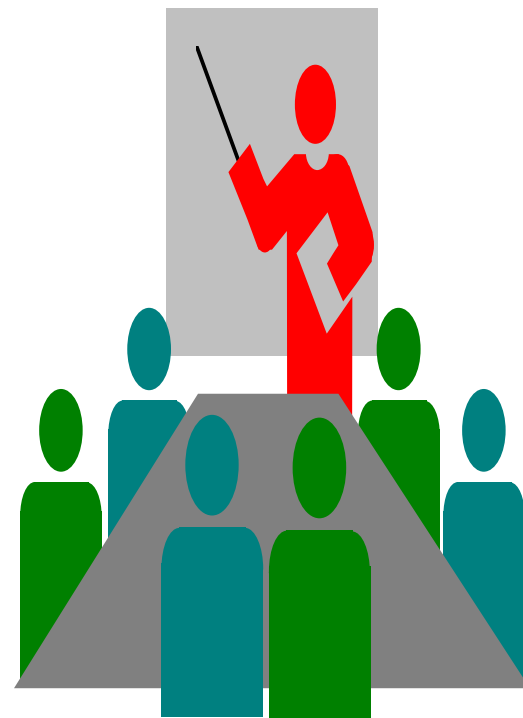
MIN 对一列求最小值

例1：求选修了课程的学生人数

```
SELECT COUNT (DISTINCT S#)
```

```
FROM SC;
```

↑
重复的只记一个



第三章 关系数据库标准语言——SQL

例2 分组统计：使用**GROUP BY**

求选修各门课程的学生人数

```
SELECT C#, COUNT (S#)
FROM SC
GROUP BY C#;
```

将表按列的值分组，列的值相同的分在一组，产生一个结果行。**GROUP BY**常和库函数一起使用，用于分组统计。

```
SELECT SC.C#, C.CN, COUNT (S#)
FROM SC, C WHERE SC.C#=C.C#
GROUP BY SC.C#, CN;
SELECT SC.C#, MIN(C.CN), COUNT(S#)
FROM SC, C WHERE SC.C#=C.C#
GROUP BY SC.C#;
```

上例中，如果要求显示课程名称，则可以
这里，实际上仅需按C#分组即可，不必再按CN分组，为此可用MIN函数作用之。因为同一分组中C#相同，CN也相同

第三章 关系数据库标准语言——SQL

例3 带条件的分组查询、统计：使用HAVING

求选修课程超过3门的学生学号

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

WHERE是选择记录的条件；
HAVING是选择分组的条件且
必须和GROUP BY一起使用
库函数只能作用于HAVING和
目标列，而不能用于WHERE

假定SC中的G用百分制表示，求最低成绩不低于85分，平均成绩不低于90分的学生学号和姓名，并按学号降序排序。

```
SELECT SC.S#, MIN(S.SN) FROM S, SC  
WHERE S.S#=SC.S#  
GROUP BY SC.S#  
HAVING MIN(SC.G) >= 85 AND AVG(SC.G) >= 90  
ORDER BY SC.S# DESC;
```

通过过滤函数用
更库函数多遍推用
GROUP BY
按学号降序排序

第三章 关系数据库标准语言——SQL

§ 4 SQL数据操纵——数据更新

一、插入数据

1、插入单个元组

INSERT

INTO 表名 [(字段名 [, 字段名] ...]

VALUES (常量 [, 常量] ...);

例1：插入一条选课记录（S1，C5）。

INSERT

INTO SC (S#, C#)

VALUES ('S1', 'C1');

第三章 关系数据库标准语言——SQL

说明：

- ★ 当在INTO后面仅指定部分属性列时，插入记录后其它列的值为空值；
- ★ 如果INTO后面没有指定属性列，则必须按表列的定义次序为每个列指定一个值；
- ★ 具有NOT NULL属性的列，必须指定值。

2、插入子查询结果

INSERT

INTO 表名 [(字段名[, 字段名] ...]

子查询；

第三章 关系数据库标准语言——SQL

例2：求每个学生的平均成绩，并按学号、姓名、平均成绩存入数据库。（设成绩G是数值型）

先创建一个表

```
CREATE TABLE AG (S# CHAR (8) ,  
                  SN CHAR (8) ,  
                  AG SMALLINT) ;
```

将计算结果插入上表中

```
INSERT
```

```
INTO AG (S#, SN, AG)
```

```
SELECT S.S#, MIN(SN), AVG (G)
```

```
FROM S, SC
```

```
WHERE S.S# = SC.S# GROUP BY S.S# ;
```

为什么用MIN() 函数



第三章 关系数据库标准语言——SQL

二、修改数据

UPDATE 表名

SET 列名 = 表达式[, 列名 = 表达式]...

[WHERE 条件];

说明:

★ 当省略**WHERE**子句时，修改表中所有记录，否则仅修改满足条件的记录;

★ 条件也可以使用子查询。

例1：将所有学生的年龄增加1岁。

UPDATE S

SET SA = SA +1 ;

第三章 关系数据库标准语言——SQL

例2：把数学系全体学生的成绩置零

```
UPDATE SC
SET G=0
WHERE 'MA'=
  (SELECT SD
   FROM S
   WHERE S.S# = SC.S# );
```



对SC中的每个选课单，检查其对应学生所在的系是否‘MA’。

```
UPDATE SC
SET G=0
WHERE S# IN
  ( SELECT S#
    FROM S
    WHERE SD='MA' );
```



找出‘MA’系的所有学生，检查SC选课单所对应的学生是否是这些学生中的一员。

第三章 关系数据库标准语言——SQL

三、删除数据：

DELETE

FROM 表名

[WHERE 条件];

只能删除表记录，不删除表结构。无条件时，删除全部记录，仅剩一个空表；有条件时删除满足条件的记录。

为物理删除命令

删除表结构用**DROPTABLE**

例1：删除不及格的学生记录。

DELETE

FROM SC

WHERE G < 60;

例2：删除物理课的全部选课单

DELETE

FROM SC

WHERE ‘物理’ =

(SELECT CN

FROM C

WHERE C.C# = SC.C#)

第三章 关系数据库标准语言——SQL

四、更新操作与数据库的一致性

增、删、改操作只能对一个表操作，可能会造成数据的不一致性。例如：

删除“物理”课程的全部信息，需使用两个操作实现，删除C表的课程记录和删除SC表的选课记录。若在完成第一个操作后发生意外，致使第二个操作未能实现，则造成数据库的不一致。因为SC中的物理课程号还存在，而被参照的C表中已没有该课程号的课程了。所以应保证这两个操作要么全做，要么全不做，为此，在数据库系统中引入了事物的概念。

为保证数据的一致性，大型数据库系统一般都提供若干策略：

第三章 关系数据库标准语言——SQL

删除主表（被参照表）中的数据时

- （1）自动删除参照表中的相应数据；
- （2）检查参照表中是否有数据参照，若有则拒绝删除。

向参照表中插入数据时

检查所有被参照表中是否有被参照的信息，若没有则拒绝插入。

修改主表中的被参照字段

检查参照表中是否有数据参照，若有则拒绝修改。

虽说SQL的DML语句只有4个，但语句中成分多样，因此简单易学、功能丰富、灵活多样。

§ 5 视 图

- 🕒 视图是从一个或几个基本表（或视图）导出的表。
（用户外模式是由若干基本表和/或若干视图构成的）
- 🕒 视图是一个虚表，只存储视图的定义，数据存在所基于的基本表中。
- 🕒 视图定义后就可象基本表一样来使用。
 - 可创建、删除视图
 - 可用来定义新的视图
 - 可在视图上查询（**SELECT**）
 - 可更新（**INSERT, DELETE, UPDATE**）视图，但受限制

第三章 关系数据库标准语言——SQL

一、视图的定义

1、建立视图

格式: **CREATE VIEW** 视图名 [(字段名[, 字段名]...)]
AS 子查询

[**WITH CHECK OPTION**];

对视图**UPDATE**或
INSERT时,记录要
满足子查询中的条件

功能: 在数据字典中存储视图的定义
(但并不执行子查询),

此后视图名就可作为一个表来使用。



属性列省略, 隐含同

例1: 建立计算机系的学生视图

```
CREATE VIEW CS_S  
AS SELECT S#, SN, SA  
FROM S  
WHERE SD='CS';
```

第三章 关系数据库标准语言——SQL

★ 组成视图的属性列名，要么全部写出，要么全部省略，省略时，隐含视图的属性列同子查询的目标列。当SELECT语句中有库函数、或字段表达式、或多表连接有同名字段时，则视图中必须指定字段名。

例2：把学生的学号和平均成绩定义为一个视图

```
CREATE VIEW S_G(S#, GAVG)
AS SELECT S#, AVG(G) FROM SC GROUP BY S#;
```

★ 视图中字段名可以和基本表中的字段名不同

例3：建一个课程号、课程名及选课人数(不少于10)的视图

```
CREATE VIEW C_N(KCH, KCM, XXRS)
AS SELECT SC.C#, MAX( CN), COUNT(S#)
FROM C, SC WHERE C.C#=SC.C#
GROUP BY SC.C# HAVING COUNT(S#)>=10 ;
```

第三章 关系数据库标准语言——SQL

★ 没有修改视图的方法，要实现此功能，唯一的途径是先删除，再重建。

★ 视图的子查询可以基于一个或多个基本表或/和视图上

2、删除视图

DROP VIEW 视图名；




★ 删除基本表或视图后，由被删除的基本表或视图导出的视图仍然存在，但已无法使用，需另行删除。

二、视图上的查询

1、执行过程

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，最后在基本表上执行修改后的查询，这一转换称为**视图消解**。

第三章 关系数据库标准语言——SQL

例1: **SELECT S#, SA**
FROM CS_S  **SELECT S#, SA**
WHERE SA < 20;  **FROM S**
WHERE SD='CS'  **AND SA < 20;**

CS_S 视图中的子查询

```
SELECT S#, SN, SA
FROM S
WHERE SD='CS'
```

Diagram description: The diagram shows the transformation of a query from a view (CS_S) to its underlying table (S). An orange arrow labeled '修改为' (modified to) points from the original query to the modified one. Red arrows show the mapping of table names (CS_S to S) and the addition of a filter condition (SD='CS') to the WHERE clause.

2、注意事项

当视图中的字段对应的是一个库函数或字段表达式时，
有些系统 **转换后的查询可能会不正确**

请看下例 

第三章 关系数据库标准语言——SQL

例2：求平均成绩90分以上的学生

```
SELECT *  
FROM S_G  
WHERE GAVG >=90;
```

修改为

```
SELECT S#, AVG(G)  
FROM SC  
WHERE AVG(G) >=90  
GROUP BY S#;
```

正确吗？

S_G视图中的定义是

```
CREATE VIEW  
S_G(S#, GAVG)  
AS  
SELECT S#, AVG(G)  
FROM SC  
GROUP BY S#
```

正确的应为：

```
SELECT S#, AVG(G)  
FROM SC  
GROUP BY S#  
HAVING AVG(G) >=90;
```

第三章 关系数据库标准语言——SQL

三、视图上的更新（INSERT, DELETE, UPDATE）

1、执行方式

将对视图的更新语句转化为对相应的基本表的更新语句，然后执行。为防止更新基本表中不属于本视图的数据，可在视图定义时加上**WITH CHECK OPTION**子句。

例1: INSERT

INSERT INTO CS_S

VALUES ('S12', 'YanXi', 19)



INSERT

INTO S(S#, SN, SA, SD)

VALUES ('S12', 'YanXi', 19, 'CS')

视图中的子查询

```
SELECT S#, SN, SA
FROM S
WHERE SD='CS'
```



第三章 关系数据库标准语言——SQL

例2：将计算机系学号为“S2”的学生姓名改为“刘辰”。

UPDATE CS_S		UPDATE S
SET SN = “刘辰”		SET SN = “刘辰”
WHERE S#=”S2”;		WHERE S#=”S2” AND SD=”CS”;

例3：删除计算机系学号为“S2”的学生。

DELETE		DELETE
FROM CS_S		FROM S
WHERE S#=”S2”;		WHERE S#=”S2” AND SD=”CS”;

2、注意事项

不是所有的视图更新都可正确转化为对基本表的更新语句

第三章 关系数据库标准语言——SQL

例如: UPDATE S_G
SET GAVG=90
WHERE S#='S1';

不能有意义地转化

👉 有些视图是可更新的, 有些视图是不可更新的。但现在还无判别方法。

👉 肯定可以更新的视图是行列子集视图



从单个表导出, 且只是去掉了基本表的某些行和某些列并保留了码

👉 处理方式: 只有从单个表导出的视图才允许更新操作, 且作一系列的限定。(限制见P127)

👉 从概念上分清不可更新视图和不允许更新视图。

四、视图的优点（P128）

- 1、能够简化用户的操作
- 2、用户能以不同的方式对待同一数据，方便灵活
- 3、提供一定程度的逻辑独立性
- 4、有利于安全保密



本节开头



本章开头



下一节

第三章 关系数据库标准语言——SQL

§ 6 SQL数据控制功能

数据控制功能包括事物管理功能和数据保护功能。即

数据的安全性、完整性、事务控制、并发控制和恢复功能

本节只讨论安全性机制，即用户对数据的存取权力。

可通过在CREATE TABLE、ALTER TABLE语句中定义码、取值唯一的列、不允许空值的列、外码（参照完整性）及其它一些约束条件来体现。

在后面章节介绍。

一、授权

1、**机制**：大的DBMS中有一个超级用户DBA，其他用户能否存在、对某类数据具有何种操作权力是由DBA决定的，系统提供授权机制。**执行过程为：**

第三章 关系数据库标准语言——SQL

- (1) 用数据控制语言把授权决定告知系统;
- (2) 系统把授权的结果存入数据字典;
- (3) 当用户提出操作请求时, 系统根据授权情况进行检查, 以决定是否执行。

第三章 关系数据库标准语言——SQL

2、权力的授予与收回

若干权力

操作对象

授予: **GRANT** 权力 [, 权力] ... [**ON** 对象类型 对象名]
TO 用户名 [, 用户名] ... **获得权力的用户**
[**WITH GRANT OPTION**];

有此项, 被授权用户可再授权给其他用户

收回:

REVOKE 权力 [, 权力] ... [**ON** 对象类型 对象名]
FROM 用户名 [, 用户名] ...;

3、操作权力分类:

见P130表3.4

4、示例： 见P131

5、说明

☆ 数据库的属主、表的属主、数据库对象的属主在他创建的对象上具有一切可能的权力，其他用户得不到主人的授权就不能在该对象上操作；

🕒 SQL的授权机制十分灵活，各种系统又作了适当的补充，

使用十分方便；

🕒 SQL的安全性控制除了上述授权机制外，还可设置口令进一步保密。

本节开头

本章开头

下一节

§ 7 嵌入式SQL

一、SQL的使用方式

1、**交互式**：在终端上每输入一条SQL语句，系统立即执行，然后等待用户输入下一条语句。

2、**自编程式**：在实际的DMBS中，都对SQL进行了扩充，增加了条件、循环等控制语句，并提供编程机制。如SYBASE中，用户可以编写存储过程并调用它。

3、**嵌入式**（嵌入到某种主语言中使用）：

宿主语言负责：运算、处理、流程控制等

SQL负责：数据库操作

第三章 关系数据库标准语言——SQL

二、嵌入式SQL使用时的問題

必须解决和主语言相互配合、连接等问题

1、标识SQL语句

用前缀，如EXEC SQL或\$等，标记SQL语句的开始；
用END-EXEC或分号 ‘；’ 等标记SQL语句的结束。

SQL语句标识是通知主语言的预编译程序将SQL语句转化为等价的主语言语句，然后再由编译程序形成目标代码

2、SQL语句与主语言之间的通信

(1) 通过SQL通信区 (SQLCA) 将SQL语句的执行状态传递给主语言。SQLCA是一个数据结构，其中有一个重要变量SQLCODE，存放SQL语句是否执行成功的信息，每执行一个SQL语句，主语言都应测试该变量。

第三章 关系数据库标准语言——SQL

(3) SQL语句中可使用主语言的程序变量（叫主变量），但要加前缀标志，一般用冒号‘：’。

(4) SQL语句和主语言的数据交换一般通过字段变量和主变量进行，但两者数据类型要匹配

(5) 一个SQL语句可以产生一组记录，而主语言一次只能处理一个记录，为此需协调两种方式

办法：用游标（Cursor），有的叫位置指针

(6) 通常用CONNECT语句来连接（申请使用）数据库

例子见P136

第三章 关系数据库标准语言——SQL

三、不需游标的SQL语句

- 说明性语句
- 数据定义语句
- 数据控制语句

最简单的一类语句，不需返回结果，不使用主变量，在主语言中只需加前缀EXEC SQL和语句结束符即可。

- 查询结果为单记录的SELECT语句
- 非CURRENT形式的UPDATE语句
- 非CURRENT形式的DELETE语句
- INSERT语句

一般均使用主变量

1、说明、数据定义、数据控制语句

见教材P137-138

第三章 关系数据库标准语言——SQL

2、查询结果为单记录的SELECT语句

例 根据主变量的值查找学生的信息

```
EXEC SQL SELECT S#, C#, G
```

```
INTO :SNO, :CNO, :G :GID
```

```
FROM SC
```

```
WHERE S#= :GS AND C#=:GC;
```

查询结果存入这三个主变量中

指示变量，<0说明取得的G为空值

待查的学号存在GIVENS#中

👉 SELECT语句的INTO、WHERE、HAVING子句中可使用主变量

👉 可在INTO子句中使用指示变量，以指明某字段是否空值

👉 若SQLCODE=100，说明没有满足条件的记录

👉 当查询到的记录多于1条时，在SQLCODE中返回错误信息

第三章 关系数据库标准语言——SQL

3、非CURRENT形式的UPDATE语句

例 将计算机系全体学生的成绩置为空值

GID = -1;

EXEC SQL UPDATE SC

SET G = :GG :GID

WHERE “CS”=

(SELECT SD

FROM S

WHERE S.S#=SC.S#);

这里使用了值为负的指示变量GID，主变量GG可为任意值

👉 WHERE、SET子句中可以使用主变量，同时SET子句中还可以使用指示变量

👉 通过检查SQLCA的值，判别更新是否成功

第三章 关系数据库标准语言——SQL

4、非CURRENT形式的DELETE语句

例 学号在主变量X1和X2之间的学生已毕业，删除他们的信息。

先删除他们的选课信息

```
EXEC SQL DELETE  
FROM SC
```

```
WHERE S# BETWEEN :X1 AND :X2;
```

再删除他们的基本情况信息

```
EXEC SQL DELETE  
FROM S
```

```
WHERE S# BETWEEN :X1 AND :X2;
```

 **WHERE子句
中可使用主变量**

第三章 关系数据库标准语言——SQL

5、INSERT语句

例 插入一条学生选课记录

GID = -1;

EXEC SQL INSERT

INTO SC (S#, C#, G)

VALUES (:SNO, :CNO, :GG :GID) ;

欲插入记录的值由主语言存放在三个主变量中，**GID**指示成绩字段值为空值，**GG**可为任意值

四、使用游标的SQL语句

下列情况必须使用游标

- 查询结果为多条记录的**SELECT** 语句
- **CURRENT**形式的**UPDATE**语句
- **CURRENT**形式的**DELETE**语句

第三章 关系数据库标准语言——SQL

1、查询结果为多条记录的SELECT语句

如：查找由主变量DEPT中给出的某个系的全体学生信息

```
EXEC SQL DECLARE SX CURSOR FOR
```

定义游标

```
SELECT S#, SN, SA
```

```
FROM S
```

```
WHERE SD=:DEPT;
```

```
EXEC SQL OPEN SX;
```

打开游标

```
DO WHILE
```

```
EXEC SQL FETCH SX INTO :S#, :SNAME, :AGE;
```

```
.....
```

推进游标

```
END;
```

```
EXEC SQL CLOSE SX;
```

关闭游标

第三章 关系数据库标准语言——SQL

EXEC SQL **DECLARE** SX CURSOR FOR

SELECT S#, SN, SA

FROM S

WHERE SD=:DEPT;

EXEC SQL **OPEN** SX;

DO WHILE

EXEC SQL **FETCH** SX

INTO :S#, :SNAME, :AGE;

.....

END;

EXEC SQL **CLOSE** SX;

游标SX

游标SX

S: S# SN SD SA

S0 X MA 18

S1 A CS 20

S2 B CS 21

S3 C MA 19

S4 D CI 19

S5 E MA 20

S6 F CS 22

S7 G CI 21

假设DEPT中为 'CS'

主变量

S# SNAME AGE

S1

A

20

第三章 关系数据库标准语言——SQL

EXEC SQL **DECLARE** SX CURSOR FOR

SELECT S#, SN, SA

FROM S

WHERE SD=:DEPT;

EXEC SQL **OPEN** SX;

DO WHILE

EXEC SQL **FETCH** SX

INTO :S#, :SNAME, :AGE;

.....

END;

EXEC SQL **CLOSE** SX;

S: S# SN SD SA

S0	X	MA	18
S1	A	CS	20
S2	B	CS	21
S3	C	MA	19
S4	D	CI	19
S5	E	MA	20
S6	F	CS	22
S7	G	CI	21

游标SX



假设DEPT中为 'CS'

主变量

S# SNAME AGE

S2

B

21

第三章 关系数据库标准语言——SQL

EXEC SQL **DECLARE** SX CURSOR FOR

SELECT S#, SN, SA

FROM S

WHERE SD=:DEPT;

EXEC SQL **OPEN** SX;

DO WHILE

EXEC SQL **FETCH** SX

游标SX

INTO :S#, :SNAME, :AGE;

.....
END;

假设DEPT中为 'CS'

EXEC SQL **CLOSE** SX;

S: S# SN SD SA

S0	X	MA	18
S1	A	CS	20
S2	B	CS	21
S3	C	MA	19
S4	D	CI	19
S5	E	MA	20
S6	F	CS	22
S7	G	CI	21

主变量 S# SNAME AGE

S6

F

22

第三章 关系数据库标准语言——SQL

```
EXEC SQL DECLARE SX CURSOR FOR
SELECT S#, SN, SA
FROM S
WHERE SD=:DEPT;
EXEC SQL OPEN SX;
DO WHILE
EXEC SQL FETCH SX
INTO :S#, :SNAME, :AGE;

.....
END;
EXEC SQL CLOSE SX;
```

S: S# SN SD SA

S0	X	MA	18
S1	A	CS	20
S2	B	CS	21
S3	C	MA	19
S4	D	CI	19
S5	E	MA	20
S6	F	CS	22
S7	G	CI	21

主变量 S# SNAME AGE

S6	F	22
----	---	----

第三章 关系数据库标准语言——SQL

2、CURRENT形式的UPDATE和DELETE语句

可修改或删除当前活动游标所指向的记录.

例子见P144-146

本节开头

本章开头

本章要求：

- 1、掌握关系系统的有关概念
- 2、了解全关系系统的十二条基本准则
- 3、掌握查询优化的一般策略
- 4、掌握关系代数的等价变换规则
- 5、掌握关系代数表达式的优化算法和优化的一般步骤

本章内容：

§ 1 关系系统

§ 2 关系系统的查询优化

请选择内容

返回

§ 1 关系系统

一、关系系统的定义

1、关系模型：

数据结构： 关系（二维表）

数据操纵： 关系代数（或关系演算）

完整性约束： 实体完整性、参照完整性、用户定义的完整性

2、关系系统的定义

☒ 关系系统是关系数据库系统的简称

☒ 从概念上讲，支持关系模型的系统称为关系系统。

☒ 按最小要求定义关系系统：

要求过于严格

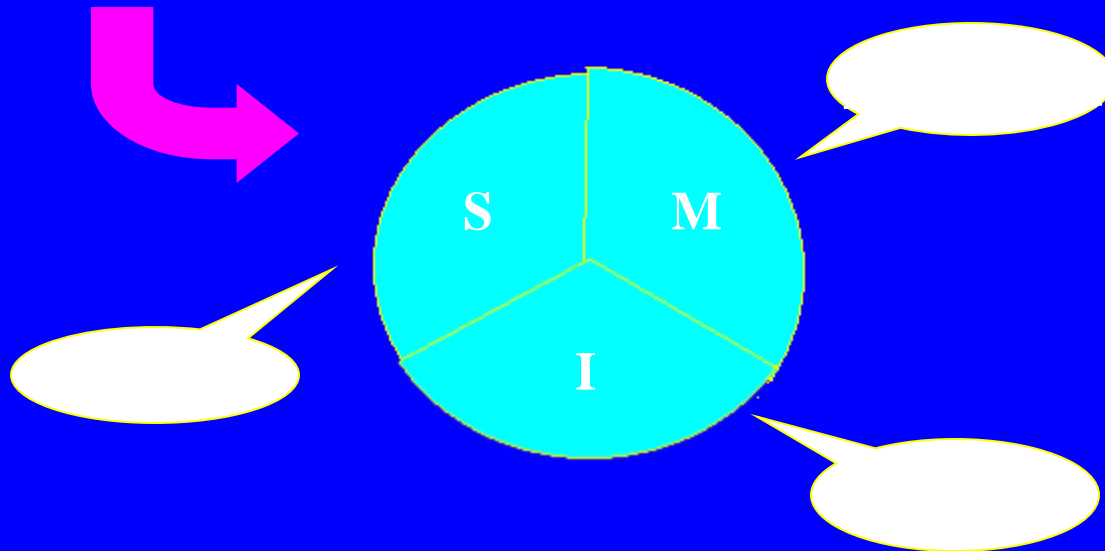
一个系统称为关系系统，当且仅当

（1）支持关系数据结构；

（2）支持选择、投影和连接运算。对运算不要求定义任何物理存取路径。

二、关系系统的分类

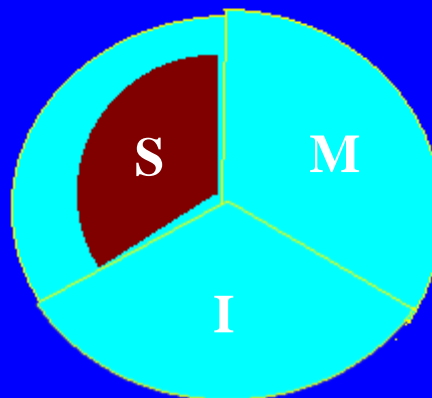
按对关系模型的支持程度来分



1、表式系统

仅支持关系结构，
不支持集合级操作

如：倒排表



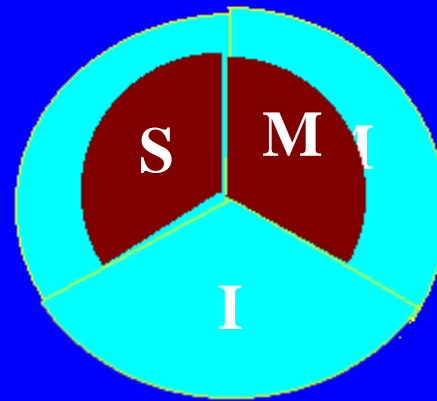
第四章 关系系统及其查询优化

2、（最小）关系系统

支持关系结构，

支持选择、投影和连接运算

如：FoxBASE、FoxPro

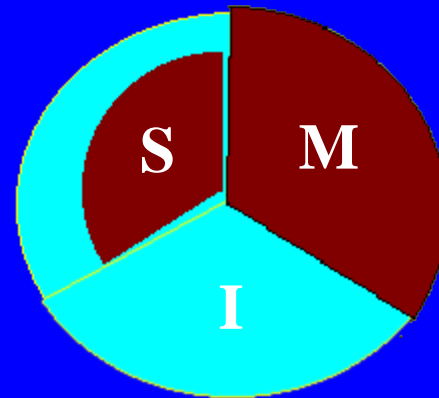


3、关系上完备的系统

支持关系结构，

支持所有的关系代数操作

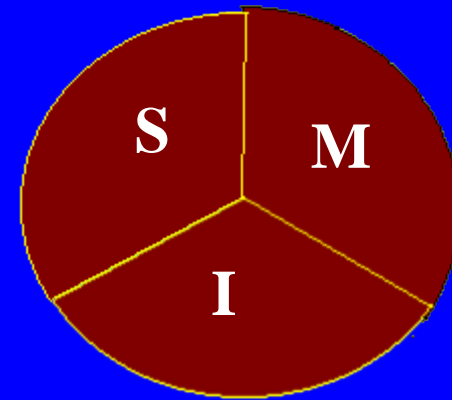
如：SYBASE、ORACLE、DB2



4、全关系系统

支持关系模型的所有特征

**SYBASE、ORACLE、
DB2**等系统已接近这个目标



三、全关系系统的十二条基本准则

基础（准则 0）：关系型**DBMS**必须能完全通过它的关系能力来管理数据库



在关系一级上支持数据的插入、删除、修改，
没有任何操作必须通过非关系的能力才能实现

第四章 关系系统及其查询优化

准则1：信息准则。逻辑上可用一种方法（表中的值）来表示所有信息。

好处：

- ◆ 提高用户生产率
- ◆ 便于DBA维护数据库
- ◆ 便于与其它软件接口

准则2：保证访问准则。依靠表名、主键、列名的组合，保证能以逻辑方式（而不是物理方式）访问到每一个数据项。

准则3：空值的系统化处理。

好处：

- ◆ 完善完整性约束
- ◆ 对库函数计算的准确性极为重要

第四章 关系系统及其查询优化

准则4： 基于关系模型的动态的联机数据字典。

数据库自身的描述（元数据）也用关系，且授权用户也可以查询。

好处： ■ 学习简单

■ 授权用户可扩充数据字典

准则5： 统一的数据子语言准则。

一种语言全面支持以下功能：

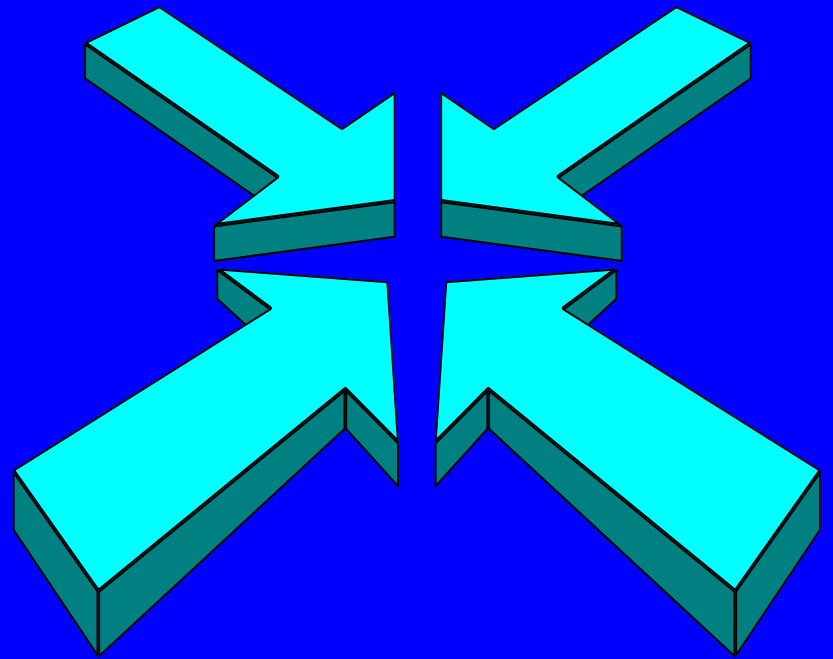
数据定义、视图定义

数据操作

完整性约束

授权

事务处理功能



准则6：视图更新准则。所有理论上可更新的视图也应该允许由系统更新。

提高逻辑独立性

准则7：高级的插入、修改、删除操作。把一个基本关系或导出关系作为单一的操作对象进行处理。

好处：

- ◆ 简化用户操作
- ◆ 便于系统优化
- ◆ 便于分布式处理

准则8：数据物理独立性。

准则9：数据逻辑独立性。

准则10：数据完整性的独立性。

完整性约束条件必须是用数据子语言定义并存储在数据字典中。

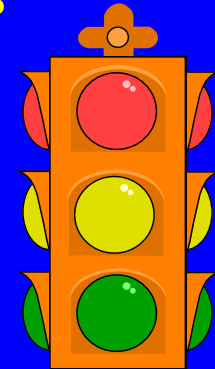
准则11：分布独立性。

数据子语言能使应用程序和终端活动在下列情况下保持逻辑不变性：

☆ 首次分布数据时； ⌚ 数据重新分布时。

准则12：无破坏准则。

如果一个关系系统具有一个低级（指一次一记录）语言，则这个低级语言不能违背或绕过完整性准则。



本节开头

本章开头

下一节

第四章 关系系统及其查询优化

§ 2 关系系统的查询优化

关系数据语言只需用户指出“干什么”，不必指出“怎么干”，为什么能做到这一点？

一个重要原因就是系统能自动进行查询优化。

查询优化的总目标：

选择有效的策略，求得给定的关系表达式的值。

一、为什么要进行查询优化？

例：求选修了课程C2的学生姓名

```
SELECT S.SN  
FROM S, SC  
WHERE S.S# = SC.S# AND SC.C# = 'C2';
```

也可用SQL语言如下实现:

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

对于一个复杂的查询，不同的用户可能会写出许许多多不同的查询方法。这些方法有的简单，有的复杂。它们的执行结果是一样的，但执行效率可能是不一样的。系统能解决这一问题吗？

对这一查询，可以考虑下面几种实现方式：

- 1、先求S和SC的笛卡尔积，然后从中选出两学号字段值相等、课程号为C2的元组：

$$Q1 = \pi_{SN}(\sigma_{S.S\#=SC.S\# \wedge SC.C\#='C2'}(S \times SC))$$

- 2、先做S和SC的自然连接，然后从中选出课程号为C2的元组：

$$Q2 = \pi_{SN}(\sigma_{SC.C\#='C2'}(S \bowtie SC))$$

- 3、先从SC中选出课程号为C2的元组，然后将该结果与S连接：

$$Q3 = \pi_{SN}(S \bowtie \sigma_{SC.C\#='C2'}(SC))$$

第四章 关系系统及其查询优化

分析三种实现策略的执行时间：

设有1000学生记录，10000选课记录，选修C2课程的学生有50名。

1、第一种策略：

$$Q1 = \pi_{SN}(\sigma_{S.S\#=SC.S\# \wedge SC.C\#='C2'}(S \times SC))$$

(1) 计算广义笛卡尔积 $S \times SC$ ：

$S \times SC$ 存于临时文件中

执行方式：

读S表

S1	A	CS	20
S2	B	CS	21
S3	C	MA	19
S4	D	CI	19
S5	E	MA	22
S6	F	CS	19

每次读若干块

读SC表

S1	C1	A
S1	C2	A
S1	C3	A

每次读一块

S1	A	CS	20	S1	C1	A
S1	A	CS	20	S1	C2	A
S1	A	CS	20	S1	C3	A
S2	B	CS	21	S1	C1	A
S2	B	CS	21	S1	C2	A
S2	B	CS	21	S1	C3	A
.....					
S6	F	CS	19	S1	C1	A
S6	F	CS	19	S1	C2	A
S6	F	CS	19	S1	C3	A

第四章 关系系统及其查询优化

读S表

S1	A	CS	20
S2	B	CS	21
S3	C	MA	19
S4	D	CI	19
S5	E	MA	22
S6	F	CS	19

读SC表

S1	C5	B
S2	C1	B
S2	C2	C

读下一块

S×SC存于临时文件中

S1	A	CS	20	S1	C1	A
S1	A	CS	20	S1	C2	A
S1	A	CS	20	S1	C3	A
S2	B	CS	21	S1	C1	A
S2	B	CS	21	S1	C2	A
S2	B	CS	21	S1	C3	A
.....					
S6	F	CS	19	S1	C1	A
S6	F	CS	19	S1	C2	A
S6	F	CS	19	S1	C3	A
S1	A	CS	20	S1	C5	B
S1	A	CS	20	S2	C1	B
S1	A	CS	20	S2	C2	C
.....					

第四章 关系系统及其查询优化

设一块能装10个学生记录或100个选课记录，每次在内存中存放5块S的元组、1块SC的元组，则S表和SC表的总块数各为100。外循环一次，内循环20次，要读取的总块数为

$$100 + 100 \times 20 = 2100 \text{ 块}$$

连接后的元组数为 $1000 \times 10000 = 1$ 千万，设每块能装10个元组，则 $S \times SC$ 的总块数为1百万块

设每秒能读写20块，则

读的时间： $2100 \text{ 块} \div 20 \text{ 块/秒} = 105 \text{ 秒}$

写的时间： $1000000 \text{ 块} \div 20 \text{ 块/秒} = 50000 \text{ 秒}$

(2) 依次读入 $S \times SC$ 的元组，然后执行选择：

读的时间： $1000000 \text{ 块} \div 20 \text{ 块/秒} = 50000 \text{ 秒}$

满足条件的元组50个，设全部放入内存（不再临时存储）

第四章 关系系统及其查询优化

(3) 在上步基础上执行投影得最终结果（此步时间不计）。

第一种策略的总时间为：

$$105 + 50000 + 50000 \approx 10\text{万秒（近28小时）}$$

2、第二种策略

$$Q2 = \pi_{SN}(\sigma_{SC.C\#='C2'}(S \bowtie SC))$$

(1) 计算自然连接

读取S表和SC表的策略不变，执行时间还是105秒。

因为SC表中的每一个学号都在S表中出现，而S表中无重复学号，故连接后的表和SC表的行数一样，为10000行，将它们临时存入盘中需

$$(10000 \div 10) \text{ 块} \div 20 \text{ 块/秒} = 50 \text{ 秒}$$

计算自然连接需时：105 + 50 = 155秒

(2) 执行选择运算

主要为读取中间文件的时间：为50秒

(3) 把上一步结果投影，时间忽略不计

第二种策略的总时间为： $155 + 50 = 205$ 秒

3、第三种策略

$$Q3 = \pi_{SN}(S \bowtie \sigma_{SC.C\#='C2'}(SC))$$

(1) 先对SC表作选择

只需读一遍SC表，需时 $100 \text{块} \div 20 \text{块/秒} = 5 \text{秒}$

中间结果只有50个记录，不需使用中间文件

(2) 作自然连接

只需读一遍S表，边读边和内存中的中间结果连接，结果仍为50个元组需时 5秒

(3) 把上一步结果投影，时间忽略不计

第三种策略的总时间为： $5 + 5 = 10$ 秒

二、优化的一般策略

1、选择、投影运算应尽可能先做

好处：减少下一步运算的数据量

2、把选择和投影运算同时进行

好处：减少扫描关系的次数

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

表S

S#	SN	SD	SA
----	----	----	----

S1	A	CS	20
----	---	----	----

S2	B	CS	21
----	---	----	----

S3	C	MA	19
----	---	----	----

S4	D	CI	22
----	---	----	----

.....

第四章 关系系统及其查询优化

3、在执行连接前对文件适当地预处理

SC:

S#	C#	G
S4	C3	B
S1	C2	A
S1	C5	B
S6	C4	A
S2	C1	B
S5	C3	B
S2	C2	C
S1	C1	A
S2	C4	C
S3	C2	B
S1	C3	A
S3	C3	C
S4	C5	D
S5	C2	C
S3	C4	B
S5	C5	B
S6	C5	A

例如：计算 $S \bowtie SC$

S: S# SN SD SA

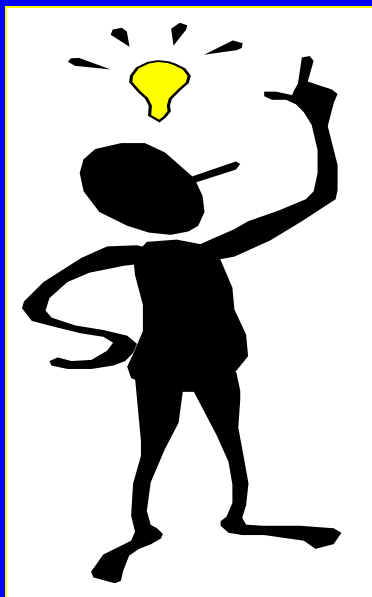
S1	A	CS	20
S2	B	CS	21
S3	C	MA	19
S4	D	CI	19
S5	E	MA	20
S6	F	CS	22

S: S# SN SD SA SC: S# C# G

S1	A	CS	20
S2	B	CS	21
S3	C	MA	19
S4	D	CI	19
S5	E	MA	20
S6	F	CS	22

S1	C1	A
S1	C2	A
S1	C3	A
S1	C5	B
S2	C1	B
S2	C2	C
S2	C4	C
S3	C2	B
S3	C3	C
S3	C4	B
S4	C3	B
S4	C5	D
S5	C2	C
S5	C3	B
S5	C5	B
S6	C4	A
S6	C5	A

对S表和SC表都
只需一遍扫描



效率就是高!

4、把投影同其前或其后的双目运算结合起来

如： $\pi_{S\#, SN, C\#} (S \bowtie SC)$

每形成一个连接后的元组，就立即取出投影字段。而不是先连接形成一个临时关系，然后在再此临时关系上投影。

又如： $SC \bowtie (\pi_{S\#, SN} (S))$

每取出S的一个元组，先取出投影字段，然后与SC进行连接

5、把某些选择和笛卡尔乘积结合起来成为连接运算

6、找出公共子表达式

三、关系代数等价变换规则

设E、E1、E2是关系代数表达式。

1、关系代数表达式的等价

若用相同的关系代替E1、E2中相应的关系变量后所得的结果关系相同，则称E1、E2等价，记作 $E1 \equiv E2$ 。

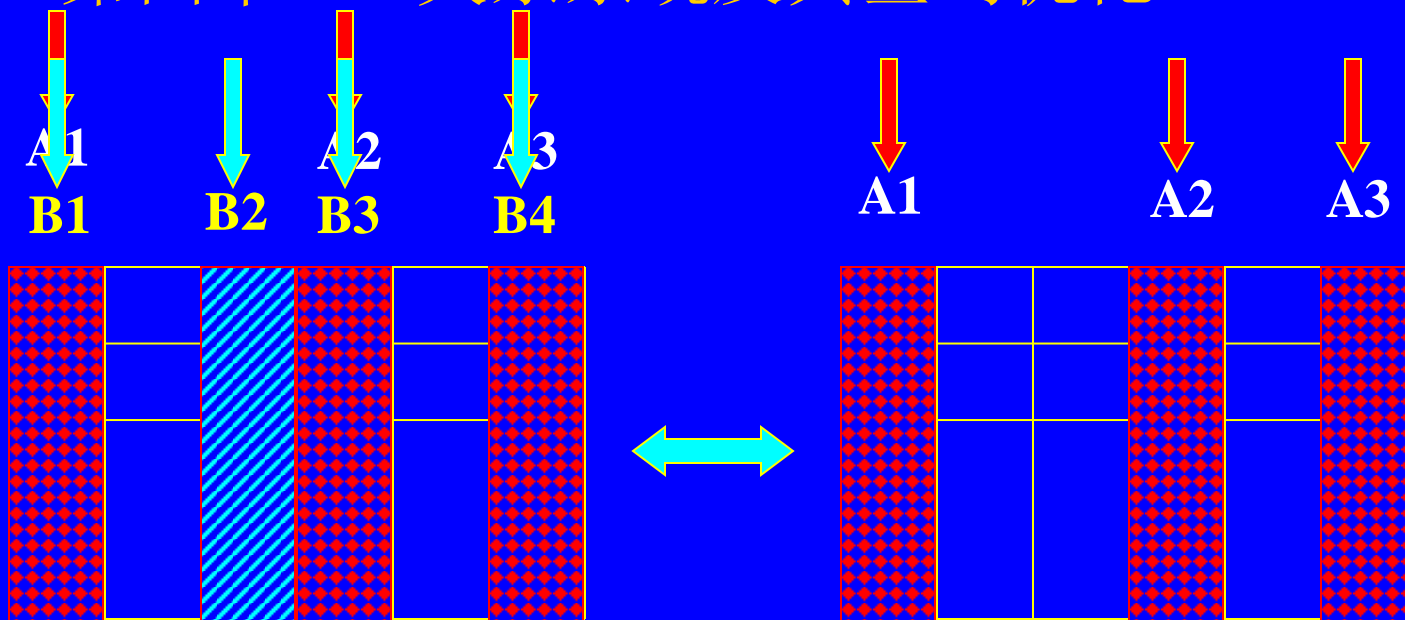
2、一元运算的串接定律（幂等律）

（1）投影的串接定律

$$\pi_{A1,A2,\dots,A_n}(\pi_{B1,B2,\dots,B_m}(E)) \equiv \pi_{A1,A2,\dots,A_n}(E)$$

其中 $\{A1,A2,\dots,A_n\} \subseteq \{B1,B2,\dots,B_m\}$

图示:



(2) 选择的串接定律

$$\sigma_{F1}(\sigma_{F2}(E)) \equiv \sigma_{F1 \wedge F2}(E)$$

3、二元运算的交换律

笛卡尔积: $E1 \times E2 \equiv E2 \times E1$

连接: $E1 \bowtie_F E2 \equiv E2 \bowtie_F E1$

自然连接: $E1 \bowtie E2 \equiv E2 \bowtie E1$

4、二元运算的结合律

笛卡尔积: $(E1 \times E2) \times E3 \equiv E1 \times (E2 \times E3)$

自然连接: $(E1 \bowtie_{F1} E2) \bowtie_{F2} E3 \equiv E1 \bowtie_{F1} (E2 \bowtie_{F2} E3)$

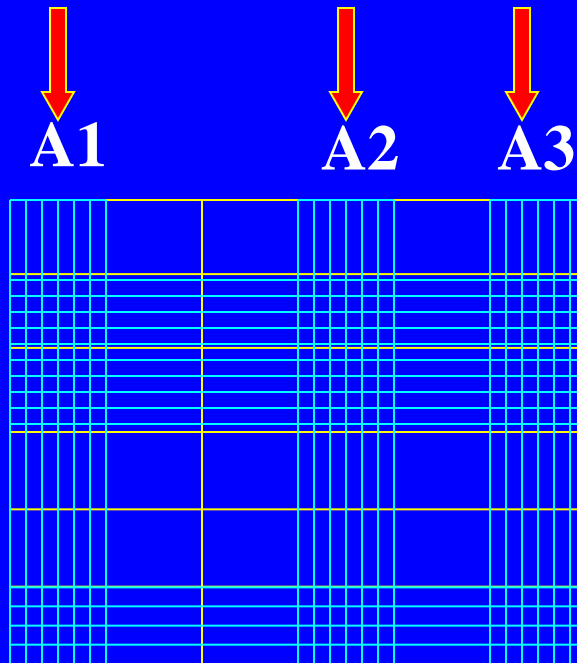
自然连接: $(E1 \bowtie E2) \bowtie E3 \equiv E1 \bowtie (E2 \bowtie E3)$

5、两个运算间的交换律

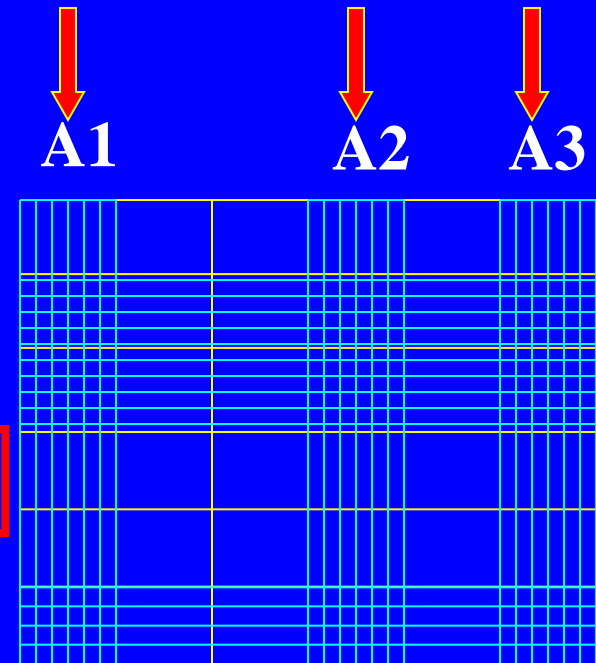
(1) 选择和投影:

$$\sigma_F(\pi_{A1,A2,\dots,A_n}(E)) \equiv \pi_{A1,A2,\dots,A_n}(\sigma_F(E))$$

其中F只涉及{A1,A2,...,An}的属性



先投影后选择



先选择后投影

结果相同

第四章 关系系统及其查询优化

若F中不属于 $\{A_1, A_2, \dots, A_n\}$ 的属性 $\{B_1, B_2, \dots, B_m\}$, 则

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

但根据投影的串接定律和上面的投影与选择的交换律, 有:

$$\begin{aligned} & \pi_{A_1, A_2, \dots, A_n}(\sigma_F(E)) \\ & \equiv \pi_{A_1, A_2, \dots, A_n}(\pi_{A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m}(\sigma_F(E))) \\ & \equiv \pi_{A_1, A_2, \dots, A_n}(\sigma_F(\pi_{A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m}(E))) \end{aligned}$$

(2) 选择与笛卡尔积

若F只涉及到E1中的属性, 则 $\sigma_F(E_1 \times E_2) \equiv \sigma_F(E_1) \times E_2$

6、一元运算对二元运算的分配律

(1) 选择对笛卡尔积的分配律

若 $F=F_1 \wedge F_2$, F_1 只涉及 E_1 中的属性, F_2 只涉及 E_2 中的属性,

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

如: $\sigma_{SD='CS' \wedge G='A'}(S \times SC)$

$$\equiv \sigma_{SD='CS'}(S) \times \sigma_{G='A'}(SC)$$

↓ S表

S1	A	CS	20
S2	B	MA	21
S3	C	CS	19

↓ SC表

S1	C1	A
S1	C2	A
S2	C2	A
S3	C1	B
S3	C3	A

笛卡尔积

S1	A	CS	20	S1	C1	A
S1	A	CS	20	S1	C2	A
S1	A	CS	20	S2	C2	A
S1	A	CS	20	S3	C1	B
S1	A	CS	20	S3	C3	A
S2	B	MA	21	S1	C1	A
S2	B	MA	21	S1	C2	A
S2	B	MA	21	S2	C2	A
S2	B	MA	21	S3	C1	B
S2	B	MA	21	S3	C3	A
S3	C	CS	19	S1	C1	A
S3	C	CS	19	S1	C2	A
S3	C	CS	19	S2	C2	A
S3	C	CS	19	S3	C1	B
S3	C	CS	19	S3	C3	A

(2) 投影对笛卡尔积的分配律

$$\begin{aligned} & \pi_{A1,A2,\dots,A_n, B1,B2,\dots,B_m} (E1 \times E2) \\ & \equiv \pi_{A1,A2,\dots,A_n} (E1) \times \pi_{B1,B2,\dots,B_m} (E2) \end{aligned}$$

其中A1,A2,...,An是E1的属性, B1,B2,...,Bm是E2的属性。

(3) 选择对连接的分配律

若F=F1∧F2, F1只涉及E1的属性, F2只涉及E2的属性, 则

$$\sigma_F (E1 \bowtie_{F3} E2) \equiv \sigma_{F1} (E1) \bowtie_{F3} \sigma_{F2} (E2)$$

因为:

$$\begin{aligned} \sigma_F (E1 \bowtie_{F3} E2) & \equiv \sigma_F (\sigma_{F3} (E1 \times E2)) \equiv \sigma_{F3} (\sigma_F (E1 \times E2)) \\ & \equiv \sigma_{F3} (\sigma_{F1} (E1) \times \sigma_{F2} (E2)) \equiv \sigma_{F1} (E1) \bowtie_{F3} \sigma_{F2} (E2) \end{aligned}$$

(4) 投影对连接的分配律

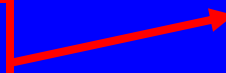
$$\begin{aligned} & \pi_{A1,A2,\dots,A_n, B1,B2,\dots,B_m} (E1 \underset{F}{\bowtie} E2) \\ & \equiv \pi_{A1,A2,\dots,A_n} (E1) \underset{F}{\bowtie} \pi_{B1,B2,\dots,B_m} (E2) \end{aligned}$$

其中F只涉及 $\{A1,A2,\dots,A_n, B1,B2,\dots,B_m\}$ 中的属性

若F涉及 $\{A1,A2,\dots,A_n, B1,B2,\dots,B_m\}$ 以外的属性，可如下处理

$$\begin{aligned} & \pi_{SN,C\#,G} (S \underset{S.S\#=SC.S\#}{\bowtie} SC) \\ & \equiv \pi_{SN,C\#,G} \pi_{S.S\#,SN,SC.S\#,C\#,G} (S \underset{S.S\#=SC.S\#}{\bowtie} SC) \\ & \equiv \pi_{SN,C\#,G} (\pi_{S\#,SN} (S) \underset{S.S\#=SC.S\#}{\bowtie} SC) \end{aligned}$$

此处投影可去掉



(5) 选择对并的分配律

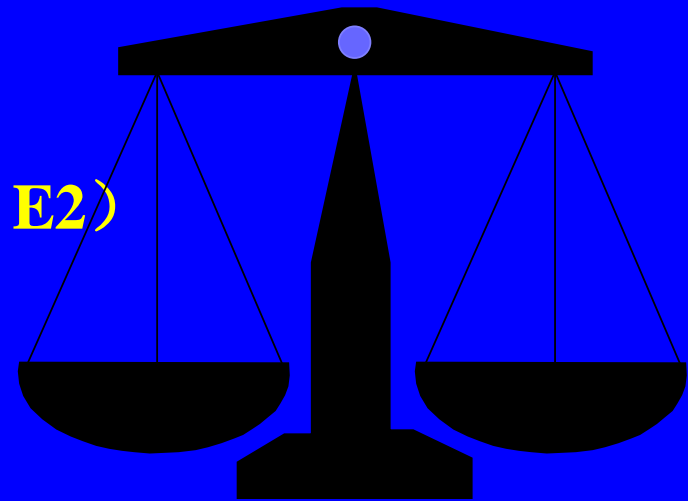
$$\sigma_F(E1 \cup E2) \equiv \sigma_F(E1) \cup \sigma_F(E2)$$

(6) 投影对并的分配律

$$\begin{aligned} & \pi_{A1, A2, \dots, An}(E1 \cup E2) \\ & \equiv \pi_{A1, A2, \dots, An}(E1) \cup \pi_{A1, A2, \dots, An}(E2) \end{aligned}$$

(7) 选择对差的分配律

$$\sigma_F(E1 - E2) \equiv \sigma_F(E1) - \sigma_F(E2)$$



四、关系代数表达式的优化

1、语法树

用来表示关系代数表达式的一棵树，其内结点表示一种运算，叶结点表示一个关系。例：

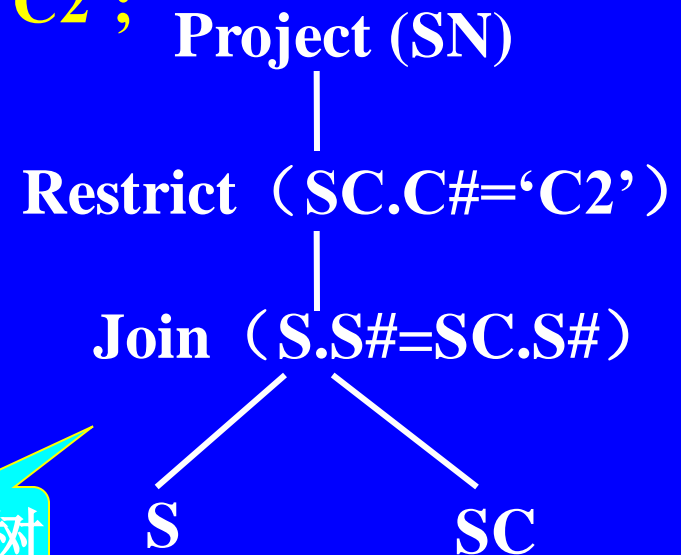
SELECT S.SN

FROM S, SC

WHERE S.S# = SC.S# AND SC.C# = 'C2'; **Project (SN)**

可转化为如下关系运算：

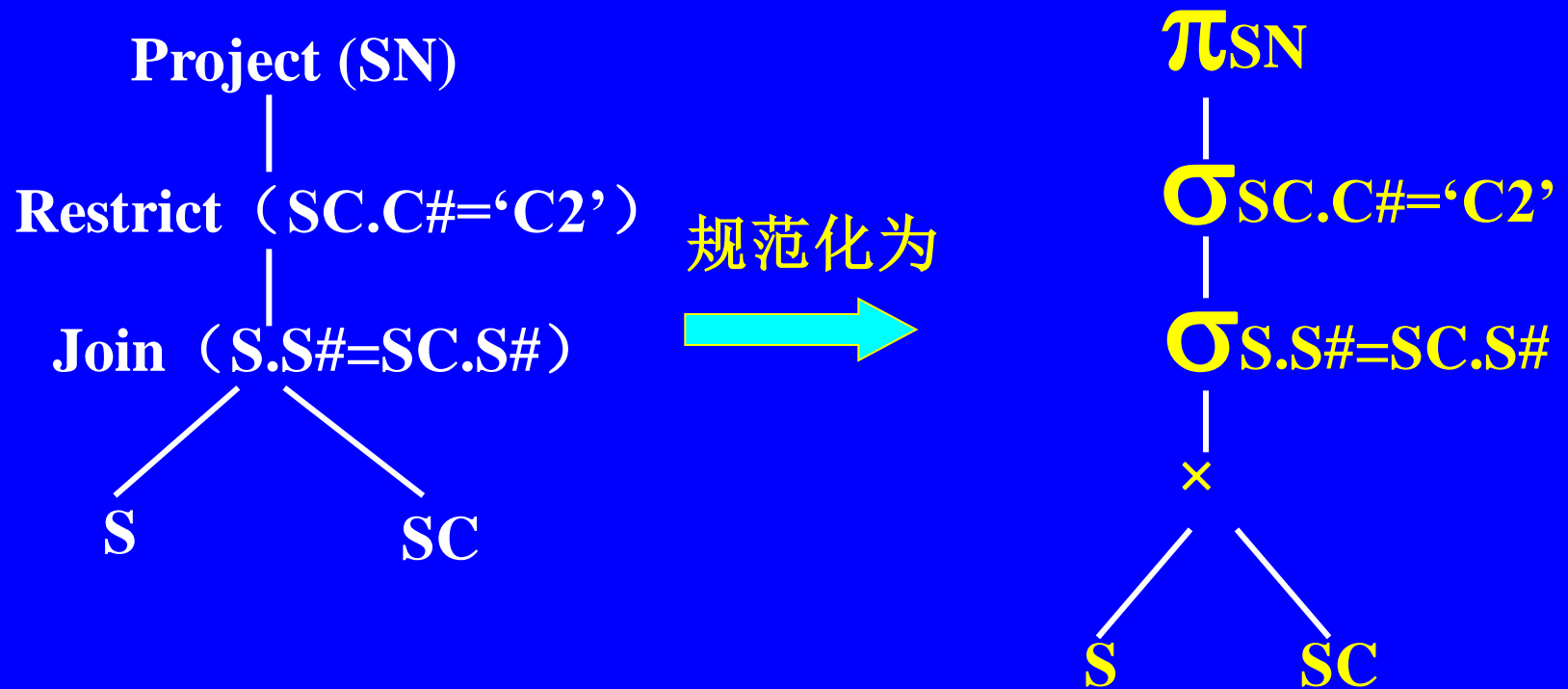
**Project (SN) (Restrict (SC.C#='C2')
(Join (S.S#=SC.S#) (S, SC)))**



语法树

第四章 关系系统及其查询优化

为简化优化算法，可将关系代数运算限制在“并、差、笛卡尔积、投影、选择”五种基本运算上。



2、关系代数表达式的优化算法

输入：一棵关系代数表达式的语法树

输出：计算该表达式的程序

(1) 分解选择

利用选择的串接定律，把形如 $\sigma_{F1 \wedge F2 \wedge \dots \wedge Fn}(E)$ 的式子变换为

$$\sigma_{F1}(\sigma_{F2}(\dots(\sigma_{Fn}(E)\dots)))$$

(2) 选择下移

对每一个选择，利用“选择的串接定律、选择和投影的交换律、选择对笛卡尔积的分配律、选择对并的分配律、选择对差的分配律”尽可能把它移到树的叶端

(3) 投影下移

对每一个投影，利用“投影的串接定律、选择和投影的交换律、投影对笛卡尔积的分配律、投影对并的分配律”尽可能把它移到树的叶端

(4) 选择、投影合并

利用“投影的串接定律、选择的串接定律、选择和投影的交换律”把选择和投影合并成单个选择、单个投影、或选择后跟投影等三种情况，使多个选择和 / 或投影能同时执行、或在一次扫描中完成

(5) 按点分组（每组只有一个二元运算）

把上面得到的语法树分组：

每个二元运算和它的一元直接祖先为一组。若它的后代直到叶子全是一元运算，则也将它们并入该组。

但对于笛卡尔积，若后面（父结点）是不能与它结合为等值连接的选择运算时，其一直到叶子的一元运算结点需单独算一组。

(6) 生成程序

按照每组的求值应在其后代组求值之后进行的顺序为每组生成一个程序，以产生整个表达式的求值程序。

第四章 关系系统及其查询优化

例子：考虑由以下关系组成的图书馆数据库

BOOKS (TITLE, AUTHOR, PNAME, LC-NO)

BORROWERS (NAME, ADDR, CITY, CARD-NO)

LOANS (CARD-NO, LC-NO, DATE)

借书证号 图书编号 借出日期

用SQL语言可如下表达：

SELECT TITLE, NAME

FROM BOOKS, BORROWERS, LOANS

WHERE BOOKS.LC-NO=LOANS.LC-NO AND

BORROWERS.CARD-NO=LOANS.CARD-NO AND

DATE < {2000-01-01};

第四章 关系系统及其查询优化

SELECT TITLE, NAME  
FROM BOOKS, BORROWERS, LOANS  转化为连接
WHERE BOOKS.LC-NO=LOANS.LC-NO AND
BORROWERS.CARD-NO=LOANS.CARD-NO AND
DATE < {2000-01-01};  转化为选择

把上述SQL语句转化为关系代数表达式:

$\pi_{\text{TITLE, NAME}} (\sigma_{\text{DATE} < \{2000-01-01\}}$

$(\text{BOOKS} \bowtie (\text{BORROWERS} \bowtie \text{LOANS})))$

若把连接用笛卡尔积来实现，上式变为：

$$\begin{aligned} & \pi_{\text{TITLE, NAME}} (\\ & \quad \sigma_{\text{DATE} < \{2000-01-01\}} (\\ & \quad \quad \pi_{\text{TITLE, AUTHOR, PNAME, LC-NO,}} (\\ & \quad \quad \quad \text{NAME, ADDR, CITY, CARD-NO, DATE} \\ & \quad \quad \sigma_{\text{BOOKS.LC-NO=LOANS.LC-NO AND}} \\ & \quad \quad \quad \text{BORROWERS.CARD-NO=LOANS.CARD-NO} \\ & \quad \quad (\text{BOOKS} \times (\text{BORROWERS} \times \text{LOANS}))))) \end{aligned}$$



π TITLE, NAME

σ DATE < {2000-01-01}

π TITLE, AUTHOR, PNAME, LC-NO,
NAME, ADDR, CITY, CARD-NO, DATE

σ BOOKS.LC-NO=LOANS.LC-NO

σ BORROWERS.CARD-NO=LOANS.CARD-NO

x

BOOKS

x

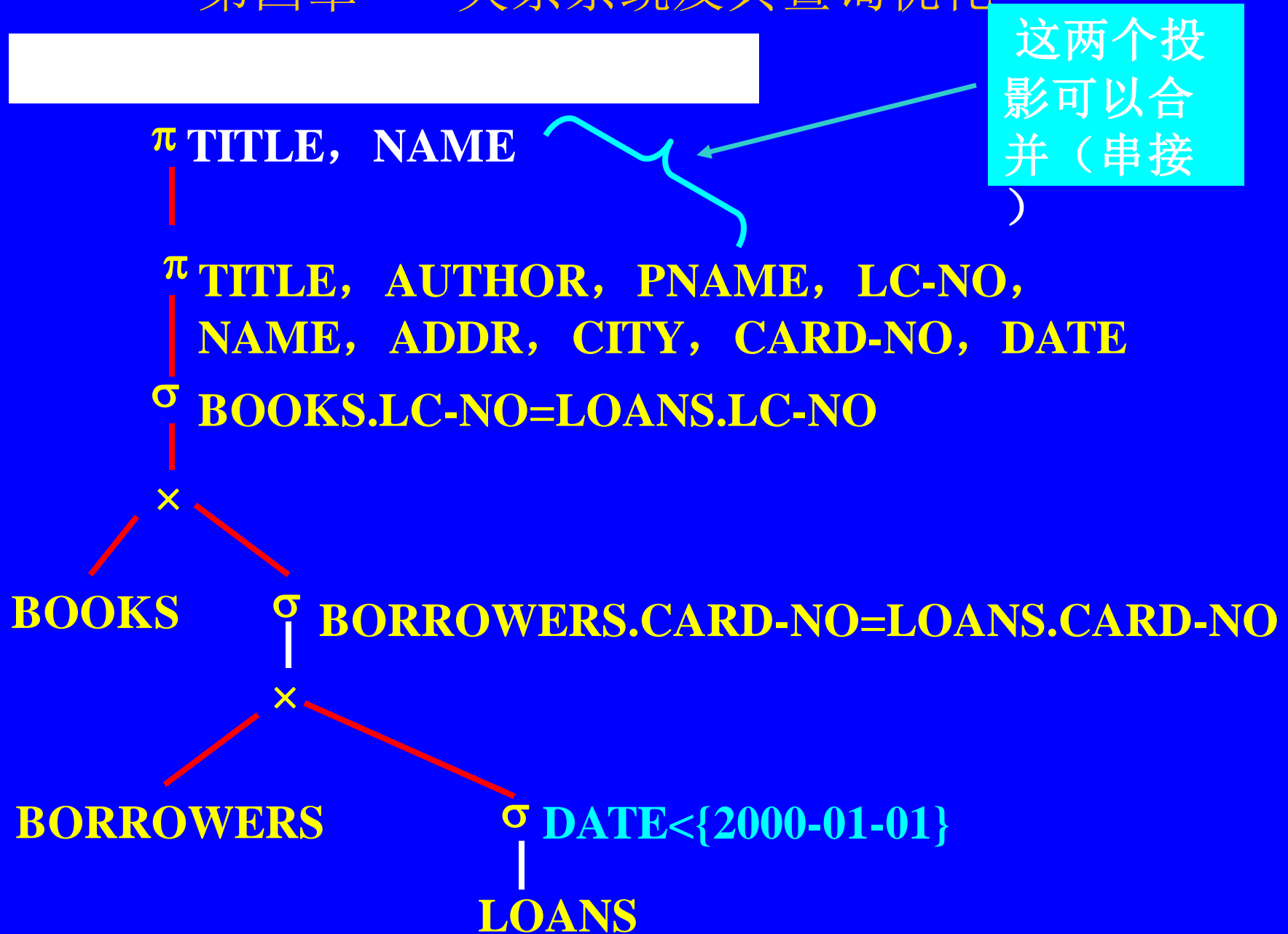
BORROWERS

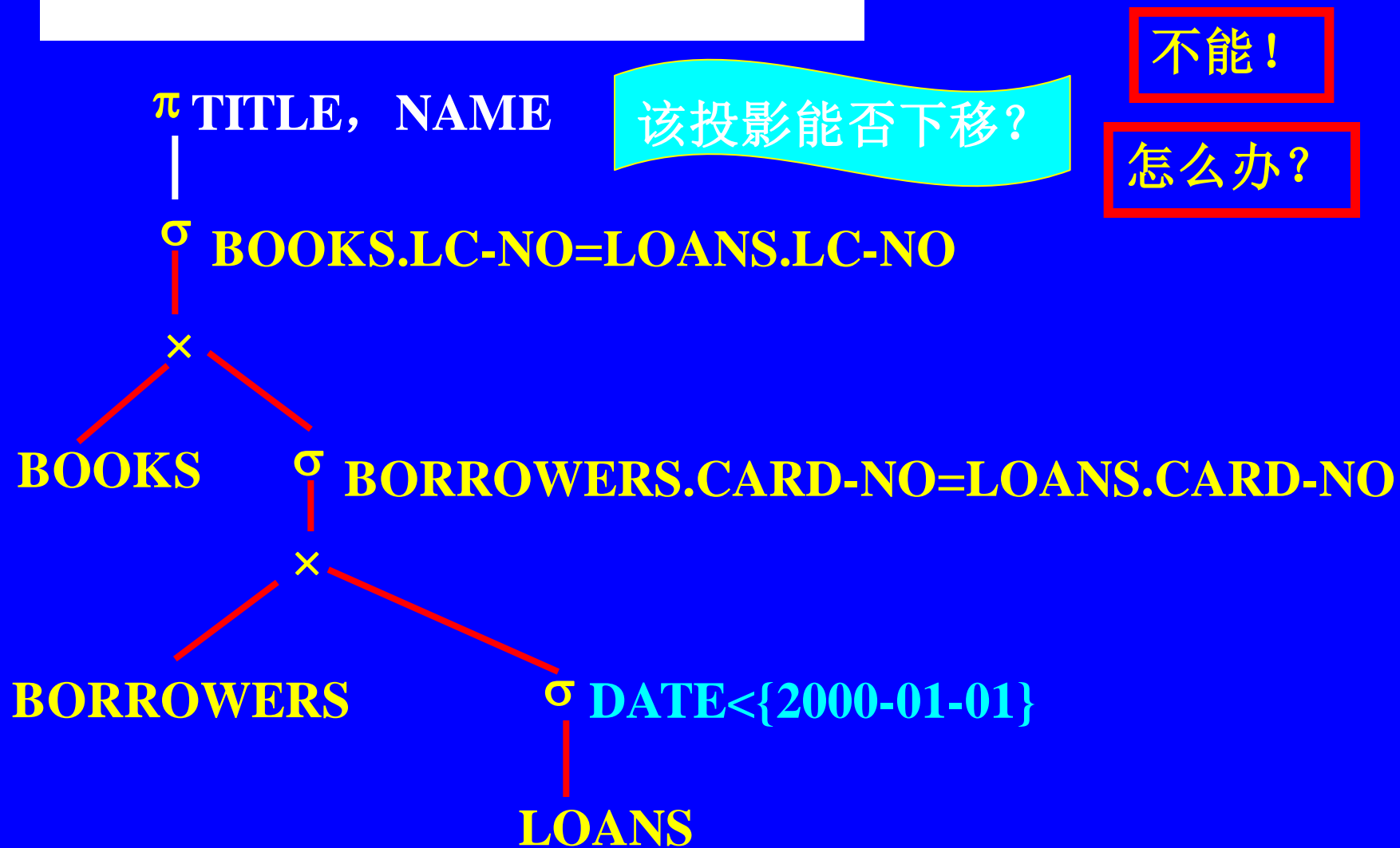
LOANS

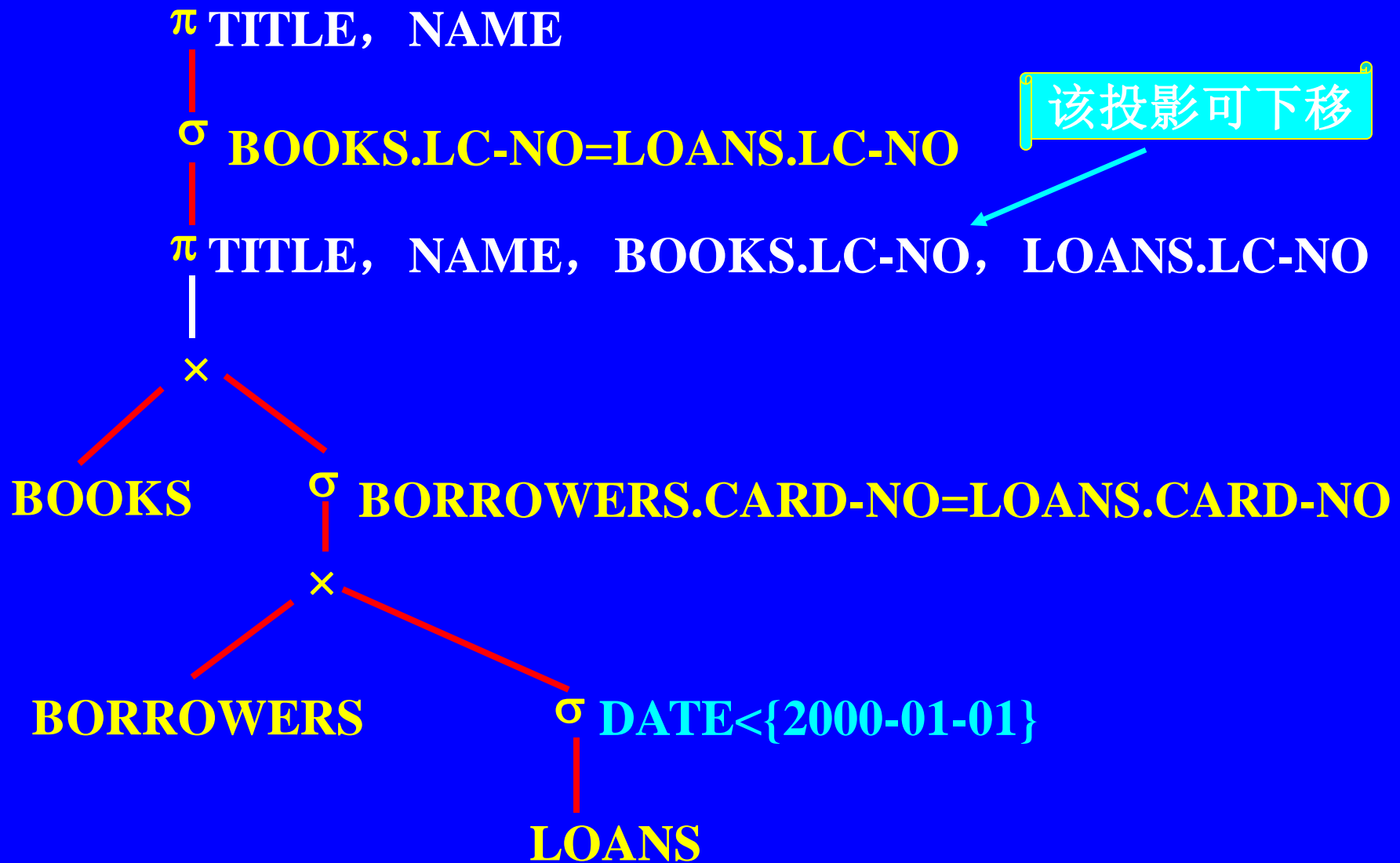
选择和投影
可以交换

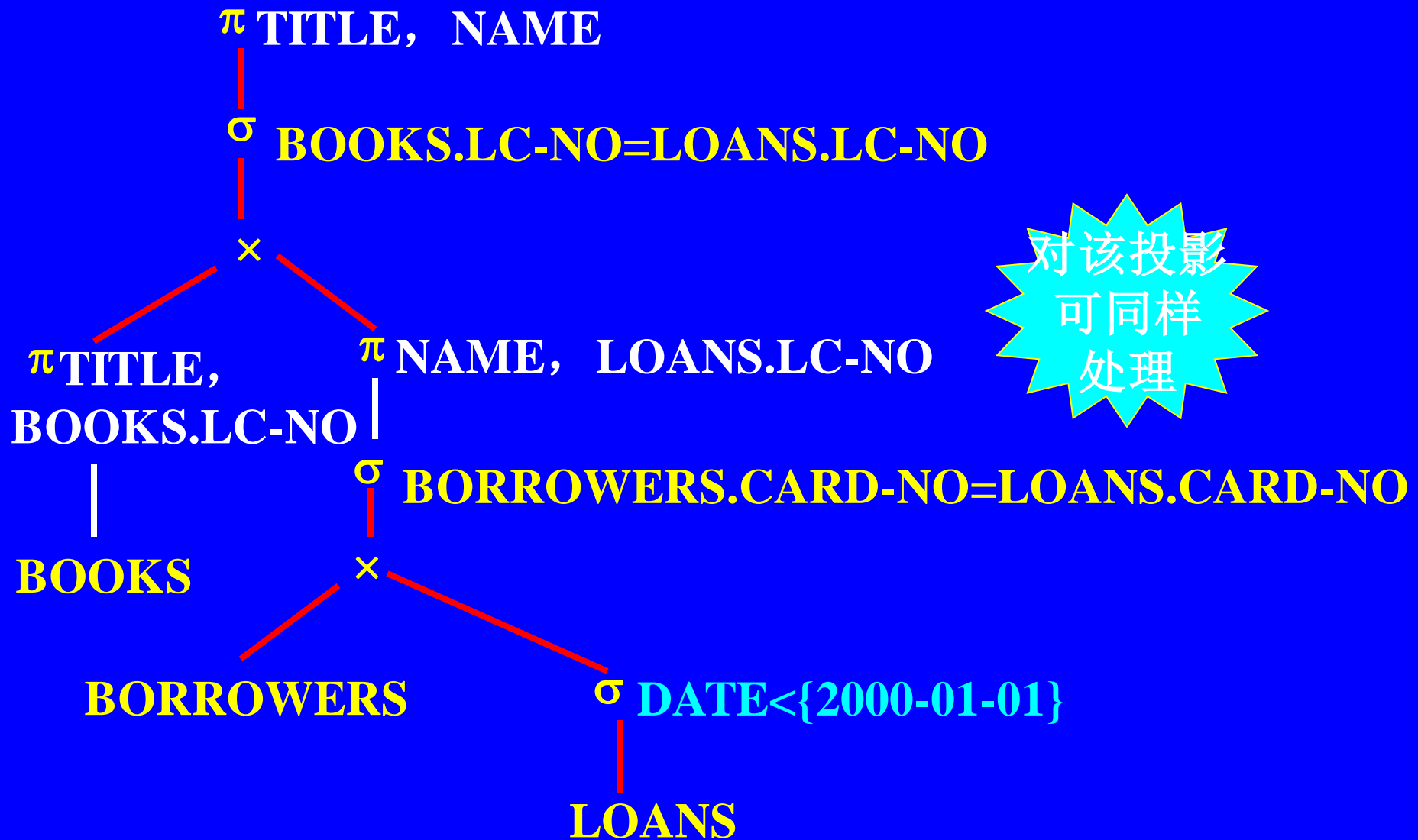
分解之后

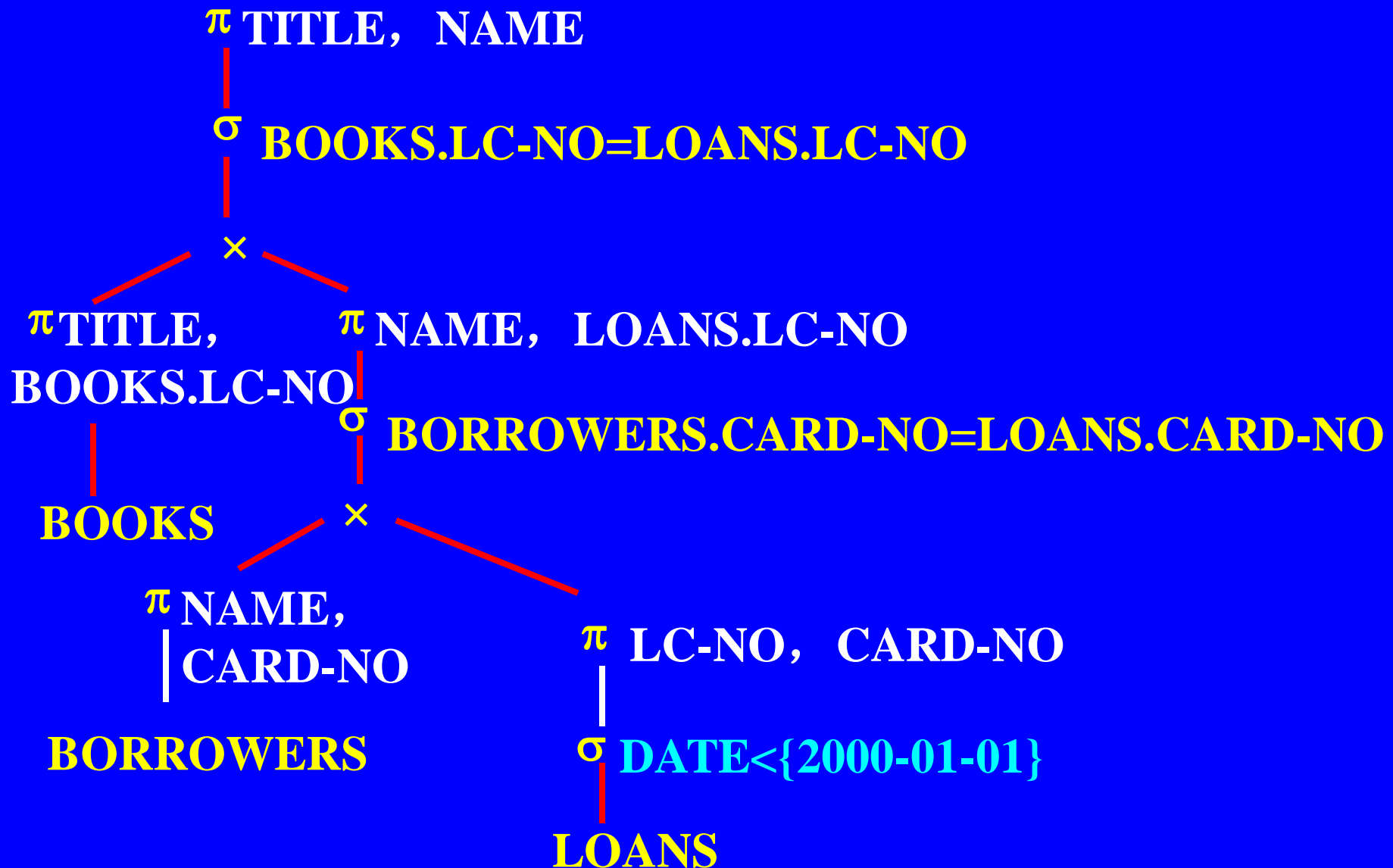


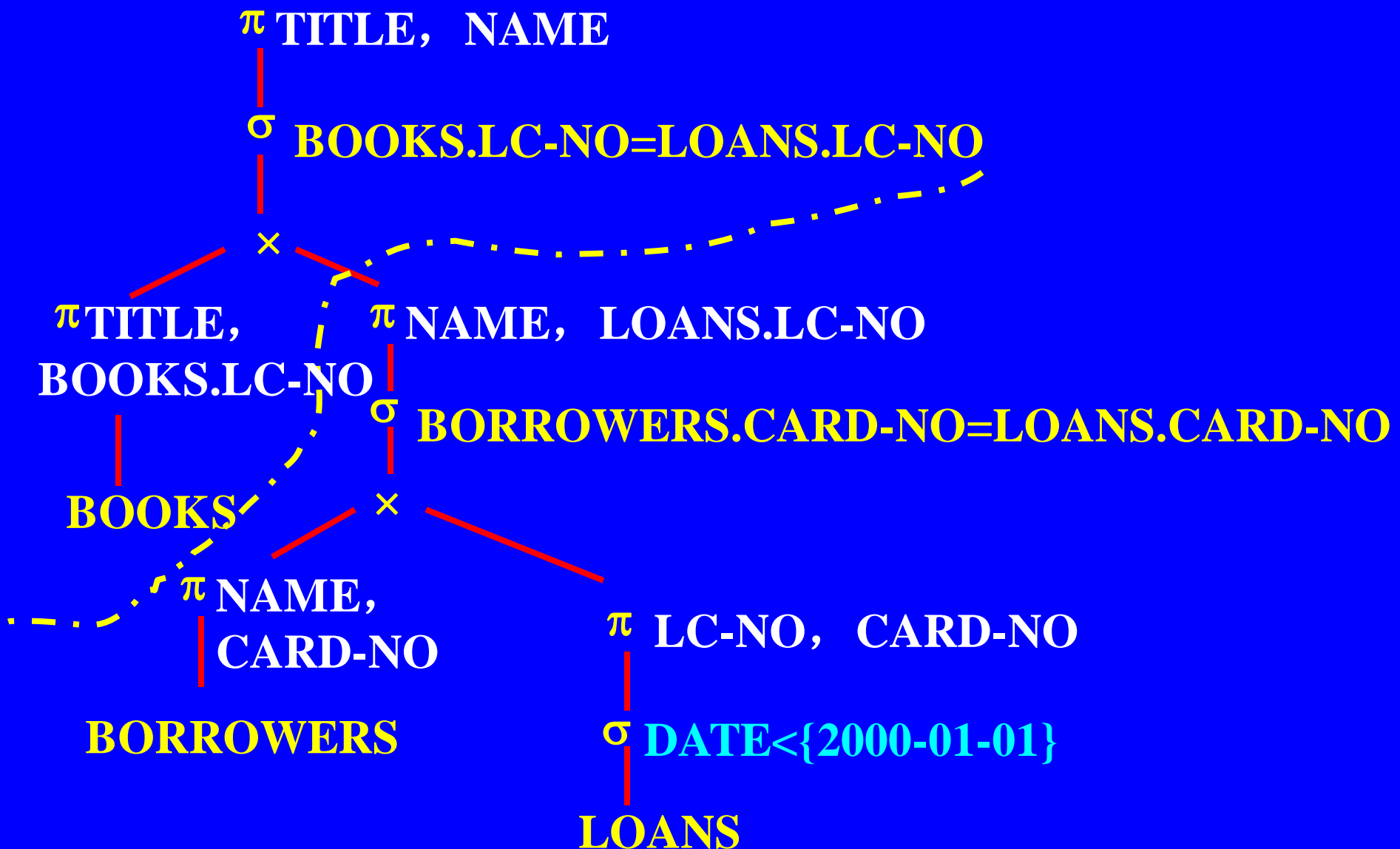












3、优化的一般步骤

(1) 把查询转换成某种内部表示

如语法树

(2) 按优化算法对语法树进行优化（标准形）

(3) 选择低层的存取路径

充分考虑索引、数据的存储分布，然后选取存储路径

(4) 生成查询计划，选择代价最小的

如对连接运算，考虑各种实现策略，从中选出代价最小的

作业：1、4

本节开头

本章开头

本章要求：

- 1、掌握为什么不合适的关系模式会带来插入异常、删除异常、存储异常、修改困难等严重问题
- 2、深刻理解函数依赖、多值依赖等有关概念
- 3、掌握关系的1NF、2NF、3NF、BCNF、4NF的概念和特征
- 4、掌握函数依赖的Armstrong公理系统、求属性集的闭包算法以及求极小函数依赖集的方法等
- 5、掌握模式分解的无损连接性和保持函数依赖性以及分解算法

本章内容：

- § 1 为什么需要对关系模式规范化？
- § 2 数据依赖
- § 3 关系的范式
- § 4 函数依赖的Armstrong公理系统
- § 5 关系模式的分解

请选择内容

返回

第五章 关系数据理论

§ 1 问题的提出

一个基本的问题：给出一组数据，如何构造一个合适的数据库模式？

例如：对关系模型，给了一组数据，应该构造几个关系？每个关系由哪些属性组成？

这就是数据库逻辑设计问题

网状、层次模型的数据库设计，主要凭设计者的经验直观地选择和确定实体集、属性以及实体间的联系。哪些实体应该合并或分解以及如何合并和分解、每个实体中应该包括哪些属性为宜、属性间的联系如何确定和处理等一系列问题的解决是没有什么固定规则和理论可循的。

第五章 关系数据理论

关系数据库的设计是借助近代数学工具而提出来的，形成了一整套定义、公理、定理及各种实用算法，产生了确定、评价关系数据库模式的好方法。

关系数据库的规范化理论

——数据库逻辑设计的有力工具

要考虑的几个问题：

- 为什么要规范化？
- 怎样规范化？
- 规范化到什么程度后最合适？

这就是本章的主题

本节首先用一个例子来说明对关系模式为什么要规范化，不经过规范化会产生什么样的结果。

第五章 关系数据理论

例：假设车间考核职工完成生产定额的关系模式如下：

W (工号, 日期, 姓名, 工种, 定额, 超额, 车间, 车间主任)

比如设某工号某年月超额完成定额的20%，其记录的内容为：

(1001, 98年11月, 张三, 车工, 180, 20%, 金工车间, 李四)

该关系的主键为？ 工号 + 日期

该关系模式存在以下四个严重问题：

(1) 数据冗余大

对同一个人来说，其姓名、工种、车间、车间主任等多次重复

.....

1001, 98年08月, 张三, 车工, xxx, xxx, 金工车间, 李四

1001, 98年09月, 张三, 车工, xxx, xxx, 金工车间, 李四

1001, 98年10月, 张三, 车工, xxx, xxx, 金工车间, 李四

1001, 98年11月, 张三, 车工, xxx, xxx, 金工车间, 李四

.....

(2) 插入异常

应该存储的信息无法存储

若新调来一个职工并将他分配到某个车间，根据上述关系模式，在对该职工统计工作之前，他的信息是装不进数据库中的。因为他的日期值是空值，而日期是主键的属性之一，不允许为空。

(1005, NULL, 天然, 车工, NULL, NULL, 金工车间, 李四)

(3) 删除异常

不该删除的信息被删除

若想删除某人的所有定额完成情况，则该职工的其他信息也都被删除。

比如在98年底要删除97年以前的所有定额完成信息，则98年由于种种原因未参加现实工作的职工的所有信息全部被删除。

(4) 修改困难，容易造成数据的不一致性

(也称更新异常)

若某车间换了主任，则该车间所有职工的上述记录都要修改；
又如某人换了车间，则其工种、车间、车间主任等信息都要修改

。

修改工作量大；
即使漏改一处都会造成数据的不一致性

；

上例充分说明对关系模式若随意设计，其后果是严重的。

本章将要讨论产生上述问题的原因以及解决办法，即如何改造一个不好的关系模式。这就是规范化理论要解决的主要问题

。

第五章 关系数据理论

比如，对于上述关系模式，若分解成下面三个关系，则前面提到的几个问题将全部或部分地得到解决：

职工关系（工号，姓名，工种，车间号）

车间关系（车间号，车间名，车间主任）

定额关系（工号，日期，定额，超额，车间号）

原因何在？ 规律何在？

本节开头

本章开头

下一节

§ 2 数据依赖

数据模型中我们讨论了实体间的联系，同时提到实体内部属性间也有联系。事实上上一节中的问题都是由于属性间的联系引起的。

一、数据依赖

1、属性间的联系：也是1:1，1:n，m:n三种

☆ 1:1联系：设A、B为某实体集中的两个属性的值集，如果对于A中的任一值，B中至多有一个值与之对应，且反之亦然。

如：车间--主任

⌚ 1:n联系：设A、B为某实体集中的两个属性的值集，如果对于A中的任一值，B中有多个值（包括0个

如：队长--学号

与之对应；而对于B中的任一值，A中至多有一个值与之对应。

🕒 **m:n联系**：设A、B为某实体集中的两个属性的值集，

如：学号--课程号 如果对于A中的任一值，B中有多个值（包括0个）与之对应，且反之亦然。

实体间的联系表示实体之间相互依赖又相互制约的关系；
属性间的联系表示属性之间相互依赖又相互制约的关系。



2、数据依赖

通过一个关系中属性间值的相互关联（主要体现于值的相等与否）体现出来的数据间的相互联系。

（是数据内在的性质，语义的体现）

两类最重要的数据依赖 { 函数依赖
多值依赖

二、关系的形式化定义

1、关系的两个主要方面

语法：属性的描述

语义：数据依赖

2、关系模式：

$R \langle U, D, \text{dom}, F \rangle$

关系名 属性组

U上的一组数据依赖

3、关系：

对关系模式 $R \langle U, F \rangle$ ，当且仅当U上的一个关系r满足F时，称r为关系模式 $R \langle U, F \rangle$ 的一个关系。



三、函数依赖

不严格地讲，函数依赖指的是一组属性值唯一决定另一组属性值的这种数据依赖。

如学生关系中，当学号确定后，其姓名也就唯一确定了。

选课关系中，当学号和课程号确定后，其成绩也就唯一确定了。

1、函数依赖（Functional Dependency，缩写FD）：

设 $R(U)$ 是属性集 U 上的关系模式， X 、 Y 是 U 的子集。若对于 R 中的任意关系 r ，对于 r 中的任意两个元组 u 、 v 都有

$$u[X]=v[X] \Rightarrow u[Y]=v[Y]$$

成立，则称 X 函数决定 Y ，或称 Y 函数依赖于 X ，记作 $X \rightarrow Y$ 。称 X 为决定因素。

例：对学生关系 $S(S\#, SN, SD, SA)$ ，有

$$S\# \rightarrow SN, S\# \rightarrow SD, S\# \rightarrow SA$$

对选课关系 $SC(S\#, C\#, G)$ ，有 $(S\#, C\#) \rightarrow G$

说明：

- ☐ 函数依赖类似于变量间的单值函数关系（一个自变量只能对应一个函数值），因此也称为单值函数依赖；
- ☐ 若 $X \rightarrow Y$ 且 $Y \rightarrow X$ ，则记作 $X \leftrightarrow Y$ ；
- ☐ 若 Y 不函数依赖于 X ，则记作 $X \nrightarrow Y$

函数依赖是指关系模式 R 的任一关系都要满足的约束条件

2、函数依赖与属性间的联系之关系

- (1) 若X、Y之间是“1:1联系”，则存在函数依赖 $X \rightarrow Y$ 和 $Y \rightarrow X$, 即 $X \leftrightarrow Y$.
- (2) 若X、Y之间是“m:1联系”，则存在函数依赖关系 $X \rightarrow Y$ 。
- (3) 若X、Y之间是“m:n联系”，则X、Y之间不存在函数

3、函数依赖分类

- (1) 非平凡的函数依赖: $X \rightarrow Y$, 但 $Y \not\subset X$ 。
- (2) 平凡的函数依赖: $X \rightarrow Y$, 但 $Y \subset X$ 。
- (3) 完全函数依赖: $X \rightarrow Y$, 且对任意的 $X' \subset X$, 都有

$$X' \twoheadrightarrow Y。记作 X \xrightarrow{f} Y$$

如在SC (S#, C#, G) 中, $(S\#, C\#) \rightarrow G$,
但 $S\# \twoheadrightarrow G$, $C\# \twoheadrightarrow G$, 因此 $(S\#, C\#) \xrightarrow{f} G$ 。

第五章 关系数据理论

(4) 部分函数依赖: $X \rightarrow Y$, 但 Y 不完全函数依赖于 X 。

记作 $X \xrightarrow{p} Y$

如在 $S(S\#, SN, SD, SA)$ 中, 因为 $S\# \rightarrow SD$,

所以 $(S\#, SN) \xrightarrow{p} SD$

(5) 传递函数依赖: 若 $X \rightarrow Y$, $Y \twoheadrightarrow X$, $Y \rightarrow Z$,

且 $Z \cap (X \cup Y) = \Phi$, 则称 Z 对 X 是传递函数依赖。

例如, 在学生关系模式 $S(S\#, SN, SD, SA)$ 中, 增加属性 SL (系的位置), 则 $S\# \rightarrow SD$, $SD \rightarrow SL$, $SD \twoheadrightarrow S\#$, 所以 $S\# \xrightarrow{\text{传递}} SL$ 。

四、多值依赖（教材P178）

1、例子：设学校中一门课由多位教员讲授，他们使用相同的参考书，比如：

“物理”，教员为汪洋、大海，参考书为《普通物理学》

、

《光学原理》、
《物理习题集》

；

“数学”，教员为大海、白云，参考书为《数学分析》、
《微分方程》、
《高等代数》；

“计算”，教员为蓝天、白云，参考书为《数学分析》、

.....

第五章 关系数据理论

用模式为 **TEACH (C, T, B)** 的关系表示上述数据:

课程C	教员T	参考书B
物理	汪洋	普通物理学
物理	汪洋	光学原理
物理	汪洋	物理习题集
物理	大海	普通物理学
物理	大海	光学原理
物理	大海	物理习题集
数学	大海	数学分析
数学	大海	微分方程
数学	大海	高等代数
数学	白云	数学分析
数学	白云	微分方程
数学	白云	高等代数
计算	白云	数学分析
•••••	•••••	•••••

该关系模式中，任何两个属性都不能函数决定第三个属性。

该关系模式存在冗余大、增删不方便等问题。

没有函数依赖，
需要另行分析

第五章 关系数据理论

课程C 教员T

参考书B

物理	汪洋	普通物理学
物理	汪洋	光学原理
物理	汪洋	物理习题集
物理	大海	普通物理学
物理	大海	光学原理
物理	大海	物理习题集
数学	大海	数学分析
数学	大海	微分方程
数学	大海	高等代数
数学	白云	数学分析
数学	白云	微分方程
数学	白云	高等代数
计算	白云	数学分析

在该关系模式中，对于一个（物理，普通物理学），有一组教员{汪洋，大海}，而对于另一个（物理，光学原理），对应的教员仍是{汪洋，大海}。因此，所对应的教员只与课程的值有关而与参考书的值无关。

这是产生问题的原因吗？

第五章 关系数据理论

2、多值依赖 (MultiValued Dependency, 缩写为MVD)

设 $R(U)$ 是属性集 U 上的关系模式, X 、 Y 、 Z 是 U 的子集, 且 $Z=U-X-Y$, 多值依赖 $X \twoheadrightarrow Y$ 成立当且仅当对 $R(U)$ 的任一关系 r , 任给的一对 (x, z) 值有一组 Y 的值, 这组值仅仅取决于 x 值而与 z 值无关。

称 X 多值决定 Y 或 Y 多值依赖于 X 。

例如, 在关系模式 $TEACH$ 中有 $C \twoheadrightarrow T$

直观上看, 若 $X \twoheadrightarrow Y$, 则 X 的一个值唯一决定一组 Y 值, 且这组值与 X 、 Y 之外的属性值无关

课程 C 教员 T 参考书 B

物理	汪洋	普通物理学
物理	汪洋	光学原理
物理	汪洋	物理习题集
物理	大海	普通物理学
物理	大海	光学原理
物理	大海	物理习题集
数学	大海	数学分析
数学	大海	微分方程
数学	大海	高等代数
数学	白云	数学分析
数学	白云	数学分
计算	白云	数学

第五章 关系数据理论

多值依赖的另一等价定义：

多值依赖 $X \twoheadrightarrow Y$ 成立当且仅当对 $R(U)$ 的任一关系 r ，若存在元组 s 、 t 使得 $s[X]=t[X]$ ，则必存在元组 w 、 $v \in r$ （ w 、 v 可以与 s 、 t 相同），使得 $w[X]=v[X]=t[X]$ ，而 $w[Y]=t[Y]$ ， $w[Z]=s[Z]$ ， $v[Y]=s[Y]$ ， $v[Z]=t[Z]$ 。

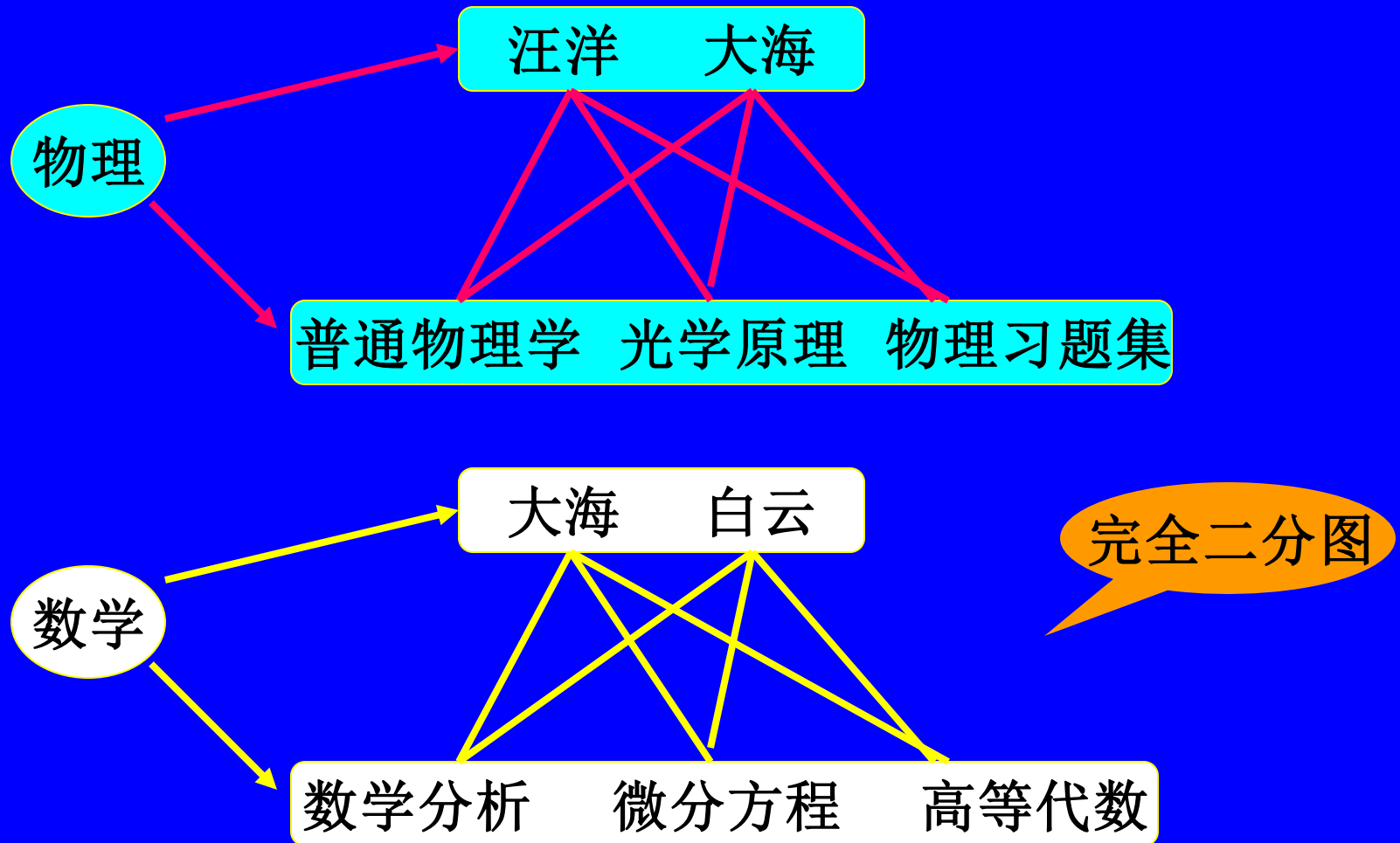
	X	Y	Z
s		s[Y]	s[Z]
t		t[Y]	t[Z]
w		w[Y]	w[Z]
v		v[Y]	v[Z]

交换 s 、 t 的 Y 值
所得新元组仍在 r 中

左图直观显示，
 x 决定一组 y 值，
这组值与 z 无关

第五章 关系数据理论

由前面例子，可看出X、Y、Z之间有下列关系：



3、多值依赖的性质：

- (1) 对称性：若 $X \twoheadrightarrow Y$, $Z = U - X - Y$, 则 $X \twoheadrightarrow Z$ 。
- (2) 函数依赖可看成是多值依赖的特例：若 $X \rightarrow Y$, 则 $X \twoheadrightarrow Y$
- (3) 若 $U = XY$ (表示 $X \cup Y$) , 则 $X \twoheadrightarrow Y$ 显然成立。

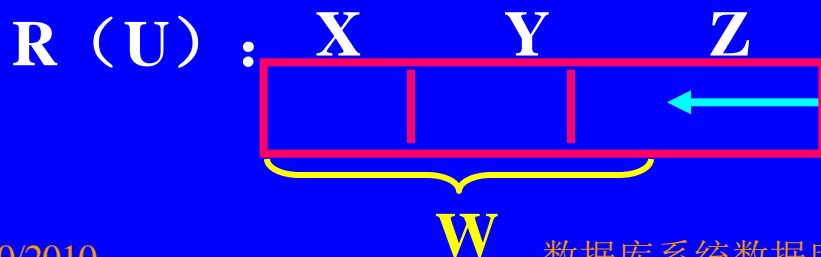
(这种多值依赖无任何实际意义, 故称为 **平凡的多值依赖**)

4、多值依赖与函数依赖的区别

- (1) 函数依赖 $X \rightarrow Y$ 的有效性仅取决于 X 、 Y , 与 X 、 Y 之外的属性无关:

$X \rightarrow Y$ 在 $\pi_{XY}(R)$ 上成立 \iff $X \rightarrow Y$ 在 $\pi_W(R)$ 上成立

其中 W 满足 $XY \subseteq W \subseteq U$ (U 是关系模式 R 的属性集)。



**X 是否函数决定 Y
与这一部分属性无关**

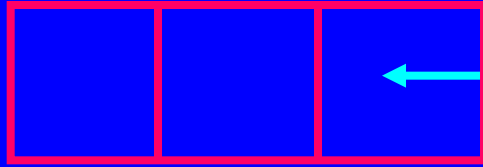
第五章 关系数据理论

多值依赖 $X \twoheadrightarrow Y$ 的有效性与 X 、 Y 之外的属性范围有关：

若 $X \twoheadrightarrow Y$ 在 U 上成立，则在 W ($XY \subseteq W \subseteq U$) 上也成立，
但反之不然。

可缩小范围但不一定能扩大范围

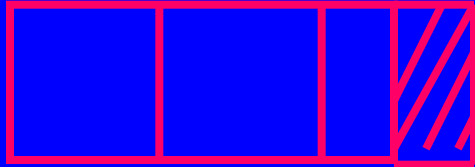
$R(U) : \quad X \quad Y \quad Z$



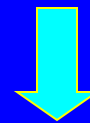
X 是否多值决定 Y
与这一部分属性密切相关

$X \twoheadrightarrow Y$ 在 U 上成立

$R(W) : \quad X \quad Y$



W



$X \twoheadrightarrow Y$ 在 W 上也成立

~~反之
不成立~~

第五章 关系数据理论

$X \twoheadrightarrow Y$ 在W上成立, 不一定有 $X \twoheadrightarrow Y$ 在U上也成立

例如, 在前面的例子中, 若增加属性“教室”(R), 则在{C, T, B, R}上 $C \twoheadrightarrow T$ 不再成立。因为若汪洋在一号教室上物理课, 大海在二号教室上物理课, 关系中会有下述两个元组:

C	T	R
物理	汪洋	普通物理学 一号教室
物理	大海	普通物理学 二号教室
.....

W

但交换汪洋和大海后的两个元组不存在

(2) 对函数依赖, 若 $X \rightarrow Y$, 则对Y的任意子集 Y' , 都有 $X \rightarrow Y'$
对多值依赖, 没有上述性质。

即 $X \twoheadrightarrow Y$ 成立 不能保证 $X \twoheadrightarrow Y'$ 成立

例：

C	T	R
物理	汪洋	普通物理学 一号教室
物理	大海	普通物理学 二号教室

在U上 $C \twoheadrightarrow \{T, R\}$ ，
但 $C \twoheadrightarrow T$ 不成立，因为交换汪洋和大海后的两个元组不存在

五、关键字

从函数依赖的角度给关键字一个形式化的定义。

1、候选关键字和主关键字：

设K是 $R\langle U, F \rangle$ 中的属性或属性组合，

若 $K \xrightarrow{f} U$ ，则称K为U的候选关键字（候选码）；

若候选关键字多于一个，则选定其中的一个作为主关键字（主键、主码）。

比以前的直观定义
准确、严谨

例：借书证关系 $C(CARD\#, S\#, SN, SD)$

$CARD\#$ 、 $S\#$ 都是候选关键字

通常选择 $CARD\#$ 作为主键

♣ K 能唯一确定一个元组

♣ K 中无多余属性

2、外键（外部码）

若 $R\langle U, F \rangle$ 中的属性或属性组合 X 不是 R 的关键字，但 X 是另一个关系的关键字，则称 X 是 R 的外键。

主键与外键提供了一个表示关系间联系的手段

3、全键（All-key，全码）

若 $R\langle U, F \rangle$ 的整个属性组是关键字，即 $U \xrightarrow{f} U$ ，则称 U 是全键。

例：关系模式 $R(P, W, A)$

演奏者 作品 听众

(P, W, A) 是 R 的全键。

注意：

一般来讲，全键是没有什么实际意义的。主键包含的属性应尽可能少为好。

4、主属性和非主属性

主属性： 包含在某个候选关键字中的属性

非主属性： 不包含在任何侯选关键字中的属性

例： $SC(S\#, C\#, G)$ 中，

$(S\#, C\#)$ 是关键字，故 $S\#, C\#$ 是主属性

G 不包含在任何关键字中，故 G 是非主属性。

§ 3 关系的范式

本节讨论下述问题：

- 如何根据关系模式属性间的数据依赖情况来判断它是否具有某些不合适的性质？
- 如何将具有不合适性质的关系模式转换为更合适的形式？

2、规范化

1、范式（Normal Form）

按关系模式所具有的数据依赖性质对关系模式的分类。也就是关系的规范化程度。

满足不同程度要求的为不同范式。

2、规范化

把一个低一级范式的关系模式通过模式分解转化为若干个高一级的关系模式的过程。

二、第一范式 (1NF)

1、定义：关系的每个分量必须是不可再分的数据项。

记作 $R \in 1NF$ 。（每个属性必须是原子的）

2、说明：

↓ 属性不可再分（不允许出现嵌套的属性定义）

↓ 属性下的值不可再分（不允许出现多个值）

↓ 这是对关系的最起码的要求，但远远不够。

（满足1NF的关系称为规范关系）

这两个表
都可变为
规范关系

例：职工情况表

职工号	部门	工资	
			基本工资	奖金

例：借书表

借书人	所借书名	借书日期
张三	B1	D1
	B2	D2
	B3	D3
李四	B2 B5	D3

第五章 关系数据理论

3、仅属于1NF的关系模式可能会产生的问题：

例：车间考核职工完成生产定额的关系模式：

W (工号, 日期, 姓名, 工种, 定额, 超额, 车间, 车间主任)

主属性

非主属性

显然 $W \in 1NF$ ，但第一节中我们已讨论知它有四个严重问题
因此仅是第一范式的关系模式完全不能满足需要。

修改困难

分析出现问题的原因：

工号函数决定非主属性姓名、工种、车间、车间主任。因此，关键字（工号，日期）部分函数决定这些属性。
这显然是产生冗余的一个主要原因。

三、第二范式（2NF）

1、定义：若 $R \in 1NF$ ，且每一非主属性都完全函数依赖于R的码，则称R是第二范式，记作 $R \in 2NF$ 。

2、属于1NF但不属于2NF的例子：

- 关系模式W（工号，日期，姓名，工种，定额，超额，车间，车间主任）
- 关系模式S_L_C（S#，SD，SL，C#，G）

↑ 学生宿舍楼，且每个系学生只住一栋楼

关键字：（S#，C#）

函数依赖：

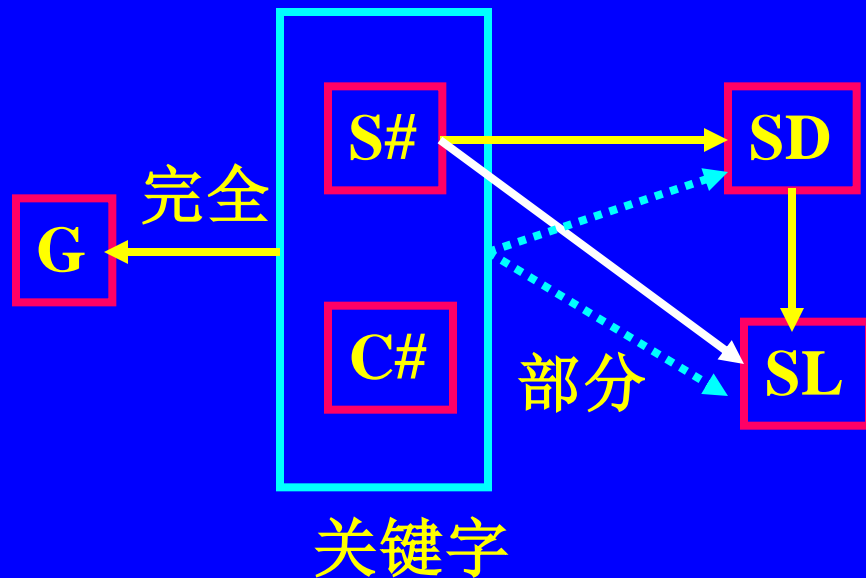
$$(S\#, C\#) \xrightarrow{f} G$$

$$S\# \rightarrow SD, SD \rightarrow SL,$$

$$S\# \xrightarrow{\text{传递}} SL$$

$$(S\#, C\#) \xrightarrow{p} SD$$

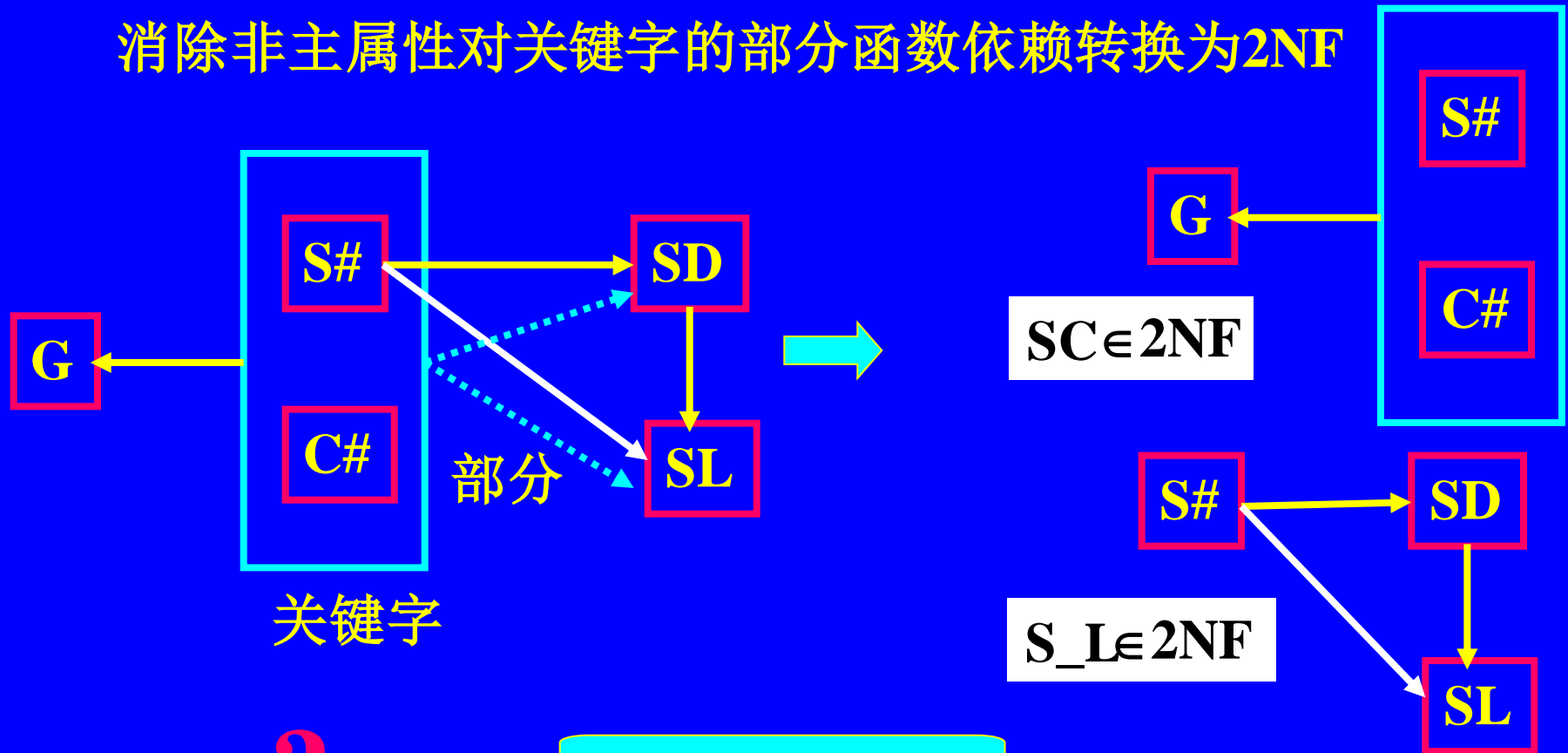
$$(S\#, C\#) \xrightarrow{p} SL$$



对关系模式S_L_C,同样存在数据冗余大、插入异常、删除异常、修改困难等问题。（产生问题的原因同W）

3、解决办法：用投影分解

消除非主属性对关键字的部分函数依赖转换为2NF



?

分解之后，与 S_L_C 相比，
 SC 和 S_L 性质如何？

第五章 关系数据理论

分解前 S_L_C (S#, SD, SL, C#, G)

分解后 SC (S#, C#, G)
S_L (S#, SD, SL)

分解之后，与S_L_C相比：

◆ 数据冗余减小

（原来伴随学生所学的每门课都要存储一遍SD、SL的值）

；

◆ 没选课的学生信息可以存储；

◆ 删除选课记录不会误删学生其他信息；

◆ 冗余数据的减少使得修改变得容易些。

1NF的上述四个问题得到了部分解决

S_L还有问题吗？

4、仅属于2NF的关系模式可能会产生的问题

(1) 数据冗余

S_L (S#, SD, SL)

数据冗余仍然较大：SL的值重复严重

(2) 插入异常

若一个系刚成立但尚无学生，该系名称等无法存储

(3) 删除异常

若一个系的学生全部毕业，删除全部学生数据的同时把该系的数据（如系名等）也都删除了

(4) 修改困难

数据冗余大势必造成修改困难
(这可以说是个必然联系)

可能的原因：

存在传递函数依赖

$S\# \rightarrow SL$

四、第三范式 (3NF)

1、定义：若关系模式 $R\langle U, F \rangle \in 1NF$ ，并且 R 中不存在码 X 、属性组 Y 和非主属性 Z ($Z \subseteq Y$) 使得 $X \rightarrow Y$ ($Y \not\rightarrow X$)， $Y \rightarrow Z$ 成立，则称 $R \in 3NF$ 。

2、定理：若 $R \in 3NF$ ，则 $R \in 2NF$ 。

分析 要证 $R \in 2NF$

↔ R 的每一非主属性都完全函数依赖于码

↔ 不存在任何非主属性部分函数依赖于码

由3NF的定义，

若 $R \in 3NF$ ，则 R 中不存在码 X 、属性组 Y 和非主属性 Z ($Z \subsetneq Y$) 使得 $X \rightarrow Y$ ($Y \not\rightarrow X$)， $Y \rightarrow Z$ 成立

希望推出

只需证：若存在非主属性部分函数依赖于码

→ $R \notin 3NF$

四、第三范式 (3NF)

1、定义：若关系模式 $R\langle U, F\rangle \in 1NF$ ，并且 R 中不存在码 X 、属性组 Y 和非主属性 $Z/(Z \subseteq Y)$ 使得 $X \rightarrow Y \wedge (Y \rightarrow X)$ ， $Y \rightarrow Z$ 成立，则称 $R \in 3NF$ 。

2、定理：若 $R \in 3NF$ ，则 $R \in 2NF$ 。

只需证：若存在非主属性部分函数依赖于码 $\longrightarrow R \notin 3NF$

证明 设 A 是 R 的一个非主属性， K 是 R 的码，且 $K \xrightarrow{p} A$

则存在 $K' \subset K$ ，使得 $K' \rightarrow A$ 。故有 $K \rightarrow K'$ ， $K' \rightarrow A$ 。

因为一个候选关键字不可能函数依赖于它的真子集，
所以有 $K' \nrightarrow K$ 。

又因为 A 是非主属性， K 是码，且 $K' \subset K$ ，故 $A \notin K'$ 。

所以 $R \notin 3NF$ 。

证毕！

要找 Y ，满足
 $K \rightarrow Y$ ， $Y \nrightarrow K$ ， $Y \rightarrow A$

3、说明

⌘ 根据以上证明，也可这样定义3NF：

若 $R \in 2NF$ ，并且R中不存在任何非主属性传递函数依赖于R的码，则称 $R \in 3NF$ 。

⌘ 若 $R \langle U, F \rangle$ 中U是全键，则一定有 $R \in 3NF$ 。

4、属于2NF但不属于3NF的例子：

◆ 关系模式S_L (S#, SD, SL)

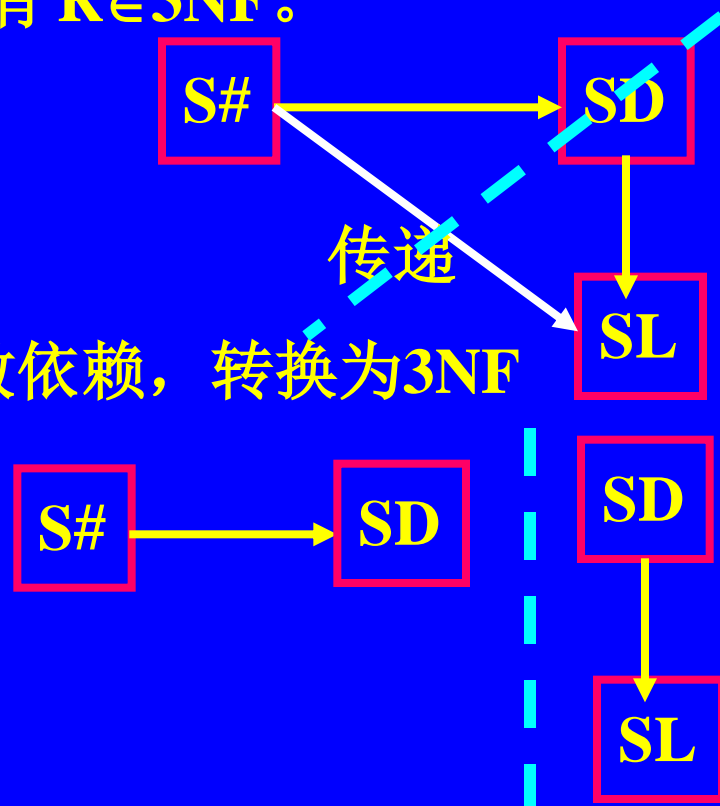
5、解决办法：用投影分解

消除非主属性对关键字的传递函数依赖，转换为3NF

如将S_L分解为：

S_D (S#, SD)

D_L (SD, SL)



6、仅属于3NF的关系模式可能会产生的问题

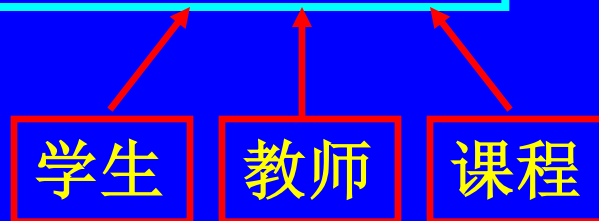
由上可知，部分函数依赖和传递函数依赖是产生异常的两个重要原因。

3NF中不存在非主属性对于关键字的部分函数依赖和传递函数依赖，因此具有较好的性质。

通常设计关系模式时至少应该是属于3NF的。

虽说3NF是广泛使用的一种关系范式，但3NF仍然存在某些“异常”。

例：关系模式 $R(S, T, J)$



每个教师只教一门课，每门课有若干教师，学生选定某门课就对应固定的教师。

第五章 关系数据理论

例：关系模式 $R(S, T, J)$

函数依赖： $(S, J) \rightarrow T$;

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

候选关键字： (S, J) ; (S, T)

R 中无非主属性，显然 $R \in 3NF$ 。

问题

J 重复，产生冗余，
带来可能的不一致性等问题

可能的原因：

R 中存在部分函数依赖 $(S, T) \rightarrow J$ 。

S	T	J
学生1	教师1	课程1
.....
学生K	教师1	课程1
学生L	教师2	课程1
.....
学生M	教师2	课程1
学生1	教师3	课程2
.....
学生K	教师3	课程2
.....

因为这是关键字

也是传递函数依赖

由例子可以看出，虽然3NF消除了一大类数据冗余问题（冗余也是产生异常操作的直接原因），但这种数据冗余是非主属性下的冗余。3NF并没有涉及主属性下的数据冗余问题。

唉，还是有问题！



五、Boyce-Codd范式 (BCNF)

1、定义：若 $R \in 1NF$ ，且对任何非平凡的函数依赖 $X \rightarrow Y$ ， X 必包含码。则 $R \in BCNF$ 。

2、另一种等价的定义：

若 $R \in 1NF$ ，并且 R 中不存

即每一决定因素都包含码

在任何属性传递函数依赖于 R 的某个码，则称 $R \in BCNF$ 。

证明 (1) 定义 \Rightarrow 等价定义

即要证：若对任何非平凡的函数依赖 $X \rightarrow Y$ ， X 必包含码，

则 R 中不存在任何属性传递函数依赖于 R 的某个码。

反证法：假设 R 中存在属性 Z 传递函数依赖于码 X ，

则必存在属性组 Y ，使得

$$X \rightarrow Y, Y \not\rightarrow X, Y \rightarrow Z.$$

显然 Y 不是码，也不包含码，矛盾。

(2) 等价定义 \Rightarrow 定义

仍用反证法：假设存在非平凡函数依赖 $X \rightarrow Y$ ， X 不包含码，设 X' 是一关键字，则 $X' \rightarrow X$ ， $X \not\rightarrow X'$ ，因此 $X' \xrightarrow{\text{传递}} Y$ ，矛盾。证毕

3、说明

- 若 $R \in \text{BCNF}$ ，则 $R \in 3\text{NF}$ 。
(也称BCNF为修正的3NF)
- BCNF 又消除了主属性对码的部分函数依赖和传递函数依赖。
- 从完全函数依赖的观点看，属于BCNF的关系模式满足：
 - ⬆ 所有非主属性对每一个码都是完全函数依赖；
 - ⬆ 所有主属性对每一个不包含它的码也是完全函数依赖；
 - ⬆ 没有任何属性完全函数依赖不是码的任何一组属性。

若 $R \in 1\text{NF}$ ，并且 R 中不存在任何非主属性传递函数依赖于 R 的某个码，则称 $R \in 3\text{NF}$ 。

第五章 关系数据理论

- 在函数依赖的范畴内,BCNF已做到彻底的分离,消除了插入异常、删除异常(3NF的“不彻底性”在于可能存在主属性对关键字的部分函数依赖和传递函数依赖)。

4、属于3NF但不属于BCNF的例子:

◆ 关系模式 $R(S, T, J)$

码: (S, T) ; (S, J)

函数依赖: $(S, J) \rightarrow T$; $(S, T) \rightarrow J$; $T \rightarrow J$

因为J部分函数依赖于码 (S, T) , 或T是决定因素, 但T不包含码。故R不属于BCNF。

5、解决办法: 用投影分解

消除主属性对码的部分或传递函数依赖, 转换为BCNF。

将 $R(S, T, J)$ 分解为:

$R_1(S, T)$ $R_2(T, J)$

第五章 关系数据理论

6、仅属于BCNF的关系模式可能会产生的问题

以前面讨论过的关系模式 **TEACH (C, T, B)** 为例

课程C	教员T	参考书B
物理	汪洋	普通物理学
物理	汪洋	光学原理
物理	汪洋	物理习题集
物理	大海	普通物理学
物理	大海	光学原理
物理	大海	物理习题集
数学	大海	数学分析
数学	大海	微分方程
数学	大海	高等代数
数学	白云	数学分析
数学	白云	微分方程
数学	白云	高等代数
.....

课程 教员 参考书

TEACH的关键字是
(C, T, B)，即全键。
因此 **TEACH** \in BCNF

问题：冗余大，
增、删不便

原因：存在多值依赖

六、第四范式 (4NF)

1、定义：设 $R \in 1NF$ ，若对于 R 的每个非平凡的多值依赖 $X \twoheadrightarrow Y$ ($Y \not\subseteq X$)， X 都包含码，那么称 $R \in 4NF$

2、定理：若 $R \in 4NF$ ，则 $R \in BCNF$ 。

3、属于 $BCNF$ 但不属于 $4NF$ 的例子：

全码

◆ 关系模式 $TEACH(C, T, B)$

(存在非平凡多值依赖 $C \twoheadrightarrow T$ ，而 C 不是关键字)

4、解决办法：用投影分解

消除不合适的多值依赖，转换为 $4NF$

对于 $TEACH(C, T, B)$ 分解为：

$C-T(C, T)$

$C-B(C, B)$

七、范式小结

1、规范化的目的

解决数据冗余、插入异常、删除异常、修改困难等问题

2、规范化的基本思想

逐步消除不合适的数据依赖，让一个关系描述一个概念、一个实体或实体间的一种联系。即“一事一地”的模式设计原则。

3、范式

消除决定因素非码的非平凡函数依赖

1NF

2NF

3NF

BCNF

4NF

...

消除非主属性对码的部分函数依赖

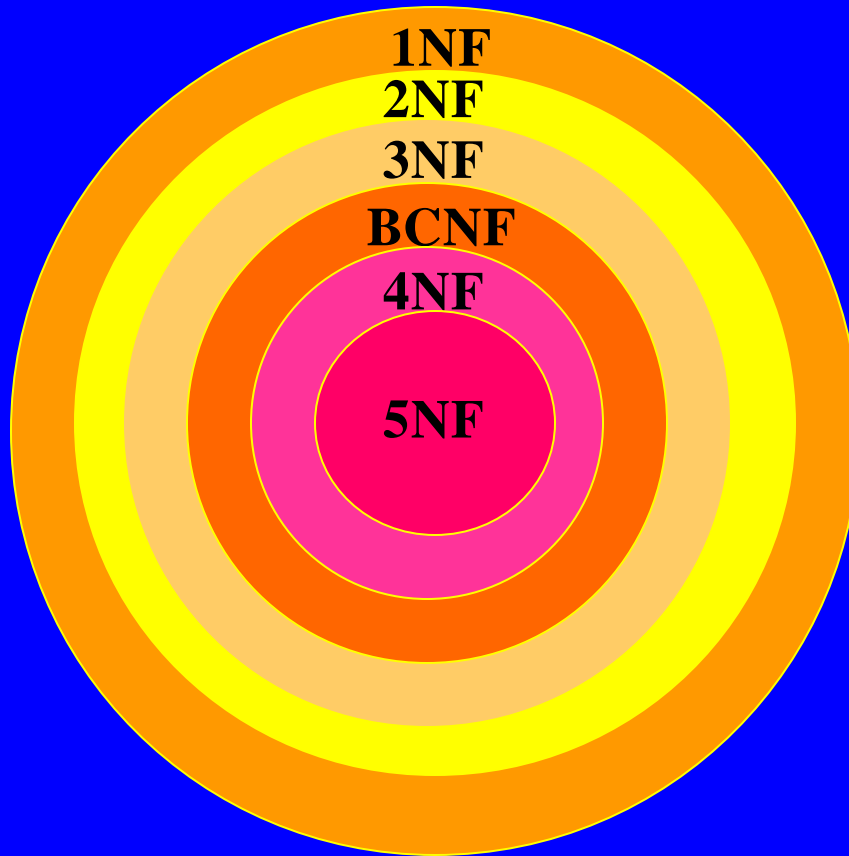
消除非主属性对码的传递函数依赖

消除主属性对码的部分和传递函数依赖

消除非平凡且非函数依赖的多值依赖



范式间的关系：



4、规范化的过程

对关系模式分解，把一个低一级关系模式分解成若干个高级的关系模式。

5、规范化与操作效率

片面追求高级的模式，会使数据库操作效率降低

本节开头

本章开头

下一节

§ 4 函数依赖的Armstrong公理系统

上一节给出了范式的定义，并用例子说明了如何确定一个关系模式属于第几范式以及如何把低一级的模式分解成高一级的模式，但都是直观的方法。

本节及下节将从理论上探讨：

如何确定一个关系模式的极小函数依赖集？

如何分解关系模式？

一、Armstrong公理系统

模式分解算法的理论基础

问题：对于给定的一组函数依赖，如何判断其它函数依赖是否成立？或者说哪些函数依赖是不独立的？如：对关系模式R有： $A \rightarrow B$ ， $B \rightarrow C$ 。

问： $A \rightarrow C$ 是否成立？

1、逻辑蕴涵

设 F 是关系模式 $R\langle U, F \rangle$ 的函数依赖集， X 、 Y 是 R 的属性子集。如果在 $R\langle U, F \rangle$ 上函数依赖 $X \rightarrow Y$ 成立，则称 F 逻辑蕴涵 $X \rightarrow Y$ 。

即对 R 的任一关系 r ，对 r 中的任意两元组 t 、 s ，
若 $t[X]=s[X]$ ，则 $t[Y]=s[Y]$

下面将会看到（我们的本意也是如此），可以用下述方式来表述逻辑蕴涵：

若从 F 中的函数依赖能够推出 $X \rightarrow Y$ ，则 F 逻辑蕴涵 $X \rightarrow Y$ 。

2、Armstrong公理

对关系模式 $R\langle U, F \rangle$ ，有如下的推理规则：

A1、自反律（Reflexivity）：若 $Y \subseteq X \subseteq U$ ，则 $X \rightarrow Y$ 为 F 所蕴涵。

简写：若 $Y \subseteq X$ ，则 $X \rightarrow Y$

第五章 关系数据理论

A2、增广律 (Augmentation) : 若 $X \rightarrow Y$ 为F所蕴涵, 且 $Z \subseteq U$, 则 $XZ \rightarrow YZ$ 为F所蕴涵。

简写: 若 $X \rightarrow Y$, 则 $XZ \rightarrow YZ$

A3、传递律 (Transitivity) : 若 $X \rightarrow Y$ 和 $Y \rightarrow Z$ 为F所蕴涵, 则 $X \rightarrow Z$ 为F所蕴涵。

简写: 若 $X \rightarrow Y$ 、 $Y \rightarrow Z$, 则 $X \rightarrow Z$

3、Armstrong公理的正确性

决定每一部分必决定全部

定理: **Armstrong**推理规则是正确的。

4、Armstrong公理的推论

(1) 合并规则: 由 $X \rightarrow Y$, $X \rightarrow Z$, 有 $X \rightarrow YZ$ 。

(2) 分解规则: 由 $X \rightarrow Y$ 及 $Z \subseteq Y$, 有 $X \rightarrow Z$ 。

(3) 伪传递规则: 由 $X \rightarrow Y$, $WY \rightarrow Z$, 有 $XW \rightarrow Z$ 。

决定全部必决定部分

证明

(1) 合并规则：由 $X \rightarrow Y$ ， $X \rightarrow Z$ ，有 $X \rightarrow YZ$ 。

因为 $X \rightarrow Z$ ，由增广律有 $XY \rightarrow YZ$ ，
因为 $X \rightarrow Y$ ，由增广律有 $XX \rightarrow XY$ ，
由传递律有 $XX \rightarrow YZ$

又因为 $X \rightarrow XX$ （自反律），所以 $X \rightarrow YZ$ （传递律）。

(2) 分解规则：由 $X \rightarrow Y$ 及 $Z \subseteq Y$ ，有 $X \rightarrow Z$ 。

因为 $Z \subseteq Y$ ，所以 $Y \rightarrow Z$ （自反律），

由 $X \rightarrow Y$ 、 $Y \rightarrow Z$ ，又有 $X \rightarrow Z$ （传递律）。

Armstrong公理

若 $Y \subseteq X$ ，则 $X \rightarrow Y$

若 $X \rightarrow Y$ ，则 $XZ \rightarrow YZ$

若 $X \rightarrow Y$ 、 $Y \rightarrow Z$ ，则
 $X \rightarrow Z$

伪传递规则的证明作为习题自己证。

根据合并规则和分解规则，可以得出一个重要的结论：

定理： $X \rightarrow A_1 A_2 \dots A_k$ 的充要条件是 $X \rightarrow A_i$ （ $i=1, 2, \dots, k$ ）。

二、函数依赖集的闭包

有了上述推理规则，对一个给定的函数依赖集F，我们自然希望知道哪些函数依赖可由F推出，哪些不能由F推出，由F能推出的函数依赖有多少等。

1、闭包的定义：在关系模式 $R\langle U, F \rangle$ 中F所逻辑蕴涵的函数依赖的全体称为F的闭包。记作 F_+ 。

例：设关系模式 $R\langle U, F \rangle$ ，其中 $U=\{X, Y, Z\}$ ， $F=\{X \rightarrow Y, Y \rightarrow Z\}$ ，则F的闭包 F_+ 为：

$X \rightarrow \Phi$	$XY \rightarrow \Phi$	$XZ \rightarrow \Phi$	$XYZ \rightarrow \Phi$	$Y \rightarrow \Phi$	$YZ \rightarrow \Phi$	$Z \rightarrow \Phi$
$X \rightarrow X$	$XY \rightarrow X$	$XZ \rightarrow X$	$XYZ \rightarrow X$	$Y \rightarrow Y$	$YZ \rightarrow Y$	$Z \rightarrow Z$
$X \rightarrow Y$	$XY \rightarrow Y$	$ZX \rightarrow Y$	$XYZ \rightarrow Y$	$Y \rightarrow Z$	$YZ \rightarrow Z$	
$X \rightarrow Z$	$XY \rightarrow Z$	$XZ \rightarrow Z$	$XYZ \rightarrow Z$	$Y \rightarrow YZ$	$YZ \rightarrow YZ$	
$X \rightarrow XY$	$XY \rightarrow XY$	$XZ \rightarrow XY$	$XYZ \rightarrow XY$			
$X \rightarrow XZ$	$XY \rightarrow XZ$	$XZ \rightarrow XZ$	$XYZ \rightarrow XZ$			
$X \rightarrow YZ$	$XY \rightarrow YZ$	$XZ \rightarrow YZ$	$XYZ \rightarrow YZ$			
$X \rightarrow XYZ$	$XY \rightarrow XYZ$	$XZ \rightarrow XYZ$	$XYZ \rightarrow XYZ$			



2、求函数依赖集的闭包问题是NP完全问题

3、属性集的闭包

设关系模式 $R\langle U, F \rangle$, $X \subseteq U$, X 关于函数依赖集 F 的闭包为:

$$X_F^+ = \{ A \mid X \rightarrow A \text{ 能由 } F \text{ 根据 Armstrong 公理推出} \}$$

函数依赖集的闭包是一个（更大的）函数依赖集
属性集的闭包 是一个（更大的）属性集合

设 F 为属性集 U 上的一组函数依赖, $X, Y \subseteq U$, $X \rightarrow Y$ 能由 F 根据 Armstrong 公理导出的充分必要条件是 $Y \subseteq X_F^+$ 。

于是判定 $X \rightarrow Y$ 能否由 F 根据 Armstrong 公理导出的问题, 就转化为求出 X_F^+ , 判定 Y 是否为 X_F^+ 的子集的问题。

4、求闭包 X_F^+ 的算法

分析：根据定义，要求所有这样的属性A： $X \rightarrow A$ 能由F经数次使用Armstrong公理推出。

A应具备何种特征？

分析Armstrong公理的形式可以看出，A应该属于X，或A属于F中某个函数依赖的右部，而该函数依赖的左部属性应是属于X的闭包。

算法思想：采用逐步扩张X的策略求X的闭包。方法是：扫描F，找出左部属性属于当前X的闭包的函数依赖，将右部属性加入当前X的闭包。经若干次扩张，直到当前闭包集不再扩大为止。

Armstrong公理

若 $Y \subseteq X$ ，则 $X \rightarrow Y$

若 $X \rightarrow Y$ ，则 $XZ \rightarrow YZ$

若 $X \rightarrow Y$ 、 $Y \rightarrow Z$ ，则 $X \rightarrow Z$

算法

输入：X, U, F;

输出：X关于F的闭包 X_F^+

(1) 初始：令 $X(0)=X$, $i=0$;

(2) 迭代：求 $B=\{A \mid (\exists V)(\exists W)(V \rightarrow W \in F \wedge V \subseteq X(i) \wedge A \in W)\}$;
 $X(i+1)=B \cup X(i)$;

(3) 判断： $X(i+1)=X(i)$?

若相等或 $X(i)=U$, 转下一步;

否则, 用 $i+1$ 取代 i , 转第(2)步继续迭代;

(4) 结束： $X(i)$ 就是 X_F^+ , 算法终止.

计算时在F中寻找未用过的左部是 $X(i)$ 的子集的函数依赖

每次迭代都添加属性到当前闭包(不增加时算法就结束)故至多迭代 $|U|-|X|$ 次算法终止

第五章 关系数据理论

例：设 $F=\{AC \rightarrow PE, PG \rightarrow A, B \rightarrow CE, A \rightarrow P, GA \rightarrow B, GC \rightarrow A, PAB \rightarrow G, AE \rightarrow GB, DP \rightarrow H\}$,

$X=BG$,

求 X_F^+

求解过程

初始： $X(0) = X = BG$

第1次迭代：找左部为B、G或BG的函数依赖，将其右部属性放入闭包

$X(1) = X(0) \cup CE = BCEG$

第2次迭代：

$X(2) = X(1) \cup A = ABCEG$

第3次迭代：

$X(3) = X(2) \cup PEBG = ABCEGP$

第4次迭代：

$X(4) = X(3) \cup AG = ABCEGP$

左部	右部	迭代号
AC	PE	3
PG	A	4
B	CE	1
A	P	3
GA	B	3
GC	A	2
PAB	G	4
AE	GB	3
DP	H	

$X(4)$ 与 $X(3)$ 相同，算法结束， $X(4)=ABCEGP$ 就是BG的闭包

第五章 关系数据理论

又如，若关系模式为 $R(A1, A2, A3)$ ， $F=\{A1 \rightarrow A2, A2 \rightarrow A3\}$ ，
则当 $X=A1$ 时， X 的闭包为 $A1A2A3$ ；

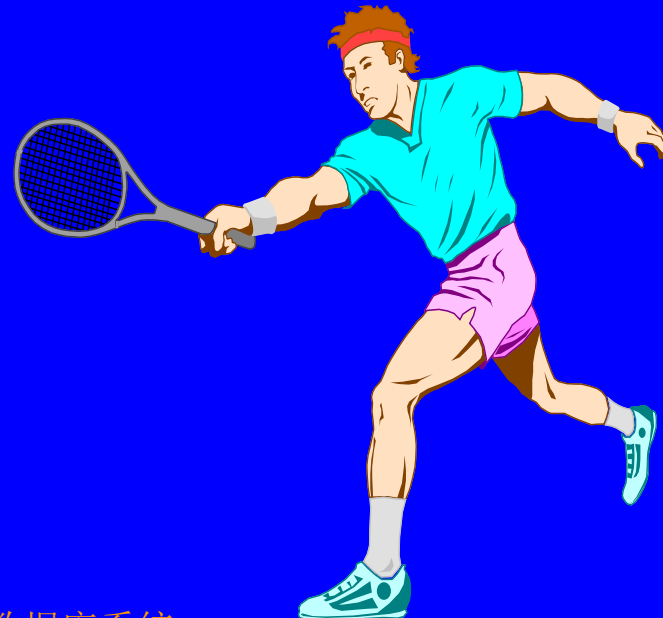
当 $X=A2$ 时， X 的闭包为 $A2A3$ ；

当 $X=A3$ 时， X 的闭包为 $A3$ ；

求 X 的闭包算法的正确性
我们不做证明。

留作习题

思考：对关系模式 $R\langle U, F \rangle$ ，
若 $X_F^+ = U$ ，
问 X 有何性质？



三、Armstrong公理系统的有效性和完备性

函数依赖集 F 的闭包 F^+ 是 F 所逻辑蕴涵的函数依赖的全体。关于逻辑蕴涵，我们已经建立了一套推理规则（Armstrong公理系统），因此自然要问：

→ 由 F 出发根据Armstrong公理能推出的函数依赖是否一定是 F 逻辑蕴涵的，即是否一定在 F^+ 中？

→ F^+ 中的每一个函数依赖，即 F 逻辑蕴涵的每一个函数依赖，是否一定可以根据Armstrong公理推出？

$F^+ = \{ \text{由} F \text{出发根据Armstrong公理能推出的函数依赖的全体} \}$



1、何谓有效性？何谓完备性？

Armstrong公理的有效性：

有效性就是正确性，Armstrong公理的正确性就是由该公理系统推出的函数依赖都是正确的。而正确性的标准就是只要F中的函数依赖为真，则由公理推出的函数依赖也为真。换句话说，由F出发根据Armstrong公理能推出的函数依赖一定在F+

中 Armstrong公理的完备性：

完备性就是要讨论用公理能否推出所有的函数依赖。因为如果存在F逻辑蕴涵的函数依赖不能用公理推出，则说明这些公理不够用，是不完全的，还必须补充新的公理。因此，完备性就是指F+中的每一个函数依赖，一定可以根据Armstrong公理推出

公理的正确性保证了推出的所有函数依赖都是正确的；
公理的完备性保证了可以推出所有的函数依赖。

2、Armstrong公理的有效性

回忆一下Armstrong公理系统（自反律、传递律和增广律）以及前面讨论过的定理“Armstrong推理规则是正确的”，可证Armstrong公理系统是有效的。

3、Armstrong公理的完备性

要证完备性，必须解决如何判断

一个函数依赖是否可由F出发根据Armstrong公理推出

如果能方便地求出 F^+ ，将有助于解决上述问题，但计算 F^+ 是NP完全问题。

前面已经给出求属性集的闭包的有效算法。对一个函数依赖 $X \rightarrow Y$ ， X 的闭包对我们要解决的问题有何帮助？

第五章 关系数据理论

根据求闭包的算法, 可知 $X \rightarrow X_F^+$

因此, 若 $Y \subseteq X_F^+$, 则 $X \rightarrow Y$ 。 反之是否成立?

引理: 对关系模式 $R\langle U, F \rangle$, $X, Y \subseteq U$,

$X \rightarrow Y$ 能由 F 根据 Armstrong 公理推出 $\iff Y \subseteq X_F^+$

证明 充分性: \longleftarrow

设 $Y = B_1 B_2 \dots B_k$, $B_i \in X_F^+$, ($i=1, 2, \dots, k$)

根据属性闭包的定义可得

$X \rightarrow B_i$ ($i=1, 2, \dots, k$) 都可由 F 根据 Armstrong 公理推出,

再由合并规则可得 $X \rightarrow B_1 B_2 \dots B_k$,

即 $X \rightarrow Y$ 可由 F 根据 Armstrong 公理推出,

必要性: \longrightarrow

此处证明从略, 留作习题

定理：Armstrong公理系统是完备的。

分析

根据定义，需证：

对 F^+ 中的每一个函数依赖，必定可由 F 用Armstrong公理推出
再由 F^+ 的定义，需证：

对 F 所蕴涵的每个函数依赖，必定可由 F 用Armstrong公理推出

因此，只需证明：

若存在函数依赖 $X \rightarrow Y$ 不能由Armstrong公理推出，则它不为 F 所蕴涵

欲证 $X \rightarrow Y$ 不为 F 所蕴涵，只需证：

存在符合关系模式 $R \langle U, F \rangle$ 的一个关系 r ，函数依赖 $X \rightarrow Y$ 在 r 上不成立

因此，需要构造一个关系 r ：

r 的属性集是 U ，且 r 满足 F 中的所有函数依赖，但 $X \rightarrow Y$ 在 r 上不成立

关键是证明这两点

第五章 关系数据理论

欲构造关系 r ，使得 $X \rightarrow Y$ 在 r 上不成立，最简单的情形是：

r	X	Y	Z
表示相等	1	1	
	1	0	

表示不相等

但这样构造的 r 显然不合要求，因为 r 不一定满足 F 中的所有函数依赖。

属性集的闭包在Armstrong公理可推导的意义下具有封闭性，我们这里要讨论的就是Armstrong公理的完备性。所以构造 r 时利用 X 的闭包更为合理：

关系 r	X_F^+	$U - X_F^+$
	11...1	11...1
	11...1	00...0

关系 r 只有两个元组，在 X 的闭包上分量值相等

第五章 关系数据理论

证明 假设函数依赖 $X \rightarrow Y$ 不能由 Armstrong 公理推出

根据前面引理易证

(1) 先证 F 中的函数依赖和 X 的闭包有下述关系:

若 $V \rightarrow W$ 是 F 中的函数依赖, 且 $V \subseteq X_F^+$, 则 $W \subseteq X_F^+$

(2) 再证前面构造的关系 r 是 $R\langle U, F \rangle$ 的关系, 即满足 F 中的全部函数依赖

设 $V \rightarrow W$ 是 F 中的任一函数依赖,

要证 $V \rightarrow W$ 在 r 上成立。

分两种情形:

第一种情形: $V \subseteq X_F^+$

根据第 (1) 步结论, 有 $W \subseteq X_F^+$

关系 r

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

$\underbrace{\hspace{1.5cm}}_W$

W 的值在仅有的两个元组上相等, 因此必有若 $t[V]=s[V]$ 则 $t[W]=s[W]$, 所以此时 $V \rightarrow W$ 在 r 上成立。

第二种情形: $V \not\subseteq X_F^+$

此时存在属性 $A \in V$, 但 $A \notin X_F^+$

r 中仅有的两个元组在属性 A 上的值不等,
所以在 r 中不存在两个元组 t 、 s 满足 $t[V]=s[V]$,
因此“若 $t[V]=s[V]$, 则 $t[W]=s[W]$ ”在关系 r 上恒成立, 故 $V \rightarrow W$ 在 r 上成立。

关系 r

	X_F^+	$U - X_F^+$	
1		A	1
1			0

V

综上可知 F 的任一函数依赖在 r 上成立, 即 r 是 $R\langle U, F \rangle$ 的关系

(3) 最后证明函数依赖 $X \rightarrow Y$ 在 r 上不成立

由假设, $X \rightarrow Y$ 不能由Armstrong公理推出, 根据引理有 $Y \not\subseteq X_F^+$
所以有 Y 的子集 Y' 满足 $Y' \subseteq U - X_F^+$, 故对 r 中仅有的两元组 t 、 s , 有
 $t[Y'] \neq s[Y']$, 进而有 $t[Y] \neq s[Y]$ 。但由于 $X \subseteq X_F^+$, 所以有 $t[X]=s[X]$,
因此得 $X \rightarrow Y$ 在 r 上不成立。即 $X \rightarrow Y$ 必不为 F 所蕴涵。

证毕

四、函数依赖集的等价和极小依赖集

1、函数依赖集的等价（覆盖）：

定义：设 F 和 G 都是关系模式 R 上的两个函数依赖集，如果 $F^+ = G^+$ ，则称 F 与 G 等价（或称 F 覆盖 G ，或 G 覆盖 F ）。

等价具有自反性、对称性、传递性，即是一等价关系。

定理： $F^+ = G^+$ 的充要条件是 $F \subseteq G^+$ 和 $G \subseteq F^+$ 。

证明：必要性显然。因为 $F \subseteq F^+$ 、 $G \subseteq G^+$ 成立，
若 $F^+ = G^+$ ，则必有 $F \subseteq G^+$ 、 $G \subseteq F^+$ 成立。

即 F 可由 G 推出
 G 也可由 F 推出

充分性：要证当 $F \subseteq G^+$ 、 $G \subseteq F^+$ 时，有 $F^+ = G^+$ 成立。

先证对任意的 $X \rightarrow Y \in F^+$ ，有 $X \rightarrow Y \in G^+$ 。这样就有 $F^+ \subseteq G^+$ 。

因为 $X \rightarrow Y \in F^+$ ，所以 $Y \subseteq X_F^+$ 。

第五章 关系数据理论

若 $F \subseteq G^+$, 则 $X_F^+ \subseteq X_{G^+}^+$

所以 $Y \subseteq X_{G^+}^+$, 从而得 $X \rightarrow Y \in (G^+)^+ = G^+$ 。

于是证得 $F^+ \subseteq G^+$ 。

同理可证 $G^+ \subseteq F^+$, 所以 $F^+ = G^+$ 。

证毕

对本定理也可先证命题: (1) 若 $F \subseteq G$, 则 $F^+ \subseteq G^+$;

(2) $F^+ = (F^+)^+$ 。

然后有: $F \subseteq G^+ \Rightarrow F^+ \subseteq (G^+)^+ = G^+;$

$G \subseteq F^+ \Rightarrow G^+ \subseteq (F^+)^+ = F^+;$



$F^+ = G^+$

2、极小（最小）函数依赖集

根据分解与合并规则，可以证明下述引理：

引理：每个函数依赖集 F 都与一个右部只有单个属性的函数依赖所构成的函数依赖集 G 等价。

第五章 关系数据理论

证明：设F中的函数依赖由两部分构成：

$$F = F_1 \cup F_2$$

右边是一个以上属性的函数依赖子集

右边是单属性的函数依赖子集

设F₂中的函数依赖形如 $X \rightarrow A_1 A_2 \dots A_k$, $k \geq 2$ 。

令F₃是将F₂中所有函数依赖分解成形如 $X \rightarrow A_i$ 的函数依赖全体，

$G = F_1 \cup F_3$ ，要证明F与G等价

对G中任一个形如 $X \rightarrow A_i \in F_3$ ，由于F中存在 $X \rightarrow A_1 A_2 \dots A_k$ ， $i \leq k$ ，根据分解规则可推出 $X \rightarrow A_i$ ，所以 $X \rightarrow A_i \in F^+$ ，而 $F_1 \subseteq F \subseteq F^+$ 。故 $G \subseteq F^+$ 。

根据合并规则，同理可证 $F \subseteq G^+$ 。

根据前面的引理，有F与G等价。

证毕

第五章 关系数据理论

定义：若函数依赖集F满足下列条件，则称F为极小（函数）

依赖集，亦称最小（函数）依赖集、最小覆盖：

(1) F的每个函数依赖的右部仅含有一个属性；右部都是单属性

(2) 对F中的任一个函数依赖 $X \rightarrow A$ ，不存在多余的函数依赖
都有F与 $F - \{X \rightarrow A\}$ 不等价；

(3) 对F中的任一个函数依赖 $X \rightarrow A$ ，每个函数依赖的左部没有多余的属性
都有F与 $(F - \{X \rightarrow A\}) \cup \{Z \rightarrow A\}$ 不等价，

其中Z是X的任一真子集。

例：关系模式 $S \langle U, F \rangle$ ， $U = \{ S\#, SD, MN, CN, G \}$ ，

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

则F是最小覆盖。

设函数依赖集 $F' = \{ S\# \rightarrow SD, S\# \rightarrow MN, SD \rightarrow MN,$

$(S\#, CN) \rightarrow G, (S\#, SD) \rightarrow SD \}$ ，

则F'不是最小覆盖。

3、如何求极小函数依赖集

定理：每一个函数依赖集F均等价于一个极小函数依赖集F_m。

证明 对F极小化处理

(1) 分解：将F中右部多于一个属性的函数依赖分解成单属性函数依赖。

逐一检查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 $\{X \rightarrow A_i \mid i=1, 2, \dots, k\}$ 取代 $X \rightarrow Y$ 。

这样每个函数依赖的右部都为单属性，由引理知该函数依赖集与F等价。

(2) 去多余依赖：

对经第一步处理后的F，逐一检查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 $X \rightarrow A$ 。

此处我们考虑 $X \rightarrow A$ 是否多余，即F是否与G等价。

因为G是F的子集，且只比F少一个函数依赖 $X \rightarrow A$ ，所以

F与G等价 $\iff X \rightarrow A$ 可由G推出 $\iff A \in X_G^+$

(3) 去左部多余属性:

对经第二步处理后的F，逐一检查F中各函数依赖 $X \rightarrow A$ ，设 $X = B_1 B_2 \dots B_m$ ，逐一考察 B_i ，若 $A \in (X - B_i)_F^+$ ，则以 $X - B_i$ 取代 X 。这样得到的函数依赖集仍与F等价，且去掉了左部多余的属性。

F 与 $(F - \{X \rightarrow A\}) \cup \{(X - B_i) \rightarrow A\}$ 等价

$\iff (X - B_i) \rightarrow A$ 可由F推出

$\iff A \in (X - B_i)_F^+$

证毕



& 此定理表明：任一函数依赖集都可极小化。定理的证明过程就是“极小化处理”过程，也就是求一个函数依赖集的极小依赖集的方法。

& F 的极小依赖集不一定唯一（例子见P187）。当以不同的顺序考察各函数依赖时，得到的极小函数依赖集可能不同。

第五章 关系数据理论

例：求下列函数依赖集F的极小依赖集。

$$F = \left\{ \begin{array}{llll} AB \rightarrow C, & D \rightarrow EG, & C \rightarrow A, & BE \rightarrow C, \\ BC \rightarrow D, & CG \rightarrow BD, & ACD \rightarrow B, & CE \rightarrow AG \end{array} \right\}$$

解：（1）分解右部为单属性

$$G = \left\{ \begin{array}{llll} AB \rightarrow C, & D \rightarrow E, & C \rightarrow A, & BE \rightarrow C, \\ & D \rightarrow G, & & \\ BC \rightarrow D, & CG \rightarrow B, & ACD \rightarrow B, & CE \rightarrow A, \\ & CG \rightarrow D, & & CE \rightarrow G \end{array} \right\}$$

（2）去掉G中多余的函数依赖

具体做法是：从G的第一个依赖开始，依次判别是否多余。设处理到函数依赖 $X \rightarrow Y$ ，从函数依赖集中先去掉它，求X关于此函数依赖集的闭包，再判属性Y是否属于该闭包。若是，则去掉 $X \rightarrow Y$ ；否则，保留。然后处理下一个。

第五章 关系数据理论

$$G = \left\{ \begin{array}{l} AB \rightarrow C, D \rightarrow E, D \rightarrow G, C \rightarrow A, BE \rightarrow C, BC \rightarrow D, \\ \cancel{CG \rightarrow B}, CG \rightarrow D, ACD \rightarrow B, \cancel{CE \rightarrow A}, CE \rightarrow G \end{array} \right\}$$

对此例：先判 $AB \rightarrow C$ 是否多余？

求 AB 关于 $G - \{AB \rightarrow C\}$ 的闭包，根据求闭包的算法，仍为 $\{A, B\}$ ，而 $C \notin \{A, B\}$ ，所以不能去掉 $AB \rightarrow C$ ；

同样可判 $D \rightarrow E, D \rightarrow G, C \rightarrow A, BE \rightarrow C, BC \rightarrow D$ 也不能去掉；
求 CG 关于 $G - \{CG \rightarrow B\}$ 的闭包，得 $ABCDEG$ ， B 在其中，
故去掉 $CG \rightarrow B$ ；

对 $G - \{CG \rightarrow B\}$ 继续进行，可得 $CG \rightarrow D, ACD \rightarrow B$ 保留，
 $CE \rightarrow A$ 去掉， $CE \rightarrow G$ 保留。

最后得：

问： $CG \rightarrow B$ 如何由 H 推出？

$CG \rightarrow CCG \rightarrow ACG \rightarrow ACCG \rightarrow ACD \rightarrow B$

$$H = \left\{ \begin{array}{l} AB \rightarrow C, D \rightarrow E, D \rightarrow G, C \rightarrow A, BE \rightarrow C, BC \rightarrow D, \\ CG \rightarrow D, ACD \rightarrow B, CE \rightarrow G \end{array} \right\}$$

(3) 从H中去掉左部多余的属性

具体做法是：依次检查左部为多属性的函数依赖，对每一个这样的函数依赖，检查其中是否有多余属性：若 $XB \rightarrow A$ ，求X关于H的闭包，看它是否包含A。

$$H = \left\{ \begin{array}{l} AB \rightarrow C, D \rightarrow E, D \rightarrow G, C \rightarrow A, BE \rightarrow C, \\ BC \rightarrow D, CG \rightarrow D, ACD \rightarrow B, CE \rightarrow G \end{array} \right\}$$

看 $AB \rightarrow C$ ，因为A关于H的闭包为{A}，B关于H的闭包为{B}，它们都不包含C，故 $AB \rightarrow C$ 无多余属性；

同样可得 $BE \rightarrow C, BC \rightarrow D, CG \rightarrow D, CE \rightarrow G$ 中无多余属性；

看 $ACD \rightarrow B$ ，因为

AC关于H的闭包为{A, C},

AD关于H的闭包为{A, D, E, G},

CD关于H的闭包为{A, B, C, D, E, G},

不含B

不含B

A多余

包含B

第五章 关系数据理论

因此在H中把 $ACD \rightarrow B$ 换成 $CD \rightarrow B$:

$$H = \left\{ \begin{array}{l} AB \rightarrow C, \quad D \rightarrow E, \quad D \rightarrow G, \quad C \rightarrow A, \quad BE \rightarrow C, \\ BC \rightarrow D, \quad CG \rightarrow D, \quad CD \rightarrow B, \quad CE \rightarrow G \end{array} \right\}$$

F的极小依赖集为:

$$F_m = \left\{ \begin{array}{l} AB \rightarrow C, \quad D \rightarrow E, \quad D \rightarrow G, \quad C \rightarrow A, \quad BE \rightarrow C \\ , \\ BC \rightarrow D, \quad CG \rightarrow D, \quad CD \rightarrow B, \quad CE \rightarrow G \end{array} \right\}$$

F还有另一个极小依赖集为:

$$F_{m2} = \left\{ \begin{array}{l} AB \rightarrow C, \quad D \rightarrow E, \quad D \rightarrow G, \quad C \rightarrow A, \\ BE \rightarrow C, \quad BC \rightarrow D, \quad CG \rightarrow B, \quad CE \rightarrow G \end{array} \right\}$$

本节开头

本章开头

下一节

§ 5 关系模式的分解

一、模式分解的有关概念

1、分解的定义：对关系模式 $R\langle U, F \rangle$ ，若模式集合

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

满足 ◆ $U = U_1 \cup U_2 \cup \dots \cup U_n$ ，且 $U_i \not\subseteq U_j$ ， $1 \leq i \neq j \leq n$ ，

◆ F_i 是 F 在 U_i 上的投影。

函数依赖集合 $\{ X \rightarrow Y \mid X \rightarrow Y \in F^+ \wedge XY \subseteq U_i \}$ 的一个覆盖

例：设模式为 $R\langle U, F \rangle$ ，其中 $U = \{ S\#, SD, MN \}$ ，

$F = \{ S\# \rightarrow SD, SD \rightarrow MN \}$ ，考虑下面几种分解：

(1) $U_1 = \{ S\# \}$ ， $U_2 = \{ SD \}$ ， $U_3 = \{ MN \}$

$F_1 = \Phi$ ， $F_2 = \Phi$ ， $F_3 = \Phi$

$\rho = \{ R_1\langle U_1, F_1 \rangle, R_2\langle U_2, F_2 \rangle, R_3\langle U_3, F_3 \rangle \}$ 是 R 的一个分解。

第五章 关系数据理论

(2) $U1 = \{ S\#, SD \}$, $U2 = \{ S\#, MN \}$
 $F1 = \{ S\# \rightarrow SD \}$,

参考前面的例子：设关系模式为 $R\langle U, F \rangle$ ，其中 $U = \{X, Y, Z\}$ ，
 $F = \{X \rightarrow Y, Y \rightarrow Z\}$ ， F 的闭包 F^+ 为：

$X \rightarrow \Phi$	$XY \rightarrow \Phi$	$XZ \rightarrow \Phi$	$XYZ \rightarrow \Phi$	$Y \rightarrow \Phi$	$YZ \rightarrow \Phi$	$Z \rightarrow \Phi$
$X \rightarrow X$	$XY \rightarrow X$	$XZ \rightarrow X$	$XYZ \rightarrow X$	$Y \rightarrow Y$	$YZ \rightarrow Y$	$Z \rightarrow Z$
$X \rightarrow Y$	$XY \rightarrow Y$	$XZ \rightarrow Y$	$XYZ \rightarrow Y$	$Y \rightarrow Z$	$YZ \rightarrow Z$	
$X \rightarrow Z$	$XY \rightarrow Z$	$XZ \rightarrow Z$	$XYZ \rightarrow Z$	$Y \rightarrow YZ$	$YZ \rightarrow YZ$	
$X \rightarrow XY$	$XY \rightarrow XY$	$XZ \rightarrow XY$	$XYZ \rightarrow XY$			
$X \rightarrow XZ$	$XY \rightarrow XZ$	$XZ \rightarrow XZ$	$XYZ \rightarrow XZ$			
$X \rightarrow YZ$	$XY \rightarrow YZ$	$XZ \rightarrow YZ$	$XYZ \rightarrow YZ$			
$X \rightarrow XYZ$	$XY \rightarrow XYZ$	$XZ \rightarrow XYZ$	$XYZ \rightarrow XYZ$			

第五章 关系数据理论

(2) $U1 = \{ S\#, SD \}$, $U2 = \{ S\#, MN \}$

$F1 = \{ S\# \rightarrow SD \}$, $F2 = \{ S\# \rightarrow MN \}$

$\rho = \{ R1 \langle U1, F1 \rangle, R2 \langle U2, F2 \rangle \}$ 也是R的一个分解。

参考前面的例子：设关系模式为 $R \langle U, F \rangle$ ，其中 $U = \{ X, Y, Z \}$ ，

$F = \{ X \rightarrow Y, Y \rightarrow Z \}$ ， F 的闭包 F^+ 为：

$X \rightarrow \Phi$	$XY \rightarrow \Phi$	$XZ \rightarrow \Phi$	$XYZ \rightarrow \Phi$	$Y \rightarrow \Phi$	$YZ \rightarrow \Phi$	$Z \rightarrow \Phi$
$X \rightarrow X$	$XY \rightarrow X$	$XZ \rightarrow X$	$XYZ \rightarrow X$	$Y \rightarrow Y$	$YZ \rightarrow Y$	$Z \rightarrow Z$
$X \rightarrow Y$	$XY \rightarrow Y$	$XZ \rightarrow Y$	$XYZ \rightarrow Y$	$Y \rightarrow Z$	$YZ \rightarrow Z$	
$X \rightarrow Z$	$XY \rightarrow Z$	$XZ \rightarrow Z$	$XYZ \rightarrow Z$	$Y \rightarrow YZ$	$YZ \rightarrow YZ$	
$X \rightarrow XY$	$XY \rightarrow XY$	$XZ \rightarrow XY$	$XYZ \rightarrow XY$			
$X \rightarrow XZ$	$XY \rightarrow XZ$	$XZ \rightarrow XZ$	$XYZ \rightarrow XZ$			
$X \rightarrow YZ$	$XY \rightarrow YZ$	$XZ \rightarrow YZ$	$XYZ \rightarrow YZ$			
$X \rightarrow XYZ$	$XY \rightarrow XYZ$	$XZ \rightarrow XYZ$	$XYZ \rightarrow XYZ$			

第五章 关系数据理论

(3) $U1 = \{ S\#, SD \}$, $U2 = \{ SD, MN \}$

$F1 = \{ S\# \rightarrow SD \}$, $F2 = \{ SD \rightarrow MN \}$

$\rho = \{ R1 \langle U1, F1 \rangle, R2 \langle U2, F2 \rangle \}$ 也是R的一个分解。

2、分解可能会带来的问题

(1) 分析第一种情形:

$U1 = \{ S\# \}$, $U2 = \{ SD \}$, $U3 = \{ MN \}$

$F1 = \Phi$, $F2 = \Phi$, $F3 = \Phi$

设关系r为:

S#	SD	MN
S1	D1	张五
S2	D1	张五
S3	D2	李四
S4	D3	王一

r1为: r2为: r3为:

S#	SD	MN
S1	D1	张五
S2	D2	李四
S3	D3	王一
S4		

问题: 关系r分解为r1、r2、r3三个关系, 但根据这三个关系, 无法回答诸如“S1是哪个系的学生?”、“D1系的系主任是谁?”等许多提问。

信息丢失

第五章 关系数据理论

分解后的关系应该能够恢复成原来的形式，这是最基本的要求。恢复通常由自然连接来实现。

r_1 、 r_2 、 r_3 自然连接的结果与 r 相比元组增加了，但信息丢失了。

分解不能丢失信息，这就是“无损连接性”的概念

(2) 分析第二种情形：

$$U_1 = \{ S\#, SD \}, \quad U_2 = \{ S\#, MN \}$$

$$F_1 = \{ S\# \rightarrow SD \}, \quad F_2 = \{ S\# \rightarrow MN \}$$

对此分解，可正确恢复出原来的关系。但原关系模式的数据冗余、插入异常、删除异常的问题仍然存在。分析原因，原来具有的函数依赖 $SD \rightarrow MN$ 现在没有了。

分解应该“保持函数依赖”

(3) 分析第三种情形:

$$U1 = \{ S\#, SD \}, \quad U2 = \{ SD, MN \}$$

$$F1 = \{ S\# \rightarrow SD \}, \quad F2 = \{ SD \rightarrow MN \}$$

对此分解，通过自然连接可正确恢复，且原来的函数依赖都保持。

3、分解准则

分解后的模式应和原来的模式“等价”。

根据以上分析，可提出以下不同标准：

- ☒ 分解具有“无损连接性” (Lossless join)
- ☒ 分解要“保持函数依赖” (Preserve dependency)
- ☒ 分解既要“保持函数依赖”，又要具有“无损连接性”

本节讨论：

- ◆ 何谓“无损连接性”和“保持函数依赖”？
- ◆ 分离程度如何？
- ◆ 如何分解？

二、分解的无损连接性和保持函数依赖性

1、无损连接性的形式化定义：

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。若对 R 的任何一个关系 r 都有

$$r = \pi_{U_1}(r) \bowtie \pi_{U_2}(r) \bowtie \dots \bowtie \pi_{U_k}(r)$$

成立，则称 ρ 具有无损连接性，简称 ρ 为无损分解。

例：设关系模式 $R(A, B, C)$ ，

分解为 $R_1(A, B)$ 、 $R_2(B, C)$ 。

不是无损分解

又设其中一个关系为：

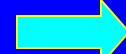
$$r \quad A \quad B \quad C \quad r_1 = \pi_{A,B}(r) \quad r_2 = \pi_{B,C}(r) \quad r_1 \bowtie r_2 \neq r$$

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



A	B
a1	b1
a2	b1

B	C
b1	c1
b1	c2



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

多一个元组

第五章 关系数据理论

例：设模式为 $R\langle U, F \rangle$ ，其中 $U=\{S\#, SD, MN\}$ ，

$F=\{S\#\rightarrow SD, SD\rightarrow MN\}$ ，

分解为： $U_1=\{S\#, SD\}$ ， $U_2=\{S\#, MN\}$

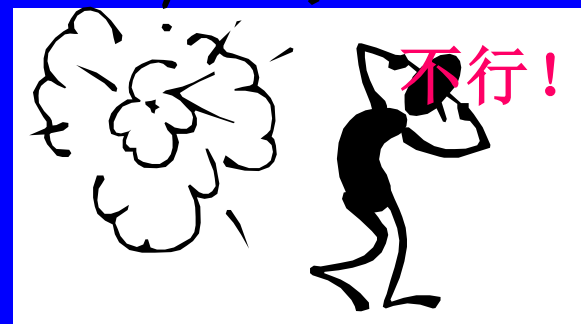
$F_1=\{S\#\rightarrow SD\}$ ， $F_2=\{S\#\rightarrow MN\}$

设关系 r 为： $r_1=\pi_{S\#,SD}(r)$ $r_2=\pi_{S\#,MN}(r)$ $r_1 \bowtie r_2 = r$

S#	SD	MN		S#	SD		S#	MN		S#	SD	MN
S1	D1	张五	→	S1	D1		S1	张五		S1	D1	张五
S2	D1	张五		S2	D1		S2	张五		S2	D1	张五
S3	D2	李四		S3	D2		S3	李四		S3	D2	李四
S4	D3	王一		S4	D3		S4	王一		S4	D3	王一

此时是否可以说这是一个无损分解？

如何判断是否为无损分解？



2、无损连接性判定算法

输入： $R\langle U, F\rangle$ 的一个分解 $\rho = \{ R_1(U_1, F_1), \dots, R_k(U_k, F_k) \}$,

$U = \{ A_1, A_2, \dots, A_n \}$,

$F = \{ FD_1, FD_2, \dots, FD_p \}$ 为最小覆盖, $FD_i: X_i \rightarrow A_i$;

输出： ρ 是否为无损连接的判定结果;

⌚ 构造一个 k 行 n 列的表：第 i 行对应关系模式 R_i ，第 j 列对应属性 A_j ；若 $A_j \in U_i$ ，则在第 i 行第 j 列处写入 a_j ；否则写入 b_{ij} 。

⌚ 逐个检查 F 中的每一个函数依赖，并修改表中的元素：

对每个 $FD_i: X_i \rightarrow A_i$ ，在 X_i 对应的列中寻找符号相同的行，并将这些行中 A_i 对应的列全改为相同的符号。修改规则为：若其中有 a_i ，则全改为 a_i ；否则全改为 b_{mli} ， m 是这些行中行号的最小值。（若某个 b_{tli} 被改动，则该列中凡是 b_{tli} 的符号都要作同样的修改。）

第五章 关系数据理论

① 判别：若某一行变成 a_1, a_2, \dots, a_n ，则算法终止（此时 ρ 为无损分解）；否则，比较本次扫描前后的表有无变化，若有，重复第①步；若无，算法终止（此时 ρ 不是无损分解）

无损连接判定举例：

设模式为 $R\langle U, F \rangle$, $U = \{A, B, C, D, E\}$, $F = \{AB \rightarrow C, C \rightarrow D, D \rightarrow E\}$, 分解为 $R_1(A, B, C)$ 、 $R_2(C, D)$ 、 $R_3(D, E)$ 。

算法第一步：构造初始表

	A	B	C	D	E
R1	a1	a2	a3	a4	a5
R2	b21	b22	a3	a4	a5
R3	b31	b32	b33	a4	a5

算法第二步：扫描F，修改表

对 $AB \rightarrow C$ ，看A、B对应的第1、2两列有无相同的行。没有，不做处理。

对 $C \rightarrow D$ ，看C对应的第3列有无相同的行
第1、2行相同，将D列的b14改为a4。

对 $D \rightarrow E$ ，进行同样处理。

算法第三步：判断

表中第一行为a1、a2、a3、a4、a5，算法终止， R_1 、 R_2 、 R_3 是R的无损分解

第五章 关系数据理论

例：设有关系模式 $R(A, B, C, D, E, F)$,

其函数依赖集合 $F = \{A \rightarrow B, C \rightarrow F, E \rightarrow A, CE \rightarrow D\}$,

分解为 $\rho = \{R_1(C, F), R_2(B, E), R_3(E, C, D), R_4(A, B)\}$

解：(1) 构造表

	A	B	C	D	E	F
R1(C,F)	b11	b12	a3	b14	b15	a6
R2(B,E)	b21	a2	b23	b24	a5	b26
R3(E,C,D)	b21	a2	a3	a4	a5	a6
R4(A,B)	a1	a2	b43	b44	b45	b46

(2) 修改表, 第一次扫描F

- $A \rightarrow B$: 无修改

- $C \rightarrow F$:
b36改为a6

- $E \rightarrow A$:
b31改为b21

- $CE \rightarrow D$: 无修改

(3) 第二次扫描F

- $A \rightarrow B$: b32改为a2
其余无修改

(4) 第三次扫描: 表无变化, 算法终止

(5) 判断: 不是无损分解

3、算法的正确性：

定理： ρ 为无损连接分解的充分必要条件是：

算法终止时，表中有一行为 a_1, a_2, \dots, a_n 。

证明略

4、分解为两个关系模式时的简单判定法

例：设关系模式为 $R\langle U, F \rangle$, $U = \{A, B, C\}$, $F = \{A \rightarrow B, C \rightarrow B\}$,
分别检查下列分解的无损连接性：

(1) $\rho_1 = \{R_1(A, B), R_2(B, C)\}$ (2) $\rho_2 = \{R_1(A, C), R_2(B, C)\}$

	A	B	C
R1(A, B)	a1	a2	b13
R2(B, C)	b21	a2	a3

对 $A \rightarrow B$ ，表无修改
对 $C \rightarrow B$ ，表无修改



	A	B	C
R1(A, C)	a1	a2	a3
R2(B, C)	b21	a2	a3

对 $A \rightarrow B$ ，表无修改
对 $C \rightarrow B$ ，修改b12为a2



第五章 关系数据理论

定理：设 $\rho = \{ R1\langle U1, F1 \rangle, R2\langle U2, F2 \rangle \}$ 是 $R\langle U, F \rangle$ 的一个分解。

那么 ρ 是无损分解的充分必要条件是

$U1 \cap U2 \rightarrow U1 - U2 \in F^+$ 或 $U1 \cap U2 \rightarrow U2 - U1 \in F^+$

即 $(U1 - U2) \in (U1 \cap U2) \stackrel{+}{F}$ 或 $(U2 - U1) \in (U1 \cap U2) \stackrel{+}{F}$ 证明略

看上例：关系模式为 $R\langle U, F \rangle$, $U = \{ A, B, C \}$, $F = \{ A \rightarrow B, C \rightarrow B \}$,

(1) $\rho1 = \{ R1(A, B), R2(B, C) \}$ (2) $\rho2 = \{ R1(A, C), R2(B, C) \}$

$U1 \cap U2 \rightarrow U1 - U2$ 为:

$B \rightarrow A$

$U1 \cap U2 \rightarrow U2 - U1$ 为:

$B \rightarrow C$

因为 $B \stackrel{+}{F} = \{B\}$
不包含A, 也不包含C,
所以 $B \rightarrow A$ 和 $B \rightarrow C$ 不为 F 所蕴涵

$\rho1$ 不是无损分解

$U1 \cap U2 \rightarrow U1 - U2$ 为:

$C \rightarrow A$

$U1 \cap U2 \rightarrow U2 - U1$ 为:

$C \rightarrow B$

因为 $C \rightarrow B \in F$, 所以

$\rho2$ 是无损分解



5、保持函数依赖性的形式化定义：

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，若 $F^+ = (F_1 \cup F_2 \cup \dots \cup F_k)^+$ ，则称 ρ 保持函数依赖。

判别定理： $F^+ = G^+$ 的充要条件是 $F \subseteq G^+$ 和 $G \subseteq F^+$ 。

6、说明

✗ 一个无损连接分解 不一定是 一个保持函数依赖的分解

✗ 一个保持函数依赖的分解 不一定是 一个无损连接分解

例：关系模式为 $R \langle U, F \rangle$ ， $U = \{ A, B, C \}$ ， $F = \{ A \rightarrow B, C \rightarrow B \}$ ，

(1) $\rho_1 = \{ R_1(A, B), R_2(B, C) \}$ (2) $\rho_2 = \{ R_1(A, C), R_2(B, C) \}$

有损连接

其 $F_1 = \{ A \rightarrow B \}$ ， $F_2 = \{ C \rightarrow B \}$

保持函数依赖

无损连接

其 $F_1 = \Phi$ ， $F_2 = \{ C \rightarrow B \}$

不保持函数依赖

三、模式分解的算法

1、模式分解的分离程度

保持函数依赖 \longrightarrow 3NF

具有无损连接性 \longrightarrow 4NF

保持函数依赖
具有无损连接性 \longrightarrow 3NF



2、转换为3NF的保持函数依赖的分解算法——合成法

算法思想： 若要将一个关系模式的极小函数依赖集中的每一个函数依赖都要保留下来，最简单的方法就是把每一个函数依赖的左右部属性组成一个模式。

如假设 $F = \{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$ ，则可考虑分解为：

$\rho = \{ R1(A, B), R2(A, C), R3(C, D) \}$

考虑合并

但 $A \rightarrow B, A \rightarrow C$ 左部相同，分解成两个模式是否有必要呢？

(左部相同的函数依赖放在一起考虑)

第五章 关系数据理论

算法5.3: 把一个关系模式分解为3NF, 且使它保持函数依赖

输入: 关系模式 $R\langle U, F \rangle$;

输出: R 的一个分解 $\rho = \{R_1, R_2, \dots, R_k\}$, 每个 R_i 均为3NF, 且 ρ 保持函数依赖

(1) 极小化: 对 F 进行极小化处理 (处理后的函数依赖集仍记为 F);

(2) 分离无关属性: 把 U 中不出现在 F 中的属性组成一个关系模式, 从 U 中去掉这些属性 (剩余的属性仍记为 U);

(3) 判是否需分解: 若有 $X \rightarrow A \in F$, 且 $XA = U$, 则 $\rho = \{R\}$, 算法终止;

(4) 分解: 对 F 分组: 左部相同的为一组, 设为 F_i , 每组所包含的全部属性形成一个属性集 U_i 。若有 $U_i \subseteq U_j$ ($i \neq j$), 去掉 U_i 。设最后保留下来 U_1, U_2, \dots, U_k , 则 $\rho = \{R_1(U_1), R_2(U_2), \dots, R_k(U_k)\}$, 算法终止。

第五章 关系数据理论

例1: 设关系模式 $R(A, B, C, G, H, R, S, T)$,

$F = \{ C \rightarrow T, CS \rightarrow G, HT \rightarrow R, HR \rightarrow C, HS \rightarrow R \}$

求: (a) R 的一个候选关键字;

(b) 将 R 分解为 3NF, 并且保持函数依赖性。

解 (a) 求关键字: 设 X 为候选关键字, 则必有 $X_F^+ = U$ 。

而 $X_F^+ - X$ 只能取 F 中函数依赖右部的属性, X_F^+ 要等于 U , 至少应有 $ABHS \in X$ 。

求 $ABHS$ 关于 F 的闭包有: $(ABHS)_F^+ = ABHSRCTG = U$
所以 $ABHS$ 是一个候选关键字。

(b) 分解 R : (1) 先对 F 极小化: F 已是最小依赖集。

(2) 分离无关属性: AB 未在 F 中出现, 分离出去。

(3) 判是否需分解: 需要。

(4) 分解: 按左部相同的原则分组。

则 $\rho = \{ R_1(A, B), R_2(C, T), R_3(C, S, G), R_4(H, T, R), R_5(H, R, C), R_6(H, S, R) \}$ 为一个保持函数依赖且达到 3NF 的分解。

第五章 关系数据理论

例2: 设有关系模式 $R(A, B, C, D, E)$, R 的一个函数依赖集为 $F = \{A \rightarrow D, A \rightarrow B, E \rightarrow D, D \rightarrow B, BC \rightarrow D, DC \rightarrow A\}$,

(a) 求 R 的候选关键字;

(b) 将 R 分解为 3NF, 并且保持函数依赖性。

解 (a) 求关键字: 设 X 为候选关键字, 则必有 $X_F^+ = ABCDE$ 。

而 $X_F^+ - X$ 只能取 F 中函数依赖右部的属性, 要使 X_F^+ 等于 $ABCDE$, 必须有 $CE \in X$ 。

再求 CE 关于 F 的闭包, 得 $(CE)_F^+ = CEDBA$ 。

所以 CE 是一个候选关键字。

(b) 分解 R :

(1) 先对 F 极小化:

① 分解右部为单属性: 不需要;

② 去掉多余函数依赖: $A \rightarrow B$ 多余;

若此时还不等于
 R 的属性集 U ,
怎么办?

第五章 关系数据理论

例2：设有关系模式 $R(A, B, C, D, E)$ ， R 的一个函数依赖集为 $F = \{A \rightarrow D, A \rightarrow B, E \rightarrow D, D \rightarrow B, BC \rightarrow D, DC \rightarrow A\}$,

(a) 求 R 的候选关键字;

(b) 将 R 分解为3NF, 并且保持函数依赖性。

③去掉左部多余属性：没有。

所以极小函数依赖集为：

$F = \{A \rightarrow D, E \rightarrow D, D \rightarrow B, BC \rightarrow D, DC \rightarrow A\}$

(2) 分离无关属性：没有。

(3) 判是否需分解：需要。

(4) 分解：

~~$\{AD, ED, DB, BCD, DCA\}$~~

$\rho = \{R_1(E, D), R_2(B, C, D), R_3(D, C, A)\}$

例3：设有关系模式 $R(A, B, C, D)$ ， R 的一个函数依赖集为
 $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$ ，
试将 R 分解为3NF，并且保持函数依赖性。

解

(1) 先对 F 极小化： F 就是最小依赖集。

(2) 分离无关属性：没有。

(3) 判是否需分解：需要。

(4) 分解：

$$\rho = \{R_1(A, B), R_2(B, C), R_3(C, D), R_4(D, A)\}$$

算法的正确性：

按算法5.3得到的关系模式 R 的分解 $\rho = \{R_1, R_2, \dots, R_k\}$ 满足：
每个 R_i 均为3NF，且 ρ 保持函数依赖。

第五章 关系数据理论

证明：(a) ρ 保持函数依赖显然成立。

(b) 证每个 $R_i \langle U_i, F_i \rangle$ 均为3NF (F_i 是 F 在 U_i 上的投影)：

设 F 是 R 的最小依赖集， F_i' 是分解算法中得到 R_i 的那组函数依赖，

要证不存在非主属性对关键字的传递函数依赖。

设 $F_i' = \{ X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n \}$, $U_i = \{ X, A_1, \dots, A_n \}$,

则 X 一定是 R_i 的候选关键字。

反证法：假设 R_i 不是由 F 按左部相同的原理分组而得。存在非主属性 A_m 和属性组合 Y 使得 $A_m \notin Y$, $X \rightarrow Y, Y \rightarrow A_m \in F_i'$, 而 $Y \rightarrow X \notin F_i'$ 。

非主属性 A_m 传递依赖于候选关键字 X

下面证 $X \rightarrow A_m$ 在 F 中多余，从而与 F 是最小依赖集矛盾。

令 $G = F - \{ X \rightarrow A_m \}$, 欲证 $X \rightarrow A_m \in G^+$ 。

第五章 关系数据理论

因为 Y 是 U_i 的子集，且不包含 A_m ，所以 $Y \subseteq X_G^+$ ，因此 $X \rightarrow Y \in G^+$ 。

要证 $X \rightarrow A_m \in G^+$ ，只需再证 $Y \rightarrow A_m \in G^+$ 。

因为 $Y \rightarrow A_m \in F_i^+$ ，所以 $A_m \in Y_F^+$ 。

若 $Y \rightarrow A_m \notin G^+$ ，则在求 Y_F^+ 的算法中，只有使用 $X \rightarrow A_m$ 才能将 A_m 引入。

根据求闭包的算法，存在 j 使得 $X \subseteq Y^{(j)}$ ，故得 $Y \rightarrow X \in F^+$ ，

与 $Y \rightarrow X \notin F_i^+$ 相矛盾，所以 $Y \rightarrow A_m \in G^+$ 成立。

证毕

说明：因为保持函数依赖的分解可能不是无损连接的分解，而不具有无损连接性就会丢失信息，所以这样的分解是没有意义的。因此单单算法5.3没有实用价值，但它是下一算法的基础。

2、转换为3NF既保持函数依赖又具有无损连接性的分解算法

算法思想:在算法5.3分解的基础上, 增加一个模式, 该模式以原模式的一个关键字作为属性组, 使该模式起到正确连接其它各模式的作用。

例如, 对 $R(A, B, C, D)$, 其最小依赖集 $F=\{A \rightarrow B, C \rightarrow D\}$, 按算法5.3得到的分解是 $\rho_1=\{AB, CD\}$, ρ 不具有无损连接性。 R 的关键字是 AC , 分解成 $\rho_2=\{AB, CD, AC\}$ 就具有无损连接性。

算法5.4: 输入: 关系模式 $R\langle U, F \rangle$;

输出: R 的一个分解 $\tau=\{R_1, R_2, \dots, R_p\}$, 每个 R_i 均为3NF,
且 τ 既保持函数依赖又具有无损连接性;

- (1) 用算法5.3将 R 分解为 $\rho=\{R_1(U_1), R_2(U_2), \dots, R_k(U_k)\}$;
- (2) 求 R 的一个关键字 X ;
- (3) 若 X 是某个 U_i 的子集, 输出 $\tau=\rho$;
否则, 输出 $\tau=\rho \cup \{R^*(X)\}$; 算法结束。

第五章 关系数据理论

算法的正确性：按算法5.4得到的关系模式 R 的分解 $\tau=\{R_1, R_2, \dots, R_p\}$ 满足：
每个 R_i 均为3NF，且 τ 既保持函数依赖又具有无损连接性。

证明：（a） τ 保持函数依赖显然成立。

（b）证每个 R_i 均为3NF：

算法5.3保证 R^* 之外的 R_i 都为3NF； R^* 中没有非主属性，因此也为3NF。

（c）证 τ 为无损分解：

分析：增加 R^* 的目的就是为了达到无损连接，

按照无损连接判定算法， R^* 所对应的行应该成为全 a 。

R^* 不一定在 τ 中，但 τ 中必存在某关系模式的属性组 T 包含 X ，

为简单起见，不妨设 T 就是 X ， R^* 就在 τ 中。

按照无损连接判定算法，在构造初始判定表时，对 R^* 所对应的行，

X 对应的列全为 a ，设 $U-X=\{A_1, A_2, \dots, A_s\}$ ，欲证经有限步执行判定算法，

该行中 A_1, A_2, \dots, A_s 所对应的列也被改为全 a 。

第五章 关系数据理论

因为X是关键字，故 $X_F^+ = U$.
所以在求X的闭包时，A1、A2、...As将进入X的闭包，不妨设进入的次序就是A1、A2、...As，则根据求闭包的算法，会产生序列：

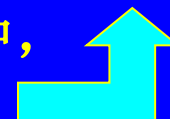
$$X \subseteq X^{(1)} \subseteq X^{(2)} \dots \subseteq X^{(i)} \dots = U$$

属性 模式	X	U-X	
		A1 ... Ai	... As
$R^*(X)$	aaa...aaaa	b b...b	b ... b
$R1(Y1A1)$	baa...abba	a b...b	b ... b
$Ri(YiAi)$	bab...babb	b a... a	b ... b
$Rs(YsAs)$	aab...abaa	a b...b	a ... a
...

满足

$$Yi \rightarrow Ai \in F, \text{ 且 } Yi \subseteq X^{(i-1)} = XA1A2 \dots A(i-1).$$

不失一般性，可设R1(Y1A1)、R2(Y2A2)、... Rs(YsAs)在τ中，
初始判定表如右上角所示。



第五章 关系数据理论

证按无损判定算法, R^* 对应的行中 A_1, A_2, \dots, A_s 所对应的列被改为全 a 。

用归纳法:

当 $s=1$ 时, 有 $Y_1 \rightarrow A_1 \in F$, 故 R_1 所在的行中 Y_1 对应的列全 a ;

因为 $Y_1 \subseteq X^{(0)} = X$, 而 R^* 所在行中 X 对应的列全 a , 所以至少这两行中 Y_1 对应的列全 a , 故相等。

考察 A_1 对应的列:

因 R_1 行对应的为 a , 故 R^* 行 A_1 列也改为 a 。

属性 模式	X	U-X
		A1 ... Ai ... As
...
$R^*(X)$	aaa...aaaa	a a ... a a ... a
...
$R_1(Y_1 A_1)$	baa...abba	a b...b b ... b
...
$R_i(Y_i A_i)$	bab...babb	b a.. a b ... b
...
$R_p(Y_p A_p)$	aab...abaa	a b...b a ... a
...

假设 $s=i-1$ 时, R^* 所在行 $A_1, A_2, A_{(i-1)}$ 对应列能够全改为 a 。

当 $s=i$ 时, 有 $Y_i \rightarrow A_i \in F$, 且 $Y_i \subseteq X^{(i-1)} = X A_1 A_2 \dots A_{(i-1)}$ 。

与 $s=1$ 时的同样道理, 根据 R_i 行的 A_i 列为 a , 可将 R^* 行 A_i 列也改为 a 。
根据归纳法原理, 结论成立。

证毕

第五章 关系数据理论

3、转换为BCNF的无损连接分解——分解法

算法思想：按自顶向下的方式，从关系模式 $R\langle U, F \rangle$ 开始，选择不是BCNF的模式将其一分为二，直到都为BCNF。（二叉分解）

算法5.5： 输入：关系模式 $R\langle U, F \rangle$ ；

输出： R 的一个分解 $\rho = \{R_1, R_2, \dots, R_k\}$ ，每个 R_i 均为BCNF，且 ρ 具有无损连接性；

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

正确性证明自己看

(2) 检查：检查 ρ 各关系模式是否都是BCNF。若是，算法终止。

(3) 分解：设 ρ 中 $R_i\langle U_i, F_i \rangle$ 不是BCNF，

则存在 $X \rightarrow A \in F_i^+$ ($A \notin X$)， X 不是 R_i 的关键字；

对 R_i 作无损分解： $\sigma = \{S_1, S_2\}$

因 X 不是 R_i 的关键字，故 XA 是 U_i 的真子集， X 且容易验证， σ 是 R_i 的一个无损分解。

$S_1\langle XA \rangle$

以 σ 代 R_i ，返回第(2)步。 $S_2\langle X(U_i - X - A) \rangle$

X	$U_i - X$	
	A	$U_i - X - A$
a...a	a	bbb ... bb
a...a	a	aaa ... aa

第五章 关系数据理论

4、转换为4NF的无损连接分解

定理：设关系模式为 $R\langle U, D \rangle$ ，其中 D 为 R 中函数依赖和多值依赖的集合，
则 $X \twoheadrightarrow Y$ 成立

$\iff R$ 的分解 $\rho = \{ R_1\langle XY \rangle, R_2\langle XZ \rangle \}$ 具有无损连接性，
其中 $Z = U - X - Y$ 。

证明：充分性。要证 若 ρ 是无损连接，则 $X \twoheadrightarrow Y$ 成立。

多值依赖的定义：

多值依赖 $X \twoheadrightarrow Y$ 成立当且仅当对 $R(U)$ 的任一关系 r ，若存在元组 s, t 使得 $s[X] = t[X]$ ，则必存在元组 $w, v \in r$ (w, v 可以与 s, t 相同)，使得 $w[X] = v[X] = t[X]$ ，而 $w[Y] = t[Y]$ ， $w[Z] = s[Z]$ ， $v[Y] = s[Y]$ ， $v[Z] = t[Z]$ 。

交换 s, t 的 Y 值
所得新元组仍在 r 中

第五章 关系数据理论

4、转换为4NF的无损连接分解

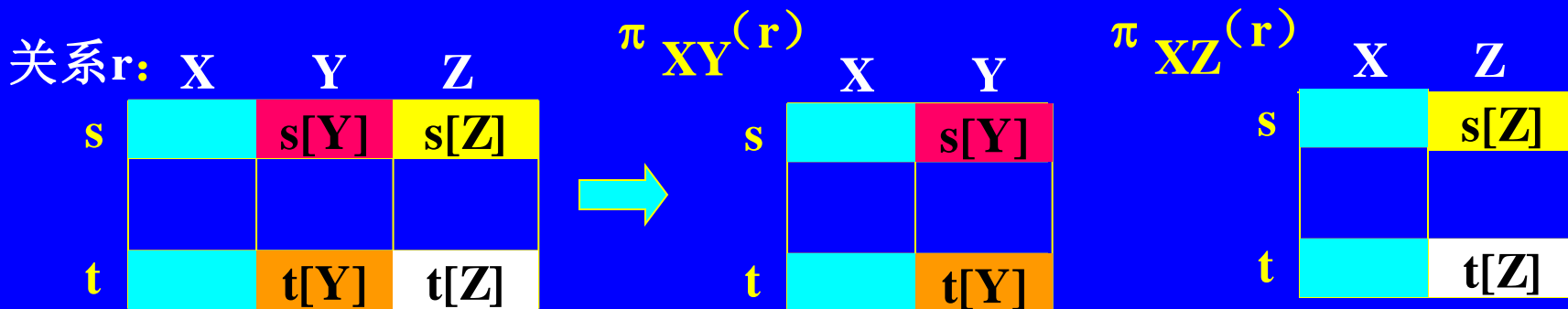
定理：设关系模式为 $R\langle U, D \rangle$ ，其中 D 为 R 中函数依赖和多值依赖的集合，
则 $X \twoheadrightarrow Y$ 成立

\iff R 的分解 $\rho = \{ R_1\langle XY \rangle, R_2\langle XZ \rangle \}$ 具有无损连接性，
其中 $Z = U - X - Y$ 。

证明：充分性。要证 若 ρ 是无损连接，则 $X \twoheadrightarrow Y$ 成立。

欲证：对 R 的任一关系 r ，若 $t, s \in r$ ，且 $t[X] = s[X]$ ，则存在 $u, v \in r$ 使得
 $u[X] = v[X] = t[X]$ ，而 $u[Y] = t[Y]$ ， $u[Z] = s[Z]$ ， $v[Y] = s[Y]$ ， $v[Z] = t[Z]$ 。

因为 ρ 是无损连接，对 R 的任一关系 r ，有 $r = \pi_{XY}(r) \bowtie \pi_{XZ}(r)$ 。
若 $t, s \in r$ ，且 $t[X] = s[X]$ ，



第五章 关系数据理论

若 $t, s \in r$, 且 $t[X]=s[X]$,

令 $u = t[X] \cdot t[Y] \cdot s[Z]$,

$v = s[X] \cdot s[Y] \cdot t[Z]$,

则 $u, v \in r$ 满足 $u[X]=v[X]=t[X]$,

而 $u[Y]=t[Y]$, $u[Z]=s[Z]$,

$v[Y]=s[Y]$, $v[Z]=t[Z]$ 。

所以 $X \rightarrow \rightarrow Y$ 成立。

必要性。要证

若 $X \rightarrow \rightarrow Y$ 成立, 则 ρ 是无损连接。

欲证: 对 R 的任一关系 r , 有

$$r = \pi_{XY}(r) \bowtie \pi_{XZ}(r)。$$

而 $r \subseteq \pi_{XY}(r) \bowtie \pi_{XZ}(r)$ 显然成立,

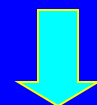
故只需证 $r \supseteq \pi_{XY}(r) \bowtie \pi_{XZ}(r)$ (自己证)

$\pi_{XY}(r)$

	X	Y
s		s[Y]
t		t[Y]

$\pi_{XZ}(r)$

	X	Z
s		s[Z]
t		t[Z]



	X	Y	Z
s		s[Y]	s[Z]
t		t[Y]	t[Z]
u		t[Y]	s[Z]
v		s[Y]	t[Z]

证毕

第五章 关系数据理论

说明：上述定理是在函数依赖和多值依赖的范畴内讨论问题，因此需要建立一套新的公理系统来保证“逻辑蕴涵”和“导出”仍是等价的。幸运的是，只要简单扩充Armstrong公理系统（增加关于多值依赖的公理），该公理系统仍是有效的和完备的。

转换为4NF的无损连接分解算法：

以算法5.5作为第一步，规范化到4NF时方法也同算法5.5（二叉分解）。

算法5.6: 输入：关系模式 $R\langle U, D \rangle$;
输出： R 的一个分解 $\rho = \{R_1, R_2, \dots, R_k\}$ ，每个 R_i 均为4NF，
且 ρ 具有无损连接性；

- (1) 用算法5.5先将 R 分解到BCNF，且具有无损连接性。
- (2) 对其中没达到4NF的模式，根据前面定理分解，直到所有模式都为4NF时为止。

作业:

P196 习题 2、3、12

补充:

1、设 $F=\{AC \rightarrow PE, PG \rightarrow A, B \rightarrow CE, A \rightarrow P, GA \rightarrow B, GC \rightarrow A, PAB \rightarrow G, AE \rightarrow GB, DP \rightarrow H\}$,

$X=AG$,

求 X_F^+

2、设有关系模式 $R(ABCDE)$ ， R 的函数依赖集为

$F=\{A \rightarrow C, B \rightarrow C, C \rightarrow D, DE \rightarrow C, CE \rightarrow A\}$,

(1) 设 $\rho=\{R1(AD), R2(AB), R3(BE), R4(CDE), R5(AE)\}$

是 R 的一个分解。试判断 ρ 是否为无损分解。

(2) 给出 R 的一个既保持函数依赖又具有无损连接的3NF分解。

本节开头

本章开头

第六章 数据库设计

本章内容：

- § 1 数据库设计概述
- § 2 需求分析
- § 3 概念结构设计
- § 4 逻辑结构设计
- § 5 物理结构设计
- § 6 数据库的实施和维护

请选择内容

返回

§ 1 数据库设计概述

数据库设计是指对一个给定的应用环境，构造最优的数据库模式，建立数据库及其应用系统，使之能够有效地存取数据，满足各种用户的应用需求。

一、数据库和信息系统

大型数据库的设计和开发是涉及多学科的综合性的技术，对涉及人员来说应具备多方面的技术知识，主要有：

- ◆ 数据库的基本知识和数据库的设计知识；
- ◆ 计算机科学的基础知识和程序设计的方法和技巧
- ◆ 软件工程的原理和方法
- ◆ 应用领域的知识

二、数据库设计的特点：

- “三分技术，七分管理，十二分基础数据”
- 数据库设计和应用系统设计相结合，即将结构设计和行为设计相结合

三、数据库设计方法简述

数据库设计方法中著名的新奥尔良方法，将数据库的设计分为四个阶段：

- ◆ 需求分析（分析用户要求）
- ◆ 概念设计（信息分析和定义）
- ◆ 逻辑设计（设计实现）
- ◆ 物理设计（物理数据库设计）

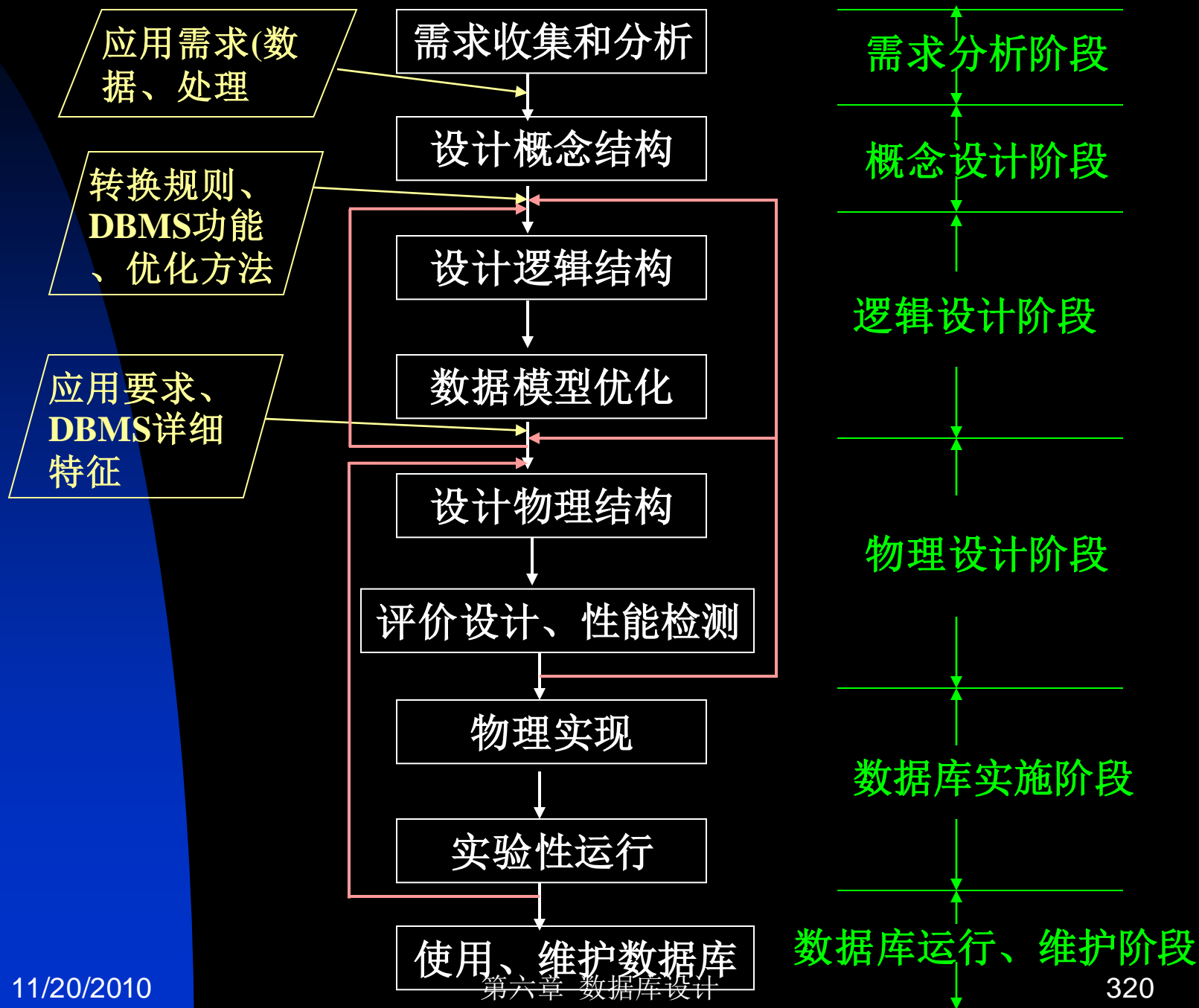
数据库设计工具：**ORACLE 公司 Design 2000**

Sybase 公司的PowerDesigner

四、数据库设计的基本步骤

数据库设计的基本步骤

- **需求分析**：了解与分析用户需求，是最困难、最费时间的一步。
- **概念结构设计**
 - ◆ 通过对用户的需求进行综合、归纳和抽象，形成一个独立于具体的**DBMS**的概念模型
- **逻辑结构设计**
 - ◆ 将概念结构转换为某个**DBMS**所支持的模型，并对其进行优化
- **物理结构设计**
 - ◆ 为逻辑数据模型选取一个最适合应用环境的物理结构（包括存取结构和存取方法）
- **数据库实施**
 - ◆ 运用**DBMS**提供的数据库语言及其宿主语言，根据逻辑设计和物理设计的结果建立数据库，编制与调试应用程序，组织数据库，并进行试运行
- **数据库运行和维护**



§ 2 需求分析

一、需求分析的任务

通过详细调查现实世界要处理的对象(组织、部门、企业等),充分了解原系统工作概况,明确用户的各种需求,然后在此基础上确定新系统的功能。

调查的重点是“数据”和“处理”,通过调查、收集与分析,获得用户对数据库的如下要求:

- 信息要求

- ◆ 指用户需要从数据库中获得信息的内容和性质,从而导出应在数据库中存储哪些数据

- 处理要求

- ◆ 用户完成什么样的处理功能,对处理的响应时间有什么要求
- ◆ 处理方式是批处理还是联机事务处理

- 安全性与完整性要求

二、需求分析的方法

- 调查组织机构情况
 - ◆ 了解该组织的部门组成情况，各部门的职责，为分析信息流程做准备
- 调查各部门的业务活动情况
 - ◆ 了解各部门的输入和使用什么样的数据
 - ◆ 如何加工这些数据
 - ◆ 输出什么信息
 - ◆ 输出到什么部门
 - ◆ 信息输出结果的格式
- 协助用户明确对新系统的各种要求
 - ◆ 信息要求、处理要求、安全性与完整性要求
- 确定新系统的边界
 - ◆ 确定那些由计算机来完成，那些由人工来完成

需求调查的方法:

- **跟班作业** 参加业务工作来了解业务活动的情况,此种方法可以准确地了解用户的需求,但是比较耗费时间
- **开会调查** 通过与用户座谈来了解业务活动情况,座谈时,参加者之间可以相互启发
- **请专人介绍**
- **询问**
- **设计调查表请用户填写**。如果调查表设计的合理,这种方法是很有效,也易于为用户接受。
- **查阅记录**

做需求调查时,往往需要同时采用上述多种方法。但是无论采用何种方法,都需要用户的配合。

三、数据字典

数据流图表达了数据和处理的关系，数据字典则是系统中各类数据描述的集合，是进行详细设计的数据收集和数据分析所获得的主要成果，数据字典在数据库设计中占有很重要的作用。数据字典通常包括数据项、数据结构、数据流、数据存储和处理过程五个部分。

■ 数据项

数据项是不可再分的数据单位。对数据项的描述通常包括以下内容

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

■ 数据结构：反应了数据之间的组合关系

数据结构描述={ 数据结构名, 含义说明, 组成: {数据项或数据结构}}

数据流：是数据结构在系统内传输的路径。

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

■ 数据存储

是数据结构停留或保存的地方, 也是数据流的来源和去向之一, 也可以是手工文档或手工凭单, 也可以是计算机文档。

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

■ 处理过程

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

简要说明: 说明该处理过程的功能及处理要求, 功能是指该处理过程用来干什么

处理要求包括处理频度要求, 如单位时间里处理多少事务、多少、的数据量、响应时间。

§ 3 概念结构设计

将需求分析得到的用户需求抽象为信息结构即概念模型的过程就是概念结构设计。

一、概念结构的主要特点：

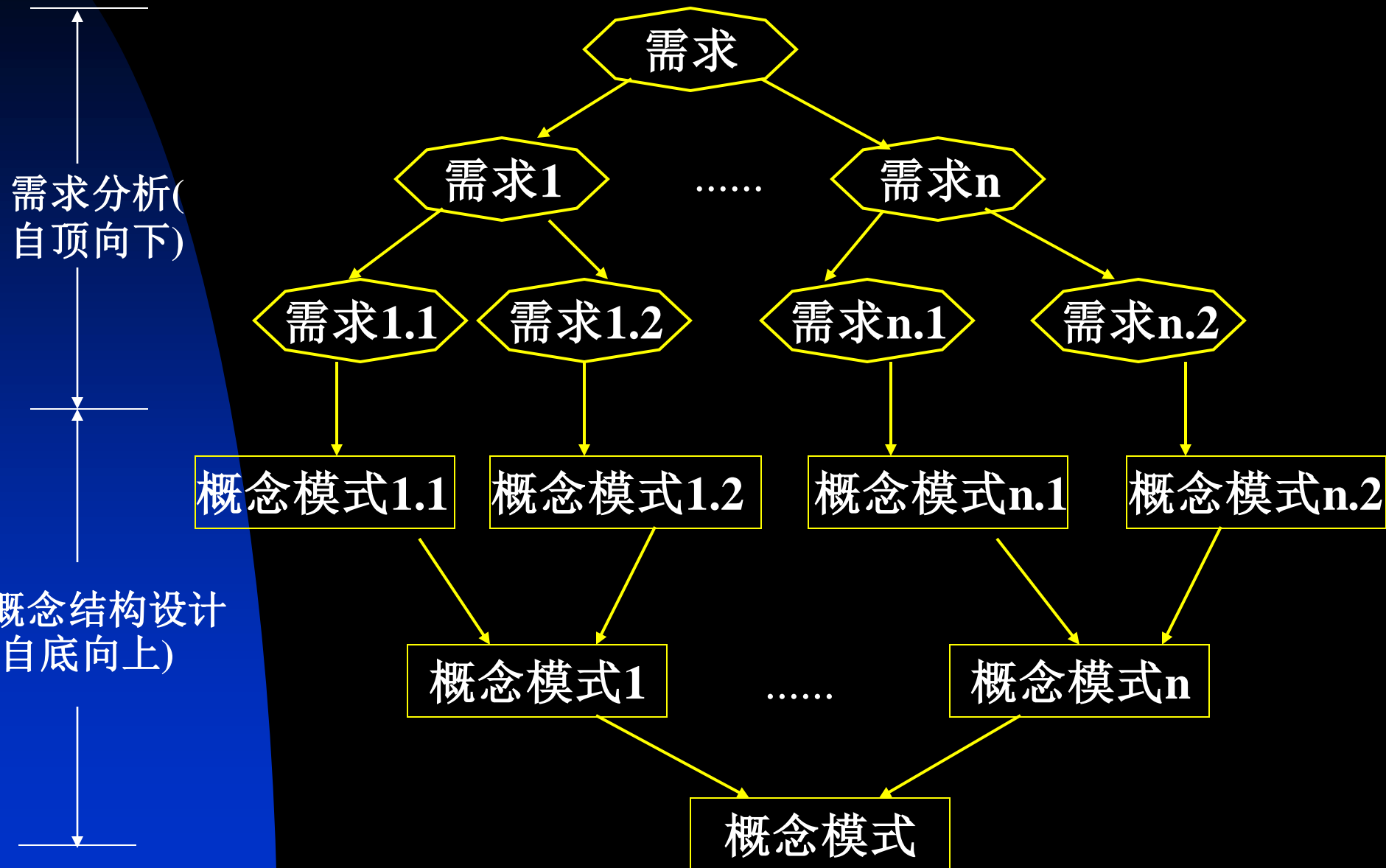
- 能真实充分地反映客观世界，包括事物和事物之间的联系，满足用户对数据的处理要求
- 易于理解
- 易于更改
- 易于向关系、网状、层次等各种数据模型转换

二、概念结构设计的方法与步骤

概念结构设计的方法：

- **自顶向下** 首先定义各全局概念框架，然后逐步细化
- **自底向上** 首先定义各局部应用的概念框架，然后将他们集中起来，得到全局概念结构
- **逐步扩张** 首先定义最重要的核心概念结构，然后向外扩充，以滚雪球的方式逐步生成其它概念结构，直至总体概念结构
- **混合策略** 将自顶向下和自底向上相结合，用自顶向下策略设计一个全局的概念结构框架，以它为骨架集成由底向上策略中设计的各局部概念框架。

自顶向下分析需求与自底向上设计概念结构



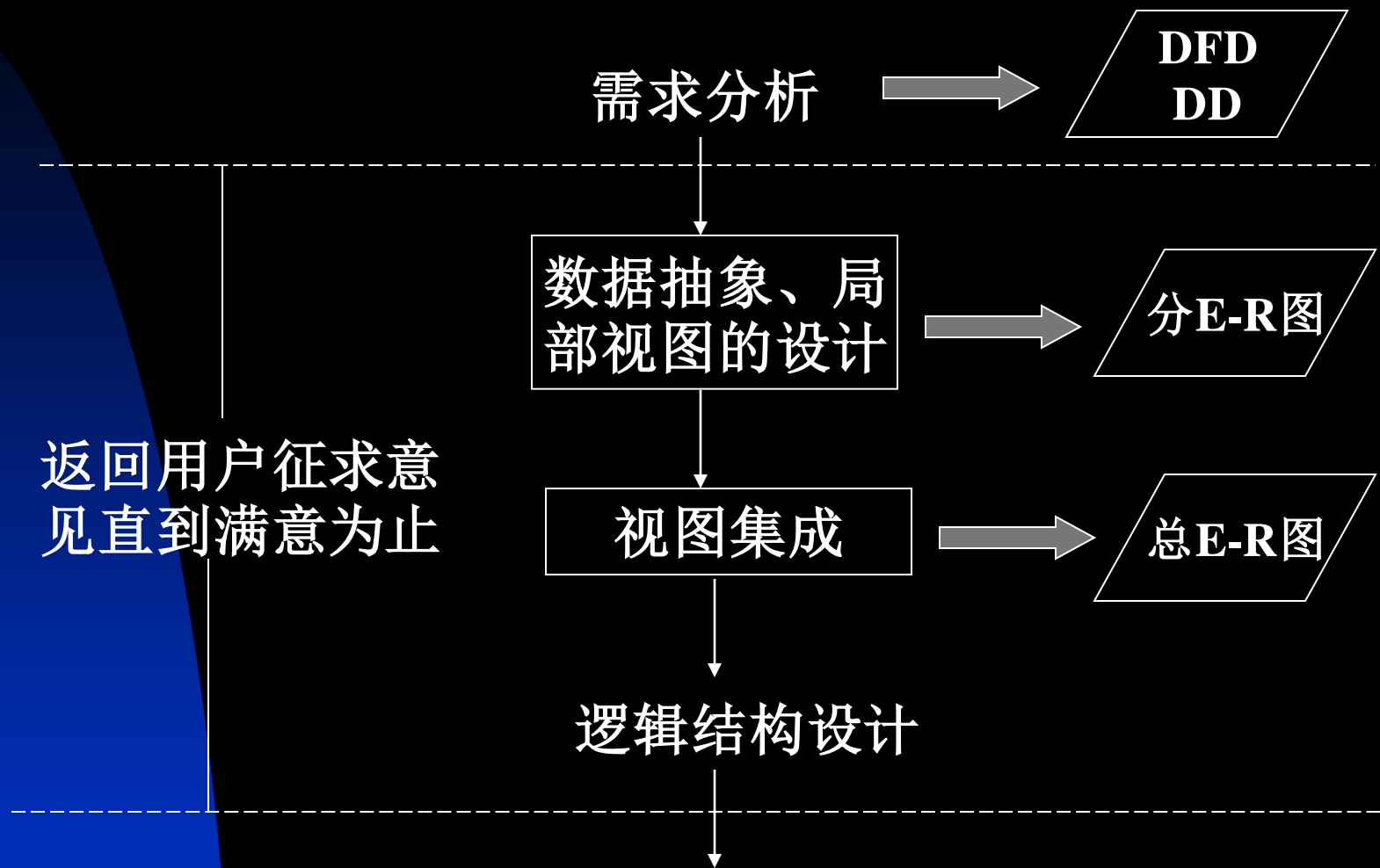


图6.9逻辑结构设计步骤

三、数据抽象与局部视图设计

抽象是对实际的人、物、事和概念进行人为处理，抽取所关心的共同特性，忽略非本职的细节，并把这些特性用各种概念精确地加以描述，这些概念组成了某些模型。

- **分类** 定义某一类概念作为现实世界中的一组对象的类型。这些对象具有某些共同的特性和行为。它抽象了对象值和型之间的 (**is member of**) 语义。
- **聚集** 定义了某一类型的组成成分。它抽象了对象内部类型和成分之间的 (**is part of**) 语义。
- **概括** 定义了类型之间的一种子集联系。它抽象了类型之间的 (**is subset of**) 的语义。如学生是一个实体型，本科生、研究生也是一个实体型。本科生和研究生是学生的子集，把学生作为超类(**Superclass**)，本科生、研究生是子类(**Subclass**)。

概念结构设计的第一步是利用抽象机制对需求分析阶段收集的数据进行分类、组织（聚集），形成实体、实体的属性，标识实体的码，确定实体之间的联系(1:1, 1:m, n:m)，设计分E-R图。具体步骤：

- 选择局部应用
- 逐一设计分E-R图

四、视图的集成

各子系统的分E-R图设计好之后，就要将各分E-R图综合成一个系统的总E-R图。视图集成的方式有两种：

- 多个分E-R图一次集成 复杂，难度大
- 逐步集成 用累加的方法一次集成两个分E-R图

每次只集成两个分E-R图，可以降低复杂度

无论采用以上那种方式，每次集成局部E-R图都要分两步走

- ◆ 合并 解决各分E-R图的冲突，将各分E-R图合并起来生成初步的E-R图。
 - ★ 属性冲突 属性值冲突和属性域冲突
 - ★ 命名冲突 同名异义和异名同义

★ 结构冲突 同一对象在不同应用上有不同的抽象
同一实体在不同的分E-R图所包含的属性个数和属性排列次序不完全相同。

◆ 修改和重构 消除不必要的冗余，生成基本E-R图

消除冗余的方法：

1. 分析方法：即以数据字典和数据流图为依据，根据数据字典中关于数据项之间的逻辑关系的说明来消除冗余。
2. 用规范化理论中函数依赖的概念来消除冗余

用规范化理论中函数依赖的来消除冗余的方法如下：

- 确定分E-R图实体之间的数据依赖

实体之间的一对一、一对多或多对多的联系可以用实体码之间的函数依赖来表示

- 求函数依赖集 F_1 的最小覆盖 G_1 ，差集为 $D = F_1 - G_1$

逐一检查 D 中的函数依赖，确定是否是冗余的联系，若是就去掉

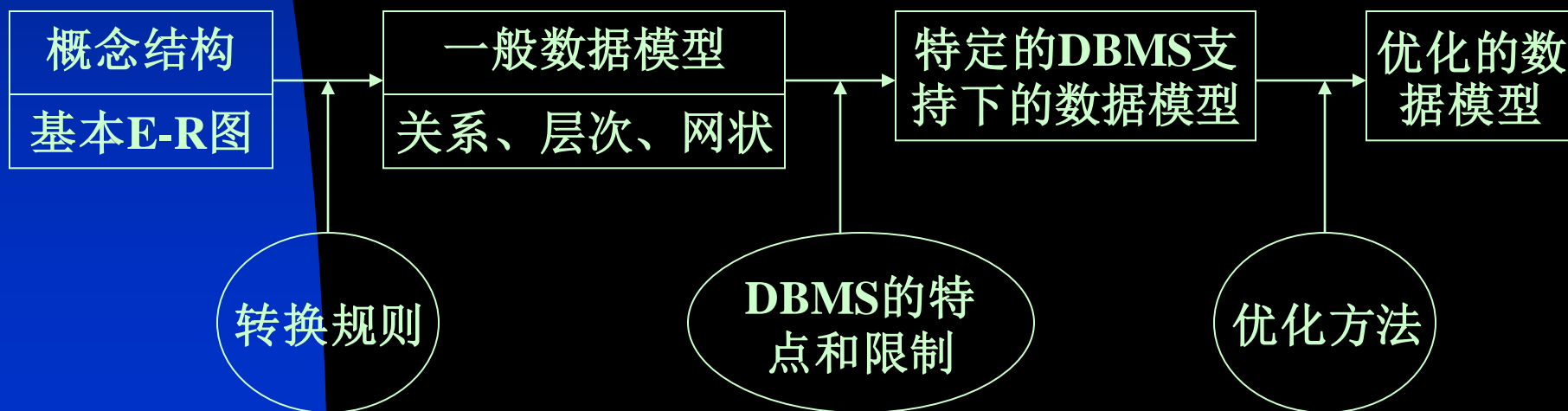
注意的问题：

- ◆ 冗余的联系一定在 D 中，但 D 中的联系不一定是冗余的
- ◆ 当实体之间存在多种联系时，要将实体之间的联系在形式上加以区分

§ 4 逻辑结构设计

逻辑结构分三步进行：

- ◆ 将概念结构转换成一般的关系、层次、网状模型
- ◆ 将转换来的关系、层次、网状模型向特定的DBMS支持下的数据模型转换
- ◆ 对数据模型进行优化



一、E-R图向关系模型的转换

由于E-R图是由实体、实体的属性和实体之间的联系组成，所以就要将实体、实体的属性和实体之间的联系转换为关系模式，这种转换遵循如下规则：

- 一个实体转换成一个关系模式。实体的属性就是关系的属性，实体的码就是关系的码。对于实体间的联系则有以下几种情况：
- 一个1:1的联系可以转换为一个独立的关系模式，也可以与任意一端对应的关系模式合并。如果转换成一个独立的关系模式，则与该联系相联的各实体的码以及联系本身的属性均转换为关系的属性，每个实体的码均是该关系的候选码，如果与某一端实体对应的关系模式合并，则需要在关系模式的属性中加入另一个关系模式的码和联系本身的属性。

- 一个1:n的联系可以转换为一个独立的关系模式，也可以与n端对应的关系模式合并。如果转换成一个独立的关系模式，则与该联系相联的各实体的码以及联系本身的属性均转换为关系的属性，而关系的码为n端关系的码。
- 一个m:n的联系可以转换为一个关系模式。与该实体相联的各实体的码以及联系本身的属性均转换为关系的属性，而关系的码为各实体码的组合。
- 三个或三个以上的实体间的一个多元联系可以转换为一个关系模式。与该多元联系相连的各实体码以及联系本身的属性均转换为关系属性，而关系的码为各实体码的组合。
- 具有相同的码的关系模式可以合并。

二、数据模型的优化

- 确定数据依赖；
- 对于各个关系模式之间的数据依赖进行极小化处理，消除冗余的联系；
- 按照数据依赖的理论对关系模式逐一进行分析，考察是否存在部分函数依赖、传递函数依赖，多值函数依赖等，确定各关系模式属于第几范式；
- 按照需求分析阶段得到的处理要求，分析这些模式对于这样的应用环境是否合适，确定对某些模式是否要进行合并或分解；
- 对关系模式进行必要的分解，提高数据操作的效率和存储空间的利用率。常用的分解方法是水平分解和垂直分解。

三、设计用户子模式

一般DBMS都提供了视图的概念，可利用这一功能为局部用户设计更合适的用户外模式，并考虑下列因素：

- 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**View**，以保证系统的安全性
- 简化用户对系统的操作

§ 5 物理结构设计

数据库的物理设计通过分为两步：

- ◆ 确定数据库的物理结构
- ◆ 对物理结构进行评价，评价的重点是时间和空间效率

一、数据库的物理设计的内容和方法

对于数据库的查询事务，需要得到如下信息：

- 查询的关系
- 查询条件所涉及到的属性
- 连接条件所涉及到的属性
- 查询的投影属性

对于数据更新事务，需要得到如下信息：

- 被更新的关系
- 每个关系上的更新操作条件所涉及到的属性
- 修改操作要改变的属性值

二、关系模式的存取方法选择

常用的存取方法：

- ◆ 索引方法
- ◆ B+树方法
- ◆ 聚簇（Cluster）方法
- ◆ HASH方法

三、确定数据库的存储结构

- 确定数据的存放位置
- 确定系统的配置

四、评价物理结构

- 评价物理数据库的方法完全依赖于所选用的**DBMS**，主要是从定量估算各种方案的存储空间、存取时间和维护代价入手，对估算结果进行权衡、比较，选择一个较优的合理的物理结构。

§ 6 数据库的实施和维护

- 数据的载入和应用程序的调试
- 数据库的试运行
- 数据库的运行和维护
 - ◆ 数据库的转储和恢复
 - ◆ 数据库的安全性、完整性控制
 - ◆ 数据库性能的监督、分析和改造
 - ◆ 数据库的重组与重构

第七章 事物管理--数据库恢复技术

本章要求:

- 1、掌握事务的概念及性质
- 2、掌握数据库恢复的基本技术和策略

本章内容:

- § 1 事务的基本概念
- § 2 故障的种类
- § 3 恢复的实现技术
- § 4 恢复策略
- § 5 具有检查点的恢复技术

请选择内容

返回

第七章 事物管理--数据库恢复技术

对数据库中存储的大量数据，有下面几个问题：

- 如何使数据资源只被相关人员合理使用？
- 如何恢复被破坏的数据？
- 如何协调多用户的工作来保证数据的一致性？
- 如何自动地发现用户的失误？

QUESTION
?

第七章 事物管理--数据库恢复技术

作为一个完善的DBMS，应该提供统一的数据保护功能来保证数据的安全可靠和正确有效！

数据保护也叫数据控制，主要包括：



本章首先讨论数据库恢复技术。

第七章 事物管理--数据库恢复技术

问题：系统软、硬件故障对系统数据造成破坏时，该如何处理？

例：银行转帐

设从帐号A中拨一笔款X到帐号B，正常的执行过程是：

☆ 查看帐号A上是否有足够的款数，即余额 $\geq X$ ？

若余额 $< X$ ，则给出提示信息，中止执行。

若余额 $\geq X$ ，则执行下面三步：

- ⌚ 在A中记上一笔支出，从余额中减去X；
- ⌚ 把值X传到B上；
- ⌚ 在B中记上一笔收入，在余额上加X，结束。



若在执行了第二步后突然断电或线路传输错误，则导致帐面不平

第七章 事物管理--数据库恢复技术

§ 1 事务的基本概念

1、事务 (transaction) 主要是更新操作

一个数据库操作序列，是数据库应用程序的基本逻辑单元。这些操作要么都做，要么都不做，是一个不可分割的执行单位。

事务标记: **BEGIN TRANSACTION** ← 事务开始

⋮
COMMIT 或 ROLLBACK

事务提交:
事务完成了其包含的所有活动，正常结束

事务回滚 (中止):
撤消已做的所有操作，回到事务开始时的状态

第七章 事物管理--数据库恢复技术

2、事务应具有的性质（P248）

（1）原子性（Atomicity）：事务执行时的不可分割性，即事务所包含的活动要么都做，要么都不做

若事务因故障而中止，则要设法消除该事务所产生的影响，使数据库恢复到该事务执行前的状态。

（2）一致性（Consistency）：事务对数据库的作用应使数据库从一个一致状态到另一个一致状态

例如：一个帐号的收支之差应等于余额。

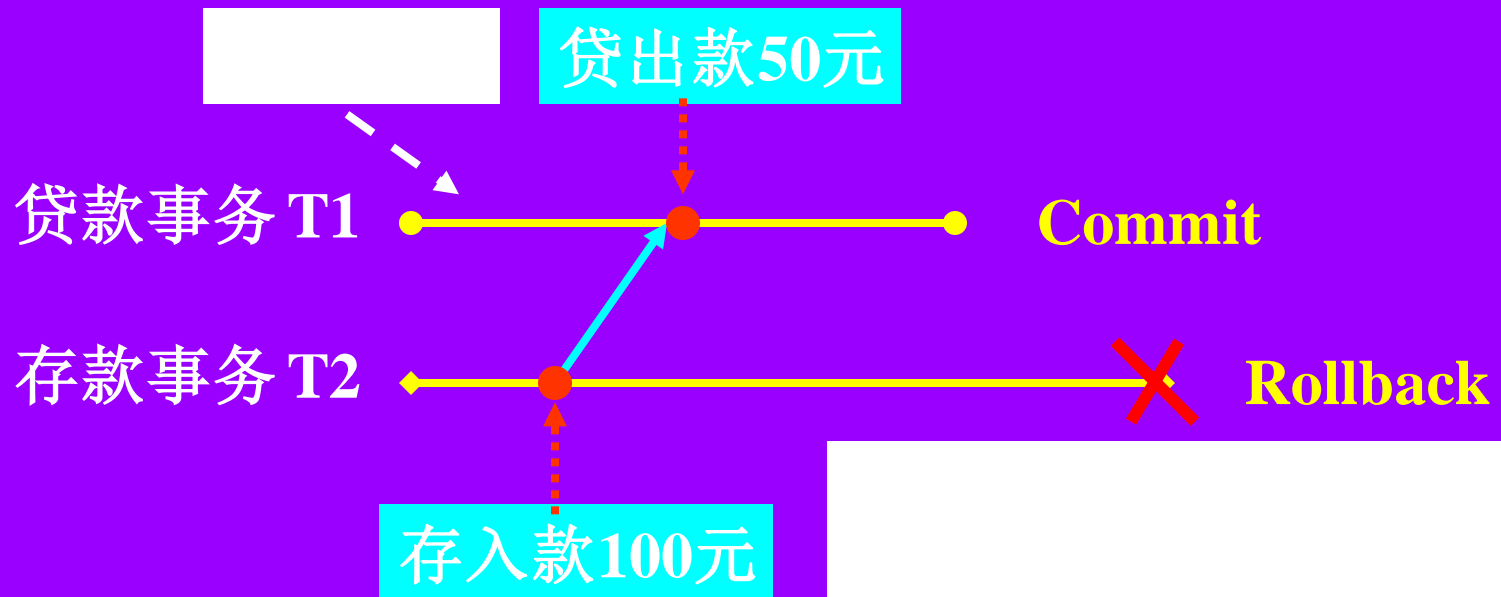
飞机订票系统，事务执行前后，座位与订出座位等信息必须一致。

第七章 事物管理--数据库恢复技术

(3) 隔离性 (Isolation) :

多事务并发执行，应象各事务独立执行一样，不能相互干扰。一个正在执行的事务其中间结果不能为其它事务所访问。

例如：有两个事务，在同一帐号上存款和贷款：



(4) 持久性 (Durability) : 一旦事务提交, 不论执行何种操作或发生何种故障, 都不应对该事务的执行结果有任何影响。

(5) 可串行性 (Serializability) : 并发控制正确性的标准用户程序在逻辑上是正确的, 它在串行执行时没有问题; 当多个事务并发执行时, 可以等价于一个串行执行序列。

3、事务管理任务

事务管理的任务就是要保证事务满足上述性质。使事务不具有上述性质的因素可能是:

- (1) 事务在运行过程中被强行终止;
- (2) 多个事务并行运行时, 不同事务的操作交叉执行

第七章 事物管理--数据库恢复技术

因此事物管理又分为两个方面：

恢复：保证事务在故障时满足上述性质的技术。

并发控制：保证事务在并发执行时满足上述性质的技术。

§ 2 故障的种类

数据库系统中可能发生各种各样的故障，分为以下几类。

1、事务内部的故障

☆ 可以通过事务程序本身发现并处理的故障

如P249银行转帐事务程序在余额小于转帐额时的情形

⌚ 非预期的故障（不能由应用程序处理）

如运算溢出、被零除、发生死锁时被选中撤消等

第七章 事物管理--数据库恢复技术

通常，我们所说的事务故障仅指非预期故障。事务故障意味着事务没有达到预期的终点（**COMMIT**或者显式的**ROLLBACK**），因此数据库可能处于不一致状态，恢复程序应在不影响其他事务的情况下，撤消故障事务的所有修改，使得故障事务就象没有运行一样。这类操作称为事务撤消（**UNDO**）。

2、系统范围内的故障：软故障

造成系统停止的任何事件，如**CPU**故障、操作系统故障、程序代码错误、断电等，使得系统必须重新启动。

第七章 事物管理--数据库恢复技术

系统故障发生时，可能使数据库处于不一致状态：

一方面，有些非正常终止事务的结果可能已写入数据库，在系统下次启动时，恢复程序必须回滚这些非正常终止的事务，撤消这些事务对数据库的影响。

另一方面，有些已完成事务的结果可能部分或全部留在缓冲区，而尚未写回磁盘上的数据库中。在系统下次启动时，恢复程序必须重做（**REDO**）所有已提交的事务，将数据库真正恢复到一致状态。

3、介质故障：硬故障

如磁盘损坏、磁头碰撞、强磁场干扰等。

这类故障发生概率很小，但破坏性极大，将破坏部分甚至整个数据库的内容，并影响使用相应数据的所有事务

4、计算机病毒

第七章 事物管理--数据库恢复技术

§ 3 恢复的实现技术

数据库故障对数据库的影响

- * 数据库本身被破坏;
- * 数据库没有破坏,正在运行的事务被非正常终止,可能造成数据库数据不正确。

数据库恢复的基本原理-----冗余

数据库任何一部分被破坏或数据不正确时,可根据存储在系统别处的数据来重建。

数据库恢复的机制(两步)

- * 建立冗余数据
常用技术: 数据转储、登记日志文件
- * 利用冗余数据实施数据库恢复

第七章 事物管理--数据库恢复技术

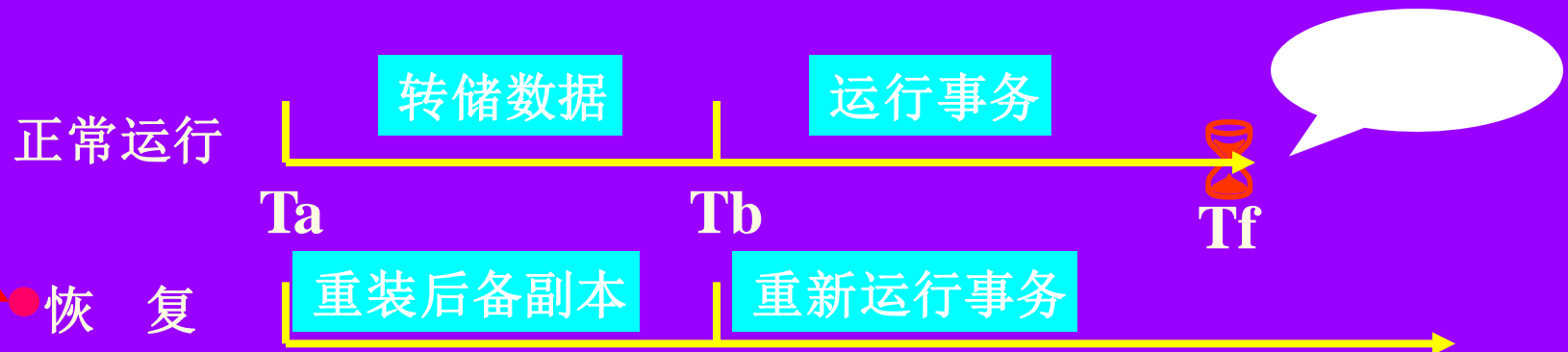
1、数据转储与恢复

转储：**DBA**定期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。

（这些备用的数据称为**后备副本**或**后援副本**）

● 恢复：当数据库被破坏后可将**后备副本**重新装入，并重新运行转储以后的所有更新事务。

例：**Ta**时刻系统停止运行事务开始转储，**Tb**时刻转储完毕重新开始运行事务，**Tf**时刻发生故障。



第七章 事物管理--数据库恢复技术

转储的状态

静态转储：转储期间  对数据库进行操作

特点：静态转储得到的一定是一个数据一致性的副本。因为转储必须等用户事务全部结束才能进行，而且新的事务必须等待转储完毕才能开始执行。但数据库的可用性被降低。

动态转储：转储期间  对数据库进行操作

特点：转储和用户事务可并发执行，即不必等待正在运行的事务结束，也不影响新事务的运行。但转储的数据可能已过时。

为此，必须建立日志文件，记录转储期间对数据库的更新活动。这样，后援副本加日志文件就能把数据库恢复到某个时刻的一致性状态。

第七章 事物管理--数据库恢复技术

转储方式

海量转储：每次转储全部数据库（一般每周一次）

增量转储：只转储上次转储后更新过的数据
（一般每天一次）

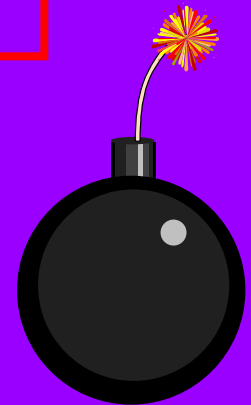
转储方法 { 动态海量 动态增量
 静态海量 静态增量

转储的缺点

费时

在转储后和故障点之间的数据更新不能恢复
动态转储时转储的数据可能已过时

注意：对大中型数据库系统来说，
转储是非常重要的！



第七章 事物管理--数据库恢复技术

2、日志文件和恢复

日志（log）：用来记录对数据库的更新操作的文件。

{ 动态转储方式必须建立日志文件
静态转储方式最好建立日志文件

日志文件的格式和内容

* 以记录为单位的日志文件

系统把 { 事务开始（**BEGIN TRANSACTION**）
事务提交（**COMMIT**）或
事务撤消（**ROLLBACK**）
对数据库的插入、删除、修改等
每一个操作作为一条记录存放到日志文件中

第七章 事物管理--数据库恢复技术

每条日志记录的主要内容

事务标识（哪个事务）	
操作类型（插删改）	
操作对象（哪条记录）	
更新前数据的旧值	← 对插入此项为空
更新后数据的新值	← 对删除此项为空

* 以数据块为单位的日志文件

将事务标识及更新前后的数据块均放在日志文件中。

日志文件的作用

静态转储：数据库毁坏后，重装后援副本，根据日志文件，重做已完成的事务，并撤消未完成的事务。

动态转储：用后援副本和日志文件综合起来恢复数据库

第七章 事物管理--数据库恢复技术

登记日志文件

原则：严格按并发事务执行的时间次序登记；
先写日志文件，后写数据库。

日志超前写规则

写数据库和写日志文件是两个不同的操作，在这两个操作之间有可能发生故障，若先写数据库数据，再写日志的话，万一在写日志前发生故障，则这次的数据库修改未登记，从而不能恢复。若写日志后发生故障而未修改数据库，则事务一定未完成，在恢复时会执行撤消处理。

如：欲将数据库中某记录字段的值由5改为8，登记日志文件后发生故障，则字段值仍为5，日志中不会登记该事务的COMMIT或ROLLBACK记录，事务未完成，恢复时对该操作做撤消处理，将字段值改为该修改操作的旧值5，数据库内容不变。

第七章 事物管理--数据库恢复技术

§ 4 恢复策略

发生故障时，利用数据库后援副本和日志文件可以将数据库恢复到某个一致性状态，但不同故障的恢复策略和方法是不一样的。

利用日志文件进行恢复

基本策略：

对于尚未提交的事务，执行撤消处理（**UNDO**）

对于已经提交的事务，执行重做处理（**REDO**）

基本方法：

扫描日志文件，确定所有已开始但尚未提交的事务（对它们需**UNDO**），再确定所有已提交的事务（对它们需**REDO**）

第七章 事物管理--数据库恢复技术

UNDO处理：若事务提交前出现异常，则对已执行的操作进行撤消处理，使数据库恢复到该事务开始前的状态。
具体做法是：反向扫描日志文件，对每个需UNDO的事务的更新操作执行反操作。即对已插入的记录执行删除，对已删除的记录重新插入，对已修改的记录用旧值代替新值。

UNDO处理是维护事务的原子性所必须的

REDO处理：重做已提交事务的操作。

具体做法是：正向扫描日志文件重新执行登记的操作

第七章 事物管理--数据库恢复技术

有些事务虽已发出COMMIT操作，但更新的结果可能只是写到缓冲区而未能写入磁盘，或磁盘上数据库被破坏，因此需要REDO处理。

例如：事务T1在学生表S上执行下面三个操作：

```
INSERT INTO S
VALUES ( 'S4', 'D', 'CS', 19) ;
DELETE FROM S
WHERE S#='S1';
UPDATE S
SET SD='CS'
WHERE S#='S2';
```

第七章

事物管理--数据库恢复技术

S1	A	CS	20
S2	B	CI	21
S3	C	MA	19

事务T1执行前的S

事务T1开始

S1	A	CS	20
S2	B	CI	21
S3	C	MA	19
S4	D	CS	19

INSERT INTO S
VALUES ('S4','D',
 'CS',19);

T1
在S中插入键
为S4的记录

S4	D	CS	19
----	---	----	----

S2	B	CI	21
S3	C	MA	19
S4	D	CS	19

DELETE FROM S
WHERE S#='S1';

T1
在S中删除键
为S1的记录

S1	A	CS	20
----	---	----	----

S2	B	CS	21
S3	C	MA	19
S4	D	CS	19

UPDATE S
SET SD='CS'
WHERE S#='S2';

T1
在S中修改键
为S2的记录

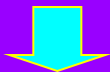
S2	B	CI	21
----	---	----	----

S2	B	CS	21
----	---	----	----

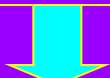
第七章

事物管理--数据库恢复技术

S1	A	CS	20
S2	B	CI	21
S3	C	MA	19



S1	A	CS	20
S2	B	CI	21
S3	C	MA	19
S4	D	CS	19



S2	B	CI	21
S3	C	MA	19
S4	D	CS	19



S2	B	CS	21
S3	C	MA	19
S4	D	CS	19

REDO处理



事务T1开始

T1

在S中插入键
为S4的记录

S4 D CS 19

T1

在S中删除键
为S1的记录

S1 A CS 20

T1

在S中修改键
为S2的记录

S2 B CI 21

S2 B CS 21

若数据库
中的状态是:

S1	A	CS	20
S2	B	CI	21
S3	C	MA	19
S4	D	CS	19

S2	B	CI	21
S3	C	MA	19
S4	D	CS	19

S2	B	CS	21
S3	C	MA	19
S4	D	CS	19

REDO处理

事务T1开始

T1
在S中插入键
为S4的记录

S4 D CS 19

T1
在S中删除键
为S1的记录

S1 A CS 20

T1
在S中修改键
为S2的记录

S2 B CI 21

S2 B CS 21

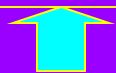
第七章

事物管理--数据库恢复技术

S1	A	CS	20
S2	B	CI	21
S3	C	MA	19



S1	A	CS	20
S2	B	CI	21
S3	C	MA	19
S4	D	CS	19



S2	B	CI	21
S3	C	MA	19
S4	D	CS	19



S2	B	CS	21
S3	C	MA	19
S4	D	CS	19

UNDO处理

事务T1开始

T1

在S中插入键
为S4的记录

S4 D CS 19

T1

在S中删除键
为S1的记录

S1 A CS 20

T1

在S中修改键
为S2的记录

S2 B CI 21

S2 B CS 21

数据库系统

11/20/2010

369

第七章 事物管理--数据库恢复技术

1、事务故障的恢复

事务故障是指事务被非正常终止，应根据日志文件对未完成事务做UNDO处理，步骤如下：

- (1) 反向扫描日志文件，查找未完成事务的更新操作；
- (2) 对该事务的更新操作执行逆操作；
- (3) 继续反向扫描日志文件，对遇到的更新操作做同样处理；
- (4) 当遇到某事务的开始标记时，停止对该事务的处理。
- (5) 重复上述过程，直到所有未完成事务全部UNDO完毕。

第七章 事物管理--数据库恢复技术

2、系统故障的恢复

系统故障造成数据库不一致的原因，一是未完成事务对数据库的更新已写入数据库，二是已提交事务的结果在故障发生前留在缓冲区没来得及写入数据库。恢复操作是撤消未完成事务，重做已完成事务。步骤如下：

- (1) 正向扫描日志文件，找出在故障发生前已提交的事务，将它们记入重做（**REDO**）队列，同时找出故障发生前尚未完成的事务，将它们记入撤消（**UNDO**）队列。
- (2) 反向扫描日志文件，对**UNDO**队列的每个事务执行逆操作，即做撤消处理。
- (3) 正向扫描日志文件，对**REDO**队列中的每个事务重新执行日志文件登记的操作。

第七章 事物管理--数据库恢复技术

3、介质故障的恢复

介质故障发生后，磁盘上的数据文件和日志文件均被破坏，恢复的方法是重装数据库和日志文件，然后重做自转储以来已完成的事务。步骤如下：

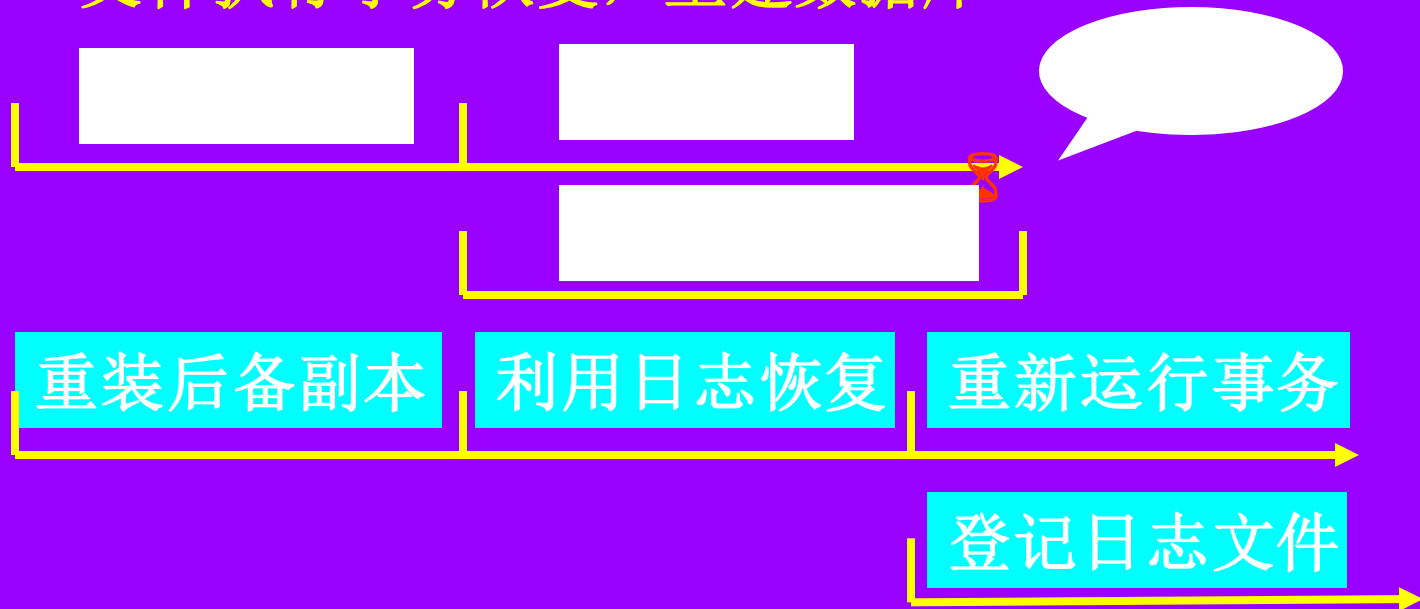
- (1) 装入最近转储的数据库后援副本，若是动态转储，则还应装入转储期间的日志文件，将数据库恢复到一致性状态。
- (2) 装入转储结束后的日志副本，重做已完成的事务。

系统故障与事务故障的恢复由系统自动完成，对用户透明，介质故障的恢复，需要DBA重装数据库和日志文件副本，然后执行相应的恢复命令。不论那种恢复，一般都要扫描整个日志文件。

第七章 事物管理--数据库恢复技术

恢复方式总结:

- ☆ 当数据库被破坏时,要重装后备副本,然后利用日志文件执行事务恢复,重建数据库



- ✧ 数据库本身未被破坏,但有些内容可能不正确,则可只利用日志文件恢复,使数据库回到某一正确状态

第七章 事物管理--数据库恢复技术

§ 5 具有检查点的恢复技术

利用日志文件恢复数据库，一般要扫描整个日志文件，日志是个流水帐，往往很长，这样做具有两个问题：

- * 搜索整个日志文件将耗费大量的时间；
- * 许多已提交事务的更新结果实际上已写入数据库中，重新做这些事务只会浪费大量的时间。

因此，确定哪些事务需**REDO**，哪些不需**REDO**，就很有意义。在日志文件中设置检查点记录

DBMS周期性地日志中记录一个检查点：将当前正在执行（尚未提交）的所有事务记录于一个记录中——检查点记录。具体工作为：

第七章 事物管理--数据库恢复技术

- ① 将内存中所有日志记录写入磁盘；
- ② 在磁盘日志文件中写入一个检查点记录；
- ③ 将内存中所有数据库记录写入磁盘数据库中；
- ④ 把检查点记录在日志文件中的地址写入一个重新开始文件中。

检查点记录的内容包括：

- ① 建立检查点时所有正在执行的事务清单；
- ② 这些事务中最近的一个日志记录地址。

参见P257图7.3。

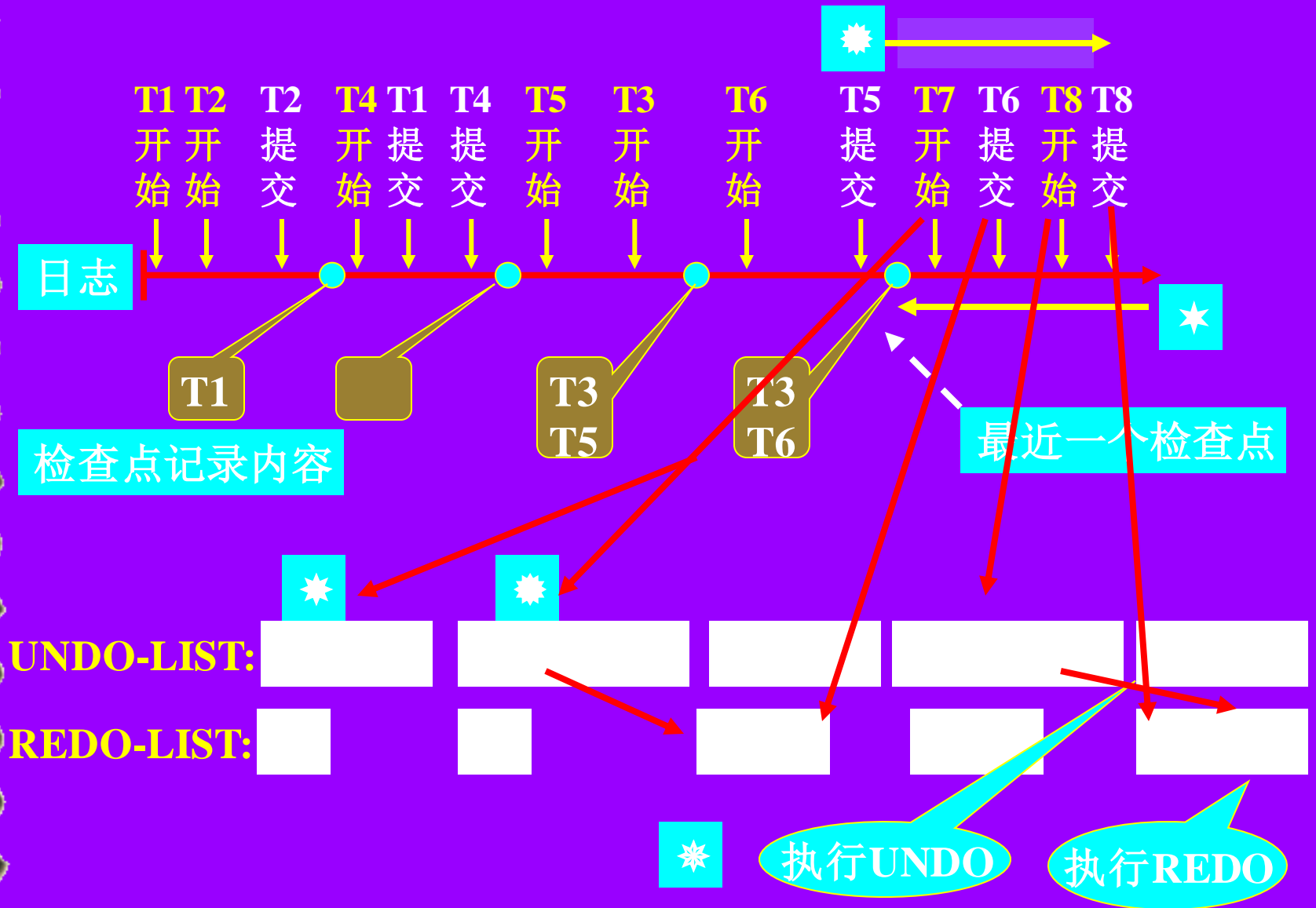
在检查点之前已提交的事务对数据库的修改在检查点之前或检查点建立时已记入磁盘，只要数据库未被破坏，不需要对这些事务执行重做（**REDO**）。

第七章 事物管理--数据库恢复技术

具有检查点的恢复算法

- ★ 根据重新开始文件中最后一个检查点记录的地址，在日志文件中找到最近的一个检查点记录；
- ★ 设置两个队列，将检查点中的所有事务放入 **UNDO-LIST**，并令 **REDO-LIST** 暂为空集；
UNDO-LIST：需要 **UNDO** 操作的事务集合；
REDO-LIST：需要 **REDO** 操作的事务集合；
- ★ 从该检查点开始扫描日志文件到文件结束为止：
凡遇有 **begin_transaction** 的事务放入 **UNDO-LIST**；
凡遇有 **commit** 的事务，将它从 **UNDO-LIST** 移入 **REDO-LIST**；
- ★ 对 **UNDO-LIST** 中的事务执行 **UNDO** 操作
对 **REDO-LIST** 中的事务执行 **REDO** 操作

第七章 事物管理--数据库恢复技术



第七章 事物管理--数据库恢复技术

§ 6 数据库镜像

前面已介绍，当数据库系统发生故障时，可利用日志文件进行数据库恢复，但前提是日志文件必须完好。然而当发生介质故障时，往往不仅数据库被摧毁，日志文件也难逃恶运，此时恢复操作就无法实施。这在银行数据库等系统中是绝对不允许的。

解决办法：

1、数据库镜像：将整个数据库或其中的关键数据同时存放在两个分离的物理磁盘上。每当主数据库更新时，**DBMS**自动把更新后的数据复制到另一个磁盘上，从而自动保证主数据库与镜像数据库的一致性。

但镜像的内容可选，如只是事务日志，或服务器上所有内容，等等。

第七章 事物管理--数据库恢复技术

数据库镜像的优缺点：

优点：可提高数据库的可用性。

- * 在介质故障时，不需关闭系统和重装后援副本，保证“不间断”地恢复；
- * 便于并发操作，当主数据库的某个对象被加排它锁时，其它应用可以读镜像数据库。

缺点：

- * 由于频繁地复制数据，会降低系统的运行效率；
- * 使用更多的磁盘设备。

2、磁盘双工：用两个不同控制器控制的磁盘存放同一内

3、双机热备份：用两个控制器同时实现

第八章 事务管理--并发控制

本章要求：

- 1、了解并发操作可能产生的数据不一致性
- 2、掌握并发控制的技术：封锁机制、三级封锁协议、活锁的避免、死锁的预防、诊断及解除
- 3、掌握并发调度的正确性标准和技术（可串行性、两段锁协议）

第八章 事务管理--并发控制

§ 1 并发控制概述

在多用户数据库系统中，当多个用户并发存取数据库时就会产生多个事务同时存取同一数据的情形。若不加控制，可能会存取和存储不正确的数据，造成数据库的不一致性。

在并发操作情况下，对事务的操作序列的调度是随机的，考虑飞机订票系统，若按下面的序列调度：

第八章 事务管理--并发控制

考虑飞机订票系统中的一个活动序列：

- 甲售票点读出某航班的机票余额A，设 $A=16$ ，
- ✕ 乙售票点读出同一航班的机票余额A，也为16，
- ✕ 甲售票点卖出一张机票，修改余额 $A \leftarrow A-1$ ，
A变为15，把A写回数据库
- ✕ 乙售票点也卖出一张机票，修改余额 $A \leftarrow A-1$ ，
A也为15，把A写回数据库。

卖出两张机票，而余额只减少1。错误！**X**

这种情况就造成数据库的不一致性，这种不一致性是由并发操作引起的。

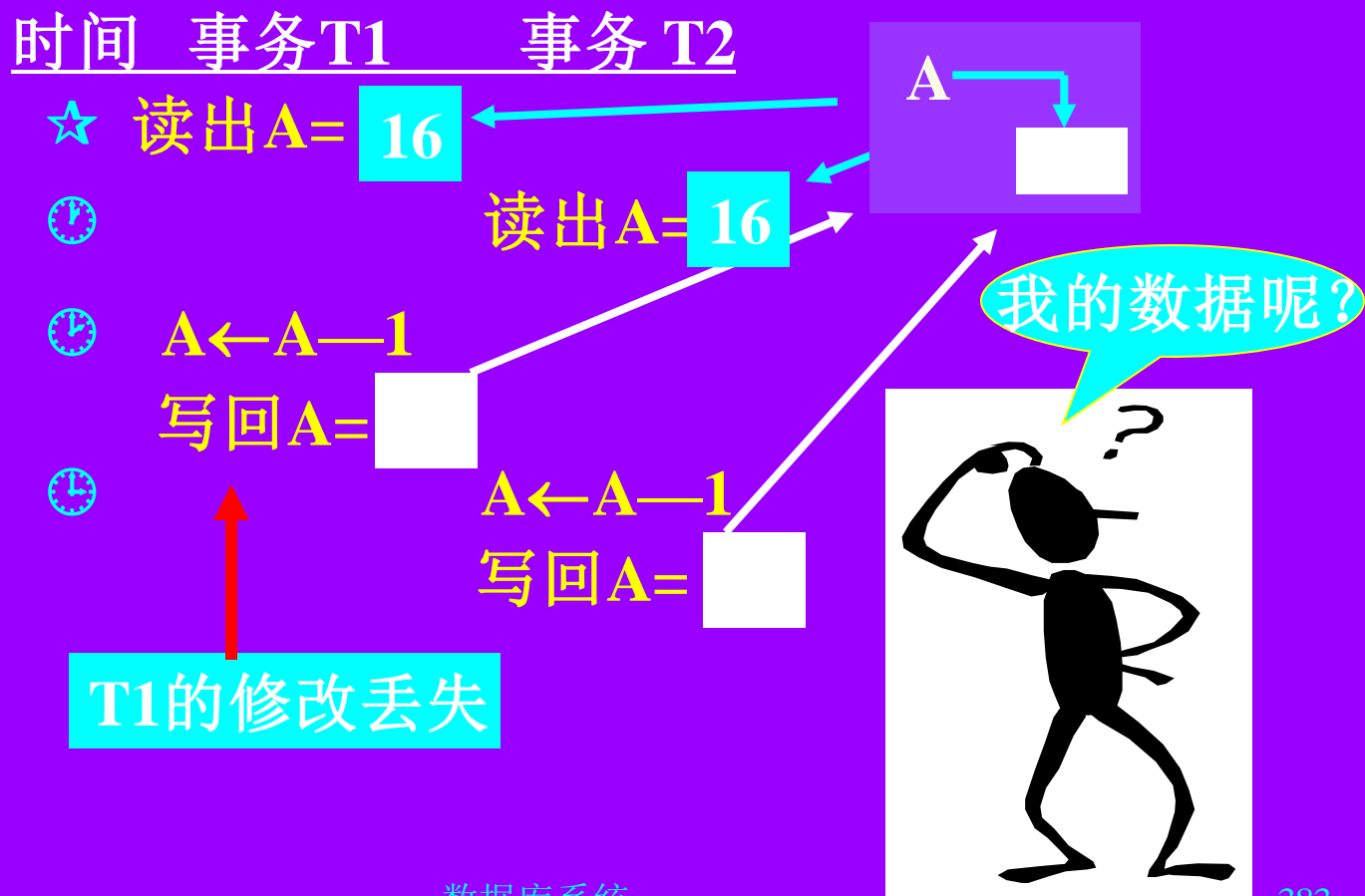
并发操作带来的数据不一致性包括三类

第八章 事务管理--并发控制

一、并发操作可能造成的一致性

1、丢失修改:

两事务读出同一数据并修改, 先写回的数据修改丢失



第八章 事务管理--并发控制

2、不可重复读

事务T1读取某一数据，事务T2读取并修改了同一数据；事务T1为了对读取值进行校对再读此数据，得到了不同的结果。

时间	事务T1	事务 T2
☆	读出A=50, B=100 求和 =150	
🕒		读出B=100 计算 $B \leftarrow B \times 2$, 写回B
🕒	读出A=50, B= <input type="text"/> 求和 = <input type="text"/>	

T1读出B的值与原来的不符，验算结果不对

第八章 事务管理--并发控制

有三种情况可造成不可重复读

(1) 事务T1读取某一数据后，事务T2对其做了修改，事务T1再次读取该数据时，发现与前次不同；

(2) 事务T1按一定条件读取了某些数据记录后，事务T2删除了其中的部分记录，事务T1再次按相同条件读取记录时，发现有些记录不存在；

(3) 事务T1按一定条件读取了某些数据记录后，事务T2插入了一些记录，事务T1再次按相同条件读取记录时，发现多了一些记录。

第八章 事务管理--并发控制

3、读出“脏”数据

事务T1修改某一数据，事务T2读取同一数据；
事务T1由于某种原因被撤消，则T2读到的就是“脏”数据。

时间	事务T1	事务T2
----	------	------

☆ 读出C=100

计算 $C \leftarrow C \times 2$ ，写回C



⌚ ROLLBACK

C恢复为100

读出



“脏”数据

T2读出的数据无效

第八章 事务管理--并发控制

产生上述三类不一致性的主要原因就是并发操作破坏了事务的隔离性。并发控制就是要用正确的方式调度并发操作，使某个事务的执行不受其它事务的干扰。

必须对并发操作进行控制

并发控制的技术是封锁，即事务在修改某个对象前，先锁住该对象，不允许其它事务读取或修改该对象，修改完毕或事务完成后再将锁打开。

第八章 事务管理--并发控制

§ 2 封锁 (Locking)

封锁的类型:

排它锁 (**Exclusive Lock**, 简称**X锁**, 又称**互斥锁**):

若事务**T**对数据对象**R**加上**X锁**, 则只允许**T**读、写**R**

,

禁止其它事务对**R**加任何锁, 相应地其它事务就无法

共享读对象**R**
共享锁 (**Shared Lock**, 简称**S锁**):

若事务**T**对数据对象**R**加上**S锁**, 则**T**可以读**R**, 但不可以写**R**, 且其它事务可以对**R**加**S锁**、但禁止加**X锁**。这保证了事务**T**在释放**R**的**S锁**之前, 其它事务只可以读**R**, 不可以修改**R**。

第八章 事务管理--并发控制

§ 3 封锁协议

1、封锁协议（Locking Protocol）

在运用X锁和S锁对数据对象加锁时，对何时申请X锁或S锁、持锁时间、何时释放等的一些约定。

2、三级封锁协议

基本方法：

★ 事务在读数据对象R时先上S锁，封锁后才能读R，否则需等待；

🕒 事务在写数据对象R时先上X锁，封锁后才能写R，否则需等待；

🕒 何时释放锁（Unlock）？

第八章 事务管理--并发控制

根据上锁的类型和释放锁的时机，分为三种情况。

1级封锁协议：

事务T在修改数据对象R之前必须先对其加X锁，直到事务结束才释放。

2级封锁协议：

1级封锁协议 + 事务T在读取数据对象R之前必须先对其加S锁，读完即释放S锁。

3级封锁协议：

1级封锁协议 + 事务T在读取数据对象R之前必须先对其加S锁，S锁也是直到事务结束才释放。

第八章 事务管理--并发控制

分析:

1级封锁协议:

事务T在修改数据对象R之前必须先对其加X锁,直到事务结束才释放。

作用:

□ 防止丢失修改

□ 保证事务是可恢复的

时间	事务T1	事务T2
----	------	------

☆ 请求X锁

对A加X锁

读出A=16



$A \leftarrow A - 1$

写回A=15

Commit

释放X锁



请求X锁
等待

⋮

对A加X锁
读出A=15

$A \leftarrow A - 1$

写回A=14

Commit

释放X锁

没有丢失修改

第八章 事务管理--并发控制

分析（续）：

1级封锁协议：

事务T在修改数据对象R之前必须先对其加X锁，直到事务结束才释放。

时间	事务T1	事务T2
----	------	------

★ 请求X锁

对C加X锁

读出C=100

计算 $C \leftarrow C \times 2$

写回C



ROLLBACK

（C恢复为100）

释放X锁

读出

仍然读出
“脏”数据

T1已对C加X锁，
为什么T2仍能读取C？

因为T2在读取C
前未对C申请任何锁。

第八章 事务管理--并发控制

分析（续）：

1级封锁协议：

事务T在修改数据对象R之前必须先对其加X锁，直到事务结束才释放。

时间	事务T1	事务T2
☆	读出A=50, B=100 求和 =150	
🕒		请求X锁 对B加X锁 读出B=100 计算 $B \leftarrow B \times 2$, 写回B Commit 释放X锁
🕒	读出A=50, B= <div></div> 求和 = <div></div>	



原因：T1在读取A、B时未申请任何锁。

仍然是不可重复读

第八章 事务管理--并发控制

分析:

2级封锁协议:

1级封锁协议 + 事务T在读取数据对象R之前必须先对其加S锁, 读完即释放S锁。

时间	事务T1	事务T2
☆	请求X锁	
	对C加X锁	
	读出C=100	
	计算 $C \leftarrow C \times 2$, 写回C	
🕒		请求S锁
		等待
		⋮
🕒	ROLLBACK (C恢复为100)	
	释放X锁	
🕒		对C加S锁
		读出C=100
		释放S锁



作用:

□ 防止丢失修改

□ 保证事务是可恢复的

□ 防止读“脏”数据

数据库系统

第八章 事务管理--并发控制

分析（续）：

时间	事务T1	事务T2
----	------	------

☆ 对A加S锁
对B加S锁
读出A=50, B=100
求和=150
释放A、B上的S锁



⌚ 对A加S锁
对B加S锁
读出A=50, B=
求和=
释放A、B上的S锁

请求X锁
对B加X锁
读出B=100
计算 $B \leftarrow B \times 2$, 写回B
Commit
释放X锁

2级封锁协议：

1级封锁协议 +
事务T在读取数据对象R之前必须先对其加S锁，读完即释放S锁。

仍然是
不可重复读

第八章 事务管理--并发控制

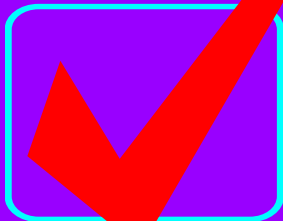
分析:

时间	事务T1	事务T2
----	------	------

☆ 对A加S锁
对B加S锁
读出A=50, B=100
求和 =150



读出A=50, B=100
求和 =150
Commit
释放A、B上的S锁



对B请求X锁
等待

⋮

对B加X锁
读出B=100
计算 $B \leftarrow B \times 2$,
Commit
释放X锁

数据库系统

3级封锁协议:

1级封锁协议+事务T在读取数据对象R之前必须先对其加S锁, S锁也是直到事务结束才释放。

作用:

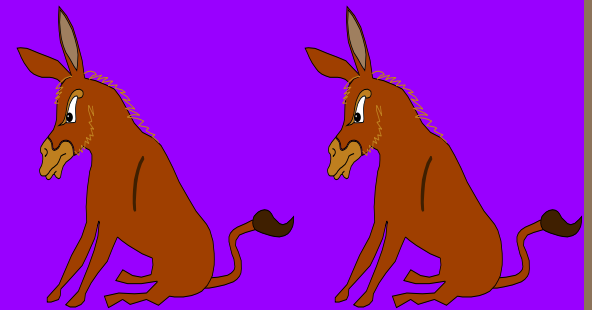
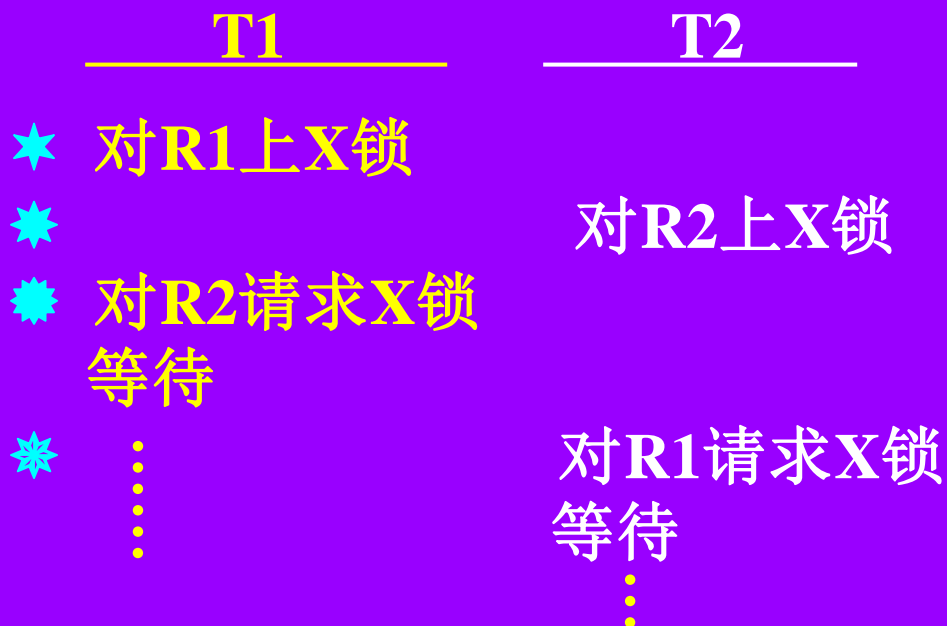
- ☐ 防止丢失修改
- ☐ 保证事务是可恢复的
- ☐ 防止读“脏”数据
- ☐ 保证可重复读

第八章 事务管理--并发控制

三级封锁协议的主要区别在于什么操作需要申请封锁，以及何时释放封锁。参见P269的表8.1。

§ 4 活锁和死锁

使用封锁机制，得不到锁的事务就要等待，这可能出现下述局面：

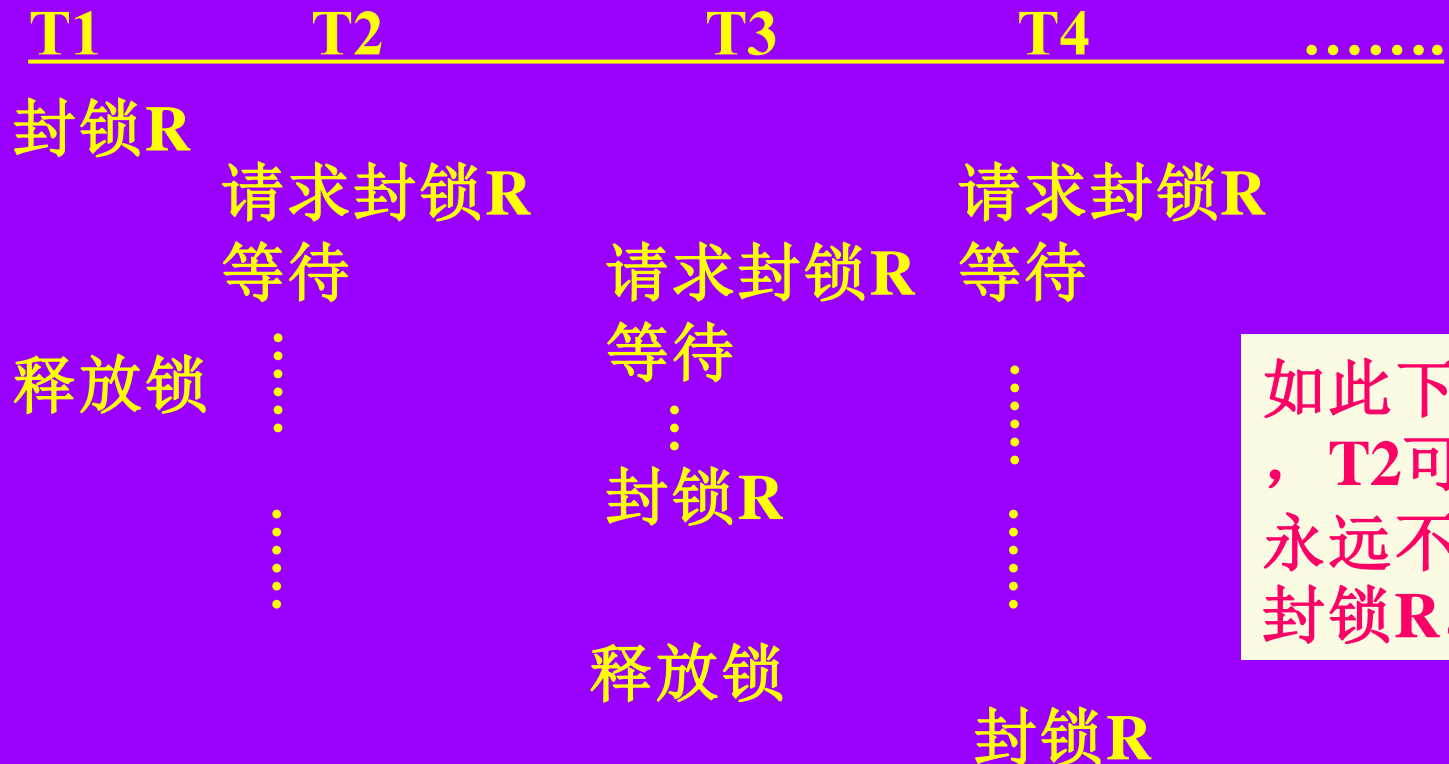


T1和T2将
永远等待下去

第八章 事务管理--并发控制

1、活锁 (Livelock) :

数据对象不断处于上锁、开锁的交替状态, 某个事务有可能为该对象上锁, 但始终没有得到上锁机会而永久等待下去的情形。例如:



如此下去
， T2可能
永远不能
封锁R。

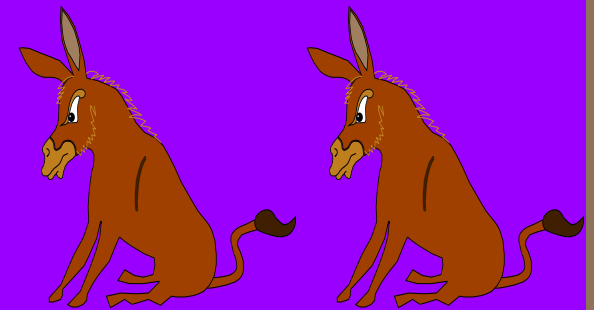
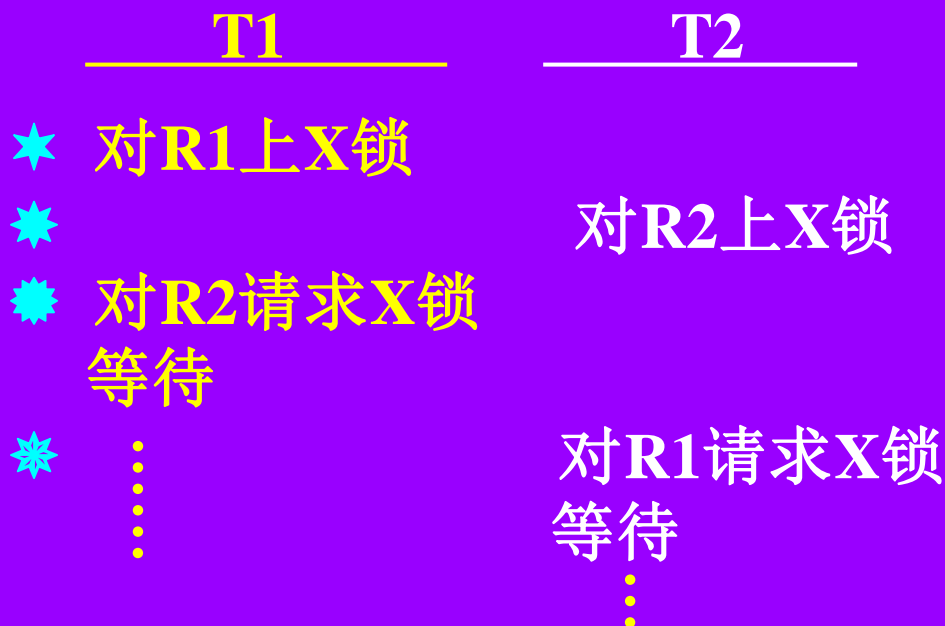
第八章 事务管理--并发控制

避免活锁：

如采用先来先服务策略。

2、死锁（Deadlock）：

多个事务因封锁冲突（竞争资源）而永远等待下去的情形。如：



T1和T2将
永远等待下去

第八章 事务管理--并发控制



(1) 一次封锁法

每个事务必须将所要求的数据对象全部上锁后才能执行读写操作，否则释放占用的资源。

存在的问题

- 使数据的上锁时间增长，降低了系统的并发度。
- 很难确定事务执行期间需封锁的数据对象。有些一开始不需要封锁的对象，随着数据库数据的变化，可能变成封锁对象，为避免此种情况发生，只能扩大封锁范围。

第八章 事务管理--并发控制

(2) 顺序封锁法

对所有数据对象规定一个封锁顺序，所有事务均按这个顺序实行封锁。

存在的问题：

- 很难维护数据对象的封锁顺序，因为数据对象很多并在不断地增加、减少。
- 很难确定事务需封锁那些对象，从而很难按规定的顺序封锁。

上述两种方法虽然都可以有效地预防死锁，但都存在一些问题，因此真正实施起来并不方便。所以预防死锁的策略不很适合数据库的特点，**DBMS**普遍采用诊断死锁并解除的方法。

第八章 事务管理--并发控制

(3) 超时法

当一个事务的等待时间超过了规定的时限，就认为发生了死锁。

存在的问题：

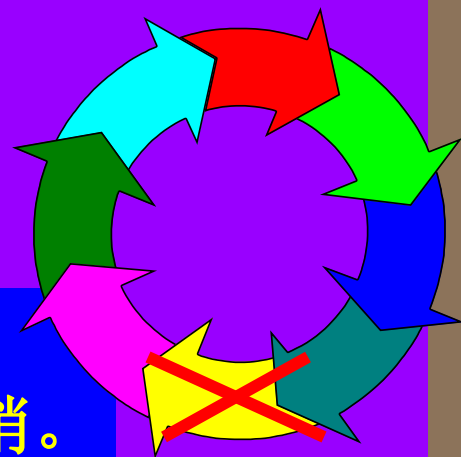
➤ 时限规定的太短，可能误判死锁，规定的太长，又不能及时发现死锁。因此很难确定一个合理的时限。

(4) 等待图法

用一个有向图表示事务等待的情况。图中节点表示事务，边表示事务间的等待关系。并发控制子系统定时检查此图，若发现有回路，则产生死锁。

发生死锁时，解除死锁的方法：

选择一个处理代价最小的事务，将其撤消。



第八章 事务管理--并发控制

§ 5 并发调度的可串行性

多事务并发执行，对并发操作的调度是随机的，如何保证正确性？

1、调度（Schedule）：

若干个事务的操作构成的序列。

2、串行与并发

对调度S中的两个事务 T_i 和 T_j ，若S中 T_i 的操作都在 T_j 的操作之前（或反之），则称 T_i 、 T_j 是串行执行的；否则称作是并发执行的。

若调度S中的所有事务都是串行执行的，则称S是串行的（Serial）。

第八章 事务管理--并发控制

4、一点说明：

对若干个事务，不同的并发调度策略其最终的执行结果不一定完全相同。但只要它们的调度是可串行化的，则都是正确调度。

如：事务T1：读B； $A \leftarrow B+1$ ；写回A。
事务T2：读A； $B \leftarrow A+1$ ；写回B。

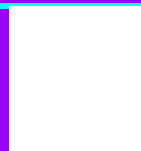
假设T1、T2执行前
 $A=2, B=2$

T1	T2
读B	
$A \leftarrow B+1$	
写A	
	读A
	$B \leftarrow A+1$
	写B

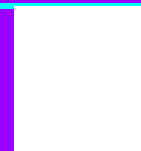
结果



T1	T2
	读A
	$B \leftarrow A+1$
	写B
读B	
$A \leftarrow B+1$	
写A	



T1	T2
读B	
	读A
$A \leftarrow B+1$	
写A	
	$B \leftarrow A+1$
	写B



第八章 事务管理--并发控制

分析右边的调度：

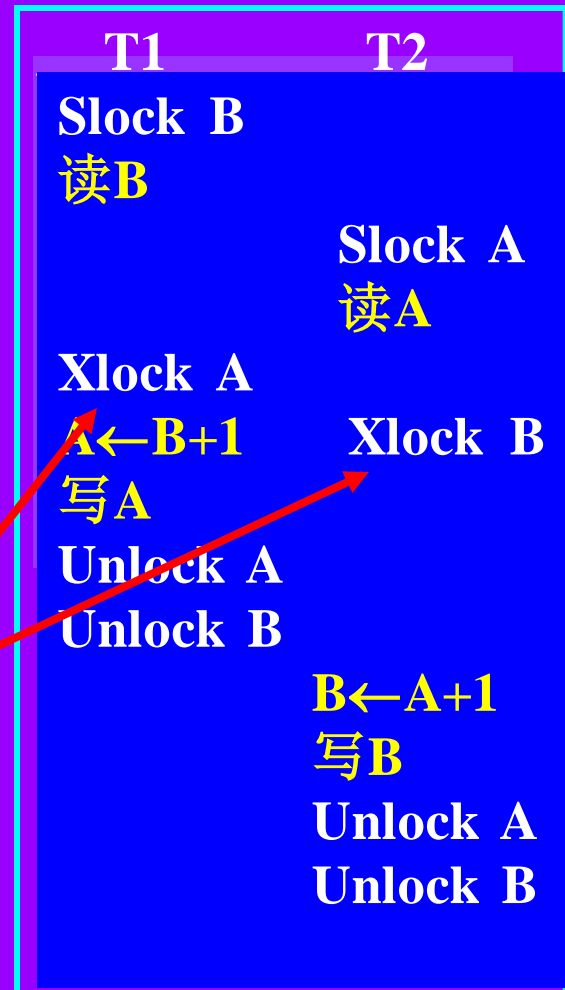
若采用2级封锁协议，
则可能得到这样的调度

仍是不可串行化的调度

若采用3级封锁协议，
则可能得到这样的调度

永久等待，造成死锁

因此，三级封锁协议不能
保证得到可串行化的调度
，也会造成死锁。



第八章 事务管理--并发控制

§ 6 两段锁协议

1、两段封锁协议（也称两相上锁协议）

（2-Phase-Locking Protocol, 简写2PL）

（1）在对任何数据进行读、写操作之前，事务首先要申请并获得对该数据的封锁（读时S锁，写时X锁）；

（2）在释放一个封锁之后，事务不再申请和获得新的封锁。

例：

Lock A Lock B Lock C Unlock B Unlock A Unlock C ✓

Lock A Lock B Unlock B Lock C Unlock A Unlock C ✗

第八章 事务管理--并发控制

含义：事务封锁分两个阶段

第一阶段是扩展阶段（**Growing Phase**），只进行上锁

第二阶段是收缩阶段（**Shrinking Phase**），只进行解锁

2、2PL的正确性

定理：若所有事务都遵守两段封锁协议，则对这些事务的任何并发调度策略都是可串行化的。

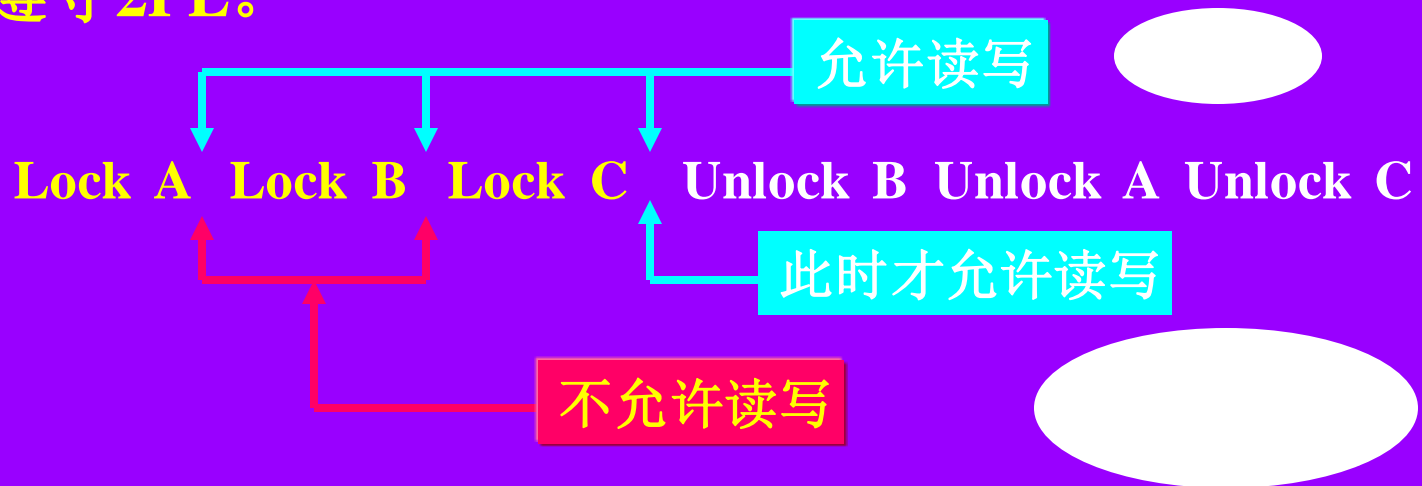
3、说明

◆ **2PL**是并发控制正确性的充分条件，但不是必要条件。即若所有事务都遵守**2PL**，则这些事务的任何并发调度都是可串行化的，反之，一个并发调度是可串行化的，不一定所有事务都遵守**2PL**。（例子见P275图8.6）

第八章 事务管理--并发控制

◆ 2PL会产生死锁（参见P275图8.7）。

2PL不隐含预防死锁的一次封锁法，但一次封锁法肯定遵守2PL。



第八章 事务管理--并发控制

§ 7 封锁的粒度

1、封锁的粒度 (Granularity)

封锁对象的大小。可以是数据库、表、记录、字段等。

{ 粒度大，封锁开销小，但影响共享
{ 粒度小，封锁开销大，但共享性好

2、多粒度封锁 (Multiple Granularity Locking)

同时支持多种封锁粒度供不同事务选择的封锁方法。

多粒度封锁方法依赖的数据结构----多粒度树

将数据库中的数据对象按相互关系和粒度大小组织成的树型结构，其中根结点表示最大数据粒度，通常为整个数据库，叶结点表示最小数据粒度。

第八章 事务管理--并发控制

多粒度封锁协议

允许对多粒度树中的每个结点独立地加锁，并且对每一个结点加锁隐含着对其后裔结点也加以同样的锁。

由事务直接加到数据对象上的封锁称为显式封锁，因上级结点加锁而引起下级对象被封锁，称为隐式封锁。

显式封锁与隐式封锁的效果是一样的。

多粒度封锁方法

为对某数据对象加锁，系统要检查：

- ◆ 该对象有无显式封锁与之冲突；
- ◆ 该对象的上级结点有无显式封锁与之冲突；
- ◆ 该对象的下级结点有无显式封锁与之冲突；

11/27/2019 当无任何冲突时方能加锁成功。

第八章 事务管理--并发控制

3、意向锁

用来指示下级结点正在被加锁的锁。对任一结点加锁时，必须先对其上级结点加意向锁。

作用

减少加锁时的封锁冲突检查工作量。只需检查上级结点与本结点是否已加了不相容的锁，并通过本结点的意向锁了解下级结点是否有不相容的锁，从而不必再检查下级结点。

三种意向锁

意向共享锁（IS锁）

如果要对某个对象加S锁，需先对其上级对象加IS锁。

意向排它锁（IX锁）

如果要对某个对象加X锁，需先对其上级对象加IX锁

第八章 事务管理--并发控制

共享意向排它锁（SIX锁）

如果要对某对象加S锁，并对其下级对象加X锁，则应对该对象加SIX锁。

意向锁的封锁和释放顺序

封锁：自上而下

释放：自下而上

第九章 数据库安全性

本章要求：

1、掌握安全性控制的一般方法：授权机制和口令机制

第九章 数据库安全性

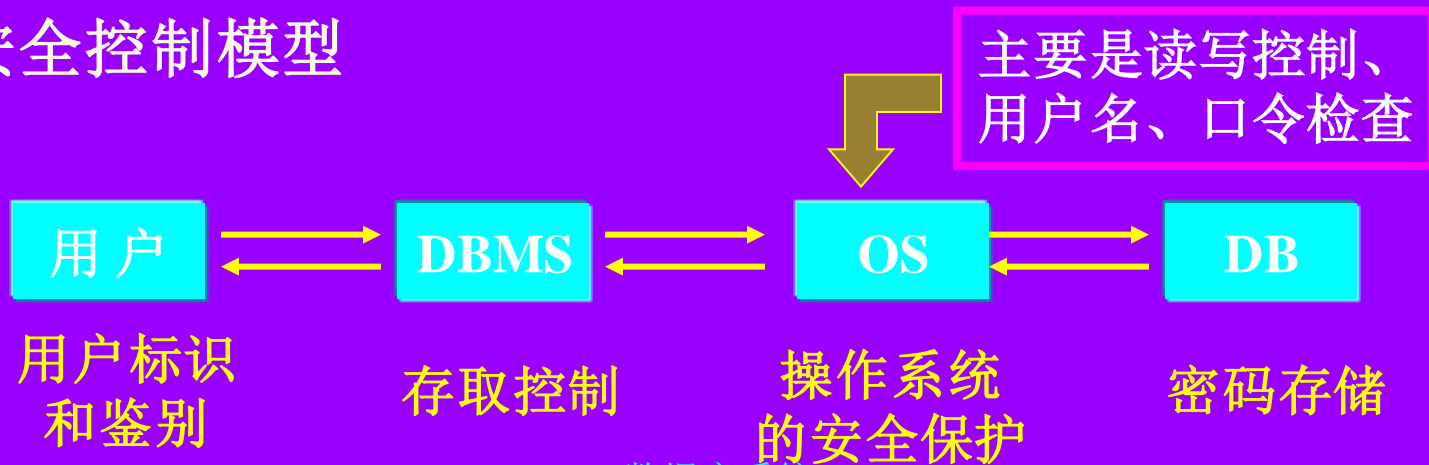
§ 1 数据库安全性控制

数据库的安全性：

保护数据库以防止不合法的使用所造成的数据泄露、更改或破坏。即：使未经授权的人员不能访问、改变和破坏数据，也就是数据库的存取控制。

两方面的含义 { 拒绝非法人员访问
拒绝程序错或操作错所造成的非法存取

安全控制模型



第九章 数据库安全性

1、用户标识和鉴别

是系统提供的最外层安全保护措施，用于鉴别用户身份，确认是否为合法用户。

🕒 自报家门：用一个标识符来标明用户身份（称为用户名）。系统内部记录着所有合法的用户标识，用户进入系统时，先要输入自己的用户名，系统确认合法后方可进入下一步的核实。

🕒 口令证实：用户选定一个密码，进一步标明身份

说明：

⊗ 用户名是公开的（便于通信），且一般不作更改

口令是保密的，且用户可自己修改。系统应定期提醒用户改变口令

第九章 数据库安全性

⊗ 为防止口令被窃，系统对口令自动加密。系统存储加了密的口令。

⊗ 为进一步加强口令保护，可采用随机数特殊处理。

⊗ 口令设防可层层进行。

⊗ 还可利用指纹、声音、IC卡、
使用条件（时间、地点等）

2、数据存取控制

数据库安全最重要的一点就是确保只授权给有资格的用户访问数据库的权限，同时保证未被授权的人员无法存取数据。

存取控制机制 { 定义用户权限
合法权限检查

第九章 数据库安全性

定义用户权限

用户权限：对数据对象允许执行的操作类型。
用户权限定义了用户在那些对象上具有那些类型的操作。系统提供一种语言来定义用户权限，并将用户权限登记在数据字典中。

模式、外模式、内模式 ↔ 建立、修改、检索
表、记录、字段 ↔ 查、增、删、改

合法权限检查

每当用户发出存取数据的操作请求后，系统根据数据字典及安全规则，检查用户是否具有相应的权限，若不具有，则拒绝存取。

第九章 数据库安全性

3、自主存取控制方法

用户对不同的对象有不同的存取权限，不同的用户对同一对象也有不同的存取权限，并且用户可将其拥有的权限转授给其他用户。

SQL标准中，数据对象的创建者自动拥有该对象的所有操作权限，并通过GRANT语句和REVOKE语句来实现权限的授予和撤消。

如：将关系Student上的查询、插入权限授予王平，并允许其将这些权限转授他人。

**GRANT SELECT, INSERT ON Student TO 王平
WITH GRANT OPTION;**

收回王平及其转授出去的在关系Student上的插入权限。

REVOKE INSERT ON Student FROM 王平 CASCADE

第九章 数据库安全性

说明:

△ 用户权限定义中数据对象范围越小，即授权粒度越细，授权子系统就越灵活，但定义与检查权限的开销也会相应增大。

△ 有些系统能够提供与数值有关的授权。即支持存取谓词。参见P291表9.5。

△ 自主存取控制存在“无意泄露”的问题

如甲将某写数据的权限授予乙，其本意是只允许乙本人操纵这些数据，但乙将这些数据复制了下来，并对副本拥有了全部的权限，乙可以不经甲同意，随意授权或转发这些数据，从而甲的意图没能得到保证。

第九章 数据库安全性

4、强制存取控制方法

对每一数据对象标以一定的密级，每一用户被授予某一个级别的许可证。对任意一个对象，只有具有合法许可证的用户才可存取。

强制存取控制中DBMS管理的全部实体

{	主体：用户或进程
	客体：数据对象， 如表、视图等

强制存取控制中，DBMS为每个主体或客体的实例指派一个敏感度标记，对于主体，称作许可证级别，对于客体，称作密级。敏感度标记分为绝密、机密、可信和公开等。

第九章 数据库安全性

主体存取客体的原则

- (1) 仅当主体的许可证级别大于或等于客体的密级时，主体才能读取客体；
- (2) 仅当主体的许可证级别等于（或小于）客体的密级时，主体才能写客体。

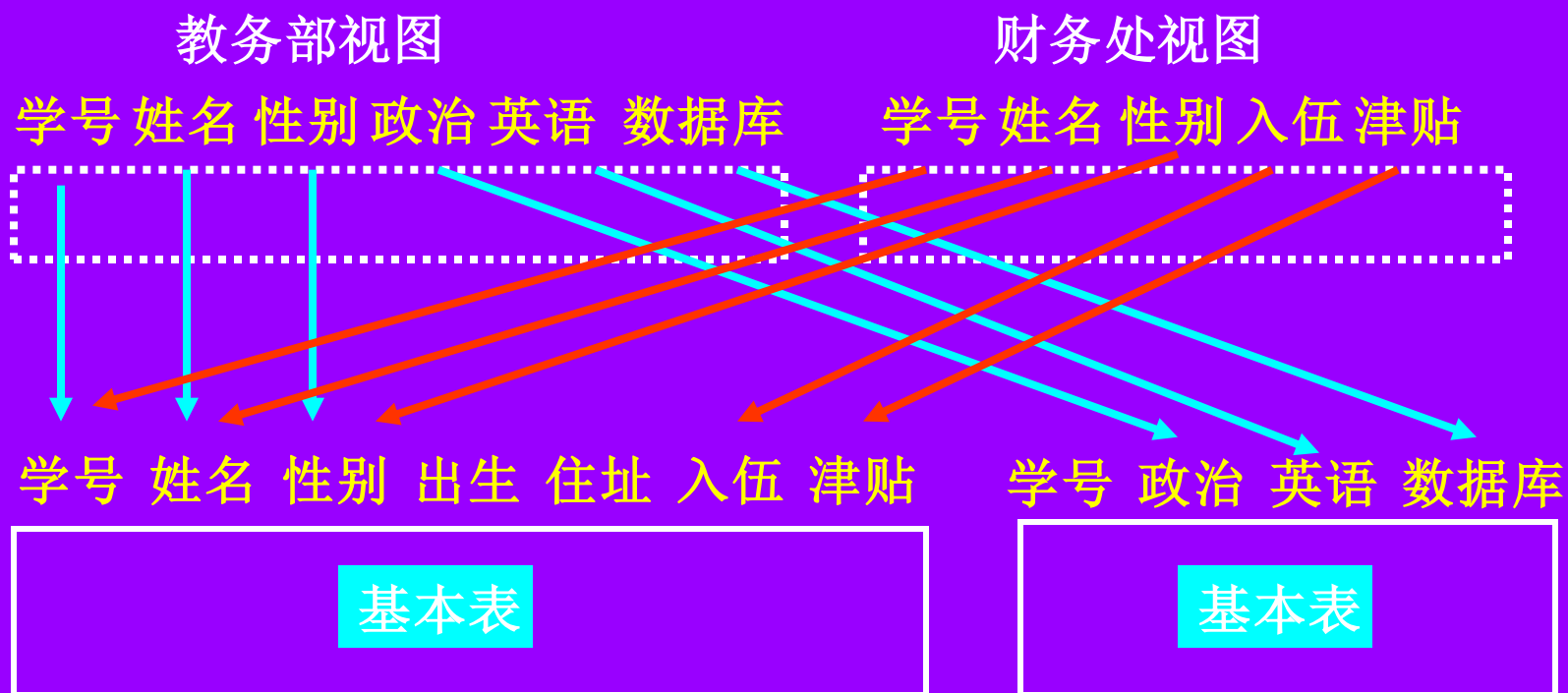
说明

- △ 上述两条原则保证了当主体的许可证级别低于客体的密级时，拒绝访问。
- △ 规则（2）中的“等于”使得高级别的用户不能修改低级别的数据。如下属部门送报给上级部门的数据，上级部门只能读取，不能修改。而“小于”使得低级用户一旦向高级用户呈报数据，则不能再反悔、修改。

第九章 数据库安全性

5、视图机制

(1) 利用视图机制，限制用户在一定的范围内使用数据，把要保密的数据通过视图机制对无权存取的用户隐藏起来，从而自动实现对数据提供一定程度的安全保护。



第九章 数据库安全性

5、视图机制

(1) 利用视图机制，限制用户在一定的范围内使用数据，把要保密的数据通过视图机制对无权存取的用户隐藏起来，从而自动实现对数据提供一定程度的安全保护。

(2) 视图机制间接地实现了支持存取谓词的用户权限定义，在不支持存取谓词的系统中，可先建立视图，然后在视图上进一步定义存取权限。

6、审计

把用户对数据库的所有操作自动记录下来放入审计日志，**DBA**可以利用审计跟踪的信息，重现导致数据库现状的一系列事件，找出非法存取数据的人、时间和内容。这可用于事后追查突破存取控制的情况。

审计的开销很大，因此主要用于安全级别较高的部门

7、数据加密

对保密级别较高的数据，可以以密文的形式存储在数据库中，从而防止数据在存储和传输中失密的情况发生。这种技术对防止存储设备丢失和线路传输中的窃听非常有效。

加密的基本思想是：应用程序在提出写数据库请求时，**DBMS**根据一定的算法将明文数据转换成不可直接识别的数据后写入数据库，应用程序在提出读数据请求时，**DBMS**从数据库读出数据，并按一定的算法将密文还原成明文后交给应用程序。

第九章 数据库安全性

只有拥有读密钥的人才能读出数据并还原为明文，
只有拥有读、写密钥的人才能读出数据----还原为明文----
修改----加密为密文----系统检查，合法后存储之。

数据加密和解密的开销也是很大的，因此，一般仅用于数据密级较高的系统。

8、SQL的安全性方法

{ 数据控制：GRANT, REVOKE
视图机制

9、安全保密与系统开销

这是一对矛盾，需权衡（保密强度）

原则：破坏的代价远超过可能获得的利益

第九章 数据库安全性

§ 2 统计数据库安全性

在统计数据库中存在着特殊的安全性问题，即可能存在着隐蔽的信息通道，使得可以通过合法的查询导出不合法的信息。

详见P295

第十章 数据库完整性

本章要求：

- 1、掌握完整性约束条件
- 2、掌握完整性控制机制
- 2、掌握SQL中的完整性方法，

本章内容：

- § 1 完整性约束条件
- § 2 完整性控制
- § 3 SQL的完整性实现

请选择内容

返回

第十章 数据库完整性

§ 1 完整性约束条件

数据库的完整性：指数据的正确性和相容性。

作用：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。主要防范对象是不符合语义的数据。

数据库的安全性：保护数据库，防止非法存取和恶意破坏
主要防范对象是非法用户和非法操作。

作用不同，目标一致

完整性检查：DBMS提供一定的机制来检查数据库中的数据，看其是否满足语义规定的条件。



完整性约束条件

第十章 数据库完整性

完整性约束条件是对数据语义上的要求

模式定义时对数据类型等的说明是对数据语法上的要求

1、完整性约束条件作用的对象

- 关系 若干元组间、关系之间的联系的约束
- 元组 元组中各个字段间的联系的约束
- 列 类型、取值范围、精度、排序等约束条件

2、静态约束和动态约束

静态约束：确定状态时，数据对象所应满足的约束条件，是反映数据库状态合理性的约束。

动态约束：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，是反映数据库状态变迁的约束。

第十章 数据库完整性

3、完整性约束条件分类（P304）

（1）静态列级约束

对一个列的取值域的说明。

① 对数据类型的约束

规定每一列数据的类型、长度、精度等。

② 对数据格式的约束

规定日期数据的格式或组合数据各部分的次序及含义等。

③ 对取值范围或取值集合的约束

如：学生成绩的取值范围为0—100，性别为男女

④ 对空值的约束

列是否可以取空值。

⑤ 其他约束

第十章 数据库完整性

(2) 静态元组约束

规定元组中各个列之间的约束关系。如：对一个包含职称和工资的元组，规定教授的工资不能低于2000元。

(3) 静态关系约束

一个关系的各个元组之间或若干关系之间的约束。

① 实体完整性约束：元组的码必须唯一。

② 参照完整性约束：外键只能取被参照关系的码值或空值。

③ 统计约束：某个字段值与多个元组的统计值之间的关系。

(4) 动态列级约束

修改列定义或列值时应满足的约束条件。

第十章 数据库完整性

如：一个已存在空值的列，不能修改为不许空的列。

② 修改列值时的约束

规定新旧值之间的关系。如新值不能小于旧值。

(5) 动态元组约束

元组中某个列被修改时，与其他列之间必须满足的约束条件。如学员的职务改为班长时，班长津贴要与之相符

(6) 动态关系约束

关系新旧状态之间的约束。如银行划款，划出帐户要增加一条划出记录，划入帐户要增加一条划入记录，且款值要相等。

第十章 数据库完整性

§ 2 完整性控制

1、DBMS的完整性控制机制

- 定义功能，提供定义完整性约束条件的机制
- 检查功能，检查用户操作是否违背了完整性约束条件
- 拒绝违背完整性约束条件的用户操作，保证数据完整性

完善的完整性控制机制应允许用户定义前述六类完整性约束条件

第十章 数据库完整性

2、立即执行约束和延迟执行约束

事务中更新语句
执行完后就进行
完整性检查

整个事务执行完后
才进行完整性检查

拒绝操作对立即执行约束，仅拒绝刚执行的更新语句，但对延迟执行约束，要拒绝整个事务，把数据库回滚到事务执行前的状态。

如银行转帐应遵循借贷平衡原则，这类事务应该是延迟执行约束。

3、完整性规则的表示

用五元组 (D, O, A, C, P) 表示

约束作用的数据对象

触发完整性检查的操作是立即还是延迟方式

约束规则

选择A作用的数据对象的谓词

违反约束时执行的过程

例如：在学生表S中，学生的年龄不的低于18，也不得高于25的约束中

D 约束作用的对象为年龄属性SA

O 当插入或修改学生记录时

A $SA \geq 18 \text{ and } SA \leq 25$

C 无（所有学生）

P 拒绝执行该操作

4、完整性约束条件分类

对关系数据库，有

实体完整性

参照完整性

用户定义的完整性



按约束条件作用的对象和约束方式仍可细分

目前实现的数据库系统一般均提供了实体完整性、参照完整性和用户自定义完整性，对于违反前两种完整性的操作，一般都采用拒绝执行的方式处理，而对于违反第三种完整性的处理要稍微复杂一些。

5、参照完整性实现中的几个问题

① 外码能否接受空值的问题

在实现参照完整性时，有些外码允许取空值，有些则不允许，因此，系统除提供定义外码的机制外，还应提供定义外码列是否允许取空值的机制。

第十章 数据库完整性

②在被参照关系中删除元组的问题

有三种不同的策略：

a. 级联删除：将参照关系中所有外码值与被参照关系中要删除元组主码值相同的元组一并删除。若还有其它关系参照了参照关系，则继续级联下去。

b. 受限删除：仅当参照关系中没有任何元组的外码值与被参照关系中要删除元组的主码值相同时，系统才执行删除操作，否则拒绝删除。

c. 置空值删除：删除被参照关系的元组，并将参照关系中相应元组的外码值置为空值。

第十章 数据库完整性

③ 在参照关系中插入元组时的问题

有两种策略：

a. 受限插入：仅当被参照关系中存在主码值与参照关系中欲插入元组的外码值相等的元组时，才执行插入操作，否则拒绝。

b. 递归插入：首先向被参照关系中插入相应的元组，其主码值等于参照关系中欲插入元组的外码值，然后向参照关系中插入元组。

第十章 数据库完整性

④ 修改关系中主码的问题

a. 不允许修改主码

不允许用UPDATE语句修改关系的主码，若确实需要修改，必须先删除元组，再将含有正确主码的元组插入。

b. 允许修改主码

允许修改主码，但必须保证主码的唯一性及非空，否则拒绝。

若修改被参照关系的主码，必须检查参照关系是否有元组的外码与修改的主码相等，若有，则可采用级联修改、拒绝修改和置空修改三种策略。

若修改参照关系的外码，必须检查被参照关系是否有主码值与此外码值相等的元组，若没有，则可以采用受限插入和递归插入两种策略。

第十章 数据库完整性

§ 3 SQL的完整性实现

1. 实体完整性

定义基本表的主关键字时，用

NOT NULL指明要求非空

UNIQUE指明要求唯一

2. 参照完整性

定义基本表时，可以定义一个主键和若干个外键

⌚ 定义外键时，用**REFERENCES**定义外键来自的表名

⌚ 可以用参照完整性任选项**ON DELETE**

有了**ON DELETE**，当要删除被参照表中被引用的主键时，为了不破坏参照完整性约束，提供三种可能的处理办法：

第十章 数据库完整性

ON DELETE



第十章 数据库完整性

例如： 若对外键DNO的定义是ON DELETE CASCADE，

则DEPT中前四个记录不能删除，第五个记录可删除

职工关系 EMP

ENO ENAME DNO

E001	张一	D002
E002	张三	D003
E003	李四	D001
E004	李五	D001
E005	刘六	D002
E006	刘七	D004
E007	王三	D003
E008	王发	D001
E009	钱好	D004

主键

部门关系 DEPT

DNO DNAME

D001	一车间
D002	二车间
D003	财务科
D004	人事处
D005	科技处

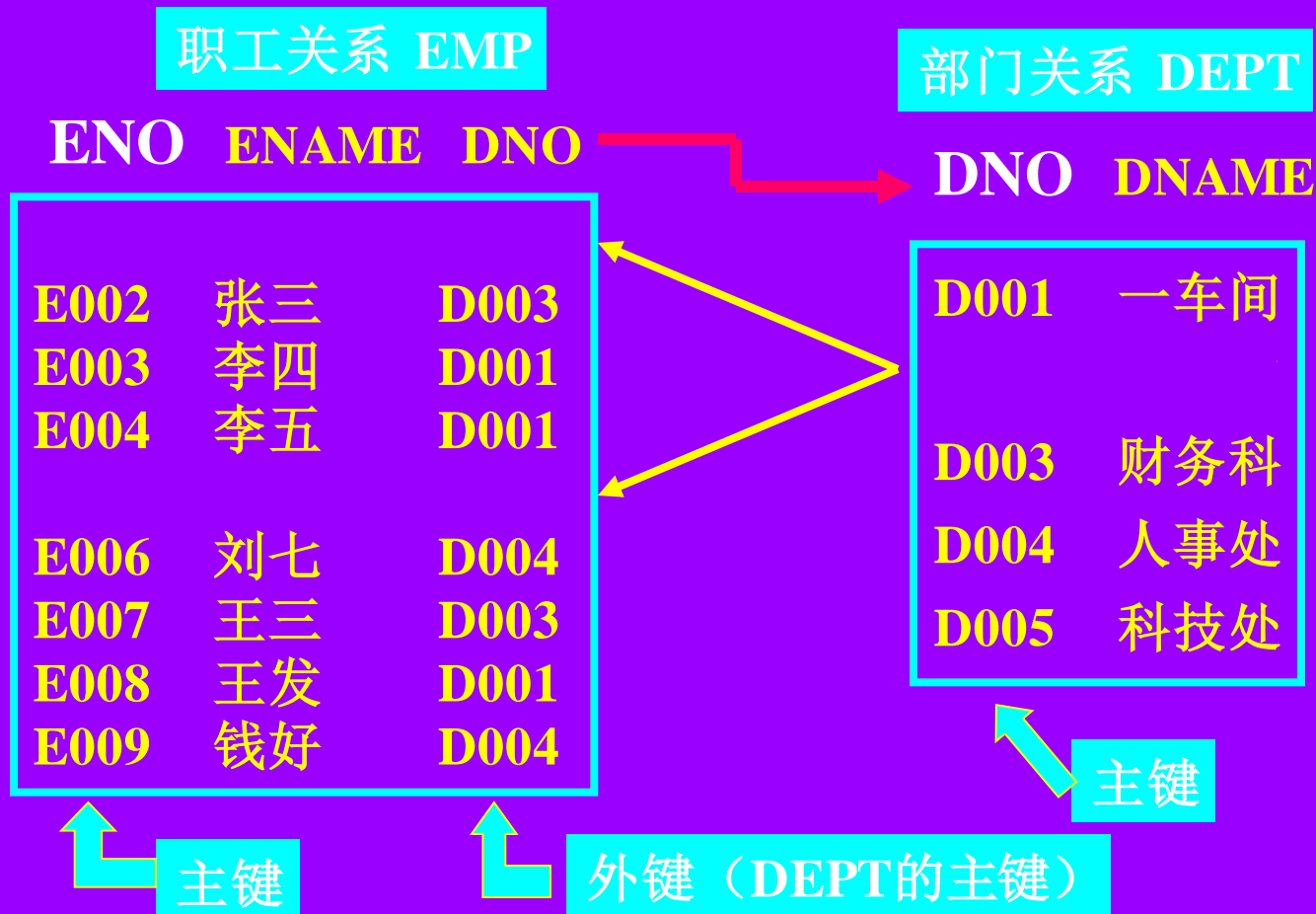
主键

外键 (DEPT的主键)

第十章 数据库完整性

例如： 若对外键DNO的定义是ONDELETE CASCADE,

当删除DEPT中D002记录时，则EMP中E001和E005两条记录也被删除



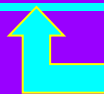
第十章 数据库完整性

例如： 若对外键DNO的定义是ONDELETE CASCADE,

当删除DEPT中D002记录时，则EMP中E001和E005两条记录也被删除

职工关系 EMP

ENO	ENAME	DNO
E002	张三	D003
E003	李四	D001
E004	李五	D001
E006	刘七	D004
E007	王三	D003
E008	王发	D001
E009	钱好	D004



主键



外键

部门关系 DEPT

DNO	DNAME
D001	一车间
D003	财务科
D004	人事处
D005	科技处

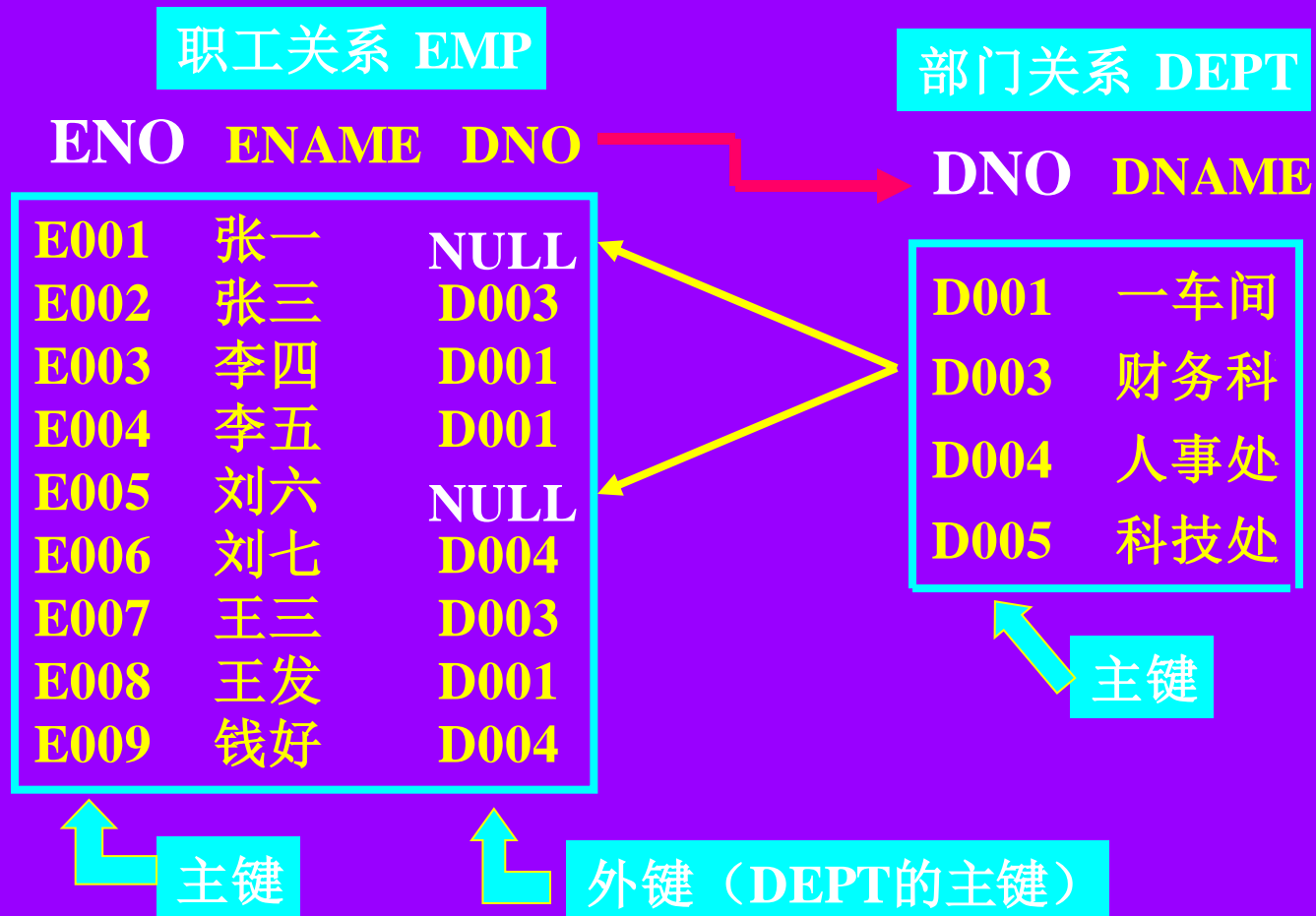


主键

第十章 数据库完整性

例如： 若对外键DNO的定义是ON DELETE ，

当删除DEPT中D002记录时，则E001和E005两条记录的DNO置为NULL



第十章 数据库完整性

3、用户定义的完整性

在定义字段时用**CHECK**子句，说明该列应满足的条件。

格式：**CHECK**（条件）

4、示例：定义学生表S，课程表C、选课表SC

```
CREATE TABLE S (S# CHAR (8) NOTNULL UNIQUE,  
                SN VARCHAR (20) NOTNULL,  
                SD VARCHAR (15) ,  
                SA SMALLINT CHECK (SA>15) ,  
                PRIMARY KEY (S#) ) ;
```

```
CREATE TABLE C (C# CHAR (4) NOTNULL UNIQUE,  
                CN VARCHAR (20) NOTNULL,  
                PRIMARY KEY (C#) ) ;
```

第十章 数据库完整性

```
CREATE TABLE SC (S# CHAR (8) NOTNULL,  
                  C# CHAR (4) NOTNULL,  
                  G CHAR (1) ,
```

组合主键

→ PRIMARY KEY (S#, C#)

FOREIGN KEY (S#)

REFERENCES S

若S中某学生记录被删除
SC中该生选课记录全删

→ ON DELETE CASCADE,

FOREIGN KEY (C#)

REFERENCES C

ON DELETE RESTRICT) ;

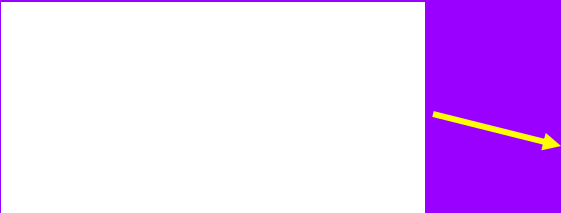
若某课程有人选修，则
C中该课程选课记录不能删

第十章 数据库完整性

又如：定义职工表 **EMP** (**ENO**, **ENAME**, **DNO**) ,
部门表 **DEPT** (**DNO**, **DNAME**)

```
CREATE TABLE DEPT (DNO CHAR (4) NOTNULL UNIQUE,  
                  DNAME CHAR (12) NOTNULL,  
                  PRIMARY KEY (DNO) ) ;
```

```
CREATE TABLE EMP (ENO CHAR (5) NOTNULL UNIQUE,  
                  ENAME CHAR (8) NOTNULL,  
                  DNO CHAR (4) ,  
                  PRIMARY KEY (ENO)  
                  FOREIGN KEY (DNO)  
                  REFERENCES DEPT  
                  ON DELETE SET NULL, ) ;
```



第十章 数据库完整性

5、说明

☎ 用SQL可在基本表定义时来定义完整性检查

优：简单、清晰

缺：不够灵活、功能不强（如插入、修改时的完整性检查）

☎ 此外，用SQL还可在基本表定义时来定义缺省（default）

格式：

DEFAULT {	值	用户指定的值
	USER	
	NULL	填入NULL

功能：在该列未输入值时，以指定值填入

例如，在S表中，若有民族属性，在定义S时可如下进行：

```
CREATE TABLE S (.....,  
                  MZ VARCHAR (10) ,  
                  .....);
```