《系统分析与设计》
System Analysis and Design

任课教师：范国祥

电　　话：0451-86418876-811(O)

13199561265(微信同号)

邮　　箱：fgx@hit.edu.cn

哈工大计算学部/

国家示范性软件学院

软件工程教研室

2022.03

数据库设计

本章主要内容

1. 数据库系统及关系数据库简介
2. 数据库逻辑模型设计
3. ERD模型及质量评价
4. 物理数据库设计及建立
5. 物理数据库提高效率的技巧
6. 描述分布式数据库的不同的体系结构模型
7. OO实体类图映射到ERD
8. 关系型数据库规范化示例

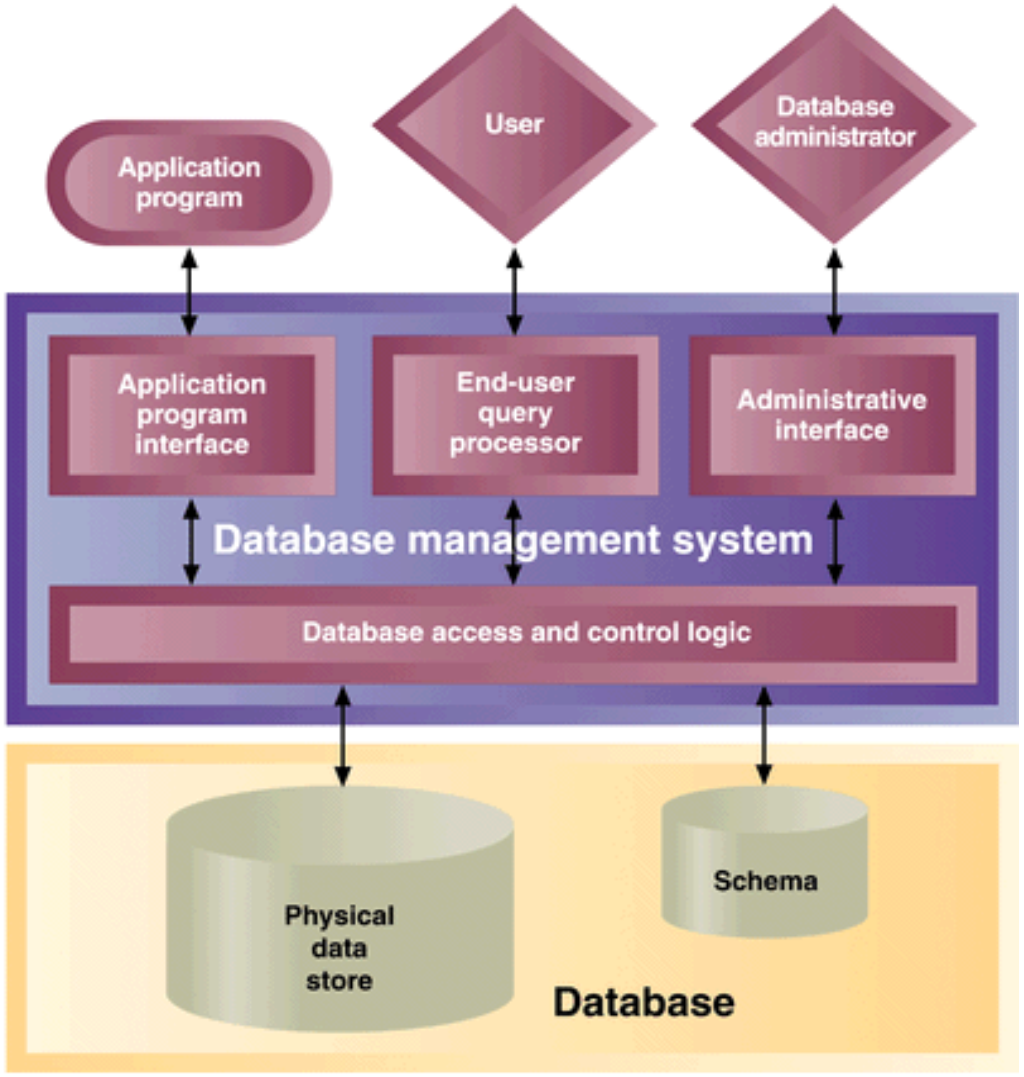
本章主要内容

1. 数据库系统及关系数据库简介
2. 数据库逻辑模型设计
3. ERD模型及质量评价
4. 物理数据库设计及建立
5. 物理数据库提高效率的技巧
6. 描述分布式数据库的不同的体系结构模型
7. OO实体类图映射到ERD
8. 关系型数据库规范化示例

数据库设计

数据库系统＝数据库(DB)＋数据库管理系统(DBMS)

- **数据库(DB)：被集中控制和管理的存储数据的完整集合**
  - 用ERD模型来描述
  - 以表（Table）对应ERD中的数据实体（Entity），是数据管理的基本单元
- **数据库管理系统(DBMS)：对数据库的访问进行管理和控制的软件**
  - 管理数据的逻辑格式定义、存储和访问
  - 允许多个用户或应用程序并发访问数据库
  - 支持通过SQL语言访问数据库



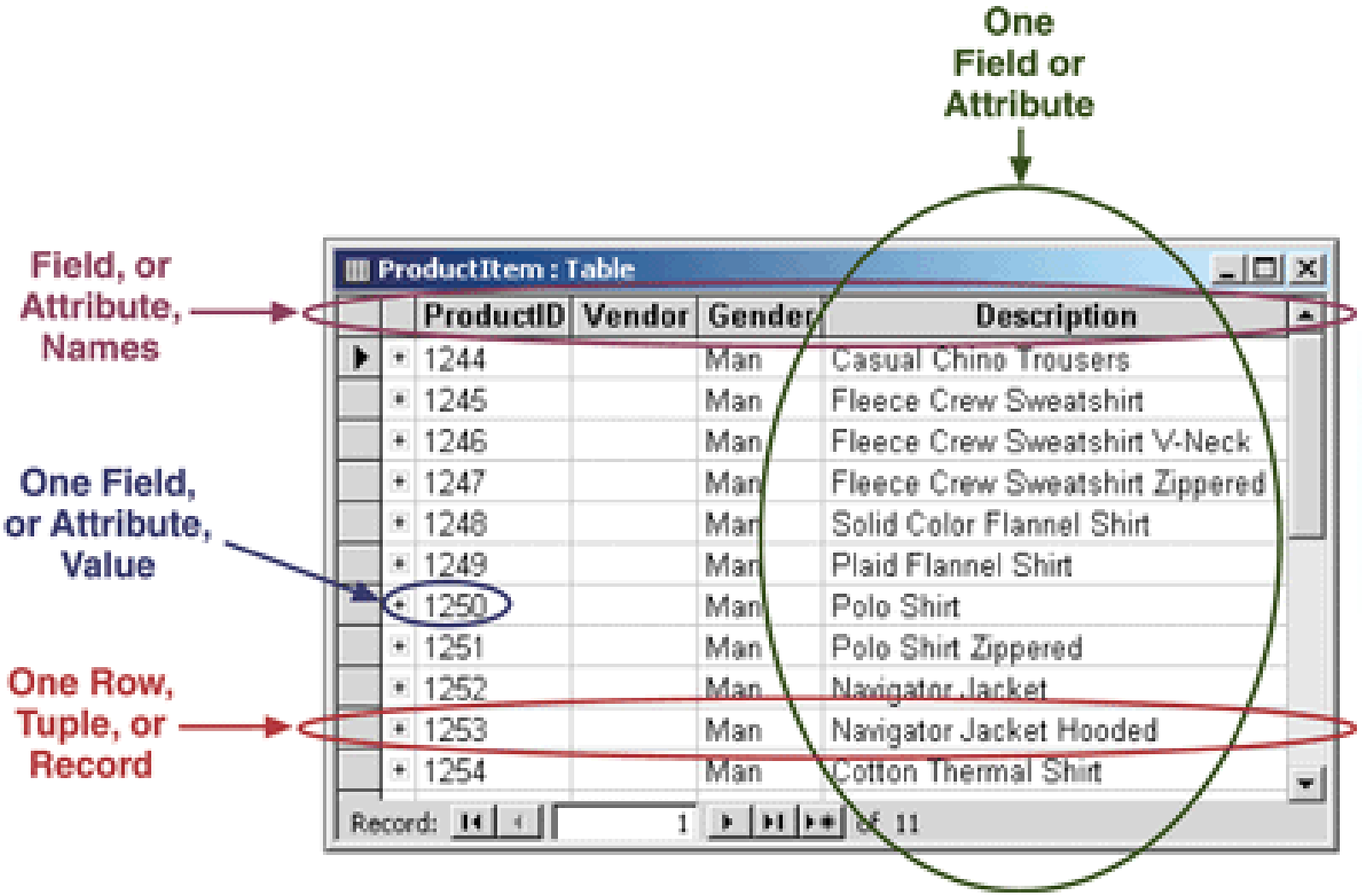
数据库系统体系结构

关系数据库

- 关系型数据库管理系统将数据存储成表和关系的结构
- 表（Table）:关系型数据库管理数据的基本单元，类似于Excel中的Sheet，与ERD模型中的实体（Entity）对应
  - 元组: 表中的行，习惯称为“记录”
  - 域:　表中的列，习惯称为“属性”
- 每个表中必须包含能唯一标识“记录”的属性或属性组，称为关键属性（Key Fields），相当于记录的索引关键词
- 数据库的表（Table）之间的关系通过Key Fields来建立
  - 主键（Primary Key）：本表中的关键属性
  - 外键（Foreign Key）：存储在本表中的其他表的关键属性，用来与其他表建立关联关系（相当于指针）

数据库设计

关系数据库：数据表（Table）的构成



数据库设计

数据库设计的根本任务

- 获取未来系统需要存储的数据实体（一定粒度的数据单元）
- 分析数据实体的内涵，以及数据实体之间的关系
- 设计物理ERD，即用于数据库系统实现的数据表结构的蓝图

数据库设计的基本过程

- 概念ERD设计 – 主要用于需求分析过程，从宏观上识别业务过程和系统层面所存在的数据实体，分析它们之间的关系。概念 ERD数据模型仅需给出存在的数据实体以及基本关系，不关注其他细节。
- 逻辑ERD设计 – 逻辑 ERD 是概念 ERD 的详细版本，明确定义每个实体中的属性，识别实体之间的关系基数，并引入关联实体，但不关注物理实现细节。
- 物理ERD设计 – 物理 ERD 是数据库的实际设计蓝图，将ERD的数据实体转化为数据的数据表（Table），针对具体的数据库系统，定义Table属性的细节（包括属性类型、长度、可否为空等）以及增加用于表达Table之间关系的新属性。

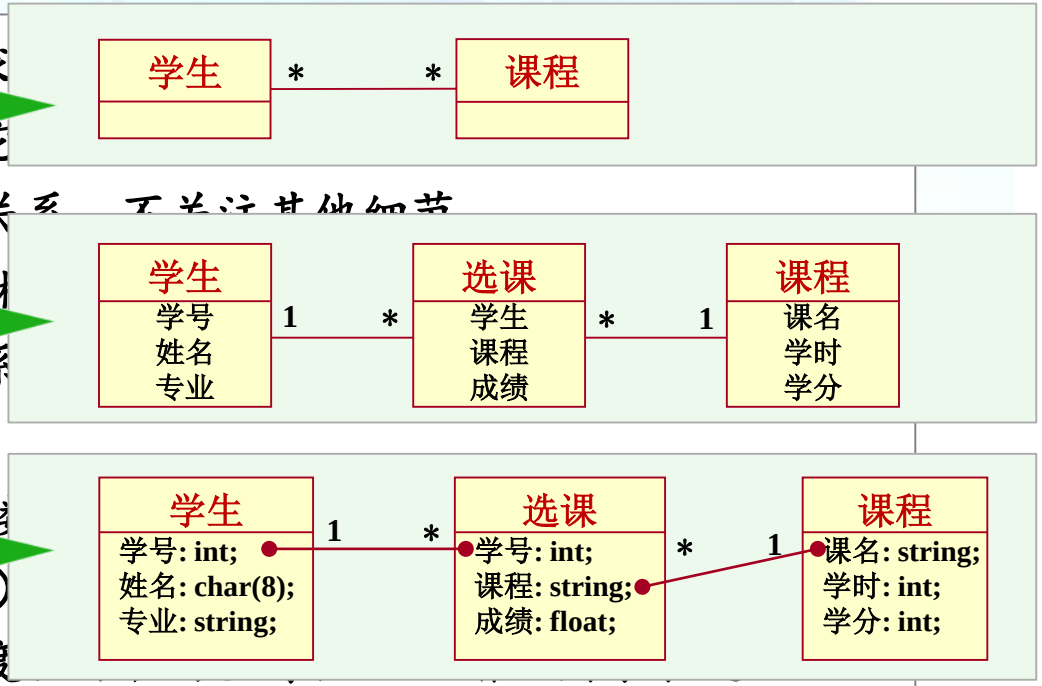
数据库设计

数据库设计的根本任务

- 获取未来系统需要存储的数据实体（一定粒度的数据单元）
- 分析数据实体的内涵，以及数据实体之间的关系
- 设计物理ERD，即用于数据库系统实现的数据表结构的蓝图

数据库设计的基本过程

- 概念ERD设计 – 主要用于需求
  层面所存在的数据实体，分析它
  给出存在的数据实体以及基本关系，不关注其他细节
- 逻辑ERD设计 – 逻辑 ERD 是
  中的属性，识别实体之间的关系
  现细节。
- 物理ERD设计 – 物理 ERD 是
  体转化为数据的数据表（Table）
  性的细节（包括属性类型、长度
  之间关系的新属性。



数据库设计

数据库设计的根本任务

- 获取未来系统需要存储的数据实体（一定粒度的数据单元）
- 分析数据实体的内涵，以及数据实体之间的关系
- 设计物理ERD，即用于数据库系统实现的数据表结构的蓝图

数据库设计的基本过程

- 概念ERD设
  层面所存在
  给出存在的
- 逻辑ERD设
  中的属性，
  现细节。
- 物理ERD设
  体转化为数
  性的细节（
  之间关系的

ERD形式	概念ERD	逻辑ERD	物理ERD
实体（名称）	√	√	√
关系	√	√	√
关系基数	随意	√	√
关联实体		√	√
属性（名称）		√	√
属性的类型、长度、说明		随意	√
主键、外键			√

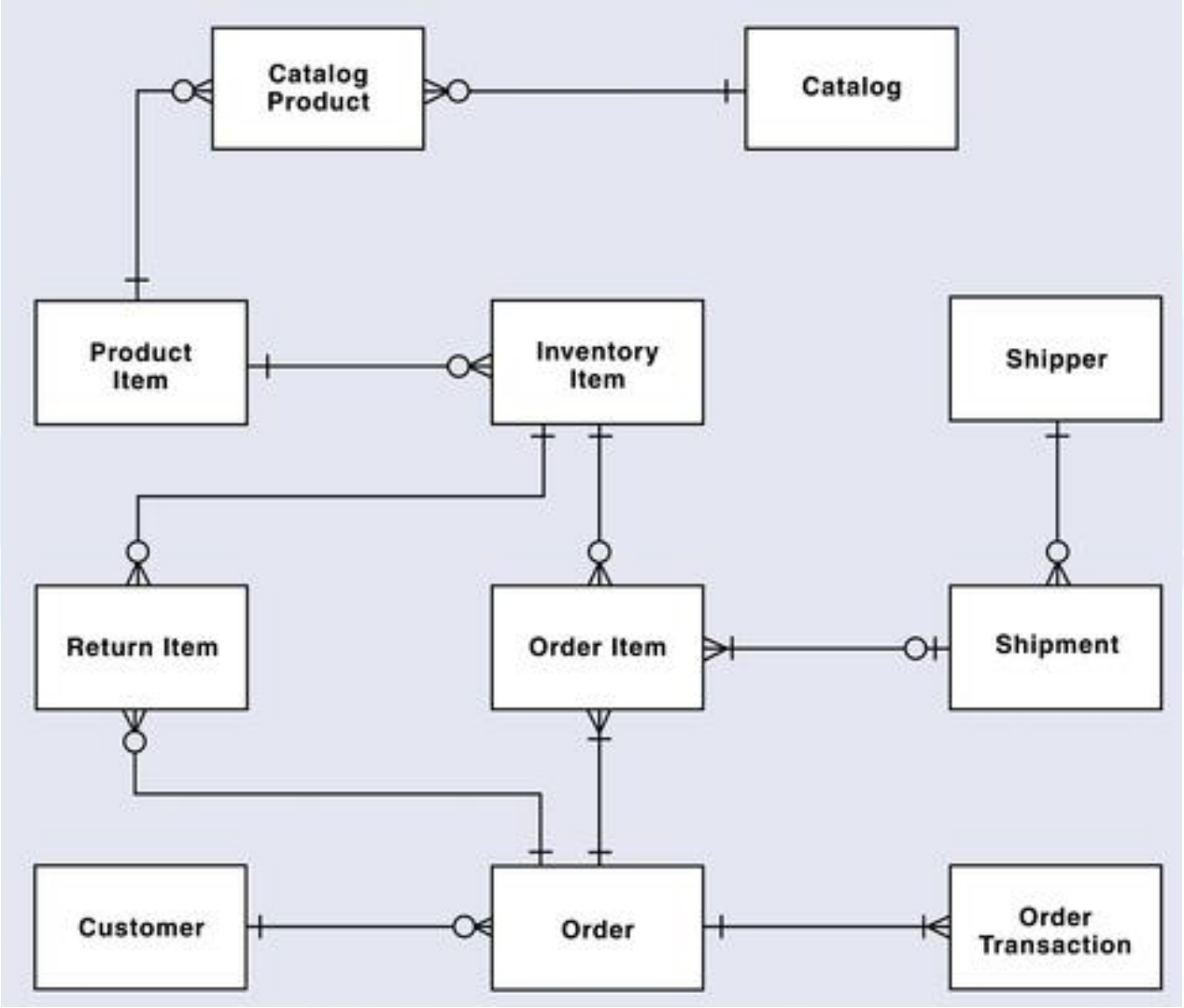
本章主要内容

1. 数据库系统及关系数据库简介
2. 数据库逻辑模型设计
3. ERD模型及质量评价
4. 物理数据库设计及建立
5. 物理数据库提高效率的技巧
6. 描述分布式数据库的不同的体系结构模型
7. OO实体类图映射到ERD
8. 关系型数据库规范化示例

数据库逻辑模型设计－ERD

1. 识别所有“自然”数据实体（Entity）
   - 来自DFD中的全部“数据存储”
   - 来自“分析类图”中的部分“类”
2. 为数据实体命名
3. 给出实体的属性
4. 识别实体之间的关联关系及关联重数
   - 关联关系需命名
   - 关联重数需判别（1:1、1:M、M:N）
5. 建立关联实体（“人造”实体）来消除M:N重关系
6. 实体规范化（一般需满足3NF）
7. 评价ERD质量并做必要的改进

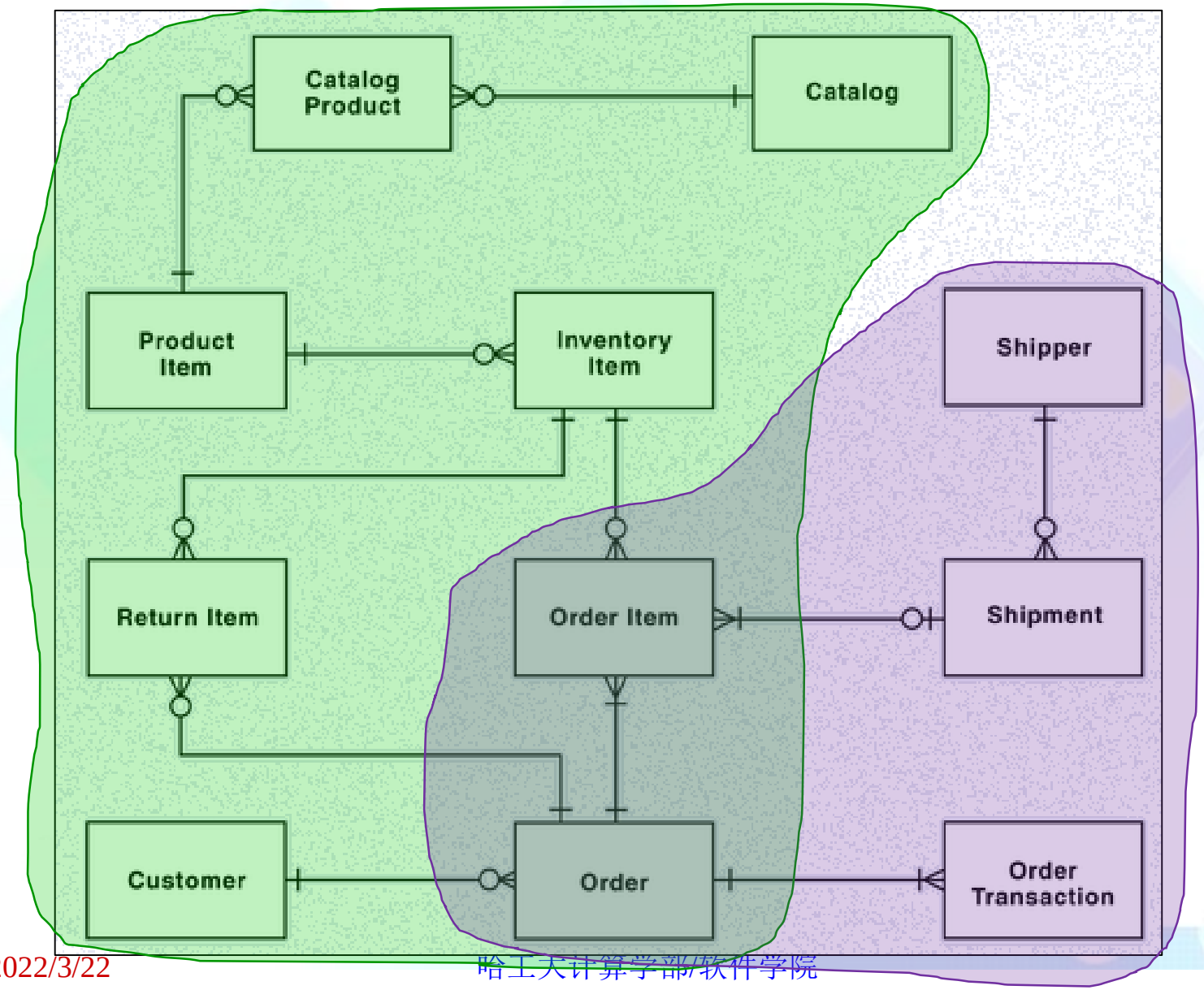
ERD图例子（Crow's feet样式）



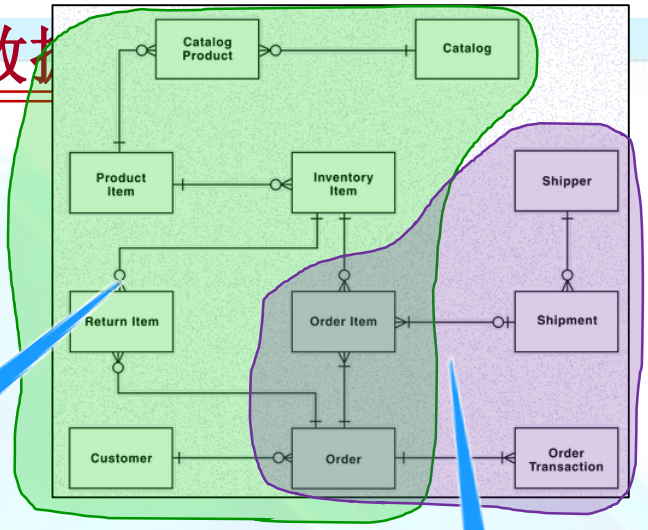
复杂ERD分组技巧

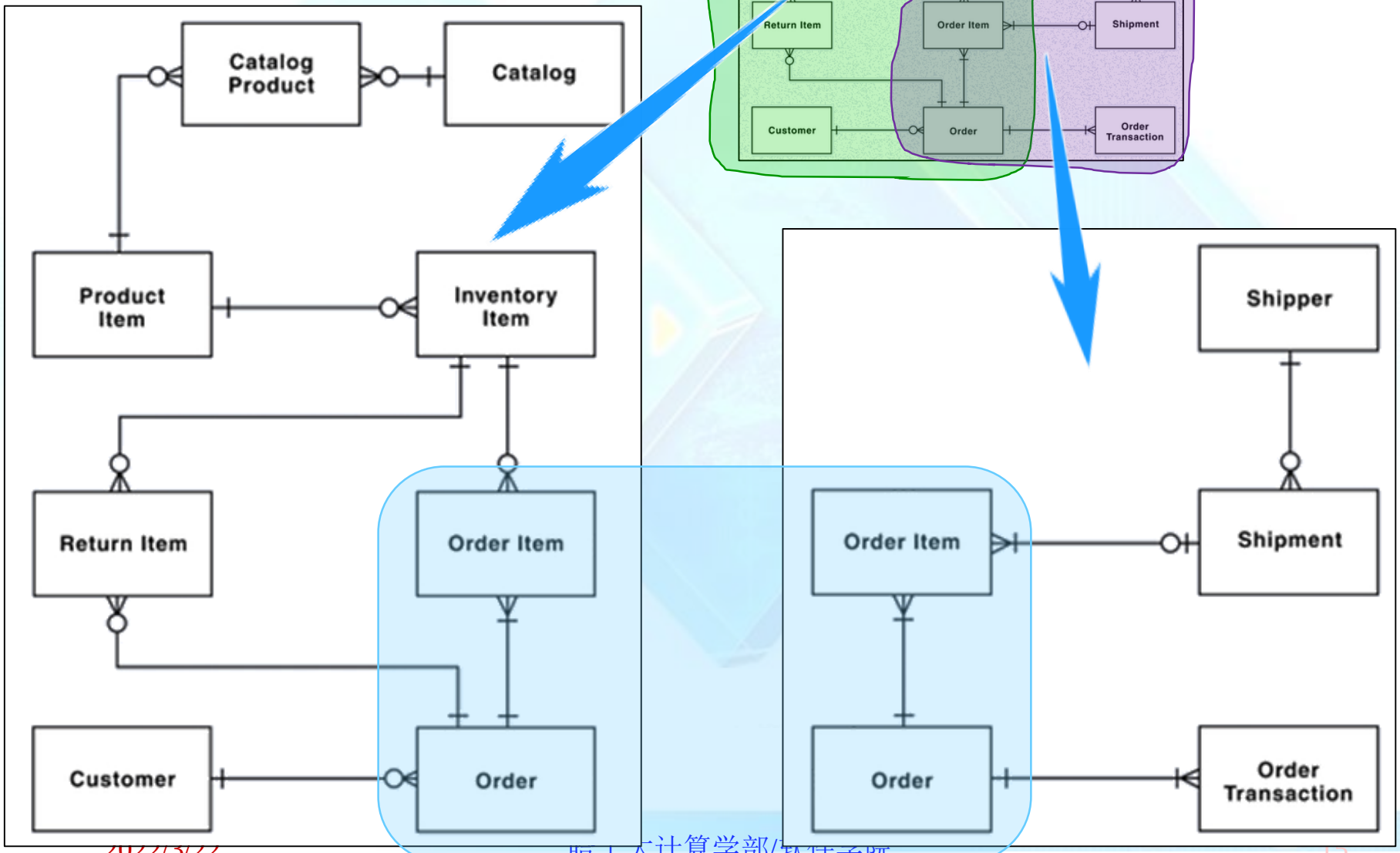
- ERD中实体非常多，以至于画在一张图中过于复杂难以清晰使用，则可以根据系统功能来分组
- 如果组之间存在相互关联的实体出现，则可以将其画多次，分别在不同的局部ERD中出现

复杂ERD分组技巧


Catalog Product
Catalog
Product Item
Inventory Item
Shipper
Return Item
Order Item
Shipment
Customer
Order
Order Transaction

复杂ERD分组技巧


Catalog Product
Catalog
Product Item
Inventory Item
Shipper
Return Item
Order Item
Shipment
Customer
Order
Order Transaction


Catalog Product
Catalog
Product Item
Inventory Item
Return Item
Order Item
Customer
Order
Shipper
Order Item
Shipment
Order
Order Transaction

本章主要内容

1. 数据库系统及关系数据库简介
2. 数据库逻辑模型设计
3. **ERD模型及质量评价**
4. 物理数据库设计及建立
5. 物理数据库提高效率的技巧
6. 描述分布式数据库的不同的体系结构模型
7. OO实体类图映射到ERD
8. 关系型数据库规范化示例

ERD质量的评价

- **ERD图的设计结果没有标准答案，即不唯一**
- **好的ERD标准：**
  - 结构清晰
  - 关联简洁
  - 实体个数适中
  - 属性分配合理
  - 没有低级冗余（重复性冗余）
- **至少符合3NF**

数据库的规范化（Normalization）

- 通过最小化数据冗余来确保数据库模型的质量
- 范式（**Normal Form**）的分类
  - **1NF** – 没有重复的属性或属性组
  - **2NF** – 是1NF 且每个非主属性均函数依赖于主属性(主键)
  - **3NF** – 是2NF且非主属性间均不存在函数依赖

能否用通俗的语言解释1NF、2NF、3NF？

数据库的规范化（Normalization）

- 范式的通俗、易懂的解释：
  - 1NF：同样的东西不能重复拥有！
  - 2NF：领导在，他就在，找到领导就能找到他！
  - 3NF：领导只有一个，不能搞宗派，不能有多级领导！
- 3NF数据库设计可以做到，但是未必是最好的数据库
  - 为了提高运行效率，必须降低范式标准，适当保留冗余数据
  - 具体做法：在逻辑模型设计时遵守3NF，降低范式标准的工作放到物理数据模型设计时考虑，即增加冗余字段

本章主要内容

1. 数据库系统及关系数据库简介
2. 数据库逻辑模型设计
3. ERD模型及质量评价
4. 物理数据库设计及建立
5. 物理数据库提高效率的技巧
6. 描述分布式数据库的不同的体系结构模型
7. OO实体类图映射到ERD
8. 关系型数据库规范化示例

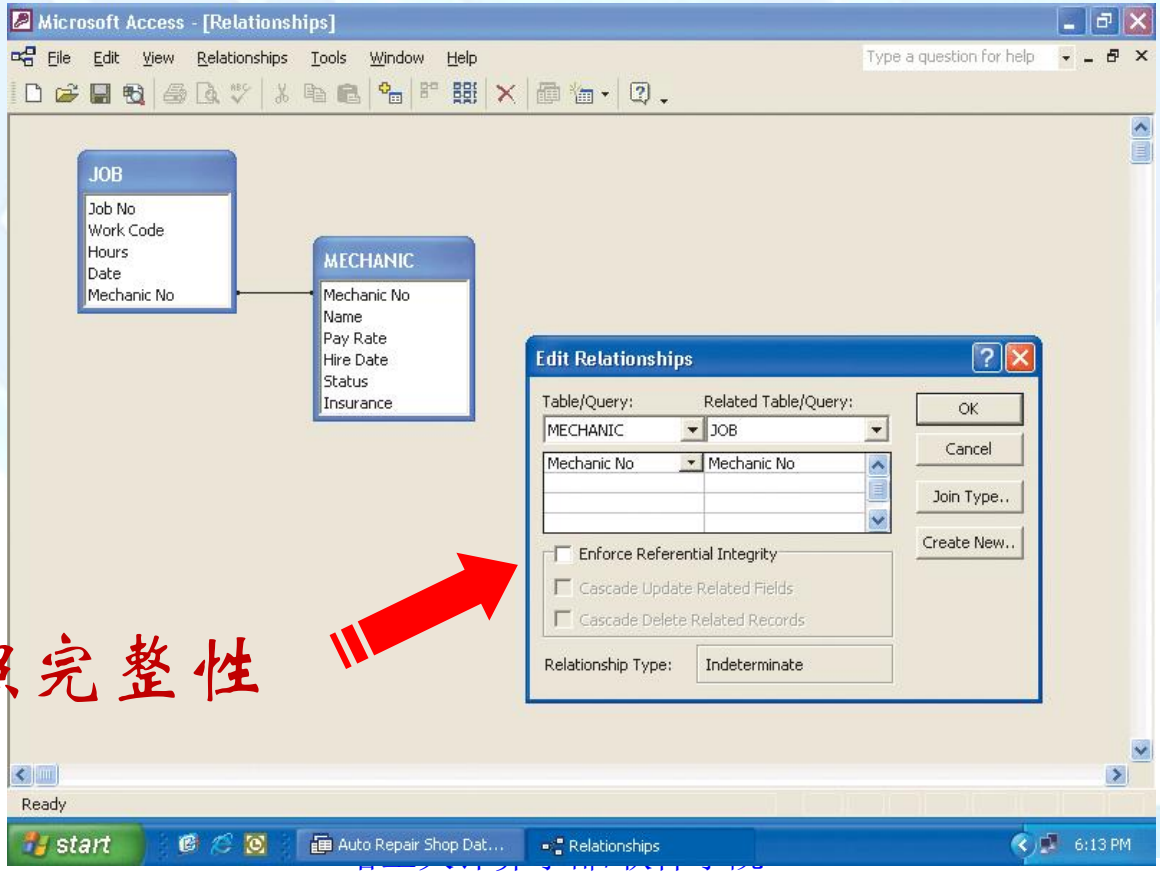
物理数据库设计及建立

根据ERD建立物理数据库：

1. 为每个ERD实体创建一个二维表（即数据库Table）
2. 根据ERD中的实体属性，在对应的Table中定义对应的字段（Field），并给出适当的数据类型和取值范围
3. 定义每个Table的主键(PK：PrimaryKey)
4. 针对1:M关联关系的子表添加外键（FK：ForeignKey）即对应主表的主键
5. 定义完整性约束

定义完整性约束（3个层次）

- 域的完整性：设定字段的取值范围
- 参照完整性：用PK、FK、表级触发器来实现
- 用户定义完整性：定义一些业务规则，用存储过程和触发器来实现



执行参照完整性

参照完整性的自动执行

- 当建立一个包含外键的记录时，DBMS 确保该外键同时作为主键出现在另一个相关表的记录中
  - 如果关联其他记录，对方必须存在
- 当删除一个记录时，DBMS 确保在其它相关的表中不会出现和该记录主键值相同的外键
  - 如果正在被其他记录关联，则不能删除
- 当主键值改变时，DBMS保证相关表中没有外键与该主键具有相同的值
  - 改变记录主键值时：①确保不冲突；②全局自动更改

本章主要内容

1. 数据库系统及关系数据库简介
2. 数据库逻辑模型设计
3. ERD模型及质量评价
4. 物理数据库设计及建立
5. 物理数据库提高效率的技巧
6. 描述分布式数据库的不同的体系结构模型
7. OO实体类图映射到ERD
8. 关系型数据库规范化示例

提高数据库运行效率的5个办法

(1) 降低范式、增加冗余；少用触发器、多用存储过程

(2) “体外运算”

当计算非常复杂、记录数巨大时，则以文件系统方式用C++等语言计算处理完成之后，最后才入库追加到表中去

（电信计费系统设计的经验）

(3) 水平/垂直分割表

- 当表中记录太多时则水平分割：以该表主键PK的某个值为界线，将该表的记录水平分割为两个表
- 当表中字段太多时则垂直分割：将原来的字段分为二组，分别建立二个1：1的表

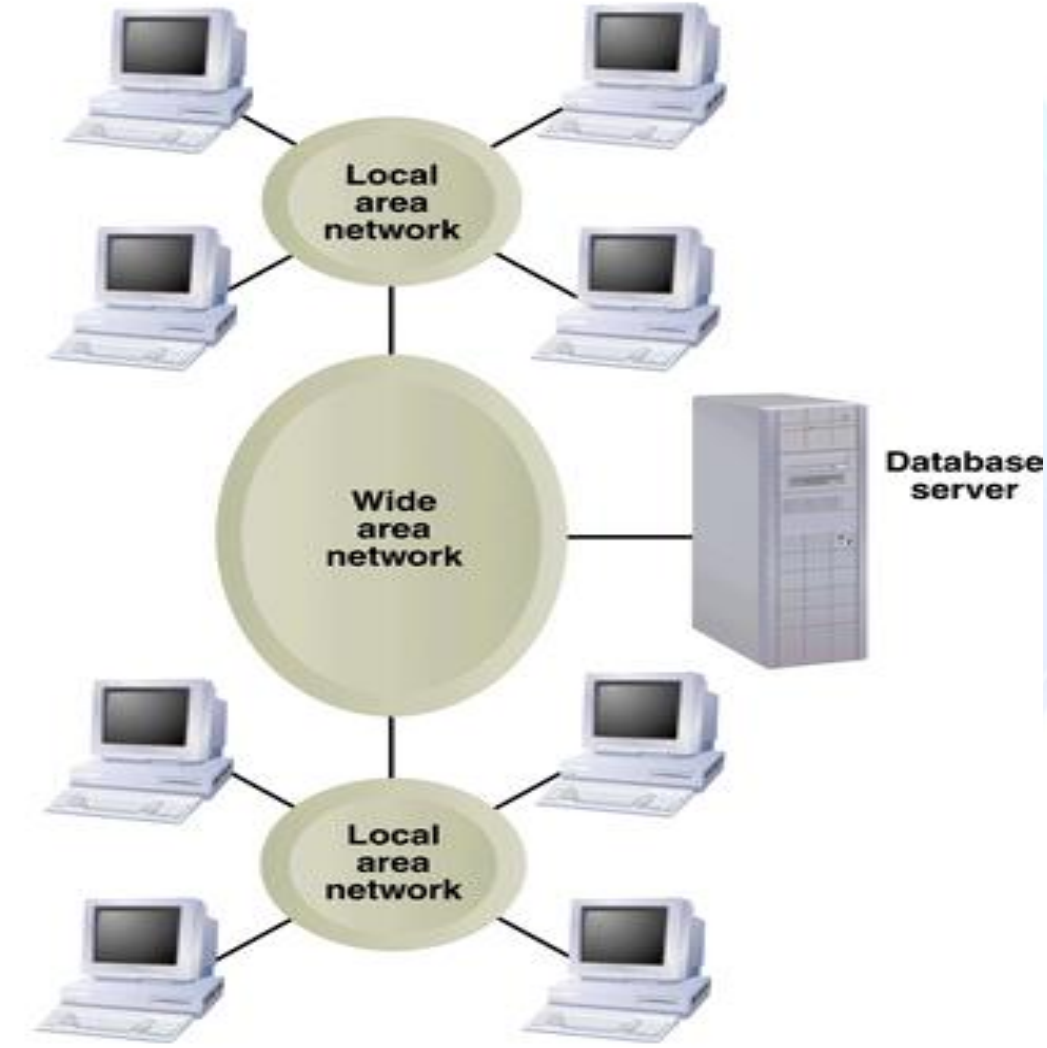
(4) 对数据库管理系统DBMS进行系统优化，即优化各种系统参数，如缓冲区个数

(5) 在使用面向数据的SQL语言进行程序设计时，尽量采取优化算法

本章主要内容

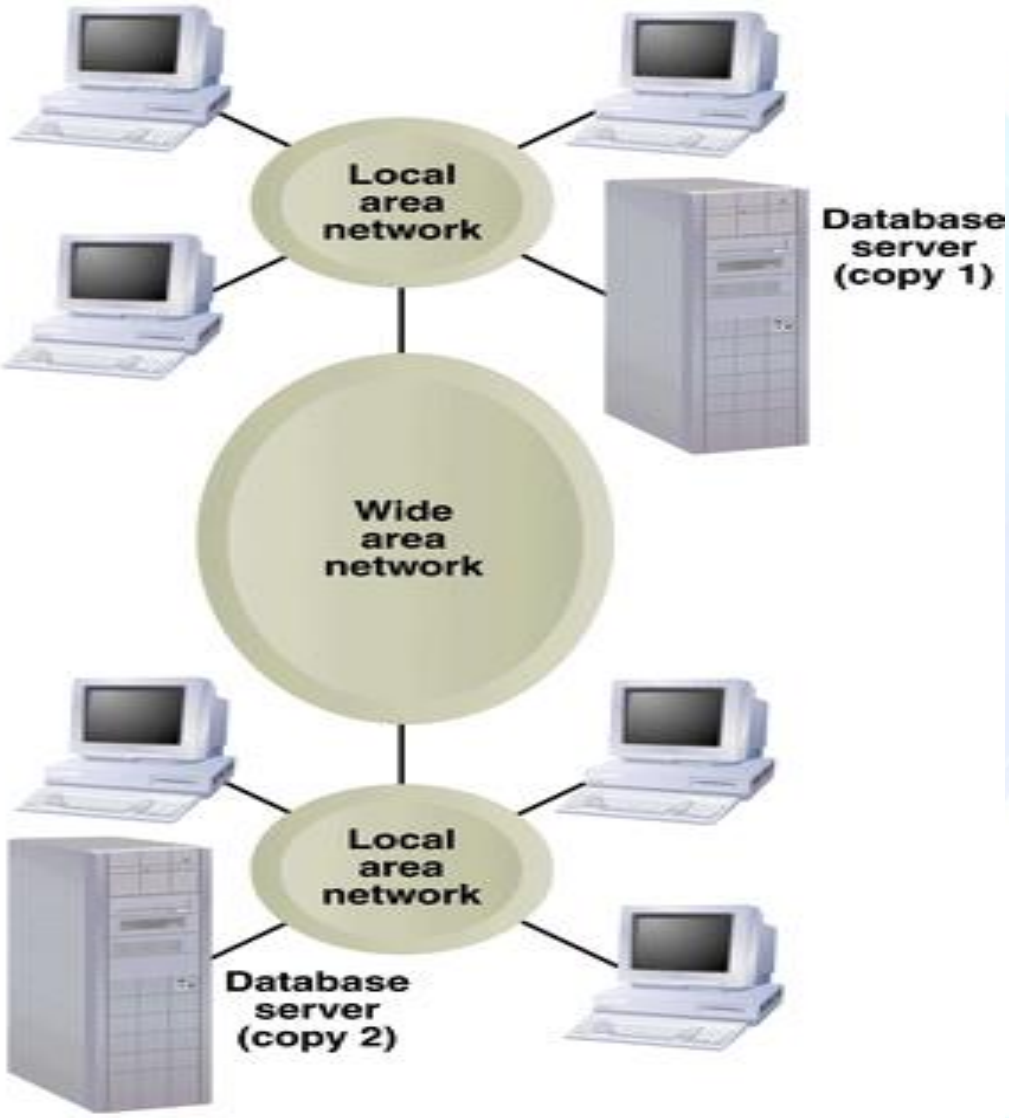
1. 数据库系统及关系数据库简介
2. 数据库逻辑模型设计
3. ERD模型及质量评价
4. 物理数据库设计及建立
5. 物理数据库提高效率的技巧
6. 描述分布式数据库的不同的体系结构模型
7. OO实体类图映射到ERD
8. 关系型数据库规范化示例

单数据库服务器体系结构



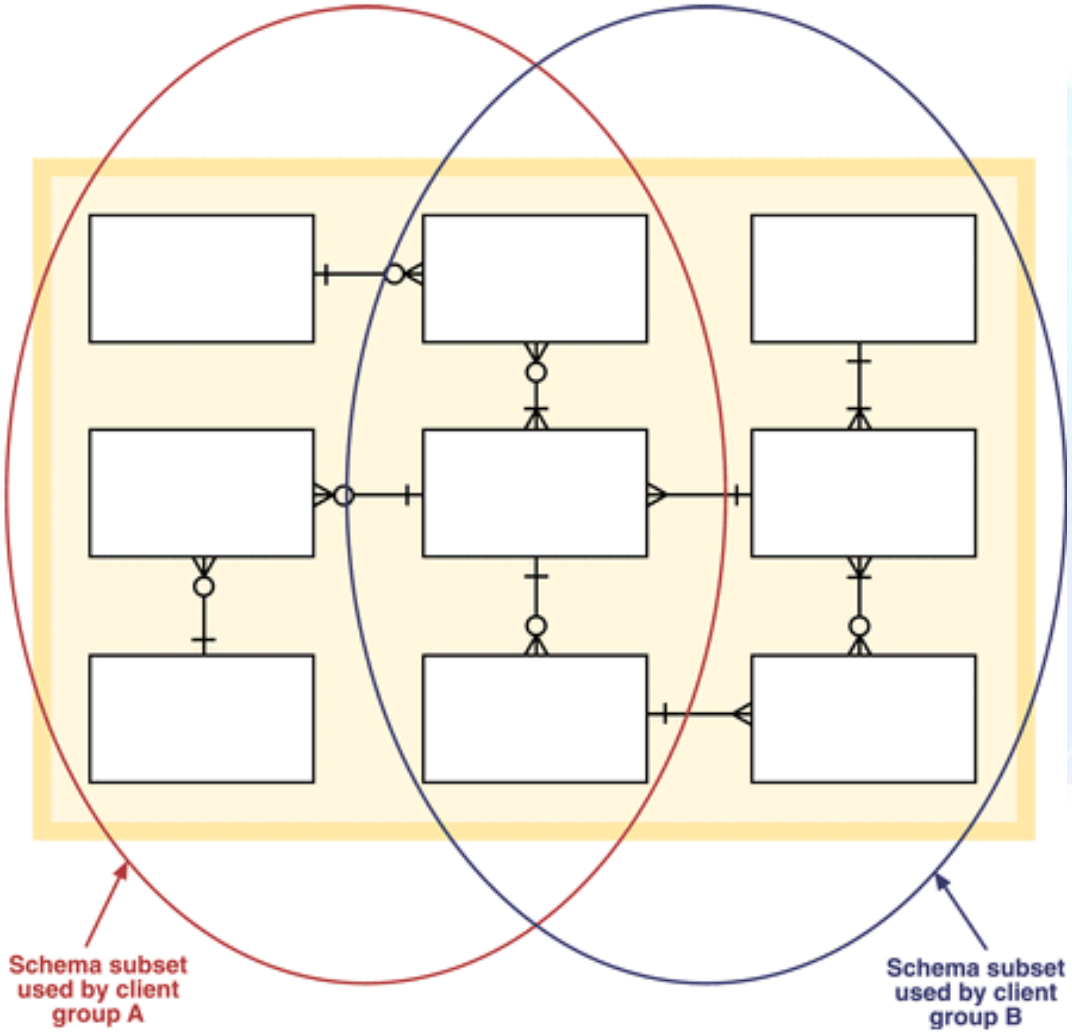
数据库设计

带备份的数据库服务器体系结构

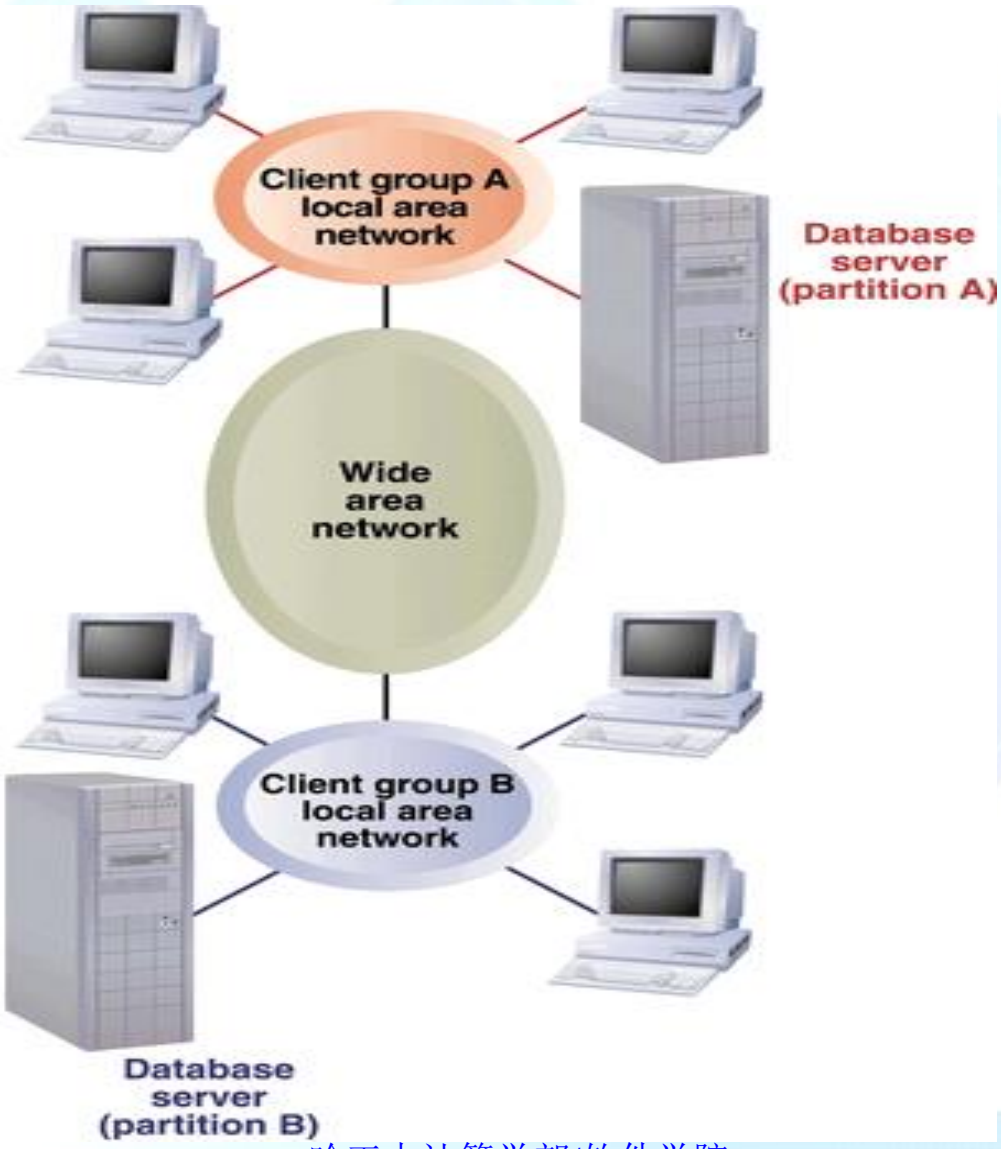


数据库设计

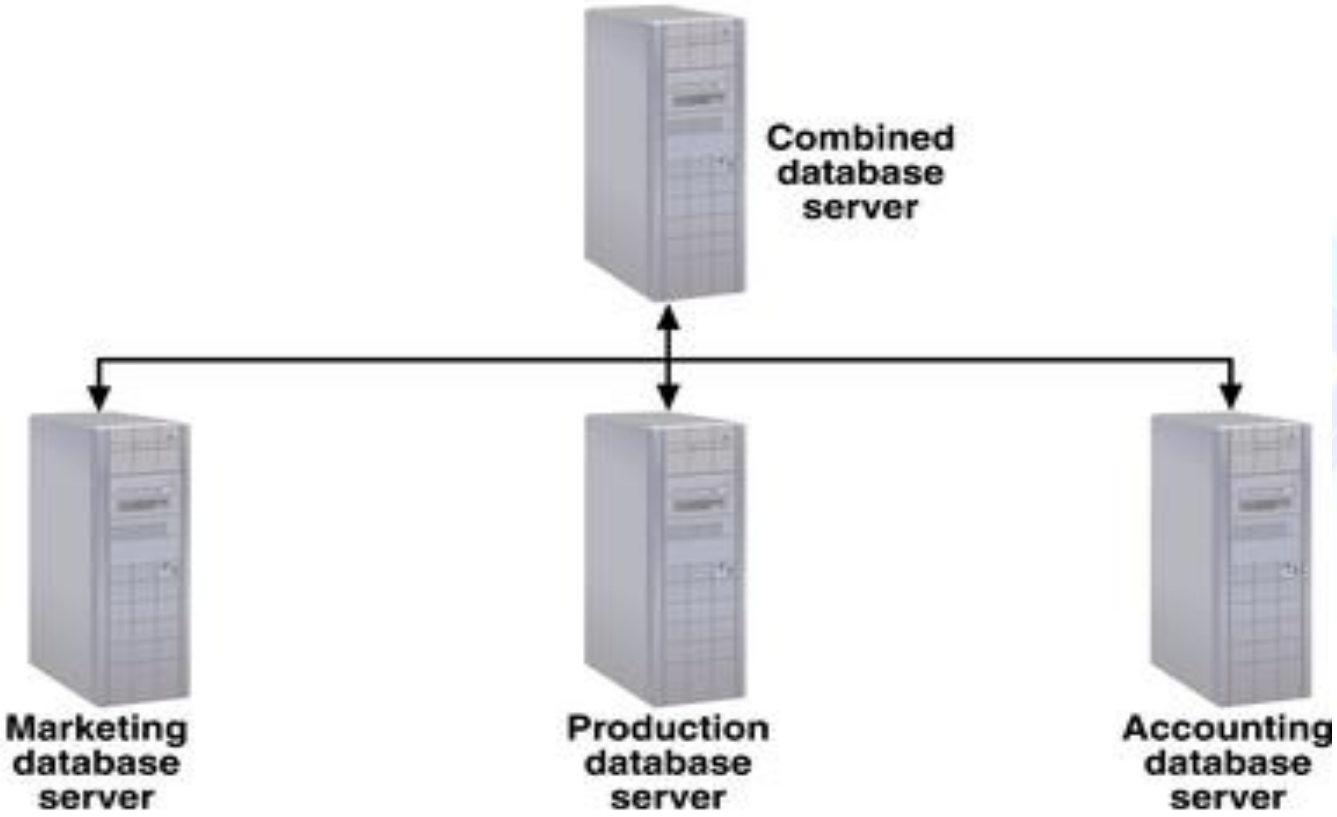
将数据库模式划分为多个客户访问子集



分区数据库服务器体系结构



联合数据库服务体系结构



本章主要内容

1. 数据库系统及关系数据库简介
2. 数据库逻辑模型设计
3. ERD模型及质量评价
4. 物理数据库设计及建立
5. 物理数据库提高效率的技巧
6. 描述分布式数据库的不同的体系结构模型
7. **OO实体类图映射到ERD**
8. 关系型数据库规范化示例

类和关系数据表的关系

- OOP以class为基本单位，所有的object都是运行在内存空间当中
- 若某些object的信息需要持久化存储，那么就需要用到database，将object的属性信息写入关系数据表
  - 假如淘宝没有“保存购物车内容”的功能（意即若不购买，下次进入之后购物车中的内容就被清空），那么“购物车”这个实体的属性就不需要关系数据表
- 在系统执行某些功能的时候，需要首先从database中将信息取出，使用各实体类的new操作构造相应的object，在程序运行空间中使用(充分利用继承/组合/聚合/关联/依赖关系在各object之间相互导航)
- 在进行OO分析和设计时完全可以不考虑数据永久存储的事情，本节就是要考虑如何使用数据库来解决永久存储的问题！

数据存储设计

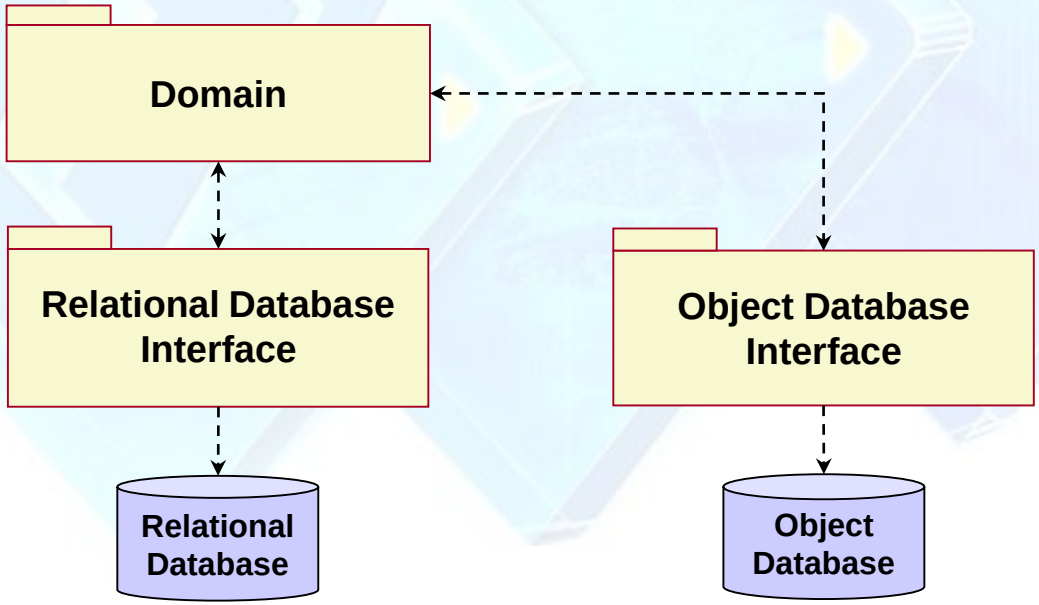
- “对象”只是存储在内存当中，而某些对象则需要永久性的存储起来 -- 持久性数据（**persistent data**）
- 数据存储策略
  - **数据文件：**
    由操作系统提供的存储形式，应用系统将数据按字节顺序存储，并定义如何以及何时检索数据
  - **关系数据库：**
    数据是以表的形式存储在预先定义好的**Schema**的类型中
  - **面向对象数据库：**
    将对象和关系作为数据一起存储
  - **存储策略的选择：**
    取决于非功能性的需求

数据存储策略的权衡（tradeoff）

- 何时选择文件？
  - 存储大容量数据、临时数据、低信息密度数据
- 何时选择数据库？
  - 并发访问要求高、系统跨平台、多应用程序使用相同数据
- 何时选择关系数据库？
  - 复杂的数据查询
  - 数据集规模大
- 何时选择面向对象数据库？
  - 数据集处于中等规模
  - 频繁使用对象间联系来读取数据

数据存储策略

- 如果使用OO数据库，那么数据库系统应提供一个接口供应用系统访问数据
- 如果使用关系数据库，那么需要一个子系统来完成应用系统中的对象和数据库中数据的映射与转换



OO设计中的数据库设计

- 核心问题：对那些需要永久性存储的数据，如何将UML类图（实体类图）映射为数据库模型（实体关系模型ERD）
- 本质：把每一个类、类之间的关系分别映射到一张表或多张关系型数据库的表中（Table）
- 解决：UML class diagram → Relation DataBase (RDB)
- 两个方面：
  - 将类（class）映射到表（table）
  - 将关联关系（association）映射到表（table）

对象关系映射(ORM)

- 对象关系映射（Object Relational Mapping，ORM）：
  - 为了解决面向对象与关系数据库存在的互不匹配的现象
  - 通过使用描述对象和数据库之间映射的元数据，将OO系统中的对象自动持久化到关系数据库中
- 目前流行的ORM产品：
  - Apache OJB
  - Hibernate
  - JPA(Java Persistence API)
  - Oracle TopLink
  - …

将对象映射到关系数据库

- 最简单的映射策略 一类一表：表中的字段对应于类的属性，表中的每一行数据记录对应类的实例(即对象)

Object Model

Person
name address

Table Model

Attribute Name	Datatype	nulls
Person_ID	int	N
Person_name	Char(20)	N
address	Char(50)	Y

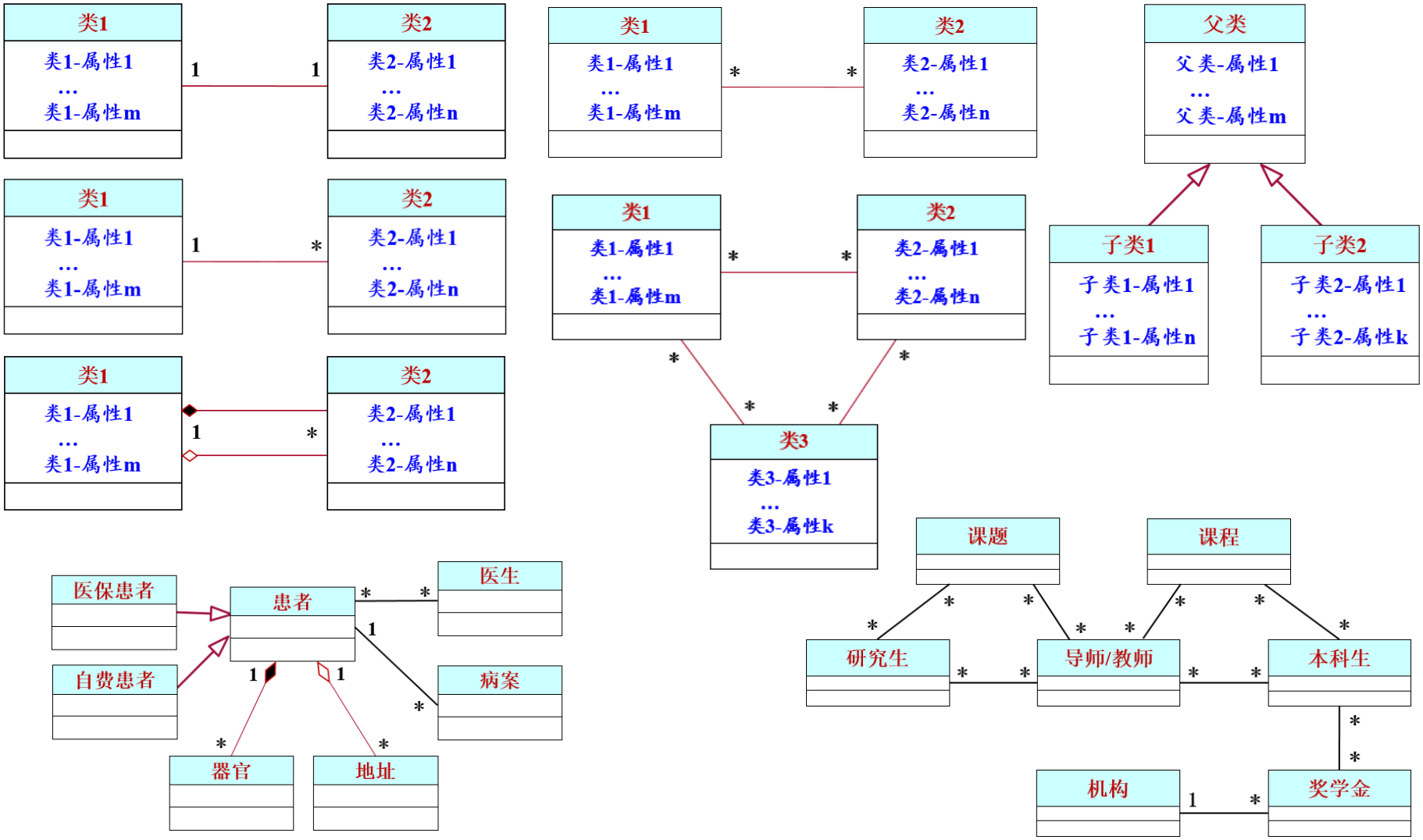
SQL Code

```
CREATE TABLE Person (person_ID int not null,
                     person name char(20) notnull,
                     address char(50),
                     PRIMARY KEY  Person_ID);
```

数据库设计

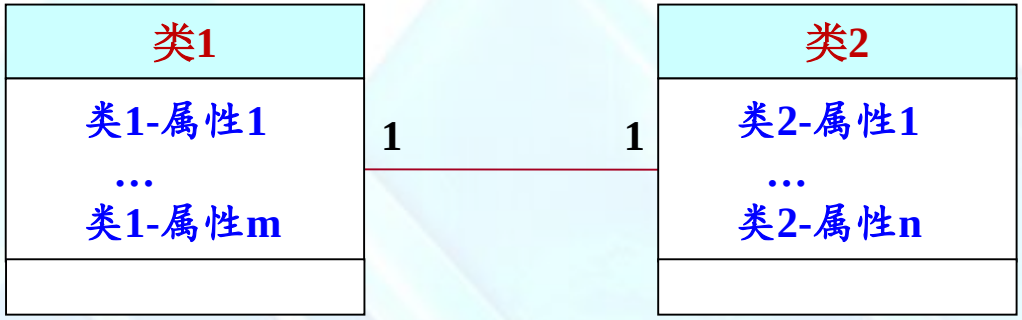
将对象映射到关系数据库

- 具有复杂关系的实体类，如何映射到关系数据库中呢？



数据库设计

实体类图 → ERD映射　典型类型一：1:1 关联关系



实例：考生 ↔ 报名表

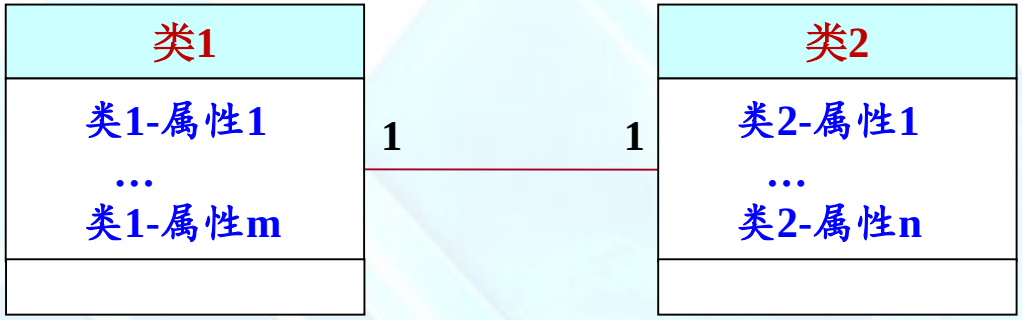
方法1：建立1个表

（1）增加PrimaryKey属性

（2）复制2个类中的所有属性

数据库设计

实体类图→ERD映射　典型类型一：1:1 关联关系



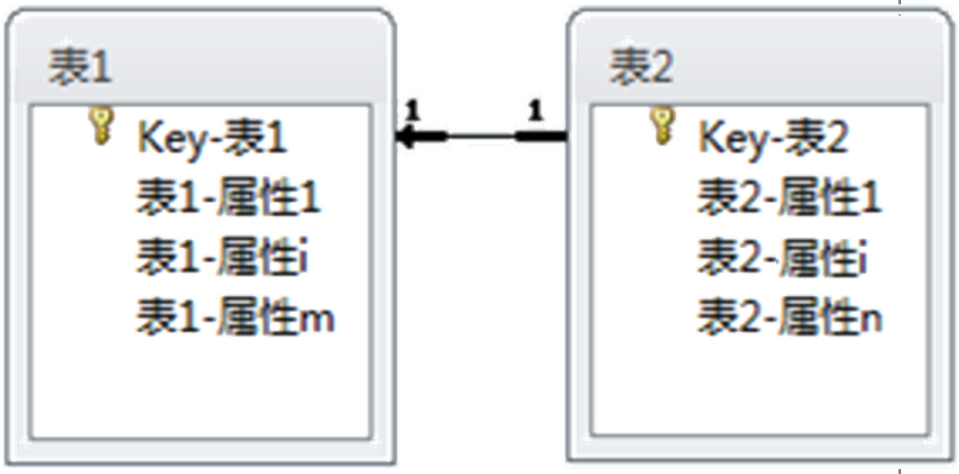
实例：考生↔报名表

方法2：建立1:1关系的2个表

（1）每个表分别增加PrimaryKey

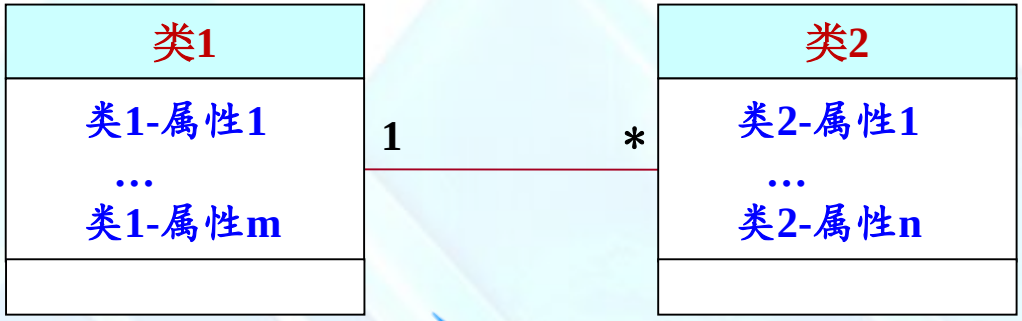
（2）每个表中分别复制对应类中的所有属性

（3）2个表的PrimaryKey建立关联，从而建立参照完整性，任何一个表作为主表均可



数据库设计

实体类图 → ERD映射　典型类型二：1:* 关联关系



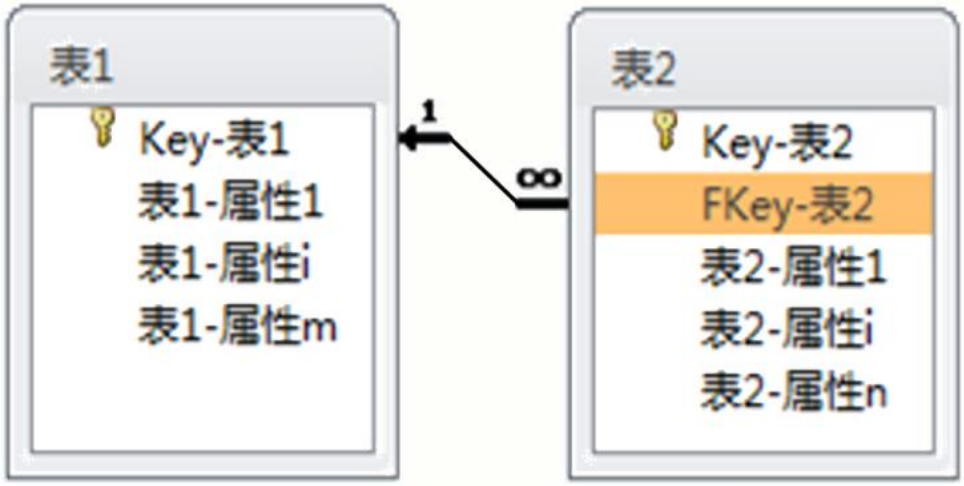
实例：父母 ↔ 子女

方法：建立1:*关系的2个表

（1）每个表分别增加PrimaryKey

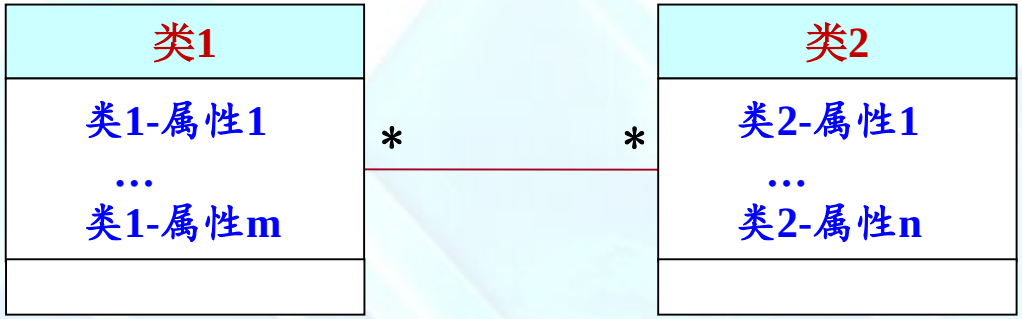
（2）每个表分别复制对应类中的所有属性

（3）表2中增加ForeignKey属性，并与表1的PrimaryKey属性关联，从而建立参照完整性，主表为“基数”为1的那个表



数据库设计

实体类图→ERD映射　典型类型三：*:* 关联关系



实例：学生↔课程

方法：建立3个表，表1、表2分别对应类1、类2，第3个表为关联表，表1、表与关联表之间均为1:*的关系

（1）3个表分别增加PrimaryKey

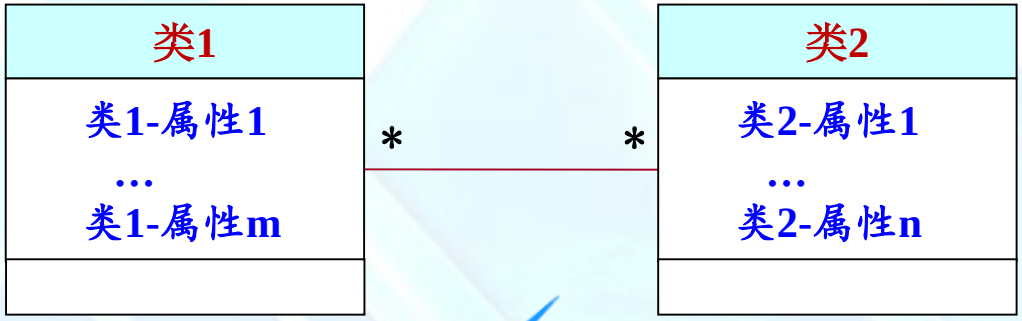
（2）表1、表2的其他属性分别复制类1、类2中的所有属性

（3）关联表中增加2个ForeignKey属性，分别与表1、表2中的PrimaryKey关联，从而建立参照完整性，表1、表2均为主表

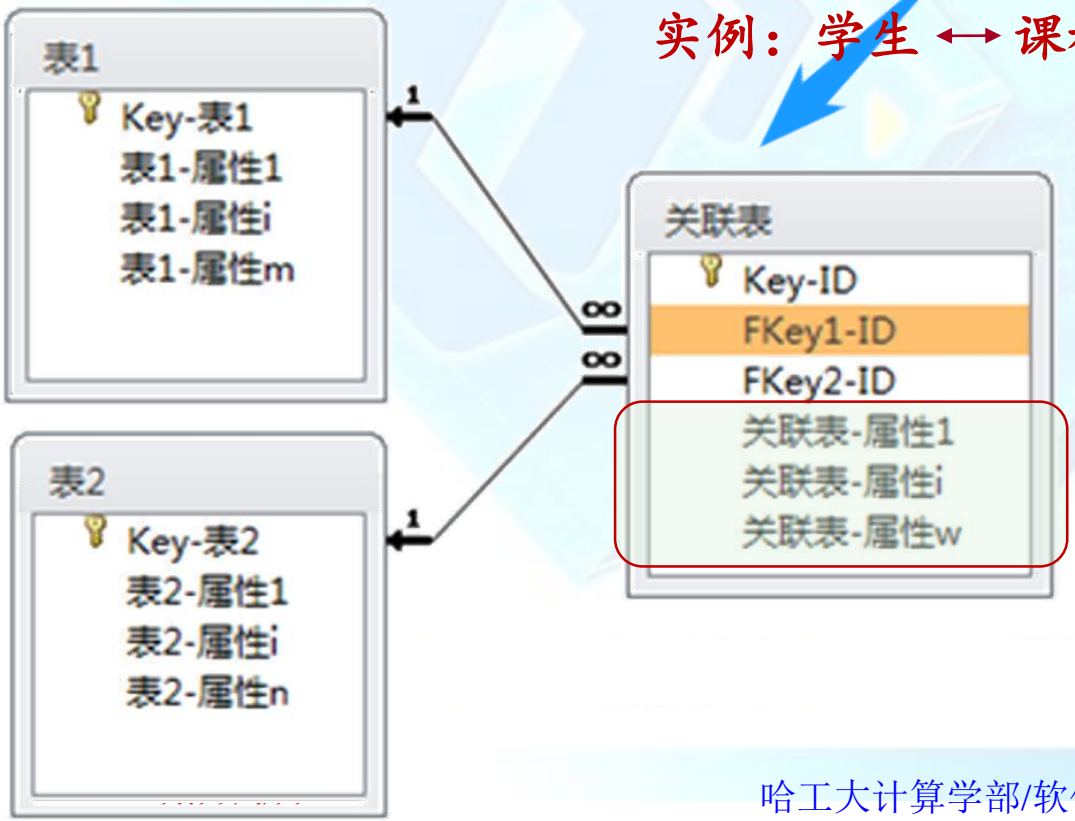
（4）关联表中经常会增加若干其他属性

数据库设计

实体类图 → ERD映射　典型类型三：*:* 关联关系



实例：学生 ↔ 课程



方法：

（1）3个表分别增加PrimaryKey

（2）表1、表2的其他属性分别复制类1、类2中的所有属性

（3）关联表中增加2个ForeignKey属性，分别与表1、表2中的PrimaryKey关联，从而建立参照完整性，表1、表2均为主表

（4）关联表中经常会增加若干其他属性

数据库设计

实体类图→ERD映射　典型类型四：循环 *:* 关联关系

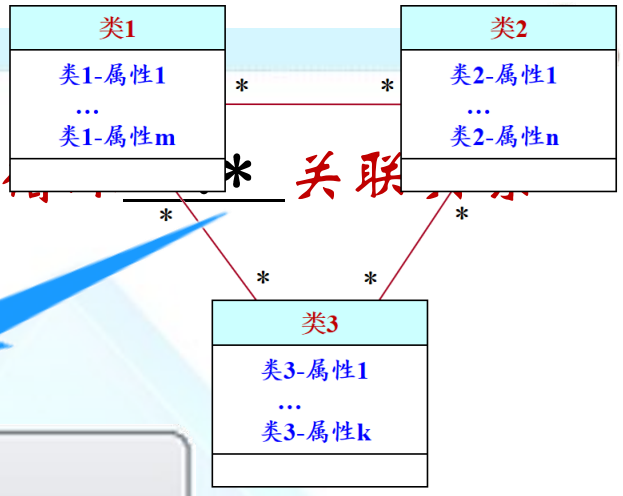
实例：学生↔课程↔教师

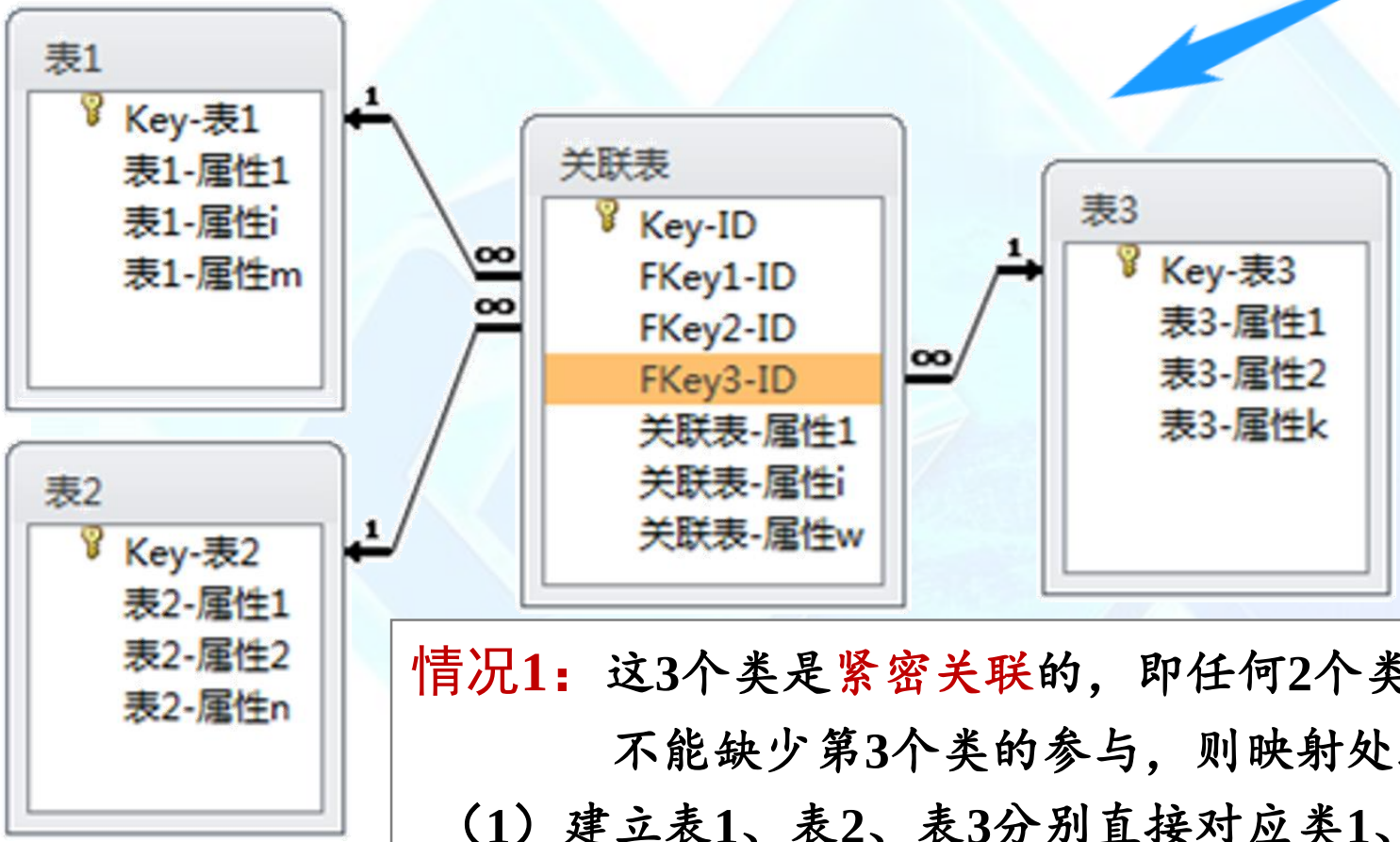
情况1：这3个类是紧密关联的，即任何2个类的关联行为都不能缺少第3个类的参与

情况2：这3个类仅仅是两两相关联

数据库设计

实体类图 → ERD映射　典型类型四：*:* 关联关系





情况1： 这3个类是**紧密关联**的，即任何2个类的关联行为都不能缺少第3个类的参与，则映射处理如下：

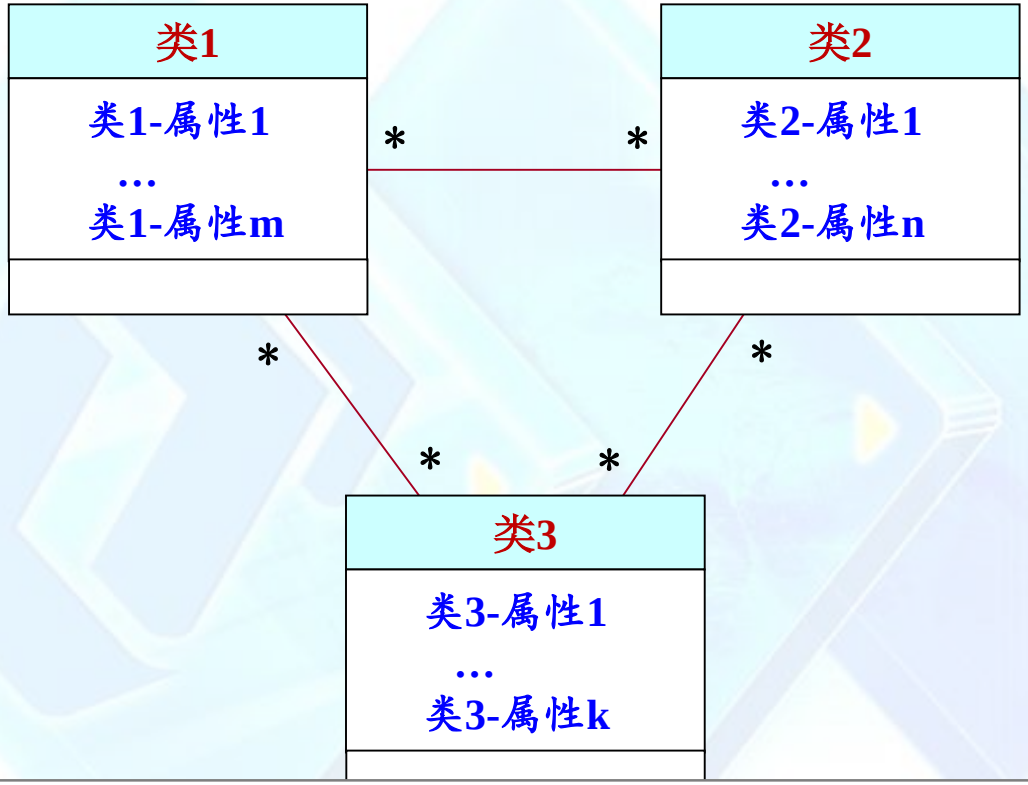
（1）建立表1、表2、表3分别直接对应类1、类2、类3；

（2）建立1个**关联表**，需要创建3个ForeignKey，分别跟表1～3的PrimaryKey关联，建立参照完整性；

（3）**关联表**中一般会增加若干其他属性。

数据库设计

实体类图→ERD映射　典型类型五：循环*:*关联关系

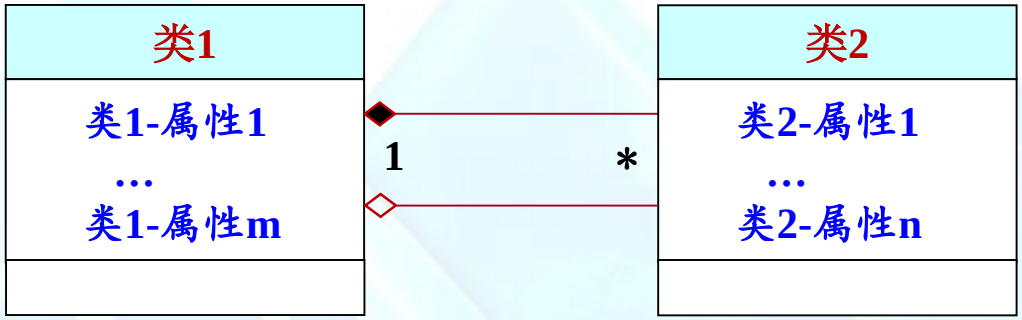


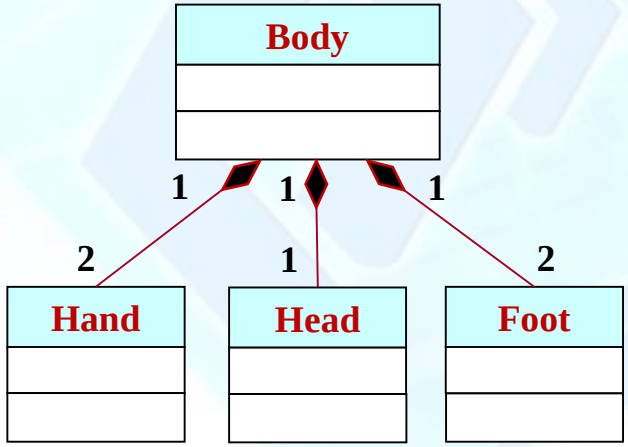
情况2：这3个类仅仅是两两相关联，即该类图包含3个“典型类型三”的情况，即表1-表2、表2-表3、表3-表1，需要创建3个关联表，每对关联的处理方法同“典型类型三”。

例如：学生-课程+成绩、教师-课程+评价、学生-教师+项目。

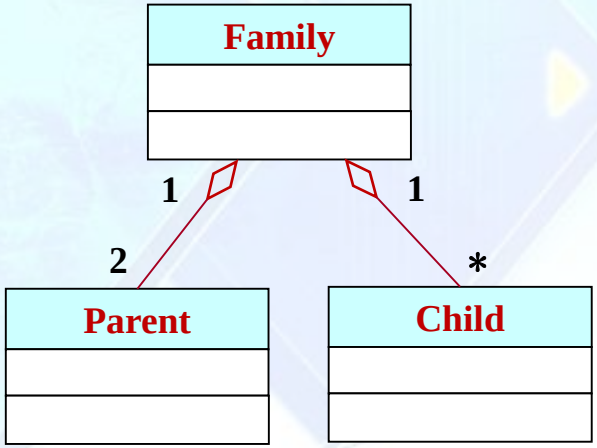
数据库设计

实体类图→ERD映射　典型类型六：组合/聚合关联关系





实例：躯体↔器官



实例：家庭↔成员

方法： 聚合/组合可以理解为特别紧密的关联关系

在映射到数据库表设计时，参照**1:1**或**1:***关系处理即可

数据库设计

实体类图→ERD映射　典型类型七：继承关系

实例：雇员←经理/职员
师生←学生/教师

方法1：分别建立父类和子类的3张数据表，形成1:*的类型

方法2：分别建立子类的2张数据表，父类并到子类的表中

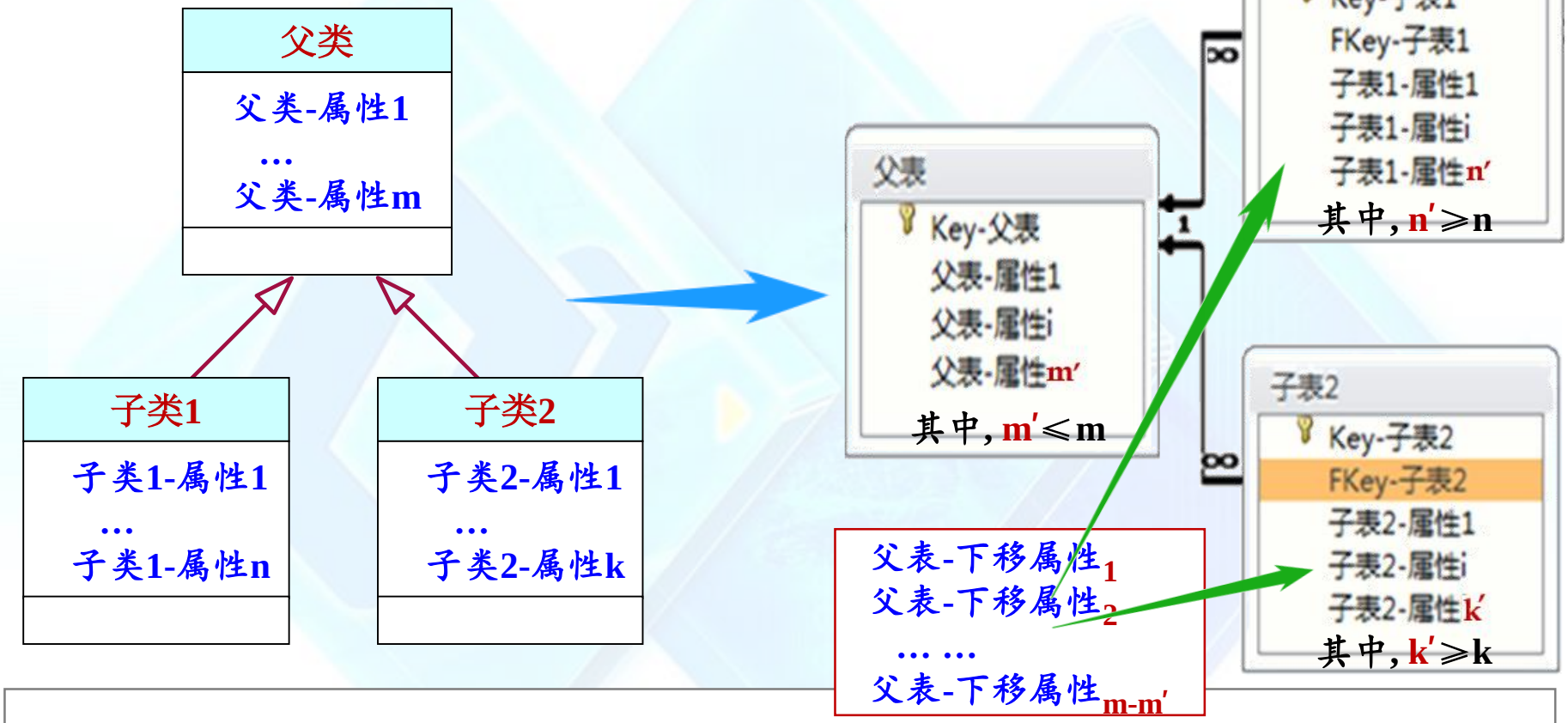
方法3：只建立父类的1张数据表，子类并到父类的表中

讨论：（1）从节省空间和访问效率上比较，方法1~3的性能逐次降低

（2）如果属性比较少的情况下，方法2或3还是可以采用的

数据库设计

实体类图 → ERD映射　典型类型七：继承关

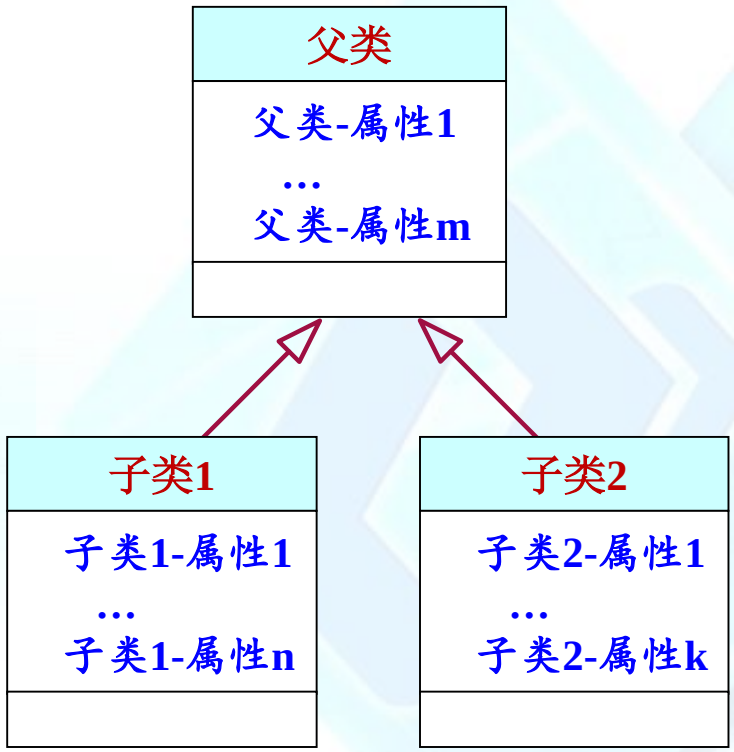


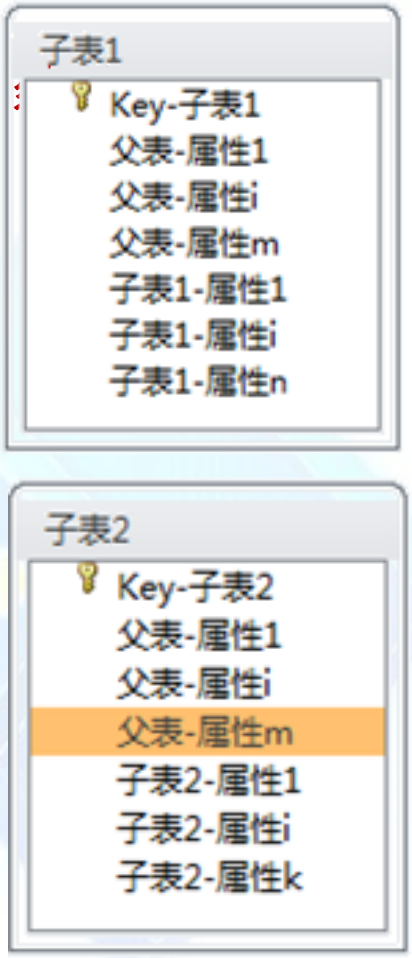
方法1：分别建立父类和子类对应的共3张数据表，形成1:*的映射类型；其中父类表保留父类的全部属性，或者将部分属性（父类与子类是1:1关系的属性）下移到子类对应的表中

讨论：最大程度减少冗余数据的存储

数据库设计

实体类图 → ERD映射　典型类型七：





方法2：分别建立子类共2张数据表，将父类的属性全部下移到各个子类所对应的数据表中

讨论：（1）会产生一定量冗余数据，浪费存储空间，表的访问效率会降低

（2）当父类中独立的属性较少时，该方案可行

数据库设计

实体类图→ERD映射　典型类型七：继承关系

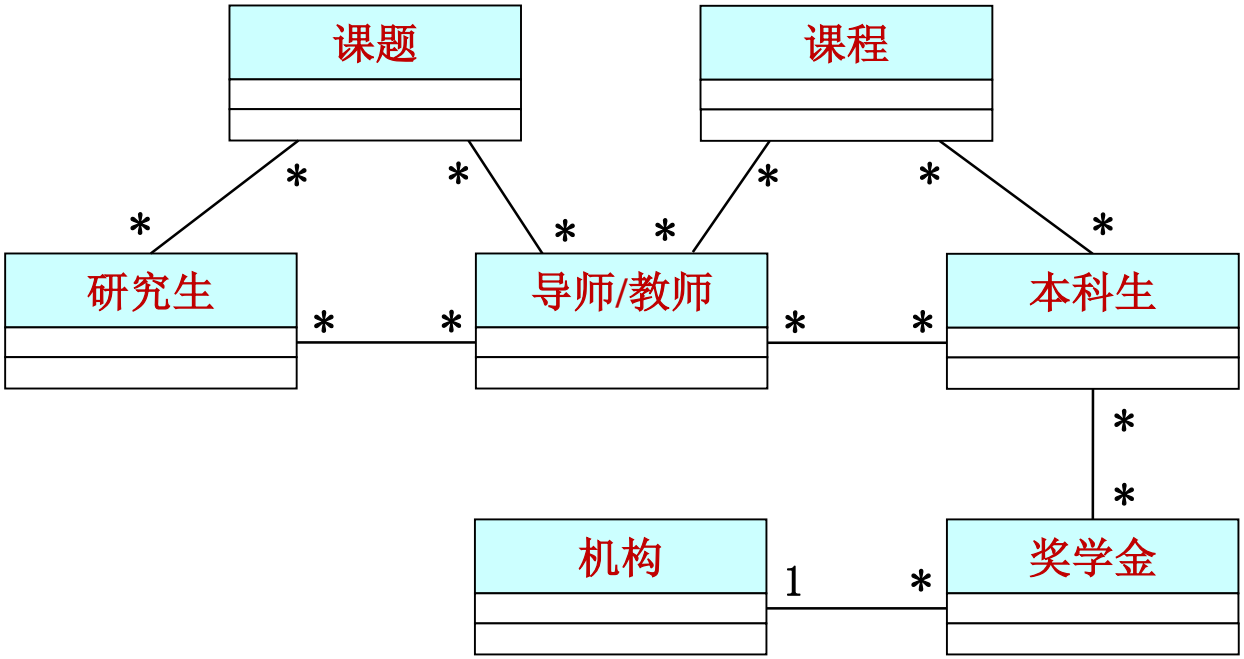
方法3：只建立1张数据表，将子类的属性上移到父类所对应的数据表中，该表包括父类和各子类的全部属性，需增加区别子类的标志性属性

讨论：（1）会产生大量的空白数据和冗余数据，比方法2更浪费存储空间，表的访问效率会大大降低

（2）只有子类中属性很少的情况下，该方法才可用

数据库设计

实体类图→ERD映射　思考一：复杂*:*关联关系



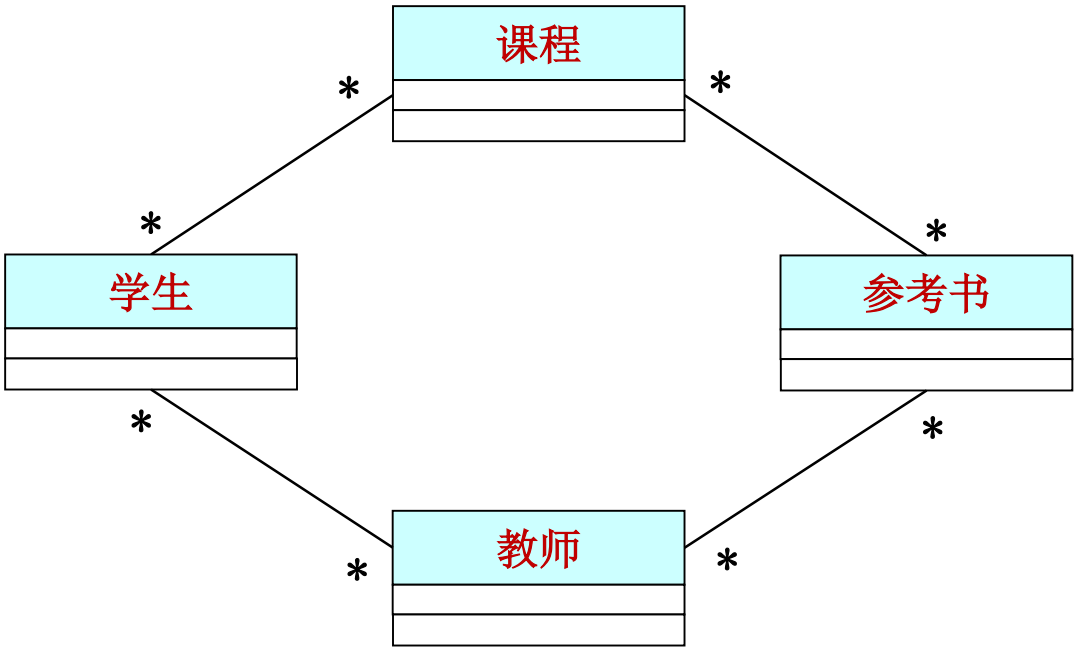
循环1：　导师 -指导- 研究生 -研究- 课题 -申请- 导师

循环2：　教师 -教授- 本科生 -选修- 课程 -讲授- 教师

关联链：本科生 -获得- 奖学金 -设立- 机构

数据库设计

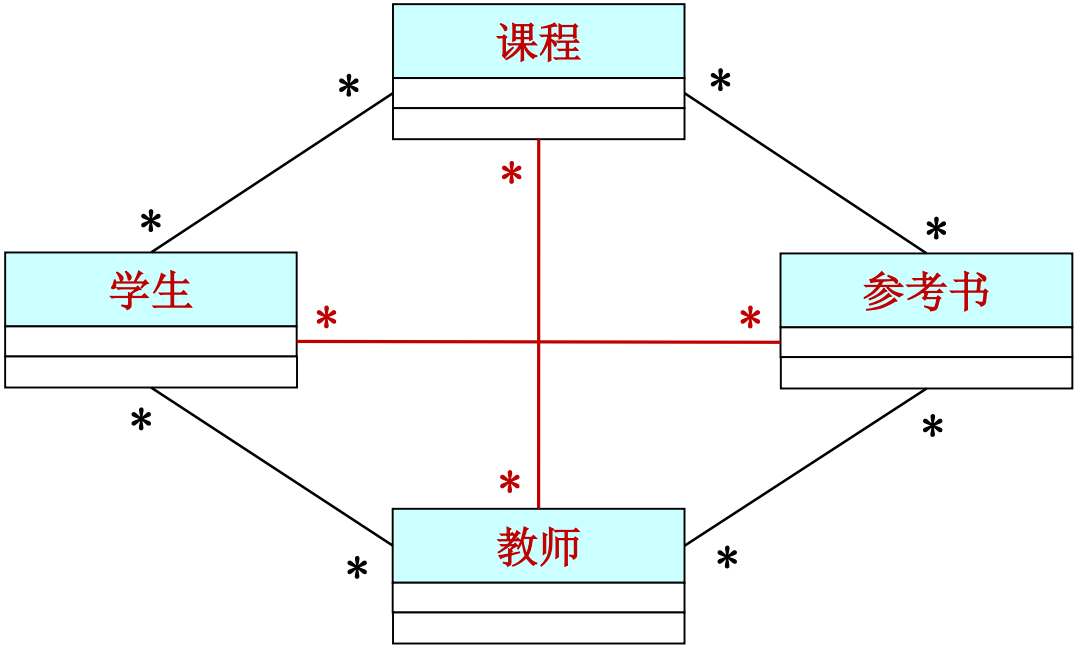
实体类图→ERD映射　思考二：大循环 *:* 关联关系



关系：教师 -指导- 学生 -选修- 课程 -选用- 参考书 -编著- 教师

数据库设计

实体类图→ERD映射　思考三：循环交叉 *:* 关联关系



大循环：　教师 -指导- 学生 -选修- 课程 -选用- 参考书 -编著- 教师

小循环1：教师 -指导- 学生 -选修- 课程 -讲授- 教师

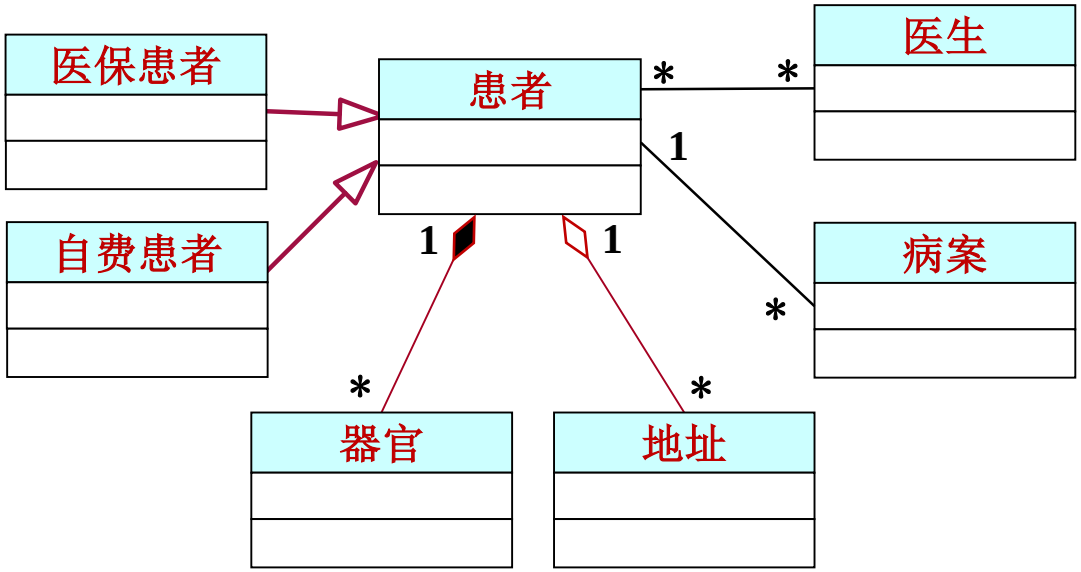
小循环2：教师 -讲授- 课程 -选用- 参考书 -编著- 教师

小循环3：教师 -指导- 学生 -购买- 参考书 -编著- 教师

小循环4：学生 -选修- 课程 -选用- 参考书 -购买- 学生

数据库设计

实体类图→ERD映射　思考四：多种关系组合



患者：分为医保患者或自费患者，用药范围/结账方式不同

患者：有自然一体的多个器官，有多个隶属关系的地址

患者：可以找多位医生诊病，也可能1次都未去过医院

医生：可以给多位患者诊疗，也可能1次都未做过诊疗

患者：可以有多份或没有病案，每份病案只能对应1位患者

本章主要内容

1. 数据库系统及关系数据库简介
2. 数据库逻辑模型设计
3. ERD模型及质量评价
4. 物理数据库设计及建立
5. 物理数据库提高效率的技巧
6. 描述分布式数据库的不同的体系结构模型
7. OO实体类图映射到ERD
8. 关系型数据库规范化示例

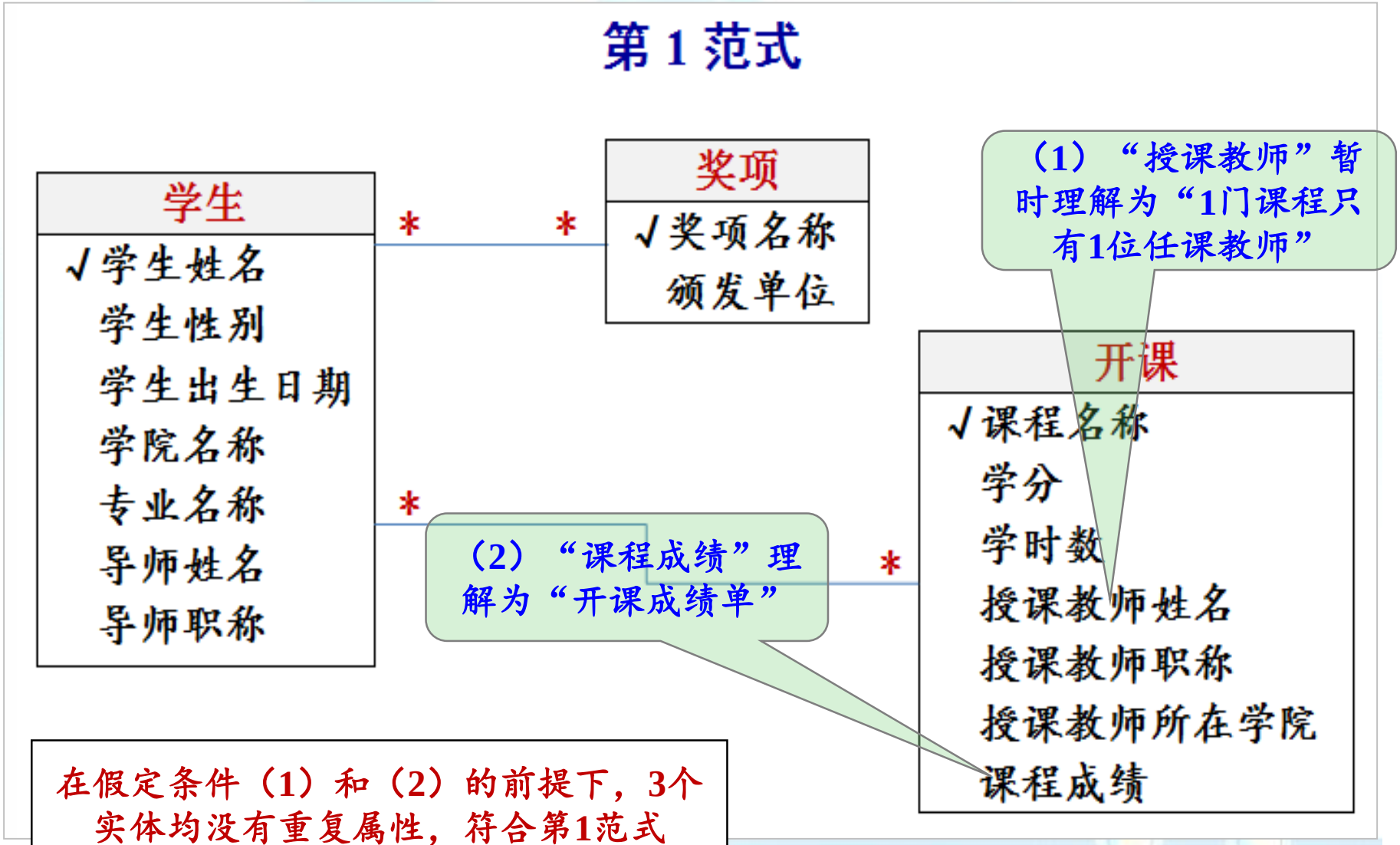
数据库设计

数据库规范化例子

第 0 范式

学生综合信息
√学生姓名
学生性别
学生出生日期
学院名称
专业名称
获奖名称(*)
颁发单位(*)
导师姓名
导师职称
课程名称(*)
学分(*)
学时数(*)
授课教师姓名(*)
授课教师职称(*)
授课教师所在学院(*)
课程成绩(*)

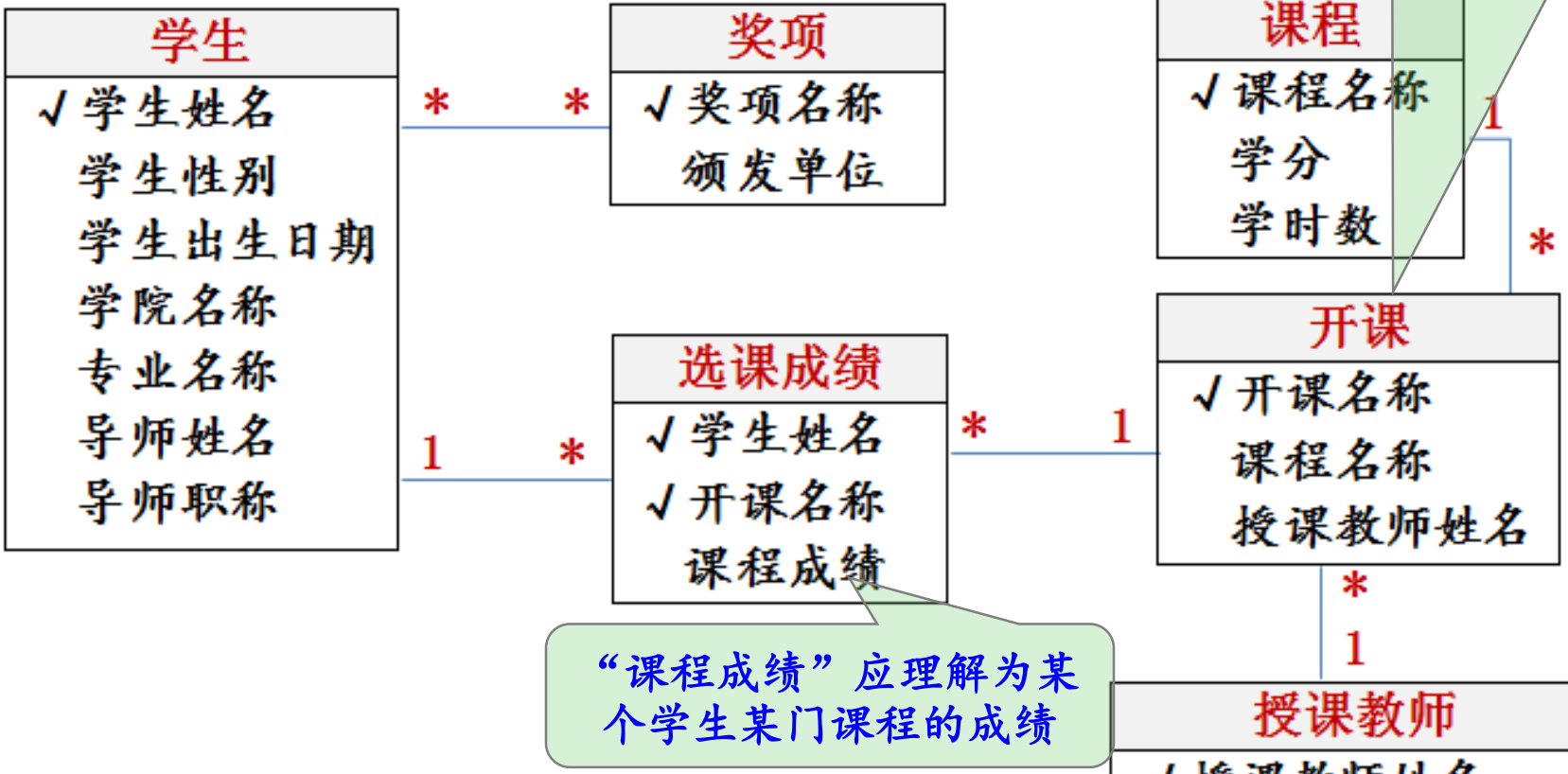
数据库规范化例子


第 1 范式
学生
√学生姓名
学生性别
学生出生日期
学院名称
专业名称
导师姓名
导师职称
奖项
√奖项名称
颁发单位
开课
√课程名称
学分
学时数
授课教师姓名
授课教师职称
授课教师所在学院
课程成绩
*
*
*
*
（1）“授课教师”暂时理解为“1门课程只有1位任课教师”
（2）“课程成绩”理解为“开课成绩单”
在假定条件（1）和（2）的前提下，3个实体均没有重复属性，符合第1范式

数据库设计

数据库规范化例子

第 2 范式

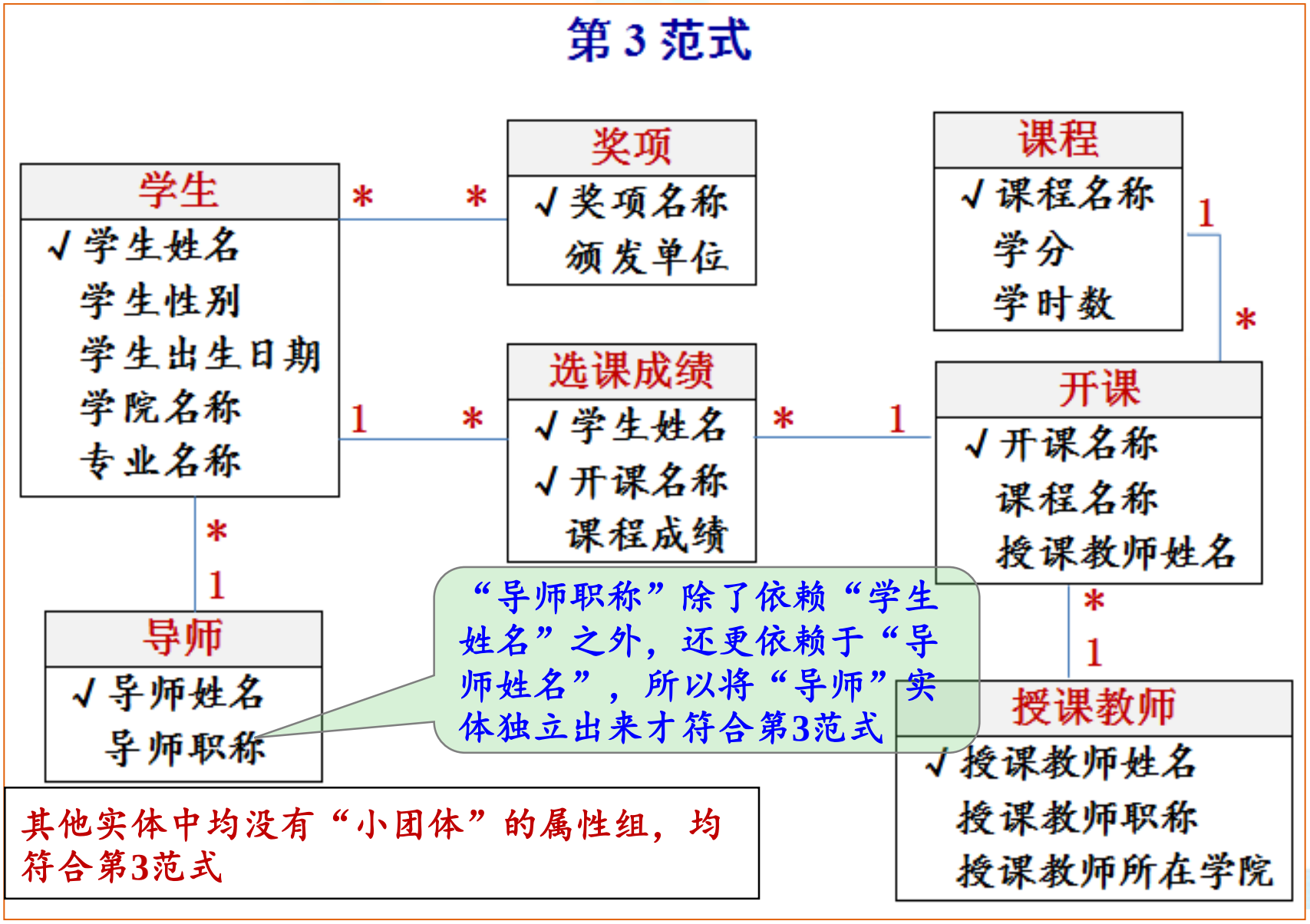


考虑“课程”与“授课教师”之间应该是“*:*”的关系才符合实际情况，增加一个关联实体才能跟“学生”建立关联关系

“课程成绩”应理解为某个学生某门课程的成绩

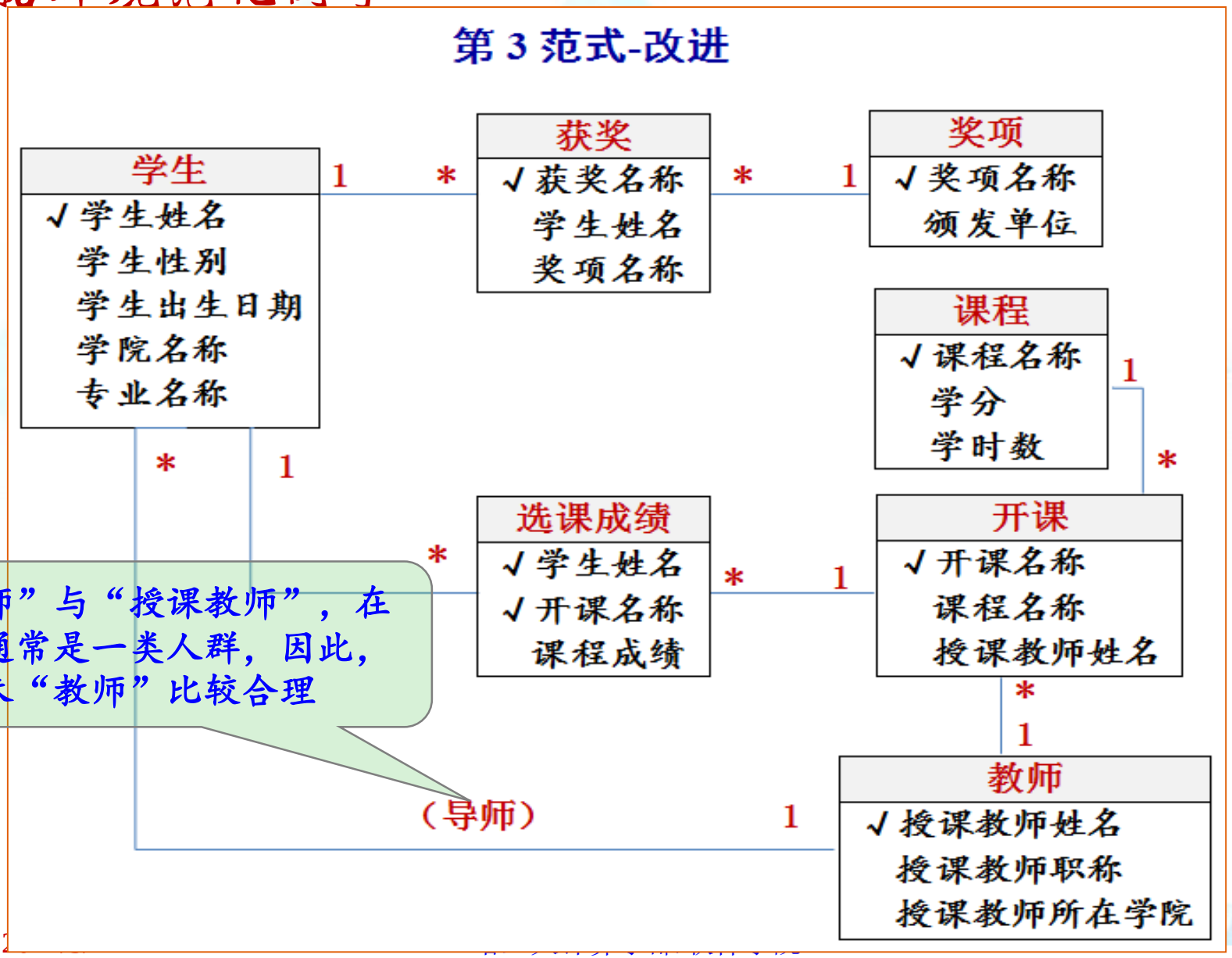
每个实体均有主属性（组合主属性），而且其他属性均依赖该属性，符合第2范式

数据库规范化例子


第 3 范式
学生
√ 学生姓名
学生性别
学生出生日期
学院名称
专业名称
奖项
√ 奖项名称
颁发单位
课程
√ 课程名称
学分
学时数
选课成绩
√ 学生姓名
√ 开课名称
课程成绩
开课
√ 开课名称
课程名称
授课教师姓名
导师
√ 导师姓名
导师职称
授课教师
√ 授课教师姓名
授课教师职称
授课教师所在学院
“导师职称”除了依赖“学生姓名”之外，还更依赖于“导师姓名”，所以将“导师”实体独立出来才符合第3范式
其他实体中均没有“小团体”的属性组，均符合第3范式

数据库设计

数据库规范化例子



数据库设计

数据库规范化例子