# 关系数据库理论I
# CHAPTER 4: THE Relational Database Theory
数据依赖

毛斐巧
深圳大学计算机与软件学院

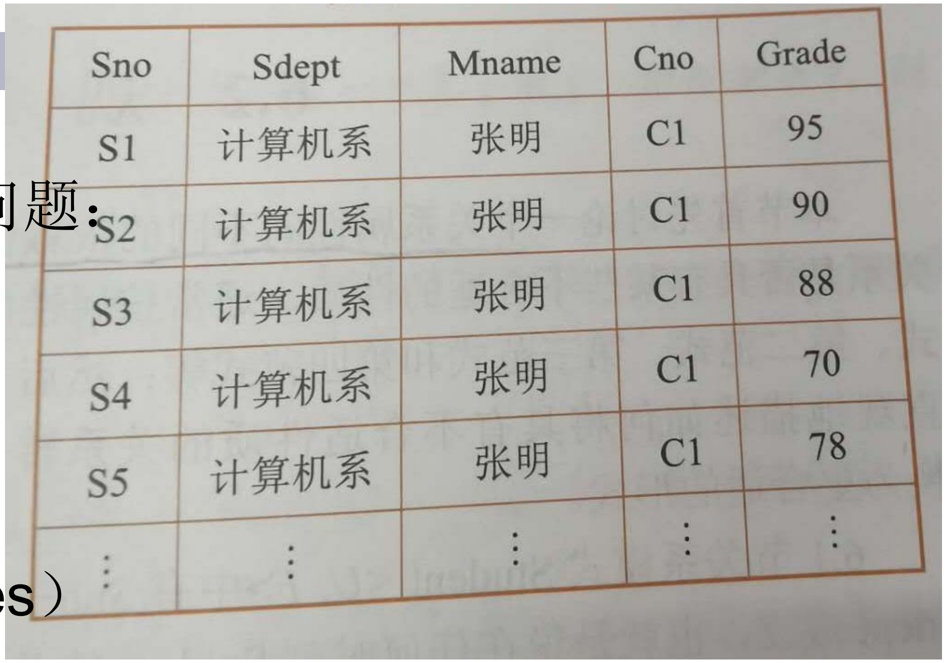
## 主要内容

- 问题的提出　　（对应6.1节）
- 数据依赖　　（对应6.2.1节）
- 确定函数依赖　　（对应6.3节）
- 计算F+和X+　　（6.3节定义6.12， 算法6.1）
- 确定候选码　　（对应6.2.2节+？）
- 模式分解　　（6.4.1和6.4.2节）

## 1.数据依赖问题的提出

- 关系数据库逻辑设计
  - 针对具体问题，如何构造一个适合于它的数据模式
  - 数据库逻辑设计的工具——关系数据库的规范化理论

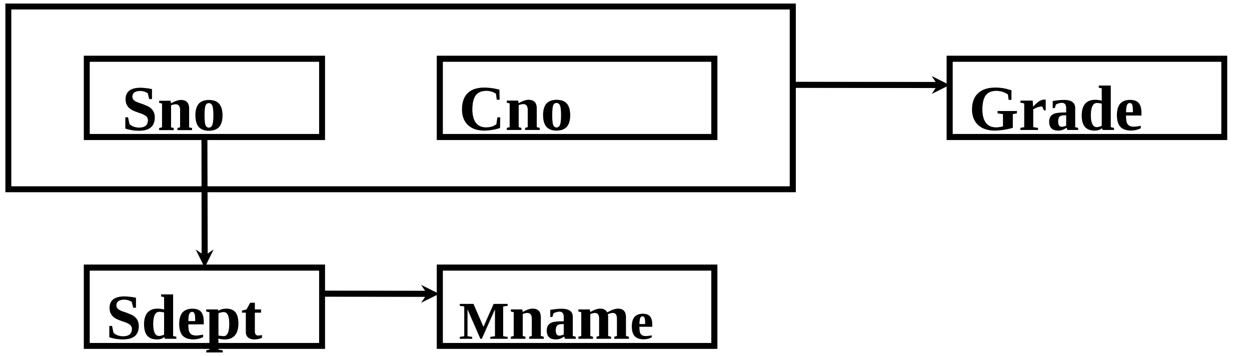
- [例6.1] 建立一个描述学校教务的数据库。
  涉及的对象包括：
  - 学生的学号（Sno）
  - 所在系（Sdept）
  - 系主任姓名（Mname）
  - 课程号（Cno）
  - 成绩（Grade）
- 假设学校教务的数据库模式用一个单一的关系模式Student来表示，则该关系模式的属性集合为：
  - U ＝{Sno, Sdept, Mname, Cno, Grade}



| Sno | Sdept | Mname | Cno | Grade |
|---|---|---|---|---|
| S1 | 计算机系 | 张明 | C1 | 95 |
| S2 | 计算机系 | 张明 | C1 | 90 |
| S3 | 计算机系 | 张明 | C1 | 88 |
| S4 | 计算机系 | 张明 | C1 | 70 |
| S5 | 计算机系 | 张明 | C1 | 78 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

- 关系模式Student<U, F>中存在的问题:
- （1）数据冗余
  - 浪费大量的存储空间
    - 每一个系主任的姓名重复出现
- （2）更新异常（Update Anomalies）
  - 数据冗余，更新数据时，维护数据完整性代价大。
    - 某系更换系主任后，必须修改与该系学生有关的每一个元组。
- （3）插入异常（Insertion Anomalies）
  - 如果一个系刚成立，尚无学生，则无法把这个系及其系主任的信息存入数据库。
- （4）删除异常（Deletion Anomalies）
  - 如果某个系的学生全部毕业了， 则在删除该系学生信息的同时，把这个系及其系主任的信息也丢掉了。

- 结论
  - Student关系模式不是一个好的模式。
  - 一个“好”的模式应当不会发生插入异常、删除异常和更新异常，数据冗余应尽可能少。
- 原因
  - 由存在于模式中的某些数据依赖引起的。



- 解决方法
  - 用规范化理论改造关系模式来消除其中不合适的数据依赖

## 2.数据依赖

- 是一个关系内部属性与属性之间的一种约束关系
  - 通过属性间值的相等与否体现出来的数据间相互联系
- 是现实世界属性间相互联系的抽象

数据依赖的主要类型

- 函数依赖（Functional Dependency，简记为FD）
- 多值依赖（Multi-Valued Dependency，简记为MVD）

## 数据依赖

- （1）函数依赖
- （2）平凡函数依赖与非平凡函数依赖
- （3）完全函数依赖与部分函数依赖
- （4）传递函数依赖

## （1）函数依赖

- 定义1 设*R(U)*是一个属性集*U*上的关系模式，*X*和*Y*是*U*的子集。若对于*R(U)*的任意一个可能的关系*r*，*r* 中不可能存在两个元组在*X*上的属性值相等， 而在*Y*上的属性值不等，则称“*X*函数确定*Y*”或“*Y*函数依赖于*X*”，记作*X*→*Y*。
  - $t1[X] = t2[X] \longrightarrow t1[Y] = t2[Y].$
- 符号:
- $X \subset R$ 表示X 是模式R属性的子集.
- $X \longrightarrow Y$ 表示*X*函数确定*Y* .

Example:

| A | B | C | D |
|---|---|---|---|
| a1 | b1 | c1 | d1 |
| a1 | b2 | c1 | d2 |
| a2 | b2 | c2 | d2 |
| a2 | b3 | c2 | d3 |
| a3 | b3 | c2 | d4 |

which is true?

$A \Longrightarrow B$

$A \Longrightarrow C$ yes

$C \Longrightarrow A$

$A \Longrightarrow D$

$B \Longrightarrow D$

$AB \Longrightarrow D$ yes

- 若$X \rightarrow Y$，X 是决定因素.
- 若$X \rightarrow Y$，并且$Y \rightarrow X$, 则记为$X \longleftrightarrow Y$。
- 若Y不函数依赖于X, 则记为$X \nrightarrow Y$。

函数依赖不是指关系模式**R**的某个或某些关系实例满足的约束条件，而是指**R**的所有关系实例均要满足的约束条件。

## （2）平凡函数依赖与非平凡函数依赖

- $X \rightarrow Y$，但$Y \not\subseteq X$则称$X \rightarrow Y$是非平凡的函数依赖( NON-Trivial FD) 。
- $X \rightarrow Y$，但$Y \subseteq X$ 则称$X \rightarrow Y$是平凡的函数依赖( Trivial FD) 。

NON-Trivial FD：(Sno, Cno) → Grade

Trivial FD ：(Sno, Cno) → Sno

(Sno, Cno) → Cno

对于任一关系模式，平凡函数依赖都是必然成立的，它不反映新的语义。
若不特别声明， 我们总是讨论非平凡函数依赖。

## （3）完全函数依赖与部分函数依赖

- 定义2 在$R(U)$中，如果$X \rightarrow Y$，并且对于$X$的任何一个真子集$X'$, 都有 $X' \nrightarrow Y$, 则称$Y$对$X$完全函数依赖，记作$X \xrightarrow{F} Y$。
- 若$X \rightarrow Y$，但$Y$不完全函数依赖于$X$，则称$Y$对$X$部分函数依赖，记作$X \xrightarrow{P} Y$
  - {CNO, CNAME} $\xrightarrow{P}$ CLOCATION
  - CNO $\xrightarrow{F}$ CLOCATION

## (4) 传递函数依赖

- 定义3 在$R(U)$中，如果$X \rightarrow Y (Y \not\subseteq X)$，$Y \nrightarrow X$，$Y \rightarrow Z$，$Z \not\subseteq Y$, 则称$Z$对$X$传递函数依赖(transitive functional dependency)。记为：$X \xrightarrow{\text{传递}} Z$。

- 注: 如果$Y \rightarrow X$, 即$X \longleftrightarrow Y$，则$Z$直接依赖于$X$，而不是传递函数依赖。
  - [例] 在关系Std(Sno, Sdept, Mname)中，有：
    - Sno → Sdept，Sdept → Mname，
    - Mname传递函数依赖于Sno

*

## 关系数据库理论

如何设计一个高质量的数据库？

去除冗余!

分解!

如何分解?

首先：确定数据依赖 （Identify Functional Dependency）

如何确定主键（主码）？

## 确定函数依赖(1)

- 平凡的函数依赖: if $Y \subseteq X \subseteq R$, then $X \longrightarrow Y$.
- If X is a superkey （超码）of R and Y is any subset of R, then $X \longrightarrow Y$ is in R.
- 基于约束规定的依赖.

  Employees(SSN, Name, Years_of_emp, Salary, Bonus)

规定: Employees hired the same year have the same salary.

依赖关系:

Years_of_emp $\longrightarrow$ Salary

## 确定函数依赖(2)

- 分析属性间的语义关系

  Addresses(City, Street, Zipcode)

  Zipcode $\longrightarrow$ City

- 从已知的数据依赖推导出新的依赖

Let R(A, B, C), F = {A $\longrightarrow$ B, B $\longrightarrow$ C}.

A $\longrightarrow$ C can be derived from F.

*A*→C 为*F*所蕴涵

基于**F**所蕴含的关系，可以确定所有的函数依赖，可以确定码。

## 确定函数依赖(3)

- 在关系模式R<U,F>中为F所逻辑蕴涵的函数依赖的全体叫作F的闭包（closure），记为$F^+$。

- A BIG $F^+$ may be derived from a small F.

Example: For R(A, B, C) and

$$F = \{A \rightarrow B,\ B \rightarrow C\}$$

$$F^+ = \{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow C,$$
$$A \rightarrow A,\ B \rightarrow B,\ C \rightarrow C,$$
$$AB \rightarrow AB,\ AB \rightarrow A,\ AB \rightarrow B,\ \dots\}$$

$F^+$是个很大的集合，如何推理？Armstrong公理

## 计算$F^+$ (1)

Armstrong公理系统(1974):

- A1 自反律（reflexivity rule）：若$Y \subseteq X \subseteq U$，则$X \rightarrow Y$ 为$F$所蕴涵。
- A2 增广律（augmentation rule）：若$X\rightarrow Y$为$F$所蕴涵，且$Z \subseteq U$，则$XZ\rightarrow YZ$ 为$F$所蕴涵。
- A3 传递律（transitivity rule）：若$X\rightarrow Y$及$Y\rightarrow Z$为$F$所蕴涵，则$X\rightarrow Z$ 为$F$所蕴涵。

## 计算$F^+$ (2)

- 根据Armstrong公理系统三条推理规则可以得到下面三条推理规则：
  - 合并规则（union rule）：

    由$X \to Y$，$X \to Z$，有$X \to YZ$。
  - 伪传递规则（pseudo transitivity rule）：

    由$X \to Y$，$WY \to Z$，有$XW \to Z$。
  - 分解规则（decomposition rule）：

    由$X \to Y$及$Z \subseteq Y$，有$X \to Z$。

Armstrong公理系统是有效的、完备的

## 属性的闭包

- 在关系模式*R<U,F>*中为*F*所逻辑蕴涵的函数依赖的全体叫作*F*的闭包，记为$F^+$。

- 设*F*为属性集*U*上的一组函数依赖，*X*、*Y* $\subseteq$ *U*， $X_F^+$={ *A*|*X*→*A*能由*F*根据Armstrong公理导出}，$X_F^+$称为属性集***X***关于函数依赖集***F***的闭包。

$$X^+ = \{ A \mid X \longrightarrow A \in F^+ \}$$

Theorem: $X \longrightarrow Y \in F^+$ if and only if $Y \subseteq X^+$.

*

## 计算X⁺ (1)

- 求闭包的算法 （算法6.1）
- 求属性集$X$（$X \subseteq U$）关于$U$上的函数依赖集$F$的闭包$X_F^+$

① 初始化：令$X^{(0)}=X$，$i=0$

② 求$B$：对$X^{(i)}$中的每个元素，依次检查相应的函数依赖，将依赖它的属性加入$B$ 。

③ 并： $X^{(i+1)}=B \cup X^{(i)}$ 。

④ 判断：$X^{(i+1)}= X^{(i)}$ 。

⑤ 若$X^{(i+1)}$与$X^{(i)}$相等或$X^{(i)}=U$ ， 则$X^{(i)}$就是$X_F^+$，
算法终止。

⑥ 若否，则$i=i+1$，返回第②步。

## 计算$X^+$ (2)

[Example]  Relation : $R<U，F>$

$U=\{A，B，C，D，E\}$；

$F=\{AB\rightarrow C，B\rightarrow D，C\rightarrow E，EC\rightarrow B，AC\rightarrow B\}$

Question:（$AB$）$_F^+$？

Solution:  Let $X^{(0)}=AB$；

(1) $X^{(1)}=AB\cup CD=ABCD$。

(2) $X^{(0)}\neq X^{(1)}$

$X^{(2)}=X^{(1)}\cup E=ABCDE$。

(3) $X^{(2)}=U$，End

→（$AB$）$_F^+=ABCDE$。

## 计算$X^+$ (3)

求（$X$）$_F^+$ 的作用？ 确定码

定理: 如果有$R(A_1, ..., A_n)$ 和函数依赖F in R, $K \subseteq R$ 是

- 超码（ superkey ）如果满足$K^+ = \{A_1, ..., A_n\}$;
- 候选码 （candidatekey）如果 K是超码， 并且对于K的任意子集 X, $X^+ \neq \{A_1, ..., A_n\}$.

- 在上面例子中
  - AB 是超码， 因为 $(AB)^+ = ABCDE$.
  - 因为 $A^+ = A$, $B^+ = BD$, A 或 B不是超码.
  - 所以AB是候选码

## Worked Example 1:

Relation schema: $R = (A, B, C, D, E)$

$F = \{A\text{->}BC, CD\text{->}E, A\text{->}D, B\text{->}D, E\text{->}A\}$

1) *Find* $A^+, B^+, BC^+$
2) Find Candidate keys of $R$

## Worked Example 1 (cont')

Compute $A^+$($F$ = {$A$->$BC$, $CD$->$E$, $A$->$D$, $B$->$D$, $E$->$A$})

1. result = *A* A->BC A->D
2. result = *ABCD* CD->E
3. result = *ABCDE*
4. *Therefore* $A^+$= *ABCDE*

## Worked Example 1 (cont')

Compute $B^+$(*F* = {*A->BC, CD->E, A->D, B->D, E->A*})

1. result = *B* B->D
2. result = *BD*
3. *Therefore* $B^+$=*BD*

## Worked Example 1 (cont’)

Compute *$BC^+$(F = {A->BC, CD->E, A->D, B->D, E->A})*

| | | |
|---|---|---|
| 1. | result = *BC* | B->D |
| 2. | result = *BCD* | CD->E |
| 3. | result = *BCDE* | E->A |
| 4. | result = *ABCDE* | |
| 5. | *Therefore $BC^+$=ABCDE* | |
| 6. | *Hence candidate keys are??* | |

## Example 2

List Candidate keys of *R*
*F* = {*A*->*BC*, *CD*->*E*, *B*->*D*, *E*->*A*}

- Let $\alpha$ be a *candidate key* for R

⇔ $\alpha \rightarrow$ R, there is no $\gamma$ s.t. $\gamma \subset \alpha$, $\gamma \rightarrow$ R

*A*? *B*? *C*? *D*? *E*?
*BC*? *BD*? *CD*?

Candidate keys : *A* *E* *BC* *CD*

## 确定候选码(1)

Let F be a set of FDs in relation schema R($A_1$, ..., $A_n$).

**Method 1** (can be automated)

```
(1) for each $A_i$, compute $A_i^+$;
        if $A_i^+ = A_1 A_2 \ldots A_n$
            then $A_i$ is a candidate key;
```

## 确定候选码(2)

```
(2) for each pair $A_iA_j$, $i \neq j$
    if $A_i$ or $A_j$ is a candidate key
        then $A_iA_j$ is not a candidate key;
    else compute $(A_i A_j)^+$;
        if $(A_iA_j)^+ = A_1 A_2 \ldots A_n$
            then $(A_i A_j)$ is a candidate key;
```

## 确定候选码(3)

```
(3) for each triple $A_iA_jA_k$, $i \neq j$, $i \neq k$, $j \neq k$
      if any subset of $A_iA_jA_k$ is a candidate key
        then $A_iA_jA_k$ is not a candidate key;
      else compute $(A_iA_jA_k)^+$;
          if $(A_iA_jA_k)^+ = A_1\ A_2\ \ldots\ A_n$
            then $(A_iA_jA_k)$ is a candidate key;
(4) . . . . . . .
```

## 确定候选码

**Method 2** (Graph Approach)

Step 1: Draw the dependency graph of F.
Each vertex corresponds to an attribute.
Edges can be defined as follows:

$$\begin{tikzcd} A \arrow[r] & B & \text{becomes} & A \arrow[r] & B \end{tikzcd}$$

$$\begin{tikzcd} & & & & B \\ A \arrow[r] & BC & \text{becomes} & A \arrow[ur] \arrow[dr] & \\ & & & & C \end{tikzcd}$$

$$\begin{tikzcd} & & & A \arrow[dr, no head] & & \\ AB \arrow[r] & C & \text{becomes} & & {} \arrow[r] & C \\ & & & B \arrow[ur, no head] & & \end{tikzcd}$$

## Finding Candidate Keys from FDs

**Step 2**: Identify the set of vertices $V_{ni}$ that have no incoming edges.

Claim 1: Any candidate key must have all attributes in $V_{ni}$.

Claim 2: If $V_{ni}$ forms a candidate key, then $V_{ni}$ is the only candidate key.

## Finding Candidate Keys from FDs

**Step 3**: Identify the set of vertices $V_{oi}$ that have only incoming edges.

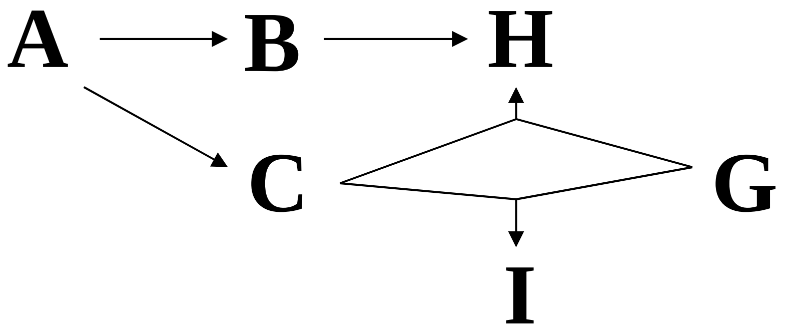
Claim 3: No candidate key will contain any attribute in $V_{oi}$.

Step 4: Use observation to find other candidate keys if there is any.

## Finding Candidate Keys from FDs

Example: Suppose R(A, B, C, G, H, I),

$F = \{A \longrightarrow BC, CG \longrightarrow HI, B \longrightarrow H\}$



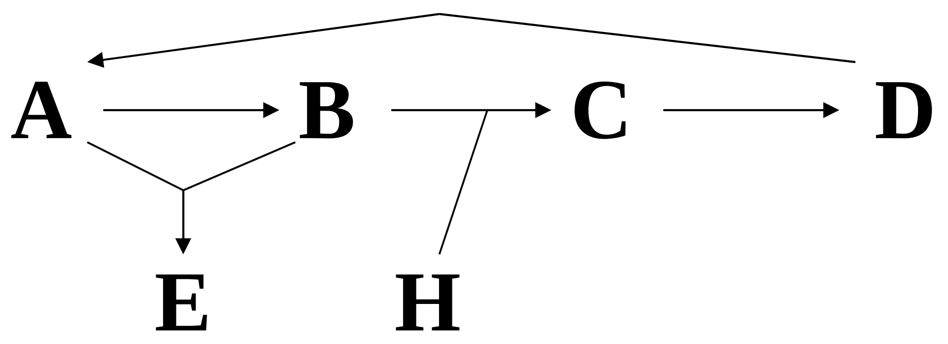
$V_{ni} = \{A, G\}$, $V_{oi} = \{H, I\}$.

Since $(AG)^{+} = ABCGHI$, AG is the only candidate key of R.

## Finding Candidate Keys from FDs

Example: Suppose R(A, B, C, D, E, H),

$F = \{ A \rightarrow B,\ AB \rightarrow E,\ BH \rightarrow C,\ C \rightarrow D,\ D \rightarrow A \}$



$V_{ni} = \{ H \}$, $V_{oi} = \{ E \}$.

Candidate keys: AH, BH, CH, DH.

## 模式分解

如何设计一个高质量的数据库？

去除冗余!

分解!

如何分解?

## Example

Consider :（动物名称，动物属性， 动物居住地）

F={动物名称→动物属性,

动物属性→动物居住地,

动物名称→动物居住地}

如何分解这个表?

## Example

SL

| 动物名称 | 动物属性 | 动物居住地 |
|---|---|---|
| 灰太狼 | 羊食 | 青青山 |
| 喜羊羊 | 草食 | 青青草原 |
| 食人鱼 | 全食 | 青青河 |
| 美羊羊 | 草食 | 青青草原 |
| 蛤蟆 | 小食 | 青青草原 |

## Example

1. Decompose it into:

   SN(动物名称)

   SD(动物属性 )

   SO(动物居住地)

## Example

| SN | SD | SO |
|---|---|---|
| 动物名称 | 动物属性 | 动物居住地 |
| 灰太狼 | 羊食 | 青青山 |
| 喜羊羊 | 草食 | 青青草原 |
| 食人鱼 | 全食 | 青青河 |
| 美羊羊 | 小食 | |
| 蛤蟆 | | |

Where I am?
Information lost !

## Example

2. SL

NL(动物名称,动物居住地)

DL(动物属性,动物居住地)

then：

NL

| Sn | So |
|---|---|
| 灰太狼 | 青青山 |
| 喜羊羊 | 青青草原 |
| 食人鱼 | 青青河 |
| 美羊羊 | 青青草原 |
| 蛤蟆 | 青青草原 |

DL

| SD | So |
|---|---|
| 羊食 | 青青山 |
| 草食 | 青青草原 |
| 全食 | 青青河 |
| 小食 | 青青草原 |

## Example

$NL \bowtie DL$

| 动物名称 | 动物居住地 | 动物属性 |
|---|---|---|
| 灰太狼 | 青青山 | 羊食 |
| 喜羊羊 | 青青草原 | 草食 |
| 喜羊羊 | 青青草原 | 小食 |
| 食人鱼 | 青青河 | 全食 |
| 美羊羊 | 青青草原 | 草食 |
| 美羊羊 | 青青草原 | 小食 |
| 蛤蟆 | 青青草原 | 草食 |
| 蛤蟆 | 青青草原 | 小食 |

?

We cannot find “动物属性” information for喜羊羊,美羊羊,蛤蟆

information lost and wrong

information is generated

## Example

3. SL：

ND(动物名称,动物属性)

NL(动物名称,动物居住地)

Then

## Example

ND

| 动物名称 | 动物属性 |
|---|---|
| 灰太狼 | 羊食 |
| 喜羊羊 | 草食 |
| 食人鱼 | 全食 |
| 美羊羊 | 草食 |
| 蛤蟆 | 小食 |

NL

| 动物名称 | 动物居住地 |
|---|---|
| 灰太狼 | 青青山 |
| 喜羊羊 | 青青草原 |
| 食人鱼 | 青青河 |
| 美羊羊 | 青青草原 |
| 蛤蟆 | 青青草原 |

## Example

ND $\bowtie$ NL

| 动物名称 | 动物属性 | 动物居住地 |
|---|---|---|
| 灰太狼 | 羊食 | 青青山 |
| 喜羊羊 | 草食 | 青青草原 |
| 食人鱼 | 全食 | 青青河 |
| 美羊羊 | 草食 | 青青草原 |
| 蛤蟆 | 小食 | 青青草原 |

No information lost.

连接无损失

## 模式分解

- 无损连接分解

Lossless Join Decomposition

- 保持函数依赖分解

Dependency-Preserving Decomposition

## 无损连接分解

Lossless Join Decomposition

ND

| 动物名称 | 动物属性 |
|---|---|

NL

| 动物名称 | 动物居住地 |
|---|---|

ND $\bowtie$ NL 连接无损失

| 动物名称 | 动物属性 | 动物居住地 |
|---|---|---|
| 灰太狼 | 羊食 | 青青山 |
| 喜羊羊 | 草食 | 青青草原 |
| 食人鱼 | 全食 | 青青河 |
| 美羊羊 | 草食 | 青青草原 |
| 蛤蟆 | 小食 | 青青草原 |

No information lost.

## 无损连接分解 (1)

Definition: Let R be a relation schema. A set of relation schemas $\{R_1, R_2, ..., R_n\}$ is a decomposition of R if $R = R_1 \cup ... \cup R_n$.

Claim: If $\{R_1, R_2, ..., R_n\}$ is a decomposition of R and r is an instance of R, then

$\{R_1, R_2, ..., R_n\}$ is a lossless (non-additive) join decomposition of R if for every legal instance r of R, we have

$$r = \pi_{R1}(r) \infty \pi_{R2}(r) \infty \; . \; . \; . \infty \pi_{Rn}(r)$$

## 无损连接分解(2)

判定一个分解是否无损连接性

适用于分解为两个关系模式

定理（同定理6.5）: Let R be a relation schema and F be a set of FDs in R. Then a decomposition of R, {R1, R2}, is a lossless-join decomposition if and only if

- $R1 \cap R2 \longrightarrow R1 - R2$; or
- $R1 \cap R2 \longrightarrow R2 - R1$.

1. 计算 $R1 \cap R2$； 指属性的交集
2. 计算 $R1 - R2$； 属性的差
3. 判定函数依赖关系是否成立

## 无损连接分解(3)

- **Example**
- F={动物名称→动物属性,动物属性→动物居住地,动物名称→动物居住地}
  - **T1 (动物名称，动物属性)**
  - **T2 (动物名称，动物居住地)**
  - T1 ∩T2=动物名称
  - T1 -T2=动物属性
  - T2 –T1=动物居住地
    - 动物名称 $\longrightarrow$ 动物属性
    - 动物名称 $\longrightarrow$ 动物居住地

无损连接分解！

## 无损连接分解(4)

- **Example**
- F={动物名称→动物属性,动物属性→动物居住地,动物名称→动物居住地}
  - **T1 (动物名称，动物居住地)**
  - **T2 (动物属性，动物居住地)**
  - T1 ∩T2=动物居住地
  - T1 -T2=动物名称
  - T2 –T1=动物属性
    - 动物居住地 $\not\rightarrow$ 动物名称
    - 动物居住地 $\not\rightarrow$ 动物属性

有损连接分解！

## 无损连接分解(5)

- **Example**
- F={动物名称→动物属性,动物属性→动物居住地,动物名称→动物居住地}
  - **T1 (动物名称，动物属性)**
  - **T2 (动物属性，动物居住地)**
  - T1 ∩T2=动物属性
  - T1 -T2=动物名称
  - T2 –T1=动物居住地
    - 动物属性 $\not\longrightarrow$ 动物名称
    - 动物属性 $\longrightarrow$ 动物居住地

无损连接分解！

## 无损连接分解(6)

判定一个分解是否无损连接性

适用于分解为多个关系模式

**6.2** 算法 判定无损连接性的算法

输入：关系模式**R(**$A_1$，$A_2$，**…,**$A_n$**),**它的函数依赖集**F**以及分解 $\rho$**={**$R_1$，$R_2$，**…,**$R_k$**}**。

方法：

（**1**）构造表：构造一个**k**行**n**列的表，第**i**行对应于关系模式$R_i$，第**j**列对应于属性$A_j$。

（**2**）填表（根据属性的分配）：如果$A_j \in R_i$, 则在第**i**行第**j**列上放符号$a_j$,否则放符号$b_{ij}$。

# 无损连接分解(6)

（3）更新表（根据F更新）：逐一检查F中的每一个函数依赖，并修改表中的元素。方法：取F中一个函数依赖X→Y，在X的列中寻找相同的行，然后将这些行中Y的分量改为相同的符号，如果其中有$a_j$,则将$b_{ij}$改为$a_j$;若其中无$a_j$，则改为某一个$b_{ij}$。

（4）循环更新：反复检查第（2）步，至无改变为止.

（5）判断：若存在某一行为$a_1,a_2,\ldots,a_k$, 则分解具有无损连接性；如果F中所有函数依赖都不能再$\rho$修改表中的内容，且没有发现这样的行，则分解$\rho$不具有无损连接性。

## example

- 举例：已知R<U,F>，U={A,B,C,D,E}，F={A→C,B→C,C→D,DE→C,CE→A}，R的一个分解为R1(AD)，R2(AB)，R3(BE)，R4(CDE)，R5(AE)，判断这个分解是否具有无损连接性。
- ① 构造一个初始的二维表，若“属性”属于“模式”中的属性，则填aj，否则填bij。

| 属性<br>模式 | A | B | C | D | E |
|---|---|---|---|---|---|
| $R_1(AD)$ | $a_1$ | $b_{12}$ | $b_{13}$ | $a_4$ | $b_{15}$ |
| $R_2(AB)$ | $a_1$ | $a_2$ | $b_{23}$ | $b_{24}$ | $b_{25}$ |
| $R_3(BE)$ | $b_{31}$ | $a_2$ | $b_{33}$ | $b_{34}$ | $a_5$ |
| $R_4(CDE)$ | $b_{41}$ | $b_{42}$ | $a_3$ | $a_4$ | $a_5$ |
| $R_5(AE)$ | $a_1$ | $b_{52}$ | $b_{53}$ | $b_{54}$ | $a_5$ |

## example

- ② 根据A→C，对上表进行处理，由于属性列A上第1、2、5行相同均为a1，所以将属性列C上的b13、b23、b53改为同一个符号b13（取行号最小值）。

| 属性 / 模式 | A | B | C | D | E |
|---|---|---|---|---|---|
| $R_1$(AD) | $a_1$ | $b_{12}$ | $b_{13}$ | $a_4$ | $b_{15}$ |
| $R_2$(AB) | $a_1$ | $a_2$ | $b_{23}$ | $b_{24}$ | $b_{25}$ |
| $R_3$(BE) | $b_{31}$ | $a_2$ | $b_{33}$ | $b_{34}$ | $a_5$ |
| $R_4$(CDE) | $b_{41}$ | $b_{42}$ | $a_3$ | $a_4$ | $a_5$ |
| $R_5$(AE) | $a_1$ | $b_{52}$ | $b_{53}$ | $b_{54}$ | $a_5$ |

| 属性 / 模式 | A | B | C | D | E |
|---|---|---|---|---|---|
| $R_1$(AD) | $a_1$ | $b_{12}$ | $b_{13}$ | $a_4$ | $b_{15}$ |
| $R_2$(AB) | $a_1$ | $a_2$ | $b_{13}$ | $b_{24}$ | $b_{25}$ |
| $R_3$(BE) | $b_{31}$ | $a_2$ | $b_{33}$ | $b_{34}$ | $a_5$ |
| $R_4$(CDE) | $b_{41}$ | $b_{42}$ | $a_3$ | $a_4$ | $a_5$ |
| $R_5$(AE) | $a_1$ | $b_{52}$ | $b_{13}$ | $b_{54}$ | $a_5$ |

## example

- ③ 根据B→C，对上表进行处理，由于属性列B上第2、3行相同均为a2，所以将属性列C上的b13、b33改为同一个符号b13（取行号最小值）。

| 属性 模式 | A | B | C | D | E |
|---|---|---|---|---|---|
| $R_1(AD)$ | $a_1$ | $b_{12}$ | $b_{13}$ | $a_4$ | $b_{15}$ |
| $R_2(AB)$ | $a_1$ | $a_2$ | $b_{13}$ | $b_{24}$ | $b_{25}$ |
| $R_3(BE)$ | $b_{31}$ | $a_2$ | $b_{33}$ | $b_{34}$ | $a_5$ |
| $R_4(CDE)$ | $b_{41}$ | $b_{42}$ | $a_3$ | $a_4$ | $a_5$ |
| $R_5(AE)$ | $a_1$ | $b_{52}$ | $b_{13}$ | $b_{54}$ | $a_5$ |

| 属性 模式 | A | B | C | D | E |
|---|---|---|---|---|---|
| $R_1(AD)$ | $a_1$ | $b_{12}$ | $b_{13}$ | $a_4$ | $b_{15}$ |
| $R_2(AB)$ | $a_1$ | $a_2$ | $b_{13}$ | $b_{24}$ | $b_{25}$ |
| $R_3(BE)$ | $b_{31}$ | $a_2$ | $b_{13}$ | $b_{34}$ | $a_5$ |
| $R_4(CDE)$ | $b_{41}$ | $b_{42}$ | $a_3$ | $a_4$ | $a_5$ |
| $R_5(AE)$ | $a_1$ | $b_{52}$ | $b_{13}$ | $b_{54}$ | $a_5$ |

## example

- ④ 根据C→D，对上表进行处理， 由于属性列C上第1、2、3、5行相同均为b13，所以将属性列D上的值均改为同一个符号a4。

| 属性 / 模式 | A | B | C | D | E |
|---|---|---|---|---|---|
| $R_1(AD)$ | $a_1$ | $b_{12}$ | $b_{13}$ | $a_4$ | $b_{15}$ |
| $R_2(AB)$ | $a_1$ | $a_2$ | $b_{13}$ | $b_{24}$ | $b_{25}$ |
| $R_3(BE)$ | $b_{31}$ | $a_2$ | $b_{13}$ | $b_{34}$ | $a_5$ |
| $R_4(CDE)$ | $b_{41}$ | $b_{42}$ | $a_3$ | $a_4$ | $a_5$ |
| $R_5(AE)$ | $a_1$ | $b_{52}$ | $b_{13}$ | $b_{54}$ | $a_5$ |

$\Rightarrow$

| 属性 / 模式 | A | B | C | D | E |
|---|---|---|---|---|---|
| $R_1(AD)$ | $a_1$ | $b_{12}$ | $b_{13}$ | $a_4$ | $b_{15}$ |
| $R_2(AB)$ | $a_1$ | $a_2$ | $b_{13}$ | $a_4$ | $b_{25}$ |
| $R_3(BE)$ | $b_{31}$ | $a_2$ | $b_{13}$ | $a_4$ | $a_5$ |
| $R_4(CDE)$ | $b_{41}$ | $b_{42}$ | $a_3$ | $a_4$ | $a_5$ |
| $R_5(AE)$ | $a_1$ | $b_{52}$ | $b_{13}$ | $a_4$ | $a_5$ |

## example

- ⑤ 根据DE→C，对上表进行处理，由于属性列DE上第3、4、5行相同均为a4a5，所以将属性列C上的值均改为同一个符号a3。

| 模式　属性 | A | B | C | D | E |
|---|---|---|---|---|---|
| $R_1(AD)$ | $a_1$ | $b_{12}$ | $b_{13}$ | $a_4$ | $b_{15}$ |
| $R_2(AB)$ | $a_1$ | $a_2$ | $b_{13}$ | $a_4$ | $b_{25}$ |
| $R_3(BE)$ | $b_{31}$ | $a_2$ | $b_{13}$ | $a_4$ | $a_5$ |
| $R_4(CDE)$ | $b_{41}$ | $b_{42}$ | $a_3$ | $a_4$ | $a_5$ |
| $R_5(AE)$ | $a_1$ | $b_{52}$ | $b_{13}$ | $a_4$ | $a_5$ |

| 模式　属性 | A | B | C | D | E |
|---|---|---|---|---|---|
| $R_1(AD)$ | $a_1$ | $b_{12}$ | $b_{13}$ | $a_4$ | $b_{15}$ |
| $R_2(AB)$ | $a_1$ | $a_2$ | $b_{13}$ | $a_4$ | $b_{25}$ |
| $R_3(BE)$ | $b_{31}$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |
| $R_4(CDE)$ | $b_{41}$ | $b_{42}$ | $a_3$ | $a_4$ | $a_5$ |
| $R_5(AE)$ | $a_1$ | $b_{52}$ | $a_3$ | $a_4$ | $a_5$ |

## example

- ⑥ 根据CE→A，对上表进行处理，由于属性列CE上第3、4、5行相同均为a3a5，所以将属性列A上的值均改为同一个符号a1。

| 模式 \ 属性 | A | B | C | D | E |
|---|---|---|---|---|---|
| $R_1(AD)$ | $a_1$ | $b_{12}$ | $b_{13}$ | $a_4$ | $b_{15}$ |
| $R_2(AB)$ | $a_1$ | $a_2$ | $b_{13}$ | $a_4$ | $b_{25}$ |
| $R_3(BE)$ | $b_{31}$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |
| $R_4(CDE)$ | $b_{41}$ | $b_{42}$ | $a_3$ | $a_4$ | $a_5$ |
| $R_5(AE)$ | $a_1$ | $b_{52}$ | $a_3$ | $a_4$ | $a_5$ |

$\Rightarrow$

| 模式 \ 属性 | A | B | C | D | E |
|---|---|---|---|---|---|
| $R_1(AD)$ | $a_1$ | $b_{12}$ | $b_{13}$ | $a_4$ | $b_{15}$ |
| $R_2(AB)$ | $a_1$ | $a_2$ | $b_{13}$ | $a_4$ | $b_{25}$ |
| $R_3(BE)$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |
| $R_4(CDE)$ | $a_1$ | $b_{42}$ | $a_3$ | $a_4$ | $a_5$ |
| $R_5(AE)$ | $a_1$ | $b_{52}$ | $a_3$ | $a_4$ | $a_5$ |

- 存在某一行为$a_1,a_2,\ldots,a_k$，则分解具有无损连接性

## 保持函数依赖分解(1)
### Dependency-Preserving Decomposition

Example: Suppose R(City, Street, Zipcode),

$F = \{CS \longrightarrow Z, Z \longrightarrow C\}$, R1(S, Z), R2(C, Z).

$\pi_{R1}(F) = \{S \longrightarrow S,\ Z \longrightarrow Z,\ SZ \longrightarrow S,$
$SZ \longrightarrow Z,\ SZ \longrightarrow SZ\}$

$\pi_{R2}(F) = \{Z \longrightarrow C,\ C \longrightarrow C,\ Z \longrightarrow Z,$
$CZ \longrightarrow C,\ CZ \longrightarrow Z,\ CZ \longrightarrow CZ\}$

定义:对关系 R 和函数依赖F, 分解 $\{R_1, R_2, ..., R_n\}$ 保持函数依赖,如果满足

$$F^+ = (F_1 \cup F_2 \cup . . . \cup F_n)^+$$

where $F_i = \pi_{R_i}(F)$, $i = 1, ..., n$.

# 保持函数依赖分解(2)

在上面例子中, {R1, R2} 是R的一个分解.

因为 $CS \longrightarrow Z \in F^+$

但是 $CS \longrightarrow Z \notin (\pi_{R1}(F) \cup \pi_{R2}(F))^+$,

所以<u>这个分解不能保持函数依赖</u>.

## 保持函数依赖分解(3)

判定是否保持函数依赖分解

Algorithm DP

Input: A relation schema R, A set of FDs F in R, a decomposition $\{R_1, R_2, ..., R_n\}$ of R.

```
for every $X \longrightarrow Y \in F$
①      if $\exists R_i$ such that $XY \subseteq R_i$
       then $X \longrightarrow Y$ is preserved;
②      else use Algorithm XYGP to find W;
         if $Y \subseteq W$ then $X \longrightarrow Y$ is preserved;
if every $X \longrightarrow Y$ is preserved
   then $\{R_1, ..., R_n\}$ is dependency-preserving;
else $\{R_1, ..., R_n\}$ is not dependency-preserving;
```

## 保持函数依赖分解(4)

```
Algorithm XYGP
  W := X;
  repeat for i from 1 to n do
       $W := W \cup ((W \cap R_i)^+ \cap R_i)$;
在每个分解后的关系$R_i$中寻找X可以确定的属性集
  until **there is no change to W**;
```

## 保持函数依赖分解(5)

**Dependency-Preserving Decomposition**

**Example**: Suppose R(A, B, C, D),

F = {$A \longrightarrow B$, $B \longrightarrow C$, $C \longrightarrow D$, $D \longrightarrow A$ },

R1(A,B), R2(B,C), R3(C,D).

Is {R1, R2, R3} dependency-preserving?

Since $AB \subseteq R1$, $A \longrightarrow B$ is preserved.

Since $BC \subseteq R2$, $B \longrightarrow C$ is preserved.

Since $CD \subseteq R3$, $C \longrightarrow D$ is preserved.

## 保持函数依赖分解(6)

For $D \longrightarrow A$, use Algorithm XYGP to compute W.

Initialization: $W = D$;

first iteration:

$W = D \cup ((D \cap AB)^+ \cap AB) = D$;

$W = D \cup ((D \cap BC)^+ \cap BC) = D$;

$W = D \cup ((D \cap CD)^+ \cap CD)$

$= D \cup (D^+ \cap CD)$

$= D \cup (ABCD \cap CD) = CD$;

## 保持函数依赖分解(7)

second iteration:

$W = CD \cup ((CD \cap AB)^+ \cap AB) = CD;$

$W = CD \cup ((CD \cap BC)^+ \cap BC)$

$= CD \cup (C^+ \cap BC) = BCD;$

$W = BCD \cup ((BCD \cap CD)^+ \cap CD)$

$= BCD;$

third iteration:

$W = BCD \cup ((BCD \cap AB)^+ \cap AB)$

$= ABCD;$

Since $A \subseteq W$, D      A is also preserved.

Hence, {R1, R2, R3} is a dependency-preserving decomposition.

## Example 1:

Let Relation Schema $R = (A, B, C, D, E)$

*Let* $F = \{A \to BC, CD \to E, B \to D, E \to A\}$

1) *Let R be decomposed into* $R_1 = (A, B, C)$ *and* $R_2 = (A, D, E)$

   Is it a lossless-join decomposition?

   Is it a dependency-preserving decomposition?

2) *If* $R_a = (A, B, C)$ $R_b = (C, D, E)$

   Is it a lossless-join decomposition?

   Is it a dependency-preserving decomposition?

## Example 1

*Let R* = *(A, B, C, D, E) and F* = *{A->BC, CD->E, B->D, E->A}*

*Let R be decomposed inot* $R_1$= *(A, B, C) and* $R_2$= *(A, D, E)*

- Is it a lossless-join decomposition?

  $$R_1 \cap R_2 = \{A\}$$

- $R_1 - R_2 = \{B，C\}$
  $\{A\}$ is superkey of $R_1$, $A\text{->}BC$ so lossless-join
- Is it a dependency-preserving decomposition?
- No

## Example 1:

$R = (A, B, C, D, E)$ $F = \{A\text{->}BC, CD\text{->}E, B\text{->}D, E\text{->}A\}$

$R_a = (A, B, C)$ $R_b = (C, D, E)$

- Is it a lossless-join decomposition? $R_a \cap R_b = \{C\}$

  $R_a - R_b = \{A,B\}$

  $R_b - R_a = \{D,E\}$

$\{C\}$ is not superkey of $R_a$ *and* $R_b$, so lossy-join

- Is it a dependency-preserving decomposition?
- Yes

## Q & 谢谢

后面为补充自选学习材料

## 2.Functional Dependency函数依赖

**Definition**: Let R(A1, ..., An) be a relation schema. Let X and Y be two subsets of {A1, ..., An}. X is said to functionally determine Y (or Y is functionally dependent on X) if for every legal relation instance r(R), for any two tuples t1 and t2 in r(R), we have

$$t1[X] = t2[X] \longrightarrow t1[Y] = t2[Y].$$

## CON…

Two notations:

- $X \subset R$ denotes that X is a subset of the attributes of R.
- $X \longrightarrow Y$ denotes that X functionally determines Y.

## CON…

Several equivalent definitions:

- $X \longrightarrow Y$ in R $\Leftrightarrow$ for any t1, t2 in r(R), if t1 and t2 have the same X-value, then t1 and t2 also have the same Y-value.
- $X \longrightarrow Y$ in R $\Leftrightarrow$ there exist no t1, t2 in r(R) such that t1 and t2 have the same X-value but different Y-values.
- $X \longrightarrow Y$ in R $\Leftrightarrow$ for each X-value, there corresponds to a unique Y-value.

## CON…

Theorem 1: If X is a superkey of R and Y is any subset of R, then $X \longrightarrow Y$ in R.

- Note that $X \longrightarrow Y$ in R is a property that must be true for all possible legal r(R), not just for the present r(R).

## CON…

- *If X*$\rightarrow$*Y*， then X is called Determinant(决定条件).
- *If X*$\rightarrow$*Y*， *Y*$\rightarrow$*X*， then written as: *X*$\longleftrightarrow$*Y.*
- If *Y* NON-**Functional dependency on** *X*, then written as :$X \nrightarrow Y$.

## FULL FD and PARTIAL FD完全函数依赖与部分函数依赖

stricter definition of FD (vs. partial FD)

- y is fully functionally dependent on x if it is functionally dependent on all of x, not just on a subset
- {CNO, CNAME} $\rightarrow$ CLOCATION : partial FD

  CNO $\rightarrow$ CLOCATION: full FD
- {SSN, CNO} $\rightarrow$ HOURS: full FD

## Transitive Functional Dependency传递函数依赖

- Transitive Functional Dependency
  - y is transitively functionally dependent on x if x functionally determines z (not a candidate key or a subset) and z functionally determines y
  - x $\rightarrow$ y if x $\rightarrow$ z and z $\rightarrow$ y
  - e.g.)Stu(SSN,Sdept,Slocation)
  - SSN $\rightarrow$ Sdept and Sdept $\rightarrow$ Slocation, then SSN $\rightarrow$ Slocation(SSN transitively determines Slocation)

## Relational Database Theory

- Designing high quality tables.
- Remove redundancy!
- Decomposition!
- How can decomposition be no error ?
- How to determine the primary key?

## Relational Database Theory

- Relation Decomposition
  - Lossless Join Decomposition
  - Dependency-Preserving Decomposition
  - Finding Candidate Keys from FDs

  Before that ….
- Identify Functional Dependency
- Consider all the implied information

# Identify Functional Dependency(1)

- Trivial FDs（平凡函数依赖）：
- if $Y \subseteq X \subseteq R$, then $X \longrightarrow Y$.
  - A special case: for any attribute A, $A \longrightarrow A$.
- Use Theorem 1: If X is a superkey of R and Y is any subset of R, then $X \longrightarrow Y$ is in R.

## Identify Functional Dependency(2)

- Created by assertions.

  Employees(SSN, Name, Years_of_emp, Salary, Bonus)

- Assertion: Employees hired the same year have the same salary.

This assertion implies:

$$\text{Years\_of\_emp} \longrightarrow \text{Salary}$$

## Identify Functional Dependency(3)

- Analyze the semantics of attributes of R.

  Addresses(City, Street, Zipcode)

  $\text{Zipcode} \longrightarrow \text{City}$

- Derive new FDs from existing FDs.

  Let R(A, B, C), F = {$A \longrightarrow B$, $B \longrightarrow C$}.

  $A \longrightarrow C$ can be derived from F.

  Denote F logically implies $A \longrightarrow C$ by

  $F \models A \longrightarrow C$.

基于**F**所蕴含的关系，可以确定所有的函数依赖，可以确定码。

## Identify Functional Dependency(4)

- Definition: Let F be a set of FDs in R. The closure（闭包） of F is the set of all FDs that are logically implied by F.
- The closure of F is denoted by $F^+$. F的闭包

$$F^+ = \{ X \longrightarrow Y \mid F \models X \longrightarrow Y \}$$

## Identify Functional Dependency(5)

- A BIG $F^+$ may be derived from a small F.

Example: For R(A, B, C) and

$$F = \{A \longrightarrow B,\ B \longrightarrow C\}$$

$$F^+ = \{\ A \longrightarrow B,\ B \longrightarrow C,\ A \longrightarrow C,$$
$$A \longrightarrow A,\ B \longrightarrow B,\ C \longrightarrow C,$$
$$AB \longrightarrow AB,\ AB \longrightarrow A,\ AB \longrightarrow B,\ \ldots\ \}$$

$|F^+| > 30.$

*F* $^+$是个很大的集合，如何推理？Armstrong公理

## Computation of F+ (1)

Armstrong‘s Axioms (1974): 公理系统

(IR1) Reflexivity rule自反律:

If $X \supseteq Y$, then $X \longrightarrow Y$.

(IR2) Augmentation rule增广律:

$\{ X \longrightarrow Y \} \models XZ \longrightarrow YZ.$

(IR3) Transitivity rule传递律:

$\{ X \longrightarrow Y, Y \longrightarrow Z \} \models X \longrightarrow Z.$

## Computation of F+ (2)

- Theorem: Armstrong's Axioms are sound and complete.
- Sound --- no incorrect FD can be generated from F using Armstrong's Axioms.
- Complete --- Given a set of FDs F, all FDs in $F^+$ can be generated using Armstrong's Axioms.

## Computation of F+ (3)

Additional rules derivable from Armstrong‘s Axioms 推理规则.

(IR4) Decomposition rule分解规则:

If $X \rightarrow Y$ and $Z \subseteq Y$ then $X \rightarrow Z$

(IR5) Union rule合并规则:

$\{ X \longrightarrow Y, X \longrightarrow Z \} \models X \longrightarrow YZ$

(IR6) Pseudo transitivity rule伪传递规则:

$\{ X \longrightarrow Y, WY \longrightarrow Z \} \models WX \longrightarrow Z$

## Computation of F+

(IR5): $\{ X \longrightarrow Y, X \longrightarrow Z \} \models X \longrightarrow YZ$

Proof:

by $X \longrightarrow Y$ and (IR2增广律):

$XX \longrightarrow XY$, i.e., $X \longrightarrow XY$;

by $X \longrightarrow Z$ and (IR2): $XY \longrightarrow ZY$;

by $X \longrightarrow XY$, $XY \longrightarrow ZY$ and (IR3传递律):

$X \longrightarrow YZ$.

## Computation of F+

(IR6): $\{ X \longrightarrow Y, WY \longrightarrow Z \} \models WX \longrightarrow Z$

Proof:

by $X \longrightarrow Y$ and (IR2): $XW \longrightarrow YW$;

by $WX \longrightarrow WY$, $WY \longrightarrow Z$ and (IR3):

$WX \longrightarrow Z$.

Claim: If $X \subseteq R$ and A, B, ..., C are attributes in R, then $X \longrightarrow A\ B \ldots C$

$\equiv \{ X \longrightarrow A, X \longrightarrow B, \ldots, X \longrightarrow C \}$

## Closure of Attributes 属性的闭包

How to determine if $F \models X \longrightarrow Y$ is true?

Method 1: Compute $F^+$. If $X \longrightarrow Y \in F^+$,

then $F \models X \longrightarrow Y$; else $F \not\models X \longrightarrow Y$.

**Problem**: Computing $F^+$ could be very expensive!

Consider $F = \{ A \longrightarrow B_1, \ldots, A \longrightarrow B_n \}$.

Claim: $|F^+| > 2^n$.

Reason: $\{A \longrightarrow X \mid X \subseteq \{B_1, \ldots, B_n\}\} \subseteq F^+$.

## Closure of Attributes 属性的闭包

Method 2: Compute $X^+$ : the closure of X under F.

$X^+$ denotes the set of attributes that are functionally determined by X under F.

$$X^+ = \{ A \mid X \longrightarrow A \in F^+ \}$$

Theorem: $X \longrightarrow Y \in F^+$ if and only if $Y \subseteq X^+$.

## Algorithm for Computing X+

- compute $X^+$ by:

```
closure := X;
repeat until no change {
        if there is an FD U → V in F
            such that U is in closure
                then add V to closure}
```

## Algorithm for Computing X+

Theorem: Given $R(A_1, ..., A_n)$ and a set of FDs F in R, $K \subseteq R$ is a

- **superkey** if $K^+ = \{A_1, ..., A_n\}$;
- **candidate key** if K is a superkey and for any proper subset X of K, $X^+ \neq \{A_1, ..., A_n\}$.

## Algorithm for Computing X+

Continue the above example:

- AB is a superkey of R since $(AB)^+$ = ABCDE.
- Since $A^+ = A$, $B^+ = BD$, neither A nor B is a superkey.
- Hence, AB is a candidate key

## Finding Candidate Keys from FDs

Let F be a set of FDs in relation schema $R(A_1, ..., A_n)$.

**Method 1** (can be automated)

```
(1) for each $A_i$, compute $A_i^+$;
      if $A_i^+ = A_1 A_2 ... A_n$
          then $A_i$ is a candidate key;
```

## Finding Candidate Keys from FDs

```
(2) for each pair $A_iA_j$, $i \neq j$
        if $A_i$ or $A_j$ is a candidate key
            then $A_iA_j$ is not a candidate key;
        else compute $(A_i A_j)^+$;
            if $(A_iA_j)^+ = A_1 A_2 \ldots A_n$
                then $(A_i A_j)$ is a candidate key;
```

## Finding Candidate Keys from FDs

```
(3) for each triple $A_iA_jA_k$, $i \neq j$, $i \neq k$, $j \neq k$
      if any subset of $A_iA_jA_k$ is a candidate key
         then $A_iA_jA_k$ is not a candidate key;
      else compute $(A_iA_jA_k)^+$;
            if $(A_iA_jA_k)^+ = A_1\ A_2\ ...\ A_n$
               then $(A_iA_jA_k)$ is a candidate key;
(4) . . . . . .
```

## 3.Relation Decomposition

- Relation Decomposition
  - Lossless Join Decomposition
  - Dependency-Preserving Decomposition

## Relation Decomposition

**Definition**: Let R be a relation schema. A set of relation schemas $\{R_1, R_2, ..., R_n\}$ is a decomposition of R if $R = R_1 \cup ... \cup R_n$.

**Claim**: If $\{R_1, R_2, ..., R_n\}$ is a decomposition of R and r is an instance of R, then

$$r \subseteq \pi_{R1}(r) \infty \pi_{R2}(r) \infty \; . \; . \; . \infty \pi_{Rn}(r)$$

Information may be lost (i.e. wrong tuples may be added) due to a decomposition.

## Lossless Join Decomposition

**Definition**: $\{R_1, R_2, ..., R_n\}$ is a lossless (non-additive) join decomposition of R if for every legal instance r of R, we have

$$r = \pi_{R1}(r) \infty \pi_{R2}(r) \infty \; . \; . \; . \infty \pi_{Rn}(r)$$

## Lossless Join Decomposition

Theorem: Let R be a relation schema and F be a set of FDs in R. Then a decomposition of R, {R1, R2}, is a lossless-join decomposition if and only if

- $R1 \cap R2 \longrightarrow R1 - R2$; or
- $R1 \cap R2 \longrightarrow R2 - R1$.

## Lossless Join Decomposition

Example:

Consider:

Prod_Manu(Prod_no, Prod_name, Price, Manu_id, Manu_name, Address)

F = { P# $\longrightarrow$ Pn Pr Mid, Mid $\longrightarrow$ Mn A },

## Lossless Join Decomposition

- Solution:

Decomposition: {Products=P# Pn Pr Mid,

Manufacturers=Mid Mn A

}

- Is it a loss less join?

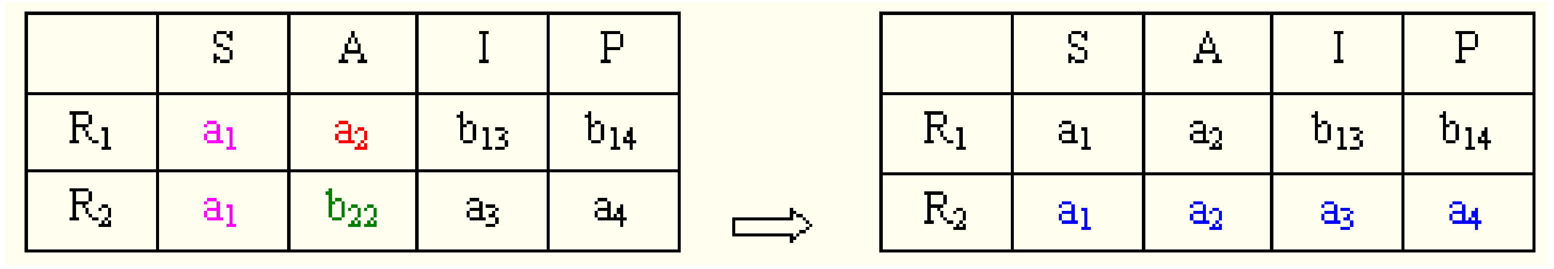
Since Products ∩ Manufacturers = Mid

Mn A = Manufacturers - Products,

it is a lossless-join decomposition.

## example

- 举例1：关系模式 R(SAIP),F={S→A,SI→P},ρ={R1(SA),R2(SIP)}，检验分解是否为无损联接?
-



| | S | A | I | P |
|---|---|---|---|---|
| $R_1$ | $a_1$ | $a_2$ | $b_{13}$ | $b_{14}$ |
| $R_2$ | $a_1$ | $b_{22}$ | $a_3$ | $a_4$ |

$\Longrightarrow$

| | S | A | I | P |
|---|---|---|---|---|
| $R_1$ | $a_1$ | $a_2$ | $b_{13}$ | $b_{14}$ |
| $R_2$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ |

## Dependency-Preserving Decomposition (1)

**Definition**: Let R be a relation schema and F be a set of FDs in R. For any R' $\subseteq$ R, the restriction of F to R' is a set of all FDs F' in $F^+$ such that each FD in F' contains only attributes of R'.

$F' = \pi_{R'}(F) = \{ X \longrightarrow Y \mid F \models X \longrightarrow Y \text{ and } XY \subseteq R' \}$

Note: $\pi_{R'}(F) = \pi_{R'}(F^+)$!

## Dependency-Preserving Decomposition (2)

Example: Suppose R(City, Street, Zipcode),

F = {$CS \rightarrow Z$, $Z \rightarrow C$}, R1(S, Z), R2(C, Z).

$\pi_{R1}(F) = \{S \rightarrow S,\ Z \rightarrow Z,\ SZ \rightarrow S,$

$SZ \rightarrow Z,\ SZ \rightarrow SZ\}$

$\pi_{R2}(F) = \{Z \rightarrow C,\ C \rightarrow C,\ Z \rightarrow Z,$

$CZ \rightarrow C,\ CZ \rightarrow Z,\ CZ \rightarrow CZ\}$

## Dependency-Preserving

**Definition**: Given a relation schema R and a set of FDs F in R, a decomposition of R, $\{R_1, R_2, ..., R_n\}$, is dependency-preserving if

$$F^+ = (F_1 \cup F_2 \cup . . . \cup F_n)^+$$

where $F_i = \pi_{R_i}(F)$, $i = 1, ..., n$.

## Dependency-Preserving Decomposition CON…

In the above example, {R1, R2} is a decomposition of R.

Since CS $\longrightarrow$ Z $\in F^+$ but

CS $\longrightarrow$ Z $\notin (\pi_{R1}(F) \cup \pi_{R2}(F))^+$,

the decomposition is not dependency-preserving.

## Dependency-Preserving Decomposition CON…

```
for every $X \rightarrow Y \in F$
  if $\exists R_i$ such that $XY \subseteq R_i$
      then $X \rightarrow Y$ is preserved;
  else use Algorithm XYGP to find W;
      if $Y \subseteq W$ then $X \rightarrow Y$ is preserved;
if every $X \rightarrow Y$ is preserved
  then $\{R_1, ..., R_n\}$ is dependency-preserving;
else $\{R_1, ..., R_n\}$ is not dependency-preserving;
```