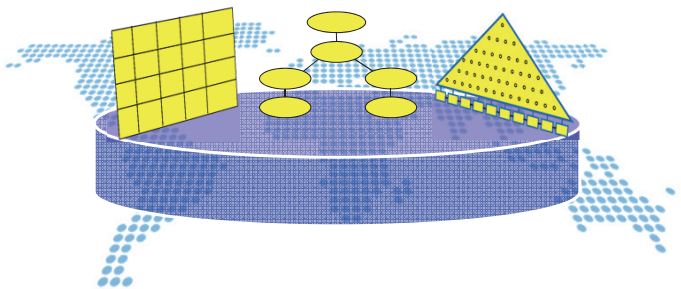
数据库系统
关系模型与SQL（4）

陈世敏
（中科院计算所）



前面内容

- ER模型，关系模型，ER➔关系模型
- SQL 初步
  - 记录的增删改
- 简单的查询+关系代数
  - 集合操作：并、交、差
  - 选择行或列：选择、投影
  - 两个关系元组之间的操作：笛卡尔积、连接
  - 其它：重命名、除
- 丰富的SQL Select功能+扩展关系代数
  - 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）
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）

Outline

- 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：索引的增删
- 外部模式设计：视图

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
- 冗余带来的问题
  - 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rating=5与wage=30之间的关系存了2次
  - 实际上耗费了更多的存储空间

数据冗余的问题：举例

Employee

ID	Name	Rating	Wage
1	Alan	5	30→40
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
  - 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
  - 例如：下面的update语句破坏了隐含条件，导致了数据异常

```
update Employee
set Wage= 40
where ID=1;
```


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
5	Eva	5	40

- 假设隐含的条件：相同的rating，工资wage也相同
- 问题
  - 插入异常：如果不知道rating对应的wage，就无法正确插入
  - 例如：下面的insert语句破坏了隐含条件，导致了数据异常

```
insert into Employee values (5, 'Eva', 5, 40);
```

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
  - 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！
  - 例如：

```
delete from Employee
where Rating=5;
```


解决方法：模式分解（我们将学习）

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

下面的内容

- 冗余类型和范式
  - 每种范式是什么？
  - 怎么判断一个关系模式是否满足某种范式？
  - 如何通过模式分解使一个关系表满足希望的范式？

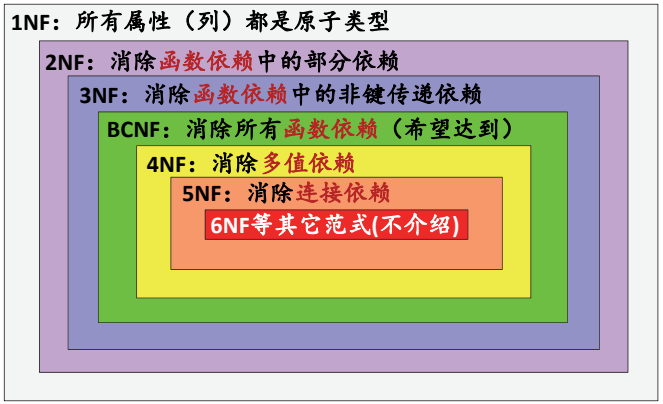
冗余的类型

- 函数依赖（Functional Dependency, FD）
- 多值依赖（Multivalued Dependency, MVD）
  - FD是一种特殊的MVD
- 连接依赖（Join Dependency, JD）
  - MVD是一种特殊的JD
- 我们会一一介绍

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）



讲解将用传统关系代数

- 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
  - 主要考虑基础的关系表的设计
  - 通常基础表定义了主键，所以没有多个相同记录
  - 不关注运算过程中的中间结果
  - 所以实际上不需要扩展关系代数

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
    - 什么是函数依赖？
    - 怎样简化函数依赖？
    - 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）

- 前提条件
  - $R(U)$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→Y
  - 任意两个元组在X上取值相同=>它们在Y上取值也相同
  - 形式化表述：对于任意的实例r，$\forall t1,t2\in r$，若$t1[X]=t2[X]$，则$t1[Y]=t2[Y]$
- 理解：相同的X取值➔相同的Y取值
- 称为：X函数确定Y，Y函数依赖于X

候选键：一种特殊的函数依赖

- 可以通过函数依赖定义候选键
  - 已知$R(U)$是属性集$U=\{A_1,\dots,A_m\}$上的一个关系模式
- 定义：$X\subseteq U$是候选键，如果满足下述条件
  - $X\rightarrow U$　　（X函数确定U）
  - $\forall Y\subset X, Y\not\rightarrow U$　　（X是最小的）
- Super key（超键）：满足上述第一个条件
  - 候选键是一种超键
  - 候选键同时满足第二个条件：它的真子集不是超键

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→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→C

- 候选键是什么？
  - 考虑包含1列的情形：A✗，B✗，C✗，D✗
  - 考虑包含2列的情形：AB✗，AC✗，AD✗，BC✗，BD✗，CD✓
  - 考虑包含3列的情形：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→C? ✗
- A→A? ✓
- ABD→A? ✓

可以简化一下吗？

- 函数依赖的分解和合并
- 目标：属性集$X \rightarrow$ 单个属性A

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：

$\because X \rightarrow YZ$

$\because YZ \rightarrow Y$（自反律）
$\therefore X \rightarrow Y$（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：

$\because X \rightarrow Z$

$\therefore XX \rightarrow XZ$（增补律）
$\therefore X \rightarrow XZ$（$XX$就是$X$）（1）

$\because X \rightarrow Y$
$\therefore XZ \rightarrow YZ$（增补律）（2）

由(1)(2)，$X \rightarrow YZ$（传递律）
证明完毕。

属性集X → 单个属性A

- $X \to YZ \iff X \to Y$而且$X \to Z$
  - 合并和分解是互为可逆的
  - 只要找出对于所有单一属性的函数依赖关系
  - 就可以得到所有的函数依赖关系
- 只需考虑“属性集X → 单个属性A”形式的函数依赖

☞已知一组函数依赖，怎样找出所有的函数依赖？

函数依赖的闭包

- 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
  - 算法
    - 反复使用Armstrong规则
    - 直到不增加新的函数依赖为止

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
- 假设已知下述函数依赖关系
  - 其中C是主键，所以C→SJDPQV
  - 一个项目买一种零件只用一个合同：JP →C
  - 一个部门从一个供货商至多购买一种零件：DS → P
- 已知F={C→SJDPQV, JP →C, DS → P}

☞求函数依赖集的闭包F^+

重新想一下：自反律？

- 自反律
  - 如果$Y \subseteq X$，那么$X \to Y$
  - 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：费力而没有很多意义

重新想一下：增补律？

- 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？
    - 从l个属性中组成Z有多少中情况？
    - 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？

- 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，求闭包重点是使用传递律
- 什么情况下使用增补律和自反律呢？
  - 为了创造条件可以使用传递律
  - 使右侧为全部属性，即左侧是候选键

传递律产生的规律？

- 已知F={C→SJDPQV, JP →C, DS → P}

求函数依赖集的闭包F^+

- C→SJDPQV，所以C→CSJDPQV（增补律）
- JP →C，所以JP → CSJDPQV（传递律）
- DS → P，所以JDS →JP（增补律）
  所以JDS → 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→SJDPQV, JP →C, DS → P}
- C→CSJDPQV, JP → CSJDPQV, JDS → CSJDPQV
- 分解可以得到
  - C→S , C→ J , C→ D , C→ P , C→ Q , C→ V
  - JP → C , JP → S , JP → J , JP → D , JP → P , JP → Q , JP → V
  - JDS → C , JDS → S , JDS → J , JDS → D , JDS → P , JDS → Q , JDS → 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

- 函数依赖F的闭包：F^+
  - 根据F可以推导出的所有函数依赖
- 属性集X的闭包：X^+
  - 相对于一个函数依赖集F
  - 计算属性集X函数确定的所有属性
  - $X \rightarrow X^+$

举例

- 已知F={C→SJDPQV, JP →C, DS → P}
- 求JDS属性的闭包？
  - 在F下，由JDS函数确定的所有属性？

属性闭包算法

```
closure = X;
do {
    changed = false;
    if (存在(Y → Z ∈ F) and
        (Y ⊆ closure) and  !(Z ⊆ closure))
    then { closure = closure ∪ Z; changed= true;}
} while (changed);
```

最后closure包含X函数确定的所有属性

举例

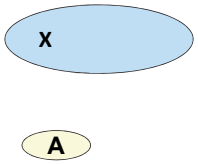
- 已知F={C→SJDPQV, JP →C, DS → P}
- 求JDS属性的闭包？

- 初始化，Closure={J,D,S}
- DS → P且DS ⊆ Closure, Closure={S,J,D,P}
- JP →C且JP ⊆ Closure, Closure={C,S,J,D,P}
- C→SJDPQV且C ⊆ Closure, Closure={C,S,J,D,P,Q,V}

函数依赖

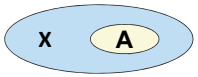
- 定义
- 规律
  - 自反律，增补律，传递律
  - 分解，合并
- 闭包
  - 函数依赖的闭包
  - 属性的闭包
- 函数依赖与2NF,3NF,BCNF的关系

可能有什么样的函数依赖呢？
“属性集X→单个属性A”



分情况讨论X和A

类型1：平凡依赖
“属性集X→单个属性A”



平凡依赖✓

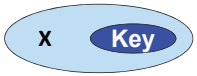
$\{A\} \subseteq X$，有$X \rightarrow A$

这种函数依赖是正常的☺
大量存在的

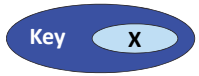
没有数据冗余问题

☞接下来，我们考虑非平凡的函数依赖
从X和A与候选键的关系来分类讨论

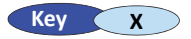
“$X \rightarrow A$”与键的关系有下述几种可能



X是超键



X是候选键的一部分



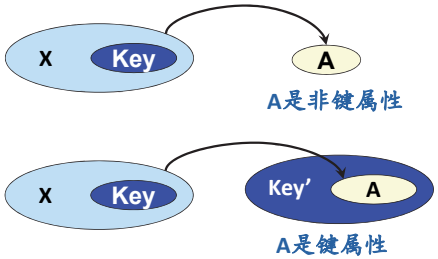
X包含非键的属性，
并且X不是超键

A是非键属性

A是键属性

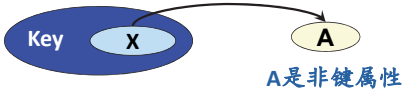
所以，有3x2=6种组合

类型2：X是超键✓



- X是超键（包括X是候选键的情形）
- X当然可以函数确定任何属性
- **正常的☺**，没有数据冗余问题

类型3：部分依赖
X是候选键的一部分，A是非键属性



A是非键属性

- 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
  - 而假设已知sid ->credit_card

类型4：非键传递依赖
X包含非键的属性，X不是超键，A是非键属性

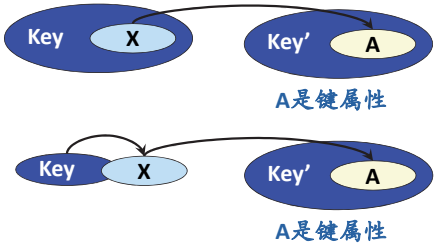
A是非键属性

- 这会导致冗余
- 在前面例子的Employee表中
  - 主键为ID
  - 但是Rating -> Wage

Employee

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

类型5：对于键属性的函数依赖

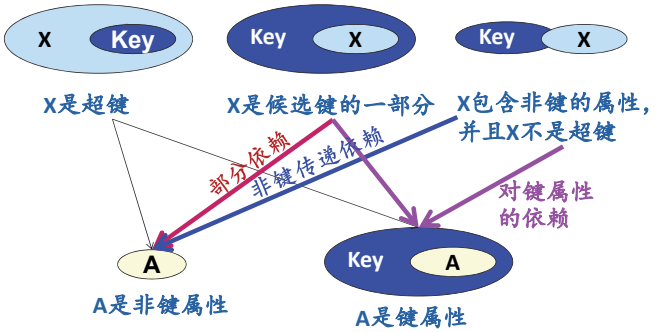


- 这些都可能产生冗余
- 一个例子：
  - ABD和CD是候选键

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

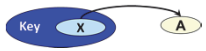
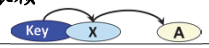
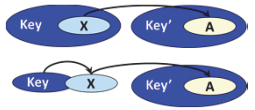
AB→C

“X→A”与键的关系有下述几种可能



所以，有3x2=6种组合

函数依赖类型与消除依赖的范式

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

具体而言

- 什么是满足1NF的模式？
  - 所有属性（列）都是原子类型，这样的模式满足1NF
  - 关系模型的基本要求
- 什么是满足2NF的模式？
  - 满足1NF，并且没有部分依赖，这样的模式满足2NF
- 什么是满足3NF的模式？
  - 满足2NF，并且没有非键传递依赖，这样的模式满足3NF
- 什么是满足BCNF的模式？
  - 满足3NF，并且没有对键属性的函数依赖，这样的模式满足BCNF

回到开始的范式关系图

那么如何判断一个关系模式是否满足1NF, 2NF, 3NF, BCNF?

- 找出所有的函数依赖关系
- 然后看看其中是否包含
  1. 存在部分依赖？
     a) Yes：只满足1NF，完成
     b) No：满足2NF，继续
  2. 存在非键传递依赖？
     a) Yes：完成
     b) No：满足3NF，继续
  3. 存在对于键属性的函数依赖？
     a) Yes：完成
     b) No：满足BCNF，完成

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→SJDPQV, JP →C, DS → P}
- 满足什么范式？

解答

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→SJDPQV, JP →C, DS → P}
- 候选键为：C, JP, JDS；非键属性：Q, V
  1. 存在部分依赖？
     a) No：满足2NF，继续
  2. 存在非键传递依赖？
     a) No：满足3NF，继续
  3. 存在对于键属性的函数依赖？
     a) Yes：完成

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：索引的增删
- 外部模式设计：视图

关系模式的分解

- $R(U)$是属性集$U=\{A_1,\dots,A_m\}$上的一个关系模式
- $X\subseteq U$，$Y\subseteq U$

操作：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
  - 分解是通过投影实现的
  - 投影后要去重

☞ 分解可能引起信息丢失，必须遵循一定的原则

无损分解

- 我们希望
  - 分解是可逆的，保留了所有信息，分解前后是等价的
  - 这种分解被称为无损分解

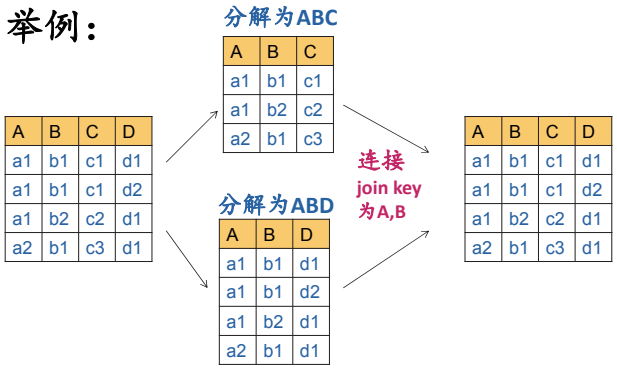
- 对R进行X和Y的分解是一个无损分解，当且仅当
  - 分解后两个表的自然连接可以得到原始表
  - 即$\pi_X(R)\bowtie\pi_Y(R)=R$（传统关系代数）

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：



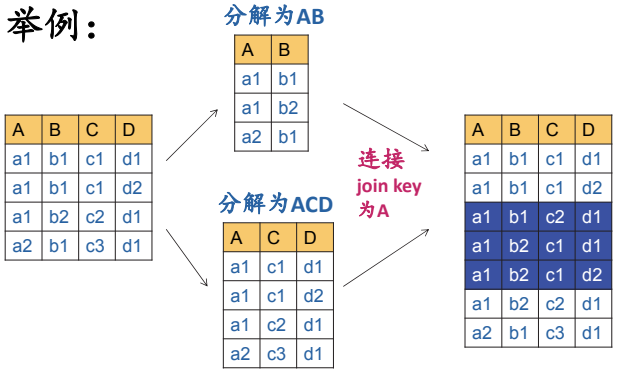
- ABC和ABD的分解是无损的

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：



- AB和ACD的分解不是无损的

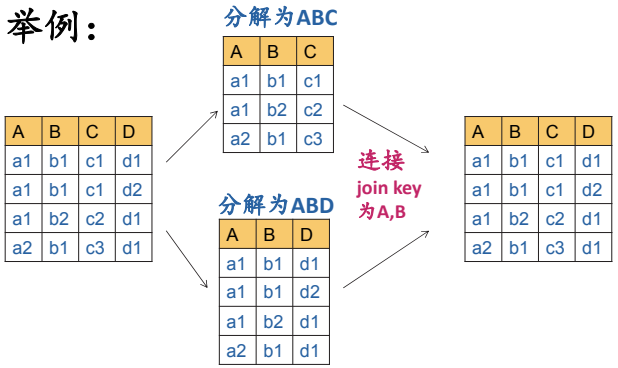
函数依赖~无损分解

- $R(U)$是属性集$U = \{A_1, \ldots, A_m\}$上的一个关系模式
- 可以证明：

  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

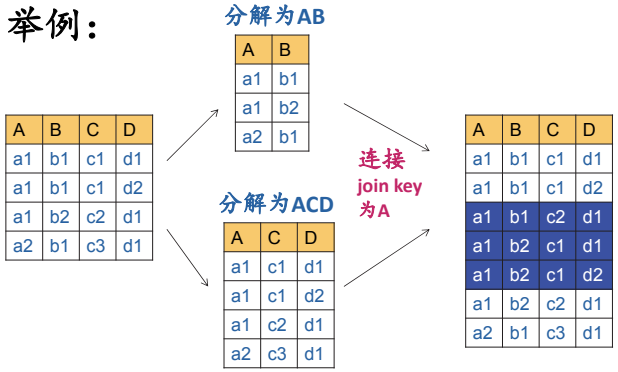
- 理解一下
  - $R1 \cap R2$是$\pi_{R1}(R) \bowtie \pi_{R2}(R)$这个连接的join key部分
  - 如果join key→$R1$成立，那么join key是R1的主键，R2的外键
  - 所以这是主键–外键之间的连接
  - 这种join是无损的
- 这是无损连接的一个充分条件

举例：



- {ABC} ∩ {ABD} = {AB}，AB->ABC
- 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

A	B
a1	b1
a1	b2
a2	b1

A	C	D
a1	c1	d1
a1	c1	d2
a1	c2	d1
a2	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} ∩ {ACD} = {A}，A不是AB的候选键，也不是ACD的候选键
- 所以上述充分条件不成立，可能不是无损分解
- 从例子中可见，AB和ACD的分解确实不是无损的

我们希望设计的关系模式满足BCNF

- BCNF只允许下述$X \to A$
  - **平凡依赖：** $A \in X$
  - **X是超键**

- 如果不满足BCNF

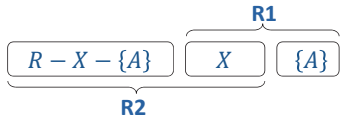
☞通过关系模式的无损分解来得到符合BCNF的模式

如何无损分解为BCNF？

- 如果$X \to A$违反了BCNF
  - $X \subset R$
  - $A \in R$是单独属性
  - $A \notin X$，不是平凡依赖
- 对R进行XA和R - A的分解
- 反复执行直至满足BCNF

证明：XA和R-A的分解是无损的

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



证明：

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

- $R1 \cup R2 = R$：$(X \cup \{A\}) \cup (R - \{A\}) = R$
- $R1 \cap R2 \to R1$：$(X \cup \{A\}) \cap (R - \{A\}) = X$，$X \to X \cup \{A\}$

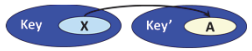
分解为BCNF：算法

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)

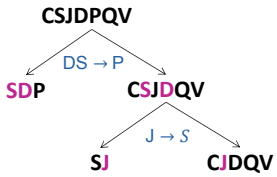
- 我们用一个字母来简记各列CSJDPQV
- 函数依赖关系F={C→SJDPQV, JP →C, DS → P, J →S}
  - 比前例增加了J →S，“每个项目只能有一个供货商”
- 满足BCNF吗?
  - 我们知道C是候选键，JP是候选键，JDS是候选键
  - 观察DS → P, J →S
  - DS是部分键，J也是部分键，P和S是键属性
  - 这些是对于键属性的函数依赖



举例

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

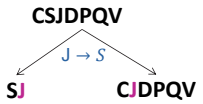
- 我们用一个字母来简记各列CSJDPQV
- 函数依赖关系F={C→SJDPQV, JP →C, DS → P, J →S}
- 分解为BCNF?



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→SJDPQV, JP →C, DS → P, J →S}
- 另一种分解



可见可能有多种分解方式

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？ **CTB是主键，没有非键属性**
- 但是，仍然存在着冗余
  - 明显对一门课而言，教材和老师是独立的，出现所有组合

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

这种冗余就是多值依赖（MVD）

多值依赖（MVD）

- $X \rightarrow\rightarrow Y$
  - 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！
  - 在X相同的情况下，Y和Z的全组合都出现

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

多值依赖（MVD）

- $X \rightarrow\rightarrow Y$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

理解一下

- $X \rightarrow\rightarrow Y$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]	y1	z1
t2	t1[X]	y2	z2
t3	t1[X]	y1	z2

理解一下

- $X \rightarrow\rightarrow Y$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]	y1	z1
t2	t1[X]	y2	z2
t3	t1[X]	y1	z2
t4	t1[X]	y2	z1

Y和Z的值是独立的，有所有的组合

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

函数依赖FD是多值依赖MVD的特例

- FD是最简单的MVD
- $X \rightarrow Y$是$X \rightarrow\rightarrow Y$
- 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，$X \to Y$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$X \to\to Y$，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$X \to Y$必须满足上述条件
  - BCNF仅允许“$Y \subseteq X$”和“X是一个超键”两种情况
    - 而$XY = R$，所以如果$X \to Y$，那么$X \to\to$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
- 存在单属性键吗？
  - Yes，满足4NF
  - No，考虑是否包含多值依赖
- 对于多值依赖$X \to\to Y$，分解为XY和R-Y两部分

连接依赖（JD, Join Dependency）

- 连接依赖 $\bowtie\{R_1, R_2, \ldots R_n\}$
  - 如果$R_1, R_2, \ldots R_n$是Rd的无损连接分解
  - 那么称为连接依赖
- 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
  - 因为可以进行无损分解
  - $X \to\to Y$，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⋈{R_1,R_2,...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：索引的增删
- 外部模式设计：视图

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
- 内容1：建立哪些索引（index）？
- 内容2：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
- 内容3：调整查询和事务

我们在这里先介绍建立索引的语法，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进一步介绍

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（不同数据库系统的语法有区别）

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*;

Outline

- 逻辑设计：ER图到关系模型
- 模式细化：范式
- 物理设计：索引的增删
- 外部模式设计：视图
  - 如何定义？
  - 怎么使用？
  - Grant和Revoke

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 *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'*;

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 *CSStCo*(*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 CSStCo
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，不能对视图进行增删改操作
  - 对于视图的insert, delete, update，如何对应回基本表？
- 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？
- 但如果视图满足某些条件，非常简单，可以更新
  - ☞称为可更新视图

可更新视图(Updatable View)

- 条件：视图定义中
  - 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';
```

理解一下

- 条件：视图定义中
  - From必须是单个关系（或单个可更新视图）R
    - 没有连接，因为很难从连接的结果反推原表的内容
  - Where如果有子查询，那么子查询不能包含R
    - 如果有子查询，也不是相关嵌套查询
  - 没有distinct和aggregation
    - 每个结果的行对应一个原始行
  - 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

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

如何解决？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'*;

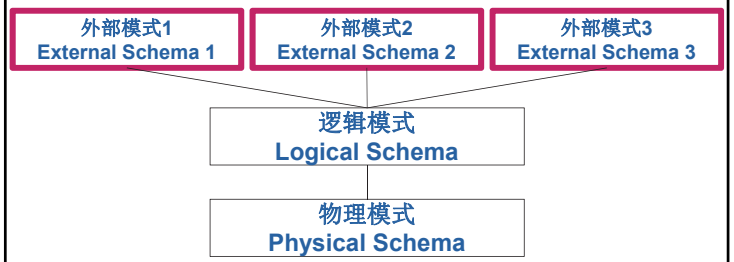
delete from CSSt
where Sname='周瑜' and Major='计算机';

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Alpha创建了一个基本表R
  - 那么，Alpha就拥有表R的所有访问权限和grant权限
- Alpha可以把表R的访问权限授予他人
  - 都是对具体记录的操作
- 但是，数据定义语句的权利(create, alter, drop)不能授予或收回

视图的访问权限

- 建立视图时，必须拥有所有用到的表的select权限

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'*;

- 建立视图时必须有Student, Course, TakeCourse的select权限

视图的访问权限

- 把这个视图授予计算机系学生用户CS_Student
  grant select on *CSStCo* to *CS_Student*;
- CS_Student可以访问视图CSStCo
- CS_Student不能直接访问Student, Course, TakeCourse
- 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：同Grant语句

- 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

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
  (Joe) 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

举例

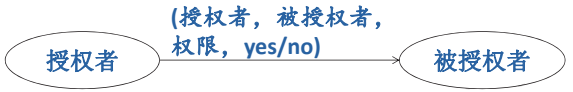
- 假设Sailors, Boats, 和Reserves表
- Joe是Sailors表的创建所有人
- Joe把Sailors的读权限和对读的授权权限授予Art
  (Joe) grant select on *Sailors* to *Art* with grant option;
- Art把Sailors的读权限和对读的授权权限授予Bob
  (Art) grant select on *Sailors* to *Bob* with grant option;
- 一段时间之后，Joe收回Art的读权限和Art进一步的授权给Bob的衍生读权限

举例

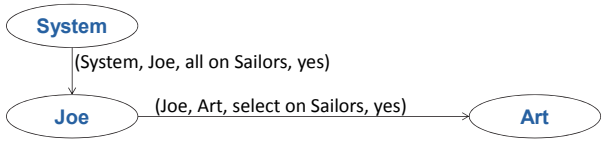
- 假设Sailors, Boats, 和Reserves表
- Joe是Sailors表的创建所有人
- Joe把Sailors的读权限和对读的授权权限授予Art
  (Joe) grant select on *Sailors* to *Art* with grant option;
- Art把Sailors的读权限和对读的授权权限授予Bob
  (Art) grant select on *Sailors* to *Bob* with grant option;
- 一段时间之后，Joe收回Art的读权限和Art进一步的授权给Bob的衍生读权限
  (Joe) revoke select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

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

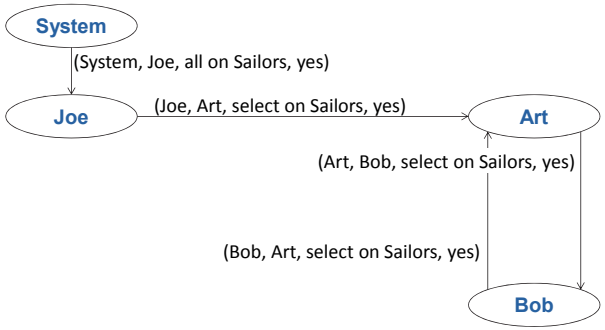


授权图举例 (授权者，被授权者，权限，yes/no)



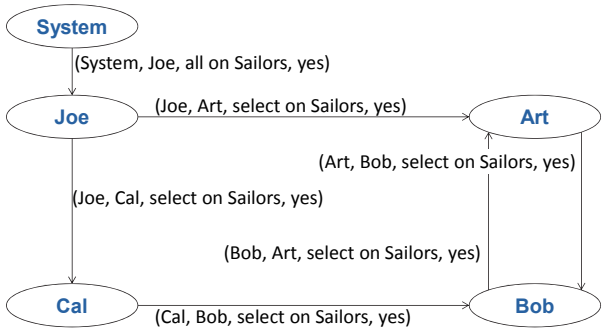
- Joe创建表Sailors
- (Joe) grant select on *Sailors* to *Art* with grant option;

授权图举例 (授权者，被授权者，权限，yes/no)



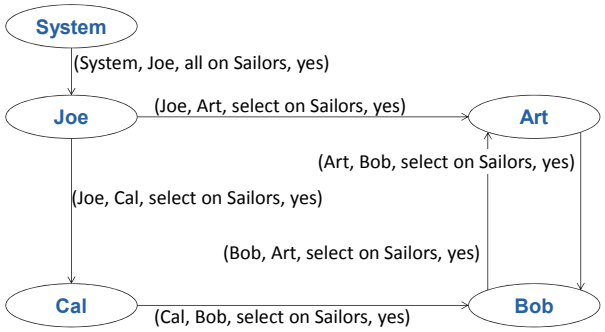
- (Art) grant select on *Sailors* to *Bob* with grant option;
- (Bob) grant select on *Sailors* to *Art* with grant option;

授权图举例 (授权者，被授权者，权限，yes/no)



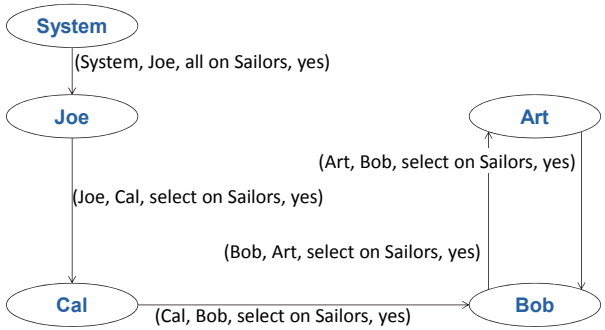
- (Joe) grant select on *Sailors* to *Cal* with grant option;
- (Cal) grant select on *Sailors* to *Bob* with grant option;

授权图举例 (授权者，被授权者，权限，yes/no)



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

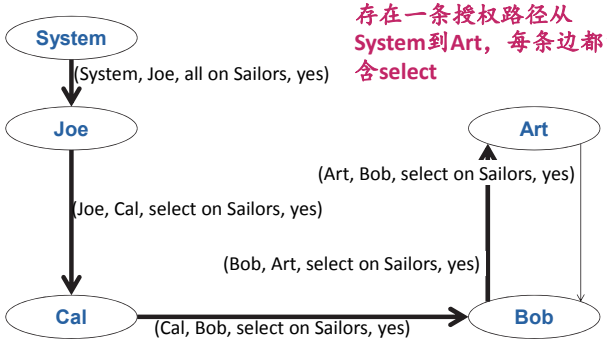
授权图举例 (授权者，被授权者，权限，yes/no)



- (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)

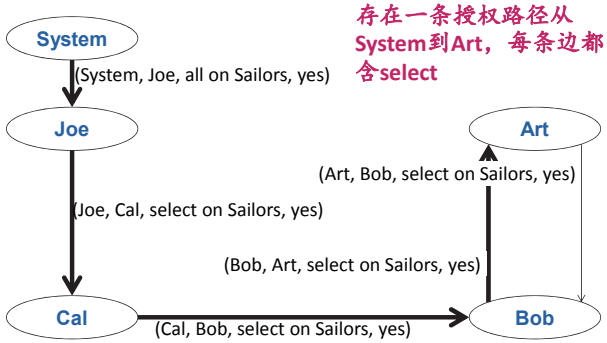
存在一条授权路径从System到Art，每条边都包含select



- (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！

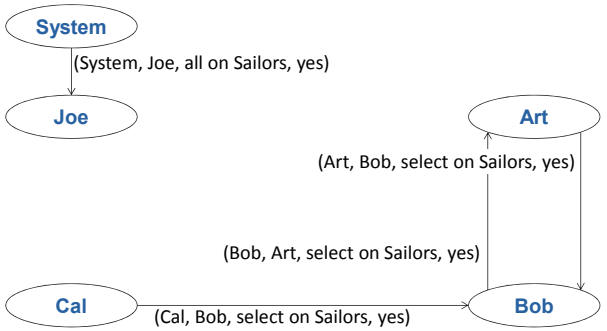
授权图举例 (授权者，被授权者，权限，yes/no)

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Owner对Sailors必须可读
  - Jack当然也就不能读YoungSailors

小结

- 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：索引的增删
- 外部模式设计：视图