Armstrong公理

- 例，$R = (A, B, C, G, H, I)$

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

F^+的某些成员：

- $A \to H$：根据传递规则，由$A \to B$和$B \to H$得到
- $AG \to I$：用G增补$A \to C$得$AG \to CG$，再由$CG \to I$根据传递规则得到
- $CG \to HI$：由$CG \to H$和$CG \to I$，可根据函数依赖的定义导出“并规则”得到，或增补$CG \to I$得到$CG \to CGI$，增补$CG \to H$得到$CGI \to HI$，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$\alpha \rightarrow \beta$与$\alpha \rightarrow \gamma$成立，则$\alpha \rightarrow \beta\gamma$成立（合并律）
  - 若$\alpha \rightarrow \beta\gamma$成立，则$\alpha \rightarrow \beta$与$\alpha \rightarrow \gamma$成立（分解律）
  - 若$\alpha \rightarrow \beta$与$\gamma\beta \rightarrow \delta$成立，则$\alpha\gamma \rightarrow \delta$成立（伪传递律）
- 以上规则可以从Armstrong公理推出
  - 例，考虑到$\alpha \rightarrow \beta\gamma$，根据自反律可得到：$\beta\gamma \rightarrow \beta$，$\beta\gamma \rightarrow \gamma$；再由传递律可得到：$\alpha \rightarrow \beta$与$\alpha \rightarrow \gamma$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^nX2^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$\{\beta_1 \beta_2 \beta_3 \cdots\} = R$。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，$R = (A, B, C, G, H, I)$

  $$F = \{A \to B,\; A \to C,\; CG \to H,\; CG \to I,\; B \to H\}$$

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

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$\alpha+$ 并检查$\alpha+$ 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$\alpha+$，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，$\alpha_1 \to \beta_1$，$\alpha_1 \to \beta_2$， $\Rightarrow \alpha_1 \to \beta_1\beta_2$

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 → C是冗余的

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

$$\Rightarrow F_c = \{A \to B,\ B \to C\}$$

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

由F: $B \to C$, => $AB \to AC$, 又 ∵$AC \to D$, ∴ $AB \to D$; 又∵$A \to B$, => $A \to AB$,
∴ $A \to D$, ∴F 蕴涵F'
由Armstrong公理，$A \to D$ 蕴涵 $AC \to D$

正则覆盖

- 函数依赖右边部分存在属性冗余
- 例，$F = \{A \to B,\ B \to C,\ A \to CD\}$，即$\{A \to B,\ B \to C,\ A \to C,\ A \to D\}$，但$A \to C$ 可由$A \to B$ 和$B \to C$ 得到。所以F 蕴涵$F' = \{A \to B,\ B \to C,\ A \to D\}$，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，$R = (A, B, C)$

  $F = \{A \to BC$

  $B \to C$

  $A \to B$

  $AB \to C\}$

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

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, ..., 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，$R = (A, B, C)$， $F = \{A \to B, B \to C\}$，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$B \to C$， $\therefore (B)^+ = \{BC\} \supseteq R_2$
    - 保持依赖：对于R_1，有$F_1 = \{A \to B\}$；对于R_2，有$F_2 = \{B \to C\}$， $\therefore (F_1 \cup F_2)^+ = F^+$
  - 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$F_1 = \{A \to B\}$；对于R_2，有$F_2 = \{A \to C\}$，$(F_1 \cup F_2)^+ = \{A \to B, A \to C\}^+ \neq F^+$，在$R_1$，$R_2$中无法不通过计算$R_1 \bowtie R_2$，来检查 $B \to C$ $\therefore$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

  $result = \alpha$

  while($result$ 有变化) do

      for each 分解后的R_i

      $t = (result \cap R_i)^+ \cap R_i$

      $result = result \cup t$

  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！