China-pub.com
下载

第11章 进一步规范化 I：1NF、2NF、3NF和BCNF

11.1 引言

本书到目前为止一直使用供应商-零件数据库作为例子，它的逻辑设计如下：

```
S   {S#, SNAME, STATUS, CITY}
    PRIMARY KEY{S#}
P   {P#, PNAME, COLOR, WEIGHT, CITY}
    PRIMARY KEY{P#}
SP  {S#, P#, QTY}
    PRIMARY   KEY{S#,P#}
    FOREIGN   KEY{S#}   REFERENCES   S
    FOREIGN   KEY{P#}   REFERENCES   P
```

现在需要考察该设计的正确性。很显然，这三个关系变量是必需的，并且供应商的地址(CITY)属于关系变量S，零件的颜色(COLOR)属于关系变量P，发货量(QTY)属于关系变量SP。那么我们是通过什么了解这些内容呢？通过考察改变设计的方式所引起的变化，会对这个问题有所了解。例如，假设把供应商的地址从供应商的关系变量 S中移到发货关系变量 SP中(直觉上这是错误的，因为很明显，“供应商的地址”只和供应商有关，而和发货无关)。图11-1和第10章的图10-1稍有不同，展示了修订过的发货关系变量的一个实例。注意：为了和原来的SP相区分，和第10章一样，把修订过的发货关系变量称为 SCP。

SCP

S#	CITY	P#	QTY
S1	London	P1	300
S1	London	P2	200
S1	London	P3	400
S1	London	P4	200
S1	London	P5	100
S1	London	P6	100
S2	Paris	P1	300
S2	Paris	P2	400
S3	Paris	P2	200
S4	London	P2	200
S4	London	P4	300
S4	London	P5	400

图11-1 关系变量SCP的实例

对该图稍作研究就可以看出其中的不足：即冗余。例如： SCP中每一个供应商 S1的元组都说明S1居住在London，每一个供应商 S2的元组都说明S2居住在Paris，等等。事实上，一个供应商提供几种零件，就有几个元组存放关于他的地址的信息。冗余将导致许多深层次的问题。如：修改后有可能在一个元组中供应商居住在 London，而在另外一个元组中却认为同一供应商居住在 Amsterdam[⊖] ，因此最好的方法或许是“一事一地”(即避免冗余)。规范化的

⊖ 在本章及下一章中，有必要作这样一个假设（非常现实）：关系变量中的谓词不能被全部实现 —— 因为如果它们能够被全部实现，则不会出现上述问题（不可能只改了其中的部分供应商是 S1的元组的地址而不修改供应商是S1的其他元组），事实上可以从另外一个角度考虑规范化要求：用一种方法构造数据库，使得单元组的更新比其他条件下（如没有完全规范化）的单元组更新在逻辑上更能被接受。因为完全规范化的关系元组的谓词将非常简单，所以可以通过规范化实现上述目的。

China-pub.com
下载

主要思想实际上就是使一些简单概念形式化，然而在数据库设计中，“形式化”确实具有实际应用价值。

当然，就像第5章所见一样，关系模型所涉及的关系都是规范化的。至于关系变量，只要它的合法实例是规范化的关系，则该关系变量就是规范化的，这样，关系模型所涉及的关系变量也都是规范化的。也就是说，关系变量（和关系）总是属于第 1范式（简称1NF）的。用另一句话说，规范化和1NF的意思是完全一样的，虽然“规范化（normalized）常常用来表示更高级别的标准（典型的是第3范式：3NF），后一种用法比较不规范但很常见。

这样，根据前面的说法，一个给定的关系变量有可能虽然是规范化的，但仍具有一些不受欢迎的性质，关系变量 SCP就是一个例子（见图 11-1）。规范化理论使人们认识这些问题，并用一些更好的关系变量代替原来的关系变量。例如：在关系变量 SCP中，运用规范化理论可以知道该关系变量存在一些什么“毛病”，并知道如何用两个更好的关系变量：一个含有属性{S#，CITY}，另一个含有属性{S#，P#，QTY}，来代替这个关系变量。

1. 范式

“进一步规范化（further normalization）”（在下文中简写为“规范化”）过程是建立在“范式”这一概念之上的。一个关系变量满足某一个范式所规定的一系列条件时，它就属于该范式。例如，如果一个关系变量属于 1NF，且满足11.3节介绍的条件时，它是属于第 2范式（2NF）的。

图11-2展示了一些范式及它们之间的关系，前三个范式（ 1NF、2NF和3NF）由Codd[10.4]定义，从图11-2可以看出，所有的规范化关系变量都属于 1NF，有一些属于1NF的关系变量还属于2NF，一些属于2NF的关系变量还属于3NF。Codd定义这些范式的原因是，他认为一个属于2NF的关系变量比一个只属于 1NF的关系变量更“好”；同样，一个属于 3NF的关系变量比一个只属于 2NF的关系变量更“好”。因此，在数据库设计中，一般应设计属于3NF的关系变量，而不是设计只属于 1NF或2NF的关系变量。

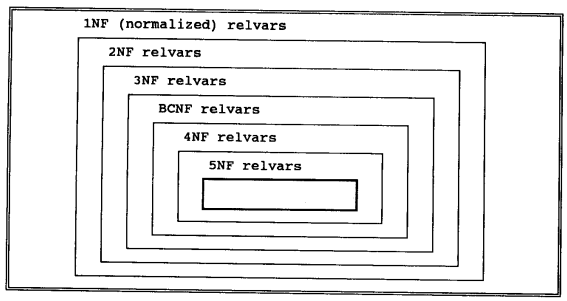

图11-2 不同级别的范式

[10.4]中还介绍了过程化思想，即所谓的规范化过程，通过它可以用一系列更“好”的关系变量（如属于3NF的关系变量）代替一个“不好”的关系变量（如只属于2NF的关系变量）。当然该过程最初定义时只适用到 3NF，但正如下一章即将看到的，该过程经过后来的扩展可以适用到5NF。该过程可以归纳为：通过一连串的归约，把一组给定的关系变量转化为一些更“好”的形式。应该注意的是该过程是“可逆”的：即总可以把该过程的输出（比方说属于3NF的关系变量的集合）重新映射到它的输入（比方说属于 2NF的关系变量的集合）。可逆性是很重要的，因为它意味着在规范化过程中是没有信息丢失的。

China-pub.com 下载

回到范式这个主题来：在第11.5节将会看到，Codd的最初关于3NF的定义[10.4]存在不足，Boyee和Codd一起，对原来的3NF作了一些修正，提出了更加严格的定义[11.2]——即那些属于修正后3NF的关系变量必然属于原来的3NF，但是一些属于原来的3NF的关系变量可能不属于修正后的3NF。为了和原来的3NF相区分，把修正后的3NF范式被称为Boyee/Codd范式（BCNF）。

后来，Fagin[11.8]提出了“第4”范式（4NF——之所以称为“第4”是因为那时BCNF仍被称为“第3”）。在[11.9]中，Fagin还定义了投影-连接范式（PJ/NF，又被称为“第5”范式或5NF）。如图11-2所示，一些属于BCNF的关系变量同时也属于4NF，而一些4NF的关系变量同时也属于5NF。

现在，读者也许会问：该过程是否有终点？是否有6NF、7NF，以至无穷的范式？虽然这是一个值得一提的问题，但本书不打算对这个问题作详细的讨论。除了图11-2所示的范式外，确实存在其他的范式，但是，从某种意义上说，5NF是最“后”的范式。这一问题将在第12章讨论。

2. 本章结构

本章主要是分析规范化概念——从1NF到BCNF（另外两个范式在第12章分析），本章安排如下：引言之后，在第11.2节讨论基本概念——无损分解，并证明函数依赖对这个概念的重要性（事实上，函数依赖构成了Codd三个范式，包括BCNF范式的基础）。第11.3节介绍前三个范式的原型，并通过例子说明一个关系变量如何经过一步一步的规范化而最终成为3NF的关系变量。11.4节介绍分解选择（alternative decomposition）——即如果存在多种分解方法，如何选择最“好”的一种。接下来，在11.5节讨论BCNF。最后在11.6节作个总结并提出一些结论性评价。

希望不要把后面介绍的内容教条化，相反，在实际工作中要在很大程度上依靠直觉。事实上，像无损分解、BCNF等术语虽然有些深奥，但是它们应该是简单的常识。一些参考文献介绍这部分内容时太正式、古板。可以从[11.5]中找到较好的学习材料。

最后，给出两个指导性的评论：

1) 前面已经提到，规范化的主要思想是：数据库设计人员在设计数据库时，他所设计的数据库的关系变量应该是属于“最终”范式（5NF）的。然而，不应把这个建议当作定则，因为在实际工作中，常常会有很多理由需要忽略规范化理论（见本章后面的习题11.7）。事实上，这也说明数据库设计是个极其复杂的工作（最起码“大型”数据库设计如此，一些小数据库的设计往往是相当简单的）。在数据库设计中，规范化理论是很有用的，但它不是灵丹妙药；因此，任何一个设计数据库的人都应该熟悉规范化理论，但这并不意味着必须只依据这一个理论。在第13章将讨论数据库其他方面的设计，这些设计和规范化理论很少甚至没有联系。

2) 前面已经提到，本章将把规范化过程作为介绍和讨论不同范式的基础，然而，这并不是说实际的数据库设计工作一定是运用这一过程实现的，事实上很有可能运用将在第13章介绍的自上而下方案而不运用本章介绍的规范化过程。规范化思想可以用来验证设计结果是否在无意中违反了规范化理论。然而，规范化过程确实为描述规范化理论提供了一个方便的框架。本章为了说明问题，假定在设计过程中采用这个过程。

11.2 无损分解和函数依赖

在开始具体讨论规范化过程之前，有必要仔细分析这一过程的一个重要性质：分解的无

China-pub.com
下载

损性（无损分解）。规范化过程涉及到把一个关系模式分解成几个关系模式，而且这种分解是“可逆”的，这样在分解过程中不会有信息丢失。也就是说，人们感兴趣的是没有信息丢失的分解。从后面的分析可以看出：一个模式分解是否是无损的这个问题和函数依赖密切相关。

以大家熟悉的关系变量S为例，该关系变量的头部为{S#，STATUS，CITY}（为简单起见，忽略SNAME）。图11-3的上部是该关系变量的一个实例（关系），标有（a）和（b）的是该关系的两种不同分解。

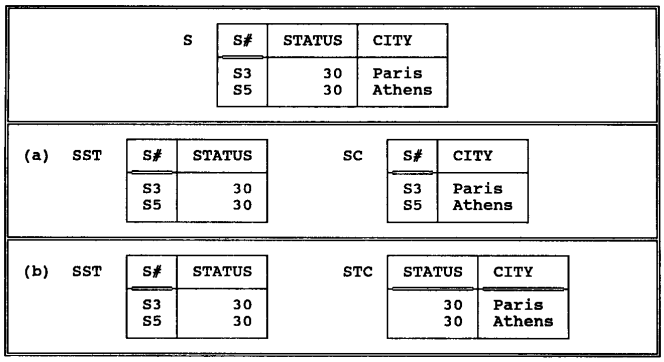

图11-3 关系变量S的一个实例及相应的两个分解

仔细分析这两种分解，可以看出：

1) 在（a）中，没有任何信息丢失。从SST和SC的值仍旧可以导出供应商S3的状态是30，地址是Paris，而供应商S5的状态是30，地址是Athens。也就是说，该分解确实是无损的。

2) 相反，在(b)中却有信息丢失，从分解后的关系中还是可以导出两个供应商的状态是 30，但不知道供应商地址，即第二个分解不是无损的，而是有损的。

到底是什么原因使得第一个分解是无损的，而第二个分解是有损的呢？通过观察，首先可以发现，“分解”过程实际上是个投影过程，图中的 SST、SC和STC都是原来的关系变量 S的一个投影，所以规范化过程中的分解实际上就是投影。在本书第二部分已经提到，人们经常说的“SST是关系变量S的投影”，更确切地说，应该是“关系变量 SST在任何时候的值都是关系变量S在相应时间的值的投影”。

第二，“在（a）中是没有信息丢失的”是指 SST和SC经过自然连接可以得到原来的 S，相反，在（b）中，SST和STC经过连接却不能得到原来的 S，所以说在分解（b）中丢失了信息[⊖]。更精确地说，“可逆”的意思就是原来的关系变量等于它的投影的连接。所以，就像规范化过程中的分解是投影操作一样，“重组（recomposition）”就是连接操作。

有一个有意思的问题：如果*R1*和*R2*是关系变量*R*的投影，且*R1*和*R2*属性集的并集包含*R*的所有属性，那么，*R1*和*R2*应满足什么条件，才能保证*R1*和*R2*的连接能得到原来的*R*？这就是函数依赖的用武之地。回到例子中来，关系变量 S满足最小函数依赖集：

S#→STATUS
S#→CITY

⊖ 更精确地说，是得到了原来S中所有的元组，同时也得到了一些“假”元组；事实上不可能得到比原来 S中更少的元组（练习：证明这个论点），因为不有从这些元组中辨别哪些是“假”的，哪些是“真”的，这样就丢失信息了。

China-pub.com 下载

如果只是说关系变量S满足这些函数依赖，还不能说明为什么关系变量S和它的投影{S#，STATUS}和{S#，CITY}的连接等价，事实上还有如下定理（Heath定理[11.4]）：

- Heath定理：假设有一个关系变量$R\{A，B，C\}$，A、B和C是属性集。如果关系变量R满足函数依赖$A\rightarrow B$，则R和投影$\{A，B\}$、$\{A，C\}$的连接等价。

把A看作S#，把B看作STATUS，把C看作CITY，该定理就可以证明关系变量S可以无损分解成它的投影{S#，STATUS}和{S#，CITY}。

同时我们知道关系变量S不能无损分解成它的投影{S#, STATUS}和{STATUS, CITY}。Heath定理不能解释为什么如此㊀，但是，可以直观地看出，在后面的分解中丢失一个函数依赖，即在后一个分解中虽然仍然满足函数依赖S#→STATUS，但不能满足函数依赖S#→CITY。

进一步了解函数依赖

下面以一些关于函数依赖的评论结束本节。

1) 左部不可约（left-irreducible）的函数依赖：第10章说过，一个左部不可约的函数依赖是指函数依赖的左边不“太大”。例如，11.1节的关系变量SCP满足函数依赖：

```
{S#，P#}→CITY，
```

然而，属性P#在该函数依赖的左边是多余的，也就是说，该关系变量还满足函数依赖：

```
S#→CITY
```

（即CITY还函数依赖于S#）。后一个函数依赖是左部不可约的，但前面那个却不是，即CITY不可约地依赖于S#，而不是不可约地依赖于{S#，P#}㊁。

左部不可约函数依赖和不可约依赖在第2范式和第3范式中是一个比较重要的概念[见11.3节]。

2) 函数依赖图：假设R是关系变量，I是R的不可约的函数依赖集（要了解不可约函数依赖集可参阅第10章）。可以用函数依赖图方便地表达函数依赖集I，图11-4给出了关系变量S、P和SP的函数依赖集，在本章的后面部分将经常引用该图。

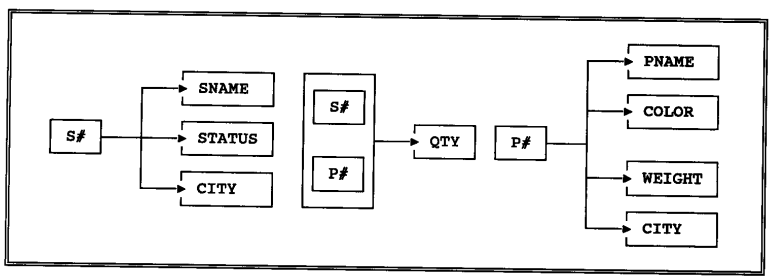

图11-4 关系变量S、P和SP的函数依赖图

可以看出，图11-4的每一个箭头都是从相关的关系变量的候选码（常常是主码）指出。根据定义，每一个候选码都有箭头指出㊂，因为对于每一个给定的候选码的值，其他属性都

㊀ 这和该定理陈述的形式有关，该定理只说明了“如果……那么……”，而不是说“当且仅当……那么……”（见本章的面的习题11.1）。在12.2节中将介绍一个更严格的Heath定理。

㊁ “左边不可约依赖”和“不可约依赖”，也常叫作“完全函数依赖”或“完全依赖”，在本书的其他早期版本中也这么叫。后面这种叫法虽然简洁，但不切题。

㊂ 更精确地说，每一个超码都有箭头指出。然而，如果该函数依赖集是不可约的，那么，所有的函数依赖（或箭头)是左边不可约的。

China-pub.com
下载

有一个唯一的值和它对应，所以这些箭头是不可少的。如果图中还存在其他的箭头，就会带来问题。所以规范化过程就是消除那些不是从候选码中指出的箭头。

3) 函数依赖是语义上的概念：函数依赖是一种特殊的完整性限制条件，所以它们也是一个语义上的概念。理解函数依赖是理解数据意义过程的一部分，例如，关系变量 S满足函数依赖S#→CITY，就是指每一个供应商的地址是唯一的。还可以从以下几个方面理解这一点：

- 在现实世界中存在这种在数据库中体现的限制条件，即每一个供应商的地址是唯一的。
- 因为这些限制是现实世界的语义表述的一部分，所以在数据库中应该得到遵守。
- 确保遵守这些限制的方法是在数据库定义时声明这些限制条件，这样，数据库就可以实现它们。
- 在数据库中声明这些限制条件的方法是定义函数依赖。

从后面的介绍中可以发现，规范化使得定义函数依赖变得非常简单。

11.3 第一、第二和第三范式

注意：为了简单起见，在本节中假定每个关系变量只含有一个候选码，并进一步假定该候选码为主码，这个假定在不太严格的定义中多次得到体现。含有多个候选码的关系变量在11.5节讨论。

现在开始介绍Codd的三个范式。为了说明问题，首先给出一个初步的、很不规范的 3NF的定义，然后讨论把任意一个关系变量转化成一个 3NF的关系变量集的过程，最后给出这三个范式的精确定义。然而可以发现， 1NF、2NF和3NF除了是BCNF及其他范式的基础外，本身的意义并不大。

下面是3NF的初步定义：

- 第三范式（非常不正式）：当且仅当关系变量的非码属性满足下列条件时，该关系变量是属于3NF的：

  （a）相互独立，且

  （b）完全依赖于主码。

关于“非码”和“相互独立”的解释如下：

- 一个“非码”是指不属于所讨论的关系变量的主码的属性。
- 两个或多个属性“相互独立”是指其中的任何一个属性都不函数依赖于其他属性的组合。这种独立性意味着其中的任何一个属性都可以独立地被修改。

作为例子，根据前面的定义，关系变量 P是属于 3NF的。属性 PNAME、COLOR、WEIGHT和CITY是相互独立的（可以更新其中的任何一个属性如 COLOR而不必同时改变其他属性如WEIGHT），并且这些属性都完全函数依赖于主码 {P#}。

上述关于3NF的定义可以用下面的更不正式的定义解释：

- 第三范式（更不正式）：当且仅当一个关系变量在任何时候，它的每一个元组都含有一个主码来识别实体，同时有一组零个或更多的相互独立的属性从不同方面描述这个实体时，该关系变量是属于3NF的。

同样，关系变量P符合这个定义：P中的每个元组有一个主码值（零件号码）来识别现实世界的一些零件，同时有四个不同的属性（零件名称、零件重量、零件颜色和零件产地），它

China-pub.com 下载

们都用来描述零件，且各自独立于其他属性。

现在，回到规范化过程上来，首先介绍第一范式的定义：

- 第一范式：当且仅当一个关系变量的所有的合法的值中，每一个元组的每个属性只含有一个值时，该关系变量属于1NF。

该定义只是说明所有的关系变量都是属于1NF的，这当然是正确的。然而，如果一个关系变量只属于第一范式（即是1NF关系变量而不是2NF关系变量，更不是3NF关系变量），则有很多理由认为它不是一个“好”的结构。为了说明这个问题，假设把关于供应商和发货的信息合在一起形成一个新的关系变量FIRST，而不是把它们分开来形成两个关系变量S和SP，关系变量SCP的定义如下：

```
FIRST {S# STATUS, CITY, P#, QTY}
      PRIMARY  KEY{S#,P#}
```

该关系变量是11.1节的SCP的扩展。除了为了说明问题而引进的附加的约束条件

```
CITY→STATUS
```

外，每一个属性都和原来的意义一样。STATUS函数依赖于CITY，表示供应商的状态由它的地址决定，如每一个居住在London的供应商的状态是20。同时，为了简单起见，省略了SNAME。FIRST的主码是{S#，P#}，函数依赖图见图11-5。

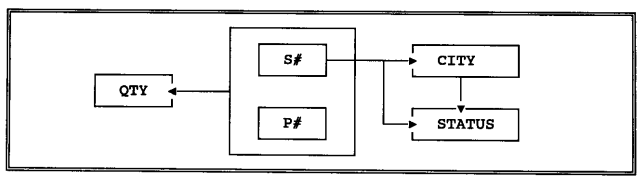

图11-5 关系变量FIRST的函数依赖图

可以看出，该关系变量的函数依赖图比属于3NF的关系变量的函数依赖图复杂，一个3NF关系变量的函数依赖图中只有从候选码中出来的箭头，而一个非3NF关系变量（如FIRST）的函数依赖图除了从候选码出来的箭头外，还有其他箭头——正是这些箭头造成了麻烦。实际上，关系变量同时违反了前面关于3NF定义中的条件（a）和（b）：非码属性不是相互独立的，因为STATUS依赖于CITY（新增加的箭头），并且非码属性不是完全依赖于主码，因为STATUS和CITY都只依赖于S#。

为了说明多余箭头所带来的麻烦，图11-6列出了关系变量FIRST的一个实例。为了和新增加的约束条件——CITY决定STATUS——保持一致，把供应商S3的STATUS由30改为10，其他属性的值和原来的一样。图中的冗余是很明显的。例如每一个S1的元组的CITY都是London，而每一个CITY是London的元组的STATUS都是20。

FIRST中的冗余会造成一系列的更新异常——即在INSERT（插入）、DELETE（删除）和UPDATE（更新）等更新操作所造成的异常。首先看和函数依赖S#→CITY相对应的供应商-地址间的冗余，在进行更新操作时都会出现问题：

- INSERT（插入）：不能插入一个居住在某个城市却没有零件供应的供应商，事实上，在图11-6中就没有居住在Athens的供应商S5，其原因是，除非S5至少供应一种零件，否则没有合适的主码（和第9章9.4节一样，本章假定主码的属性没有默认值）。
- DELETE（删除）：删除FIRST中的某个元组时，不仅删除了一个供应商的某种零件的发

China-pub.com
下载

货，而且有可能把该供应商的其他信息（如地址）丢失。例如：把图 11-6中的主码是{S3，P2}的元组删除，则关于S3的地址是Paris的信息就丢失了（INSERT和INSERT问题事实上就像一个硬币的两面）。

FIRST

S#	STATUS	CITY	P#	QTY
S1	20	London	P1	300
S1	20	London	P2	200
S1	20	London	P3	400
S1	20	London	P4	200
S1	20	London	P5	100
S1	20	London	P6	100
S2	10	Paris	P1	300
S2	10	Paris	P2	400
S3	10	Paris	P2	200
S4	20	London	P2	200
S4	20	London	P4	300
S4	20	London	P5	400

图11-6 关系变量FIRST 的实例

注意：引起这些问题的关键是 FIRST把太多的信息绑在一起，当从中删除一个元组时，删除了太多的信息。更精确地说， FIRST中同时包含了有关供应商的信息和有关发货的信息，当删除有关发货的信息时，同时也删除了有关供应商的信息。解决这个问题的方法当然是“分解”——即把发货的信息放在一个关系变量中，而把有关供应商的信息放在另一个关系变量中。这样，就可以把规范化过程概括为“分解”过程：把逻辑上独立的信息放在独立的关系变量中。

- UPDATE（更新）：一般来讲，给定供应商的地址在FIRST会出现多次，这就会给更新带来困难。例如，如果供应商S1从London搬到Amsterdan，这时，我们就面临这样的选择：要么找出FIRST中所有和供应商S1有关的元组，并把地址改为 Amsterdan，要么就保留一个可能不一致的结果（S1的地址有可能在一个元组里是 Amsterdan，而在另一个元组里是London）。

解决这些问题的方法前面已经提到，就是用关系变量：

```
SECOND{S#, STATUS, CITY}
```

和

```
SP{S#, P#, QTY}
```

代替原来的关系变量FIRST。

关系变量SECOND、SP的函数依赖见图11-7，这两个关系变量的实例见图 11-8。可以看出，供应商S5的信息已被包含在内（只在关系变量 SECOND中，在关系变量SP中却没有），而关系变量SP和往常的发货关系变量已完全一样。

应该清楚，修改过的结构克服了前面描述的所有问题：

- INSERT（插入）：即使S5没有供应一种零件，通过把合适的元组插入关系变量 SECOND中就可以插入居住在Athens的供应商S5的信息。
- DELETE（删除）：可以在关系变量SP中删除有关S3和P2的元组，却不会丢失S3的地址是Paris的信息。
- UPDATE（删除）：在修改过的结构中，给定供应商的地址只出现一次，而不是多次，因为在关系变量SECOND（主码是S#）中，给定供应商只有一个元组。也就是说， S#-CITY间的冗余已被消除。在SECOND元组中对S1的地址作一次修改，就可以把 S1的地

China-pub.com 下载

址由London改为Amsterdam。

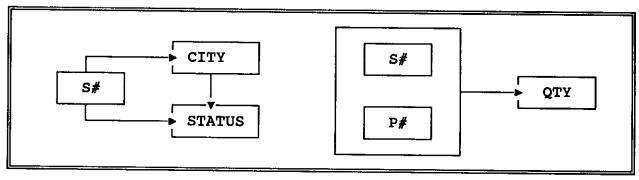

图11-7 关系变量SECOND和SP的函数依赖图

SECOND

S#	STATUS	CITY
S1	20	London
S2	10	Paris
S3	10	Paris
S4	20	London
S5	30	Athens

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

图11-8 关系变量SECOND和SP的实例

比较图11-5和图11-7可以看出，关系变量FIRST经模式分解分解为SECOND和SP的作用是消除不完全函数依赖，也正是消除了这些不完全函数依赖才解决了前面描述的问题。也可以说，在关系变量FIRST中，属性CITY描述的不是由主码所确定的实体的信息，即发货，而是和发货有关的供应商（当然，属性STATUS也是如此）的信息。正是把这两种信息混在一起才导致了前面的问题。

下面给出第二范式的定义[㊀]：

- 第二范式：（假定只有一个候选码，且该候选码是主码）当且仅当一个关系变量属于1NF，且该关系变量的每一个非码属性都完全函数依赖于主码时，该关系变量属于2NF。

关系变量SECOND和SP都是2NF的（主码分别是{S#}和{S#，P#}）。关系变量FIRST不是属于2NF的，一个不属于1NF和2NF的关系变量总是可以归约成与之等价的2NF的关系变量的集合。这个归约过程包括用适当的投影替代原来的关系变量，因为这些投影经过连接可以得到原来的关系变量，所以这些投影和原来的关系变量是等价的。在我们的例子中，关系变量SECOND和SP是FIRST的投影[㊁]，而FIRST是SECOND和SP的自然连接。

规范化过程的第一步可以归纳为利用投影消除“非不可约的”函数依赖，即给定如下关系变量：

```
R { A, B, C, D }
    PRIMARY KEY {A, B}
  /*假定满足A→D*/
```

根据规范化理论，可以用下列两个关系变量替代原来的关系变量：

㊀ 严格地说，2NF应该根据一个给定的函数依赖集定义，但在不太正式的上下文中可以忽略，这也适用于其他范式（当然不包括第一范式）。

㊁ 在SECOND中可以含有FIRST中没有出现的元组（如图11-8中的供应商S5）这个事实除外。也就是说，新的结构可以表达原来的结构中不能表述的信息，从这种意义上说，新的结构更能真实地反映现实世界。

China-pub.com
下载


```
R1 {A, D}
   PRIMARY  KEY{A}

R2 {A, B, C}
   PRIMARY  KEY{A, B}
   FOREIGN  KEY{A}  REFERENCES R1
```

利用外码-主码的连接可以从*R1*和*R2*重新得到*R*。

回到前面的例子，SECOND-SP结构仍然存在问题，然而，SP是符合要求的。事实上，SP已经是3NF了，在本节可以忽略。但是，SECOND因属性间缺乏独立性，仍然会产生问题。SECOND的函数依赖图仍然比3NF关系变量复杂，更确切地说，STATUS依赖于S#虽然是完全函数依赖，但是它是不可约的传递函数依赖（通过CITY）：每一个S#决定一个CITY，而每一个CITY决定一个STATUS。一般来说，就像第10章介绍的那样，只要$A \rightarrow B$和$B \rightarrow C$都能满足，则必有传递函数依赖：$A \rightarrow C$。而传递依赖一样也会造成更新异常（现在把注意力集中在和函数依赖CITY→STATUS相关的CITY-STATUS的冗余上）。

- INSERT（插入）：不能插入一个没有供应商却具有状态的地址——即不能插入这样的信息：任何居住在Rome（CITY）的供应商的状态（STATUS）为50，除非已经有一个居住在Rome的供应商。
- DELETE（删除）：当在SECOND中删除与某一城市有关的元组时，不仅删除了与该城市有关的供应商的信息，而且删除了与该城市有关的状态（STATUS）信息。例如，在SECOND中删除S5的元组，则同时丢失Athens的状态是30的信息（同样，DELETE（删除）和INSERT（插入）是同一硬币的两面）。

  注意：这个问题同样是由于“捆绑”：关系变元SECOND中同时含有与供应商和城市有关的信息。同样解决这些问题的方法是“分解”，即把供应商的信息放在一个关系变元中，而把关于城市的信息放在另外一个关系变元中。
- UPDATE（修改）：某一城市的状态在SECOND中有可能出现多次（该关系变量仍存在冗余）。因此，如果要把London的状态由30改为20，则同样面临这样的选择：在SECOND中查找所有与London有关的元组，并把状态（STATUS）由30改为20或则得到一个有可能不一致的结果（在一个元组中London的状态是30，而在另一个元组中London的状态是20）。

再一次用投影代替原来的关系变量（本例中是SECOND）以解决这些问题，即用投影：SC {S#，CITY}和CS {CITY，STATUS}代替SECOND {S#，CITY，STATUS}。关系变量SC和CS的函数依赖图见图11-9，图11-10是这两个关系变量的一个实例。同样，这种归约是可逆的，因为SECOND是SC和CS通过CITY的自然连接。

同样应该很清楚，这种修订过的结构克服了前面描述的更新问题，该问题的具体讨论作为练习。和图11-7、图11-9相比，可以看出进一步分解的作用是消除STATUS对S#的传递函数依赖，也正是消除了这种传递函数依赖才解决了更新异常问题。直观地说，在关系变量SECOND中，属性STATUS所描述的并不是由主码所标识的实体，即供应商的信息，而是供应商的地址的信息。同样，把这两种信息放在同一关系变量中会带来问题。

下面给出第三范式的定义：

- 第三范式（假定关系变量只有一个候选码，且该候选码是主码）：当且仅当一个关系变

China-pub.com 下载

量属于2NF且该关系变量的所有非码属性都不传递依赖于主码时，该关系变量属于 3NF。注意：“不传递依赖”蕴涵不互相依赖，从这个意义上说，该术语的解释和本节开始的解释一样。

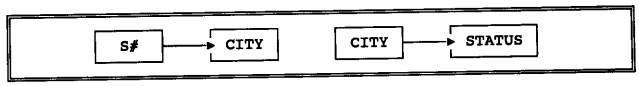

图11-9 关系变量SC和CS的函数依赖图

SC

S#	CITY
S1	London
S2	Paris
S3	Paris
S4	London
S5	Athens

CS

CITY	STATUS
Athens	30
London	20
Paris	10
Rome	50

图11-10 关系变量SC和CS的实例

关系变量SC和CS都属于3NF（它们的主码分别是{S#}和{CITY}）。关系变量SECOND不属于3NF。一个属于2NF但不属于3NF的关系变量总是可以归约成一些属于3NF的关系变量的集合。前面已经提到，这种归约是可逆的，即在归约中是没有信息丢失的，然而归约后的 3NF关系变量集中含有一些在原来的2NF关系变量中不能描述的信息，如Rome的状态是50[⊖]。

规范化过程的第二步可以归纳为利用投影消除非码属性间的传递函数依赖，也就是说，给定关系变量：

```
R {A, B, C}
   PRIMARY  KEY{A}
    /*假定满足函数依赖：B→C */
```

根据规范化理论，用如下两个关系变量 *R1*、*R2*替代原来的关系变量 *R*：

```
R1 {B, C}
    PRIMARY  KEY{B}
R2 {A, B}
    PRIMARY  KEY{A}
    FOREIGN  KEY{B}  REFERENCES R1
```

利用外码和主码相匹配机制，*R1*和*R2*通过连接可以重新得到 *R*。

最后应该强调一点，一个给定关系变量的规范化级别是语义问题，而不仅仅是个与关系变量在某一特定时间的值有关的问题。也就是说，不能只观察一个关系变量的一些实际值就判断该关系变量是否是属于3NF的，而必须通过了解数据的意义，如函数依赖等才能作出判断。还应该注意的是即使知道这些函数依赖，也不能通过分析一些给定的数据就判断该变量是否是属于3NF的，通过这些具体的数据最多只能判断所讨论的数据是否违反了这些函数依赖，如果没有违反，则这些数据和该关系变量属于3NF的假设是一致的，当然，这不能保证这种假设是正确的。

11.4 保持函数依赖

在归约过程中，常常会出现这样的问题，即一个给定的关系变量可以有不同的无损分解

⊖ 正如关系变量的组合 SECOND-SP比属于1NF但不属于2NF的关系变量 FIRST更能表现现实世界一样，关系变量组合SC-CS比属于2NF但不属于3NF的关系变量SECOND更能表现现实世界。

China-pub.com
下载

方法。还是以11.3节的关系变量SECOND为例，该关系变量满足函数依赖S#→CITY和CITY→STATUS及传递函数依赖S#→STATUS（参考图11-11，该图中传递函数依赖以虚线箭头表示）。11.3节的分析表明，该关系变量的更新异常可以通过模式分解把它分解成下面两个属于3NF的关系变量得到解决：

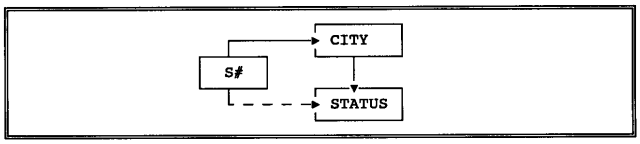

图11-11 关系变量SECOND的函数依赖图

```
SC {S#, CITY}
CS {CITY, STATUS}
```

假设把这种分解叫作“分解*A*”。同样还有另外一种分解（“分解*B*”）：

```
SC{S#, CITY}
SS{S#, STATUS}
```

在这两种分解中，投影SC是一样的。分解*B*也是无损的，而且它的两个投影都是属于3NF的，但有许多原因可以认为分解*B*不如分解*A*“好”。例如在分解*B*中，还是不能插入某个地址具有某种状态的信息，除非恰好有一个供应商居住在这个城市。

下面仔细分析这个例子。首先应该注意，分解*A*的投影和图11-11的实箭头相符，而分解*B*中的一个投影和图11-11的虚箭头相符。事实上，在分解*A*中的两个投影是相互独立的，只要对某个投影的更新在上下文是合法的——只要不违反所更新投影的主码唯一性限制条件，它们就可以单独更新而不用考虑其他投影[⊖]，这样，更新后的投影经过连接仍能得到合法的SECOND（即连接不会违反SECOND的函数依赖）。相反，在分解*B*中，为了保证不违反函数依赖CITY→STATUS（如果两个供应商的地址一样，则它们的状态必须一样），对其中任何一个投影更新时都要对两个投影实行监控（考虑把供应商S1的地址由London改为Paris应作些哪些更新）。也就是说，在分解*B*中，两个投影不是相互独立的。

用第8章的话来讲，这个问题的关键是函数依赖CITY→STATUS变成了一个跨越两个关系变量的数据库限制条件（这意味着在今天的一些产品中必须用过程来维护）。相反，在分解*A*中是传递函数依赖S#→STATUS成了数据库限制条件，而这种数据库限制条件只要S#→CITY和CITY→STATUS这两个关系变量级的限制条件实现后就会自动被实现，而要实现这两个关系变量级的限制条件是非常容易的，只要实现相应的主码唯一性限制条件即可。

投影独立性概念为从可能存在的多种分解方法中选择一个合适的分解提供了一个准则。具体地说，一个投影相互独立的分解比相互不独立的分解“好”。Rissanen[11.6]介绍了判断关系变量*R*的两个投影*R1*和*R2*是否相互独立的方法，指出当且仅当满足下列条件时，*R1*和*R2*是相互独立的：

- *R*中的所有函数依赖都是*R1*和*R2*的函数依赖的逻辑推论（logical consequence），并且
- *R1*和*R2*的相同属性至少组成它们之中一个的候选码。

考察前面定义的分解*A*和*B*。在分解*A*中，因为它们的的共同属性CITY是CS的主码，而且SECOND中的函数依赖不是分别出现在CS和SC中，就是SC和CS的函数依赖的推论，所以分

⊖ 当然是除了从SC到CS的参照完整性。

China-pub.com 下载

解*A*中的两个投影是相互独立的；而在分解 *B*中，尽管它们的共同属性 S#是分解后的两个投影的候选码，但是，原来的函数依赖 CITY→STATUS却不能从分解后的关系变量 SC、SS中推导出来，所以，在分解*B*中的两个投影是不互相独立的。注意：存在第三种分解可能：即用投影{S#，STATUS}和{CITY，STATUS}代替SECOND，因为这种分解不是无损的，所以是不合法的（练习：证明这个分解不是无损的）。

一个不能被分解成几个相互独立的投影的关系变量称为“原子”关系变量 [11.6]。然而应该注意，事实上，一个给定的关系变量如果不是原子的并不意味着一定要把它分解成几个原子的关系变量，例如，供应商的关系变量 S和零件的关系变量 P并不是原子的，却根本没必要进行进一步的分解，当然SP是原子的，那就更没有必要进行分解了。

规范化过程中把一个关系变量分解成相互独立的投影的思想被称为“保留函数依赖”。最后，对这个概念作更准确的解释：

1) 假设给定一个关系变量 *R*，运用规范化过程对它进行规范化 —— 用一系列的关系变量 *R1*，*R2*，…, *Rn*（都是*R*的投影）代替*R*。
2) 假设*R*的函数依赖集是*S*，而*R1*，*R2*，*R3*…, *Rn*的函数依赖集分别是*S1*，*S2*，…, *Sn*。
3) 每个函数依赖集*Si*只和投影*Ri*的属性有关(*i*=1, 2, …, *n*)，这样实现*Si*就非常简单。但是，真正要实现的是原来S中的函数依赖。这样就希望分解成*R1*，*R2*，…, *Rn*后，在实现*S1*，*S2*，…, *Sn*中的限制条件的同时就等价于实现S中的限制条件——即要求分解是保持函数依赖的。
4) 假设*S'*是*S1*，*S2*，…, *Sn*的并集，一般来讲，为了使分解是保持函数依赖的，只需 *S*和*S'*的闭包相同即可，而不必使*S'*=*S*（关于函数依赖集的闭包参考第 10.4节）。
5) 现在还没有一个有效的方法来计算函数依赖集的闭包，所以计算两个函数依赖集的闭包是否相等是不可能的。然而，现在已经有一种验证一个分解是否保持函数依赖的方法，具体算法已超出本书的讨论范围，要进一步了解可参考 Ullman的书[7.13]。

注意：本章后面的练习 11.3的答案中给出了一种算法，运用该算法可以把一个任意给定的关系变量无损分解（保持函数依赖）成一系列的 3NF的投影的集合。

11.5 BOYCE/CODD范式

现在，把前面关于每个关系变量只含有一个候选码的假设去掉，来考虑一般情况下可能出现的问题。事实上，Codd原来的3NF[10.4]没有很好地处理一般情况下的问题，更确切地说，是没有很好地处理具有下列特性的关系变量：

1) 具有一个或多个候选码，比如：
2) 候选码是复合的，并且
3) 候选码之间是重叠的（即至少有一个属性是相同的）。

3NF后来被由Boyce和Codd提出的一个更为严格的定义取代，该定义同时适用于上述情况[11.2]。然而，由于该定义比原来的 3NF定义要严格，所以有必要给它一个新的名称而不是沿用原来的3NF，这样就被叫作Boyce/Codd范式（BCNF）㊀。注意实际生活中，同时满足上述条件的情况并不常见。对于一个不满足上述条件的关系变量， 3NF和BCNF是等价的。

首先回顾一下前面的内容：决定因素是指函数依赖的左边，平凡的函数依赖指左边是右

㊀ “第三”范式的定义事实上等价于 1971年Heath在[11.4]中首次给出的BCNF范式的定义，因此，更恰当的名称应是“Heath范式”。

China-pub.com
下载

边的超集的函数依赖。下面给出BCNF的定义：

- Boyce/Codd范式：如果一个关系变量的所有非平凡的、完全的函数依赖的决定因素是候选码，则该关系变量属于Boyce/Codd范式（BCNF）。

或用一种比较不正式的定义：

- Boyce/Codd范式：（不正式的定义）如果一个关系的唯一的决定因素是候选码，则该关系变量属于Boyce/Codd范式（BCNF）。

也就是说，函数依赖图中唯一的一个箭头是从候选码中出来的。前面已经说过，每一个候选码总有箭头出来，而BCNF认为这里没有其他箭头，也就是说，在规范化过程中已没有箭头可消除。应该注意这两个BCNF的定义的区别，在非正式定义中默认：(1) 决定因素不“太大”；(2)所有的函数依赖都是非平凡的函数依赖。为了使问题简单化，除非有特别声明，在本书的剩余部分一直沿用这两个假设。

值得指出的是，从概念上讲，BCNF比3NF简单，它没有明确地引用第一范式和第二范式的概念，也没有引用传递依赖的概念。而且，BCNF的定义虽然比3NF严格，但是任何一个关系变量都可以无损分解成一系列BCNF关系变量的集合。

在讨论含有多个候选码的关系变量之前，首先讨论既不属于3NF又不属于BCNF的关系变量FIRST和SECOND，以及既属于3NF又属于BCNF的关系变量SP、SC和CS。在关系变量FIRST中，有三个决定因素：即{S#}、{CITY}和{S#，P#}，其中{S#，P#}是候选码，所以FIRST不属于BCNF。关系变量SECOND含有两个决定因素：{S#}和{CITY}，其中{S#}是候选码，所以SECOND也不属于BCNF。而关系变量SP、SC和CS属于BCNF，因为每个关系变量的唯一的决定因素是候选码。

下面考虑具有两个相互分离——即不重叠候选码的关系变量。假设在供应商关系变量S{S#，SNAME，STATUS，CITY}中，{S#}和{SNAME}都是候选码（即在任何情况下，供应商的姓名（SNAME）是唯一的）。并且假设属性STATUS和CITY是互相独立的，即在11.3节的假设CITY→STATUS不再成立，则该关系变量的函数依赖图见图11-12。

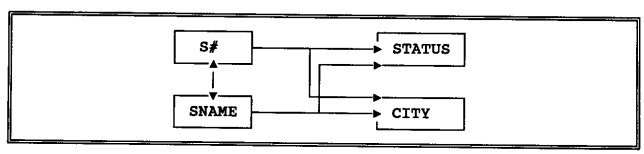

图11-12 关系变量S的函数依赖图（SNAME是候选码，且函数依赖CITY→STATUS不成立）

虽然该关系变量的函数依赖图看起来比3NF的“复杂”，但该关系变量是属于BCNF的，因为该关系变量的函数依赖图中，所有的箭头都从候选码中出来。从本例子可以看出，一个关系变量有多于一个的候选码不一定是“不好”的（当然，在数据库定义时两个候选码都要声明，以便DBMS实现它们的唯一性限制条件）。

下面讨论一个候选码相互重叠的例子。候选码相互重叠是指两个或两个以上的候选码至少含有一个以上的公共属性。为了和第8章关于该问题的讨论保持一致，在后面的例子中不打算把多个候选码中的一个作为主码，在本节的图中，也不对任何属性加下划线。

例1：假设供应商的姓名是唯一的，看下面的关系变量：

```
SSP {S#, SNAME, P#, QTY}
```

它的候选码是{S#，P#}和{SNAME，P#}。该关系变量属于BCNF吗？答案是“不是”，因为

China-pub.com 下载

S#和SNAME都是决定因素（它们互相决定，所以{S#}和{SNAME}都是决定因素），但不是候选码。图11-13是该关系变量的一个实例。

SSP

S#	SNAME	P#	QTY
S1	Smith	P1	300
S1	Smith	P2	200
S1	Smith	P3	400
S1	Smith	P4	200
..	.....	..	...

图11-13 关系变量SSP的实例

从图中可以看出，关系变量SSP和11.3节的FIRST、SECOND一样有冗余，同样会产生更新异常。例如，把供应商S1的姓名（SNAME）由Smith改为Robinson同样会出现这样的问题：或者查找所有S1的元组并修改它，或者得到一个可能不一致的结果。然而根据以前的定义，SSP是属于3NF的，因为原来的定义没有规定属于其他候选码的属性必须完全依赖于每一个候选码，所以属性SNAME不完全依赖于候选码{S#，P#}的事实被忽略了。注意，这里的3NF是指[10.4]中关于3NF的原始定义，而不是本书11.3节的简化形式。

解决该问题的方法当然是把该关系变量分解成两个投影：

```
SS{S#，SNAME}
SP{S#，P#，QTY}
```

或者

```
SS{S#，SNAME}
SP{SNAME，P#，QTY}
```

这两种分解是等价的，它们都属于BCNF。

现在，应该停下来仔细回顾本章的目的。显然，原来的只包含一个关系变量SSP的设计是“不好”的，其中的问题很明显，任何一个合格的数据库设计人员即使不了解BCNF的思想也不会提出这样的设计。常识会告诉设计人员采用SS-SP这种结构的设计会更好。但什么是“常识”呢？数据库设计人员选择SS-SP结构而不选择SSP结构的理论基础是什么呢？

答案当然是函数依赖理论和Boyce/Codd范式。也就是说，这些概念（函数依赖、BCNF，已经讨论的和即将讨论的各种范式）都是形式化的“常识”。该领域的所有定理的目的就是识别一些常识，然后把它们形式化——这当然不是一个简单的工作！但如果成功了，就可以把这些规则“机械化”：即可以编写程序，用机器去实现这些规则。规范化理论的批评者恰巧忽略了这一点，他们认为这些思想都是常识，而没有意识到给这些常识一个精确的形式化定义的重要性。

例2：考虑具有属性S、T和J的关系变量STJ（有些人可能认为该关系变量是病理方面的），我们用S表示学生，J表示课程，T表示教课程J的教师。STJ中的元组{S:*s*，J:*j*，T:*t*，}表示学生*s*选修了教师*t*教的课程*j*。该关系变量中有下列约束条件：

- 某个学生选定某门课程就对应一个固定的教师。
- 每一个教师只教一门课，而每门课有若干教师。

图11-4给出该关系变量的一个实例。

那么，该关系变量满足什么样的函数依赖呢？从第一个约束条件可以得出函数依赖：{S，J}→T，从第二个函数依赖可以得出函数依赖：T→J。因为一门课可以有多名教师，所以不能满足函数依赖J→T。所以该关系变量的函数依赖如图11-15所示。

China-pub.com
下载

SJT

S	J	T
Smith	Math	Prof. White
Smith	Physics	Prof. Green
Jones	Math	Prof. White
Jones	Physics	Prof. Brown

图11-14 关系变量STJ的实例

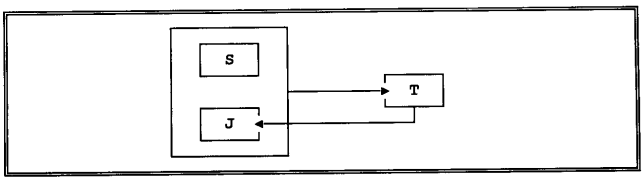

图11-15 关系变量STJ的函数依赖

同样，该关系变量中存在相互重叠的候选码，即{S，J}和{S，T}。而且该关系变量属于3NF但不属于BCNF，所以它同样会产生更新异常——例如，如果要删除Jones选修的课程physics，则会丢失教physics的教师Brown教授的信息。产生该问题的原因是属性T是决定因素，而不是候选码。同样可以把该关系变量分解成两个投影：

```
ST{S，T}
TJ{T，J}
```

练习：根据图11-14给出上述关系变量的一个实例，并写出相应的函数依赖图，证明这两个关系变量是属于BCNF的，并指出各自的候选码，检验它们是否会产生更新异常。

然而，上述分解虽然避免了更新异常，却带来了新的问题：根据Rissanen的理论，该分解的两个投影不是互相独立的，更确切地说函数依赖

```
{S，J}→T
```

不能从函数依赖

```
T→J
```

（它是这两个投影中唯一的一个函数依赖）中推导出来。结果，这两个投影不能被独立地更新。例如，不能在ST中插入元组{Smith, Brown教授}，因为Smith已经选修了Green教授教的physics课，然而系统如果不检查关系变量TJ就不能检查出这个事实。人们常常面临两难的选择：（1）把关系变量分解成属于BCNF的关系变量；（2）把关系变量分解成互相独立的关系变量，即不能同时满足两方面的要求。

注意，事实上，关系变量SJT虽然不属于BCNF，但它是原子的（见11.4节）。从而可以看出，一个“原子”关系变量不能分解成相互独立的关系变量，并不意味着它根本不能分解（指无损分解）。直观地说，“原子”并不是一个很好的术语，因为在数据库设计中，“原子”既不是充分的，也不是必要的。

例3：考察具有有属性S（学生）、J（课程）和P（名次）的关系变量EXAM，EXAM中的一个元组{S:*s*，J:*j*，P:*p*}指学生*s*的课程*j*的成绩在班里的排名是*p*，假定该关系变量满足下列约束条件：

- 在同一门课程中，任何一个学生的排名都不相同。

这样，该关系变量的函数依赖图见图11-16。

China-pub.com 下载

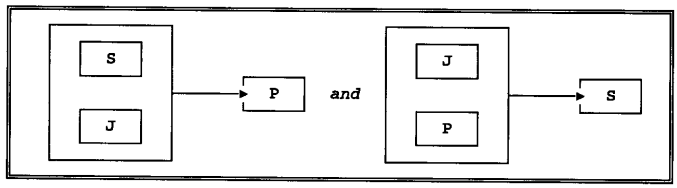

图11-16 关系变量EXAM的函数依赖图

同样，该关系变量有两个相互重叠的候选码{S，J}和{J，P}，因为（1）给定一个学生和他选修的一门课程，都有唯一的一个名次和他对应；（2）给定一门课程和名次，只有一个学生和它对应。但是，很明显，该关系变量是属于BCNF的，因为这些候选码都是决定因素，而且也不会出现前面所说的更新异常（练习：验证这个观点）。所以相互重叠的候选码并不一定会导致产生所讨论的问题。

从上面的分析可以看出，BCNF消除了一些原来的3NF定义中仍可能存在的问题，而且BCNF的定义也比3NF简洁，因为它没有涉及到1NF、2NF、主码及传递依赖等概念。而且，该定义所涉及的概念——候选码，可以用更基础的概念——函数依赖代替具体定义，见[11.2]）。另一方面，主码、传递依赖等概念在实际工作中也很重要，因为它们有助于数据库设计人员了解如何一步一步地把任意一个关系变量归约成一系列属于BCNF的关系变量的集合。

最后应该说明的是本章练习11.3的答案给出了把任意一个关系变量无损分解成一系列属于BCNF的投影集的算法。

11.6 具有关系值属性的关系变量

在第5章中已经说过，一个关系可能含有这样一个属性，这个属性的值也是一个关系（例如，图11-17）。同样，一个关系变量也可能含有一个值是关系的属性。从数据库设计的角度看，这种关系变量是不正常的，因为它们是不对称的[⊖]——更不用说它们的谓词有可能非常复杂！——而这种不对称会造成一系列的实际问题。例如，在图11-17中，供应商和零件是不对称处理的。结果，查询（对称的）

（1）查出供应零件P1的供应商的号码S#

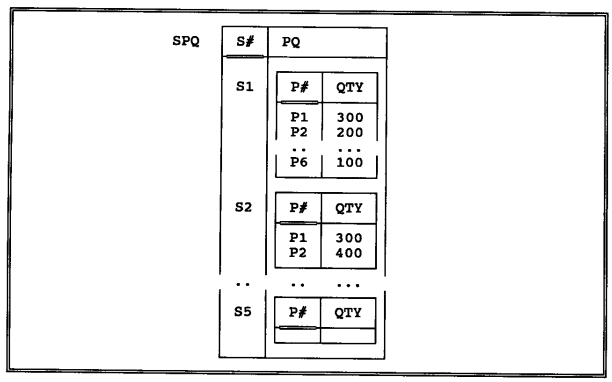

图11-17 一个具有关系值属性的关系

⊖ 由于历史原因，这种关系变量被认为是不合法的——即不是规范化的，它们甚至不属于1NF[10.4]，参见第5章。

China-pub.com
下载

(2) 查出由供应商S1提供的零件的号码P#

有不同的查询表达式：

```
1) (SPQ  WHERE  P#('P1') IN PQ{P#}) {S#}
2) ((SPQ  WHERE  S#=S#('S1')){PQ}{P#}
```

假定关系变量SPQ的值是由图11-17所示的关系的集合。

更糟糕的是更新，例如考虑下列更新操作：

(1) 插入一个发货记录，它的供应商号是S6，零件号是P5，数量是500。

(2) 插入一个发货记录，它的供应商号是S2，零件号是P5，数量是500。

对于通常的发货关系变量SP，这两个操作根本没什么区别——都是插入一条记录，但是，对于关系变量SPQ却相反，这两个操作有本质的区别（更不用说这两个操作比在SP上作相同的操作更复杂）：

```
1) INSERT  INTO  SPQ  RELATION
            {TUPLE{S#,S#,{'S6'},
                  PQ  RELATION {TUPLE{P# ('P5'),
                                      QTY  QTY(500)}}}};
2) UPDATE  SPQ  WHERE  S#=S#('S2')
       INSERT  INTO  PQ  RELATION {TUPLE{P#('P5'),
                                         QTY  QTY(500)}};
```

关系变量(至少是基本关系变量)一般不希望含有关系值属性，因为这种相对简单的逻辑结构使得对它们的操作也变得相对简单。然而应该清楚，这只是一个指导方针，而不是教条，在实际工作中，一个含有关系值属性的关系变量可能更能说明问题——即使是基本关系变量。图11-18是目录关系变量RVK的一个实例的一部分，该关系变量列出了数据库中所有的子关系变量以及它们的候选码，其中关系变量中CK的值也是关系，而且是该关系变量唯一的候选码的一个组成部分。RVK的定义如下：

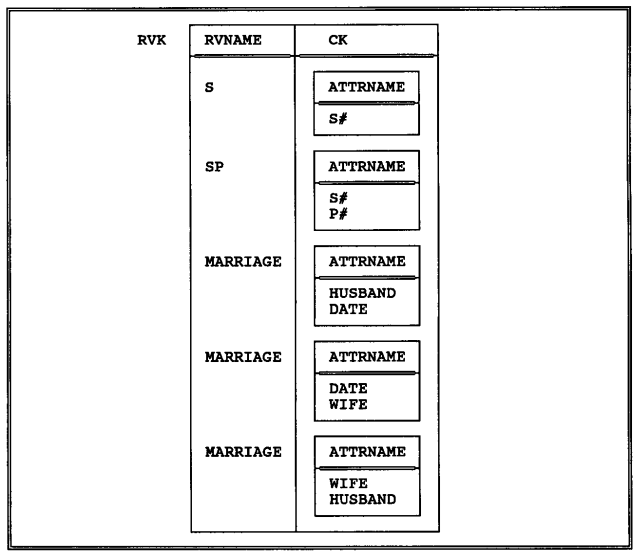

图11-18 目录关系变量RVK的实例

China-pub.com 下载

```
VAR  RVK  BASE  RELATION
       { RVNAME  NAME,ECK  RELATION {ATTRNAME,MNAME}}
         KEY  {RVNAME,ECK};
```

注意：本章练习 11.3**介绍了如何消除关系值属性的方法（可以参照第** 6.8**节的分组还原（**UNGROUP**）操作）——如果必须消除这种关系值属性的话（事实上常常是需要的）**[⊖]**。**

11.7 小结

到目前为止，已经介绍了规范化的前两章。在这两章中，主要讨论了第一、第二、第三和 Boyce/Codd**范式。因为属于任何一个范式的关系变量都自动属于比它低一级的范式，反之则不然——存在一些关系变量，它属于某一范式，但不属于比该范式更高级别的范式，所以这些不同的范式（包括下一章要讨论的第四范式和第五范式）就组成一条线。而且，任意一个关系变量都可以归约成** BCNF**（事实上可以归约成** 5NF**），几乎任何一个关系变量都可以用一个等价的**BCNF**（或**5NF**）关系变量的集合替代。这种归约的目的是避免冗余和消除某些更新异常。**

归约过程包括用一些投影替代给定的关系变量，这样通过这些投影的连接可以得到原来的关系变量，即这种归约过程是“可逆”的（或者说这种分解是无损的）。函数依赖在分解过程中同样具有重要作用， Heath**定理告诉人们如果满足给定的函数依赖，则这种分解是无损的。**

这两章里还讨论了 Rissanen**的“相互独立的投影”概念，并建议分解成相互独立的投影比分解成相互不独立的投影要“好”。当把一个关系变量分解成相互独立的投影时，该分解保持了函数依赖。很不幸，把一个关系变量无损分解成** BCNF**的集合和保持函数依赖这两个目的有时是相互冲突的，不能全部满足。**

最后给出3NF**和**BCNF**的精确定义（参见**Zaniolo[11.7]**），首先是**3NF**的定义：**

- **第三范式（**Zaniolo**的定义）：假设**R**是关系变量，**X**是**R**的属性集的子集，**A**是**R**的任意一个属性，当且仅当每一个函数依赖** $X \rightarrow A$ **至少满足下列条件中的一个时，该关系变量属于**3NF**：**

1）X**包含**A**（这样，该函数依赖是平凡的）。**

2）X**是个超码。**

3）A**属于**R**的某个候选码。**

只要把3NF**定义中的第三种可能去掉，即可得到** Boyce/Codd**范式的定义（这也说明** BCNF **比**3NF**严格）。顺便提及，本章引言所介绍的** 3NF**定义的缺陷产生的原因就是存在第三种可能性。**

练习

11.1 **证明**Heath**定理。该定理的逆定理成立吗？**

11.2 **所有的二目关系变量都是属于** BCNF**的。该命题正确吗？**

11.3 **图**11-19**是某公司人事部门数据库中要存放的信息，以层次结构（如** IBM**的层次数据库**

[⊖] **并且是可能的！应该注意到在关系变量中是不可能消除（最起码不能直接消除）这种子属性的，除非增加** CKNAME **(候选码属性)。**

China-pub.com
下载

系统IMS）的形式体现。图中：

- 该公司有若干部门。
- 每个部门有若干职工、项目和办公室。
- 每个职工都有工作经历（该职工所从事过的工作的集合）。对每一项工作，该职工都有一个工资历史记录（该职工从事该工作时所得的工资）。
- 每个办公室有若干个电话。

该数据库包含以下信息：

- 每个部门的信息包括：部门号（唯一）、预算费和部门领导人的职工号（唯一）。
- 每个职工的信息包括：职工号（唯一）、他当前参加的项目的项目号、办公室号码、电话号码、每个职工所从事的各个工作的的名称、加薪日及工资额。
- 描述每个项目的属性：项目号（唯一）和预算。
- 描述办公室的属性：办公室号码、楼层、办公室内的电话号码（唯一）。

根据这些信息，设计适当的关系变量，指出各个关系变量所应满足的函数依赖，并声明与这些函数依赖有关的假设。

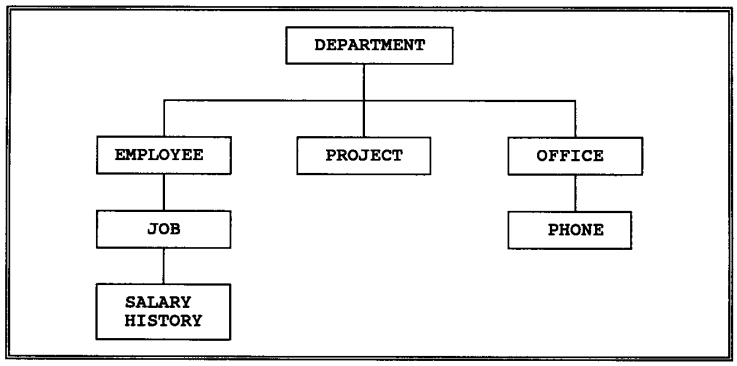

图11-19 一个公司的数据库（层次结构）

11.4 一个订货系统数据库中包括顾客、存货和订单等内容，下面是该数据库应包含的内容：

- 每个顾客的信息：
  - 顾客号（唯一）
  - 收货（ship-to）地址（每个顾客可以有多个）
  - 余额
  - 赊购限额
  - 折扣
- 描述每份订单的属性：
  - 订单头信息：
    - 顾客号
    - 收货地址
    - 定货日期
  - 订单细则（每一份订单可以有多项订单细则）：
  - 货物编号

China-pub.com 下载

订货数量

- 描述每种货物的属性：

  货物编号（唯一）

  制造厂商

  每个厂商实际存货量

  每个厂商规定的最低存货量

  货物的详细描述

由于处理的需要，每份订货单的每一项订货细则还应有一个未发货量，该值开始时是发货量，随着发货将减为0。为这些数据设计一个数据库。对于不同的情况给出数据依赖的假设。

11.5 假设上题中只有很少量的顾客（例如1%或更少）有多个发货地址，这是符合实际情况的。由于这些极少数的例外而又不能忽视的情形使我们不能按一般的方法来处理问题。请回顾上一题的答案，并改进它。

11.6 关系变量TIMETABLE习题10.13的翻版）有以下属性：

D	一星期中的一天（1~5）
P	一天中的一个时间段（1~8）
C	教室号码
T	教师姓名
S	学生姓名
L	课程名称

元组{*D*：*d*，*P*：*p*，*C*：*c*，*T*：*t*，*S*：*s*，*L*：*l*}指在时间段{*D*：*d*，*P*：*p*}时，学生*s*选修了教师*t*教的课程*l*，上课地点是*c*。可以假定，每节课持续一个时间段，并且一个星期中所有课程的名称都是唯一的，请把该关系变量归约成更好的结构。

11.7 关系变量NADDR（练习10.14的翻版）的属性是：NAME（姓名，唯一）、STREET（街道）、CITY（城市）、STATE（州）和ZIP（邮编）。对于任意一个邮编，只有一个城市和州与之对应；同样，对于任意给定的一个街道、城市或州，只有一个邮编和它对应，那么NADDR是否属于BCNF？是否属于3NF或2NF？能设计出更好的结构吗？

参考文献和简介

除了下面列出的参考书外，还可以参考 Codd关于1NF、2NF和3NF的论文[10.4~10.5]。

11.1 Philip A. Bernstein: “Synthesizing Third Normal Form Relations from Functional Dependencies,” *ACM TODS* 1, No.4 (1976年12)

本章讨论了把一个“大”关系变量分解成一些较“小”的关系变量的方法。在这篇文章中，Bernstein考虑了该问题的逆问题：用“较小”的关系变量合成“较大”的关系变量（也就是说具有更多属性）。但是这个问题并不是这么描述的，在该文中，他把该问题描述成根据给定的属性集和函数依赖构造一个属于3NF的关系变量集。然而，因为脱离关系变量这一具体的上下文，属性和函数依赖没有任何意义，所以，把含有函数依赖的二元关系变量作为基本结构，将会比把多个属性及其函数依赖作为基本结构更精确。

China-pub.com
下载

注意：可以把一组给定的属性集和函数依赖集看作是定义一个满足给定函数依赖集的通用关系变量（参见[12.9]）。在这种情况下，“合成过程”就可以看作是把该“通用关系变量”分解成3NF的关系变量集，但在目前讨论中仍然使用原来的关于“合成”的解释。

这样，合成过程就成了从一组二元关系变量和适用于这些关系变量的函数依赖集中合成*n*元关系变量的过程，而且，这些 *n*元关系变量应该是属于 3NF的（做这些工作时还没有BCNF的定义）。

该方法有一个缺陷(由Bernstein发现)：合成算法只能处理语法（syntactic）问题而不能处理语义(semantics)问题。例如：给定函数依赖：

A→B（适用于关系变量*R*{*A*，*B*}

B→C（适用于关系变量*S*{*B*，*C*}

A→C（适用于关系变量*T*{*A*，*C*}

第三个函数依赖有可能是冗余的（即可以由第一个和第二个函数依赖蕴涵），也可能不是冗余的，它将取决于*R*、*S*和*T*本身的意义。例如，假设*A*是雇员编号，*B*是办公室号码，*C*是部门编号，如果*R*表示雇员的办公室，*S*表示拥有该办公室的部门，*T*表示该部门的职工，考虑一个职工在不属于他所在的部门的办公室办公的情况，则可以看出第三个函数依赖不是由第一个和第二个函数依赖所蕴涵的。合成算法假定，这两个 C是一样的（事实上，它根本不识别关系变量的名称），这样就需要外部机制（如人的干预）去避免语义上的非法操作。在这种情况下，在最初定义函数依赖时就需要设计者给这两个属性定义不同的名称，如在*S*中定义为*C1*，而在*T*中定义为*C2*。

11.2 E. F. Codd: “Recent Investigations into Relational Data Base Systems,” Proc. IFIP Congress, Stockholm, Sweden (1974), and elsewhere.

该文涉及的主题比较多，但主要是给出了改进的第三范式的定义。该文中的第三范式实际上就是众所周知的BCNF。该文还讨论了视图及视图更新、数据子语言、数据交换和进一步研究的方向等问题。

11.3 C. J. Date: “A Normalization Problem,” in *Relational Database Writings 1991-1994*. Reading, Mass.: Addison-Wesley (1995).

为了证明这些抽象的理论，该文以航班管理数据库为例，分析了规范化问题，并利用它考察了数据库设计及显式的完整性约束条件的声明等问题。下面是该数据库中的函数依赖：

```
{ FLIGHT } → DESTINATION
{ FLIGHT } → HOUR
{ DAY, FLIGHT } → GATE
{ DAY, FLIGHT } → PILOT
{ DAY, HOUR, GATE } → DESTINATION
{ DAY, HOUR, GATE } → FLIGHT
{ DAY, HOUR, GATE } → PILOT
{ DAY, HOUR, PILOT } → DESTINATION
{ DAY, HOUR, PILOT } → FLIGHT
{ DAY, HOUR, PILOT } → GATE
```

这个例子还说明，只要以规范化理论为基础，就能设计出好的数据库。

China-pub.com 下载

11.4 I. J. Heath: “ Unacceptable File Operations in a Relational Database,” Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access, and Control, San Diego, Calif. (November 1971).

该文给出了3NF的定义，3NF实际上是最早的BCNF。该文还验证了11.2节提到的Heath定理。应该注意的是，本章所讨论的规范化的三个步骤实际上就是Heath定理的应用。

11.5 William Kent: “ A Simple Guide to Five Normal Forms in Relational Database Theory,” *CACM 26*, No. 2 (February 1983).

3NF（更精确地讲是BCNF）具有吸引力的两大原因：每个属性必须代表码（整个码）的一个事实，而且仅仅是代表码。

11.6 Jorma Rissanen: “ Independent Components of Relations,” *ACM TODS2*, No. 4 (December 1977).

11.7 Carlo Zaniolo: “ A New Normal Form for the Design of Relational Database Schemata,” *ACM TODS 7*, No. 3 (September 1982).

是11.7节提到的3NF和BCNF定义的基础。该文的主要目的是定义一个范式：基本码范式（elementary key normal form），即EKNF，该范式在克服了3NF和BCNF的不足（3NF的定义太“宽松”，而BCNF的计算又太复杂）的同时，保留了两者的优点。该文还说明，利用Bernsteirn[11.1]算法得到的关系变量属于EKNF，而不是属于3NF。

部分练习答案

11.1 根据Heath定理，如果关系变量$R\{A，B，C\}$满足函数依赖$A\rightarrow B$（A，B，C是属性集），那么R和投影$R1\{A，B\}$、$R2\{A，C\}$的连接等价。在定理的证明过程中，元组$\{A：a，B：b，C：c\}$简写为（a，b，c）

首先证明，把R分解成投影$R1$和$R2$，然后进行连接而没有元组丢失。假设（a，b，c）R，那么（a，b）$R1$，并且（a，c）$R2$，所以（a，b，c）$R1\ JOIN\ R2$。

接着证明，所有$R1$和$R2$连接的元组都是R的元组（即连接没有产生假的元组）。假设（a，b，c）$R1\ JOIN\ R2$，为了在连接过程中产生这样的元组，必须有元组（a，b）$R1$，（a，c）$R2$。这样必然存在一个元组（a，b'，c）R，为了得到元组（a，c）$R2$，必有（a，b'）$R1$。又因为（a，b）$R1$，而$A\rightarrow B$，所以$b'=b$，所以（a，b，c）R。

Heath定理的逆定理是，如果关系变量$R\{A，B，C\}$和它的两个投影$\{A，B\}$和$\{A，C\}$的连接等价，那么R必满足函数依赖：$A\rightarrow B$。该定理是错误的，例如在下一章图12-2所示的就是一个关系，它和它的两个投影的连接等价，但该关系变量不满足任何非平凡的函数依赖。

11.2 该论点并不十分正确。下面的反例取自文献[5.5]。考虑关系变量USA{COUNTRY，STATE}，可以把STATE看作COUNTRY的一个成员。该关系变量的每一个元组的COUNTRY都是美国，这样该关系变量满足函数依赖

```
{}→COUNTRY
```

决定因素{}并不是一个候选码，所以USA不属于BCNF。它可以无损分解成两个一目

China-pub.com
下载

的投影。当然，它是否有必要进一步地规范化是一个有待于讨论的课题。

值得注意的是，有时空集{}也会成为候选码（参考练习8.7）。

11.3 图11-20画出了一些重要的函数依赖，所有这些函数依赖都是根据习题的文字说明和语义假设导出（将在后面详细说明）。

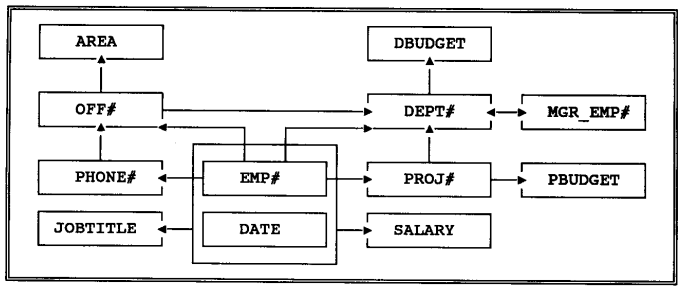

图11-20 习题11.3的函数依赖图

语义假设：

- 一个职工不能同时成为多个部门的领导人。
- 一个职工不能同时在多个部门就职。
- 一个职工不能同时参加多个项目。
- 一个职工不能同时属于两个不同的办公室。
- 一个职工不能同时拥有两部或两部以上的电话。
- 一个职工不能同时从事两种工作。
- 一个项目不能同时分配给多个部门。
- 一个办公室不能同时分配给多个部门。
- 部门号、职工号、项目号、办公室号码及电话号码是全局唯一的。

步骤0：建立初始关系变量结构

观察原来的层次结构，得到一个具有关系值属性的关系变量DEPT0：

```
DEPT0 { DEPT#, DBUDGET, MGR_EMP#, XEMP0, XPROJ0, XOFFICE0 }
      KEY { DEPT# }
      KEY { MGR_EMP# }
```

属性DEPT#、DBUDGET和MGR_EMP#容易理解，但是属性XEMP0、XPROJ0和XOFFICE0本身是个关系，需要做进一步的解释：

- DEPT0的每一个给定元组的属性XPROJ0是一个关系，含有两个属性：PROJ#（项目编号）和PBUDGET（项目预算费）。
- 同样，DEPT0的每一个给定元组的属性XOFFICE0是一个关系，含有三个属性：OFF#（办公室编号）、AREA（办公室面积）和XPHONE0。其中XPHONE0也是一个关系，它只有一个属性：PHONE#（电话号码）。
- 最后，DEPT0的每一个给定元组的属性XEMP0是一个五目关系：EMP#（职工号）、PROJ#（项目号）、OFF#（办公室编号）、PHONE#（电话号码）和XJOB0。其中XJOB0是一个二目关系，它的属性是：JOBTITLE（工作名称）和XSALHIST0，XSALHIST0是一个二目关系，它的属性是：DATE（日期）和SALARY（工资）。

这样，整个层次结构就可以用下面的嵌套结构表示：

China-pub.com 下载

```
DEPT0 { DEPT#, DBUDGET, MGR_EMP#,
        XEMP0 { EMP#, PROJ#, OFF#, PHONE#,
                XJOB0 { JOBTITLE,
                        XSALHIST0 { DATE, SALARY } } },
        XPROJ0 { PROJ#, PBUDGET },
        XOFFICE0 { OFF#, AREA, XPHONE0 { PHONE# } } }
```

注意：这里没有声明主码，而是把那些至少具有局部唯一性（unique whthin parent）的属性用斜体字表示。事实上，根据前面的假设，DEPT#、EMP#、PROJ#、OPP#和PHONE#都具有全局唯一性(globally unique)。

步骤1：消除关系值属性

为了使问题简单化，假设每一个关系变量都有一个主码——因为某些原因，人们常常指定一个候选码作为主码（原因并不重要）。在关系变量DEPT0中，假定{DEPT#}为主码（则{MGR_EMP#}为可选的主码（alternate key））。

根据第11.6节的分析可知，具有关系值属性的结构是不好的[⊖]，所以现在开始消除DEPT0中的关系值属性：

- 为DEPT0中的每一个关系值属性XEMP0、XPROJ0和XOFFICE0设计一个关系变量，它们的属性是各自本来的属性和DEPT0主码的组合，主码是原来“局部唯一”的属性和DEPT0的主码的组合（这种主码存在冗余，后面会消除这种冗余）。然后把XEMP0、XPROJ0和XOFFICE0从DEPT0中删除。
- 如果还有具有关系值属性关系变量*R*，则继续对*R*进行上述处理。

这样就得到了一组没有关系值属性的关系变量，它们都是属于1NF的，但不一定是属于其他范式的。

```
DEPT1 { DEPT#, DBUDGET, MGR_EMP# }
      PRIMARY KEY { DEPT# }
      ALTERNATE KEY { MGR_EMP# }

EMP1 { DEPT#, EMP#, PROJ#, OFF#, PHONE# }
     PRIMARY KEY { DEPT#, EMP# }

JOB1 { DEPT#, EMP#, JOBTITLE }
     PRIMARY KEY { DEPT#, EMP#, JOBTITLE }

SALHIST1 { DEPT#, EMP#, JOBTITLE, DATE, SALARY }
         PRIMARY KEY { DEPT#, EMP#, JOBTITLE, DATE }

PROJ1 { DEPT#, PROJ#, PBUDGET }
      PRIMARY KEY { DEPT#, PROJ# }

OFFICE1 { DEPT#, OFF#, AREA }
        PRIMARY KEY { DEPT#, OFF# }

PHONE1 { DEPT#, OFF#, PHONE# }
       PRIMARY KEY { DEPT#, OFF#, PHONE# }
```

步骤2：归约成2NF

下面通过消除部分函数依赖，把步骤1得到的关系变量归约成与之等价的2NF关系变量的集合。

DEPT1：已经是2NF。

EMP1：首先可以看出主码中的DEPT#是冗余的，可以把{EMP#}作为该关系变量的

[⊖] 这种消除关系值属性的过程实际上就是反复运用第6章6.8节介绍的取消分组(UNGROUP)操作，直到得到所期望的结果。而且该过程在消除关系值属性的同时也保证消除任何不是函数依赖（一对一的单值函数依赖）的多值函数依赖(MVDS)，结果所得的关系变量属于4NF而不是属于BCNF(见第12章)的。

China-pub.com
下载

主码，这样，该关系变量就是2NF的。

JOB1：同样，DEPT#也可以不包括在主码中。因为DEPT#函数依赖于EMP#，所以DEPT#部分依赖于主码{EMP#，JOBTITLE}，因此该关系变量不属于2NF。可以用下列关系变量替代：

```
JOB2A { EMP#, JOBTITLE }
      PRIMARY KEY { EMP#, JOBTITLE }
```

和

```
JOB2B { EMP#, DEPT# }
      PRIMARY KEY { EMP# }
```

然而，JOB2A是SALHIST2（见下文）的一个投影，JOB2B是EMP1（在后面又称为EMP2）的投影，所以这两个关系变量都可以消除。

SALHIST1：和JOB1一样，可以把DEPT#完全删除，而且，JOBTITLE也可以不作为主码属性，而把{EMP#，DATE}作为主码，从而得到2NF的关系变量：

```
SALHIST2 { EMP#, DATE, JOBTITLE, SALARY }
         PRIMARY KEY { EMP#, DATE }
```

PROJ1：和EMP1一样，可以把DEPT#看作非码属性，这样该关系变量就是2NF的。

OFFICE1：和PROJ1一样。

PHONE1：因为，关系变量{DEPT#，OFF#}是OFFICE1（后面称为OFFICE2）的投影，所以，可以把属性DEPT#去掉。OFF#又函数依赖于PHONE#，所以可以把PHONE#作为主码，这样可以得到一个2NF的关系变量：

```
PHONE2 { PHONE#, OFF# }
       PRIMARY KEY { PHONE# }
```

应该注意的是，因为可以存在没有分配给某个职工的的办公室和电话，该关系变量不一定是EMP2的投影，所以该关系变量不能省去。

这样就得到一个2NF的关系变量集：

```
DEPT2 { DEPT#, DBUDGET, MGR_EMP# }
      PRIMARY KEY { DEPT# }
      ALTERNATE KEY { MGR_EMP# }

EMP2 { EMP#, DEPT#, PROJ#, OFF#, PHONE# }
     PRIMARY KEY { EMP# }

SALHIST2 { EMP#, DATE, JOBTITLE, SALARY }
         PRIMARY KEY { EMP#, DATE }

PROJ2 { PROJ#, DEPT#, PBUDGET }
      PRIMARY KEY { PROJ# }

OFFICE2 { OFF#, DEPT#, AREA }
        PRIMARY KEY { OFF# }

PHONE2 { PHONE#, OFF# }
       PRIMARY KEY { PHONE# }
```

步骤3：归约成3NF

现在通过消除关系变量中的传递函数依赖，把2NF的关系变量归约成3NF的关系变量。观察上面的属于2NF的关系变量，发现唯一一个不属于3NF的关系变量是EMP2。在这个关系变量中，OFF#和DEPT#都传递函数依赖于主码{EMP#} —— OFF#通过PHONE#，DEPT#通过PROJ#或OFF#（然后通过PHONE#）传递依赖于{EMP#}，所以可以把EMP2

China-pub.com 下载

分解成下面的3NF关系变量集：

```
EMP3 { EMP#, PROJ#, PHONE# }
     PRIMARY KEY { EMP# }

X { PHONE#, OFF# }
  PRIMARY KEY { PHONE# }

Y { PROJ#, DEPT# }
  PRIMARY KEY { PROJ# }

Z { OFF#, DEPT# }
  PRIMARY KEY { OFF# }
```

然而，X就是PHONE2，Y是PROJ2的投影，Z是OFFICE2的投影，所以最后的3NF关系变量集就变得更简单：

```
DEPT3 { DEPT#, DBUDGET, MGR_EMP# }
      PRIMARY KEY { DEPT# }
      ALTERNATE KEY { MGR_EMP# }

EMP3 { EMP#, PROJ#, PHONE# }
     PRIMARY KEY { EMP# }

SALHIST3 { EMP#, DATE, JOBTITLE, SALARY }
         PRIMARY KEY { EMP#, DATE }

PROJ3 { PROJ#, DEPT#, PBUDGET }
      PRIMARY KEY { PROJ# }

OFFICE3 { OFF#, DEPT#, AREA }
        PRIMARY KEY { OFF# }

PHONE3 { PHONE#, OFF# }
       PRIMARY KEY { PHONE# }
```

最后，可以看出，所有这些3NF的关系变量都是属于BCNF的。

应该注意的是，因为PROJ3的投影{PROJ#，DEPT#}和EMP3、PHONE3及OFFICE3的投影是完全一样的，所以，如果增加一些合理的语义限制条件，这个BCNF关系变量集是强冗余的[5-1]。

最后发现，可以从函数依赖图中“得到”BCNF关系变量。应该注意的是，并不是说任何一个关系变量都可以分解成BCNF关系变量集——在实际工作中，有可能出现不能分解成BCNF关系变量集的情况，下面将给出更精确的说明。假设给定关系变量 R满足函数依赖集S，下面的算法（步骤 0~8）能保证将R分解成3NF（不是BCNF）关系变量集D，在分解过程中是“无损”和保持函数依赖的：

0. 初始化D，D={}。
1. 假设I是S的最小覆盖。
2. 假设X是出现在I中的函数依赖$X\rightarrow Y$的左边的属性集。
3. 假设函数依赖集I中所有左边是X的函数依赖是$X\rightarrow Y1$，$X\rightarrow Y2$，…，$X\rightarrow Yn$。
4. 假设$Y1$，$Y2$，…，Yn的并集是Z，$Z= Y1\cup Y2\cup Yn$。
5. 用D和R在X、Z上的投影的并集取代原来的D。
6. 对每个不同的X重复步骤3~5。
7. 假设属性集$A1$，$A2$，…，An是R的属性集，但不是D的任何一个关系变量的属性，用D和R在$A1$，$A2$，…，An投影的并集取代原来的D。
8. 如果R中的一个候选码没有包含在D的任何一个关系变量中，则用D和R在这个候选码上的投影的并集取代原来的D。

下面的算法（步骤0~3）能将关系变量R分解成BCNF关系变量集D。分解过程中能保

China-pub.com
下载

证是无损的，但不能保证保持函数依赖：

0. 初始化D，使D只含R。
1. 对D中每一个不属于BCNF的关系变量T执行步骤2~3。
2. 假设T的函数依赖$X \rightarrow Y$违反了BCNF的要求。
3. 用两个投影，即在X、Y上的投影和在除Y之外的所有属性上的投影代替原来的T。

回到公司人事数据库这个话题上来：作为附加练习，继续前面的设计，把前面的设计和外码的声明结合起来——这和规范化的关系不大，但和一般的数据库设计有关。

11.4 图11-21是练习的函数依赖图，其语义假设如下：

- 任何两个顾客的收货地址都不相同。
- 每一个订单都有一个唯一的订单号码。
- 每个订单的订单细则在这个订单里有一个唯一的编号。

相应的BCNF关系变量集如下：

```
CUST { CUST#, BAL, CREDLIM, DISCOUNT }
     KEY { CUST# }

SHIPTO { ADDRESS, CUST# }
       KEY { ADDRESS }

ORDHEAD { ORD#, ADDRESS, DATE }
        KEY { ORD# }

ORDLINE { ORD#, LINE#, ITEM#, QTYORD, QTYOUT }
        KEY { ORD#, LINE# }

ITEM { ITEM#, DESCN }
     KEY { ITEM# }

IP { ITEM#, PLANT#, QTYOH, DANGER }
   KEY { ITEM#, PLANT# }
```

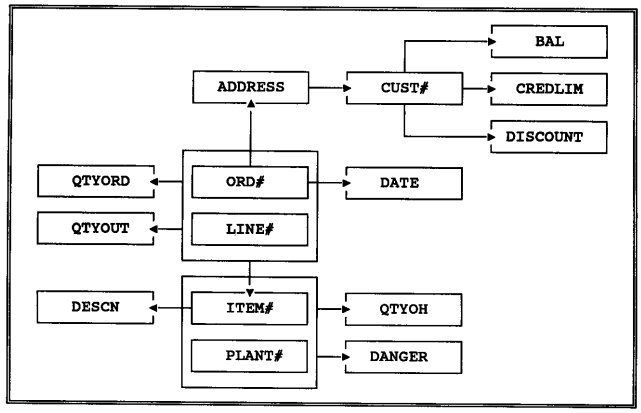

图11-21 练习11.4的函数依赖图

11.5 考虑到该过程要用订单处理程序处理，假定在输入订单时要声明顾客号、收货地址和订单细节（货物号和数量）

```
RETRIEVE CUST WHERE CUST# = input CUST# ;
check balance, credit limit, etc. ;
RETRIEVE SHIPTO WHERE ADDRESS = input ADDRESS
                  AND   CUST#   = input CUST#
/* this checks the ship-to address */ ;
IF everything is OK THEN process the order ; END IF ;
```

China-pub.com 下载

如果99%的顾客只有一个收货地址，则把地址放在与 CUST不同的一个关系变量中的效率是很低的（如果只考虑 99%的只有一个收货地址的顾客，则 ADDRESS（地址）函数依赖于CUST#（顾客号））。下面对这个问题进行改进。对于每个顾客，指定一个合法收货地址作为主地址，则对于 99%的顾客，该地址就是他的唯一地址，其它地址存放在关系变量SECOND中，关系变量CUST的定义如下：

```
CUST { CUST#, ADDRESS, BAL, CREDLIM, DISCOUNT }
     KEY { CUST# }
```

关系变量SHIPTO可以用下面的关系变量替代：

```
SECOND { ADDRESS, CUST# }
       KEY { ADDRESS }
```

这里CUST存放主地址，而SECOND中存放所有的第二地址（和相应的顾客号），这两个关系变量都是属于BCNF的。现在这个订单处理程序如下所示：

```
RETRIEVE CUST WHERE CUST# = input CUST# ;
check balance, credit limit, etc. ;
IF retrieved ADDRESS ≠ input ADDRESS THEN
   RETRIEVE SECOND WHERE ADDRESS = input ADDRESS
                   AND   CUST#   = input CUST#
   /* this checks the ship-to address */ ;
END IF :
IF everything is OK THEN process the order ; END IF ;
```

该方法具有如下优点：

- 对于99%的顾客的处理变得简单（当然更有效）了。
- 如果输入订单时把收货地址省略了，则可以用主地址作为默认地址。
- 如果一个顾客的不同收货地址的折扣不同，那么如果采用原来的方法，则属性DISCOUNT必须被移到关系变量SHIPTO中，使处理变得很复杂。而在修正过的方法中，主要折扣（和主地址相对应）可以采用 CUST中的DISCOUNT，而其他折扣可以存放在SECOND的DISCOUNT中，这两个关系变量都是 BCNF的，而且对于99%的顾客的处理还是很简单的。

总的说来，把特殊情况分离开来是个有效的方法，它可以充分利用两者的优点，既达到简化处理的目的，又使设计的关系变量不违反 BCNF的限制。

11.6 图11-22显示了一些比较重要的函数依赖，它的可能的关系变量集如下：

```
SCHED { L, T, C, D, P }
      KEY { L }
      KEY { T, D, P }
      KEY { C, D, P }

STUDY { S, L }
      KEY { S, L }
```

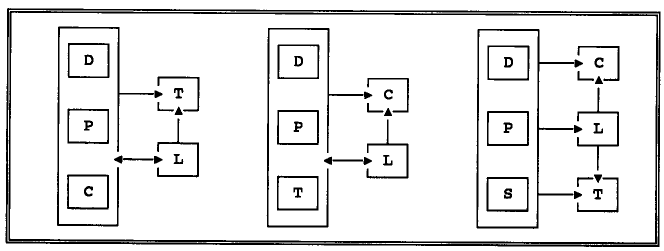

图11-22 练习11.6的函数依赖图

China-pub.com
下载

11.7 NADDR属于2NF，但不属于3NF。一个更好的设计是：

```
NSZ { NAME, STREET, ZIP }
    KEY { NAME }

ZCS { ZIP, CITY, STATE }
    KEY { ZIP }
```

这两个关系变量都是属于BCNF的。

- 因为STREET（街道）、CITY（城市）和STATE（州）总是连在一起的，而ZIPCODE（邮编）不是经常变动的，所以这种分解可以认为是不必要的（即只对相关的依赖实施规范化，而不必对所有的依赖都进行规范化）。
- 考虑到如果要得到某一个人的完整地址就需要进行连接操作，因此可以得到这样一个结论：规范化到BCNF范式对更新有利，但不利于查询——没有完全规范化造成的冗余给更新带来了困难但对查询有利⊖。不加控制的冗余会带来更新异常，但在某些情况下，一些受控的冗余还是可以接受的（如由DBMS声明的冗余，它将由DBMS管理）。
- 函数依赖{STREET，CITY，STATE}→ZIP并没有在该设计中没有得到体现，事实上必须通过声明用户自定义限制条件或定义过程来单独维护它。其实，按照Rissanen的观点，关系变量NSZ和ZCS不是相互独立的[11.6]。

⊖ 另一方面，这种冗余也会给某些查询带来不方便，就像在下一章的12.5节看到的一样，它将使某些查询更难以明确表达。