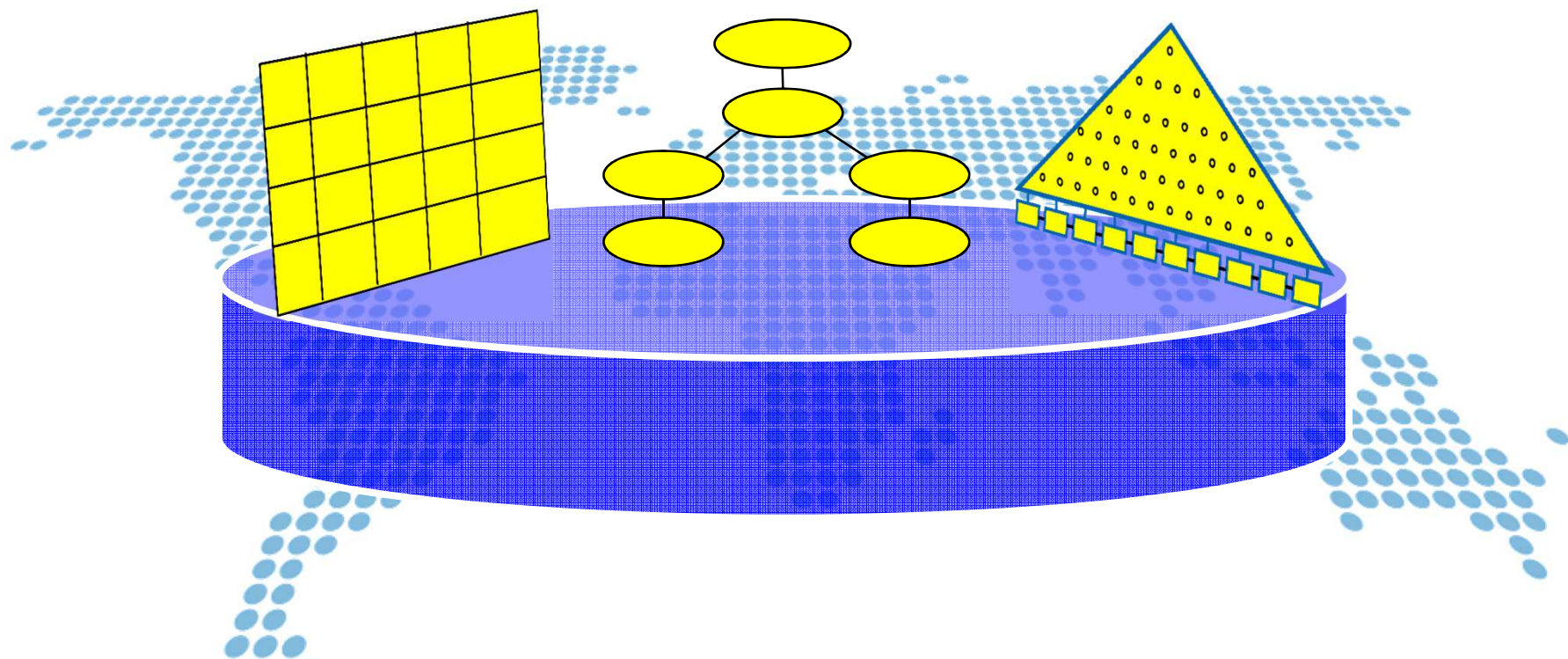


数据库系统

关系模型与SQL (3)

陈世敏

(中科院计算所)



上节内容

- 关系演算
 - 元组关系演算
 - 域关系演算
 - 关系演算与传统关系代数的表达能力
- 丰富的SQL Select功能
 - 扩展关系代数
 - 单个Select语句: aggregation, group by, having, order by
 - 嵌套Select语句: in, exists, unique, op ANY/ALL
- 完整性约束
 - domain, unique, primary key, not null
 - foreign key, 执行
 - check
 - Assertion, trigger

数据库设计过程

- 1) 需求分析
- 2) 概念设计 (ER模型等)
- 3) 逻辑设计 (ER模型→关系模型)
- 4) 模式细化 (规范化等)
- 5) 物理设计 (物理模式, 性能等)
- 6) 应用与安全设计 (定义外部模式等)

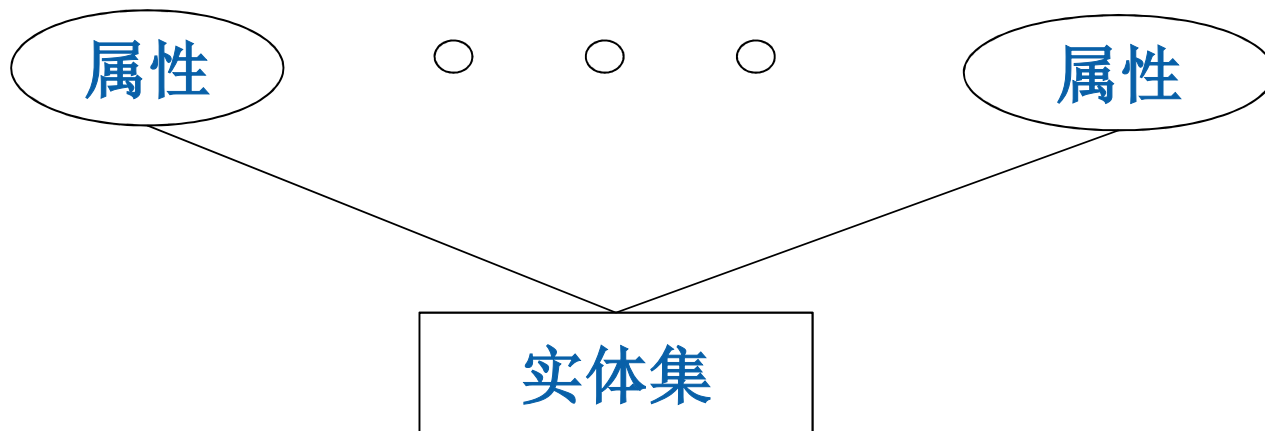
本节内容

- 物理设计本节只涉及一部分, 更多内容在数据库内部实现的基础上再讨论

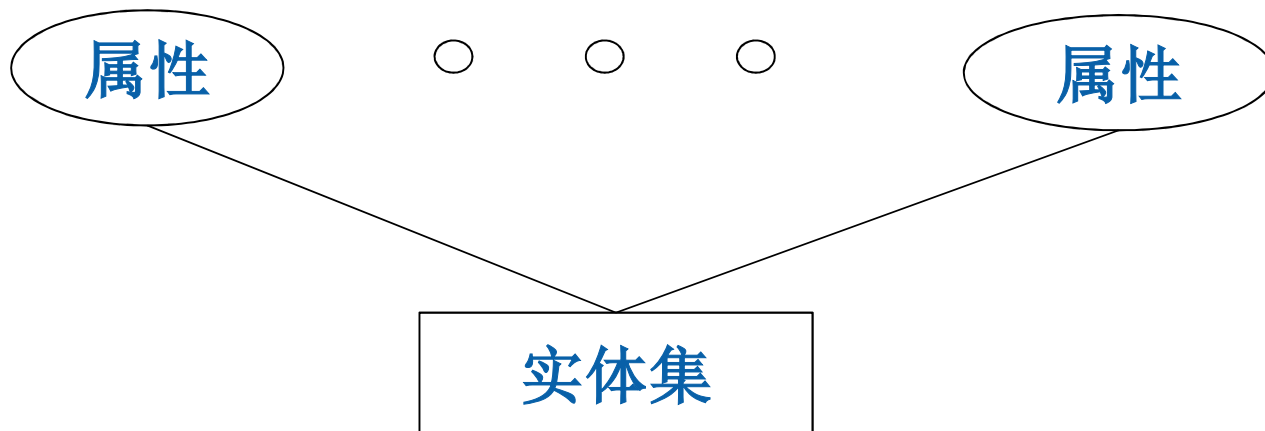
Outline

- 逻辑设计：ER图到关系模型
- 模式细化：范式
- 物理设计和外部模式设计：视图，索引

ER图实体集和属性的表示

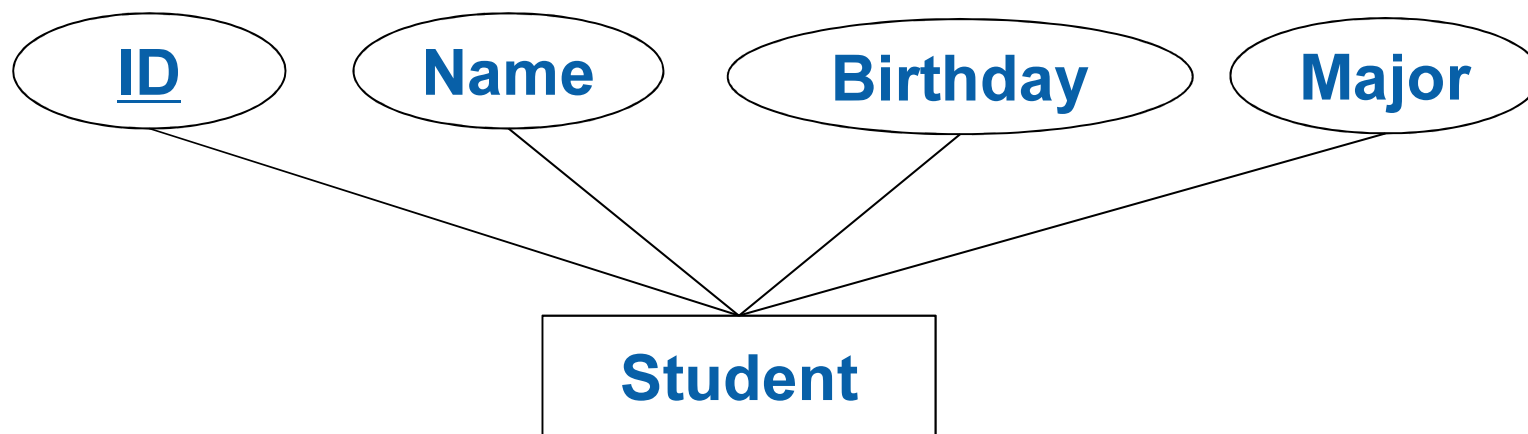


ER图的实体集→关系表



```
create table 实体集名 (  
    属性1 类型,  
    属性2 类型,  
    . . .  
);
```

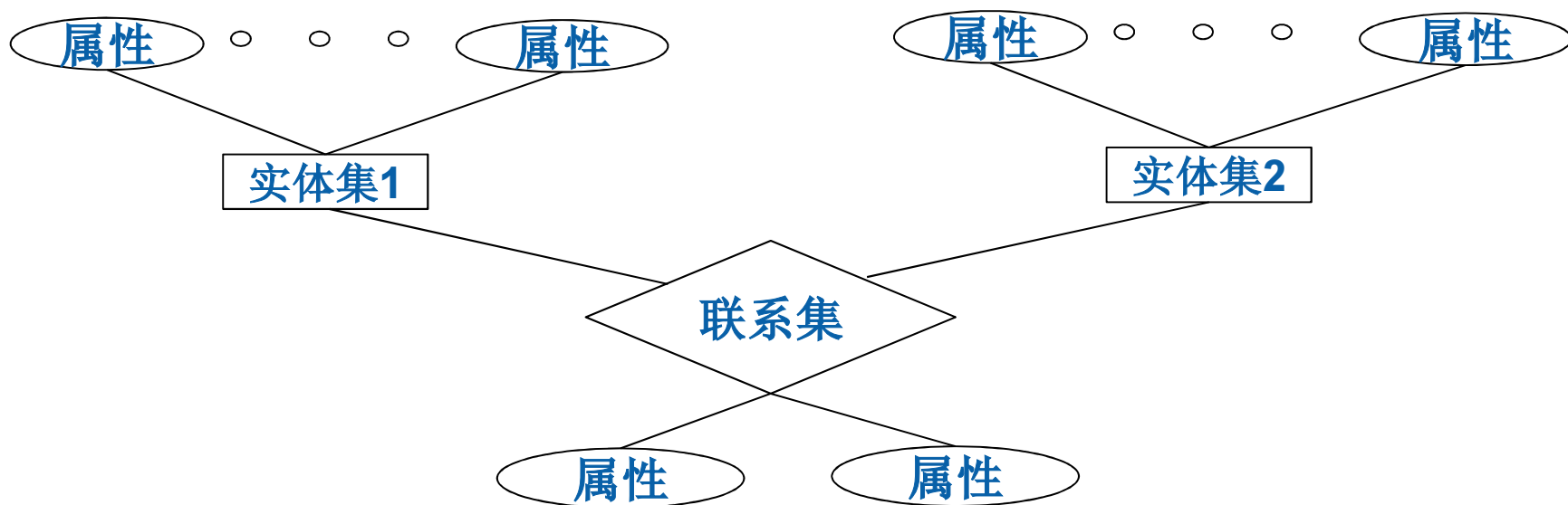
ER图的实体集→关系表：举例



```
create table Student (  
    ID integer primary key, ←——下划线对应主键  
    Name varchar(20),  
    Birthday date,  
    Major varchar(20)  
);
```

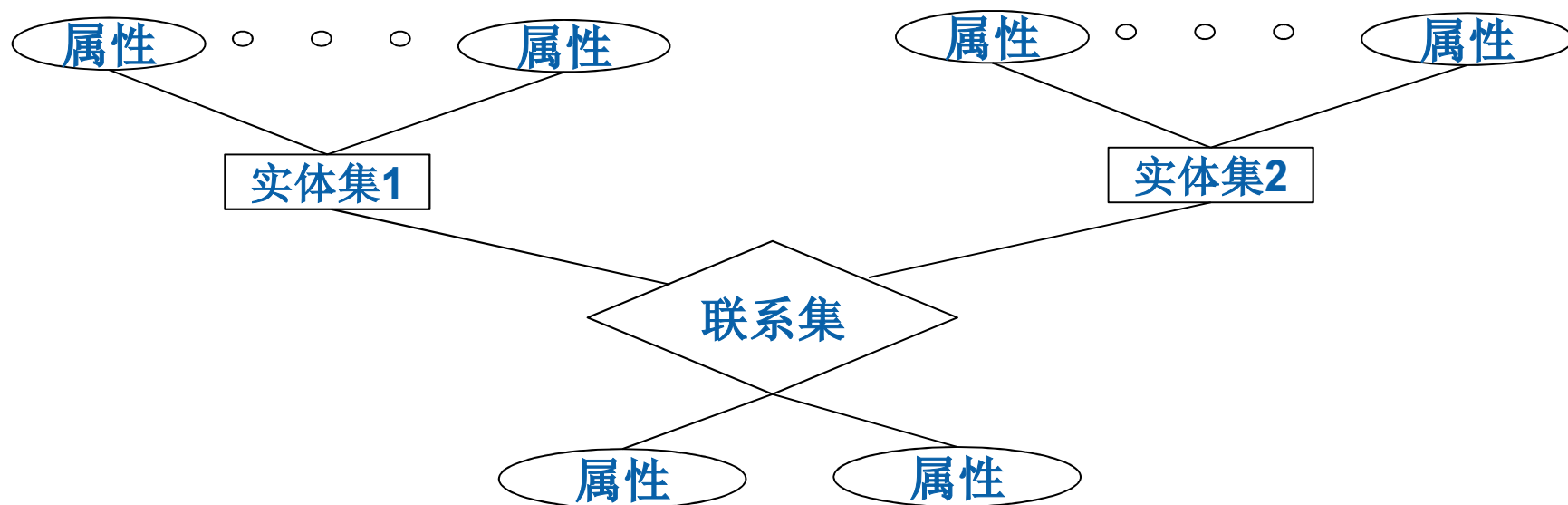
ER图的联系集→关系表

- 联系集也可以映射为关系表



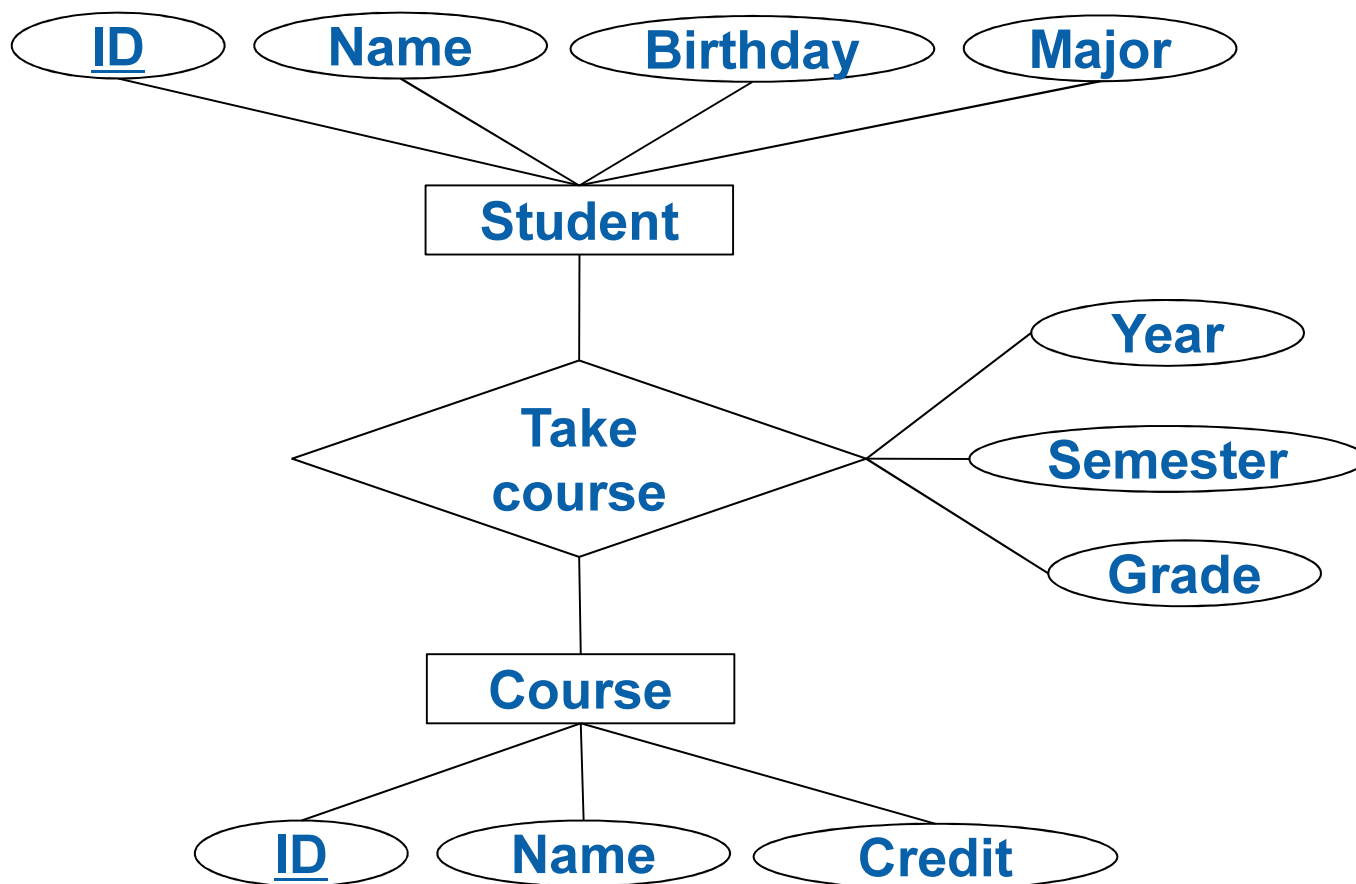
- 包含来自参与实体的主键，作为外键
- 包含联系集的属性

ER图的联系集→关系表

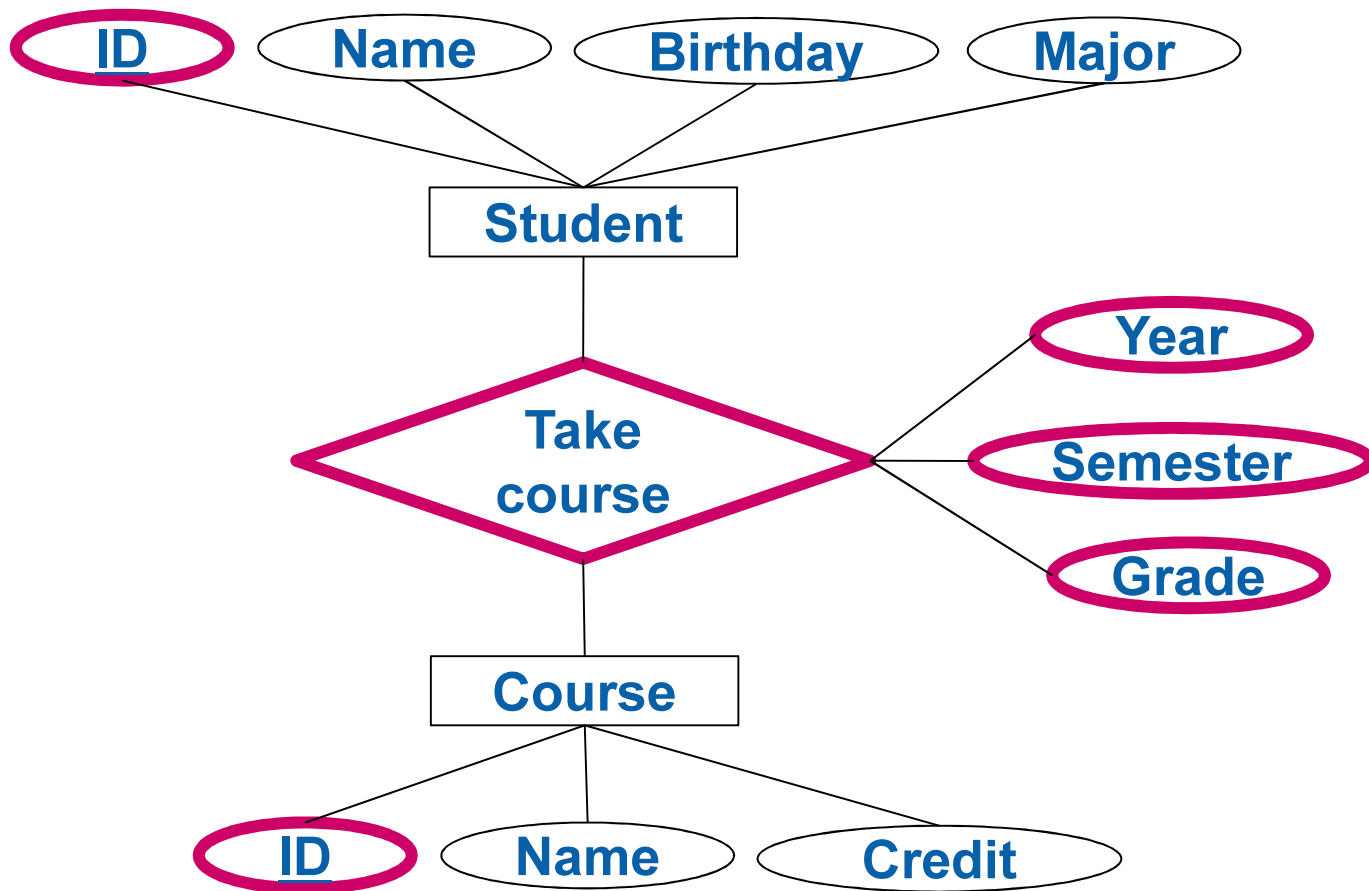


```
create table 联系集名 (  
    实体集1主键 类型, 实体集2主键 类型,  
    。 。 。  
    联系集属性1 类型, 联系集属性2 类型,  
    。 。 。  
    foreign key ...  
);
```

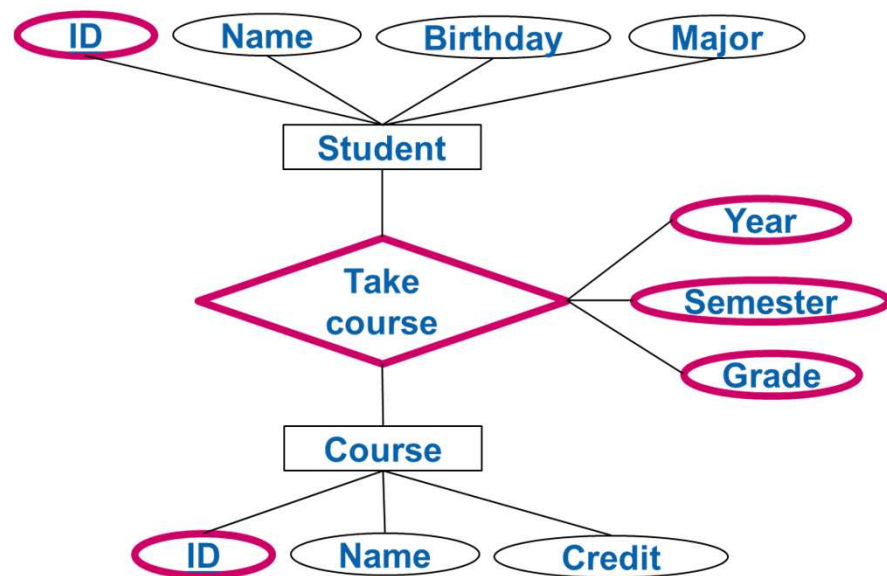
ER图的联系集→关系表：举例



ER图的联系集→关系表：举例

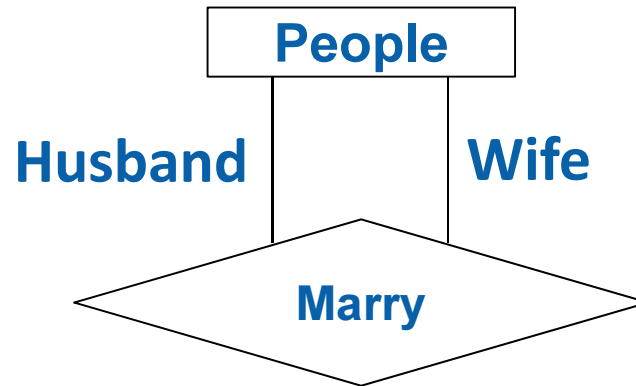


ER图的联系集→关系表：举例



```
create table TakeCourse (  
    CourseID integer,  
    StudentID integer,  
    Year year,  
    Semester enum('Spring', 'Summer', 'Fall'),  
    Grade float,  
    primary key (CourseID, StudentID),  
    foreign key (CourseID) references Course(ID),  
    foreign key (StudentID) references Student(ID)  
);
```

单一实体集内部的联系

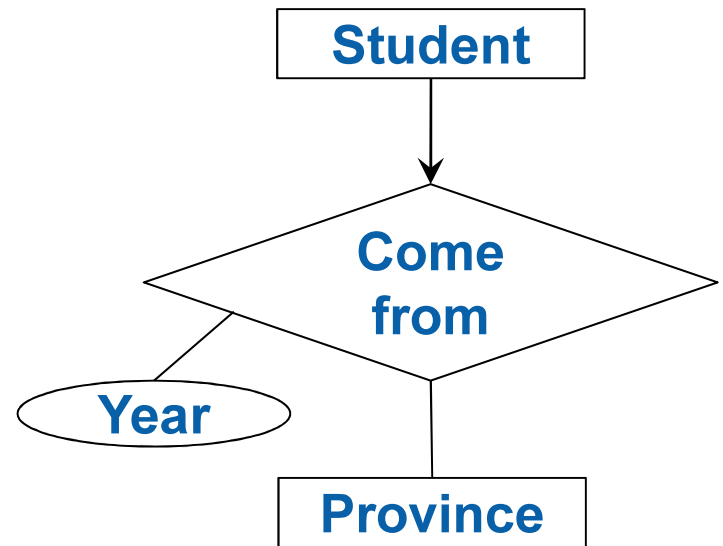


```
create table Marry (  
    HusbandID integer,  
    WifeID integer,  
    primary key (HusbandID, WifeID),  
    foreign key (HusbandID) references People(ID),  
    foreign key (WifeID) references People (ID)  
);
```

ER图的联系集：Key Constraint

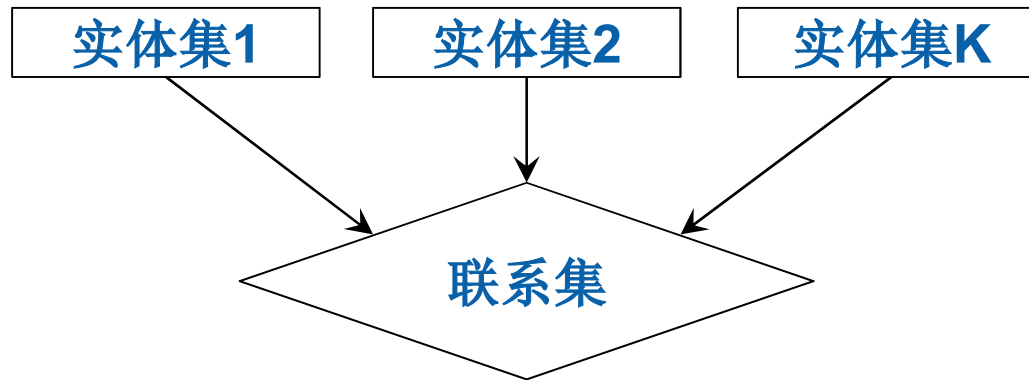
- 每个Student至多参与一个ComeFrom联系

□ 于是Student的主键也是ComeFrom的候选键



```
create table ComeFrom (  
    StudentID integer,  
    ProvinceID integer,  
    Year integer,  
    primary key (StudentID),  
    foreign key (StudentID) references Student(ID),  
    foreign key (ProvinceID) references Province(ID)  
);
```

ER图的联系集：Key Constraint

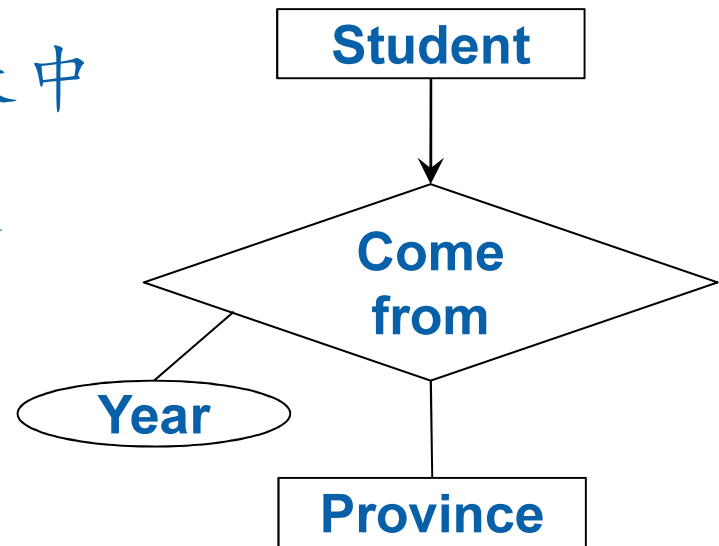


主要注意的是联系集的键

- 每个键约束的实体集，其主键都是联系集的候选键
- 选择其中一个为联系集的主键（primary key）
- 其它为联系集候选键（unique）

ER图的联系集：Key Constraint，方法2

- 把ComeFrom归并到Student表中
- 不需要单独建立ComeFrom表



create table *Student* (

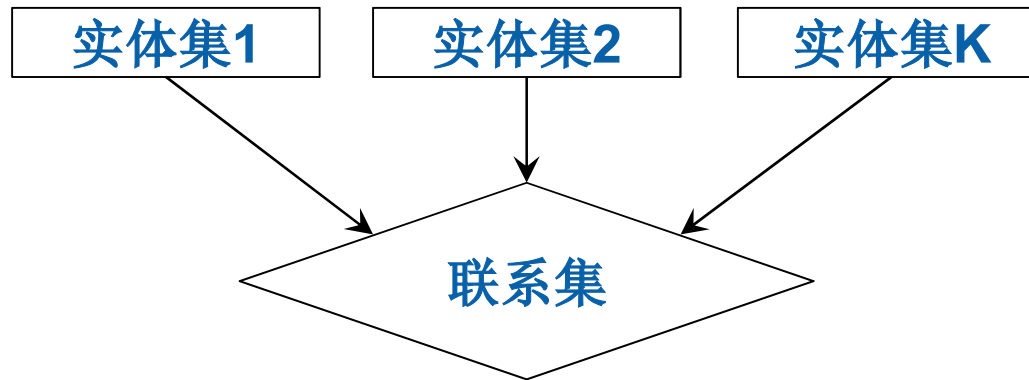
.....

ProvinceID integer,

Year integer,

foreign key (*ProvinceID*) references *Province*(*ID*)
);

ER图的联系集：Key Constraint，方法2



- 选择任何一个键约束的实体集，假设其关系为R
- 扩展R，增加下述列
 - 联系集的属性
 - 参与联系的其它实体集的主键，增加外键

ER图的联系集：

Key Constraint + Total Participation

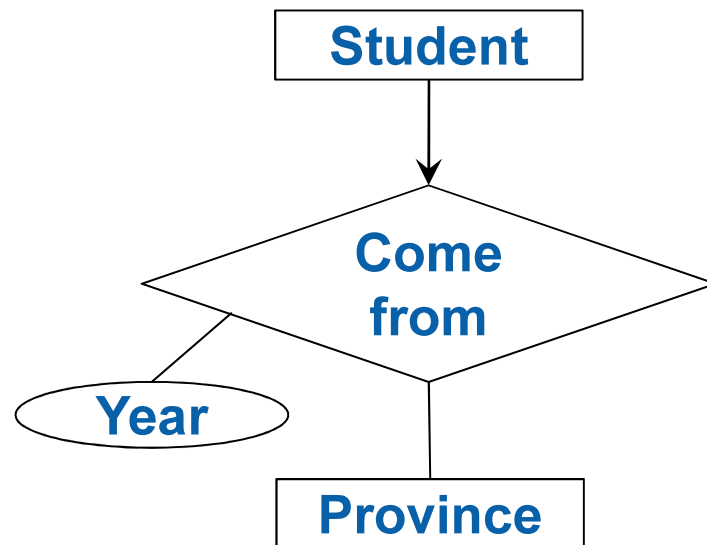
- Key Constraint + Total Participation

- Key Constraint: 至多参与1次
- Total Participation: 至少参与1次
- 实体集每个实体参与1次且仅参与1次联系

- 那么可以在上述方法2上增加not null

ER图的联系集：举例

Key Constraint + Total Participation



create table *Student* (

.....

ProvinceID integer not null,

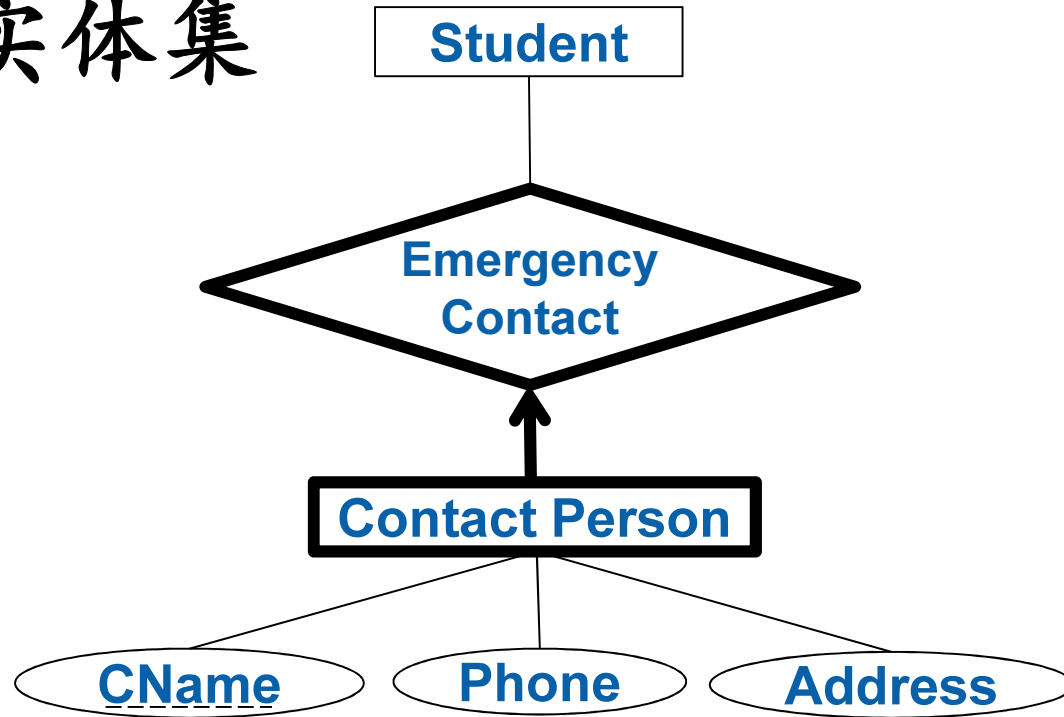
Year integer not null,

foreign key (*ProvinceID*) references *Province*(*ID*)
);

ER图的联系集：普通Total Participation？

- 普通的完全参与，无法简单表达
- 只能用Check或Assertion来表达
 - 代价很高
- 通常就不勉强了

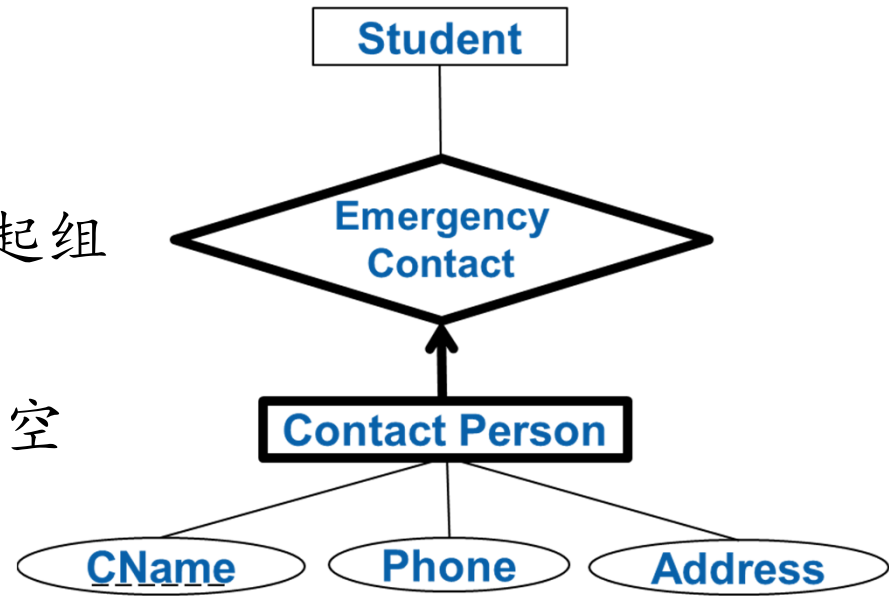
ER图的弱实体集



- 弱实体满足key constraint + total participation
 - 所以可以用前面的方法
- 不同点
 - 部分键
 - 删除主实体时，也删除相应的弱实体

ER图的弱实体集

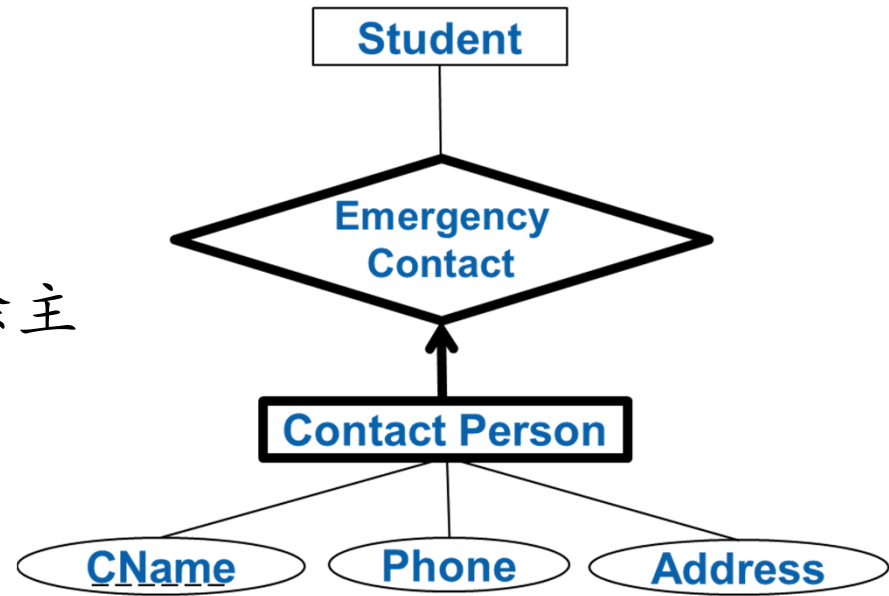
- 部分键必须和主实体的主键一起组成主键
- Primary key 保证了StudentID 非空



```
create table ContactPerson (  
    StudentID integer,  
    CName varchar(20),  
    Phone integer, Address varchar(100),  
    primary key (StudentID, CName),  
    foreign key (StudentID) references Student(ID)  
        on delete cascade  
);
```

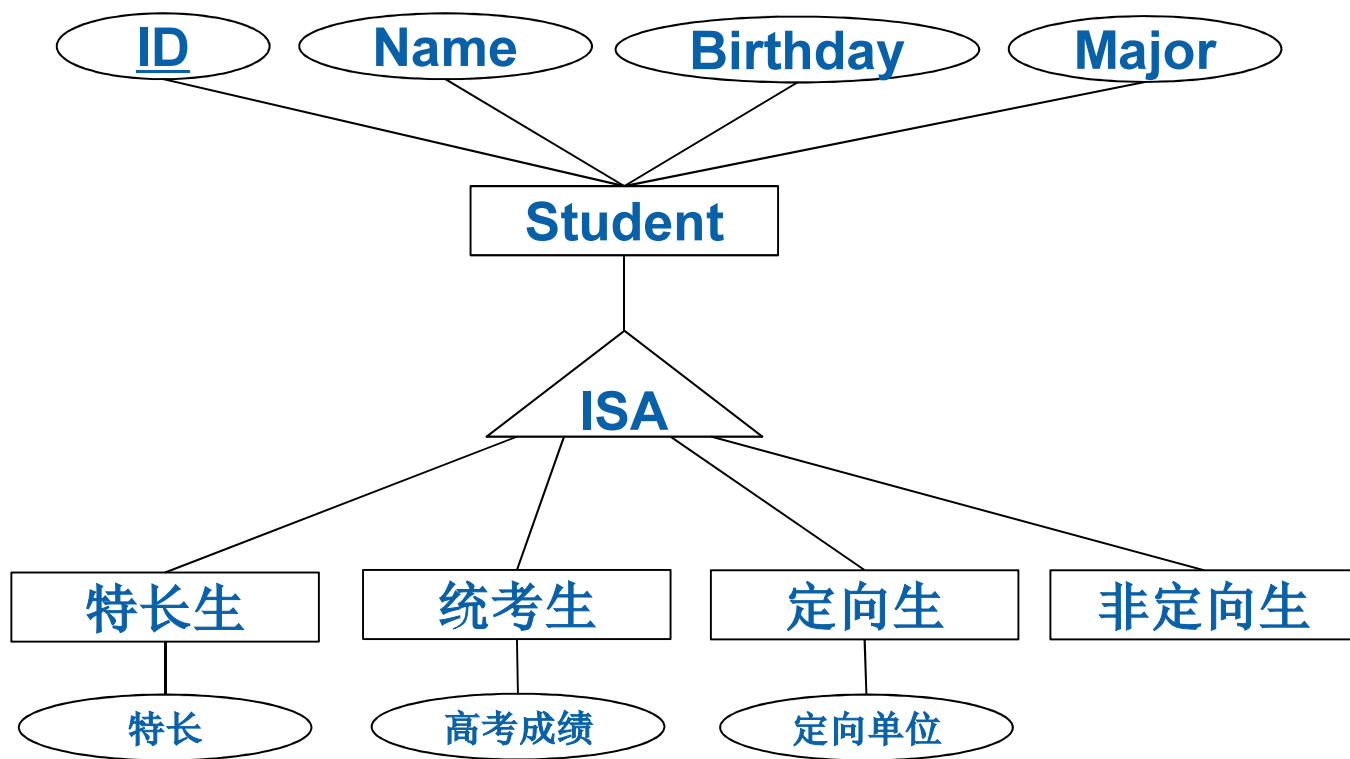
ER图的弱实体集

- StudentID 外键引用采用了 on delete cascade，保证在删除主实体的同时删除弱实体



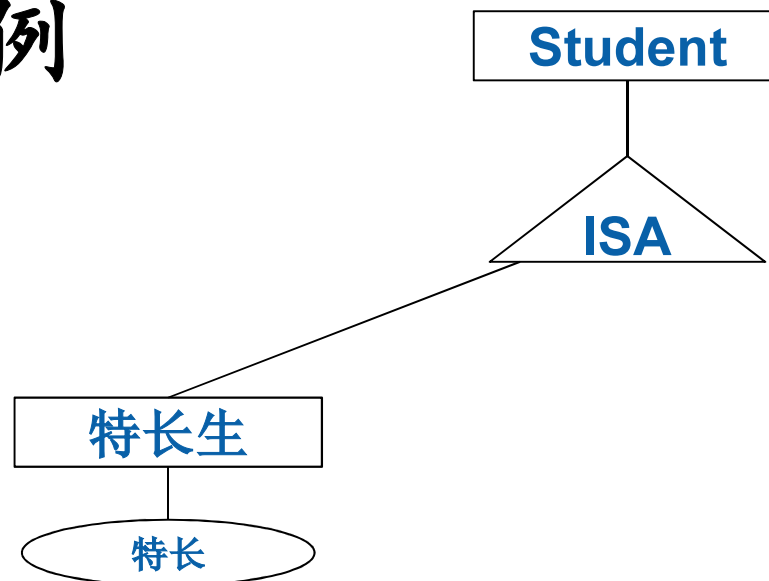
```
create table ContactPerson (  
    StudentID integer,  
    CName varchar(20),  
    Phone integer, Address varchar(100),  
    primary key (StudentID, CName),  
    foreign key (StudentID) references Student(ID)  
        on delete cascade  
);
```

ER图的类层次



- 把子实体创建为一个关系
- 在子实体的关系中引用父实体的主键

ER图的类层次：举例



```
create table StudentWSpecialty (  
    StudentID integer,  
    Specialty varchar(20),  
    primary key (StudentID),  
    foreign key (StudentID) references Student(ID)  
        on delete cascade  
);
```

Outline

- 逻辑设计：ER图到关系模型
- 模式细化：范式
 - 数据冗余的问题
 - 范式介绍
 - 函数依赖与2NF,3NF,BCNF
 - 分解为BCNF
 - 多值依赖和连接依赖
- 物理设计和外部模式设计：视图，索引

数据冗余的问题

- 冗余：同样的数据在数据库中被存储了多次
- 问题
 - 冗余存储
 - 更新异常
 - 插入异常
 - 删除异常

数据冗余的问题：举例

Employee

ID	Name	Rating	Wage
1	Alan	5	30
2	Bob	7	50
3	Carol	7	50
4	Dan	5	30

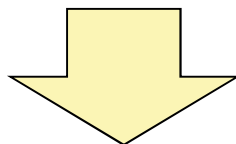
- 假设隐含的条件：相同的rating，工资wage也相同
- 问题
 - 冗余存储：例如5与30之间的关系存了2次
 - 更新异常：修改wage时，必须修改所有rating相同的记录，否则就不一致了
 - 插入异常：如果不知道rating对应的wage，就无法正确插入
 - 删除异常：删除了所有具有rating=5的值时，关于rating=5的wage信息也消失了

模式分解

Employee

ID	Name	Rating	Wage
1	Alan	5	30
2	Bob	7	50
3	Carol	7	50
4	Dan	5	30

可以通过分解模式来消除冗余



Employee

ID	Name	Rating
1	Alan	5
2	Bob	7
3	Carol	7
4	Dan	5

WageRate

Rating	Wage
5	30
7	50

冗余的类型

- 函数依赖 (Functional Dependency, FD)
- 多值依赖 (Multivalued Dependency, MVD)
- 连接依赖 (Join Dependency, JD)

范式 (Normal Form)

- 范式就是规范的形式
- 关系模式的每种范式：消除了一种冗余
- 模式求精细化需要使设计符合一定的范式

范式 (Normal Form)

1NF: 所有属性 (列) 都是原子类型

2NF: 消除函数依赖中的部分依赖

3NF: 消除函数依赖中的非键传递依赖

BCNF: 消除所有函数依赖 (希望达到)

4NF: 多值依赖

5NF: 连接依赖

6NF及其它范式(不介绍)

使用传统关系代数

- 在关系模式设计时，消除冗余
- 那么，关系将是一个集合，而不是包/多集
- 所以，我们下面都将用传统关系代数

Outline

- 逻辑设计：ER图到关系模型
- 模式细化：范式
 - 数据冗余的问题
 - 范式介绍
 - 函数依赖与2NF,3NF,BCNF
 - 分解为BCNF
 - 多值依赖和连接依赖
- 物理设计和外部模式设计：视图，索引

函数依赖 (FD)

- $R(U)$ 是属性集 $U = \{A_1, \dots, A_m\}$ 上的一个关系模式
- $X \subseteq U, Y \subseteq U$
- 函数依赖 $X \rightarrow Y$
 - 任意两个元组在 X 上取值相同 \Rightarrow 它们在 Y 上取值也相同
 - 对于任意的实例 r , $\forall t_1, t_2 \in r$, 若 $t_1[X] = t_2[X]$, 则 $t_1[Y] = t_2[Y]$
- X 函数确定 Y , Y 函数依赖于 X

函数依赖：举例

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

$AB \rightarrow C$

观察一下

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

$AB \rightarrow C$

- 候选键是什么？

- 包含1列：A✗, B✗, C✗, D✗

- 包含2列：AB✗, AC✗, AD✗, BC✗, BD✗, CD✓

- 包含3列：ABC✗, ABD✓, ACD✗, BCD✗

- 上述函数依赖中AB不是候选键，（找候选键复杂性可以是指数级的）

- $A \rightarrow C?$ ✗

- $A \rightarrow A?$ ✓

- $ABD \rightarrow A?$ ✓

候选键是一种特殊的函数依赖

- $R(U)$ 是属性集 $U = \{A_1, \dots, A_m\}$ 上的一个关系模式
- $X \subseteq U$ 是候选键, 如果满足下述条件
 - $X \rightarrow U$ (X函数确定U)
 - $\forall Y \subset X, Y \nrightarrow U$ (X是最小的)
- 满足第一个条件的叫Super key (超键)
- 两个条件都满足就是候选键
 - 候选键也是超键

函数依赖的推导

- Armstrong规则

- 自反律: 如果 $Y \subseteq X$, 那么 $X \rightarrow Y$
- 增补律: 如果 $X \rightarrow Y$, 那么任取 Z , $XZ \rightarrow YZ$
- 传递律: 如果 $X \rightarrow Y$ 而且 $Y \rightarrow Z$, 那么 $X \rightarrow Z$

- 可以得到

- 分解: 如果 $X \rightarrow YZ$, 那么 $X \rightarrow Y$ 而且 $X \rightarrow Z$
- 合并: 如果 $X \rightarrow Y$ 而且 $X \rightarrow Z$, 那么 $X \rightarrow YZ$

分解的证明

- 如果 $X \rightarrow YZ$, 那么 $X \rightarrow Y$ 而且 $X \rightarrow Z$

- 证明:

$$\because X \rightarrow YZ$$

$$\because YZ \rightarrow Y \text{ (自反律)}$$

$$\therefore X \rightarrow Y \text{ (传递律)}$$

同理, 可以证明 $X \rightarrow Z$

证明完毕。

合并的证明

- 如果 $X \rightarrow Y$ 而且 $X \rightarrow Z$, 那么 $X \rightarrow YZ$

- 证明:

$$\because X \rightarrow Z$$

$$\therefore XX \rightarrow XZ \text{ (增补律)}$$

$$\therefore X \rightarrow XZ \text{ (} XX \text{ 就是 } X \text{)} \quad (1)$$

$$\because X \rightarrow Y$$

$$\therefore XZ \rightarrow YZ \text{ (增补律)} \quad (2)$$

由 (1) (2), $X \rightarrow YZ$ (传递律)
证明完毕。

属性集 $X \rightarrow$ 单个属性 A

- 分解：如果 $X \rightarrow YZ$ ，那么 $X \rightarrow Y$ 而且 $X \rightarrow Z$
- 合并：如果 $X \rightarrow Y$ 而且 $X \rightarrow Z$ ，那么 $X \rightarrow YZ$
- 关心 “属性集 $X \rightarrow$ 单个属性 A ” 形式的函数依赖
 - 合并和分解是互为可逆的
 - 只要得到对于所有单一属性的函数依赖关系
 - 就可以合并得到所有的函数依赖关系

函数依赖的闭包

- 给定了一个函数依赖组成的集合F
 - 根据F可以推导出更多的函数依赖
- 函数依赖集F的闭包： F^+
 - 从F可以推导出的所有函数依赖的集合
 - 反复使用Armstrong规则，直到不增加为止

举例：函数依赖的闭包

Contracts(contractid, supplierid, projectid,
deptid, partid, quantity, value)

- 我们用一个字母来简记各列CSJDPQV
 - 函数依赖关系
 - 其中C是主键，所以 $C \rightarrow SJDPQV$
 - 一个项目买一种零件只用一个合同： $JP \rightarrow C$
 - 一个部门从一个供货商至多购买一种零件： $DS \rightarrow P$
 - 已知 $F = \{C \rightarrow SJDPQV, JP \rightarrow C, DS \rightarrow P\}$
- ☞ 求函数依赖集的闭包 F^+

重新想一下

• 自反律

□ 如果 $Y \subseteq X$, 那么 $X \rightarrow Y$

□ 假设 X 中有 k 个属性, 那么有多少种可能的自反律规律?

– 换言之, X 有多少个不同的非空子集?

– 指数个: $2^k - 1$

□ 假设关系模式中有 m 个属性, 那么有多少种可能的自反律?

– 包含 1 个属性的集合 X 有 $\binom{m}{1}$ 个, 有 $2^1 - 1$ 种自反律

– ...

– 包含 K 个属性的集合 X 有 $\binom{m}{k}$ 个, 有 $2^k - 1$ 种自反律

– ...

– 总计有: $\sum_{k=1}^m \binom{m}{k} (2^k - 1)$

• 列举自反律产生的函数依赖: 费力而没有什么意义

重新想一下

- 增补律

- 如果 $X \rightarrow Y$ ，那么任取 Z ， $XZ \rightarrow YZ$

- 假设除去 X 和 Y 之外，还有 l 个属性，那么有多少种可能的增补律规律？

- 指数个： $2^l - 1$

- 列举增补律产生的函数依赖：费力而没有很多意义

重新想一下

- 传递律

□ 如果 $X \rightarrow Y$ 而且 $Y \rightarrow Z$, 那么 $X \rightarrow Z$

- 相对于自反律和增补律而言, 很少!
- 所以, 我们的重点是传递律的规律

传递律产生的规律？

- 已知 $F = \{C \rightarrow SJDPQV, JP \rightarrow C, DS \rightarrow P\}$

求函数依赖集的闭包 F^+

- $C \rightarrow SJDPQV$ ，所以 $C \rightarrow CSJDPQV$ （增补律）
- $JP \rightarrow C$ ，所以 $JP \rightarrow CSJDPQV$ （传递律）
- $DS \rightarrow P$ ，所以 $JDS \rightarrow JP$ （增补律）
所以 $JDS \rightarrow CSJDPQV$ （传递律）
- 所以，这里 C 、 JP 、 JDS 都是候选键

对单个属性的函数依赖

- 已知 $F = \{C \rightarrow SJDPQV, JP \rightarrow C, DS \rightarrow P\}$
- $C \rightarrow CSJDPQV, JP \rightarrow CSJDPQV, JDS \rightarrow CSJDPQV$
- 分解可以得到
 - $C \rightarrow S, C \rightarrow J, C \rightarrow D, C \rightarrow P, C \rightarrow Q, C \rightarrow V$
 - $JP \rightarrow C, JP \rightarrow S, JP \rightarrow J, JP \rightarrow D, JP \rightarrow P, JP \rightarrow Q, JP \rightarrow V$
 - $JDS \rightarrow C, JDS \rightarrow S, JDS \rightarrow J, JDS \rightarrow D, JDS \rightarrow P, JDS \rightarrow Q, JDS \rightarrow V$
- 当然还有自反律和增补律可以产生的大量规律

属性集的闭包

- 给定了一个函数依赖组成的集合F
 - 根据F可以推导出更多的函数依赖
- 属性集X的闭包： X^+
 - 相对于一个函数依赖集F
 - 计算X函数确定的所有属性集

属性闭包算法

closure = X;

do {

 changed = false;

 if (存在($Y \rightarrow Z \in F$) 且 ($Y \subseteq \text{closure}$))

 then { closure = closure \cup Z; changed= true;}

} while (changed);

最后closure包含X函数确定的所有属性

举例

- 已知 $F = \{C \rightarrow SJDPQV, JP \rightarrow C, DS \rightarrow P\}$
- 求 JDS 属性的闭包？
- 初始化, $\text{Closure} = \{J, D, S\}$
- $DS \rightarrow P$ 且 $DS \subseteq \text{Closure}$, $\text{Closure} = \{S, J, D, P\}$
- $JP \rightarrow C$ 且 $JP \subseteq \text{Closure}$, $\text{Closure} = \{C, S, J, D, P\}$
- $C \rightarrow SJDPQV$ 且 $C \subseteq \text{Closure}$, $\text{Closure} = \{C, S, J, D, P, Q, V\}$

函数依赖

- 定义

- 规律

- 自反律, 增补律, 传递律
 - 分解, 合并

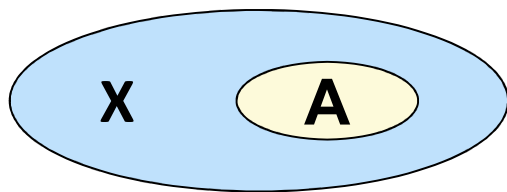
- 闭包

- 函数依赖的闭包
 - 属性的闭包

- 函数依赖与2NF,3NF,BCNF的关系

可能存在的函数依赖

“属性集 $X \rightarrow$ 单个属性 A ”

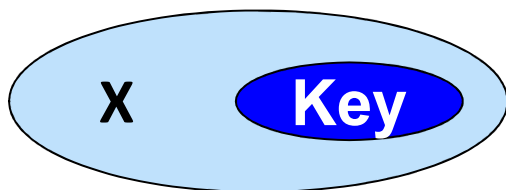


平凡依赖✓

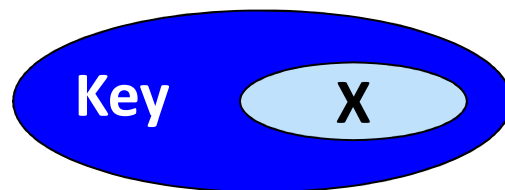
$\{A\} \subseteq X$ ，由自反律知道函数依赖存在，这种函数依赖是正常的

☞ 接下来，我们考虑 X 和 A 与键的关系

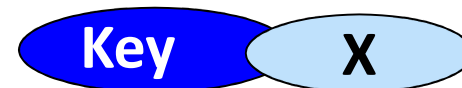
“ $X \rightarrow A$ ”与键的关系



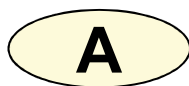
X包含候选键



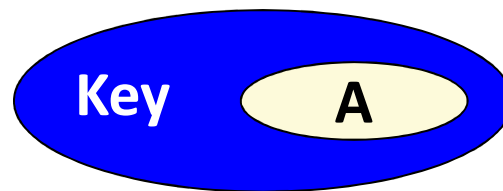
X是候选键的一部分



其它情况：X包含非键的属性，又不包含完整的候选键



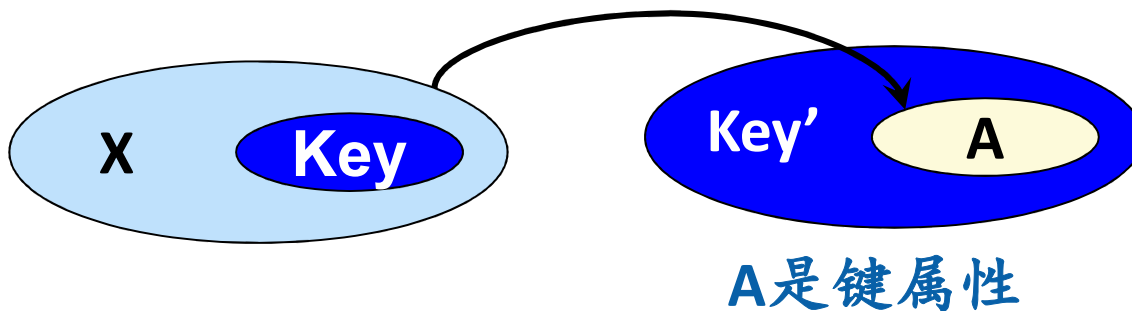
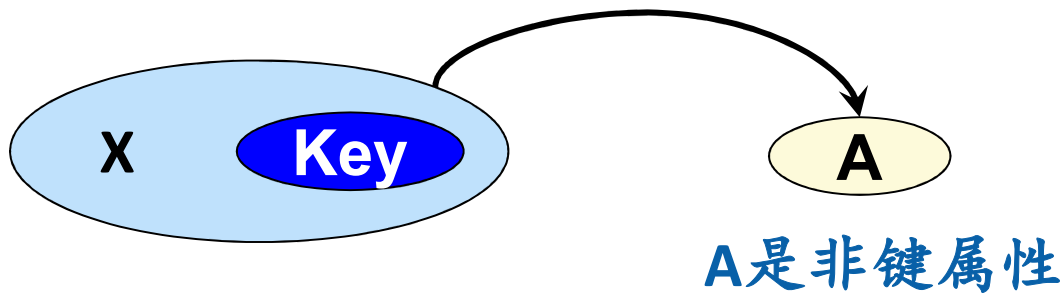
A是非键属性



A是键属性

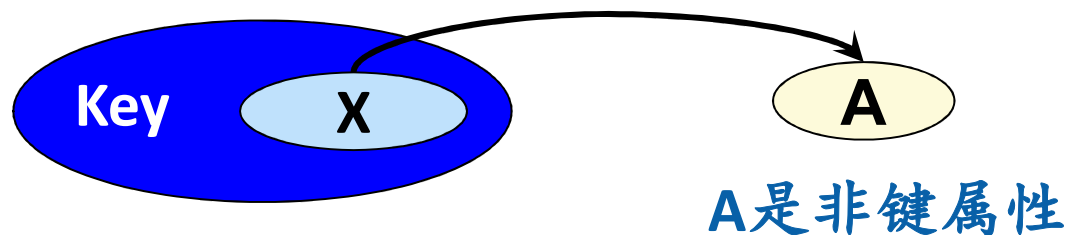
所以，有 $3 \times 2 = 6$ 种组合

X是超键✓



- X是超键，X当然可以函数确定任何属性
- 所以，这是很正常的

部分依赖



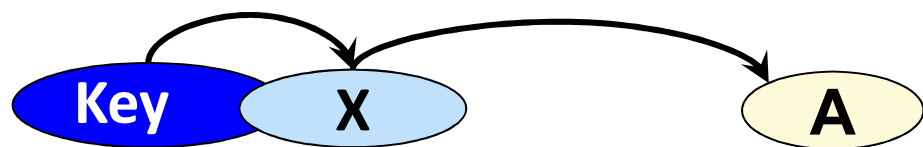
- 这会导致冗余

- 例如：

```
create table Reserves(sid integer, bid integer, day date,  
                     credit_card char(16) );
```

- 主键是sid, bid, day的组合
- 但是sid -> credit_card

非键传递依赖



A是非键属性

- 这会导致冗余

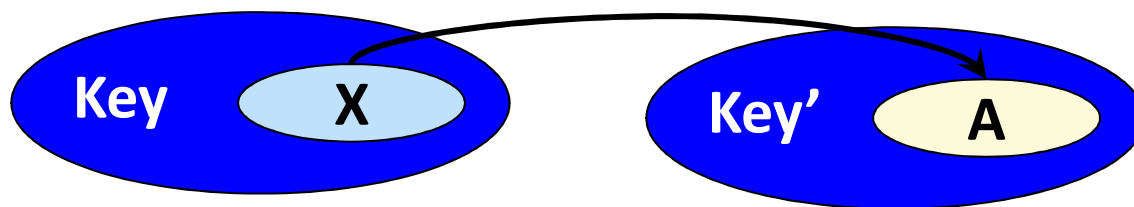
- 例如：

- 在前面例子中Employee表中，
主键为ID，
但是存在Rating -> Wage

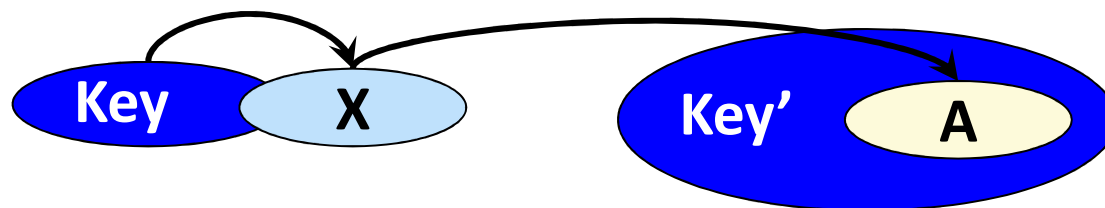
Employee

ID	Name	Rating	Wage
1	Alan	5	30
2	Bob	7	50
3	Carol	7	50
4	Dan	5	30

对于键属性的函数依赖



A是键属性



A是键属性

- 这些都可能产生冗余

- 一个例子:

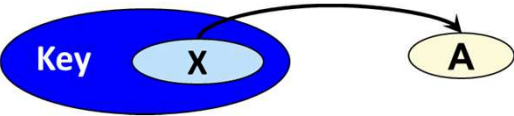
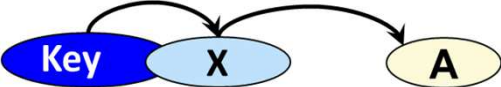
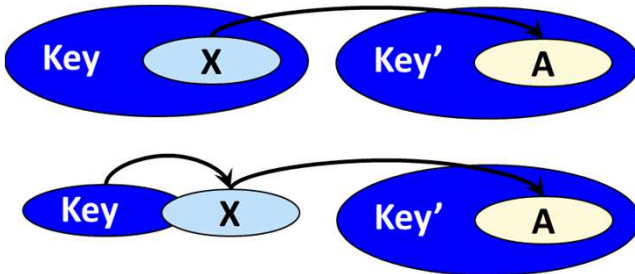
□ ABD和CD是候选键

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

$AB \rightarrow C$

消除函数依赖的范式

Boyce-Codd

	2NF	3NF	BCNF
平凡依赖	✓	✓	✓
X是超键	✓	✓	✓
部分依赖 	😊	😊	😊
非键传递依赖 	😞	😊	😊
对于键属性的函数依赖 	😞	😞	😊

Outline

- 逻辑设计：ER图到关系模型
- 模式细化：范式
 - 数据冗余的问题
 - 范式介绍
 - 函数依赖与2NF,3NF,BCNF
 - 分解为BCNF
 - 多值依赖和连接依赖
- 物理设计和外部模式设计：视图，索引

关系模型的分解

- $R(U)$ 是属性集 $U = \{A_1, \dots, A_m\}$ 上的一个关系模式
- $X \subseteq U, Y \subseteq U$
- 对 R 进行 X 和 Y 的分解
 - 把 R 分解为 $\pi_X(R)$ 和 $\pi_Y(R)$ (传统关系代数)
- 注意：关系模型的分解是通过投影实现的
 - 投影后要去重

无损分解

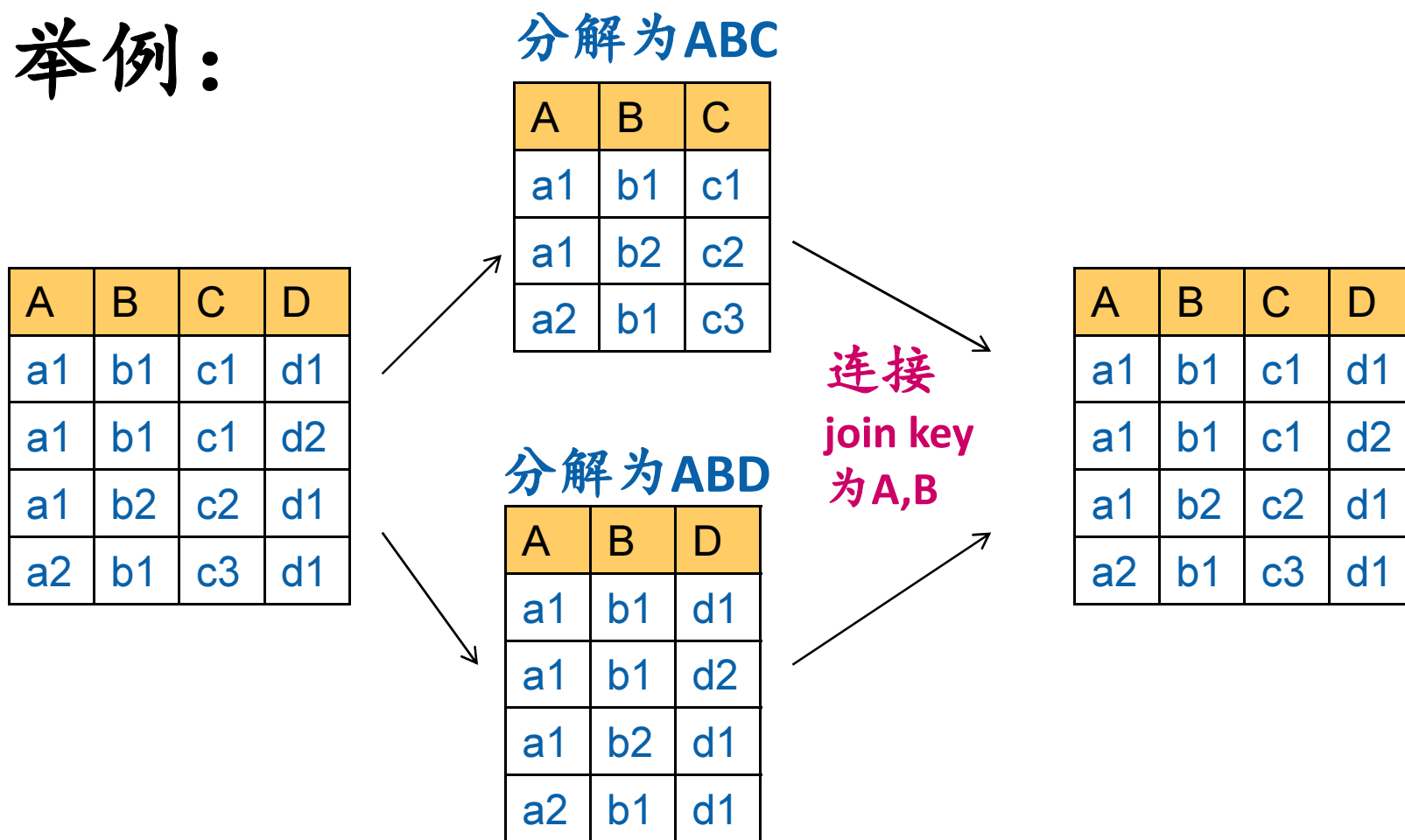
- 对R进行X和Y的分解
- 无损分解
 - 分解后两个表的自然连接可以得到原始表
 - $\pi_X(R) \bowtie \pi_Y(R) = R$ (传统关系代数)

举例：

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

- ABC和ABD的分解是无损的吗？

举例：



- ABC和ABD的分解是无损的

举例：

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

- AB和ACD的分解是无损的吗？

举例：

分解为AB

A	B
a1	b1
a1	b2
a2	b1

分解为ACD

A	C	D
a1	c1	d1
a1	c1	d2
a1	c2	d1
a2	c3	d1

连接
join key
为A

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b1	c2	d1
a1	b2	c1	d1
a1	b2	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

- AB和ACD的分解不是无损的

函数依赖与无损分解

- $R(U)$ 是属性集 $U = \{A_1, \dots, A_m\}$ 上的一个关系模式
- 如果 $R1 \cup R2 = R$ 且 $R1 \cap R2 \rightarrow R1$,
那么对 R 进行 $R1$ 和 $R2$ 的分解是无损分解
- 理解一下
 - $R1 \cap R2$ 是 $\pi_{R1}(R) \bowtie \pi_{R2}(R)$ 这个连接的 join key 部分
 - 如果 $\text{join key} \rightarrow R1$ 成立, 那么 join key 是 $R1$ 的主键, $R2$ 的外键
 - 所以这是主键-外键之间的连接
 - 这种 join 是无损的

我们希望设计的关系模式是BCNF

- BCNF只允许 $X \rightarrow A$

- 平凡依赖: $A \in X$

- X 是超键

- 如果不满足BCNF

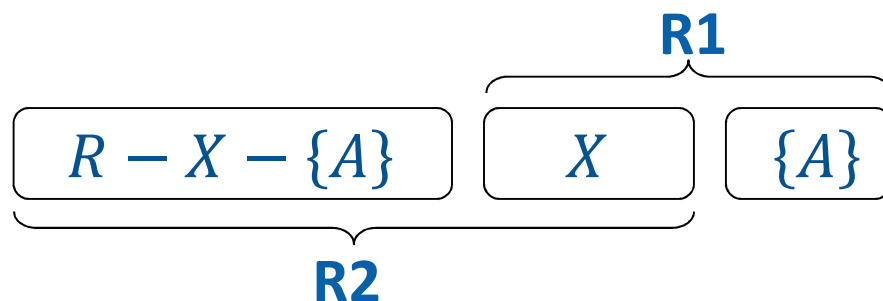
- ☞ 通过关系模式的无损分解来得到符合BCNF的模式

无损分解为BCNF

- 如果 $X \rightarrow A$ 违反了BCNF
 - $X \subset R$
 - $A \in R$ 是单独属性
 - $A \notin X$, 不是平凡依赖
- 那么对 R 进行 XA 和 $R-A$ 的分解是无损的

无损分解为BCNF

- 如果 $X \rightarrow A$ 违反了BCNF
- 那么对R进行XA和R-A的分解是无损的



- 设 $R1 = X \cup \{A\}$, $R2 = R - \{A\}$, 满足前述无损分解定理
 - $R1 \cup R2 = R$: $(X \cup \{A\}) \cup (R - \{A\}) = R$
 - $R1 \cap R2 \rightarrow R1$: $(X \cup \{A\}) \cap (R - \{A\}) = X$, $X \rightarrow X \cup \{A\}$

分解为BCNF：算法

```
while (存在关系R和 $X \rightarrow A$ 不满足BCNF) {  
    对R进行XA和R-A的分解;  
}
```


举例

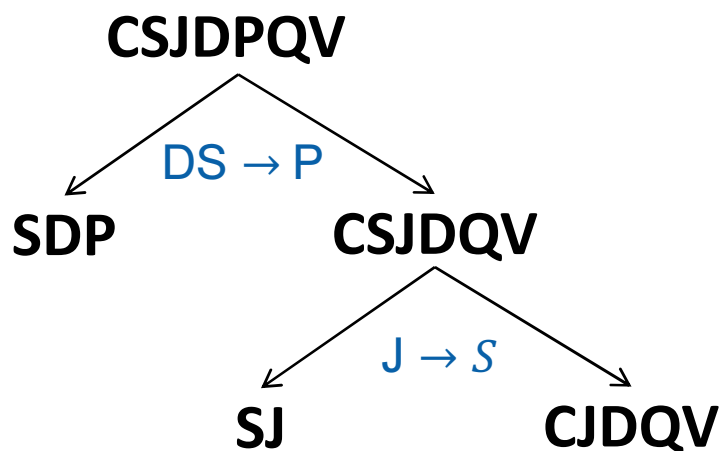
Contracts(contractid, supplierid, projectid,
deptid, partid, qty, value)

- 我们用一个字母来简记各列CSJDPQV
- 函数依赖关系 $F = \{C \rightarrow SJDPQV, JP \rightarrow C, DS \rightarrow P, J \rightarrow S\}$
 - 比前例增加了 $J \rightarrow S$, “每个项目只能有一个供货商”
- 是BCNF吗?
 - 不是, DS不是超键, J也不是超键

举例

Contracts(contractid, supplierid, projectid,
deptid, partid, qty, value)

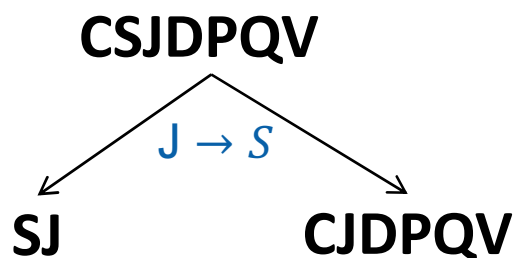
- 我们用一个字母来简记各列CSJDPQV
- 函数依赖关系 $F = \{C \rightarrow SJDPQV, JP \rightarrow C, DS \rightarrow P, J \rightarrow S\}$
- 分解为BCNF?



举例

Contracts(contractid, supplierid, projectid,
deptid, partid, qty, value)

- 我们用一个字母来简记各列CSJDPQV
- 函数依赖关系 $F = \{C \rightarrow SJDPQV, JP \rightarrow C, DS \rightarrow P, J \rightarrow S\}$
- 另一种分解



Outline

- 逻辑设计：ER图到关系模型
- 模式细化：范式
 - 数据冗余的问题
 - 范式介绍
 - 函数依赖与2NF,3NF,BCNF
 - 分解为BCNF
 - 多值依赖和连接依赖
- 物理设计和外部模式设计：视图，索引

多值依赖 (MVD)

Course	Teacher	Book
Physics101	Green	Mechanics
Physics101	Green	Optics
Physics101	Brown	Mechanics
Physics101	Brown	Optics
Math301	Green	Set Theory
Math301	Green	Vectors
Math301	Green	Geometry

- 上述CTB是满足BCNF

- 为什么？

- 但是，存在着冗余

- 明显对一门课而言，教材和老师是独立的

分解可以消除这种冗余

Course	Teacher	Book
Physics101	Green	Mechanics
Physics101	Green	Optics
Physics101	Brown	Mechanics
Physics101	Brown	Optics
Math301	Green	Set Theory
Math301	Green	Vectors
Math301	Green	Geometry

Course	Teacher
Physics101	Green
Physics101	Brown
Math301	Green

Course	Book
Physics101	Mechanics
Physics101	Optics
Math301	Set Theory
Math301	Vectors
Math301	Geometry

多值依赖

- $X \twoheadrightarrow Y$

- $Z = R - XY$, 对任意具体的 $X=x$

- 使得 $\pi_{Y,Z}(\sigma_{X=x}(R)) = \pi_Y(\sigma_{X=x}(R)) \times \pi_Z(\sigma_{X=x}(R))$

- 注意：笛卡尔积！

X	Y	Z
a	b1	c1
a	b2	c2
a	b1	c2
a	b2	c1

多值依赖

- $X \twoheadrightarrow Y$ 的正式定义

- $Z = R - XY$

- $\forall R$ 的实例 $r, \forall t1, t2 \in r, t1[X] = t2[X]$, 那么

- $\exists t3 \in r$, 使得 $t1[XY] = t3[XY]$ 且 $t2[Z] = t3[Z]$

- 这个定义与前一页，关于每个X值上，Y与Z投影笛卡尔积的说明是一致的

多值依赖

- $X \twoheadrightarrow Y$ 的正式定义

- $Z = R - XY$

- $\forall R$ 的实例 $r, \forall t1, t2 \in r, t1[X] = t2[X]$, 那么
 $\exists t3 \in r$, 使得 $t1[XY] = t3[XY]$ 且 $t2[Z] = t3[Z]$

	X	Y	Z
t1	t1[X]	t1[Y]	
t2	t1[X]		t2[Z]
t3	t1[X]	t1[Y]	t2[Z]

多值依赖

- $X \twoheadrightarrow Y$ 的正式定义

- $Z = R - XY$

- $\forall R$ 的实例 $r, \forall t1, t2 \in r, t1[X] = t2[X]$, 那么
 $\exists t3 \in r$, 使得 $t1[XY] = t3[XY]$ 且 $t2[Z] = t3[Z]$

- $t1$ 和 $t2$ 是对称的, 交换它们就有
 $\exists t4 \in r$, 使得 $t2[XY] = t4[XY]$ 且 $t1[Z] = t4[Z]$

	X	Y	Z
t1	t1[X]	t1[Y]	t1[Z]
t2	t1[X]	t2[Y]	t2[Z]
t3	t1[X]	t1[Y]	t2[Z]
t4	t1[X]	t2[Y]	t1[Z]

Y和Z的值是
独立的, 有
所有的组合

函数依赖FD是多值依赖MVD

- FD是一种特殊的MVD
- $X \rightarrow Y$ 是 $X \twoheadrightarrow Y$
- 为什么？
 - 当X取一个值时，根据 $X \rightarrow Y$ ，Y只有唯一的值！
 - 所以， $\pi_Y(\sigma_{X=x}(R))$ 只有一个记录
 - $\pi_{Y,Z}(\sigma_{X=x}(R)) = \pi_Y(\sigma_{X=x}(R)) \times \pi_Z(\sigma_{X=x}(R))$

平凡多值依赖

- 情况1: $Y \subseteq X$

- 这时, $X \rightarrow Y$ 函数依赖成立, 当然多值依赖也成立

- 情况2: $XY = R$

- 这时, $Z = R - XY = \emptyset$

4NF（第四范式）

- R满足4NF，如果对每一个R上的 $X \twoheadrightarrow Y$ ，下面条件有一个成立
 - $Y \subseteq X$ 或 $XY = R$ ，或
 - X是一个超键
- 注意：如果R满足4NF，那么它一定满足BCNF
 - 函数依赖也是多值依赖
 - 函数依赖 $X \rightarrow Y$ 必须满足上述条件
 - BCNF包括了“ $Y \subseteq X$ ”和“X是一个超键”两种情况
 - 而 $XY = R$ ，所以 $X \rightarrow XY$ ， $X \rightarrow R$ ，X是超键
 - 所以一定是BCNF

多值依赖的充分条件

- 由Date和Fagin提出
- 如果R是BCNF，而且存在一个单属性的候选键，那么R是4NF

于是

- 首先得到BCNF
- 包含单属性键的关系已经是4NF
- 对于剩余的关系，考虑其是否包含多值依赖
- 对于多值依赖 $X \twoheadrightarrow Y$ ，分解为XY和R-Y两部分

连接依赖 (JD, Join Dependency)

- 连接依赖 $\bowtie\{R_1, R_2, \dots, R_n\}$
 - 如果 R_1, R_2, \dots, R_n 是 R_d 的无损连接分解
- 连接依赖 JD 是多值依赖 MVD 的进一步推广
 - 一个 MVD 也是一个 JD
 - $X \twoheadrightarrow Y$, 是 $\bowtie\{XY, R-Y\}$, join key 是 X

5NF（第五范式）

- 如果对R上的每个连接依赖 $\bowtie\{R_1, R_2, \dots, R_n\}$ ，下述条件有一个成立：
 - 其中某个 $R_i = R$
 - 这个连接依赖为一组函数依赖所蕴含，这些函数依赖的左侧都是候选键
- 充分条件
 - 如果R是3NF，而且R的每个候选键都是一个属性，那么R也是5NF。

Outline

- 逻辑设计：ER图到关系模型
- 模式细化：范式
- 物理设计和外部模式设计：视图，索引

物理数据库设计

- 目的

- 了解数据库的负载
 - 提高数据库的性能

- 建立哪些索引 (index) ?

- 调整逻辑模式

- 不同的方法产生不同规范化结果
 - 垂直划分
 - 非规范化?
 - 用外部模式隐藏这些改变

- 调整查询和事务

我们在这里先介绍
建立索引等的语法,
具体的物理数据库
设计将在讲解了数
据库系统内部原理
后再介绍

创建索引 (Index)

- DDL的一部分

- Create, drop

- 建立索引

create index *索引名* on *表名* (*列名*, ..., *列名*);

例如：在Student表中name列上建立索引

create index *StudentName* on *Student* (*Name*);

删除索引 (Index)

- 删除索引

Oracle, IBM DB2, Postgresql: drop index 索引名;

MS SQL Server: drop index 表名.索引名;

MySQL: alter table 表名 drop index 索引名;

视图 (View)

- 视图是一个关系表

- 但是，它的内容不是直接存储的
- 而是在需要的时候根据视图的定义计算得到的

- 假设我们经常要查询计算机系学生姓名和所选课程

create view *CSSStCo* as

```
select S.ID, S.name as Sname, C.name as Cname  
from   Student S, Course C, TakeCourse TC  
where  TC.StudentID=S.ID and TC.CourseID=C.ID  
       and S.Major='计算机';
```

这个视图包含了三个列：学号，姓名，课程名

创建和删除视图

创建视图

create view 视图名 as
select 语句;

视图的列和列的类型由Select子句决定

删除视图

drop view 视图名;

创建视图时，对属性重命名

create view *视图名* (*列名*, ..., *列名*) as
select 语句;

例如：

```
create view CSSStCo(StudentID, Sname, Cname) as  
select S.ID, S.name, C.name  
from Student S, Course C, TakeCourse TC  
where TC.StudentID=S.ID and TC.CourseID=C.ID  
and S.Major=‘计算机’;
```

视图的使用

- 与基本关系表Table相同

- 例如：查询张飞选的课

```
select Cname  
from StudentCourse  
where Sname = '张飞';
```

注意：实际上，数据库系统把这个查询转化为嵌套查询：

```
select Cname  
from (select S.ID, S.name as Sname, C.name as Cname  
      from Student S, Course C, TakeCourse TC  
      where TC.StudentID=S.ID and TC.CourseID=C.ID  
        and S.Major='计算机')  
where Sname = '张飞';
```


视图的更新 (View Update)

- 大部分情况下，视图不能更新
- 例如：
 - *CSStCo(StudentID, Sname, Cname)*
 - 如果插入一条记录，很难直接对应到某个基本表中
 - 如：StudentID如果不存在会怎么样？
- 在某些条件下，视图非常简单时，可以更新视图

可更新视图(Updatable View)

- 条件

- From必须是单个关系（或单个可更新视图）R
- Where如果有子查询，那么子查询不能包含R
- 没有distinct和aggregation
- Select必须包含足够多的列，确保插入时可以用NULL或默认值填充剩余的列

- 这样，视图的一个插入操作可以直接对应到基本表

- 例如

```
create view CSSt(StudentID, Sname) as  
  select S.ID, S.name  
  from   Student S  
  where S.Major='计算机';
```

更新可更新视图

```
create table Student (  
    ID integer, Name varchar(20),  
    Major varchar(20), Year year, GPA float  
);
```

```
insert into CSSt(StudentID, Sname) values (12345, '周瑜');
```

被转化为

```
insert into Student(ID, Name) values (12345, '周瑜');
```

那么，就在基本表Student中插入下述记录：

(12345, '周瑜', NULL, NULL, NULL)

实际上有些奇怪，插入的记录不满足Major='计算机'的条件

更新可更新视图

可以增加Major列来避免这种问题

```
create view CSSt(StudentID, Sname, Major) as  
  select ID, Name, Major  
  from   Student S  
  where  S.Major='计算机';
```

insert into CSSt(StudentID, Sname, Major) values (12345, '周瑜', '计算机');

在基本表Student中插入下述记录:

(12345, '周瑜', '计算机', NULL, NULL)

更新可更新视图

同理，update和delete也都是转化为基本表的操作进行的

例如，为了保证delete的正确性，需要加上view中where条件

```
create view CSSt(StudentID, Sname, Major) as  
  select ID, Name, Major  
  from   Student S  
  where  S.Major=‘计算机’;
```

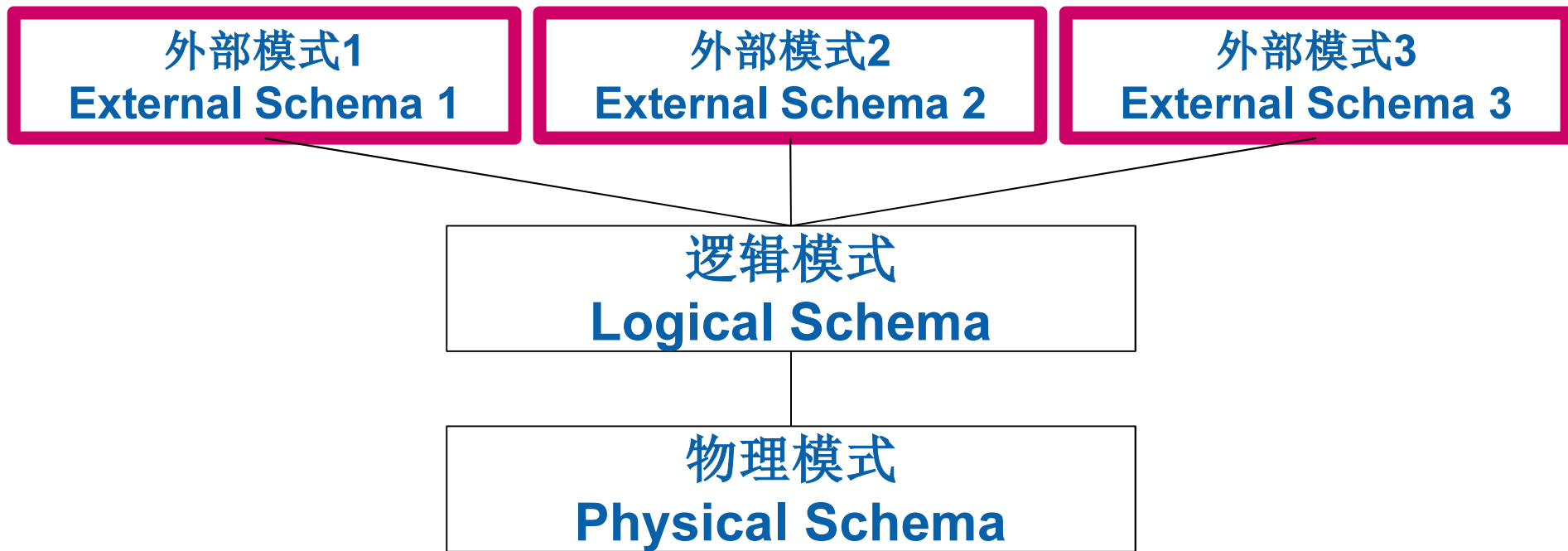
```
delete from CSSt  
where Sname=‘周瑜’ and Major=‘计算机’;
```

如果没有Major=‘计算机’，就可能删除其它系同名的学生！

更新可更新视图

- 比较Tricky
- 如果实在需要，那么就一定仔细检查，保证正确

数据独立性和安全



- 出于安全和数据独立性的需要，对于不同的用户（群体），通常只使其看到逻辑模式的一个部分
 - 例如：学生可以看到自己的选课信息，但不能看到其他同学的选课信息
- 在逻辑模型上使用View来得到

访问控制 Grant

- SQL通过Grant和Revoke命令来进行访问控制
- Grant给特定用户授予Table或View的访问权限

grant *访问权限* on *对象* to *用户* [with grant option]

□ *对象*: 基本表table或视图view

□ *访问权限*

- Select: 读所有列
- Insert: 插入; Insert(列名): 插入, 只允许插入指定的列
- Update: 修改; Update(列名): 修改, 只允许修改指定的列
- Delete: 删除
- References(列名): 可以在其它定义中, 用外键引用这个列
- All: 所有权限

□ *With grant option*: 这个用户可以把得到的权限授权给别人

基本表的访问权限

- 用户U创建了一个基本表R
 - 那么，U就拥有表R的所有访问权限和grant权限
- U可以把表R的访问权限授予他人
 - 允许查询
 - 允许增、删、改
 - 允许再授权
- 但是，数据定义语句的权利(create, alter, drop)等不能授予或收回

视图的访问权限

- 建立视图时，必须有涉及的所有表的select权限
- 然后，就可以把视图的select权限授予用户

例如：

create view *CSStCo* as

```
select S.ID, S.name as Sname, C.name as Cname  
from   Student S, Course C, TakeCourse TC  
where  TC.StudentID=S.ID and TC.CourseID=C.ID  
       and S.Major=‘计算机’;
```

- 把这个视图授予计算机系学生用户 *CS_Student*
grant select on *CSStCo* to *CS_Student*;
□ 但是，*CS_Student*不能直接访问*Student*, *Course*,
TakeCourse。于是，有效地保护了这些基本表。

去除访问权限：Revoke

revoke [grant option for] 访问权限 on 对象 from 用户
[cascade|restrict]

□ 对象：基本表table或视图view

□ 访问权限

- Select: 读所有列
- Insert: 插入; Insert(列名): 插入, 只允许插入指定的列
- Update: 修改; Update(列名): 修改, 只允许修改指定的列
- Delete: 删除
- References(列名): 可以在其它定义中, 用外键引用这个列
- All: 所有权限

□ cascade: 对应于with grant option, 把这个用户授权给别人的衍生权限也收回

□ restrict: 收回指定用户的权限, 若有衍生权限则拒绝执行

□ grant option for: 只收回进一步授权的权限

举例

- 假设Sailors, Boats, 和Reserves表
- Joe是Sailors表的创建所有人
- Joe把Sailors的读权限和对读的授权权限授予Art
grant select on *Sailors* to *Art* with grant option;
- Art把Sailors的读权限和对读的授权权限授予Bob
grant select on *Sailors* to *Bob* with grant option;
- 一段时间之后, Joe收回Art的读权限和Art进一步的授权给Bob的衍生读权限
revoke select on *Sailors* from *Art* cascade;
□ 这里Bob的读权限也被收回了

举例

- (Joe) grant select on *Sailors* to *Art* with grant option;
(Joe) grant select on *Sailors* to *Bob* with grant option;
(Art) grant select on *Sailors* to *Bob* with grant option;
(Joe) revoke select on *Sailors* from *Art* cascade;
- 这里Bob的读权限由Joe和Art授权，虽然Joe收回了Art及其衍生读权限，但是Bob从Joe直接的授权仍然有效。

举例

- (Joe) grant select on *Sailors* to *Art* with grant option;
(Joe) revoke grant option for select on *Sailors* from *Art* cascade;
- 收回了Art对Sailors的读进一步授权的权利
- 但是，Art的读权限本身仍然保留
- 但是，更加复杂的grant和revoke需要借助授权图来判断权限

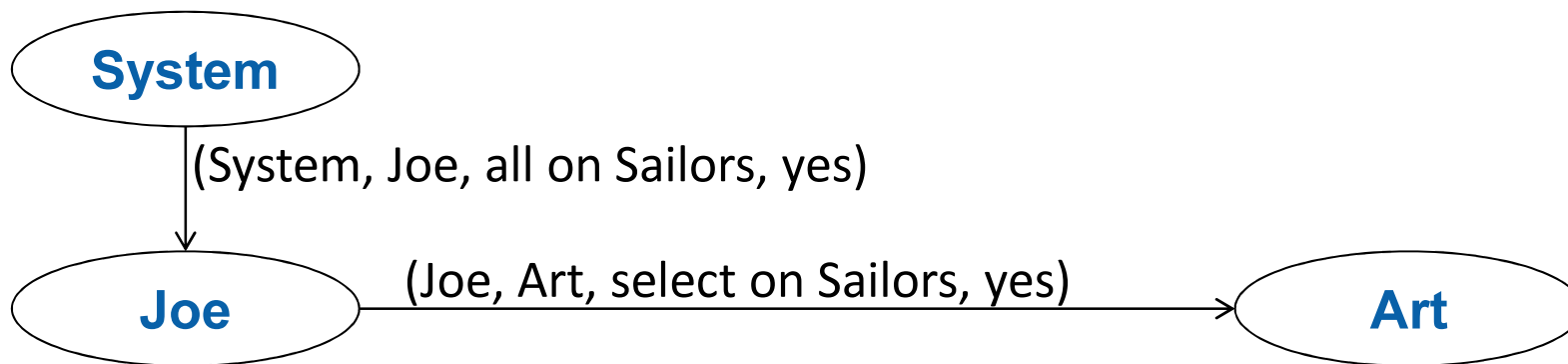
授权图

- 每执行一次grant，数据库系统记录
 - 授权者：执行grant的用户，或者是system
 - 被授权者：得到授权的用户
 - 权限
 - 是否具有with grant option: yes/no
- 于是，可以把授权用图来表示



授权图举例

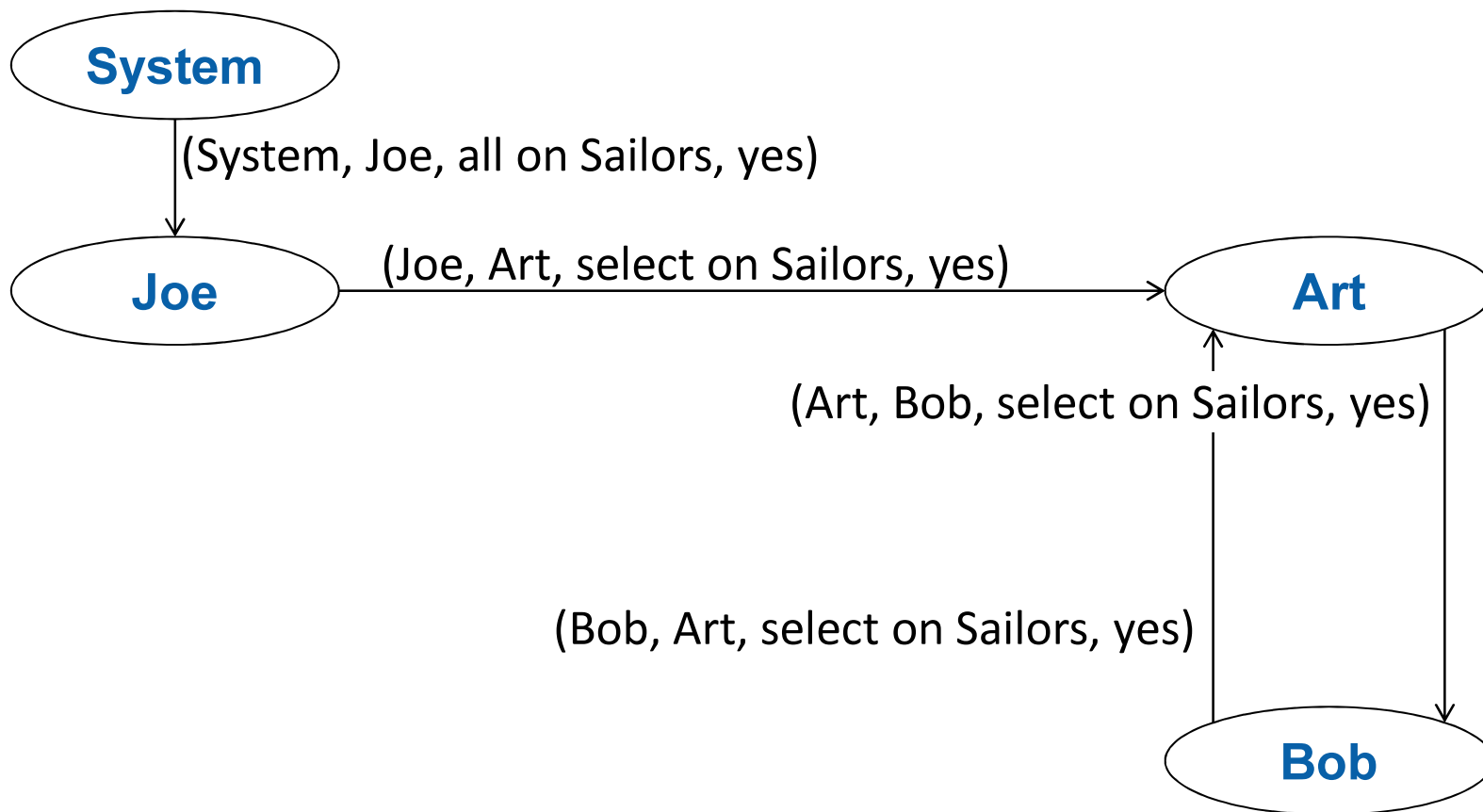
(授权者, 被授权者, 权限, yes/no)



- Joe创建表 *Sailors*
- (Joe) grant select on *Sailors* to *Art* with grant option;

授权图举例

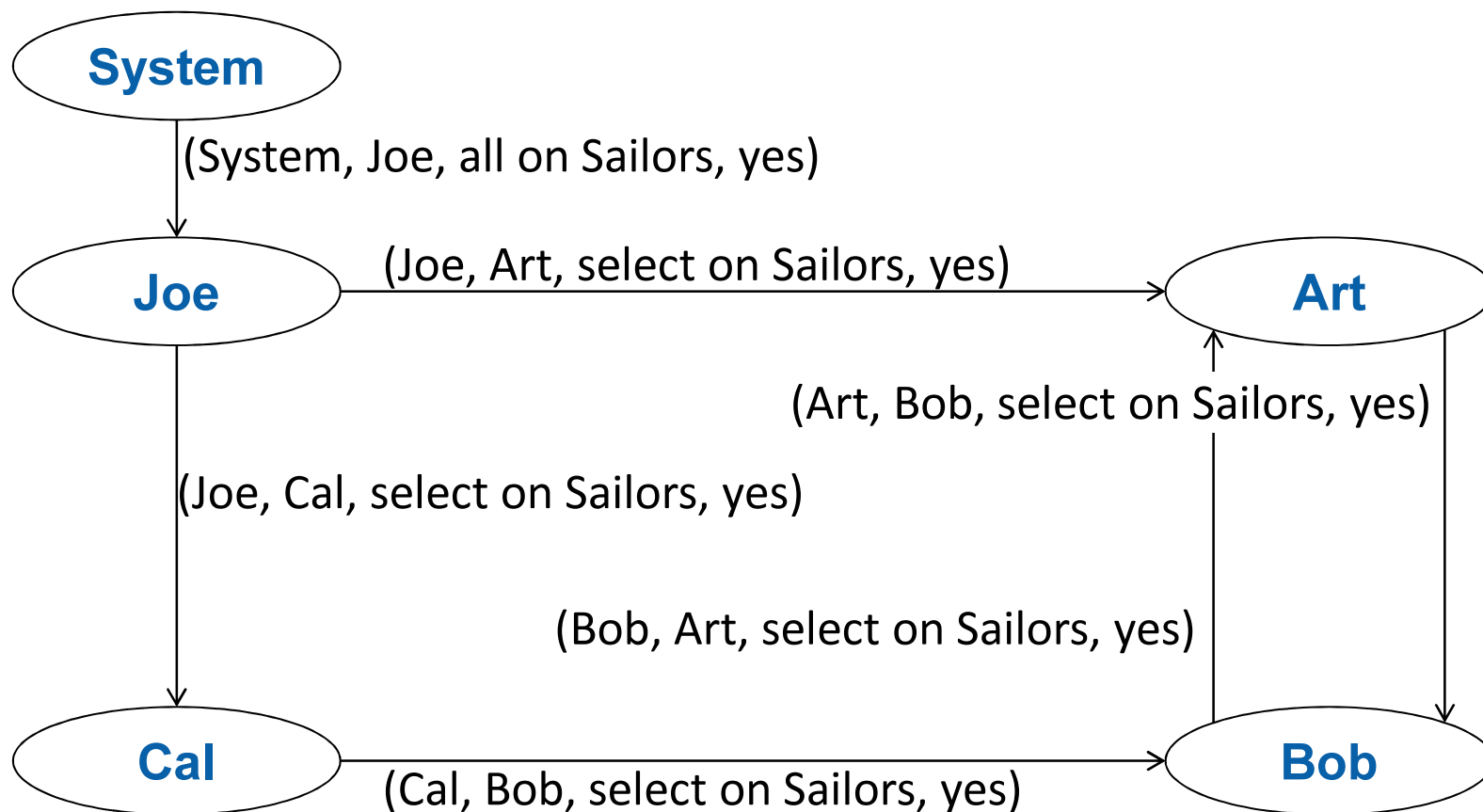
(授权者, 被授权者, 权限, yes/no)



- (Art) grant select on *Sailors* to *Bob* with grant option;
- (Bob) grant select on *Sailors* to *Art* with grant option;

授权图举例

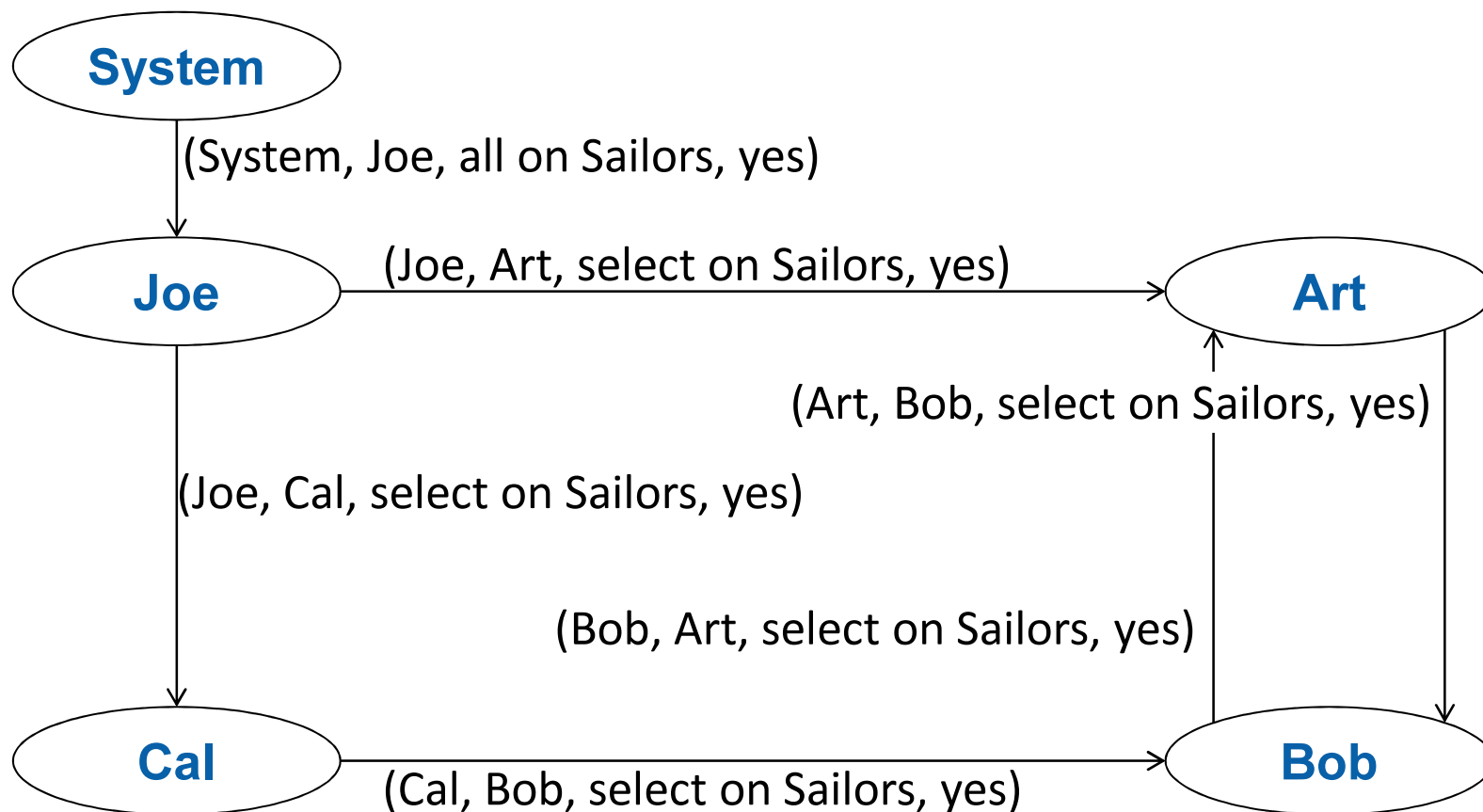
(授权者, 被授权者, 权限, yes/no)



- (Joe) grant select on *Sailors* to *Cal* with grant option;
- (Cal) grant select on *Sailors* to *Bob* with grant option;

授权图举例

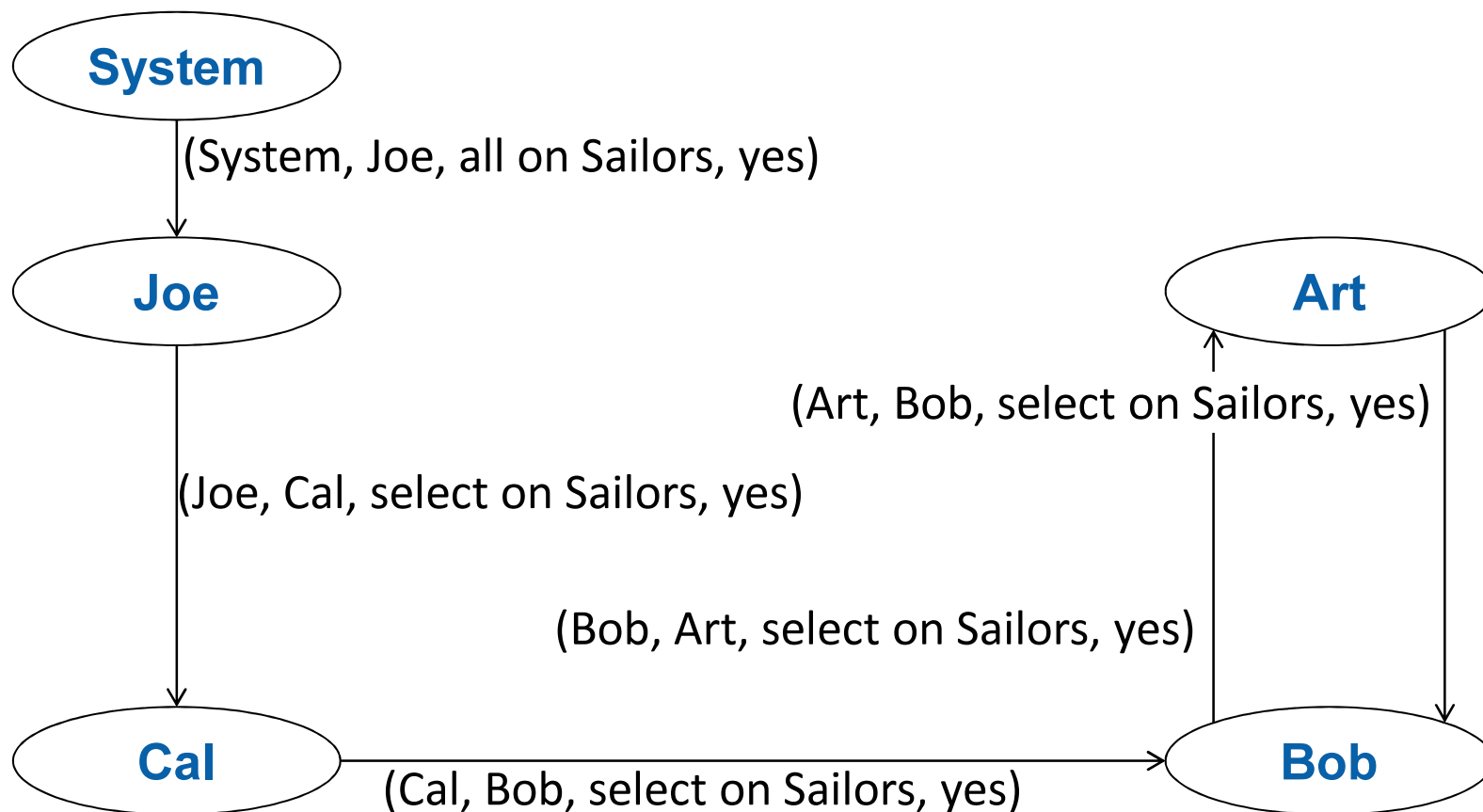
(授权者, 被授权者, 权限, yes/no)



- (Joe) revoke select on *Sailors* from *Art* cascade;

授权图举例

(授权者, 被授权者, 权限, yes/no)

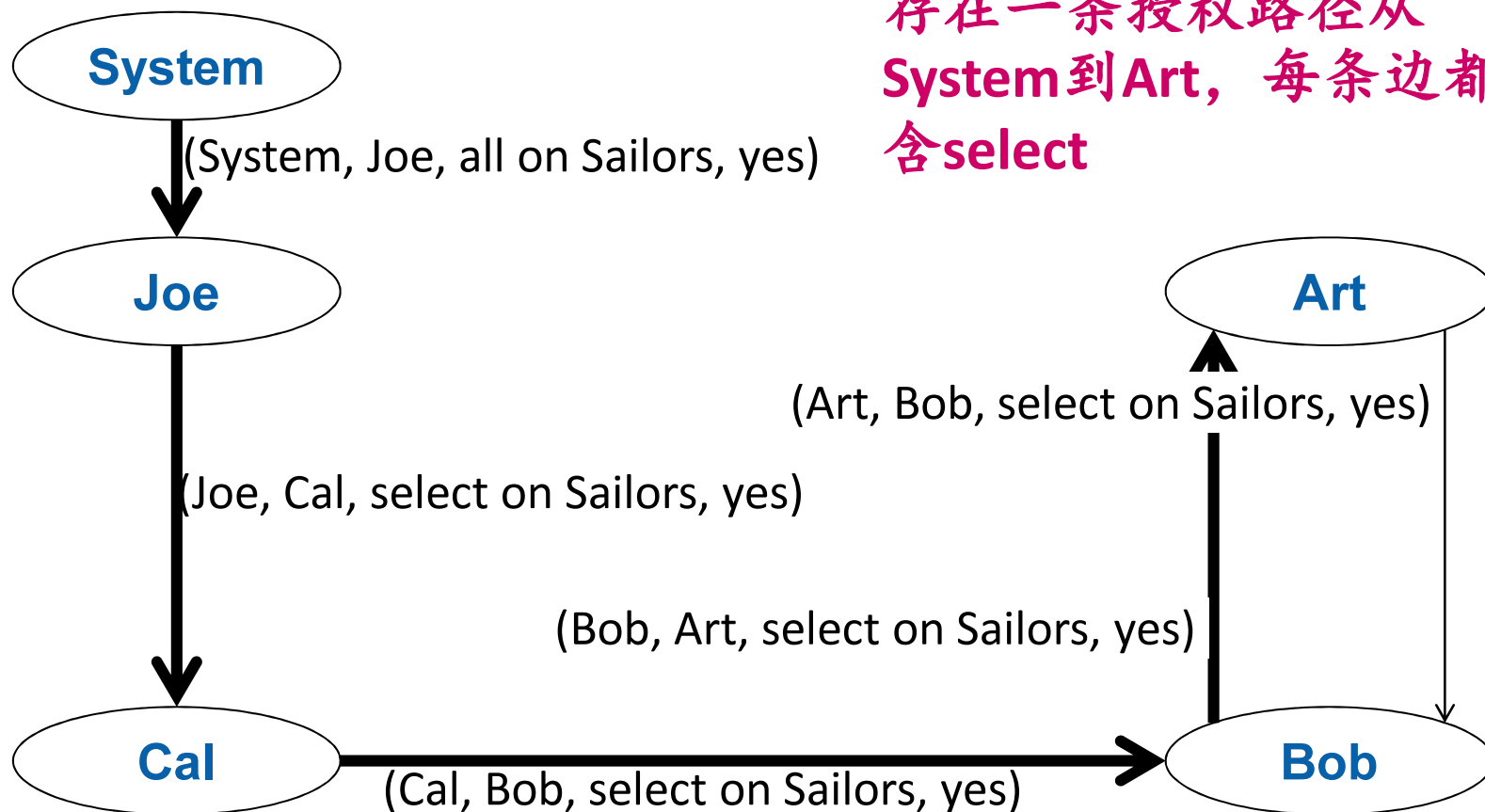


- (Joe) revoke select on *Sailors* from *Art* cascade;
- 删除了一条边, 问: Art是否还有权读Sailors?

授权图举例

(授权者, 被授权者, 权限, yes/no)

存在一条授权路径从
System到Art, 每条边都包
含select

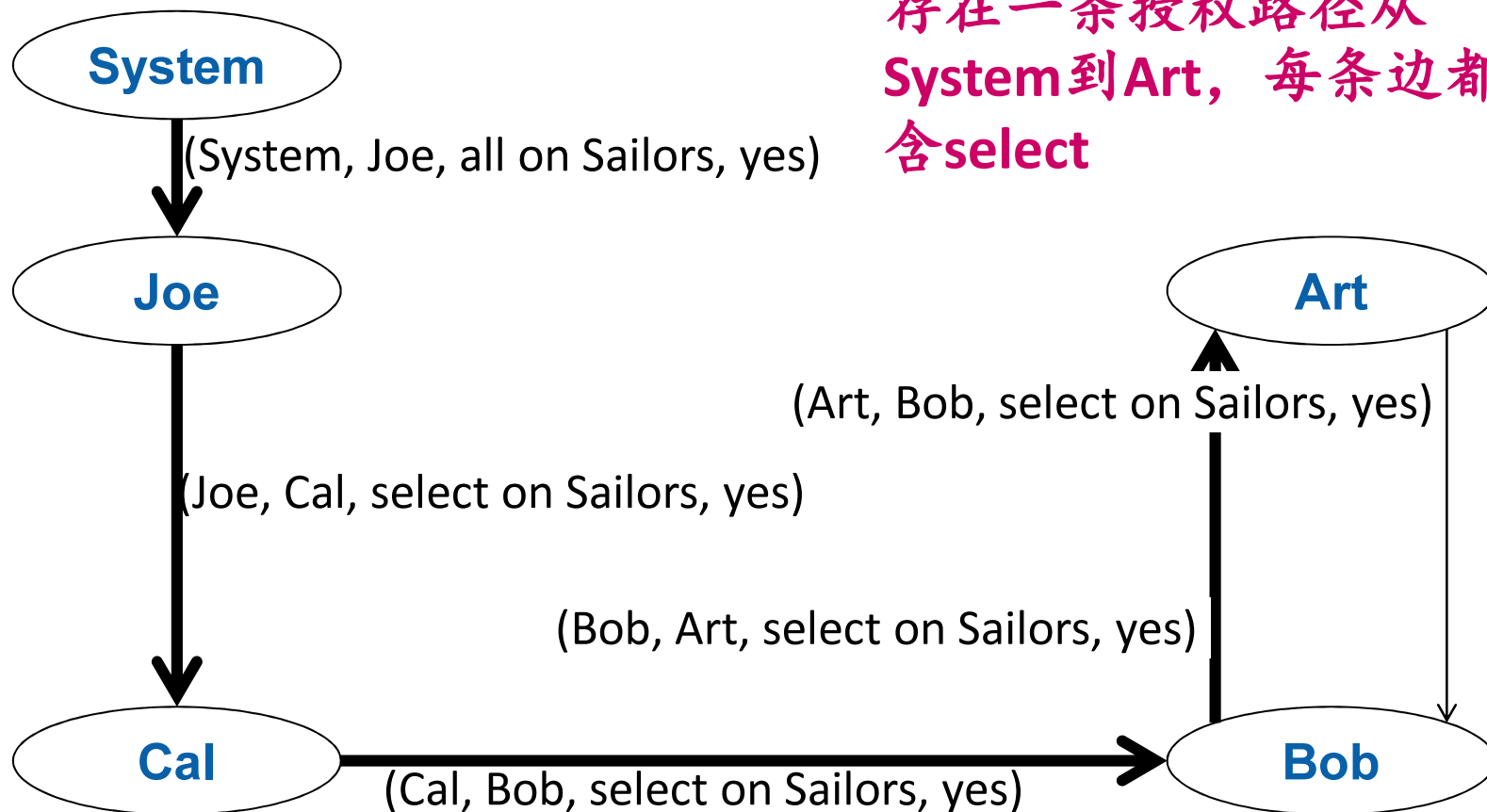


- (Joe) revoke select on *Sailors* from *Art* cascade;
- 删除了一条边, 问: Art是否还有权读Sailors? 是的!

授权图举例

(授权者, 被授权者, 权限, yes/no)

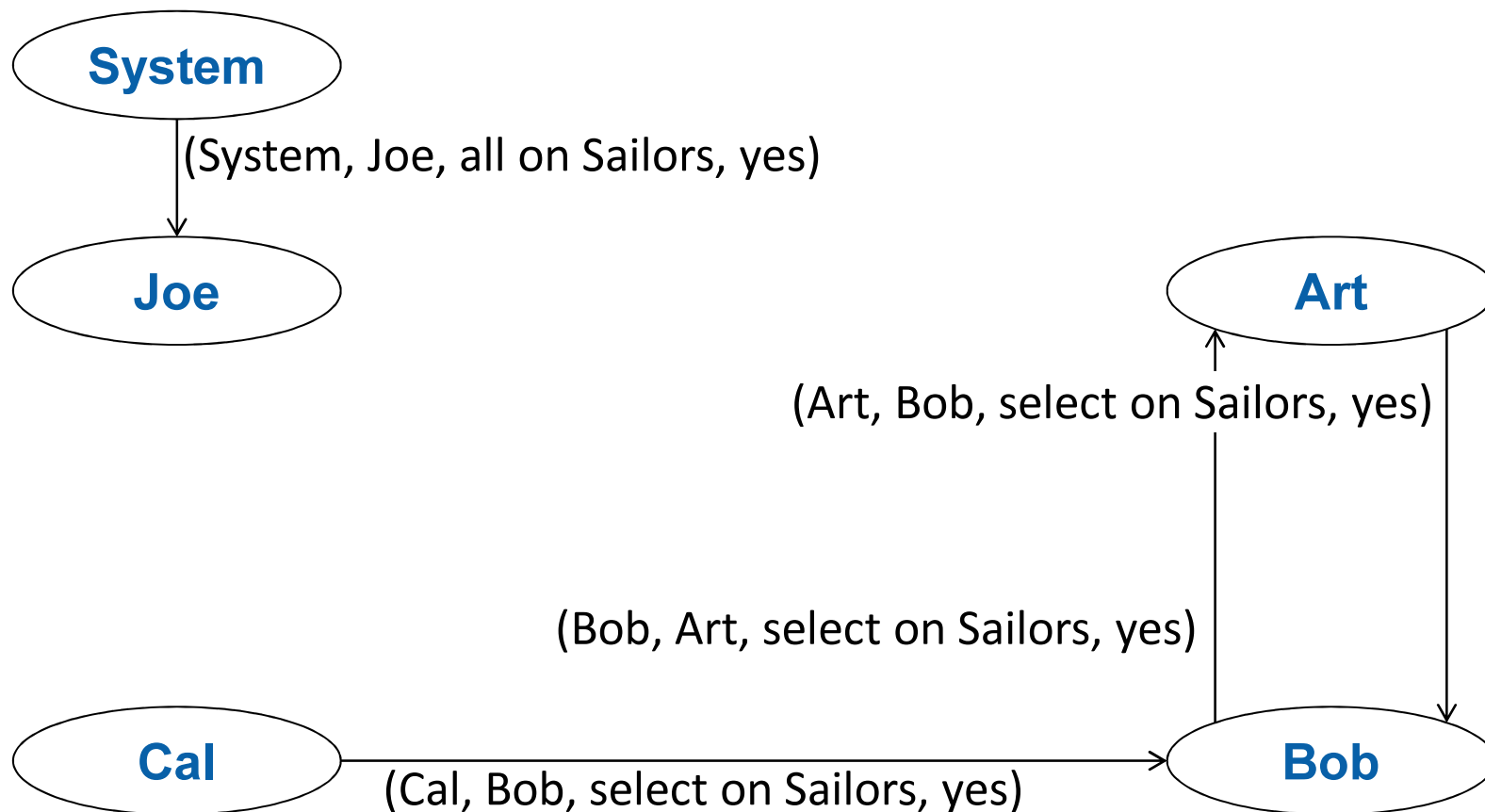
存在一条授权路径从
System到Art, 每条边都包
含select



注意: 虽然Bob向Art的授权取得自Art, 但是出于实现方便和现实需要, 只考虑当前的授权图, 而不考虑边生成的顺序

授权图举例

(授权者, 被授权者, 权限, yes/no)



- (Joe) revoke select on *Sailors* from *Cal* cascade;
- 又删除了一条边, 问: Art是否还有权读Sailors? 否

基本表的授权和视图

- Joe创建了Sailors表
- Joe把Sailors表的读权限授权给Michael, 如何写?

- Michael在Sailors表上建立了一个视图

```
create view YoungSailors as  
  select *  
  from Sailors  
  where age < 20;
```

- 然后, Michael把YoungSailors的读权限授予Jack, 如何写?
- 这时, Joe收回了Michael对于Sailors表的读权限, 如何写?
- 会发生什么?
 - YoungSailors这个视图被删除! 要求视图的所有人对Sailors必须可读
 - Jack当然也就不能读YoungSailors

小结

- 逻辑设计：ER图到关系模型
- 模式细化：范式
 - 数据冗余的问题
 - 范式介绍
 - 函数依赖与2NF,3NF,BCNF
 - 分解为BCNF
 - 多值依赖和连接依赖
- 物理设计和外部模式设计：视图，索引