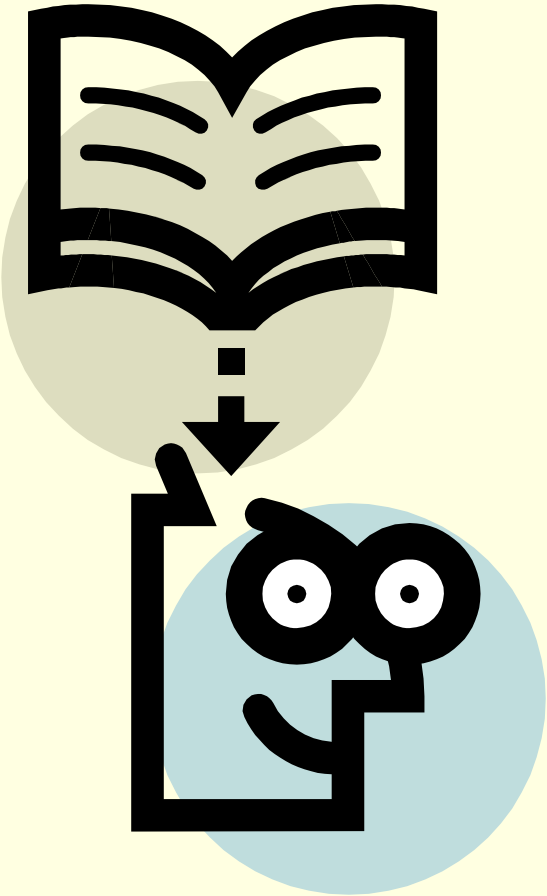
第11章 数据库设计
Chapter 11. Database Design

目录 Contents

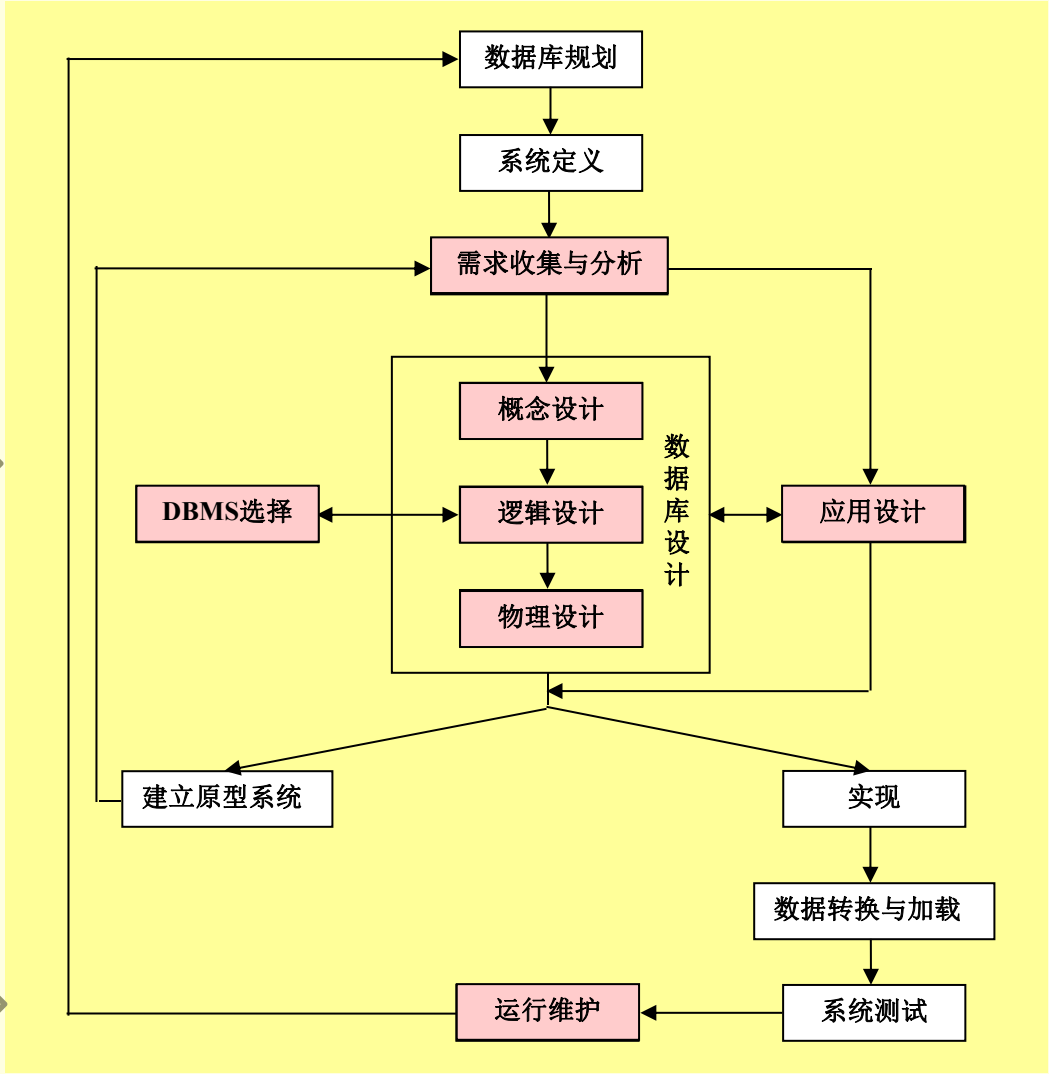


11.1 概述

- 概念回顾：
  - 数据库生命周期

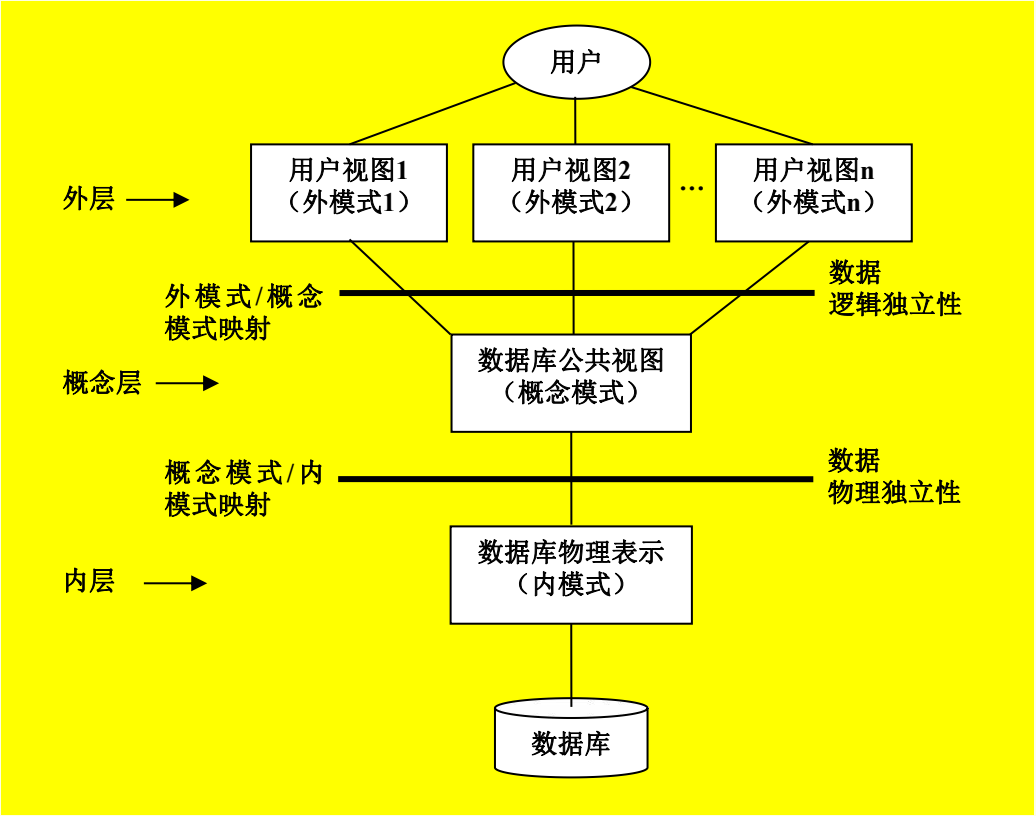
数据库设计
- 需求分析
- 概念设计
- 逻辑设计
- 物理设计

相关问题
- 数据库调整
- 数据库重组
- 数据库重构



11.1 概述

- ANSI-SPARC体系结构中的三种模式：
  - **外层是多个外模式**，每个外模式描述数据库中与特定用户相关的部分，即数据库的**用户视图**
  - **概念层是一个概念模式**，描述数据库中包含什么数据以及数据之间的关系，即数据库的**公共视图**
  - **内层是一个内模式**，描述数据库中数据是如何存储的，即数据库的**物理表示**



11.1 概述

- 数据库设计的任务、方法、过程
  - **任务**
    - ✓ 根据一个企业/单位/机构的信息（处理）需求及数据库系统的支撑环境（硬件、网络、OS、DBMS等）的特点，设计出满足用户需求的数据模式（外模式、概念模式、内模式）及典型应用。
  - **目标**
    - ✓ 对常用的、大多数典型应用（主要是数据库查询）能使用方便、运行性能满意。

11.1 概述

- 方法

(1) 面向过程的方法（process-oriented approach）

- ✓ 以处理需求为主，兼顾信息需求；
- ✓ 可能在数据库使用的初始阶段能比较好地满足应用的需要，获得好的性能，但随着应用的发展和需求变化，往往会导致数据库的较大变动或重构。

(2) 面向数据的方法（data-oriented approach）

- ✓ 以信息需求为主，兼顾处理需求；
- ✓ 较好地反映数据的内在联系，不但可以满足当前应用的需要，还可以满足潜在（将来）应用的需要；
- ✓ 与信息工程方法(IEM)一致：遵循James Martin于1980年代提出的数据中心原理，对业务过程目标进行分析和数据建模的基础上，分阶段开发信息系统的方法。

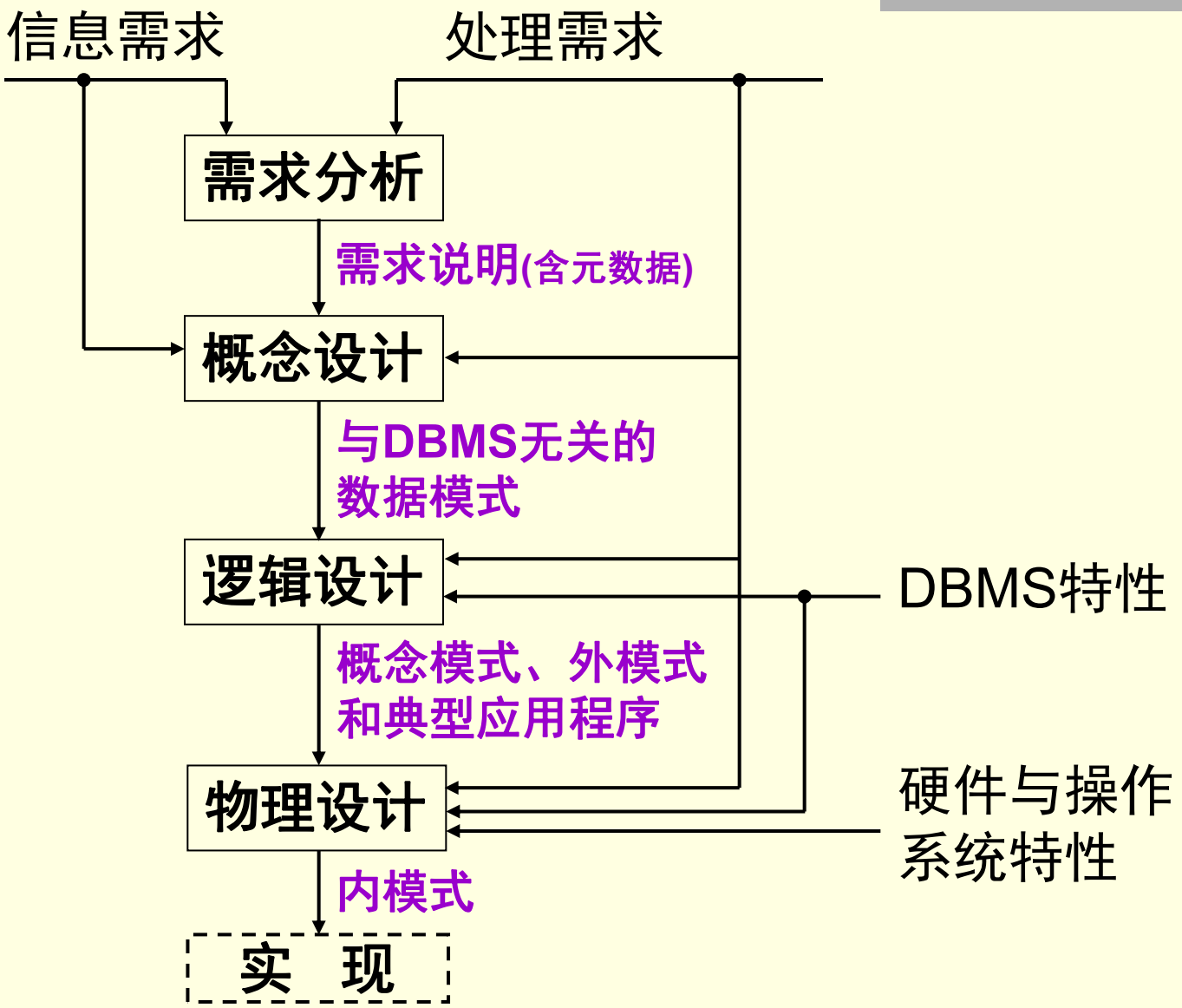
重硬件、轻软件 (1960's~70's) → 重软件、轻数据 (70's~80's) → 重数据(90's~)

11.1 概述

- **特点**
  - ✓ **反复性（iterative）**
    数据库设计不可能“一气呵成”、“一蹴而就”，需要反复推敲和修改完善才能完成。
  - ✓ **试探性（tentative）**
    数据库设计不同于求解数学题，设计结果一般不是唯一的。设计过程往往是一个试探的过程。各式各样的要求和制约因素之间往往是矛盾的。
  - ✓ **分步性（multistage）**
    数据库设计常常由不同的人员分阶段进行。一是由于技术上分工的需要；二是为了分段把关，逐级审查，保证设计的质量和进度。

11.1 概述

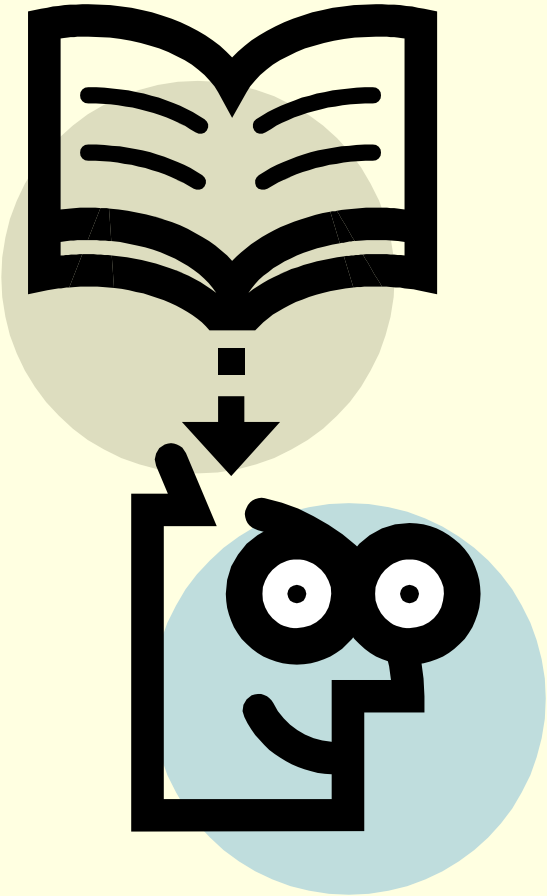
数据库设计的过程



11.1 概述

- 需求分析（Requirements Analysis）
  - 定义需求说明（specification of requirements）
  - 构建元数据库（metadata repository）
- 概念设计（Conceptual Design）
  - 概念建模（conceptual (E-R/UML) modeling）
- 逻辑设计（Logical Design）
  - 模式转换（E-R/UML to relational conversion）
  - 模式规范化（normalization）
  - 完整性约束设计（integrity constraints design）
  - 外模式设计（external schema design）
- 物理设计（Physical Design）
  - 内模式设计 （internal schema）

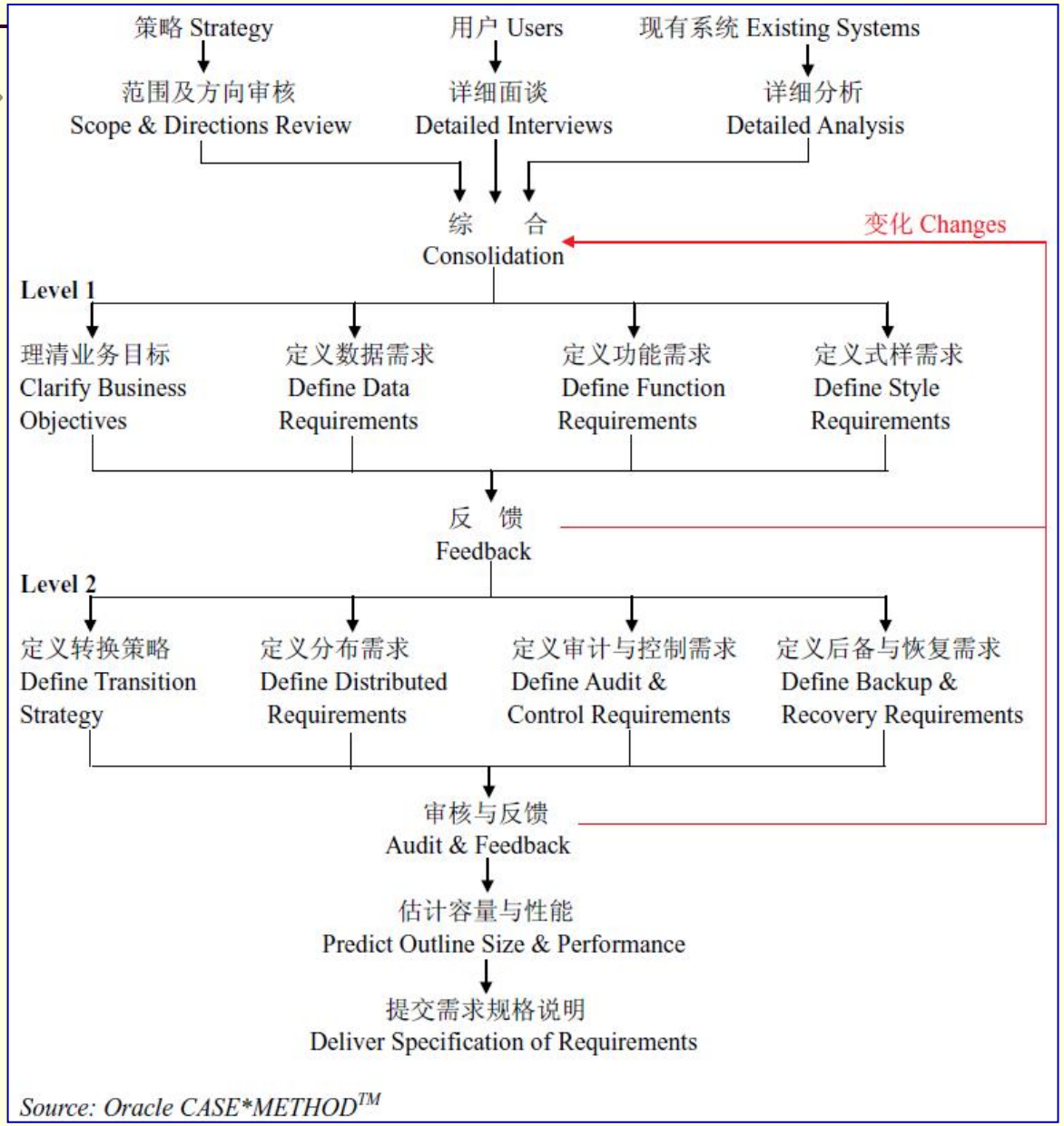
目录 Contents



11.2 需求分析

分析过程

- 要求分析人员
  - ✓ 掌握数据库技术
  - ✓ 熟悉公司的业务
  - ✓ 善于与用户沟通
  - ✓ 具有丰富的经验



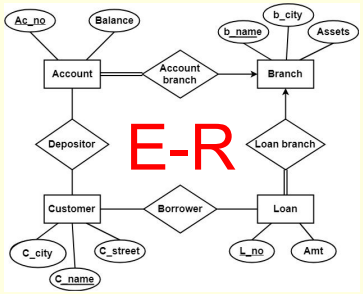
11.2 需求分析

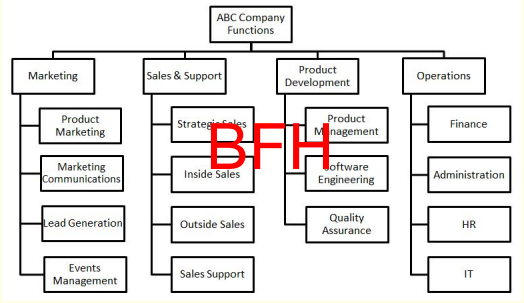
- 提交成果
  - 需求规格说明:
    - ✓ 功能-实体、功能-业务单位、实体-业务单位矩阵
    - ✓ 初步的（高层的）实体联系（ E-R ）图
    - ✓ 业务功能层次结构（BFH）图
    - ✓ 数据流图（DFD）
    - ✓ 数据分布、约束、输出格式需求
    - ✓ 数据量，功能使用频率，用户期望性能
    - ✓ 各种策略：审计，控制，后备/恢复，转换，etc.

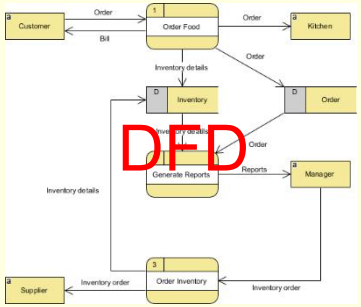
Example business function-to-data entity matrix

Business Function (users) \ Data Entity Types	Customer	Product	Raw Material	Order	Work Center	Work Order	Invoice	Equipment	Employee
Business Planning	X	X						X	X
Product Development		X	X		X			X	
Materials Management		X	X	X	X	X		X	
Order Fulfillment	X	X	X	X	X	X		X	X
Order Shipment	X	X		X	X		X		X
Sales Summarization	X	X		X			X		X
Production Operations		X	X	X	X	X		X	X
Finance and Accounting	X	X	X	X	X		X	X	X

Matrix





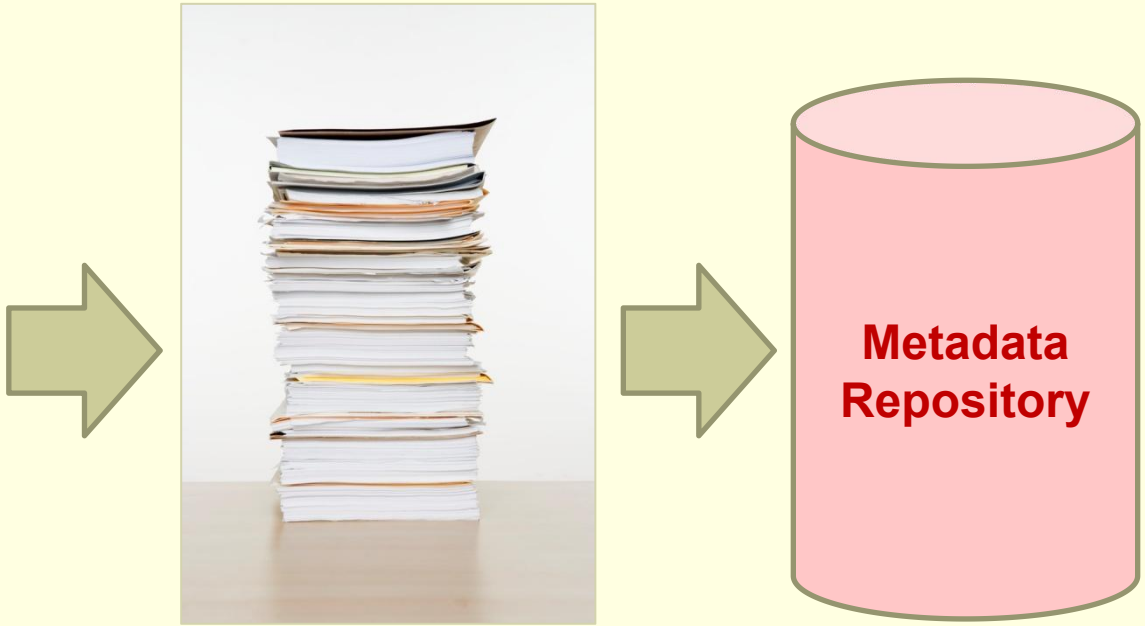


11.2 需求分析

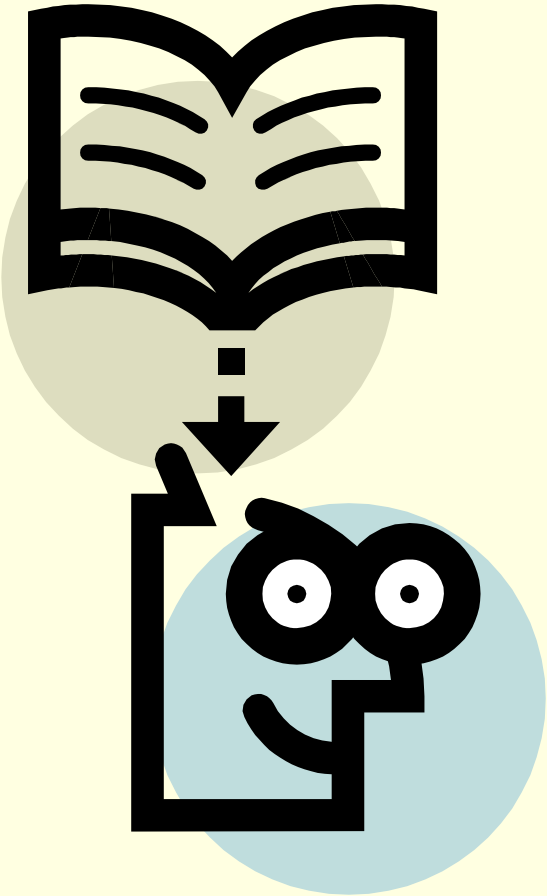
- 提交成果
  - 元数据库：

    对每个数据，记录以下信息：

    - ✓ 数据名
    - ✓ 类型/长度
    - ✓ 编码规则
    - ✓ 更新要求
    - ✓ 使用频率
    - ✓ 数据量
    - ✓ 语义约束
    - ✓ 保密要求



目录 Contents



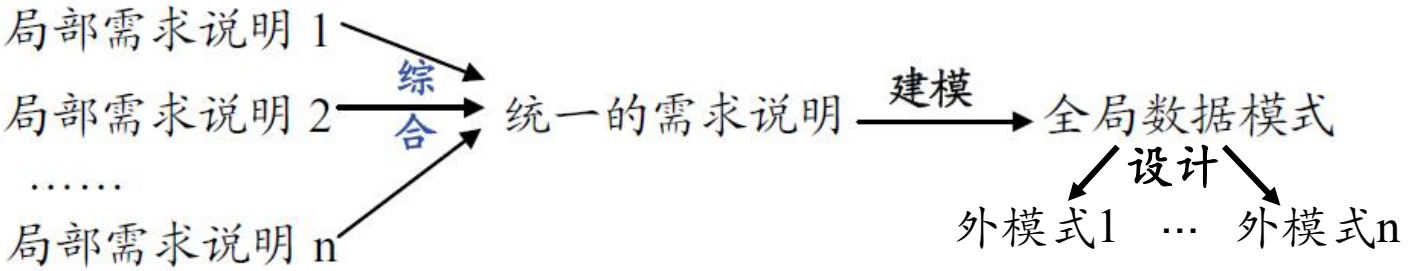
11.3 概念设计

11.3.1 概述

- 任务：对数据实体及其联系进行概念建模
- 技术：实体联系（E-R）建模或UML类图建模
- 方法：（E-R建模方法——在第2章中已学过）

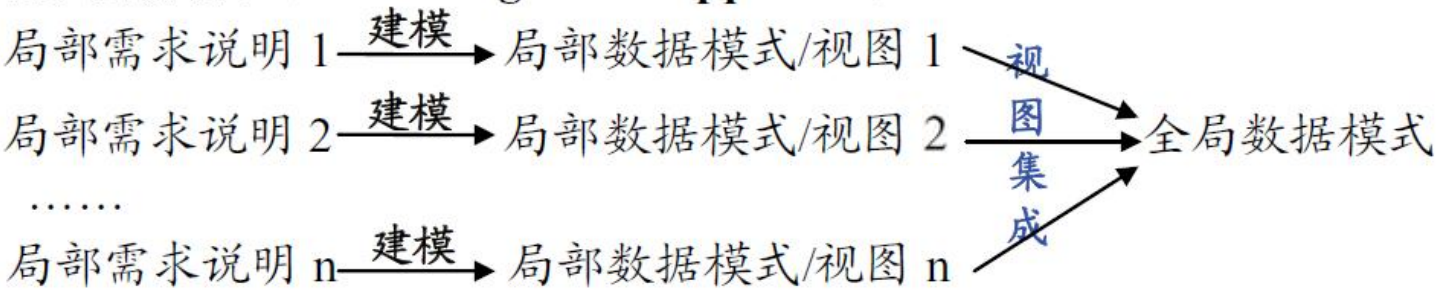
较适合于小型DB设计

- 集中式模式设计法（Centralized Schema Design Approach）：



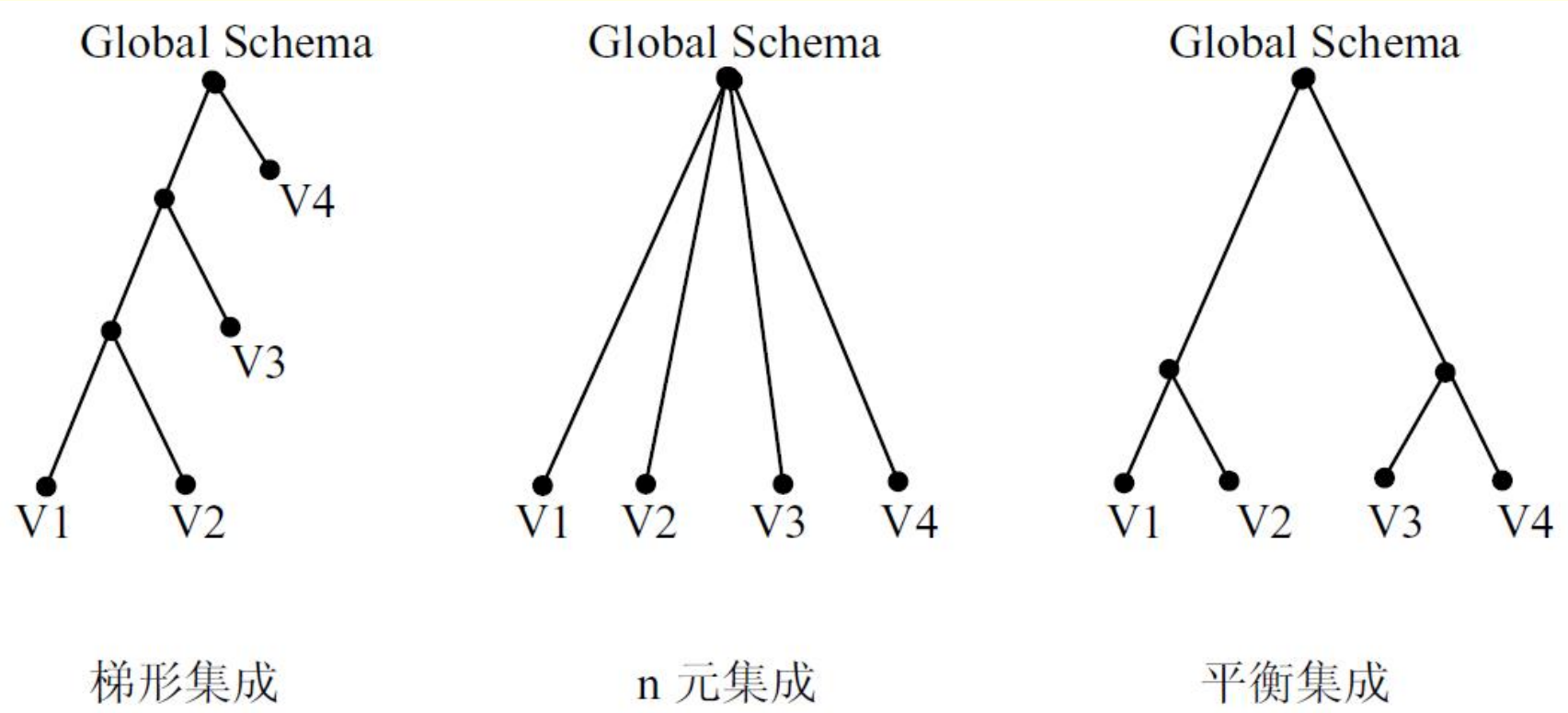
较适合于大型DB设计

- 视图集成法（View Integration Approach）：



11.3 概念设计

- “视图集成法” 中的视图集成策略



11.3 概念设计

- 视图集成时，须解决的主要问题：

  - **确认视图中的对应和冲突**

    对应是指语义上等价的概念（实体、属性或联系）。

    冲突指有矛盾的概念。常见的冲突有：

    （1）命名冲突。同名异义；同义异名。

    （2）概念冲突。同一概念在一个视图中作为实体，在另一视图中作为属性或联系。

    （3）域 冲 突。相同属性在不同视图中有不同的数据类型或取值范围（包括取值使用不同的度量单位）。

    （4）约束冲突。不同视图可能有不同的联系语义约束。

  - **修改视图，解决部分冲突**

    例如教材P227的图11-2中，【同义异名】“入学时间”和“何时入学”两个属性名可以统一成“入学时间”；【同名异义】学生分为大学生和研究生两类，课程也分为本科生课程和研究生课程两类；学号一律用字符串表示，等。

  - **合并视图，形成全局模式**

    尽可能合并对应的部分，保留特殊的部分，删除冗余部分；

    必要时对模式进行适当修改，力求使模式简明清晰。

11.3 概念设计： 视图集成例（教材图11-2和11-3）

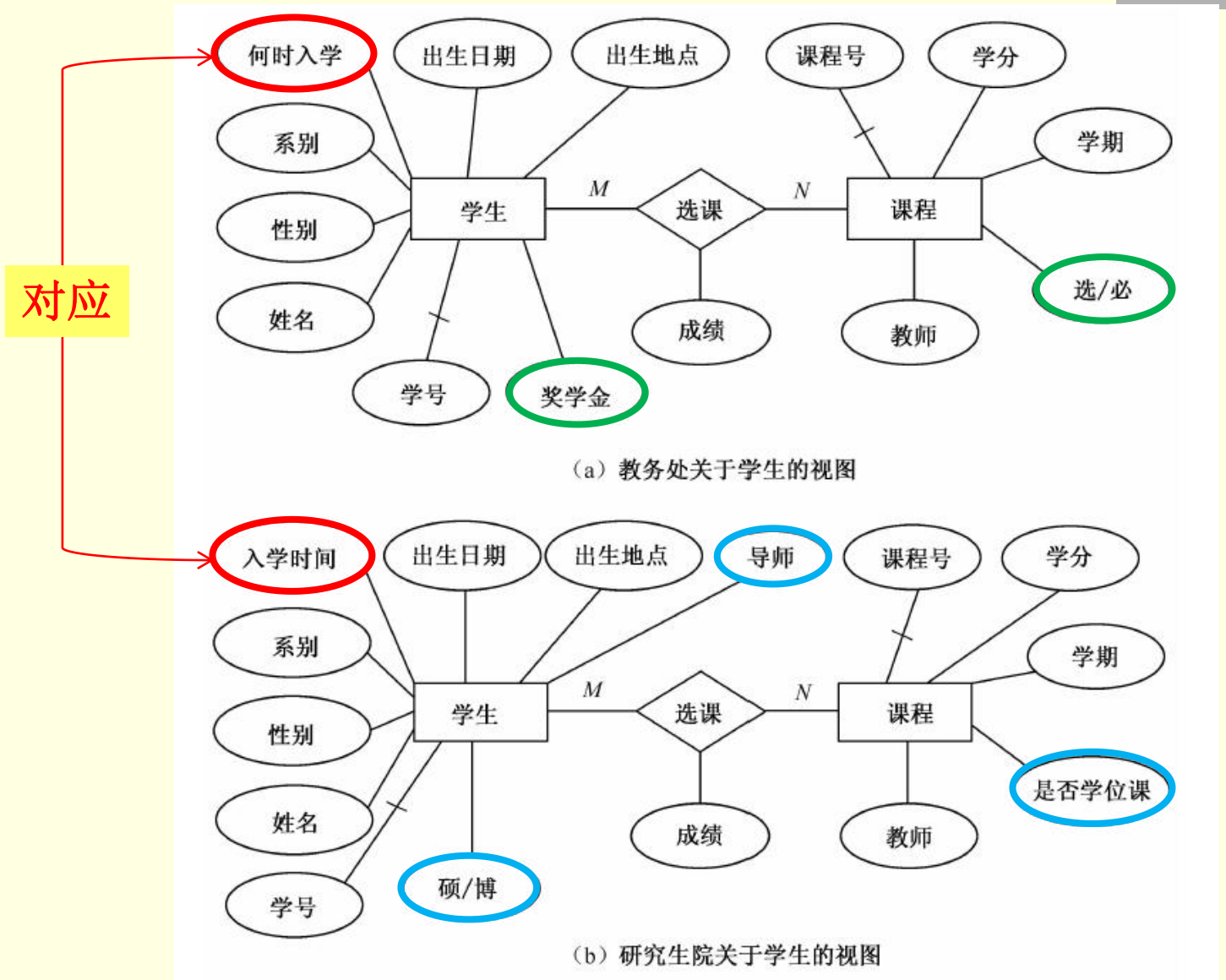


图11-2

11.3 概念设计： 视图集成例（教材图11-2和11-3）

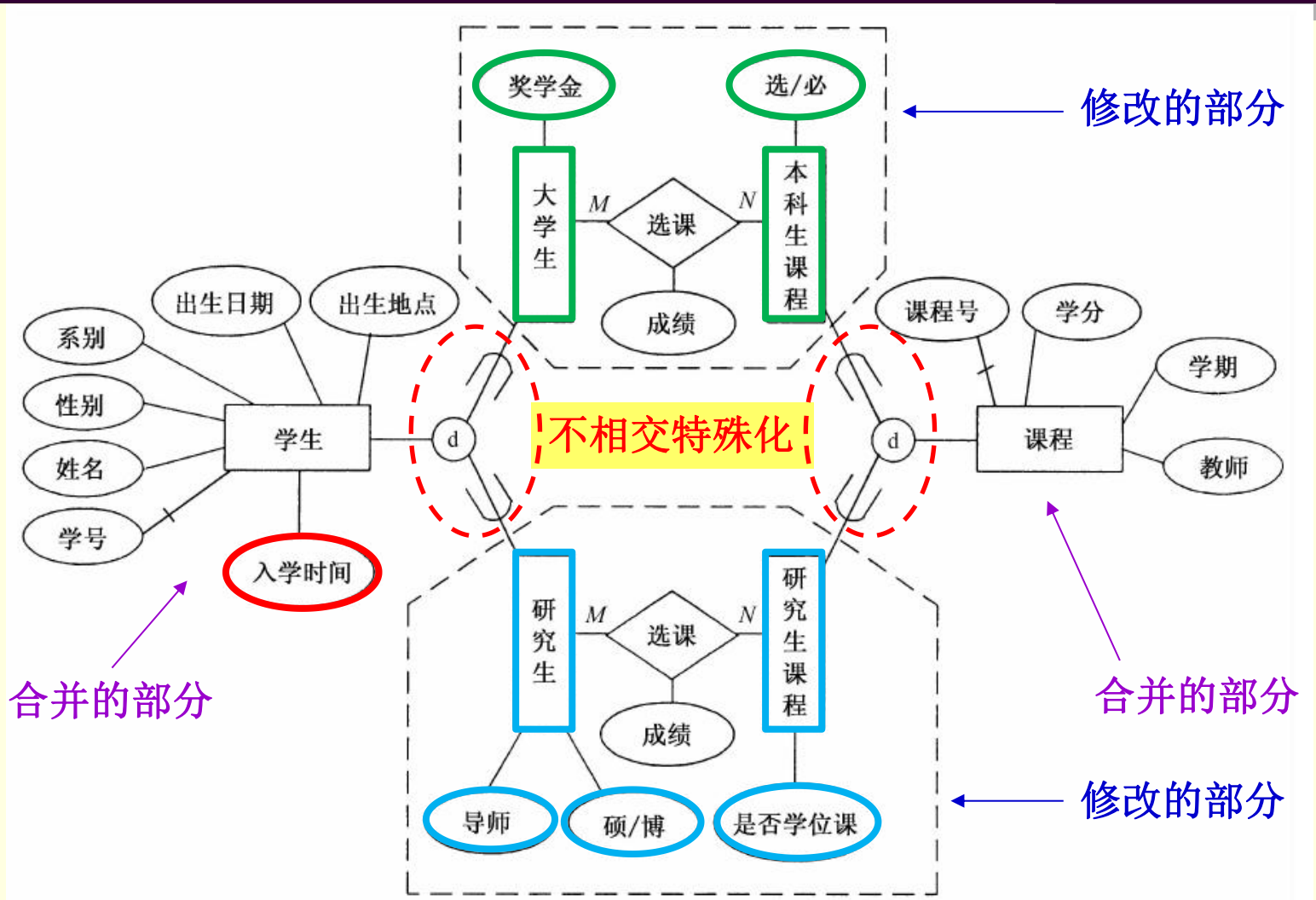
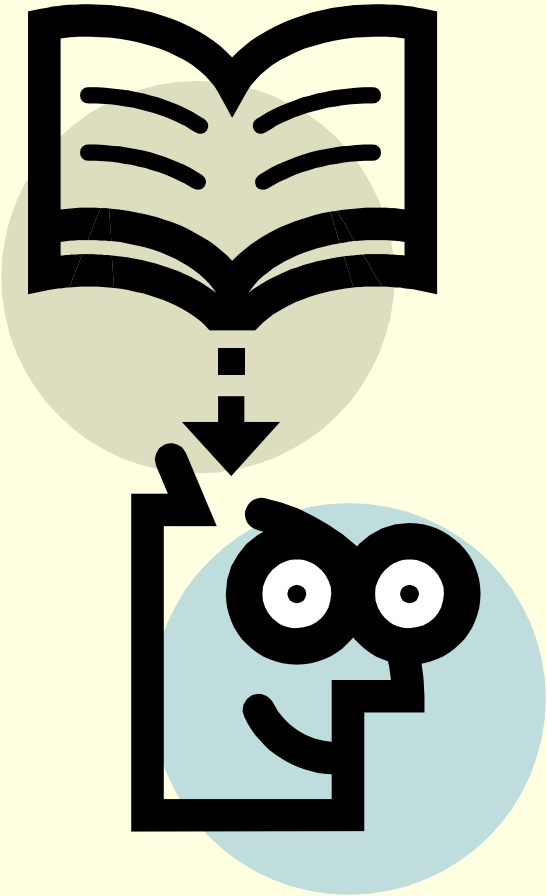


图 11-3　图 11-2 中两个视图的集成

目录 Contents



11.4 逻辑设计

11.4.1 任务与技术

- 任务
  - ✓ 设计数据库的逻辑（概念）模式和外模式。

- 技术
  - ✓ E-R模式向关系模式的转换 （主要技术）
  - ✓ 关系模式规范化（第10章中已学过）
  - ✓ 模式的调整
  - ✓ 外模式设计

11.4 逻辑设计

11.4.2 E-R向关系转换

- 两条转换规则：

实体及其属性 —转换→ 创建关系模式（含属性）

联系及其属性 —转换→ 根据规则，通过关系模式间的键引用（FK→PK），将联系及其属性
①尽量置于为参与该联系的某个实体所创建的关系模式中，
②或只能置于为该联系单独创建的关系模式中

11.4 逻辑设计

- **实体的转换：**
  - ✓ 模式及属性的命名：要易读、好记。 e.g. 合同
    - 好: contract (contractNo, contractName, firstPart, secondPart, signDate, totalPrice)
    - 不好: hetong (bh, mc, jf, yf, qdrq, zj)
  - ✓ 属性域的处理：
    - 选择RDBMS / SQL DBMS中的内置数据类型：
      4大类基本数据类型：Numeric
      String
      Date & Time
      Boolean
      或自定义数据类型。

11.4 逻辑设计

选修课程
C语言, C++语言, Java语言

- **非原子属性的处理：**
  - ✓ （对E-R中的集合属性）——在关系表中纵向展开；
  - ✓ （对E-R中的元组属性）——在关系表中横向展开。

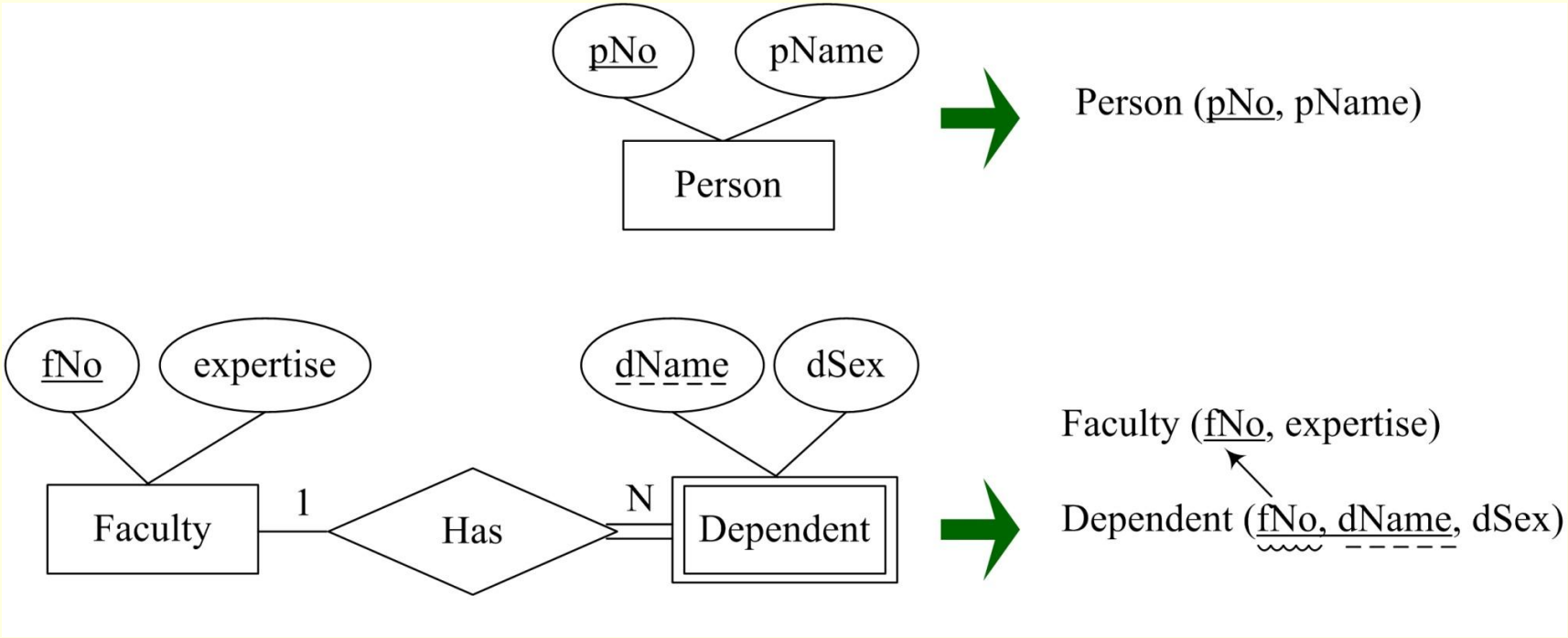
如：选修课的集合{C, C++, Java}

如：通信地址（邮编，省，市…）

- **键的处理：**
  - ✓ E-R中实体键成为关系模式的键
    （如有多个，选定其中之一作为模式的主键 (PK)）
- **弱实体的处理：**
  - ✓ 相应的关系模式中应包含其所有者实体的（主）键
    - e.g. 家属（职工号，家属姓名，年龄，与职工的关系）
    - PK = {职工号，家属姓名}；FK = {职工号}。

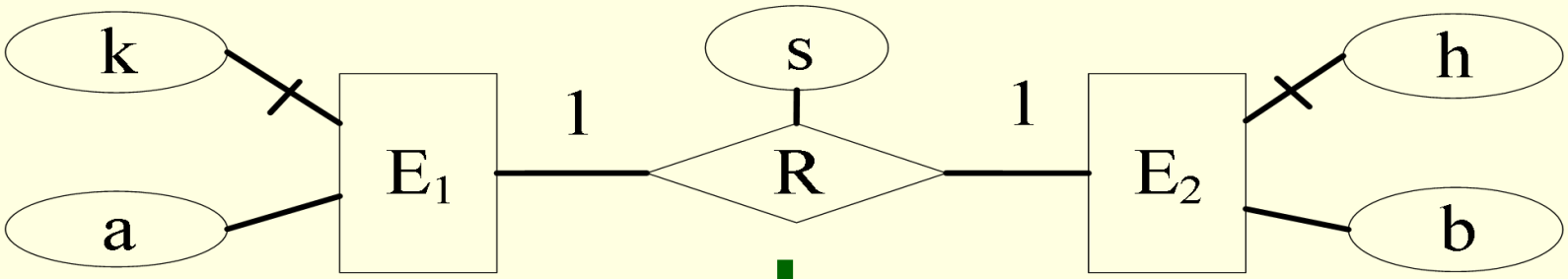
11.4 逻辑设计

实体与弱实体转换举例：



11.4 逻辑设计

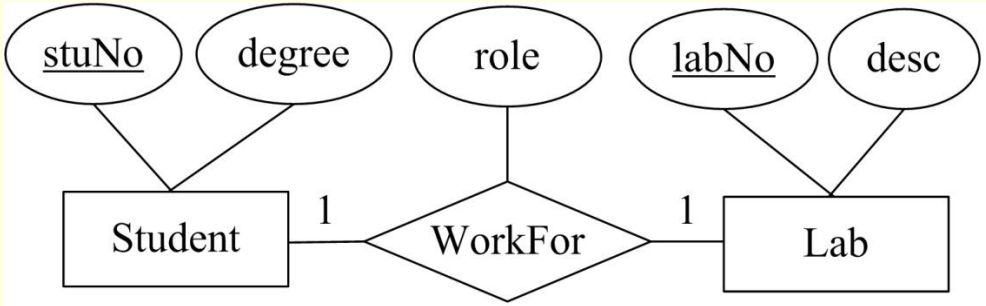
- **1:1联系的转换：**



若E1（或E2）是全参与的	若E1和E2均不是全参与的
R1 (k, a, h, s) 【表示了联系】	R1 (k, a)
外键不取null值	R12 (k, h, s) 【表示了联系】
R2 (h, b) 模式个数尽量少	R2 (h, b)
注： — PK ～ FK ····· 候补键 ⟶ 键引用	

11.4 逻辑设计

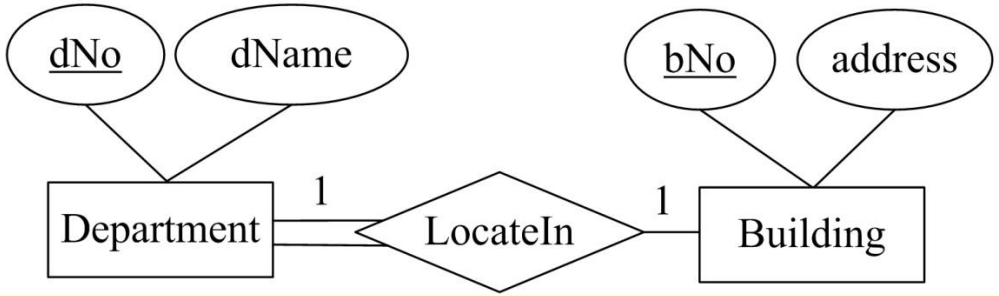
1:1联系转换举例：



Student (stuNo, degree)

WorkFor (stuNo, labNo, role)

Lab (labNo, desc)



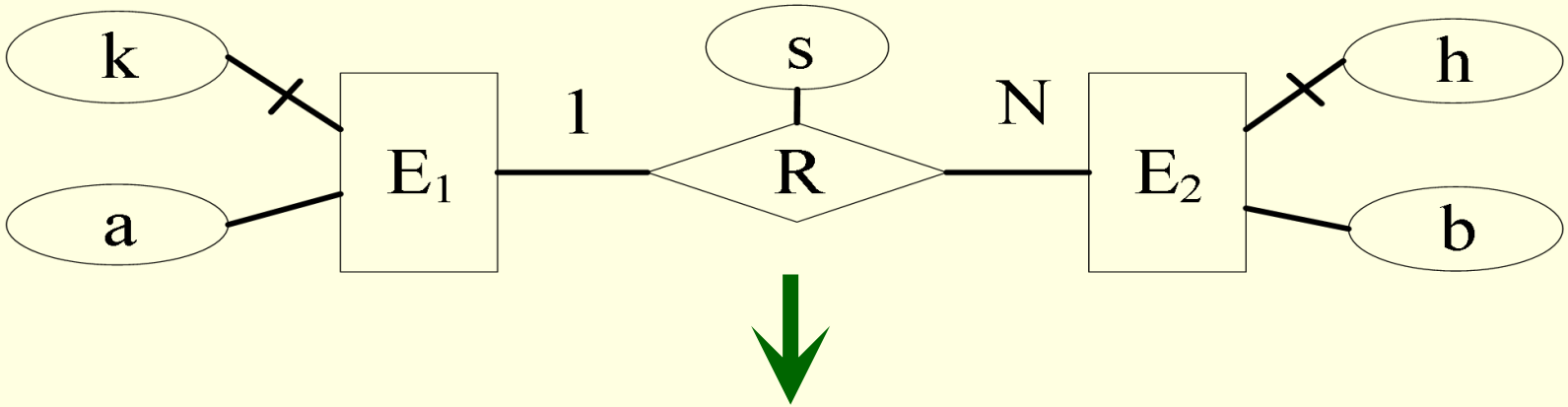
Building (bNo, address)

Department (dNo, dName, bNo)

外键不取null值　模式个数尽量少

11.4 逻辑设计

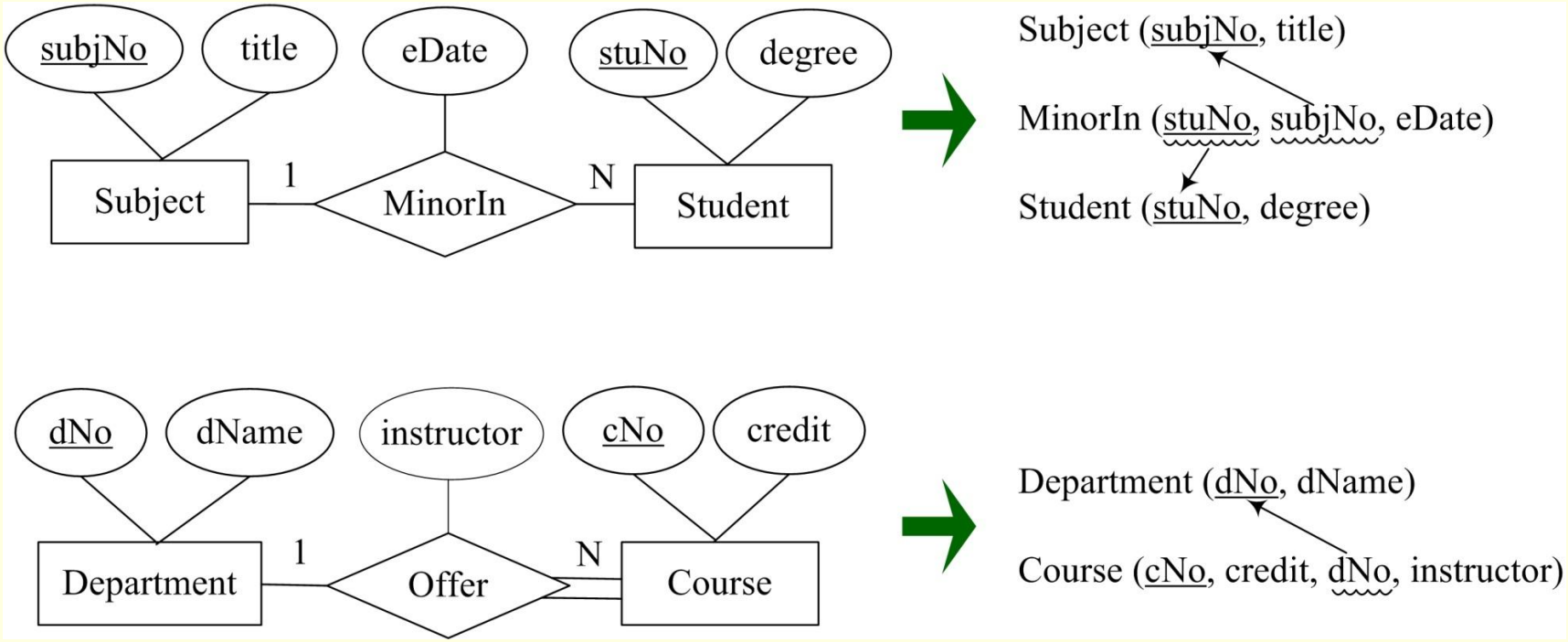
- **1:N联系的转换：**



若E2是全参与的	若E2不是全参与的
R1 (k, a) 模式个数尽量少	R1 (k, a)
外键不取null值	R12 (h, k, s) 【表示了联系】
R2 (h, b, k, s) 【表示了联系】	R2 (h, b)
注： — PK ～ FK ⟶ 键引用	

11.4 逻辑设计

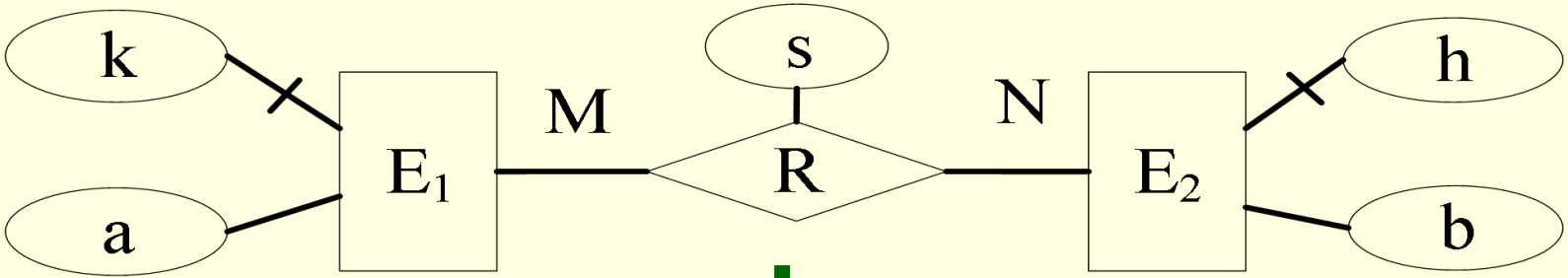
1:N联系转换举例：



外键不取null值　模式个数尽量少

11.4 逻辑设计

- **M:N联系的转换：**

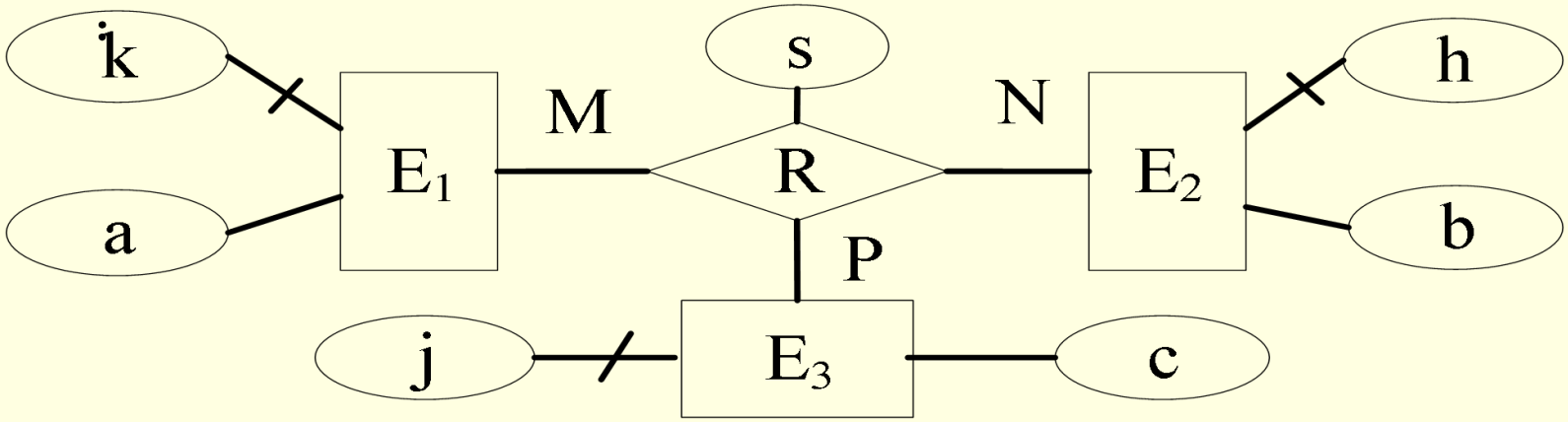


不管E1、E2是否全参与	
R1 (k, a)	模式个数尽量少
R12 (h, k, s) 【表示了联系】	外键不取null值
R2 (h, b)	
注： — PK ～FK ⟶ 键引用	

11.4 逻辑设计

小概率事件

- **M:N:P三元联系的转换**（如果不是1:1:P且P端全参与的话）



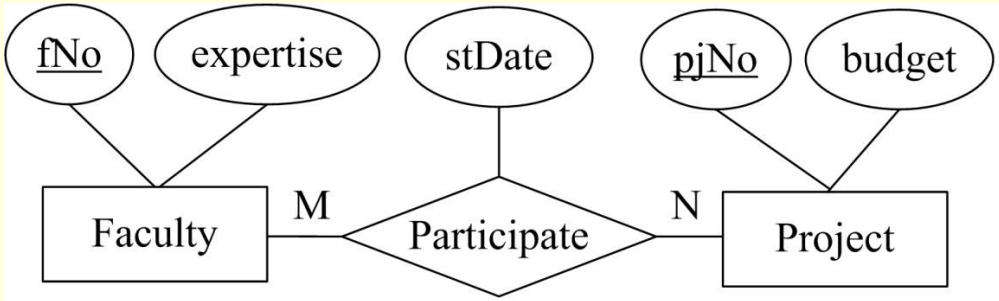
模式个数尽量少

外键不取null值

不管E1、E2、E3是否是全参与：
R1 (k, a)
R2 (h, b)
R3 (j, c)
R4 (k, h, j, s)　【表示了联系】

11.4 逻辑设计

M:N联系转换举例：



Faculty (fNo, expertise)

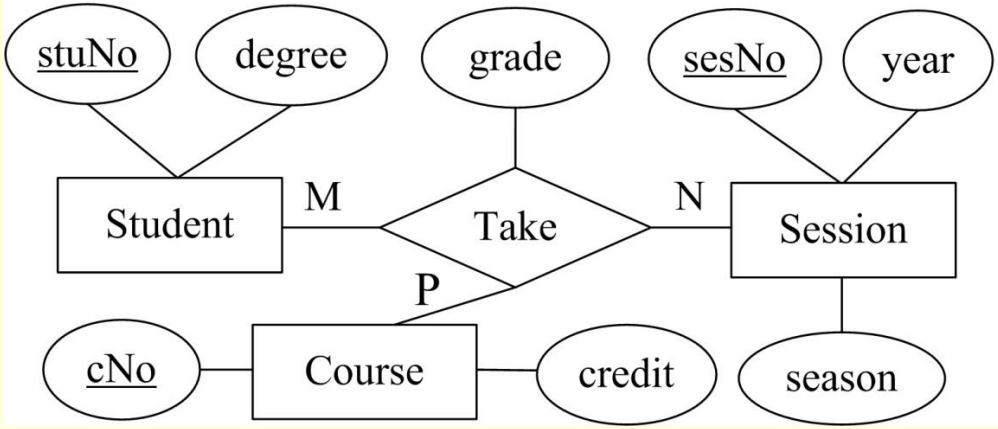
Particiate (pjNo, fNo, stDate)

Project (pjNo, budget)

模式个数尽量少

外键不取null值

M:N:P联系转换举例：



Student (stuNo, degree)

Course (cNo, credit)

Take (stuNo, cNo, sesNo, grade)

Session (sesNo, year, season)

11.4 逻辑设计

- EER图中特殊化（即：is-a关系）的转换：

回忆：特殊化（specialization）与普遍化（generalization）：

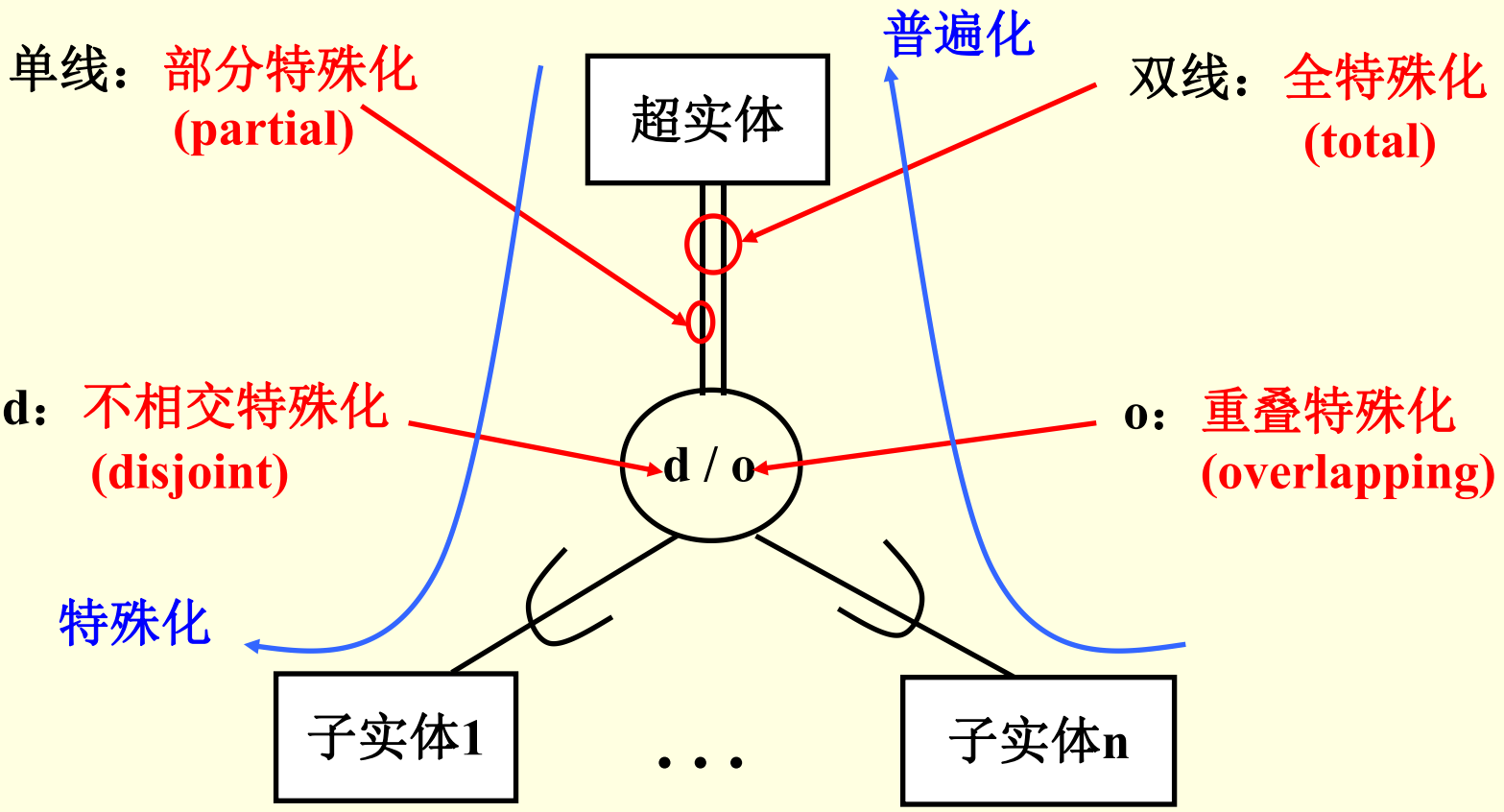
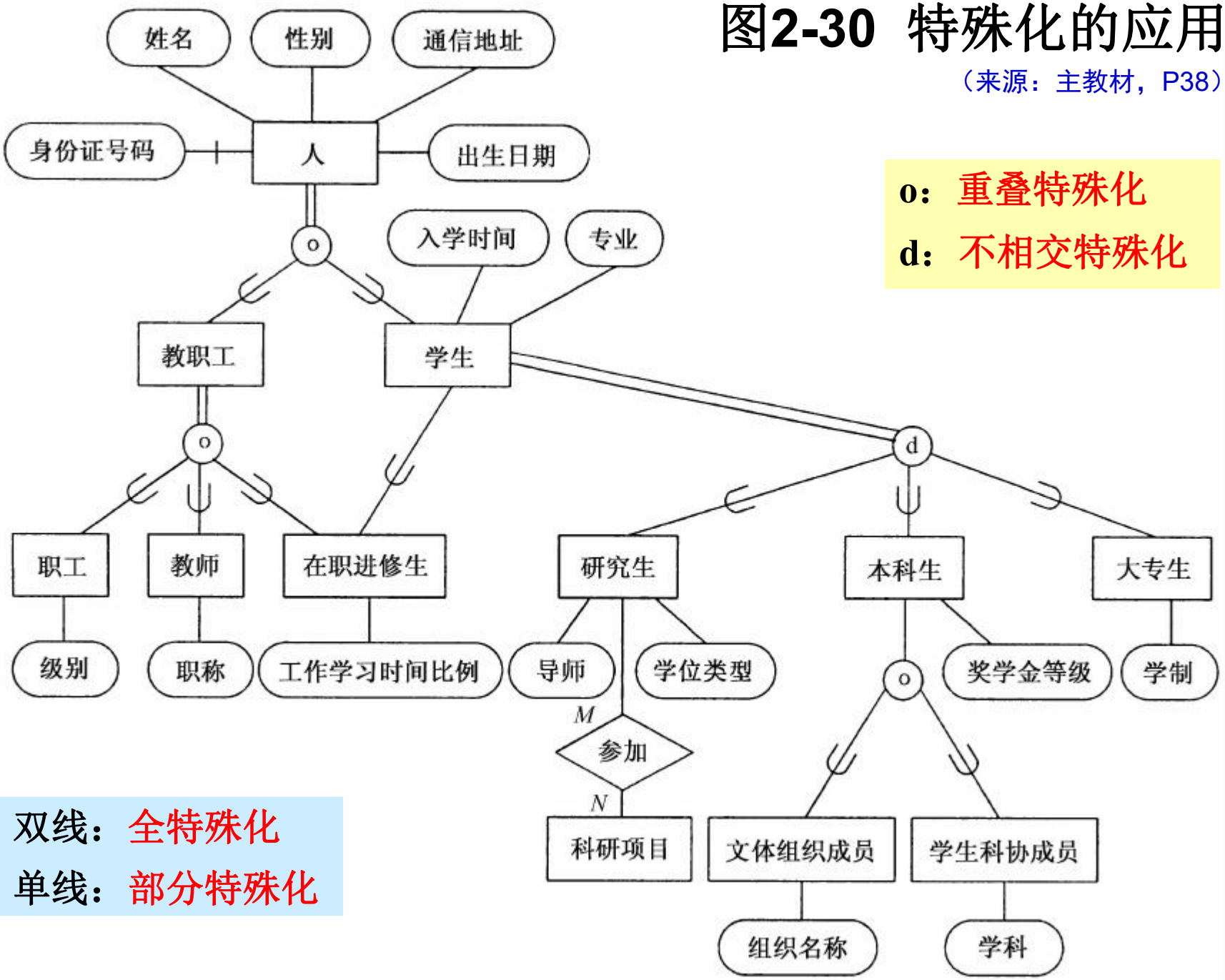


图2-30 特殊化的应用

（来源：主教材，P38）



o：重叠特殊化

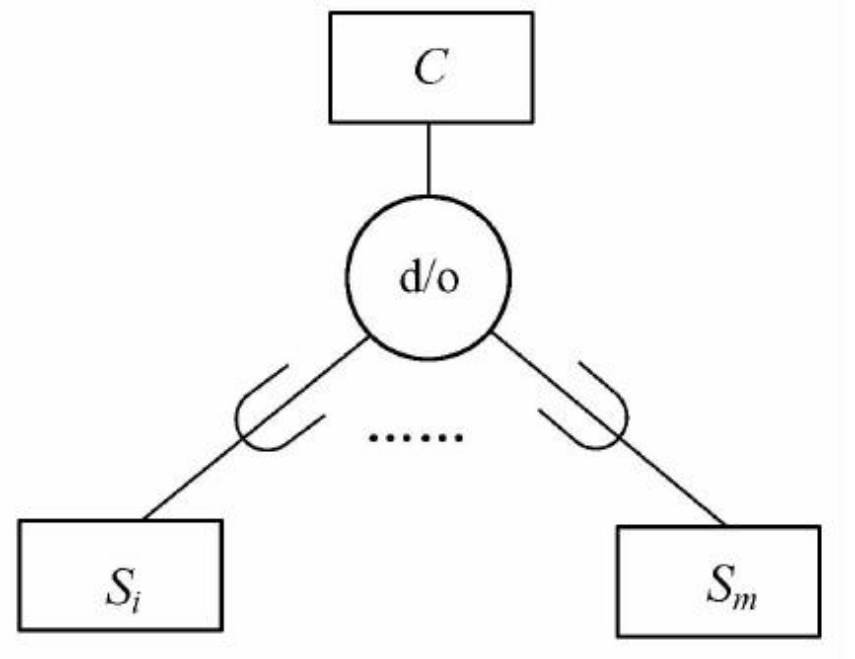
d：不相交特殊化

双线：全特殊化

单线：部分特殊化

11.4 逻辑设计

- **EER图中特殊化（即：is-a关系）的转换：**



设超实体集C的属性集是$\{\underline{k}, a_1, a_2, \cdots, a_n\}$

m个子实体集S_i的属性集是 $\text{Attr}(S_i), 1 \le i \le m$

方案一：$R(\underline{k}, a_1, a_2, \cdots, a_n)$

$R_i(\underline{k}, \text{Attr}(S_i)), 1 \le i \le m$

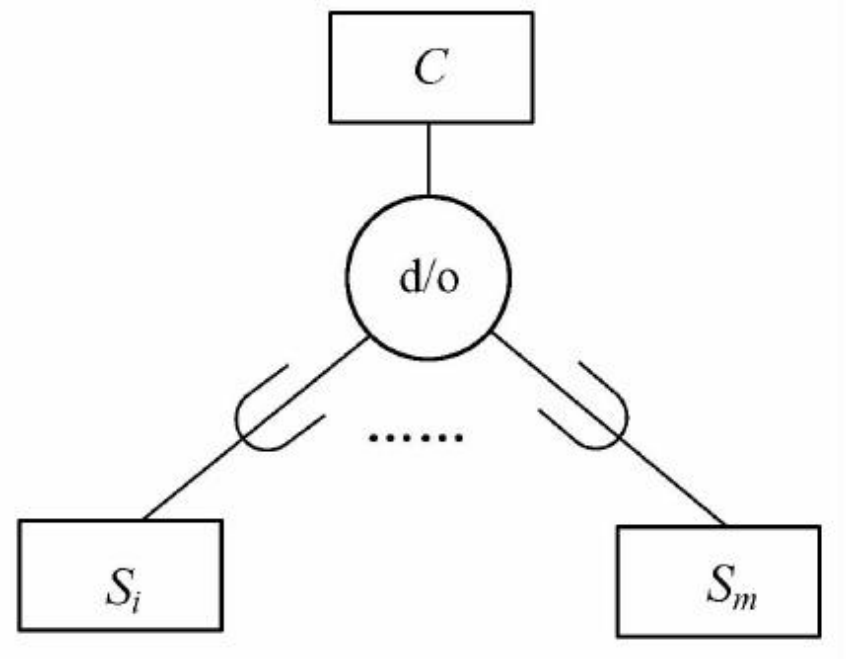
方案一的优点：
若不相交特殊化，则不存在数据冗余/更新异常。

缺点：
要查询到子实体集R_i的全部属性，如 $a_j, 1 \le j \le n$ ，需进行连接 $R \rhd\lhd_{R.k=R_i.k} R_i$

11.4 逻辑设计

教材中方案三、四存在很多null值，不常用！

- **EER图中特殊化（即：is-a关系）的转换：**



设超实体集C的属性集是$\{\underline{k}, a_1, a_2, \cdots, a_n\}$

m个子实体集S_i的属性集是 $\mathrm{Attr}(S_i),\ 1 \le i \le m$

方案二：（不相交且全特殊化）

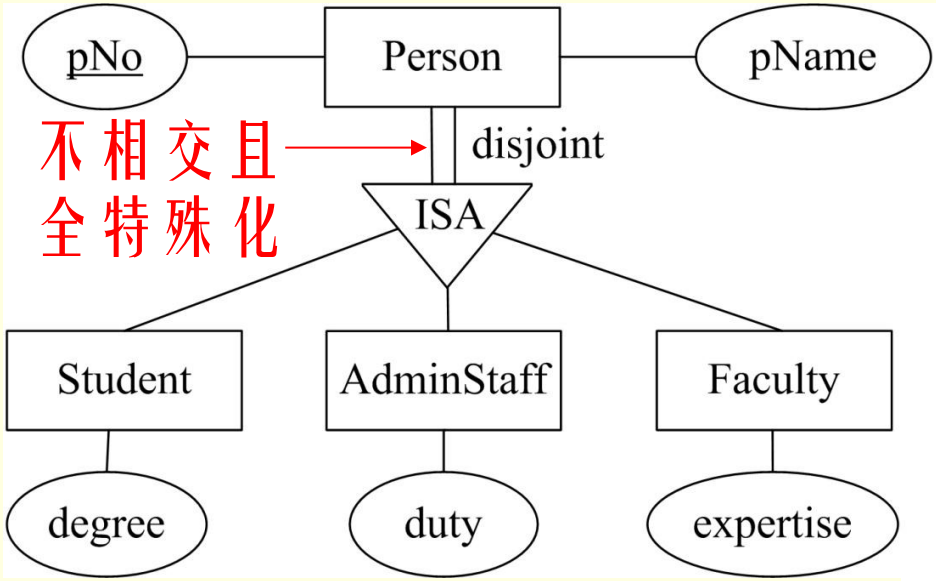
$R_i(\underline{k}, a_1, a_2, \cdots, a_n, \mathrm{Attr}(S_i)),\ 1 \le i \le m$

方案二的优点：
无需连接；模式个数少。
缺点：
$a_1, a_2, \cdots, a_n$ 数据冗余m次；仅适合于不相交（否则冗余）且全特殊化（否则丢失信息）的情形。

11.4 逻辑设计

- **EER图中特殊化（即：is-a关系）的转换**

举例



允许部分特殊

若重叠特殊化，则数据冗余

方案一：

Person (pNo, pName)

Faculty (fNo, expertise)

AdminStaff (AdminNo, duty)

Student (stuNo, degree)

方案二：

Faculty (fNo, pName, expertise)

AdminStaff (AdminNo, pName, duty)

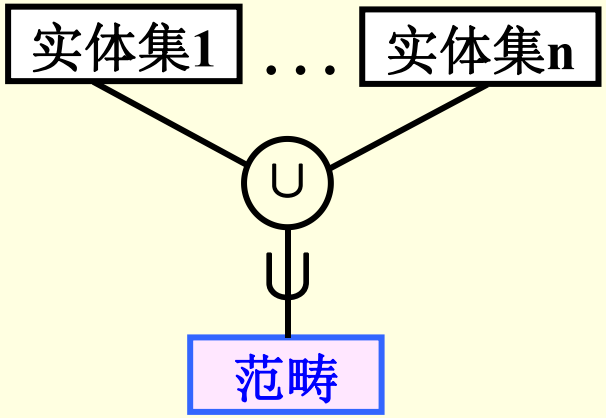
Student (stuNo, pName, degree)

若部分特殊化，则丢失Person信息
若重叠特殊化，则数据冗余

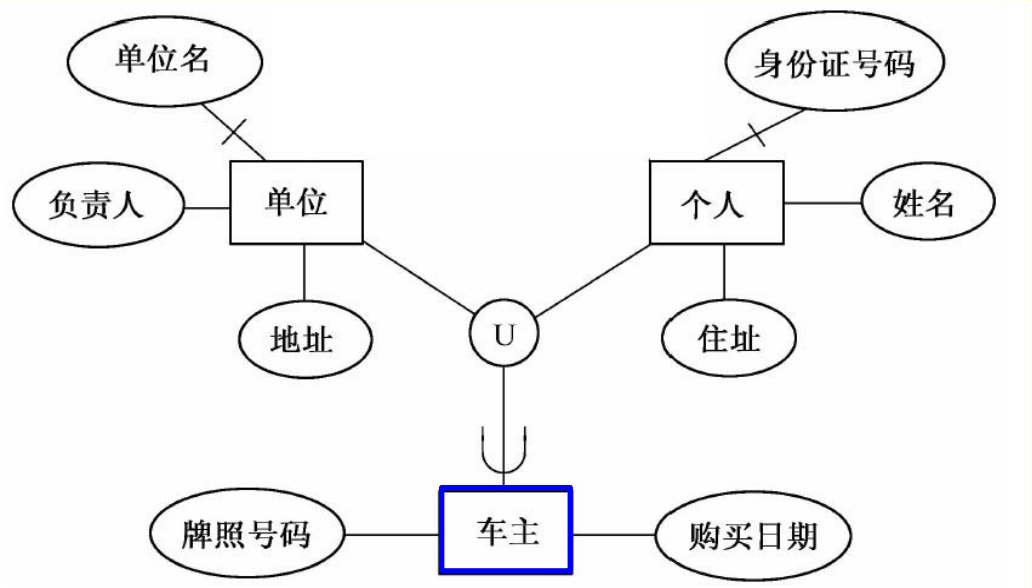
11.4 逻辑设计

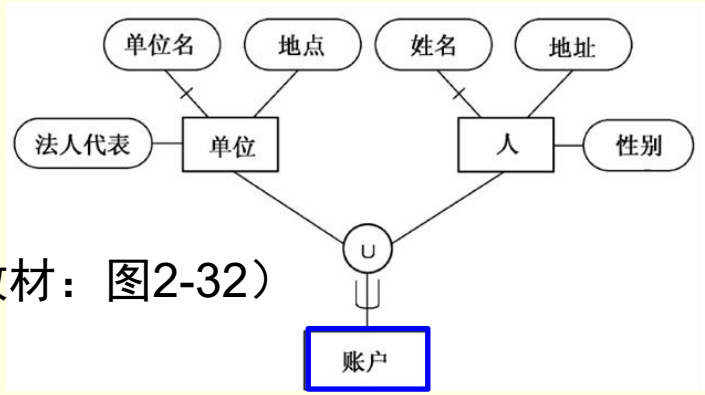
- **EER图中范畴的转换：**

回忆：范畴（category）——不同类型的实体组成的新实体。



举例：（主教材：图11-20）

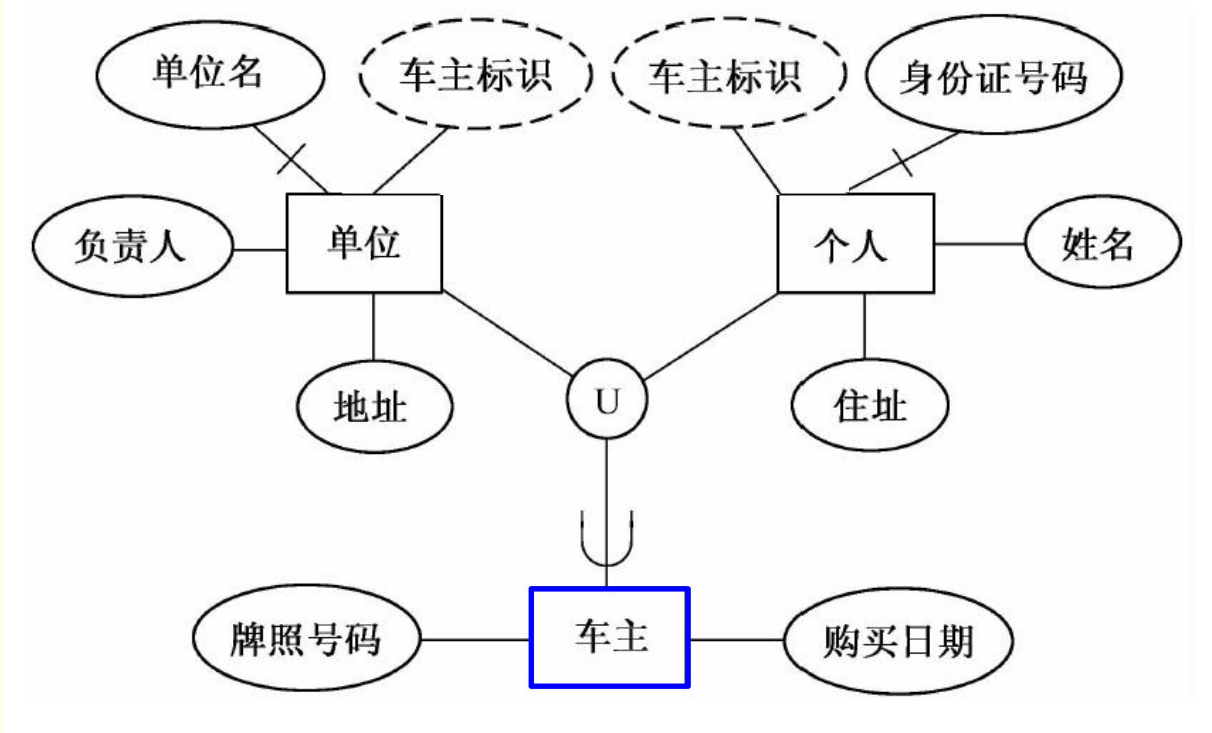




（主教材：图2-32）

11.4 逻辑设计

- **EER图中范畴的转换：**



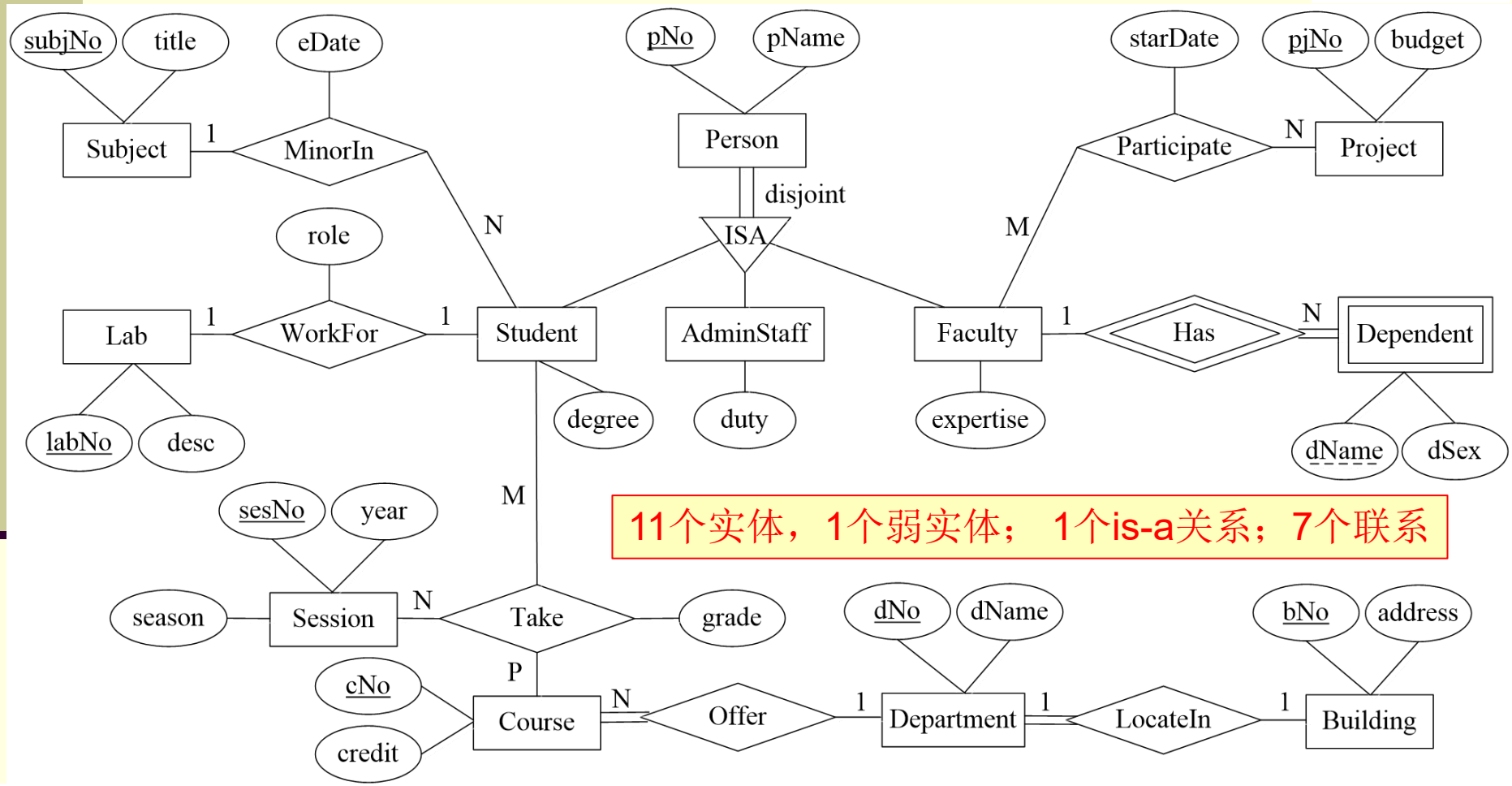
实体集的类型各异，范畴的键不宜直接使用实体集的键，需为实体集定义一个统一类型的键，称替身键（surrogate key），其相当于各个实体集的候补键。
如：“车主标识”

R_1(单位名，负责人，地址，车主标识)　（车主标识是候补键）
R_2(身份证号码，住址，姓名，车主标识)　（车主标识是候补键）
R_3(车主标识，购买日期，牌照号码)

11.4 逻辑设计

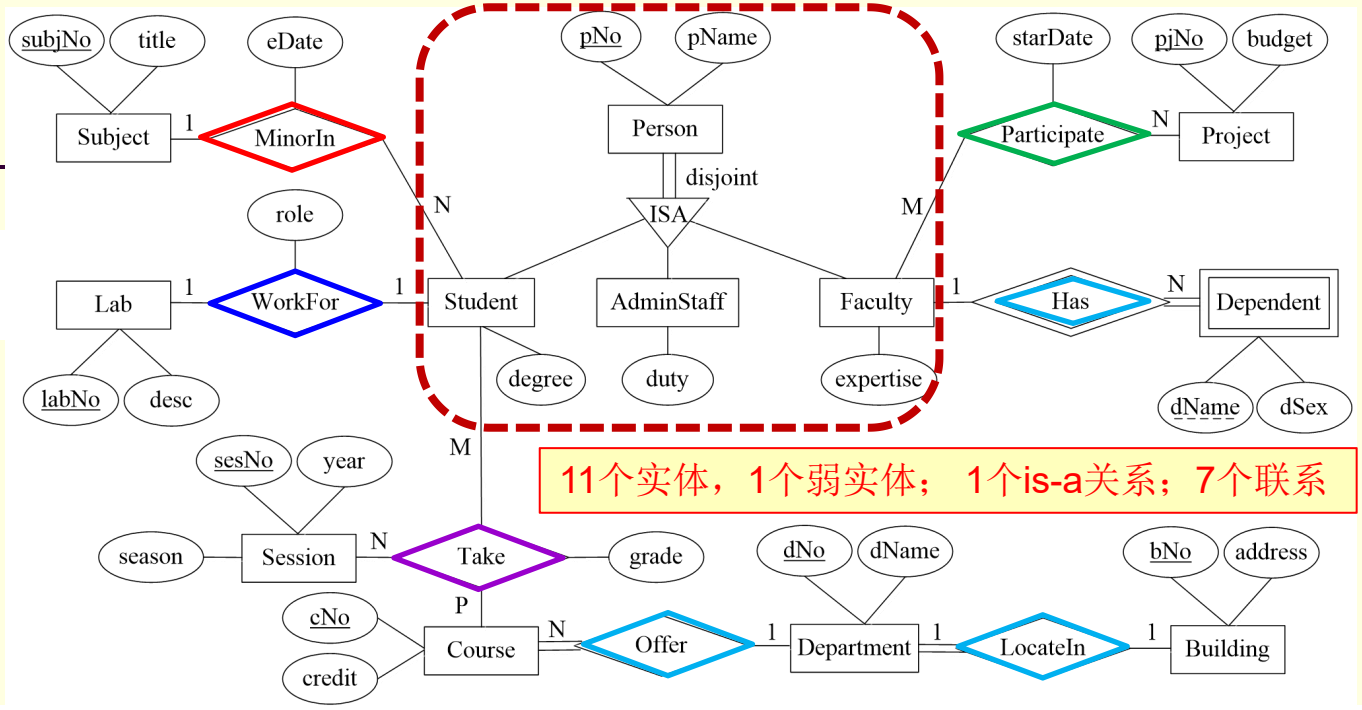
E-R向关系转换举例：

E-R图



转换：

前文E-R图



关系DB模式

Person (pNo, pName)

Subject (subjNo, title)

is-a关系

Project (pjNo, budget)

AdminStaff (adminNo, duty)

MinorIn (stuNo, subjNo, eDate)

Particiate (pjNo, fNo, starDate)

Session (sesNo, year, season)

Student (stuNo, degree)

Faculty (fNo, expertise)

Take (stuNo, cNo, sesNo, grade)

Has联系

WorkFor (stuNo, labNo, role)

Dependent (fNo, dName, dSex)

Course (cNo, credit, deptNo)

Lab (labNo, desc)

Offer联系

Building (bNo, address)

Department (dNo, dName, bNo)

LocateIn联系

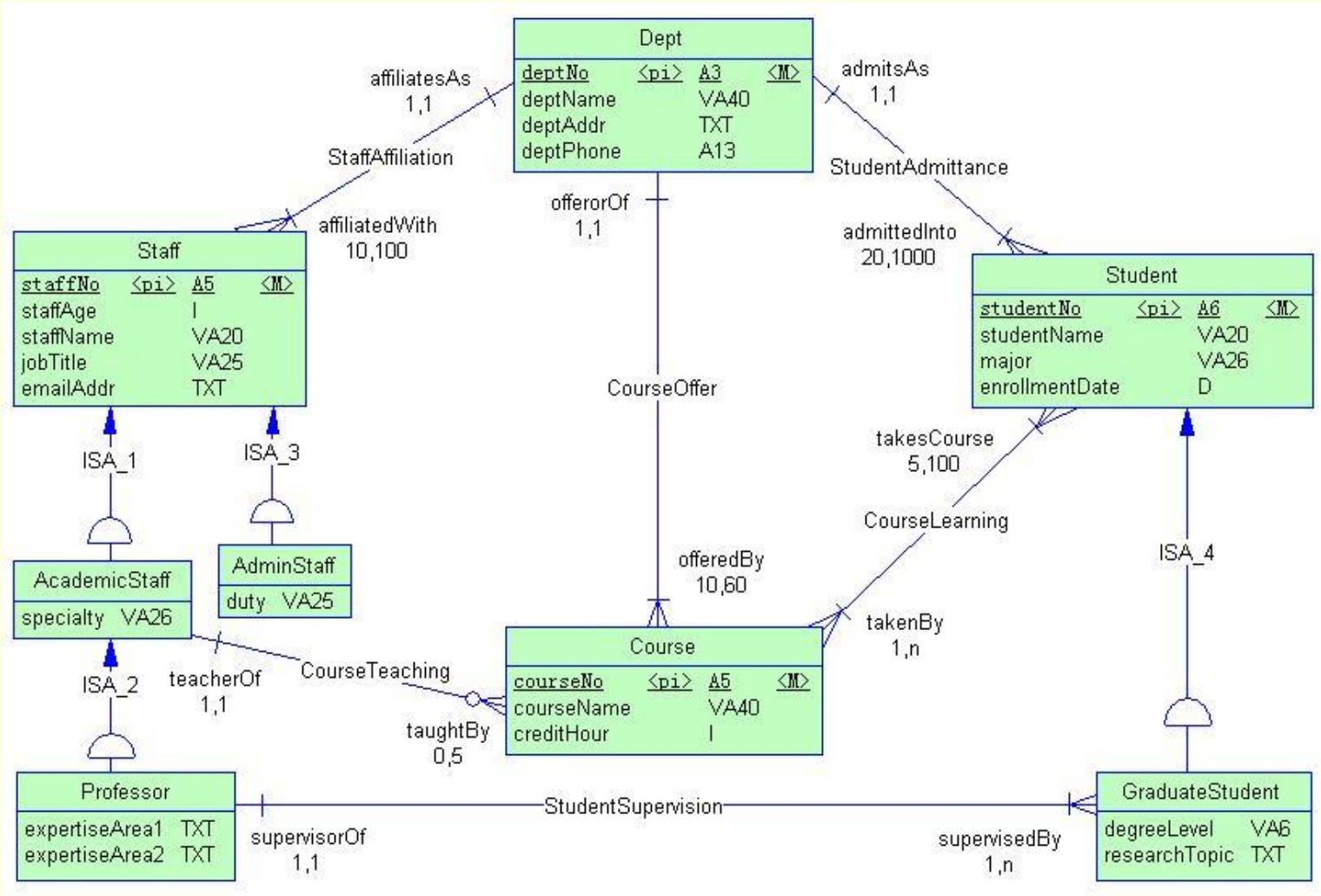
12个实体模式 + 4个联系模式 = 16个

11.4 逻辑设计

- 【补充】CASE工具（计算机辅助数据库设计）
  - **小型数据库：**可人工设计，但设计质量取决于设计人员的技术、经验和对应用业务的熟悉程度。
  - **大型数据库：**人工设计效率低、周期长，难以保证质量，故普遍借助支持数据库设计全过程的计算机辅助软件工程（CASE）工具。
- 主要CASE工具有：
  - 现erwin, Inc的erwin Data Modeler (DM)
  - 原Rational公司的Rational Rose（IBM公司收购后成为IBM Rational Rose XDE）
  - 原Sybase现SAP的PowerDesigner
  - Oracle公司的Oracle Designer

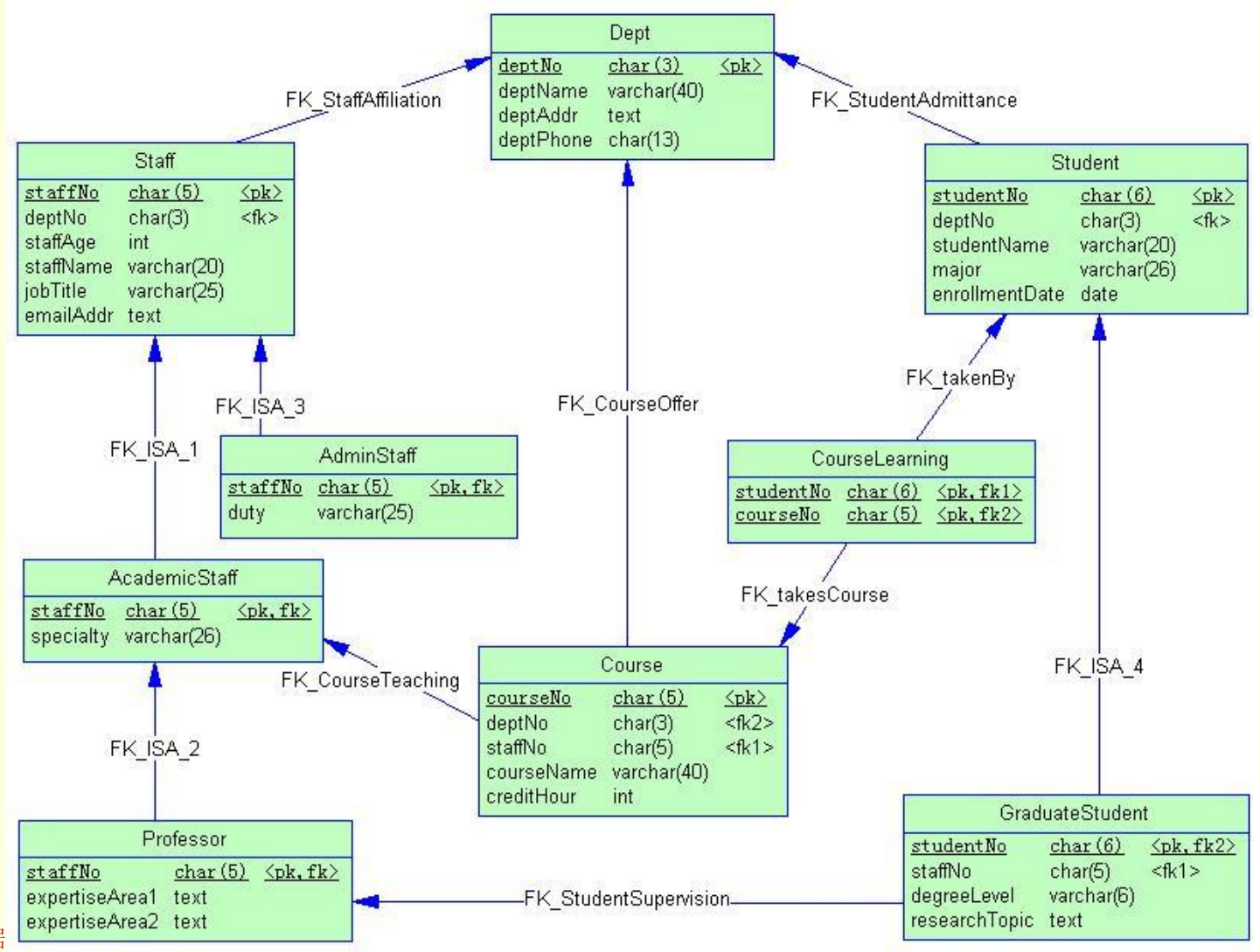
数据库设计CASE工具举例

- 用PowerDesigner 10.0创建的E-R图：



数据库设计CASE工具举例

- 用PowerDesigner将此E-R图转换成PDM图：



数据库设计CASE工具举例

- PDM图对应的RDB模式（下划线：PK；斜体：FK）

数据库设计CASE工具举例

- 用PowerDesigner产生MySQL 4.0 DDL语句：

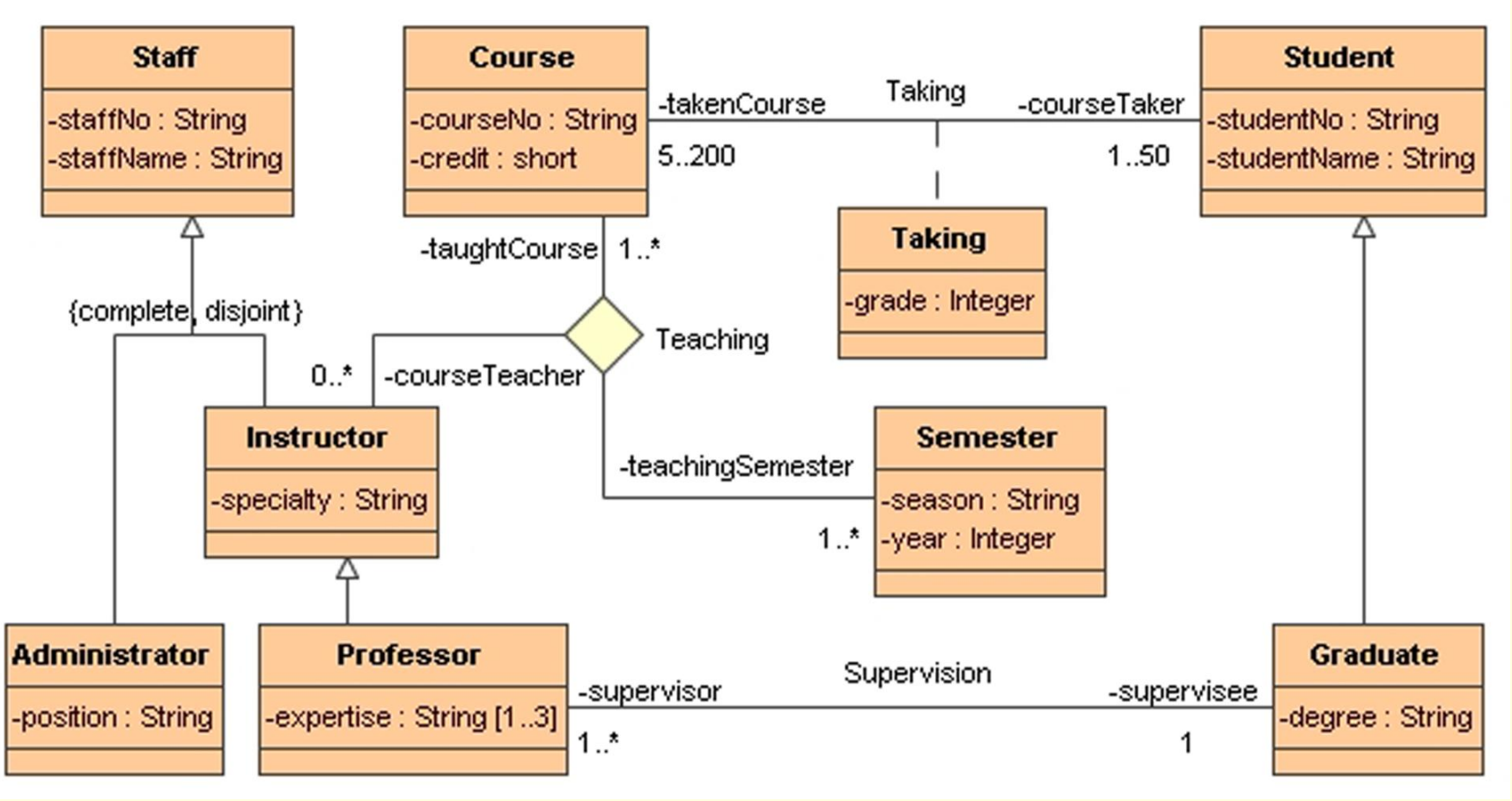
```
/*==============================================================================*/
/* DBMS name:      MySQL 4.0                                                    */
/* Database:       univdept                                                     */
/* Created on:     2005-6-18 17:32:21                                           */
/*==============================================================================*/
DROP DATABASE IF EXISTS univdept;
CREATE DATABASE univdept;
USE univdept;

CREATE TABLE Dept (
   deptNo          char(3)  NOT NULL,
   deptName        varchar(40),
   deptAddr        text,
   deptPhone       char(13),
   PRIMARY KEY (deptNo)
) TYPE = InnoDB;

CREATE TABLE Staff (
   staffNo         char(5)  NOT NULL,
   deptNo          char(3)  NOT NULL,
   staffAge        int,
   staffName       varchar(20),
   jobTitle        varchar(25),
   emailAddr       text,
   PRIMARY KEY (staffNo),
   INDEX StaffAffiliation_FK (deptNo),
   CONSTRAINT FK_StaffAffiliation FOREIGN KEY (deptNo)
     REFERENCES Dept (deptNo) ON DELETE RESTRICT ON UPDATE CASCADE
) TYPE = InnoDB;
```

数据库设计CASE工具举例

- 用No Magic, Inc.’s MagicDraw UML Community Edition v11.5创建的UML类图：



11.4 逻辑设计

11.4.3 关系模式的规范化（第10章已学过）

- 如果E-R图中实体的属性对键不存在部分依赖和传递依赖，那么按前述规则由E-R图转换而得的关系模式一般就不需要再规范化了。

规范化知识回顾：

- **规范化：**一个关系模式按有关范式的要求**分解**成多个关系模式的过程。
- **模式分解有两种准则：(1)** 只满足无损分解（lossless decomposition）要求； **(2)** 既满足无损分解，又满足保持依赖（preserving dependencies）
- **理论研究结论：**总有将一个关系模式规范化成3NF的无损且保持依赖的分解；总有将一个关系模式规范化成BCNF的无损分解。

Q & A:

- **无损分解：**要求分解前后的关系模式是等价的，即对任何相同的查询总是产生相同的查询结果。——可通过**连接**分解后的诸关系来重构原关系。
- **保持依赖分解：**要求分解后的关系模式中的函数依赖仍然能**逻辑蕴涵**原关系模式中的函数依赖。——是一种理想的模式分解。

11.4 逻辑设计

规范化知识回顾：

● **规范化与数据库性能间的关系：**模式分解总体上会降低数据访问性能，因此模式规范化程度并非越高越好。规范化与数据库性能之间需要权衡。一般情况下：

(1) 对于更新频繁、查询较少的关系模式（基表），应提高规范化程度，以减少更新异常；

(2) 对于查询频繁、更新较少（或无更新）的关系模式（基表），通常可降低规范化程度，以提高数据访问的性能。

● **数据库的性能与数据库的物理设计（在下一小节介绍）关系十分密切，但数据库的逻辑设计（如：模式规范化）对数据库的性能也有较大影响。**

11.4 逻辑设计

11.4.4 模式的调整

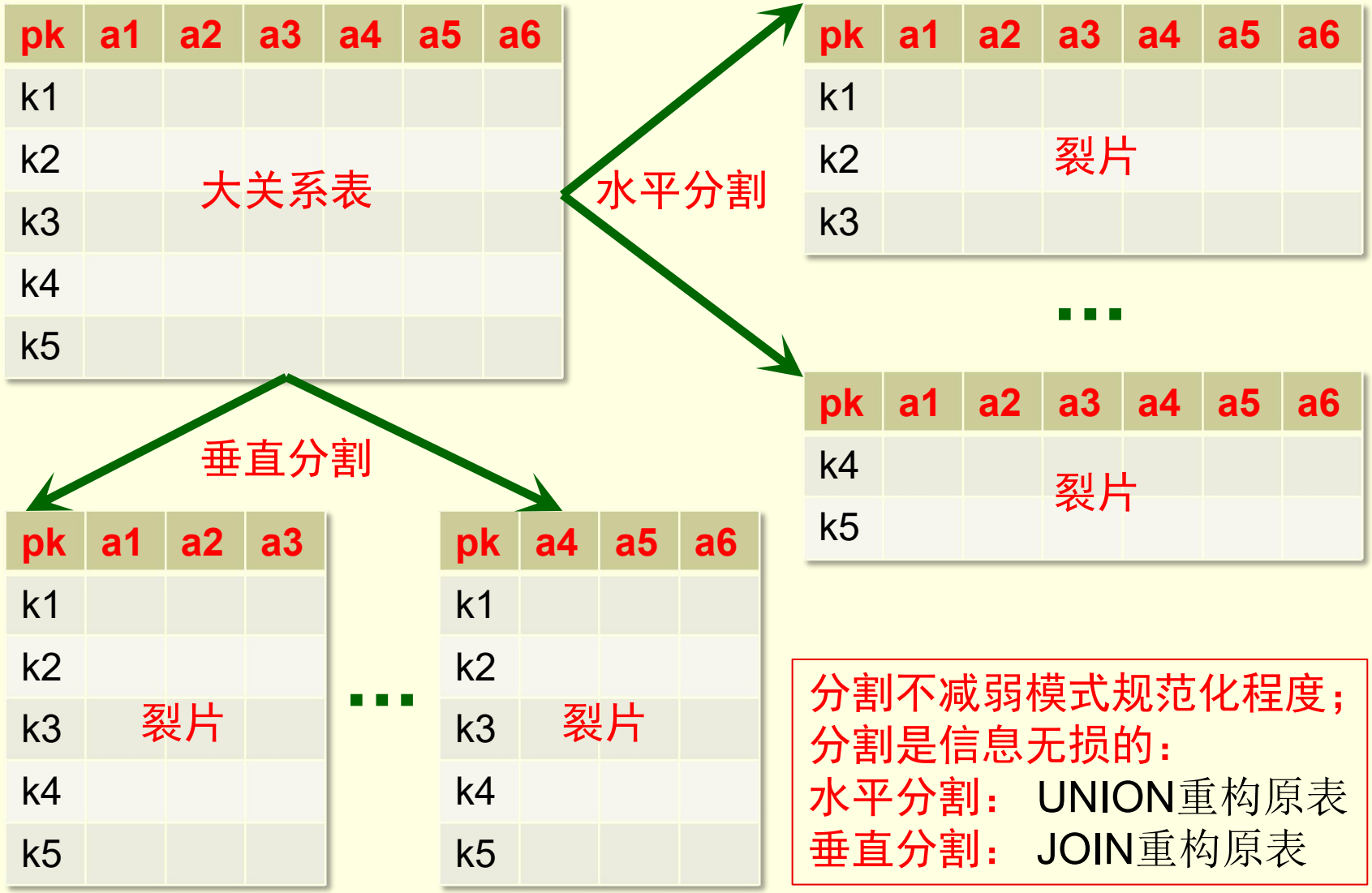
- **(1) 为改善DB性能的调整：**
  - ✓ **减少连接运算：**对关系模式进行逆规范化（denormalization）
  - ✓ **减小表的大小：**对大关系表进行水平/垂直分割（segmentation）
  - ✓ **尽量使用快照：**快照（snapshot）是一个关系表的定期刷新的（本地）只读副本

11.4 逻辑设计

逆规范化（denormalization）：

- 连接运算的代价很大，严重影响DB查询性能。
- 逆规范化是规范化（模式分解）的逆过程，即将规范化程度较高的（如BCNF/3NF）几个关系模式合并为一个规范化程度较低的（如3NF/2NF/1NF）关系模式。——这相当于提前连接好几个关系表，以减少查询时的连接运算。
- 逆规范化会导致数据冗余，易产生更新异常。故:
  - ✓ 对于更新频繁的表，其关系模式不宜进行逆规范化；
  - ✓ 对于不经常更新的表，其关系模式在逆规范化后可通过完整性约束维护机制（例如，触发器）来防止发生更新异常。

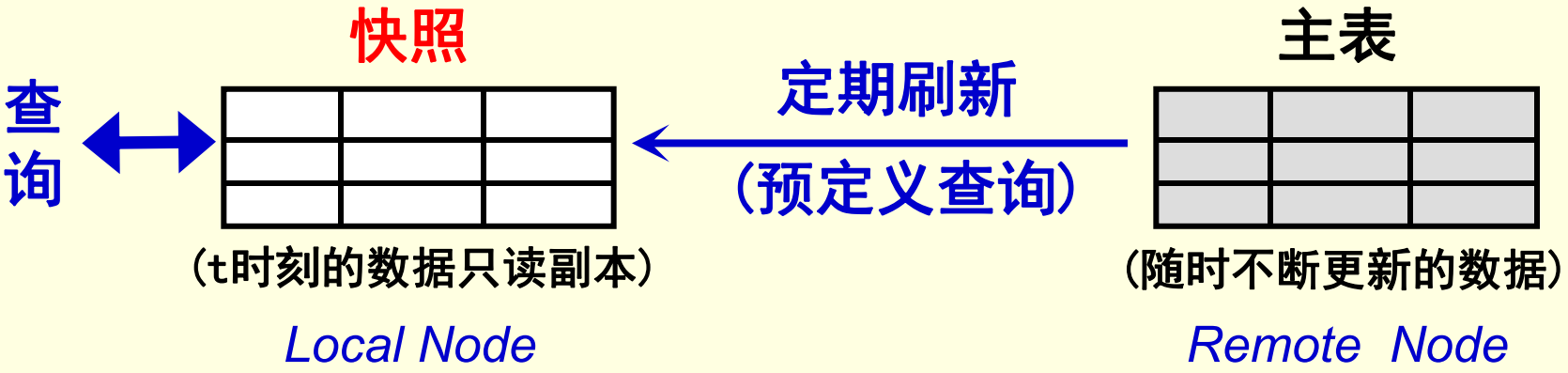
11.4 逻辑设计



11.4 逻辑设计

快照（snapshot）：

- 快照：是一个关系表的定期刷新的（本地）只读副本。
- 许多应用只需访问某个时间点的数据值，而并不需要数据的最新当前值。例如：周/月/季/年度报表。
- 这类应用可访问快照——需周期性地刷新快照中的数据。
- 快照还可在本地建立，以降低远程访问的网络通信代价。



11.4 逻辑设计

- **(2) 为节省存储空间的模式调整：**
  - ✓ 减少属性值的长度：
    用编码代替属性值；用缩写代替全称。
  - ✓ 减少重复数据所占存储空间：
    用长度短的假属性（dummy）值来代表长度较长的重复属性值。

均是牺牲查询时间来换取存储空间节省

设某关系模式中存在函数依赖 $A \rightarrow B$，如果B的取值（即基数$|B|$）较少，但每个值的长度较长，而A的取值重复较多，那么B的同一个值就有可能在多个元组（表行）中重复出现（即会占用很大的存储空间）。
此时可引入一个存储编码/缩写的属性或假属性C，显然，函数依赖为：$(A \rightarrow C, C \rightarrow B)$。让$A \rightarrow C$的属性值存储在原关系表中，而$C \rightarrow B$的属性值存储在新关系表中。

11.4 逻辑设计

学号	姓名	奖学金(*A*)	家庭经济状况(*B*)
1	张1	2000	人均月收入为300-1000元，无其他经济来源
2	张2	2000	人均月收入为300-1000元，无其他经济来源
3	张3	2000	人均月收入为300-1000元，无其他经济来源
4	李1	1800	人均月收入为300-1000元，有其他经济来源
5	李2	1800	人均月收入为300-1000元，有其他经济来源
6	王1	1500	人均月收入为1000-2000元，无其他经济来源
7	王2	1500	人均月收入为1000-2000元，无其他经济来源
8	王3	1500	人均月收入为1000-2000元，无其他经济来源
9	赵1	1000	人均月收入为1000-2000元，有其他经济来源
10	赵2	1000	人均月收入为1000-2000元，有其他经济来源
11	刘1	800	人均月收入为2000-5000元，无其他经济来源
12	刘2	800	人均月收入为2000-5000元，无其他经济来源
13	刘3	800	人均月收入为2000-5000元，无其他经济来源
14	江1	500	人均月收入为2000-5000元，有其他经济来源
15	江2	500	人均月收入为2000-5000元，有其他经济来源
16	万1	300	人均月收入为5000-10000元，无其他经济来源
17	沈1	200	人均月收入为5000-10000元，有其他经济来源
18	曹1	100	人均月收入为10000元以上，无其他经济来源
19	白1	0	人均月收入为10000元以上，有其他经济来源

$A \rightarrow B$

*A*值重复

属性*B*的大量重复值浪费存储空间

11.4 逻辑设计

学号	姓名	奖学金(*A*)	收入(*C*)
1	张1	2000	1A
2	张2	2000	1A
3	张3	2000	1A
4	李1	1800	1B
5	李2	1800	1B
6	王1	1500	2A
7	王2	1500	2A
8	王3	1500	2A
9	赵1	1000	2B
10	赵2	1000	2B
11	刘1	800	3A
12	刘2	800	3A
13	刘3	800	3A
14	江1	500	3B
15	江2	500	3B
16	万1	300	4A
17	沈1	200	4B
18	曹1	100	5A
19	白1	0	5B

$A \rightarrow B$

引入假属性*C*后
可节省存储空间

$(A \rightarrow C,\ C \rightarrow B)$

但要查询某学生实际家庭经济状况*B*时需两表JOIN操作
(牺牲时间代价)

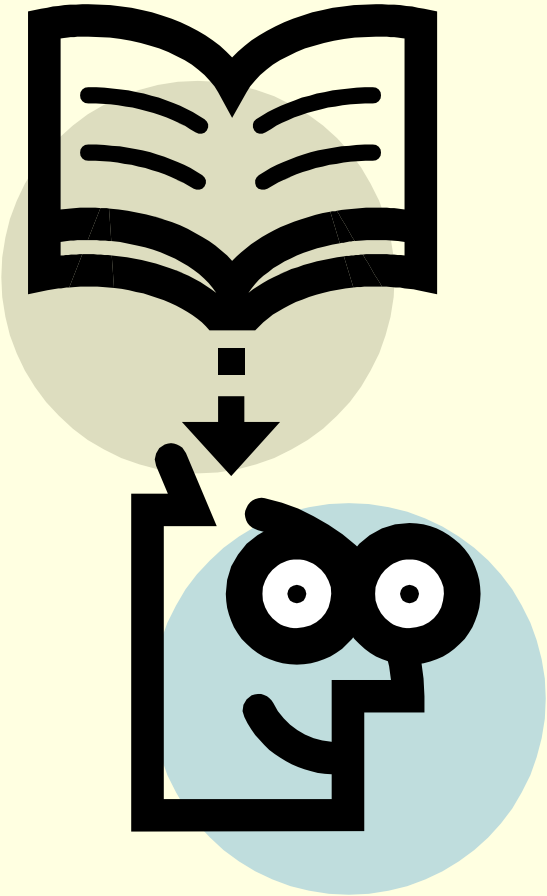
收入(*C*)	家庭经济状况(*B*)
1A	人均月收入为300-1000元，无其他经济来源
1B	人均月收入为300-1000元，有其他经济来源
2A	人均月收入为1000-2000元，无其他经济来源
2B	人均月收入为1000-2000元，有其他经济来源
3A	人均月收入为2000-5000元，无其他经济来源
3B	人均月收入为2000-5000元，有其他经济来源
4A	人均月收入为5000-10000元，无其他经济来源
4B	人均月收入为5000-10000元，有其他经济来源
5A	人均月收入为10000元以上，无其他经济来源
5B	人均月收入为10000元以上，有其他经济来源

11.4 逻辑设计

11.4.5 外模式的设计

- 外模式：描述数据的不同（用户）视图
- 外模式的设计依据：局部E-R模式（即用户视图）
- 外模式的SQL实现：定义视图 + 限制基表访问
  - ✓ CREATE VIEW … AS SELECT … WHERE …;
  - ✓ GRANT/REVOKE ... [ON ...] TO/FROM ...;
  - ✓ 视图的作用：
    - (1) 提供了一定的逻辑数据独立性
    - (2) 更好地适应不同用户对数据的特定需求
    - (3) 屏蔽某些数据，有利于数据保密
    - (4) 视图上查询可简化用户的查询操作

目录 Contents



11.5 物理设计

11.5.1 概述

- 任务：

  设计数据库的内（存储）模式，包括：选择合适的存储机制（存储结构和存取方法），合理安排存储空间。

- 目标：(1) 提高DB的查询性能（乃主要目的！）

  (2) 有效利用存储空间 （有时与(1)矛盾）

- 技术：

  ✓ 存储机制（存储结构和存取方法）的选择：

  无簇表：(1) 普通表　　簇表：(3) 散列簇表

  (2) 索引的表　　(4) 索引簇表

  ✓ 分区设计：

  数据在多个磁盘上合理分布，以合理地安排I/O，从而提高DB性能。

11.5 物理设计

11.5.2 存储机制的选择

- **索引的选择：**
  - ✓ **建立：**

    一般，PK及UNIQUE属性列上的索引是由DBMS自动建立的；

    其他属性列上的索引需用户（一般DBA）手工建立。
  - ✓ **使用：**

    索引的使用由DBMS自动决定，即对用户是透明的。

11.5 物理设计

- **下列情况，宜在有关属性列上建索引：**
  ——即选用(2)索引的表

  ① 常用来连接（JOIN）相关表的列上宜建索引。

  常用的连接是通过PK/FK来实现的，PK列上一般已由DBMS自动建立了索引，因此，最好在FK 列上也建索引（假如对FK列的更新不是太频繁的话）。

  ② 以读为主的表，其常用查询涉及的列上宜建索引。

  假如范围查询满足条件的行数 ≤ 20%的表行，等值查询满足条件的行数 ≤ 5%的表行，且查询条件不是IS NOT NULL的话。

  ③ 对于只需访问索引块而不需访问数据块的查询是常用查询的表，其相应列上宜建索引。

  (1) 查询某属性值的MIN, MAX, AVG, SUM, COUNT等聚集函数值，且查询语句没有GROUP BY子句；

  (2) 查询某属性值是否存在（即：使用EXISTS或NOT EXISTS谓词）。

11.5 物理设计

- **下列情况，不宜在有关属性列上建索引：**
  ——即选用(1)普通表，除非可建簇集（见后面）

① 很少出现在查询条件中的列。（没意义）

② 属性取值很少的列。（没效果，e.g. 性别）

③ 数据太少的表。（没必要，全表扫描很快）

④ 属性值分布严重不均匀的列。（反而不如全表扫描）

⑤ 经常频繁更新的表/列。（索引维护开销大，得不偿失）

⑥ 属性值过长的列。（难以建索引，如：Oracle规定不能在LONG或LONG RAW类型的列上建索引）

11.5 物理设计

- **簇集的选择：**
  - ✓ **建立：**

    簇集需用户（一般是DBA）手工建立。
  - ✓ **使用：**

    簇集的使用由DBMS自动决定，即对用户是透明的。

    （实际中常用散列簇集，较少用索引簇集）

11.5 物理设计

- 下列情况，宜在有关属性列上建簇集：
  ① 更新较少的表，且对其只进行等值查询时，宜建散列簇集。
  ——即选用(3)散列簇表
  ② 更新较少的表，且对其既进行等值查询又进行范围查询时，可建索引簇集。
  ——即选用(4)索引簇表

- 下列情况，不宜在有关属性列上建簇集：
  ① 更新频繁的表，则宁可选用(1)普通表。
  ② 更新较少的表，但对其只进行范围查询时，则宁可用(2)索引的表。

11.5 物理设计

11.5.3 分区设计的原则

数据库的存储介质一般由多个磁盘阵列所构成。数据在磁盘阵列上的分布（即分区设计 / Partition Design）也是数据库物理设计的内容之一。一般原则是：

- 为了减少访盘冲突、提高I/O并行性，可水平分割（大）关系，将多个裂片分散到多个磁盘阵列中；
- 为了均衡磁盘阵列的I/O负荷，尽可能地使热点数据（hotspot data）分散到多个磁盘阵列中。
- 保证关键数据（如数据字典）的快速访问，缓解系统访问瓶颈。

例如，在Oracle系统中，通常将一个数据库划分为一个或多个称为表空间（table space）的逻辑存储单元，一个表空间所对应的一个或多个数据文件可物理存储于不同的磁盘阵列上，通过这样的方式可简单地实现分区设计。

The End

- 【补充题】试对四个实体：学生、班级、课程、教师，及它们之间的四个联系：组成、选课、任课、班导师进行E-R建模，然后把E-R数据模式（E-R图）转换成关系数据库模式，并写出创建相应基表的SQL DDL语句，同时定义PK与FK、以及你认为必要的其他完整性约束。
  【本课程的最后一道作业题！**请在截止时间（12月5日23:59）之前提交答案！**】
- 建议用Microsoft Visio来画E-R图