

## 4. 关系数据库设计理论：函数依赖

**函数依赖**  $X \rightarrow Y$ ：如果两个元组在关系  $X$  中的属性相同，那么它们在关系  $Y$  中的属性必须相同（给定  $X$ ，“函数”值  $Y$  唯一确定）。

- 称为  $X$  函数决定  $Y$  或  $Y$  函数依赖于  $X$ ；
- 函数依赖是一个唯一值限制 (unique-value constraints)；
- 函数依赖等约束是对关系模式的限制（一般是根据语义确定的，而不是根据实例中的值确定的）；
- 如果  $Y \subseteq X$ ，那么  $Y$  依赖于  $X$  称为**平凡**的函数依赖；
- 表示：  $A_1, A_2, \dots, A_n \rightarrow B_1, B_2, \dots, B_m$ ；
  - 另一种常用的表示方法：  $X \rightarrow A_1, A_2, \dots, A_n$  等价于  $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n$ 。

**关系的键值**：  $K$  是关系  $R$  的键值当：**1**  $K \rightarrow$  all attributes of  $R$ ；**2**  $K$  是极小的，即没有  $K$  的真子集满足条件 **1**。

- 例如，在表 `SC` 中， `(sno, cno)` 是键值，而 `sno` 或 `cno` 均不是。

**超级键值** (superkey) 包含键值 (key) 的集合为超级键值 (superkey)。

**候选键值** (candidate key)：关系  $R$  可能有多个键值，他们都被称为候选键值。

**首选键值** (primary key)：可以从候选键值中选定一些称为首选键值，在首选键值下方划线表示。

**蕴含于**：函数依赖集合  $S$  蕴含于函数依赖集合  $T$ ；如果关系实例满足函数依赖集合  $T$ ，那么一定满足函数依赖集合  $S$ 。

- 函数依赖集合  $T$  “大”，所以关系实例的范围“小”。

**等价**：如果函数依赖集合  $S$  蕴含函数依赖集合  $T$ ，且函数依赖集合  $T$  蕴含函数依赖集合  $S$ ，则函数依赖  $S$  与函数依赖  $T$  等价。

- 分离规则：若  $x \rightarrow A_1, A_2, \dots, A_n$ ，则  $x \rightarrow A_1, x \rightarrow A_2, \dots, x \rightarrow A_n$ ；
- 组合规则：若  $x \rightarrow A_1, x \rightarrow A_2, \dots, x \rightarrow A_n$ ，则  $x \rightarrow A_1, A_2, \dots, A_n$ ；
- 平凡依赖关系：如果  $X \rightarrow Y$ ，则  $X \rightarrow (Y - X)$ （去掉平凡关系）；
  - 平凡：  $\forall i, A_i \in X$ ；
  - 非平凡：  $\text{some } A_i \in X$ ；
  - 完全非平凡：  $\forall i, A_i \notin X$ 。
- 传递规则：若  $X \rightarrow Y, Y \rightarrow Z$ ，则  $X \rightarrow Z$ 。

**闭包** (closure)：给定关系  $R$  和属性集合  $X$  以及函数依赖集合  $S$ ，那么  $X$  在  $S$  下的闭包是属性集合  $Y$ ，其中  $\{X \rightarrow Y\}$  蕴含于  $S$ ，记作  $X^+$ 。

- 平凡地，  $X \subseteq X^+$ 。

**一些问题**

- 如何判定  $\{X \rightarrow A\}$  是否蕴含于  $S$ ？

- **算法**：计算出  $X$  在  $S$  下的闭包  $X^+$ ；若是，则  $\{X \rightarrow A\}$  蕴含于  $S$ ；若否，则  $\{X \rightarrow A\}$  不蕴含于  $S$ 。
- $\{A_1, A_2, \dots, A_n\}$  是否是键值？
  - $\{A_1, A_2, \dots, A_n\}$  是超级键值当且仅当  $\{A_1, A_2, \dots, A_n\}^+$  包含了  $R$  的全部属性；
  - **算法**：先判断是否是超级键值；接着，若对于所有  $i$ ，令  $S = \{A_1, \dots, A_n\} - \{A_i\}$ ， $S^+$  不包含  $R$  的全部属性；则  $\{A_1, A_2, \dots, A_n\}$  为键值。
- 函数依赖的闭集 (closing set)? (给定函数依赖集合，找到其可以推导出的所有函数依赖)
  - **算法1**：找到属性集合所有子集的闭包；
  - **算法2**：利用阿姆斯特朗公理：对于任意属性集合  $X, Y, Z$ ，
    - 自反性：如果  $Y \subseteq X$ ，那么  $X \rightarrow Y$  在闭集中；
    - 增补性：如果  $X \rightarrow Y$ ，那么  $XZ \rightarrow YZ$  在闭集中；
    - 传递性：如果  $X \rightarrow Y$  且  $Y \rightarrow Z$ ，则  $X \rightarrow Z$  在闭集中。
  - 精简的结果： $X \rightarrow Y$ ，则可以省略  $Z \rightarrow Y$ ，其中  $Z \subseteq X$ 。
  - 函数依赖集合与自己的闭集等价。
- 找到函数依赖集合的**最小基 (minimal basis)**:
  - 给定一个关系  $R$  上的函数依赖集合  $S$ ，找到一个函数依赖集合  $T$  与  $S$  等价，使得  $T^+ = S^+$ ，则称  $T$  是  $S$  的基。
  - **最小基**：一个满足下面三个条件的基，
    - 所有函数依赖的右部分是单属性；
    - 其的任何非空真子集均不是基；
    - 没有函数依赖的左部分是多余的（例如  $AB \rightarrow C$  与  $A \rightarrow C$  应该去掉前者）。
  - **算法**
    - 分离所有依赖的右部分。
    - 不断尝试能否去掉一个函数依赖；
    - 不断尝试能否去掉函数依赖的左部分的属性。
- 找到函数依赖集合的**投影 (projection)**
  - 给定  $R$  上的函数依赖  $F$ ，找到函数依赖在  $R_1 = \pi_L(R)$  上的投影（最小基）  $F_1$ ，其中
    - $F_1$  蕴含于  $F$ ；
    - $F_1$  仅仅包含  $R_1$  中的属性。
  - **算法**:
    - 对于所有  $R_1$  中属性集合的子集  $X$ ，计算  $X^+$  在  $F$  下的闭包。可以省略  $X = \emptyset, X = \{\text{all attributes}\}$ ；如果  $X^+ = \{\text{all attributes}\}$ ，则跳过  $X$  的所有超集。
    - 将所有满足  $A \in (X^+ - X) \cap R_1$  的  $X \rightarrow A$  加入  $F_1$ 。
    - 于是我们得到了一个基  $F_1$ ，利用上面的算法计算出最小基即可。

## 5. 关系数据库设计理论：范式

假设有数据库 `StuCourse(sno, name, age, dept, cno, title, credit)`

**冗余 (redundancy)**：一个信息重复出现多次，如上述数据库中，`name, age, dept` 与 `title, credit` 等等出现多次。

**异常 (anomalies):** 更新异常 (修改某个学生的年龄可能要修改许多数据项)、删除异常 (如果只有一名学生选了某个课程, 当这门学生退课后将会丢失课程信息)、插入异常 (如果一个学生暂时还未选择任何课程, 那么当前数据库中不会有这个学生的信息)。

**解决方案:** 将关系分解成更多的小关系, 将不同的事物放入不同的表中 (如将上述数据库分成 `Student`, `Course`, `SC` 三个数据库)。

**分解 (decomposing relations)**

- 将一个关系分解成许多更小的关系来消除冗余。
- 将  $R(A_1, A_2, \dots, A_n)$  分解成  $S(B_1, B_2, \dots, B_m)$  与  $T(C_1, C_2, \dots, C_k)$ , 满足
  - $\{A_1, A_2, \dots, A_n\} = \{B_1, B_2, \dots, B_m\} \cup \{C_1, C_2, \dots, C_k\}$ ;
  - $S = \pi_{B_1, B_2, \dots, B_m}(R)$ ;
  - $T = \pi_{C_1, C_2, \dots, C_k}(R)$ 。

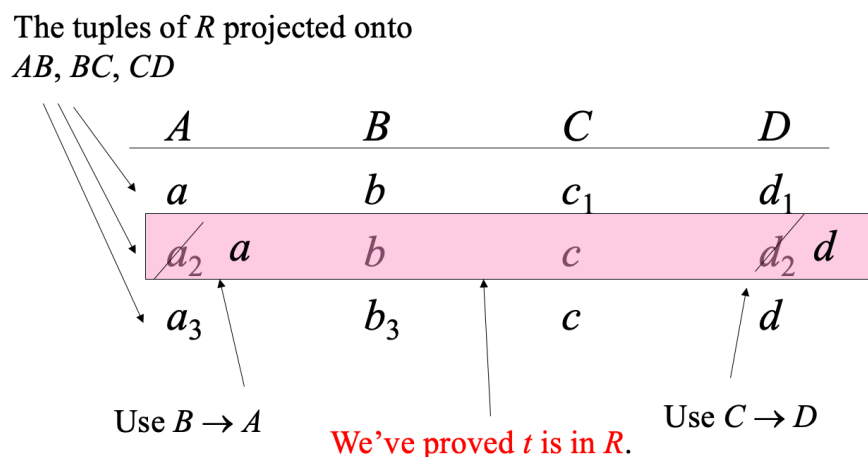
**Boyce-Codd 范式 (BCNF):**  $R$  是一个 BCNF 当且仅当对于任意非平凡的函数依赖  $X \rightarrow Y$ ,  $X$  都是  $R$  的超级键值 (superkey)。

- 例如, 在本章最初的数据库中, 存在  $sno \rightarrow name, age, dept$ , 而  $sno$  不是超级键值, 故该数据库不是 BCNF。
- 任何关系  $R$  都可以分解成若干关系  $R_1, R_2, \dots, R_n$  满足
  - 每个  $R_i$  都是 BCNF;
  - $R$  可以通过  $R_1, R_2, \dots, R_n$  重新构成。
- **分解策略:**
  - 找到一个违反 BCNF 的函数依赖  $X \rightarrow Y$ ;
  - 计算出  $X^+$ ;
  - 将  $R$  分解成  $R_1$  与  $R_2$ , 其中  $R_1 : X^+$ ,  $R_2 : (R - X^+) \cup X$ ;
  - 重复这个过程直到任何  $R_i$  都是 BCNF, 其中可能需要用到函数依赖的投影;
  - 可以证明, 这个算法一定能够成功。
  - **【注意】** 给定  $R$  与函数依赖集合  $F$ , 我们只需要考虑  $F$  中的函数依赖, 而不需要考虑  $F$  能推出的函数依赖。

**无损分解:** 将原始表分解成若干表, 将分解后的结果进行自然连接能够恰好得到原始表 (元组不增多、也不遗漏), 则这个分解称为无损分解, 这个连接称为无损连接 (lossless join)。

- 将  $R$  分解成 Boyce-Codd 范式的方法是无损分解。
  - 因为我们是按照函数依赖  $B^+ \rightarrow C$  分解的, 对于任意  $(A, B^+, C)$ , 其中  $A$  为其他无关属性。那么  $(A, B)$  中  $B$  的值唯一决定  $(B^+, C)$  中的  $B^+$  的值 (闭包的性质); 因此, 两个集合进行自然连接不会出现多余的元素。
- **追逐测试 (chase test):** 我们将原始表分解而成的若干表进行自然连接, 任取结果的一个元组  $t$ , 我们要判断  $t$  是否在原来的  $R$  中。一般地, 设  $t = (a, b, c, \dots)$ 。我们按照如下步骤进行测试:
  - 维护一个列表  $A, B, C, \dots$ ;
  - 对于一个分解表, 将其含有的属性用不带下标的小写字母表示, 其余属性用带下标的 (下标和之前不重复的) 小写字母表示, 加入列表中。例如一个分解表含有  $AB$ , 则将  $(a, b, c_1, \dots)$  加入列表中。
  - 对于每一个函数依赖, 如  $B \rightarrow C$ , 将所有列表中  $B$  属性相同的行的  $C$  属性设为相同值 (若存在无下标的元素, 则优先设为无下标元素)。如由于函数依赖的唯一性,  $(a_1, b, c, \dots), (a, b, c_1, \dots)$  被修改成  $(a_1, b, c, \dots), (a, b, c, \dots)$ 。

- 最后，如果存在一行  $(a, b, c, \dots)$  全部为无下标的小写字母，那么所有元组都能够原表中找到，分解是无损分解。
- 一个追逐测试的例子：



**依赖保持：**在对数据库进行更改时，我们需要高效验证依赖关系是否保持，如果依赖关系的左右部分被分为两个子关系中，那么验证依赖保持将低效（需要进行自然连接后再判断）。

**主属性 (prime)**  $A$  被称为主元素当且仅当其是一些键值的成员。

**第三范式 (3NF)：**一个关系  $R$  是第三范式当且仅当对于任意不平凡的函数依赖  $X \rightarrow Y$ ，有以下两条件至少之一成立：

- $X$  是超级键值 (superkey)；
- 对于所有  $A \in Y - X$  都被某些键值包含，即  $A$  是主元素。
- 【注】 $X$  是超级键值意味着保持  $X \rightarrow Y$  依赖要求我们不能分割（否则依赖保持将低效）；  
 $\forall A \in Y - X$ ， $A$  都被某些键值包含意味着保持  $X \rightarrow Y$  要求我们不能分割  $Y$ （否则依赖保持将低效）；但是如果  $\exists A \subseteq Y - X$ ， $A$  不被某些键值包含，我们可以分离出  $X \rightarrow A$  同时满足依赖保持的条件。
- 分解策略
  - 找到给定函数依赖的最小基 (minimal basis)；
  - 为最小基中的每一个函数依赖创建一个表，如有  $X \rightarrow A$ ，则创建  $T(X, A)$ ；
  - 如果这些表中不包含  $R$  的键值（即不存在超级键值），则构造一个仅包含  $R$  键值的关系；
  - 将这些表中平凡的表删除（被包含于另一个表）。

### 3NF v.s. BCNF

- BCNF 满足无损分解，但不一定满足依赖保持。
- 3NF 满足无损分解与依赖保持。

**多值依赖 (multivalued dependency, MVD)：**如果任意两个  $R$  中的元组在  $X$  上有着相同的值，那么交换他们的  $Y$  值得到的两个新的元素都在  $R$  中，则  $R$  上存在一个多值依赖  $X \twoheadrightarrow Y$ 。

- 若  $R(X, Y, Z)$  且  $X \twoheadrightarrow Y$ ，则  $Y$  与  $Z$  独立。
- 平凡的多值依赖：
  - 若  $Y \subseteq X$ ，则  $X \twoheadrightarrow Y$ ；
  - 如果  $R = X \cup Y$ ，则  $X \twoheadrightarrow Y$ 。
- 非平凡的多值依赖： $X \twoheadrightarrow Y$  时， $Y$  中的属性不出现在  $X$  中且  $X \cup Y$  不是  $R$  中的全部元素。

- 函数依赖是多值依赖的特例，即若  $X \rightarrow Y$ ，则  $X \twoheadrightarrow Y$ 。
- 传递性：若  $X \twoheadrightarrow Y$  且  $Y \twoheadrightarrow Z$ ，那么  $X \twoheadrightarrow Z$ 。
- 补集性质：若  $X \twoheadrightarrow Y$ ，那么  $X \twoheadrightarrow Z$ ，其中  $Z$  是所有不在  $X$  与  $Y$  中的参数。
- 不具有分离性！即  $X \twoheadrightarrow YZ$  不能推出  $X \twoheadrightarrow Y$  且  $X \twoheadrightarrow Z$ 。

**第四范式 (4NF)**：目标是消除由于 MVD 产生的冗余。一个关系  $R$  是第四范式当且仅当对于任意不平凡的 MVD  $X \twoheadrightarrow Y$ ， $X$  都是超级键值，这时候所有不平凡的 MVD 都是函数依赖。

- 若  $R$  为 4NF，则其一定为 BCNF。（根据 4NF 定义，所有的 MVD 都是函数依赖；同时违反 BCNF 的必定违反 4NF）
- 分解策略
  - 找到一个违反 4NF 的多值依赖  $X \twoheadrightarrow Y$ ；若找不到，则  $R$  已为 4NF；
  - 将  $R$  分成  $R_1(X, Y)$  与  $R_2(X, Z)$ ，其中  $Z = R - (X \cup Y)$ ；
  - 继续分解  $R_1, R_2$ 。

范式的关系： $4NF \Rightarrow BCNF \Rightarrow 3NF \Rightarrow 2NF \Rightarrow 1NF$ 。

#### 多值依赖与函数依赖

- 函数依赖得到  $Y$  中的值相等；
- 多值依赖得到  $R$  中必然含有某些其他元组（形成新的元组）。

**通过追逐测试推导函数依赖  $X \rightarrow Y$** （给定 FD 与 MVD）

- 一开始，用一个有两行的表来代表任意两个元组，其中两行中  $X$  中属性相同；
- 通过 FD、MVD 进行推导（FD 时使得元素相同往无下标的情况靠近，MVD 时增加行数）；
- 最后两行的  $Y$  若相等，则有  $X \rightarrow Y$ 。

**通过追逐测试推导多值依赖  $X \twoheadrightarrow Y$** （给定 FD 与 MVD）

- 一开始，用一个有两行的表来代表任意两个元组，其中两行中  $X$  中属性相同不带下标；第一行  $Y$  中元素不带下标，第二行  $R - (X \cup Y)$  中元素不带下标，其余元素带有不同的下标；
- 通过 FD、MVD 进行推导（FD 时使得元素相同往无下标的情况靠近，MVD 时增加行数）；
- 最后若产生全部无下标元组，那么说明  $X \twoheadrightarrow Y$ ；否则去重后就构造了一个反例。

**投影情况下的追逐测试**：可以先写出全部属性，最后只检查投影中含有的属性即可。