第三部分　数据库设计

这一部分主要介绍数据库的设计（更确切地说是关系数据库的设计），数据库设计问题可以简单地描述为：如果要把一组数据储存在数据库中，该如何为这些数据设计一个合适的逻辑结构？——用另一句话来说就是，如何决定存在哪些关系变量，以及各个关系变量中应该有哪些属性？这个问题的重要性是显而易见的。

在详细地讨论这个问题前，首先应该注意以下几个问题：

(1) 应该注意，在这里只讨论逻辑（或者是概念）设计，而不是物理设计。当然，这并不是说物理设计不重要，相反，物理设计也是十分重要的，然而：

1) 物理设计可以看作是逻辑设计后的一个与逻辑设计相独立的工作。也就是说，数据库设计的“正确”方法是首先做一个纯逻辑的（例如：关系）设计，然后，作为一个单独的后续步骤，把逻辑设计映射到特定的 DBMS支持的物理结构上（用第 2章的话来说就是物理设计应该在逻辑设计的基础上产生，而不是用其他方法产生）。

2) 根据定义，物理设计从某种角度来说是依赖于特定的 DBMS的，在本书这种通用教材中不宜把它作为一个主题来讨论。而逻辑设计正相反，它是独立于 DBMS的，并且有成型的理论可以应用。当然，这些理论也将在本书中讨论。

遗憾的是，现实世界并不完美，在实际工作中，实际的物理阶段的设计往往会对逻辑设计产生影响（在本书中已经多次提到，现在的 DBMS只支持从逻辑结构到物理结构的简单映射），用另一句话说，数据库设计是逻辑设计——物理设计——逻辑设计这样一个反复进行的过程，有时必须反复多次，在此过程中有时必须作出妥协。然而，我们还是支持原先的观点：正确的数据库设计方法是首先进行数据库的逻辑设计而不考虑数据库的物理设计，因此，该书的这一部分主要讨论“首先如何使数据库的逻辑设计正确”。

(2) 虽然所讨论的主要是关系数据库的设计，但是即将讨论的这些思想和非关系数据库也是密切相关的。也就是说，在非关系数据库的设计中，正确的方法是首先作一个正确的关系设计，然后作为一个单独的步骤，把关系设计映射到任何一个 DBMS支持的非关系结构（如层次结构）上。

(3) 数据库设计应该说是一种艺术而不是一种科学。尽管有一些科学理论可以应用到这个问题上，而且这些科学理论也是后面三章讨论的对象；然而，有很多很多的设计问题在这些科学规则中根本没有提到。因此，很多数据库理论家和从业者提出了数据库设计方法学（design methodology）[⊖]，从某种程度来说，它们都可以用来处理到目前来说还比较难处理的问题，即找出一个合理的逻辑设计，但是这些方法有些相当严格，另一些则不然。因为这些方法在一定程度上只适用于某些特定场合，所以没有客观的标准来判断选择哪种方法较好。然而，在第 13章中介绍一种众所周知的方法，这种方法因为它的许多优点而被广泛使用。在

⊖ 术语方法学原指对方法的研究，但是这里指的是某种科学或艺术的学习和研究的方法与规则。

该章中还简单介绍了一些商业上支持的方法。

(4) 需要说明的是，以下两个假设是这一部分讨论的基础：

1) 数据库设计不仅仅是得到一个正确的数据结构，数据完整性是数据库设计的关键要素之一。这一点将在以后的各个章节中不断地重复和强化。

2) 主要讨论具有应用独立性的设计。也就是说，主要讨论数据本身的问题，而不是数据怎样被使用的问题。应用独立性之所以重要，是因为在设计阶段不可能知道对数据的所有使用方式。人们总是希望自己的设计是鲁棒的，即不希望设计的数据库在遇到一个在设计阶段没有考虑到的应用需求时把它认为是非法数据。用另一句话来说（采用第2章的术语），人们所要做的主要是使“概念”模式正确，即设计一个独立于硬件、操作系统、DBMS、语言及用户的抽象的逻辑结构，而对于前面所说的因为实现问题而作出的妥协则不感兴趣。

(5) 前面已经说过，数据库设计主要是确定在数据库中应有哪些关系变量，以及每个关系变量中应该有哪些属性。事实上，还要确定应该定义哪些域或类型，但是本书没有对这个主题作过多的介绍，因为到目前为止有关这个问题所做的研究还不多（[13.11]和[13.39]除外）。

这一部分的结构安排如下：第10章提出一些基础理论，第11、12两章涉及规范化思想，该思想直接建立在前面介绍的理论基础上，目的是给非形式化的声明一个形式化的意义，以便说明某种设计在某些方面比另一种设计“好”。第13章介绍语义建模(semantic modeling)，特别介绍了“实体/关系”(entry/relationship，即ER）模型的概念，并介绍这个概念如何运用于自上而下的数据库设计问题（从现实世界的实体开始，以规范的关系设计结束）。

第10章 函数依赖

10.1 引言

本章，首先介绍一下基本概念，即函数依赖。[10.7]中把这一概念描绘为“不是很基础，但十分近似基础的”概念。这个概念对以后各章要讨论的几个问题（特别是第11章要讨论的数据库设计理论）都非常重要。

函数依赖主要是指一个关系变量中一个属性集和另一个属性集间的多对一关系。例如：在发货关系变量SP中存在由属性集{S#，P#}到属性集{QTY}间的函数依赖，它的意思是，对于关系变量SP的任意一个合法的值（关系）：

1) 对于任意给定的属性集S#和P#的值，只有一个QTY的值与之对应；但是

2) 可以存在许多S#和P#的不同的值，而它们所对应的QTY的值相同。

注意：我们通常所举的SP的例子（见例3.8）确实满足以上两个条件。

在10.2节中将进一步介绍函数依赖的概念，严格区分那些只在某些特定的条件下才能满足给定的关系变量的函数依赖和在任何条件下都能满足给定的关系变量的函数依赖。已经提到过，函数依赖是科学解决许多实际问题的基础，其原因是函数依赖具有一些有趣的形式化的特性，这使得可以用一种形式化的、严格的方法处理所讨论的问题。10.3~10.6节详细介绍了这些形式化特性及它们在实际工作中的重要性。10.7节给出简单总结。

注意：为了理解后面三章内容所应了解的概念在10.2节和10.3节做了介绍，所以第一次阅读本书时，对其余部分可以只简单地看看，等消化吸收了后面三章的内容后再回过头来仔细阅读这些内容。

10.2 基本概念

为了解释本节的一些概念，把发货关系变量稍作修改，使它除了含有原来的属性：S#（供应商编号）、P#（零件编号）和QTY(发货量)外，增加一个属性CITY（城市），该属性表示供应商的地址，为了防止混淆，把这个关系变量称为SCP。关系变量SCP的一个可能的值见图10-1。

SCP

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

图10-1 关系变量SCP的一个实例

现在，有必要分清楚以下两种不同的情况：(a)给定的关系变量在某一特定时间的值；(b)给定关系变量在不同时候所有可能的值。首先根据情况(a)讨论函数依赖，然后，把函数依赖的概念扩展到情况(b)。下面是情况(a)的定义：

- 假设r是一个关系，X和Y是r的属性集的任意子集，当且仅当r中任一给定的X的值，在r中存在一个唯一的Y与之对应。也就是说，如果X相等，Y也相等，则Y函数依赖于X，表示为$X \rightarrow Y$（读作X函数决定Y，或简单读作X指向Y）。

例如：图10-1所示的关系满足下述函数依赖：

```
{S#}→{CITY}
```

因为这个关系的任一给定的S#值都有一个给定的CITY与之对应。事实上，该关系还满足下列函数依赖：

```
{S#, P#}→{QTY}
{S#, P#}→{CITY}
{S#, P#}→{CITY, QTY}
{S#, P#}→{S#}
{S#, P#}→{S#, P#, CITY, QTY}
{S#}→{QTY}
{QTY}→{S#}
```

(练习：检查这些函数依赖的正确性。)

函数依赖表达式的左边和右边有时分别称为自变量（determinant)和应变量(depentent)。根据定义，自变量和应变量都是属性集。当某个属性集只含有一个属性时，即单元素集（singleton set）时，可以把花括号对省去，例如：

```
S#→CITY
```

前面已经解释过，上面的定义只适用情况(a)，即只适用于单独的一个关系。然而当考虑

关系变量，特别是基本关系变量时，人们所感兴趣的就不是对关系变量的某个或某些关系适用的函数依赖，而是对关系变量的所有可能的值都适用的函数依赖。例如在 SCP中，函数依赖

```
S#→CITY
```

对SCP的所有可能值都适用，因为在任何情况下，一个给定的供应商有一个确定的地址（CITY），所以在SCP中的任意两个元组，如果它的供应商编号（ S#）相等，则它们的地址（CITY）相等。事实上，“ 任何时间 ”都适用的函数依赖（例如对所有 SCP可能的值）是关系变量SCP的完整性约束条件 —— 它是对SCP所有被认为合法的值的一个限制。下面用第 8章的语法定义这种约束的表述：

```
CONSTRAINT S#_CITY_FD
COUNT(SCP{S#})=COUNT(SCP{S#,CITY})；
```

语法S#→CITY可以看作是上述表述的简写。

下面是在情况（b）下函数依赖的定义（对情况 a的扩展用黑体字表示）：

- 设*R*是关系变量，*X*、*Y*是*R*的属性集的任意子集，当且仅当对于*R*的所有可能的合法值，*X*的值和*Y*的值密切相关。也就是说，对于*R*的所有可能的合法值，当两个元组的*X*值相等时，*Y*值也相等，则*Y*函数依赖于*X*，表示为：

```
X→Y
```

今后说“ 函数依赖 ”指的是要求更加严格的、具有时间独立性的函数依赖，而不必详细说明函数依赖成立的条件。

下面是关系变量SCP的一些函数依赖（具有时间独立性的函数依赖）：

```
{S#，P#}→QTY
{S#，P#}→CITY
{S#，P#}→{CITY，QTY}
{S#，P#}→S#
{S#，P#}→{S#，P#，CITY，QTY}
{S#}    →CITY
```

特别要注意下列函数依赖，它们在图 10-1的情况下成立，但并不是任何时间对关系变量SCP都成立。

```
S#→QTY
QTY→S#
```

换句话说，在图10-1中，命题“ 给定供应商的发货量是相等的 ”是真的，但并不是对关系变量SCP的所有可能的合法值都是真的。

如果*X*是关系变量*R*的候选码，则关系变量 *R*的任意属性*Y*一定函数依赖于*X*（这一点在8.8节提到过，它可以从候选码的定义中得出）。例如，在关系变量*P*中，应该有：

```
P#→{P#，PNAME，COLOR，WEIGHT，CITY}
```

事实上，如果关系变量*R*满足函数依赖*A*→*B*，而*A*不是候选码 ㊀，则*R*一定存在冗余。例如在关系变量SCP中，表示给定的供应商（ S#）一般居住在给定的城市（ CITY）的函数依赖S#→CITY会出现多次（参见图10-1）。下一章将对这个问题作详细的介绍。

㊀ 并且该函数依赖不是平凡的函数依赖 [见10.3节]，*A*也不是超码 [见10.5节]，而且*R*至少有两个元组。

即使只考虑任何时间都满足的函数依赖，一个给定的关系变量的完整函数依赖集还是很庞大的，如关系变量SCP所示（练习：给出SCP的完整的函数依赖集）。应该寻找一个方法把这个集合缩小到一个可管理的范围——事实上，本章的剩余部分主要讨论这个问题。

为什么这个问题值得讨论呢？原因之一是函数依赖表示完整性约束条件，而DBMS需要实现这种完整性限制条件。给定一个函数依赖集 S，如果能找到一个集合 T，T远远小于S，而集合T的函数依赖蕴涵集合 S的所有函数依赖，则DBMS只要实现函数依赖集 T，函数依赖集S中的所有函数依赖会自动实现，因此，寻找函数依赖集 T具有实践上的重要性。

10.3 平凡的函数依赖和非平凡的函数依赖

注意：在本章的剩余部分常常把“函数依赖”简称为“依赖”，同样对“函数依赖于”和“函数决定”等作相应的简化。

缩小函数依赖集大小的一个简单方法是消除平凡的函数依赖，一个不可能不满足的函数依赖称为平凡的函数依赖。在前面提到的关系变量 SCP的一个函数依赖就是平凡的函数依赖，即函数依赖：

```
{S#, P#}→S#
```

事实上，当且仅当函数依赖的右边是左边的子集（不一定是真子集）时，该函数依赖才是平凡的函数依赖。

正如名称所示，平凡的函数依赖并没有实际意义，实际上人们所感兴趣的是非平凡的函数依赖，因为只有它们才和“真正的”完整性约束条件相关。然而，在讨论规范化时，必须处理所有的依赖：平凡的函数依赖和非平凡的函数依赖。

10.4 依赖集的闭包

前面已经提到过，有些函数依赖蕴涵另一些函数依赖，作为一个简单的例子，函数依赖：

```
{S#, P#}→{CITY, QTY}
```

蕴涵下面两个函数依赖：

```
{S#, P#}→CITY
{S#, P#}→QTY
```

作为一个比较复杂的例子，假设一个关系变量 R有三个属性A、B和C，如果R满足函数依赖$A\rightarrow B$和$B\rightarrow C$，很容易就可以发现R同样满足函数依赖$A\rightarrow C$。这里函数依赖$A\rightarrow C$是传递函数依赖的一个例子——即C通过B传递依赖于A。

函数依赖集S所蕴涵的函数依赖的全体称为函数依赖集 S的闭包，记为S^+，很显然，人们需要一个算法用以从一个给定集合 S中求出它的闭包 S^+，对这个问题的阐述最早出现在Armstrong的论文里，他提出了一组推理规则（常被叫作Armstrong公理），通过这些推理规则，可以从给定的函数依赖中推出新的函数依赖。这些规则可以用许多等价的方法阐述，下面是其中最简单的一种：假设 A、B和C是给定的关系变量 R的属性集的任意子集，并把 A和B的并集记位AB，则：

1) 自反律（reflexivity）：如果B是A的子集，则$A\rightarrow B$。
2) 增广律（augmentation）：如果$A\rightarrow B$，则$AC\rightarrow BC$。

3) 传递律（transitivity）：如果$A \rightarrow B$且$B \rightarrow C$，则$A \rightarrow C$。

以上的每一个规则都可以从函数依赖的定义直接证明（当然，第一条规则正好是平凡的函数依赖的定义）。并且，该公理是完备的，即给定一个函数依赖集S，该函数依赖集所有蕴涵的函数依赖都可以利用这些规则从S中导出，另外，该公理是有效的，即所有不是由函数依赖集S蕴涵的函数依赖不能根据该公理系统从S导出。也就是说，可以利用该公理系统推导S的闭包S^+。

从上面的规则可以导出其它规则（如下所示），这些规则在实际工作中可以用来简化从S中计算S^+的工作（在下面列出的规则中，D是关系变量R的属性集的另外一个子集）。

4) 自含规则（self_determination）：$A \rightarrow A$。

5) 分解规则（decomposition）：如果$A \rightarrow BC$，则$A \rightarrow B$，且$A \rightarrow C$。

6) 合并规则（union）：如果$A \rightarrow B$且$A \rightarrow C$，则$A \rightarrow BC$。

7) 复合规则（composition）：如果$A \rightarrow B$，$C \rightarrow D$，则$AC \rightarrow BD$。

Darwen证明了下面的定理[10.6]，该规则被称为“通用一致性定理”（General Unification Theorem）：

8) 如果$A \rightarrow B$且$C \rightarrow D$，则$A \cup (C - B) \rightarrow BD$（“∪”表示并集，“-”表示差集）。

该定理之所以被称为“通用一致性定理”，是因为很多早期的规则可以被看作该定理的特例。

例：假设有一个关系变量R，A、B、C、D、E、F是它的属性，R满足下列函数依赖：

$A \rightarrow BC$

$B \rightarrow E$

$CD \rightarrow EF$

可以看出，这里对符号稍微作了相容的扩充，例如，用BC表示B和C中的所有属性（精确地说，BC表示B和C的并集）。注意：如果想用更具体的例子，可以假定A是雇员号，B是部门号，C是经理的雇员号，D是一个经理负责的工程的工程号（对于某个经理，工程号是唯一的），E是部门名称，F表示某个经理分配某个工程的时间。

从下面可以看出，在R中函数依赖$AD \rightarrow F$是成立的，且是给定函数依赖集的闭包的一个成员：

1) $A \rightarrow BC$ （给定）

2) $A \rightarrow C$ （1，分解规则）

3) $AD \rightarrow CD$ （2，增广律）

4) $CD \rightarrow EF$ （给定）

5) $AD \rightarrow EF$ （3和4，传递律）

6) $AD \rightarrow F$ （5，分解规则）

10.5 属性集的闭包

原则上讲，通过算法：反复运用前面讲的规则，直到不再产生新的函数依赖为止，就可以计算出一个函数依赖集S的闭包S^+。但是，因为这种算法的效率是很低的，所以在实际工作中，几乎没有必要计算闭包本身。在这一部分将介绍如何计算一个闭包的子集：即该子集包含所有左边是特定的属性集Z的函数依赖。更精确地说，给定一个关系变量R，R的一个属性集Z，以及R的一个函数依赖集S，从中确定R中所有的函数依赖于Z的属性集——即属性集Z关

于函数依赖集S的闭包Z^{+}⊖。图10-2是计算该闭包的算法（练习：证明该算法的正确性）。

```
CLOSURE[Z,S] := Z ;
do "forever" ;
   for each FD X → Y in S
      do ;
         if X ≤ CLOSURE[Z,S]              /* ≤ = "subset of" */
         then CLOSURE[Z,S] := CLOSURE[Z,S] ∪ Y ;
      end ;
   if CLOSURE[Z,S] did not change on this iteration
   then leave loop ;                  /* computation complete */
end ;
```

图10-2 计算属性集Z关于S的闭包$Z+$

例：假设给定一个关系变量R，A、B、C、D、E、F是它的属性集，以及函数依赖集S：

$A \rightarrow BC$

$E \rightarrow CF$

$B \rightarrow E$

$CD \rightarrow EF$

下面开始计算属性集$\{A, B\}$关于函数依赖集S的闭包$\{A, B\}^{+}$

1) 初始化：令集合CLOSURE[Z，S]=$\{A, B\}$。
2) 进行四次内循环，一次一个函数依赖。第一次循环（对于函数依赖 $A \rightarrow BC$）时发现它的左边是到目前为止计算的集合CLOSURE[Z，S]的子集，所以把属性（B和）C加入集合CLOSURE[Z，S]，这样，集合CLOSURE [Z，S]就变为$\{A, B, C\}$。
3) 第二次循环（对于函数依赖$E \rightarrow CF$）发现它的左边不是到目前为止计算的结果的子集，因此该结果保持不变。
4) 第三次循环（对于函数依赖 $B \rightarrow E$），把E加入集合CLOSURE[Z，S]，这样，CLOSURE[Z，S]=$\{A, B, C, E\}$。
5) 第四次循环（对于函数依赖$CD \rightarrow EF$），集合保持不变。
6) 再一次经过四次内循环，第一次循环结果不变，第二次结果扩展到$\{A, B, C, E, F\}$，第三次和第四次不变。
7) 再一次经过四次内循环，集合CLOSURE[Z，S]保持不变，这样，整个过程结束，结果：$\{A, B\}^{+}=\{A, B, C, E, F\}$。

根据前面的讨论可以得出这样一个重要结论：给定一个函数依赖集 S，可以方便地判断函数依赖$X \rightarrow Y$是否可以从S中导出，因为当且仅当Y是属性集X关于S的闭包的子集时，$X \rightarrow Y$才能根据Armstrong公理由S中导出。也就是说，可以用一种简单的方法，该方法不必精确计算函数依赖集S的闭包S^{+}就能判断函数依赖$X \rightarrow Y$是否属于函数依赖集S的闭包S^{+}。

还可以得出另外一个重要的结论。在第8章中讲过，关系变量R的超码是关系变量R的属性集，该关系变量的一些侯选码是该属性集的子集（不必是真子集）。由此可以得出以下结论：给定关系变量R的超码一定是R的属性集的一个子集K，对于R中的任意属性集的子集A有函数依赖

$K \rightarrow A$

⊖ 左边为Z的函数依赖集包含所有具有$Z \rightarrow Z'$(Z'是Z^{+}的某个子集)形式的函数依赖，这样，函数依赖集S的闭包S^{+}是关于所有可能的属性集Z的这种函数依赖集的并集。

成立，也就是说，当且仅当 K 关于给定函数依赖集的闭包 K^+ 是 R 的所有属性的集合时，K 为超码（当且仅当 K 是不可约的超码时，K 是候选码）。

10.6 最小函数依赖集

假设 $S1$ 和 $S2$ 是两个函数依赖集，如果所有为 $S1$ 所蕴涵的函数依赖都为 $S2$ 所蕴涵，—— 即 $S1^+$ 是 $S2^+$ 的子集，则 $S2$ 是 $S1$ 的覆盖 ⊖，DBMS 只要实现了 S2 中的函数依赖，就自动实现 $S1$ 中的函数依赖。

如果 $S2$ 是 $S1$ 的覆盖，同时 $S1$ 是 $S2$ 的覆盖 —— 则 $S1$ 和 $S2$ 等价，即 $S1^+=S2^+$。很显然，如果 $S1$ 和 $S2$ 等价，则 DBMS 只要实现 $S1$ 中的函数依赖，就自动实现 $S2$ 中的函数依赖，反之亦然。

当且仅当函数依赖集满足以下条件时，该函数依赖集为最小函数依赖集 ⊜：

1) 每个函数依赖的右边（应变量）只含有一个属性（即它是单元素集合）。
2) 每个函数依赖的左边（自变量）是不可约的 —— 删除自变量的任何一个属性都将改变闭包 S^+（即会使 S 转变为一个不等价于原来的 S 的集合）。这种函数依赖被称为左部不可约的函数依赖。
3) 删除 S 中任何一个函数依赖都将改变它的闭包 S^+，即使 S 转变为一个不等价于原来的 S 的集合。

关于第 2 点和第 3 点，在这里要指出的是，为了知道如果删除某些元素是否会改变闭包，不必要清楚地知道闭包的内容。例如：观察大家熟悉的零件关系变量 P,有下列函数依赖：

```
P#→PNAME
P#→COLOR
P#→WEIGHT
P#→CITY
```

显而易见，该函数依赖集是最小依赖集：每个函数依赖中右边只含有一个属性，同样，左边也是不可约的，且任何一个函数依赖都不能被删除而不改变闭包（即不丢失信息）。相反，下面的函数依赖集不是最小依赖集。

1) P#→{PNAME，COLOR}　　：第一个函数依赖的右边不是单属性集
   P#→WEIGHT
   P#→CITY

2) {P#，PNAME}→COLOR　　：第一个函数依赖左边的 PNAME 可以删除而不改变闭包（即左边是可约的）
   P#→PNAME
   P#→WEIGHT
   P#→CITY

3) P#→P#　　：第一个函数可以删除而不改变闭包
   P#→PNAME
   P#→COLOR
   P#→WEIGHT
   P#→CITY

任何一个函数依赖集至少存在一个最小函数依赖集。假设函数依赖集为 S，根据分解规则，可以假定每个函数依赖的右边是单属性的而不会失去它的一般性（如果右边不是单属性的，则可以利用分解规则把它分解成单属性），接着考察每个函数依赖 f 左边的每一个属性 A，如果把

⊖ 有些书中覆盖的意思和本书的等价集相同。
⊜ 在书面上常称为“最小”。

A从f的左边删除而并不改变闭包，则把A从f的左边删除，然后考察S中剩余的每一个函数依赖f，如果把f删除而不改变闭包，则把f从S中删除，最后所得的集合S是和原来的函数依赖集S等价的最小函数依赖集。

例：假设给定关系变量R、A、B、C、D是R的属性集，R满足函数依赖：

$A \rightarrow BC$
$B \rightarrow C$
$A \rightarrow B$
$AB \rightarrow C$
$AC \rightarrow D$

现在计算该函数依赖的最小函数依赖集。

1) 把所有的函数依赖写成右边是单属性的函数依赖：

$A \rightarrow B$
$A \rightarrow C$
$B \rightarrow C$
$A \rightarrow B$
$AB \rightarrow C$
$AC \rightarrow D$

很显然，函数依赖$A \rightarrow B$出现了两次，可以删除其中的一次。

2) 可以把C从函数依赖$AC \rightarrow D$的左边删除，因为$A \rightarrow C$，根据增广律可以得出$A \rightarrow AC$，给定$AC \rightarrow D$，根据传递律可以得出$A \rightarrow D$。所以C在函数依赖$AC \rightarrow D$的左边是冗余的。

3) 接着发现可以删除函数依赖$AB \rightarrow C$，因为$A \rightarrow C$，根据增广律可得$AB \rightarrow CB$，又根据分解规则可以导出$AB \rightarrow C$。

4) 函数依赖$A \rightarrow C$由函数依赖$A \rightarrow B$和$B \rightarrow C$蕴涵，所以它可以删除。最后剩下下列函数依赖：

$A \rightarrow B$
$B \rightarrow C$
$A \rightarrow D$

该集合是不可约。

一个函数依赖集I是不可约的，且等价于某个函数依赖集S，则说I是S的最小等价依赖集。这样，如果要实现一个函数依赖集S，系统只要实现它的一个最小依赖集就足够了（重复一次：要计算最小依赖集I不必计算闭包S^+）。应该清楚的是给定函数依赖集的最小依赖集并不一定是唯一的（参考练习10.12）。

10.7 小结

函数依赖是一个关系变量中的两个属性集间的多对一关系。给定关系变量R，A和B是R的属性集的子集，当且仅当对于R的任意两个元组，如果A相等，则B必相等，那么可以说B函数依赖于A，记为$A \rightarrow B$。每一个关系变量R一定满足某些平凡的函数依赖。当且仅当函数依赖的右边（应变量）是左边（自变量）的子集时，该函数依赖为平凡的函数依赖。

一个函数依赖蕴涵其他的函数依赖。给定一个函数依赖集S，S中各个函数所蕴涵的函数依赖的集合成为S的闭包，记为S^+。S^+必然是S的超集。Armstromg公理系统为计算S的闭包S^+提供了一个有效且完备的基础理论（然而，事实上并不用它来计算）。从Armstromg公理系统

中可以导出其他一些有用的规则。

如果给定一个关系变量 R 的属性集的子集 Z 和一个该关系变量满足的函数依赖集 S，那么 Z 关于 S 的闭包 Z^+ 是 R 的属性集的一个子集，对于该子集中的任意一个属性 A，函数依赖 $Z \rightarrow A$ 都成立。如果 Z^+ 包含 R 的所有属性，则 Z 为超码（如果超码不可约，则该超码是候选码）。本章介绍了一种计算 Z 关于函数依赖集 S 的闭包 Z^+ 的算法，从而有了一种判断给定的函数依赖 $X \rightarrow Y$ 是否是 S^+ 的成员的简便方法：当且仅当 Y 是 X^+ 的子集时，$X \rightarrow Y$ 是 S^+ 的成员。

当且仅当两个函数依赖集 $S1$ 和 $S2$ 相互覆盖时，这两个函数依赖集等价。即当且仅当 $S1^+ = S2^+$ 时，$S1$ 和 $S2$ 等价。每一个函数依赖集至少等价于一个最小函数依赖集。一个最小函数依赖集应满足以下三个条件：（1）该函数依赖集的每一个函数依赖的右边只有一个属性，（2）删除该函数依赖集的任何一个函数依赖都将改变它的闭包；（3）删除该函数依赖集中任意一个函数依赖左边的任意一个属性都将改变该集合的闭包。如果 I 是 S 的最小依赖集，则实现 I 的函数依赖时自动实现 S 的函数依赖。

最后，应该注意的是前面介绍的思想不仅仅适用于函数依赖，还适用于一般的完整性限制条件：

- 一些限制条件是平凡的。
- 一些限制条件蕴涵其他限制条件。
- 给定一个限制条件的集合所蕴涵的限制条件的集合可以认为是给定限制条件集合的闭包。
- 确定一个限制条件是否属于一个闭包的问题 —— 即该限制条件是否被其他限制条件蕴涵的问题 —— 是个有趣的实际问题。
- 寻找一个限制条件集合的最小依赖集是个有趣的实际问题。

只是因为有一套有效且完备的公理系统才使得函数依赖比一般的完整性约束条件更容易处理。在本章和第 12 章的“参考文献”部分给出了一些参考文献，这些参考文献中介绍了其他一些约束 —— “MVDs”、“JDs”和“INDs” —— 对这些约束规则，该公理系统仍然适用。本书不再像介绍函数依赖一样介绍这些约束条件。

练习

10.1 假设 R 是一个 n 目的关系变量，该关系变量最多可以满足多少个函数依赖（包括平凡的函数依赖和非平凡的函数依赖）？

10.2 什么是 Armstrong 公理系统的有效性，什么是 Armstrong 公理系统的完备性？

10.3 根据函数依赖的基本定义，证明自反律、增广律和传递律。

10.4 证明 10.3 中的三个规则蕴涵自含规则、分解规则和合并规则。

10.5 证明 Darwen 的“通用一致性定理”，在证明时运用了前两题的什么规则？哪些规则可以看作是该定理的特例？

10.6 名词解释：（1）函数依赖集的闭包；（2）属性集关于给定函数依赖集的闭包。

10.7 列出关系变量 SP 中所有可能的函数依赖。

10.8 关系变量 $\{A, B, C, D, E, F, G\}$ 满足下列函数依赖

$A \rightarrow B$

$BC \rightarrow DE$

$AEF \rightarrow G$

计算{A，C}关于这个函数依赖集的闭包{A，C}$^+$，该函数依赖集是否蕴涵函数依赖$ACF \rightarrow DG$？

10.9 什么情况下说$S1$和$S2$等价？

10.10 什么是最小函数依赖集？

10.11 下面两个函数依赖集等价吗？

1) $A \rightarrow B$　　$AB \rightarrow C$　　$D \rightarrow AC$　　$D \rightarrow E$

2) $A \rightarrow BC$　　$D \rightarrow AE$

10.12 关系变量R{A，B，C，D，E，F}满足下列函数依赖：

$AB \rightarrow C$
$C \rightarrow A$
$BC \rightarrow D$
$ACD \rightarrow B$
$BE \rightarrow C$
$CE \rightarrow FA$
$CF \rightarrow BD$
$D \rightarrow EF$

找出该函数依赖集的最小依赖集。

10.13 关系变量TIMETABLE定义了如下属性（字段）：

D：一星期中的每一天（1~5）

P：一天中的一个时间段（1~8）

C：教室号码

T：教师姓名

L：课程名

当且仅当在时间{D：d，P：p}时，教师t在教室c教授课程l时元组{D：d，P：p，C：c，T：t，L：l}才在关系变量中出现。可以假定课程持续一个时间段，一个星期里所有课程的名称唯一，那么在该关系变量中存在哪些函数依赖？它的候选码有哪些？

10.14 关系变量NADDR定义了如下属性（字段）：NAME（姓名，唯一）、STREET（街道）、CITY（城市）、STATE（州）和ZIP（邮编），对于给定的邮政编码，只有唯一的州和城市与之对应。同样，给定一个街道、城市和州，只有唯一的一个邮政编码和它对应。给出该关系变量的一个最小函数依赖集和它的候选码。

10.15 关系变量R{A，B，C，D，E，F，G，H，I，J}满足下列函数依赖：

$ABD \rightarrow E$
$AB \rightarrow G$
$B \rightarrow F$
$C \rightarrow J$
$CJ \rightarrow I$
$G \rightarrow H$

该函数依赖集是最小依赖集吗？给出该关系变量的候选码。

参考文献和简介

10.1 W. W. Armstrong: “ Dependency Structures of Data Base Relationships,” Proc. IFIP Congress, Stockholm, Sweden (1974).

该文第一次用形式化的语言描述了函数依赖（它是 Armstrong公理系统的基础），该文还精确描述了侯选码这个概念。

10.2 Marco A. Casanova, Ronald Fagin, and Christos H. Papadimitriou: " Inclusion Dependencies and Their Interaction with Functional Dependencies," Proc. I st ACM SIGACT-SIGNIOD Symposium on Principles of Database Systems. Los Angeles, Calif. (March 1982).

内含依赖（inclusion dependencies, INDs）可以被看作是参照完整性的概括，例如，内含函数依赖

SP．S#↛S．S#

（和参考书上使用的符号不同）

说明出现在关系变量SP中的S#的集合是出现在关系变量S中的S#的集合的子集，这个例子事实上就是参照完整性，然而，内含依赖并不要求它的左边是外码或它的右边是候选码。注意：因为内含依赖和普通的函数依赖一样是反映多对一的关系，所以它们之间有共同点，但是内含依赖往往是跨越关系的，而函数依赖则是存在于一个关系变量内部的。

本文提供了一套完备且有效的内含依赖推理规则：

1) A↛B。

2) 如果AB↛CD，则A↛C且B↛D。

3) 如果A↛B且B↛C，则A↛C。

10.3 R. G. Casey and C. Delobel: " Decomposition of a Data Base and the Theory of Boolean Switching Functions," *IBM J. R&D 17*, No. 5 (September 1973).

本文指出：对于一个任意给定的关系变量，该关系变量所满足的函数依赖（该文中称为函数关系)集可以表示成" 布尔转化函数（boolean switching function）" 的形式，而且这种函数是唯一的。因为虽然原来的函数依赖可以用很多表面上看起来不同但实际上是等价的形式表示，每种表示形式都可以转化成一个表面上不同的布尔函数 —— 但这些函数可以利用布尔代数规则规约成相同的规范形式。所以，关系变量的分解问题（无损分解，见第11章）就和众所周知的布尔代数问题 —— 寻找和原来的关系变量及函数依赖相对应的布尔函数的" 质蕴涵的最小覆盖（a covering set of prime implicants） —— 等价。从而将关系变量的分解问题转化为等价的布尔代数问题，人们就可以运用一些熟悉的技术来解决该问题。

本文和[10.8]及第12章的一些参考文献最早把函数依赖理论和其他理论作了比较。

10.4 E. F. Codd: " Further Normalization of the Data Base Relational Model," in Randall J. Rustin (ed.). *Data Base Systems, Courant Computer Science Symposia Series 6*. Englewood Cliffs, N.J.: Prentice-Hall (1972).

本文最早提出了函数依赖这一概念（IBM的内部备忘除外)。本文的标题中的" 规范化 " 指的是第11章介绍的数据库设计规则。该文的目的是论证函数依赖这一概念在数据库设计中的作用。函数依赖事实上是对数据库设计问题的第一次科学思考，函数依赖本身的应用也很广泛。

10.5 E. F. Codd: " Normalized Data Base Structure: A Brief Tutorial," Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access, and Control, San Diego, Calif. (November 1971).

对参考书[10.4]中一些思想的介绍。

10.6 Hugh Darwen: " The Role of Functional Dependence in Query Decomposition," in C. J. Date and Hugh Darwen, *Relational Database Writings 1989-1991*. Reading, Mass.: Addison-Wesley (1992).

本文介绍了一个推理规则集，通过该规则集可以从一个关系变量所满足的函数依赖集中推导出任意一个从该关系变量导出的关系变量所能满足的函数依赖集。通过这些函数依赖集就可以确定由原来的关系变量导出的关系变量的候选码，这就是第 8章和第9章偶尔提到的候选码推导规则。本文还介绍了如何利用这些规则来显著提高 DBMS的性能。

10.7 Hugh Darwen: " OObservations [*sic*] of a Relational Bigot," presentation to BCS Special Interest Group on Formal Aspects of Computing Science, London, UK (December 21st, 1990).

10.8 R. Fagin: " Functional Dependencies in a Relational Database and Propositional Logic," *IBM J. R&D 21*, No. 6 (November 1977).

本文认为，Armstrong公理系统是和命题逻辑中的蕴涵系统严格等价的，即本文在函数依赖和命题之间定义了一个映射，然后证明当且仅当与函数依赖 f相对应的命题是与函数依赖集S相对应的命题集的逻辑推论时，函数依赖 f才是函数依赖集S的推论。

10.9 Claudio L. Lucchesi and Sylvia L. Osborn: " Candidate Keys for Relations," *J. Comp. and Sys. Sciences 17*, No. 2 (1978).

给出了计算一个满足给定函数依赖集的关系变量的所有候选码的算法。

部分练习答案

10.1 函数依赖是一个形式为 $A \rightarrow B$的断言，其中A、B是关系变量R的属性集，因为一个 n目的关系变量可以有2^n个不同的属性集，每个A和B可以有2^n不同的值，所以最多可以有 2^{2n}个函数依赖。

10.5

1) $A \rightarrow B$	(给出)
2) $C \rightarrow D$	(给出)
3) $A \rightarrow B \cap C$	(连接依赖, 1)
4) $C - B \rightarrow C - B$	(自含规则)
5) $A \cup (C - B) \rightarrow (B \cap C)(C - B)$	(复合规则，3，4)
6) $A \cup (C - B) \rightarrow C$	(是5的简化)
7) $A \cup (C - B) \rightarrow D$	(传递律，6，2)
8) $A \cup (C - B) \rightarrow B \cup D$	(复合规则，1，7)

证毕

所用的规则如上所示，以下是 Darwen定理的特例：合并规则、传递律、增广律和复合规则。下面这个用途广泛的规则也是 Darwen定理的特例：

如果$A \rightarrow B$且$AB \rightarrow B$，则$A \rightarrow C$

10.7 关系变量SP满足的所有函数依赖 —— 即SP的函数依赖的闭包如下所示：

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

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

{ S#, P# }      → { S#, P#, QTY }
{ S#, P# }      → { S#, P# }
{ S#, P# }      → { P#, QTY }
{ S#, P# }      → { S#, QTY }
{ S#, P# }      → { S# }
{ S#, P# }      → { P# }
{ S#, P# }      → { QTY }
{ S#, P# }      → { }
{ P#, QTY }     → { P#, QTY }
{ P#, QTY }     → { P# }
{ P#, QTY }     → { QTY }
{ P#, QTY }     → { }

{ S#, QTY }     → { S#, QTY }
{ S#, QTY }     → { S# }
{ S#, QTY }     → { QTY }
{ S#, QTY }     → { }

{ S# }          → { S# }
{ S# }          → { }

{ P# }          → { P# }
{ P# }          → { }

{ QTY }         → { QTY }
{ QTY }         → { }

{ }             → { }
```

10.8 $\{A, C\}^{+}=\{A, B, C, D, E\}$，后一部分的答案是“是”。

10.11 它们是等价的，假设把各个函数依赖按如下顺序编号：

1) $A \rightarrow B$
2) $AB \rightarrow C$
3) $D \rightarrow AC$
4) $D \rightarrow E$

首先，第3个函数依赖可以用下面的替代：

3) $D \rightarrow A$和$D \rightarrow C$

接着，根据第1和第2个函数依赖可知，第2个函数依赖可以用下面的替代：

2) $A \rightarrow C$

这样就剩下如下函数依赖：$D \rightarrow A$、$A \rightarrow C$和$D \rightarrow C$，由于$D \rightarrow C$为$D \rightarrow A$和$A \rightarrow C$所蕴涵，所以可以去掉，最后剩下：

3) $D \rightarrow A$

第1个函数依赖集和下面的最小依赖集等价：

$A \rightarrow B$
$A \rightarrow C$
$D \rightarrow A$
$D \rightarrow E$

第2个函数依赖集：

$A \rightarrow BC$
$D \rightarrow AE$

也和上面的最小依赖集等价，所以这两个函数依赖集等价。

10.12 第一步是把给定函数依赖集中的每一个函数依赖写成右边是单元素的函数依赖：

1) $AB \rightarrow C$
2) $C \rightarrow A$
3) $BC \rightarrow D$
4) $ACD \rightarrow B$

5) $BE \rightarrow C$
6) $CE \rightarrow A$
7) $CE \rightarrow F$
8) $CF \rightarrow B$
9) $CF \rightarrow D$
10) $D \rightarrow E$
11) $D \rightarrow F$

现在：

- 2蕴涵6，所以可以把6去掉。
- 8蕴涵$CF \rightarrow BC$（增广律），它和3一起蕴涵$CF \rightarrow D$（传递律），所以可以把10去掉。
- 8蕴涵$ACF \rightarrow AB$（增广律），11蕴涵$ACD \rightarrow ACF$（增广律），所以$ACD \rightarrow AB$（传递律），并且$ACD \rightarrow B$，这样4就可以去掉。

现在已经不可以进行进一步的归约，所以得到最小依赖集：

$AB \rightarrow C$
$C \rightarrow A$
$BC \rightarrow D$
$BE \rightarrow C$
$CE \rightarrow F$
$CF \rightarrow B$
$D \rightarrow E$
$D \rightarrow F$

另一种方法：

- 2蕴涵$CD \rightarrow ACD$（复合规则），它和4一起蕴涵$CD \rightarrow B$，所以可以用函数依赖$CD \rightarrow B$替代4。
- 2蕴涵6，所以可以把6去掉。
- 2和10蕴涵$CF \rightarrow AD$（复合规则），它又蕴涵$CF \rightarrow ADC$（增广律），它和4一起蕴涵$CF \rightarrow B$，所以可以把8去掉。
- 现在已经不能进行进一步的归约，所以得到最小依赖集：

$AB \rightarrow C$
$C \rightarrow A$
$BC \rightarrow D$
$CD \rightarrow B$
$BE \rightarrow C$
$CE \rightarrow F$
$CF \rightarrow D$
$D \rightarrow E$
$D \rightarrow F$

因此，可以看出，同一个函数依赖集可以有不同的最小依赖集。

10.13 候选码是：L、DPC和DPT。

10.14 把NAME、ST REET、CITY、STATE和ZIP分别简写成N，R，C，T和Z，则有：

$N \rightarrow RCT$，$RCT \rightarrow Z$，$Z \rightarrow CT$

等价的最小依赖集为：

$N \rightarrow R$　$N \rightarrow C$　$N \rightarrow T$　$RCT \rightarrow Z$　$Z \rightarrow C$　$Z \rightarrow T$

唯一的一个候选码是N。

10.15 提示：首先，该函数依赖集不是最小的，因为$C \rightarrow J$和$CJ \rightarrow I$一起蕴涵$C \rightarrow J$。第二，很明显，{A，B，C，D，G，J}是一个超码，即所有出现在函数依赖左边的属性的集合是超码，但是，因为$C \rightarrow J$，所以把J从超码中去掉，同时因为$AB \rightarrow G$，所以把G从超码中去掉。这样，超码中就剩下A、B、C和D，由于它们都没有在函数依赖集的任何一个函数依赖的右边出现，所以它们都不能从超码中去掉，最后得到的候选码是{A，B，C，D}。