数据库系统原理

陈岭

浙江大学计算机学院

9 函数依赖和关系模式分解

- 第一范式
- 关系数据库设计中易犯的错误
- Armstrong公理系统
- 函数依赖理论
- 关系模式分解

第一范式

- 如果某个域中元素被认为是不可分的，则这个域称为是原子的
  - 非原子域的例子：
    - 复合属性：名字集合
    - 多值属性：电话号码
    - 复杂数据类型：面向对象的
- 如果关系模式R的所有属性的域都是原子的，则R称为属于第一范式（1NF）
- 非原子值存储复杂并易导致数据冗余
  - 我们假定所有关系都属于第一范式

第一范式

- 如何处理非原子值
  - 对于组合属性：让每个子属性本身成为成为一个属性
  - 对于多值属性：为多值集合中的每个项创建一条元组
- 原子性实际上是由域元素在数据库中如何被使用决定的
  - 例，字符串通常会被认为是不可分割的
  - 假设学生被分配这样的标识号：CS0012或EE1127，如果前两个字母表示系，后四位数字表示学生在该系内的唯一号码，则这样的标识号不是原子的
  - 当采用这种标识号时，是不可取的。因为这需要额外的编程，而且信息是在应用程序中而不是在数据库中编码

关系数据库设计中易犯的错误

- 关系数据库设计要求我们找到一个“好的”关系模式集合。一个坏的设计可能导致
  - 数据冗余
  - 插入、删除、修改异常
  - 假设，我们用以下模式代替*instructor*模式和*department*模式：

    inst_dept(ID, name, salary, dept_name, building, budget)

关系数据库设计中易犯的错误

- 数据冗余
  - 每个系的*dept_name*, *building*, *budget*数据都要重复一次
  - 缺点：浪费空间，可能会导致不一致问题

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

关系数据库设计中易犯的错误

- 更新异常
  - 更新复杂，容易导致不一致问题。例，修改*dept_name*，很多相关元组都需要修改
- 插入/删除异常
  - 使用空值(null)：存储一个不知道所在系的教师信息，可以使用空值表示*dept_name*, *building*, *budget*数据，但是空值难以处理

模式分解

- 模式分解
  - 例，可以将关系模式(A, B, C, D)分解为：(A, B)和(B, C, D)，或(A, C, D)和(A, B, D)，或(A, B, C)和(C, D)，或(A, B)、(B, C)和(C, D)，或(A, D)和(B, C, D)
  - 例，将关系模式*inst_dept*分解为：
    - *instructor*(*ID, name, dept_name, salary*)
    - *department*(*dept_name, building, budget*)
- 原模式(R)的所有属性都必须出现在分解后的(R_1, R_2)中：$R = R_1 \cup R_2$
- 无损连接分解
  - 对关系模式R上的所有可能的关系r

$$r = \prod_{R1}(r) \bowtie \prod_{R2}(r)$$

模式分解

- 例，分解$R = (A, B)$，得到$R_1 = (A)$和$R_2 = (B)$

$r =$

A	B
α	1
α	2
β	1

A
α
β

$\Pi_A(r)$

B
1
2

$\Pi_B(r)$

$\Pi_A(r) \bowtie \Pi_B(r) =$

A	B
α	1
α	2
β	1
β	2

$\neq$ r

不好的分解！

因为它不是无损连接分解，是非法的

模式分解

- 目标：设计一个理论
  - 以判断关系模式R 是否为“好的”形式（不冗余）
  - 当R 不是“好的”形式时，将它分解成模式集合$\{R_1, R_2, \ldots, R_n\}$使得
    - 每个关系模式都是“好的”形式
    - 分解是无损连接分解
  - 我们的理论基于：
    - 函数依赖（ functional dependencies ）
    - 多值依赖（ multivalued dependencies ）

函数依赖

- 设R是一个关系模式，且有属性集$\alpha \subseteq R$，$\beta \subseteq R$
- 函数依赖

$$\alpha \rightarrow \beta$$

借用了数学上的函数概念：
x → f(x)

在R上成立当且仅当对任意合法关系$r(R)$，若r的任意两条元组t_1和t_2在属性集α上的值相同，则他们在属性集β上的值也相同。即，

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- β函数依赖于α，α函数决定β

α	β	γ
a	f	1
b	h	2
a	f	3
c	f	4

函数依赖

- 函数依赖：一种完整性约束，表示特定的属性值之间的关系，可以用来判断模式规范化和建议改进
- 例，考虑$r(A, B)$及其下列实例r

A	B
1	4
1	5
3	7

  - 对此实例，$A \rightarrow B$不成立，但$B \rightarrow A$成立

∵若B属性值确定了，则A属性值也唯一确定了。于是有 $B \rightarrow A$

函数依赖

- 函数依赖是码概念的推广
- K是关系模式R的超码当且仅当$K \rightarrow R$
- K是R的候选码当且仅当
  - $K \rightarrow R$，并且
  - 没有$\alpha \subset K$，使$\alpha \rightarrow R$（不存在K的真子集α，使之满足$\alpha \rightarrow R$）

函数依赖

- 函数依赖使我们可以表达用超码无法表达的约束，考虑模式

  $inst\_dept(\underline{ID}, name, salary, \underline{dept\_name}, building, budget)$

  - 我们期望下列函数依赖成立：

    $$dept\_name \rightarrow building$$
    $$ID \rightarrow building$$

  - 而不期望下列函数依赖成立：

    $$dept\_name \rightarrow salary$$

函数依赖的使用

- 我们用函数依赖来：
  - 检查关系在给定函数依赖之下是否合法
    - 若关系r在函数依赖集F下是合法的，则称r满足F

r =

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

F＝{

A→C

AB →D ⇒ {A, B} →D

ABC →D}

但，

A ↛ B, A ↛ D, B ↛ A, C ↛ A, C ↛ D,
B ↛ C, C ↛ B, B ↛ D, …

函数依赖的使用

- 对合法关系集合指定约束
  - 如果R上的所有合法关系都满足F, 则称F在R上成立

$r_1(R)=$

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

F＝{

A→C,

AB →D,

ABC →D

}

$r_2(R)=\cdots$

$r_3(R)=\cdots$

……

注：容易判别一个r是否满足给定的F。不易判别F是否在R上成立。不能仅由某个r推断出F。R上的函数依赖F, 通常由定义R的语义决定

函数依赖

- 被所有关系实例都满足的函数依赖称为平凡的
  - 例，$A \rightarrow A$，$AB \rightarrow A$

    $(ID, name) \rightarrow ID$

    $ID \rightarrow ID$
  - 一般地，若$\beta \subseteq \alpha$，则$\alpha \rightarrow \beta$是平凡的。即，

    平凡的函数依赖：若$\beta \subseteq \alpha$，$\alpha \rightarrow \beta$

    非平凡的函数依赖：若$\beta \nsubseteq \alpha$，$\alpha \rightarrow \beta$

函数依赖集的闭包

- 给定函数依赖集F，存在其他函数依赖被F逻辑蕴含
  - 例，如果$A \rightarrow B$且$B \rightarrow C$，则可推出$A \rightarrow C$
- 被F逻辑蕴含的全体函数依赖的集合称为F的闭包，用F^+表示F的闭包
  - 例，$F = \{ A \rightarrow B,\ B \rightarrow C \}$，$F^+ = \{ A \rightarrow B,\ B \rightarrow C,\ A \rightarrow C,\ A \rightarrow A,\ AB \rightarrow A,\ AB \rightarrow B,\ AC \rightarrow C,\ A \rightarrow BC,\ \cdots \}$
- 如何计算出F^+
  - 例，$R = (A,\ B,\ C,\ G,\ H,\ I)$

$$F = \{ A \rightarrow B,\ A \rightarrow C,\ CG \rightarrow H,\ CG \rightarrow I,\ B \rightarrow H \}$$，那么 $F^+ = ?$

Armstrong公理

- 可利用Armstrong公理找出F^+：
  - 若$\beta \subseteq \alpha$，则$\alpha \rightarrow \beta$(自反律)
  - 若$\alpha \rightarrow \beta$，则$\gamma\alpha \rightarrow \gamma\beta$(增补律)
  - 若$\alpha \rightarrow \beta$且$\beta \rightarrow \gamma$，则$\alpha \rightarrow \gamma$(传递律)
- Armstrong公理是
  - 正确有效的(不会产生错误的函数依赖)
  - 完备的(产生所有成立的函数依赖即F^+)

Armstrong公理

- 例，$R = (A, B, C, G, H, I)$

  $F = \{A \rightarrow B$

  $A \rightarrow C$

  $CG \rightarrow H$

  $CG \rightarrow I$

  $B \rightarrow H\}$

F^+的某些成员：

- $A \rightarrow H$：根据传递规则，由$A \rightarrow B$和$B \rightarrow H$得到
- $AG \rightarrow I$：用G增补$A \rightarrow C$得$AG \rightarrow CG$，再由$CG \rightarrow I$根据传递规则得到
- $CG \rightarrow HI$：由$CG \rightarrow H$和$CG \rightarrow I$，可根据函数依赖的定义导出“并规则”得到，或增补$CG \rightarrow I$得到$CG \rightarrow CGI$，增补$CG \rightarrow H$得到$CGI \rightarrow HI$，再利用传递规则得到

Armstrong公理的补充定律

- 可用下列规则进一步简化F^+的手工计算
  - 若$\alpha \to \beta$与$\alpha \to \gamma$成立，则$\alpha \to \beta\gamma$成立（合并律）
  - 若$\alpha \to \beta\gamma$成立，则$\alpha \to \beta$与$\alpha \to \gamma$成立（分解律）
  - 若$\alpha \to \beta$与$\gamma\beta \to \delta$成立，则$\alpha\gamma \to \delta$成立（伪传递律）
- 以上规则可以从Armstrong公理推出
  - 例，考虑到$\alpha \to \beta\gamma$，根据自反律可得到：$\beta\gamma \to \beta$，$\beta\gamma \to \gamma$；再由传递律可得到：$\alpha \to \beta$与$\alpha \to \gamma$成立

计算F^+

- 下列过程计算函数依赖集F的闭包：

```
F+ = F
repeat
    for each  F+中的函数依赖f
        对f应用自反律和增补律
        将结果函数依赖加入F+
    for each  F+中的一对函数依赖f1 和f2
        if  f1和f2可以使用传递律结合起来
            将结果函数依赖加入F+
until F+ 不再变化
```

- 由于包含n个元素的集合含有个2^n子集，因此共有$2^n X 2^n$个可能的函数依赖
- 后面会介绍完成此任务的另一过程

属性集的闭包

- 如何判断集合α是否为超码
  - 一种方法是：计算F^+，在F^+中找出所有$\alpha \rightarrow \beta_i$，检查{ β_1 β_2 β_3 … }= R。但是这么做开销很大，因为F^+可能很大
  - 另一种方法是：计算α的闭包
- 定义：给定一个属性集α，在函数依赖集F 下由α函数确定的所有属性的集合为F 下α的闭包(记做α^+)
  - 检查函数依赖$\alpha \rightarrow \beta$是否属于F^+ $\Leftrightarrow$ $\beta \subseteq \alpha^+$
  - 判断α是否为超码： $\alpha \rightarrow R$属于F^+ $\Leftrightarrow$ $R \subseteq \alpha^+$

属性集的闭包

- 计算α^+的算法

```
result := a;
while(result有变化) do
  for each β → γ in F do
          begin
                  if β ⊆ result then result := result ∪ γ
          end
  a+ := result
```

避免了找F^+(反复使用公理)的麻烦

属性集的闭包

- 例1，$R = (A, B, C, G, H, I)$

  $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$

- $(AG)^+$
  - result = AG
  - result = $ABCG$ ($A \rightarrow C$ and $A \rightarrow B$)
  - result = $ABCGH$ ($CG \rightarrow H$ and $CG \subseteq AGBC$)
  - result = $ABCGHI$ ($CG \rightarrow I$ and $CG \subseteq AGBCH$)

属性集的闭包

- *AG*是候选码吗?
  - *AG*是超码吗？
    - 即，AG $\rightarrow$ R？由于$(AG)^+ \supseteq$ R，所以AG是超码
  - 存在*AG*的子集是超码吗?
    - $A^+ \rightarrow R$? 由于$(A)^+ = ABCH$，所以$(A)^+ \not\supseteq R$，所以A不是超码
    - $G^+ \rightarrow R$? 由于$(G)^+ = G$，所以$(G)^+ \not\supseteq R$，所以G不是超码
  - 综上，*AG*是候选码

属性集的闭包

- 例2，$R=(A,\ B,\ C)$，$F=\{A\rightarrow B,\ BC\rightarrow A\}$，$R$的候选码是什么？

  $\because (BC)^+=(BCA)\supseteq R,\ (AC)^+=(ACB)\supseteq R,\ (AB)^+=(AB)\supseteq R$

  $\therefore$候选码是AC，BC

属性闭包的用法

- 属性闭包算法有多种用途：
  - 测试超码（$\alpha \rightarrow R$?）
    - 为检测α是否超码，可计算α^+ 并检查α^+ 是否包含R 的所有属性
  - 测试函数依赖（$\alpha \rightarrow \beta$?）
    - 为检测函数依赖$\alpha \rightarrow \beta$是否成立（即是否属于F^+），只需检查是否$\beta \subseteq \alpha^+$
    - 即，可计算α^+，并检查它是否包含β
    - 这个检查简单而高效，非常有用
  - 计算F的闭包（$F^+ = ?$）
    - 对每个 $\gamma \subseteq R$，计算γ^+，再对每个$S \subseteq \gamma^+$，输出函数依赖$\gamma \rightarrow S$

正则覆盖

- DBMS总是检查确保数据库更新不会破坏任何函数依赖。但如果*F*很大，其开销就会很大。因此我们需要简化函数依赖集
- 直观地说，*F*的正则覆盖（记做F_c）是指与*F*等价的“极小的”函数依赖集合
  - F_c中任何函数依赖都不包含无关属性
  - F_c中函数依赖的左半部都是唯一的
    - 例，$\alpha_1 \rightarrow \beta_1$，$\alpha_1 \rightarrow \beta_2$，$\Rightarrow \alpha_1 \rightarrow \beta_1\beta_2$

正则覆盖

- 如何计算F_c：删除多余属性，存在以下三种情况
  - 函数依赖集中存在可由其他函数依赖推导出的函数依赖
    - 例，在F中 A $\rightarrow$ C是冗余的

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

      $\Rightarrow$ $F_c = \{A \rightarrow B,\ B \rightarrow C\}$
  - 函数依赖左边部分存在属性冗余
    - 例，$F = \{A \rightarrow B,\ B \rightarrow C,\ AC \rightarrow D\}$，即$\{A \rightarrow B,\ B \rightarrow C,\ AC \rightarrow D,\ A \rightarrow D\}$ $\Rightarrow$ $\{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow D\}$。所以$F$ 蕴涵F' = $\{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow D\}$，因此属性$C$ 是多余的

由F: $B \rightarrow C$, => $AB \rightarrow AC$, 又 $\because AC \rightarrow D$, $\therefore$ $AB \rightarrow D$; 又$\because A \rightarrow B$, => $A \rightarrow AB$, $\therefore$ $A \rightarrow D$, $\therefore F$ 蕴涵F'

由Armstrong公理, $A \rightarrow D$ 蕴涵 $AC \rightarrow D$

正则覆盖

- 函数依赖右边部分存在属性冗余
- 例，$F = \{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow CD\}$，即$\{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow C,\ A \rightarrow D\}$，但$A \rightarrow C$ 可由$A \rightarrow B$ 和$B \rightarrow C$ 得到。所以F 蕴涵$F' = \{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow D\}$，因此属性$C$ 是多余的

无关属性

- 考虑函数依赖集合F 及其中的函数依赖$\alpha \rightarrow \beta$
  - 如果$A \in \alpha$ 并且F 逻辑蕴含$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$，则称属性A 在$\alpha$ 中是无关的
    - 例，给定$F = \{A \rightarrow C,\ AB \rightarrow C\}$

      B 在$AB \rightarrow C$ 中是无关的，因为$A \rightarrow C$ 逻辑蕴含$AB \rightarrow C$
  - 如果$A \in \beta$ 并且$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$逻辑蕴含$F$，则称属性$A$ 在β 中是无关的
    - 例，给定 $F = \{A \rightarrow C,\ AB \rightarrow CD\}$

      C 在$AB \rightarrow CD$ 中是无关的，因为即使删除C 也能推出$A \rightarrow C$

检测属性是否无关

- 为检测属性$A \in \alpha$ 在α 中是否无关
  - 计算在F 下的$(\alpha - \{A\})^+$
  - 检查$(A - \{\alpha\})^+$ 是否包含β。如果是，则A 是无关的
- 为检测属性$A \in \beta$ 在β中是否无关
  - 计算在$F' = (F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$下的$\alpha^+$
  - 检查α^+ 是否包含A。 如果是，则A 是无关的

$\alpha=\{A\alpha'\}$，$\{A\alpha'\} \to \beta$。若F 蕴涵$\alpha' \to \beta$，则A 多余。故只要证明$\beta \in (\alpha')^+$

$\beta=\{A\beta'\}$，$\alpha \to \{A\beta'\}$。若F'蕴涵$\alpha \to A$，则A可删。故只要在F'下证明$A \in (\alpha)^+$

正则覆盖

- 计算F 的正则覆盖

  repeat

  对F 中的依赖利用合并规则

  $\alpha_1 \rightarrow \beta_1$ 和 $\alpha_1 \rightarrow \beta_2$ 替换成 $\alpha_1 \rightarrow \beta_1 \beta_2$

  找出含有无关属性的函数依赖$\alpha \rightarrow \beta$（在α 或β 中）

  如果找到无关的属性，从$\alpha \rightarrow \beta$ 中删去

  until F 不再变化

  - 注：删除某些无关的属性之后，可能导致合并规则可以使用，所以必须重新应用

正则覆盖

- 例，$R = (A, B, C)$

  $F = \{A \rightarrow BC$

  $B \rightarrow C$

  $A \rightarrow B$

  $AB \rightarrow C\}$

- 合并$A \rightarrow BC$ 及$A \rightarrow B$ 得到$A \rightarrow BC$
  - 集合变成 $\{A \rightarrow BC,\ B \rightarrow C,\ AB \rightarrow C\}$
- A 在$AB \rightarrow C$ 中是无关的，因为$B \rightarrow C$ 逻辑蕴含$AB \rightarrow C$
  - 集合变成 $\{A \rightarrow BC,\ B \rightarrow C\}$
- C 在$A \rightarrow BC$ 中是无关的，因为$A \rightarrow BC$ 可由$A \rightarrow B$ 和$B \rightarrow C$ 逻辑推出
- 正则覆盖是：$F_C = \{A \rightarrow B,\ B \rightarrow C\}$

模式分解

- 规范化的目标
  - 以判断关系模式R 是否为“好的”形式（不冗余，无插入、删除、更新异常）
  - 当R 不是“好的”形式时，将它分解成模式集合$\{R_1, R_2, \ldots, R_n\}$使得
    - 每个关系模式都是“好的”形式
    - 分解是无损连接分解
    - 分解是保持依赖

模式分解

- 分解应有的特性：
- 1. 原模式（R）的所有属性都必须出现在分解后的（R_1，R_2）中：$R = R_1 \cup R_2$
- 2. 无损连接分解
  - 对关系模式R上的所有可能的关系r

$$r = \prod_{R1}(r) \bowtie \prod_{R2}(r)$$

  - R 分解成R_1 和R_2 是无损连接，当且仅当下列依赖中的至少一个属于F^+

$$R_1 \cap R_2 \rightarrow R_1$$
$$R_1 \cap R_2 \rightarrow R_2$$

无损连接分解的条件：

分解后的二个子模式的共同属性必须是R_1或R_2的码。（适用于一分为二的分解）

模式分解

- 3. 保持依赖
  - 有效地检查更新操作（以确保没有违反任何FD），允许分别验证子关系模式R_i，而不需要计算分解后的关系的连接
  - F 在R_i上的限定是： $F_i \subseteq F^+$， 即F^+ 中所有只包含R_i 中属性的函数依赖F_i 的集合
  - $(F_1 \cup F_2 \cup \cdots \cup F_n)^+ = F^+$ ，F_i是F^+ 中仅包含R_i属性的依赖集
- 4. 没有冗余
  - R_i最好满足BCNF或3NF（ BCNF和3NF将在下一课中讲解 ）

模式分解

- 例，$R = (A, B, C)$，$F = \{A \rightarrow B, B \rightarrow C\}$，有两种分解方式
  - 第一种方式：$R_1 = (A, B)$和$R_2 = (B, C)$
    - 无损连接分解：$R_1 \cap R_2 = \{B\}$并且$B \rightarrow C$，$\therefore (B)^+ = \{BC\} \supseteq R_2$
    - 保持依赖：对于R_1，有$F_1 = \{A \rightarrow B\}$；对于$R_2$，有$F_2 = \{B \rightarrow C\}$，$\therefore (F_1 \cup F_2)^+ = F^+$
  - 第二种方式：$R_1 = (A, B)$和$R_2 = (A, C)$
    - 无损连接分解：$R_1 \cap R_2 = \{A\}$ and $(A)^+ = \{AB\} \supseteq R_1$
    - 对于R_1，有$F_1 = \{A \rightarrow B\}$；对于$R_2$，有$F_2 = \{A \rightarrow C\}$，$(F_1 \cup F_2)^+ = \{A \rightarrow B, A \rightarrow C\}^+ \neq F^+$，在$R_1$，$R_2$中无法不通过计算$R_1 \bowtie R_2$，来检查 $B \rightarrow C$
      $\therefore$是非保持依赖

模式分解

- 为检查依赖$\alpha \rightarrow \beta$ 在R 到R_1 ，R_2 ，… ，R_n的分解中是否得到保持，可进行下面的简单测试

```
result = α
while(result 有变化) do
        for each 分解后的Ri
        t = (result ∩ Ri)+ ∩ Ri
        result = result ∪ t
```

对于F 中的某个$\alpha \rightarrow \beta$ ，投影到各个R_i 中，判别是否有某个R_i 能保持函数依赖$\alpha \rightarrow \beta$

  - 若$result$ 包含β 中的所有属性，则$\alpha \rightarrow \beta$ 得到保持
  - 若对F 中的每个$\alpha \rightarrow \beta$ 都能有一个R_i 满足函数依赖，则该分解保持依赖

总结

- 描述了原子域和第一范式的假设
- 给出了数据库设计中易犯的错误，这些错误包括信息重复和插入、删除、修改异常
- 介绍了函数依赖的概念，展示了如何用函数依赖进行推导
- 理解F^+，α^+，F_C
- 介绍了如何分解模式，一个有效的分解都必须是无损的
- 如果分解是保持依赖的，则给定一个数据库更新，所有的函数依赖都可以由单独的关系进行验证，无须计算分解后的关系的连接

谢谢！