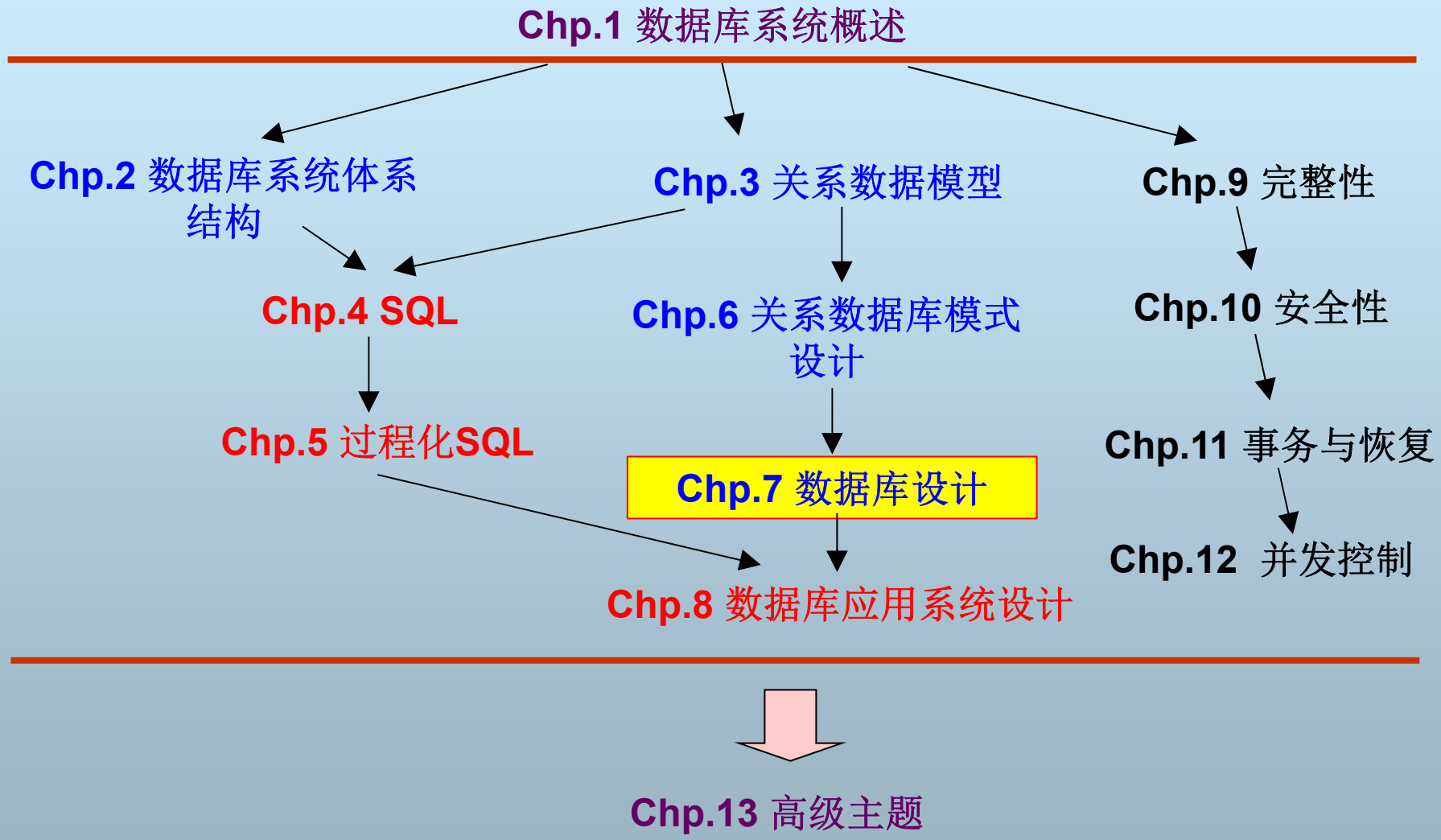
第7章 数据库设计

金培权（jpq@ustc.edu.cn）

课程知识结构


Chp.1 数据库系统概述
Chp.2 数据库系统体系结构
Chp.3 关系数据模型
Chp.9 完整性
Chp.4 SQL
Chp.6 关系数据库模式设计
Chp.10 安全性
Chp.5 过程化SQL
Chp.7 数据库设计
Chp.11 事务与恢复
Chp.12 并发控制
Chp.8 数据库应用系统设计
Chp.13 高级主题

一、什么是数据库设计

- 对于给定的应用环境，构造最合适的数据库模式，并利用现成的DBMS，建立数据库及其应用系统，使之能够有效地存储数据，满足各种用户的需求
  - 面向特定应用
  - 逻辑设计
  - 物理设计

二、数据库设计方法

- 数据库设计是一种方法而不是工程技术，缺乏科学的方法论支持，很难保证质量
- 规范化设计方法：运用软件工程的思想方法进行数据库设计
  - 新奥尔良方法（New Orleans）
    - 需求分析、概念设计、逻辑设计、物理设计
  - 基于ER模型的方法
  - 基于关系模式的设计方法
  - 基于3NF的设计方法
  - 计算机辅助数据库设计方法

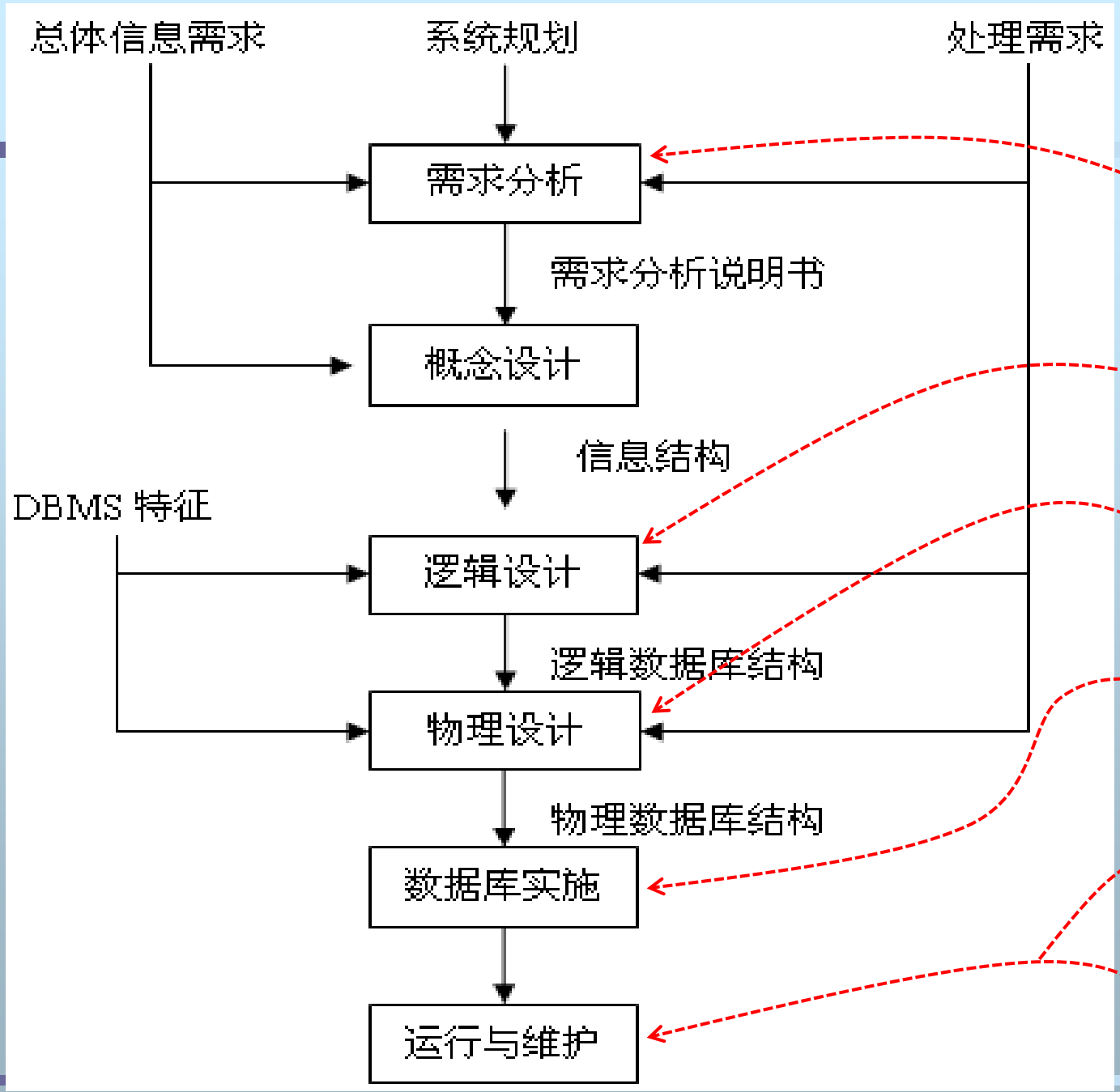
数据库设计不同阶段上的具体实现技术和方法

我们的选择

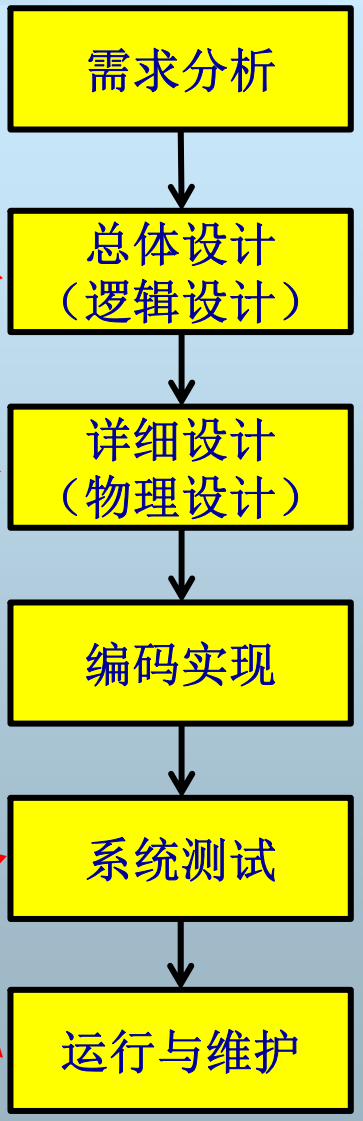
- 以新奥尔良方法为基础，基于ER模型和关系模式，采用计算机辅助进行数据库设计
  - 概念设计：基于ER模型
  - 逻辑设计：基于关系模式设计
  - 计算机辅助设计工具
    - ERWIN（CA）
    - Power Designer（Sybase, now SAP）
    - Workbench（MySQL）
    - Visible Analyst（Visible）
    - Navicat Data Modeler（PremiumSoft）
    - ……

三、数据库设计步骤

- 需求分析
- 概念设计
- 逻辑设计
- 物理设计
- 数据库实施
- 数据库运行与维护


总体信息需求
系统规划
处理需求
需求分析
需求分析说明书
概念设计
信息结构
DBMS 特征
逻辑设计
逻辑数据库结构
物理设计
物理数据库结构
数据库实施
运行与维护

对比：软件工程


需求分析
总体设计
（逻辑设计）
详细设计
（物理设计）
编码实现
系统测试
运行与维护

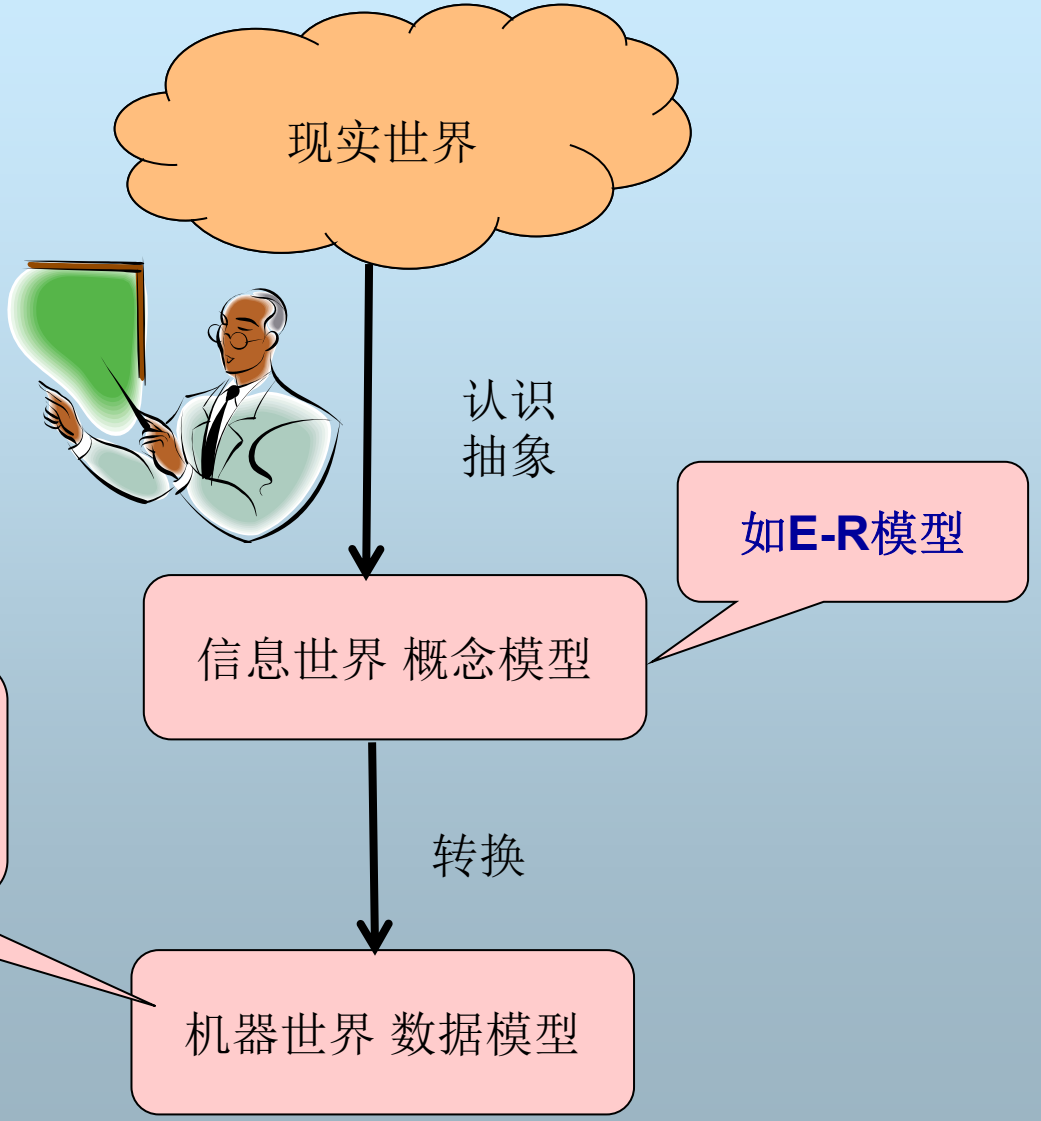
输入输出

- 输入：总体信息需求、处理需求、DBMS特征
  - 总体信息需求：数据库应用系统的目标、数据元素的定义、数据在组织中的使用描述
  - 处理需求：每个应用需要的数据项、数据量以及处理频率
  - DBMS特征：DBMS说明、支持的模式、程序语法
- 输出：数据库设计说明书（完整的数据库逻辑结构和物理结构、应用程序设计说明）

四、概念设计（ER模型设计）

- 产生反映组织信息需求的数据库概念结构，即概念模型
  - 概念模型独立于数据库逻辑结构、DBMS以及计算机系统
  - 概念模型以一组ER图形式表示
- 概念设计侧重于数据内容的分析和抽象，以用户的观点描述应用中的实体以及实体间的联系

数据抽象的层次


现实世界
认识
抽象
如E-R模型
信息世界 概念模型
如关系模型、层次模型、网状模型、面向对象模型
转换
机器世界 数据模型

1、ER模型的概念

- **ER模型(Entity-Relationship Model)**
  - **1976，Peter .P. Chen（陳品山）提出的概念设计方法**
  - **以ER图的方式表达现实世界实体及实体间的联系**

Louisiana State University

Peter Chen. The Entity-Relationship Model--Toward a Unified View of Data. *ACM Transactions on Database Systems*, Vol. 1(1), p.9-36,1976

One of the 38 most influential papers in Computer Science

1、ER模型的概念

- **ER模型要素**
  - **实体 Entity**
    - 包含实体属性
  - **实体与实体间的联系 Relationship**
    - 包含联系类型和联系属性

（1）实体与联系

- 实体（Entity）
  - 现实世界中可标识的对象
  - 如学生、学校、发票、教室、系、班级……
  - 物理存在的实体（教室、学生）、代表抽象概念的实体（课程）
  - 应用中的数据以实体的形式呈现
  - 一个实体具有唯一的标识，称为码（Key）
- 联系（Relationship）
  - 实体和实体之间发生的关联
  - 一个实体一般都与一个或多个实体相关

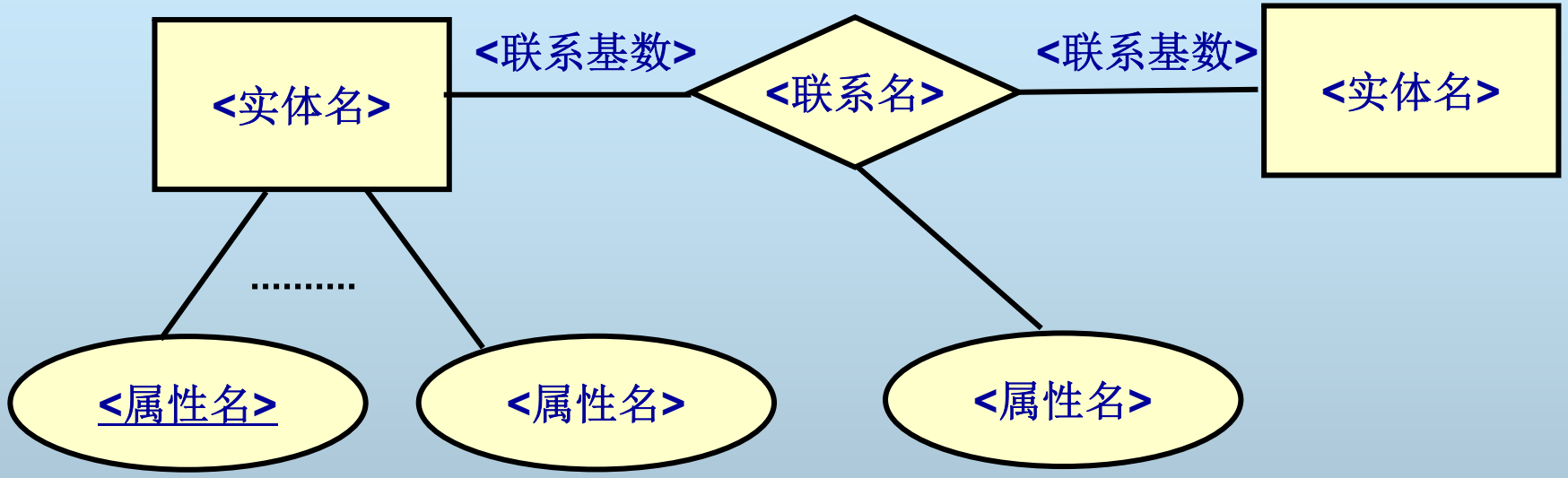
（2）联系的类型

- 1对1联系（1:1）
  - 学校和校长、学生和学生简历……
  - A和B是1:1联系指一个A只有一个B与其关联，并且一个B也只有一个A与其关联
- 1对多联系（1:N）
  - 公司和职工、系和学生、客户和订单……
  - A和B是1:N联系指一个A可以有多个B与其关联，但一个B只有1个A关联
- 多对多联系（M:N）
  - 学生和课程、教师和课程、医生和病人……
  - 一个A可有多个B对应，一个B也可有多个A对应

（3）联系的确定

- 联系的确定依赖于实体的定义和特定的应用，同样的实体在不同应用中可能有不同的联系
  - 部门和职工：若一个职工只能属于一个部门，则是1:N，若一个职工可属于多个部门，则是M:N
  - 图书馆和图书：若图书的码定义为索书号，则为M:N（一个索书号可能有几本相同的书）；若图书的码为图书条码，并且每本书有一个唯一条码，则为1:N联系

（4）ER图的符号

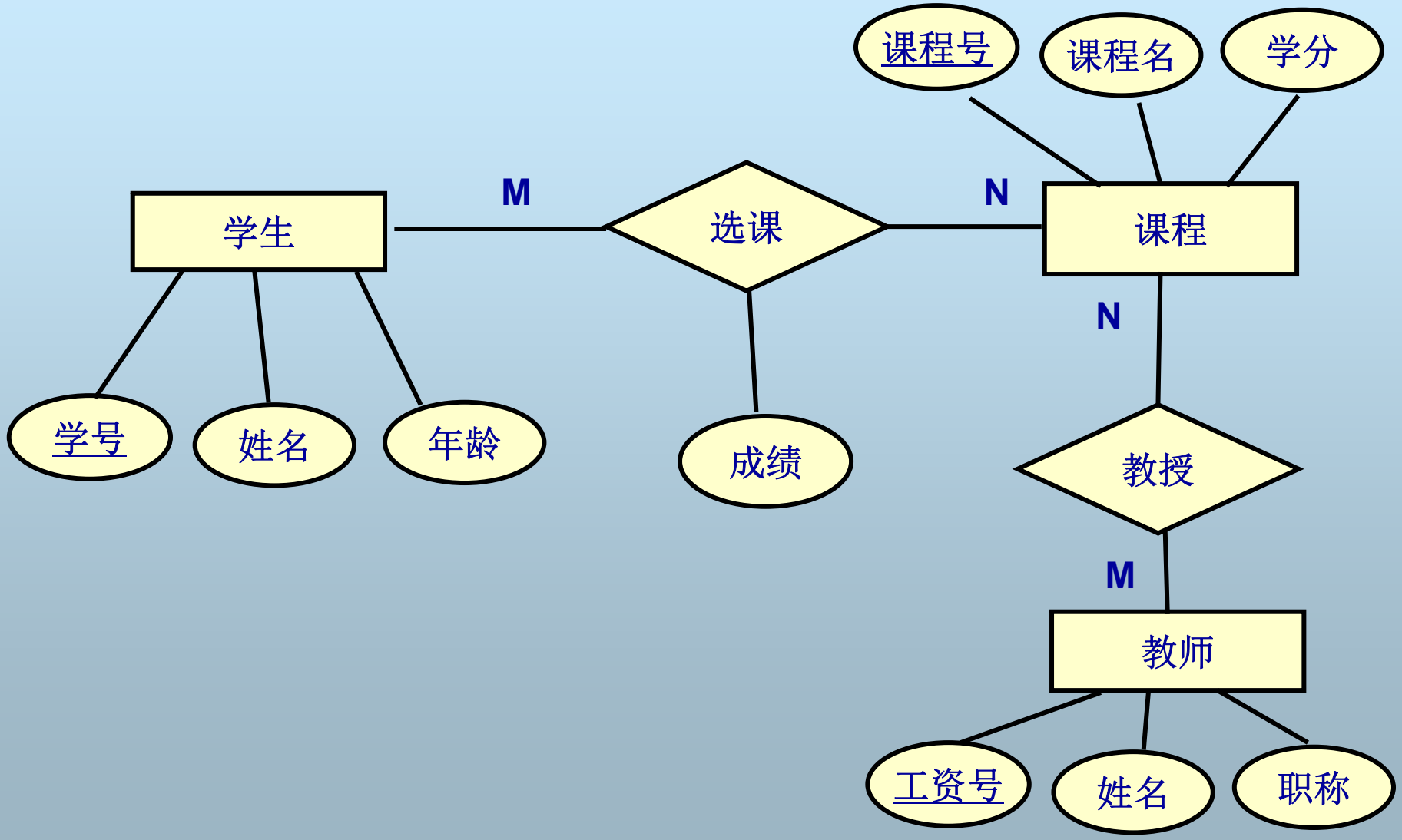


矩形：表示实体
菱形：表示联系，两端写上联系的基数

（1:N, M:N, 1:1）

椭圆形：表示属性，实体的码加下划线，联系也可有属性

（5）ER图例子：教学应用

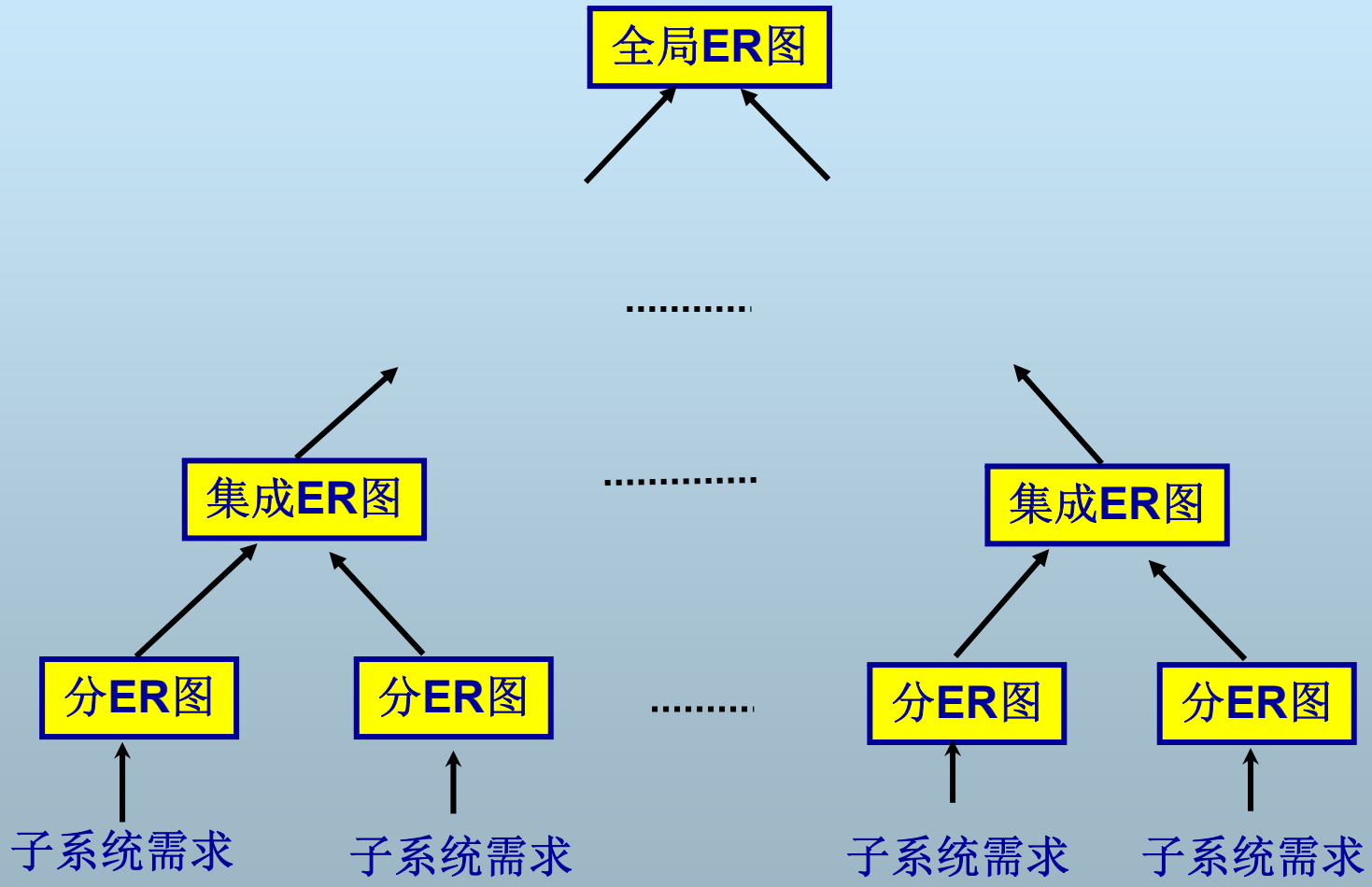


2、ER设计的步骤

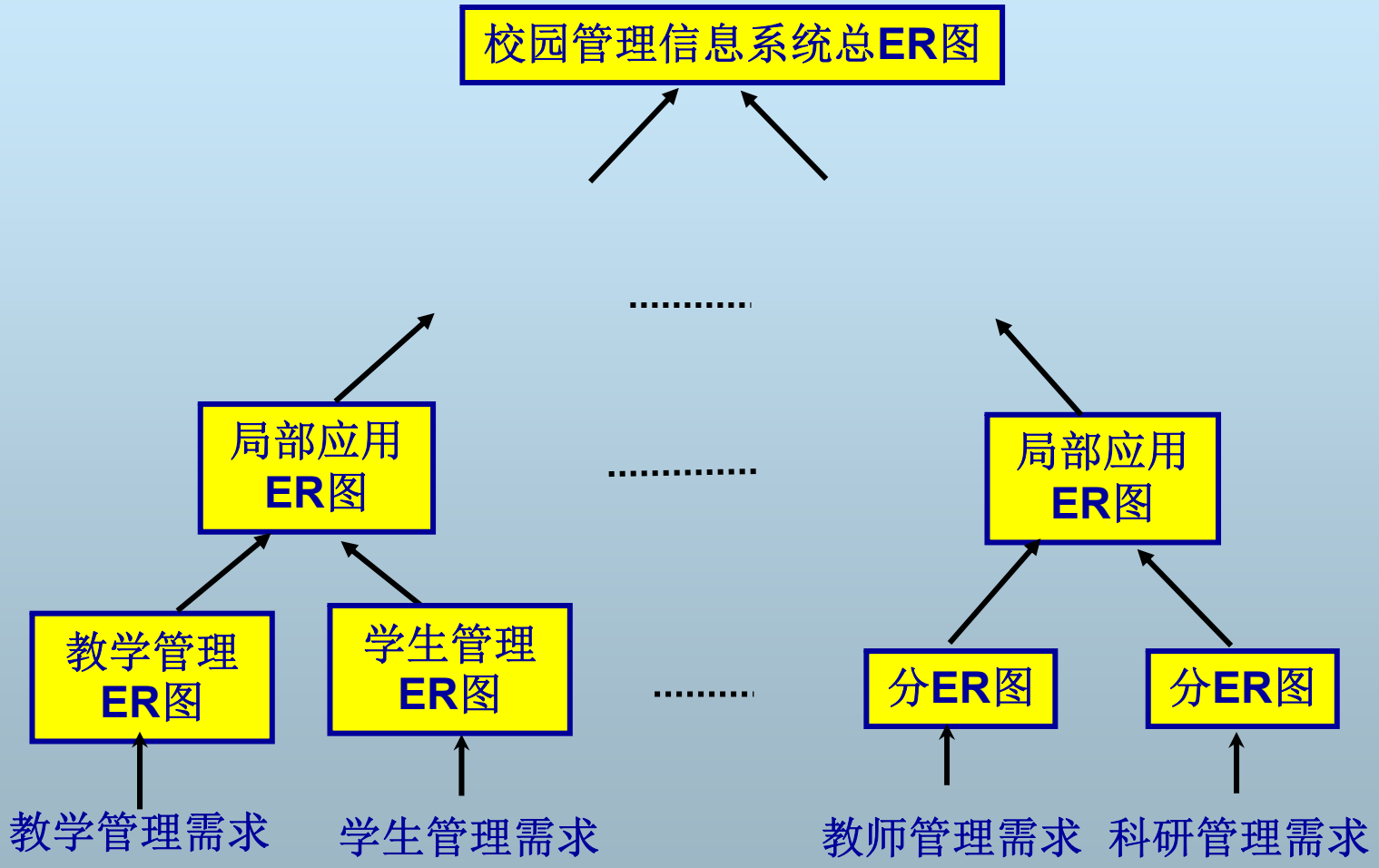
- 自顶向下进行需求分析，自底向上进行ER设计
  - 分ER模型设计（局部ER图）
  - ER模型集成
  - ER模型优化

如果应用比较简单则可以合为一个步骤

(1)ER设计的步骤示意


全局ER图
集成ER图
集成ER图
分ER图
分ER图
分ER图
分ER图
子系统需求
子系统需求
子系统需求
子系统需求

（2）ER设计步骤例子


校园管理信息系统总ER图
局部应用
ER图
局部应用
ER图
教学管理
ER图
学生管理
ER图
分ER图
分ER图
教学管理需求
学生管理需求
教师管理需求
科研管理需求

（3）分ER设计

- 通过实体、联系和属性对子系统的数据进行抽象，产生分ER图
  - 确定实体
  - 确定实体属性
  - 确定联系和联系属性
- 设计原则
  - 实体要尽可能得少
  - 现实世界中的事物若能作为属性就尽量作为属性对待

A）确定实体

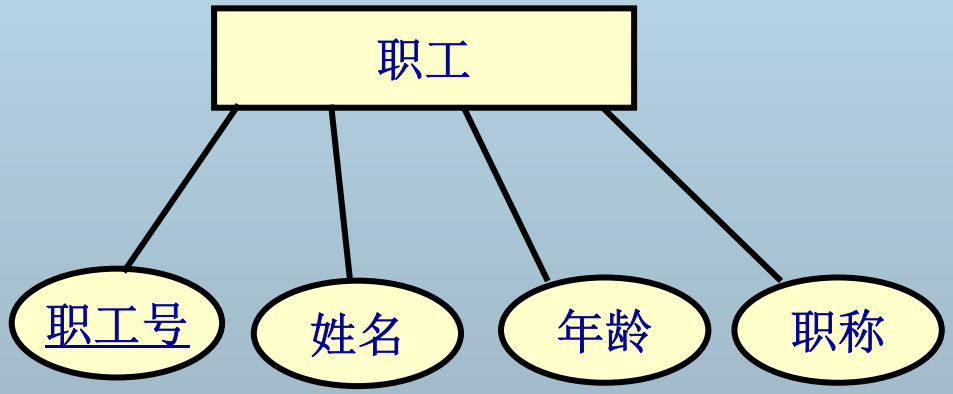
- 实体是一个属性的集合
- 需求分析阶段产生的数据字典中的数据存储、数据流和数据结构一般可以确定为实体
  - 数据字典五个部分：数据项、数据结构、数据流、数据存储和数据处理

B）确定实体属性

- 实体和属性之间没有形式上可以截然划分的界限
  - 首先确定实体的码
  - 只考虑系统范围内的属性
  - 属性应具有域
  - 属性一般要满足下面的准则
    - 属性必须不可分，不能包含其它属性
    - 属性不能和其它实体具有联系

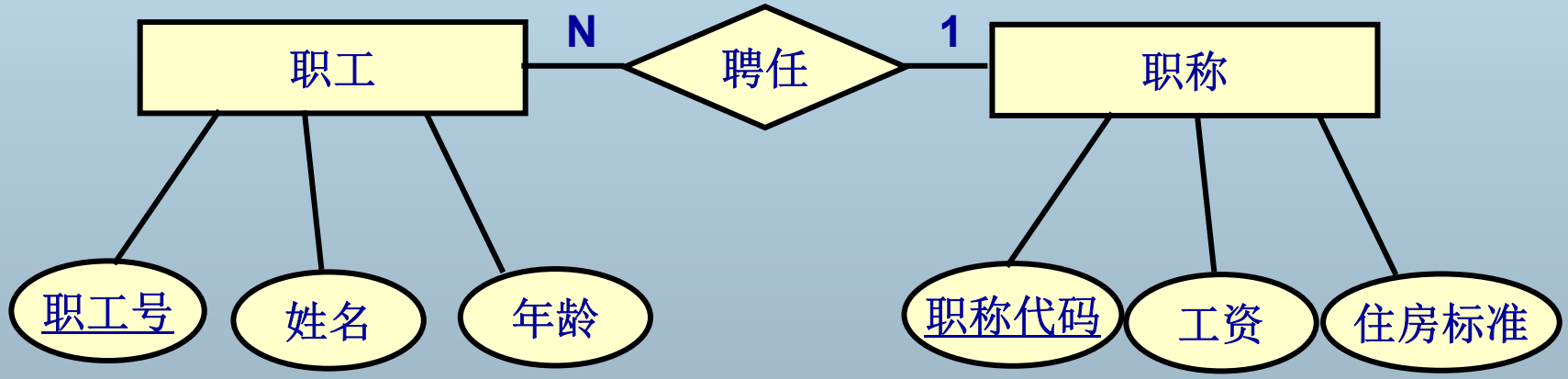
属性设计例子1

- 职工是一个实体，职工号、姓名、年龄是职工的属性，如果职工的职称没有进一步的特定描述，则可以作为职工的属性



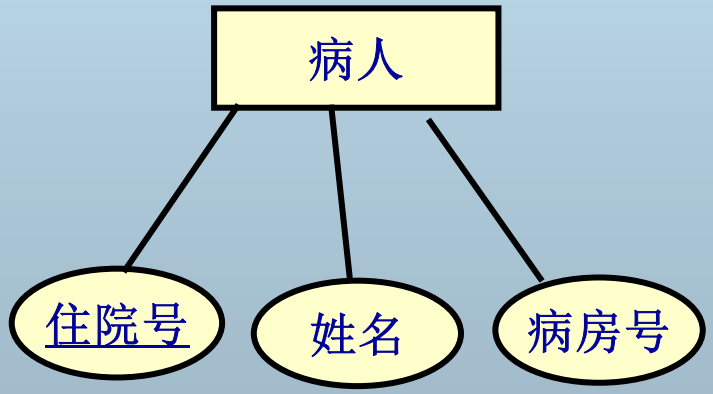
属性设计例子1

- 如果职称与工资、福利等挂钩，即职称本身还有一些描述属性，则把职称设计为实体比较恰当



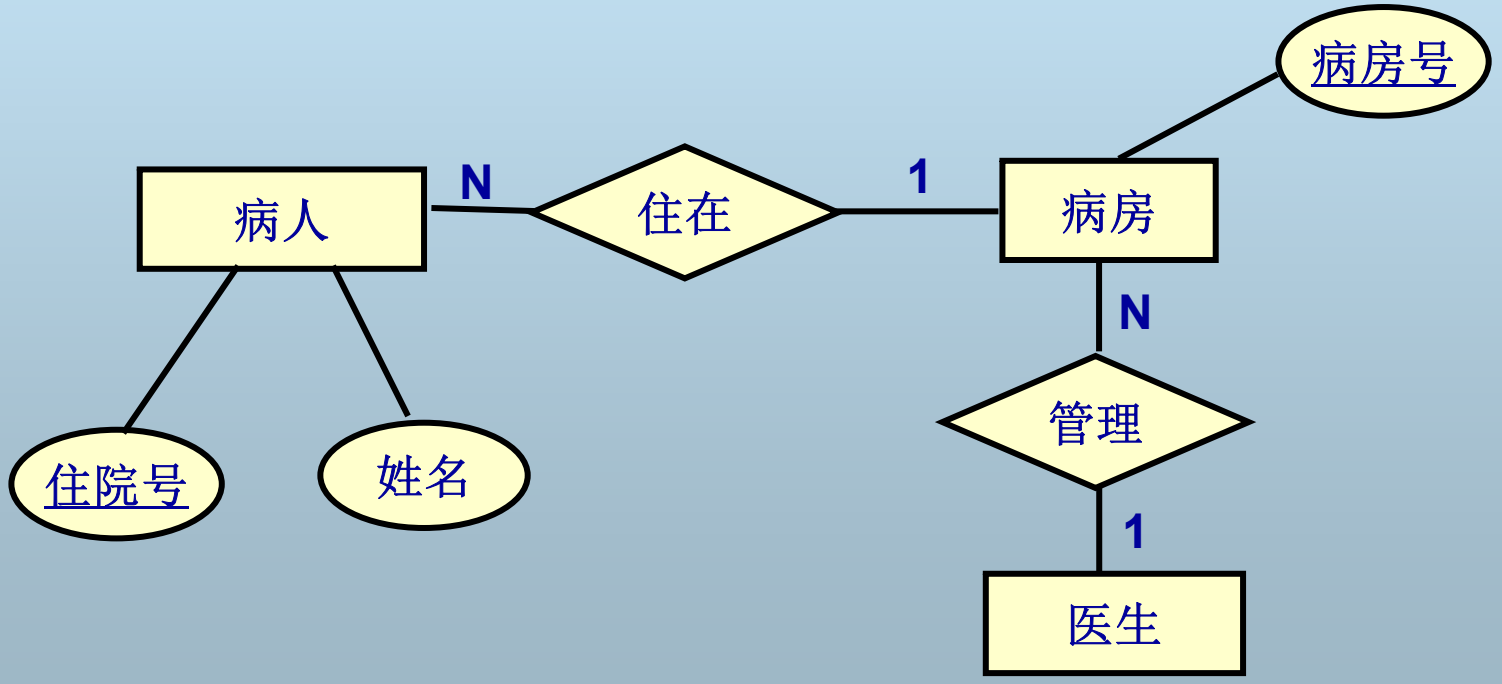
属性设计例子2

- 医院管理中，一个病人只能住在一个病房里，因此病房号可以作为病人实体的一个属性



属性设计例子2

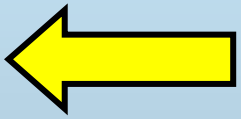
- 但如果病房与医生实体存在负责联系，即一个医生要负责管理多个病房，而一个病房的管理医生只有一个



C）确定联系和联系属性

- 根据数据需求的描述确定
  - 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}
  - 参考书：“系统分析与设计”或“软件工程”
- 联系的基数
  - 0个或1个（国家和总统：1个国家可以有0个或1个总统）
  - 0个或1个或多个（学院和系）
  - 1个或多个（班级和学生）
  - 1个（公司和法人）
  - 确定的k个（候选人和推荐人：一个候选人必须有3个候选人）

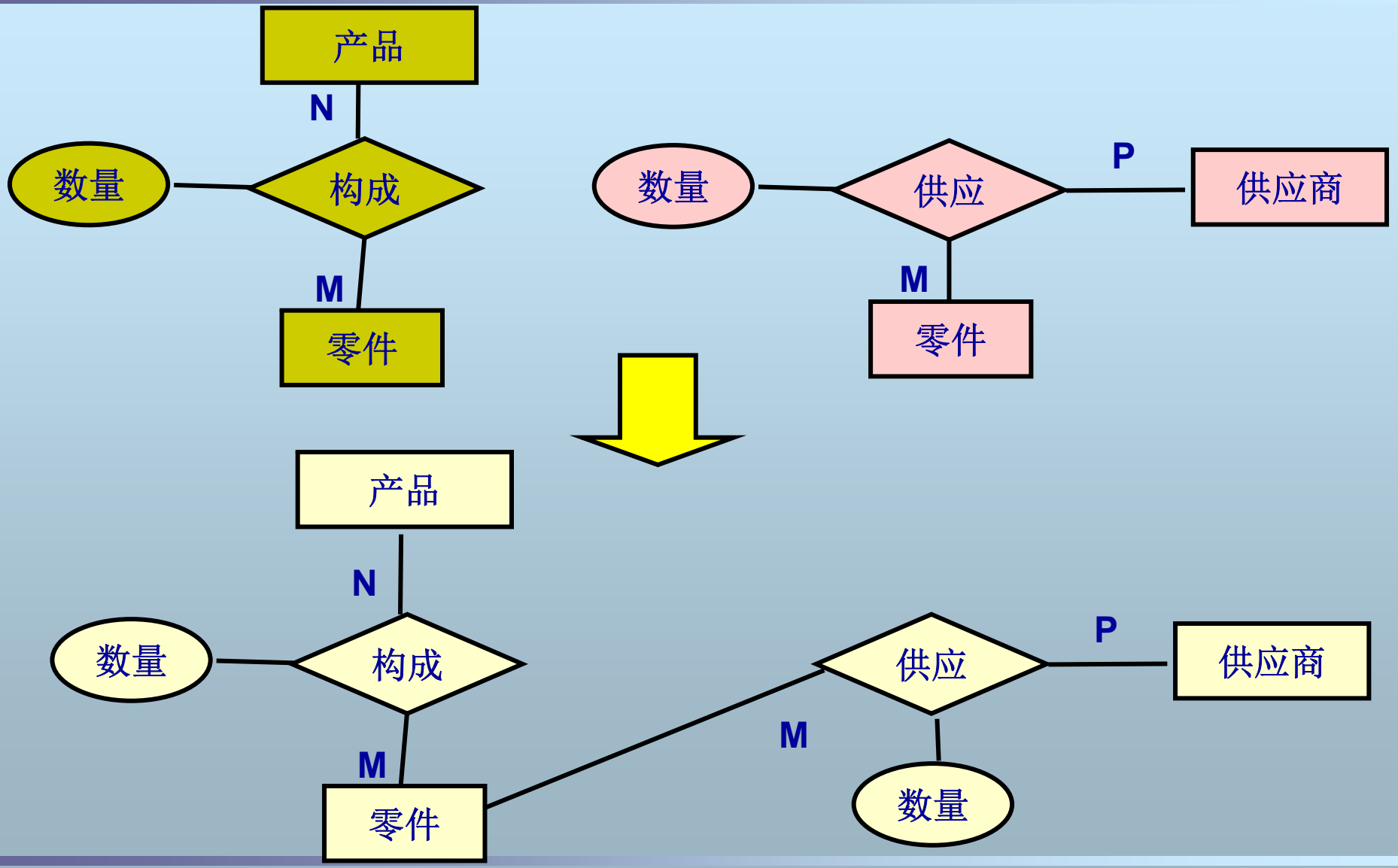
2、ER设计的步骤

- 自顶向下进行需求分析，自底向上进行ER设计
  - 分ER模型设计（局部ER图）
  - ER模型集成 
  - ER模型优化

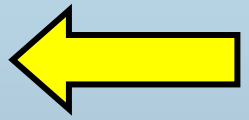
（4）ER集成

- 确定公共实体
- 合并分ER图
- 消除冲突
  - 属性冲突：类型冲突、值冲突
    - 例如性别、年龄
  - 结构冲突：实体属性集不同、联系类型不同、同一对象在不同应用中的抽象不同
  - 命名冲突：同名异义、异名同义
    - 实体命名冲突、属性命名冲突、联系命名冲突

ER集成示例



2、ER设计的步骤

- 自顶向下进行需求分析，自底向上进行ER设计
  - 分ER模型设计（局部ER图）
  - ER模型集成
  - ER模型优化 

（5）ER模型的优化

- 目标
  - 实体个数要少，属性要少，联系尽量无冗余
- 具体优化手段
  - 合并实体类型
  - 消除冗余属性
  - 消除冗余联系

A）合并实体

- 一般1:1联系的两个实体可以合并为一个实体
- 如果两个实体在应用中经常需要同时处理，也可考虑合并
  - 例如病人和病历，如果实际中通常是查看病人时必然要查看病历，可考虑将病历合并到病人实体中
    - 减少了连接查询开销，提高效率

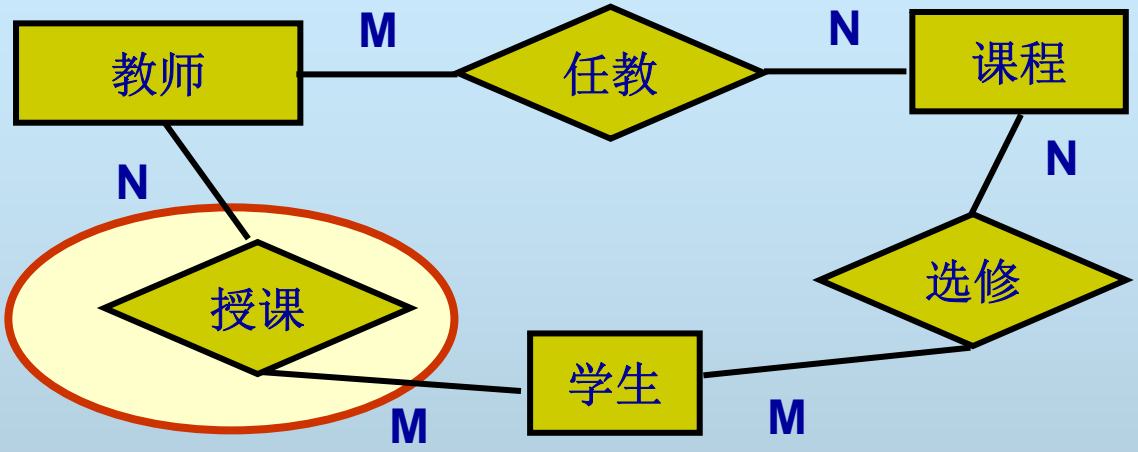
B）消除冗余属性

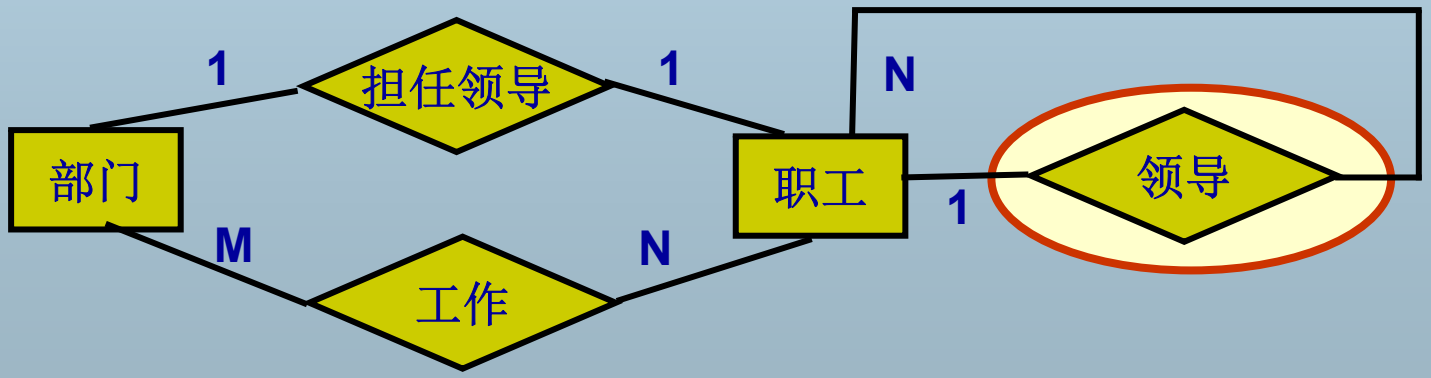
- 分ER图中一般不存在冗余属性，但集成后可能产生冗余属性
  - 例如，教育统计数据库中，一个分ER图中含有高校毕业生数、在校学生数，另一个分ER图中含有招生数、各年级在校学生数
  - 每个分ER图中没有冗余属性，但集成后“在校学生数”冗余，应消除

B）消除冗余属性

- 冗余属性的几种情形
  - 同一非码属性出现在几个实体中
  - 一个属性值可从其它属性值中导出
    - 例如出生日期和年龄

C）消除冗余联系

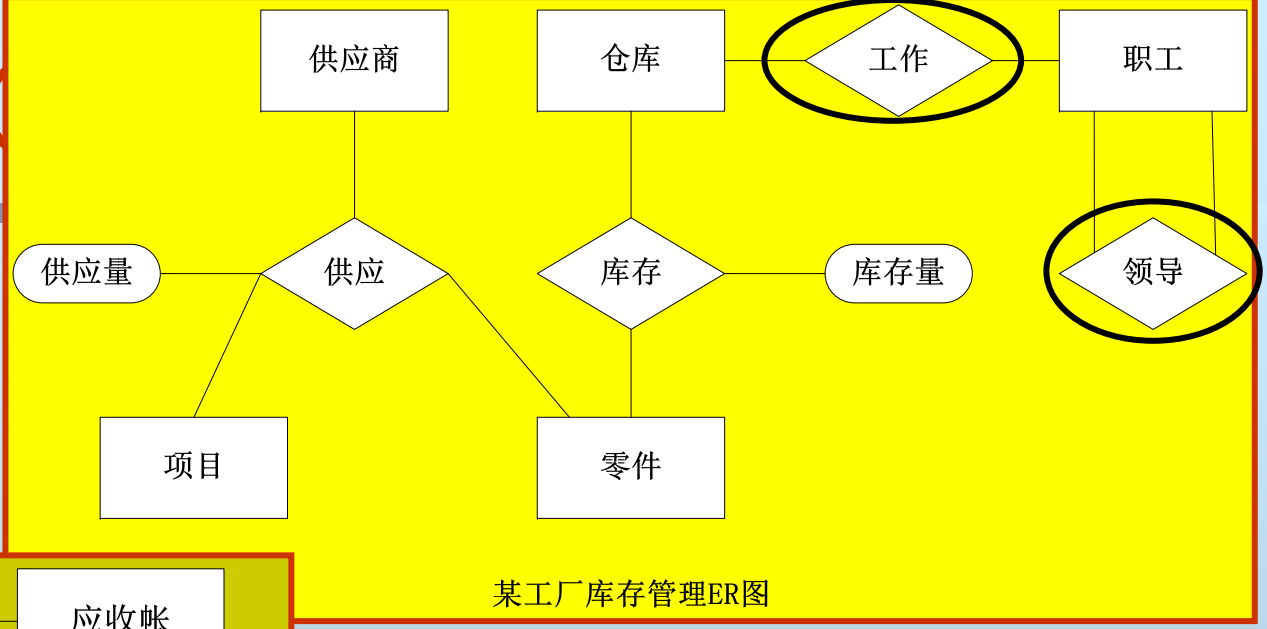




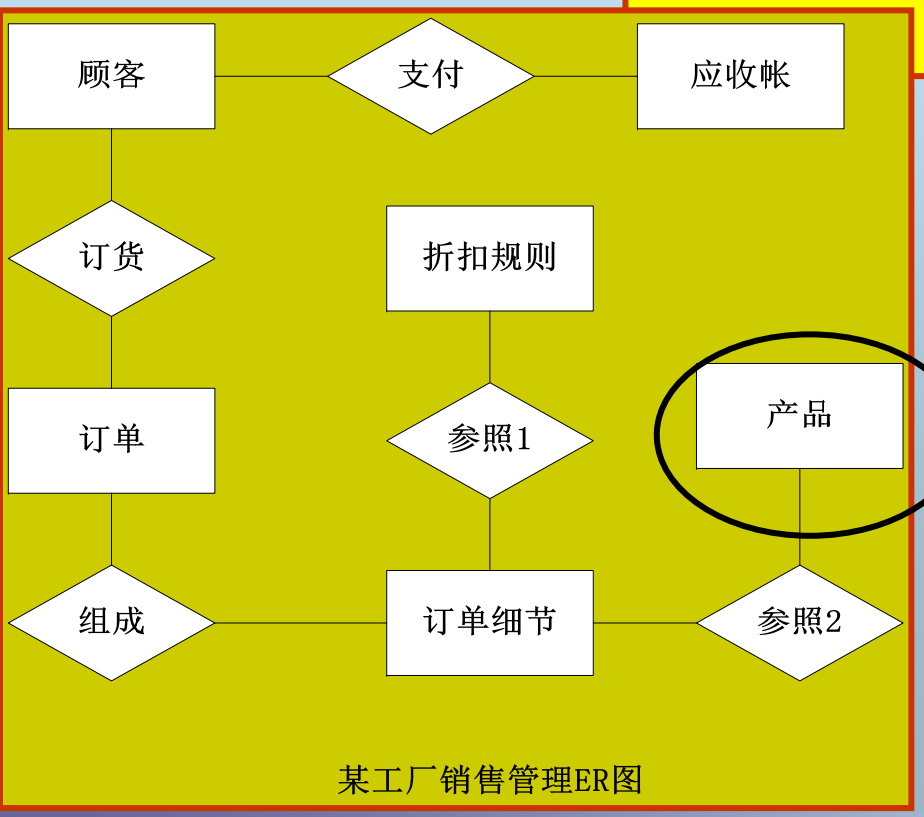
项目＝产品

职工与仓库的联系冗余

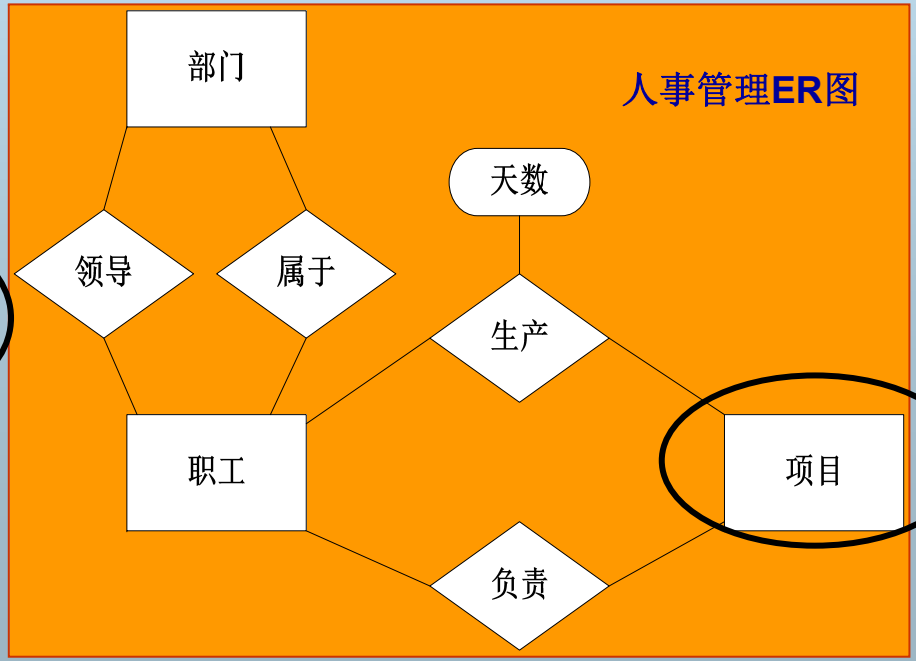
职工与职工的领导联系冗余

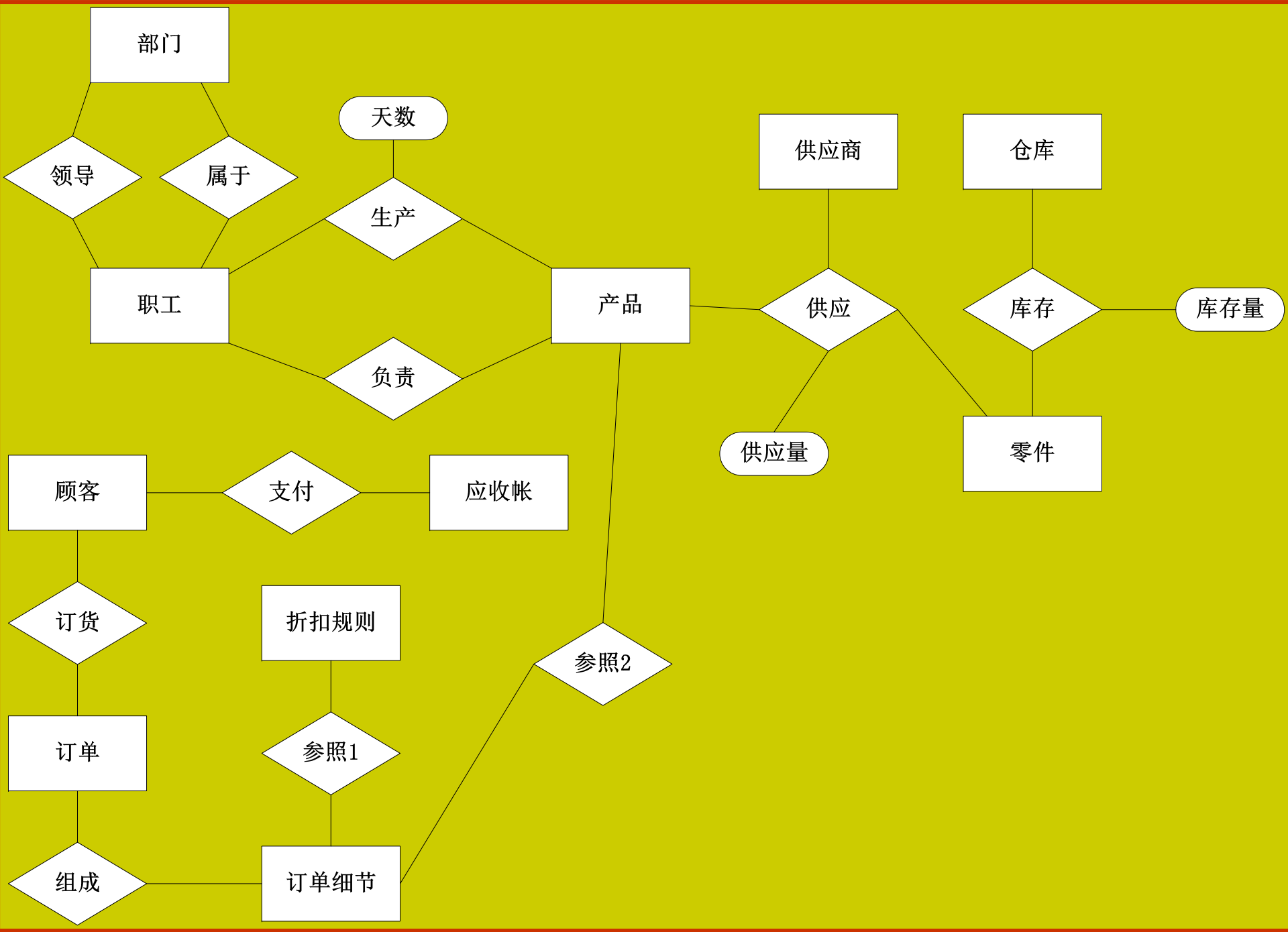


某工厂库存管理ER图



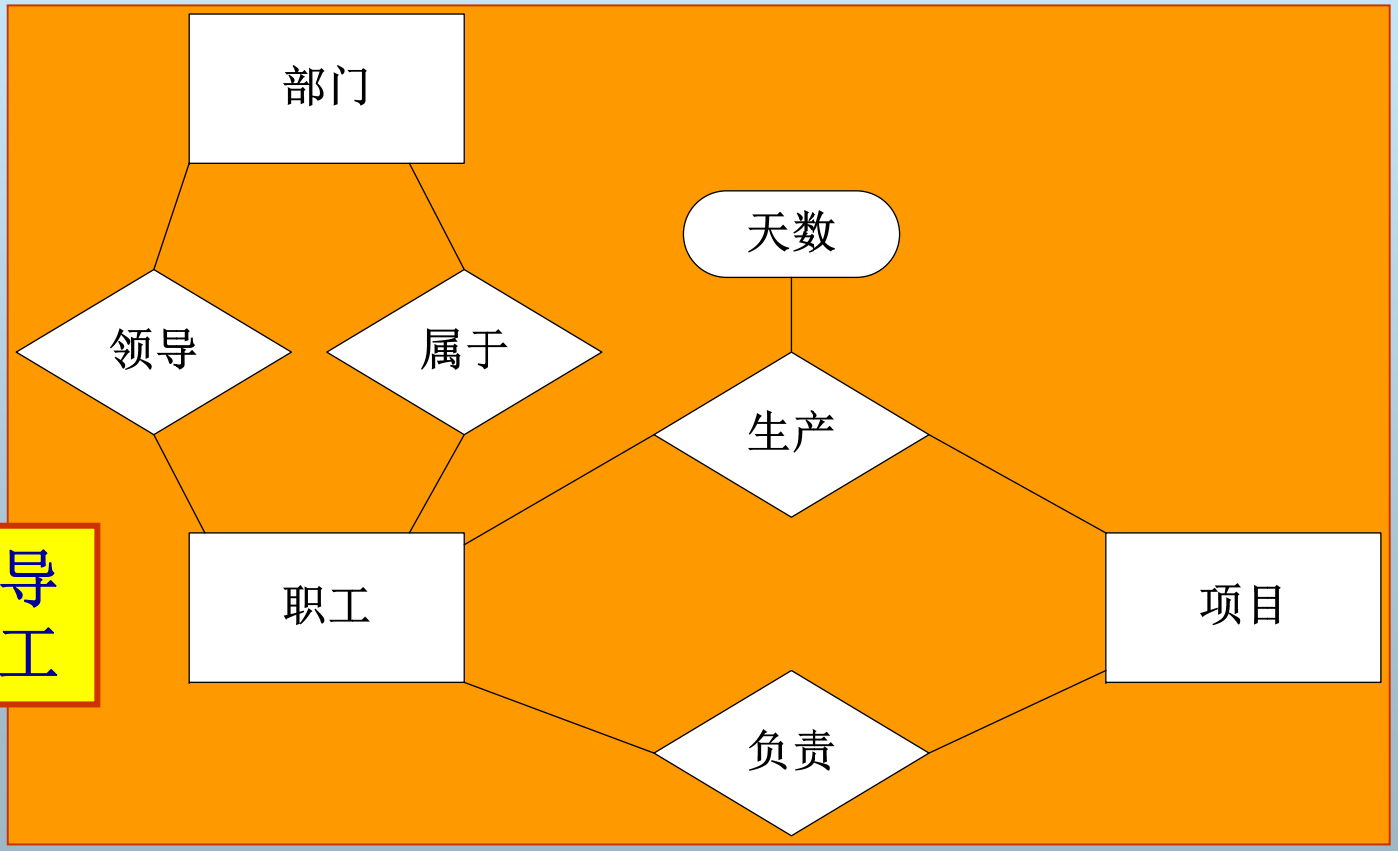
某工厂销售管理ER图




部门
领导
属于
天数
生产
职工
产品
负责
供应商
供应
供应量
仓库
库存
库存量
零件
顾客
支付
应收帐
订货
订单
组成
折扣规则
参照1
参照2
订单细节

3、ER模型的扩展

- 传统的ER模型无法表达一些特殊的语义



无法区分领导者和一般职工

3、ER模型的扩展

- 弱实体
- 子类（特殊化）与超类（一般化）

（1）弱实体（weak entity）

- 一个弱实体的存在必须以另一实体的存在为前提
  - 弱实体所依赖存在的实体称为常规实体（regular entity）或强实体（strong entity）
  - 弱实体有自己的标识，但它的标识只保证对于所依赖的强实体而言是唯一的。在整个系统中没有自己唯一的实体标识

（1）弱实体（weak entity）

- 弱实体的例子
  - 一个公司的人事系统中，需要管理职工和职工的子女信息
  - 子女是弱实体，职工是强实体
  - 是否弱实体要看具体应用：例如在社区人口管理系统中，子女就不是弱实体，即使双亲都不存在了，子女仍应存在于人口系统中

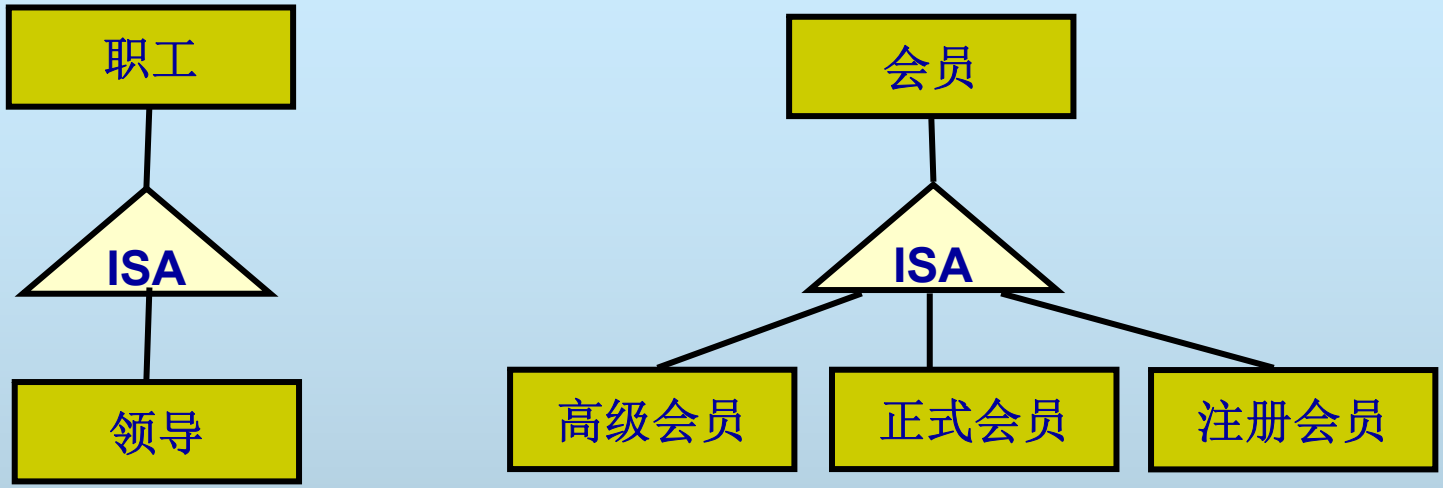
（2）弱实体的表示

- 弱实体用双线矩形表示，存在依赖联系用双线菱形表示，箭头指向强实体

（3）子类（特殊化）与超类（一般化）

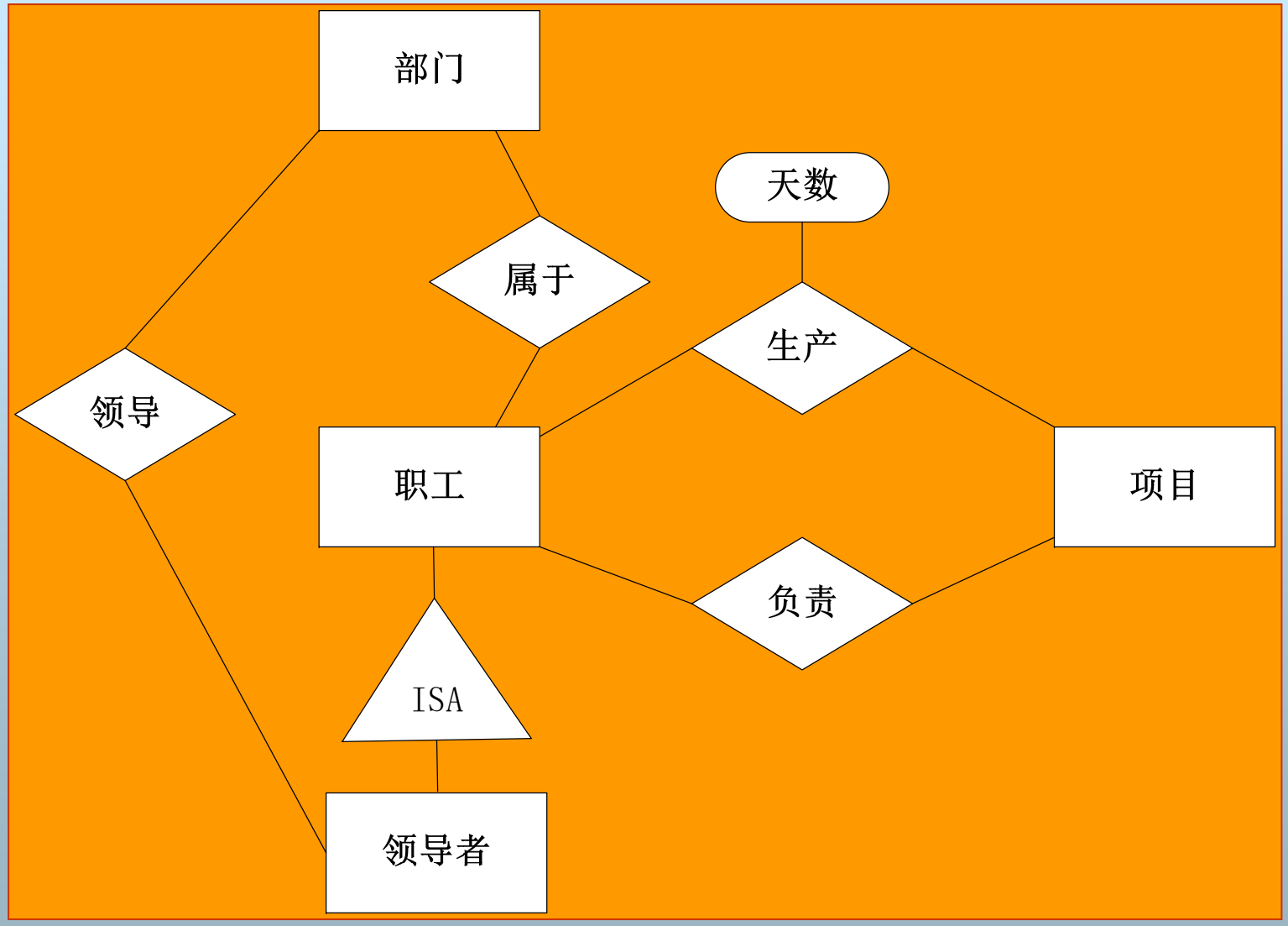
- 子类（Subtype）和超类（Supertype）
  - 两个实体A和B并不相同，但实体A属于实体B，则A称为实体子类，B称为实体超类
  - 子类是超类的特殊化，超类是子类的一般化
  - 子类继承了超类的全部属性，因此子类的标识就是超类的标识
  - 例如，研究生是学生的子类，经理是职工的子类
- 在ER设计时，可以根据实际情况增加子类，也可以根据若干实体抽象出超类

（4）子类符号

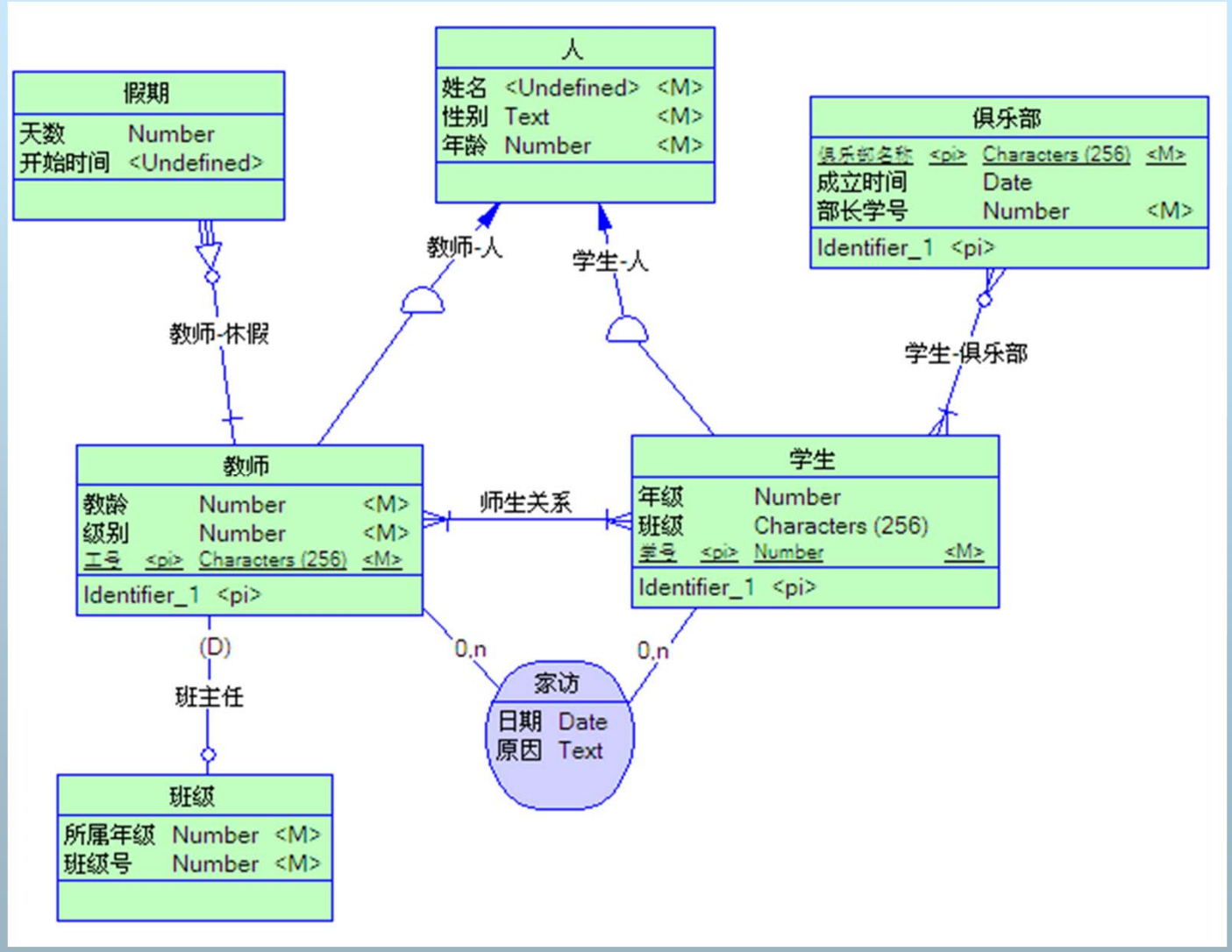


- **ISA表示子类与超类关系**

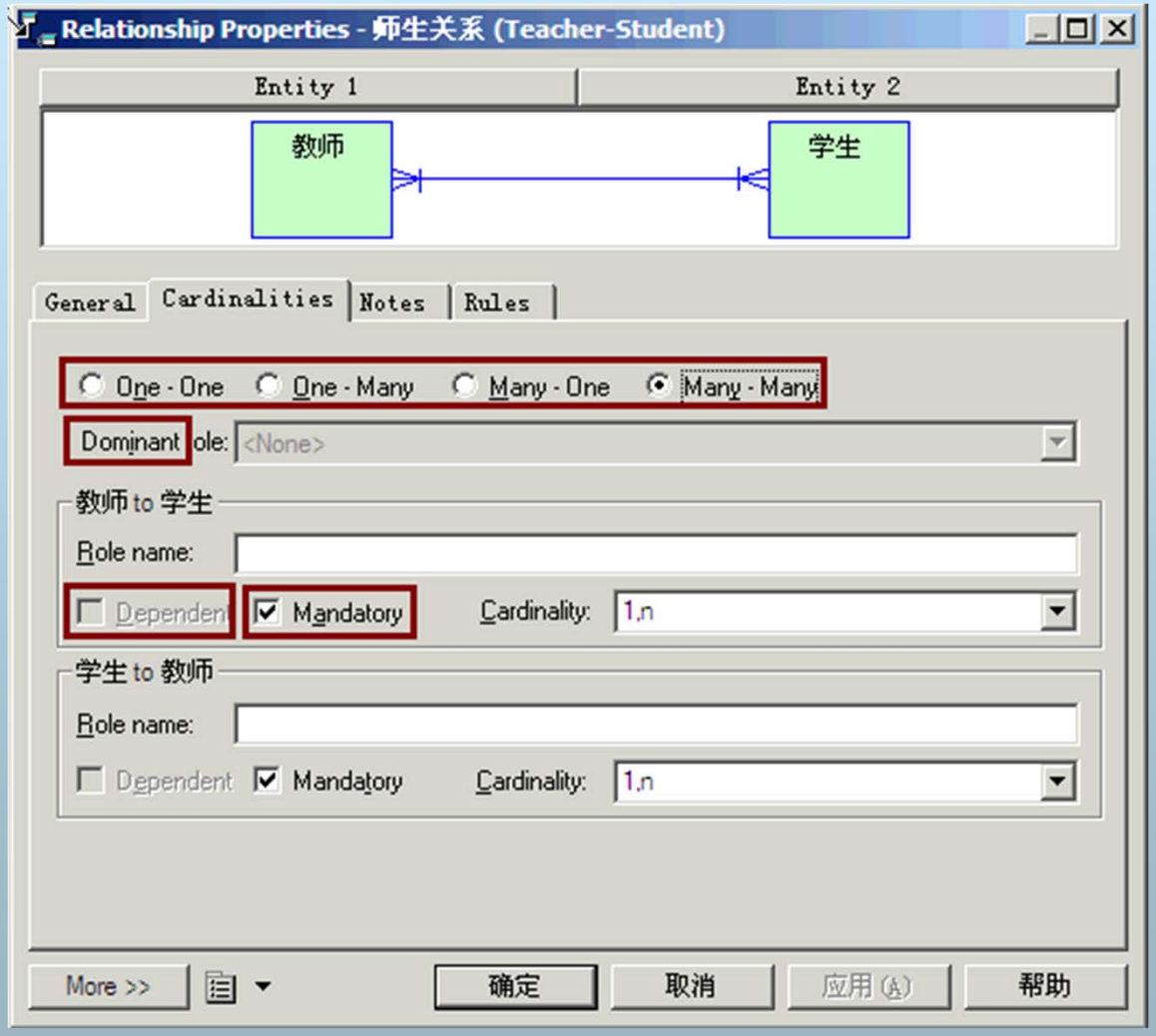
（5）子类例子



Power Designer中的符号



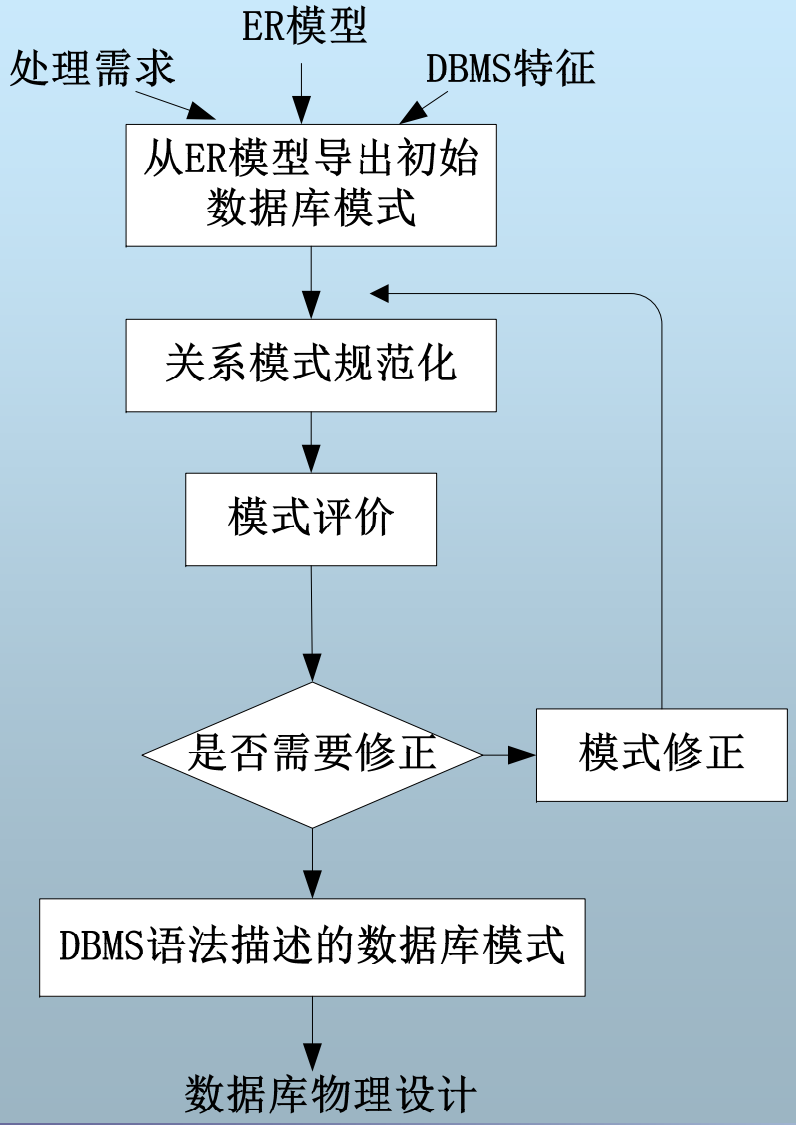
Power Designer中的符号



五、数据库逻辑设计

- 根据概念模型设计出与DBMS支持的数据模型相符合的数据库逻辑结构
- 主要工作
  - ER模型向关系模型的转换
  - 关系模型优化
  - 关系模型修正

1、数据库逻辑设计步骤



① **ER模型转换成关系数据库模式**

② 关系数据库模式的规范化

③ 模式评价

④ 模式修正

⑤ 最终产生一个优化的全局关系数据库模式

⑥ 子模式设计

2、ER模型向关系模型转换

- 基本ER模型的转换
- 扩展ER模型的转换

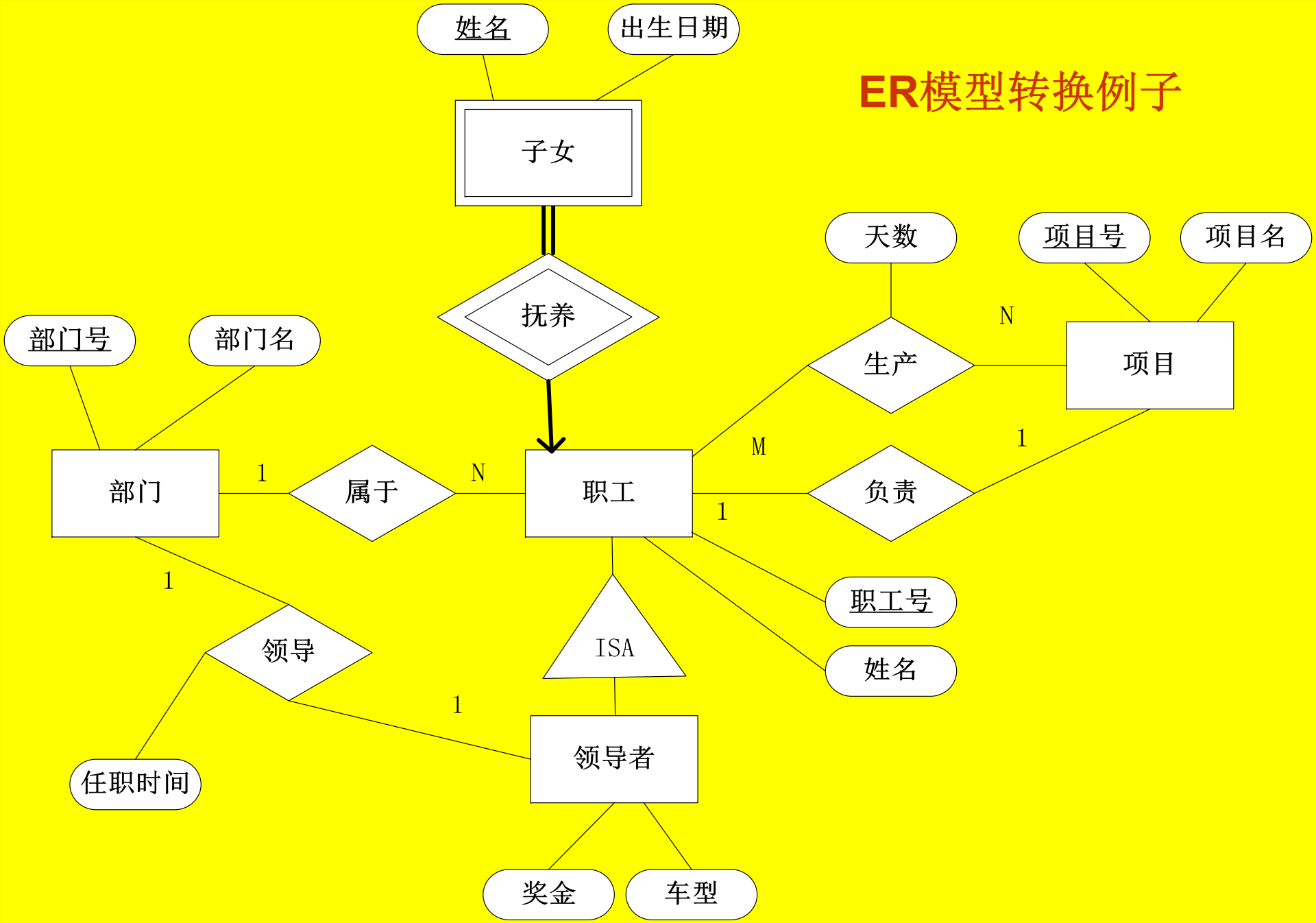
（1）基本ER模型转换到关系模型

- 实体转换
  - 每个实体转换为一个关系模式，实体的属性为关系模式的属性，实体的标识成为关系模式的主码
- 联系转换
  - **1:1**：将任一端的实体的标识和联系属性加入另一实体所对应的关系模式中，两模式的主码保持不变
  - **1:N**：将**1**端实体的标识和联系属性加入**N**端实体所对应的关系模式中，两模式的主码不变
  - **M:N**：新建一个关系模式，该模式的属性为两端实体的标识以及联系的属性，主码为两端关系模式的主码的组合

（2）扩展ER模型转换到关系模型

- 弱实体转换
  - 每个强实体转换为一个关系模式，强实体的属性成为关系模式的属性，实体标识成为主码
  - 每个弱实体转换为一个关系模式，并加入所依赖的强实体的标识，关系模式的主码为弱实体的标识加上强实体的标识
- 子类转换
  - 父类实体和子类实体都各自转换为关系模式，并在子类关系模式中加入父类的主码，子类关系模式的主码设为父类的主码

ER模型转换例子


姓名
出生日期
子女
抚养
天数
项目号
项目名
部门号
部门名
生产
项目
N
部门
1
属于
N
职工
M
负责
1
1
1
领导
ISA
职工号
姓名
1
领导者
任职时间
奖金
车型

转换实体

实体转换为关系模式：

1、部门（部门号，部门名）

2、职工（职工号，姓名）

3、项目（项目号，项目名）

4、领导者（奖金，车型）

5、子女（姓名，出生日期）

先考虑弱实体“子女”，加入“职工号”，并修改主码为“职工号＋姓名”

5、子女（姓名，出生日期，职工号）

在考虑子类“领导者”，加入父类标识“职工号”作主码

4、领导者（奖金，车型，职工号）

转换联系

实体转换得到关系模式：

1、部门（部门号，部门名）

2、职工（职工号，姓名）

3、项目（项目号，项目名）

4、领导者（奖金，车型，职工号）

5、子女（姓名，出生日期，职工号）

考虑每个联系：

1、部门:领导者（1:1）：领导者（奖金，车型，职工号，部门号，任职时间）

或者　部门（部门号，部门名，职工号，任职时间）

2、部门:职工（1:N）：职工（职工号，姓名，部门号）

3、职工:项目（1:1）：项目（项目号，项目名，职工号）

4、职工:项目（M:N）：增加模式：职工_项目（项目号，职工号，天数）

得到初步的关系数据库模式

1. 部门（部门号，部门名）
2. 职工（职工号，姓名，部门号）
3. 项目（项目号，项目名，职工号）
4. 领导者（奖金，车型，职工号，部门号，任职时间）
5. 子女（姓名，出生日期，职工号）
6. 职工_项目（项目号，职工号，天数）

3、关系数据库模式的规范化

- 确定范式级别
- 实施规范化处理

（1）确定范式级别

- 范式级别的确定
  - 根据数据依赖确定已有的范式级别
    - 根据需求写出数据库模式中存在的所有函数依赖
    - 消除冗余数据依赖，求出最小的依赖集
    - 确定范式级别

（1）确定范式级别

- 范式级别的确定
  - 根据实际应用的需要（处理需求）确定要达到的范式级别
    - 时间效率和模式设计问题之间的权衡
      - 范式越高，模式设计问题越少，但连接运算越多，查询效率越低
      - 如果应用对数据只是查询，没有更新操作，则非BCNF范式也不会带来实际影响
      - 如果应用对数据更新操作较频繁，则要考虑高一级范式以避免数据不一致
    - 实际应用中一般以3NF为最高范式

（2）规范化处理

- 确定了初始数据模式的范式，以及应用要达到的范式级别后
- 按照规范化处理过程，分解模式，达到目标范式
  - “模式设计”部分的内容

4、模式评价

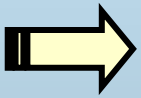
- 检查规范化后的数据库模式是否完全满足用户需求，并确定要修正的部分
  - 功能评价：检查数据库模式是否支持用户所有的功能要求
    - 必须包含用户要存取的所有属性
    - 如果某个功能涉及多个模式，要保证无损连接性
  - 性能评价：检查查询响应时间是否满足规定的需求。
    - 由于模式分解导致连接代价
    - 如果不满足，要重新考虑模式分解的适当性
    - 可采用模拟的方法评价性能

5、模式修正

- 根据模式评价的结果，对已规范化的数据库模式进行修改
  - 若功能不满足，则要增加关系模式或属性
  - 若性能不满足，则要考虑属性冗余或降低范式
    - 合并：若多个模式具有相同的主码，而应用主要是查询，则可合并，减少连接开销
    - 分解：对模式进行必要的分解，以提高效率
      - 水平分解
      - 垂直分解
      - 分库分表

（1）水平分解

学号	……	所在系
01		1
02		2
03		6
……		……



学号	……	所在系
01		1
12		1
13		1
……		……

学号	……	所在系
02		2
18		2
45		2
……		……

……

（1）水平分解

- 冷热数据划分
  - 80/20原则：一个关系经常被使用的数据只占20%
  - 将20%热数据单独划分为一个模式，使得大部分的查询都可以在较小规模的数据集上执行
  - 减少了应用处理的数据量，提高效率

（2）垂直分解

- 把关系模式按属性集垂直分解为多个模式
- 在实际中，应用可能经常存取的只是关系的某几个列，可考虑将这些经常访问的列单独拿出组成一个关系模式
- 若一个关系中，某几个属性的值重复较多，并且值较大，可考虑将这些属性单独组成关系模式，以降低存储空间

（2）垂直分解

50万种商品

商品号	……	商品说明书（4KB）

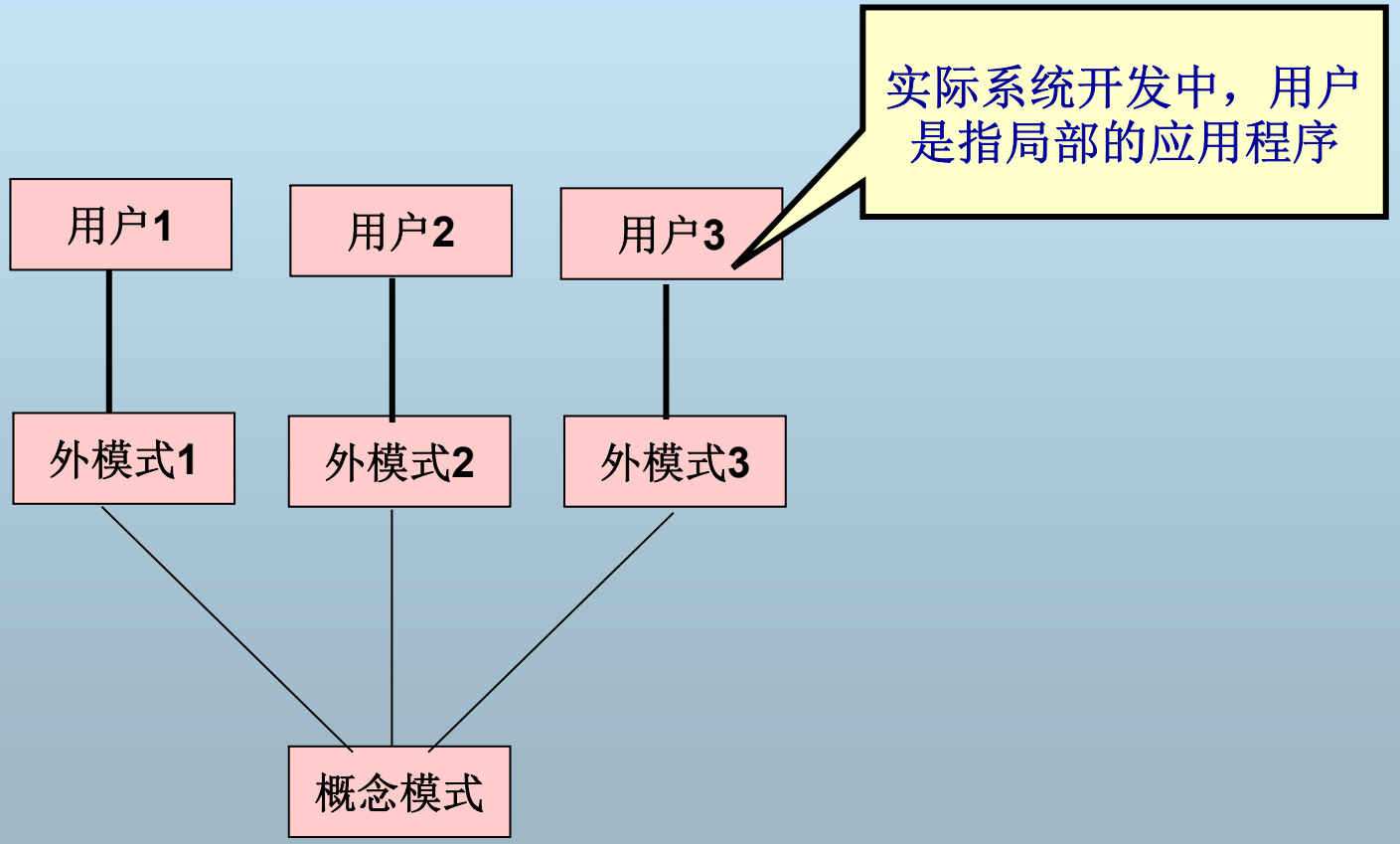
商品表约减少了
2GB数据量

商品号	……	说明编号（2B）

说明编号	商品说明书

6、设计用户子模式（视图）

- 根据局部应用的需求，设计用户子模式



（1）设计用户子模式（视图）的考虑

- 使用更符合用户习惯的别名
  - ER图集成时要消除命名冲突以保证关系和属性名的唯一，在子模式设计时可以重新定义这些名称，以符合用户习惯
- 给不同级别的用户定义不同的子模式，以保证系统安全性
  - 产品(产品号，产品名，规格，单价，产品成本，产品合格率)
    - 为一般顾客建立子模式：产品1(产品号，产品名，规格，单价）
    - 为销售部门建立：产品2(产品号，名称，规格，单价，成本，合格率)
- 简化用户程序对系统的使用
  - 可将某些复杂查询设计为子模式以方便使用

六、数据库物理设计

- 设计数据库的物理结构
  - 为关系模式选择存取方法
  - 设计数据库的存储结构
- 物理设计的考虑
  - 查询时间效率
  - 存储空间
  - 维护代价
- 物理设计依赖于给定的计算机系统

1、选择存取方法

- 存取方法：数据的存取路径
  - 例如图书查询
- 存取方法的选择目的是加快数据存取的速度
  - 索引存取方法
  - 聚簇存取方法
  - 散列存取方法

可以使用什么样的存取方法依赖于具体的**DBMS**

DBMS中最流行的索引：B＋树

- 一种树型的多级索引结构
- 每个节点都是一个磁盘块（页）
  - 例如MS SQL Server的页为8KB, MySQL默认页为16KB
- 树的层数与数据大小相关，通常为3层
- 所有结点格式相同：n个值，n＋1个指针
- 所有叶结点位于同一层
- 适用于主索引，也可用于辅助索引

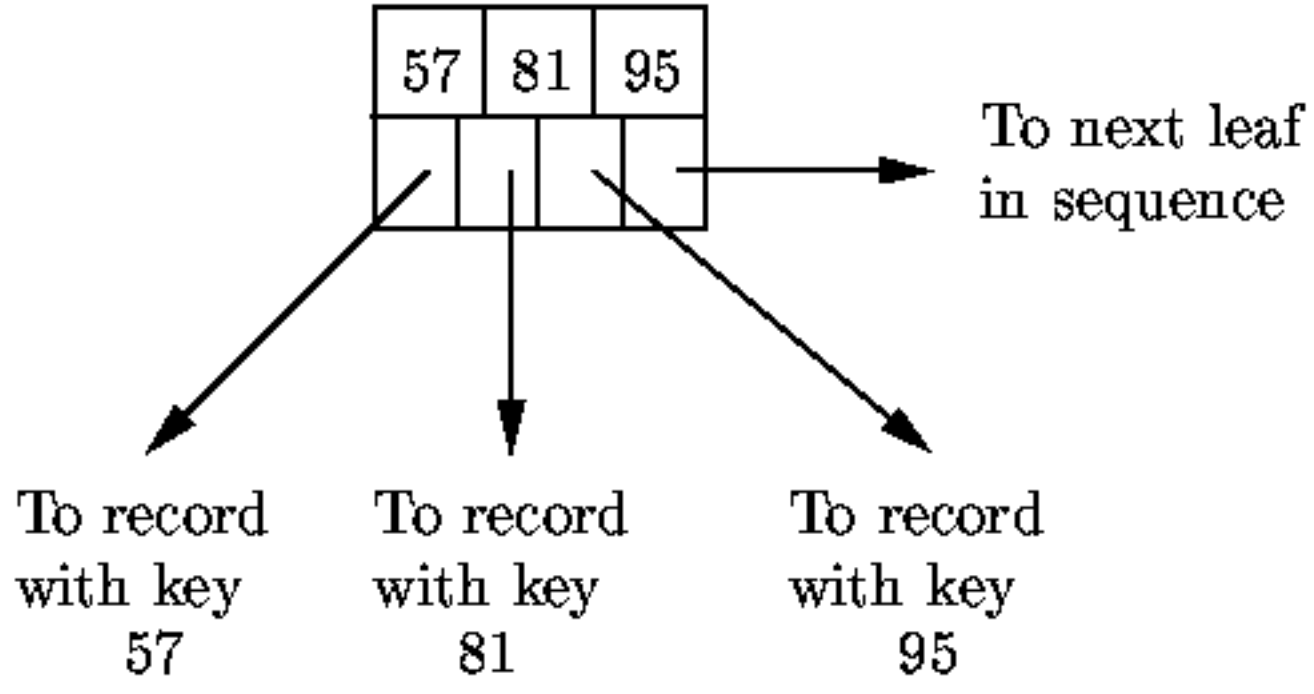
主索引与辅助索引

```
MovieStar(name char(10) PRIMARY KEY, address char(20))
```

- **Name上创建了主索引，记录按name有序**
  - **主索引即MovieStar的聚簇索引**
- **Address上创建辅助索引**

```
Create Index adIndex On MovieStar(address)
```

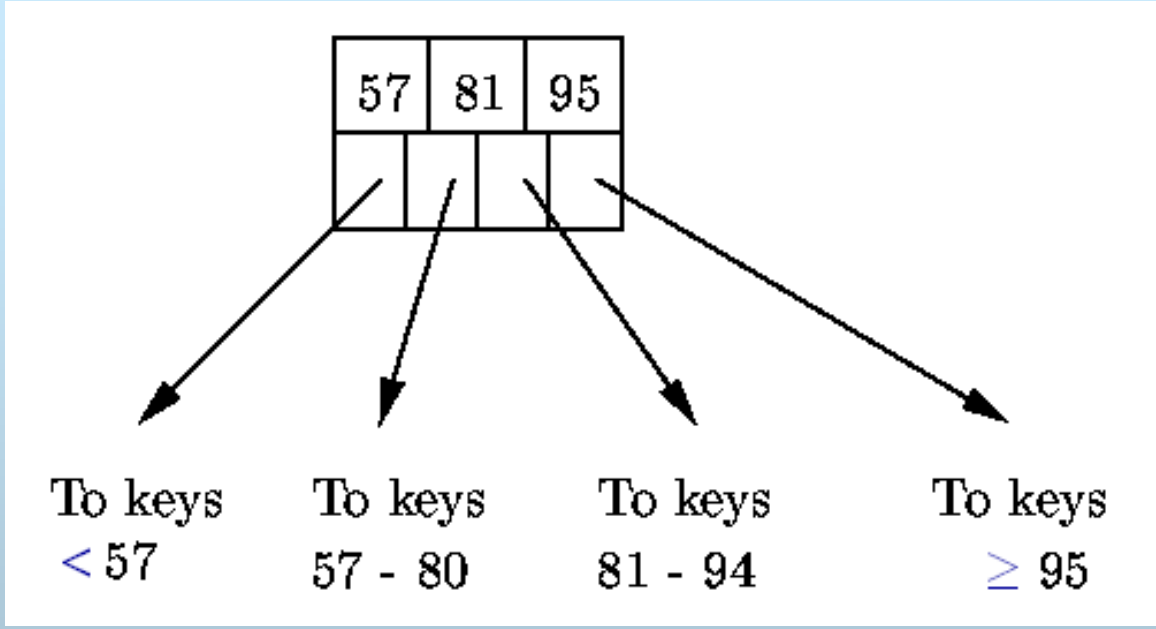
B+树的叶结点



- 1个指向相邻叶结点的指针
- n对键一指针对

- 至少$\lfloor (n+1)/2 \rfloor$个指针指向键值

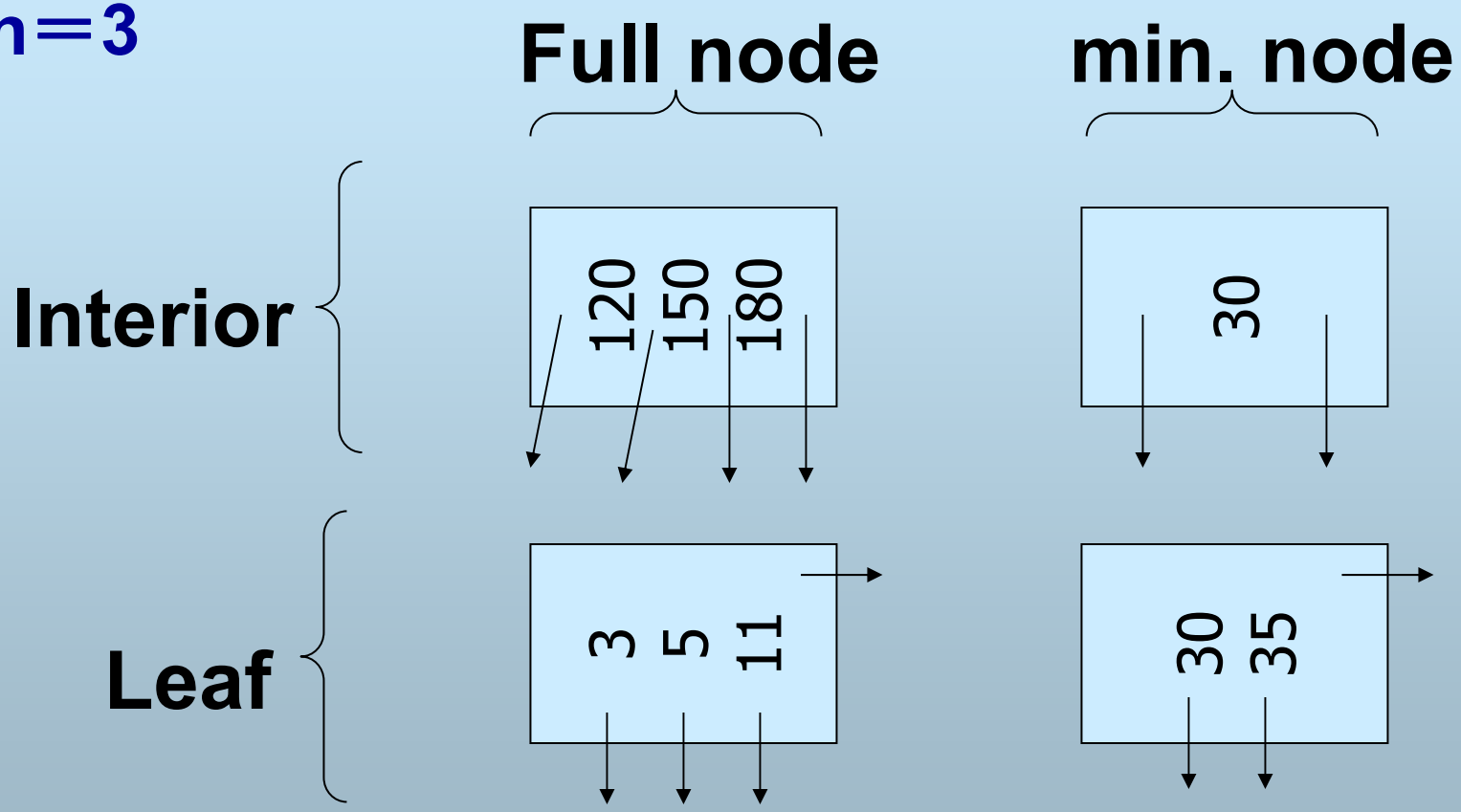
B+树的中间结点



- n个键值划分n+1个子树
- 第 i 个键值是第 i+1 个子树中的最小键值
- 至少⌈(n+1)/2⌉个指针指向子树
- 根结点至少 2 个指针

B＋树结点例子

n＝3



B＋树查找

- 从根结点开始
- 沿指针向下，直到到达叶结点
- 在叶结点中顺序查找

B＋树插入

- 查找插入叶结点
- 若叶结点中有空闲位置（键），则插入
- 若没有空间，则分裂叶结点
  - 叶结点的分裂可视作是父结点中插入一个子结点
  - 递归向上分裂
  - 分裂过程中需要对父结点中的键加以调整
  - 例外：若根结点分裂，则需要创建一个新的根结点

B＋树删除

- 查找要删除的键值，并删除之
- 若结点的键值填充低于规定值，则调整
  - 若相邻的叶结点中键填充高于规定值，则将其中一个键值移到该结点中
  - 否则，合并该结点与相邻结点
    - 合并可视作在父结点中删除一个子结点
  - 递归向上删除
- 若删除的是叶结点中的最小键值，则需对父结点的键值加以调整

B＋树的效率

- 访问索引的I/O代价＝树高（B＋树不常驻内存）或者0（常驻内存）
- 树高通常不超过3层，因此索引I/O代价不超过3（总代价不超过4）
  - 通常情况下，根节点常驻内存，因此索引I/O代价不超过2（总代价不超过3）

B＋树的效率

- 设块大小8KB，键2B（smallint），指针2B，则一个块可放约2048个索引项

层数	索引大小（块数/大小）	索引记录空间
1	1/8KB	2047
2	(1＋2048)/约16M	约419万(2^{22})
3	(1+2^{11}+2^{22})/约32G	约85亿(2^{33})

2、设计数据库的存储结构

- 确定数据的存放位置
  - 针对应用环境和DBMS特性，合理安排数据存储位置
    - 表和索引可考虑放在不同的磁盘上，使查询时可以并行读取
    - 日志文件和备份文件由于数据量大，而且只有恢复时使用，可放到磁带上
- 确定系统配置
  - 系统初始参数不一定适合应用
    - 并发用户数、同时打开的数据库对象数、缓冲区分配参数、物理块的大小等

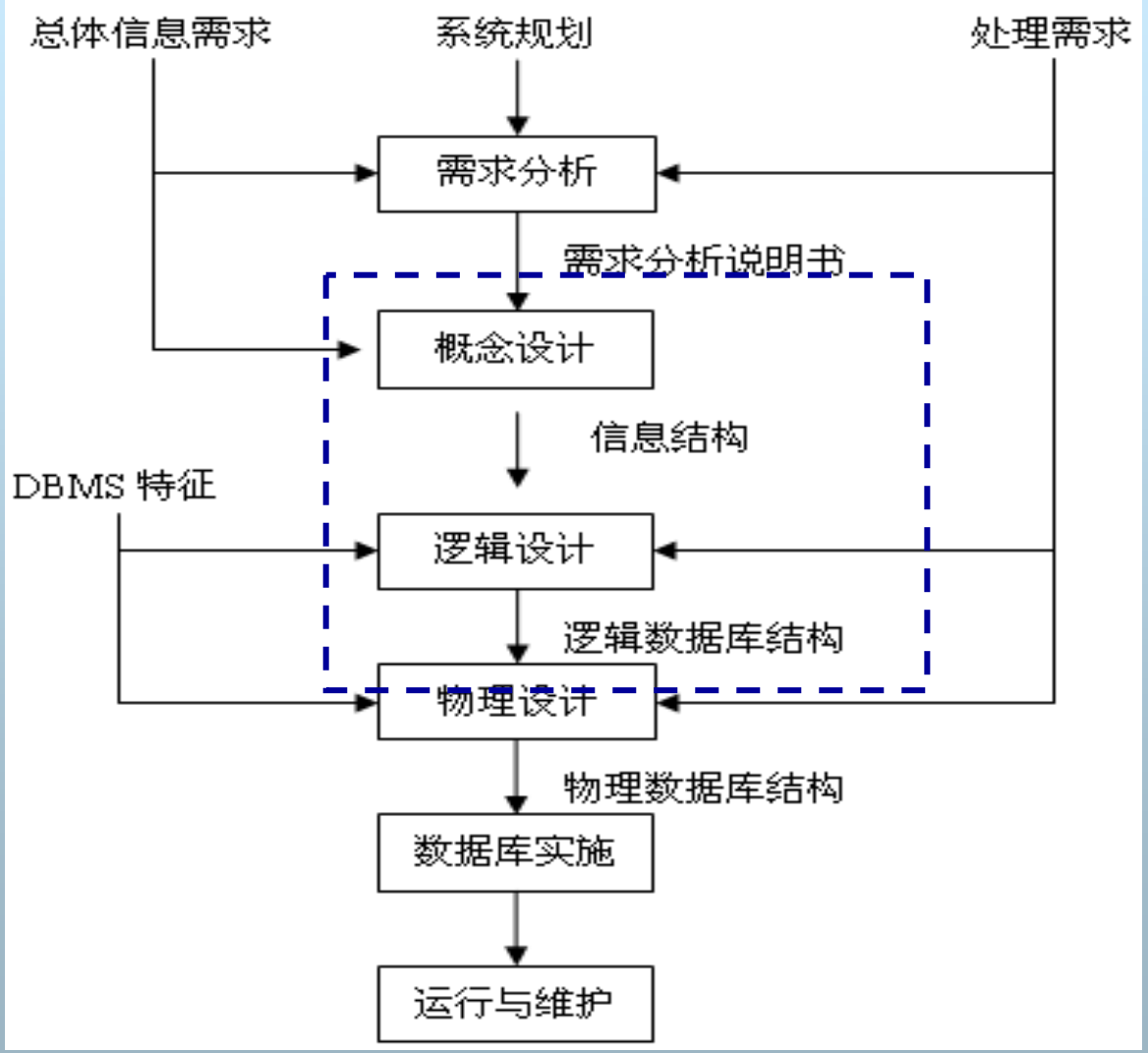
七、数据库实施

- 建立实际的数据库结构
  - CREATE TABLE
  - CREATE INDEX
  - ……
- 初始数据装入
- 安全性设计和故障恢复设计
- 应用程序的编码和调试

八、运行和维护

- 试运行
  - 根据初始数据对数据库系统进行联调
  - 执行测试：功能、性能
- 维护
  - 数据备份和恢复
  - 数据库安全性控制和完整性控制
  - 数据库性能的分析和改造
  - 数据库的重组织

本章小结


总体信息需求
系统规划
处理需求
需求分析
需求分析说明书
概念设计
信息结构
DBMS 特征
逻辑设计
逻辑数据库结构
物理设计
物理数据库结构
数据库实施
运行与维护