高级数据库系统及其应用

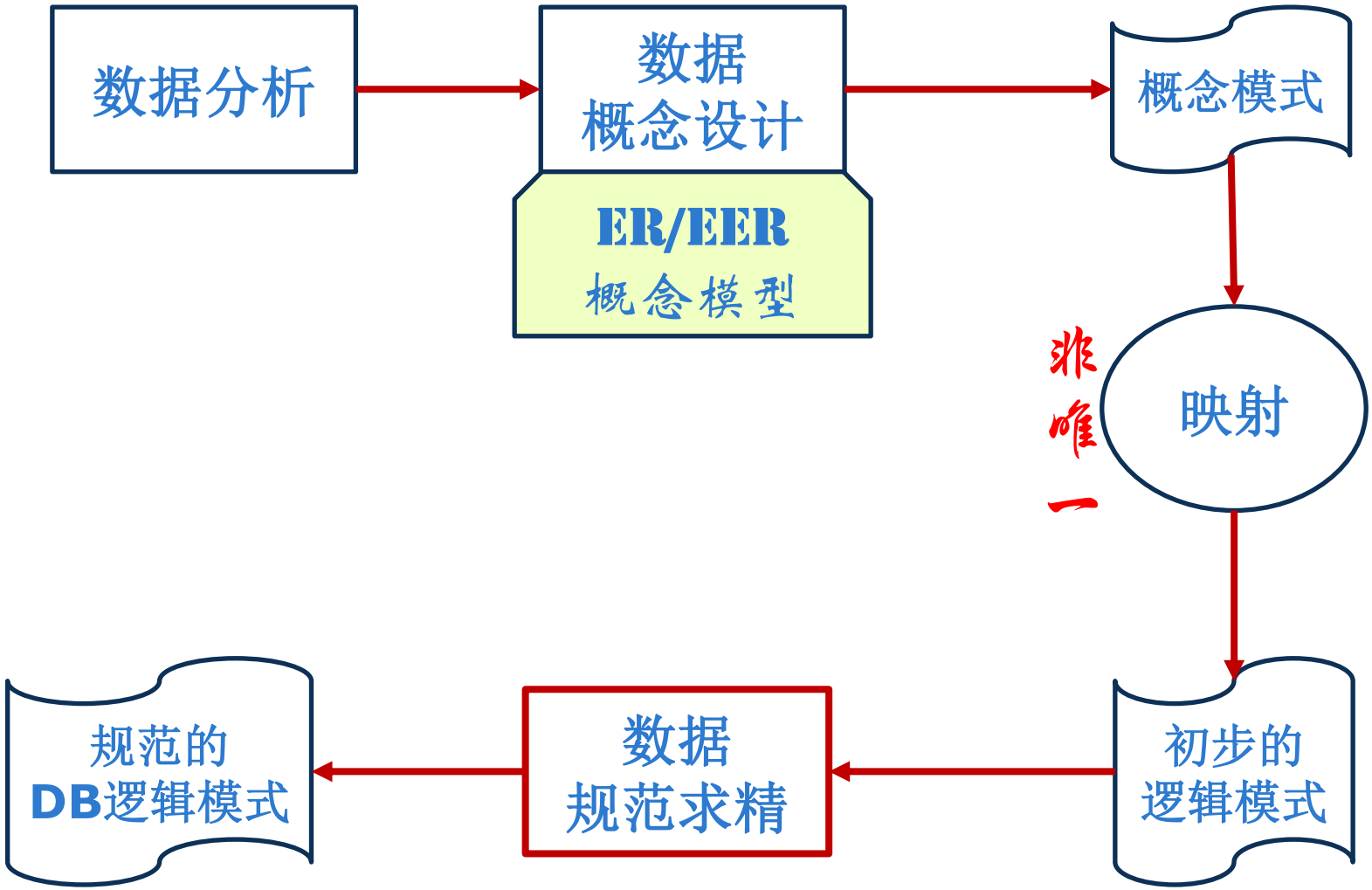
第1部分　数据库系统基础

第3章 数据库表结构设计

xshxie@ustc.edu.cn

LOGO

DB-表结构设计过程


数据分析
数据
概念设计
ER/EER
概念模型
概念模式
非唯一
映射
初步的
逻辑模式
数据
规范求精
规范的
DB逻辑模式

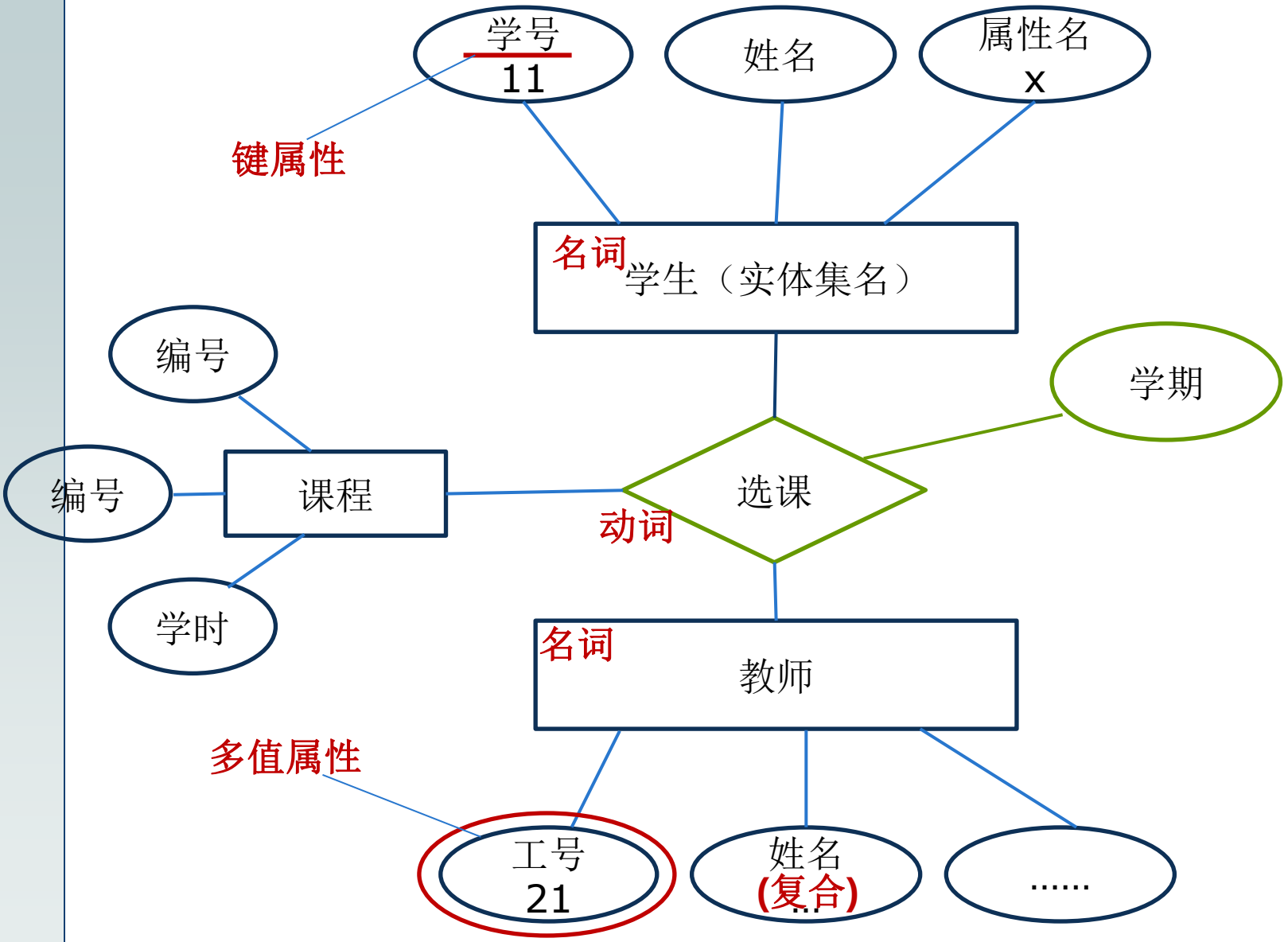
第3章 数据库表结构设计

ER模型简介

1. 构成ER模型的基本概念

- 实体与属性
- 实体类型、实体集与键
  - **实体类型**：指具有相同属性的实体模式结构，由名和属性来描述。
  - **实体集**：具有相同实体类型的所有实体集合。
    - 实体类型描述了相同结构实体集的模式或内涵；
    - 实体集则描述了实体类型的外延。
    - ER图中不区分实体类型和实体集（被视为同义词）。
- 关系、关系类型和关系集
  - 关系集通常是动词，且可以有自己的属性
- ER模型的其它概念

基本ER图元


学号
11
姓名
属性名
x
键属性
名词
学生（实体集名）
编号
学期
编号
课程
选课
动词
学时
名词
教师
多值属性
工号
21
姓名
(复合)
……

ER模型的其它概念

- **关系约束**
  - 指与关系集相关的约束，通过约束表达可限制参与关系各实体的可能组合。
  - 主要类型：基数词约束、键约束和参与约束。
- **弱实体集**
  - 指只能附属其它实体集而存在的实体集。

ER图设计举例（1）

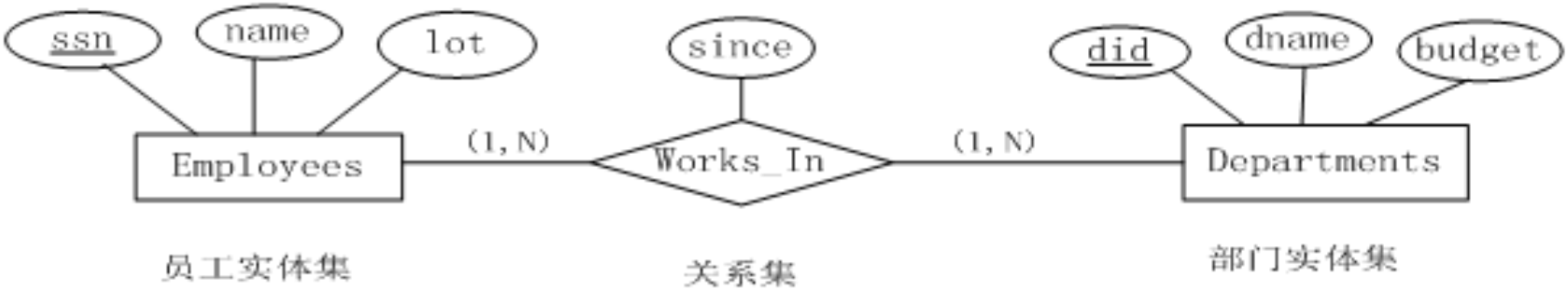
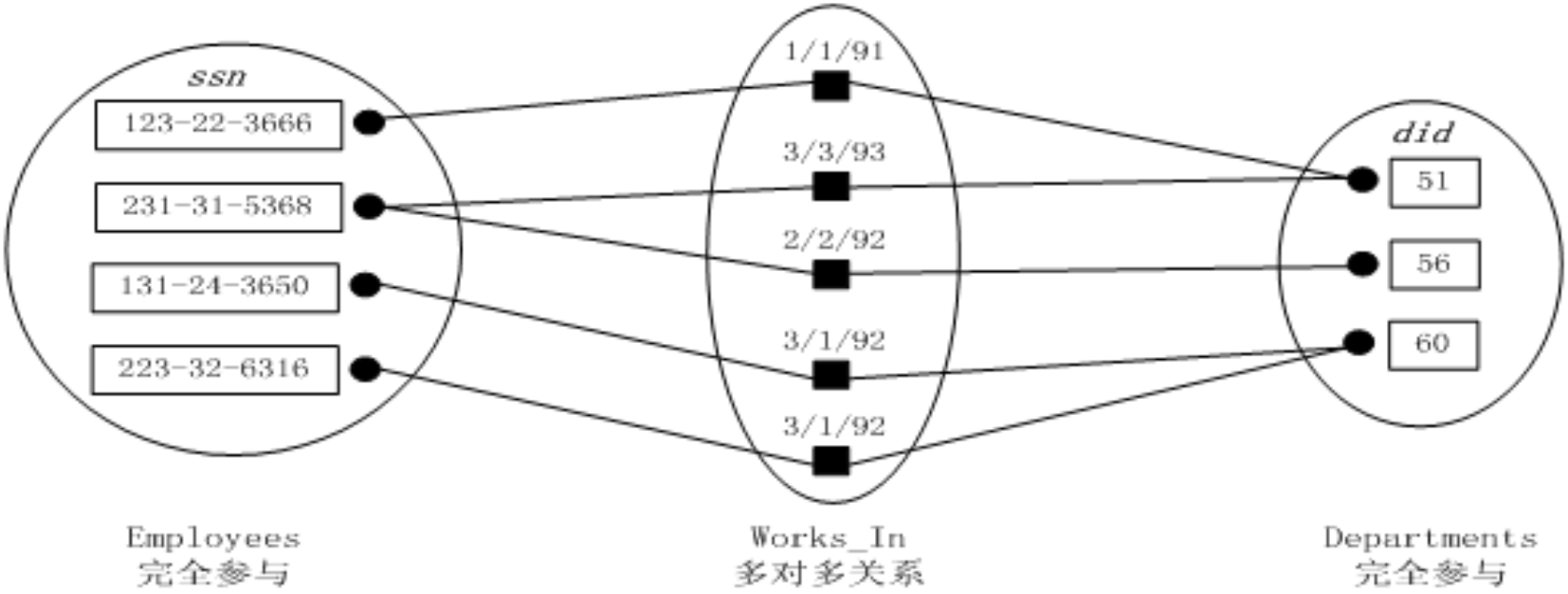


图3.2 （a）ER图中的实体集与关系集



（b）关系集Works_In的一个实例

ER图设计举例 （2）

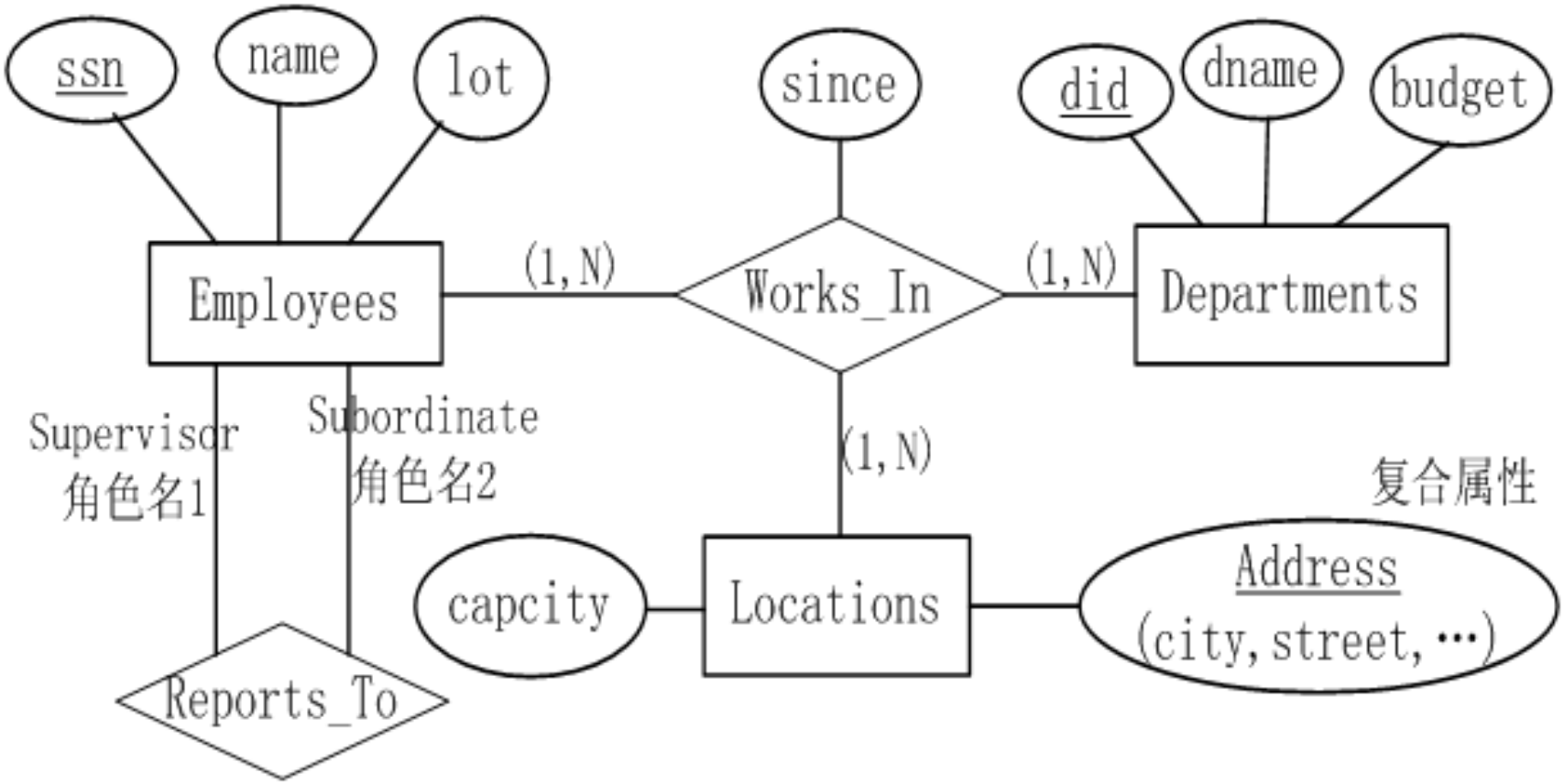


图3.3

EER核心概念（1）

- 类
  - 指实体的集合或实体集，也可包括实体类(型)、子类、超类和类别等。
  - EER中，任何类都允许参与一个关系。
- 子类、超类
  - 子类S是一个类，子类中的实体必然也是超类C的一个实体，即有关系：S⊆C 成立
  - 超类/子类关系也称为ISA关系，记做C/S。
  - 子类实体除了可以从超类实体中继承所有的属性外，还可以有自己专有的属性和关系。

EER核心概念（2）

- 特化
  - 特化$Z=\{S_1,S_2,\ldots,S_n\}$是具有相同超类G的一个子类集合，每个G/S_i是一个超类/子类关系。
    - 用“特化”指代由特化过程所获得的--特化子集。
    - G被称为泛化实体类型。
  - 特化的种类（约束）

特化是概念上的求精，而泛化则是概念上的综合。
显然，由泛化获得超类方法，易得到完全特化的子集。

- 泛化
  - 是特化的逆过程，允许我们忽略多个实体集之间的性质差异，找出它们的共同点——抽象出超类。

特化及其约束的EER表示-示例

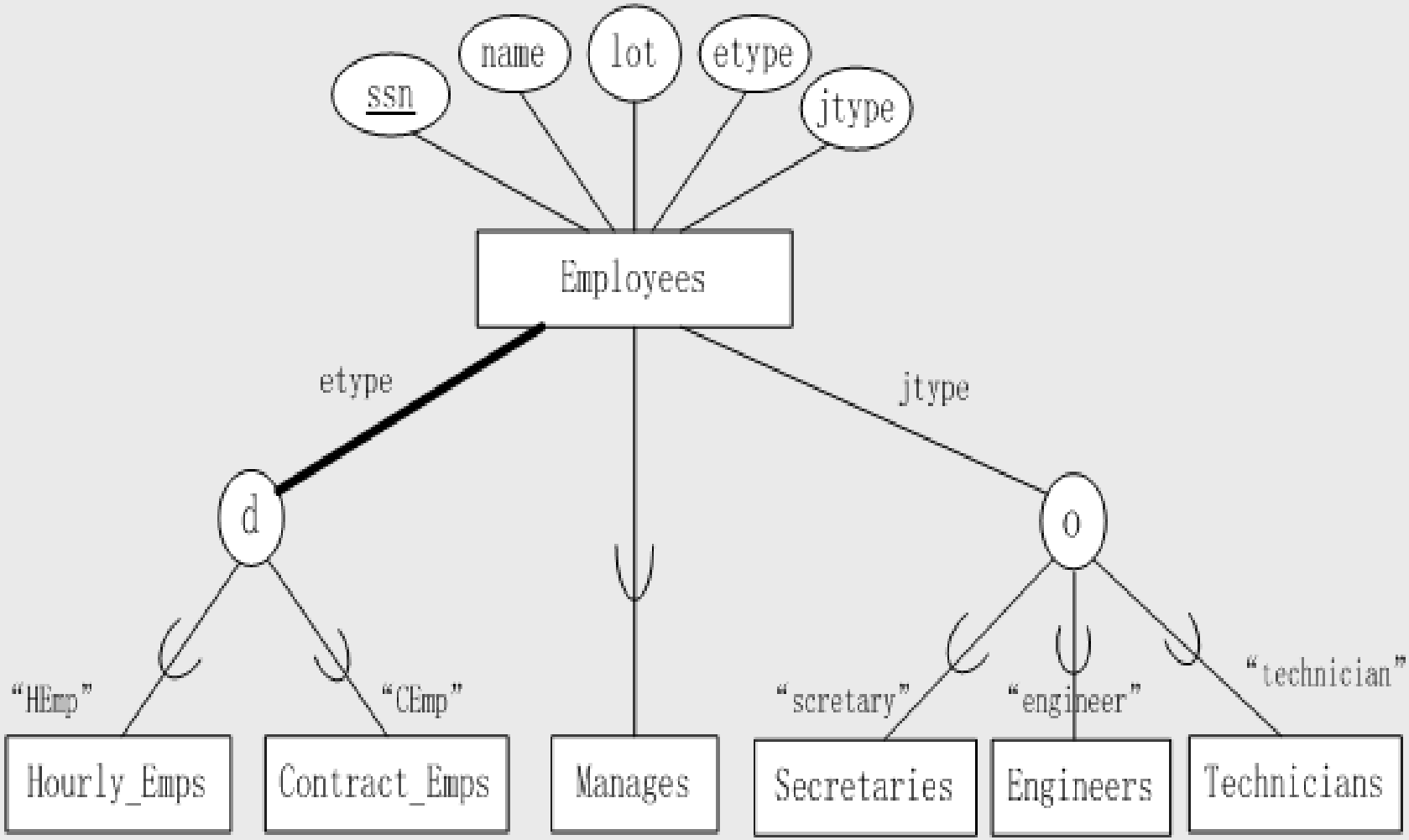
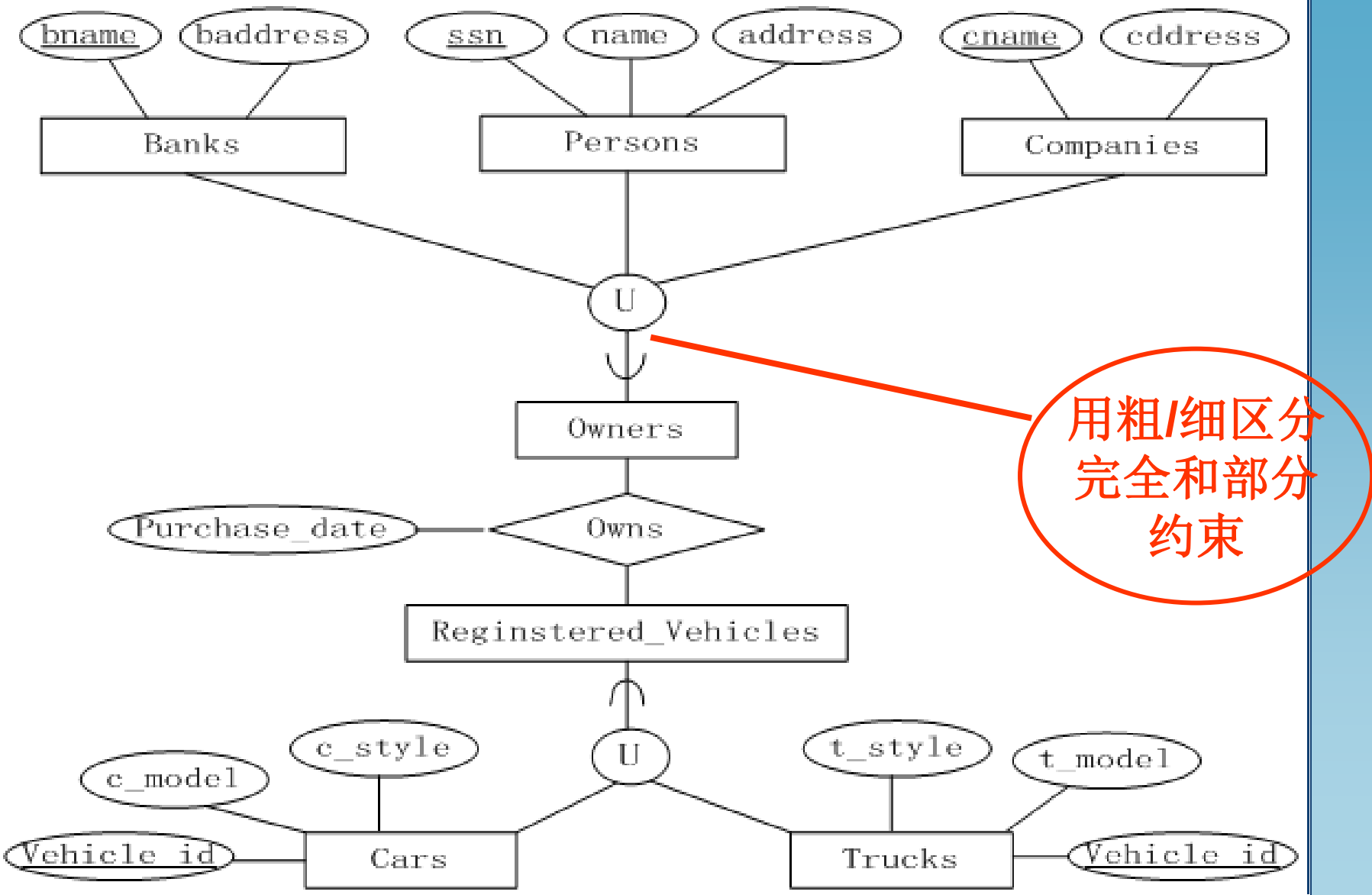


图3.7 一个特化及其约束的EER表法示例

EER核心概念（3）

- **类别(category)**
  - 类别有时也被称为union子类。
  - 类别T是一个类，它是n个判定超类$D_1,D_2,\ldots,D_n$（$n>1$）并集的一个子集。
    - 其形式表示为：$T\subseteq(D_1 \cup D_2 \cup \ldots \cup D_n)$
- **union子类的约束**
  - 完全约束：子类包含了其所有超类并集中的所有成员；
  - 部分约束：子类只包含并集的一个子集。

UNION子类及其约束的EER表示（图3.8）



Company DB模式的EER表示

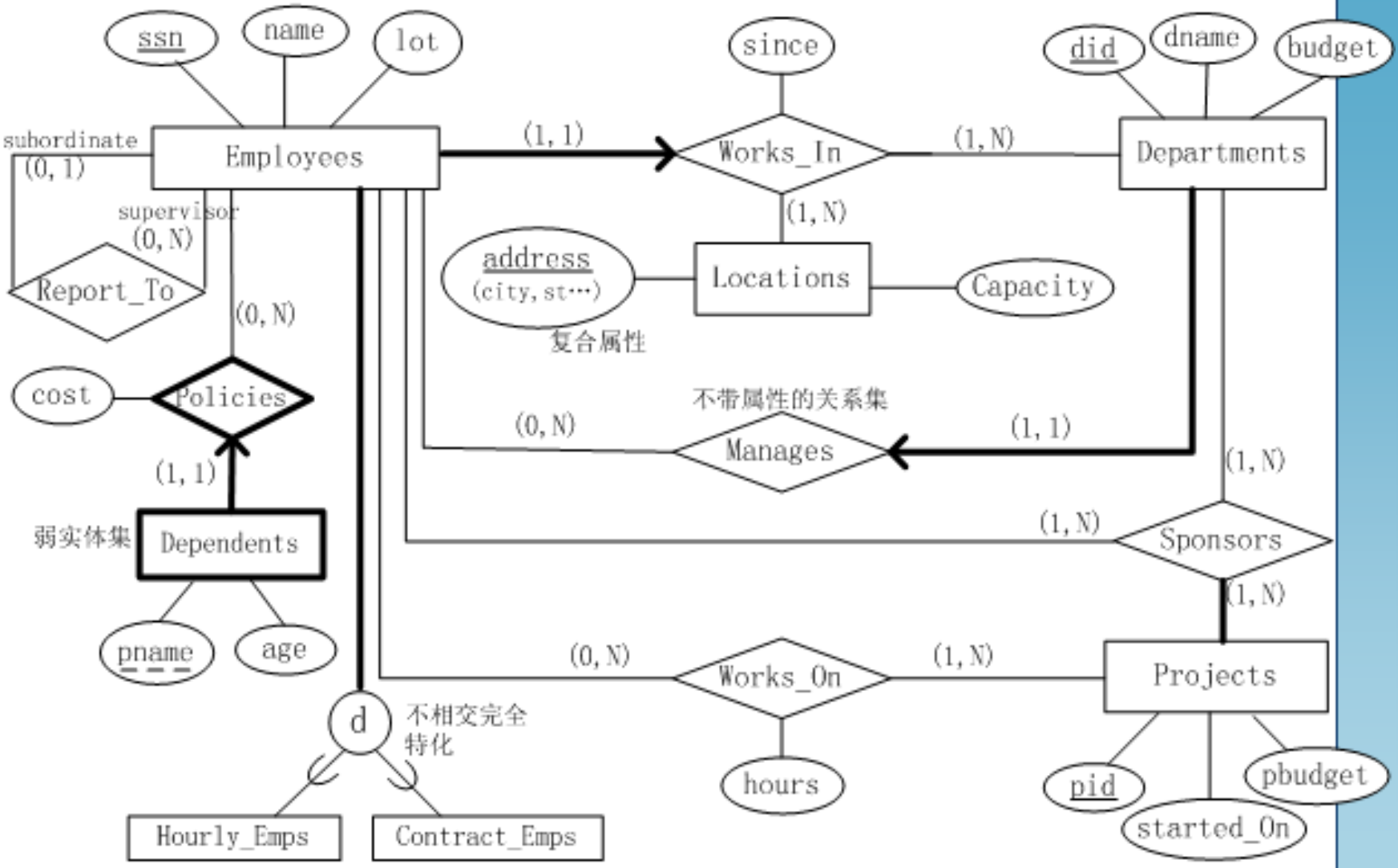


图3.9 (a) 基于ER模型表达的Company-DB概念模式

Company DB模式的UML表示

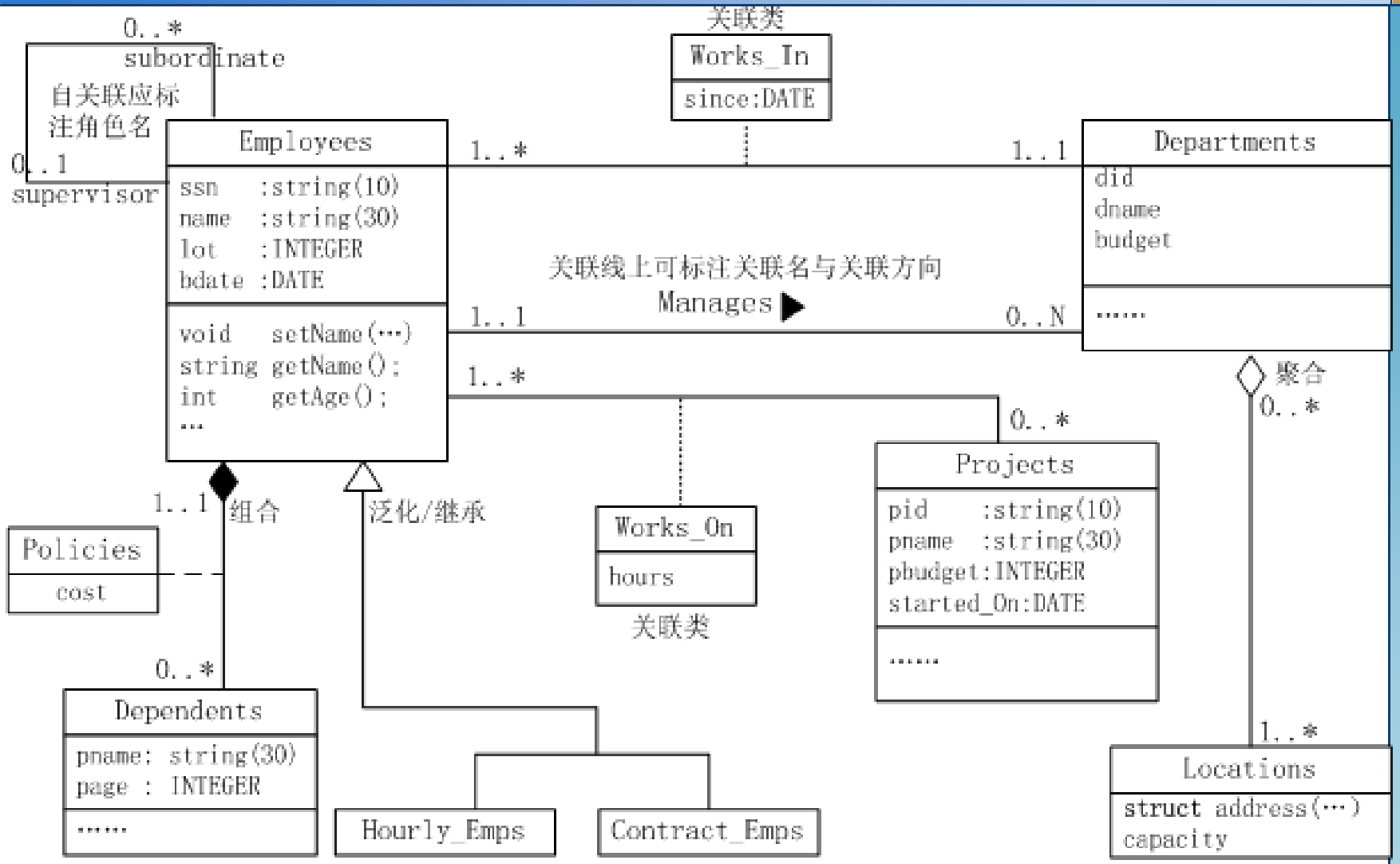


图3.9 (b) 基于UML类图表达的Company-DB概念模式

3.3 逻辑数据库设计:映射ER/EER模式到关系模式

映射常规实体集到关系表

- **一个常规实体集可直接地映射到一个关系表：**
  - 将实体集的每个属性，作为关系表的一个属性。
  - 用SQL-92 DDL建表语句基本上可以完全捕获这些信息，包括域约束和主键约束。

举例：图3.9（a）中的Employees实体集映射

```
CREATE TABLE Employees (
                    ssn    CHAR(11),
                    name VARCHAR(30),
                    lot       INTEGER,
                    PRIMARY KEY (ssn)  )
```

映射关系集到关系表

（一）映射含键约束的关系集

方法1：独立关系表法

- 映射关系集***R***到独立的关系表***R’***。

示例：图3.9(a)的Manages关系集 映射：

```
CREATE TABLE Manages  (
        ssn       CHAR(11),
        did       INTEGER,
        since     DATE,
        PRIMARY KEY (did),
        FOREIGN KEY (ssn) REFERENCES Employees,
        FOREIGN KEY (did) REFERENCES Departments )
```

映射关系集到关系表

（一）映射含键约束的关系集

方法1：独立关系表法

方法2：外键方法

- 将关系集的相关信息合并到具有键约束的参与实体集中（一对多关系的‘一’端）。

示例： 图3.9(a)的Manages关系集，也可以按如下映射

```
CREATE TABLE Dept_Mgr ( did      INTEGER,
                        dname    VARCHAR(20),
                        budget   REAL,
                        ssn      CHAR(11),
                        since    DATE,
                        PRIMARY KEY (did),
                        FOREIGN KEY (ssn) REFERENCES Employees)
```

映射关系集到关系表

（一）映射含键约束的关系集

方法1：独立关系表法

方法2：外键方法

方法3：合并关系法

- 若关系集的所有参与实体集都有键约束且都是完全参与。这时，也可合并所有参与实体集到一个关系。

（二）在映射关系集时考虑参与约束

- 图3.9（a）中的Manages，除了键约束（每部门至多有一经理）外，还含有一完全参与约束（每部门至少需要有一经理）。考虑到这一点，Dept_Mgr:ssn应设置NOT NULL。

（三）无键约束和参与约束的关系集映射

- 对这类关系集，一般只能用独立关系表法(方法1)进行映射。

映射EER扩展结构——多值/复合结构属性

- **关系模式不支持多值属性，必须为关系模式中的每个多值属性，分别创建一个独立的关系。**
  - 令关系模式为*R*，*MA*是*R*的一个多值属性，为*MA*创建的关系表为*M*。
    - *M*的属性应包含*R*的主键属性*k* ，以便关联到R。
    - 原关系模式*R*中可去掉多值属性MA.
- **令关系模式为*R*，*CA*是*R*的一个复合属性。对于*CA*，有两种建模方法：**
  - 方法1：将复合属性的每个结构成份，分别作为一个属性，加到所属的关系表中。
  - 方法2：为复合属性*CA*单独建立一个关系表。

映射EER扩展结构——类层次结构

- 映射处理EER图中的ISA层次结构。
  - 假设超类C被特化为m个子类{S_1 , … , S_m}
  - Attr(C) = {k, a_1 , … , a_n}，PK(C) = k。
- 方法1：映射超类和每个子类到一个不同的表。

示例：对图3.9(a)中Employees特化

- 先映射常规实体集Employees。
- 映射子类Hourly_Emps关系，其属性包括:
  - *(ssn, hourly_wages, hours_worked)*
  - *ssn*在该子类表中既作为主键，也作为关联超类的外键。
  - 若超类元组被删除，必须CASCADE删除相应的子类元组。
- 类似方式,映射子类Contract_Emps关系表。

映射EER扩展结构——类层次结构

- 方法1：映射超类和每个子类到一个不同的表。
- 方法2：仅创建子类关系表。
  - 为每个子类$S_i(1 \le i \le m)$创建一个关系L_i，且有属性$Attr(L_i)= \{k, a_1, \ldots, a_n\} \cup \{S_i$的其它专有属性$\}$，$PK(L_i)=k$。
  - 该方法只适用于超类完全参与的特化类型。
- 方法3：仅创建含1个类标志属性的单个关系。
- 方法4：仅创建含m个类标志属性的单个关系。
  - 该方法能适应子类有重叠特化的情况，但会产生大量的null值。

映射EER：union子类（1）

1）对超类实体集有各自不同键的情况

示例：映射图3.8中Owners及其相关UNION子类：

Persons (*ssn*, *name*, *address*, *owner_id*);
Banks (*bname*, *baddress*, *owner_id*);
Companies (*cname*, *caddress*, *owner_id*);
Owners (*owner_id*)。

2）对超类实体集有有相同键的情况

- 这时，无需使用代理键。

示例：映射图3.8中Registered_Vehicles及相关子类

Reginstered_Vehicles(vehicle_id, licenseNumber);
Cars (vehicle_id, cstyle, cmodel, …);
Trucks (vehicle_id, tstyle, tmodel, …);
Owns (owner_id, vehicle_id, puchase_date)。

ER模型至关系模型映射小结

- 步骤1：映射常规实体集。
- 步骤2：映射弱实体集。
- 步骤3：映射ER模式中的关系集。
- 步骤4：映射ER模式中的聚集关系集。
- 步骤5：映射与EER模型相关的扩展结构。

3.4 关系模式求精与规范化

3.4.1 模式求精问题（综述）

- 模式求精的基本任务是基于分解技术，来处理初始关系模式中存在的问题。信息的冗余存储是引发这些问题的根源。虽然分解能删除冗余，但它也可能导致一些额外的问题，如信息损失或导致某些强制性约束丢失，必须慎重使用。

（一）冗余可能引发问题

- 浪费空间
- 更新异常
  - 同样的信息被存储多份，如某份数据被更新，而其它份信息未做相应更新，就会造成DB数据的不一致。
- 插入异常
  - 如果不附带冗余存储一些相关的信息，新的信息可能无法存储到DB中。
- 删除异常
  - 删除某信息时,可能会附带删掉一些不希望删除的信息

冗余可能引发问题举例

- 考虑**Hourly_Emps**

  (ssn, name, lot, rating, hourly_Wages, Hours_worked)

  - 缩写为Hourly_Emps (*SNLRWH*)
  - 假定小时工资主要取决于员工等级，即给定*R*值，就可唯一确定*W*值。这是一个典型的函数依赖约束关系，它会导致存储冗余，其副作用有多个方面：
    - 同等级员工对应的元组中，*R*/*W*信息完全相同。同样的信息被存储多次，浪费存储空间。
    - 如果删除了给定*R*值的所有元组，将丢失这组*R*/*W*所隐含的IC约束信息，这是一种删除异常。
    - 无法单独记录员工等级与小时工资的*R*/*W*关系。这是一种插入异常。

（二）利用分解技术消除冗余

- 函数依赖约束(**FDs**)或其它相近的**ICs**可被用来识别冗余点，并给出处理冗余的指导性建议。
- 分解技术的核心思想
  - 通过将原关系替换（分解）为一组更小关系，来解决冗余问题。
  - 例如，通过将**Hourly_Emps**分解为如下的两个小关系，就可以很好消除原有冗余引起的相关问题。
    - Hourly_Emps2(*ssn, name, lot, rating, hours_worked*)
    - Wages(*rating , hourly_wage*)

（三）分解可能引发的相关问题

- 分解能很好解决冗余问题，但必须慎重使用，否则可能会带来其它问题。在使用分解时，须反复提问以下两个重要问题：
  - 我们的确需要分解一个关系吗？
    - 对该问题，已有若干规范来帮助回答这个问题。
  - 一个给定的分解会引起那些其它问题？
    - 对该问题，可借助分解的两个重要特性来帮助回答
      - 用无损连接(*lossless-join*)；
      - 依赖保持(*dependency- preservasion*)

3.4.2 函数依赖(*functional dependency*,FD)

- 函数依赖，是DB中两组属性间存在的一种约束，是一类更广义的键概念约束。其形式定义如下：
  - 令*R*:关系模式，*r*是*R*的一个实例。

    *X*和*Y*是*R*的两个非空属性子集。
  - 若 $\{\forall<t_1,t_2>|\ t_1 \epsilon r,\ t_1 \epsilon r\}$,

    只要 $t_1.X=t_2.X$,

    必有 $t_1.Y=t_2.Y$。 就称*Y*函数依赖于*X*，记为：*X*→*Y*。
- 两类特殊的函数依赖
  - 完全函数依赖 若 X---<X1,X2> 且 <X1,X2>→Y

    但X1→Y 或 X2→Y 均不成立
  - 部分函数依赖 X1→Y 或 X2→Y 至少有一成立
- 通常，模式设计者会显式指定一组函数依赖。

函数依赖推理（1）

- 在满足***F*:{FDs}**的所有合法关系实例中，通常还会隐含一些其它可从***F***推理获得的函数依赖。
  - 例如，对**Workers(*ssn*, *name*, *lot*, *did*, *since*)**
    - 显式FDs
      - **FD1: *ssn*→*did*，FD2: *did*→*lot* *保持***
- 定义（隐含函数依赖 ***f***）
  - 给定**FDs**集***F***，如果**FD: *f*** 也能在满足***F***的每个关系实例中保持，则称**FD: *f*** 是隐含在***F***中的函数依赖。
- 定义（函数依赖集闭包F^+）
  - 将包括给定的**FDs**集***F***，加上***F***所隐含的所有***f***，合称为***F***闭包，简记为F^+。

函数依赖推理 （2）

- 由给定FDs集F，推导或计算出F^+的规则
  - 自反规则IR1：如 $X \supseteq Y$，则 $X \rightarrow Y$。
  - 增广规则IR2：如 $X \rightarrow Y$，则 $XZ \rightarrow YZ$，Z是任意属性组。
  - 传递规则IR3：如果 $X \rightarrow Y, Y \rightarrow Z$，则 $X \rightarrow Z$。

    两增补规则：
  - 合并或加法规则IR4：如果 $X \rightarrow Y, X \rightarrow Z$，则 $X \rightarrow YZ$。
  - 分解或投影规则IR5：如果 $X \rightarrow YZ$，则 $X \rightarrow Y, X \rightarrow Z$。
- 定义（平凡函数依赖）
  - 如果$X \rightarrow Y$且$X \supseteq Y$，则称$X \rightarrow Y$是平凡的(trivial)。
    - 显然，利用各类推理规则，我们不难由已知的FDs推出所有的平凡依赖关系。

函数依赖推理（3）

- 定义（函数依赖集覆盖）
  - 对于函数依赖集*F*，如果另一个函数依赖集*E*中的每个函数依赖同时也在*F*+中，则称*F*覆盖了*E*。
- 定义（函数依赖集等价）
  - 对于两个函数依赖集*E*和*F*，如果*E*+=*F*+，则称*E*和*F*是等价的。
- 定义（函数依赖集最小覆盖）一个FDs集*F*的最小覆盖是满足以下三个条件的一组FDs集*G*:
  - *G*中的每个依赖关系都是规范的*X*→*A*形式，这里，*A*是一个单属性；
  - 闭包*F*+等价于闭包*G*+。
  - 如果通过删除*G*中的任何一个依赖关系，得到另一个依赖集*H*，则必有*H*+≠*G*+。

3.4.3 基本规范范式（1）

第一范式

- 对于一个关系*R*，如果它的每个字段只包含不可分割的原子值（即没有复合值或值集字段），则*R*满足第一范式，记为*R*∈1NF。
  - 1NF独立于键和函数依赖；关系模型能自然满足1NF约束。

第二范式

- 对于一个关系*R*，如果它的每个非键属性*A*都完全依赖于*R*的某个键，则*R*满足第二范式，记为*R*∈2NF。

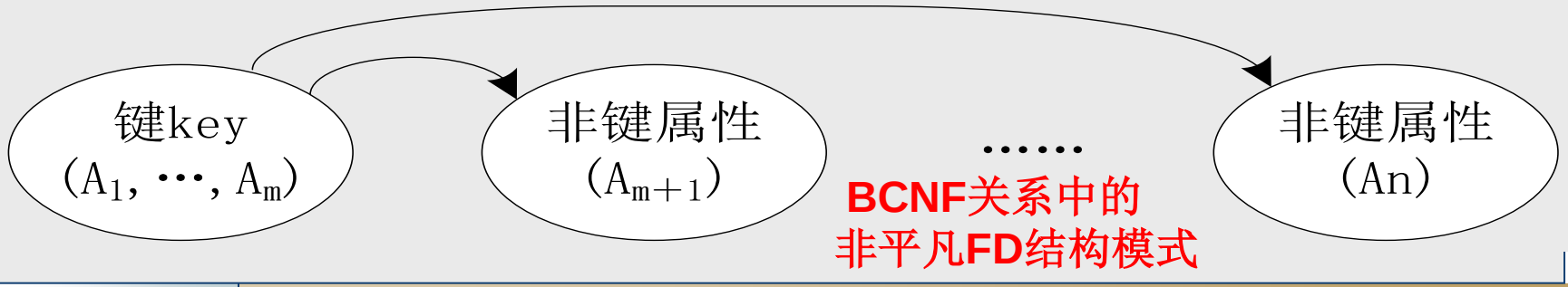
3.4.3 基本规范范式（2）

Boyce-Codd 范式

- 令*R*是一关系模式，*X*和A分别是*R*的属性子集。如果对*R*中保持的每个FD:*X*→*A*，能至少满足以下两条件之一，就称*R*满足Boyce-Codd范式，简记为*R*∈BCNF。
  1) A∈X，即*X*→*A*是一个平凡的FD；
  2) *X*是一个超键。
- 可证明：判断*R* ∈ BCNF,只需检查*F*+中每个非平凡FD左边是否为超键。

直观分析“满足BCNF”的关系表

- BCNF能确保关系表在FD信息视角下无冗余。



基本规范范式（3）

- **第三规范**
  - 令*R*是一关系模式，*X*与*A*分别是*R*的属性子集。如果对*R*中保持的每个FD:*X*→*A*，能至少满足以下三条件之一， 就称*R*满足第三范式，简记为*R*∈3NF。
  - A∈X，即*X*→*A*是一个平凡的FD；
  - *X*是一个超键；
  - *A*是*R*的部分键。
- **3NF比BCNF多了第三个条件，也允许*A*是键的一部分。显然，每个BCNF关系肯定是3NF关系**

3.4.4 无损分解与依赖保持分解

1. 无损分解

❖ 无损分解定义

- 令*R*为一关系模式，*F*是*R*上的FDs集
- 将*R*分解为两个属性组*X*和*Y*，如果对*R*的每个满足*F*的实例*r*，满足$\pi_X(r) \bowtie \pi_Y(r) = r$，就称该分解是无损连接的。

❖ 无损分解应用

- 将*R*分解为属性组*R1*和*R2*是无损连接的，当且仅当*R1*∩*R2*→ *R1*或*R1*∩*R2*→ *R2*保持。
- 举例：Hourly_Emps(*SNLRWH);* FD:*R*→*W*

一个不满足无损连接的分解示例（图3.14 ）

S	P	D
s1	p1	d1
s2	p2	d2
s3	p1	d3

实例r

S	P
s1	p1
s2	p2
s3	p1

$\pi_{SP}(r)$

P	D
p1	d1
p2	d2
p1	d3

$\pi_{PD}(r)$

S	P	D
s1	p1	d1
s2	p2	d2
s3	p1	d3
s1	p1	d3
s3	p1	d1

$\pi_{SP}(r) \bowtie \pi_{PD}(r)$

3.4.4 无损分解与依赖保持分解

2. 依赖保持分解

- 定义（依赖集投影）
  - 令关系*R*被分解为两个属性组*X*和*Y*,*F*是*R*上保持的FDs
  - *F*在*X*上的投影(F_X):是*F*+中那些仅包含*X*中属性的FDs
    - 依赖$U \rightarrow V$在F_x中，当且仅当*U*和*V*中的所有属性都在*X*中。
- 定义（依赖保持分解）
  - 带有FDs集*F*的关系模式*R*，分解为*X*和*Y*两个属性组是依赖保持的，当且仅当$(F_X \cup F_Y)^+ = F^+$。
- 依赖保持为什么考虑***F***闭包***F+***而不是***F*** ？