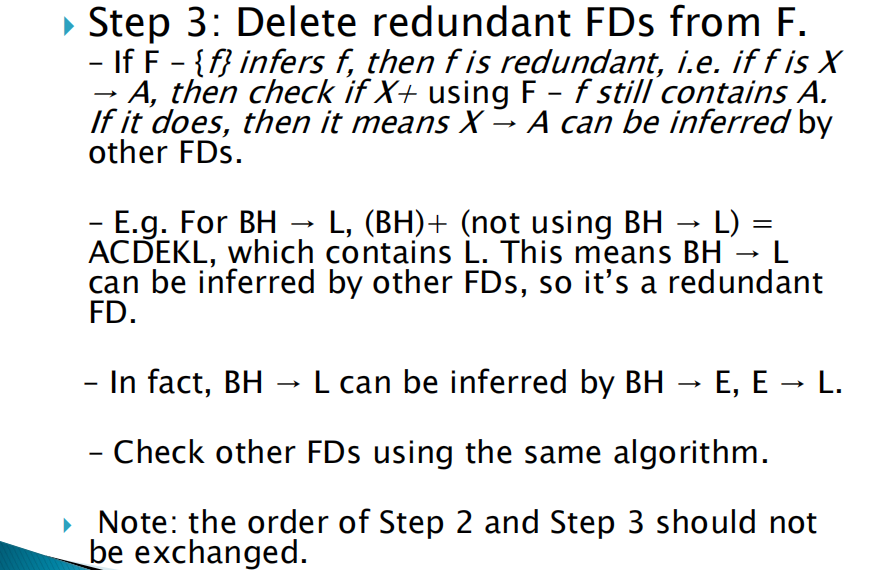
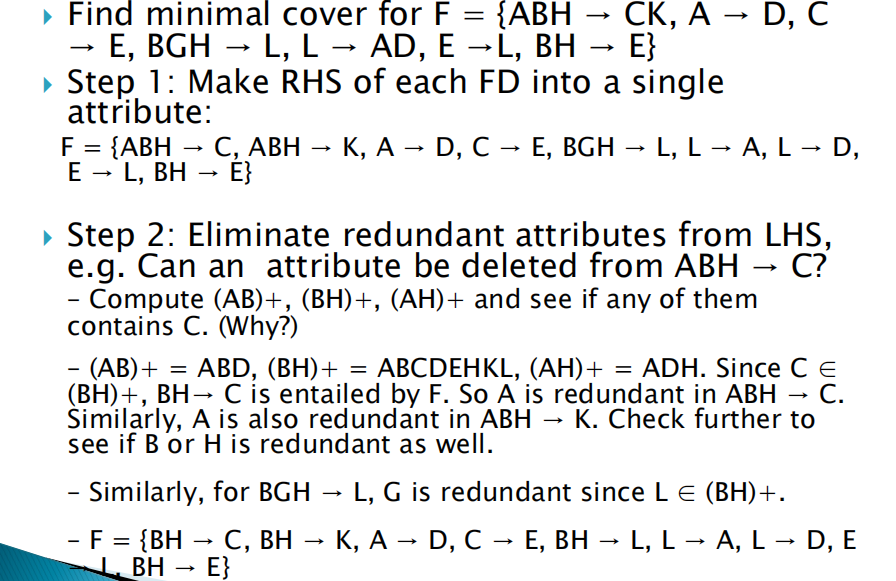
1. **函数依赖的判断**
2. 判断一个函数依赖X →Y 是否在F的闭包中，只需要计算属性X的闭包 (记为X+)，如果X的闭包中有Y，那么函数依赖X →Y就在F的闭包中
3. **超键的判断**
4. 只需要计算 X+, 检查 X+ 是否包括R的所有属性
5. **候选码的判断**
6. 子集不是超键，自身是超键
7. **BC范式的检验**
8. 列出所有的非平凡函数
9. 依赖找出所有R的超键
10. 确认每一个函数依赖箭头左边的属性集是R的超键
11. **BC范式的分解**

给出R和函数依赖集F：

1. 求出候选码
2. 观察函数依赖集，如果左边不是超码（候选码），则不满足条件
3. 用不满足条件的函数依赖（A->B）进行分解，这样分解之后就满足了
   1. R1=AB（这样就满足了）
   2. R2=（R-R1)∪A
   3. F2={…}去掉B的所有函数依赖，尽可能写全
4. 对F2进行步骤1的计算。
5. 重复直到所有的满足条件

通常情况, 有多个函数依赖违背BCNF. 那么，我们处理它们的顺序的不同会导致分解的结果不同，但是都满足BCNF范式约束

1. **三范式的检验**
2. 使用属性闭包检查对于F中的每个函数依赖α→β，a是否是一个超键
3. 如果不是，再检测β中的属性是否包含在R的某一个候选键中（要清楚R的所有候选键）
4. 注意：计算候选键的代价很高，是NP难问题；但是分解到3NF的操作是多项式复杂度
5. **三范式的分解**
6. 如果 X → Y 不保持函数依赖, 增加关系XY，但是问题是XY可能会违背3NF，因此不考虑 FDsF, 而是使用F的最小的函数依赖集
7. 计算正则覆盖
8. 先计算最小函数依赖，然后合并

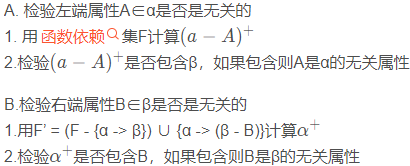


此时我们只是得到了最小函数依赖，然后我们合并左边相同的函数依赖，就可以得到正则覆盖

1. 直接计算正则覆盖

重复如下步骤直到函数依赖没有变化：

1. 首先，使用合并律将所有左部相同的函数依赖合并成一个
2. 在合并后的函数依赖集中，对于每一个函数依赖，寻找一个无关属性，将它删除，直到没有无关属性可以删



1. 对于正则覆盖里的每一个函数依赖，设立一个关系
2. 如果任何候选键都没被包含，那么就增加候选键关系
3. 如果有被包含的函数依赖，删除
4. **分解是否保持函数依赖**
5. 如果F上的每一个[函数依赖](https://so.csdn.net/so/search?q=%E5%87%BD%E6%95%B0%E4%BE%9D%E8%B5%96&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_35293120/article/details/_blank)都在其分解后的某一个关系上成立，则这个分解是保持依赖的（充分条件）。
6. 如果上述判断失败，并不能断言分解不是保持依赖的，因为上面只是充分条件，还要使用下面的通用方法来做进一步判断。
7. 对F上的每一个不能在一个分解后的关系上直接判断的函数依赖α→β使用下面的过程：

result:=α;

while(result改变)do

for each 分解后的Ri

t=(result∩Ri)+ ∩Ri

result=result∪t

如果result包含了β，那么就是保持了函数依赖

1. **可串行化**

一个并行调度的结果和某个串行调度的结果相同

1. **冲突可串行化**

一个并行调度，经过非冲突指令的交换，可以变成一个串行化的调度。

1. **冲突可串行化的判断方法**

优先图

1. **可串行化和冲突可串行化**

可串行化包含冲突可串行化，一个调度是冲突可串行化的，一定是可串行化的，一个调度是可串行化的，不一定是冲突可串行化的。

1. **恢复算法（无检查点）**

* 正向扫描日志文件（即从头扫描日志文件）
  + Redo队列: 在故障发生前已经提交的事务Ti
    - 日志包括<Ti start>
    - 以及<Ti commit>或<Ti abort>
  + Undo队列:故障发生时尚未完成的事务Ti
    - 日志包括<Ti, start>
    - 但不包括<Ti commit>
    - 也不包括<Ti abort>
* 对Undo队列事务进行UNDO处理
  + 反向扫描日志文件，对每个UNDO队列中Ti 更 新过的数据项恢复为旧值，写入日志<Ti ，Xj，V1> ，直到 遇到<Ti start>记录，说明该事务撤销完成， 写入一条 <Ti abort>记录
* 对Redo队列事务进行REDO处理
  + 正向扫描日志文件，对每个REDO队列中的Ti更新过的数据项设置为新值，（abort也要做，因为abort之前的日志中已经记录了撤销操作的条目），即对于所有的<Ti ，Xj，V2>和<Ti ，Xj，V1，V2>,把Xj的值赋为V2

1. **恢复算法（有检查点）**

只考虑检查点以后的日志

* 不执行操作：
  + 在检查点前提交
* 执行重做操作：
  + T2：在检查点前开始执行，在检查点后、故障点前提交
  + T3：在检查点后开始执行，在故障点前提交
* 执行撤销操作：
  + 在检查点前开始执行，在故障点后提交
  + 在检查点后开始执行，在故障点后提交

1. **判断是否是无损连接**

### 当分解ρ只有两组的时候，有现有公式: R1∩R2→R1-R2或R1∩R2→R2-R1  这两个条件满足任何一个都是无损连接

ρ={R1<U1,F1>,R2<U2,F2>,...,Rk<Uk,Fk>}是关系模式R<U,F>的一个分解，U={A1,A2,...,An}，F={FD1,FD2,...,FDp}，并设F是一个最小依赖集，记FDi为Xi→Alj，其步骤如下：

① 建立一张n列k行的表，每一列对应一个属性，每一行对应分解中的一个关系模式。若属性Aj Ui，则在j列i行上真上aj，否则填上bij；

② 对于每一个FDi做如下操作：找到Xi所对应的列中具有相同符号的那些行。考察这些行中li列的元素，若其中有aj，则全部改为aj，否则全部改为bmli，m是这些行的行号最小值。

**如果在某次更改后，有一行成为：a1,a2,...,an，则算法终止。且分解ρ具有无损连接性**，否则不具有无损连接性。

对F中p个FD逐一进行一次这样的处理，称为对F的一次扫描。

③ 比较扫描前后，表有无变化，如有变化，则返回第② 步，否则算法终止。如果发生循环，那么前次扫描至少应使该表减少一个符号，表中符号有限，因此，循环必然终止。

举例1：已知R<U,F>，U={A,B,C}，F={A→B}，如下的两个分解：

① ρ1={AB,BC}

② ρ2={AB,AC}

判断这两个分解是否具有无损连接性。

①因为AB∩BC=B，AB-BC=A，BC-AB=C

所以B→A ¢F+，B→C ¢ F+

故ρ1是有损连接。

② 因为AB∩AC=A，AB-AC=B，AC-AB=C

所以A→B €F+，A→C ¢F+

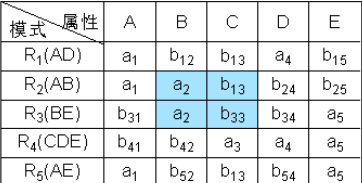
故ρ2是无损连接。

举例2：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



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）。



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）。



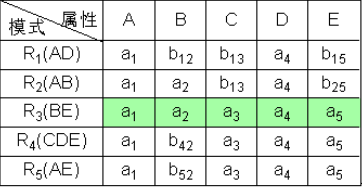
④ 根据C→D，对上表进行处理，由于属性列C上第1、2、3、5行相同均为b13，所以将属性列D上的值均改为同一个符号a4。



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。



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。



⑦ 通过上述的修改，使第三行成为a1a2a3a4a5，则算法终止。且分解具有无损连接性。

1. **时间戳算法**

假设事务Ti发出read(Q)操作（事务时间戳小于冲突操作时间戳时回滚）

* 若TS(Ti)<W-TS(Q)，则Ti需要读入的Q值已被覆盖。因此，read操作被拒绝，Ti回滚
* 若TS(Ti)≥W-TS(Q)，则执行read操作，而R-TS(Q)的值被设为R-TS(Q)与TS(Ti)中的较大者（有可能后面的事务比自己先读）

假设事务Ti发出write(Q)操作

* 若TS(Ti)<R-TS(Q)，则Ti产生的Q值是先前所需要的值，但系统已假定该值不会被产生。因此，write操作被拒绝，Ti回滚(新事务已读，不应再改)
* 若TS(Ti)<W-TS(Q)，则Ti产生的Q值已过时。因此，write操作被拒绝，Ti回滚(新事务已写，不应再改)
* 其他情况，系统执行write操作，并将W-TS(Q)的值设为TS(Ti)(属于正常写，更新时间戳)