# 4关系数据库的设计

## 4.1 第一范式

若果关系R中的所有属性都是原子的，则该范式满足第一范式。

## 4.2 函数依赖

令 *R* 表示一个关系模式

α ⊆ *R 且 β* ⊆ *R*

函数依赖

**α → *β***

在*R*上成立，当且仅当任何一个合法的关系*r*(R)中,如果两个元组 *t*1和 *t*2 在r中具有相同的α的取值，也必须在*β*上有相同的取值：  *t*1[α] = *t*2 [α] ⇒ *t*1[*β* ] = *t*2 [*β* ]

我们可以利用好函数依赖重新定义“超码”和“候选码”

* *K* is a superkey for relation schema *R* if and only if *K* → *R*
* *K* is a candidate key for *R* if and only if
  + *K* → *R*, and
  + for no α ⊂ *K,* α → *R*
* 平凡的函数依赖：

α → *β* is trivial if *β* ⊆ α

4.3 函数依赖的闭包（closure）

所有能够被F逻辑蕴含的函数依赖的集合称之为F的闭包。（即那些F成立，则其必然也成立的函数依赖的集合）。

闭包用**F*+***表示。**F+**是F的超集。

我们在求**F*+***时，需要迭代使用下面Armstrong定理

1. 自反律：if *β* ⊆ α, then α → *β*
2. 增强律：α → *β,* then γ α → γ *β*
3. 传递率：α → *β,* and *β* → γ, then α → γ

下面的三个定律可以通过上面三个定律推导出：

1. 合并律

α → *β* holds *a*nd α→ γ holds, then α → *β* γ holds

1. 分解律

α → *β* γ holds, then α → *β* holds and α→ γ holds

1. 伪传递律

α → *β a*nd γ *β* → δ hold, then α γ → δ holds

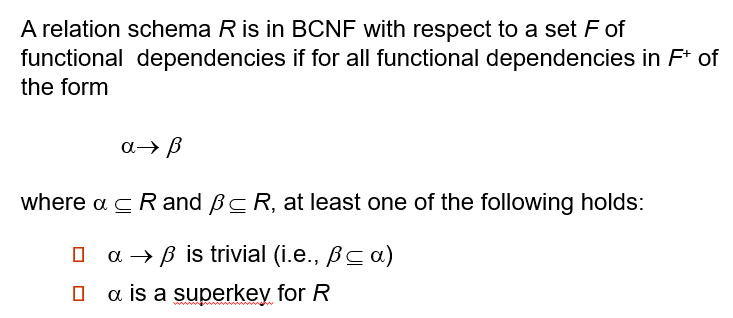
计算F+的方法如下：

*F* + = *F*

**repeat**  
 **for each** functional dependency *f* in *F*+  apply reflexivity and augmentation rules on *f* add the resulting functional dependencies to *F* + **for each** pair of functional dependencies *f*1and *f*2 in *F* +  **if** *f*1 and *f*2 can be combined using transitivity  
 **then**

add the resulting functional dependency to *F* +**until** *F* + does not change any further

4.4 Boyce-Codd Normal Form (BCNF)

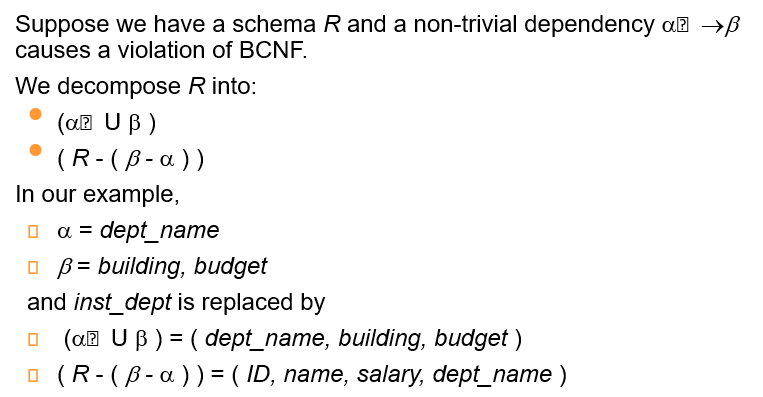


F+中的任何一个α→ *β*要么是平凡的，要么α是超码。只有满足上述条件，该关系模式R才满足BCNF范式。

★例子：

*instr\_dept* (*ID, name, salary, dept\_name, building, budget* )

因为*dept\_name*→ *building, budget*成立，所以上面这个关系模式不属于BCNF。



4.5 第三范式

一个关系模式R是第三范式，当所有的在*F*+中的α → *β* 至少满足如下三个条件之一：

（1）α → *β*是平凡的；

（2）α是关系模式R的一个超码；

（3）*β-*α中每个属性A均属于某个候选码，但不要求这些属性同属于一个候选码。

满足BCNF一定满足第三范式。第三范式是BCNF的最小松弛。

4.6 数据库设计的目标

令R表关系模式，F为R上的函数依赖。

首先，判断R是否是一个好的范式。

如果不是，需要对R进行分解，将R分解为{R1, R2, ..., Rn}。

（1）分解后的R1, R2, ..., Rn必须都是好的范式；

（2）分解是无损的；

（3）最好还能保持函数依赖

4.7 属性集合的闭包

给定一个属性集合α，在函数依赖集合F下的α的闭包是能由α函数依赖所决定的属性的集合。

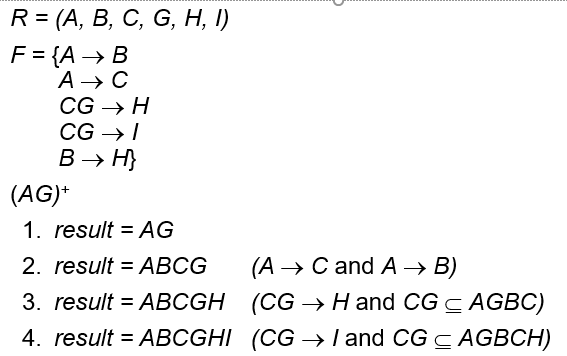
计算α在函数依赖集合F下的闭包α+。

*result* := ;  
**while** (changes to *result*) **do  
 for each** β→ γ **in** *F* **do  
 begin  
 if** β⊆ *result* **then**

*result* := *result* ∪ γ

**end**

**★例子：**



4.8 无关属性

判断F中的α → β中是否存在无关属性：

若*A* ∈ α，A是无关属性，当F能够蕴含(*F* – {α → β}) ∪ {(α – *A*) → β}.

若*A* ∈ β，A是无关属性，当(*F* – {α → β}) ∪ {α→ (β– *A*)}能够蕴含F.

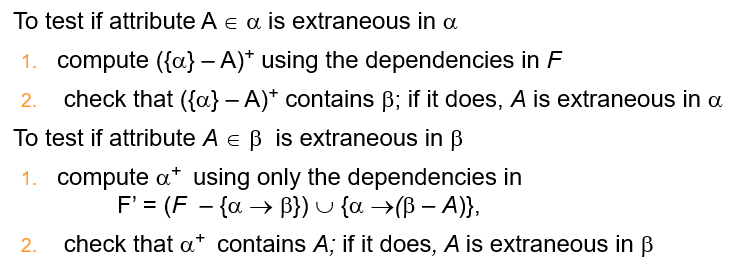
Example: Given *F* = {*A* →*C*, *AB*→*C* }

*B* is extraneous in *AB* →*C* because {*A* →*C, AB* →*C*} logically implies *A* → *C* (I.e. the result of dropping *B* from *AB* →*C*).

Example: Given *F* = {*A*→ *C*, *AB*→*CD}*

*C* is extraneous in *AB*→*CD* since *A*B→*C* can be inferred even after deleting *C*

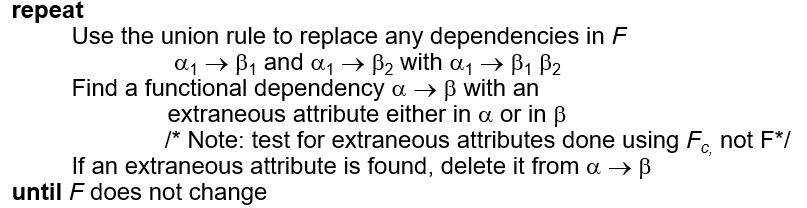
除了用定义之外，还可以通过属性的闭包来判断某一属性是否为无关属性：



4.9 正则覆盖

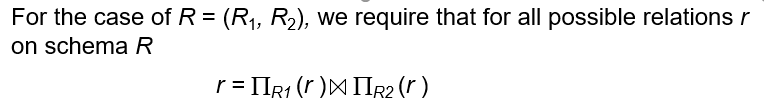
正则覆盖Fc与函数依赖集F是完全等价的，即Fc蕴含F，F蕴含Fc。且Fc满足，其所含的函数依赖中没有无关属性，各个函数依赖的左侧没有重复。

计算正则覆盖的代码如下：

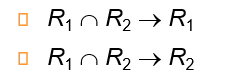


4.10 无损连接

定义：



判断方法：只要其中一个成立即可。



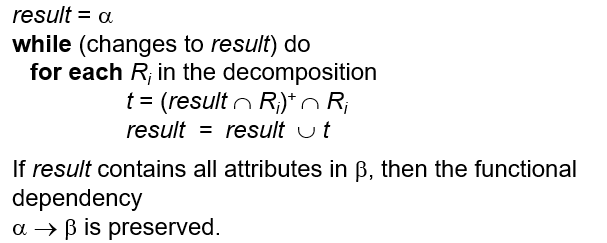
4.11 函数依赖保持

令*Fi*为F+中只包含Ri中属性的函数依赖的集合。

如果分解是无损的，当如下等式成立：

C:\Users\zhang\AppData\Local\Temp\1559268056(1).png

判断α → β这个函数依赖是不是保持的程序

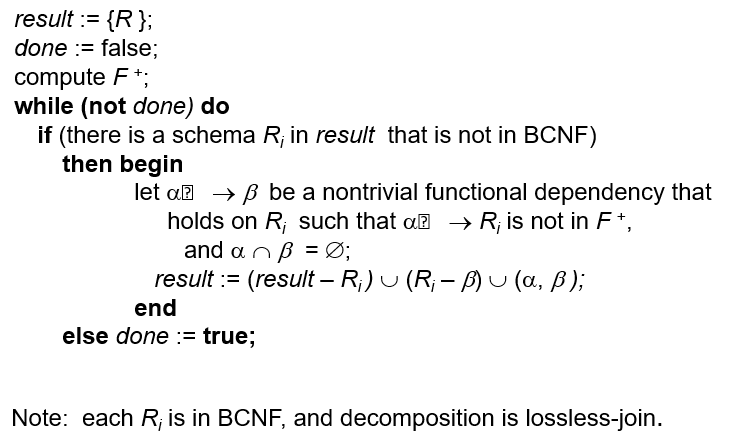


4.12 BCNF判断

判断关系R是否属于BCNF，只需要判断F中的函数依赖是否违反BCNF的要求。

判断R分解后的结果是否满足BCNF，则需要判断各个分解后的关系模式在F+违反了BCNF的要求。

4.13 BCNF分解



4.14 判断第三范式

判断第三范式只需要在F上进行，无需在F+上。

第三范式的分解

