数据库系统原理

陈岭

浙江大学计算机学院

10 BCNF、3NF和4NF

- BCNF
- 3NF
- 多值依赖
- 4NF
- 数据库设计过程

Boyce-Codd范式

- 具有函数依赖集合F 的关系模式R 属于BCNF的条件是，当且仅当对F^+ 中所有函数依赖$\alpha \rightarrow \beta$（其中，$\alpha \subseteq R$ 且$\beta \subseteq R$），下列至少有一项成立
  - $\alpha \rightarrow \beta$ 是平凡的函数依赖（即$\beta \subseteq \alpha$）
  - α 是R 的超码(即，$R \subseteq \alpha^+$ ，$\alpha \rightarrow R$）

Boyce-Codd范式

- 例，$R = (A, \quad B, \quad C)$

  $F = \{A \rightarrow B$

  $B \rightarrow C\}$

  键= $\{A\}$

- R 不属于BCNF（$\because F^{+} = \{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow C, \quad \cdots\}$，因为$B \rightarrow C$，$B$ 不是超码）
- 分解成$R_1 = (A, \quad B)$，$R_2 = (B, \quad C)$
  - R_1 与R_2 属于BCNF（A 是R_1的超码，B 是R_2的超码）
  - 无损连接分解（$R_1 \cap R_2 = B$，并且B 是R_2的超码）
  - 保持依赖（$F_1 = \{A \rightarrow B\}$ 在R_1上保持依赖，$F_2 = \{B \rightarrow C\}$ 在R_2上保持依赖）

检查是否为BCNF

- 为检查非平凡依赖 $\alpha \rightarrow \beta$ 是否违反BCNF的要求
  - 计算α^+
  - 检验α^+ 是否包含R 的所有属性，即，是否为R 的超码
- 简化的测试：为检查具有函数依赖集合F 的关系模式R 是否属于BCNF，只需检查F 中的函数依赖是否违反BCNF即可，而不需检查F^+ 中的所有函数依赖
  - 可以证明如果F 中没有违反BCNF的依赖，则F^+ 中也没有违反BCNF的依赖

> ∵ F^+是由Armstrong的3个公理从F 推出的，而任何公理都不会使FD左边变小(拆分)，故如果F中没有违反BCNF的FD(即左边是superkey)，则F^+中也不会

检查是否为BCNF

- 但是，当检查R 的分解后的关系时仅用F是错误的
- 例，考虑 R (A，B，C，D)，具有F = {$A \rightarrow B$，$B \rightarrow C$}
  - 分解R 到R_1(A，B) 与R_2(A，C，D)
  - F 中的函数依赖都不是只包含(A，C，D)中的属性， 因此我们可能错误地认为R_2 满足BCNF
  - 事实上， F^+ 中的依赖$A \rightarrow C$ 显示R_2 不属于BCNF

可在F下判别R 是否违反BCNF，但须在F^+下判别R 的分解式是否违反BCNF

BCNF分解算法

result := {*R*};
done := false;
compute F^+;
while (not *done*) do
  if (*result* 中存在模式R_i 不属于BCNF)
    then begin
      令$\alpha \rightarrow \beta$ 是R_i 上的一个非平凡函数依赖
        使得$\alpha \rightarrow R_i$ 不属于F^+，且$\alpha \cap \beta = \varnothing$;
      result := (*result* − R_i) ∪ (R_i − β) ∪ (α, β);
      end
    else *done* := true;

将R_i分解为二个子模式：$R_{i1}=(\alpha, \beta)$和$R_{i2}=(R_i - \beta)$，α是R_{i1}和R_{i2}的共同属性

R_{i1}　　R_{i2}

- 注意：每个R_i 都属于BCNF，且分解是无损连接的

BCNF分解示例

- *class* (*course_id, title, dept_name, credits, sec_id, semester, year, building, room_number, capacity, time_slot_id*)
- 函数依赖：
  - *course_id* → *title, dept_name, credits*
  - *building, room_number* → *capacity*
  - *course_id, sec_id, semester, year* → *building, room_number, time_slot_id*
- 候选码：{*course_id, sec_id, semester, year*}
- BCNF分解：
  - *course_id* → *title, dept_name, credits*，但是*course_id* 不是超码

BCNF分解示例

- 我们将*class* 分解为：
  - *course*(*course_id, title, dept_name, credits*)
  - *class-1* (*course_id, sec_id, semester, year, building, room_number, capacity, time_slot_id*)
- *course*是BCNF
- *building, room_number* → *capacity* 在*class-1*上存在依赖，但是{*building, room_number*}不是*class-1*的超码
- 我们将*class-1* 分解为：
  - *classroom* (*building, room_number, capacity*)
  - *section* (*course_id, sec_id, semester, year, building, room_number, time_slot_id*)
- *classroom* 和*section* 是BCNF

BCNF与保持依赖

- BCNF分解不总是保持依赖的
- 例，$R = (J, K, L)$

  $F = \{JK \rightarrow L$

  $L \rightarrow K\}$

J---student，K---course，L---teacher（一门课有多个教师，一个教师上一门课，一个学生选多门课，一门课有多个学生选）

- 两个候选码：JK 和JL
- R 不属于BCNF（$\because L \rightarrow K$，L 不是超码）
- R 的任何分解都不满足保持依赖：$JK \rightarrow L$
  - 例，$R_1 = (L, K)$，$R_2 = (J, L)$ $\subseteq$ BCNF，但不保持依赖

BCNF与保持依赖

- 因此，我们并不总能满足这三个设计目标：
  - 无损连接
  - BCNF
  - 保持依赖

3NF：动机

- 存在这样的情况
  - BCNF不保持依赖
  - 但是，有效检查更新是否违反FD是重要的
- 解决方法：定义一种较弱的范式，称为第三范式（3NF）
  - 允许出现一些冗余（从而会带来一些问题）
  - 但FD可以在单个关系中检查，不必计算连接
  - 总是存在到3NF的无损连接分解，且是保持依赖的

3NF

- 关系模式R 属于第三范式（3NF）当且仅当对所有F^+ 中的函数依赖

$$\alpha \rightarrow \beta$$

下列条件中至少一个成立：

  - $\alpha \rightarrow \beta$ 是平凡的函数依赖（即，$\beta \in \alpha$）
  - α 是R 的超码
  - $\beta - \alpha$ 中的每个属性A 都包含于R 的一个候选码中（即$A \in \beta - \alpha$ 是主属性，若$\alpha \cap \beta = \varnothing$，则$A = \beta$ 是主属性）

- 注：各属性可能包含在不同候选码中

3NF

- 若一个关系属于BCNF则必属于3NF
- 第三范式相对于BCNF来说放宽了约束，允许非平凡函数依赖的左面不是超码。因为候选码是最小的超码，它的任何一个真子集都不是超码
- 第三个条件是对BCNF条件的最小放宽，以确保每一个模式都有保持依赖的3NF分解

讨论：国内其他教材关于3NF的定义：不存在非主属性对码的部分依赖和传递依赖。该定义实际是说，当β为非主属性时，α必须是码；但当β为主属性时，则α无限制。 二种定义本质上是一致的

3NF

- 例，$R = (J, K, L)$

  $F = \{JK \rightarrow L, L \rightarrow K\}$

- 两个候选码：JK 和JL
- R 属于3NF
  - $JK \rightarrow L$，JK 是超码
  - $L \rightarrow K$，K 包含在一候选码中
- 但BCNF分解将得到(J, L)和(L, K)，$JK \rightarrow L$不保持依赖
  - 检查$JK \rightarrow L$，需要连接操作
- 此模式中存在冗余

J---student，K---course，L---teacher（一门有多个教师，一个教师上一门课，一个学生选多门课，一门课有多个学生选）

BCNF与3NF的比较

- 因3NF的冗余引起的问题

  $R = (J, K, L)$

  $F = \{JK \rightarrow L, L \rightarrow K\}$

J---student，K---course，L---teacher（一门有多个教师，一个教师上一门课，一个学生选多门课，一门课有多个学生选）

- 属于3NF但不属于BCNF的模式存在以下问题：
  - 信息重复（如，联系l_1和k_1）
  - 需要使用空值（如，表示联系l_2和k_2，这里没有对应的J值）

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
$null$	l_2	k_2

检查是否为3NF

- 优化：只需检查F 中的FD，而不必检查F^+ 中的所有FD
- 对每个依赖$\alpha \rightarrow \beta$，利用属性闭包来检查α 是否为超码
- 如果α 不是超码，必须检查β 中的每个属性是否包含在R 的某个候选码中
  - 这个检查较昂贵，，因为它涉及求候选码
  - 检查是否属于3NF是NP-hard的
  - 有趣的是，分解到第三范式可以在多项式时间内完成

3NF分解算法

令F_c 是F 的正则覆盖;
i := 0;
for each F_c 中的函数依赖$\alpha \rightarrow \beta$ do{
 if 没有模式R_j ($1 \leq j \leq i$)包含$\alpha\beta$
 then begin
 $i := i + 1$;
 $R_i := (\alpha, \beta)$
 end}
if 没有模式R_j ($1 \leq j \leq i$)包含R 的候选码
 then begin
 $i := i + 1$;
 R_i := R 的任意候选码;
 end
return (R_1, R_2, ..., R_i)

将F_c中的每个$\alpha \rightarrow \beta$ 分解为子模式$R_i := (\alpha, \beta)$，从而保证 dependency-preserving

保证至少在一个R_i中存在R的候选码，从而保证 lossless-join

3NF示例

- 关系模式：*cust_banker_branch* = (*customer_id*, *employee_id*, *branch_name*, *type*)
- 函数依赖：
  - *customer_id*, *employee_id* → *branch_name*, *type*
  - *employee_id* → *branch_name*
  - *customer_id*, *branch_name* → *employee_id*
- 计算正则覆盖：
  - *branch_name* 在第一个函数依赖中是多余的
  - 没有其他的多余属性，因此，我们得到F_C =
    - *customer_id*, *employee_id* → *type*
    - *employee_id* → *branch_name*
    - *customer_id*, *branch_name* → *employee_id*

3NF示例

- 通过for循环，我们得到以下子关系模式：
  - (*customer_id, employee_id, type*)
  - (*employee_id, branch_name*)
  - (*customer_id, branch_name, employee_id*)
  - 由于(*customer_id, employee_id, type*)包含原关系模式的候选码，分解到此为止
- 在循环结束后，检查并删除模式。如，(*employee_id, branch_name*)是其他模式的子集，应该删除
  - 结果与考虑函数依赖的顺序无关

3NF示例

- 最后，得到3NF分解的子关系模式：
  - (*customer_id, employee_id, type*)
  - (*customer_id, branch_name, employee_id*)

BCNF与3NF的比较

- 总是可以将一个关系分解到3NF并且满足
  - 分解是无损的
  - 保持依赖
- 总是可以将一个关系分解到BCNF并且满足
  - 分解是无损的
  - 但可能不保持依赖

练习

- 例1，F = {AB → E, BE → I, E → G, GI → H}，使用Armstrong公理证明AB → GH

  证明：AB →E，由增补率得：AB →BE；BE→I，由增补率得：BE→EI；

  ∴AB →EI　　----(1)

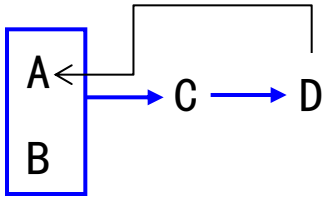
  E →G，由增补率得：EI →GI；GI →H，由增补率得：GI →GH

  ∴EI →GH　　----(2)

  ∴ (1)，(2)通过传递律得：AB →GH

练习

- 例2，对于关系模式：$R(A,\ B,\ C,\ D)$，$F = \{AB \to C,\ C \to D,\ D \to A\}$
  - 1）列出R 的所有候选码
  - 2）将R 分解为到BCNF
  - 3）上述分解是否为保持依赖的



答：1）$(AB)^+ = (ABCD) \supseteq R$，$(BC)^+ = (ABCD) \supseteq R$，

$D \to A$，$BD \to AB$；$AB \to C$；∴ $(BD)^+ = (ABCD) \supseteq R$，

∴AB，BC，BD 是候选码，且R 不满足BCNF

2）$R1(C, D)$；$R2(A, B, C)$，

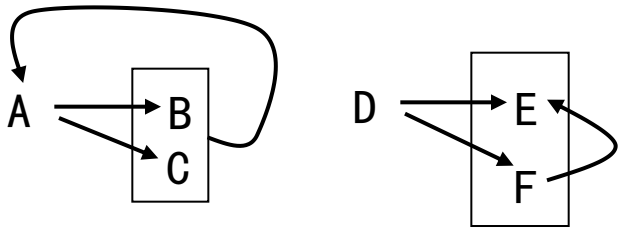
（$R1$满足BCNF，$R2$不满足BCNF，∵$C \to D$，$D \to A$，∴ $C \to A$，C不是$R2$的超码）；

$R21(A, C)$，$R22(B, C)$，$R21$ 满足BCNF，$R22$ 满足BCNF

3）$D \to A$，$AB \to C$ 不是保持依赖的

练习

- 例3，对于关系模式：$R(A, B, C, D, E, F)$，$F = \{A \rightarrow B, A \rightarrow C, BC \rightarrow A, D \rightarrow EF, F \rightarrow E\}$
  - 1）列出R 的所有候选码
  - 2）R 属于BCNF还是3NF，或都不属于
  - 3）如果R 不属于BCNF，将其分解到BCNF
  - 4）上述分解是否为保持依赖的



答：1）候选码为：AD，BCD

2）$\because F \rightarrow E$，F 不是超码，E 不在超码中，$\therefore$ 不属于BCNF，也不属于3NF

3）$R1=(A, B, C)$，$R2=(A, D, E, F)$，（由$A \rightarrow BC$）

$R21=(D, E, F)$，$R22=(A, D)$，$\because F \rightarrow E$，$R21$不属于BCNF，

$R211=(F, E)$，$R212=(D, F)$

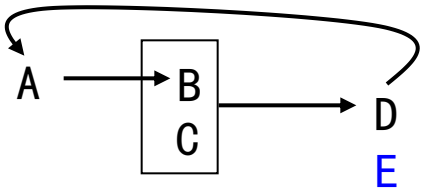
练习

答：3) 方法2：$R1=(B, C, A)$， $R2=(B, C, D, E, F)$； $R21=(F, E)$， $R22=(B, C, D, F)$，
$R221=(D, F)$， $R222=(B, C, D)$

4) $D \rightarrow E$是保持依赖吗？

练习

- 例4，对于关系模式：$R(A, B, C, D, E)$，$F = \{A \rightarrow B, BC \rightarrow D, D \rightarrow A\}$
  - 1) 列出R 的所有候选码
  - 2) R 属于BCNF还是3NF，或都不属于
  - 3) 如果R 不属于BCNF，将其分解到BCNF
  - 4) 上述分解是否为保持依赖的



答：1) 候选码为：ACE, BCE, CDE

2) ∵F 中右边的每一个属性都在候选码中，∴ 属于3NF

3) $R1=(A, B)$，$R2=(A, C, D, E)$，

$R21=(A, D)$，$R22=(C, D, E)$

练习

答：3) 方法2，$R1=(A, B)$，$R2=(A, C, D, E)$，

$R21=(A, C, D)$，$R22=(A, C, E)$，

$R211=(A, D)$，$R212=(C, D)$

4) $BC \rightarrow D$ 不是保持依赖的

设计目标

- 关系数据库设计的目标是：
  - BCNF
  - 无损连接
  - 依赖保持
- 如果不能达到这些，也可接受
  - 缺少保持依赖
  - 因3NF引起的冗余
- 除了超码之外，SQL并没有提供直接声明函数依赖的方法。可以通过断言声明FD，但检测代价太大
- 因此即使我们有一个保持依赖的分解，使用SQL，我们也不能有效地检测左部不是码的函数依赖

多值依赖

- 有时属于BCNF的模式仍然未充分规范化
- 考虑数据库

  $classes(course,\ teacher,\ book)$

  定义$(c, t, b) \in classes$，意思是教师t 可以教课程c，而b 是需用于课程c 的教材
- 数据库将为每门课程列出能讲授该课程的教师的集合，以及需用的书的集合(不管谁讲授该课)
  - *course*: *teacher* =1:n，*course*: *book* = 1:n，*teacher*和*book*是多值属性，并且*teacher*和*book*相互独立

多值依赖

course	*teacher*	*book*
database	Avi	DB Concepts
database	Avi	DB system (Ullman)
database	Hank	DB Concepts
database	Hank	DB system (Ullman)
database	Sudarshan	DB Concepts
database	Sudarshan	DB system (Ullman)
operating systems	Avi	OS Concepts
operating systems	Avi	OS system (Shaw)
operating systems	Jim	OS Concepts
operating systems	Jim	OS system (Shaw)

classes

- 由于没有非平凡依赖，(*course*, *teacher*, *book*)是唯一的键，因此该关系模式属于BCNF

多值依赖

- 冗余与插入异常： 如果Sara是能教数据库的新教师，必须插入两条元组

  (database, Sara, DB Concepts)

  (database, Sara, Ullman)

多值依赖

- 因此，最好将*classes* 分解成：

course	*teacher*
database	Avi
database	Hank
database	Sudarshan
operating systems	Avi
operating systems	Jim

teaches

key={course, teacher}

course	*book*
database	DB Concepts
database	DB system (Ullman)
operating systems	OS Concepts
operating systems	OS system (Shaw)

text

我们将看到这两个关系都属于第四范式(4NF)

多值依赖

- 设有关系模式R，令$\alpha \subseteq R$ 及$\beta \subseteq R$。多值依赖

$$\alpha \rightarrow\rightarrow \beta$$

在R 上成立当且仅当在任意合法关系$r(R)$中，对所有满足$t_1[\alpha] = t_2[\alpha]$的元组对$t_1$ 和t_2，必存在元组t_3 和t_4 使得：

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_4[\beta] = t_2[\beta]$$
$$t_3[R - \alpha - \beta] = t_2[R - \alpha - \beta]$$
$$t_4[R - \alpha - \beta] = t_1[R - \alpha - \beta]$$

令$R - \alpha - \beta = z$

$t_3[z] = t_2[z]$

$t_4[z] = t_1[z]$

多值依赖

- 用表来表示 $\alpha \rightarrow\rightarrow \beta$

	α	β	$R-\alpha-\beta$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$
	t1[ ]=t2[ ] =t3[ ]=t4[ ]	t1[ ]=t3[ ] t2[ ]=t4[ ]	t1[ ]=t4[ ] t2[ ]=t3[ ]

- 如果$\beta \subseteq \alpha$，或$\alpha \cup \beta = R$，则$\alpha \rightarrow\rightarrow \beta$ 是平凡的

多值依赖

- 另一种定义

	α	β	$R-\alpha-\beta$
t_1	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$a_{j+1} \dots a_n$
t_2	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$b_{j+1} \dots b_n$
t_3	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$b_{j+1} \dots b_n$
t_4	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$a_{j+1} \dots a_n$
	t1[ ]=t2[ ] =t3[ ]	t1[ ]=t3[ ]	t2[ ]=t3[ ]

对任意三个元组都成立

多值依赖

- 例，设关系模式R 的属性集合被分成三个非空子集

  $Y,\ Z,\ W$

  称$Y \rightarrow\rightarrow Z$（$Y$ 多值决定Z）当且仅当对所有可能的关系$r(R)$，若

  $\langle y_1,\ z_1,\ w_1 \rangle \in r$ 及 $\langle y_2,\ z_2,\ w_2 \rangle \in r$

  则

  $\langle y_1,\ z_1,\ w_2 \rangle \in r$ 及 $\langle y_1,\ z_2,\ w_1 \rangle \in r$

- 注意由于Z 和W 的行为完全对称，若$Y \rightarrow\rightarrow W$ 则$Y \rightarrow\rightarrow Z$

多值依赖

- 在前面的例子中：

  $course \rightarrow\rightarrow teacher$

  $course \rightarrow\rightarrow book$

- 上述形式定义表达了这种概念：给定Y（$course$）的特定值，则有一个Z（$teacher$）值的集合和一个W（$book$）值的集合与之相关联，而这两个集合在某种意义上是相互独立的

多值依赖理论

- 根据多值依赖的定义，可导出下列规则：
  - 若 $\alpha \rightarrow \beta$，则$\alpha \rightarrow\rightarrow \beta$，即函数依赖是多值依赖的特例
- D 的闭包D^+ 是D 逻辑蕴含的所有函数依赖和多值依赖的集合
  - 可根据函数依赖与多值依赖的形式定义来从D 计算$D+$
  - 我们只对在实践中较常见的简单多值依赖可用这样的推理
  - 对于复杂的依赖，最好利用一套推理规则来对依赖集合进行推理

4NF

- 关系模式R 关于函数依赖及多值依赖集合D 属于4NF当且仅当对D^+ 中所有形如$\alpha \rightarrow\rightarrow \beta$ 的多值依赖，其中$\alpha \subseteq R$ 且$\beta \subseteq R$，下列条件中至少一个成立：
  - $\alpha \rightarrow\rightarrow \beta$ 是平凡的（即，$\beta \subseteq \alpha$ 或$\alpha \cup \beta = R$）
  - α 是模式R 的超码
- 若关系属于4NF，则它必属于BCNF

4NF分解算法

result: = {*R*};
done := false;
compute D^+;
令D_i 表示D^+ 在R_i 上的限制
while (not *done*)
if (在*result* 中有不属于4NF的模式R_i) then
begin
令$\alpha \rightarrow\rightarrow \beta$ 是在R_i 上成立的一个非平凡多值依赖，它使得$\alpha \rightarrow R_i$ 不属于D_i，并且$\alpha \cap \beta = \phi$;
result := (*result* − R_i) ∪ (R_i − β) ∪ (α, β);
end
else *done*:= true;

R_{i1}　　R_{i2}

注：每个R_i 属于4NF，且分解是无损连接的

4NF

- 例，$R=(A,\ B,\ C,\ G,\ H,\ I)$

  $F=\{A \rightarrow\rightarrow B$

  $B \rightarrow\rightarrow HI$

  $CG \rightarrow\rightarrow H\}$

- R 不属于4NF，因为$A \rightarrow\rightarrow B$，$A$ 不是R 的超码
- 分解：
  - a）$R_1=(A,\ B)$　　（R_1 属于4NF）
  - b）$R_2=(A,\ C,\ G,\ H,\ I)$　　（R_2 不属于4NF）
  - c）$R_{21}=(C,\ G,\ H)$　　（R_{21} 属于4NF）
  - d）$R_{22}=(A,\ C,\ G,\ I)$　　（R_{22} 属于4NF）

更多的范式

- 连接依赖概化了多值依赖
  - 并引出了另一种范式投影-连接范式（PJNF）
- 还有一类更一般化的约束，它引出一种称作域-码范式（domain-key normal form）
- 使用这些一般化的约束的一个实际问题是，它们不仅难以推导，而且也还没有形成一套具有正确有效性和完备性的推理规则用于约束的推导
- 因此，很少使用

数据库设计过程

- 假设给定了模式R
  - R 可能是经转换E-R图到表而生成的
  - R 可能是单个包含所有属性的关系（称为全关系）
  - 规范化将R 分解成较小的关系
  - R 可能是某种特定设计的结果，然后再测试/转换成范式

E-R模型和规范化

- 当我们小心地定义当E-R图，并正确地标识所有实体，则从E-R图生成的关系模式就不需要太多进一步的规范化
- 但是，在现实(不完善的)设计中可能存在从实体的非码属性到其他属性的FD
  - 例如：*employee* 实体具有属性 *department_name* 和*building*，以及FD *department_name* $\rightarrow$ *building*
  - 好的设计应该将*department* 作为实体
- 从联系集的非码属性引出FD是可能的，但极少——多数联系是二元的

为了性能去规范化

- 当我们小心地定义当E-R图，并正确地标识所有实体，则从E-R图生成的关系模式就不需要太多进一步的规范化
- 为提高性能可能使用非规范化的模式
- 例如：假定每次访问一门课程时，所有的先修课都必须和课程信息一起显示，需将*course*和*preq*连接
- 做法1：使用包含*course*以及*preq*的所有上述属性的反规范化关系
  - 查找迅速
  - 额外空间以及额外更新执行时间
  - 程序员的额外编码工作以及更多出错可能性

为了性能去规范化

- 做法2：使用如下定义的物化视图

$$course \bowtie prereq$$

  - 优缺点同上，除了程序员的额外编码工作和避免可能的错误

其他设计问题

- 有些数据库设计问题不能被规范化解决
- 应避免的坏的数据库设计：
  - 例1，不用*earnings*(*company-id*, *year*, *amount*)，而是用具有同样模式(*company-id*, *earnings*) 的*earnings-2000*, *earnings-2001*, *earnings-2002*，等等
  - 这属于BCNF，但使得跨年度查询很困难，并且每年都需要新表
  - 例2，*company-year*(*company-id*, *earnings-2000*, *earnings-2001*, *earnings-2002*)
  - 也属于BCNF，但每年都需要添加新的属性
  - 这是一个crosstab的例子，将属性值作为了列名
  - 可用于spreadsheets，以及数据分析工具

总结

- 我们概述了一个将关系分解成BCNF的算法，有一些关系不存在保持依赖的BCNF分解
- 用正则覆盖将关系分解成3NF，它比BCNF的条件弱一些。属于3NF的关系也会含有冗余，但是总存在保持依赖的3NF分解
- 介绍了多值依赖的概念，它指明仅用函数依赖无法指明的约束
- 用多值依赖定义了4NF
- 其他的范式，如PJNF和DKNF，消除了更多细微形式的冗余。但是，它们难以操作而且很少使用
- 数据库设计过程中，要考虑的规范化问题

谢谢！