

# 数据库设计与关系理论

## Database Design & Relational Theory

Normal Forms & All That Jazz

O'REILLY

机械工业出版社  
China Machine Press

C.J. Date 著  
卢涛 译

李颖 苏旭晖 审校

O'Reilly精品图书系列

数据库设计与关系理论

Database Design and Relational Theory:  
Normal Forms and All That Jazz

[英]戴特 (Date, C. J.) 著

卢涛 译

ISBN: 978-7-111-43292-0

本书纸版由机械工业出版社于2013年出版，电子版由华章分社（北京华章图文信息有限公司）全球范围内制作与发行。

版权所有，侵权必究

客服热线：+ 86-10-68995265

客服信箱：service@bbbvip.com

官方网址：www.hzmedia.com.cn

新浪微博 @研发书局

腾讯微博 @yanfabook

# 目 录

[题献](#)

[O'Reilly Media, Inc. 介绍](#)

[业界评论](#)

[译者序](#)

[作者简介](#)

[前言](#)

[先决条件](#)

[逻辑与物理设计的对比](#)

[致谢](#)

[第一部分 设置环境](#)

[第1章 篇首语](#)

[1.1 从文献摘录的一些引用](#)

[1.2 关于术语的说明](#)

[1.3 正在运行的示例](#)

[1.4 键](#)

[1.5 设计理论的地位](#)

[1.6 本书的目的](#)

[1.7 结束语](#)

[习题](#)

[第2章 预备知识](#)

[2.1 概览](#)

[2.2 关系及关系变量](#)

[2.3 谓词和命题](#)

[2.4 更多的供应商和零件](#)

## 习题

# 第二部分 函数依赖、BOYCE/CODD范式及相关事宜

## 第3章 规范化：一些通则

### 3.1 规范化用于两个目的

### 3.2 更新异常

### 3.3 范式层次结构

### 3.4 规范化和约束

### 3.5 结束语

## 习题

## 第4章 函数依赖和BCNF（非正式的）

### 4.1 第一范式

### 4.2 函数依赖

### 4.3 键的重新审视

### 4.4 第二范式

### 4.5 第三范式

### 4.6 Boyce/Codd范式

## 习题

## 第5章 函数依赖和BCNF（正式的）

### 5.1 初步定义

### 5.2 函数依赖

### 5.3 Boyce/Codd范式

### 5.4 希思定理

## 习题

## 第6章 保持函数依赖

[6.1 遗憾的冲突](#)

[6.2 第二个例子](#)

[6.3 第三个例子](#)

[6.4 第四个例子](#)

[6.5 一个能够工作的过程](#)

[6.6 恒等分解](#)

[6.7 关于冲突的更多内容](#)

[6.8 独立投影](#)

[习题](#)

[第7章 FD公理化](#)

[7.1 阿姆斯特朗公理](#)

[7.2 附加规则](#)

[7.3 证明附加规则](#)

[7.4 另一种闭包](#)

[习题](#)

[第8章 反规范化](#)

[8.1 “反规范化是为了性能”吗](#)

[8.2 反规范化是什么意思](#)

[8.3 什么不是反规范化（I）](#)

[8.4 什么不是反规范化（II）](#)

[8.5 反规范化是有害的（I）](#)

[8.6 反规范化是有害的（II）](#)

[8.7 结束语](#)

[习题](#)

[第三部分 连接依赖、第五范式及其他相关事项](#)

## 第9章 连接依赖及5NF（非正式的）

### 9.1 连接依赖的基本思路

### 9.2 一个属于BCNF但不属于5NF的关系变量

### 9.3 循环规则

### 9.4 结束语

### 习题

## 第10章 连接依赖及5NF（正式的）

### 10.1 连接依赖

### 10.2 第五范式

### 10.3 被键蕴含的JD

### 10.4 一个有用的定理

### 10.5 FD不是JD

### 10.6 更新异常再探

### 习题

## 第11章 隐式依赖关系

### 11.1 无关的分量

### 11.2 结合分量

### 11.3 不可约的JD

### 11.4 小结

### 11.5 追逐算法

### 11.6 结束语

### 习题

## 第12章 多值依赖和4NF

### 12.1 一个介绍性的例子

[12.2 多值依赖（非正式的）](#)

[12.3 多值依赖（正式的）](#)

[12.4 第四范式](#)

[12.5 公理化](#)

[12.6 嵌入式依赖](#)

[习题](#)

[第13章 额外的范式](#)

[13.1 相等依赖](#)

[13.2 第六范式](#)

[13.3 超键范式](#)

[13.4 无冗余范式](#)

[13.5 域-键范式](#)

[13.6 结束语](#)

[习题](#)

[第四部分 正交](#)

[第14章 正交设计原则](#)

[14.1 规范化的两个欢呼声](#)

[14.2 一个启发性的例子](#)

[14.3 一个更简单的例子](#)

[14.4 元组与命题](#)

[14.5 第一个例子再探](#)

[14.6 第二个例子再探](#)

[14.7 最终版本](#)

[14.8 澄清](#)

[14.9 结束语](#)



## 习题

### 第五部分 冗余

#### 第15章 我们需要更多的科学

##### 15.1 一点历史

##### 15.2 数据库设计是谓词设计

##### 15.3 例1

##### 15.4 例2

##### 15.5 例3

##### 15.6 例4

##### 15.7 例5

##### 15.8 例6

##### 15.9 例7

##### 15.10 例8

##### 15.11 例9

##### 15.12 例10

##### 15.13 例11

##### 15.14 例12

##### 15.15 管理冗余

##### 15.16 改善定义

##### 15.17 结束语

## 习题

### 第六部分 附录

#### 附录A 主键是良好的，但不是必需的

##### A.1 为PK：AK区别辩护的论据

##### A.2 具有多个键的关系变量

[A.3 发票和发货例子](#)

[A.4 是否每个实体类型一个主键](#)

[A.5 申请人及员工的实例](#)

[A.6 结束语](#)

[附录B 冗余回顾](#)

[附录C 重要论文回顾](#)

[附录D 习题答案](#)

[第1章](#)

[第2章](#)

[第3章](#)

[第4章](#)

[第5章](#)

[第6章](#)

[第7章](#)

[第8章](#)

[第9章](#)

[第10章](#)

[第11章](#)

[第12章](#)

[第13章](#)

[第14章](#)

[第15章](#)

## 题献

在计算中，优雅不是可有可无的奢侈品，而是决定成败的品质。

——艾兹格·W·迪科斯彻（Edsger W.Dijkstra）

不良的设计是设计师最常见的毛病。

——赫西奥德（Hesiod）

要注意的是，如果本书有任何部分是平淡的，那么其中包含设计精髓。

——理查德·斯蒂尔爵士（Sir Richard Steele）

一个正式的设计原则理念，往往会由于含糊的文化/哲学谴责，如：“扼杀创造力”，而遭到拒绝，更为明显的是……“人文”上的一个浪漫的愿景，其实是将技术上的无能给理想化了……

因为可靠性和智能控制的缘故，我们必须保持设计简单且井井有条。

——艾兹格·W·迪科斯彻（Edsger

我的设计是非常值得自豪的。

——佚名

## O'Reilly Media, Inc. 介绍

O'Reilly Media通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始，O'Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了Make杂志，从而成为DIY革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版，在线服务或者面授课程，每一项O'Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

## 业界评论

“O'Reilly Radar博客有口皆碑。”

——Wired

“O'Reilly凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O'Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照Yogi Berra的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去Tim似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”



## 译者序

2012年年底，当谢晓芳编辑把这本书在亚马逊网站的链接发给我时，我为有机会翻译数据库领域内知名作者的著作感到荣幸，由于他的著作已有多本中文翻译版出版，这既给我提供了良好的学习机会，又给我增添了压力，希望自己不要破坏原书的风貌。

这本书比较薄，因为它是一系列书中的第二本，专门讲数据库设计中用到的关系理论。它的每一章也都比较短，具体介绍了各种范式、依赖、约束、冗余的定义和规范化的算法、方法，并澄清了不少文献中的谬误，叙述还是相当通俗易懂的。个人觉得本书专门介绍了设计领域的最新进展，如6NF等新的范式，这在市面的书籍中是独一无二的，对于数据库设计人员很有参考价值。

本书对于数据库开发人员也有价值，因为它为解决具体问题提供了思路，比如有一道邀请参加聚会的习题，在第3届国际SQL/NO SQL大赛中作为竞赛题目。本书提供的计算函数依赖闭包的算法是那道题的核心知识点。

这本译作的完成离不开大家的帮助。



首先感谢我妻子李颖一如既往的支持，她是英语专业毕业，作为我的第一读者，帮助我检查出很多生硬的译文，并把语句梳理得更加通顺。另一方面，她用了很长时间照顾家庭，发挥了不可替代的作用。

感谢老战友苏旭晖（ITPUB Oracle开发版版主，网名newkid），他在国外工作，技术水平和文笔俱佳。他帮助我解决了大量语言和技术难点，使本书译文在技术上更准确。

感谢谢晓芳编辑，她对译文从专业的角度进行把关，对我不擅长的内容进行了很多润色，使之错误更少。我从她修改的译文中也学到了很多。

最后还要感谢我儿子卢〇一，他知道我在翻译书稿就默默配合我，让我能够专注于本书的翻译，本书的出版也有他的一份贡献。

最后希望这本书对读者有帮助。为了争取译好这本书，我尽我所能翻阅了作者其他著作的中译本，参考了前辈的译文，也参考了离散数学中的相关术语。但由于译者经验和水平有限，译文中难免有不妥之处，恳请读者批评指正！



# 作者简介

C.J.Date是一位专门研究关系数据库技术的独立作者、讲师、研究员和顾问。他最著名的著作是《数据库系统导论（原书第8版）》[\[1\]](#)，这本书已经销售了大约85万册，并被世界范围内的几百所大学和综合性大学作为教材采用。他同时也是许多数据库管理书籍的作者，包括最近出版的下列作品：

- 《数据库、类型和关系模型》[\[2\]](#)

- Trafford出版社：Logic and Databases: The Roots of Relational Theory（2007）

- From Apress出版社：The Relational Database Dictionary,Extended Edition（2008）

- From Trafford出版社：Database Explorations: Essays on The Third Manifesto and Related Topics（coauthored with Hugh Darwen, 2010）

- 《数据库探索：对第三宣言和相关主题的评论》

■From Ventus出版社: Go Faster! The TransRelational<sup>TM</sup> Approach to DBMS Implementation (2011)

■From O'Reilly出版社: SQL and Relational Theory: How to Write Accurate SQL Code (2nd edition, 2012)

### ■《SQL与关系数据库理论》

Date先生在2004年被选入计算行业名人堂。他因具有一流的将复杂的技术专题用一种清晰且很容易理解的方式加以解释的能力而享有盛誉。

[1]该书由机械工业出版社引进并出版。

[2]该书由机械工业出版社引进并出版。

# 前言

本书源自《Database in Depth: Relational Theory for Practitioners》[\[1\]](#)（O'Reilly, 2005年）一书中相对较短的一章。那本书后来被《SQL and Relational Theory: How to Write Accurate SQL Code》[\[2\]](#)（O'Reilly, 2009年）所取代，因为关于数据库设计的材料有点偏离后一本书的主题，所以此部分材料不再作为一章出现，而是转变成一个（有点长）的附录。随后，我开始着手后一本书第2版的写作。[\[3\]](#)在此过程中，我发现关于设计的主题有许多内容需要写，附录有扩展到与书的其余部分比例失调的危险。正如我已经指出的，由于关于设计的主题与后来那本书的主要重点非常不吻合，因此，我决定快刀斩乱麻地把这部分材料独立出来，为它单独写一本书：即你现在看到的这本。

上述背景立即引出了以下三点。

■首先，本书假设你熟悉《SQL与关系数据库理论》中包含的知识（特别是，假设你知道到底什么是关系、属性和元组）。我对这种情况并不感到抱歉，因为本书是针对数据库专业人员的，而数据库专业人员，无论如何都应该真正地

熟悉我早期那本书中的大部分内容。

■其次，尽管有前一点的说明，但这本书与我早期那本书之间不可避免地会有少量的重叠。但是，我已经尽力将这种重叠限制到最低限度。

■再次，同样不可避免地，本书参考了我早期那本书中的许多内容。现在，大多数在本书中参考的其他出版物都以完整的形式注明，如下面的例子所示：

Ronald Fagin: “Normal Forms and Relational Database Operators,” Proc.1979 ACM SIGMOD Int.Conf.on Management of Data,Boston,Mass. (May/June 1979) .

但是，尤其对参考我早期那本书的情况，从此刻开始我会按照它的中文书名《SQL与关系数据库理论》[\[4\]](#)的形式单独给出。而且，我这个缩写书名专门指的是该书的第2版（在任何有区别的地方）。

实际上，多年来，我已经在许多出版物的不同章节中涉及了设计理论的各个方面，而本书的目的之一是保留我早期那些书中的精华。但它不只是将以前出版的资料拼凑在一起，我衷心地希

望它不会被视为一本大杂烩。原因之一是，它包含很多新的材料。另外，它对整个主题的描述更连贯，而且在我看来视角更好（我自己多年来已经学到了很多！）。事实上，尽管某些部分的文字基于一些早期的出版物，但我确信那些有问题的材料已完全改写并且得到了改善。

现在，关于数据库设计的书籍并不短缺，那么本书有什么与众不同呢？事实上，我不认为市面上有与这本书非常相似的书。市面上的确有许多讲述设计实践的书籍（在我看来，质量参差不齐），但那些书（再次，以我偏执的看法）通常没有很好地解释基本理论。市面上也有几本关于设计理论的书，但它们往往太偏理论，并在表达风格上也过于学术化，而不适合实践者使用。我想要做的是弥合这种差距，换句话说，我想用实践者应该能够理解的方式来解释理论，并且我想说明为什么理论具有很大的实际意义。我并不想都做到详尽无遗，也不想讨论理论的每一个细节，但我想把精力都集中在对我似乎很重要的组成部分上（不过，当然，我对确凿内容的处理，就其本身而言，应该是精确和准确的）。另外，我的目标是将正式和非正式的内容审慎地交融，换句话说，我想提供一个理论的简单介绍，以便：

a.你可以使用重要的理论成果，以帮助你做实际的设计，并且

b.如果你倾向于学习理论，你就可以去阅读更学术的文章，并理解它们。

为了保持可读性，我特意把本书写得相当短小精炼，我也特意使每章都比较短。（我是一个在容易理解的小块中传递信息的坚定信徒。）而且，每章都包含了一组习题（其中大部分习题答案可参见书后的附录D），即使你不能把所有的习题都做一遍，我也建议你好好做一些。其中有些习题的目的是为了说明如何在实践中应用理论的思路，其余的习题提供（如果不是在习题中，就是在答案中）了更多与主题相关的信息，它们远远超过了本书正文包含的内容；还有些习题，例如，要求你证明一些简单的分析结果，其意图是让你对“关于像一个理论家那样思考”涉及哪些方面，获得一定的了解。总之，我想对什么是设计理论和为什么它是这样的提出一些见解。



## 先决条件

我的目标受众是数据库专业人员，特别是对数据库设计的兴趣不凡的数据库专业人员。因此，我特别假定你相当熟悉关系模型，或者至少熟悉该模型的某些方面（第2章将更加详细地深入探讨这些内容）。正如已经指出的，熟悉

《SQL与关系数据库理论》对理解本书会有很大的帮助。注意：我想要提及，我还有一个在这本书的基础上举办的研讨会的现场录像。欲知详情，请参阅

[www.justsql.co.uk/chris\\_date/chris\\_date.htm](http://www.justsql.co.uk/chris_date/chris_date.htm)。

[1] 中文版书名：《深度探索关系数据库——实践者的关系理论》。

[2] 中文版书名：《SQL与关系数据库理论》。

[3] 已在2012年由O'Reilly出版。

[4] 该书第1版的中文翻译版已经出版，书名为《SQL与关系数据库理论》，在本书中，暂且假设第2版的中文翻译版与第1版书名相同。

## 逻辑与物理设计的对比

本书是关于设计理论的，因此，从定义上看，它是关于数据库逻辑设计，而不是物理设计的。当然，我不是说物理设计不重要（当然不是），我是说它是一个不同的活动，独立于逻辑设计并在其后实施。更详尽地讲，做设计的“正确”方式如下所示：

- 1.首先做一个清晰的逻辑设计。然后，作为一个单独的和随后的步骤：

- 2.将逻辑设计映射到目标DBMS正好支持的物理结构上。[\[1\]](#)

请注意，因此，物理设计应当从逻辑设计推导而来，而不是相反。（事实上，在理想的情况下，系统应能够“自动”从逻辑设计获得物理设计，而在这个过程中完全不需要人为参与。）

再次重申，本书是关于设计理论的。因此，它不涉及各种特设的设计方法，比如多年来先后提出的实体/关系模型之类的设计方法。当然，我知道这些方法中的一部分在实践中使用相当广泛，但事实仍然是，它们从坚实的理论基础方法得到的东西相对较少。因此，它们大多超出了这

样一本书的范围。不过，我会不时对那些方法作少量的评论（特别是在第8章和第15章及附录A中）。

**[1]**DBMS=数据库管理系统。请注意，DBMS和数据库在逻辑上是不同的！遗憾的是，业界通常使用术语数据库指某些DBMS产品，如Oracle，或碰巧在特定计算机上安装的这样的特定产品的一个副本。本书不采取这种用法。因为问题是，如果把DBMS称作数据库，那又该怎么称呼数据库呢？

## 致谢

我要感谢Hugh Darwen、Ron Fagin、David McGoveran和Andy Oram，他们一丝不苟地审阅了本书的初稿。这些审校者帮助我纠正了许多误解（其实，多得超乎我的预想）。当然，不用说，本书任何遗留的错误，责任都在我。我也想感谢Chris Adamson对特定技术问题给予我的帮助，以及我的妻子Lindy在我创作这本书的整个过程中对我一如既往的支持。

C.J.Date

加利福尼亚州希尔兹堡

# 第一部分 设置环境

本书此部分包含两章，它们的章名（分别是“篇首语”和“预备知识”）大体上是不言而喻的。

## 第1章 篇首语

(On being asked what jazz is: )

Man,if you gotta ask,you'll never know.

——Louis Armstrong (attrib.)

本书英文版有一个副书名“Normal Forms and All That Jazz”（范式以及其他<sup>[1]</sup>）。很显然，需要对此有些解释！首先，当然，我谈论的是设计理论，大家都知道范式是该理论一个重要的分量，因此这是副书名第一部分的来源。但是，理论不仅仅是范式，它还有更多的内容。这个事实说明了副书名的第二部分。第三，这是很遗憾的情况，从实践者的角度来看，无论如何，设计理论充满了术语和概念，它似乎很难理解且似乎与实际的设计工作没多大关系。这就是为什么我用通俗的形式（不是说俚语的性质）说出副书名的后一部分的原因，我想传达这样一种想法或印象，即虽然有时我们一定会处理“困难”的材料，但我会尽我所能让这些材料既不令人气馁又不令人胆怯。但当然，是否我已经成功地达成这一目标需要你来判断。

我还想就设计理论在实际设计工作中是否有

用武之地这一问题多说几句。请允许我澄清，没有人可以或应该要求数据库设计是很容易的。但是一个好的理论知识一定会起帮助作用。事实上，如果你想要正确地做设计，如果你想让你建立的数据库像它们理所应当的那样强大、灵活和准确，那么你就必须掌握设计理论。至少，如果你想自称是一位专业人士，那么你别无选择。设计理论是数据库设计的科学基础，就像关系模型是一般数据库技术的科学基础一样。正如任何专门从事一般的数据库技术工作的人都需要熟悉关系模型，所以从事数据库设计工作的人需要特别熟悉设计理论。适当的设计是非常重要的！毕竟，数据库位于我们在计算机领域做的这么多事情的中心，所以如果它的设计不好，就会产生非常广泛的负面影响。

## 1.1 从文献摘录的一些引用

因为我们将要讨论很多关于范式的问题，所以我想这可以从文献中的几个引用开始说起，虽然它们也许不引人深思，但是有趣的。当然，范式整体概念的出发点是第一范式（First Normal Form, 1NF），因此一个明显的问题是：你知道什么是1NF吗？如下面的引用展示的（来源省略，以保护犯了错误的人），很多人不知道。

■为了实现第一范式，表中的每个字段必须传达唯一的信息。

■如果一个实体的所有属性都是单值，那么它符合第一范式（1NF）。

■当且仅当所有相关的域只包含原子值，一个关系才符合1NF。

■如果表中包含没有重复的组属性，那么它符合1NF。

现在，可能有人会说，即使这些引用不全对，也至少有一些是隐约正确的，但其实它们全都不可救药地马虎，即使它们一般是正确的。

（如果你想知道，我将在第4章中对1NF准确地加以定义）。

让我们仔细看看这是怎么回事。这里再次给出上述第一条引用，现在给出完整的引用。

■为了实现第一范式，表中的每个字段必须传达唯一的信息。例如，如果你有一个Customer表，其中有两列用于保存电话号码，你的设计会违反第一范式。第一范式是很容易实现的，因为人们很少会遇到在一个表中保存重复信息的需



求。

所以很明显，这里正在谈论的设计，看起来像这样：

CUSTNO	PHONENO1	PHONENO2	...
--------	----------	----------	-----

现在，我不能确定这是否是一个很好的设计，但它肯定不违反1NF。（我之所以不能确定这是一个很好的设计，因为我不知道究竟什么是“用于保存电话号码的两列”。短语“在一个表中保存重复信息”表明，我们把相同的电话号码记录了两次，但这样的解释表面上是荒谬的，但是，即使这种解释是正确的，它仍然构不成对1NF的违反）。

下面是对另一个文献的引用：

■第一范式.....表明表都不应该存在字段的“重复组”.....重复组是指你一遍又一遍地重复相同的基本属性（字段）。很好地说明这个情况的一个例子是，当你想存储你在一家杂货店买的物品时.....[然后大概是为了说明重复组的概念，该作者举了一个表的例子，表名为Item（物品），它的各列分别称为Customer、Item1、

Item2、Item3和Item4（客户、物品1、物品2、物品3和物品4）]:

CUSTOMER	ITEM1	ITEM2	ITEM3	ITEM4
----------	-------	-------	-------	-------

那么，这样的设计几乎可以肯定是坏的，如果客户购买的物品不是正好四个，会发生什么情况？——但它坏的原因，并不在于它违反1NF，正如前一个例子，其实，这是一个符合1NF的设计。虽然不严密地说，1NF确实意味着“没有重复的组”，但“重复组”不是“当你一遍又一遍地重复相同的基本属性”。（第4章解释1NF实际上是什么，并将解释它的本质。）

这一个（在互联网上找到的求助帖）怎么样？除了我添加的一些黑体字，我绝对逐字地引用了它：

■我一直在试图找到在Access中规范化表的正确方式。按我的理解，它会从第一范式开始，再到第二范式，然后是第三范式。通常情况下，那样可能就足够了，但有时需要进一步规范到第五范式和第六范式。然后，也有科布（Cobb）[\[2\]](#)第三范式。这一切对我来说很有意义。我将要在

下周开始讲授一门课，我刚刚得到了这门课的教科书。但它说的东西与我的理解完全不同。它指出，第二范式只适用于有一个多字段主键的表，第三范式只适用于有一个单字段键的表。而它对第四范式的解释是，对于某个符合第一范式的表，如果主键和非键字段之间没有独立的一对多的关系，那么它符合第四范式。请问有人可以帮我澄清吗？

还有一个（这次带有一个“有帮助”的回应）：

■>我不明白“规范化”是什么意思。你能具体说明你说的规范化规则指的是什么吗？

>我的模式以什么样的方式没有规范化？

规范化：将重复的东西替换为参考原来的东西的过程。

例如，已知“约翰是一个人”并且“约翰在军队服役”，人们可以看出，在第二个句子中的“约翰”与第一个句子中的“约翰”是重复的。使用系统提供的方法，第二句应存储为“->约翰在军队服役”。

[1]原文All That Jazz是一个英文习语，意为其他，作者提到爵士乐（Jazz）是双关语。

[2]可能是Codd之误，即BCNF中的C。

## 1.2 关于术语的说明

我敢肯定，你已注意到，在上一节中的引用的几个定义都是用大家最熟悉的“对用户友好”的术语表示的，比如，表、行和列（或字段）。相比之下，在本书中，我会倾向于支持更正式的术语，如：关系（**relation**）、元组（**tuple**）（英文单词**tuple**发音通常与**couple**（夫妇）押韵）和属性（**attribute**）。如果我这个决定使上下文有点难以理解，我对此表示道歉。但我这么做确实有我的理由。正如我在《SQL与关系数据库理论》[\[1\]](#)一书中说过的：

如果对用户更友好的术语能有助于使观点更易于接受，那么我会普遍认同使用它们的想法。然而，在目前的情况下，在我看来，令人遗憾的是，它们不能使观点更易于接受，相反，它们歪曲了观点，而且恰恰对真正的理解带来了严重的障碍。事实是，关系不是表，元组不是行，属性不是列。而且虽然在非正式的环境下假装正式术语可能是可以接受的（确实，我自己经常正好是这么做的），但是我认为，只有当我们都理解对用户更加友好的术语只是对真理的一个近似表述并且完全无法捕获本质上到底发生了什么时，这么做才是可以接受的。换句话说，如果你了解真

实状况，那么明智地使用对用户友好的术语可能是一个好主意，但为了从一开始就学习和领悟真实状况，你真的需要掌握正式术语。

基于上述原因，让我补充一点，（正如我在序中说的）我确实假设你确切知道什么是关系、属性和元组，尽管事实上，可以在第5章发现这些结构的正式定义。

还有另外一个需要我解决的术语问题。关系模型当然是一个数据模型。然而，遗憾的是，在数据库领域里，后一个术语有两个完全不同的含义。[\[2\]](#)数据模型的第一个也是更基本的含义是这样的。

■定义：数据模型（第一个含义）是对数据结构与数据操作等的一种抽象、“自包含的”逻辑定义，这些内容共同组成用户交互的抽象机制。

这是我们心目中的含义，特别是当我们谈论关系模型时：在关系模型中，数据结构是关系，而数据操作是投影、连接等关系操作。（至于说定义中的“等”，它涵盖键、外键，以及各种相关的概念等事宜）。

术语数据模型的第二个含义如下。

■定义：数据模型（第二个含义）是某些特定企业的数据，尤其是持久性数据的一个模型。

换言之，数据模型的第二个含义仅仅是一个（逻辑上的，并可能有些抽象的）数据库设计。例如，我们可能会讲一些银行、医院，或者政府部门的数据模型。

在解释了这两种不同含义的数据模型之后，我想提醒你注意下面这个比喻，我认为它很好地阐明了二者之间的关系。

■数据模型的第一个含义就像是一种编程语言，其结构可以用来解决很多具体的问题，但它们本身并没有与任何具体问题直接发生联系。

■数据模型的第二个含义就像是一个用编程语言编写的特定程序，它使用第一个含义的模型所提供的设施，依据该术语的第一个含义，解决一些具体问题。

综上所述，如果我们谈论的是数据模型的第二个含义，那么我们可能会合理地用复数形式讲多个“关系模型”，或“一个”（不定冠词）关系模型。但是，如果我们谈论的是数据模型的第一个含义，那么只有一种关系模型，而它称为“关系

模型”（具有定冠词）。

现在，你可能已经知道，大多数数据库设计方面的著作，尤其如果它们的重点是“程序”而不是底层的理论，那么它们使用的术语“模型”或“数据模型”，只表示其第二个含义。但是，请注意，小心！——本书不遵循这种做法，事实上，除了偶尔提到关系模型，我根本不使用“模型”这个术语。

[1]我在前言中提醒过你，在整本书中，我使用《SQL与关系数据库理论》作为我的《SQL and Relational Theory: How to Write Accurate SQL Code》（2nd edition, O'Reilly, 2012）一书的缩写形式。

[2]这一观点无疑是正确的。然而，一位审校者想让我补充的是，数据模型这两方面的含义本质上可以被认为是在不同的抽象层次上的相同概念。



## 1.3 正在运行的示例

现在介绍一个例子，我将把它用作在本书其余章节讨论的基础，它就是大家熟悉的（而不是陈腐的）供应商和零件数据库例子。（我很抱歉又一次拖出老战马，但我相信，在各种不同的书籍和出版物中使用本质上相同的例子可以有助于学习，而不是阻碍学习）。示例值如图1-1所示。[\[1\]](#)详细说明如下所示。

■ **供应商（supplier）**：关系变量（Relvar）S表示供应商。[\[2\]](#)每个供应商有一个对该供应商唯一的供应商编号（SNO），一个不一定是唯一的（虽然图1-1中的SNAME值正好是唯一的）名称（SNAME），一个状态值（STATUS），表示供应商之间的某种排名或偏好类型，以及一个位置（CITY）。

■ **零件（part）**：关系变量P表示零件（更准确地是零件的种类）。每一种零件有一个唯一的零件编号（PNO），一个不一定是唯一的名字（PNAME），一种颜色（COLOR），一个重量（WEIGHT）和一个那种零件的存储地点（CITY）。

■ **发货（shipment）**：关系变量SP表示发货

（它显示哪些零件是由哪些供应商提供的或发货的）。每次发货有一个供应商编号（SNO）、一个零件编号（PNO）和一个零件数量（QTY）。另外，在我假设的例子中，在任何时候对于一个给定的供应商和一种给定的零件，最多有一次发货。所以每次发货有一个唯一的供应商编号/零件编号组合。

S

SNO	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	30	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

PNO	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12.0	London
P2	Bolt	Green	17.0	Paris
P3	Screw	Blue	17.0	Paris..
P4	Screw	Red	14.0	London
P5	Cam	Blue	12.0	Paris
P6	Cog	Red	19.0	London

SP

SNO	PNO	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 1-1 供应商和零件数据库示例值

[1]图1-1中显示的值在两个小的方面与我的其他书籍有所不同。首先，供应商S2的状态为30，而不是10。其次，零件P3的城市是巴黎（Paris），而不是奥斯陆（Oslo）。我这么做的原因此后将变得明确。

[2]如果你不知道是什么关系变量，现在你只需要把它当做通常数据库意义上的一个表。请参阅第2章查看更深入的解释。

## 1.4 键

在进一步讨论之前，我需要复习熟悉的“键”这个术语在关系意义上的概念。首先，我敢肯定，你知道，每一个关系变量至少有一个候选键。一个候选键基本上只是一个独特的标识符；换句话说，它是属性的一个组合，通常正好是单个属性的一个“组合”，但不总是如此。关系变量中的每一个元组都有一个唯一的组合值。例如，对于图1-1的数据库。

■因为每家供应商都有一个唯一的供应商编号且每个零件都有一个唯一的零件编号，所以{SNO}是S的一个候选键，而{PNO}是P的一个候选键。

■至于发货，因为假设在任何时候对于一个给定的供应商和一个给定的零件最多有一次发货，所以{SNO,PNO}是SP的候选键。

注意，候选键以大括号的方式表示，再重复一遍，候选键总是属性（即使在集合中只包含一个属性时）的组合或集合，而集合在论文上的常规表示法是用大括号包含的一个逗号分隔的元素列表。注意：有用的术语逗号分隔列表

（commalist）的定义如下：假定xyz是一些语法构

造（例如，“属性名称”）。那么，术语xyz的逗号分隔列表表示零个或多个xyz组成的序列，其中每一对相邻的xyz都由逗号分隔（逗号之前或之后或前后的一个或多个空格是任选的）。

接下来，我敢肯定，你也知道，主键是一个为了某种特殊处理而以某种方式挑选出来的候选键。现在，如果所涉及的关系变量只有一个候选键，那么如果我们称它为主键，也不存在任何真正的区别。但是，如果在关系变量中有两个或两个以上候选键，那么通常选择其中之一作为主键，这意味着，它以某种方式“比其他候选键更平等”。例如，假设供应商总是有一个唯一的供应商编号和一个唯一的供应商名称，那么{SNO}和{SNAME}两者都是候选键。然后，比如，我们可能会选择{SNO}作为主键。

现在注意，我说的是通常选择一个主键。事实上，虽然这是通常的，但它不是100%必要的。如果只有一个候选键，那么我们别无选择也没有什么问题；但如果有两个或多个候选键，那么选择其中一个并把它当作主键，似乎有一点点的随意性，至少对我来说是这样的。（当然在有的情况下，似乎没有任何充分的理由作出这样的选择，甚至有可能有很好的理由不这样做。附录A详细阐述了有关事项。）因为熟悉的原因，我通

常会遵循自己的主键原则，在这本书中，像图1-1中的画面，我会用双下划线表示主键的属性，但我想强调的是，它其实是候选键，而不是主键，这从关系的角度来看是显著的，从设计理论的角度来看也的确如此。部分是由于这样的原因，从这里开始，以后我将使用非限定性的术语“键”表示任何候选键，无论所涉及的候选键是否已额外地指定为主键。（为了避免你误会，其实主键比其他候选键享有的特殊处理主要是语法，无论如何，这不是根本的，也不是非常重要的。）

更多的术语：首先，一个涉及两个或两个以上属性的键称为复合键（而一个非复合键有时称为简单键）。其次，如果一个给定的关系变量有两个或两个以上键而其中之一被选为主键，那么其他键有时称为备用键（alternate key，见附录A）。再次，外键是某个关系变量R2的属性FK的组合或集合，其中每个FK的值必须等于某个关系变量R1的某个键K的某个值（R1和R2不一定非要是不同的）。<sup>[1]</sup>参考图1-1，例如，在关系变量SP中{SNO}和{PNO}两者都是外键，分别对应关系变量S和P中的键{SNO}和{PNO}。

<sup>[1]</sup>这个定义是故意偏简单的（尽管就目前而言，它已经足够好）。可以在《SQL与关系数据库理论》中找到一个更好的定义。



## 1.5 设计理论的地位

为了重申我在序中说的某些内容，对于术语设计的意思，我指的是逻辑设计，而不是物理设计。逻辑设计关注的是数据库对用户来说看起来像什么（不严格地说，这意味着，存在什么关系变量以及应用在那些关系变量上的是什么约束），相反，物理设计关注的是如何把给定的逻辑设计映射到物理存储上。<sup>[1]</sup>术语设计理论专指逻辑设计，而不是物理设计——物理设计的要点是必须依赖于目标DBMS的某些方面（尤其是性能方面），而逻辑设计是，或者应该是，独立于DBMS的。在整本书中，以后非限定性术语设计应理解为专指逻辑设计，除非有与此相反的明确陈述。

现在我们知道，设计理论不是关系模型的一部分，更确切地说，它是在关系模型之上建立的一个独立理论。（一般认为把它作为关系理论的一部分是适当的，但再次重申，说它是关系模型本身的一部分是不对的。）因此，设计概念，如进一步规范化，本身基于更基本的概念，如关系代数的投影和连接操作，而这些操作是关系模型的一部分。（所有这些都表明，当然可以认为，设计理论是关系模型的一个合理的结果，至少部



分如此。换句话说，虽然它与一般的关系模型不符合，但符合在它基础上建立的设计理论。）

逻辑设计的总体目标是实现一个设计，它首先独立于硬件，出于明显的原因；其次独立于操作系统和数据库管理系统，再次同样出于明显的原因；或许有点争议的最后一条，独立于应用程序（换句话说，我们主要关心数据是什么，而不是它将如何使用）。在这个意义上独立于应用程序是可取的，它有一个很好的理由，因为在通常情况下（也许在所有时候），在设计时数据的所有用途并非都是已知的。因此，我们希望有一个设计，它将是健壮的，在这个意义上，它不会因为在原设计时没有预见到的应用需求的到来而失效。注意，这种状况的一个重要的后果是，我们没有（或至少不应该）让设计出于物理性能方面的原因妥协。设计理论不应该由性能方面的考虑驱动。

回到设计原理。正如我们将要看到的，理论包括一些正式的定理，为设计人员提供要遵循的实践指引的定理。所以，如果你是一个设计师，你需要熟悉这些定理。让我赶紧补充说，我不是说，你必须知道如何证明这些定理（虽然事实上它们的证明往往是很简单的），我的意思是，你必须知道这些定理的内容，即，你要知道结果，

你还必须准备应用这些结果。这是定理的好处：一旦有人已经证明了它们，它们的结果就可供任何人在任何需要它们的时候使用。

现在，有时人们声明，所有的设计理论确实会增强你的直觉，这个说法不是完全没道理的。我说这句话是什么意思？那么，考虑供应商和零件数据库。该数据库的一个明显的设计如图1-1所示，我的意思是说，这是“显而易见”的，有三个关系变量是必需的，属性STATUS（状态）属于关系变量S，属性COLOR（颜色）属于关系变量P，属性QTY（数量）属于关系变量SP，并以此类推。但是，这些东西为什么是明显的呢？假设我们尝试了一种不同的设计，例如，假设我们将属性STATUS从关系变量S中移出，并移入到关系变量SP中（直观上这就是错误的，因为状态是供应商的属性，而不是发货的属性）。图1-2显示了这个修订的发货关系变量的一个示例值，我把它称为STP，以避免混淆：[\[2\]](#)

STP	SNO	STATUS	PNO	QTY
	S1	20	P1	300
	S1	20	P2	200
	S1	20	P3	400
	S1	20	P4	200
	S1	20	P5	100
	S1	20	P6	100
	S2	30	P1	300
	S2	30	P2	400
	S3	30	P2	200
	S4	20	P2	200
	S4	20	P4	300
	S4	20	P5	400

图 1-2 关系变量STP的示例值

只需要大致浏览这幅图就足以看出这样的设计有什么错误：它是冗余的，因为供应商S1的每一个元组都告诉我们S1的状态是20，供应商S2的每个元组都告诉我们S2的状态是30，等等。[\[3\]](#)设计理论告诉我们，不要以明显会导致这种冗余的方式设计数据库，它也告诉我们（虽然是隐含地）这种冗余的后果将是什么。换句话说，正如我们所见的，设计理论的主要目的是减少冗余。（顺便说一句，我评论说，部分是由于这些原因，也许有点不近人情，理论已描述作为不良例

子的一个良好来源。)

现在，如果设计理论真的只是增强了你的直觉，那么它可能（而且确实已经）被批评的理由是，它确实不过都是些常识。通过举例的方式，再次考虑关系变量STP。正如我已经说过的，那个关系变量显然是糟糕的设计；冗余是显而易见的，造成的后果也是明显的，任何称职的人类设计师将“自然地”避免这样的设计，即使该设计师完全不具备明确的设计理论知识。但这里的“自然地”是什么意思呢？人类设计师在选择一个更“自然”（和更好）的设计时，应该应用什么样的原则呢？

答案是：它们正好是设计理论所涉及的原则（例如，规范化的原则）。换句话说，称职的设计师的大脑里实际上已经有这些原则，即使他们从来没有正式研究过它们，不能命名它们，或准确地表达它们。所以，原则是常识，但它们是形式化的常识。（常识可能是普通的，但说清楚它到底是什么并不总是容易的！）设计理论所做的是用精确的方式说明常识某些特定方面的组成。在我看来，理论真正的成就（或无论如何真正的成就之一）是：它形式化地说明了某些常识性原则，从而为机械化这些原则的可能性打开了门（即，将它们集成到计算机化的设计工具中）。

理论的批评者往往忽视了这一点，他们相当正义地声称，这些观点大多只是常识，但他们似乎并没有意识到，用一个精确且正式的方法表述常识的含义是一个重大的成就。

作为上述内容的一种后记，我注意到，常识可能并不总是那么普通的。下面从休斯飞机公司的罗伯特·R·布朗<sup>[4]</sup>的一篇论文中提取的略加编辑的内容说明了这一点。作者以“一个简化的真实例子”（他的原话）开始，此例涉及员工文件（包括员工编号、员工姓名、电话号码、部门编号和经理姓名字段）和部门文件（包括部门编号、部门名称、经理姓名和经理的电话号码字段），这些全都有直观的意义。然后，他继续说：

这个例子所基于的实际的数据库有更多文件和更多字段，以及更多的冗余。当设计师被问及这样设计的原因时，他列举了性能和做连接的困难。即使在我的例子中冗余对你而言应该是清楚的，它也并没有体现在设计文档中。在包含许多文件和许多字段的大型数据库中，若没有做大量的资料分析且没有与用户所在组织中的专家进行长时间的讨论，是不可能找到冗余的。

顺便说一句，还有一个我非常喜欢的引用，

其实，我用它作为《SQL与关系数据库理论》的题词——这支持了我的论点：实践者确实需要知道自己所在领域的理论基础。这是达·芬奇说的（并且有着500年之久的历史！），它是这样的（用黑体显示）：

没有理论武装的实践迷恋者正如领航员上了一艘没有舵或罗盘的船，并且他从来没有任何确定的航行目的地。实践应始终基于一种完善的理论知识。

[1]警告，但是，其他作家使用术语逻辑设计和物理设计指别的事物，并且使用其他术语表示我说的含义。注意辨别。

[2]出于明显的原因（因为S已经用作SNO的缩写），我使用T而不是S作为STATUS的缩写，整本书都这样用。

[3]你可能也会注意到另外一个问题：此设计不能很好地代表这样的供应商，如目前根本不提供零件的供应商S5。这样的“更新异常”将在第3章中讨论。

[4]Robert R.Brown: “Database Systems in Engineering: Key Problems, and Potential Solutions,” in the proceedings of a database symposium held in Sydney, Australia (November 15th-17th, 1984) .



## 1.6 本书的目的

如果你像我一样，会在文献和现场演示及类似的场合遇到过设计理论方面的很多术语，如投影连接范式、追逐、连接依赖、保持函数依赖

（FD）和许多其他的术语，我敢肯定，你不时想知道它们的确切含义都是什么。因此，本书目标之一是解释这些术语：仔细和准确地定义它们，解释它们的针对性和适用性，以及从总体上消除似乎围绕着它们的任何神秘气息。如果我成功达成这一目标，我就已经找到了一个很好的方式，去解释设计理论是什么样的以及为什么它是重要的（事实上，本书一个可能的替代书名是《数据库设计理论：它是什么以及为什么你应该关心它》）。总的来说，本书旨在为数据库专业人士提供一个关于设计理论的轻松介绍。更具体地说，本书旨在：

- 对你应该已经很熟悉的设计的各方面知识，从你可能不熟悉的角度来复习

- 深入剖析你可能不熟悉的方面

- （用许多例子）提供所有相关概念的清晰和准确的解释和定义



■不在已广为人知的内容（如2NF和3NF<sup>[1]</sup>）上花大量时间

说是这么说，我也应该说，数据库设计不是我最喜欢的科目。我之所以不太喜欢它是因为这个科目有许多部分还是有点.....当然，这只是我的一家之见。正如我刚才所说的，设计理论是数据库设计的科学基础。然而，可悲的是，还有许多设计问题是理论根本（仍然）不能解决的。因此，虽然本书描述的形式原则确实代表科学设计的一部分，但我在别处论及的其他内容本质上仍然属于一种艺术创作。事实上，本书传达的信息正是我们在这领域需要更多的科学。

为了增加一些乐观的元素，我想提醒你注意以下事项。设计理论是为了（至少部分地）捕捉数据的含义，并且像Codd自己曾在这一概念上所说的：<sup>[2]</sup>

以一个相当正式的方式捕捉.....比数据的含义更多的内容是一个永无止境的任务.....我们的目标仍然是极其重要的，因为即使是很小的成功也可以为数据库设计领域带来理解和秩序。

事实上，我会走得更远：如果你的设计违反了任何已知的科学，那么，如我在其他地方（在

一个稍微不同的情况下)所写的,你可以肯定的一件事是,它会出问题。虽然可能很难确切地说明什么会出问题,并且可能很难说是会以一个或大或小的方式出问题,但是你知道,它们将出问题,因为它们违背了理论。可见,理论是很重要的。

[1]不过,由于完整性的原因,我至少会给出那些已熟知的概念的精确定义。因为我敢肯定它们真的是为人熟知的,所以我会在定义它们之前时不时提到它们。

[2]这段引用来自Codd的论文“Extending the Database Relational Model to Capture More Meaning,”ACM TODS 4, No.4, 1979(斜体字是我的作品)。当然,Ted Codd是关系模型的发明者,他也是定义规范化的一般概念,特别是前三个范式(1NF、2NF、3NF)的第一人。

## 1.7 结束语

本书在写作中成长，事实证明，虽然上一节的一些话略有消极语气，但真的有不少的好材料需要涉及。更重要的是，这些材料在完善中。因此，虽然前几章可能会显得进展相当缓慢，我想你随后会发现步伐赶上来了。导致这一点的部分原因是需要引入一些术语和概念，这些观点不是真的很难，但它们似乎有点繁多，至少在你熟悉这些术语之前是这样的。出于这个原因，至少在这本书的某些部分，我将把材料介绍两次，首先从非正式的角度，然后从一个更正式的角度。

（伯特兰·罗素有一次令人回味地说：作品可以是可读的或精确的，但两者不能同时实现，我想努力使这两者兼得。）

用来自伯特兰·罗素的另一个引用[\[1\]](#)来结束本章似乎是适当的：

我曾被指控有改变自己观点的习惯……我自己对这个习惯没有感到丝毫的惭愧。在20世纪初已经活跃的物理学家会梦想夸口说，在过去的半个世纪中，他的观点并没有改变吗？……我很重视和努力追求的一种哲学是科学的，在这个意义上，有一些明确的知识要获得，并且新的发现可以使任何坦诚的人承认以前犯下的不可避免的错

误。对于我说过的话，无论是过去还是最近说的，我不要求它们是像神学家声称他们的教义那样的真理。我只要求，至少在当时我所表达的观点是明智的并值得保持，如果随后的研究并没有表明这些观点需要进行修改，我会感到非常惊讶的。[这些观点不是]作为教皇的宣告，而仅作为在促进清晰和准确的思维的时候，我可以做到最好的程度。最重要的是，准确一直是我的目标。

我在别处也引用了这个片断：特别是在我的书《An Introduction to Database Systems》[\[2\]](#)（第8版，Addison-Wesley出版社，2004年）的序言中。我提及后一本书的原因是，它包括对在本书中涉及的一些材料更深入的其他教学处理。但是，世界已经改变。我希望我自己对理论的理解比我写早期的书时更好，而且我会坦率地说现在我想修订那本书在某些方面的处理方式。早期那本书的处理方式的一个问题是，我试图通过对任何给定关系变量只采用一个键，从而可以无害地把它视为主键，使材料更易于理解。但是，简化的假设的结果是我给出的那几个定义（例如，2NF和3NF）不完全准确。这一事实导致了一定的混乱——部分是我的错，我坦率地承认，但部分也是采用脱离上下文定义的人们的错。

[\[1\]](#)这段引言来自《The Bertrand Russell Dictionary

of Mind, Matter and Morals》(ed., Lester E. Denonn; Citadel Press, 1993) 的前言。这里已经对它稍微作了修改。

[2] 中文版《数据库系统导论(原书第8版)》由机械工业出版社引进并出版。

## 习题

这些习题的目的是提供后续章节范围的一些知识点，也或许是测试你现有的知识水平。仅根据本章所学的知识，不能回答它们。

1.1 关系模型不要求关系变量属于任何特定的范式，这是否正确？

1.2 数据冗余是否总是应该消除？这可以做到吗？

1.3 3NF和BCNF之间的区别是什么？

1.4 每一个“全键”关系变量都满足BCNF，这是否正确？

1.5 每一个二元关系变量都满足4NF，这是否正确？

1.6 每一个“全键”关系变量都满足5NF，这是否正确？

1.7 每一个二元关系变量都满足5NF，这是否正确？

1.8 如果关系变量只有一个键和一个其他属

性，那么它满足5NF，这是否正确？

1.9 如果一个关系变量满足BCNF，但不满足5NF，那么它必须是全键，这是否正确？

1.10 你能给出5NF的一个精确定义吗？

1.11 如果关系变量满足5NF，那么它是没有冗余的，这是否正确？

1.12 反规范化的精确定义是什么？

1.13 什么是希思定理？为什么它很重要？

1.14 什么是正交设计的原理？

1.15 是什么让一些连接依赖（JD）不可消除而有些却可以？

1.16 什么是依赖保持？为什么它很重要？

1.17 什么是追逐？

1.18 你可以说出多少范式的名称？

## 第2章 预备知识

The world is everything that is the case.

——Ludwig Wittgenstein: Tractatus Logico-Philosophicus

假设你是一个数据库专业人士，我指的是这样一个人，他是一个数据库从业者并对关系理论有适度的熟悉程度。请注意，很抱歉，我不得不说这一点，但它是事实——不管有多深的SQL知识，这对于满足这一要求的后半部分都是不够的。正如我在《SQL与关系数据库理论》中写的：

我敢肯定，你知道一些关于SQL的知识，但——我在这里为可能冒犯的语气向你道歉——如果你关于关系模型的知识只来自你的SQL知识，那么恐怕你对关系模型的了解还不够深入，而且你知道的可能是一些似是而非的东西。我怎么下面这句话都不为过：SQL和关系模型是不一样的东西。

那么，本章旨在告诉你一些我希望你已经知道的事情。如果你确实知道，那么本章将作为一个复习；如果你不知道，我希望它足以充当一个



教程。更特别的是，我想要做的是详细说明关系理论的特定基本方面的一些细节，这也是阅读后续章节的基础。以我的经验，这里列举的这些方面的问题，数据库从业者往往是不知道的（至少是不明确的）。当然，我也会依靠关系理论的其他方面，但在我利用它们的时候，如果我认为有必要，我会详细说明那些内容。

## 2.1 概览

首先，我对刚才提到的关系理论的这些基本方面作一个简单的总结（主要是为后续参考）。

- 任何给定的数据库包含关系变量（英文简称为relvar）。

- 在任何给定的时间，任何给定的关系变量的值都是一个关系值（简称为关系）。

- 每个关系变量都代表一个特定的谓词。

- 在任何给定的关系变量中，每个元组都代表一个特定的命题。

- 按照封闭世界假设（The Closed World

Assumption,CWA), 关系变量R在时间T包含所有且只包含那些代表关系变量R对应的谓词在时间T计算结果为TRUE的实例元组。

接下来的两节(这很大程度上依赖于《SQL与关系数据库理论》中的内容)详尽阐述了这些想法。

## 2.2 关系及关系变量

再看看在第1章的图1-1中的供应商和零件数据库。该图显示了三个关系，即：在某个特定的时间正好存在于数据库中的关系。但是，如果我们在某个不同的时间观察数据库，我们可能会在它们的位置看到三个不同的关系出现。换句话说，S、P、SP其实是变量（精确地说是关系变量），就像一般的变量，它们在不同的时间有不同的值。特别是，因为它们是关系变量，所以它们在任何给定时间的值，当然就是关系值。

作为进一步研究这些概念的基础，考虑下面的图2-1。这个图的左边显示了来自图1-1的发货关系的一个非常精简的版本；右边显示的是，执行一定的更新后得出的关系。使用前一段列出的术语，那么，我们可以说，我们在图2-1左边看到的值，是关系变量SP在某个特定的时间T1的值；我们在图2-1右边看到的值，是相同的变量在某个大概更晚的时间T2，已经插入一个额外的元组后的值。

在时间T1的关系变量SP

SNO	PNO	QTY
S1	P1	300
S2	P1	300

在时间T2的关系变量SP

SNO	PNO	QTY
S1	P1	300
S2	P1	300
S1	P2	200

标题

正文

关系

谓词：供应商SNO供应数量为QTY的零件PNO

命题：供应商S1供应数量为300的零件P1（等）

图 2-1 关系值和变量的一个例子

所以在关系值和关系变量之间存在一个逻辑差异。麻烦的是，数据库社区历来使用相同的术语“关系”来充当两个概念的缩写，这种做法肯定会导致混乱，至少在涉及本书主题（如进一步规范化）的上下文中。因此，本书从此刻起将非常仔细地地区分两者——当我谈关系值时，我的意思是关系值，而当我谈关系变量时，我的意思是关系变量。不过，大部分时候，我也会把“关系值”缩写为“关系”（正如我们大部分时候把整数值（integer value）缩写为整数（integer））。例如，我要说的供应商和零件数据库包含三个关系

变量，（更精确地说，三个基本关系变量；视图也是关系变量，但在这本书中，关于视图我说得很少）。

旁白：其实，关于视图有一件事我确实想说。（视图和基本关系变量）的互换性原理指出，实际上，至少只要用户关注视图看上去应该就像基本关系变量。（我不是指定义为纯粹的速记视图，我是指目的是以某种方式将用户与“真正”的数据库隔离的视图，请参阅第15章关于这一点的阐述。）一般情况下，其实用户不与只包含基础关系变量的数据库（以下简称“真正”的数据库）交互，而是与其中包含基础关系变量和视图的某种混合的所谓的“用户数据库”交互。但是，对用户而言，用户数据库应该看起来和感觉就像真正的数据库，因此，本书讨论的所有设计原则（例如，规范化的原则）不仅在真正的数据库上应用得好，而且在这样的用户数据库上应用得同样好。出于这个原因，我会在整本书中随意使用不加限定的术语“关系变量”，依靠上下文来表示术语是否同等地指基础关系变量和视图，或者明确地仅指基础关系变量（或视图）。旁白结束。

让我们回到图2-1。这幅图表明，关系有两部分，一个“标题”（heading）和一个“正

文”(body)。基本上，标题是一组属性，正文是一组符合该标题的元组。例如，图2-1中所示的两个关系都有一个由三个属性组成的标题，并且，该图左侧的关系有一个由两个元组组成的正文，而在右侧有一个由三个元组组成的正文。因此，请注意，关系并没有真正地包含元组，至少不是直接地（它包含一个正文，而该正文又包含元组）。然而，在实践中，为了简单，我们确实通常这样谈论，好像关系直接包含元组。由此产生的要点如下所示。

■标题和正文这两个术语也以明显的方式延伸至关系变量。当然，关系变量的标题（像关系的标题）永远不会改变——它与所有可能的关系的标题相同，可能把这些可能的关系赋值给所涉及的关系变量。相比之下，正文会发生变化，具体而言，在所涉及的关系变量上的更新完成时它发生变化。

■当我在本书的第二部分进行更正规的处理时，我要（重新）把标题定义为属性名的集合。但是就目前而言，这两个定义之间的差异并不重要。

■在事实上，把一个标题定义为“属性名/类型名”对的一个集合（并且需要所有涉及的属性名

是不同的) 仍然会是更正确的。例如, 我要在整本书的例子中, 假设属性SNO、PNO的类型都是CHAR (任意长度的字符串) 而属性QTY的类型是INTEGER (整数)。[1] 当我谈论元组符合一些标题时, 我的意思是所涉及的元组中的每个属性值必须有一个相关类型的值。例如, 为了让一个元组符合关系变量SP的标题, 它必须具有属性SNO、PNO和QTY (并且没有其他的属性), 而且这些属性的值类型必须分别是CHAR、CHAR和INTEGER。(综上所述, 我现在也必须说类型对于关系设计理论的目的不是很重要。这就是为什么我在本书中随便地对什么是一个标题进行简化定义的原因。而且, 我也随便地在我的许多示例关系变量定义中只给出属性名, 甚至没有提及类型。)

■ 在一个给定标题中的属性数量是这个标题的度 (degree)。它也是任何具有该标题的关系或关系变量的度。同样, 在一个给定的正文中元组的数量是该正文的基数 (cardinality), 它还是任何具有该正文的关系或关系变量的基数。[2] 注意: 术语“度”也同时用于元组和键 (包括外键)。例如, 关系变量SP中的元组都与关系变量本身一样, 度为3, 关系变量的唯一键的度是2, 而在该关系变量中的两个外键, {SNO} 和

{PNO}，度均为1。

■度（如标题的度、关系的度等）可以是任何非负整数。度为1称为一元（unary）；度为2称为二元（binary）；度为3称为三元（ternary）；……，而更一般地，度为n称为n元（n-ary）。

[1]把QTY定义为NONNEGATIVE\_INTEGER（非负整数，带有明显的语义）类型会是更合适的，但支持这种类型的DBMS即使有也是极少的。当然，我们可以将它作为用户定义的类型引入，但我不想在本书中讨论用户定义的类型细节。

[2]我说“任何”具有该正文的关系，但实际上，当且仅当所涉及的正文为空时，两个不同的关系才可以有相同的正文。如果不是这种情况，那么正好有一个关系具有该正文（见第5章中关系的正式定义）。



## 2.3 谓词和命题

再次考虑发货关系变量S P。像所有关系变量一样，该关系变量应该代表现实世界中的某些部分。事实上，我可以更精确地表述：关系变量的标题代表一个特定的谓词（predicate），这意味着，它是与现实世界的某些部分有关的一种通用声明（它是通用的，因为它是参数化（parameterized）的，正如稍后将要解释的）。所涉及的谓词是很简单的：

供应商SNO供应数量为QTY的零件PNO。

这个谓词是对关系变量SP的预期解释（intended interpretation），换句话说，即其含义（meaning）。

旁白：也许我应该对我在本书中对术语“谓词”的使用方式多说几点。首先，你可能已经熟悉该术语，因为SQL广泛使用它来指布尔型或者真值表达式（它讨论比较谓词、IN谓词、EXISTS谓词等）。然而，尽管SQL中的这种用法不完全错误，但是它确实篡夺了一个很一般的术语——在数据库环境中一个极其重要的术语——并给它一个相当专门的意义，这就是为什么我自己倾向于不遵循这个用法的原因。

其次，为了准确，我应该解释，谓词真的不是如前所述的一个语句，相反，它是由该语句作出的断言。例如，关系变量S的谓词是它现在的样子，无论它用英语、西班牙语或者任何语言表示。然而，为了简单起见，我将假定遵循一个谓词本身确实也只是一个通常用自然语言表达的语句。注意：类似的言论也适用于命题（见下文）。

最后，虽然我现在已经解释了我认为的这个术语的含义，但你应该知道（尽管前一段这么说）即使在逻辑学家之间，对于一个谓词确切是指什么也似乎没有共识。特别是，一些作者把一个谓词作为一个本身没有任何意义的纯粹形式结构，并认为我的解释是与谓词不同的东西。我不想在这里对这些问题进行争论，作为进一步讨论，我推荐你阅读由C.J.Date和Hugh Darwen著（Trafford, 2010）的《Database Explorations: Essays on The Third Manifesto and Related Topics》中的文章“什么是谓词”。旁白结束。

不严格地说，你可以把谓词想象为一个真值函数（truth valued function）。像所有的函数一样，它有一组参数，当调用它时，它返回一个结果；并且（因为它的返回类型是真值）结果不是TRUE就是FALSE。例如，在前面刚刚显示的谓词

的例子中，参数是SNO、PNO和QTY（当然与关系变量的属性相对应），而它们所代表适用类型的值（在这个简单的例子，分别是CHAR、CHAR和INTEGER）。当调用该函数时（用逻辑学家的说法是，实例化该谓词）我们用实参替换形参。假设替换的实际参数分别为S1、P1和300。那么，我们就得到了下面的陈述。

供应商S1供应数量为300的零件P1。

其实这个陈述是一个命题（proposition），它是在逻辑上是计算结果无条件地不是TRUE就是FALSE的某个东西。下面是几个例子。

1.爱德华·阿比<sup>[1]</sup>，写了《扳手党》（The Monkey Wrench Gang）。

2.威廉·莎士比亚写了《扳手党》。

第一个命题是成立的，第二个则是错误的。不要陷入认为命题必须是真实的常见陷阱！但是，在这一刻我说的命题都应该是真实的，正如我现在解释的。

■首先，每个关系变量有一个关联的谓词，称为所涉及的关系变量的关系变量谓词。（所以

上述“供应商SNO供应数量为QTY的零件PNO”谓词是关系变量SP的关系变量谓词）。

■设关系变量R具有谓词P。那么在某个给定的时间T，出现在R中的每一个元组t都可以视为代表一个特定的命题p，它们是在时间T，通过用t的属性值作为参数调用（或实例化）P得出的。

■而且（非常重要！）我们按照惯例假设，以这种方式获得的每一个命题p的计算结果都为TRUE。

例如，考虑到图2-1左边所示的为关系变量SP的示例值，假定下列命题在时间T1的计算结果都为TRUE：

供应商S1供应数量为300的零件P1。

供应商S2供应数量为300的零件P1。

更重要的是，我们再进一步：如果在某个给定的时间T，在某个关系变量中一个特定元组有理由可能出现，但实际上没有出现，那么我们就有权假定在那个时间T，相应的命题是假的。例如，元组

---

（采用一个明显的速记符号）肯定是一个合理的SP元组，但在时间T1，它没有出现在关系变量SP中，我再次参照图2-1——所以我们有权假定在时间T1，下面的命题是不成立的：

供应商S1供应数量为200的零件P2。

（另一方面，这个命题在时间T2是成立的）。

总而言之：在任何给定的时间，一个给定的关系变量R包含所有且仅包含那些在所涉及的时间代表成立的命题（R的关系变量谓词的真实实例）的元组，或者，至少是我们在实践中总是假设成立的那些东西。换句话说，在实践中，我们采用所谓的封闭世界假设。而且，由于这种假设如此重要（它强调的只是当我们使用一个数据库时我们所做的一切，即使它很少明确地承认），所以为了记录在案，这里我想要详细说明它。

■定义n：设关系变量R具有谓词P。那么封闭世界假设（CWA）指出，（a）如果在时间T，元组t出现在R中，那么认为在时间T，与t相对应的P的实例p是成立的；相反地，（b）如果在时间

T, 元组t有理由可能出现在R中, 但它没有出现, 那么认为在时间T, 与t相对应的P的实例p是不成立的。换言之（尽管是不严格地）：在一个给定的时间, 当且仅当当时元组t满足R的谓词时, 它才出现在关系变量R中。

[\[1\]](#)Edward Abbey是美国作家。

## 2.4 更多的供应商和零件

现在，让我们回到供应商和零件数据库，使用在第1章中图1-1所示的示例值作为例子。这里现在利用称为Tutorial D的语言来表示在该数据库中的三个关系变量的定义：

---

```
VAR S BASE RELATION
{SNO CHAR,SNAME CHAR,STATUS INTEGER,CITY
CHAR}
KEY{SNO};
VAR P BASE RELATION
{PNO CHAR,PNAME CHAR,COLOR CHAR,WEIGHT
RATIONAL,CITY CHAR}
KEY{PNO};
VAR SP BASE RELATION
{SNO CHAR,PNO CHAR,QTY INTEGER}
KEY{SNO,PNO}
FOREIGN KEY{SNO}REFERENCES S
FOREIGN KEY{PNO}REFERENCES P;
```

---

如我所说，这些定义是利用称为Tutorial D的语言表示的。现在，我认为此语言是相当不言自明的，但是，如果需要的话，可以在

《Databases,Types,and the Relational Model: The Third Manifesto (3rd edition)》(C.J.Date和Hugh Darwen著, Addison-Wesley, 2006)一书中找到它的全面描述。[\[1\]](#)注意：顾名思义，这本书还介

绍并解释了《第三宣言》（The Third Manifesto），关系模型的一种精确但有些正式的定义和配套的类型理论（顺便说一下，包括一个全面的类型继承模型）。特别是，它使用名称D作为符合《第三宣言》所制定的原则的任何一种语言的通用名称。任何数量的不同的语言都可以有资格作为一个有效的D。然而，可悲的是，SQL不是其中之一，这就是为什么在本书中的例子用Tutorial D来表示，而不是用SQL表示的原因（在任何有区别的地方）。（当然，Tutorial D是一个有效的D，事实上，它显然是适合说明和讲述《第三宣言》的观点的工具）。

旁白：这是一个提及在本书中所使用的术语是建立在《宣言》的基础上的最佳时机。因此，它们确实不同于在一些设计理论文献中偶然发现的术语。例如，那些文献通常不谈论关系的标题，而是使用术语关系模式（relation schema）[\[2\]](#)。那些文献也没有谈论关系变量（relvar）；相反，它们把本书中指的赋值给一些关系变量的（关系）值称作相应的模式的一个实例（instance）。旁白结束。

返回关系变量的定义。如你所见，这些定义都包括一个键的说明，这意味着，曾经分配给那些关系变量的每一种可能关系都需要满足相应的



键约束。（回想一下第1章，每个关系变量确实具有至少一个键。）例如，可能曾经分配给关系变量S的每一个关系需要满足如下约束，即在该关系中的两个不同元组不能具有相同的SNO值。更重要的是，我要在整本书中假设，除非有明确的与此相反的陈述，在关系变量S中也存在下面的函数依赖（FD）：

---

$$\{CITY\} \rightarrow \{STATUS\}$$

---

你可以把这个FD非正式地读作“STATUS函数依赖于CITY”，或读作“CITY函数决定STATUS”，或者更简单地只说“CITY箭头（arrow）STATUS”。它的意思是，每一个可能会分配给关系变量S的关系，都需要满足如下约束条件，如果在此关系中的两个元组具有相同的CITY值，那么它们也必须具有相同的STATUS值。[\[3\]](#)观察图1-1给出的示例值关系变量S确实满足此约束。注意：虽然本书第二部分和第三部分将阐述大量有关FD的内容，但我敢肯定，你无论如何已经很熟悉这个基本思想了。

现在，正因为键规格说明用于声明键约束，所以我們也需要某种形式的语法，以便能够声明FD约束。Tutorial D并没有提供用于这一目的的具

体语法<sup>[4]</sup>（到目前为止，SQL同样也没有提供）。但它们确实可以用有点迂回的方式来表达，例如：

---

```
CONSTRAINT XCT
COUNT (S{CITY}) =COUNT (S{CITY,STATUS}) ;
```

---

说明：在Tutorial D中，形式为 $r\{A_1, \dots, A_n\}$ 的表达式表示关系 $r$ 在属性 $A_1, \dots, A_n$ 上的投影。如果关系变量 $S$ 的当前值是 $s$ （一个关系），那么，（a）表达式 $S\{CITY\}$ 表示 $s$ 在 $CITY$ 上的投影，（b）表达式 $S\{CITY,STATUS\}$ 表示 $s$ 在 $CITY$ 和 $STATUS$ 上的投影；及（c）整体约束（我随意将它命名为 $XCT$ ）要求这两个投影的基数（COUNT）相等。（要求这两项计数相等相当于要求保持所需的FD约束，如果这不是很明显，那么尝试用图1-1中的示例数据来解释它。）

旁白：如果你觉得在约束 $XCT$ 公式中的那些计数要求在某种程度上有点不雅，下面是另一种避免使用它们的写法：

---

```
CONSTRAINT XCT
WITH (CT: =S{CITY,STATUS}) :
AND (JOIN{CT,CT  RENAME{STATUS  AS  X}},
STATUS=X) ;
```

---

说明：首先，WITH规格说明

（“WITH（.....）：”）仅用于引入一个名称，CT，以后可在整体表达式中反复使用它，以避免写好几次它所代表的表达式。其次，Tutorial D的RENAME操作大体上是不言自明的（但无论如何，在附录D中的习题2.15答案中有它的定义）。再次，Tutorial D的表达式AND（rx,bx），其中，rx是一个关系表达式，bx是一个布尔表达式，它当且仅当bx表示的条件对关系中由rx表示的每一个元组的计算结果都为TRUE时返回TRUE。旁白结束。

尽管有上述复杂情况，但是假设在整本书中，FD都可以使用前面所描述的更简单的箭头符号来说明。类似的言论也适用于其他类型的依赖关系（尤其是连接依赖和多值依赖，它们分别在第9章与第12章中阐述）。

我会用一个小难题结束本章。假设唯一适用于供应商和零件数据库的约束是上述FD和指定的键（外键）约束，那么我们就可以说，关系变量S、P和SP分别属于第二范式、第五范式和第六范式。要了解这些言论的意义，请继续往下看！

[1]自从这本书第一次出版后，实际上Tutorial D已

在某种程度地修改和扩展了。修订版（这是我将在这本书中使用的版本）的描述可以在

《Database Explorations: Essays on The Third Manifesto and Related Topics》（C.J.Date和Hugh Darwen著（Trafford, 2010））和网站

[www.thethirdmanifesto.com](http://www.thethirdmanifesto.com)上找到（顾名思义，该网站还包含许多现行的有关《第三宣言》的信息）。

[2]我一定不能给人留下标题和（关系的）模式完全是一样的东西的印象。相反，一个模式是一个标题和一定的依赖关系（dependency）（尤其是，包括但不限于局限于在本书后面详细讨论的函数依赖和连接依赖）的组合。

[3]这个关于FD的含义是什么的例子也足以说明为什么这种依赖关系称作函数依赖。详细说明：数学中的函数是从一个集合A到某个集合B的一个映射，其中集合A与集合B不一定是不同的，这个映射的属性是A中每一个元素只能映射到B中的一个元素（但在A中可以有任意数量的不同元素映射到B中的相同的元素）。因此，在这个例子中，我们可以说，从S中的CITY值集合到S中的STATUS值集合存在一个映射，而此映射确实是一个数学函数。

[4]它没有提供此语法的原因之一是，如果遵循本书所讨论的设计建议，无论如何应该很少需要显

式地声明函数依赖。

## 习题

这些习题的目的是测试你的关系理论知识。它们中的大多数不能仅用本章的内容来回答。然而，这里提到的一切以及附录D中这些习题的答案，在《SQL与关系数据库理论》中都有详细讨论。

2.1 什么是信息原则（Information Principle）？

2.2 下面的语句中哪些是正确的？

a.关系（关系变量也一样）的元组间没有顺序。

b.关系（关系变量也一样）的属性间没有顺序。

c.关系（关系变量也一样）永远不会有任何未命名的属性。

d.关系（关系变量也一样）永远不会有两个或两个以上同名的属性。

e.关系（关系变量也一样）永远不会包含重复元组。

f.关系（关系变量也一样）永远不会包含空值。

g.关系（关系变量也一样）总是属于1NF的。

h.用于关系属性定义的类型可以是任意复杂的。

i.关系（关系变量也一样）自己有类型。

2.3 下面哪些陈述是正确的？

a.标题的每个子集也是标题。

b.正文的每个子集也是正文。

c.元组的每个子集也是元组。

2.4 关系理论的文章中通常会出现术语域（domain），但本章正文并没有提及它。你怎么看这个事实？

2.5 定义命题和谓词。请举例说明。

2.6 说明供应商和零件数据库中关系变量S、P、SP的谓词。

2.7 设DB是你正好熟悉的任何数据库，并设R是在DB中的任何关系变量。R的谓词是什么呢？注意：这个习题的要点是让你把本章正文讨论的一些想法应用到你自己的数据上，试图让你用这些术语思考一般数据。显然，这个习题有的正确答案不唯一。

2.8 用你自己的方式解释“封闭世界假设”。可能存在“开放世界假设”（The Open World Assumption）这样的东西吗？

2.9 尽可能精确地定义元组和关系这两个术语。

2.10 尽可能精确地说明（a）两个元组相等和（b）两个关系相等的含义。

2.11 元组就是集合（分量集合），那么你是否认为定义通常的集合操作（求并集、交集等）适用于元组的版本是有意义的？

2.12 再说一遍，元组就是集合。但空集是合法的集合，因此，我们可以把相关的组件集为空的元组定义为空元组（empty tuple）。它内在的含义是什么？你能想到这样的元组的任何用途吗？



2.13 键是属性集合，而空集是合法的集合，因此，我们可以把相关的属性为空的键定义为空键（empty key）。它内在的含义是什么？你能想到这样的键的任何用途吗？

2.14 一个谓词有一个参数集合，而空集是合法的集合，因此，谓词可以有一个为空的参数集合。它内在的含义是什么？

2.15 规范化原则大量使用关系操作符投影和连接。尽可能精确地给出这两种操作的定义。此外，也尝试给出属性重命名操作符（Tutorial D 中的RENAME）的定义。

2.16 关系代数的操作符形成一个封闭的系统。你是怎样理解这句话的？

## 第二部分 函数依赖、BOYCE/CODD 范式及相关事宜

尽管范式不是设计理论的全部，但不可否认，它们占这个理论的一个非常大的部分，而且它们构成了本书第二和第三部分的主要议题。本部分最远谈到Boyce/Codd范式（BCNF），这是关于函数依赖（Functional Dependency,FD）的范式。

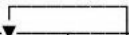
## 第3章 规范化：一些通则

Normal: see abnormal.

——from an early IBM PL/I reference manual

在本章中，我想在我们开始进入细节（我们将在第4章做这件事）之前，先对进一步规范化的某些一般方面进行澄清。我将从仔细观察来自图1-1的关系变量S的示例值（为方便起见，在下面把它重新绘制为图3-1）开始。

S



SNO	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	30	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

图 3-1 供应商关系变量——示例值

现在回忆这个关系变量具有的函数依赖（FD）：

---

{CITY}→{STATUS}

（我用图3-1中的箭头表明了这一事实）。因此FD的存在，[11](#)这表明，此关系变量属于第二范式（2NF），但不属于第三范式（3NF）。因此，这个关系变量存在冗余，具体来说，即一般一个给定城市有一个给定状态的事实出现了很多次。而进一步规范化（further normalization）的原则（从此处开始，大部分时间我会把它缩写为只是规范化，前面不加限定词）因此会提议我们将此关系变量分解成两个度更小的关系变量SNC和CT，如图3-2所示（当然，那些关系变量显示的值是与图3-1显示的关系变量S的示例值对应的。）

SNC		
SNO	SNAME	CITY
S1	Smith	London
S2	Jones	Paris
S3	Blake	Paris
S4	Clark	London
S5	Adams	Athens

CT	
CITY	STATUS
Athens	30
London	20
Paris	30

图 3-2 关系变量SNC和CT——示例值

从这个例子中可以得出如下两点。

■首先，分解肯定消除了冗余，一个给定城市有一个给定状态的事实，现在只出现一次。

■其次，分解过程基本上是一个执行投影的过程——图3-2所示的每个关系都是如图3-1所示的关系的投影。[\[2\]](#)事实上，我们可以写出下面两个等式：

---

---

$$\begin{aligned} \text{SNC} &= \text{S}\{\text{SNO}, \text{SNAME}, \text{CITY}\} \\ \text{CT} &= \text{S}\{\text{CITY}, \text{STATUS}\} \end{aligned}$$

---

---

（回顾一下，第2章介绍的投影的Tutorial D的语法形式 $r\{A_1, \dots, A_n\}$ ，其中 $r$ 是某个关系表达式，而 $A_1, \dots, A_n$ 是属性名）。[\[3\]](#)

■再次，分解过程是无损的（nonloss，也称为lossless），即在分解的过程中不会丢失信息，因为在图3-1中所示的关系可以通过对图3-2中所示的关系进行连接（join）来重建：

---

---

$$\text{S} = \text{JOIN}\{\text{SNC}, \text{CT}\}$$

---

---

（再次是Tutorial D语法）。因此，我们可以

说，在图3-1中的关系和图3-2中的关系对是信息等价（information equivalent）的，或更精确地说，对于任何可以针对图3-1中的关系执行的查询，都有一个可以针对图3-2中的关系执行且产生相同结果的查询与它对应（反之亦然）。显然，分解的这种“无损性”（losslessness）是一个重要的属性，无论我们做什么方式的规范化，当我们做到这一点时，我们一定不会丢失任何信息。

■从前面的描述可知，正因为投影是分解操作（相对于按常规理解的规范化），所以连接是相应的重构（recomposition）操作。

## 3.1 规范化用于两个目的

到目前为止，我所介绍的是大家很熟悉的东西。但现在我要指出，如果你一直在认真阅读，你可能会合理地指责我要了一个小花招……具体而言，我已经考虑了无损的关系分解的含义，但我们应该谈论的规范化，不是一个分解关系的问题，而是一个分解关系变量的问题。

假设我们执行建议的分解，将关系变量S分解为关系变量SNC和CT。注意，现在我确实正在谈论关系变量而不是关系，但是，为了明确起见，让我们假设这些关系变量分别具有在图3-1和

图3-2中所示的示例值。再次为了明确起见，让我们专门地专注于关系变量CT。此关系变量确实是一个关系变量（我的意思是，它是一个变量），这样我们就可以更新它。例如（使用第2章介绍的元组速记符号），我们可能会插入元组：

---

---

('Rome', 10)

---

---

但执行该更新后，关系变量CT包含一个在关系变量S中没有对应元组的元组（到目前为止，它在关系变量SNC中也没有一个对应元组）。现在，这种可能性经常用来（实际上，Codd自己在他的第一篇关于规范化的论文，详见附录C，里就用了它）作为一个摆在首位的赞成做规范化的论据：规范化后的两个关系变量的设计能够表示原始的一个关系变量的设计不能表示的某些信息。（在当前这个例子中，它可以表示，目前没有供应商的那些城市的状态信息。）但同样的事实也意味着，这两个设计其实不是真正信息等价的，此外，关系变量CT也不完全是关系变量S的一个“投影”<sup>[4]</sup>——它包含这样的元组，这个元组既不是关系变量S中的任何元组的投影，也不是以其他方式派生自它们的。<sup>[5]</sup>或者说（也许还有更多的），CT也不是SNC和CT的连接的投影，所以，那个连接在一定意义上是“丢失信

息”的，具体地说，它丢失了罗马（Rome）的状态是10的信息。[\[6\]](#)

如果我们从关系变量SNC删除元组

---

（'S5', 'Adams', 'Athens'）

---

将出现类似的情况。执行该更新后，我们可以不严格地说，[\[7\]](#)关系变量S包含一个在关系变量SNC中没有对应元组的元组（虽然它确实在关系变量CT中有一个对应元组）。所以，这再次表明，这两个设计不是真正信息等价的，并且此时关系变量S不完全是关系变量SNC和CT的“连接”，因为它包含一个不与关系变量SNC中的任何元组相对应的元组。

因此，这两个设计其实不是信息等价的。但难道我刚才没有说分解的“无损性”是一个重要的性质吗？难道我们没有一般性的假设，如果设计B是通过对设计A规范化而产生的，那么设计B和设计A应该是信息等价的吗？这究竟是怎么回事呢？

为了回答这些问题，观察关系变量谓词是有帮助的。SNC的谓词是：



供应商SNO名为SNAME并位于城市CITY。

而CT的谓词是：

城市CITY具有状态STATUS。

现在假定即使某个城市没有供应商，这个城市也允许具有状态；换句话说，假设关系变量CT可以包含一个在关系变量SNC中没有对应元组的元组，例如（Rome，10）。<sup>[8]</sup>那么，只由关系变量S构成的设计是完全不正确的。这就是说，如果允许谓词城市CITY具有状态STATUS存在一个真实的实例而不需要——在同一时间对于同一个CITY值——谓词供应商SNO名为SNAME并位于城市CITY存在一个真实的实例，那么一个只由关系变量S构成的设计，不能如实地反映现实世界的情况（因为这样的设计不能表示一个没有供应商的城市的状态）。

同样，假定即使某个城市没有状态，也允许供应商位于这个城市，换句话说，假定允许关系变量SNC包含一个在关系变量CT中没有对应元组的元组，比如，（S6，Lopez,Madrid）。那么，关系变量S的设计同样是完全不正确的，因为这个设计需要其中每一个有供应商的城市必须有一定的状态。

这里再用另一种方式来看待上述论点。假设仅由关系变量S构成的设计也忠实地反映了现实世界中的情况。那么，关系变量SNC和CT应当服从以下完整性约束（“SNC中的每个城市都出现在CT中，反之亦然”）：

---

---

CONSTRAINT.....SNC{CITY}=CT{CITY};

---

---

但对于正在讨论的例子，这种约束明显是不满足需要的，其实这是我以后将要称为相等依赖（equality dependency）或EQD的一个例子。注意：为简单起见，正如你所见，我没有费心给该约束设置一个名称。事实上，从这个时候开始，在这本书中我的所有例子全都忽略这样的名字，除非有一些令人信服的理由不这样做。

概括起来，我们看到的规范化可以（并且应当）用来解决下面两种相当不同的问题：

1.它可以用来修复一个逻辑上不正确的设计，如在本节前面讨论过的例子。习题：应用来自1.5节的STP例子时会出现类似这个例子中的问题吗？（答案：是的，它们会。）

2.它可以用于在其他逻辑正确的设计中减少

冗余。（显然，一个设计并非必须存在上述意义上的逻辑错误，才呈现冗余。）

在实践中，往往因为没有明确区分这两种情况而出现很大的混乱。事实上，大多数文献集中在第2种情况，为明确起见，我自己会在下文中有任何区别的地方，假设是第2种情况，但请不要忽视第1种情况，它在实践中即使不比第2种情况更重要，至少也是与第2种情况同样重要的。

我应进一步指出的是，从严格意义上讲，术语投影和连接只适用于第2种情况。这是，因为在第1种情况中，正如我们已经看到的那样，“新”关系变量不一定是“旧”关系变量的投影，“旧”关系变量也不一定是“新”关系变量的连接（如果你看出了我的意思）。事实上，谈论关系变量（而不是关系）的投影和连接的意思是什么呢？我已经在别处阐述了：[\[9\]](#)

根据定义，投影、连接等操作是专门应用于关系值的。当然，尤其是，它们应用于的值正好是关系变量的当前值。因此，下面的说法显然是有道理的，例如，关系变量S在属性{CITY,STATUS}上的投影，表示在关系变量S的当前值（代表的）关系的那些属性上的投影结果

（代表）的关系。但是，在某些情况（例如，规范化）下，在一个稍微不同的意义上使用如“关系变量S在属性{CITY,STATUS}上的投影”的表达式其实是方便的。具体而言，我们可能会虽然不严密，但很方便地说，某个关系变量CT是关系变量S在属性{CITY,STATUS}上的投影，更确切地说，它表示在任何时候，关系变量CT的值都是关系变量S当时的值在那些属性上的投影。因此，在一定意义上，我们可以谈论关系变量本身的投影，而不是仅仅用关系变量的当前值的投影的方式来谈论。类似的情况也适用于所有的关系操作。

换句话说，即使是在第1种情况下，我们确实仍然使用“投影/连接”术语。虽然这样的说法是有些不适用的（而不是说草率的），但它至少是简洁的。但是，如果说分解并不是一个投影过程，而是一个联想过程，回想我们在投影（对重构和连接也同样）的时候所做事的过程，但两者不是完全相同的过程。这种说法真的是更准确的。

[1]为了准确地说明该问题，而且因为除了那些由唯一键{SNO}蕴含的FD外，没有其他的FD存在。请参阅第4章。

[2]其他类型的分解是可能的，但我会假定不局限

定的“分解”具体就是指通过投影分解，直至另行说明为止。

[3] Tutorial D还支持形式为 $R\{ALL\ BUT\ B_1, \dots, B_m\}$ 的语法，这表示 $r$ [译者注： $r$ 怀疑是 $R$ ]在除了 $B_1, \dots, B_m$ 之外所有属性上的投影。例如，在本例中对应于SNC的投影可以替代地表达为： $S\{ALL\ BUT\ STATUS\}$ 。

[4]关于我在这里将术语“投影”用引号括起来的原因，参见本节后面的解释。

[5]关于一个元组可能是另一个元组的投影的观点，参见附录D中习题2.11的答案。

[6]由于这个原因，如SNC和CT的连接有时也称为有损连接（lossy join）。然而，可能最好避免用这个术语，因为它也可以用来指另一种连接，例如， $S$ 在 $\{SNO, SNAME, STATUS\}$ 和 $\{CITY, STATUS\}$ 上的投影的连接，这种连接由于不同的原因而丢失信息。参见在第5章对后一个例子的讨论，也请参见习题3.2。

[7]实际上，假装关系变量 $S$ 、 $SNC$ 、 $CT$ 是共存的（仿佛彼此一起存在）。

[8]在这里，我采用了一个草率的规则，它从常规的文本中把真正应该用来括起字符串值的单引号省略了，所以我写 $(Rome, 10)$ ，而不是 $('Rome', 10)$ 。更重要的是，从此刻开始往后，我会一直遵守这个规则。

[9]例如，在《The Relational Database Dictionary,Extended Edition》（Apress，2008）中。

## 3.2 更新异常

当涉及规范化时，更新异常（update anomaly）的概念是经常提及的。现在，应该明确的是，任何形式的冗余总会导致异常，因为不严格地说，冗余意味着，某些信息被表示了两次，所以总是有两个表示不一致的可能性（即，如果一个更新，而另一个没有更新）。更特别的是，让我们考虑在关系变量S中存在下面FD的情况：

---

$$\{CITY\} \rightarrow \{STATUS\}$$

---

这个FD产生的冗余，即一个给定城市有一个给定状态出现多次的事实——已经讨论过了。它会导致如下异常（这些例子假定使用图3-1显示的关系变量S的示例值）。

■插入异常：除非在罗马（Rome）有一个供应商，否则我们不能插入罗马的状态是10的事实。

■删除异常：如果我们删除了在雅典（Athens）的唯一供应商，我们就失去了一个事实，即雅典的状态是30。

■修改异常：我们不能改变（“修改”）一个给定供应商的城市而不改变该供应商的状态（一般来说）。同样，我们也不可以修改一个给定供应商的状态而不对在相关城市的所有供应商进行相同的修改。

把关系变量S更换为两个“投影”关系变量SNC和CT解决了这些问题（到底是怎么解决的？）。此外，为了记录在案，请允许我说明关系变量S（如前所述）属于第二范式，但不属于第三范式，而关系变量SNC和CT都属于第三范式，事实上，它们也属于BCNF。在一般情况下，BCNF是由上面列出的各种异常所引起的问题的解决方案。



### 3.3 范式层次结构

正如你所知道的，存在很多不同的范式。图3-3是我们第一次给出的范式层次结构（但请立即注意，我将在本书第13章扩展这个层次结构）。注意：你可能会认为这个层次结构是倒过来的，因为它在底部显示了最高的范式并在顶部显示最低的范式。我不想争论这点，我只想说，这个图的显示方式更符合（在我看来）事实，例如，所有属于2NF的关系变量都属于1NF，但一些属于1NF的关系变量不属于2NF。这个图的详细阐述如下所示。

■有几种不同的范式：第一范式、第二范式、第三范式等。该图显示了6个这样的范式，但你可以看到，它们不是按照第一，……，第六（不完全）来标记的，在第三个和第四个范式之间有一个闯入者，BCNF。[图3-3](#)第4章将解释这个古怪的术语这样命名的原因，现在，让我仅说名字BCNF是Boyce/Codd范式的简写形式。注意：尽管BCNF例外，一般地使用术语第n范式来指不同程度的规范化是方便的，本书后面的章节会不时采取这种用法。

■在一般情况下，规范化的级别越高越好，从设计的角度来看，因为规范化的级别越高，防

止的冗余就越多，且发生更新异常的情况就减少。



BCNF 和 5NF 是重要的范式  
(至少直到另行说明之前)

图 3-3 范式层次结构 (I)

■除1NF之外的所有范式都是依据某些特定依赖（在此上下文中，只是完整性约束的另一个术语）定义的。主要的依赖有函数依赖（FD）和连接依赖（JD）。注意：在文献中交替使用从属（dependence）和依赖（dependency）。本书会坚持用依赖（dependency）。

■对前一点的简要阐述：FD是定义BCNF的基础，而JD是定义5NF的基础。正如图3-3所示，BCNF和5NF是最重要的范式（至少直至另行说明

之前)。

■关系变量可能属于第 $n$ 范式，但不属于第 $(n+1)$ 范式 ( $n=1, \dots, 4$ )。

■如果关系变量 $R$ 属于第 $(n+1)$ 范式，那么它肯定属于第 $n$ 范式 ( $n=1, \dots, 4$ )。换句话说，第五范式(5NF)蕴含着第四范式

(4NF)，等等。因此，例如，说关系变量 $R$ 属于BCNF并不排除 $R$ 也属于5NF的可能性。然而，因为在实践中，比如，“关系变量 $R$ 属于BCNF”指的是“ $R$ 属于BCNF，但不属于任何更高的范式”，这样的陈述的意思是常见的。所以请注意，本书不采用这种用法。

■如果关系变量 $R$ 属于第 $n$ 范式，但不属于第 $(n+1)$ 范式 ( $n=1, \dots, 4$ )，那么它总是可以通过投影的方式无损分解，所以，(a)那些投影通常情况下属于第 $(n+1)$ 范式并且(b)  $R$ 等于那些投影的连接。

■最后，特别是根据前一点，任何给定的关系变量 $R$ 总是可以分解成5NF的投影。换言之，5NF始终是可以实现的。

关于冗余的概念的注意事项：第1章指出，

设计理论主要旨在减少冗余，本章又反复提到这个概念，尤其是，本章指出，规范化的程度越高，能防止的冗余就越多。但是，要给冗余下一个准确的定义似乎是相当困难的，其实，比我在本书此处设想的要困难得多。出于这个原因，在这里我甚至不打算尝试定义它，我只是要假定，直至另行说明为止，我们至少可以在看到它时识别出它（但即使如此，这实际上仍然是一个非常大的假设）。第15章深入探讨冗余的概念。

[\[1\]](#)在BCNF和4NF之间也有一个缺口，这反映了一个事实，层次结构中的前四个范式和最后两个范式之间存在一种概念上的跳跃。参见本书第三部分。

### 3.4 规范化和约束

规范化还产生另外一个常常被忽视的问题。再次考虑把关系变量S分解为其在{SNO,SNAME,CITY}上的投影SNC和其在{CITY,STATUS}上的投影CT的例子。有三种情况需要考虑。

1.假设原来只由关系变量S组成的设计，至少在逻辑上是正确的（即，它只是存在冗余）。正如3.1节指出的，那么，两个投影之间存在一定的约束（即“相等依赖”）：

---

---

CONSTRAINT.....SNC{CITY}=CT{CITY};

---

---

（“每个出现在SNC的城市也出现在CT，反之亦然”）。

2.另外，就像我们先前做的假设，CT可能包含一个元组，如（Rome, 10），它在SNC中没有对应元组。此外，假设相反的情况是不可能出现的，即SNC永远不能包含一个在CT中没有对应元组的元组。在这种情况下，这两个投影（从SNC到CT）之间存在一个外键约束（foreign key constraint）：

---

---

3.第三种可能性（也许比前两个更不可能），CT和SNC可能都允许包含在另一个（投影）中没有对应元组的元组。例如，可能出现这种情况，比如，供应商S6，名为洛佩兹（Lopez），位于马德里（Madrid），但马德里没有状态。在这种情况下，一个完全合理的设计将涉及元组（S6, Lopez, Madrid）在SNC中出现，而在CT中没有出现马德里的一个元组的情况，显然，因此，这两个关系变量之间完全不存在涉及城市的约束（至少，为了说明这个例子，让我们同意不存在这样的约束）。

现在，稍微简化一些，我已经说过，一个属于第n个范式的关系变量R总是可以无损地分解成属于第(n+1)范式的投影。然而，正如上述讨论表明，这样的分解通常是指现在至少需要存在一个新的约束。让事情变得更糟的是，这里的约束是一个多关系变量（multirelvar）约束（即，它跨越两个关系变量，或可能超过两个）。所以这是一个权衡：我们是要分解的好处，还是要避免多关系变量约束？[\[1\]](#)

旁白：可以这样说，至少在这个SNC和CT例

子中，分解也意味着增加一个现在不必存在的约束：即， $FD\{CITY\} \rightarrow \{STATUS\}$ 。但这种说法也不是完全有效的，在这方面，分解做的工作是把约束从一个关系变量移动到另一个关系变量（实际是从关系变量S移动到关系变量CT，它是作为{CITY}是一个键的约束的副作用而存在的）。结束旁白。

现在，在这个正在讨论的简单例子中，几乎可以肯定做分解的好处比不这样做的好处多。但是，情况并不总是这样，实际上，在更复杂的情况下，分解还是不分解，会是一个相当棘手的问题。在后面，为了避免大量的重复文字，我会倾向于认为我们总是想要做分解，但请不要忘记有时可以有不这样做的有说服力的论据，尤其是在本书第三部分讨论的一些例子中，它们比现在这个例子要复杂得多。

**[1]**当然，如果必须要保持这种约束，那么应该做这件事的是系统而不是用户，但至少将定义约束，且用户必须要意识到它的存在。

## 3.5 结束语

我想通过解决我在本书中至今还没有讨论的一个问题来结束本章。这是一个术语问题。具体而言：究竟为什么1NF、2NF以及其他的东西称为范式？也就是说，为什么规范化称作规范化？

这些问题的答案来自数学（虽然想法蔓延到几个相关的学科，特别要包括计算学科，例如，考虑浮点数计算）。在数学中，我们经常会发现自己不得不面对一些巨大的，甚至可能是无限的，某种对象的集合：例如，所有矩阵的集合，所有有理数的集合，或跟我们的研究对象近一点的，所有关系的集合。在这种情况下，希望能为所涉及的对象找到一个规范形式（canonical form）的集合。这里是它的一个定义。

■定义：给定一个集合 $s_1$ ，再加上已定义的那个集合的元素之间等价的概念，当且仅当在等价的概念下， $s_1$ 的每一个元素 $x_1$ 都只相当于 $s_1$ 的子集 $s_2$ 的一个元素 $x_2$ 时， $s_2$ 是 $s_1$ 的一个规范形式的集合，（并且元素 $x_2$ 是元素 $x_1$ 的规范形式）。<sup>[1]</sup>各种适用于 $x_1$ 的“有趣”特性，也都适用于 $x_2$ ，因此，我们就可以只研究小集合 $s_2$ ，而不是大集合 $s_1$ ，以证明各种“有意义”的定理或结果。



举一个说明这个概念的简单例子，设 $s_1$ 为非负整数集合 $\{0, 1, 2, \dots\}$ ，并设两个这样的整数当且仅当它们除以5会得到相同的余数时才是等价的。那么，我们可以把 $s_2$ 定义为集合 $\{0, 1, 2, 3, 4\}$ 。作为应用在这个例子中的一个“有意义”的定理，设 $x_1, y_1, z_1$ 是 $s_1$ 中的任意三个元素（即，任何非负整数），并设它们在 $s_2$ 中的规范形式为 $x_2, y_2, z_2$ ；那么当且仅当 $y_2 * z_2$ 的积等价于 $x_2$ 时， $y_1 * z_1$ 的积等价于 $x_1$ 。

那么，范式（normal form）只是规范形式的另一种说法。因此，当我们在数据库上下文中谈论范式时，我们正在谈论的是数据的一种规范表示。这一点的清晰解释是：正如我们知道的，用关系来表示任何给定的数据集都可以采取许多不同的方式。当然，事实上，所有的这些方式必须是信息等价的，也就是说，信息等价是我们在这个特殊的上下文中要求的那种等价。然而，由于各种原因，这些（表示给定的信息的）方式中的一些方式是比较其他的方式更为首选的。而那些首选的方式当然是关系范式，也就是本书的很大一部分主题。

至于术语“规范化”（normalization），它仅仅是指把某个给定的对象映射到它的规范等价对象的一般过程。因此，特别是在数据库的上下文

中，它用来（如我们所知）指把某个给定的关系变量映射到一个关系变量集合的过程，集合中这些关系变量（a）当整体考虑时，是与原始关系变量信息等价的，但（b）其中每个关系变量都独立地属于某个首选的范式。

针对上述情况，我也许应该添加以下内容。据我所知，关于这个问题，在Codd的早期著作中，他自己从来没有提到过他引入术语“范式”或“规范化”的原因。但多年之后，他确实记录了自己的解释：[\[2\]](#)

记者：“规范化”这个词的来源是什么？

Codd：对我来说，把一些规则引入数据库设计中，这似乎是重要的。我把它叫做规范化，因为当时的总统尼克松说了很多关于与中国的关系正常化的话。[\[3\]](#)我想，如果他能把关系规范化，我也能。

[\[1\]](#)还需要 $s_2$ 的每个元素 $x_2$ 至少等于 $s_1$ 的一个元素 $x_1$ ，这是合理的。让我也提醒你注意以下言论，转述附录D中的习题2.3的答案：在整本书中，表达的形式“B是A的一个子集”必须理解为包括B和A相等的可能性。例如，集合 $\{x,y,z\}$ 是它本身的一个子集。当我要排除这种可能性时，我会明确

地谈论术语“真子集”（proper subset），例如，集合 $\{x,z\}$ 是集合 $\{x,y,z\}$ 的一个真子集。当然，任何集合都不是它本身的一个真子集。

[2]在“A Fireside Chat: Interview with Dr.Edgar F.Codd”（DBMS Magazine 6, No.13, December 1993）.

[3]英文单词“normalization”的中文含义包括“规范化”和“正常化”。

## 习题

3.1 考虑1.5节中的STP例子。请用这个例子给出可能产生更新异常的一些示例。也请给出一个适当的分解，并显示分解是如何避免这些异常现象的。

3.2 无损分解基于关系可以分解成投影的概念，这种分解通过把这些投影用连接重新结合在一起，以恢复原始关系的方式进行。事实上，如果关系 $r$ 的投影 $r_1$ 和 $r_2$ 使得在 $r$ 中的每一个属性至少在 $r_1$ 和 $r_2$ 中的一个中保持，那么连接 $r_1$ 和 $r_2$ 总是会产生 $r$ 的每一个元组。证明这一论断。（从这一事实可知，不是无损分解的问题不是出在连接丢失了元组，而是出在它产生额外的，或“虚假的”元组。因为我们一般有没有办法知道在连接中的哪些元组是假的，哪些是真的，所以分解已经丢失了信息。）

3.3 “规范化用于两个目的”。用你自己的话解释这句话。你认为这一点是广泛理解的吗？

## 第4章 函数依赖和BCNF（非正式的）

It is downright sinful to teach the abstract before the concrete.

——Z.A.Melzak: Companion to Concrete Mathematics

正如我们在前面的章节中看到的那样，Boyce/Codd范式（简称BCNF）是依据函数依赖定义的。事实上，BCNF是真正的关于函数依赖的范式（我们把讲解提前一会儿，就像5NF是真正的关于连接依赖的范式）。本章的总体目标是解释这个观点，正如本章标题所表明的，但是，在这个阶段，各种解释和相关定义都是（当然是故意的）有点非正式的。（虽然非正式，但并非不准确，我不会说任何蓄意的谎言。）第5章会对此材料提供一个更正规的处理。

### 4.1 第一范式

让我们从头开始：设关系 $r$ 具有属性 $A_1, \dots, A_n$ ，每个属性分别具有类型 $T_1, \dots, T_n$ 。那么，根据定义，如果元组 $t$ 出现在关系 $r$ 中，那么 $t$ 中的属性 $A_i$ 的值的类型是 $T_i$ （ $i=1, \dots, n$ ）。例如，如果 $r$ 是出货关系变量SP的当

前值（见第1章中的图1-1），那么在r中的每一个元组都具有一个类型为CHAR的SNO值，一个类型也是CHAR的PNO值，以及一个类型为INTEGER的QTY值。

现在，我可以给出第一范式的一个准确定义：[\[1\]](#)

■定义：设关系r具有属性A1, ....., An，每个属性的类型分别是T1, ....., Tn。那么当且仅当对于所有出现在r中的元组t,t中的属性Ai的值的类型都是Ti (i=1, ....., n) 时，r属于第一范式（1NF）。

换句话说，根据定义，每一个关系都属于1NF！再换句话说，1NF只是意味着，关系中的每个元组的每个属性正好包含一个相应类型的值。请特别注意，在1NF中，对这些属性的类型是没有限制的。[\[2\]](#)它们甚至可以是关系类型，即，具有关系值属性（Relation Valued Attribute,RVA）的关系是合法的（你听到这一点可能会感到惊讶，但这是真实的）。图4-1给出了这方面的一个例子。

SNO	PQ	
..	.....	
S2	PNO	QTY
	P1	300
	P2	400
S3	PNO	QTY
	P2	200
..	.....	

图 4-1 一个带有关值属性的关系

稍后，我还会说更多关于RVA的内容，但我首先需要转移话题，说明几个小的要点。首先，我需要定义规范化的关系的含义。

■定义：当且仅当关系r属于1NF时，它是规范化（normalized）的。

换句话说，规范化和第一范式的意思是完全一样的，所有规范化的关系都属于1NF，所有属于1NF的关系都是规范化的。造成这个有些奇怪

的状况的原因在于规范化是原来的（历史的）术语，直到人们开始谈论2NF和更高水平的规范化，需要一个术语来描述那种不属于这些更高范式之一的关系时，1NF这个术语才引入。当然，目前术语“规范化”经常用来表示一些更高的范式（通常特别是3NF）；事实上，我已经在前面的章节中使用了这种方式，你可能已经注意到。但是，从严格意义上讲，这种用法是草率和不正确的，而且最好避免这么用，除非不会造成混乱。

至于我的第二个“小的要点”：注意，本节直至现在的讨论（特别是定义）都是用关系而不是关系变量的方式描述的。但是，因为按照定义，可以赋值给关系变量的每一个关系都属于1NF，所以如果我们用明显的方式把1NF的概念也扩展应用到关系变量是没有坏处的——而且这样做也是理想的，因为（正如我们将看到的）所有其他范式都定义为适用于关系变量，而不是关系。事实上，可以这么认为，1NF采用关系而不是关系变量的方式来定义与以下事实有关，令人遗憾的是，在许多年之前明确地得出这样的区分（即，关系和关系变量之间的区别）。

回到RVA。我已经说过了，实际上，带有RVA的关系变量是合法的，但现在我需要补充的是，从设计的角度来看，至少，这样的关系变量



通常是（并不总是）要避免的。现在，这个事实并不意味着，你应该完全避免RVA（特别是，查询结果包括RVA是没有问题的），它只是意味着，我们不想把RVA带入到“数据库设计”中。在本书中我不想在这个问题上深入讨论很多细节，我只是想说，带有RVA的关系变量往往看起来非常像早期的非关系型的系统（如IMS）中的层次结构，[\[3\]](#)并且在层次结构中曾出现的所有的老问题，都会因此再次暴露出来。这里是其中一些问题的列表，以供参考。

■最根本的一点是，层次结构是不对称的，因此，虽然它们可能使某些任务“更容易”，但它们肯定使其他的任务更困难。

■作为前一点的具体说明，尤其是查询是不对称的，并且比其对称的等价查询更复杂。例如，考虑针对图4-1的关系表达查询“获取供应商S2供应的零件的零件编号”和“获取供应零件P2的供应商的供应商编号”会涉及什么。这些查询的自然语言版本是彼此对称的，但它们在SQL或Tutorial D或其他一些形式语言中的表达方式，毫无疑问是不对称的（留给读者练习）。

■类似的说法也适用于完整性约束。

■类似的说法也适用于更新，但问题更严重。

■在一般情况下，对于如何选择“最好”的层次结构，不存在指导方针。

■即使是“自然”的层次结构，如组织结构和物料清单结构，通常情况下，它们用非层次结构设计来表达仍然是最好的。

现在你可能会感到疑惑，如果按照定义所有关系变量都属于1NF，那么不属于1NF可能意味着什么。也许令人惊讶的是，这个问题确实有一个合理的答案。问题是，今天的商业数据库管理系统（DBMS）完全不能正确地支持关系变量（或关系），相反，它们支持一种结构，为方便起见，我会称之为表（table），虽然对于这个术语，我不一定是指把自己局限在特别是在SQL系统中找到的那种表。但表，而不是关系变量，可能确实不属于1NF。详细说明如下。

■定义：当且仅当一个表是某个关系变量的一个直接和忠实的表示时，它属于第一范式（1NF），等价地，这样的表是规范化的。

因此，当然由此产生的问题是：表是一个关

系变量的一个直接和忠实的表示是什么意思呢？有5个基本的要求，所有这些要求都是由在任何给定时间一个关系变量的值（当然）总是一个特定的关系这一事实直接导致的。

1.各行不存在从上到下的顺序。

2.各列不存在从左到右的顺序。

3.不存在重复的行。

4.每一个行和列的交叉点都正好包含一个适用类型的值，而不是包含其他东西。

5.所有列都是常规的（见下文）。

要求1~3是不言自明的，[\[4\]](#)但其他两个要求或许值得多做一点解释。下面是一个违反要求4的表的例子。

SNO	PNO
S2	P1 , P2
S3	P2
S4	P2 , P4 , P5

因为在PNO列的值不是单个零件编号，而是零件编号组（group）（S2组包含两个零件编号，S3组包含一个零件编号，S4组包含三个零件编号）。所以这与要求4发生冲突。

注意：在上述例子中，如果列PNO不是定义为CHAR类型，而是定义为某个用户定义的类型，或许也称为PNO类型，违反要求4的情况也许会更清楚。那么，在该列中的值不是该类型本身，而是相当于类型为“PNO组”，这可能会更加明显。这样的考虑为术语重复组（repeating group）指出了—一个相当准确的定义。

■定义：当且仅当列C的定义为类型T，但出现在该列中的值不是类型T的值，而是类型为T的值的组（换句话说，集合或列表或数组等。）时，它是重复组列，也称为多值属性（multivalued attribute）。

如果你仍然对重复组列和RVA之间的差异感到混淆，那么请再看一下图4-1。在该图中的RVA，即，属性PQ不是重复组列（关系不允许有重复的组！）。相反，它是一个属性，该属性的类型正好是一个特定的关系类型——具体来说，使用Tutorial D语法来描述，是关系类型：

---

而该属性的值，精确地说，是这种类型的关系。因此，这幅图中的关系确实遵守了1NF的定义。

顺便说一句，要求4也意味着禁止空值（null）（因为空值不是值）。

现在来看要求5（“所有列都是常规的”）：这个要求意味着，首先，每一列都有一个名称，这个名称在应用于所涉及的表的列名中是唯一的；其次，任何行都不允许包含任何额外的、超过要求4所规定的常规列值的内容。例如，不存在不能通过常规的列引用（即列名）访问，而只能通过特殊的操作访问的“隐藏”列，并且不存在对于行调用常规操作有非常规效果的列。因此，特别是，不存在除了常规的键值以外（不存在隐藏的行ID或“对象ID”，如遗憾地在今天的一些SQL产品中发现的）的标识符（identifier），并且不存在隐藏的时间戳（timestamp），如在文献中针对特定的“时态数据库”（temporal database）建议的。

总结：如果有违反以上5个要求的任何情况，那么所涉及的表就没有“直接和忠实地”表示

关系变量，并且后果是不堪设想的。尤其是，关系操作，如连接不再保证如预期的那样工作（如果你像我以为的那样熟悉SQL，那么你已经知道这一点了）。关系模型处理的是关系（更准确地说，意思是，关系值和关系变量），而且只处理关系。

[1]一位审校者指责我用这个定义篡改了历史。也许会被指控有罪，但我这么做确实有我的原因，具体来说，在我看来，这个概念的所有早期“定义”，要么过于含糊而无法使用，要么完全错误。有关它的进一步讨论，请参阅《SQL与数据库关系理论》。

[2]虽然关系模型确实没有限制。但在附录D中的练习2.2的答案的解释中提到，有两个小例外，我在这里只是稍微简化地对它们加以说明：首先，数据库中的任何关系都不能具有任何指针类型的属性；其次，如果关系 $r$ 是属于类型 $T$ 的，则 $r$ 的属性本身没有可以属于类型 $T$ 的（请思考这一点！）。然而，这些例外与1NF毫无关系。

[3]而且，关于这点也许有更多的较新的情况，如XML（见习题4.12）。

[4]虽然我顺便注意到，特别是要求2实际上表明SQL表从来不是规范化的，除非它们可能正好只有一列。然而，在《SQL与关系数据库理论》中建议的原则允许你，除此以外，不妨把这样的表

（大部分）看作是规范化的。

## 4.2 函数依赖

关于1NF就说这么多，现在我可以开始讨论一些关于更高的范式的问题。现在，我已经说过Boyce/Codd范式（BCNF）是依据函数依赖定义的，而事实上，第二范式（2NF）以及第三范式（3NF）同样是用这种方式定义的。那么，下面就是函数依赖的一个定义：

■定义：设X和Y是关系变量R的标题的子集，那么当且仅当只要R的两个元组具有一致的X，它们就具有一致的Y时，在R中存在函数依赖（FD）：

$$X \rightarrow Y$$

这里的X和Y分别是决定因素（determinant）和依赖因素（dependant），而FD整体可以读作“X函数决定Y”或“Y函数依赖于X”，或者更简单地，就读作“X箭头Y”。

例如，正如我们所知道的，在关系变量S中存在 $FD\{CITY\} \rightarrow \{STATUS\}$ 。顺便说一句，请注意表达式中的大括号，定义中的X和Y都是R的标题的子集，因此它们是（属性的）集合，即使（如在本例中）它们碰巧是单例集合（singleton



set)。由于同样的原因，X和Y的值是元组，即使（如在本例中）它们碰巧都是度为1的元组。

再来看另一个例子，在关系变量S中还存在  $FD\{SNO\} \rightarrow \{SNAME, STATUS\}$ ，因为对于那个关系变量，{SNO}是一个键，事实上，是唯一的键，并且总是存在“从键出来的箭头”（见4.3节）。注意：为了防止不是很明显，我用术语“从X出来的箭头”来表示存在某个Y，使得在相关的关系变量中存在  $FD\ X \rightarrow Y$ （其中X和Y是关系变量的标题的子集）。

现在，记住一个要点：如果在关系变量R中存在  $FD\ X \rightarrow Y$ ，那么对所有X的超集  $X^+$  和所有Y的子集  $Y^-$ ，在关系变量R中也存在  $FD\ X^+ \rightarrow Y^-$ （当然，只要  $X^+$  仍然是标题的一个子集）。换句话说，你可以随时往决定因素中添加属性或从依赖因素中减少属性，而你所得到的仍然是所涉及的关系变量中存在的一个FD。例如，下面是关系变量S中存在的另一种FD：

---

$$\{SNO, CITY\} \rightarrow \{STATUS\}$$

---

（我从  $FD\{SNO\} \rightarrow \{SNAME, STATUS\}$  开始，往决定因素中添加了CITY，并从依赖因素中删除

了SNAME。)

我还需要解释FD是平凡的代表什么意思。

■定义：当且仅当FD  $X \rightarrow Y$ 没有办法可以违反时，它是平凡（trivial）的。

例如，对任何具有属性STATUS和CITY的关系变量，下面的函数依赖都是平凡的：[\[1\]](#)

---

---

```
{CITY,STATUS} → {CITY}
{CITY,STATUS} → {STATUS}
{CITY} → {CITY}
{CITY} → {}
```

---

---

简要阐述（但为了简单起见，只考虑第一个例子）：如果两个元组具有相同的CITY和STATUS值，那么它们当然有相同的CITY值。事实上，这是显而易见的，当且仅当Y是X的一个子集时，FD  $X \rightarrow Y$ 是平凡的。现在，当我们在做数据库设计时，我们通常不用操心平凡的函数依赖，因为它们是平凡的，但是当我们想正式和精确地处理这些问题——特别是，当我们正试图开发设计的理论时，那么我们就需要考虑一切FD，包括平凡的和非平凡的。

[1]特别是关于这些例子中的最后一个，见习题4.10。

## 4.3 键的重新审视

第1章概括地讨论了键的概念，但现在是获得关于此概念更确切的一些知识并介绍更多术语的时候了。首先，这里记录的是术语“候选键”（candidate key）的一个确切定义。

■定义：设K是关系变量R的标题的一个子集，那么当且仅当它同时具有以下两个属性时，K是R的一个候选键（或简写为仅是键）。

1.唯一性（uniqueness）：任何有效的R值都不能包含两个具有相同的K值的不同元组。

2.不可约性（irreducibility）：K的任何真子集都不具有唯一性属性。

旁白：这是我们遇到的第一个涉及某种不可约性的定义，但我们在后续的章节中还会见到更多——正如我们会看到的，在整个设计理论领域，一种或另一种不可约性是无处不在的，并且是重要的。特别是对于键的不可约性，它之所以重要的原因之一（不是唯一原因）是，如果我们指定一个不是不可约的“键”，那么DBMS将不能够执行正确的唯一性约束。例如，假设我们告诉DBMS（骗它的！）{SNO,CITY}是一个键，并且

事实上是关系变量S唯一的键，DBMS就不能强制约束供应商的编号是“全局”唯一的。相反，它只能强制执行较弱的约束，即供应商的编号是“局部”唯一的，在这个意义上，它们在相关的城市内是唯一的。旁白结束。

因为这个概念是如此熟悉，[\[1\]](#)所以我不打算进一步讨论上述定义，但请注意观察接下来的几个定义是如何依赖于它的。

■定义：关系变量R的一个键属性（key attribute）是指R的一个属性，它至少是R的一个键的一部分。

■定义：关系变量R的一个非键属性（nonkey attribute）是指R的一个属性，它不属于R的任何键。[\[2\]](#)

例如，在关系变量SP中，SNO和PNO是键属性而QTY是非键属性。

■定义：当且仅当整个标题是一个键（在这种情况下，它一定是唯一的键）时，等价地说，当且仅当整个标题的任何真子集都不是一个键时，关系变量是“全键”（all key）。注意：如果关系变量是“全键”，那么它肯定有没有非键属

性，但反之则不成立，一个关系变量可能是这样的，它的所有属性都是键属性，但它并不是“全键”（正确吗？）。

■定义：设SK是关系变量R的标题的一个子集，那么当且仅当它具有以下属性时，SK是R的一个超键（superkey）。

1.唯一性：R的任何有效值都不能包含两个具有相同的SK值的不同元组。

更简洁地说，R的超键是R的标题的具有唯一性但不一定不可约的一个子集。换句话说，我们可以不严格地说，超键是一个键的一个超集（之所以说“不严格”，自然是因为所涉及的超集仍然必须是相关标题的一个子集）。因此，请注意，所有的键都是超键，但“大部分”超键不是键。注意：非键的超键有时称作真超键。

子键的概念的定义也很方便。

■定义：设SK是关系变量R的标题的一个子集，那么当且仅当它至少是R的一个键的一个子集时，SK是R的一个子键（subkey）。注意：非键的子键有时称作真子键（proper subkey）。

举例来说，考虑关系变量SP，它只有一个键，{SNO,PNO}。这个关系变量有：

a.两个超键：

---

---

{SNO,PNO}  
{SNO,PNO,QTY}

---

---

请注意，对于任何关系变量R，标题始终是一个超键。

b.4个子键：

---

---

{SNO,PNO}  
{SNO}  
{PNO}  
{}

---

---

需要注意的是，对于任何关系变量R，属性的空集始终是一个子键。

在结束此节前，请注意，如果H和SK分别是关系变量R的标题和超键，那么在R中一定存在  $FD \quad SK \rightarrow H$ ，等价地，对H的所有子集Y，在R中一定存在  $FD \quad SK \rightarrow Y$ 。其原因是，如果R的两个元组具有相同的SK值，那么实际上它们必须是同一

个元组，在这种情况下，它们显然必须具有相同的Y值。当然，所有这些说法适用于一种重要的特殊情况，即SK不仅是一个超键而且是一个键的情况。如前所述（当然是非常不严格的），总是有从键出来的箭头。事实上，我们现在可以做一个更一般的陈述：总是有从超键出来的箭头。

[1]但要注意，没有要求关系变量只有一个键的建议。恰恰相反，其实一个关系变量可以有任意数量的不同的键，这个数量只受一个东西的限制，即涉及的关系变量的度的一个逻辑后果。见习题4.9。

[2]作为一个有历史意义的注记，我备注，键属性和非键属性在Codd的原规范化论文中分别称为主属性与非主属性（见附录C）。



## 4.4 第二范式

我要引入一个新的概念，FD不可约性（注意，这是第二种不可约性），这样我才可以得到2NF、3NF和BCNF的定义。

■定义：当且仅当FD  $X \rightarrow Y$ 存在于R中而对于X的任何真子集 $X^-$ ， $X^- \rightarrow Y$ 不存在于R中时，FD  $X \rightarrow Y$ 对于关系变量R是不可约的（如果R是不言自明的，或只是不可约的（irreducible））。

例如，对于关系变量SP,FD{SNO,PNO} $\rightarrow$ {QTY}是不可约的。注意：这种不可约性有时也更明确地称为左不可约性（因为它其实是在我们正在谈论的FD的左侧），但这里为了方便起见，我选择省略那个“左”字。

那么，最后，你可以有理由认为，我可以对2NF作如下定义。

■定义：当且仅当对于关系变量R的每一个键K和R的每个非键属性A,FD  $K \rightarrow \{A\}$ （必须是在R中存在的）都是不可约的时，R属于第二范式（2NF）。

注意：下面的定义与上面刚刚给出的定义在

逻辑上是等同的（见本章末尾的习题4.4），但有时可能会更加有用：

■定义：当且仅当对于关系变量R存在的每一个平凡的FD  $X \rightarrow Y$ ，以下三点至少有一点是正确的：（a）X是一个超键；（b）Y是一个子键；（c）X不是一个子键；R属于第二范式（2NF）。

由此产生的要点如下所示。

在设计过程中把2NF作为最终目标，将是极不寻常的。事实上，2NF和3NF主要具有历史意义，认为它们是通向BCNF的最好垫脚石，后者更加具有务实意义（也具有理论意义）。

2NF的定义在文献中经常采取的形式是“R属于2NF当且仅当它属于1NF且……”不过，这样的定义通常是基于对1NF的含义的错误理解。正如我们所看到的，所有关系变量都属于1NF，因此，“它属于1NF且”的话实际上没有添加什么限制。

让我们来看一个例子。其实，对于范式，通常看反例比看例子本身更具启发性。因此，考虑关系变量SP修订后的版本——让我们把它称作

SCP，它有一个额外的属性CITY，代表适用的供应商的城市。下面是一些示例元组：

SNO	CITY	PNO	QTY
S1	London	P1	300
S1	London	P2	200
S1	London	P3	400
..	.....	..	...
S2	Paris	P1	300
S2	Paris	P2	400
..	.....	..	...

此关系变量显然存在冗余：供应商S 1的每一个元组都告诉我们，S 1是在伦敦（London），供应商S2的每个元组都告诉我们，S2是在巴黎

（Paris），等等。此关系变量不属于2NF，其唯一的键是{SNO,PNO}，并且 $FD\{SNO,PNO\} \rightarrow \{CITY\}$ ，因此一定成立，但是，此FD并不是不可约的：我们可以从决定因素中删除PNO，这样保留下来的 $\{SNO\} \rightarrow \{CITY\}$ ，仍是在关系变量中存在的FD。等价地，我们可以说 $FD\{SNO\} \rightarrow \{CITY\}$ 存在并且是非平凡的，而且，（a）{SNO}不是一个超键，（b）{CITY}不是一个子键，且（c）{SNO}是一个子键等——那么根据上面给出的第二种定义，此关系变量不属于

2NF。

## 4.5 第三范式

■定义：当且仅当对关系变量R存在的每一个非平凡的FD  $X \rightarrow Y$ ，要么（a）X是一个超键，要么（b）Y是一个子键时，R属于第三范式（3NF）。

这个定义产生的要点如下所示。

■重复上一节的论述（也许与流行的观点相反），3NF主要具有历史意义——它应该充其量不过是通往BCNF的道路上的垫脚石。注意：我之所以这么说“也许与流行的观点相反”，是因为许多常见的3NF“定义”（至少在流行的文献中常见的）实际上是BCNF的定义——而BCNF，正如我已经指出的，是很重要的。需要注意辨别。

■3NF在文献中的定义通常采取的形式是“R属于3NF，当且仅当它属于2NF且.....”，我更喜欢一个没有提及2NF的定义。但是，请注意，我定义的3NF其实可以派生自我给2NF的第二个定义，通过删除条件（c）（“X不是一个子键”）即可得到。因此3NF蕴含着2NF，也就是说，如果一个关系变量属于3NF，那么它肯定属于2NF。

我们已经看到了一个属于2NF，但不属于

3NF的关系变量的例子，即，供应商关系变量S（参见第3章中的图3-1）。详细说明：正如我们知道的，因为此关系变量存在非平凡的FD{CITY}→{STATUS}；此外，{CITY}不是一个超键，且{STATUS}不是一个子键，所以此关系变量不属于3NF。（但它肯定属于2NF，习题：确认这种说法！）

## 4.6 Boyce/Codd范式

正如我刚才所说，Boyce/Codd范式（BCNF）是关于FD的范式，但现在我可以精确地定义它。

■定义：当且仅当对于关系变量R中存在的每一个非平凡的FD  $X \rightarrow Y$ , X都是一个超键时，R属于Boyce/Codd范式（BCNF）。

这个定义产生的要点如下所示。

■从它的定义可以得出，一个属于BCNF的关系变量中只能存在如下函数依赖，要么它是平凡的（很明显，我们不能摆脱这些），要么是从超键出来的箭头（我们也不能摆脱这些）。或许有些人喜欢说：每个事实都是关于键、关于整个键并且只关于键的事实，但我必须立即补充，这种非正式的描述，虽然它直观上是有吸引力的，但它不是真正准确的，因为它假定了其他的东西，即假定只存在一个键。

■此定义没有提及2NF或3NF。然而，请注意，该定义可以由3NF定义删除（b）（“Y是一个子键”）得到。因此BCNF蕴含着3NF，也就是说，如果某个关系变量属于BCNF，那么它肯定

属于3NF。

作为一个属于3NF但不属于BCNF的关系变量的例子，考虑一个出货关系变量的修订版，让我们把它称作SNP，它有一个额外的属性SNAME，表示适用的供应商的名称。假设供应商名称一定是唯一的（即，在相同的时间没有两家供应商拥有相同的名称）。那么下面是一些示例元组。

SNP	SNO	SNAME	PNO	QTY
	S1	Smith	P1	300
	S1	Smith	P2	200
	S1	Smith	P3	400
	..	.....	..	...
	S2	Jones	P1	300
	S2	Jones	P2	400
	..	.....	..	...

我们再次观察到一些冗余：供应商S1的每一个元组都告诉我们S1的名称是史密斯（Smith），供应商S2的每个元组都告诉我们S2的名称是琼斯（Jones），等等；同样，史密斯的每一个元组都告诉我们史密斯的供应商编号是S1，琼斯的每个元组都告诉我们琼斯的供应商编号是S2，等等。并且此关系变量不属于BCNF。首先，它有两个键，{SNO,PNO}和{SNAME,PNO}。[\[1\]](#)其次，标



题的每个子集，尤其是{QTY}（当然），都函数依赖于这两个键。再次，然而，在此关系变量中也存在 $FD\{SNO\} \rightarrow \{SNAME\}$ 和 $\{SNAME\} \rightarrow \{SNO\}$ ，因为这些FD肯定不是平凡的，它们也不是从超键出来的箭头，所以此关系变量不属于BCNF（虽然它属于3NF）。

最后，我敢肯定你知道，规范化的原则说：如果关系变量R不属于BCNF，那么将它分解成属于BCNF的投影。在关系变量SNP这个例子中，以下分解中的任何一个都将满足这个目标。

■在{SNO,SNAME}和{SNO,PNO,QTY}上投影。

■在{SNO,SNAME}和{SNAME,PNO,QTY}上投影。

顺便说一下，我现在可以解释为什么BCNF是另类的，因为它没有一个形式为“第n范式”的名称。如下内容是我从Codd首次描述了新的范式的论文中引用的：[\[2\]](#)

最近，Boyce和Codd制定了以下定义：一个[关系变量]R属于第三范式，如果它属于第一范式并且对R的每一个属性集合C，如果有任何不属于

C的属性函数依赖于C，那么R中的所有属性都函数依赖于C[即，C是一个超键]。

因此，Codd在这里给出了他认为是“新的和改进的”第三范式的定义。麻烦的是，新定义不仅过去而且现在确实比旧定义强，也就是说，任何属于新定义的3NF的关系变量肯定是属于旧定义的3NF的，但反之是不成立的——关系变量可以属于旧定义的3NF，而不属于新定义的3NF（上面所讨论的关系变量SNP，是一个很好的例子）。所以，“新的和改进的”定义实际上是一个新的且更强的范式，因此需要其自己的一个独特的名称。然而，在充分认识到这一点的时候，费金（Fagin）已经定义了他所称的第四范式，这个名字已不可用了。<sup>[3]</sup>因此，它只能用特殊的名称“Boyce/Codd范式”。

<sup>[1]</sup>这就是为什么我在展示示例元组时，没有显示任何双下划线的原因，因为它有两个候选键，似乎没有任何好的理由，让其中之一“比另一个更平等”。

<sup>[2]</sup>E.F.Codd: “Recent Investigations into Relational Data Base Systems,” Proc. IFIP Congress, Stockholm, Sweden (1974) .

<sup>[3]</sup>实际上，当雷蒙德（Raymond Boyce）首次想出了那个成为“新的和改进的”范式时，他确实称

之为第四范式！（他第一次描述此概念的论文发表在IBM Technical Disclosure Bulletin 16, No.1（June 1973），该论文的标题是“Fourth Normal Form and its Associated Decomposition Algorithm”。）我不知道为什么这个名字后来被拒绝了（虽然我有我的怀疑）。

## 习题

4.1 关系变量SP中存在多少FD？其中哪些是平凡的？哪些是不可约的？

4.2 FD这个概念依赖于元组相等的概念，这是真的吗？

4.3 从你自己的工作环境给出下面的例子。  
(a) 一个不属于2NF的关系变量； (b) 一个不属于3NF但属于2NF的关系变量； (c) 一个不属于BCNF但属于3NF的关系变量。

4.4 请证明在本章正文中给出的两个2NF定义在逻辑上是等价的。

4.5 如果某个关系变量不属于2NF，那么它一定有一个复合键，这正确吗？

4.6 每个二元关系变量都属于BCNF，这正确吗？

4.7 （与习题1.4相同），每一个“全键”关系变量都属于BCNF，这正确吗？

4.8 编写Tutorial D CONSTRAINT（约束）的语句来表示在关系变量SNP中存在FD{SNO}

$\rightarrow \{\text{SNAME}\}$ 和 $\{\text{SNAME}\} \rightarrow \{\text{SNO}\}$ （见4.6节）。  
注意：这是所有章节中的第一个请你用Tutorial D语言给出答案的习题。当然我知道你可能并不完全熟悉这种语言，因此，在所有这样的习题中，例如，下面的习题4.14和4.15，请只管尽你所能去做。我确实认为你至少值得在所涉及的习题中尝试它。

4.9 设R是一个度为n的关系变量。R中有可能存在的FD（平凡的以及非平凡的）的最大数目是多少？R中可能存在的键的最大数目又是多少？

4.10 假定FD  $X \rightarrow Y$ 中的X和Y都是属性集合，如果其中一个是空集，会发生什么情况？

4.11 你能想到，有一个带有RVA的基本关系变量，它确实是合理的情况是怎样的？

4.12 在写作《XML Databases》的时候，业界对此有很多讨论。但是，XML文档的性质本质上是层次结构的，所以你觉得在本章正文中对层次结构的批评适用于XML数据库吗？（是的，如我在本章前面的脚注中指出的，它们适用。那么你的结论是什么？）

4.13 第1章提到，在关系的表格图片中，我以双下划线表示主键的属性。然而，在那时，我没有正式地讨论关系与关系变量之间的差异；现在我们知道，一般来说键适用于关系变量，而不是关系。然而，从那以后我们已经看到了几个代表如前所述的关系的表格图片（我的意思是，不只是某个关系变量的一个示例值的关系）——例如，图4-1为三个例子<sup>[1]</sup>——并且在那些图片中我肯定使用了双下划线惯例。那么，关于该惯例我们现在可以说什么呢？

4.14 （再次摘自本章正文）请用Tutorial D描述下面对图4-1所示的关系的查询：

- a. 获取供应商S2供应的零件的零件编号。
- b. 获取供应零件P2的供应商的供应商编号。

4.15 假设我们需要更新数据库，以显示供应商S2供应数量为500的零件P5。请分别对（a）图1-1所示的非RVA设计，（b）图4-1所示的RVA设计给出Tutorial D描述所需的更新。

4.16 这里有一些来自技术文献的1NF定义。鉴于本章正文给出的解释，你对它们有什么评论？

■第一范式（1NF）.....表明属性的域必须只包括原子值（简单的，不可分割的），并且一个元组中任何属性的值必须是一个来自该属性域的单一个值（single value）.....1NF不允许一组值、一个元组的值，或两者的组合作为一个单独的元组（single tuple）的属性值.....1NF不允许“关系内部的关系”或“作为元组内的属性值的关系”.....1NF所允许的唯一属性值是单个原子（不可分割的）值（Ramez Elmasri and Shamkant B.Navathe,Fundamentals of Database Systems, 4th edition,Addison-Wesley, 2004）

■如果某个关系的每个字段都只包含原子值，即，没有列表或集合，那么它属于第一范式（Raghu Ramakrishnan and Johannes Gehrke,Database Management Systems, 3rd edition,McGraw-Hill, 2003）。

■第一范式就是下面这种情况，每一个元组的每一个分量都是一个原子值（Hector Garcia-Molina,Jeffrey D.Ullman,and Jennifer Widom,Database Systems: The Complete Book,Prentice Hall, 2002）。[\[2\]](#)

■如果认为某个域的每个元素都是不可分割的单元，那么此域是原子（atomic）的.....我们

说，如果在一个关系模式R中的所有属性的域都是原子的，那么R属于第一范式（1NF）

（Abraham Silberschatz, Henry F. Korth, and S. Sudarshan, Database System Concepts, 4th edition, McGraw-Hill, 2002）。[3]

■当且仅当一个关系满足只包含标量值的条件时，它称为属于第一范式（简称1NF）

（C.J. Date, An Introduction to Database Systems, 6th edition, Addison-Wesley, 1995）。

[1] 是的，我确实是说3个例子。

[2] 中文版《数据库系统全书》由机械工业出版社引进并出版。

[3] 中文版《数据库系统概念》由机械工业出版社引进并出版。



## 第5章 函数依赖和BCNF（正式的）

What's formal is normal.

What's not so is not.

And if normal is formal,

Informal is what?

——Anon: Where Bugs Go

现在，我想回退一步，不妨深吸一口气，并再次考虑FD和BCNF，但这一次，我想正式地做这件事（我为涉及少量的重复道歉）。你很快就会发现，本章的处理方法比第4章更加抽象，如果你完全学会了第4章的内容，那么理解本章是不难的，但本章内容肯定会是更加正式的。出于这个原因，除非你已经掌握了第4章的所有内容，否则我根本不希望你看本章。（当然，这应该不难做到，因为你对第4章的大部分内容无论如何是肯定熟悉的。）

预先说明一个普遍观点：既然BCNF是关于FD的范式，本章就不会论及2NF或3NF（甚至1NF）。因为我已经大致地说过，2NF和3NF本身

是没什么意义的。

## 5.1 初步定义

在本节中，我只是给出几个熟悉的但绝对重要的概念的**定义**，但很少有进一步的阐述，这里的定义比通常在文献中找到的定义更精确（在某种程度上，也比在本书中前面的那些定义更精确）。我把用来说明定义的例子留作一个习题。

■**定义**：一个标题 $H$ 是一个属性名集合。注意：这里特意把这个定义写得与第2章给出的定义不完全一样，请参阅第2章。此外，由于在这里并不重要的原因，《第三宣言》不使用 $H$ 而使用 $\{H\}$ 来表示一个标题。但是，对于本书的目的， $H$ 是更简单的形式，是更方便的。

■**定义**：一个具有标题 $H$ 的元组是一个有序对 $\langle A, v \rangle$ （在 $H$ 中出现的每个属性名 $A$ 都有这样的一对）的集合，其中 $v$ 是一个值。注意：如果 $H$ 的意义是众所周知的或与当前的目的不相关的，那么具有标题 $H$ 的元组这句话可以缩写为仅元组。

■**定义**：设 $t$ 是一个具有标题 $H$ 的元组并设 $X$ 是 $H$ 的一个子集，那么 $t$ 在 $X$ 的属性上的（元组）投影 $t\{X\}$ 是一个带有标题 $X$ 的元组——即，仅包含

那些出现在 $X$ 中的 $\langle A, v \rangle$ 对的 $t$ 的子集。[1]注意：这里定义的通常关系投影操作符的版本适用于单个元组（见习题2.11）。注意：一个元组的每一个投影本身就是一个元组。

■定义：一个关系 $r$ 是一个有序对 $\langle H, h \rangle$ ，其中 $h$ 是由所有具有标题 $H$ 的元组（ $r$ 的正文）组成的一个集合。 $H$ 是 $r$ 的标题，而 $H$ 的属性是 $r$ 的属性。 $h$ 的元组是 $r$ 的元组。

■定义：设 $r$ 是关系 $\langle H, h \rangle$ 并设 $X$ 是 $H$ 的一个子集，那么 $r$ 在 $X$ 的属性上的（关系）投影 $r\{X\}$ 是关系 $\langle X, x \rangle$ ，其中 $x$ 是所有元组 $t\{X\}$ 组成的一个集合，使得 $t$ 是 $h$ 的一个元组。

■定义：设关系 $r_1, \dots, r_n$  ( $n \geq 0$ ) 是可连接（joinable）的，即，设它们具有相同类型的同名属性。那么 $r_1, \dots, r_n$ 的连接（join） $\text{JOIN}\{r_1, \dots, r_n\}$ ，是一个关系，它的（a）标题是 $r_1, \dots, r_n$ 的标题的并集，且（b）正文是所有元组 $t$ 组成的一个集合，其中 $t$ 是来自 $r_1$ 的一个元组， $\dots$ ，以及来自 $r_n$ 的一个元组的并集。注意：这种连接有时也更明确地称为自然（natural）连接。请注意，它是一个 $n$ 元操作符，而不是仅仅是一个二元操作符（ $n=2$ 仅仅是一个常见的特例，关于 $n < 2$ 的情况，参见习题3.1）。

同时请注意在操作数关系 $r_1, \dots, r_n$ 没有同名的属性这个重要的特例下，连接退化为笛卡儿积。

■定义：带有标题 $H$ 的一个关系变量（relvar）是一个变量 $R$ ，使得仅当值 $r$ 是具有标题 $H$ 的关系时，值 $r$ 才可以赋值给该变量。 $H$ 的属性是 $R$ 的属性。同时，如果把关系 $r$ 赋值给 $R$ ，那么根据该赋值， $r$ 的正文和元组分别是 $R$ 的正文和元组。注意：正如定义所说，仅当（强调）关系 $r$ 与关系变量 $R$ 具有相同的标题时， $r$ 才可以赋值给 $R$ 。事实上，当且仅当：（a） $r$ 与 $R$ 有相同的标题，且（b） $r$ 满足所有应用于 $R$ 的约束，关系 $r$ 可以赋值给关系变量 $R$ ——这里的“所有应用于 $R$ 的约束”包含 $R$ 中存在的FD，但一般情况下不只限于这样的FD。

[1]这里用的符号有点儿混乱。如果 $X$ 是 $H$ 的一个子集，则 $X$ 是一个属性名称集合，比如 $\{A_1, \dots, A_n\}$ ，因此， $t$ 在 $X$ 上的投影 $t\{X\}$ ，明显地必须用双括号来书写，比如这样： $t\{\{A_1, \dots, A_n\}\}$ 。当然，我们不这样做，我要在本书其余部分忽略此混乱。但是，我至少应该解释为什么它会出现。基本上，之所以出现这种情况，因为我们把符号 $X$ 两个截然不同的诠释混为一谈了。一方面，我们使用该符号表示上述集合——

即，元素是 $A_1, \dots, A_n$ 的集合；另一方面，我们使用它来表示属性名 $A_1, \dots, A_n$ 的逗号分隔列表作为一个人工书写形式，代表这些元素（比如，在论文中）。前者是符号 $X$ 实际表示的内容，后者是我们表述那个符号的意义的语法形式。

## 5.2 函数依赖

现在我将妥善处理函数依赖的概念。同样，我将提出明确的定义，但本节更多地讨论这些定义和它们的一些影响。

■定义：设H是一个标题，那么关于H的函数依赖（FD）是一种形式为 $X \rightarrow Y$ 的表达式，其中X（决定因素）和Y（依赖因素）两者都是H的子集。注意：如果H是不言自明的，那么关于H的FD这句话可以缩写为仅FD。

下面是几个例子：

---

---

$$\{\text{CITY}\} \rightarrow \{\text{STATUS}\}$$
$$\{\text{CITY}\} \rightarrow \{\text{SNO}\}$$

---

---

请注意，FD是关于某个标题定义的，而不是关于某个关系或某个关系变量定义的。例如，刚才所示的这两个FD就定义在任何包含属性CITY、STATUS和SNO的标题上（也有可能还有其他属性）。

还请注意，从形式上看，FD仅仅是一个表达式：这个表达式在解释某个特定的关系时，根据定义，它就成为一个计算结果不是TRUE就是

FALSE的命题。例如，如果上面所示的两个FD关于（图1-1）关系变量S的当前值（代表）的关系来解释时，那么第一个计算结果为TRUE，而第二个计算结果为FALSE。当然，在某些特定情况下，仅当表达式的计算结果为TRUE时，才把这样一个表达式非正式地定义为一个FD，这种定义方法是常见的。但这样的定义没有办法说明未能满足给定的关系或违反某个FD的情况。为什么会这样呢？因为，根据非正式的定义，不满足的FD，首先将不会是一个FD。例如，我们将无法说关系变量S的当前值（代表）的关系违反了上面所示的第二个FD。

我真的怎么强调上述这点都不为过。对于大多数人来说，它代表着思维上的转变；但是如果你想了解设计理论都是关于什么的，那么这是一个必须进行的转变。这个要点是这样的：大部分关于FD的著作，包括首次推出这个概念的早期研究论文，实际上都完全没有定义FD的概念！他们说的都是类似“Y函数依赖于X，当且仅当任意两个元组在X上一致时，它们也在Y上一致”的内容，当然，这是完全正确的，但它不是FD的定义，它是FD待满足（satisfied）的含义的定义。但是，如果我们想开发一种FD理论，那么我们显然需要能够脱离某个特定的关系或关系变量的上下文来谈论作为对象本身的FD。更具体地说，我

们需要把上述FD的概念与在某个上下文中可能有某个解释或含义的概念脱离。事实上，设计理论可以看作一小块逻辑，逻辑完全不是关于含义的，而是关于正式的操作的。

接下来是这个定义。

■定义：设关系 $r$ 具有标题 $H$ 并且设 $X \rightarrow Y$ 是一个关于 $H$ 的FD，比如 $F$ 。如果对 $r$ 的每一对元组 $t_1$ 和 $t_2$ ，每当 $t_1\{X\}=t_2\{X\}$ 时，都有 $t_1\{Y\}=t_2\{Y\}$ 成立，则 $r$ 满足 $F$ ；否则 $r$ 违反（violate） $F$ 。

注意，满足或违反某些特定FD的是关系，而不是关系变量。例如，关系变量 $S$ 的当前值（代表）的关系同时满足下面这两个FD：

---

---

$$\begin{aligned}\{CITY\} &\rightarrow \{STATUS\} \\ \{SNAME\} &\rightarrow \{CITY\}\end{aligned}$$

---

---

并且违反这个函数依赖：

---

---

$$\{CITY\} \rightarrow \{SNO\}$$

---

---

■定义：当且仅当可以赋值给关系变量 $R$ 的每一个关系都满足 $F$ 时，关系变量 $R$ 中存在FD  $F$ （等



价的说法是，关系变量R服从（is subject to）FD F）。关系变量R中存在的FD是R的FD。

重要提示：请注意这里描述的术语的区别——FD是被关系满足（或违反）的，但是在关系变量中存在（或不存在）的。请同时注意，我会在整本书中遵循这种区别。举例来说，以下FD在关系变量S中存在：

---

---

$$\{\text{CITY}\} \rightarrow \{\text{STATUS}\}$$

---

---

而下面这些FD则在关系变量S中不存在：

---

---

$$\begin{aligned}\{\text{SNAME}\} &\rightarrow \{\text{CITY}\} \\ \{\text{CITY}\} &\rightarrow \{\text{SNO}\}\end{aligned}$$

---

---

（将它与前一个定义的例子对比。）所以，现在，我们最终知道了给定的关系变量服从给定的FD的确切含义。

## 5.3 Boyce/Codd范式

我们对FD的本质有了正确的认识后，就可以继续解决一个关系变量属于BCNF意味着什么。我再次通过一系列精确的定义开始这一过程。

■定义：设 $X \rightarrow Y$ 是一个关于标题H的FD，比如F，那么当且仅当它被每个带有标题H的关系满足时，F是平凡（trivial）的。

第4章定义了一个平凡FD是一个不可能违反的FD。那个定义当然没有错，但刚刚提出的这个定义是更可取的，因为它明确地提到了相关的标题。第4章也提到，我们可以很容易地看到当且仅当Y是X的一个子集时，FD  $X \rightarrow Y$ 是平凡的。那也是真实的，但我现在可以说，后一个事实不是一个真正的定义，而是一个定理，从上述定义出发很容易证明它。（另一方面，上述定义对于确定一个给定FD是否是平凡的不是非常有帮助，而定理却是很有帮助的。出于这个原因，我们可以把定理当作一个可操作的（operational）定义，因为它确实提供了一个可以很容易应用的有效测试。）[\[1\]](#)为了记录在案，让我明确地陈述此定理。

■定理：设 $X \rightarrow Y$ 是一个FD，比如F。则当且

仅当依赖因素Y是决定因素X的一个子集时，F是平凡的。

现在回到定义上面。

■定义：关系变量R的一个超键是R的标题H的一个子集SK，其中R中存在FD  $SK \rightarrow H$ （“ $SK \rightarrow H$ 是R的一个FD”）。该FD是R上的一个超键约束。

例如，{SNO}、{SNO,CITY}和{SNO,CITY,STATUS}都是关系变量S的超键。

■定义：当且仅当FD  $X \rightarrow Y$ 在关系变量R中存在且对于X的任何真子集 $X-$ ，在R中都不存在 $X- \rightarrow Y$ 时，FD  $X \rightarrow Y$ 关于关系变量R是不可约的（或只是不可约的，如果R是不言自明的）。

例如，FD  $\{CITY\} \rightarrow \{STATUS\}$ 关于关系变量S是不可约的。相比之下，FD  $\{CITY,SNO\} \rightarrow \{STATUS\}$ 虽然肯定是S中存在的FD，但它关于S是可约的（reducible）。注意，虽然这类FD是关于某个标题定义的，但是FD不可约性是关于某个关系变量定义的。换句话说，如前所述的FD只是一个语法概念（FD只是一个需要一定语法形式的表达式），而FD不可约性是一个语义学问题

（它必须处理相关的关系变量的含义）。注意：我不会在后面假定我们正在谈论的只是不可约的FD，虽然在实践中，我们通常这么做。

■定义：关系变量R的一个键是R的标题H的一个子集K，其中FD  $K \rightarrow H$ 是R的一个不可约的FD。该FD是R上的一个键约束。

请注意在上述定义中对FD不可约的需求。

■定义：设关系变量R具有标题H，并且设  $X \rightarrow Y$  是一个关于H的FD，比如F。则当且仅当每一个满足R的键约束的关系r也满足F时，R的键才蕴含F。

这个定义需要某些详细的阐述。首先，说某个关系满足某个键约束的含义是，它满足适用的唯一性要求；并且如果它满足构成某个键的属性集合的唯一性要求，当然，它也满足这个属性集合的每一个超集（当然，只要这个超集是相关的标题的一个子集）的唯一性要求，换句话说，对于每一个相应的超键满足唯一性要求。因此，定义中的“满足R的键约束”这句话可以替换为“满足R的超键约束”而不造成任何重大的区别。同样，概念“键才蕴含”也可以替换成“超键才蕴含”而不造成任何重大的区别。

其次，如果定义中提到的FD  $F$ 是平凡的，会发生什么情况呢？那么，在这种情况下，根据定义， $F$ 被带有标题 $H$ 的每一个关系 $r$ 满足，所以，更不用说， $F$ 肯定被满足 $R$ 的键约束的每一个关系 $r$ 满足。所以总是平凡地蕴含平凡的FD键。

然后，假设 $F$ 是非平凡的。那么，很容易证明下面的定理。

■定理：设 $F$ 是关系变量 $R$ 中存在的FD，则当且仅当它是 $R$ 上的一个超键约束时， $R$ 的键才蕴含 $F$ 。

换句话说，就像平凡的FD，正式的定义对于确定键是否蕴含一个给定FD不是非常有帮助，但定理很有帮助。出于这个原因，我们可以把定理作为一个可操作的定义，因为它提供了一个有效的测试，并可以很容易在实践中应用。

而现在，我终于可以定义BCNF了。

■定义：当且仅当 $R$ 的键蕴含关系变量 $R$ 的每一个FD时， $R$ 属于Boyce-Codd范式（BCNF）。

然而，根据本节已经讨论过的各种定义和定理，我们可以看到以下“可操作的”的定义也是有

效的。

■定义：当且仅当对于关系变量R中存在的每一个非平凡的FD  $X \rightarrow Y$ , X都是R的一个超键时，R属于Boyce-Codd范式（BCNF）。

正如第4章所述，从这个定义可以得出，一个属于BCNF的关系变量中只存在两种FD，要么是平凡的（很明显，我们不能摆脱这些），要么是从超键出来的箭头（我们也不能摆脱这些）。当我谈到“摆脱”一些FD时，虽然现在我想补充的是，我担心我有一点草率（我希望这是偶尔的情况）……例如，考虑关系变量S，此关系变量服从FD  $\{CITY\} \rightarrow \{STATUS\}$  以及其他FD；因此，正如在第3章中说明的，建议把此关系变量分解为它分别在  $\{SNO, SNAME, CITY\}$  上的投影SNC和  $\{CITY, STATUS\}$  上的投影和CT。但是，如果我们这样做，那么关系变量S中还存在的FD  $\{SNO\} \rightarrow \{STATUS\}$  “消失了”。在某种意义上，其实我们的确是“摆脱它”了。但是，FD消失了意味着什么？答案是：它被替换成多关系变量约束（即，跨越两个或更多关系变量的约束）了。因此，这个约束当然仍然存在，它只是不再是一个FD而已。[\[2\]](#)当我在本书中其他地方说“摆脱”一些依赖时也是这个意思。

[1]这里描绘的区别有时被界定为是语义与语法（semantic vs.syntactic）的区别。要详细地说明这点：原来的定义——当且仅当F被带有相关标题的每一个关系满足时，F是平凡的，是语义，因为它定义了概念的含义，相反，我称之为“可操作的”定义——当且仅当Y为X的一个子集时，F是平凡的，这是语法，因为它以一个纯粹语法的方式提供了一个可以执行的检查。（在后面，我们会多次遇到语义和语法概念之间的这种区别。事实上，后面马上介绍一个关于FD不可约性的概念的典型例子，请参阅它。）

[2]它是一个FD，但它是两个关系变量（即，SNC和CT）的连接中存在的一个FD，而不是在一个如前所述的单独的关系变量中存在的FD。但是，请注意，对这两个关系变量强制执行键约束将“自动”执行多关系变量的约束；也就是说，所涉及的多关系变量约束是明确声明的约束所蕴含的，或者是明确声明的约束的一个合乎逻辑的结果。

## 5.4 希思定理

再次考虑关系变量S，以及它的FD{CITY}  $\rightarrow$  {STATUS}。假设我们不是像在第3章那样把此关系变量分解为关系变量SNC和CT，而是分解为关系变量SNT和CT，这里的CT和以前一样，但SNT具有标题{SNO、SNAME、STATUS}而不是{SNO、SNAME、CITY}。图1-1所示的S值相对应的SNT和CT的示例值如图5-1所示。我希望你能从这幅图中看到以下要点。

■关系变量SNT和CT都属于BCNF（键分别是{SNO}和{CITY}，且在這些关系变量中存在的仅有的非平凡的FD都是“从超键出来的箭头”）。

■但是，与第3章中的分解不同，这种分解不是无损的，而是有损的（lossy）。例如，我们不能从图5-1中得出供应商S2是在巴黎（paris）还是在雅典（Athens），注意，如果我们将这两个投影进行连接会发生什么情况，[\[1\]](#)所以我们已经丢失了信息。



SNT

SNO	SNAME	STATUS
S1	Smith	20
S2	Jones	30
S3	Blake	30
S4	Clark	20
S5	Adams	30

CT

CITY	STATUS
Athens	30
London	20
Paris	30

图 5-1 关系变量SNT和CT——示例值

让我们来仔细看看这个例子。首先，下面是关系变量SNT和CT的谓词。

■SNC：供应商SNO名为SNAME并且具有状态STATUS。

■CT：城市CITY具有状态STATUS。

因此，这两个关系变量的连接的谓词是：

供应商SNO名为SNAME并且具有状态STATUS并且城市CITY具有状态STATUS。

现在回想一下，关系变量S的谓词（见附录D中习题2.6的答案）：

供应商SNO名为SNAME并且它位于城市CITY，该城市具有状态STATUS。

后一个谓词与连接的谓词显然是不一样的。更准确地说，如果某个给定的元组 $t$ 满足后一个谓词，那么元组 $t$ 也满足连接的谓词，但反之则是不成立的。这就是为什么连接“丢失信息”或“有损”的原因，只是因为某个元组出现在连接中，我们不能假设它也出现在原来的关系变量 $S$ 中。

那么究竟是什么让一些分解是无损的而其他分解却是有损的呢？这是规范化理论的核心问题，因此正式地陈述如下：

设 $r$ 是一个关系，并设 $r_1, \dots, r_n$ 为 $r$ 的投影。为了使 $r$ 等于这些投影的连接，必须满足什么样的条件呢？（顺便说一下，注意，正如前面提到的，在这里有个隐性假设，即连接是一个 $n$ 元操作）。

这个问题的一个重要的（尽管是片面的）答案是Ian Heath（伊恩·希思）在1971年提供的。那时他证明了如下定理：

■希思定理（针对关系）：设关系 $r$ 具有标题 $H$ ，并设 $X, Y, Z$ 都是 $H$ 的子集，且 $X, Y, Z$ 的并

集等于H。设XY代表X和Y的并集，XZ也代表类似的含义。如果r满足FD  $X \rightarrow Y$ ，则r等于其在XY和XZ上的投影的连接。

举例来说，再次考虑供应商关系（即，如图1-1所示的关系变量S的当前值）。此关系符合FD  $\{CITY\} \rightarrow \{STATUS\}$ 。因此，设X为{CITY}，Y为{STATUS}，且Z为{SNO, SNAME}，希思定理告诉我们，把此关系分解为其在{CITY, STATUS}和{CITY, SNO, SNAME}<sup>[2]</sup>上的投影是无损的，正如事实上我们所知道的。

现在，重要的是要明白，（再次重复）Heath对原来问题的回答是片面的。我会用前面的例子来解释这意味着什么。基本上，这个定理确实告诉我们，分解成投影SNC和CT（参见第3章中图3-2）是无损的，但是，它并没有告诉我们分解成SNT和CT（见图5-1）是有损的。换句话说，如果我们在FD的基础上分解，就像我们在图3-2的例子中所做的，那么希思定理提出此分解将是无损的，但如果我们在一些其他的基础上分解，例如我们在图5-1的例子中所做的，那么此定理对此事无能为力。因此，对于一个给定的（二元）分解是无损的，该定理给出了一个充分条件，但不是一个必要条件。因此，可以推论出，即使关系r不满足FD  $X \rightarrow Y$ ，也有可能以无损的方式，把r分

解为其在XY和XZ上的投影。注意：本书后面将介绍希思定理的一个更强的形式，同时给出必要条件和充分条件（见第12章）。

顺便说一句，我还要说，在Heath证明了他的定理的论文中，也给出了他所谓的“第三”范式的定义，这实际上是BCNF的一个定义。由于该定义比Boyce和Codd的定义早三年多，在我看来，BCNF按理说应该称为Heath范式。但事实并非如此。

还记得不，3.1节提到了类似下面这样的话：

如果你一直在认真阅读，你可能会合理地指责我在前面的讨论中耍了一个小小的花招。具体而言，我已经考虑了关系的无损分解的含义，但我们应该谈论规范化问题，不是一个分解关系的问题，而是一个分解关系变量的问题。

这些说法在这里也适用！因此，让我们回到关系变量.....再次考虑关系变量S。假设我们确实决定执行分解成“投影”关系变量SNC和CT的建议；而且，假设我们希望分解是无损的，事实上，我们一定会这样希望。换句话说，我们要的分解是这样的，在任何时候，关系变量S的当前值等于SNC和CT的连接的当前值。[\[3\]](#)也就是说，

我们希望S服从以下完整性约束（下面将它称为YCT）：

---

```
CONSTRAINT YCT  
S=JOIN{S{SNO,SNAME,CITY}, S{CITY,STATUS}};
```

---

现在，回顾第2章，S肯定服从以下约束（XCT）：

---

```
CONSTRAINT XCT  
COUNT (S{CITY})=COUNT (S{CITY,STATUS});
```

---

这里提醒一下，这个约束只是说S中存在  $FD\{CITY\} \rightarrow \{STATUS\}$ 。因此，根据希思定理，我们可以看到，关系变量S的每一个可能值，因为它必须满足约束XCT，所以它也肯定满足约束YCT。因此，约束XCT蕴含约束YCT（详细地说，它的意思是，如果关系变量S服从XCT（它确实服从），那么它肯定也服从YCT）。因此，约束YCT确实存在，而且正如需要的，把关系变量S分解为关系变量SNC和CT的确是无损的。因此，无论如何我们可以把希思定理应用到关系变量，而不只是关系。因此，让我们相应地再说明一遍。

■希思定理（针对关系变量）：设关系变量R具有标题H，并且设X、Y、Z是H的子集，其中X、Y、Z的并集等于H。设XY表示X和Y的并集，XZ也表示类似的含义。如果R服从FD  $X \rightarrow Y$ ，那么R可以无损地分解成其在XY和XZ上的投影。

关于无损分解（对于BCNF或其他）的一般问题，我还想进一步说明一点。再次考虑关系变量S，以及它的FD  $\{CITY\} \rightarrow \{STATUS\}$ 。根据希思定理，此关系变量可以无损地分解成其在  $\{SNO, SNAME, CITY\}$  和  $\{CITY, STATUS\}$  上的投影。但是，它显然也可以无损分解成这两个投影和（比如）其在  $\{SNAME, STATUS\}$  上的投影，也就是说，如果我们把所有这些投影连接起来，就回到了开始的地方。（如果不能立即明显地看出这点，那么使用我们的常用关系变量S的示例值自己检查这个说法。）然而，在重建原始关系变量的过程中，第三个投影显然是没有必要的。现在，在我们做数据库设计时，出于显而易见的原因，通常只考虑那些在重建过程中每个投影都需要的分解方式，但本书讨论的是一般的分解，所以我不会把自己限制在每个投影都需要的那些分解方式上（当然，除非有明确的相反陈述）。

[\[1\]](#)参见3.1节的脚注中关于“有损连接”的讨论。

[2]或者，因为我们在书写时会“更自然地”倾向于交换两个属性集合并用“更自然的”顺序指定各个属性，即写成{SNO、SNAME、CITY}和{CITY、STATUS}。

[3]在这里，我再次采用合宜的假设，设关系变量S、SNC和CT共存（仿佛彼此一起存在）。第3章的注7是第一次假设。

## 习题

5.1 在本章正文定义的连接版本是一个 $n$ 元的操作符，而不仅是一个二元的操作符。但如果 $n=1$ 或 $n=0$ ，会发生什么情况呢？

5.2 请尽你所能精确地定义一个关系变量服从一个函数依赖的含义。

5.3 考虑以下FD：

- 
- a.  $\{CITY\} \rightarrow \{STATUS\}$
  - b.  $\{SNO, CITY\} \rightarrow \{STATUS\}$
  - c.  $\{SNO\} \rightarrow \{SNO\}$
  - d.  $\{SNO, CITY\} \rightarrow \{SNO\}$
  - e.  $\{SNO\} \rightarrow \{SNO, CITY\}$
  - f.  $\{SNAME, SNO\} \rightarrow \{STATUS, CITY\}$
  - g.  $\{SNO\} \rightarrow \{STATUS\}$
  - h.  $\{SNAME\} \rightarrow \{STATUS, SNO\}$
- 

哪些FD是平凡的？哪些是如图1-1给出的关系变量 $S$ 的当前值满足的？哪些存在于关系变量 $S$ 中？哪些对于关系变量 $S$ 是不可约的？

5.4 请证明希思定理，也证明该定理的逆命题是无效的。注意：关于这方面，也请参阅第11章的练习11.3。



5.5 说一个超键蕴含一个FD到底是什么意思？或者，说一个键蕴含一个FD又是什么意思？

5.6 下面是一个谓词：在某日 $d$ 的课时 $p$ ，学生 $s$ 正在参加课程 $l$ ，这是教师 $t$ 在教室 $c$ 讲授的（这里的 $d$ 是一星期中的一天，周一至周五——而 $p$ 是在一日之内的一个课时——1~8）。课程持续一个课时并在那一周讲授的所有课程中具有唯一的课程标识符。为此数据库设计一个BCNF关系变量集合。这些关系变量的键分别是什么？

5.7 设计一个数据库。其中的实体是员工和程序员；每一个程序员都是员工，但有些员工不是程序员；员工有员工编号、名称和工资属性；程序员有一个（单独的）编程语言技能属性。如果程序员可以有任意数量的技能，那么数据库的设计有什么区别？

5.8 本章正文中给出的键的定义的形式与第4章中给出的键的定义的形式有所不同。请问这些定义在逻辑上是等价的吗？

## 第6章 保持函数依赖

Nature does require.

Her times of preservation.

——William Shakespeare: Henry VIII

再次考虑我们常用的供应商关系变量S。由于{SNO}是一个键，因此该关系变量肯定服从 $FD\{SNO\} \rightarrow \{STATUS\}$ 。因此，设X为{SNO}，Y为{STATUS}，Z为{SNAME,CITY}，希思定理告诉我们，可以把此关系变量分解为关系变量SNC和ST，这里的SNC具有标题{SNO,SNAME,CITY}，而ST具有标题{SNO,STATUS}。与图1-1所示的S值对应的SNC和ST的示例值，如图6-1所示。

SNC

SNO	SNAME	CITY
S1	Smith	London
S2	Jones	Paris
S3	Blake	Paris
S4	Clark	London
S5	Adams	Athens

ST

SNO	STATUS
S1	20
S2	30
S3	30
S4	20
S5	30

## 图 6-1 关系变量SNC和ST——示例值

在这个分解中：

■关系变量SNC和ST都属于BCNF——两者的键都是{SNO}，这些关系变量中存在的仅有的非平凡的FD都是“从超键出来的箭头”。

■更重要的是，这个分解肯定是无损的（实际上是由希思定理保证的），即如果我们把SNC和ST连接在一起，就重新得到了S。

■不过， $FD\{CITY\} \rightarrow \{STATUS\}$ 已经丢失了，当然，我的意思是，它被一个特定的多关系变量约束替代了，正如第5章解释的。[\[1\]](#)所涉及的约束可以如下所示：

---

```
CONSTRAINT.....  
COUNT ( (JOIN{SNC,ST}) {CITY}) =  
COUNT ( (JOIN{SNC,ST}) {CITY,STATUS}) ;
```

---

说明：这个约束说的是，如果我们连接SNC和ST，得到一个结果（假定称它为S），在这个结果中，不同城市的数量与不同的城市/状态对的数量相等。而事实上，存在后一种属性相当于原关系变量S中存在 $FD\{CITY\} \rightarrow \{STATUS\}$ 。

因此，我们已经“丢失”了一个FD。这句话的含义是什么？当然正如我们刚刚看到的，替代它的多关系变量的约束是更难以说明的。也许更重要的一点是，它是更难以执行的（就是，如第3章图3-2所示，比它本来首选分解成投影SNC和CT的执行更困难）。<sup>[2]</sup>例如，假设我们更新关系变量SNC，把供应商S1的城市，从伦敦更改为雅典，那么我们也必须同时更新关系变量ST，把供应商S1的状态从20更改为30，因为如果不这么做，那么将SNC和ST重新连接在一起将产生一个不是关系变量S的一个合法值的结果。（与此相反，如果我们更新关系变量SNC，把供应商S2的城市从巴黎更改为雅典，那么我们不必也更新关系变量ST，但我们还是要检查关系变量ST，以确定这一事实。）

旁白：对于一个架构良好的数据库管理系统（DBMS），不必让用户对关系变量ST做必要的检查，而让系统“自动”做这件事，这是有可能的。甚至也有可能让系统“自动”执行任何必要的补充更新。然而，即使有这样的系统，仍然存在约束很难执行的情况（即，仍然有更多的工作要做，即使是系统而不是用户做这些工作）。无论如何，发生这种情况的可能性都只是写作的时候的一个白日梦，在一般情况下，当今的商业化产

品通常甚至都不允许说明多关系变量约束；上述可能性在今天是遥不可及的，而处理（特别是执行）这样的约束是用户的责任。旁白结束。

因此，这传递出的信息是：尽量选择保持了FD，而不是丢失它们的分解方法。（对于这个例子，把在{SNO,STATUS}上的投影更换为在{CITY,STATUS}上的投影解决了这个问题。）换句话说，不严格地说，如果在原来的关系变量中存在FD  $X \rightarrow Y$ ，那么尽量不要选择使得X在其中一个关系变量中而Y在另一个关系变量中的分解方法。注意：当然，我在此假设分解不在FD  $X \rightarrow Y$ 本身的基础上进行，因为如果是这样，我们会实际得到有两个X的分解，其中一个X将与Y在同一个关系变量中（一定如此）而另一个X不是这样。我也默许假设， $X \rightarrow Y$ 是在原关系变量中存在的FD总集的所谓的一个不可约覆盖的一部分。本章后面将讨论不可约覆盖。

## 6.1 遗憾的冲突

FD保持的基本思想是简单的，然而，遗憾的是，关于这个问题有相当多的内容需要说明。首先，我想展示有些人可能会认为是极其罕见的一个例子。假设我们有一个关系变量SJT，它具有属性S（学生）、J（课程）、T（教师），而谓

词是“学生S由教师T讲授课程J”。应用以下业务规则：[\[3\]](#)

■对于每一门课程，这门学科的每名学生只由一位教师讲授。

■每位教师只讲授一门课程。

■每名学生学习多门课程，因此由几位教师讲授（一般情况下）。

■每门课程由数名学生学习（一般情况下）。

■每门课程由几位教师任教（一般情况下）。

■学习同一课程的不同学生可能会也可能不会由同一位教师讲授。

这关系变量符合这些规则的一个示例值如图6-2所示。

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

图 6-2 关系变量SJT的示例值

关系变量SJT中存在的FD是什么？从第一条业务规则，我们可以得出 $\{S,J\} \rightarrow \{T\}$ 。从第二条业务规则，我们可以得出 $\{T\} \rightarrow \{J\}$ 。对剩余规则的仔细分析将表明，除了平凡或可约（或两者兼有）的FD以外，不存在任何其他的FD。因此，存在的非平凡且不可约的FD只有下面这两个：

$$\{S,J\} \rightarrow \{T\}$$

$$\{T\} \rightarrow \{J\}$$

那么此关系变量的键都是什么呢？首先， $\{S,J\}$ 是一个键，因为显然整个标题函数依赖于 $\{S,J\}$ ，而不函数依赖于 $\{S,J\}$ 的任何真子集。此外， $\{S,T\}$ 也是一个键，这是因为：

a.这肯定是事实，假设存在FD $\{T\} \rightarrow \{J\}$ ，那么整个标题函数依赖于 $\{S,T\}$ 。

b.这也是事实，假设 $FD\{S\} \rightarrow \{J\}$ 和 $\{T\} \rightarrow \{S\}$ 不存在，那么整个标题不函数依赖于 $\{S,T\}$ 的任何真子集。

所以此关系变量有两个键，即 $\{S,J\}$ 和 $\{S,T\}$ 。<sup>[4]</sup>也许对于这点还有更多内容， $\{T\}$ 不是一个键，因此关系变量SJT服从一个不是“一个从键出来的箭头”的FD（更正式地说明这个随意的说法是：键不蕴含它）。因此，此关系变量不属于BCNF，尽管它属于3NF。（习题：检验这种说法。）并且它存在冗余，例如，根据在图6-2中显示的示例值，White（怀特）教授讲授数学这一事实出现了两次。正如你所期望的，它也存在更新异常，例如，再次对于图6-2，我们不能在不丢失Brown（布朗）教授讲授物理的信息的条件下删除Jones（琼斯）学习物理这一事实。

现在，我们可以通过适当分解关系变量解决这些问题。通过对 $FD\{T\} \rightarrow \{J\}$ 应用希思定理（设X、Y和Z分别是 $\{T\}$ 、 $\{J\}$ 、 $\{S\}$ ），我们得到以下无损分解：

---

---

TJ	{T,J}
KEY	{T}
TS	{T,S}
KEY	{T,S}

---

---



我会把下面的问题留作一个习题，显示与图6-2所示的SJT的值对应的这两个关系变量的值，显示它们属于BCNF，并检验这个分解确实避免了上述冗余和更新异常。尤其要注意， $FD\{T\} \rightarrow \{J\}$ 在分解之后成为一个键约束，相反，在原来的设计中，必须分别说明和执行它。

但还有一个问题。实际上，虽然分解成TJ和TS确实避免了某些异常，但遗憾的是，这种分解引入了其他问题。具体来说，FD

---

$$\{S,J\} \rightarrow \{T\}$$

---

丢失了（当然 $FD\{T\} \rightarrow \{J\}$ 不蕴含它，这是在分解的结果中存在的唯一的非平凡FD）。因此，关系变量TJ和TS不能独立地更新。例如，把元组插入

---

(Smith,Prof.Brown)

---

TS的尝试必须遭到拒绝，因为Brown（布朗）教授讲授物理而Smith（史密斯）已经由Green（格林）教授讲授物理，但这个事实在不检查TJ的情况下不能检测到。

总之，前面的例子说明了如下事项：我们在做无损分解时通常有两个目标，BCNF投影和保持FD，而可悲的是，这些目标可能是互相冲突的（也就是说，不总是能够同时实现两者）。

我曾经在本章初稿的此处，写了以下内容：

那么，我们应该放弃哪个目标？你看，如果我可以，我会告诉你，但我不能。SJT示例演示的是标准化的理论，虽然它是重要的，但它其实是远远不够的，我的意思是说，有许多问题它没有回答。因此，它传递出的信息是：我们需要更多的科学！规范化理论肯定是科学的，但它并不能解决所有的设计问题。

然而，在我的一个审校者的提示中，我得出结论，这个段落可能是夸大的。在这里我们不太需要更多的科学，需要的是更好的实现！也就是说，在类似本例的情况下，容忍一个不够规范化的设计的主要论据，是当今的DBMS在处理类似本例中“丢失”的FD的多关系变量约束时相当尴尬这一事实。因此，让我再树立一个标准，并明确地说明我对此问题的观点。在我看来，在这些有冲突的情况下，保持FD是需要放弃的目标。[\[5\]](#)

[\[1\]](#)在这种情况下，说FD“丢失”是常见的，但它是

不合适的轻率说法——真正发生的情况是（再次重复）所涉及的FD已经被另一个约束替代了。但关键是，另一个约束不是一个如前所述的FD。

[2]也可能有性能损失。现在，我真的不应该提及此事，正如第1章指出的，我永远不希望性能方面的考虑成为我的逻辑设计背后的驱动力。但在目前的情况下，性能正好也是加强我的主要论点的另一个因素。

[3]业务规则是通常用自然语言表达的一个语句，它应该捕捉到数据库中的数据是指什么或者它是如何被约束的的某个方面。关于这个术语任何更精确的定义没有达成共识，但至少大多数作家会同意，关系变量的谓词是一个重要的特例。

[4]你可以看到这两个键有重叠部分。顺便说一下，正如在第4章中处理关系变量SNP的方式，我选择不使这些键成为主键，这就是为什么在图6-2中没有双下划线的原因。

[5]在本例中，当然，如果我们希望（比如说）即使在那一刻Black（布莱克）教授没有学生，也能够记录布莱克教授教物理的事实，我们必须按照指示分解关系变量。

## 6.2 第二个例子

上一节的开始曾经指出，SJT的例子可能被认为是极其罕见的。然而，现在我要声称它不是极其罕见的，完全不是极其罕见的；我要给出几个例子，我认为它们表明，保持FD的问题，往往比你想象的出现得更频繁。

一般认为规范化是一个从1NF到2NF再到3NF（等）顺序步进的过程。让我们同意这个过程，即从1NF到2NF再到3NF（等）顺序步进是公认的“传统的规范化过程”。在这一节和接下来的两节中，我想提出一系列的例子来证明，过于盲目地遵循传统的规范化过程不一定是一个好主意。我的第一个例子涉及一个看起来像这样的关系变量：

---

---

RX1{SNO,PNO,CITY,STATUS,QTY}

---

---

名称RX1的意思是relvar example 1（关系变量示例1）的缩写；谓词是供应商SNO位于城市CITY，此城市具有状态STATUS，并且该供应商供应数量为QTY的零件PNO。假设在此关系变量中存在以下FD：

---

---

$$\begin{aligned}\{SNO\} &\rightarrow \{CITY\} \\ \{CITY\} &\rightarrow \{STATUS\} \\ \{SNO, PNO\} &\rightarrow \{QTY\}\end{aligned}$$

---

显而易见，在此关系变量中也暗示了存在下列函数依赖：[\[1\]](#)

---

$$\begin{aligned}\{SNO\} &\rightarrow \{STATUS\} \\ \{SNO, PNO\} &\rightarrow \{CITY, STATUS\}\end{aligned}$$

---

事实上，第二组FD可以扩展到 $\{SNO, PNO\} \rightarrow H$ ，其中H是整个标题；换言之， $\{SNO, PNO\}$ 是关系变量RX1的一个键。

现在回顾一下，当且仅当对于关系变量R的每个键K和每个非键属性A,  $FD \quad K \rightarrow \{A\}$ 都不可约时，它属于2NF。那么，显然，RX1不属于2NF，因为 $FD \{SNO, PNO\} \rightarrow \{CITY\}$ 是R X1的一个FD，但它并不是不可约的，具体而言，因为在此关系变量中也存在 $FD \{SNO\} \rightarrow \{CITY\}$ ，所以前一个FD不是不可约的。传统的规范化过程因此会建议，通过在 $FD \{SNO\} \rightarrow \{CITY\}$ 上应用希思定理来分解此关系变量。但是，如果我们这样做，这就是我们将得到的结果：

---

RX1A {SNO,CITY}  
KEY {SNO}  
RX1B {SNO,PNO,STATUS,QTY}  
KEY {SNO,PNO}

---

现在请注意，在这个分解中 $FD\{CITY\} \rightarrow \{STATUS\}$ 丢失了。因此，一个直接的教训是，保持FD的问题是 与从1NF到2NF的步骤相关的——而不只是与从3NF到BCNF的步骤相关，这是上一节的SJT例子所示的步骤。

旁白：这里的关系变量RX1A肯定属于2NF。与此相反，关系变量RX1B不属于2NF，因为 $FD\{SNO,PNO\} \rightarrow \{STATUS\}$ 是可约的。因此，我们可以再次应用希思定理来把它分解为它在 $\{SNO,STATUS\}$ 和 $\{SNO,PNO,QTY\}$ 上的投影，这两个投影都属于2NF；但实际上伤害已经造成，因为 $FD\{CITY\} \rightarrow \{STATUS\}$ 已经丢失。旁白结束。

在这个例子中，我们如何才能保持此FD呢？一个答案是：不在 $FD\{SNO\} \rightarrow \{CITY\}$ 的基础上分解，而在 $FD\{SNO\} \rightarrow \{CITY,STATUS\}$ 的基础上分解。但是，请注意，这个FD不是原始明确列出的FD之一，也不是我说的显然是那些明确FD所蕴含的，因此，不太可能选中它作为分解的基

础。然而，假设我们选择它，并执行相应的分解。那么结果如下：

---

```
RX1A' {SNO,CITY,STATUS}
KEY {SNO}
RX1B' {SNO,PNO,QTY}
KEY {SNO,PNO}
```

---

在这种分解中，STATUS出现在键为{SNO}的关系变量中，而不出现在键为{SNO,PNO}的关系变量中，从而保持 $FD\{CITY\} \rightarrow \{STATUS\}$ 。注意：当然，因为这里的关系变量RX1A仍然不符合3NF，所以我们可能会想进一步把它分解。但是，我们需要小心一点，具体来说，我们需要在 $FD\{CITY\} \rightarrow \{STATUS\}$ 的基础上分解，而不能在 $\{SNO\} \rightarrow \{STATUS\}$ 的基础上分解，否则我们将再次丢失一个FD。但是 $\{CITY\} \rightarrow \{STATUS\}$ 是传统的规范化过程会告诉我们使用的FD，因此，在这里不应该有什么问题。

解决上述问题的另一种方法将是，在 $FD\{CITY\} \rightarrow \{STATUS\}$ 的基础上分解原始关系变量RX1：

---

```
RX1A'' {CITY,STATUS}
KEY {CITY}
```

RX1B" {SNO,PNO,CITY,QTY}  
KEY {SNO,PNO}

---

这种分解也保持 $FD\{CITY\} \rightarrow \{STATUS\}$ 。但是，请注意，这个FD不是一个导致2NF违反（它不是“一个从真子键出来的箭头”）的FD，所以，再次强调，如果我们按照传统规范化的过程，我们在实践中，在这个阶段不太可能会选择它作为分解的一个基础。另请注意，因为这里的关系变量RX1B”仍然不属于3NF，所以我们可能会想进一步把它分解。我准备把进一步分解的详细方法留给你思考。

[1]第7章和第11章将讨论关于隐式地存在的函数依赖（“隐式函数依赖”）的大量问题。



## 6.3 第三个例子

让我们来看看另外一个例子。假设把供应商划分为类（C1、C2等），这样，我们有一个看起来像这样的关系变量RX2（如我对RX1所做的，为简单起见，我忽略供应商名称）：

---

---

```
RX2{SNO,CLASS,CITY,STATUS}  
KEY{SNO}
```

---

---

谓词是“供应商SNO属于类CLASS，它位于城市CITY，并具有状态STATUS”。也假设（a）每个类都只有一个关联的状态，且（b）每个城市都只有一个关联的状态，但（c）另一方面类和城市是完全相互独立的。那么存在下面的FD：

---

---

```
{CLASS}→{STATUS}  
{CITY}→{STATUS}
```

---

---

注意：我也假设有一个业务规则，大意是说，对于任何给定的供应商，城市的状态与类的状态是相等的（这就是为什么我们能够只用一个STATUS属性侥幸成功的原因）。

现在回顾一下，当且仅当关系变量R对于它

存在的每一个非平凡的FD  $X \rightarrow Y$ ，都有要么X是一个超键，要么Y是一个子键时，R属于3NF。那么，很显然，RX2不属于3NF，因为在FD{CITY}  $\rightarrow$  {STATUS}中，{CITY}不是一个超键，且{STATUS}不是一个子键。因此传统的规范化过程会建议，通过在FD{CITY}  $\rightarrow$  {STATUS}上应用希思定理来分解此关系变量。但是，如果我们这么做，下面就是我们所得到的结果（两个都属于3NF的投影关系变量）：

---

```
RX2A{CITY,STATUS}
KEY{CITY}
RX2B{SNO,CLASS,CITY}
KEY{SNO}
```

---

现在请注意，在这种分解中，FD{CLASS}  $\rightarrow$  {STATUS}丢失了。（当然，如果我们用另一个FD取代FD{CITY}  $\rightarrow$  {STATUS}作为分解的基础，那么后一个FD将会丢失。）所以，现在我们看到保持FD的问题，也可以是与从2NF到3NF的步骤相关的。

现在，我们可以在这个例子中，通过在FD{SNO}  $\rightarrow$  {CLASS,CITY}的基础上分解来保持FD，但是这个FD再次不大可能被选定为分解的基础，因为它不是明确表示的。[\[1\]](#)尽管如此，分

解的结果如下：

---

```
RX2A' {CLASS,CITY,STATUS}
KEY {CLASS,CITY}
RX2B' {SNO,CLASS,CITY}
KEY {SNO}
```

---

在这种分解中，在RX2B'中{CLASS,CITY}是一个（复合）外键，它引用RX2A'。关系变量RX2B'属于3NF。然而，关系变量RX2A'甚至都不属于2NF，因为 $FD\{CLASS,CITY\} \rightarrow \{STATUS\}$ 显然是可约的。因此，如果我们决定保持此关系变量， $FD\{CLASS\} \rightarrow \{STATUS\}$ 和 $\{CITY\} \rightarrow \{STATUS\}$ 都必须单独说明和执行。另外，我们可以把此关系变量分解为其在 $\{CLASS,STATUS\}$ 和 $\{CITY,STATUS\}$ 上的投影，在这种情况下，必须单独说明并强制执行一个适当的多关系变量约束。留给读者的习题：那个约束会是什么样子的呢？

[1]它不大可能作为分解的基础还有一个原因，是因为{SNO}是关系变量RX2的一个键（其实是唯一的键）。如果不是这样，我们可能会预期看到两个分别说明的FD，即， $\{SNO\} \rightarrow \{CLASS\}$ 和 $\{SNO\} \rightarrow \{CITY\}$ 。

## 6.4 第四个例子

现在考虑上一节的例子的修订版，再次把供应商划分为类，但每个类都只有一个相关的城市（每个城市依次只有一个相关的状态，如前一个例子）。因此，我们有一个看起来像这样的关系变量RX3（为简单起见，我忽略供应商名称）：

---

$$RX3\{SNO, CLASS, CITY, STATUS\}$$

---

事实上，RX3当然与RX2具有相同的标题，但RX3的谓词是不同的：供应商SNO是类CLASS的一部分，这个类有关联的城市CITY，这个城市具有状态STATUS。此外还存在如下FD：

---

$$\begin{aligned}\{SNO\} &\rightarrow \{CLASS\} \\ \{CLASS\} &\rightarrow \{CITY\} \\ \{CITY\} &\rightarrow \{STATUS\}\end{aligned}$$

---

关系变量RX3不符合3NF，因为在FD $\{CLASS\} \rightarrow \{CITY\}$ 中， $\{CLASS\}$ 不是一个超键且 $\{CITY\}$ 不是一个子键。（只要作相应的修改， $\{CITY\} \rightarrow \{STATUS\}$ 是同样的）。因此传统的规范化过程建议我们通过 $\{CLASS\} \rightarrow \{CITY\}$ 上应用希思定理来分解此关系变量。但

是，如果我们这么做，就会得到如下结果：

---

```
RX3A{CLASS,CITY}  
KEY{CLASS}  
RX3B{SNO,CLASS,STATUS}  
KEY{SNO}
```

---

RX3A属于3NF但RX3B仅属于2NF，并且如你所见， $FD\{CITY\} \rightarrow \{STATUS\}$ 丢失了。事实上，在 $FD\{CITY\} \rightarrow \{STATUS\}$ 的基础上分解会是更好的：

---

```
RX3A'{CITY,STATUS}  
KEY{CITY}  
RX3B'{SNO,CLASS,CITY}  
KEY{SNO}
```

---

RX3A'属于3NF而RX3B'仅属于2NF，但至少 $FD\{CITY\} \rightarrow \{STATUS\}$ 已保持。更重要的是，我们现在可以继续从 $FD\{CLASS\} \rightarrow \{CITY\}$ 的基础上分解RX3B'，从而获得如下结果：

---

```
RX3BA'{CLASS,CITY}  
KEY{CLASS}  
RX3BB'{SNO,CLASS}  
KEY{SNO}
```

---

这些关系变量都属于3NF。

现在，我们已经看了4个会丢失或可能丢失函数依赖的不同分解例子。关于这个话题还有更多内容，但一个明确的信息是：事实上，在实践中经常被教导的传统的规范化过程，在以下几个方面存在不足。具体地说，分别如下。

■传统的观点认为，保持FD是与从3NF到BCNF的步骤相关的，但正如我们已经看到的，情况不一定如此。

■使用常规方法通常建议的FD作为基础进行分解不一定是最好的。

■这一过程还假定，通过从1NF到2NF再到3NF（等）的顺序步骤可以找到最佳的设计方案。

当然，“第一”、“第二”等术语的命名方法加强了这最后一个观点.....但在某种程度上，那些命名实在不过是历史的偶然。我的意思是，如果第一个定义的范式是BCNF，发生这件事的可能性很大，因为那个定义在概念上是如此简单，包括因为它没有提到FD不可约性、非键属性、子键、1NF，2NF或3NF，那么就真的永远根本不会

有任何需要作为特定的范式创建2NF和3NF。[\[1\]](#)

[\[1\]](#)为了支持这一论点，我想引用一些Codd本人在他介绍了2NF和3NF的论文（见附录C）中说过的话：“[这些]范式相关的基本思路很简单，但它们有许多微妙的后果。笔者发现，解释和引导这些范式的精确定义需要用无数例子。”

## 6.5 一个能够工作的过程

现在展示一个过程，可以保证产生所有关系变量都属于3NF（虽然不一定属于BCNF）并保持所有FD的一种分解。[\[1\]](#)为方便起见，我在此后将它称作“3NF过程”。输入是一个关系变量R和在R中存在的FD的所谓的不可约覆盖（irreducible cover），比如C。我将花很短的时间解释什么是不可约覆盖，顺便说一句，我再次提到不可约这个词，但首先请让我说明这个过程。

1. 设S是标题的一个集合。S初始化为空集 {}。

2. 设X是在C中的某个FD的左边部分（决定因素），设C中的FD及其左边部分X的完整集合是  $X \rightarrow Y_1, \dots, X \rightarrow Y_n$ ；并设  $Y_1, \dots, Y_n$  的并集是Y。把X和Y的并集添加到S中。为每个不同的X执行此步骤。

3. 设U是R中不包含S中任何元素的属性的集合。如果U是非空的，则把U添加到S中。

4. 如果S中的元素不是R的超键，则把R的某个键K添加到S中。



此过程结束时，S的元素是一个3NF关系变量集合的标题，其中R可以无损地分解，且不会丢失任何FD。请特别注意，整个过程都没有明确提及2NF，甚至也没有以某种垫脚石的形式提及它。

那么，它是如何工作的呢？显然，一个不可约覆盖的概念是重要的。为了解释这一概念，让我先重申我过去已经呼吁几次的一些东西：即一些FD蕴含其他函数依赖的事实。作为一个简单的例子，FD  $X \rightarrow Y$  和  $Y \rightarrow Z$  一起蕴含FD  $X \rightarrow Z$ ——我的意思是，如果关系r满足前两个FD，那么它也一定满足第三个FD。或者，也许更重要的是：如果在关系变量R中存在前两个FD，那么也一定存在第三个FD。我们在上一节看到一个说明，在关系变量RX3中，因为存在FD  $\{\text{CLASS}\} \rightarrow \{\text{CITY}\}$  和  $\{\text{CITY}\} \rightarrow \{\text{STATUS}\}$ ，所以也存在FD  $\{\text{CLASS}\} \rightarrow \{\text{STATUS}\}$ 。

因此，一些FD蕴含其他FD。给定一个FD集合F，那么，我们可以明智地谈论F的一个覆盖。它的定义如下。

■定义：一个FD集合F的一个覆盖是一个FD集合C，其中F中的每个FD都被C中的FD蕴含。

作为一个简单的例子，设F是集合：

---

---

$\{X \rightarrow Y, Y \rightarrow Z, X \rightarrow Z\}$

---

---

那么以下两个都是F的覆盖：

---

---

$\{X \rightarrow Y, Y \rightarrow Z\}$

$\{X \rightarrow Y, Y \rightarrow Z, X \rightarrow Z\}$

---

---

这个例子说明了两点：首先，在一般情况下，覆盖不是唯一的；其次，FD的任何一个集合无疑是它本身的一个覆盖，因为无论如何每个FD都蕴含其本身。再次，更重要的一点是：对于一个给定的集合F执行一个覆盖C中的FD将“自动”执行集合F中的FD。因此，给定某个需要执行的FD集合F，就已经足以找到F的某个覆盖C并且用执行在C中的FD来代替它。（尤其是，执行F的一个不可约覆盖中的FD就足够了，而且这点很快就会变得清晰。）

现在，我可以定义什么是不可约的覆盖。

■定义：一个FD集合F的一个覆盖C是不可约（irreducible）的，当且仅当它具有以下所有的属性。

1.单例依赖因素（singleton dependant）：在C中的每个FD的右侧都只有一个属性。

2.不可约的决定因素（irreducible determinant）：在C中的每个FD本身是不可约的。注意：我在这里有点草率。回顾一下第4章和第5章，FD不可约的定义只是针对于某个关系变量的，但在这里我也没有说过任何与F中的FD存在于某个关系变量有关的话，因此，没有允许我们合法地谈论关于FD的不可约性的上下文。但是，我的意思是，左侧的任何属性都不可以在不丢失C是F的一个覆盖这个性质的情况下去掉。

3.没有冗余的FD：任何FD都不可以在不丢失C是F的一个覆盖这个性质的情况下从C中去掉。

由此产生的最明显的问题是：对于一些特定的FD集，我们怎样才能找到那个集合的一个不可约覆盖呢？第7章会正式地回答这个问题。而现在，让我只举一个例子，即在我们平时的供应商关系变量S中存在的FD的一个不可约覆盖：

---

$$\begin{aligned}\{SNO\} &\rightarrow \{SNAME\} \\ \{SNO\} &\rightarrow \{CITY\} \\ \{CITY\} &\rightarrow \{STATUS\}\end{aligned}$$

---

稍加说明：这是肯定的情况，S中存在的每一个FD都被这三个FD共同所蕴含，所以这三个FD肯定构成一个覆盖。此外，这三个FD中的每一个都具有单例依赖因素；没有属性可以从任何决定因素中去掉；并且没有任何FD可以去掉。因此，这个覆盖实际上是一个不可约覆盖。与此相反，下面的几个FD集虽然是S中存在的FD的覆盖，但都不是不可约的（在每种情况下，为什么不是呢？）

---

$\{SNO\} \rightarrow \{SNAME, CITY\}$   
 $\{CITY\} \rightarrow \{STATUS\}$   
 $\{SNO, SNAME\} \rightarrow \{CITY\}$   
 $\{SNO\} \rightarrow \{SNAME\}$   
 $\{CITY\} \rightarrow \{STATUS\}$   
 $\{SNO\} \rightarrow \{SNAME\}$   
 $\{SNO\} \rightarrow \{CITY\}$   
 $\{CITY\} \rightarrow \{STATUS\}$   
 $\{SNO\} \rightarrow \{STATUS\}$

---

现在，让我们回到3NF过程。特别是，让我们来看对于SJT例子，它如何工作。[\[2\]](#)只是提醒您，此关系变量有属性S、J和T；键{S,J}和{S,T}；且服从FD{T}→{J}。因此，存在下列FD：

---

$\{S, J\} \rightarrow \{T\}$

$$\{S,T\} \rightarrow \{J\}$$
$$\{T\} \rightarrow \{J\}$$

---

然而，很容易看出，这里的FD  $\{S,T\} \rightarrow \{J\}$  是冗余的（其实，当本章前面第一次讨论这个例子时，我实际上同样假设过），并且因此，其他两个FD一起形成一个不可约覆盖（下面把它称为C）：

---

$$\{S,J\} \rightarrow \{T\}$$
$$\{T\} \rightarrow \{J\}$$

---

现在，我们可以应用3NF过程。我们从标题的一个空集S开始，第二个步骤做两件事情：它把在C中具有相同左侧的FD收集在一起——实际上在例子中已经完成的一些东西——然后添加这些集合（实际上是标题）

---

$$\{S,J,T\}$$
$$\{T,J\}$$

---

到S中。第三个步骤没有任何效果，因为原始关系变量的每个属性现在至少包含S的一个元素。最后一个步骤也没有效果，因为S的  $\{S,J,T\}$  元素是原始关系变量的一个超键。因此，总的来

说，3NF过程告诉我们关系变量S能够以保持FD的方式无损分解为它在 $\{S,J,T\}$ 和 $\{T,J\}$ 上的投影。由此产生的要点如下所示。

■在 $\{S,J,T\}$ 上的投影当然是与原始关系变量相同的！换句话说，它是一个恒等投影（identity projection，见下一节），而且在这里并没有发生太多的分解。

■保持第二个关系变量（即在 $\{T,J\}$ 上的投影）和原始的关系变量实际上似乎没有什么必要，除非我们希望说，例如，Black（布莱克）教授教物理而不必在同一时间存在一些学生实际上由布莱克教授授课。如果我们不想要这种能力，我们可能不会希望保持第二个关系变量。因此，由3NF产生的分解不一定是一个推荐的分解，但再次重复，它是所有关系变量都属于3NF且所有FD都保持的一个分解。

我会把它留作一个习题（习题6.4）来显示对关系变量RX1、RX2、RX3应用3NF过程时会发生什么。同时，我想用关于BCNF的几句话结束本节。首先，我们可以添加其他（第5个）步骤到此3NF过程，如下所示：

5. 设Z是S的一个元素，使得关系变量R在属

性Z上的投影P不属于BCNF，设 $X \rightarrow Y$ 是P中存在的C的一个元素（即一个FD）；并设X不是P的一个超键，把S中的Z替换为（a）X和Y的并集及（b）Z和Y之间（按此顺序）的差集 $Z - Y$ 。为每个不同的Z和每个不同的X执行此步骤。

现在，把3NF过程应用到关系变量SJT产生一个由标题 $\{T, J\}$ 和 $\{S, J, T\}$ 组成的集合S。SJT在 $\{T, J\}$ 上的投影属于BCNF，但SJT在 $\{S, J, T\}$ 上的（恒等）投影则不属于BCNF，因为在后一个投影中存在FD $\{T\} \rightarrow \{J\}$ ，而 $\{T\}$ 不是一个超键。因此，我们应用第5步，删除标题 $\{S, J, T\}$ 并插入（a） $\{T\}$ 和 $\{J\}$ 的并集——但此插入有没有影响，因为该并集已经是S的元素——和（b） $\{S, J, T\}$ 和 $\{J\}$ 之间的差集，按照这个顺序。因此，最后S以下面的标题作为元素：

---

---

$\{S, T\}$   
 $\{T, J\}$

---

---

这些是SJT可以无损分解成的一组BCNF关系变量的标题（那些关系变量的键分别是 $\{S, T\}$ 与 $\{T\}$ ）。正如你所见到的，因此，往3NF过程中添加第5步将其转换成一个BCNF过程，但不能保证FD将保持。（当然，事实上，提供这样的保证是

不可能的，因为我们已经知道，BCNF和保持FD是相互冲突的目标。）但是，任何丢失的FD都是不能在不违反BCNF的情况下保持的。

其实，我们可以在某种程度上简化问题，直接到BCNF（即，绕过3NF）如下（它的输入与3NF过程的输入相同；即，它是由一个关系变量R和在R中存在的FD的一个不可约覆盖C组成的）。

1.初始化S，使它只包含标题R。

2.（与3NF过程的步骤2相同）设X是在C中的某个FD的左边部分（决定因素），设C中的FD及其左边部分X的完整集合是 $X \rightarrow Y_1, \dots, X \rightarrow Y_n$ ；并设 $Y_1, \dots, Y_n$ 的并集是Y。把X和Y的并集添加到S中。对每个不同的X都执行此步骤。

3.设Z是S的一个元素，使得关系变量R在属性Z上的投影P不属于BCNF，设 $X \rightarrow Y$ 是P中存在的C的一个FD；并设X不是P的一个超键，把S中的Z替换为（a）X和Y的并集及（b）Z和Y之间（按此顺序）的差集 $Z - Y$ 。对每个不同的Z和每个不同的X执行此步骤。



当此过程结束时，S的元素是一组BCNF关系变量的标题，R可以无损地分解，但不一定不会丢失FD。此外，为了备案，让我指出此过程并没有提及2NF或3NF。

[1]我给出此过程部分是由于历史的原因。如果你愿意，你可以跳过它。

[2]当然SJT已经属于3NF，但我们仍然可以对它应用此过程——我有我希望这样做的理由，这随后将成为明显的理由。

## 6.6 恒等分解

虽然这是与本章的主旋律有一点离题的，但我想简要地阐述一下恒等投影的概念。这里的定义（我针对关系变量定义它，但当然类似的概念也适用于关系）。

■定义：一个给定关系变量的恒等投影（identity projection）是关系变量在它的所有属性上的投影。

现在，这应该是显而易见的，任何关系变量都可以无损地，尽管平凡地，分解成它的恒等投影。然而，有些人根本不喜欢把这样的分解当作是一个分解，（正如我针对SJT示例说过的，“在这里并没有发生太多的分解”）。如果你碰巧是这类人中的一个，那么，你可能更喜欢通过以下方式观察这个问题。设关系变量R具有标题H，那么，在R中存在 $FD \{ \} \rightarrow \{ \}$ ，其中 $\{ \}$ 为属性的空集（当然，这个FD是平凡的，在每个关系变量中都存在），这肯定是真实的。因此，按照希思定理，设X、Y和Z分别为 $\{ \}$ 、 $\{ \}$ 和H，R可以无损分解成其投影R1和R2，其中：

1. R1的标题XY是 $\{ \}$ 和 $\{ \}$ 的并集，它可以化简成只是 $\{ \}$ ；即，R1是R在完全没有属性上的投

影，且它的值只有两种，如果R是空的，它是TABLE\_DUM，否则，它是TABLE\_DEE。[\[1\]](#)

2.R2的标题XZ是{}和H的并集，它可以化简成只是H；即，R2是R在它的所有属性上的投影，或者换句话说，是R的恒等投影。

我希望这是明确的，它的分解是无损的——R无疑等于R1和R2的连接。（另一方面，R1和R2的组合也确实不符合通常的要求，即要求在重建过程中，两个投影都应是不必要的）。

虽然这里的主题可能称为恒等分解（identity decomposition），但请让我说明任何关系变量也总是能分解（再次平凡地，但这次是“水平”分解，而不是“垂直”分解）为相应的恒等限制。[\[2\]](#)下面是一个定义（我再次针对关系变量定义它，但是类似的概念当然也适用于关系）。

■定义：一个给定的关系变量R的恒等限制（identity restriction）是R的任何限制条件恒为真的限制，换句话说，R的任何限制在逻辑上等价于以下形式之一：

---

---

R WHERE TRUE

---

---

注意：在逻辑上，恒为真的条件，如布尔表达式 $CITY=CITY$ ，称为同义反复（tautology）。因此，我们可以说，关系变量 $R$ 的恒等限制是限制条件为一个同义反复的任何限制。

顺便指出，任何给定的关系变量 $R$ 也总是有一个空的限制，我们可以这样表示它：

---

---

$R \text{ WHERE FALSE}$

---

---

一个给定的关系变量 $R$ 的恒等限制和空限制的（不相交的）并集当然恒等于 $R$ 。注意：在逻辑上，恒为假的条件，如布尔表达式 $CITY \neq CITY$ ，称为矛盾（contradiction），因此，我们可以说，关系变量 $R$ 的空限制是限制条件为一个矛盾的任何限制。[\[3\]](#)

[\[1\]](#)TABLE\_DUM和TABLE\_DEE分别是下面两种关系的昵称，即，没有属性也没有元组的唯一关系，以及没有属性但有一个元组的唯一关系。

（之前我们遇见过这些关系，在附录D中习题2.8的答案中）。关于它们的进一步讨论，请参阅《SQL与关系数据库理论》。

[\[2\]](#)关于限制（restriction）的精确定义和限制条件（restriction condition）的相关概念，请参阅附录

D中习题13.13的答案。

[3] 术语矛盾并不意味着与它在普通的语境中的含义在逻辑上是完全一样的，但就目前而言这种差异并不重要。

## 6.7 关于冲突的更多内容

要回到这一章的主旋律：到现在为止，我们已经看到了几个可能会丢失FD的例子。在这些例子中，我们可以通过小心来避免丢失FD，但在一种情况下（例如SJT），保持FD和BCNF分解的目标实际上是相互冲突的。因此，出现了一个显而易见的问题：我们是否能够描述真正有冲突的情况呢？答案是肯定的，事实上，这是容易做到的。

设R是我们正在处理的关系变量，并设C是在R中存在的FD的一个不可约覆盖。构造如下的一个FD图（FD graph）。

- 1.为R的每个属性都构造一个结点。

- 2.设 $X \rightarrow Y$ 是C中的一个FD。其中X涉及两个或更多个属性；构造一个“超级结点”，使它恰好包含在X中命名的属性的结点。（如果你在纸面上这样做，你可以画一个圈起个别属性结点的圆）。认为超级结点是结点。为C中具有复合决定因素的每个FD重复此步骤。

- 3.设 $X \rightarrow Y$ 是C中的一个FD。绘制从X的结点到Y的结点的有向弧，为C中的每个FD重复此步

骤。

4.当且仅当完成的图形包含任何回路（一个回路是从一个结点到其自身的有向弧序列）时，R才不能无损分解成BCNF投影而不丢失FD。

作为一个练习，尝试对本章前面讨论的各个例子应用上述过程。当你这样做的时候，你会很快明白在这里到底发生了什么（如果你还没有这样做）。详细说明：只有当关系变量包含的FD的模式与从SJT的例子中获得的模式类似时，才会有一个真正的冲突。

## 6.8 独立投影

我想回到本章开始的例子来结束这一章。只是提醒你，这个例子涉及把我们平常的供应商关系变量S无损分解到其在SNC ( $\{SNO, SNAME, CITY\}$ ) 和ST ( $\{SNO, STATUS\}$ ) 上的投影。那个分解丢失了FD  $\{CITY\} \rightarrow \{STATUS\}$ ，导致为了强制执行每个城市都有一个状态的约束，有时对其中任何一个投影做更新，需要更新另一个投影。相比之下，“明智”地分解成SNC ( $\{SNO, SNAME, CITY\}$ ) 和CT ( $\{CITY, STATUS\}$ ) 上的投影则没有这样的问题——对其中任何一个投影做更新都不需要考虑另一个投影。[\[1\]](#)

为了本次讨论，让我把类似分解成SNC和ST的分解称作坏的分解，而把类似分解成SNC和CT的分解称作好的分解。那么，正如我们已经看到的那样，一个好的分解中的投影可以相互独立地更新，因此，有时也明确地将这种投影称为独立投影 (independent projection)。相反，在相同的意义上，在一个坏的分解中的投影不是独立的。因此，我们可以说，为了保持FD，我们希望分解得到的各个技能都是独立的。Jorma Rissanen



提出的一个定理，可以在这方面有所帮助。但是，在我说明此定理之前，首先让我对两个投影是独立的含义给出一个准确定义。

■定义：关系变量R的投影R1和R2是独立的，当且仅当在R中存在的每一个FD在R1和R2的连接中也存在。

下面是此定理：

■Rissanen定理：设关系变量R具有标题H，它具有投影R1和R2，它们分别具有标题H1和H2；进一步，设H1和H2都是H的真子集，设它们的并集等于H，并设它们的交集是空的。[\[2\]](#)那么，当且仅当（a）其共同属性构成至少其中一个投影的超键，且（b）在R中的每个FD都被其中至少一个投影存在的FD蕴含时，投影R1和R2是独立的。

考虑把S分解为SNC和CT投影的“好”分解。这两个投影是独立的，因为（a）其共同属性的集合是{CITY}，且{CITY}是CT的一个（超）键，并且（b）在S中存在的每个FD或者在两个投影中的一个中存在或被在投影中存在的那些FD蕴含（请参阅第7章）。与此相反，考虑分解为SNC和ST投影的“坏”分解。这里的投影是不独立

的，因为 $FD\{CITY\} \rightarrow \{STATUS\}$ 不能由在那些投影中存在的那些FD推断出——虽然至少其共同属性的集合 $\{SNO\}$ ，确实同时是两个投影的键。

Rissanen在独立投影上所做的工作（在1977年做的，或至少在那时发表的）为我们奠定了现在所说的保持FD的理论基础，这可以说是有历史意义的贡献。

[1]除了可能在CT的 $\{CITY\}$ 和SNC的 $\{CITY\}$ 之间有一个外键约束，甚至是一个相等依赖。

[2] $H_1$ 和 $H_2$ 的交集不为空这个条件在Rissanen定理的原始陈述中存在，但似乎是不必要的。

## 习题

6.1 来自6.1节的关系变量SJT服从 $FD\{S,J\} \rightarrow \{T\}$ 。如果我们把SJT分解为其在 $\{T,J\}$ 上的投影TJ和在 $\{T,S\}$ 上的投影TS，请用Tutorial D语法书写一个约束的声明来表达取代此FD的多关系变量约束。

6.2 （参见本章正文）假设把来自6.3节的关系变量R X2A'分解成其在 $\{CLASS,STATUS\}$ 和 $\{CITY,STATUS\}$ 上的投影。正如这一节注明的，现在必须单独说明并强制执行一个适当的多关系变量约束。这个约束是什么样子的呢？

6.3 以下关系变量用来代表一组美国地址：

---

$ADDR\{STREET,CITY,STATE,ZIP\}$

---

一个典型的元组可能类似如下所示（Tutorial D语法）：

---

$TUPLE\{STREET'1600 Pennsylvania Ave.',$   
 $CITY'Washington', STATE'DC', ZIP'20500'\}$

---

并非完全不合理地假设，在这个关系变量中

存在如下FD，且它们是不可约的：

---

$$\{\text{STREET,CITY,STATE}\} \rightarrow \{\text{ZIP}\}$$
$$\{\text{ZIP}\} \rightarrow \{\text{CITY,STATE}\}$$

---

你会如何分解这个关系变量？

6.4 请显示对来自本章正文的关系变量RX1、RX3和RX2（注意这里的序列）应用3NF过程的效果。

6.5 下面是一个谓词：明星S在电影M中扮演角色R，这部电影是导演D执导的并在Y年发布的；进一步，明星S出生在日期B，并因此有星座Z和中国生肖C，且Z和C共同确定了S的星座运势H。请给出一个捕捉上述状况的FD集合。并请说明你根据可能有效的“业务规则”作出的任何假设。此外，应用BCNF过程来获得一个适当的BCNF关系变量集合。该过程是否丢失了任何FD呢？

## 第7章 FD公理化

[The]true and solid and living axioms.

——Francis Bacon: The New Organon

我已经指出一些FD蕴含着另外的FD这一点触动多次了，现在该是获得更具体的知识的时候了。但是，首先，我要介绍一些符号表示法，因为：（a）减少了在正式的证明及诸如此类的东西中所需的按键，（b）有时也可以有助于宛如看见森林同时看见树木。

你可能还记得，第5章阐述的希思定理包括下面的句子：设 $XY$ 表示 $X$ 和 $Y$ 的并集， $XZ$ 也表示类似的含义。我要为大家介绍的符号表示法基本上是这个简单想法的扩展（虽然它有点不合逻辑，但是它非常方便）。具体地说，这种表示法使用 $XY$ 形式的表达式来表示：

■ $\{X\}$ 和 $\{Y\}$ 的并集，如果 $X$ 和 $Y$ 分别表示单独的属性（即，它们是单独的属性名）。

■ $X$ 和 $Y$ 的并集，如果 $X$ 和 $Y$ 表示属性的集合（即，它们是属性名的集合）。

如果X表示单个属性，它也允许{X}缩写为仅X（例如，在一个FD中）。注意：为了方便起见，我将从此刻开始把这种符号表示法称为希思符号（Heath notation）。

## 7.1 阿姆斯特朗公理

我们已经看到，从形式上来说，一个FD只是一个形式为 $X \rightarrow Y$ 的表达式，其中X和Y都是集合（实际上是属性名称的集合，但是从形式上看，集合包括什么真的无所谓）。现在，让我们假设我们得到某个FD集合（比如，F）。然后，我们可以应用某种形式的推理规则（inference rules）从F中的FD来推导进一步的FD——F中的FD蕴含的FD，这意味着，如果F中的FD在某个关系变量R中存在，那么派生的FD也在R中存在。所涉及的规则由阿姆斯特朗在1974年首次提出，因此，这些规则通常称为阿姆斯特朗推理规则

（Armstrong's inference rules）或（更常见）阿姆斯特朗公理（Armstrong's axioms）。它们可以用各种等价的方法来表示，其中以下几个规则可能是最简单的。

1. 如果Y是X的一个子集，则 $X \rightarrow Y$ （“自反律”，reflexivity）。

2.如果 $X \rightarrow Y$ ，则 $XZ \rightarrow YZ$ （“增广律”，augmentation）。

3.如果 $X \rightarrow Y$ 和 $Y \rightarrow Z$ ，则 $X \rightarrow Z$ （“传递律”，transitivity）。

注意，根据一个FD的预期解释，这些规则直观上是合理的。也就是说，因为我们知道FD是什么“意思”，所以我们可以很容易地看出，例如，如果FD  $X \rightarrow Y$ 和 $Y \rightarrow Z$ 都在关系变量R中存在，那么FD  $X \rightarrow Z$ 一定也在关系变量R中存在。注意：供应商关系变量S说明了这个特定的规则——FD $\{SNO\} \rightarrow \{CITY\}$ 和 $\{CITY\} \rightarrow \{STATUS\}$ 都在这个关系变量中存在，所以FD $\{SNO\} \rightarrow \{STATUS\}$ 也在这个关系变量中存在。

因此这样的规则是合理的。但更重要的是，它们都健全（sound）且完备（complete）。健全性和完备性是在一般的形式系统中经常遇到的概念。在这里考虑的形式系统中，这是它们的含义。

■完备性（completeness）：如果FD  $f$ 由给定的集合F中的FD蕴含，那么它可以由F中的FD通过规则推导得出。（再次重复，说某个FD  $f$ 是由在某个集合F中的FD蕴含的意思是，如果F中的

FD都存在，则f也存在）。

■健全性（soundness）：如果FD  $f$ 不由给定的集合F中的FD蕴含，那么它不能由F中的FD通过规则推导得出。[\[1\]](#)

这些规则从而形成了所谓的FD的公理化（axiomatization）。其结果是，它们可以用于推导任何给定FD的集合F的所谓的闭包 $F^+$ 。下面是一个定义。

■定义：设F是一个FD集合。那么，F的闭包（closure） $F^+$ 是由F中的那些FD所蕴含的所有FD的集合。

更重要的是，这个推导过程可以机械化（mechanized），也就是，阿姆斯特朗规则可以纳入（例如）一个设计工具中，这样已知在某个关系变量R中存在的FD的一个集合F，将能够计算那个集合F的闭包 $F^+$ ，或换句话说，计算出在该关系变量中存在的所有FD的一个完整集合。这个事实的意义应该是显而易见的。

[\[1\]](#)如果你有逻辑学的背景，你可能会喜欢如下描述：健全性意味着所有的定理都是恒真命题（tautology）；完整性意味着所有的恒真命题都



是定理。或者更直观地（并且感谢Hugh Darwen）：健全性意味着如果你能证明这一点，那么它是真实的，而完整性意味着如果它是真实的，那么你可以证明这一点。

## 7.2 附加规则

可以由原来的3个规则推导出以下几个附加的推论规则。这些附加的规则可以用来简化从F计算F<sup>+</sup>的实际任务，下面是一些例子。

4.  $X \rightarrow X$  (“自含律”， self determination)。

5. 如果  $X \rightarrow Y$  和  $X \rightarrow Z$ ，则  $X \rightarrow YZ$  (“合并律”， union)。

6. 如果  $X \rightarrow Y$  和  $Z \rightarrow W$ ，则  $XZ \rightarrow YW$  (“复合律”， composition)。

7. 如果  $X \rightarrow YZ$ ，则  $X \rightarrow Y$  和  $X \rightarrow Z$  (“分解律”， decomposition)。[\[1\]](#)

接下来的一节将展示如何从原来的3个规则推导出这4个规则。但是，首先让我举两个例子来显示这些规则（原始的及/或附加的）可以如何使用。第一个例子，假设我们有一个关系变量R，它具有属性A、B、C、D、E、F，并且我们得知，在这个关系变量中存在下面的FD：

$A \rightarrow BC$

$$B \rightarrow E$$

$$CD \rightarrow EF$$

我现在将显示在R中还存在FD  $AD \rightarrow F$ （我想你会同意，这不是一个一眼就能看出事实）。<sup>[2]</sup>它的证明如下：

1.  $A \rightarrow BC$ （已知）

2.  $A \rightarrow C$ （对1应用分解律）

3.  $AD \rightarrow CD$ （对2应用增广律）

4.  $CD \rightarrow EF$ （已知）

5.  $AD \rightarrow EF$ （对3和4应用传递律）

6.  $AD \rightarrow F$ （对5应用分解律）

对于第二个例子，回顾一下第6章中不可约覆盖的概念。只是为了提醒你，（a）对于一个给定的FD集合F，它的一个覆盖是一个FD集合C，使得在F中的每一个FD都由C中的那些FD蕴含，且（b）当且仅当它是具有下列所有性质的一个集合时，此覆盖C才是不可约的。

1.单例依赖因素（singleton dependant）：C中的每个FD的右侧只有一个属性。

2.不可约的决定因素（irreducible determinant）：左侧的任何属性都不可以在不丢失C是F的一个覆盖这个性质的情况下去掉。

3.没有冗余的FD：任何FD都不可以在不丢失C是F的一个覆盖这个性质的情况下从C中去掉。

现在，我假设在第6章中（默认）的每一个FD集合F都有一个不可约覆盖。其实，这是很容易看到的。

■根据分解律，不失一般性，我们可以假设F中的每一个FD都有一个单例右侧的形式。

■接下来，对于F中的各个FD，检查其左侧的每个属性A，如果从左侧删除A对闭包 $F^+$ 没有影响，那么就删除A。

■对于保留在F中的每个FD，如果从F中删除此FD对闭包 $F^+$ 没有影响，那么就删除此FD。

最终版本的F是不可约的，并且是原始版本的F的一个覆盖。

下面用一个具体例子来说明这个实际找到一个不可约覆盖的过程。让我们假设FD集合（它的元素是在某个具有属性A、B、C、D的关系变量的R中所有可能存在的FD）如下：

$$A \rightarrow BC$$

$$B \rightarrow C$$

$$A \rightarrow B$$

$$AB \rightarrow C$$

$$AC \rightarrow D$$

那么对于这个给定集合，下面的过程会产生它的一个不可约覆盖。

1.首先，把FD改写为每个FD都具有一个单例右侧的形式：

$$A \rightarrow B$$

$$A \rightarrow C$$

$$B \rightarrow C$$

$$A \rightarrow B$$

$$AB \rightarrow C$$

$$AC \rightarrow D$$

注意，既然FD  $A \rightarrow B$ 出现了两次，那么其中一次出现可以去掉。

2. 可以从FD  $AC \rightarrow D$ 的左侧去掉属性C，这是因为已知 $A \rightarrow C$ ，所以根据增广律可得 $A \rightarrow AC$ ，且已知 $AC \rightarrow D$ ，所以根据传递律可得到 $A \rightarrow D$ ；因此在 $AC \rightarrow D$ 左侧的C是多余的。

3. FD  $AB \rightarrow C$ 可以去掉，这同样是因为已知 $A \rightarrow C$ ，所以根据增广律可得到 $AB \rightarrow CB$ ，再根据分解律可得到 $AB \rightarrow C$ 。

4. FD  $A \rightarrow C$ 是由FD  $A \rightarrow B$ 和 $B \rightarrow C$ 蕴含的，因此它可以去掉。

于是我们只剩下：

$$A \rightarrow B$$

$$B \rightarrow C$$

$A \rightarrow D$

这个集合是不可约的。

[1]两个要点：首先，不要将这种分解与本书其他地方所讨论的无损分解混淆。其次，注意，在这里所定义的复合和分解不是完全互逆的；具体地说，分解的逆是复合的一种特例，其中Z由X替换而W由Z替换。

[2]如果你喜欢一个更具体的例子，设A为员工编号，B为部门编号，C为经理的员工编号，D为经理（唯一的经理）管理的项目的项目编号，E为部门名称，F为经理在特定的项目上花费的时间的百分比。那么FD  $A \rightarrow BC$ 、 $B \rightarrow E$ 、 $CD \rightarrow EF$ 直观上都是合理的。（ $AD \rightarrow F$ 呢？）

## 7.3 证明附加规则

为了兑现前面的承诺，本节将展示如何从原始的规则1~3推导出规则4~7。

4.  $X \rightarrow X$ （“自含律”）。

证明：根据自反律立即可得。

5. 如果  $X \rightarrow Y$  且  $X \rightarrow Z$ ，则  $X \rightarrow YZ$ （“合并律”）。

证明：已知  $X \rightarrow Y$ ，因此根据增广律可得到  $X \rightarrow XY$ ；同样，已知  $X \rightarrow Z$ ，因此根据增广律可得到  $XY \rightarrow YZ$ ；再根据传递律可得到  $X \rightarrow YZ$ 。

6. 如果  $X \rightarrow Y$  且  $Z \rightarrow W$ ，则  $XZ \rightarrow YW$ （“复合律”）。

证明：已知  $X \rightarrow Y$ ，因此根据增广律可得到  $XZ \rightarrow YZ$ ；同样，已知  $Z \rightarrow W$ ，因此根据增广律可得到  $YZ \rightarrow YW$ ，再根据传递律可得到  $XZ \rightarrow YW$ 。

7. 如果  $X \rightarrow YZ$ ，则  $X \rightarrow Y$  且  $X \rightarrow Z$ （“分解律”）。

证明：已知  $X \rightarrow YZ$  且根据反身律可得到



$YZ \rightarrow Y$ ，因此根据传递律可得到  $X \rightarrow Y$ （同理可得到  $X \rightarrow Z$ ）。

## 7.4 另一种闭包

总之，一个FD集合 $F$ 的闭包 $F^+$ 是 $F$ 中的那些FD蕴含的所有FD的集合。现在原则上，我们可以通过反复应用阿姆斯特朗规则（和/或由此衍生的规则）从 $F$ 计算 $F^+$ ，直到它们不再产生新的FD。然而，在实践中，很少需要计算 $F^+$ 本身（或许，这也只是因为刚刚概述的这个过程是非常低效的）。但现在我想展示我们如何能计算 $F^+$ 的某个特定子集：即，该子集包括关于一个给定的决定性因素的所有FD。更确切地说，我将展示，已知一个标题 $H$ ， $H$ 的一个子集 $Z$ 和关于 $H$ 的FD集合 $F$ ，我们如何计算出 $Z$ 在 $F$ 下的所谓闭包 $Z^+$ 。下面是它的一个定义。

■定义：设 $H$ 是一个标题，设 $F$ 是一个关于 $H$ 的FD集合，并设 $Z$ 是 $H$ 的一个子集，那么 $Z$ 在 $F$ 下的闭包 $Z^+$ 是使得 $Z \rightarrow C$ 被 $F$ 中的FD蕴含的 $H$ 的最大子集 $C$ 。

顺便说一下，请注意，我们现在有两种不同的闭包（尽量不要把它们混淆）：一个FD集合的闭包，以及在一个FD集合中的一个属性集合的闭包。[\[1\]](#)还请注意，我们对于这两种闭包都使用相同的“上标加号”符号。

下面是计算 $Z$ 在 $F$ 下的闭包 $Z^+$ 的一个简单的伪代码算法：

---

```
 $Z^+ := Z;$   
do "永远";  
for  $F$  中的每个 FD  $X \rightarrow Y$   
do;  
if  $X$  是  $Z^+$  的一个子集  
then 把  $Z^+$  替换为  $Z^+$  和  $Y$  的并集;  
end;  
if  $Z^+$  在本次迭代没有发生改变  
then 退出; /* 计算完成 */  
end;
```

---

让我们做一个例题。假设给定的标题是 ABCDEG，且我们要计算属性集合 AB 在以下 FD 集合  $F$  下的闭包  $AB^+$ ：

$A \rightarrow BC$

$E \rightarrow CG$

$B \rightarrow E$

$CD \rightarrow EG$

现在，让我们逐步查看算法：

1.首先，我们把结果 $AB^+$ 初始化为属性集合AB。

2.现在，我们执行内循环4次，即对给定的FD中的每个FD都执行1次。在第1次迭代（对于FD  $A \rightarrow BC$ ）中，我们发现决定因素A确实是到目前为止计算的 $AB^+$ 的一个子集，所以我们往结果中添加属性B和C。 $AB^+$ 现在是集合ABC。

3.在第2次迭代（对于FD  $E \rightarrow CG$ ）中，我们发现决定因素E不是到目前为止的计算结果的子集，从而 $AB^+$ 保持不变。

4.在第3次迭代（对于FD  $B \rightarrow E$ ）中，我们往 $AB^+$ 中添加E，因此，它现在的值是ABCE。

5.在第4次迭代（对于FD  $CD \rightarrow EG$ ）中， $AB^+$ 保持不变。

6.现在我们再次执行内循环4次。在第1次迭代中，结果保持不变，在第2次迭代中，它扩展为ABCEG，在第3次和第4次迭代中，它仍然保持不变。

7.现在我们再次执行内循环4次。结果保持不变，所以整个过程结束，结果是 $AB^+ = ABCEG$ 。

那么，我希望能从这个例子中看出，已知H、F和Z，计算 $Z^+$ 本质上是简单的。最重要的事情是这样的：已知某个FD集合F（关于某个标题H），我们可以很容易地分辨某个特定的FD  $X \rightarrow Y$ （关于相同的标题H）是否由F蕴含，因为当且仅当Y是X在F下的闭包 $X^+$ 的一个子集时，才会如此。换句话说，我们现在有了一个不必实际计算 $F^+$ ，而确定一个给定的FD  $X \rightarrow Y$ 是否在F的闭包 $F^+$ 中的简单方法。

从一个属性集合的闭包的定义也可以得出，一个关系变量R的超键正是R的标题的那些子集SK，其中SK在相关的FD集合下的闭包 $SK^+$ 是R的整个标题。

[1]更不用说适用于关系代数操作的那种闭包。

## 习题

7.1 说阿姆斯特朗规则是健全的是什么意思？说它是完备的又是什么意思？

7.2 什么是一个FD集合的闭包？请显示在发货关系变量SP中存在的FD集合的闭包。

7.3 请根据满足FD的含义的定义，显示反身律、增广律和传递律是合理的。

7.4 （请在回溯到本章正文的条件下尝试这个习题）请证明上个习题中的3个规则蕴含着自含律、合并律、复合律和分解律。

7.5 下面的定理<sup>[1]</sup>是Hugh Darwen（休·达尔文）提出的：<sup>[2]</sup>

如果 $X \rightarrow Y$ 且 $Z \rightarrow W$ ，那么 $XV \rightarrow YW$ ，这里 $V = Z - Y$ 。

证明这个定理。你用了来自上两个习题中的哪些规则？来自这些习题的哪些规则可以作为此定理的特例推导出呢？

7.6 请找出下面的FD集合的一个不可约覆

盖:

$$AB \rightarrow C \quad BE \rightarrow C$$

$$C \rightarrow A \quad CE \rightarrow FA$$

$$BC \rightarrow D \quad CF \rightarrow BD$$

$$ACD \rightarrow B \quad D \rightarrow EF$$

7.7 考虑下面的FD:

$$A \rightarrow B$$

$$BC \rightarrow DE$$

$$AEF \rightarrow G$$

请问FD  $ACF \rightarrow DG$ 被上面的集合蕴含吗?

7.8 当且仅当每一个集合都是另一个集合的一个覆盖时, 两个FD集合是等价 (equivalent) 的。请问下列集合等价吗?

---

$$\{A \rightarrow B, AB \rightarrow C, D \rightarrow AC, D \rightarrow E\}$$
$$\{A \rightarrow BC, D \rightarrow AE\}$$

---

请注意，对于任何给定的一个FD集合F，如果一个集合C是F的一个不可约的覆盖，那么F肯定等价于C，进一步讲，当且仅当两个集合具有相同的不可约覆盖时，它们才是等价的。

7.9 设关系变量R具有属性A、B、C、D、E、F、G、H、I和J，并服从以下FD：

$$ABD \rightarrow E \quad C \rightarrow J$$

$$AB \rightarrow G \quad CJ \rightarrow I$$

$$B \rightarrow F \quad G \rightarrow H$$

这个集合是可约的吗？R有哪些键呢？

[1]也称为通用一致性定理，见《数据库系统导论（原书第8版）》第11章。

[2]Hugh Darwen: “The Role of Functional Dependence in Query Decomposition,” in C.J.Date and Hugh Darwen, Relational Database Writings 1989-1991 (Addison-Wesley, 1992) .



## 第8章 反规范化

What's normal, anyway?

——Anon.: Where Bugs Go

我想说几句关于反规范化的话。在本书中，至今为止，我还没有考虑（至少没仔细考虑）任何高于BCNF的规范化级别。但反规范化，哪怕有一丁点儿意义，也不能只专门地适用于BCNF，我的意思是，它不能只是专门地回落到低于BCNF的某规范化级别，而是，它必须意味着，从任何给定的规范化级别回落到某个较低的级别。

话虽如此，但是，我也需要说属于BCNF，而不属于某个更高的范式的关系变量是相当不寻常的（虽然不是完全未知的，我要赶紧补充）。因此，在实践中，反规范化确实通常专门指回落到某个BCNF以下的规范化级别，这就是本书包含本章的原因。


### 8.1 “反规范化是为了性能”吗

自从SQL产品第一次上市以来，有必要“为了性能而反规范化”的说法已广泛宣传。（似是而

非的)支持这个说法的是类似这样的一些论据:

- 1.规范化意味着很多的关系变量。
- 2.很多的关系变量意味着大量的存储文件。
- 3.大量的存储文件意味着大量的I/O。

例如,在供应商和零件的案例中,要求获取供应红色零件的供应商详情涉及两个二元连接,也许第一个是供应商和发货的连接,第二个是第一个连接的结果和零件的连接。如果这3个关系变量对应3个物理上独立的存储文件,那么这两个连接将需要大量的I/O,因此性能不佳。

正如已经指出的那样,这种说法是似是而非的,至少在原则上。究其原因,是关系模型并没有规定关系变量必须一一映射到存储文件。例如,在供应商和零件的案例中,没有合乎逻辑的理由,对于为什么我们不能把3个关系变量的连接物理地(可能甚至是冗余地)存储为磁盘上的单个存储文件, 这可能明显减少正在考虑的查询的I/O。但是,这一点与目前的目的是不相关的,这是因为以下原因。

■首先,这是大多数DBMS厂商已经让我们严

重失望的一个领域，大多数SQL产品确实将关系变量一一映射到存储文件，差不多全部如此。<sup>[2]</sup>即使是例外者（指不在上述之列的厂商），也未能向我们提供我们所期待的那样的数据独立性，或关系系统在理论上应有的独立性。因此，遗憾的是，作为一个实际问题，这个“似是而非”的说法对当今大多数SQL产品是有效的。

■其次，即使关系变量没有一一映射到存储文件，反规范化可能在存储文件级别仍然是理想的。事实上，不一一映射可取的一个重要原因正是它们将允许在它所属的物理层反规范化，而不必通过逻辑层显示出来，从而（避免）恶化。

所以我会认为，出于讨论的目的，有时确实不得不在一定级别或其他级别上做反规范化。但是，反规范化是什么呢？

<sup>[1]</sup>当然，我在这里说得很轻率。特别是，我忽略了一些供应商或一些零件没有相应的发货的可能性。

<sup>[2]</sup>我知道从关系变量到存储文件的映射并不总是如我在这里假设的那样完全是一对一的，例如，一些产品允许几个关系变量共享相同的存储文件，而有些则允许一个单独的关系变量跨越几个存储文件。但是这些事实不会显著影响整体状

况，在这里为简单起见，忽略它们。

## 8.2 反规范化是什么意思

奇怪的是，虽然这种做法被如此广泛地提倡，但关于反规范化的实际组成似乎是相当混乱的。（教科书也没有太大的帮助，即使是那些专门针对数据库设计主题的书，其中大部分甚至没有提到它，而那些提到它的又很少提供它的定义，并且肯定不会非常深入地讨论这个问题）。例如，前一段时间，我有机会读了一篇专门致力于在商业SQL产品中的反规范化问题的论文。<sup>[1]</sup>在下面，我会把该论文引用为“反规范化论文”。不过，论文作者一开始就反对反规范化。引用如下：

我认为规范化的原则应被视为戒律……除非你面对的性能问题是金钱、硬件可扩展性、当前的SQL技术、网络优化、并行化或其他性能技术都解决不了的（略做改写，添加粗体）。

我完全同意这一立场。事实上，我记得自己说过很多同样内容的话：我于1990年写了一篇关于SQL系统在实践中使用的论文<sup>[2]</sup>，其中写道“仅当其他一切都失败时”，我才把反规范化作为一个性能技巧推荐。然而，遗憾的是，“反规范化论文”的其余部分往往表明，作者真的不知道什么是反规范化；在上面引述的公开声明之后，该

论文继续给出了一些“为性能而设计”的8个例子，但除了其中一个外，所有例子都完全与反规范化绝对无关！

但是，我得替作者说句公道话，重申一遍：似乎很难在文献中找到反规范化的一个确切定义。当然，人们可能认为这样的定义是没必要的，（a）反规范化，无论它是什么东西，它都一定是规范化的逆操作，并且（b）反过来说，规范化肯定是精确定义了的。然而，为了记录在案，我马上会提供一些反规范化的精确定义可能看起来像什么观点。然而，在我这样做之前，首先让我说清楚，我没有与反规范化论文中提出的具体设计技巧针锋相对的意思，事实上，早在1982年我写的一篇论文中，我自己就提出了其中几种相同的技巧。[\[3\]](#)我唯一争论的是它把它们专门作为反规范化技巧的事实。

所以这里是我自己的定义，这是非常值得的（如果它看起来有点冗长，我为此道歉）。我开始于规范化关系变量R意味着通过下列方法减少冗余这一观察。

- 1.将R用投影 $R_1, \dots, R_n$ 的集合替代，并且在 $R_1, \dots, R_n$ 中，至少有一个的规范化级别比R更高。

2.对于R所有可能的值 $r$ ，如果把 $R_1, \dots, R_n$ 相应的值 $r_1, \dots, r_n$ （分别）再次重新连接在一起，那么那个连接的结果等于 $r$ 。

因此，得到下面的定义。

■定义：反规范化关系变量 $R_1, \dots, R_n$ 的一个集合，意味着通过下列方法增加冗余：

1.把 $R_1, \dots, R_n$ 由它们的连接 $R$ 替代，其中 $R$ 的规范化级别至少比 $R_1, \dots, R_n$ 中的一个更低。

2.对于 $R_1, \dots, R_n$ 所有可能的值 $r_1, \dots, r_n$ （分别）， $R$ 对应的值 $r$ 在 $R_i$ 的属性上的投影的结果等于 $r_i$ （ $i=1, \dots, n$ ）。

由此产生的要点如下所示。

■注意，反规范化是一个应用于一个关系变量集合的过程，而不是孤立地考虑单个关系变量的过程。例如，考虑分别具有标题 $\{SNO, SNAME, CITY\}$ 和 $\{CITY, STATUS\}$ 的关系变量 $SNC$ 和 $CT$ （一些示例值见图3-2）。这两个关系变量属于BCNF。如果我们将它们连接在一起，我们将得到供应商关系变量 $S$ （这只属于

2NF, 不属于3NF, 因此也不属于BCNF), 因此关系变量S可以看作关系变量SNC和CT的一个反规范化。当然, 更重要的是, 关系变量S比关系变量SNC和CT包含更多的冗余。

■如果 (a) 通过对一开始的R执行投影获得了R<sub>1</sub>, ..., R<sub>n</sub>, 换句话说, 如果反规范化真的是较早的规范化的反向操作 (姑且这么说), 比如上一段所述的供应商的例子, 并且 (b) 做较早的规范化纯粹是为了减少冗余, 而不是修复逻辑上不正确的设计 (见第3章中这两种可能性之间的区别的评论), 那么 (c) 对R的所有可能的值r, r在R<sub>i</sub>的属性上的投影必须得到r<sub>i</sub> (i=1, ..., n) 的需求, 将自动满足。

支持反规范化的论点基本上是, 它使检索更容易表达, 并且使它们的执行 (性能) 更好。[\[4\]](#)我会在后面的章节审查这种说法在何种程度上可能是有效的。但是, 我首先想指出的是, 一旦我们做出反规范化的决定, 那么我们已经走上了一个非常危险的斜坡。现在的问题是: 我们在哪里停止呢? 目前的情况与规范化不同, 后者对于持续的过程有清晰的逻辑理由, 直到我们达到可能的最高范式。这样, 我们是否能够得出这样的结论: 我们应该持续反规范化, 直到我们达到尽可能低的范式? 当然不是, 但对于决定过程应该停



止的确切位置并没有逻辑的标准。换句话说，在选择反规范化时，我们已经放弃了一个至少有坚实的科学和逻辑理论支持作后盾的立场，取而代之的是一个本质上纯粹务实的立场（通常，也基于对整个问题的有些狭隘的视角）。

[1] Sam Hamdan: “Denormalization and SQL-DBMS,” SQL Forum 4, No.1 (January/February 1995)。

[2] “SQL Dos and Don'ts,” in Relational Database Writings 1985-1989 (Addison-Wesley, 1990)。

[3] “A Practical Approach to Database Design,” in Relational Database: Selected Writings (Addison-Wesley, 1986)。

[4] 有时也声称反规范化使数据库更容易理解。习题8.2将解决这一具体问题。

## 8.3 什么不是反规范化（I）

我已经说过了，反规范化意味着增加冗余。不过，这并不表示增加冗余就意味着反规范化！这是反规范化论文落入的一个陷阱；它描述的设计手段确实增加了冗余（通常情况下），但它们不是（如前面提到的，一个唯一的例外）反规范化本身的应用。（在逻辑上，如果 $p$ 蕴含 $q$ 成立，不能因此得出 $q$ 蕴含 $p$ 成立的结论，如果否认这一点就是一个众所周知的错误推理的例子：事实上，它是如此众所周知，以至于享有一个特殊的名字，虚假转换的谬论（Fallacy of False Conversion））。

让我们来看看几个来自反规范化论文的例子。在第一个例子中，假设关系变量ITEM和SALES看起来像这样：

---

```
ITEM{INO,INAME}  
KEY{INO}  
SALES{SNO,INO,QTY}  
KEY{SNO,INO}  
FOREIGN KEY{INO}REFERENCES ITEM
```

---

谓词分别是“物品INO具有名称INAME”以及“数量QTY的物品INO在商店SNO销售”。出于

性能方面的考虑，该论文建议添加一个TOTAL\_QTY属性到关系变量ITEM中，对于任何给定的物品，其值是该物品在所有商店的总销售额。不过，虽然得出的设计确实包含一些冗余，但事实仍然是两个关系变量都仍然属于BCNF（请特别注意，在修订版本的关系变量ITEM中存在FD{INO} $\rightarrow$ {TOTAL\_QTY}）。换句话说，在这个例子中没有如前所述的反规范化。

第二个例子涉及该论文所谓的“内部数组”：

---

```
EMP {ENO, JAN_PAY, FEB_PAY, ....., DEC_PAY}  
KEY {ENO}
```

---

它的谓词是员工ENO1月收入金额JAN\_PAY, ....., 12月收入金额DEC\_PAY。据推测，虽然该论文没有尽可能明确地说，这个“元组方式”的设计是为了对照（并且由于性能方面的原因，可能优于）下面的“属性方式”模型：

---

```
EMP {ENO, MONTH, PAY}  
KEY {ENO, MONTH}
```

---

但两者的设计都属于BCNF。同样，这里也没有反规范化；其实，先提前透露一点后面的内

容（参见第15章），我会说，也没有增加冗余。（另一方面，原来的“元组方式”的设计可能是糟糕的，如果你考虑查询“获得至少1个月薪酬不到5000美元的员工信息，并加上所涉及的月份”，你就会看到它的坏处。）

然而，另一个例子涉及把一个关系变量 RESELLERS“水平”分解为两个独立的关系变量（ACTIVE\_RESELLERS和 INACTIVE\_RESELLERS）。换句话说，原始关系变量通过限制（而不是投影）分解，并通过合并（而不是连接）两个限制重建。因此，在这里我们显然根本不是在说传统意义上的规范化，更不是在说传统意义上的反规范化。[\[1\]](#)

我将给出另一个来自反规范化论文的例子。这个例子以如下STORE和EMP关系变量开始：

---

```
STORE{SNO,REGION,STATE, .....}  
KEY{SNO,REGION,STATE}  
EMP{ENO,SNO,REGION,STATE, .....}  
KEY{ENO}  
FOREIGN      KEY{SNO,REGION,STATE}REFERENCES  
STORE
```

---

谓词是SNO商店位于STATE州内的REGION

区域并且ENO员工在位于STATE州内的REGION区域的SNO商店工作。冗余是显而易见的，因此建议为商店引入一个替代标识符，比如SID，从而修改设计为如下内容：

---

```
STORE{SID,SNO,REGION,STATE, .....}  
KEY{SID}  
KEY{SNO,REGION,STATE}  
EMP{ENO,SID, .....}  
KEY{ENO}  
FOREIGN KEY{SID}REFERENCES STORE
```

---

但修改后的设计不仅不涉及反规范化，而且这实际上减少了冗余！[\[2\]](#)因为，一个特定的SNO与REGION和STATE的关联现在只出现一次，而不是为所涉及的商店的每一位员工存储一次。

（详细地说，它显然不是反规范化，因为，撇开其他的事情不谈，大家肯定都同意的一件事是，反规范化应该增加冗余）。

顺便说一下，我知道这最后一个例子可能给人的印象是：我认为代理键是一个好主意。然而，可悲的是，它们并非总是一个好主意。事实是，代理键，虽然它们可能会解决一些问题，但也可能引入它们自己的更多问题。要了解进一步的讨论，请参阅习题8.3及第15章。

在结束本节时，我想要很明确地表明，上述讨论并非要希望抨击反规范化论文或其作者。事实上，引自该论文的以下陈述应该明确地表明，在更大的问题上，该作者与我实际上是持同一立场的：

[我们应该]停止批评关系模型，并明确区分什么是SQL，什么是关系.....这两者是完全不同的。

我完全同意这个立场，也同意我们要在逻辑层面担心这些反规范化事项的唯一原因是当今的SQL产品在这方面的失败。正如我在其他地方指出的，其实在一个理想的系统中，[\[3\]](#)在逻辑层面，我们将永远不会有反规范化。即使在通常远远不够理想的当今的系统中，我仍然相信我们应该把反规范化只作为最后的手段。也就是说，只有当所有其他提高性能的策略出于某种原因都不符合需求时，我们才应该从一个完全规范化的设计回退。（当然，我会在这里赞同通常规范化有性能影响的假设，因为通常情况下，这种影响在当前的SQL产品中确实存在。）

[\[1\]](#)这是真实的，有可能在限制和并集的基础上，而不是在投影和连接的基础上（我会在本书第四部分关于这种可能性进行更详细的阐述）定义一

种新的规范化。如果我们这么做了，那么，我想我们也就有一种新的反规范化。不过，我敢肯定，这样的考虑，不是反规范化论文用 RESELLERS 的例子说明的东西。

[2] 或者不是这样？同样，请参见第15章，其中专门包括对代理键的使用的进一步讨论。

[3] 例如，在我的书《数据库系统导论（原书第8版）》中。如需进一步讨论，请参阅我的书《Go Faster! The TransRelationaltm Approach to DBMS Implementation》（Ventus Publishing, 2002, 2011）。

## 8.4 什么不是反规范化（II）

到目前为止，在本章中，我已经对我认为的反规范化是什么给出了一个合理的定义，并且我也已经给出了一些关于它不是什么的例子。不过，觉得这个术语应该用在任何一种精确的或合乎逻辑的意义上，也许只是我的一个错误。当然，总体而言，在业界，它的使用非常不精确，事实上，它似乎用来——特别是在数据仓库环境中——指可以看作糟糕的设计实践的任何事情。更重要的是，所涉及的做法往往是明确推荐的！（即是由那些以这种方式说话的人提出的）。这种不良做法的例子包括：

- 使用重复的组；

- 允许重复的行；

- 使用空值，更糟糕的（？），允许在键中包含空值；

- 在同一列中混合使用不同类型的信息（使用一个单独的“标志”列来指定所涉及的列个别值的类型）；

- 使用一个单一的文本列来代表那些逻辑上



应该是不同的多列。

我想在这里关于星型模式（star schema）添加注释，因为“星型模式”的概念经常是和“反规范化”一起提到的。<sup>[1]</sup>这一概念背后的基本想法如下。假设出于分析的目的，我们希望收集业务事务的历史，例如，假设在供应商和零件的案例中，我们希望记录每次发货发生的特定时间间隔。因此，我们可能会利用一个时间间隔标识符（Time Interval Identifier, TINO）来确定时间间隔，并且引入另一个关系变量TI来将时间间隔本身与这些标识符相关联。修订后的发货关系变量SP和新的时间间隔关系变量TI可能如图8-1所示。在星型模式的术语中，SP是事实表（the fact table），而TI是一个维度表（a dimension table）。供应商关系变量S和零件关系变量P也是维度表（见图8-2）。<sup>[2]</sup>这个整体结构称为“星型模式”，因为对应的实体/关系图和一颗恒星在想象中是相似的，即事实表被维度表包围并通过“辐条”或“射线”与之连接，如图8-2所示。（当然，这些“辐条”或“射线”代表外键引用）。

SP

SNO	PNO	TINO	QTY
S1	P1	T13	300
S1	P1	T15	100
S1	P2	T11	200
S1	P3	T12	400
S1	P4	T11	200
S1	P5	T15	100
S1	P6	T14	100
S2	P1	T13	300
S2	P2	T14	400
S3	P2	T11	200
S3	P2	T13	200
S4	P2	T11	200
S4	P4	T13	200
S4	P5	T12	400
S4	P5	T11	400

TI

TINO	FROM	TO
TI1	t0	t1
TI2	t2	t3
TI3	t4	t5
TI4	t6	t7
TI5	t8	t9

图 8-1 示例事实表 (SP) 和维度表 (TI)

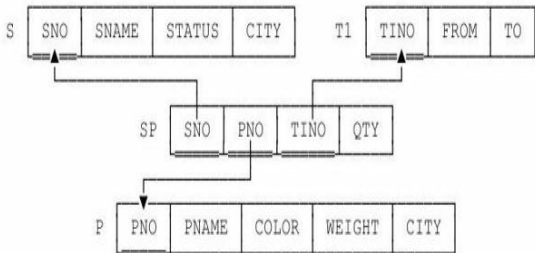


图 8-2 用于供应商和零件（与时间间隔）的星形模式

现在，你可能会想知道，星型模式和传统的关系型设计之间的区别是什么。事实上，正如这个正在讨论的简单例子，星型模式很可能与良好的关系设计相同。然而，在更复杂的情况下，维度表往往没有完全规范化（这么做的目的显然是为了避免连接）。[\[3\]](#)更重要的是，其他关系设计建议往往也违反了规范化的要求（参见本节前面的讨论项目列表）。

星型模式及相关事宜的详细讨论超出了本书的范围，你可以在我的书《数据库系统导论（原书第8版）》中找到一个更广泛的讨论。

[1]但至少有一个权威声称，把星型模式称为反规范化是误导性的。“在描述一个星型模式时使用反规范化，意味着设计开始于规范化。大多数设计不是以这样的方式产生的。非规范化（not normalized）将是一个更好的描述”（引自《Star Schema: The Complete Reference》，by Chris Adamson, McGraw-Hill, 2010）。

[2]为了简单起见，我选择忽略（只是为了目前的讨论）在关系变量S中存在FD{CITY}→{STATUS}，因此，关系变量S是没有完全规范化的事实。

[3]在这方面，考虑来自一本关于数据仓库的书的建议：“[反对]规范化……完全是为了节省磁盘空间[原文]而努力规范化一个维度数据库中的任何表都是浪费时间……维度表不能规范化……规范化维度表破坏浏览的能力”（摘自Ralph Kimball, 《The Data Warehouse Toolkit》，John Wiley & Sons, 1996）。

## 8.5 反规范化是有害的（I）

在本节中，为了支持你应该把反规范化只作为最后的手段的立场，我想提出一个论点，一个符合逻辑并且你可能没有见过的论点。从本质上讲，这个论点是：虽然（众所周知）反规范化可能在逻辑上对更新是不好的，它也可能在逻辑上不利于检索，在这个意义上，它可能使某些查询语句更难编写。（或者说，它可以使它们更容易编写错误，意思是说，如果执行它们，你得到的答案可能对查询语句本身是正确的，但对问题是错误的。）让我举例说明。

再次考虑关系变量S，带有FD{CITY} → {STATUS}。如前所述，这个关系变量可以被视为反规范化关系变量SNC（带有属性SNO、SNAME和CITY）和CT（带有属性CITY和STATUS）的结果。现在考虑查询“获取供应商所在城市的状态值的平均值”。根据图3-2中的示例值，因为雅典、伦敦和巴黎的状态值分别为30、20和30，所以平均值为80/3，把这个值保留到小数点后三位是26.667。那么下面是一些用SQL编写该查询的尝试（为简单起见，因为我假设S是非空的，所以我们不必担心如果我们尝试对一个空集应用AVG操作，在SQL中会发生什么情

况)： [\[1\]](#)

1.

---

```
SELECT AVG (STATUS) AS RESULT  
FROM S
```

---

结果（不正确）：26。这里的问题是，伦敦的状态和巴黎的状态都计算了两次。也许，我们需要在AVG调用内部添加一个DISTINCT。让我们来试一下。

2.

---

```
SELECT AVG (DISTINCT STATUS) AS RESULT  
FROM S
```

---

结果（不正确）：25。不，我们需要检查的是不同的城市，而不是不同的状态值。我们可以通过分组做到这一点。

3.

---

```
SELECT CITY,AVG (STATUS) AS RESULT  
FROM S
```

结果（不正确）：（Athens，30），（London，20），（Paris，30）。这一写法给出的是每个城市的状态平均值，而不是整体的状态平均值。也许我们需要的是对平均值再求平均值吧？

4.

---

```
SELECT CITY,AVG（AVG（STATUS））AS RESULT  
FROM S  
GROUP BY CITY
```

---

结果：语法错误——SQL标准相当正确地不允许以这种方式嵌套调用“集合函数。”[\[2\]](#)再多做一次尝试。

5.

---

```
SELECT AVG（TEMP.STATUS）AS RESULT  
FROM（SELECT DISTINCT S.CITY,S.STATUS  
FROM S）AS TEMP
```

---

结果（最终正确）：26.667。但请注意，这

个表达式与其本身完全规范化的设计（关系变量SNC和CT）相比较是多么复杂：

6.

---

```
SELECT AVG (STATUS) AS RESULT  
FROM CT
```

---

[1]如果我们不能作出这样的假设，不同的SQL表达式将需要什么样的改写？

[2]我说“相当正确”，只是因为我们在SQL的特定上下文中，更正统的语言（如Tutorial D）肯定允许我们做这样的嵌套调用（或相当于这样的调用的模拟）。让我来解释一下。考虑SQL表达式“SELECT SUM (QTY) AS RESULT FROM SP WHERE QTY>100”（为了清楚起见，我特意切换到一个不同的例子）。这里的SUM调用的参数真正是由表达式QTY FROM SP WHERE QTY>100提供的，而一种更正统的语言因此会把这整个表达式用圆括号括起来。但是，SQL不这么做。因此，形式为AVG (SUM (QTY))的表达式是非法的，因为SQL不能确定表达式两侧的哪个部分必须涉及AVG参数，以及哪个部分必须涉及SUM参数。



## 8.6 反规范化是有害的（II）

我刚才说了，支持反规范化的论点是，它使检索更容易表达，让它们有更好的执行性能。但是，这种说法真的经得起仔细分析吗？让我们来仔细看看。

首先，它使检索更容易表达显然不完全正确；上一节就给出了详细的反例，但可以用更简单的例子得出这一点。举例来说，考虑分别对（a）图1-1的规范化设计和（b）一个反规范化设计，其中关系变量S、SP和P被单个“连接”关系变量（比如，SSPP）替代，编写“获取所有供应商详细信息”的查询会涉及什么。下面是Tutorial D的写法：

---

---

```
a.S  
b.SSPP {SNO,SNAME,STATUS,CITY}
```

---

---

或者，如果你喜欢SQL：

---

---

```
a.SELECT*  
FROM S  
b.SELECT DISTINCT SNO,SNAME,STATUS,CITY  
FROM SSPP
```

---

---

第二点是也有很多查询可能执行性能更差。造成这种状况的原因有几个。其中之一是，反规范化导致的冗余性，反过来又可能导致需要做重复消除（注意，在上述第二个SQL写法中的DISTINCT）。另一种原因如下所示。

■再次假设供应商、发货和零件的连接表示为一个单独的存储文件。为简单起见，还假设任何给定的存储文件由一个物理上连续存储的记录集合组成，其中存储文件的每一个记录表示目前出现在关系变量的每个元组。

■为了符合这个论点，我们也假设这种物理结构对于查询“获取提供红色零件的供应商的详细信息”，将合理地执行良好。但查询“获取所有供应商的详细信息”，在这个结构上的执行性能将会比把三个关系变量映射到三个物理上独立的存储文件的结构更差！为什么呢？因为在后一种设计中，所有供应商存储的记录在物理上是连续的，而在前一种设计中，它们实际上将分散在更广泛的范围内，因此将需要更多的I/O。类似的言论也适用于任何只访问供应商或只访问零件，或只访问发货而不是执行某种连接的查询。

■也请注意，再次因为反规范化增加冗余，它将很有可能导致更大的存储记录，而这个事实

也可能会导致更多而不是更少的I/O。例如，一个4KB的页面可以容纳两个2KB的存储记录，但只能容纳一个3KB的存储记录，因此，反规范化增加了50%的冗余可能会增加100%的I/O（当然，这里我说得很不严格）。

我的下一个观点是，即使我们接受反规范化让检索更容易表达且有更好的执行性能的说法，但可以肯定，它使更新更难以表达且有更差的执行性能。现在，这一点是广泛理解的（如我之前说的），但是，没有如此广泛理解的是，反规范化也打开了违反完整性的的大门。例如，在关系变量S（而不是投影关系变量SNC和CT）中，有人——无论是系统还是用户，并且在目前的实践中可能是后者——将不得不负责维护 $FD\{CITY\} \rightarrow \{STATUS\}$ ；并且如果不做这种维护，就会丢失完整性。（相比之下，在两关系变量的设计中，必须做的所有工作是对CT执行键约束——这个一定是由系统而不是用户做的——并且然后将会“自动地”维护每个城市都有一个状态的事实。）

我最终的观点是，无论我们谈论的是以下哪个内容：

a.真正的反规范化，它只在物理层执行，或

b.在当今的SQL产品中我们必须执行的一种反规范化，这也会影响到逻辑层。

当人们说“为了性能反规范化”时，他们实际上是指特定的应用程序（specific application）的性能，这点并没有得到足够广泛的理解。如我在前面说过的，反规范化通常是基于对整体问题的有点狭隘的观点。任何给定的物理设计都可能会对一些应用程序有利，而对另外一些应用程序不利（即，在其对性能的影响方面）。

## 8.7 结束语

在这一章中，我已经对赞成不反规范化给出了一些强有力的论据；因此，实际上我已经给出了赞成规范化的论据。[\[1\]](#)良好的设计通常是完全规范化的，这是千真万确的。但重要的是要明白，相反的命题未必成立！也就是说，一个设计可能是完全规范化的，但它仍然是糟糕的。例如，关系变量S在属性SNO和STATUS上的投影ST肯定属于BCNF，其实，它属于可能的最高范式，正如我们将在本书第三部分看到的，但它显然不是一个好的设计，正如我们在第6章中所看到的。

[\[1\]](#)一位审阅者指出，规范化的另一个优势是，它往往简化复杂查询的编写，因为它通过简单的子表达式的嵌套使表达式更加模块化。

## 习题

8.1 有时候有人声称，二元关系变量比关系模型支持的一般的 $n$ 元关系变量的优势在于二元关系变量总是属于BCNF（也许，这意味着，在所有事情当中，我们并不需要担心规范化和反规范化）。请证明这种说法的正确性，或通过提供一个反例表明它是不正确的。

8.2 以下内容摘录自对数据库顾问的公开采访。[\[1\]](#)它是从顾问的一个声明开始的。

顾问：这个问题.....在很大程度上是由于规范化数据分布在多个[关系变量].....但是，如果数据是反规范化的，许多查询会更容易理解.....

记者：请问反规范化有可能降低数据的完整性和降低支持意料之外的查询的灵活性吗？

顾问：规范化和它强调的消除冗余存储，是一个纯粹的事务处理问题。当用户查看数据时，他们以一种冗余的形式看到它。为了将数据转换成一种对用户有用的形式，必须通过一个连接将它反规范化，这实质上是为了更好地利用数据而将其动态反规范化的一种方式。

现在的问题是，用户无法容忍连接的时间和成本。为了解决这个问题，许多公司在越来越多的决策支持数据库中复制数据，它代表数据的反规范化视图。

在你看来，在上述顾问的意见中有什么错误（如果有）？

8.3 本章正文中提到了使用替代标识符或代理键的可能性。事实上，许多设计师建议使用人工或代理键代替有时也称为“自然”键的东西。例如，我们可以添加一个属性（比如SPNO）到我们通常的发货关系变量（当然，确保它具有唯一性）中，然后把{SPNO}当作此关系变量的一个代理键。（但是，请注意，{SNO,PNO}仍然是一个键，它只是不再是唯一的键）。因此，在通常的关系意义上，代理键是键，但（a）它们总是正好仅涉及一个属性，并且（b）它们的值仅作为它们代表的实体的替代（即，它们仅仅用于代表实体存在的事实——它们绝对没有承担任何额外的含义或任何形式的包袱）。在理想的情况下，这些替代值是系统生成的，但无论它们是系统生成的还是用户生成的，都不影响上述代理键的基本思想。两个问题：代理键与元组标识符是同样的东西吗？你认为使用代理键是一个好主意吗？

8.4 如果两个设计是信息等效的，那么一定可以使用关系代数操作将它们互相转换。因此，考虑对于本章正文的员工关系变量的下列竞争的设计：

---

```
EMP {ENO,JAN_PAY,FEB_PAY, ....., DEC_PAY}
KEY {ENO}
EMP {ENO,MONTH,PAY}
KEY {ENO,MONTH}
```

---

请使用Tutorial D或SQL（或你自己首选的数据库语言）来显示这些设计中的每个是如何转换成另一个的。

[1]该内容选自Data Base Newsletter, 22, No.5 (September/October 1994)。



## 第三部分 连接依赖、第五范式及其他 相关事项

本书这一部分讲述连接依赖和第五范式的内容，就如上一部分讲述函数依赖和Boyce/Codd范式的内容。它还将规范化必须处理的大量不严格的事项联系在一起，作为一般意义上的设计理论的重要组成部分。

## 第9章 连接依赖及5NF（非正式的）

If you can't beat them, join them.

——Anon.

正如Boyce/Codd范式是依据函数依赖来定义的，第五范式（5NF）是依据连接依赖（JD）来定义的，[\[1\]](#)正如第4章指出的，事实上，5NF是关于JD的范式，就像BCNF是关于FD的范式。因此在本书该部分中对这些概念的处理方式与在第二部分中对BCNF和FD问题的处理方式类似。换句话说，我计划在第10章中正式地处理这些内容，在本章则非正式地处理它们。

请让我立即补充如下：虽然5NF确实是关于JD的“范式”，但是不应该认为这种状况意味着5NF是规范化进程的最终目标。实际上，恰恰相反：如我们将在第13章看到的，至少有两个其他范式，更加适合这个头衔。然而，从教学的角度来看，以及从一个历史的角度来看，我认为最好先详细讨论5NF。（我提到这一点，仅仅是为了避免给人一种错误的印象，我的一位审校者认为，我应该以与此不同的顺序介绍相关内容，但我不同意。）

原来，在以前的文章中，我倾向于认为JD只是一个广义的FD。现在我认为这种看法是错误的，或者至少是误导性的，我现在觉得最好把JD当作一个完全不同的现象。当然，FD和JD都是依赖关系（即约束），并且它们在某些方面确实彼此相似，尤其是，在关系变量R中存在一个特定的JD意味着，R可以无损地以某种方式分解的事实，正如在关系变量R中存在一个特定的FD也意味着，R可以无损地以某种方式分解的事实。这也是成立的，因为每一个FD蕴含着一个JD，所以，如果关系变量R中存在某个FD F，那么在R中也一定存在某个JD J。但并不是所有的FD都蕴含JD，事实上，很不严格地说，但是我必须强调，我说的是非常不精确的，我们可以说，5NF与不被FD蕴含的JD有关。也就是说，如果某个关系变量R属于BCNF，但它服从某个不被FD蕴含的JD，那么它就可能是与5NF的概念相关的。

那么，当且仅当一个关系变量服从的所有FD都被键蕴含时，它属于BCNF。因此，你可能会想到，当且仅当一个关系变量服从的所有JD都被键蕴含时，它属于5NF。[\[2\]](#)然而，后一种概念（即JD被键蕴含的概念）比其FD对应概念界定起来要麻烦一点；事实上，我们很快就会看到，围绕这些观点有非常丰富的理论，并且这个理论的

某些部分在一开始可能有点铺天盖地（不是说混乱）。你需要保持清醒的头脑！正如在这些问题上的认识远比我更深刻的人曾经对我说过的：JD 很神秘。

写了这么多的序言，现在让我们来研究具体细节。

## 9.1 连接依赖的基本思路

在这本书中到目前的大部分时间，我一直在做一个心照不宣的假设，即，我一直在假设，当我们分解某个关系变量时，我们总是通过把该关系变量正好替换为它的两个投影来实现的。[\[3\]](#)

（请注意，希思定理，它为到目前为止我已经说的关于无损分解的内容提供正式支持，特别是针对正好分解成两个投影的分解。）更重要的是，这样的假设是完全必要的，只要我们的目标仅仅是BCNF，换句话说，它成功地引导我们达到了这个特定目标。所以，当你发现虽然不能无损地分解为两个投影，但它能无损地分解成三个投影（或可能超过三个）的关系变量存在时，你可能会感到惊讶。

顺便说一句，我明显注意到，1969年，Codd在他的第一篇关于关系模型的论文中（见附录

C) 举了一个例子，表明他意识到这种可能性。然而，这个例子显然被原始论文的大多数读者忽略了，因此，理所当然地，当数年后（准确地说，是1977年）重新发现这种可能性时，研究界似乎对此感到惊奇。

现在，我刚才说，虽然不严格，5NF与不被FD蕴含的JD有关。我现在可以补充，尽管再次说得非常不严格，它与不能无损地分解成两个投影，但可以无损地分解成三个或更多个投影的关系变量有关。换句话说，当这些情况出现时，即，当存在不由FD蕴含的JD和只能无损地分解为两个以上投影的关系变量时，你真的必须掌握JD和5NF。

因此，当我们说一些JD在一些关系变量中存在时，我们究竟是什么意思？这里有一个定义。

■定义n: 设 $X_1, \dots, X_n$ 是关系变量R的标题H的子集，那么当且仅当R可以无损地分解成其在 $X_1, \dots, X_n$ 上的投影（即，当且仅当R的每一个合法的r值等于相应的投影 $r_1, \dots, r_n$ 的连接）时，连接依赖（JD）

---

$$\Join\{X_1, \dots, X_n\}$$

---

在R中存在。X1, ....., Xn是此JD的分量 (component), JD整体可以理解为“星型X1, ....., Xn”或“连接X1, ....., Xn”, 虽然我赶紧补充说, 这里的“连接”真的不是适当的语句, 因为连接运算符 (至少按照通常理解) 连接的是关系, 而X1, ....., Xn不是关系, 而是标题。

通过一个简单的例子, 再次考虑供应商关系变量S。我们知道, 因为此关系变量服从  $FD\{CITY\} \rightarrow \{STATUS\}$ , 所以希思定理告诉我们它可以无损地分解成其在  $\{SNO, SNAME, CITY\}$  和  $\{CITY, STATUS\}$  上的投影。换句话说, 在该关系变量中存在下面的JD:

---

---

$$\star \{ \{SNO, SNAME, CITY\}, \{CITY, STATUS\} \}$$

---

---

产生的要点如下所示。

■注意, 从定义可以得出, 分量X1, ....., Xn的并集必须等于H (即, H的每个属性都必须出现在这些分量中的至少一个中), 因为否则R不可能等于对应这些分量的投影的连接。

■不同的作者用不同的符号来表示JD, 我用一种特殊的星号, 但在研究文献中, 符号  $\bowtie$

（“领结”）是更经常遇到的。

■指出下面这一点可能是有帮助的，即，说存在某个JD等于说，如果我们把指定的投影连接在一起，我们将永远不会得到任何“假”元组（如习题3.2称呼它们那样）。

■下面的观点也可能是有帮助的。我将通过一个简单但略显抽象的例子来解释这些。设关系变量R具有属性A、B、C、D（只有这些属性），并设在R中存在 $JD \bowtie \{AB, BC, CD\}$ （“希思符号”，见第7章），让我用符号“ $\in$ ”来表示“出现在”的意思（见附录D习题5.4的答案）。那么说在R中存在已知的JD等于说以下内容：[\[4\]](#)

---

---

```
if EXISTS c1 ( EXISTS d1 ( (a,b, c1, d1) ∈ R ) ) AND
EXISTS a2 ( EXISTS d2 ( (a2, b,c, d2) ∈ R ) ) AND
EXISTS a3 ( EXISTS b3 ( (a3, b3, c,d) ∈ R ) )
then (a,b,c,d) ∈ R
```

---

---

说明：设在R中有一个具有A=a和B=b的元组及一个具有B=b和C=c的元组及一个具有C=c和D=d的元组。那么元组(a,b)、(b,c)和(c,d)将分别出现在R在AB、BC和CD上的投影中，这样我们把这三个投影连接在一起，元组(a,b,c,d)就会出现。此外，下面的陈述也显然

是正确的：如果元组  $(a,b,c,d)$  出现在  $R$  中，那么元组  $(a,b)$ 、 $(b,c)$  和  $(c,d)$  肯定会出现在这三个投影中（因此，在上述形式语句中的“如果”（if）实际上可以被“当且仅当”（If and only if）替换）。

作为对最后一点的一个简单说明，假定在关系变量  $S$  中存在下面的 JD：

---

---

$$\odot \{ \{SNO, SNAME, CITY\}, \{CITY, STATUS\} \}$$

---

---

表示当且仅当  $S$  中有一个元组具有  $SNO=s$  且  $SNAME=n$  且  $CITY=c$ ，并且在  $S$  中有一个元组具有  $CITY=c$  且  $STATUS=t$  时，元组  $(s,n,t,c)$  在  $S$  中出现。

要先对这个例子作个延伸，正如我们知道的，关系变量  $S$  中存在指定的 JD 是希思定理的一个合乎逻辑的结果。事实上，我们现在可以对希思定理重新表述如下。

■希思定理（对于关系变量，依据 JD 重新表述）：设关系变量  $R$  具有标题  $H$ ，并设  $X$ 、 $Y$  和  $Z$  是  $H$  的子集，其中  $X$ 、 $Y$  和  $Z$  的并集等于  $H$ 。设  $XY$  表示  $X$  和  $Y$  的并集， $XZ$  表示类似的含义。如果  $R$  服从



FD  $X \rightarrow Y$ ，则R服从JD  $\odot \{XY, XZ\}$ 。

如前所述，因此，FD蕴含JD，但正如我们将看到的，不是所有的JD都被FD所蕴含。不过，在我阐述这一点之前，我想强调一个要求，即，给定的JD的分量的并集必须等于相关的标题。没有适用于FD的类似规定；对于FD，左侧和右侧的并集不必等于相关的标题，它们只必须是该标题的子集。在一定意义上，这种区别可能有助于说明一点（至少在直觉上）：JD和FD真的是不同种类的东西。

现在，在上述例子中的JD：

---

---

$$\odot \{\{SNO, SNAME, CITY\}, \{CITY, STATUS\}\}$$

---

---

是二元的：它有两个分量，并且它对应一个分解成两个投影的无损分解。相比之下，在关系变量S中存在另一个JD：

---

---

$$\odot \{\{SNO, SNAME\}, \{SNO, CITY\}, \{CITY, STATUS\}\}$$

---

---

这个JD是三元的，但它是派生的，实际上，通过两个二元JD的“级联”得到。

■首先，我们已经知道S中存在二元JD $\bowtie$  $\{\{SNO, SNAME, CITY\}, \{CITY, STATUS\}\}$ 。

■但因为S在 $\{SNO, SNAME, CITY\}$ （对应于二元JD的分量之一）上的投影中存在FD $\{SNO\} \rightarrow \{SNAME\}$ ，[\[5\]](#)所以该投影中存在二元JD $\{\{SNO, SNAME\}, \{SNO, CITY\}\}$ 。

因此，在原始关系变量中存在给定的三元JD。相比之下，我将在下一节中给出一个例子，它不是从二元JD通过级联派生得来的三元JD，因此也得到一个例子，它可以无损地分解成三个投影，但不能无损地分解成两个投影的关系变量。

[\[1\]](#)在4NF中也是如此，但在第12章前，我会忽略4NF（大多数情况下）。

[\[2\]](#)这个很不正式的定义会是我在本章给出的唯一一个5NF定义。

[\[3\]](#)当然，我这里指在整个过程中的单个步骤。显然，在一般情况下，重复的步骤（即，重复的单独分解）将产生由两个以上投影组成的结果，即使每一个单独的分解只分解成两个投影。

[\[4\]](#)在以下定义中，关键字EXISTS表示存在量词（existential quantifier）。如果你需要一本有关这个事项的教程，请参阅《SQL与关系数据库理论》。

[5]这里我用了一个很容易证明的定理：即，当且仅当关系变量R本身存在FD  $X \rightarrow Y$ 时，在R的某个投影中存在此FD（见第12章的习题12.5）。

## 9.2 一个属于BCNF但不属于5NF的关系变量

我将以我们一贯的发货关系变量SP经修订的版本（我称之为SPJ）开始。修订包括：（a）删除属性QTY，及（b）引入一个新的属性JNO（“工程编号”）。谓词是供应商SNO供应零件PNO给JNO工程，及一个如图9-1所示的示例值。请注意，因为此关系变量是“全键”，所以一定属于BCNF。[\[1\]](#)

SPJ	SNO	PNO	JNO
	S1	P1	J2
	S1	P2	J1
	S2	P1	J1
	S1	P1	J1

图 9-1 关系变量SPJ——示例值

现在，假设下面的业务规则有效。

■如果（a）供应商s供应零件p，并且（b）零件p被提供给工程j，并且（c）工程j由供应商s供应，那么，（d）供应商s为工程j供应零件p。[\[2\]](#)

更具体地说，这个规则说的是，如果（例如）以下三个都是真命题。

- a.史密斯提供活动扳手给某个工程。
- b.有人提供活动扳手给曼哈顿工程。
- c.史密斯提供某些东西给曼哈顿工程。

那么，以下也是一个真命题：

- d.史密斯提供活动扳手给曼哈顿工程。

换句话说，如果关系变量SPJ包含代表命题a、b、c的元组，那么它一定也包含一个代表命题d的元组。[\[3\]](#)需要注意的是图9-1所示的示例值满足此要求，（设S1是史密斯，P1是活动扳手，J1是曼哈顿工程）。

不过，命题a、b、c通常并不意味着命题d。具体地说就是，如果我们只知道命题a、b、c成立，那么我们知道，史密斯提供活动扳手给某个工程j；我们知道，某个供应商s供应活动扳手给曼哈顿工程；并且我们知道，史密斯提供一些零件p给曼哈顿工程，但我们不能有效地推断出s是史密斯，我们不能有效地推断出p是活动扳手，

也不能有效地推断出j是曼哈顿工程。像这样的错误推断的例子，有时也称为连接陷阱（the connection trap）。然而，在目前的情况下，业务规则告诉我们，没有陷阱，也就是说，在这个特定的案例中，我们可以从命题a、b、c有效地推断出命题d。

现在，让我们更仔细地考虑这个例子。只在这一刻，让我使用SP、PJ、JS表示SPJ分别在{SNO,PNO}、{PNO,JNO}、{JNO,SNO}上的投影。那么，我们就有以下几条结论。

■通过投影和连接的定义，

---

---

$$\begin{aligned} &\text{IF } (s,p,j) \in \text{JOIN}\{\text{SP},\text{PJ},\text{JS}\} \\ &\text{THEN } (s,p) \in \text{SP} \\ &\text{AND } (p,j) \in \text{PJ} \\ &\text{AND } (j,s) \in \text{JS} \end{aligned}$$

---

---

并且因此存在s'、p'和j'，使得

---

---

$$\begin{aligned} &(s,p,j') \in \text{SPJ} \\ &\text{AND } (s,p',j) \in \text{SPJ} \\ &\text{AND } (s',p,j) \in \text{SPJ} \end{aligned}$$

---

---

旁白：我很抱歉在上述内容中略微缺乏对称

性，但如果我们要以页面上的有序逗号分隔列表来展示元组，这就是不可避免的，因为元组按照定义是无序的。旁白结束。

■但是，通过业务规则，

---

IF  $(s, p, j') \in SPJ$   
AND  $(s, p', j) \in SPJ$   
AND  $(s', p, j) \in SPJ$

---

那么，我们一定有：

---

$(s, p, j) \in SPJ$

---

■所以，如果  $(s, p, j)$  出现在SP、PJ、JS的连接中，那么它也出现在SPJ中。而且反过来显然也是正确的，即，如果  $(s, p, j)$  出现在SPJ中，那么它肯定会出现在SP、PJ、JS的连接中。

因此，当且仅当  $(s, p, j)$  出现在SP、PJ、JS的连接中时，它出现在SPJ中。因此，关系变量SPJ的每一个合法值都等于其在 $\{SNO, PNO\}$ 、 $\{PNO, JNO\}$ 、 $\{JNO, SNO\}$ 上的投影的连接，因此，JD

---

肯定存在于关系变量SPJ中。

现在注意前述JD是三元的——它有三个分量。更重要的是，它并不被FD蕴含。<sup>[4]</sup>因此，它肯定不被键蕴含（回顾第5章，一个键约束只是FD的一个特例）。因此，关系变量SPJ虽然属于BCNF（因为它是“全键”），但它不属于5NF。

为了更好一点地了解这种状况，回头看如图9-1所示的示例SPJ值是有帮助的。图9-2显示了（a）对应于该示例值SP、PJ和JS的投影值，（b）连接SP和PJ投影（在{PNO}上）的效果，以及（c）连接该结果与JS的投影（在{JNO, SNO}上）的效果。正如你所看到的，连接前两个投影产生原始SPJ关系的一个副本加上一个额外的（“假”）元组，连接其他投影随后消除额外的元组，从而让我们回到原来的SPJ关系。此外，不管我们选择哪一对投影做第一个连接，虽然在每种情况下，中间结果是不同的，但净效果是一样的。习题：检查这种说法。

因此，再次重申， $JD \odot \{SP, PJ, JS\}$ 存在于关系变量SPJ中（如果现在你允许我使用SP、PJ、SJ的名称，不是指上述这样的投影，而是指相应



的标题的子集），换句话说，该JD（可谓）捕获了原有业务规则的本质。因此，关系变量SPJ可以相应地无损分解。更重要的是，它可能应该是，因为它存在冗余；具体而言，根据图9-1的示例值，命题“供应商S1提供零件P1给工程J1”既明确地由元组（S1，P1，J1）表示，又是此JD和其他三个元组代表的命题的隐含逻辑推论。

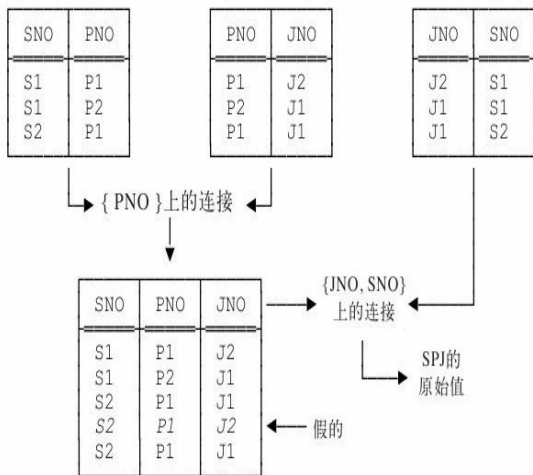


图 9-2 SPJ等于其所有的三个二元投影的连

接，但不等于任何两个投影的连接

更多术语：我们说类似应用于SPJ示例中的一个JD是元组强迫（tuple forcing）的，因为如果某些元组出现，它迫使一些额外的元组也出现。例如，在图9-1中，三个元组（S1，P1，J2），（S1，P2，J1），和（S2，P1，J1）的出现强制了元组（S1，P1，J1）的出现。请注意，并非所有的JD都是元组强迫的；例如，连接依赖  $\bowtie\{\{SNO, SNAME, CITY\}, \{CITY, STATUS\}\}$  存在于关系变量S中，但毫无疑问它并不迫使额外的元组出现。注意：我们提前片刻了解，后面将会表明，一个服从元组强迫的JD的关系变量不能属于5NF（虽然如SPJ示例所示，它可以属于BCNF）。

[1]事实上，它也属于4NF，然而，它不属于在第13章所谓的无冗余范式（redundancy free normal form），RFNF，因此它也不属于5NF。

[2]也许有人会说，严格来说，“供应商s供应零件p”，这里确实应该是“供应商s供应零件p给某个工程”（类似“零件p被提供给工程j”和“工程j是由供应商s提供的”）。这是否是事实取决于情况的完整语义，为了直观简洁起见，我故意没有说得很彻底。我会在第15章中回顾这个问题。

[3]我再次有一些马虎。例如，考虑命题a（“史密

斯供应活动扳手给某个工程”）。如果“某个工程”在这里的意思是“某个未指定的工程”，即，存在这样的工程，但我们不知道它是什么，那么此命题不是SPJ的谓词的实例，且没有SPJ元组可能代表它。但一个SPJ元组肯定可以代表命题“史密斯供应活动扳手给某个指定的工程”，更重要的是，如此表达的命题，还蕴含了命题“史密斯供应活动扳手给某个工程”（即“存在一个工程j，史密斯供应活动扳手给j”）。我希望那是十分明显的！

**[4]**证明：关系变量SPJ中仅有的FD都是平凡的，且“满足那些平凡的FD的每一个关系也满足此JD”显然不成立。例如，如图9-1所示，包含前三个元组但不包含第四个元组的关系就不是这样。

## 9.3 循环规则

现在观察SPJ例子的业务规则的循环性质（“如果s与p有关且p与j有关且j与s有关，那么s和p和j都必须直接相关，在这个意义上，它们都必须一起出现在同一个元组中”）。让我们同意把这条规则称作“三路循环”。那么，我们一般可以说，对于某个大于2的n，如果存在一个n路循环规则，我们可能将面临一个关系变量，它（a）属于BCNF且不属于5NF，因此（b）可以无损地分解成n个投影且不能无损地分解成更少的投影。[\[1\]](#)

虽然如此，我也必须说，根据我的经验，在实践中这样的循环规则是罕见的，这意味着，在实践中，大多数关系变量，如果它们至少属于BCNF，那么它们很可能也属于5NF。事实上，在实践中，找到一个属于BCNF而不属于5NF的关系变量，这是相当不寻常的。尽管这不寻常，但也不是没听说过——我曾经遇到过真实世界中的一些例子。换句话说，这样的关系变量罕见的事实，不意味着你不必担心它们，或不必担心JD和5NF。恰恰相反：JD和5NF实际上是你的设计师的工具箱中的工具，而且（在其他条件相同时），你应该尝试确保在你的数据库中的所有关

系变量都属于5NF。 [2]

[1]如果规则是稍微简单的形式“如果s与p有关且s与j有关，那么s和p和j必须都直接相关”，那么我们可能有一个属于BCNF，但不属于4NF（更不用说，因此不属于5NF）的关系变量。请参阅第12章。

[2]除了在第13章中注明的情况。

## 9.4 结束语

我将用一些其他意见来结束本章。首先，请注意，我在本书该部分假设（事实上，我在前一部分也是这么做的），我们仅关心这样的依赖，它们与用投影作为分解操作符并用连接作为相应的重构操作符有关。在这样的假设下，从连接依赖的定义立即可以得出，在某种意义上，JD是“最终”类别的依赖；也就是说，没有“更高”类别的依赖，所以JD只是更高类别依赖的一个特例。因此进一步可得出（虽然我还没有真正正确地定义它），第五范式是关于投影和连接（这说明了它的另一个名称，投影-连接（projection-join）范式）的最终范式。[\[1\]](#)

其次，我已经多次提到属于BCNF但不属于5NF的关系变量，事实上，我已经心照不宣地假设，如果关系变量R属于5NF，那么它肯定属于BCNF。事实上，这个假设是正确的。为了备案，让我也明确说明5NF始终是可以实现的，也就是说，任何不属于5NF的关系变量总是可以分解成一个5NF投影集合，但不一定不丢失依赖，当然，因为我们在第7章已经知道，分解为BCNF和保持依赖关系可能是相互冲突的目标。

再次，从5NF的定义可以得出，一个属于

5NF的关系变量R保证不存在可以通过投影来删除的冗余。换句话说，说R属于5NF等于说，把R进一步无损地分解为投影，虽然是可能做到的，但这么做肯定不会删除任何冗余。但是，请非常慎重，说R属于5NF并不是说R不存在冗余。（与此相反的观念是另一种流行的误解。见第1章的习题1.11）。事实上，有许多种冗余是上述投影无力删除的——这是我在1.4节说明的观点，大意思是说，还有目前的设计理论根本不能解决的许多问题。举例来说，可以考虑图9-3，它显示关系变量CTXD的示例值，它属于5NF但仍然存在冗余。谓词是教师TNO花了DAYS天用教科书XNO讲授课程CNO。唯一的键是{CNO,TNO,XNO}。正如你所看到的，（比如）教师T1教课程C1这个事实出现了两次，课程C1使用教科书X1的事实也出现了两次。[\[2\]](#)

CTXD	CNO	TNO	XNO	DAYS
	C1	T1	X1	7
	C1	T1	X2	8
	C1	T2	X1	9
	C1	T2	X2	6

图 9-3 5NF关系变量CTXD——示例值

让我们更仔细地来分析这个例子：

■由于{CNO,TNO,XNO}是一个键，关系变量服从以下函数依赖：

---

$$\{CNO,TNO,XNO\} \rightarrow \{DAYS\}$$

---

这是一个“从键出来的箭头”。

■因此，DAYS依赖于CNO、TNO、XNO三者，并且它不能出现在任何少于所有三者的关系变量中。

■因此，此关系变量根本没有适用于将其分解成投影的（非平凡）分解——此关系变量属于5NF。注意：当且仅当一个分解是在平凡的依赖（FD或JD）基础上时，它才是平凡的，并且当且仅当这个分解不是平凡的时，它才是非平凡的。第4章和第5章讨论了平凡的FD，平凡的JD将在第10章中讨论。

■因此，更不用说，肯定没有分解成投影的分解方法可以消除这些冗余。

[1]除了6NF之外（请再次参阅第13章）。

[2]一位审校者坚持认为这些重复并没有真正构成



冗余。我不想在这里争论这一点；我只是提醒你，我将会在第15章中详细研究究竟是什么构成冗余的整个问题。

## 习题

9.1 （与本章正文重复），请验证对图9-2所示的二元关系中的任何一对执行连接产生的结果都包含一个“假”的元组（即没有一个出现在图9-1中的元组），并请验证将第三个二元关系与中间结果相连接就消除了上述那个假的元组。

9.2 写一个Tutorial D CONSTRAINT语句来表达本章正文所讨论的关系变量SPJ中存在的JD。

9.3 为以下的案例设计一个数据库。被代表的实体是销售代表、销售区域和产品。每名代表负责一个或多个区域的销售，每个区域有一名或多名负责销售的代表。每名代表负责销售一个或多个产品，每个产品都有一名或多名负责销售的代表。每个产品在一个或多个区域销售，每个区域有一个或多个产品出售。最后，如果代表 $r$ 负责区域 $a$ ，而产品 $p$ 在区域 $a$ 中出售，且代表 $r$ 销售产品 $p$ ，则 $r$ 在 $a$ 销售 $p$ 。

9.4 如果可能，举一个自己的工作环境中的某个例子，其关系变量属于BCNF但不属于5NF。

## 第10章 连接依赖及5NF（正式的）

After great pain,a formal feeling comes.

——Emily Dickinson

正如第5章是由对第4章所介绍的内容更正规的处理方式组成的，本章也包括对第9章所介绍的内容更正规的处理方式。但你很快就会看到，本章覆盖的内容与第5章相比是相当多的。让我只是提前透露，正如第5章对2NF或3NF讨论得很少，关于4NF，本章几乎没什么可说的，其实，至少从某些观点来看，正如2NF和3NF，4NF也主要具有历史意义。不过，对于它我在后面的章节（第12章）中还会更多地论及4NF。

### 10.1 连接依赖

我从对JD是什么给出一个精准的定义开始，然后是故意与第5章中的对应文本类似的一些说明文字。（类似的言论也适用于下一节。）

■定义：设H是一个标题，那么一个关于H的连接依赖（JD）是一种形式为 $\{X_1, \dots, X_n\}$ 的表达式，其中， $X_1, \dots, X_n$ （JD的分量）是H的子集，且它们的并集等于H。注意：如果H

是不言自明的，那么“关于H的JD”这句话可以缩写为仅JD。

下面是一些例子：

---

---

$\odot\{\{SNO, SNAME, CITY\}, \{CITY, STATUS\}\}$   
 $\odot\{\{CITY, SNO\}, \{CITY, STATUS, SNAME\}\}$   
 $\odot\{\{SNO, SNAME\}, \{SNO, STATUS\}, \{SNAME, CITY\}\}$   
 $\odot\{\{SNO, CITY\}, \{CITY, STATUS\}\}$

---

---

需要特别注意，JD（如FD）是关于某个标题定义的，而不是关于某个关系或某个关系变量定义的。例如，在刚才显示的几个JD中，前三个是关于标题 $\{SNO, SNAME, STATUS, CITY\}$ 定义的，第四个是关于标题 $\{SNO, STATUS, CITY\}$ 定义的。

还请注意，从形式上看（再次如同FD），JD只是表达式：当这些表达式就一些具体的关系解释时，就成为根据定义，计算结果不是TRUE就是FALSE的命题。例如，如果上面所示的前两个JD就关系变量S（图1-1）的当前值表示的关系解释，那么第一个JD的计算结果为TRUE而第二个为FALSE。当然，在某些特定的情况下，仅当它在这种情况下的计算结果为TRUE时，把 $\odot\{X_1, \dots, X_n\}$ 非正式地定义为一个JD是常见的。然而，这样的定义造成无法说给定的关系无法满足

或违反某个JD——因为，根据非正式的定义，一个不满足的JD首先就不会是一个JD。例如，我们将无法说属于关系变量S的当前值的关系违反了上面显示的JD中的第二个。

下面是另一个碰巧被关系变量S的当前值（事实上，被此关系变量的所有合法值）满足的JD的例子：

---

---

$$\odot \{ \{ \text{SNO}, \text{SNAME}, \text{CITY} \}, \{ \text{CITY}, \text{STATUS} \}, \{ \text{CITY}, \text{STATUS} \} \}$$

---

---

此JD对应一个无损的分解，其中一个投影在重建过程中不是必需的。事实上，它明显地相当于先前所示的四个JD中的第一个JD： [\[1\]](#)

---

---

$$\odot \{ \{ \text{SNO}, \text{SNAME}, \text{CITY} \}, \{ \text{CITY}, \text{STATUS} \} \}$$

---

---

这意味着，相同的两个分量之一可以去掉，而不会带来显著的损失。由于这样的原因，我会放心地说任何给定的JD的分量构成一个集合，即使在JD中的书面形式分量的逗号分隔列表可能包含重复，这是集合本身永远不会有。（当然，这就是把逗号分隔列表括在大括号中的原因）。

继续这个定义。

■定义：设关系 $r$ 的标题为 $H$ ，并设 $\odot\{X_1, \dots, X_n\}$ 是一个关于 $H$ 的JD，比如 $J$ ，如果 $r$ 等于其在 $X_1, \dots, X_n$ 上的投影的连接，那么 $r$ 满足 $J$ ；否则 $r$ 违反 $J$ 。

注意，是关系，而不是关系变量，满足或违反某些特定的JD。例如，关系变量 $S$ 的当前值表示的关系同时满足这些JD：

---

---

$$\odot\{\{SNO, SNAME, CITY\}, \{CITY, STATUS\}\}$$
$$\odot\{\{SNO, SNAME\}, \{SNO, STATUS\}, \{SNAME, CITY\}\}$$

---

---

并违反这个JD：

---

---

$$\odot\{\{CITY, SNO\}, \{CITY, STATUS, SNAME\}\}$$

---

---

注意，并没有出现关系满足或违反JD $\odot\{\{SNO, CITY\}, \{CITY\}\}$ （我们原来的4个示例JD集合中的最后一个JD）的问题，因为JD不是关于这个关系的标题定义的。

■定义：设关系变量 $R$ 具有标题 $H$ ，并设 $\odot\{X_1, \dots, X_n\}$ 是一个关于 $H$ 的JD，比如 $J$ ，

那么当且仅当可以被赋值给关系变量R的每一种关系都满足J时，JD J在关系变量R中存在（等价于，关系变量R服从JD J）。关系变量R中存在的JD是R的JD。

请注意，这里描述的术语的区别——关系满足（或违背）JD，但它存在（或不存在）于关系变量中。我会在后续内容中坚持这种区别。举例来说，在关系变量S中存在以下JD：

---

$$\odot\{\{SNO, SNAME, CITY\}, \{CITY, STATUS\}\}$$

---

而不存在这些JD：

---

$$\begin{aligned} &\odot\{\{SNO, SNAME\}, \{SNO, STATUS\}, \{SNAME, CITY\}\} \\ &\odot\{\{CITY, SNO\}, \{CITY, STATUS, SNAME\}\} \end{aligned}$$

---

（对比遵循以前的定义的例子。）所以，现在，最终，我们精确地知道了对于一个给定的关系变量服从一个给定的JD是什么意思——并且立即可以得到，当且仅当关系变量R服从 $\text{JD} \odot \{X_1, \dots, X_n\}$ 时，它可以无损地分解成其在 $X_1, \dots, X_n$ 上的投影。

**[1]**一般来说，当且仅当每一个满足其中一个JD的

关系也满足另一个JD时，两个JD是等价的。在接下来的章节，我将对此主题（JD的等价）作更多的说明。



## 10.2 第五范式

现在，过去当我们在谈论FD和BCNF时，我们曾陷入平凡的FD、FD不可约性、被键蕴含的FD及各种相关事项的讨论中。我敢肯定，你现在会期望对于JD和5NF也出现类似的概念，但细节会更复杂一点。不过，平凡的JD的概念其实是很简单的。

■定义：设 $\bowtie\{X_1, \dots, X_n\}$ 是一个关于标题H的JD，比如J，那么当且仅当每个具有标题H的关系满足它时，J是平凡的。

从这个定义中，可以很容易地证明如下结果：

■定理：设 $\bowtie\{X_1, \dots, X_n\}$ 是一个关于标题H的JD，比如J，那么当且仅当某个 $X_i$  ( $1 \leq i \leq n$ ) 等于H（因为每一个具有标题H的关系必定满足每个形式为 $\bowtie\{\dots, H, \dots\}$ 的JD）时，J是平凡的。

我们可以把这个定理当作一个可操作的（或“语法”）定义，因为它提供了一个有效的测试，该测试可以很容易地在实践中应用。（相比之下，正式的或“语义”定义对于确定一个给定的

JD是否平凡的实际问题是没有多少用处的。)

我会把JD不可约性的讨论推迟到第11章，在此之前，我想解释一个被键蕴含的JD究竟是什么意思。

■定义：设关系变量R具有标题H，并设  $\{X_1, \dots, X_n\}$  是一个关于H的JD，比如J，那么当且仅当满足R的键约束的每一个关系r同时也满足J时，J被R的键所蕴含。

这个定义需要一些阐述。首先，说某个关系满足某个键约束也就是说，它满足适用的唯一性要求，并且如果它满足一些构成键的属性的唯一性要求，它肯定满足那个属性集合的每个超集的唯一性要求（当然，只要该超集是相关标题的一个子集），换句话说，满足每一个相应的超键的唯一性要求。因此，定义中的“满足R的键约束”那句话可以替换为“满足R的超键约束”，而不会造成任何明显的区别。同样，概念“被键蕴含”也可以替换为“被超键蕴含”，而不会造成任何明显的区别。

其次，如果定义中提到的JD J是平凡的，会发生什么情况呢？那么，在这种情况下，根据定义，每一个具有标题H的关系r满足J，而更不用

说因此被每一个满足R的键约束关系r肯定满足J。因此，平凡的JD总是平凡地“被键蕴含”。

再次，那么，假设J是非平凡的。我们如何确定某个非平凡的JD是否被某个关系变量的键蕴含呢？这个问题确实有一个令人满意的答案，但它有点复杂，因为这个原因，我会推迟到下一节再讨论它。在此之前，我想给5NF下一个定义，并对该定义作一些解释。

■定义：当且仅当关系变量R的每个JD都被R的键蕴含时，R属于第五范式（5NF），也称为投影-连接范式（PJ/NF）。

应该明确的是，如果一个JD被R的键所蕴含，那么可以肯定它存在于R中（即，它肯定是“R的一个JD”）。但是反之则不成立：一个JD可以存在于R中而不被R的键所蕴含，换句话说，5NF的定义的整个要点是：在一个5NF关系变量中存在的JD仅是我们不能摆脱的JD——这意味着那些被键蕴含的JD（包括那些作为一个特例的平凡JD）。[\[1\]](#)

我想通过指出BCNF和5NF定义之间的一个直观上吸引人的类似点结束本节：

■当且仅当R中存在的每一个FD都被R的键蕴含时，R属于BCNF。

■当且仅当R中存在的每一个JD都被R的键蕴含时，R属于5NF。

然而，它们也存在一个显著的差异。在BCNF的定义中，我们可以把“被键蕴含”这句话简化为“被某个键蕴含”，不严格地说，这意味着，每个存在的FD都是从某个孤立地考虑的特定键出来的箭头（当然，对于每一个这样的FD，不一定是相同的键）。相比之下，没有适用于5NF的这种简化——存在的JD是被作为整体的键的组合蕴含，不一定只是被某个孤立考虑的键蕴含。例如，假设在某一时刻关系变量有两个键，{SNO}和{SNAME}。那么以下JD（我们已经重复见过几次的JD）：

---

---

✧{{SNO,SNAME}, {SNO,STATUS}, {SNAME,CITY}}

---

---

将在这个关系变量中存在。但是，实际上它不存在，因为{SNAME}其实不是一个键。练习：构造一些示例数据来证明这些说法的真实性。

[1]像往常一样，“摆脱”一个（任何种类的）依赖的真正含义是用某个多关系变量约束替代它。

## 10.3 被键蕴含的JD

那么，如何才能确定给定的非平凡JD是否被键所蕴含呢？其实，有一个做这项工作的算法，即成员（membership）算法（由Fagin（费金）提出）。它的工作原理是这样的：设关系变量R具有标题H，且设 $\bowtie\{X_1, \dots, X_n\}$ 是一个关于H的JD，比如J，那么：

- 1.如果J的两个不同的分量都包含R的相同的键K，那么在J中把它们替代为它们的并集。

- 2.重复上面的步骤，直到再没有进一步的替代是可行的。

那么当且仅当J现在是平凡的，即，当且仅当最终版本的J含有H作为它的一个分量时，原来的JD才被R的键所蕴含。[\[1\]](#)（顺便说一下，请注意，正是那个平凡的JD导致算法轻易地取得成功。）

让我们来看看几个例子。首先，考虑我们平时的供应商关系变量S，现在有另一个JD，让我们把它称为J1，它存在于这个关系变量：

---

---

$\bowtie\{\{SNO, SNAME\}, \{SNO, STATUS\}, \{SNO, CITY\}\}$

---

通过反复应用希思定理，我们已经知道在S中存在这个JD。然而，注意它的分量{SNO,SNAME}和{SNO,STATUS}现在都包含键{SNO}，因此，应用成员算法，我们可以用它们的并集{SNO,SNAME,STATUS}来替代它们。J1现在看起来像这样：

---

$\odot \{ \{ \text{SNO}, \text{SNAME}, \text{STATUS} \}, \{ \text{SNO}, \text{CITY} \} \}$

---

注意：（a）这个修订版本的J1本身是一个关于关系变量S的标题的JD，而且（b）关系变量S也服从它——这两项事实结合在一起应该使我们对这个算法的内部机制有些了解（参见后面的进一步解释）。

其次，后一个JD的分量{SNO,SNAME,STATUS}和{SNO,CITY}也都包含键{SNO}，这样我们还可以用它们的并集{SNO,SNAME,STATUS,CITY}来替代它们，从而得到：

---

$\odot \{ \{ \text{SNO}, \text{SNAME}, \text{STATUS}, \text{CITY} \} \}$

---

这个进一步修订的J1还是一个关于S的标题的JD。然而，这都是说，关系变量S等于其恒等投影的“连接”（回顾一下习题5.1，单个关系r的连接 $\text{JOIN}\{r\}$ 恒等于r）；换句话说，进一步修订的J1，只是说，S可以“无损地分解”成其恒等投影。但这种观点显然是成立的：任何关系变量都始终可以“无损地分解”成其恒等投影，正如我们在第6章中所看到的。事实上，现在这个JD是形式上平凡的，因为它包含一个等于相关标题的分量。因此，最初说明的JD J1是被关系变量S的键蕴含的。

再举一个反例，现在考虑以下也在关系变量S中存在的JD——让我们把它称为J2：

---

---

$\odot\{\{SNO, SNAME, CITY\}, \{CITY, STATUS\}\}$

---

---

因为此关系变量的唯一的键， $\{SNO\}$ ，肯定不同时包含在这个（二元）JD的两个分量中，成员算法对其没有作用。因此，该算法的输出等于输入（即保持原JD J2不变）；输出的任何分量都不等于整个标题，所以J2并没有被键蕴含（因此，关系变量S不属于5NF）。

最后，让我们考虑一些更抽象的例子。设关



系变量R具有属性A、B、C、D、E和F（只有这些属性）；设R具有键{A}、{B}、{C,D}，并且没有其他键，并设AB表示属性{A,B}的集合，其他属性的名称组合的处理与此类似（“希思符号”见第7章）。现在考虑以下JD：

1.  $\odot\{AB, ACDE, BF\}$

2.  $\odot\{ABC, ACD, BEF\}$

3.  $\odot\{AB, AC, ADEF\}$

4.  $\odot\{ABC, CDEF\}$

5.  $\odot\{ABD, ACDE, DF\}$

对这些JD应用成员算法，我们发现，第1~3个JD是被键蕴含的（因此，R一定服从它们），而第4~5个JD却不是被键蕴含的。简要地阐述一下：第1个和第2个JD都被一对键{A}和{B}共同蕴含，但不被任何单独的键蕴含；相反，第3个JD被孤立地考虑的键{A}蕴含。当且仅当{C}是一个键时，第4个JD被键蕴含（实际上被单独的键蕴含），但{C}不是一个键；更重要的是，那个JD不可能在R中存在，因为如果它存在于R中，那么无论如何{C}必须是一个键（想想它

吧)。至于第5个JD，它显然不是键所蕴含的，它可能会也可能不会在R中存在，但如果它在R中存在，那么R不属于5NF。

那么，在这些例子中究竟是怎么回事？让我试着解释一下我一直在说的东西背后凭的是什么样的直觉，（你可能想利用供应商关系变量S和JD $\bowtie$ { {SNO,SNAME}, {SNO,STATUS}, {SNAME,CITY} }的方面尝试解释随后发生的事，重申一下，假设{SNO}和{SNAME}为该关系变量的两个键）：

■设 $X_1, \dots, X_n$ 是关系变量R的标题H的子集，并且 $X_1, \dots, X_n$ 的并集等于H。

■设J是JD $\bowtie$ { $X_1, \dots, X_n$ }，并设J被R的键所蕴含。

■设r是R的当前值（代表）的关系。

■在集合{ $X_1, \dots, X_n$ }中任意选择两个不同的元素（分量），比如 $X_1$ 和 $X_2$ 。

■设 $r_1$ 和 $r_2$ 分别是r在 $X_1$ 和 $X_2$ 上的投影。

现在，如果 $X_1$ 和 $X_2$ 两者都包含R的同一个键K，那么 $r_1$ 和 $r_2$ 的连接 $r_{12}$ （其标题 $X_{12}$ 将是 $X_1$ 和

X2的并集）将是一个严格一对一的连接，并且因此 $r_1$ 和 $r_2$ 可以被 $r_{12}$ 替换而不损失信息。（同时，在J中的X1和X2可以被X12替换。）由于原来版本的J是被R的键蕴含的，按照定义，重复地进行这样的替换，最终将产生一个关系（a）它等于原来的关系 $r$ ，尤其是（b）它将因此具有一个等于整个标题H的标题。

现在让我指出，到目前为止我说的东西在相关的关系变量R只有一个键K这种常见的特殊情况下，会变得更加简单。在这种情况下，当且仅当以下两点成立时， $JD \bowtie \{X_1, \dots, X_n\}$ 被键所蕴含：

a.至少 $X_1, \dots, X_n$ 之一包含R的每个属性。（当然，在一般情况下以及在这个特殊的情况下，这要求总是适用的）。

b. $X_1, \dots, X_n$ 中的每一个都包含R的唯一键K——换言之， $X_1, \dots, X_n$ 中的每一个都是一个超键。

因此，如果R只有一个键K，那么当且仅当R中存在的每个JD的每一个分量都包含该键K时，R属于5NF。[\[2\]](#)（但是，请注意，最重要的！我这里假设仅考虑关于R不可约的JD，请参阅第11

章。) 举例来说, 考虑零件关系变量P, 在此关系变量中存在的唯一的不可约JD  $\bowtie \{X_1, \dots, X_n\}$  使得每个  $X_i$  ( $i=1, \dots, n$ ) 都包括唯一的键  $\{PNO\}$ 。这些JD显然都被那个唯一的键蕴含, 因此, 关系变量P属于5NF。下面是所涉及的JD之一:

---

---

$\bowtie \{ \{PNO, PNAME, COLOR\}, \{PNO, WEIGHT, CITY\} \}$

---

---

因此, 关系变量P可以无损地分解成其在JD的分量上的投影。至于我们是否实际上要执行那个分解, 当然是另一回事。只要我们知道如果我们想做我们就可以这样做, 这就够了。

让我通过重新审视来自第9章的SPJ例子来结束本节。为方便起见, 那个关系变量的一个示例值如图10-1所示(重复图9-1的内容)。谓词是供应商SNO供应零件PNO给JNO工程, 并且下面的业务规则是有效的:

■如果供应商s供应零件p且零件p被提供给工程j且工程j是由供应商s供应的, 那么供应商s供应零件p给工程j。

SPJ	SNO	PNO	JNO
	S1	P1	J2
	S1	P2	J1
	S2	P1	J1
	S1	P1	J1

图 10-1 关系变量SPJ——示例值

现在，我们从第9章（如那一章所述）知道，下面的JD抓住了此业务规则的本质，且因此存在于SPJ中：

---


$$\odot\{\{SNO,PNO\}, \{PNO,JNO\}, \{JNO,SNO\}\}$$


---

现在，我们可以看到这个JD并不被此关系变量唯一的键（即 $\{SNO,PNO,JNO\}$ ）蕴含，因为成员算法失败，所以SPJ不属于5NF。因此，它可以无损地分解成它的三个二元投影，并且如果我们减少冗余，可能应当这么分解。这三个投影都属于5NF（它们根本不存在非平凡的JD）。

[1]在实践中，当然，我们可能并不需要一路走到最后并计算最终版本的J，我们可以在产生一个等于H的分量时尽快退出。

[2]注意，对于关系变量S，情况并非如此——那个关系变量服从至少一个JD，该JD至少有一个分量不包括唯一键{SNO}（即， $\{ \{SNO, SNAME, CITY\}, \{CITY, STATUS\} \}$ ），所以它不属于5NF。

## 10.4 一个有用的定理

第9章指出，在实践中，发现一个关系变量属于BCNF，且不属于5NF，是相当不寻常的。事实上，有一个解决这个问题的定理。

■定理：设R是一个BCNF关系变量，并设R没有复合键，则R属于5NF。（回顾一下第1章，一个复合键是由两个或多个属性组成的。）

这个定理是非常有用的。它说的是，如果你能得到BCNF（这足够容易），如果在你的BCNF关系变量中没有复合键（这是经常发生的，但并非总是如此），那么你一般不用担心JD及5NF的复杂性，无须进一步考虑此事，你就知道此关系变量就是属于5NF的。注意：其实这个定理适用于3NF，而不适用于BCNF，也就是说，它真正说的是，一个没有复合键的3NF关系变量属于5NF。但务实地说，每一个BCNF关系变量都属于3NF，并且在任何情况下BCNF比3NF更重要（并且在概念上更简单）。

警告：我不知道为什么，但人们往往误解了上述定理。具体而言，既然一个没有复合键的BCNF关系变量“自动”属于5NF，人们似乎常常认为，简单地给BCNF关系变量引入一个代理键

（按照定义是非复合的）“自动”表示关系变量现在属于5NF。但是，这个看法根本不对！如果在引入代理键前关系变量不属于5NF，那么在引入代理键之后它也不会属于5NF。特别是，如果它在引入代理键前有一个复合键，那么在引入代理键之后，它仍有一个复合键。



## 10.5 FD不是JD

每一个FD都是JD，或者（正如第9章所述）JD是一种广义FD的陈述，在不太正式的文献中是相当普遍的，事实上，在我以前的书籍和著作中，我自己就曾经说过这样的话。但是，这样的言论完全是不正确的。说每一个FD都蕴含一个JD（其实我们已经从希思定理中知道这个情况）将是更好的陈述。换句话说，如果R服从一个特定的FD，比如F，那么它肯定服从一个特定的JD，比如J。然而，相反的陈述是不正确的——R可以服从相同的JD J而不服从同样的FD F，如我现在显示的：

■设关系变量R具有属性A、B、C（只有这些属性），设F是FD  $AB \rightarrow C$ ，并设R服从F（再次使用希思符号）。

■根据希思定理，那么，R服从  $JD \star \{ABC, AB\}$ 。（参考在第9章给出的希思定理的描述，设X为AB，Y为C，且Z是属性的空集）把此JD称为J。

■但是，这个JD J是平凡的，其存在于每个具有标题ABC的关系变量R中，无论那个关系变量是否服从FD  $AB \rightarrow C$ 。

## 10.6 更新异常再探

第3章简单地介绍了FD（具体而言，在不属于BCNF的关系变量中存在的FD）可能引起的特定的更新异常。但是，坦率地说，更新异常的概念从来没有很精确的定义（至少，不在那种情况下），从另一个角度看，说更新异常的问题就是冗余问题，可能是最好的。那么，JD（具体而言，在不属于5NF的关系变量中存在的JD）引起的更新异常又是怎样的？正如我们已经看到的，这样的JD确实导致冗余，因此我们可以预计它们也会导致更新异常。事实上，它们确实会，更重要的是，在这种情况下，这个概念可以更精确地定义（或者，无论如何，它已经是更加精确了），正如我们将看到的。

考虑图10-2，它显示了关系变量SPJ两个可能的值，左侧的图所示的关系与图10-1所示的关系相同，右侧的图是从左侧的图去掉两个元组后得到的。


SPJ	SNO	PNO	JNO
	S1	P1	J2
	S1	P2	J1
	S2	P1	J1
	S1	P1	J1

SPJ	SNO	PNO	JNO
	S1	P1	J2
	S1	P2	J1

图 10-2 关系变量SPJ两个可能的值

现在注意，如果关系变量SPJ的当前值是图10-2左侧的关系，那就存在一个删除异常（deletion anomaly）：我们不能只删除元组（S1，P1，J1），因为那个删除执行后的结果违反JD，并因此不是此关系变量的一个合法的值。同样，如果关系变量SPJ的当前值是图10-2右侧的关系，那就存在一个插入异常（insertion anomaly）：我们不能只插入元组（S2，P1，J1），因为该插入执行后的结果（再次，出于同样的原因）不是此关系变量的一个合法值。

现在，在这个例子中是JD是元组强迫的。（回顾一下第9章，如果一个JD要求“如果某些元组出现，某些额外的元组被迫也出现”，那么它是元组强迫的。）而元组强迫的JD概念（或不如说这一概念背后的直觉）允许我们给出这种更新异常的定义，它们在存在这样一个JD时可能发生

——实际上是比它们的FD对应物更精确的定义。具体的定义如下：

■定义：设在关系变量R中存在JD J，那么当且仅当存在一个关系r和一个元组t，它们每个都与R有相同的标题，并且符合下面两个条件时，R存在关于J的删除异常：

a.r满足J，并且

b.关系r'（其正文是从r中去除t）违反J。

■定义：设在关系变量R中存在JD J，那么当且仅当存在一个关系r和一个元组t，它们每个都与R有相同的标题，并且符合下面两个条件时，R存在关于J的插入异常：

a.r满足J，并且

b.关系r'其正文是从r中追加t的满足R的键约束，但违反J。

由此产生的要点如下所示。

■请仔细注意上述异常都是明确地依据某个JD J定义的，并且如果J是元组强迫的，那么它们肯定会发生，正如我们已经看到的那样。然而，

在第13章中，我们将会看到，即使J不是元组强迫的，关系变量也可能存在关于J的插入异常（虽然不是删除异常）。

■虽然它们的定义比它们的FD对应定义更精确，但上述异常仍可以被视为从另一个角度看的冗余问题——当然，虽然在这里，我们指的是由一个JD造成的冗余，而不是由一个FD造成的冗余。

■如果关系变量R存在更新异常，且这些异常是由一个JD（元组强迫的或其他的）导致的，那么用一个5NF投影集合替代R将解决这个问题。也就是说，这样的异常不能发生在5NF关系变量中。

但是，请注意，不是所有的更新异常都是FD和JD造成的。事实上，在始终存在一个其插入会导致违反所涉及的约束的元组这个意义上，说完整性约束（尽管不是全部）可能会引起插入异常可能是正确的。（举一个简单的例子，假设有一个大意是供应商的状态值必须在范围1~100（含上下限）之间的约束）。与此相反，相对较少的限制，可能会引起一个删除异常。（一种情况是大意是必须始终至少有两个不同的供应商的约束。另一种情况是外键约束。例如，在供应商和

零件数据库中，如果删除供应商会导致违反相关的外键约束，那么就不能这样做。）

[1]它们可能更精确，但它们也略有可疑，因为它们谈论插入或删除一个单独的元组。正如在《SQL与关系数据库理论》中解释的，INSERT和DELETE实际上是对整个关系，而不是单独的元组工作的。

## 习题

10.1 以下问题与第1章的习题重复，但现在，你应该有能力更好地回答它们（假设之前你不能回答）。

a.（习题1.6）每一个“全键”关系变量都满足5NF，这是否正确？

b.（习题1.7）每一个二元关系变量都满足5NF，这是否正确？

c.（习题1.8）如果关系变量只有一个键和一个其他属性，那么它满足5NF，这是否正确？

d.（习题1.9）如果一个关系变量满足BCNF，但不满足5NF，那么它必须是全键，这是否正确？

e.（习题1.10）你能给出5NF的一个确切的定义吗？

f.（习题1.11）如果关系变量满足5NF，那么它是没有冗余的，这是否正确？

10.2 尽你所能地给出关系变量服从一个连接依赖的含义的精确定义。

10.3 在发货关系变量SP中存在多少JD呢？

10.4 说一个JD被超键蕴含是什么意思？

10.5 什么是一个平凡的JD？一个平凡的FD是一个平凡的JD的特例吗？

10.6 请举一个（a）元组强迫的JD例子，（b）非元组强迫的JD例子。

10.7 考虑在第9章习题9.3的答案中所讨论的关系变量RAP。给出可以在此关系变量中发生的插入异常和删除异常的例子。

10.8 以下内容引用自某本数据库教科书：“第五范式涉及的依赖关系是默默无闻的。它与可分解为子关系（正如我们一直在做的），但不能重构。出现这种情况的条件没有明确的、直观的意义。我们不知道这种依赖关系会引起什么样的后果，甚至不知道它们是否有任何实际后果。”你对此有什么看法？



## 第11章 隐式依赖关系

What are you implying?

——20th century catchphrase

在前面的章节中，我们已经看到了一些有关一定的依赖蕴含其他依赖的观点的说明。具体而言，我们在第7章中看到了FD是如何被其他FD蕴含的，我们在第9章和第10章看到了JD是如何被FD蕴含的。现在是时候去仔细研究相关事宜了。（特别要注意的是，如果我们需要说出某个关系变量属于什么范式，那么我们需要知道该关系变量中存在的所有依赖，包括隐含的和明确的。）因此，本章将讨论其他一些事情，包括：

■无关的JD分量

■结合JD分量

■不可约的JD

■添加JD分量

这些讨论将为解释什么是所谓的追逐（the chase）铺平道路，它将在11.5节描述。

## 11.1 无关的分量

再次考虑关系变量S和它的FD{CITY}  $\rightarrow$  {STATUS}。正如我们从前面的章节所知道的：

■此关系变量可以无损地分解成其在 {SNO,SNAME,CITY} 和 {CITY,STATUS} 上的投影。

■显然它也可以无损地分解成上述两个投影以及在（比如）{SNAME,CITY}上的投影。

■然而，第三个投影在重构原始关系变量的过程中显然是不需要的。

现在，让我用JD的方式重新描述上述例子：  
关系变量S服从JD

---

---

$\bowtie \{ \{ \text{SNO}, \text{SNAME}, \text{CITY} \}, \{ \text{CITY}, \text{STATUS} \} \}$

---

---

并且也服从JD

---

---

$\bowtie \{ \{ \text{SNO}, \text{SNAME}, \text{CITY} \}, \{ \text{CITY}, \text{STATUS} \}, \{ \text{SNAME}, \text{CITY} \} \}$

---

---

然而，在后一个JD中， $\{SNAME, CITY\}$ 分量是无关紧要的：它是另一个分量的一个真子集，而且由于这个原因，对应的投影在重建原始关系变量的过程中是不需要的。

受上述例子的启发，我现在可以给某个JD中的某个分量是无关紧要的是什么含义下一个准确的定义。

■定义：设 $\{X_1, \dots, X_n\}$ 是一个JD，比如J，那么当且仅当（a）在J中存在某个 $X_j$ ，使得 $X_i$ 是 $X_j$ 的一个真子集或（b）在J中存在某个 $X_j$ （ $i < j$ ），使得 $X_i = X_j$ 时， $X_i$ 是在J中无关的（irrelevant）分量。[\[1\]](#)

我选择术语无关的原因如下：如果 $X_i$ 是在J中无关的，那么每一个满足J的关系，同时也满足J'，其中J'是从J中去掉 $X_i$ 后得到的关系。更重要的是，反过来也是正确的：每一个满足J'的关系也满足J。换句话说，JD J和J'是等价（equivalent）的：每个满足其中一个的关系必定也满足另一个。因此，无关的分量不仅始终可以去掉，它们还始终可以添加，而不会造成明显影响。

[\[1\]](#)如果我们假设分量 $X_1, \dots, X_n$ 都是不同

的，那么我们可以去掉这个定义的（b）部分。

## 11.2 结合分量

现在，我们已经看到，一些JD蕴含了其他JD（就像有些FD蕴含了其他FD）。但无关的分量距离这个故事的结束还很远。下一点如下（我把它标记为一个定理，但它很明显并几乎配不上如此庄重的一个名称）。

■定理：设J是一个JD，并设J'是通过把J的两个分量替换成它们的并集而得的，那么J蕴含J'（即，每一个满足J的关系也满足J'）。

举例来说，每一个满足以下JD的关系变量S的合法值（这是来自上一节的JD，它带有一个无关的分量）：

---

---

$$\odot\{\{SNO, SNAME, CITY\}, \{CITY, STATUS\}, \{SNAME, CITY\}\}$$

---

---

因此也满足下面这个JD：

---

---

$$\odot\{\{SNO, SNAME, CITY\}, \{CITY, STATUS, SNAME\}\}$$

---

---

习题：请检查上述声明的有效性，若有可能甚至试图正式地证明这一点，如果它不是显而易

见的。（另外，通过以这种方式结合分量，从给定的一个JD可以推导出多少个不同的JD？）由此产生的要点如下。

■当我在第10章中解释成员算法（用于测试JD是否被键蕴含）背后的直觉时，我心照不宣地利用了上述定理。

■注意，该定理包括“蕴含”这个词，而不是“等价”这个词： $J$ 蕴含 $J'$ ，但反过来是不成立的——一般情况下， $J'$ 不蕴含 $J$ ，并因此 $J$ 和 $J'$ 是不等价的（再次，在一般情况下）。注意，其实这点是很容易看到的：如果我们持续地把分量替换为它们的并集，我们最终会得到一个与整个标题相等的分量，且因此产生的JD  $J'$ 将是平凡的——而每个JD都等于某个平凡的JD显然是不正确的。

■虽然不可否认，上面所示的两个JD中的第二个（二元的那个）在关系变量 $S$ 中存在，但在那个JD的基础上无损地分解此关系变量不会是一个好主意。注意：习题11.4要求你进一步解释这一观点，但你可能会想花一点时间来说服自己，这是正确的。另外，让我提前片刻透露，我可以说，所涉及的JD其实是关于 $S$ 不可约的（见下一节）。因此，这个例子说明了，虽然不可约JD是很重要的，但它们不一定对应于良好的分解。非

正式地，换句话说，我们需要区分“好”与“坏”的JD，这里的“好”和“坏”指的是相应分解的质量。如需进一步讨论，请参阅第14章。

## 11.3 不可约的JD

到目前为止，一个JD蕴含另一个JD的概念在本质上或多或少合乎语法的——我还没有真正花费太多精力研究我们正在谈论的JD在某个特定的关系变量中存在的问题。（注意，无论是无关的分量的定义，还是关于把分量用它们的并集替换的定理，都没有提到一个关系变量，甚至也没有提到一个标题。）但是现在，让我们考虑确实在某个关系变量中存在的JD。那么，我们有下面的定理：

■定理：设JD  $J$ 在关系变量 $R$ 中存在；那么 $J$ 等价于也在 $R$ 中存在的某个不可约JD（不一定是唯一的）。然而，注意，这里的等价应当在某个特定的关系变量的上下文中理解；对于两个JD，它们同时在一个关系变量中存在，却不同时在另一个关系变量中存在，这是可能的。在这样的情况下，相对于第一个关系变量，这两个JD可能等价也可能不等价，但相对于第二个关系变量，它们肯定是不等价的。

我将确切地解释在某一时刻一个JD不可约到底是什么意思。但是，在我这样做之前，我想只是提醒你它与FD的不可约有一些类似。回顾一下本书第二部分，在关系变量 $R$ 中存在的每一个FD



都蕴含同样在关系变量R中存在的某个不可约FD。（这点是容易看出的：只要不断地从决定因素中去掉一些属性，直到剩下的是关系变量中不再存在的一个FD。）同样，在关系变量R中存在的每个JD也都蕴含了（事实上，等价于（一个更强的声明）——某个也在关系变量R中存在的不可约JD。

那么，JD不可约是什么意思呢？这里有一个定义：

■定义：设 $\bowtie\{X_1, \dots, X_n\}$ 是关系变量R中存在的一个JD，比如J，并设不存在 $\{X_1, \dots, X_n\}$ 的一个真子集 $\{Y_1, \dots, Y_m\}$ 使得 $\text{JD}\bowtie\{Y_1, \dots, Y_m\}$ 也在R中存在，则J是关于R不可约的（或只是不可约的，如果R是不言自明的）。

由此产生的要点如下所示。

■容易看出，在关系变量R中存在的每个JD都蕴含也在关系变量R中存在的一个不可约JD——只要不断地从给定的JD中去掉一些分量，直到剩下的是在R中不再存在的一个JD，那么在R中存在的最后一个JD是不可约的。

■从另一个方式也很容易看出这个蕴含：从不可约JD开始，然后持续地把去掉的分量添加回去，直到达到原来的JD。在这个过程中的每一步，JD的当前版本都将是R中存在的一个JD。注意：将这一要点和以前的要点结合在一起，就可以得出如下结论：（a）在R中存在的每个JD都等价于在R中存在的某个不可约JD（事实上，如前面提到的），并因此，（b）在R中存在的不可约JD其实蕴含在R中存在的所有JD。

■如果在J中某个分量Xi是无关的，那么对于每一个存在于J中的关系变量，J肯定是可约的（因为Xi可以去掉，而不造成显著损失）。但是，正如我现在显示的，即使所有分量都是有关的，对于一些关系变量，J可能仍然是可约的。

再次考虑供应商关系变量S。然而，为了简单起见，让我们同意忽略属性SNAME，更重要的是，让我们同意用“S”这个名字来表示这个精简版本的关系变量，直至另行通知为止。现在考虑下列JD：

---

---

$$\odot\{\{SNO,CITY\}, \{CITY,STATUS\}, \{SNO,STATUS\}\}$$

---

---

让我们把这个JD称为J1，它没有无关的分

量。不过，我将会显示：（a）它在关系变量S中存在，但（b）它对于此关系变量是可约的，因为{CITY,STATUS}分量可以去掉，剩下的仍然是S的一个JD。注意：实际上，在这个例子中的可约性在直观上是显而易见的，因为（要精确地说明此事项）S在{CITY,STATUS}上的投影显然等于S{SNO,CITY}和S{SNO,STATUS}的连接在{CITY,STATUS}上的投影。因此，{CITY,STATUS}分量实际上没有添加任何东西。再次重复，因此：可约性是“显而易见”的，但现在我要证明这一点。

a.那么，首先，假设下列元组出现在S中（元组的简化表示法；s1和s2表示供应商的编号，c1和c2表示供应商的城市，t1和t2表示状态值）：[\[1\]](#)

---

s1	c1	t2
s1	c2	t1
s2	c1	t1

---

因为{SNO}是一个键，所以在S中存在下面的FD：

---

{SNO}	→	{CITY}
{SNO}	→	{STATUS}

---

因此，我们可以得出 $c1=c2$ 和 $t1=t2$ 的结论，并且下面这个元组：

---

---

$s1\ c1\ t1$

---

---

必然在S中出现，因为实际上它与上图所示的原始列表中的第一个元组是相同的（或者，与第二个元组同样是相同的）。但是，如果你明白我的意思，说原来的“三个”元组导致“第四”个元组出现，也就是说，准确地说，JD J1（即J1的内容）存在。因此，J1确实在S中存在。

b.现在，根据 $FD\{SNO\} \rightarrow \{CITY\}$ 和 $\{SNO\} \rightarrow \{STATUS\}$ （这两者都在S中存在，据我们所知）中的任何一个和希思定理，我们可以得出，下面这个JD——让我们称它为J2——必然在关系变量S中存在：

---

---

$\bowtie\{\{SNO,CITY\}, \{SNO,STATUS\}\}$

---

---

但是，J2的分量构成了J1的分量的一个真子集。因此J1对于S是可约的，分量 $\{CITY,STATUS\}$ 可以从J1中去掉，而没有任何损失，在这个意义上，剩下的仍然是S的一个JD。

现在观察，上述证明没有用到 $FD\{CITY\} \rightarrow \{STATUS\}$ 在S中存在的事实（事实上，即使该FD并不存在，结果仍然是有效的）。但现在让我们举一个确实用到了该FD的例子。

■首先，我们现在知道下面的JD（来自前面的例子中的J1）在S中存在：

---

---

$$\star\{\{SNO,CITY\}, \{CITY,STATUS\}, \{SNO,STATUS\}\}$$

---

---

■不过， $FD\{CITY\} \rightarrow \{STATUS\}$ 也存在，并因此根据希思定理，下面的JD，让我们称它为J3，也存在：

---

---

$$\star\{\{CITY,STATUS\}, \{CITY,SNO\}\}$$

---

---

■J3的分量构成了J1的分量的一个真子集，并且因此再次得出，J1对于S是可约的。具体地说，分量 $SNO,STATUS\}$ 可以从J1去掉，而没有任何损失，在这个意义上，剩下的仍然是S的一个JD。

因此，注意，对于关系变量S，原始JD J1等价于两个不同的JD：即，J2和J3。

当然，一般情况下，问题是：已知关系变量R和在R中存在的一个JD J，我们如何才能找到J的一个不可约等价物（关于这个问题的更精确的意义是，一个JD，它既与J等价，又是不可约的，其中等价和不可约都理解为相对于R的）？那么：

■如果在J中的某个分量是无关的，该分量显然可以去掉。

■如果所有分量都是相关的，我们只能尝试去掉其中一个，然后：

a.如果剩下的仍然是R的JD，我们把另一个分量去掉，并重复这个过程。

b.如果剩下的不是R的JD，我们恢复已删除的分量，并尝试去掉另一个分量。

最后，我们将得到一个JD，它等价于原来的JD，并且是不可约的。

我们怎么判断某个JD实际上是R的JD呢？那么，如果它是一个明确定义的JD，显然没有问题，但如果它没有明确定义，我们使用追逐法，我将在11.5节中描述它。

[1]注意，这些元组中的每一个是如何对应于给定的JD（J1）的一个分量的。

## 11.4 小结

让我来总结一下我们目前学到的知识。总的来说，一些JD蕴含其他的JD。具体来说，我们已经讨论了以下内容。

■无关的分量：每个JD  $J$  等价于一个通过从 $J$ 添加或去掉无关的分量得到的JD  $J'$ 。

■结合分量：每个JD  $J$  蕴含一个通过从 $J$ 把两个分量替换为它们的并集得到的JD  $J'$ 。

■不可约性：在关系变量 $R$ 中存在的每个JD  $J$  至少等价于一个JD  $J'$ （不一定与 $J$ 不同），它在 $R$ 中存在并且是不可约的（其中等价和不可约都必须理解为是相对于 $R$ 的）。因此， $R$ 的不可约的JD其实蕴含 $R$ 的所有JD。

下面的观点也是有效的（我还没有讨论细节，但它们很直观）。

■添加属性：如果JD  $J$ 在关系变量 $R$ 中存在，那么每个通过添加 $R$ 的某个属性到 $J$ 的某个分量，从 $J$ 得到的JD  $J'$ 也在关系变量 $R$ 中存在。

■添加分量：如果JD  $J$ 在关系变量 $R$ 中存在，



那么每个通过添加R的标题的任何子集作为另一个分量，从J得到的JD J'也在关系变量R中存在。

然而，注意，在这两种情况下，我们正在谈论的都是蕴含，而不是等价。例如，在关系变量S（但为了简单起见，再次忽略SNAME）中， $JD_{\odot}\{SNO,STATUS\}\{SNO,CITY\}$ 存在，因此下面的JD也存在： $\odot\{\{SNO,STATUS\},\{SNO,CITY\},\{CITY\}\}$ 。然而，反过来是不成立的，如果后一个JD存在，不能得出前一个JD存在的结论。[\[1\]](#)

我们也可以说：如果J是在关系变量r中存在的一个JD，且J蕴含另一个JD J'，其中J'是通过从J去掉J的分量中的一些属性，和/或删除J的整个分量得到的，那么J无疑是一个“坏”JD（参见11.2节底部有关好的和坏的JD的话题的言论）。然而，并不是所有的“坏”JD都可以用这个简单的方式获得，正如我们将看到的。

现在，我想在一定程度上概括这个讨论。首先，从这里开始，根据上下文要求，我将采用术语“依赖”来表示FD或JD或表示两者。[\[2\]](#)现在，整本书至今为止，每当我考虑依赖被其他依赖蕴含的问题时，我心照不宣地把我的注意力限制在被单独的依赖蕴含的那些依赖上。但是，更普遍

的，事实证明，某些依赖关系的集合可能蕴含其他依赖关系的集合。请让我举一个例子。

考虑一个关系变量SPT，它具有属性SNO、PNO、STATUS（只有这些属性），其中各个属性有其通常的意义。假设我们已知，并非不合理的，以下依赖（一个FD和一个JD）都在这个关系变量中存在：

---

$$\{SNO, PNO\} \rightarrow \{STATUS\}$$
$$\bowtie \{\{SNO, PNO\}, \{SNO, STATUS\}\}$$

---

现在，从条件的语义上看，可以明显地看出（a）{SNO}不是SPT的键，且（b） $FD\{SNO\} \rightarrow \{STATUS\}$ 在SPT中隐式地存在（顺便说一句，因此SPT不属于2NF）。请注意，我说“隐式”，是因为没有明确告知我们该FD存在。现在的问题是：只根据明确说明的FD和JD存在，我们可以证明（a）和（b）吗？也就是说，不理睬任何语义，我们可以显示（a）和（b）在形式上是有效的吗？（毕竟，如果我们希望系统能够推断出依赖关系，这就是它将不得不做的工作。当我在这里使用“语义”这个术语时，系统不知道有关它的任何东西。）[\[3\]](#)

所以，让我们尝试一下。首先，假设下列元组出现在SPT中：

---

```
s1 p1 t2
s1 p2 t1
```

---

也假设 $p1 \neq p2$ 。为了显示 $FD\{SNO\} \rightarrow \{STATUS\}$ 存在，现在我们需要做的就是显示 $t1$ 和 $t2$ 必然是相等的。我们首先记录这两个元组对应于给定JD $\bowtie \{\{SNO, PNO\}, \{SNO, STATUS\}\}$ 的分量的投影：

---

```
s1 p1 s1 t2
s1 p2 s1 t1
```

---

把这些投影连接在一起，我们得到了原始的两个元组以及两个额外的元组（在下面显示为粗体）：

---

```
s1 p1 t2
s1 p1 t1
s1 p2 t2
s1 p2 t1
```

---

因为给定的JD存在，两个额外的元组实际上

必须与原来的两个一起出现在此关系变量中。不过， $FD\{SNO, PNO\} \rightarrow \{STATUS\}$ 也存在，因此， $t_1=t_2$ ，并因此， $SNO \rightarrow FD\{SNO\} \rightarrow \{STATUS\}$ 存在（每一个SNO为s1的元组也具有STATUS t1）。这是待证明的（b）部分。同时，根据假设，有 $p_1 \neq p_2$ （注意，在论证中至今没有使那个假设无效），因此 $FD\{SNO\} \rightarrow \{PNO\}$ 不存在，所以 $\{SNO\}$ 不是一个键，这就是待证明的（a）部分。

所以我们看到，任何给定的关系变量都服从显式依赖（这些都是显式声明的）和隐式依赖（这些是显式声明的依赖蕴含的）。为了记录在案，让我把这些点要合并成一个适当的定义。

■定义：设R是一个关系变量。与R相关的显式依赖有两个集合：在R中存在的显式FD的一个XFD集合和在R中存在的显式JD的一个XJD集合。XFD中的FD和XJD中的JD一起构成R的显式依赖。那些不在XFD或XJD中，却是XFD和XJD中的依赖的逻辑后果的FD和JD是R的隐式依赖。R的显式依赖和R的隐式依赖一起构成R的依赖。只有当关系r满足R的所有依赖时，才可以把它赋值给R。

[1]当然，前一个JD确实在我们运行的例子中存

在，这是由于 $FD\{CITY\} \rightarrow \{STATUS\}$ 也存在。但一般后一个JD并不蕴含前一个。

[2]正如我们所知，其他类型的依赖（例如，相等依赖，这是第3章中和其他地方顺便提到的，并在第13章中进一步讨论）也存在，但在这个时候我故意将其排除在考虑之外。

[3]我顺便记录，证明的（b）部分将立即由习题11.3，即，作为希思定理的一个扩展版本的逆命题得出。

## 11.5 追逐算法

到目前为止，我们已经从本章看到的一切，提出了下面这个很明显的问题。

已知某个依赖（FD或JD或两者的混合物）的集合D，被这个集合里的那些依赖蕴含的依赖d都是什么内容？

这个问题的部分答案，正是追逐算法（the chase algorithm），准确地说，它是测试某个依赖d是否是由依赖的集合D蕴含的一个算法，更具体地说，已知集合D和某个依赖d，追逐算法将产生以下结果之一。

a.说明D蕴含d，

b.通过提供一个明确的反例，也就是，一个满足D中所有的依赖，但违反d的关系，来说明D不蕴含d。

事实上，我们其实已经在行动中看到了追逐的一些例子。前一节展示了一个给定的FD和JD是如何共同蕴含一个特定的FD，却没有蕴含另一个FD的（后者实际上是一个键约束，当然，它是FD的一个特例）。而在这之前的一节中，我给出

了两个例子，其中一个给定的FD和JD一起蕴含一个特定的JD（顺便说一句，因此显示给定的JD是可约的）。所有这些例子其实都是追逐的应用。但现在让我们更具体地研究。为了做到这一点，首先需要引入更多一点的术语。

■考虑FD。抽象地（虽然当然很不严格）说，一个FD具有以下形式：“如果出现某些元组  $t_1, \dots, t_n$ ，那么这些元组的某些属性必须具有相同的值。”出于这个原因，FD有时说成是相等生成（equality generating）依赖。

■现在考虑JD。抽象地说，但同样非常不严格，一个JD具有以下形式：“如果出现某些元组  $t_1, \dots, t_n$ ，那么一个特定的元组  $t$  必须也出现。”因此，JD有时说成是元组生成（tuple generating）依赖。

在进一步讨论之前，我必须提醒你，不要将元组生成依赖和元组强迫依赖混淆。[\[1\]](#)一个元组强迫依赖是一个JD，它具有的属性是：如果出现元组  $t_1, \dots, t_n$ ，那么有别于  $t_1, \dots, t_n$  中任何一个的某个元组  $t$  被迫出现。与此相反，一个元组生成依赖（a）不要求“生成的”元组与给定的元组是不同的，并且（b）实际上根本不必是一个如前所述的JD。（但是，本书所讨论的唯一的元

组生成依赖确实明确是JD。因此，就目前而言，你可以把“元组生成依赖”当作是指一个JD，因此，我们可以说，所有元组强迫依赖都是元组生成依赖，但一些元组生成依赖不是元组强迫依赖。）

相等生成依赖和元组生成依赖都包括一个前提集合（即元组 $t_1, \dots, t_n$ ）和一个结论。对于一个元组产生依赖，得出的结论是生成的元组 $t$ ；而对于相等生成依赖，结论是有一个特定的等式成立的事实。

现在，我可以解释如前所述的追逐算法了。也许我首先应该说，这是基本常识，事实上，它往往更容易演示，而不是描述。但是，概括地说，它的工作原理是这样的。我们正试图确定依赖 $d$ 是否由集合 $D$ 中的依赖得出。我们按照如下步骤进行：

- 1.我们写下代表 $d$ 的前提的元组。

- 2.我们把 $D$ 中的依赖应用到这些元组（可能产生额外的元组），并不断重复这个过程，直到没有进一步的变化发生为止。

整个过程最终将产生如下结果之一：



a.d的结论的一种表示方式，在这种情况下，d确实由D得出，

b.一个满足D但不满足d的关系，在这种情况下，d不由D得出。

让我们编写一个例子。设给定的依赖关系集合如下（再次使用希思符号）：

---

---

$$\{A \rightarrow C, B \rightarrow C, C \rightarrow D, CE \rightarrow A, DE \rightarrow C\}$$

---

---

（其实它们都是FD，正如你可以看到的。）考虑下面的JD（把它称为J）：

---

---

$$\odot \{AB, AD, AE, BE, CDE\}$$

---

---

现在，我会证明给定FD其实蕴含J（我认为你会同意，这不是一个显而易见的状况）。

第一步是把代表JD J的前提的元组写下来，现在，让我详细说明这个JD指的到底是什么东西。

如果以下的所有情况属实：

- 一个具有 $A=a$ 和 $B=b$ 的元组出现,
  - 一个具有 $A=a$ 和 $D=d$ 的元组出现,
  - 一个具有 $A=a$ 和 $E=e$ 的元组出现,
  - 一个具有 $B=b$ 和 $E=e$ 的元组出现,
  - 一个具有 $C=c$ 和 $D=d$ 和 $E=e$ 的元组出现,
- 那么

■一个具有 $A=a$ 和 $B=b$ 和 $C=c$ 和 $D=d$ 和 $E=e$ 的元组必须出现。

然而, 事实证明, 不用 $a$ 、 $b$ 、 $c$ 、 $d$ 和 $e$ 表示, 而用带后缀的 $x$ 和 $y$ 表示属性值更便于使用。具体地说, 我将使用 $x_1 \sim x_5$ 分别代替 $a \sim e$ , 且在所有其他位置我会用带后缀的 $y$ , 例如, 我将使用 $y_{23}$ 表示在“第二个”前提元组中的“第三个”或 $C$ 值。因此, 全部前提元组看起来是这样的: [\[2\]](#)

---

```

x1 x2 y13 y14 y15
x1 y22 y23 x4 y25
x1 y32 y33 y34 x5
y41 x2 y43 y44 x5
y51 y52 x3 x4 x5

```

---

那么，当（且仅当）五个FD蕴含此JD，这些元组将“生成”下面这个元组：

---

x1 x2 x3 x4 x5

---

让我们看看它是否能生成这个元组，即，让我们应用给定的依赖。

■从 $A \rightarrow C$ ，我们可以得出 $y_{13}=y_{23}=y_{33}$ ，同样，从 $B \rightarrow C$ ，我们可以得出 $y_{13}=y_{43}$ 。因此，我们可以把 $y_{23}$ 、 $y_{33}$ 和 $y_{43}$ 都用 $y_{13}$ 替换。前提元组变成如下所示（替换以粗体显示）：

---

x1 x2 y13 y14 y15  
x1 y22 y13 x4 y25  
x1 y32 y13 y34 x5  
y41 x2 y13 y44 x5  
y51 y52 x3 x4 x5

---

■从 $C \rightarrow D$ ，我们得出 $y_{14}=y_{34}=y_{44}=x_4$ 。进行替换后如下：

---

x1 x2 y13 x4 y15  
x1 y22 y13 x4 y25  
x1 y32 y13 x4 x5  
y41 x2 y13 x4 x5

y51 y52 x3 x4 x5

---

■从 $CE \rightarrow A$ ，我们得出 $y41=x1$ 。进行替换后如下：

---

x1 x2 y13 x4 y15  
x1 y22 y13 x4 y25  
x1 y32 y13 x4 x5  
x1 x2 y13 x4 x5  
y51 y52 x3 x4 x5

---

■从 $DE \rightarrow C$ ，我们得出 $y13=x3$ 。进行替换后如下：

---

x1 x2 x3 x4 y15  
x1 y22 x3 x4 y25  
x1 y32 x3 x4 x5  
x1 x2 x3 x4 x5  $\Leftarrow$  成功：全都是（带后缀的）x！  
y51 y52 x3 x4 x5

---

■这里的“第四”个元组全都是x，所以JD J确实由给定的FD得出。

让我们来看看另外一个例子。设我们给定的依赖集合只由 $JD\{AB, AC\}$ 组成。这个集合是否蕴含 $FD\ A \rightarrow B$ 呢？注意：我们已经知道答案是否定

的，因为我们在此谈论的是希思定理的逆命题，我们也从习题5.4知道希思定理的逆命题是假命题。但是，让我们看看追逐告诉我们什么：

■前提元组：

---

$x_1 y_{12} y_{13}$

$x_1 y_{22} y_{23}$

---

那么当且仅当FD由此JD蕴含时，应用这个JD到这些元组将必定使 $y_{12}$ 和 $y_{22}$ 相等。结果是这样吗？那么，

■给定的JD“生成”如下所示的元组：

---

$x_1 y_{12} y_{23}$

$x_1 y_{22} y_{13}$

---

■合在一起的四个元组满足JD，但不满足FD，尤其是，它们不要求 $y_{12}=y_{22}$ 。因此，此FD不由给定的JD得出。

[\[1\]](#)同样的道理，不要混淆了相等生成依赖和相等依赖，后者在本章前面的脚注中和其他地方提到了，并将在第13章中进行详细讨论。

[2]严格地说，前提元组根本不是真正的元组，因为它们包含变量，而不是值。同样，前提元组合在一起也并不真正构成一个关系。我建议从这里开始忽视这些要点，但我至少应该顺便提到（部分是由于这样的原因），研究文献通常指出那些前提元组构成的不是一个关系，而是一个场景（tableau）。

## 11.6 结束语

在本章中，我们已经看到了JD蕴含JD，一个JD和一个FD一起蕴含一个FD,FD蕴含一个JD，并且，在前面的章节中，FD蕴含FD。但是，注意，追逐让我们做的所有事情是确定一个特定的依赖是否由给定的依赖得出。它不会做的是，让我们从给定的一个集合推断或产生新的依赖（这就是为什么我在接近上一节开头的地方说，追逐只提供了该问题的部分答案的原因）。对于这一点，我们就需要针对FD和JD的一个公理化。虽然阿姆斯特朗规则为FD本身提供了一个健全和完整的公理化，遗憾的是，把FD和JD结合起来考虑，没有这样的公理化存在是一个众所周知的事实。<sup>[1]</sup>

<sup>[1]</sup>参见，例如，《Foundations of Databases》，by Serge Abiteboul, Richard Hull, and Victor Vianu (Addison-Wesley, 1995) 一书。

## 习题

11.1 考虑供应商和零件数据库的零件关系变量P。为简单起见，我们把属性PNO、PNAME、COLOR、WIGHT和CITY分别重命名为A、B、C、D和E，并且让我们再次使用希思符号。以下的JD都是针对P的标题定义的：

---

- a.  $\odot \{AC, ABDE\}$
  - b.  $\odot \{ACD, ABDE\}$
  - c.  $\odot \{AE, ABCD\}$
  - d.  $\odot \{AB, ACD, CE\}$
  - e.  $\odot \{AB, ACD, AE\}$
  - f.  $\odot \{AB, BCD, DE\}$
  - g.  $\odot \{ABC, ACDE, CE\}$
  - h.  $\odot \{ABCD, BDE, BCE\}$
  - i.  $\odot \{AB, ABC, BCD, CDE, AD\}$
  - j.  $\odot \{AB, BC, CD, DE, AD\}$
  - k.  $\odot \{ABD, CDE, ABC, BE, ABE\}$
  - l.  $\odot \{A, AB, ABC, ABD, ACE\}$
- 

在这些JD中，哪些是平凡的？哪些涉及无关的分量呢？哪些蕴含了在列表中的其他哪些JD？哪些JD对是彼此等价的？哪些被图1-1所示的关系变量P的示例值满足？哪些在关系变量P中存在？哪些相对于P是不可约的？

11.2 在本习题中的依赖都是针对一个由属



性ABCD组成的标题定义的。

a.FD的集合 $\{A \rightarrow B, A \rightarrow C\}$ 是否蕴含  
 $JD \odot \{AD, ABC\}$ ?

b.FD的集合 $\{C \rightarrow D, B \rightarrow C\}$ 是否蕴含  
 $JD \odot \{AB, BC, CD\}$ ?

c.FD的集合 $\{A \rightarrow B, B \rightarrow C\}$ 是否蕴含  
 $JD \odot \{AB, BC, CD\}$ ?

d. $JD \odot \{BC, ABD\}$ 是否蕴含  
 $JD \odot \{AB, BC, CD\}$ ?

11.3 我们从习题5.4知道，希思定理的逆命题是假的。然而，该定理有一个扩展版，其逆命题是真的。它的内容如下所示。

■希思定理（扩展版）：设关系变量R具有标题H，并设X、Y和Z是H的子集，其中X、Y和Z的并集等于H。设XY代表X和Y的并集，对于XZ的处理类似。如果R服从FD  $X \rightarrow Y$ ，那么（a）R服从 $JD \odot \{XY, XZ\}$ ，（b）XZ是R的一个超键。

证明这个定理的（b）部分。并且证明（a）和（b）合在一起蕴含 $X \rightarrow Y$ 存在（扩展版希思定理的逆命题）。

11.4 考虑以下JD，它们都在关系变量S中存在：

---

$$\begin{aligned} & \odot \{ \{ \text{SNO}, \text{SNAME}, \text{CITY} \}, \{ \text{CITY}, \text{STATUS} \}, \\ & \{ \text{SNAME}, \text{CITY} \} \} \\ & \odot \{ \{ \text{SNO}, \text{SNAME}, \text{CITY} \}, \{ \text{CITY}, \text{STATUS}, \text{SNAME} \} \} \end{aligned}$$

---

我在本章正文中指出，（在11.2节中），虽然这些JD中的第一个蕴含第二个，但在第二个JD（即使它是不可约的）的基础上分解关系变量S，将不会是一个好主意。这为什么不是好主意呢？

## 第12章 多值依赖和4NF

Who's on first, What's on second, I Don't Know's on third.

——Bud Abbott and Lou Costello: Naughty Nineties

第10章提到，4NF与2NF和3NF一样，主要具有历史意义。然而，这种描述也许有点不公平，因为以下几个方面。

■首先，4NF是关于所谓的多值依赖或MVD的范式。现在，MVD真的只是一种特殊的JD，因此，如果你了解一般的JD，在一定意义上，你已经知道了MVD。然而，MVD本身仍然是值得研究的（原因之一是，在实践中它们很可能比不是MVD的那些JD更常见）。

■其次，MVD比一般的JD有更直观的现实世界解释，因此往往比较容易理解一点。

■再次，与一般的JD不同，MVD确实有一个公理化，正如我们将会看到的。

因此，让我们来仔细研究。

## 12.1 一个介绍性的例子

在这一节和下一节中，我将从一个相对非正式的角度来探讨MVD，在这两节之后，我会再次考虑它们，但这次会更正式，并使用更多正式的理解来引出4NF。我将从一个定义开始。

■定义：一个多值依赖（Multivalued Dependency, MVD）是一个正好由两个分量组成的连接依赖。

从这个定义可以得出，基于一个MVD的一个无损分解总是正好产生两个投影（回顾前面的内容，一般的JD可以是n路的（其中 $n > 2$ ），相比之下，MVD总是正好2路的）。进一步得出以下JD（例如）其实是一个MVD：

---


---

☼{{SNO, SNAME, CITY}, {CITY, STATUS}}

---

---

现在，我们已经在本书中多次看到了这个特殊的JD，它在关系变量S中存在。但我在第9章没有说一个函数依赖（即， $FD\{CITY\} \rightarrow \{STATUS\}$ ）蕴含这个JD的吗？事实上，我说了。因此，这个例子显示，FD蕴含一些MVD。但不是所有的MVD都是这样，在一定意义上，你

可能会想到那些不这样的MVD是有意义的。因此，让我们来看看那些“有意义的MVD”之一。考虑图12-1，它显示了一个称为CTX的关系变量的示例值。谓词如下：课程CNO可以由教师TNO讲授并使用教科书XNO。

CTX	CNO	TNO	XNO
	C1	T1	X1
	C1	T1	X2
	C1	T2	X1
	C1	T2	X2

图 12-1 关系变量CTX——示例值

现在，关系变量CTX是“全键”，因此，可以肯定它属于BCNF。然而，正如你看到的，它存在冗余，例如，教师T1可以讲授课程C1这个事实出现了两次，而课程C1使用教科书X1这个事实也出现了两次。（因此，它也存在某些更新异常，见习题12.3）。造成这些冗余的原因是，我假设（也许不是很切合实际）教师和教科书彼此是完全独立的，也就是说，不管是谁实际讲授某门特定课程的任何特定的课，都使用同样的教科书。我也假设一位特定的教师或一本特定的教科书可以与任意数量的课程关联。因此：

■每门课程c有一位可以讲授该课程的教师集合T和它使用的教科书集合X。

■而且，每一门这样的课程c，对于一位来自T的教师t和一本来自X的教科书x的每一个可能的组合，在CTX中都有一个元组与之对应。（粗略地讲，每个CNO值与所有与该CNO值对应的TNO和XNO值的笛卡儿积一起出现。）

为了更精确地说明问题，在关系变量CTX中存在下面的约束（回顾第9章，符号“ $\in$ ”是指“存在于”）：

---

---

$$\begin{aligned} & \text{IF } (c, t_1, x_1) \in \text{CTX} \\ & \text{AND } (c, t_2, x_2) \in \text{CTX} \\ & \text{THEN } (c, t_1, x_2) \in \text{CTX} \\ & \text{AND } (c, t_2, x_1) \in \text{CTX} \end{aligned}$$

---

---

但说这个约束存在相当于说下面的JD存在：

---

---

$$\bowtie \{ \{ \text{CNO}, \text{TNO} \}, \{ \text{CNO}, \text{XNO} \} \}$$

---

---

因此，CTX服从此JD，并进一步得出，此关系变量可以并可能应该分解成其在 $\{ \text{CNO}, \text{TNO} \}$ 和 $\{ \text{CNO}, \text{XNO} \}$ 上的投影。习题：展示与图12-1中的

关系变量CTX的示例值对应的这些投影值，并检查冗余消失的情况。（但现在需要强制执行的多关系变量约束是什么呢？）

顺便说一句，我备注，如前所示的约束可以通过简单地把最后一行去掉，从而无损地从四行减少为三行。我的意思是，如果元组（c,t1, x1）和（c,t2, x2）都出现，那么元组（c,t1, x2）必须出现（这就是约束的第三行所表达的意思），因此，调换前两个元组的位置，可以得出，如果（c,t2, x2）和（c,t1, x1）出现，那么（c,t2, x1）也必须出现。但四行版本的约束是对称的，在美学上也是令人满意的，并且也许更容易理解。

顺便说一下，你可能会想CTX的冗余是不必要的。更具体地说，你可能会想，对于一个给定的CNO，此关系变量不需要显示所有可能的TNO/XNO组合。例如，两个元组显然足以代表课程C1有两名教师和两本教科书的信息。现在的问题是，哪两个元组？任何具体的选择都会导致一个关系变量有非常不明显的解释和非常奇怪的更新行为。（请尝试说明这样的关系变量的谓词！即，尝试说明对于那个关系变量决定某个给定的更新在逻辑上是否可以接受的标准。如果你尝试了这个练习，我想你会看到为什么冗余在

CTX中无论如何是必要的。)

[1]这个例子是来自第9章的CTXD的例子的一个修改后的版本。



## 12.2 多值依赖（非正式的）

人们很早就认识到如CTX这样的“有问题”的BCNF关系变量的存在，也在那个时候认识到了处理它们的方法，至少直观地（见习题12.8）。然而，直到1977年，这些直观的想法才由费金提出的MVD概念打下了坚实的理论基础。[\[1\]](#)让我解释一下。

关系变量CTX服从 $JD \bowtie \{\{CNO, TNO\}, \{CNO, XNO\}\}$ 。然而，我们同样可以说，它服从以下这对MVD：

---

---

$$\begin{aligned}\{CNO\} &\twoheadrightarrow \{TNO\} \\ \{CNO\} &\twoheadrightarrow \{XNO\}\end{aligned}$$

---

---

注意：MVD  $X \twoheadrightarrow Y$  可以读作“X多值决定Y”或“Y多值依赖于X”，或者更简单地说，“X双箭头Y”。

综合起来看，上述MVD直觉上的意思是：课程不只有一名教师讲授或者只用到一本教科书（即， $FD \{CNO\} \rightarrow \{TNO\}$  和  $\{CNO\} \rightarrow \{XNO\}$  不存在），但这些课程确实也有一个教师集合和一个教科书集合。更重要的是，对于一门给定的课

程，教师和教科书的集合是完全相互独立的。

（正如我早些时候说过的，无论是谁讲授某门课程的某个特定的课，都使用同样的教科书。同样，对于某门课程，实际上使用什么教科书也没关系，同样的教师都可以讲授它。）因此，我们可以说：

■对于一门给定的课程 $c$ 和一本给定的教科书 $x$ ，与那个 $(c,x)$ 对关联的教师 $t$ 的集合仅依赖于 $c$ ——我们选择哪个特定的 $x$ 没有什么区别。

■同样，对于一门给定的课程 $c$ 和一位教师 $t$ ，与那个 $(c,t)$ 对关联的教科书 $x$ 集合也仅依赖于 $c$ ——与我们选择哪个特定的 $t$ 也没有什么区别。

请注意，图12-1所示的关系变量CTX的示例值确实遵守这些规则。

重复一遍，关系变量CTX服从一对MVD。其实，一般情况下，显示（请参阅下一节）这一点很容易，假设关系变量R具有标题H且H的子集X、Y、Z使得X、Y和Z的并集等于H，那么当且仅当MVD  $X \twoheadrightarrow Y$ 也在R中存在时，MVD  $X \twoheadrightarrow Z$ 在R中存在。MVD总是以这种成对的方式一起存在。由于这个原因，它通常写成“单行”，如下所示：

---

---

$$X \twoheadrightarrow Y|Z$$

---

（“X双箭头Y竖线Z”）。在CTX关系变量的情况下，例如，我们有：

---

$$\{CNO\} \twoheadrightarrow \{TNO\}|\{XNO\}$$

---

现在，我们可能会非常不严格地说，一个MVD与一个FD是一样的，除了，它不是“对于这些中的一个，存在那些中的一个与之对应”，它是“对于这些中的一个，存在那些中的一个集合与之对应”（这个非正式的描述，使得MVD比一般的JD更容易理解一点）。但总是成对一起存在这一点是重要的（注意，没有适用于FD的类似的东西）。事实上，如果MVD概念的定义是太不准确的（事实上，正如我刚刚做的！），可能会错误地得出这样的结论：对于每对相关关系变量的标题的子集X和Y，都有一个从X到Y的MVD。例如，在发货关系变量SP中，对于每个供应商编号，肯定有一个发货数量集合与之对应，但MVD{SNO}  $\twoheadrightarrow$  {QTY}并不存在，情况不是下面说的这样，对于一个给定的供应商编号s和给定的零件编号p，与（s,p）对关联的数量q的集合只依赖于s。

[1] 费金在MVD上的研究工作早于一般的JD概念的广泛采用，这也就是为什么MVD最初被视为一个单独的现象的原因。

## 12.3 多值依赖（正式的）

本节的定义与前面的章节为FD和JD给出的那些定义是相似的，因此只出现少量的进一步解释。

■定义：设H是一个标题，那么一个关于H的多值依赖（MVD）是一个形式为 $X \twoheadrightarrow Y$ 的表达式，其中X（决定因素）和Y（依赖因素）都是H的子集。注意：如果H是不言自明的，那么关于H的MVD这句话可以缩写为仅MVD。

请注意，MVD与FD和JD一样，是关于某个标题，而不是关于某个关系或某个关系变量定义的。还要注意，从形式的角度来看（再次像FD和JD），MVD只是表达式：当这个表达式针对一些具体的关系解释时，它就成为按照定义，判断结果为TRUE或FALSE的命题。

■定义：设关系r具有标题H；设 $X \twoheadrightarrow Y$ 是一个关于H的MVD，比如M，设Z是所有不包含在X或Y任意一个中的H的属性集合（换句话说，Z是X和Y的并集关于H的补集，而X、Y和Z的并集等于H）。如果r满足 $JD \star \{XY, XZ\}$ ，则r满足M，否则r违反M。

请注意，前述的定义在Y和Z上是对称的，由此得出R满足MVD  $X \twoheadrightarrow Y$ ，当且仅当它满足MVD  $X \twoheadrightarrow Z$ （我们因此可以把它们写成“一行”，如在上一节所指出的）。

■定义：当且仅当每一个能赋值给关系变量R的关系都满足MVD M时，在关系变量R中存在MVD M（等价地，关系变量R服从MVD M）。在关系变量R中存在的MVD是R的MVD。

从这个定义和前一个定义中可以得出，当且仅当R服从MVD  $X \twoheadrightarrow Z$ 时，R服从MVD  $X \twoheadrightarrow Y$ 。

■费金（Fagin's）定理：当且仅当MVD  $X \twoheadrightarrow Y|Z$ 在关系变量R中存在时，R可以无损地分解为其在XY和XZ上的投影。

费金定理是第5章承诺的“希思定理更强的形式”。也就是说，希思定理只给出了一个关系变量可无损分解成两个投影的一个充分条件，费金定理既提供了必要条件又提供了充分条件。当然，费金定理是“显而易见的”，因为我们现在知道了一般的JD，事后看来，如果先定义和适当地研究一般的JD，那么根本没有对MVD下任何正式定义的需要。但费金定理是在适当地研究一般的JD之前证明的，并且在那时它是一个新的、重要

的结果，更重要的是，它仍然具有现实意义，因为MVD确实对应一种相当常见的业务规则，而同样的规则却不能合理地用第9章和第10章讨论的在 $n > 2$ 时的“循环” $n$ 路JD表示。

## 12.4 第四范式

听到有平凡的MVD这样一个东西，你不会感到惊讶。

■定义：设 $X \twoheadrightarrow Y$ 是一个关于标题H的MVD，比如M，那么当且仅当每个具有标题H的关系满足它时，M是平凡的。

根据这个定义，可以很容易地证明下面的定理（见习题12.7）。

■定理：设 $X \twoheadrightarrow Y$ 是一个关于标题H的MVD，比如M，那么当且仅当（a）Y是X的子集或（b）X和Y的并集等于H时，M是平凡的。

你可能也不会对接下来的一个定义感到惊讶。

■定义：设关系变量R具有标题H， $X \twoheadrightarrow Y$ 是一个关于标题H的MVD，比如M，那么当且仅当每一个满足R的键约束的关系r也满足M时，M被R的键蕴含。

与FD和JD一样，这里的“被键蕴含”也可以写作“被超键蕴含”，而不会产生任何显著的区别。



另外，如果M是平凡的，那么每一个具有标题H的关系r满足它，所以更不用说每一个满足R的键约束的关系r满足它。因此，平凡的MVD总是平凡地“被键蕴含”。因此，假设M是非平凡的，那么很容易证明下面的定理。

■定理：设M是在关系变量R中存在的一个非平凡的MVD，那么当且仅当它精简为一个从R的超键出来的FD（即，双箭头实际上精简为一个单箭头，并且决定因素是一个超键）时，M被R的键蕴含。

那么，现在我可以定义4NF了。

■定义：当且仅当关系变量R的每个MVD都被R的键蕴含时，R属于第四范式（4NF）。

然而，根据本节已经讨论过的各种定义和定理，我们可以看到以下“可操作的”定义也是有效的。

■定义：当且仅当对于在关系变量R中存在的每一个非平凡的MVD  $X \twoheadrightarrow Y$ , X都是R的一个超键（换句话说，每一个这样的MVD都精简为“从一个超键出来的FD”）时，R属于第四范式（4NF）。

当然，如果一个MVD是被R的键蕴含的，那么它肯定在R中存在（即，它肯定是“R的一个MVD”，但是，反过来是不成立的：一个MVD可以在R中存在，而不被R的键蕴含（关系变量CTX提供了这方面的一个例子）。因此，4NF的定义的重点是，在一个4NF关系变量中存在的仅有的MVD是我们无法摆脱的，这意味着，它们被R的键蕴含（包括作为特例的平凡的MVD）。[\[1\]](#)

现在回顾第10章的BCNF和5NF定义之间的类似点。事实上，该类似点也延伸到4NF定义中。也就是说，我们有以下定义。

■当且仅当在R中存在的每一个FD都被R的键蕴含时，R属于BCNF。

■当且仅当在R中存在的每一个MVD都被R的键蕴含时，R属于4NF。

■当且仅当在R中存在的每一个JD都被R的键蕴含时，R属于5NF。

现在，在BCNF和4NF的定义中，我们可以把“被R的键蕴含”，简化为仅“被R的某个键蕴含”，然而，如第10章中指出的，在5NF的定义中作同样的简化是不正确的，从这个意义上说，

4NF与BCNF的相似度比它和5NF的相似度更高。另一方面，4NF与5NF的相似度也比它和BCNF的相似度更高，在这个意义上的4NF和5NF定义都依赖于上下文——我指的是，在一个4NF或5NF关系变量中存在MVD和JD都至少隐含地包括那个关系变量所有的属性，而同样的说法对BCNF是不正确的。（正如我刚才所说，MVD总是会成对出现这一点是非常重要的。没有适用于FD的类似说法。）

现在回顾第6章的FD保持的概念。从本质上讲，这个概念的内容如下：如果在关系变量R中存在FD  $X \rightarrow Y$ ，那么建议对R进行分解的方式是——假设此分解完全是必要的，并进一步假设，此分解基于除 $X \rightarrow Y$ 本身以外的某个FD——以一种使X和Y在相同的投影中一起保持的方式进行分解。这个概念也延伸到MVD，即，如果我们把整个概念中的FD  $X \rightarrow Y$ 全都替换为MVD  $X \twoheadrightarrow Y$ ，上述建议仍然适用。

在结束本节前，让我明确指出以下几点。

a.如果关系变量R属于5NF，它肯定也属于4NF，类似的，如果关系变量R属于4NF，它肯定也属于BCNF。

b. 一个关系变量可以属于4NF，但不属于5NF（见习题12.4）。

c. 4NF总是可以实现的。（当然，事实上，我们已经知道这一点了，因为我们知道5NF是可以实现的，而现在我们又知道5NF蕴含4NF。）

[1] 与往常一样，“摆脱”任何形式的一个依赖的真正的意思是用某个多关系变量约束来替换它。

## 12.5 公理化

正如我在这一章的开篇提到的，MVD，与一般的JD不同的是，它也有一个公理化，或者换句话说，从给定的MVD产生“新”的MVD的一套健全和完备的规则。所涉及的规则如下所示。

1.如果Y是X的一个子集，则 $X \twoheadrightarrow Y$ （“自反律”（reflexivity））。

2.如果 $X \twoheadrightarrow Y$ 且Z是W的一个子集，那么 $XW \twoheadrightarrow YZ$ （“增广律”（augmentation））。

3.如果 $X \twoheadrightarrow Y$ 且 $Y \twoheadrightarrow Z$ ，则 $X \twoheadrightarrow Z$ （“传递律”（transitivity））。

4.如果（a）X、Y和Z的并集等于相关的标题H，且（b）Y和Z的交集是X的一个子集，那么（c） $X \twoheadrightarrow Y|Z$ （“互补律”（complementation））。

你看，这四条规则几乎不像阿姆斯特朗关于FD的规则那么容易理解或记住（或者无论如何在我看来是如此的）。部分出于这个原因，在这里我不会试图证明它们，我也不会在实践中显示它们。不过，我至少会说，从原来的四条规则可以

得到进一步的一些规则，如下所示。

5.如果 $X \twoheadrightarrow Y$ 且 $YZ \twoheadrightarrow W$ ，那么 $XZ \twoheadrightarrow W$ - $YZ$ （“伪传递律”（pseudotransitivity））。

6.如果 $X \twoheadrightarrow Y$ 且 $X \twoheadrightarrow Z$ ，那么 $X \twoheadrightarrow YZ$ （“合并律”（union））。

7.如果 $X \twoheadrightarrow YZ$ 且 $W$ 是 $Y$ 和 $Z$ 的交集，那么 $X \twoheadrightarrow Y-Z$ ,  $X \twoheadrightarrow Z-Y$ 且 $X \twoheadrightarrow W$ （“分解律”（decomposition））。

以下规则既涉及MVD又涉及FD。

8.如果 $X \rightarrow Y$ ，那么 $X \twoheadrightarrow Y$ （“复制律”（replication））。

9.如果（a） $X \twoheadrightarrow Y$ ，且（b） $Z \rightarrow W$ ，且（c） $W$ 是 $Y$ 的一个子集，且（d） $Y$ 和 $Z$ 的交集为空，那么（e） $X \rightarrow W$ （“聚结律”（coalescence））。

下面是一个额外的派生规则。

10.如果 $X \twoheadrightarrow Y$ 且 $X \rightarrow Z \rightarrow Y$ 且 $Y \rightarrow Z$ ，那么 $X \rightarrow Z \rightarrow Y$ （“混合伪传递律”（mixed pseudotransitivity））。



## 12.6 嵌入式依赖

回顾来自第9章的关系变量CTXD（一个与图9-3重复的示例值，显示在图12-2中）。这个关系变量可以作为本章前面讨论过的关系变量CTX的一个扩展版本。谓词是教师TNO用教科书XNO在课程CNO上花费DAYS天，<sup>[1]</sup>而唯一的键是{CNO,TNO,XNO}。

正如我们在第9章中所看到的，关系变量CTXD存在冗余；<sup>[2]</sup>但它仍属于5NF，这意味着，除了平凡的JD外，没有JD（因此更不用说没有MVD）存在。因此，特别是，MVD不存在，<sup>[3]</sup>但它们确实在CTXD在{TNO,CNO,XNO}上的投影中存在。出于这个原因，称这些MVD是嵌入在原始关系变量CTXD中的。在一般情况下，已知某个具有标题H的关系变量R，一个关于R的嵌入式依赖是指一种在R本身中不存在的投影中，但确实存在R在H的某个真子集上的投影中存在的依赖。如上面例子说明的，因此，（也正如第9章中指出的，尽管用不同的词），嵌入式依赖导致冗余，但这种冗余不能通过投影操作消除。因此，与这种冗余相关的约束，必须单独说明和执行（见习题12.2）。



$$\{ \text{CNO} \} \twoheadrightarrow \{ \text{TNO} \} \mid \{ \text{XNO} \}$$

CTXD	CNO	TNO	XNO	DAYS
	C1	T1	X1	7
	C1	T1	X2	8
	C1	T2	X1	9
	C1	T2	X2	6

图 12-2 5NF关系变量CTXD——示例值

顺便说一句，注意，前述嵌入式概念适用于JD（并因此适用于MVD），[\[4\]](#)但不适用于FD。也就是说，已知某个关系变量R和R的一个投影，其中这个投影的标题包括X和Y，那么当且仅当FD  $X \rightarrow Y$ 在R本身中存在时，它在该投影中存在。例如，如前所述的FD{CITY}  $\rightarrow$  {STATUS}在关系变量S中存在，因此它也在该关系变量的每一个保留这两组属性的投影中存在。

[\[1\]](#)这是我在第9章中给出的谓词，但更准确的一个版本可能是：课程CNO可以由教师TNO讲授并使用教科书XNO，且教师TNO用教科书XNO在课程CNO上花费DAYS天。而且我们可能还要添加DAYS是大于零的。请参阅第15章关于此问题的

进一步讨论。

[2]正如在第9章中指出的，我的一个审校者质疑这种说法。再次，请参阅第15章关于此问题的进一步讨论。

[3]我在这里有一点不严谨，根据本章前面给出的定义， $MVD\{CNO\} \twoheadrightarrow \{TNO\} | \{XNO\}$ 不可能在关系变量CTXD中存在，因为它们不包括DAYS属性。不过，我想你明白我的意思。

[4]举一个不是嵌入式MVD的嵌入式JD的例子，假设来自第9章的关系变量SPJ被扩展为包括一个数量属性，QTY，从而形成一个新的关系变量SPJQ。假设 $FD\{SNO, PNO, JNO\} \rightarrow \{QTY\}$ 在SPJQ中存在，（即 $\{SNO, PNO, JNO\}$ 是一个键）。那么， $\bowtie\{\{SNO, PNO\}, \{PNO, JNO\}, \{JNO, SNO\}\}$ 是一个JD，它存在于SPJQ在 $\{SNO, PNO, JNO\}$ 上的投影中，但不在SPJQ本身中存在。

## 习题

12.1 请给出 (a) 一个属于BCNF但不属于4NF且度至少为三的关系变量的例子，并给出 (b) 一个属于BCNF但不属于4NF的二元关系变量的例子。

12.2 请编写Tutorial D CONSTRAINT语句来表达 (a) 在关系变量CTX中存在的MVD，以及 (b) 在关系变量CTXD中存在的嵌入式MVD，这里的关系变量CTX和CTXD和在本章正文中的一样。

12.3 考虑来自本章正文的关系变量CTX。这个关系变量可能会出现什么样的更新异常呢？

12.4 请给出一个属于4NF但不属于5NF的关系变量的例子。

12.5 请证明，已知某个关系变量R和R的一个投影，其中这个投影的标题包括X和Y，那么当且仅当FD  $X \rightarrow Y$ 在R本身中存在时，它在该投影中存在。

12.6 请显示，如果关系变量R服从FD  $X \rightarrow Y$ ，那么它也服从MVD  $X \twoheadrightarrow Y$ 。

12.7 设 $X \twoheadrightarrow Y$ 是一个关于标题H的MVD，比如M，请证明，当且仅当（a）Y是X的子集或（b）X和Y的并集等于H时，M是平凡的。顺便提及，请注意，从这个结果可以得出，已知MVD  $X \twoheadrightarrow Y|Z$ 对（相对于标题H定义，其中H等于X、Y和Z的并集），那么当且仅当 $X \twoheadrightarrow Z$ 是平凡的时， $X \twoheadrightarrow Y$ 是平凡的。

12.8 在实践中，通常采用以下经验法则。

设关系变量R具有标题H，并设R的标题H被分成不相交的子集X、Y和Z。另外，设X是唯一的键并设Y和Z都是关系值。那么，再次使用希思符号，R应替换成R1和R2，其中 $R1 = (R \{XY\})$  UNGROUP (Y) 且  $R2 = (R \{XZ\})$  UNGROUP (Z)。注意：UNGROUP是Tutorial D的一个操作符，我在附录D中的习题4.14的答案中用过它。在《SQL与关系数据库理论》和其他地方详细地讨论了它。

这个经验法则与本章讨论的主题是怎么关联的？

12.9 （习题9.3的修订版本）请为以下的案例设计数据库。要表示的实体是销售代表、销售区域和产品。每名代表是负责一个或多个区域的

销售，每个区域有一名或多名负责销售的代表。每名代表负责销售一种或多种产品，每种产品都有一名或多名负责销售的代表。每种产品都在每个区域销售，但是，在同一地区没有两个代表销售相同的产品。每个代表在其负责的每个区域销售相同的产品集合。

12.10 以下依赖是关于一个由属性ABCD组成的标题定义的：

---

$B \rightarrow D$

$A \twoheadrightarrow B|C$

---

使用追逐算法来显示这些依赖蕴含了MVD  $A \twoheadrightarrow C|D$ 。注意：我在这里使用了一个特定的速记符号，依照它，这里的 $A \twoheadrightarrow B|C$ 和 $A \twoheadrightarrow C|D$ 分别表示 $A \twoheadrightarrow B|CD$ 和 $A \twoheadrightarrow C|DB$ 。进一步的解释参见附录D中这个习题的答案。

## 第13章 额外的范式

Where's it all going to end?

——Tom Stoppard: *Rosencrantz and  
Guildenstern Are Dead*

Now, this is not the end. It is not even the  
beginning of the end.

But it is, perhaps, the end of the beginning.

——Winston Churchill: *The End of the  
Beginning*

为了解释第9章的内容，在本书中，到目前为止我假设，我们仅关心[11](#)那些利用投影做分解操作，并利用连接做相应的重构操作的依赖。我也说过，根据这样的假设，可以得出，5NF是最终的范式。不过，我也确实在一个脚注中说过，有所谓的“第六”范式或6NF的东西。其实，事实证明，我们不但可以定义如前所述的6NF，还可以定义其他几个范式，这些都没有违反那些关于可用的分解与重构操作的相同假设。图13-1（来自第3章的图3-3的扩展版本）展示了这些额外的范式（即，RFNF、SKNF和6NF，在图13-1中用

粗斜体显示)是如何姑且容纳在总体方案中的。出于完整性,本章将描述这三个范式以及(简要地)更多的几个范式。

1NF  
2NF  
3NF  
BCNF  
4NF  
***RFNF***  
***SKNF***  
5NF  
***6NF***

图 13-1 范式的层次结构 (II)

## 13.1 相等依赖

在描述如前所述的各种额外的范式之前,我需要花一点时间在另一项初级问题。回顾一下第3章中的示例,其中关系变量S被替换为其分别在{SNO,SNAME,CITY}和{CITY,STATUS}上的投影SNC和CT。在对那个例子的讨论中,我曾指出下面的约束:

---

在分解的结果中存在（或至少可能存在），并且我提到这个约束实际上是一个相等依赖。这里有一个定义。

■定义：设 $R_1$ 和 $R_2$ 分别是具有标题 $H_1$ 和 $H_2$ 的关系变量。此外，设 $X_1$ 和 $X_2$ 分别是 $H_1$ 和 $H_2$ 的子集，从而存在一个可能为空的属性重命名集合，使得对投影 $R_1\{X_1\}$ 应用这些重命名之后的结果 $R$ 具有标题 $X_2$ 。那么在 $R_1$ 和 $R_2$ 之间的一个相等依赖（equality dependency）（EQD）大意是 $R$ 和 $R_2\{X_2\}$ 必须相等的一个陈述。（更一般地，一个EQD是要求两个关系相等的任何约束。）

其实，相等依赖是一个称为包含（inclusion）依赖的更普遍现象的一个重要特例。

■定义：设 $R_1$ 和 $R_2$ 分别是具有标题 $H_1$ 和 $H_2$ 的关系变量。此外，设 $X_1$ 和 $X_2$ 分别是 $H_1$ 和 $H_2$ 的子集，从而存在一个可能为空的属性重命名集合，使得对投影 $R_1\{X_1\}$ 应用这些重命名之后的结果 $R$ 具有标题 $X_2$ 。那么从 $R_1$ 到 $R_2$ 的一个包含依赖（IND）大意是 $R$ 必须包含在 $R_2\{X_2\}$ 中的一个陈述（即 $R$ 是 $R_2\{X_2\}$ 的一个子集）。（更一般地，



一个IND是要求一个关系包含在另一个关系中的任何约束。)

从后一个定义产生的要点如下所示。

■外键约束是IND的一个特例。例如，在供应商和零件数据库中，在关系变量SP中，{SNO}是一个外键，它引用关系变量S的键{SNO}；因此，从SP到S有一个IND,SP在{SNO}上的投影含在S在{SNO}上的投影中。但需要注意的是（使用上述定义的符号），不像特定的外键约束，一般的IND不要求X2是R2的一个键（甚至是超键）。

■正如已经指出的那样，EQD也是IND的一个特例。更具体地说，EQD“A=B”等价于一对IND，即IND“A包含在B中”和“B包含在A中”。换句话说，一个EQD实际上是一个双向的IND。

我们随后将要看到大量特定的EQD例子，而不是一般的IND。事实上，这种状况应该是显而易见的：正如我们已经知道的，把一个关系变量无损地分解成投影通常至少会导致IND，经常会导致EQD。然而，在某种程度上，EQD不是无损分解产生的一个有意义的结果。其原因是，这样一个EQD的存在往往变成冗余的一个标志，因为

如果（正如第3章说明的）一些信息记录了两  
次，就可能需要用一個EQD來保持兩種表示方式  
的一致性。

順便說一下，如果你之前還沒有聽到很多關  
於EQD的內容，你可能會奇怪，既然它們的概念  
如此重要，為什麼沒有提及它們呢。在我看來，  
這種遺漏最可能的原因是SQL語言……如果你曾  
經嘗試過練習，你將會知道，EQD在SQL中是非  
常難以編寫的，因為SQL沒有表達關係比較的直接  
方式。<sup>[2]</sup>可以在15.14節的例12中發現支持這一  
論點的一個明顯的例子。


<sup>[1]</sup>那就是說，除了相等依賴和包含依賴。

<sup>[2]</sup>在這裡提到的SQL，我是指SQL標準所定義的  
SQL。在主流的實現中，情況甚至更糟，大多數  
EQD根本不能編寫，這都是由於這樣的事實，即  
所涉及的實現不允許約束中包含子查詢。

## 13.2 第六范式

尽管如此，或至少是隐含的，在本章中，我们不会违背我们关于分解与重构操作的通常假设。我将以与我们的书（《Temporal Data and the Relational Model》（Morgan Kaufmann, 2003），Hugh Darwen, Nikos Lorentzos）完全一样的方式开始关于第六范式的讨论，并且我定义以下内容：

- a. 广义版本的投影和连接操作，
- b. 连接依赖的一种广义形式，
- c. 一个新的范式，称为6NF。

正如那本书的书名可能会表明的，已证明这些发展是对时态数据特别重要的，并且在那本书中详细讨论它们。然而，这种时态数据超出了现在你正在读的这本书的范围，我想在这里做的所有事情是给出适用于“常规”数据（即，非时态数据）的6NF的定义，（且我假设从此刻起的所有数据都是在这个意义上“常规”的）。只诉诸经典的定义中的投影和连接，因此（于是只诉诸经典的定义的JD），下面是6NF的定义。

■定义：当且仅当在关系变量R中存在的所有

JD都是平凡的时，换句话说，在R中的JD的形式都是 $\{ \dots, H, \dots \}$ 时，其中H是标题，R属于第六范式（6NF）。

当然，我们永远不能摆脱平凡的依赖，因此，除了平凡的分解以外，一个属于6NF的关系变量根本不能无损地分解。出于这个原因，有时说一个属于6NF的关系变量是不可约的（注意，另一种不可约性）。我们通常的发货关系变量SP属于6NF，来自第9章的关系变量CTXD也是如此；相比之下，我们通常的零件关系变量P属于5NF，但不属于6NF。（相比之下，当然，我们通常的供应商关系变量S甚至都不属于3NF）。

那么，从这个定义立即可以得出每个属于6NF的关系变量肯定属于5NF，即，6NF蕴含5NF。（这就是为什么它使用第六范式的名称是合理的，因为6NF确实代表沿着从1NF到2NF.....5NF的经典道路又迈进了一步。）更重要的是，6NF总是可以实现的。由于下列原因，它在直观上也是有吸引力的：如果把关系变量R替换为其6NF投影 $R_1, \dots, R_n$ ，那么 $R_1, \dots, R_n$ 的谓词全都是简单的，且R的整体谓词是这些简单谓词的合取（即，这是一个合取谓词）。让我立刻解释一下我这些言论的意思。

■定义：如果一个谓词不涉及连接词和复合（或组合），那么它是简单的；否则它不是简单的。

■定义：连接词是一个逻辑操作符，如AND（与）、OR（或）或NOT（非）。

■定义：合取谓词是两个或两个以上的其他谓词的与（AND）。注意：本定义是很不严格的，但就当前目的而言，它已经足够好。

例如，假设我们把关系变量P替换为其分别在属性{PNO,PNAME}、{PNO,COLOR}、{PNO,WEIGHT}和{PNO,CITY}上的投影PN、PL、PW和PC。这些投影的谓词如下（注意，它们都是简单谓词）。

■PN：零件PNO的名称是PNAME。

■PL：零件PNO的颜色是COLOR。

■PW：零件PNO的重量是WEIGHT。

■PC：零件PNO存储在城市CITY。

P本身的谓词是这四个谓词的与（AND），[\[2\]](#)因此，如此例所示，可以认为属于

6NF的关系变量把数据的含义分解成不能再进一步分解的片（它们表示的含义，有时也称为“原子事实”，也许最好称为“不可约的事实”）。严格地说，我们可以说属于6NF的关系变量的谓词不涉及任何与运算。

旁白：在这方面，我想简要地提醒你，分别来自第12章和第9章的关系变量CTX和SPJ,CTX的谓词肯定是合取的（课程CNO可以由教师TNO讲授且（AND）课程CNO使用教科书XNO），而把关系变量分解成它在{CNO,TNO}和{CNO,XNO}上的二元（其实也是6NF）投影最终会消除这个AND。SPJ的谓词也是合取的，尽管在我表达它的简化形式中没有出现同样的连接词。这里有一个更完整的版本：供应商SNO提供零件PNO给某个工程JNO且（AND）零件PNO被某个供应商SNO提供给工程JNO且（AND）工程JNO由某个供应商SNO提供零件PNO。同样，把关系变量分解成三个二元（其实也是6NF）投影会消除这些AND。结束旁白。

现在这里有一个关于6NF的清晰描述（其实，这是一个定理）。

■定理：当且仅当（a）关系变量R属于5NF，（b）它的度为n，且（c）不存在度小于n-

1的键时，它属于6NF。

例如，设关系变量PLUS具有属性A、B、C（因此它的度是三），并设关系变量谓词是 $A+B=C$ ，那么PLUS属于5NF，它有三个键（即AB、BC和CA，再次使用希思符号），但是，这些键的度没有小于二的，因此PLUS属于6NF。[3]

顺便说一下，请不要误会我的意思，我不是说关系变量应始终属于6NF，或规范化应该总是要求尽可能实现6NF。有时，一些更低的范式（比如5NF）至少是足够的。更重要的是，重复第8章中的内容，一个完全规范化的设计（这表示关系变量都属于5NF，甚至6NF）可能仍然是糟糕的。例如，供应商关系变量S在{SNO,STATUS}上的投影肯定属于6NF，但它不是一个良好的设计，正如我们在第6章中所看到的。

还有一点要考虑的是，把一个5NF关系变量替换为6NF投影将可能导致需要执行某些相等依赖（EQD）。正如我们在上一节看到的，一个EQD大意是某些关系变量的某些投影必须相等的一个约束（很不严格地说）。例如，如果我们把上面所讨论的关系变量P分解为其在PL、PW、PN和PC上的投影，那么可能要应用下面的约束：

---

```
CONSTRAINT.....PL{PNO}=PN{PNO};  
CONSTRAINT.....PW{PNO}=PN{PNO};  
CONSTRAINT.....PC{PNO}=PN{PNO};
```

---

另一方面，在其他地方解释过，[\[4\]](#)以类似正在讨论的分解为基础可以很好地处理缺失的信息。假设每一个零件总是有一个已知的名字，但并不一定有一个已知的颜色、重量或城市。那么一个没有已知颜色的零件在关系变量PL中根本没有元组（对于重量与城市和关系变量PW与PC，分别是同样的）。当然，相等依赖将成为分别从PL到PN、从PW到PN和从PC到PN的包含依赖（实际上是外键约束）。

上述讨论的实质如下（我将用举例的方式把它表达出来，只为定性）：如果每一个零件总是有两个或多个属性，比如名称和颜色，那么把这两个属性分离为不同的投影可能是一个坏主意，但如果一些属性是“可选的”（换句话说，有可能是“缺失”或未知的），那么在该属性自己的一个关系变量中放置它可能是一个好主意。

[\[1\]](#)所以我毕竟没有真正违背我们通常的假设。

[\[2\]](#)换句话说，每个零件都确切地有一个名称、颜色、重量和城市。事实上，正是因为这样的情



况，如果我们不想分解，我们实际上并不需要把关系变量P分解成其投影PW、PN、PL和PC，单个关系变量P可以有效地作为这四个关系变量的组合的简写。

[3]事实上，PLUS可能是一个关系常量（constant），而不是一个关系变量，但它仍然有键。

[4]参见《SQL与关系数据库理论》和《SQL and Relational Theory or the book Database Explorations: Essays on The Third Manifesto and Related Topics》，作者为Hugh Darwen和我自己（Trafford, 2010）二者中的任何一本书。

### 13.3 超键范式

我想讨论的下一个范式，简单地说，是超键范式（SuperKey Normal Form, SKNF）。让我马上补充说，SKNF本身似乎并不是非常重要的；我提到它的主要原因完全是，许多教科书把本质上是它的定义作为第五范式的一个不正确的定义给出了。例如，下面是从我自己的教科书提取的一个转述<sup>[1]</sup>（警告！它是错误的！）：

关系变量R属于5NF，当且仅当在R中存在的每一个平凡的JD都被R的键蕴含时，其中：

a. 当且仅当 $X_1, \dots, X_n$ 中的至少一个等于R的标题时， $JD \bowtie \{X_1, \dots, X_n\}$ 是平凡的。

b. 当且仅当 $X_1, \dots, X_n$ 中的每个都是R的一个超键时， $JD \bowtie \{X_1, \dots, X_n\}$ 被R的键蕴含。

当然，这个定义的a部分是正确的，但b部分不正确。为什么不正确呢？考虑下面的反例。设关系变量SNC是供应商关系变量S在属性 $\{SNO, SNAME, CITY\}$ 上的投影。SNC属于5NF。然而，如下JD：

---

在关系变量SNC中存在，且{SNAME,CITY}分量不是该关系变量的超键。

现在观察，上述“定义”明确是指非平凡的JD。因此，你可能会想，为了纠正它，我们需要做的，是把“非平凡的”替换为“不可约的”（请注意，上面展示的，在SNC中存在的一个JD，是可约的，{SNAME,CITY}分量可以去掉，并且不造成损失）。然而，事实并非如此。下面是一个更复杂的反例。

■设关系变量R具有属性A、B、C（只有这些属性），设AB、BC、CA中的每个都是R的键，并设 $JD \bowtie \{AB, BC, CA\}$ （把它称作J）在R中存在。

■那么（a）除了平凡的依赖外，没有额外的依赖是由J和那些键蕴含的；（b）J对于R是不可约的。（这两点可能不是很明显，但它们其实是正确的。）

由此可以得出，R不属于5NF（成员算法在J上失败），而J的每个分量都是一个超键。

注意：如果你喜欢一个更具体的例子，设A、B、C分别是“最喜爱的颜色”，“最喜爱的食物”和“最喜爱的作曲家”，并设谓词是存在一个人，其最喜爱的颜色是A，最喜爱的食物是B，最喜爱的作曲家是C。另外，设业务规则的大意是：

■任何两个不同的人都没有多个共同喜爱的东西。

■不存在这样的三个不同的人，对于每种喜爱的东西，这三人中的两人都有共同的选择。

习题：为这个关系变量编造一些示例数据。如果你尝试做这个习题，我想你就会明白为什么指定的键约束和JD是有意义的。

通过上述启发的方式，那么，让我们来定义另一个范式。

■定义：当且仅当对于在关系变量R中存在的每个不可约JD  $\{X_1, \dots, X_n\}$ ，都有 $X_1, \dots, X_n$ 中的每个都是R的一个超键时，R属于超键范式（SKNF）。

现在，我已经说过了，SKNF真的不是很有

意义。这是真的，但至少有一个涉及它的定理。

■定理：5NF蕴含SKNF，但反过来是不正确的，SKNF蕴含4NF，但反过来是不正确的。

换句话说，SKNF严格地位于4NF和5NF之间（即，它是强于4NF且弱于5NF）的。话虽如此，不过，我要补充一点，在相关的关系变量R只有一个键的常见的特例下，（事实上从定义看是明显的）SKNF和5NF是一致的，这也许正是思考SKNF本身没多大意义的另一个原因。然而，事实是，SKNF和5NF之间有一个逻辑上的区别，这就是为什么我在本章包含它的原因。

[1] 《数据库系统导论（原书第8版）》  
（Addison-Wesley出版社，2004年）。

## 13.4 无冗余范式

第9章我讨论了一个惊人的事实：即，存在这样的关系变量，它不能无损地分解为两个投影，但可以无损地分解为两个以上的投影。现在让我为你提供另外一个惊喜：即，5NF，虽然它对于消除整本书至今我们一直在谈论的那种冗余（即那种可以通过投影操作消除的冗余）是充分的，但实际上不是必要的。现在，我希望你会与我在2010年第一次遇到这种状况时同样惊讶！5NF是费金于1979年定义的，并在此后30年左右的时间里，人们普遍认为，要使关系变量不存在冗余，它必须属于5NF（意思是，为了消除我们一直在谈论的那种冗余，5NF肯定是必要的，但一般不充分）。但是，事实证明，5NF对于实现这一目标是完全没有必要的，更特别的是，事实证明，对于消除冗余，一个严格弱于5NF但强于第四范式（4NF）的新范式与5NF是完全等效的。其实，原因很明显，新的范式称为无冗余范式（Redundancy Free Normal Form, RFNF），对于这个目的被证明是必要的且充分的。因此，如果规范化的目标是减少冗余，那么要瞄准的目标不是5NF，而是RFNF。

旁白：这里使用的名称RFNF来自自由体·达尔文、罗恩·费金和我最近写的一篇论文的初稿（见

附录G)。但是，在本书将要出版时，我们发现有一篇更早的论文（“Redundancy Elimination and a New Normal Form for Relational Database Design”，Millist W.Vincent，1998）已经使用了该名称用来指别的东西（更具体地说，与“我们的”范式不同的东西）。因此，由于显而易见的原因，我们打算为我们的RFNF选择一个新的名称；[\[1\]](#)但是，为了当前的目的，我会继续使用名称RFNF来指我们的范式，除非有明确的相反陈述。如果这会导致任何混乱，我对此道歉。在附录B中，对于Vincent的RFNF，以及我们的RFNF，我还有更多内容要说。结束旁白。

为了说明这些想法，我将从一个例子开始：来自第9章关系变量SPJ的一个修改后的形式，我把它称为SPJ'。像以前一样，关系变量具有属性SNO（供应商编号）、PNO（零件编号）、JNO（工程编号），谓词是与以前一样的：供应商SNO提供零件PNO给工程JNO。此外，下面的业务规则是有效的（再次与以前的一样）。

1.如果供应商s供应零件p且把零件p提供给工程j且工程j由供应商s供应，那么供应商s为工程j供应零件p。

但是，现在假设以下业务规则也是有效的。

2.任何给定的供应商s最多为一个工程j供应给定的零件p。

那么，正如我们之前所看到的，下面的JD捕获了这些规则中的第一个的本质，因此，在SPJ'中存在：

---

---

$$\odot \{ \{SNO, PNO\}, \{PNO, JNO\}, \{JNO, SNO\} \}$$

---

---

同样，以下的FD捕获了第二个规则的本质，因此它也在SPJ'中存在：

---

---

$$\{SNO, PNO\} \rightarrow \{JNO\}$$

---

---

从这个FD可以得出 $\{SNO, PNO\}$ 是SPJ'的一个键。此外，它可以表明，除了平凡的FD或JD，没有其他的FD或JD存在。因此，由于前述JD肯定不是由唯一的键蕴含的（成员算法失败），SPJ'不属于5NF，尽管它属于BCNF（正如SPJ不属于5NF但属于BCNF，本质上是相同的原因）。[\[2\]](#)注意：只是顺便说一句，这个例子因此揭穿了两种流行的误解（见习题10.1）：第一个误解是，只有一个键和一个非键属性的关系变量必定属于5NF，第二个误解是，一个不属于5NF



的BCNF关系变量必定是全键。

现在，假设此关系变量包含以下三个元组：

---

t1=s1 p1 j2  
t2=s1 p2 j1  
t3=s2 p1 j1

---

（符号s1和s2表示供应商的编号；p1和p2表示零件编号；j1和j2表示工程编号；而t1、t2和t3是这三个元组的引用标签。）那么，由于JD存在，必须也出现下面的元组：

---

t4=s1 p1 j1

---

但是，因为{SNO,PNO}是一个键，这样元组t1和t4就具有相同的键值，所以它们实际上是同一个（并且因此，j1=j2）。因此，我们在第9章的SPJ中观察到的那种冗余不会发生在SPJ'上。

（更具体地，在这种情况下，元组t4不是一个“额外”的元组。参见在下一个段落中关于元组强迫的JD的讨论）。换句话说，SPJ'，即使它不属于5NF，没有且实际上不可能遭受5NF旨在解决的那种冗余。因此，在一定意义上，看起来好像5NF对于这个目的可能是过于强了。

现在，让我提醒你元组强迫（tuple forcing）JD的概念。基本上，如果一个JD是这样的，如果某些元组的出现，迫使一些额外的元组也出现，那么它是元组强迫的。前面的章节多次提到这个概念，但我从来没有真正适当地定义它，所以现在在我提供这样的一个定义。

■定义：设J是一个关于标题H的JD，并设J在关系变量R中存在。那么J可能或可能不会得出下面的结果，即，如果某些特定的元组 $t_1, \dots, t_n$ 出现在R中，那么一个特定的额外元组 $t$ 也被迫出现在R（其中，“额外的”表示 $t$ 是有别于 $t_1, \dots, t_n$ 中的任何一个的）。如果它确实得出这样的结果，那么J是关于R元组强迫的。注意：很容易看出，任何这样的J必须是：（a）平凡的，（b）不被R的任何FD蕴含，且（c）不被R的键蕴含。[\[3\]](#)

在第9章中的关系变量SPJ是一个元组强迫的JD（事实上，这正是那个关系变量的设计的错误之处）。但同样的批评并不适用于SPJ'的例子：在SPJ'中存在的JD不会强迫任何额外的元组出现，这是由于也在那个关系变量中存在的这个FD。这就是尽管SPJ'与SPJ一样不属于5NF，但它不遭受与SPJ遭受的同样一种冗余的原因。

那么，受上述例子的启发，我可以定义另一个新的范式。不过，我需要一步一个脚印地定义它。首先，我们需要对我们正在谈论的冗余下一个准确的定义。

■定义：当且仅当关系变量R不属于BCNF时，它是FD冗余的。

■定义：当且仅当在关系变量R中存在一些元组强迫的JD时，它是JD冗余的。

注意，这两种冗余是互不蕴含的；即，一个关系变量可以是FD冗余的但不是JD冗余的，或是JD冗余的但不是FD冗余的。[\[4\]](#)例如：

■来自第9章的关系变量SPJ（其属性是SNO、PNO、JNO；它的键是这三个属性的组合；且存在 $JD \bowtie \{\{SNO, PNO\}, \{PNO, JNO\}, \{JNO, SNO\}\}$ ）属于BCNF并因此不是FD冗余的，但它很明显是JD冗余的。

■供应商关系变量S（其属性是SNO、SNAME、STATUS和CITY；它的键是SNO；且存在 $FD\{CITY\} \rightarrow \{STATUS\}$ ）不属于BCNF，并因此是FD冗余的。但在该关系变量中不存在元组强迫的JD，因此它不是JD冗余的。

现在继续这个定义。

■定义：当且仅当关系变量R既没有FD冗余又没有JD冗余时，它是无冗余的。[\[5\]](#)

请注意，根据上述定义，5NF关系变量肯定无冗余。一个SKNF关系变量也是如此，到目前为止；但为了使一个关系变量无冗余，它不必属于5NF，甚至都不必属于SKNF（见下文）。

■定义：当且仅当关系变量R无冗余时，它属于无冗余范式（RFNF）。

换句话说，当且仅当关系变量R既不是FD冗余的，也不是JD冗余的：等价地，当且仅当它属于BCNF且不存在元组强迫JD时，它属于RFNF。

当然，虽然上述的定义是精准的，但它没有什么实际用处，因为它对确定一个给定的关系变量是否确实属于RFNF的问题没有多大帮助。但是，有一个确实在这方面有帮助的定理。

■定理：当且仅当关系变量R属于BCNF，且对于在R中存在的每一个明确的JD  $J, J$ 中的某个分量都是R的一个超键时，R属于RFNF。

这个定理为一个关系变量属于RFNF提供了必要条件和充分条件。因此，我们可以把这个定理当作一个对RFNF有用的、可用的测试：实际上，可以作为RFNF的一个有效的定义。注意：该定理是指R的显式JD，但实际上，我们可以删除这个限定词，而剩下的东西仍然是正确的（即，当且仅当在R中存在的每个JD都有一个超键的分量时，R属于RFNF）。然而，在一定意义上，包含限定词使这个定理“更严格”。特别是，它表示为了测试所涉及的关系变量是否属于RFNF，没有必要检查一个关系变量的隐式JD。（事实上，我们仍然可以通过把“每一个显式JD”替换为“每一个显式不可约的JD”使定理更严格。然而，在实践中，某个显式JD是不太可能不是不可约的，所以这一点也许不是非常重要。）

下面的定理表明，RFNF确实严格地位于4NF和5NF之间，事实上，它严格地位于4NF和SKNF之间。[\[6\]](#)

■定理：5NF蕴含SKNF；SKNF蕴含RFNF；且RFNF蕴含4NF。相反的蕴含不成立。

然后回顾一下，RFNF严格地弱于5NF，尽管它在消除冗余方面丝毫不亚于5NF。

下面是提供了简单有用和实际测试的另外两个定理。

■定理：设R是一个3NF关系变量，并设R没有复合键，那么R属于RFNF。（回顾一下，一个复合键是由两个或两个以上的属性组成的。）

■定理：设R是一个BCNF关系变量，并设R有一个非复合键，那么R属于RFNF。

这些定理都为一个关系变量属于RFNF提供了一个充分条件，尽管不是必要条件。注意，所涉及的条件有一个引人注目的特性，即，它们仅涉及FD，而不涉及JD。注意：事实上，这些定理中的第一个应是毫不奇怪的，因为我们已经从第10章10.4节）知道，一个没有组合键的3NF关系变量属于5NF。因此，更不用说，这样的关系变量也属于RFNF。至于第二个定理，我们应该很清楚，如果R属于BCNF，并有一个非复合键K，那么K必须包含在R中存在的每个JD的至少一个分量中，得出上述结果的原因马上会揭晓。

这场可以称为RFNF的正式部分的讨论到此就接近尾声了。不过，我想仔细看看启发的例子（关系变量SPJ），因为关于这个例子还有更多有用的东西可说。回顾一下， $FD\{SNO, PNO\}$

→{JNO}和JD⊗{{SNO,PNO}, {PNO,JNO}, {JNO,SNO}}都在该关系变量中存在。但是，从一个直观的角度来看，这些事实蕴含了什么呢？好吧，假设此关系变量包含下面这三个元组：

---

t1=s1 p1 j2  
t2=s1 p2 j1  
t3=s2 p1 j1

---

还假设， $s1 \neq s2$ ， $p1 \neq p2$ ，且 $j1 \neq j2$ 。那么，由于这个JD的存在，下列元组必须也出现：

---

t4=s1 p1 j1

---

但{SNO,PNO}是一个键，所以元组t1和t4必定是同一个，同样，j1也必定等于j2，这些都有悖于我们的原始假设。因此，如果关系变量只包含元组t1和t2，插入元组t3的尝试一定会失败，正是因为它会导致这一矛盾。因此，我们必须实现以下（有点怪异）的业务规则。

■如果（a）供应商s1为工程j2提供零件p1且（b）供应商s1还为工程j1提供零件p2（ $p1 \neq p2$ ， $j1 \neq j2$ ），那么，（c）任何供应商，即使是s1，都不可以为工程j1提供零件p1。[7]

更重要的是，应该明确，下列同样离奇的规则也必须生效（注意对称性）：

■如果（a）供应商s1为工程j2提供零件p1且（b）供应商s2为工程j1提供零件p1（ $s1 \neq s2$ ,  $j1 \neq j2$ ），那么，（c）任何零件，即使是p1，都不可以由供应商s1提供给工程j1。

■如果（a）供应商s1为工程j1提供零件p2且（b）供应商s2为工程j1提供零件p1（ $s1 \neq s2$ ,  $p1 \neq p2$ ），那么，（c）任何工程，即使是j1，都不可以由供应商s1提供零件p1。

事实上，这三条业务规则可以合并成如下所示的一条业务规则。让我们同意，只在此刻，假设关系变量SPJ'的每个元组代表一次发货（由某个零件的某个供应商为某个工程）。那么就不能存在三次不同的发货x、y和z，使得x和y涉及相同的供应商，且y和z涉及相同的零件，且z和x涉及相同的工程。

关于SPJ的例子还有另外一点。请再次分析导致上述三条业务规则中的第一条的原因。该分析表明，元组t3不能与元组t1和t2一起出现。因此，SPJ'出现了一个插入异常，尽管事实上，它属于RFNF（及因此事实上，没有元组强迫的JD



存在)。与此相反，它并没有出现删除异常（假设，那就是，它存在的唯一约束是说明的FD和JD及它们的逻辑后果）。因此，5NF和RFNF之间的一个区别是这样的：尽管它们都是无冗余的，5NF还保证“没有插入异常”，而RFNF不保证这一点。（再次假设FD和JD是仅有的要考虑的约束条件。）

当然，这是从SPJ的例子得到的很诱人的结论，即，属于RFNF，而不属于5NF的关系变量，有可能在实践中是罕见的。然而，在两个范式之间有一个清晰的逻辑区别，并因此，至少从减少冗余的观点来看，要瞄准的目标真的应该是RFNF，而不是5NF。（作为奖励，我注意到，对RFNF的测试也比5NF更加容易一点）。

注意：由于事实上，SPJ的例子还以另一种方式增强了上述观点。由于此关系变量服从 $JD \bowtie \{\{SNO, PNO\}, \{PNO, JNO\}, \{JNO, SNO\}\}$ ，它可以无损地分解成其分别在 $\{SNO, PNO\}$ ， $\{PNO, JNO\}$ 和 $\{JNO, SNO\}$ 上的投影。这些投影每个都是“全键”，且事实上属于5NF。然而，这种分解“丢失”了 $FD\{SNO, PNO\} \rightarrow \{JNO\}$ ！正如我们在第6章所看到的，丢失依赖通常是不推荐的。因此关系变量SPJ说明了这一点，5NF不仅是有时太强了，而且有时可能是肯定禁止的。

[1]“我们的”RFNF因此已经更名为ETNF（Essential Tuple Normal Form，必需元组范式）。如需进一步讨论，请参阅附录C中的“最新消息”。

[2]事实上，它不只属于BCNF，它实际上属于4NF。同样，它不只是不属于5NF，它甚至都不属于SKNF（请原谅此处故意笨拙的措辞）。

[3]你可能会想这些条件中的第三个是另外两个条件的一个合乎逻辑的结果，但情况并非如此。例如，考虑一个具有属性A、B、C、D和键A及键B的关系变量R；设在R中不存在除了那些由R的键蕴含的FD外其他的FD。那么，可以很容易地看到，在R中存在非平凡的 $JD \bowtie \{AB, AD, BC\}$ ，因为这个JD是被这些键一起蕴含的。然而，它不被任何一个单独的键蕴含，换句话说，它不被在R中存在的如前所述的任何单独FD蕴含。

[4]这种状况强调了我第9章说的东西：即，最好是把FD和JD当作不同的现象，而不是把FD当作JD的一个特例。

[5]这里的术语无冗余（redundancy free）也许不是很好的选择，因为它给另外一个很一般的术语赋予了一个非常特殊的意义。

[6]关系变量SPJ是属于RFNF但（正如在较早的脚注中注明的）不属于SKNF的关系变量的一个具体的例子。

[7]出于同样的原因，任何供应商，甚至是s1，也都不可以为工程j2提供零件p2。注意：当然，类似的言论也适用于即将要讨论的“同样离奇”的规则。

## 13.5 域-键范式

域-键范式（DK/NF）与到目前为止在本书中讨论的所有范式都不同，它根本不是依据如前所述的FD、MVD和JD定义的。<sup>[1]</sup>DK/NF真的是一种“理想的”范式：它是可取的，因为根据定义，一个属于DK/NF的关系变量是保证不存在特定的更新异常的；然而，可悲的是，它不是总是可以实现的，而且“究竟何时能实现”这个问题也未获答复。即便如此，让我们展开研究。

DK/NF是用域约束（domain constraint）和键约束（key constraint）的方式定义的。当然，我们已经非常熟悉键约束（定义见第5章）。至于域约束，我想提醒你，域这个词本质上只是类型的另一种说法（见附录D中习题2.4的答案）。因此一个域约束在逻辑上应该与一个键约束是同样的事情，换句话说，它应该仅仅是一个值集合的规格说明，这些值构成所涉及的类型（有关这个概念的进一步说明，请参见《SQL与关系数据库理论》）。然而，用在目前上下文中的这个词有一个稍微特殊的意义。具体地说，用在这里的术语域约束，表示的是一个约束，大意是一个给定属性的值必须取自某个规定的值集合：例如，供应商关系变量S上的一个约束，大意是状态值

（都是整数，即类型为INTEGER）必须在1~100范围内，包含上下限。

那么，这就是一个定义。

■定义：当且仅当在关系变量R中存在的所有关系变量约束都被在R中存在的域约束和键约束蕴含时，R属于域-键范式（DK/NF）。[2]

因此在一个DK/NF关系变量上强制执行约束在概念上很简单，因为只要执行相关的域约束和键约束就足够了，其他所涉及的关系变量上的所有约束（不只是FD、MVD和JD，而是应用到所涉及的关系变量的所有关系变量约束），都将自动强制执行。

DK/NF是费金在1981年首次定义的，并且正是DK/NF论文首次给出了插入异常和删除异常的明确定义。我在第10章中定义过这些概念，但在那里的定义是专门依据JD制定的。这里记录的是一般定义（注意，它们是指一般的约束，而不只碰巧是FD、MVD或JD这些约束中的一种）：[3]

■定义：当且仅当存在关系变量R的一个合法值r和与R具有相同标题的一个元组t，使得通过追加t到r所得的关系满足R的键约束，但不是R的一

个合法值（即它违反R上的某个关系变量约束）时，R存在插入异常。

■定义：当且仅当存在关系变量R的一个合法值r和r的一个元组t，使得通过从r删除t得到的关系不是R的一个合法值（即它违反R上的某个关系变量约束）时，R存在删除异常。

最后，我们有下面的定理：

■定理：只要每一个相关的属性都可以取至少两个不同的值，DK/NF就蕴含5NF。

这就是说（很不严格地），每个DK/NF关系变量都属于5NF，因此也属于RFNF（等等），当然，虽然它不一定属于6NF。事实上，在（大概是不可能的）存在的仅有的约束都特别是FD和JD的特殊情况下，DK/NF和5NF是一致的。

[1]它是依据键约束定义的，正如我们将看到的，而键约束反过来是FD的一个特例，所以这个评论也许有点偏颇。

[2]关系变量约束是可以通过孤立地检查相关关系变量进行测试的任何约束。如需进一步讨论，请参阅《SQL与关系数据库理论》。

[3]这些定义，与第10章中的定义一样，稍微有点

可疑，因为它们谈论插入或删除个别元组。

## 13.6 结束语

这是一次多么漫长和奇怪的旅行.....前面的章节描述了1NF、2NF、3NF、BCNF、4NF、5NF（最后3个有点长），而现在我们又遇到了四个范式：RFNF、SKNF、6NF和DK/NF（最后一个是有有点奇怪的一个）。但即使这样，故事并没有结束。在这总结性的一节，只是为了完整性，我将简要地提到不时在其他文献中已定义的其他一些范式。

### 13.6.1 基本键范式（EKNF）

基本键范式是由Zaniolo在1982年提出的。[\[1\]](#)下面是它的定义。

■定义：当且仅当对在关系变量R中存在的每一个非平凡的FD  $X \rightarrow Y$ ，要么（a）X是一个超键，要么（b）Y是某个基本键的子键时，R属于基本键范式（EKNF）。当且仅当存在R的某个属性A使得FD  $K \rightarrow \{A\}$ 是非平凡的且不可约的时，键K是基本的。

EKNF严格地位于3NF和BCNF之间，也就是说，BCNF蕴含EKNF,EKNF蕴含3NF，反向的蕴含均不存在。EKNF声明的意图是“抓住3NF和




BCNF的显著性质”，同时避免两者的问题（即，3NF“太宽容”，而BCNF“容易导致计算的复杂性”）。此外，我也应该说，EKNF在文献中没有太多的引用。

[\[1\]](#) Carlo Zaniolo: “A New Normal Form for the Design of Relational Database Schemata, ”ACM TODS 7, No.3 (September 1982) .

## 13.6.2 过强的PJ/NF

回顾一下，5NF最初称为PJ/NF，而PJ/NF表示每个JD都被键蕴含（很不严格地说）。事实上，在他提出了PJ/NF的论文中，费金还提出了他称之为过强的PJ/NF的范式，这表示（再次很不严格地说），每个JD都被某个孤立地考虑的特定的键蕴含。后者就是人们直觉上所期待的常规PJ/NF（即5NF）的定义，请回顾在第10章和第12章中有关BCNF、4NF、5NF之间的定义的相似性的评论。即使如此，下面是这个定义。

■定义：当且仅当关系变量R的每个JD都被R的某个键蕴含时，R属于过强的PJ/NF。

过强的PJ/NF显然蕴含5NF（即“常规”的PJ/NF），但反过来是不正确的。一个单独的反例足以证明后一个事实：考虑一个关系变量R，它具有属性A、B、C、D（只有这些属性），并只有键{A}及{B}。设在R中仅有的依赖都是被这些键蕴含的（所以R肯定属于5NF）。现在考虑 $JD \bowtie \{AB, BC, AD\}$ 。应用成员算法，我们看到，这个JD在R中存在，但它不是任何一个孤立地考虑的键的一个后果，这也可以通过检查成员算法看出。因此，R属于5NF（或PJ/NF）但不属于过强的PJ/NF。

[1] 同样的例子，在本章前面关于元组强迫的JD的定义的一个脚注中曾用到过。

### 13.6.3 “限制-合并”范式

考虑来自供应商和零件数据库的零件关系变量P。此刻之前所描述的规范化理论告诉我们关系变量P属于一个“好”的范式，事实上，它属于5NF，因此，它保证不存在可以通过投影删除的异常。但是，为什么要把所有零件都保存在一个单一关系变量中呢？如果有一个设计，它把红色的零件都保存在一个关系变量（比如RP），蓝色的保存在另一个关系变量（比如BP），等等，这种设计怎么样呢？换句话说，通过限制而不是投影的方法把原来的零件关系变量分解的可能性怎么样呢？这样产生的结构是一个良好的设计还是一个糟糕的设计？（事实上，正如我们会在本书第四部分中看到的，除非我们非常谨慎，否则它几乎肯定会是糟糕的，但是，这里要说的是，对于这种情况，如前所述的经典的规范化理论绝对是没什么可说的。）

因此设计研究的另一个方向是研究通过投影以外的某个操作对关系变量进行分解的影响。在这个例子中，前面已经提到，分解操作是（不相交）的限制，相应的重构操作是（不相交）的合并。因此，有可能创建一种“限制-合并”规范化理论，类似于（但正交于）我们在此之前一直在考虑的投影连接规范化理论。我不希望在这里更具

体地讨论这个问题，可以肯定地说，沿着这些方向的一些初步的想法可以在下列地方发现：

（a）在费金的PJ/NF论文，其中还讨论了一个称为PJSU/NF的范式，和（b）史密斯的论文，其中讨论了一个称为（3，3）NF的范式。[\[1\]](#)顺便说一句，后一篇论文表明：（3，3）NF蕴含BCNF，但一个（3，3）NF关系变量不一定属于4NF，一个4NF关系变量也不一定属于（3，3）NF。如前所述，因此，简化到（3，3）NF与简化到4NF（及5NF）是正交的。

[\[1\]](#)J.M.Smith, “A Normal Form for Abstract Syntax, ”Proc.4th Int.Conf.on Very Large Data Bases,Berlin,Federal German Republic (September 1978) .

## 习题

13.1 绘制记忆中的范式层次结构，至少包括9种范式。

13.2 请给出6NF的定义。

13.3 有时说6NF关系变量是不可约的，并且我注意到在本章正文中提到，这种关系变量的不可约又是与设计理论相关的许多种不可约性都不相同的。你能找出多少种不可约性？

13.4 请给出（a）FD冗余；（b）JD冗余；（c）RFNF的定义。

13.5 让我们再来看本章正文中的关系变量P到6NF投影PL、PN、PW和PC的分解。你能想到对此设计的任何改进吗？

13.6 假设你有一个代表婚姻的关系变量R，它具有属性A、B、C且谓词是人A与人B在日期C结婚。假定不存在多配偶制，假设也没有两个人彼此结婚不止一次。R有哪些键？ $JD \bowtie \{AB, BC, CA\}$ 存在吗？R所属的最高范式是什么？

13.7 我在本章正文说过，一个关系变量可能属于SKNF，但不属于5NF，并且举了以下这样的关系变量作为一个例子：

■设关系变量R具有属性A、B、C（只有这些属性），设AB、BC、CA每个都是R的键，并设在R中存在 $JD \bowtie \{AB, BC, CA\}$ ，把它称作J。

但是，你可能有理由对这个例子有点怀疑。为了更具体，你可能想知道，服从指定的键约束和指定的JD（虽然我确实继续给出了此例子一个稍微更具体的版本）的一个关系变量能否存在？通过证明所有可能的依赖（FD和JD）集合其实都是一致的来显示这个例子无论如何是合理的，在这个意义上，至少总能找到一个满足集合中所有依赖的关系。

13.8 13.4节的关系变量SPJ'服从一个所谓的“对称的”JD。即， $JD \bowtie \{\{SNO, PNO\}, \{PNO, JNO\}, \{JNO, SNO\}\}$ ，但还展示出了某种不对称性，在JD的三个分量中，其中只有一个分量对应于一个键。直观地，你可能期望其他两个分量也对应于键。请说明情况不一定如此。

13.9 为以下案例设计一个数据库。要代表的实体是足球比赛中的某支球队的比赛项目。对

于已经完成的比赛，我们希望记录“进球数”和“失球数”，但是，这两个属性对还未举行的比赛显然是毫无意义的。你的关系变量属于什么范式呢？

13.10 设关系变量SCP具有属性SNO、PNO、CITY，谓词是供应商SNO和零件PNO均位于城市CITY。SCP能从我们平时的S、P和SP关系变量派生出吗？它属于什么范式？你是否能想到这个例子可能面对的任何传统的看法？

13.11 请给出D K/N F的定义。请举出一个属于6N F但不属于D K/N F的一个关系变量的例子。

13.12 SKNF与过强的PJ/NF有什么区别？它们事实上有区别吗？

13.13 尽你所能地给出关系操作符“限制”和“合并”的精确定义。

13.14 在本章正文中，我非正式地展示了为何把一个关系变量精简为6NF投影相当于把一个合取谓词精简为简单谓词。可能存在类似“析取谓词”这样的东西吗？一个关系变量如何可能对应于这样一个谓词呢？把这样一个谓词精简为简



单谓词会涉及什么呢？

## 第四部分 正交

重申一遍我在第1章中所说的东西，数据库设计不是我最喜欢的主题。原因是，设计实践只有很少一部分是真正的科学：当然，规范化确实是科学的，但其他的没有多少科学。然而，本书这个部分讨论的正交性的主题，确实代表其余仍然（可惜）相当主观的领域中另一小块科学。

## 第14章 正交设计原则

Orthogonal At right angles to; independent.

——David Darling: The Universal Book of Mathematics

注意：本章的一部分曾以相当不同的形式，最早出现在我的书《Date on Database: Writings 2000-2006》（Apress, 2006）中。

首先，我将以快速回顾规范化的原则和对规范化符合其目标的程度分析开始本章的阐述。下面是这些原则的一个总结。

1. 一个不属于RFNF的关系变量应该“完全规范化”，即分解成（至少是）RFNF投影的一个集合。

2. 通过连接把这些投影重新结合在一起，原来的关系变量应该是可重构的，即，分解应该是无损的。

3. 在分解过程中应保持依赖（FD和JD），至少在不违反原则1的前提下可以这样做。

4. 在重建过程中，每个投影都应该是必需

的。

## 14.1 规范化的两个欢呼声

正如我已经多次重申的，规范化是数据库设计底层的科学（或至少是科学的很大一部分）。但它不是万能的，通过考虑它的目标是什么，以及它对这些目标的实现程度，我们可以很容易地看到这一点。这些目标如下所示。

- 实现一个设计，它是现实世界的一个“好”的代表（即一个直观上易于理解并为未来的发展奠定了良好基础的设计）

- 减少冗余

- 从而避免否则可能会出现某些更新异常

- 简化语句和执行某些完整性约束

我会逐个地分析每个目标。

- 真实世界的良好的代表：规范化在这方面做得出色。在这里我没有可批评的。

- 减少冗余：在这个问题上，规范化是一个良好的开端，但它仅仅是一个开端。一方面，执

行投影是一个过程，并且我们已经看到了并非所有的冗余都可以通过投影删除，事实上，有许多种的冗余是规范化根本无法解决的。（第15章将讨论这个问题的细节。）另一方面，即使在无损分解的情况下，投影也可能会导致丢失依赖，正如我们在第6章及其他地方看到的。

■避免更新异常：这一点，至少部分地，只是前一个目标的另一种说法。众所周知，没有正确地规范化的设计，之所以可能会遭受特定的更新异常，正是因为它们带来的冗余。例如，在关系变量STP中，（参见第1章中的图1-2），供应商S1可能会显示为在一个元组具有状态20而在另一个元组具有状态25。当然，只有在完整性约束执行得不够理想时，这种特殊的异常才可能会出现.....也许思考更新异常问题的一个更好的方式是这样的：如果设计是适当地规范化的，那么它比如果没有这么做，防止此类异常状态所需的约束会更容易表达，而且可能更容易执行（请参阅下一段）。然而，思考它的另一种方式是：如果设计是适当地规范化的，那么它比如果没有这么做，单个元组更新<sup>[1]</sup>在逻辑上会更加可以接受

（因为非规范化的设计意味着冗余，即多个元组说的是同样的事情，而冗余意味着，有时我们必须要在同一时间更新几件事情）。

■简化语句和约束的执行：正如我们从前面的章节中知道的，一些依赖蕴含其他依赖。（更一般地，实际上，任何形式的约束都可以蕴含其他约束，举一个简单的例子，如果发货数量必须小于或等于5000，它们必然小于或等于6000。）现在，如果约束A蕴含约束B，那么说明并执行A实际上会“自动地”说明和执行B（事实上，除了可能以文档的方式说明外，B将完全不再需要单独列出）。规范化到5NF给出了一个非常简单的方法来说明和执行某些重要的制约，基本上，我们要做的是定义键和实现它们的唯一性（这是我们无论如何希望做的），那么所有适用的JD（因此所有的MVD和FD）实际上将自动得到说明，并自动执行，因为它们都被那些键蕴含。所以规范化也在这方面做得很好。（当然，我在这里忽略了规范化过程中很可能会引起的各种多关系变量约束。）

另一方面，除了上述已经给出的原因以外，这里还有说明为什么规范化不是什么灵丹妙药的几个原因。

■首先，JD、MVD和FD不是仅有的约束种类，规范化不利于其他任何种类的约束。

■其次，给出特定的关系变量的一个集合，

经常会存在无损地分解成5NF投影的多种不同分解方法，并且在这种情况下，只有很少或根本没有正式的指导，告诉我们应该选择哪一种分解方法。（说实话，虽然我怀疑在实践中这种缺失是否可能导致重大问题。）

■最后，还有很多的设计问题，规范化根本不能解决。例如，应该只有一个供应商关系变量，而不是伦敦的供应商有一个，巴黎的供应商也有一个，等等，这告诉我们什么？这当然不是传统上理解的规范化。

话虽如此，但我必须说清楚，我不希望上述意见被视为任何形式的攻击。正如我在第8章所说的，我相信任何一个没有完全规范化的设计都应该强烈地禁止。但事实仍然是规范化（“设计的科学部分”）真的没有做如我们所希望的那么多的工作，所以，现在已经有一小块额外的科学提供给我们，这可以说是很好的。这就是正交性主题所涉及的全部内容。注意：正交的概念随着时代的变迁已经演变。因此，本章的部分内容与以前的著作（大多数是我自己的）在这一问题上是有分歧的。更重要的是，我也很怀疑是否本章的内容就是定论。我确实相信这一章截至目前是准确的，然而，对这些材料的进一步精炼也许是有可能的，且理想的。需要注意辨别。

## 14.2 一个启发性的例子

为简单起见，假设 $FD\{CITY\} \rightarrow \{STATUS\}$ 在关系变量S中不存在（请注意，我会在本章中通篇保持这个假设）。请看以下对此关系变量的分解：

---

```
SNC{SNO,SNAME,CITY}
KEY{SNO}
STC{SNO,STATUS,CITY}
KEY{SNO}
```

---

它的示例值如图14-1所示。正如图14-1所示，分解几乎不是非常明智的（特别要注意，给定的供应商位于一个城市的事实出现了两次），但它遵守所有的规范化原则——两个投影都属于5NF；分解是无损的；依赖得以保留；并且在重建过程中这两个投影都是必需的。



SNC

SNO	SNAME	CITY
S1	Smith	London
S2	Jones	Paris
S3	Blake	Paris
S4	Clark	London
S5	Adams	Athens

STC

SNO	STATUS	CITY
S1	20	London
S2	30	Paris
S3	30	Paris
S4	20	London
S5	30	Athens

图 14-1 关系变量SNC和STC的示例值

直观地看，上述设计的问题是显而易见的：当且仅当元组  $(s,t,c)$  出现在STC时，元组  $(s,n,c)$  在SNC中出现；等价地，当且仅当正好相同的元组  $(s,c)$  出现在STC在SNO和CITY上的投影时，元组  $(s,c)$  在SNC在SNO和CITY上的投影中出现。为了稍微更正式地说明这个问题，我们可以说这个设计存在以下相等依赖（EQD）：

---



---

CONSTRAINT.....SNC{SNO,CITY}=STC{SNO,CITY};

---



---

并且这个EQD造成了明显的冗余。

然而，再次重申，上述设计遵守规范化的所有既定原则。因此，这些原则本身是不够的，我

们需要别的东西来告诉我们这个设计有什么错误（别的正式的东西，因为大家都非正式地知道它在什么地方错了）。把此事换一种说法，在我们减少冗余的努力中，规范化原则提供了一套正式的原则来指导我们，但这套原则本身是不够的，因为这个例子清楚地表明。我们需要另一个原则。换句话说，正如我在本书中已经不止一次说过的，我们需要更多的科学。

## 14.3 一个更简单的例子

为了能看到我们需要的原则可能看起来像什么，让我们考虑一个简单的例子。正如你所知道的，如前所述的规范化（特别是在上一节的例子中所用的规范化）是关于“垂直”分解关系变量的（这意味着通过投影分解）。但是，“水平”分解（即，通过限制分解）显然也是可能的。考虑如图14-2所示的设计，将零件关系变量P水平地分离（事实上，划分）成两个关系变量，其中一个包含重量小于17.0磅的零件（“轻零件”，LP），另一个包含重量大于或等于17.0磅的零件（“重零件”，HP）。（为了明确起见，我假设WEIGHT的值代表的是用磅计量的重量。）

谓词如下。

■LP: 零件PNO名为PNAME, 具有颜色COLOR和重量WEIGHT (小于17.0), 并存储在城市CITY中。

■HP: 零件PNO名为PNAME, 具有颜色COLOR和重量WEIGHT (大于或等于17.0), 并存储在城市CITY中。

LP	PNO	PNAME	COLOR	WEIGHT	CITY
	P1	Nut	Red	12.0	London
	P4	Screw	Red	14.0	London
	P5	Cam	Blue	12.0	Paris

HP	PNO	PNAME	COLOR	WEIGHT	CITY
	P2	Bolt	Green	17.0	Paris
	P3	Screw	Blue	17.0	Paris
	P6	Cog	Red	19.0	London

图 14-2 关系变量LP和HP的示例值

注意: 通过求LP和HP关系变量 (不相交) 的并集, 就可以恢复原来的关系变量P。

我们为什么也许要进行这样的水平分解? 坦

率地说，我并不知道这样做有什么合乎逻辑的好理由，但当然，这并不是说没有这样的理由存在。即使如此，注意，我们可以并应当说明适用于这些关系变量的两个约束：

---

CONSTRAINT LPC AND (LP,WEIGHT<17.0) ;  
CONSTRAINT HPC AND (HP,WEIGHT≥17.0) ;

---

（我在第2章提醒过你，Tutorial D的表达式AND (rx,bx)，其中，rx是一个关系表达式，bx是一个布尔表达式，当且仅当bx对rx所表示的关系中的每一个元组的计算结果都为TRUE时，这个表达式返回TRUE。）

因此，我们这里有一个稍微不寻常的情况：具体地说，在关系变量LP和HP的案例中，零件的谓词可以而且应该以显式约束的形式正式地捕获。事实上，需要说明和执行这样的约束这一事实恰恰可能会被视为妨碍设计。但是，即使水平分解在逻辑层面上是禁止的，仍有很多实际的理由（关于恢复、安全性能等事项）要求在物理层做这样的分解，因此，在当今的DBMS中，给定的逻辑层和物理层往往几乎是步调一致的，即，那些DBMS并不具有它所应该有的那样多的数据独立性，因此，至少在目前的产品中，在逻辑层

面也执行这样一个分解，有可能有务实的原因。

现在，无论你可能对上述论点有什么样的看法，至少在图14-2中没有明显坏的设计（为了这个例子，让我们无论如何赞成这一点）。<sup>[2]</sup>但假设我们只是稍微不同地定义关系变量LP，具体地说，假设我们把它定义为包含重量小于或等于17.0磅的这些零件（当然，也相应地调整谓词和约束LPC）。下面所示的图14-3是图14-2修改后的版本，它显示这个修改后的设计中发生了什么。正如你可以看到的，现在绝对有坏事发生，具体地说，现在零件P2和P3的元组在图14-3所示的两个关系变量中都出现（换句话说，现在有某种冗余）。更重要的是，这些元组必定出现在这两个关系变量中！反之，假设（比如说）零件P2的元组出现在HP中，而不出现在LP中。那么，由于LP不包含零件P2的元组，根据封闭世界假设我们可以合理地得出结论（见第2章），零件P2的重量不可能是17.0磅。但随后我们从HP看到，零件P2其实确实重17.0磅，因此，这个数据库是不一致的，因为它包含一个矛盾。（当然，在一个数据库中的不一致性是极不可取的，正如我在《SQL与关系数据库理论》中表明的，你永远不能相信你从不一致的数据库中得到的结果，事实上，从这样的数据库你完全可以得到任何可能的

结果，甚至是产生如 $0=1$ 这样荒谬的事情的结果！)

LP	<u>PNO</u>	PNAME	COLOR	WEIGHT	CITY
	P1	Nut	Red	12.0	London
	P2	Bolt	Green	17.0	Paris
	P3	Screw	Blue	17.0	Paris
	P4	Screw	Red	14.0	London
	P5	Cam	Blue	12.0	Paris

HP	<u>PNO</u>	PNAME	COLOR	WEIGHT	CITY
	P2	Bolt	Green	17.0	Paris
	P3	Screw	Blue	17.0	Paris
	P6	Cog	Red	19.0	London

图 14-3 关系变量LP（修订版）和HP的示例值

现在，不难看出图14-3的设计中的问题：LP和HP的谓词“重叠”，在这个意义上，完全相同的元组 $t$ 可同时满足它们。更重要的是，如果 $t$ 是这样一个元组，且如果在某些给定的时间，元组 $t$ 代表一个“真正的事实”，那么，按照封闭世界假设，在所涉及的时间，该元组 $t$ 必然同时出现在两个关系变量（当然，因此导致冗余）。事实上，

我们拥有另一个EQD:

---

CONSTRAINT..... (LP WHERE WEIGHT=17.0) =  
(HP WHERE WEIGHT=17.0) ;

---

再说一遍，这个例子中的问题是，我们允许两个关系变量有重叠的谓词。那么，很显然，我们正在寻找的原则应当会给出类似这样的建议：不要那样做！让我们尝试更精确一点地说明这种情况。

■定义（第一次尝试）：正交设计原则指出，如果关系变量R1和R2是不同的，那么任何一个元组都必须不存在下面的性质：如果它出现在R1中，那么它也必须出现在R2中，反之亦然。<sup>[3]</sup>注意：这里的术语正交源于该原则实际上说的是关系变量应该是相互独立的（如果它们的含义在上述意义上重叠，那么它们不会是独立的）这一事实。

在下文中，我有时会把上述原则（或者更确切地说，这一原则的各种版本）缩写为正交性原则，或有时只写为正交性。

旁白：与本书中的其他地方一样，可能会指

责我在上述阐述中耍了一个小小的花招。让我们再次观察图14-3，特别是观察零件P2的元组。该元组同时出现在LP和HP中，因为它既是LP的谓词的一个真正的实例，又是HP的谓词的一个真正的实例。真的是这样吗？零件P2的这些谓词的实例实际上如下：

■LP：零件P2名为螺栓，具有颜色绿色和重量17.0（这是小于或等于17.0），并存储在城市巴黎中。

■HP：零件P2名为螺栓，具有颜色绿色和重量17.0（这是大于或等于17.0），并存储在城市巴黎中。

这两个命题是不一样的！当然，它们肯定彼此相等，但为了认识到这个等价性，我们需要知道“ $17.0 \leq 17.0$ ”和“ $17.0 \geq 17.0$ ”都是成立的，那么我们就需要运用一点逻辑推理。（问题的关键是，对于我们人类是明显的东西，对于机器不一定也是明显的，为了完整性，我真的应该在我的论点中清楚阐述缺少的步骤。）结束旁白。

现在，在轻与重的零件的例子中坚持正交性原则一定能避免如图14-3所示的冗余。然而，请注意，所列的原则来仅适用于标题完全相同的关



系变量，如LP和HP，因为如果所涉及的关系变量有不同的标题，当然就不可能发生相同的元组出现在两个不同的关系变量中的情况。因此，你可能会认为正交性原则是没有多大用处的，因为在实践中同一个数据库中不太可能有两个具有相同标题的关系变量。<sup>[4]</sup>如果这就是全部，我可能会同意你的看法，我的意思是，在这种情况下，生活将是相当简单的，而本章可以到此为止（定义这样一个非常明显的原理甚至都可能不值得用盛大的标签“原则”）。但是，当然，对此问题还有很多可阐述的内容。为了进一步探讨这种可能性，我首先需要仔细观察元组和命题之间的关系。

## 14.4 元组与命题

我们知道，在某个特定的时间出现在某个特定的关系变量R中的每一个元组都代表一定的命题，所涉及的命题是关系变量R的关系变量谓词的一个实例，认为这个谓词（按照约定）在所涉及的时间中是真实的。例如，这里又是来自图14-2和图14-3的关系变量HP的一个谓词：

零件PNO名为PNAME，具有颜色COLOR和重量WEIGHT（这大于或等于17.0），并存储在城市CITY中。

除此以外，此关系变量包含零件P6的一个元组，该元组表示上述谓词的下列实例：

零件P6名为嵌齿轮，具有颜色红色和重量19.0（这是大于或等于17.0），并被存储在城市伦敦。

粗略地讲，那么，我们可以说数据库“包含命题”。记得吗，在这本书的前面部分中我已经说过多次，当且仅当数据库把同样的事情描述了两次时，它包含某种冗余。现在，我可以让这句话更确切一点：

■定义：当且仅当数据库包含同一个命题的两个不同的陈述时，该数据库包括冗余。

那么，鉴于元组代表命题，这很容易诱使人把上述定义翻译成下面这样：当且仅当数据库包含同一元组的两次不同的出现时，该数据库包含某种冗余。[\[5\]](#)遗憾的是，后一个“定义”充其量也是相当简化的。让我们更仔细地研究它。

首先，至少我们真的不希望相同的元组在相同的关系变量中和相同的时间出现不止一次，因为这样的状况肯定将构成“把同样的事情描述两次”。（因为我曾经听Codd说过：如果某件事情

是真的，把它描述两次并不使它更真实。）当然，这种需求由关系模型本身负责，根据定义，关系永远不会包含重复元组，因此，对于关系变量也同样如此，因此我们可以忽略这种可能性。注意：换句话说，人们可能会认为，一个避免冗余的愿望是选择集合（根据定义，它不包含重复的元素）而不是“背包”，作为建立一个坚实的数据库理论的正确数学抽象基础的动机之一（可能是一个小的动机）。SQL辩护者请注意！

旁白：我顺便注意到，现在我们对“重复元组”（人们随时都在用这个短语，但我很怀疑如果在被催促时他们是否能够精确地定义它）的概念有了一个精确的描述。从严格意义上讲，当然，当且仅当两个元组是完全相同的元组时，它们才是重复的，就像当且仅当两个整数是完全相同的整数时，它们才是重复的。从逻辑的角度来看，“重复元组”这个短语实际上并没有多大意义（说两个不同的元组重复是一个矛盾）。人们在使用这句话时，他们真正谈论的是相同元组的重复出现。由于这个原因，在数据库环境中经常遇到的“重复消除（duplicate elimination）”这个我们都知道的短语，更好的表达应该是“消除重复”（duplication elimination）。但是，我离题了……让我们回到主要的讨论中。结束旁白。

其次，我们也不希望相同的子元组不止一次地出现在同一个关系变量（再次，在同一时间）中。<sup>[6]</sup>但是，经典的规范化负责这项任务；例如，正是因为 $FD\{CITY\} \rightarrow \{STATUS\}$ 在关系变量S中存在，导致相同的 $\{CITY, STATUS\}$ 对多次出现（每次出现具有相同的含义），所以建议我们把那个关系变量替换为其在 $\{CITY, STATUS\}$ 和 $\{SNO, SNAME, CITY\}$ 上的投影。

我的下一个观点是完全相同的元组可以表示任何数量的不同命题，这可以很容易地看到。举一个简单的例子，设SC和PC分别是关系变量S在CITY上的投影和关系变量P在CITY上的投影。根据我们通常的示例值，那么，一个只包含CITY值“London”的元组同时出现在SC和PC中，但是这两个出现代表不同的命题。具体地说：在SC中的出现代表命题在伦敦至少有一个供应商，而在PC中的出现代表命题在伦敦至少有一个零件（为了本例的目的，在这两种情况下都稍微作了简化）。

更重要的是，在这里我一定要更正式一点地阐述，同一个命题也可以用任何数量的不同元组表示。这是因为，从形式上看，相关的属性名称是元组的一部分（如果你需要确认这一点，请检查在第5章中元组的定义）。因此，例如，我们

可能有我们常用的发货关系变量SP，它的属性是SNO、PNO、QTY，且谓词是：

供应商SNO提供数量为QTY的零件PNO。

另外，我们可能还有一个属性为SNR、PNR和AMT的关系变量PS，它的谓词是：

供应商SNR提供数量为AMT的零件PNR。

（使用Tutorial D语法）下列元组可能分别出现在关系变量SP和PS中：

---

---

```
TUPLE{SNO'S1', PNO'P1', QTY 300}
```

```
TUPLE{SNR'S1', PNR'P1', AMT 300}
```

---

---

这些元组显然是不同的，但它们都代表同一个命题：

供应商S1提供数量为300的零件p1。

事实上，两个关系变量SP和PS中的任意一个都可以用另一个形式来定义，当下面的约束（实际上再次是EQD）一起显示时：

---

---

CONSTRAINT.....

```

PS=SP RENAME{SNO AS SNR,PNO AS PNR,QTY AS
AMT};
CONSTRAINT.....
SP=PS RENAME{SNR AS SNO,PNR AS PNO,AMT AS
QTY};

```

---

一个包含这两个关系变量的数据库显然将包含某种冗余。[\[7\]](#)

上述讨论的实质是这样的：在元组和命题之间有一个多对多的关系，即，任意数量的元组可以代表同一个命题，任意数量的命题可以用相同的元组表示。鉴于这种状况，那么，下面尝试对正交性原则做出更精确一点的说明。

■定义（第二次尝试）：设关系变量 $R$ 和 $R_2$ 是不同的，并设它们分别具有标题 $\{A_1, \dots, A_n\}$ 和 $\{B_1, \dots, B_n\}$ 。设关系变量 $R_1$ 的定义如下：

---


$$R_1 = R \text{ RENAME } \{A_1 \text{ AS } B_1', \dots, A_n \text{ AS } B_n'\}$$


---

其中， $B_1', \dots, B_n'$ 是 $B_1, \dots, B_n$ 的某种置换。（注意， $R_1$ 和 $R_2$ 因此有相同的标题。）那么正交设计原则指出，不存在每个都不恒为假的限制条件 $c_1$ 和 $c_2$ ，使得相等依赖（ $R_1 \text{ WHERE}$

$c1) = (R2 \text{ WHERE } c2)$  存在。

从第二次尝试产生的要点如下。

■这个版本的原则确实解决了图14-3的设计的问题：首先，设R和R2分别是LP和HP，从而定义R1如下：

---

---

$R1 = LP \text{ RENAME } \{PNO \text{ AS } PNO, \dots, CITY \text{ AS } CITY\}$

---

---

（换句话说，设R1恒等于R）其次，设c1和c2都是限制条件WEIGHT=17.0。那么相等依赖  $(R1 \text{ WHERE } c1) = (R2 \text{ WHERE } c2)$  成立，因此这样的设计违反了正交设计原则。注意：如这个例子所示，只要c1和c2不恒为假，就存在这样特定的元组，它们必须同时在R1和R2中出现，假设它们代表“真正的事实”——而基本上，这种情况是我们想要取缔的。（相比之下，如果c1和c2恒为假，那么限制R1 WHERE c1和R2 WHERE c2都将是空的，因而不会违反任何正交性。）

■事实上，这个版本的原则包含以前的版本，因为（a）我们可以使R1恒等于R（如在前面的列表项中那样，使重命名实际上是“无操作”）且（b）我们可以设c1和c2中的每个都只是

TRUE。（正如我刚才所说的，以前版本的原则确实假设所涉及的关系变量具有相同的标题。然而，正如本节的讨论所表明的，我们不能把我们的注意力只限制在那种简单的情况中。）

■回顾一下第6章，在逻辑上，恒为假的东西（例如，布尔表达式 $\text{WEIGHT} \geq 17.0$  AND  $\text{WEIGHT} < 17.0$ ）称为一个矛盾。因此，此c1和c2不恒为假的要求可以这样表述：c1和c2两者在逻辑意义上都不是一个矛盾。

## 14.5 第一个例子再探

现在，让我们回到我们的启发例子，其中把关系变量S“垂直”地分解为其分别在 $\{\text{SNO}, \text{SNAME}, \text{CITY}\}$ 和 $\{\text{SNO}, \text{STATUS}, \text{CITY}\}$ 上的投影SNC和STC。（当然，轻与重零件的例子涉及水平分解）。现在请注意，虽然SNC和STC的度必定是相同的，但是任何给定的元组都没有办法可以同时出现在两者之中：SNC中的元组具有SNAME属性，而STC中的元组没有SNAME属性，而代之以一个STATUS属性。更重要的是，我们没有办法可以简单地把SNC中的SNAME属性重命名为（比如说）STATUS，并从而产生一个与STC具有相同标题的关系变量，因为在SNC中SNAME的类型是CHAR而在STC中STATUS的类型



是INTEGER。（重命名属性更改的是名称，而不是类型。）因此，我们的第二次尝试定义的正交性原理仍是不充分的。事实上，在目前的情况下，它根本不适用。

现在回顾一下上述设计中存在的问题是什么：当且仅当完全相同的元组（s,c）出现在STC在SNO和CITY上的投影中时，元组（s,c）才在SNC在SNO和CITY上的投影中出现。也就是说，以下EQD存在：

---

---

$$\text{CONSTRAINT} \dots \text{SNC}\{\text{SNO}, \text{CITY}\} = \text{STC}\{\text{SNO}, \text{CITY}\};$$

---

---

让我们同意此刻先忽视属性重命名的问题，因为它与这个例子是不相关的。那么，关键的一点是，这个EQD不是存在于如前所述的不同的数据库关系变量之间的，而是存在于相同的数据库关系变量的不同的投影之间的。具体地说，存在于对该数据库关系变量进行“纵向”分解所产生的投影中。但是，当然情况并不总是如此，我的意思是，SNC和STC有可能独立地定义为两个完全不同的关系变量，而不曾存在（在设计师的脑海里，可以这么说）一个与它们的连接相等的关系变量。它们甚至可能不是关系变量本身，而是两个这样的不同关系变量的投影。所有这一切都导

致对正交性原理进行定义的第三次尝试。

■定义（第三次尝试）：设关系变量 $R_1$ 和 $R_2$ 是不同的，并设 $JD \bowtie \{X_1, \dots, X_n\}$ 关于 $R_1$ 不可约。<sup>[8]</sup>设存在某个 $X_i$  ( $1 \leq i \leq n$ ) 和 $R_1$ 在 $X_i$ 上的投影（比如说 $R_1X$ ）上的某个可能为空的属性重命名集合，这个重命名把 $R_1X$ 映射到比如说 $R_1Y$ ，这里的 $R_1Y$ 具有和 $R_2$ 的标题的某个子集 $Y$ 相同的标题。此外，设 $R_2$ 在 $Y$ 上的投影是 $R_2Y$ 。那么，如果相等依赖 $R_1Y = R_2Y$ 存在，则 $R_1$ 和 $R_2$ 违反了正交设计原则。

现在，这看起来有点复杂，但基本上它说的是，在 $R_1$ 的无损分解中，没有任何投影可以信息等价于 $R_2$ 的任何投影。事实上，正如你也许可以看到的，（不论多么复杂的）定义中的许多复杂性来自处理重命名问题的需要。为了有意义，这里有一个更简单的版本的定义，它忽视这种复杂性。

■定义（第三次尝试，忽略重命名）：设关系变量 $R_1$ 和 $R_2$ 是不同的，并设 $JD \bowtie \{X_1, \dots, X_n\}$ 是关于 $R_1$ 不可约的。设某个 $X_i$  ( $1 \leq i \leq n$ ) 与 $R_2$ 的标题的某个子集 $Y$ 是相同的，此外，设 $R_1$ 在 $X_i$ 上的投影和 $R_2$ 在 $Y$ 上的投影分别是 $R_1X$ 和 $R_2Y$ 。那么，如果存在相等依赖 $R_1X = R_2Y$ 成立， $R_1$ 和

R2违反正交设计原则。

现在观察坚持这第三个版本的原则是怎么通过启发性的示例解决问题的，在这个例子中，关系变量S分别被分解成其在{SNO,SNAME,CITY}和{SNO,STATUS,CITY}上的投影SNC和STC。假设分解已经完成。那么：

a.数据库现在包含两个不同的关系变量，即SNC和STC。

b.由于希思定理和 $FD\{SNO\} \rightarrow \{SNAME\}$ 在关系变量SNC中存在的事实， $JD \bowtie \{\{SNO,SNAME\}, \{SNO,CITY\}\}$ 在关系变量SNC中存在且实际上是关于SNC不可约的。

c.因此，关系变量SNC在{SNO,CITY}上的投影是SNC的一个有效的无损分解的一部分。但在那个投影和STC在那些相同的属性上的投影之间存在一个相等依赖。因此，正如刚才详细说明的（“第三次尝试”），该设计违反正交性原则。

注意：这里的b点以及c点可以用以下内容代替，而不改变整体的信息：

b.由于希思定理和 $FD\{CITY\} \rightarrow \{STATUS\}$ 在

关系变量STC中存在的事实，  
 $JD \bowtie \{\{CITY, STATUS\}, \{CITY, SNO\}\}$ 在关系变量STC中存在且其实关于STC是不可约的。

c.因此，关系变量STC在 $\{CITY, SNO\}$ 上的投影是STC的一个有效的无损分解的一部分。但在那个投影和SNC在那些相同的属性上的投影之间存在一个相等依赖。因此，正如刚才详细说明的（“第三次尝试”），该设计违反正交性原则。

我现在注意，第三个版本的正交性原则也让我处理了一件来自第11章的未完成的工作。你可能还记得，我曾在那一章中指出，在关系变量S中存在的以下JD其实关于那个关系变量是不可约的：

---

$$\bowtie \{\{SNO, SNAME, CITY\}, \{CITY, STATUS, SNAME\}\}$$

---

我还曾说，在这个JD的基础上分解关系变量S，将不会是一个好主意（且习题11.4问了为什么不是）。那么，现在我们可以看到，如果进行那样的分解：

a.现在该数据库包含两个不同的关系变量（我称它们为SNC和CTN），它们的标题分别是

$\{SNO, SNAME, CITY\}$ 和  
 $\{CITY, STATUS, SNAME\}$ 。

b. 由于希思定理和  $FD\{CITY\} \rightarrow \{STATUS\}$  在关系变量CTN中存在的事实，  
 $JD \bowtie \{\{CITY, STATUS\}, \{CITY, SNAME\}\}$  在那个关系变量CTN中存在且其实关于CTN是不可约的。

c. 因此，关系变量CTN在  $\{CITY, SNAME\}$  上的投影是CTN的一个有效的无损分解的一部分。但在那个投影和SNC在那些相同的属性上的投影之间存在一个相等依赖。换言之，该设计再次违反正交性原则。

这个例子的实质是这样的：根据正交设计原则，在一个“坏”JD的基础上做一个分解是禁止的。（在本例中的JD是“坏”的，是因为属性SNAME可以从  $\{CITY, STATUS, SNAME\}$  分量中去掉而没有显著损失。）更重要的是，遵守正交原则的一个后果是，在本章开始部分给出的第四条规范化原则（即，每个投影在重建过程中都是必要的）会自动得到满足（所以，无论如何，正交性和规范化之间存在一个逻辑上的某种联系）。

## 14.6 第二个例子再探

遗憾的是，在上一节所定义的正交性原则的第三个版本仍然缺失一些东西，而且重新审视轻与重的零件的例子显示缺失的是什么：它缺少有关限制的处理。（在该示例中，那个EQD既不是在如前所述的数据库关系变量之间的，也不是在那种关系变量的投影之间的，而是在那种关系变量的特定限制之间的。）换句话说，第三个版本的原则未能包含第二个版本。与此相反，如下所示的定义同时处理了限制问题和投影问题。

■定义（第四次尝试）：设关系变量R1和R2是不同的，并设 $JD \bowtie \{X_1, \dots, X_n\}$ 关于R1不可约。设存在某个 $X_i$  ( $1 \leq i \leq n$ ) 和R1在 $X_i$ 上的投影（比如说R1X）上的某个可能为空的属性重命名集合，这个重命名把R1X映射到（比如说）R1Y，这里的R1Y具有和R2的标题的某个子集Y相同的标题。此外，设R2在Y上的投影是R2Y。那么，当且仅当存在每个都不恒为假的限制条件c1和c2，使得相等依赖  $(R1Y \text{ WHERE } c1) = (R2Y \text{ WHERE } c2)$  存在时，则R1和R2违反正交设计原则。

## 14.7 最终版本

不管你相信与否，上述定义还有一个小问题……考虑某个版本的供应商关系变量（我称之

为SCC)具有属性SNO、CITYA和CITYB。对于任何给定的供应商,设SCC服从CITYA和CITYB的值恒等的约束。结果:冗余!当然,这是一个疯狂的设计,但它是一个可能的设计,并且对正交性原则进行扩展,以处理这样的设计也将是很好。而下面的最终(?)定义应该达到这个目的(我会把它作为一道习题留给你,让你来找出它到底是如何做到的)。

■定义(“最终”版本):设R1和R2是关系变量(不一定是不同的),并设 $JD \bowtie \{X_1, \dots, X_n\}$ 关于R1不可约。设存在某个 $X_i$  ( $1 \leq i \leq n$ )和R1在 $X_i$ 上的投影(比如说R1X)上的某个可能为空的属性重命名集合,这个重命名把R1X映射到(比如说)R1Y,这里的R1Y具有和R2的标题的某个子集Y(如果R1和R2是同一个关系变量,那么Y是不同于 $X_i$ 的)相同的标题。此外,设R2在Y上的投影是R2Y。那么,当且仅存在每个都不恒为假的限制条件c1和c2,使得相等依赖(R1Y WHERE c1) = (R2Y WHERE c2)存在时,则R1和R2违反正交设计原则。

这个版本的原则涵括了以前的所有版本。

## 14.8 澄清

抱歉我必须披露，关于正交性的文献中有相当多的混乱，即使其基本思路如此简单。更遗憾的是，我不得不说这种混乱有一部分是我的错，我以前关于这一主题的一些著作是（直截了当地说）完全错的。所以，让我借这个机会尝试并澄清。其基本要点是这样的：正交性指出关系变量不应该有重叠的含义，它不是说关系变量不应该有相同的标题（或更一般地，“重叠”的标题）。下面是由休·达尔文提供的一个简单例子，它说明了两者的不同。考虑谓词员工ENO在度假且员工ENO正在等待电话号码分配。针对这种情况最明显的设计涉及两个度为1的关系变量，它们看起来像这样（概要）：

---

```
ON_VACATION{ENO}  
KEY{ENO}  
NEEDS_PHONE{ENO}  
KEY{ENO}
```

---

显然，完全相同的元组可以同时出现在两个关系变量中。但是，即使如此，这两次出现代表两个不同的命题，并且没有任何冗余，并且没有违反正交性。

现在请注意，刚才讨论的例子与在本章前面图14-2和图14-3的轻与重的零件的例子（关系变



量LP和HP)之间有某种差异。正如我们前面所看到的,在后一种情况下,我们可以写一个正式的约束,大意是WEIGHT值必须在一定范围内,一个给定的元组为了能被LP或HP接受或被两者同时接受,必须满足它。然而,我们没法写一个给定的元组必须满足的一个正式的约束,以便它被ON\_VACATION或NEEDS\_PHONE接受或被两者同时接受。如果用户断言,一个特定的元组要插入,比方说,ON\_VACATION中,那么系统只需信任该用户;没有办法执行一种检查来确定该元组确实属于ON\_VACATION,而不属于(或也属于)NEEDS\_PHONE。

下面是休·达尔文提供的另一个例子,也可能错误地认为它违反了正交性,但实际上并不违反。假设我们有三个关系变量,它们看起来像这样(概要): [\[9\]](#)

---

```
EARN{ENO,SALARY}
KEY{ENO}
SALARY_UNK{ENO}
KEY{ENO}
UNSALARIED{ENO}
KEY{ENO}
```

---

它的示例值如图14-4所示。

EARN\$		SALARY_UNK	UNSALARIED
ENO	SALARY	ENO	ENO
E1	85,000	E2	E4
E3	70,000		

图 14-4 关系变量EARN\$、SALARY\_UNK、UNSALARIED的示例值

这三个关系变量的谓词如下所示。

■EARN\$: 员工ENO的薪金是SALARY。

■SALARY\_UNK: ENO员工有一份薪金，但我们不知道它是什么。

■UNSALARIED: 员工ENO没有薪金。

现在，关系变量SALARY\_UNK和UNSALARIED确实有相同的标题，但即使相同的元组可以同时出现在两者之中，也不会有任何冗余，因为所涉及的出现将代表两个不同的命题。当然，实际情况的语义是这样的，无论如何，任何元组不应同时出现在两者之中（换句话说，关系变量是不相交的）。下面是用来表达这个事实的必要的完整性约束：

---

```
CONSTRAINT.....IS_EMPTY (SALARY_UNK JOIN  
UNSALARIED) ;
```

---

（正如在附录D中习题6.4的答案，如果关系r是空的，Tutorial D的表达式

---

```
IS_EMPTY (r)
```

---

返回TRUE，否则返回FALSE）。

## 14.9 结束语

最后，我想对正交的概念做一些进一步（有点杂项）的观察。首先，正交设计的总体目标，与规范化类似，是减少冗余，从而避免否则可能会出现特定的更新异常。事实上，不严格地说，规范化减少关系变量内部的冗余，而正交减少关系变量之间的冗余，在这个意义上，正交是对规范化的补充。

更重要的是，正交也以另一种方式补充了规范化。再次考虑如图14-1所示的把关系变量S分解为其投影SNC和STC的（坏）分解。正如我们前面所看到的，该分解遵循所有通常的规范化原

则；换句话说，是正交而不是规范化，告诉我们这个设计是坏的。

我的下一个观点是，类似规范化的原则，正交设计原则基本上只是常识，但（再次类似规范化）它是形式化的常识，而且，我在第1章中关于这种形式化的言论在这里也适用。正如我在那一章中说的：

设计理论所做的是[形式化]某些常见的原则，从而打开机械化这些原则（即，将其纳入到计算机化的设计工具中）的可能性的的大门。理论的批评者往往忽视了这一点，不错，他们声称这些想法大多只是常识，但他们似乎并没有意识到以一个精确的和正式的方法说出常识的含义是一个重大的成就。

我的最后一点是这样的：假设我们从通常的零件关系变量 $P$ 开始，但出于设计的目的决定把关系变量分解成一组限制，例如轻与重的零件。那么正交性原则告诉我们，所涉及的限制应该是两两不相交的（当然，它们的并集（这其实是一个不相交的并集），也应该把我们带回到原关系变量）。注意：在以前的著作中，我把满足这一要求的分解称为正交分解。不过，我现在认为概括这个术语，并用它表示遵守正交性原则的任何

分解将是更好的。修订后的定义把较早的定义作为一个特例包括了。

## 习题

14.1 试述正交设计原则的最终版本，而不要回头查看本章正文。

14.2 考虑你碰巧熟悉的任何数据库的设计。它是否涉及对正交设计原则的任何违反呢？是否有任何的约束（尤其是“重叠”的）应该声明，但一直没声明呢？

14.3 考虑14.8节中的第二个例子（一个涉及关系变量EARN\$、SALARY\_UNK和UNSALARIED的例子）。你觉得在这个例子中所示的设计是无冗余的吗？

14.4 假设我们用一组关系变量LS、PS、AS……（为每个不同的供应商城市设置一个关系变量，例如，LS关系变量包含在伦敦的供应商元组），取代供应商关系变量S。这些关系变量都具有相同的属性，即，SNO、SNAME和STATUS（没有必要保留CITY属性，因为如果我们这样做，在每个关系变量中，它的值将是恒定的）。这样的设计是否违反正交性？你还能想到

它的任何其他问题吗？

顺便说一下，如果我们确实在关系变量LS、PS、AS等中保留属性属性，这个设计实际上违反规范化的原则！为什么违反，到底违反的是什么原则呢？

[1]也许更好的表达是，单例集合更新更可以接受。

[2]这里实际上可能有坏的东西。考虑，例如，如果零件P1的重量加倍，会发生什么情况。

[3]“反之亦然”是很重要的。考虑一个修订版的供应商和零件数据库，其中（a）从关系变量SP删除属性QTY，且（b）添加另一个关系变量SAP，它具有标题{SNO,PNO}和谓词供应商SNO能够提供零件PNO。那么可能有一个有效的约束，它的内容是，只有当一个给定的元组SP也出现在SAP中时，它才可以在SP中出现，且这样一个合理的状况不（且显然不应该）违反正交性。

[4]在这一章中，不像前面的章节中，事实上，标题概念包括相关的属性类型有时是重要的，因此，如果存在任何差别，术语标题必须（重新）进行相应的解释。举例来说，标题{PNO

CHAR,WEIGHT RATIONAL}和{PNO CHAR,WEIGHT INTEGER}，虽然它们涉及相同的属性名称，但它们是不一样的标题，正是因为

这两个WEIGHT属性是不同的类型。所以说，为简单起见，我将在整个本章的其余部分继续尽我所能地忽略属性的类型。

[5]有一位审校者极力主张，这个“诱惑”根本不诱人！也许确实如此，但我仍然认为这是值得讨论的。

[6]这个陈述也太过于简单。一个稍微好一点的陈述是：如果同样的子元组的不同出现代表的是同一个命题，我们不希望它出现不止一次——但这种说法也是不完美的。然而，要尝试使之更精确仍然需要具备比我目前想介绍的更多知识。请参阅第15章获得进一步的解释。

[7]因此，这个例子表明了一个明显的经验法则是：当你开始设计过程时（就我而言这意味着，当你写下谓词和其他业务规则时），总是对相同的属性使用相同的名称；不要通过使用，例如，SNO和SNR都指供应商编号，QTY和AMT都指数目等来“玩文字游戏”。按照这一原则，将（无论如何）使两个不同的元组不太可能代表同一个命题。

[8]请注意，根据JD不可约性的定义（见第11章），这个JD肯定在R1中存在。

[9]这个例子说明了在关系设计中处理“缺失信息”的一个推荐的方法（在《SQL与关系数据库理论》中详细讨论）。

## 第五部分 冗余

在本书中，我们一直在关注在设计中避免冗余。但是，到底什么是冗余呢？



## 第15章 我们需要更多的科学

What I tell you three times is true.

——Lewis Carroll: The Hunting of the Snark

注意：本章的一部分曾以相当不同的形式，最早出现在我的书《Date on Database: Writings 2000-2006》（Apress, 2006）中。

冗余 形容词。不需要的，弥漫性，过分的，额外的，无关紧要的，过度的，转弯抹角，冗言的，冗长，重复，分外，超编，多余的，过剩的，同义反复的，多余的，不必要的，多余的，冗长，啰唆。

我在这里给出上述壮观的同义词列表，纯粹是为了它可能有的内在价值（我在《钱伯斯20世纪辞典》发现它（Chambers Twentieth Century Thesaurus），这本书也给出了以下反义词的精彩列表：简洁、基本的、必要的。即使如此，我们已经看到，一般的设计理论可以视为一套减少冗余的原则和技术，（从而减少否则可能会出现潜在的某些不一致和更新异常）。但究竟什么是冗余？我们似乎没有这个术语的一个非常准确的定义，我们只是有一个有点模糊的概念：至少如

果对它不加妥善管理，就可能会导致问题。<sup>[1]</sup>

为了使这个问题得到稍微好一点的处理，我们首先需要明确区分系统的逻辑层面和物理层面。显然，在这两个层面，设计的目标是不同的。在物理层面，冗余几乎肯定会以一些形态或形式存在。这里有两个原因：

- 索引等“快速访问路径”结构必然需要一定的冗余度，因为某些数据值都将存储在这些辅助结构和为数据值提供“快速访问”的结构中。

- 以某种方式物理存储的派生关系变量（和/或派生关系）（称为快照或汇总表或物化查询或物化视图）<sup>[2]</sup>显然也包括一些冗余。

当然，在物理层面上冗余的原因是性能。但物理层面上的冗余对逻辑层面（或应该是这样）没有影响，它是由DBMS托管的，并且是从来不被用户直接看到的。我在这里提到它实际上只是为了排除它。从此刻开始，我就只关心逻辑层面上的冗余。

那么，在逻辑层面上，很容易脱口而出，冗余总是不好的。不过，当然这种说法是过于简单化的，因为如果没有别的办法，还可以使用视图

的机制。让我离题片刻来阐述后面这一点。这是众所周知的，但无论如何值得明确说明，视图（与规范化类似，虽然原因各不相同）服务于下面两个相当不同的目的。

1.实际定义视图V的用户，显然知道定义V的表达式X的形式。该用户可以在任何希望使用表达式X的地方使用名称V，但这些用途基本上只是速记（如使用一种编程语言中的宏）。

2.相比之下，仅仅获悉视图V存在和可以使用的用户应该是不知道表达式X的，事实上，对于该用户，V应该看上去和感觉就像一个基本关系变量。[\[3\]](#)

举一个第1种情况的例子，假设用户意识到数据库包含两个关系变量R1和R2，并把它们的连接定义为一个视图；那么，对该用户来说，显然这个视图是多余的，它可以去掉，并且不丢失任何信息。因此，为了确定性，我要从此刻开始假设（除非有明确的相反陈述）数据库中的任何关系变量都不是根据其他任何关系变量定义的，因此，至少这种特殊的冗余目前是不常见的。排除了这种可能性，那么，很容易诱使人树立一个目标并再次说逻辑层面的冗余始终是不可取的。但是，为了采取这样的立场，我们需要能够说出我

们认为的冗余的含义，否则这个立场不可能有意义。即使我们能给出该术语的一个很好的定义，这个立场是真正站得住脚的（即逻辑层面的冗余总是不好的）吗？有可能消除所有冗余吗？甚至这么做是可取的吗？

当然，这些问题具有相当的务实重要性。事实上，我认为这里值得注意的是，Codd把他的第一篇（1969年）关于关系模型的论文命名为“**Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks**”（添加了粗体；见附录C）。而通常认为他的第二篇（1970年）论文“A Relation Model of Data for Large shared Data Banks”（再次见附录C）是该领域的开创性论文（尽管这种描述对其1969年的前身有点不公平），它包括长度几乎相等的两个部分，其中第二个部分称为“冗余性和一致性”（第一个部分称为“关系模型和范式”）。因此，Codd显然把他在冗余上的想法当作他在关系的研究工作上的贡献的重要分量：在我看来，这是正确的，因为他确实至少为我们提供了一个框架，利用此框架我们就可以精确地和系统地解决这个问题。

那么，我在前一章曾展示了一个不适用于这一章的冗余的“定义”，即当且仅当某个数据库包含相同元组的两次不同的出现时，该数据库包含

冗余。但是，我们可以正确地给出以下定义。

■定义：当且仅当某个数据库直接或间接地包含相同命题的两个不同的表示方式时，该数据库包含冗余。

麻烦的是，虽然这个定义显然是正确的，但它对于减少冗余的实际问题并没有多大帮助。但它至少蕴含着以下这个更好一点的定义。

■定义（初步版）：设 $D$ 是一个数据库设计，设 $DB$ 是一个符合 $D$ 的数据库的值（即在 $D$ 中提到的关系变量值的集合），并设 $p$ 是一个命题。此外，设 $DB$ 包含（显式或隐式地）表示 $p$ 的某个元组，或元组的某种组合的某次具体的出现。如果 $DB$ 还包含也（显式或隐式地）表示 $p$ 的某个元组，或元组的组合某个不同的出现，那么 $DB$ 包含冗余且 $D$ 允许冗余。

规范化原则和正交设计原则的目标正是为了减少上述狭义的冗余。但是，请注意，这个定义说的全都是如果出现某些元组（而不是当且仅当），那么存在冗余，即它不是一个完备的定义。事实上，我们在本章中可以看到几个例子，即使它们不包含代表同一个命题的不同元组或元组组合，但它们仍包含冗余。更重要的是，所涉

及的大多数例子是完全规范化和完全正交的。换句话说，规范化和正交的原则，虽然是必要的，并无疑是重要的，但仍是远远不够的。

## 15.1 一点历史

在我开始讨论如何以及为什么规范化和正交性是不够的之前，我想略加讨论Codd在他的前两篇解决冗余问题的论文中的尝试。在他1969年的论文中，他曾这样说：

如果一个关系集合至少包含可从该集合中的其余关系派生的一个关系，那么它是强冗余的。

他在1970年的论文中对上述定义更加规范：

如果一个关系的集合至少包含具有可从该集合中的其余关系的投影派生的一个投影的一个关系，那么它是强冗余的。

我应该解释一下，Codd说的当一个关系 $r$ 可从一个关系集合 $S$ 派生，他的意思是 $r$ 等于将一系列的关系操作（连接、投影等）应用到来自 $S$ 的关系所得的结果。但是，我对他的定义有几点意见。

■首先，术语“关系”应整体更换为术语“关系变量”。（当然，直到多年后，才提出这个术语，并且Codd根本从来没有使用过它。）

■其次，我们可以忽略限定词“强”。Codd区分“强”冗余和他所谓的弱冗余，但弱冗余对我们来说是无关紧要的。其原因是，弱冗余所涉及的只是那些相等依赖，它们并非在所有时间存在，但由于关系值在所涉及的时刻恰好存在，而在特定的时间确实恰好存在。事实上，在我看来，Codd在这里正在竭力应对关系和关系变量之间的逻辑差异！见前面的项目符号项。“强”冗余适用于关系变量（这就是当我们谈论数据库设计时我们通常所说的冗余）。相比之下，“弱”冗余适用于关系，而不是关系变量（它只是在某些特定的时间恰好具有的关系变量值的一个人为现象）。

■把1969年的定义纳入1970年的定义，当然，因为（我们从第6章知道）每个关系变量 $R$ 都恒等于 $R$ 的一个特定的投影，即恒等投影。

■更重要的一点，1970年的定义作为一般的冗余定义仍是不够的，至少有以下两个原因。

a.它还包括一些可能性，我们通常根本不会把它们作为冗余。例如，假设供应商和零件数据

库服从每一个零件必须至少由一个供应商提供的约束。那么关系变量SP在{PNO}上的投影将一定等于关系变量P在{PNO}上的投影，并且我们将有一个“强冗余”。注意：也许用来说明相同的一点的更现实的例子，将是人事数据库上的一个约束，其大意是每一个部门必须至少有一名员工。

b.同时，它排除了我们肯定会认为是冗余的很多可能性，例如，第14章中的轻与重的零件的例子（第2版，如图14-3所示）。本章后面的各节给出一些深入的例子。

■更重要的是，在1970年定义中引用的投影应改为对应于不可约JD的分量的投影。（那么，前面项目符号项中的两个目标中的第一个就会消除。）

关于Codd的定义的最后一点：Codd确实至少在这两篇论文中说：“我们应当在数据库中把定义所有冗余的一个语句集合与[数据库]关联”。Codd这里说的“语句”指的是Tutorial D CONSTRAINT语句（当然，或逻辑上等同于此类语句的东西）。换句话说，Codd当然希望系统能够识别冗余，他希望这些冗余得到相应的管理。然而，遗憾的是，然后他接着说：



一个不一致的产生.....可以在内部记录，因此，如果它没有在一些合理的时间内得到纠正.....系统能够通知安保人员[原文]。另外，该系统可以[告知用户]这样或那样的关系，现在需要改变以恢复一致性.....理想的情况下，[不同的补救措施]应该是可能的.....针对关系的不同子集合。

注意：冗余，更确切地说，管理不善的冗余肯定会由引起“不一致”（或者，我宁愿把它们称为完整性违规），但并非所有的完整性违规都是冗余造成的，当然。更重要的一点，我相信数据库应该从不包含任何不一致之处，至少对用户而言应该如此，正如我在前面的章节所说的，你永远不能相信你从一个不一致的数据库得到的结果。换句话说，“纠正不一致”，需要在单个语句级（甚至不在事务级）立即进行。[\[4\]](#)参见15.15节。

[\[1\]](#)不过，我想提醒你，我们至少有那种可以通过投影来删除的冗余的一个确切定义（见第13章）。

[\[2\]](#)顺便提一下，最后一个术语是强烈不建议使用的，因为所涉及的结构不是一个视图。视图是虚拟的，而不是物化的（至少对关系模式而言），因此，物化视图是一个自相矛盾的术语。适当的

术语是快照。

[3]重点是“应该”，我在这里描述了一个理想的情况。我敢肯定，你知道当今的现实是相当混乱的。

[4]参见《SQL与数据库关系理论》对这个可能相当非正统的意见的辩护。让我以详细阐述的方式补充一点，这个立场表示要求系统支持多重赋值（multiple assignment），这是赋值的一种形式，它允许多个变量，特别是几个关系变量“同时”更新（也就是说，在一个语句的范围内）。

## 15.2 数据库设计是谓词设计

虽然关于谓词和约束，我在前面的章节中说了很多，但我并没有明确地指出这些概念之间的区别。所以，现在让我来弥补这方面的不足。首先，一个给定的关系变量 $R$ 的谓词（为了定性，有时更明确地说是关系变量的谓词）是 $R$ 预期的解释或意义。当然，关系变量 $R$ 的每一个用户应该（或假设）了解相应的谓词，然而，请注意，（至少在当今的实现中）谓词是用自然语言表示的，因此从本质上一定是有些非形式的。

因此，谓词是非形式的。与此相反，约束是形式的。从本质上讲，一个约束条件是一个布尔表达式，它用一些正式语言，如SQL或Tutorial D来表示，并通常包含对数据库中关系变量的引用，在所有时间的计算结果都必须为TRUE。设 $R$ 是关系变量。那么，很容易认为，所有无论直接或间接提到 $R$ 的约束的逻辑“与”（AND），都是关系变量 $R$ 的约束。因此，请注意，关系变量 $R$ 的谓词只能由用户理解，而关系变量 $R$ 的约束既能由用户“理解”又能由系统“理解”。事实上，关系变量 $R$ 的约束可视为系统对关系变量 $R$ 的谓词的近似。当然，理想的情况下，我们希望 $R$ 总是满足其谓词的，然而，我们所能期望的最好结果是，

它总是满足它的约束。[\[1\]](#)

假设现在有一个数据库，它应该是所谓的“感兴趣的微观世界”的语义的忠实代表，因此谓词和约束是与数据库的设计业务高度相关的。我们可以说，谓词是这些语义非正式的方式，而约束是正式的方式。因此，我看到的数据库设计的过程是这样的：

- 1.首先，我们尽量谨慎地确定关系变量的谓词（以及其他业务规则）。

- 2.然后，我们将这些谓词和规则映射到关系变量和约束中。

作为上述过程的结果，我们可以看到，考虑设计理论（规范化等）的另一种方式如下：这是一套帮助业务确定谓词（和因此导致的约束）的原则和技术。这一观点是本章以下大部分内容的基础。

顺便说一句，我注意到，上述讨论有助于解释了为什么我对E/R（“实体/关系”）的建模和类似的图形方法不太热衷。（你可能已经注意到了在前面的章节中完全没有E/R图和类似的东西！）E/R模型和诸如此类的方案的问题是，它

们与形式逻辑相比不够强大，太不够强大。特别是，它们没有包括类似对量词（EXISTS和FORALL）的足够支持的任何东西<sup>[2]</sup>，这是一个严重的疏漏，因为除了描述最简单的约束之外，都需要这样的支持或相当于这种支持的东西。<sup>[3]</sup>因此，这些方案和图完全不能代表除了少量固然重要的，但有限的约束之外的其余约束。因此，尽管在一个较高的抽象水平使用这种图表以阐明整体设计是可以接受的，但认为这样的图表实际上就是设计的整体的观点是误导性的，并在某些方面是相当危险的。恰恰相反：设计是图表确实显示的关系变量再加上它们没有显示的约束<sup>[4]</sup>。

<sup>[1]</sup>顺便说一句，我说的封闭世界假设适用于谓词，而不适用于约束。也就是说，（a）如果元组 $t$ 在时间 $T$ 出现在关系变量 $R$ 中，则 $t$ 当然在时间 $T$ 同时满足关系变量 $R$ 的谓词和该关系变量的约束；（b）如果元组 $t$ 貌似可以出现在关系变量 $R$ 中但实际上没有出现，则 $t$ 在时间 $T$ 肯定不会满足 $R$ 的谓词，但它仍然满足 $R$ 的约束（因为如果它不满足的话，那么它首先不可能“貌似可以出现”）。

<sup>[2]</sup>由于量词是弗雷格在1879年发明的，这种疏漏使得E/R图之类的工具（我的一个朋友曾经这样说是“1879年之前的那种逻辑”！注意：关于量

词和相关事项的教程可以在《SQL与数据库关系理论》和许多其他地方获得。

[3] Tutorial D也没有明确的量词支持，但无论如何仍然可以以Tutorial D表达量词形式的东西，即，Tutorial D至少有一些相当于量词支持的东西。

[4] 两个条件：首先，图表确实显示了一些约束（基本上是键和外键约束），正如已经指出的。其次，事实上它们可能无法表示所有的关系变量（一些E/R模型的方案在它们的图形中不包括与我们的例子中对应于多对多关系的SP相类似的关系变量）。

## 15.3 例1

这是我的声明：作为一个研究领域的设计理论，在一般情况下，仍然是敞开的。为了支持这一声明，在本节及未来几节我想举一些设计的例子，它们：（a）完全规范化且完全正交（至少在大多数情况下），但（b）仍然存在各种冗余（同样在大多数情况下）。

我的第一个例子，考虑下面这个简单的关系变量，它代表一个姓名和地址集合（谓词是人NAME居住在地址ADDR）：

---

---

```
NADDR{NAME,ADDR}
KEY{NAME}
```

---

---

假设在此关系变量中的属性ADDR的值是元组（tuple valued），所涉及的元组具有属性STREET、CITY、STATE和ZIP。（是的，元组值的属性或TVA是合法的，正如关系值的属性或RVA是合法的，并出于同样的理由，请参阅第4章。）这个关系变量的示例值如图15-1所示。

NADDR

NAME	ADDR /* tuple valued */			
Jack	STREET	CITY	STATE	ZIP
	1 Main St.	SFO	CA	94100
Jill	STREET	CITY	STATE	ZIP
	2 Main St.	SFO	CA	94100

图 15-1 关系变量NADDR（ADDR的值是元组）——示例值

现在为了这个例子的目的，如我们在习题6.2中所做的，假设每当两个ADDR值有相同的ZIP分量时，它们也有同样的CITY和STATE分量。上述设计显然包含一些冗余。然而，它没有违反规范化，尤其是函数依赖

---

$$\{\text{ZIP}\} \rightarrow \{\text{CITY}, \text{STATE}\}$$

---

不存在。（为什么不存在呢？答案：因为FD



定义为在属性间存在，而不是在属性的分量间存在。）

即便如此，我还想指出，将NADDR替换为下面表达式的返回值后，上述FD确实存在于结果中：

---

---

NADDR UNWRAP (ADDR)

---

---

注意：Tutorial D的UNWRAP（解包）操作符实际上把某个值为元组的属性替代为一组属性，其中TVA的每个分量对应一个属性。因此，上述表达式返回的结果具有属性NAME、STREET、CITY、STATE和ZIP。（当然，这个结果仍然只属于2NF，甚至不属于BCNF，它仍然存在冗余。）

我们可以从这个例子得出结论：解包TVA是一个好主意。但它是一个足够被奉为一个良好的设计原则的好主意吗？[\[1\]](#)

[\[1\]](#)在他的书《An Introduction to Relational Database Theory》（Ventus, 2010），休·达尔文（Hugh Darwen）表明，在这方面可能是，且我们可能会考虑一个打包-解包范式（wrap-unwrap

normal form)。在同一本书中，他还建议，取消分组RVA也是一个好主意，因此，我们可能也相应考虑分组-取消分组范式（group-ungroup normal form）。

## 15.4 例2

Codd可能会禁止例1的设计，理由是属性ADDR的值不是“原子”的（虽然我不知道他曾经在他的哪篇著作中明确地解决了如前所述的元组值属性的问题）。现在，我个人不同意这个立场，根据我已经在《SQL与关系数据库理论》和其他地方详细地解释了的原因，但这一点是不值得为之战斗的，因为我们可以明显地把该元组值的属性更换为如图15-2所示的一个类型为CHAR的属性。Codd肯定会允许修改后的设计，但它仍遭受完全类似于例1中的冗余。

NADDR

NAME	ADDR /* type CHAR */
Jack	1 Main St., SFO, CA 94100
Jill	2 Main St., SFO, CA 94100

图 15-2 经修订的关系变量NADDR（ADDR文本值）的示例值

如果你不喜欢这个例子，考虑如果属性ADDR是某个用户定义的类型（比如说ADDRESS），而不是CHAR类型会发生什么情况。



## 15.5 例3

如果DATE类型属性包括星期几和日历日期（如在，例如，“星期二，2011年1月18日”），那么关于这些属性可能出现类似例2中的那些冗余。

## 15.6 例4

下面的例子是我们熟悉的员工和程序员例子的一个非常简单的版本，其中所有的程序员都是员工，但有些员工不是程序员（如在习题5.7中）。注意，有些人会说，在这个例子中，员工和程序员分别对应于一个实体超类型（entity supertype）和一个实体子类型（entity subtype）。即使如此，这里是传统的设计：

---

```
EMP {ENO}
KEY {ENO}
PGMR {ENO,LANG}
KEY {ENO}
```

---

为了简单起见，我假设非程序员除了ENO外没有其他有意义的属性（如果他们有这样的属性，对于本例并没有显著差异），且程序员有一个额外的属性LANG（编程语言技能，例如，“Java”或“SQL”或“Tutorial D”）。现在，我们有一个选择：在EMP中记录所有员工，或只在EMP中记录非程序员。哪个更好呢？

■如果我们只在EMP中记录非程序员，员工成为或不再是一个程序员所涉及的处理是稍微非平凡的，在这两种情况下我们都要从一个关系变

量删除一个元组，并往另一个关系变量插入一个元组。我们还需要说明并执行下面的约束：

---

CONSTRAINT.....IS\_EMPTY (EMP JOIN PGMR) ;

---

另外，请注意，如果我们想要一些其他关系变量包含员工的引用，该方案的。通常情况下，引用是一个简单的外键，但如果员工横跨两个关系变量EMP和PGMR，它不能（至少，不能作为常规理解的外键）是一个简单的外键。这样的考虑的实质是，这种特殊的设计可能是不推荐的。

■但另一方面，如果我们在EMP中记录所有的员工，那么在我们的设计中有一些冗余：如果e是一个程序员，e肯定是员工，那么，为什么说得这么明确呢？

## 15.7 例5

现在，为了说明一个附加的观点，我想稍微扩展例4。假设关系变量EMP确实包括至少一个额外的属性JOB；进一步假定当且仅当在一个给定的员工的元组中JOB值在EMP中具有值“Programmer”（程序员）（对应于其他关系变量也许还有其他的JOB值，例如，“Janitor”（看门人））时，该员工是程序员，并在关系变量PGMR中表示。顺便提一下，这种情况在实践中并非绝无仅有的。现在一定有冗余，因为该设计服从如下相等依赖（以及许多类似的，可能）：

---

---

CONSTRAINT.....

(EMP WHERE JOB='Programmer')

{ENO}=PGMR{ENO};

---

---

然而，请注意，在这个例子中没有违反正交性，即使所有员工，包括程序员，都在EMP中表示。假设他们是，那么这显然是PGMR在{ENO}上的投影等于EMP在{ENO}上的投影的特定子集（它不是一个如前所述的限制）的情况。（习题：为什么它不是这样的限制呢？）但这些投影都没有和在相关关系变量中存在的任何不可约JD的一个分量相对应。[\[1\]](#)（如果此刻你需要刷新你



的记忆，请检查第14章中的最终版本的正交设计原则）。因此，一个数据库可以完全正交，但仍表现出一定的冗余度。

[1]实际上关系变量EMP与PGMR都属于6NF且在它们中存在的仅有的不可约JD都是平凡的。

## 15.8 例6

Codd在一篇写于1979年的论文“Extending the Database Relational Model to Capture More Meaning,”ACM TODS 4, No.4, December 1979, 中提出了一个特定的设计原则，可以描述如下（略作简化）。

■设E是一个“实体类型”，并设ID是一个数据类型，使得类型E的每个实体都正好有一个ID类型的主要标识符（我的术语，不是Codd的）。例如，E和ID可能分别是实体类型“供应商”和数据类型“字符串”。

■设 $P_1, \dots, P_n$ 是一组“属性类型”，使得类型E的每个实体至多有每个类型为 $P_1, \dots, P_n$ 的一个属性。例如，在供应商的案例中， $P_1$ 、 $P_2$ 、 $P_3$ 可能是属性类型“name”、“status”和“city”（“名称”、“状态”和“城市”）（所以在这个例子中， $n=3$ ）。注意：只是为了本次讨论，我假设供应商可以有三个属性的任何子集，特别是包括空集。

■那么，数据库应该包含：

a.正好一个E-关系变量，它包含在任何给定

时间存在的类型为E的这些实体的ID值。

b.对于每个 $P_i$  ( $i=1, \dots, n$ )，正好有一个P-关系变量，它包含在任何给定的时间存在并在那个时间有一个类型 $P_i$ 的属性的E类型实体的 (ID值,  $P_i$ 值) 对。

我将把这个原则称为“RM/T原则”，因为它的一部分 (Codd) 在他1979年的论文中，作为“扩展关系模型的RM/T” (T代表塔斯马尼亚岛，在那里Codd第一次提出了他的扩展模型的想法)。把这个原则具体地应用到供应商的案例中，我们得到的设计，大概 (这里为简单起见，我忽略了应该有一个从{CITY}到{STATUS}的FD的事实) 如下所示：

---

```
S{SNO}
KEY{SNO};
SN{SNO,SNAME}
KEY{SNO}
FOREIGN KEY{SNO}REFERENCES S
ST{SNO,STATUS}
KEY{SNO}
FOREIGN KEY{SNO}REFERENCES S
SC{SNO,CITY}
KEY{SNO}
FOREIGN KEY{SNO}REFERENCES S
```

---

这些关系变量都是不可约的，等价地说，每个关系变量都属于6NF，[\[1\]](#)图15-3给出了一组示例值。注意：所涉及的值并不意味着和我们常用的示例值相同，虽然它们非常接近。注意，特别是：（a）供应商S3没有状态，（b）供应商S4没有状态，也没有城市，及（c）供应商S5没有名字，没有状态，也没有城市。

S	SN	ST	SC
SNO	SNO SNAME	SNO STATUS	SNO CITY
S1	S1 Smith	S1 20	S1 London
S2	S2 Jones	S2 30	S2 Paris
S3	S3 Blake		S3 Paris
S4	S4 Clark		
S5			

图 15-3 “RM/T设计”供应商——示例值

正如第13章指出的，这样的设计其实有不少值得推荐的地方（至少，对一个架构良好的DBMS会如此）。然而，就目前而言，我想要做的所有事情是请你注意以下内容：只要每一个类型为E的实体至少有n个属性中的一个，那么设计肯定包含某种冗余（事实上，可以说是Codd自己

在他1970年的论文中定义的强冗余），因为，在任何给定的时间，E-关系变量的值将等于P-关系变量在标识符属性上的投影的并集：

---

```
CONSTRAINT.....S{SNO}=
UNION{SN{SNO}, ST{SNO}, SC{SNO}};
```

---

这种冗余适用于图15-3，例如，如果我们删除供应商S5（也就是说，如果每个供应商至少有三个属性（即名称、状态和城市）中的一个）。留给读者的习题：这里的冗余与在例4中讨论的冗余有区别吗？它们有什么区别？如果例4中的员工有额外的属性（例如薪金），会有什么差异吗？

进一步注意，在每一个E类型实体实际上都有所有的n个属性的（通常？）特殊情况下，这个设计实际上变得“甚至更冗余”。图15-4是图15-3修改后的版本，它说明了这种情况。请注意，在这个图中（很不严格地说），{SNO}现在是每个关系变量中的一个外键，它引用彼此的唯一键{SNO}，等价地说，任何一个关系变量在{SNO}上的投影都等于任何其他关系变量在{SNO}上的投影。更精确的问题，实际上有一个与这四个关系变量中的每对相关的相等依赖：

---

CONSTRAINT.....

IDENTICAL{S{SNO}, SN{SNO}, ST{SNO},

SC{SNO}};

---

注意：IDENTICAL（相同）是由休·达尔文和我作为Tutorial D的额外操作符提出的[\[2\]](#)，它的语义如下：如果关系 $r_1, \dots, r_n$ 全都相等，表达式

---

IDENTICAL{ $r_1, \dots, r_n$ }

---

返回TRUE；否则返回FALSE（你可以认为它是一种 $n$ 元“=”操作符）。

S	SN	ST	SC
SNO	SNO SNAME	SNO STATUS	SNO CITY
S1	S1 Smith	S1 20	S1 London
S2	S2 Jones	S2 30	S2 Paris
S3	S3 Blake	S3 30	S3 Paris
S4	S4 Clark	S4 20	S4 London
S5	S5 Adams	S5 30	S5 Athens

图 15-4 图15-3的修订版

然而，请注意，即使在这种极端的情况下，

该设计也不违反正交性。更重要的是，我再说一遍，假设有一个架构良好的DBMS，这种设计有不少值得推荐的地方。特别是，相等依赖及因此造成的冗余，将在这种系统中得到“自动”管理和维护（见15.15节）。

[1]在历史准确性的意义上，我注意到，Codd在他的RM/T论文所描述的P-关系变量并不一定属于6NF，因为他没有坚持每个P-关系变量只包含单个“属性”。

[2]在《Database Explorations: Essays on The Third Manifesto and Related Topics》（Trafford, 2010）和其他地方。

## 15.9 例7

考虑一家公司的每一位员工都需要正好属于一个部门，每个部门都至少需要有1名员工。图15-5显示了这种情况的RM/T设计的示例值（概要）：[\[1\]](#)

EMP		DEPT		EMPDEPT	
ENO		DNO		ENO	DNO
E1		D1		E1	D1
E2		D2		E2	D2
E3		D3		E3	D2
E4				E4	D3
E5				E5	D3

图 15-5 员工和部门——示例值

但是，关于这些示例值，我们看到正好有5名员工和三个部门。由于每个员工都必须正好属于一个部门，每一个部门必须至少有一名员工，为什么不定义一个部门（比如说D3），作为“默认”部门，并采用一条规则，即在EMP中提到并且没有在EMPDEPT提到的任何员工都属于默认的部门呢？在图15-4中，这个规则将允许我们从EMPDEPT中忽略元组（E4，D3）和（E5，



D3)。请注意，如果我们不采用这样的规则，那么设计显然再次包含某种冗余，具体地说，它服从下列相等依赖：

---

CONSTRAINT	EVERY_EMP_HAS_A_DEPT
EMP{ENO}=EMPDEPT{ENO};	
CONSTRAINT	EVERY_DEPT_HAS_AN_EMP
DEPT{DNO}=EMPDEPT{DNO};	

---

但是，在我看来至少有两个因素会妨碍采用这样的“默认部门”的设计。第一个因素是，作出的默认部门的选择很可能是任意的。第二个因素是，现在我们需要非常关注关系变量EMPDEPT的含义！明显的谓词“员工ENO属于部门DNO”无法正常工作。为什么会这样呢？因为，根据该谓词（和默认的部门是D3），省略的元组（E5，D3）（根据封闭世界假设）将意味着该员工E5不属于部门D3！因此，谓词是类似这样的：

员工ENO属于部门DNO（这不是默认的部门编号D3）。

现在，这个谓词能够工作（我认为），但它是相当棘手的。假设如图15-5所示的元组（E1，D1）出现在关系变量中，那么相应的命题是：

员工E1属于部门D1（这不是默认的部门编号D3）。

当然，这个命题的判断结果为TRUE。到此为止没有问题。但是，现在假设在此关系变量中没有员工E5的元组。当然，预期的解释是：员工E5属于部门D3，但封闭世界假设实际上说的是什么呢？首先注意，例如，具体的元组（E5，D1）没有出现。那么，根据封闭世界假设，下面的命题必定是真命题：

不是员工E5属于部门D1（这不是默认的部门编号D3）这样的情况。

或者更形式化一点：

---

---

$\text{NOT (E5 is in D1 AND D1} \neq \text{D3)}$

---

---

按照德·摩根律，这个表达式等价于：

---

---

$\text{E5 is not in D1 OR D1} = \text{D3}$

---

---

由于 $\text{D1} = \text{D3}$ 为假，因此表达式可以精简为只是“E5不属于D1”，这就是我们想要的（我的意思是，这是一个真命题）。

类似的分析表明，我们可以推断，E5肯定不属于不是默认部门D3的任何部门。但是，默认部门会怎么样呢？那么，因为元组（E5，D3）不出现，所以下面的命题必定是真命题：

---

NOT (E5 is in D3 AND D3≠D3)

---

等价的表达式是：

---

E5 is not in D3 OR D3=D3

---

由于D3=D3为真，因此这个表达式可以精简为只是TRUE。但是，请注意，这个命题实际上并没有告诉我们E5属于D3！现在，假设E5确实存在，且肯定不属于其他任何部门（？），也许我们可以推断出后一个事实。但我严重怀疑在实践中用户是否情愿必须处理这种令人费解的和逻辑变化无常的论证。

**[1]**习题：在那个图中的EMPDEPT是为员工、部门，或两者兼有的P-关系变量吗？证明你的答案！把这一点再推进一步：在这个例子中，RM/T设计可能不是最佳选项，因为在EMP和EMPDEPT之间一定有一个一一对应，并似乎没

有不把那两个关系变量合二为一的理由。

## 15.10 例8

考虑图15-6所示的设计（对来自前一章的图14-4略作修订，有点“像RM/T”的版本）：

EMP	EARN\$	SALARY_UNK	UNSALARIED															
<table><tr><th>ENO</th></tr><tr><td>E1</td></tr><tr><td>E2</td></tr><tr><td>E3</td></tr><tr><td>E4</td></tr></table>	ENO	E1	E2	E3	E4	<table><tr><th>ENO</th><th>SALARY</th></tr><tr><td>E1</td><td>85,000</td></tr><tr><td>E3</td><td>70,000</td></tr></table>	ENO	SALARY	E1	85,000	E3	70,000	<table><tr><th>ENO</th></tr><tr><td>E2</td></tr></table>	ENO	E2	<table><tr><th>ENO</th></tr><tr><td>E4</td></tr></table>	ENO	E4
ENO																		
E1																		
E2																		
E3																		
E4																		
ENO	SALARY																	
E1	85,000																	
E3	70,000																	
ENO																		
E2																		
ENO																		
E4																		

图 15-6 员工和薪金的“RM/T设计”——示例值

这些关系变量的谓词如下：

■EMP：员工ENO被公司所雇用。

■EARN\$：员工ENO有薪金SALARY。

■SALARY\_UNK：ENO员工有一份薪金，但我们不知道它是多少。

■UNSALARIED：员工ENO没有薪金。

我现在注意到，关系变量SALARY\_UNK或关系变量UNSALARIED之一是冗余的，在关系变量EMP而不在关系变量EARNNS中表示的任何员工必须正好在其他两个关系变量之一中表示，因此，我们可以删除，比如说，关系变量SALARY\_UNK而不会丢失任何信息。然而，似乎没有任何充分的理由选择SALARY\_UNK和UNSALARIED之一而让另一个被淘汰，而对称的考虑会主张赞成这两个都保留，并保持冗余（？）。

旁白：对称性通常是另一个良好的设计原则。引用波利亚：[\[1\]](#)“努力把对称的东西进行对称的处理，并且不要大肆破坏任何自然的对称。”但是，例8和其他类似它的例子（或许例7也是）显示对称和非冗余有时是相互冲突的目标。结束旁白。

[\[1\]](#)G.Polya: How To Solve It (2nd ed., Princeton University Press, 1971)。

## 15.11 例9

这个例子是由休·达尔文提供的。它是基于出现在英国的开放大学的现实生活中的一个情况，设我们有一个关系变量，它看起来像这样：

---

---

```
SCT{SNO,CNO,TNO}
KEY{SNO,CNO,TNO}
```

---

---

谓词是：导师TNO在课程CNO上辅导学生SNO。图15-7显示了此关系变量的一个示例值。冗余是显而易见的：例如，在图15-7中显示的示例值中，S1学生参加课程C1的事实，课程C1由导师T1辅导的事实，以及导师T1辅导学生S1的事实中都不止一次地表示了。[\[1\]](#)（请注意， $JD \bowtie \{\{SNO,CNO\}\{CNO \ TNO\}, \{TNO,SNO\}\}$ 在关系变量SCT中不存在）。

SCT	SNO	CNO	TNO
	S1	C1	T1
	S1	C1	T2
	S2	C1	T1
	S2	C1	T2
	S1	C2	T1
	S2	C2	T1

图 15-7 关系变量SCT——示例值

现在，我们可能会考虑在像这样的例子中减少冗余的策略之一是利用代理键（简称代理）[\[2\]](#)。例如，我们可能会引入一个属性X，比如说，其值作为（SNO,CNO）对的代理，如图15-8所示。（注意，在这个图中我已经把{X}作为关系变量XSC的主键了。然而，{SNO,CNO}组合当然仍然也是一个键）。



XSC	X	SNO	CNO
	x1	S1	C1
	x2	S2	C1
	x3	S1	C2
	x4	S2	C2

XT	X	TNO
	x1	T1
	x1	T2
	x2	T1
	x2	T2
	x3	T1
	x4	T1

图 15-8 为 (SNO,CNO) 组合使用代理

这种方法带来的一个难题是：我们决定为 (SNO,CNO) 组合，而不为 (CNO,TNO) 组合或 (TNO,SNO) 组合使用代理，这个决定的基础是什么？我们的选择是不对称的。此外，代理也不是没有自己的问题。下面是代理的一些问题

[3]：

■代理可能使更新更复杂（在本质上，用户必须自己做外键检查）。

■雪上加霜的是，该系统的外键检查，仍然有许多工作要做：（a）这些工作将永远不会失败，且（b）是纯粹的开销。

■查询和更新语句变得更长，编写更繁琐，更容易出错，更难以调试，更难以维护。

## ■更多完整性约束成为必要。

然而，就目前而言，真正的问题是：引入代理确实有助于减少冗余吗？我不想在这里尝试解决这个问题，我会在15.16节回过头来解决它。

[1]你可能不同意那些重复构成冗余。但是，如果你不同意，我请你现在保留反对意见，在本章后面部分我会更详细地研究这个例子。

[2]事实上，Codd在他关于所有实体类型的RM/T原则中提倡使用代理。在此建议中，他继续了Patrick Hall、John Owlett和Stephen Todd在他们的论文“Relations and Entities”中的开拓性工作，这个论文发表在：G.M.Nijssen (ed.)，Modelling in Data Base Management Systems (North-Holland/Elsevier Science, 1975)。

[3]这些问题在我的《Relational Database Writings 1989-1991》(Addison-Wesley, 1992)一书的论文“Composite Keys”中进行了阐述。

## 15.12 例10

为了减少如图15-7的例子中的冗余，我们可能会考虑的另一个策略是提出一些关系值属性或RVA。图15-9给出了一个例子。然而，这种方法的一个明显的问题（撇开所有RVA始终出现的常见问题）再次是不对称：我们决定为导师而不是学生或课程使用一个RVA的依据是什么？并且在任何情况下，这种策略确实能减少冗余吗？再次，我会在15.16节回过头来解决它。

SNO	CNO	TNOREL			
S1	C1	<table><tr><td>TNO</td></tr><tr><td>T1</td></tr><tr><td>T2</td></tr></table>	TNO	T1	T2
TNO					
T1					
T2					
S2	C1	<table><tr><td>TNO</td></tr><tr><td>T1</td></tr><tr><td>T2</td></tr></table>	TNO	T1	T2
TNO					
T1					
T2					

S1	C2	<table><tr><td>TNO</td></tr><tr><td>T1</td></tr></table>	TNO	T1
TNO				
T1				
S2	C2	<table><tr><td>TNO</td></tr><tr><td>T1</td></tr></table>	TNO	T1
TNO				
T1				

图 15-9 为导师使用一个RVA

## 15.13 例11

这个例子只是一个位置标记。在我们的书《Temporal Data and the Relational Model》（Morgan Kaufmann, 2003）中，休·达尔文、尼科斯（Nikos Lorentzos）和我展示了时态数据中可能出现的一些更深层次种类的冗余，我们为处理这些问题提出了一些新的设计原则和技术。特别是，这本书是新范式6NF（第13章讨论）的源头，但重要的是要明白，6NF在时态上下文中有有一个扩展的定义（并且“更相关”）。

## 15.14 例12

我最后举一个关于常见的实际情况的典型例子。它宽泛地以Fabian Pascal的书《Practical Issues in Database Management: A Reference for the Thinking Practitioner》(Addison-Wesley, 2000)中的一个例子为基础。我们给出如下所示的两个关系变量(且直至另行通知为止,专门假设它们是基本关系变量):

---

```
PAYMENTS{CUSTNO,DATE,AMOUNT}
KEY{CUSTNO,DATE}
FOREIGN KEY{CUSTNO}REFERENCES TOTALS
TOTALS{CUSTNO,TOTAL}
KEY{CUSTNO}
```

---

关系变量TOTALS中的属性TOTAL是通常称为派生数据的一个例子,因为对于任何给定的客户它的值是由汇总所涉及的客户的所有付款得出的。事实上,以下相等依赖存在:

---

```
CONSTRAINT C12 TOTALS=SUMMARIZE PAYMENTS
BY{CUSTNO}:
{TOTAL: =SUM (AMOUNT) };
```

---

注意: SUMMARIZE是Tutorial D对SQL的带

有一个GROUP BY的SELECT语句的模拟（很不严格地说！）。[\[1\]](#)如果你觉得使用SQL比Tutorial D更舒适，请让我也给出上述约束的一个SQL版本：

---

```
CREATE ASSERTION C12 CHECK
  (NOT EXISTS
    (SELECT*
     FROM TOTALS
     WHERE NOT EXISTS
       (SELECT*
        FROM (SELECT CUSTNO,SUM (AMT) AS TOTAL
              FROM PAYMENTS
              GROUP BY CUSTNO) AS TEMP
        WHERE TOTALS.CUSTNO=TEMP.CUSTNO) )
    AND
    NOT EXISTS
      (SELECT*
       FROM (SELECT CUSTNO,SUM (AMT) AS TOTAL
              FROM PAYMENTS
              GROUP BY CUSTNO) AS TEMP
       WHERE NOT EXISTS
         (SELECT*
          FROM TOTALS
          WHERE TOTALS.CUSTNO=TEMP.CUSTNO) ) ) ;
```

---

关于SUMMARIZE的进一步说明，请参阅《SQL与关系数据库理论》。

那么，派生数据显然是冗余的，但要再次注

意，这里没有违反规范化或正交性（特别是，关系变量PAYMENTS与TOTALS都属于6NF）。我将在后一节中更详细地分析这个例子。

[1]其实SUMMARIZE可能会从下一个版本的Tutorial D中删除，因为包含SUMMARIZE的表达式总可以用关系EXTEND运算符和所谓的映像关系（image relations）更“体面”地表达。例如，本例中的SUMMARIZE表达式，可以由下列表达式替代：  
EXTEND PAYMENTS{CUSTNO}:  
{TOTAL: =SUM (! PAYMENTS,AMOUNT)}。  
欲了解更多有关一般的EXTEND和专门的映像关系的信息，请参阅《SQL与关系数据库理论》。

## 15.15 管理冗余

特定的相等依赖存在（约束C12）的事实清楚地显示出一个事实，即上一节例12的设计是冗余的。至少在原则上，有四个基本的方法来处理由该实例显示出的这种冗余：

- 1.只用原始设计
- 2.声明约束
- 3.使用视图
- 4.使用快照

让我们来仔细看看。

### 1.只用原始设计

由于当今的大多数DBMS实现提供有限的功能，这也许是在实践中最有可能遇到的方法。我们的想法很简单，就是：

- a.正如上一节所示的那样对关系变量PAYMENTS与TOTALS进行定义。
- b.不在DBMS中声明约束C12。



c.维护派生数据百分之百是用户的责任。

（或者无论如何是一些用户的责任，维护可能由触发的过程执行，但仍然有某个用户必须写那个程序的代码。）[\[1\]](#)

实际上，此方法是额外工作和性能改进的折衷，额外工作源自一部分用户或至少某个用户在执行某些更新（以及相关关联的性能冲击），性能改进源自在执行某些查询时得到的性能改进。但它是无保证的，如果用户在执行一些更新过程中犯了（实际上）会导致违反约束C12的错误，那么，有麻烦了。

[\[1\]](#)特别注意，关系变量TOTALS实际上决不应该更新，除了那些为了保持两个关系变量“同步”所需要的更新。（事实上，通过必要的修正，类似的观察也适用于其他三种方法。）

## 2.声明约束

在这种方法中，约束C12是在DBMS中显式声明的，且DBMS负责执行它。但是，维护派生数据仍然是用户的责任，正如在前一种方法中的做法。更重要的是，如果用户可靠和正确地执行此任务，约束检查将永远不会失败，并且因此，它实际上会构成对用户的更新的纯开销。但是，我们不能免除该约束，正是因为我们需要系统检查用户正在可靠和正确地执行维护任务。

### 3.使用视图

如果我们实际上可以通知系统派生数据的定义规则，并让系统自动执行推导过程，显然，这 will 比简单地声明该约束更好。我们确实可以做到，这正是视图机制所做的。具体地说，我们可以用相同名称的视图（或“虚拟关系变量”）来替代基本关系变量TOTALS，因此：

---

```
VAR TOTALS VIRTUAL/*Tutorial D定义一个视图的语法*/  
  (SUMMARIZE PAYMENTS BY{CUSTNO}:  
   {TOTAL: =SUM (AMOUNT)});
```

---

现在，用户不再需要担心维护派生数据；而且，现在也不可能违反约束C12，甚至都不再需要说明它，除了可能是非正式地（也许作为告诉用户该视图的语义的一种方法）。但是，请注意，必须明确地告知用户不要设法维护总计值！但是，这一事实并不意味着，必须告知用户关系变量TOTALS是一个视图，它只是意味着必须告知用户，系统将有效地执行此维护任务。

## 4.使用快照

但是，视图解决方案的缺点是，推导过程是每次引用视图时进行的（即使在最后一次引用它后没有进行过更新）。因此，如果练习的目标是在更新时做推导工作，以便提高后续查询性能，视图的解决方案显然是不够的。在这种情况下，我们应该使用一个快照，而不是一个视图：

---

```
VAR TOTALS SNAPSHOT
(SUMMARIZE PAYMENTS BY {CUSTNO};
{TOTAL: =SUM (AMOUNT) })
REFRESH ON EVERY UPDATE;
```

---

快照概念源于Michel Adiba写的一篇文章。[\[1\]](#)基本上说，像视图一样，快照派生自关系变量；但是，不同于视图的是，它们是真实的，不是虚拟的，也就是说，它们不仅仅通过其依据其他关系变量下的定义表示，而且（至少在概念上）通过其自己数据的单独实体化副本（**materialized copy**）表示。换句话说，定义一个快照很像执行一个查询，不同的是：

a.查询结果作为一个只读关系变量（只读，也就是说，除了定期刷新——参见下面的b点）保存在数据库中指定的名称（在本例中是

TOTALS) 中。

b.快照是定期（在本例中是在每次更新时）刷新的，也就是说，它的当前值将废弃，重新执行查询，新的执行结果将成为新的快照值。

## REFRESH子句的一般形式是

---

REFRESH EVERY <now and then>

---

这里的<now and then>可能是，例如，MONTH或WEEK或DAY或HOUR或n MINUTES或MONDAY或WEEKDAY（依此类推）。特别是，规格说明REFRESH[ON]EVERY UPDATE表示快照与它派生自的一个或多个关系变量保持永久同步，在例12的情况下，这大概正是我们想要的。

现在，到目前为止，本节已经专门研究了例12和“派生数据”。然而，事实是，所有形式的冗余都可以当成派生数据：如果x是冗余的，那么根据定义x可以由在数据库中的其他东西推导得出。（把术语派生数据的使用限制到例12所示的那种情况因此是误导性的，并且是不推荐的。）因此上述的分析，特别是，处理派生数据的4种不同方法可以推广到适用于所有种类的冗余，至

少在原则上是可以的。请特别注意，上述分别使用视图和快照的第三个和第四个方法，两者都构成有时也称为受控的冗余的例子。冗余受控是指，如果它确实存在（且用户意识到这一点），但用以确保它不会导致任何不一致性的“蔓延的更新”任务是由系统而不是由用户管理的。不受控制的冗余可能是一个问题，但受控的冗余不应该是。事实上，我想更进一步，我想说虽然消除百分之百的冗余可能是不可能的，甚至也许是不值得的，但起码应该对那些没有消除的任何冗余加以控制。特别是，我们需要对快照的支持。

（幸运的是，许多商业产品现在支持快照，虽然它们用了不推荐使用的物化视图这个名称）。

[1] Michel Adiba: “Derived Relations: A Unified Mechanism for Views, Snapshots, and Distributed Data,” Proc. 1981 Int. Conf. on Very Large Data Bases, Cannes, France (September 1981)。也参见早期版本的“Database Snapshots”, by Michel E. Adiba and Bruce G. Lindsay, IBM Research Report RJ2772 (March 7th, 1980)。

## 15.16 改善定义

我故意把本节留到了本章的（几乎）最后。考虑发货关系变量SP，及其谓词供应商SNO供应数量为QTY的零件PNO。还要考虑第1章的图1-1中所示的关系变量值代表的关系。请注意：

a.在该关系中有两个元组分别是（S1， P5， 100）和（S1， P6， 100）。

b.这两个元组都包含（S1， 100）作为一个子元组。

那个子元组的两次出现意味着什么？那么，出现在（S1， P5， 100）中的子元组是指：

1.供应商S1提供数量为100的某个零件。

（为了未来的引用，我把这个命题编号，注意，这的确是一个命题。）而出现在（S1， P6， 100）中的子元组指的是完全一样的东西！所以我们这里的情况难道不是数据库中包含表示同样命题的某个元组的两次不同的出现吗？换句话说，按照本章导言部分给出的定义，数据库难道没有包含某种冗余吗？

在我尝试回答这个问题之前，我想对同一点提供一个简单的说明。再次参考图1-1，可以考虑该图中所示的供应商S1的6个发货（SP）元组。显然，那些元组都包含度为1的子元组（S1）。且那个子元组的6次出现都意味着同样的事情。

## 2. 供应商S1提供某个数量的某个零件。

我们甚至可以把这个说法更推进一步，并考虑，SP的每一个元组（其实，每一个可能的元组，不管它有什么属性），总是有0元组作为一个子元组的事实。因此，那个子元组在图1-1所示的发货关系中的12次“出现”（如果你明白我的意思！）都代表下列命题。

## 3. 某个供应商提供某个数量的某个零件。

那么现在我们的手中有冗余吗，或没有？让我观察，上述命题1~3都包含某个存在量词。下面是那些命题的稍微正式的版本：

1. 存在某个零件PNO，使得供应商S1提供数量为100的零件PNO。

2. 存在某个零件PNO，使得存在某个数量QTY，使得供应商S1提供数量为QTY的零件



PNO。

3.存在某个供应商SNO，使得存在某个零件PNO，使得存在某个数量QTY，使得供应商SNO提供数量为QTY的零件PNO。

在这些命题中，第三个命题中的SNO，在第二个和第三个命题中的QTY，以及所有三个命题中的PNO都不是参数，而是逻辑学家称为绑定变量的东西，因为事实上，它们都是被短语“存在某个.....使得”“存在性量化”的。注意：如果你不熟悉这些概念（即，绑定变量和存在性量词），那么你可以在《SQL与关系数据库理论》中找到一个教学处理。不过，我认为，即使你没有这些事情的任何先验知识，目前的讨论应该也是很容易理解的。

与上述情况相比，基础关系变量SP中的元组代表的命题不包括任何存在量词。这是因为它们都只是关系变量谓词（即，它们都只是通过对谓词的形参代入实参得到的）的实例化，并且，如下所示的谓词，反过来也不包含这样的量词：

供应商SNO提供数量为QTY的零件PNO。

要总结这一点，那么，以下意见看起来仿佛

适用。

a.所谓给定的命题（给定关系变量中的元组表示的命题）是无量词的。

旁白：说给定的命题总是无量词的可能不完全符合事实，例如，考虑一个属性为WEIGHT和HEIGHT的关系变量和谓词某人具有体重WEIGHT和身高HEIGHT。但是，在这样的情况下，我们可以通过一个称为斯科伦化（skolemization）（得名于逻辑学家T.A斯科伦）的过程有效地消除量词。在这个例子中，这个过程包括用人 $p$ 具有体重WEIGHT的和身高HEIGHT（其中 $p$ 是指某人或未知的某些人）的谓词替代原有的谓词。后面的谓词是无量词的。结束旁白。

b.相反，派生命题（至少，与对给定的关系变量中的元组进行投影所得的元组对应的派生命题）至少包括一个存在量词。

现在，我们肯定不希望说我们一贯的发货关系变量SP本质上是冗余的（不是吗？）。所以，看上去我们可能想说的是顺着以下思路的东西。

如果表示同一个命题两次，但这个命题是存

在性量化的，那么那个重复不计为冗余。

但是，等一下，供应商关系变量S以及它的FD{CITY}→{STATUS}怎么样？一个完全类似上述例子的论据似乎表明，例如，作为图1-1中描绘的供应商的两个元组中的一个子元组出现的元组（London，20），代表以下命题。

存在某个供应商SNO，使得存在着某个名字SNAME，使得供应商SNO命名为SNAME，具有状态20，并位于城市伦敦。

显然，这个命题是存在性量化的，但它表示了两次，而且我们确实想把这个重复计为冗余。（正如我们所知道的，关系变量S不属于BCNF）。那么，这是怎么回事呢？

像往常一样，我认为，通过仔细观察那个谓词就可以找到这个问题的答案。首先，回顾一下第13章，如果某个谓词不涉及任何连接词，那么它是简单谓词，如果它是用AND连接的一组其他谓词（就目前而言，我假设那些其他的谓词都是简单的），那么它是合取谓词。现在，关系变量SP的谓词是，供应商SNO提供数量为QTY的零件PNO，在上述意义上是简单的。与此相反，关系变量S的谓词是合取谓词，它可以分解成一组简

单谓词。我可以通过把谓词说明为以下略为生硬但其实更符合逻辑的形式，让后一个事实马上更明显：

供应商SNO名为SNAME且

供应商SNO位于城市CITY且

城市CITY具有状态STATUS。

那么，对于这个版本的谓词，以下内容应该是明确的，即：

a.关系变量S服从非平凡的、不可约的  $JD \bowtie \{SN, SC, CT\}$ ，其中名称SN、SC和CT分别表示标题  $\{SNO, SNAME\}$ 、 $\{SNO, CITY\}$  和  $\{CITY, STATUS\}$ 。（相比之下，关系变量SP仅有的JD都是平凡的，且SP属于6NF。当然，关系变量S，甚至不属于BCNF）。

b.因此，关系变量S可以按照JD无损地分解。相应的投影的谓词如下：

SN： 供应商SNO名为SNAME。

SC： 供应商SNO位于城市CITY。

CT：城市CITY具有状态STATUS。

因为这些谓词不是存在性量化的，所以相应的命题也不是存在性量化的。[\[1\]](#)

c.关系变量S肯定包含对应于SN、SC和CT的子元组，但那些对应于SN和SC的子元组永远不会重复，因为{SNO}是一个键。相反，那些对应于CT的子元组是重复的，至少有可能（如我们从图1-1知道的），这种重复并不构成冗余。

那么，受到所有上述内容的启发，我提供了以下作为数据库呈现冗余的含义的一个公认的“最终”定义。

■定义（“最终”版本）：设D是一个数据库设计，设DB是符合D的一个数据库值（即，D中提到的关系变量值的一个集合），并设p是一个不包含任何存在性量化的命题。如果DB包含p的两个或多个不同的表示（显式（译者注：原文写错）或隐式），则DB包含冗余，且D允许冗余。

请特别注意，根据这个定义，即使一个数据库完全符合正交设计的原则和所有的规范化原则，它仍然可以呈现冗余。但是，请注意，这个定义仍在指出，如果（而不是当且仅当）有一定

的条件成立，则有冗余，我想把“如果”替换为“当且仅当”，但在这里我对我的信念还没有足够的勇气。目前还没有。

即使如此，让我们来考虑来自本章前几节的例1～例12，并观察，特别是它对于那些例子存在什么影响。请仔细注意：为了简单起见，在下面的整个分析中，我用的都是不带限定词的命题，来表示一个没有存在性量化的命题的意思。

### 例1～例2

这两个例子都显示冗余，因为命题“城市SFO和州CA是邮编94100代表的城市和州”表示了两次。

### 例3

假设两个不同的元组都包含DATE值“星期二，2011年1月18日”，那么数据库清楚地显示冗余，因为命题2011年1月18日是星期二明确表示了两次。事实上，即使DATE值只出现一次，也有冗余！原因是，即使在这种情况下，命题2011年1月18日是星期二，无论是显式或隐式，都表示那个值的一部分，星期几（在本例中是星期二），可以根据算法由其余的值（在本例中是

2011年1月18日) 确定。

#### 例4

设员工E1既在关系变量EMP中表示, 又在关系变量PGMR中表示。相应的两个命题是E1是一个员工和E1是一个程序员。前一个命题是后一个命题与所有的程序员都是员工的命题组合的一个合乎逻辑的结果。(请注意, 实际上, 后一个命题是由在PGMR中的{ENO}是引用EMP{ENO}的一个外键这个事实表示的)。因此, 命题E1是一个员工表示了两次, 一次是显式的, 另一次是隐式的。

#### 例5

设员工E1在关系变量EMP中表示, 并设E1的JOB值为“Programmer”, 即“程序员”(所以员工E1也在关系变量PGMR中表示)。那么, 命题E1是一个程序员显然是用两种不同的方式明确表示的。

#### 例6

对于确实至少具有三个属性(名称、状态和城市)之一的供应商, 这个例子在细节上作必要

修改后，在本质上与例4是相同的。（对于一个没有这些属性的供应商，没有冗余）。

### 例7

命题员工E5属于部门D3既由EMPDEPT中的一个元组显式表示，又由（a）和（b）二者的组合隐式地表示，其中，（a）表示命题每一个员工都正好属于一个部门（这实际上是由有关的外键定义表示的），（b）表示在EMPDEPT中缺乏代表命题员工E5属于部门D1和命题员工E5属于部门D2的元组。

### 例8

命题员工E4没有薪金既由UNSALARIED中的一个元组显式表示和（b）二者的组合隐式地表示，其中，（a）表示命题每一位员工有一个已知的工资或未知的工资，或没有薪金（这应该再次由一个特定的已声明的完整性约束表示），（b）表示员工E4在EARNNS或SALARY\_UNK中的任意一个中缺乏一个元组。

### 例9

现在，这是一个有意义的例子。此前，我说



过下面的话：

这一冗余.....是显而易见的：例如，学生S1参加课程C1的事实，课程C1由导师T1辅导的事实，以及导师T1辅导学生S1的事实都在[图15-7]所示的示例值中不止一次地表示。

我也说过谓词是导师TNO在课程CNO上辅导学生SNO。但是，如果真正存在所说明的冗余，谓词不能那么简单。相反，它必须是类似这样的：

学生SNO参加课程CNO且

课程CNO由导师TNO辅导且

导师TNO辅导学生SNO且

导师TNO在课程CNO上辅导学生SNO。

因而一个更完整的设计包含的关系变量如下：

■S{SNO, .....}、C{CNO, .....}和  
T{TNO, .....}分别代表学生、课程、导师；

■SC{SNO,CNO, .....}、CT{CNO,TNO,

.....}和TS{TNO,SNO, .....}分别显示哪些学生都参加哪些课程、哪些课程由哪些导师辅导, 以及哪些导师辅导哪些学生;

■SCT{SNO,CNO,TNO}与原有版本的例子相同。

现在请注意:

1.关系变量SC等于S{SNO}与C{CNO}的连接(实际上是笛卡儿积)的某个子集(且对于CT和TS是类似的)。

2.关系变量SC也等于SCT在{SNO,CNO}上的投影(且再次对于CT和TS是类似的)。注意:为了更精确地说明这种情况, 当且仅当任何学生都不可以在没有为该课程分配导师的情况下参加某门课程时, SC等于SCT在{SNO,CNO}上的投影(且再次对于CT和TS是类似的)。因此, 我们正在这里做一些语义假设, 这些假设原本没有阐明且可能有效也可能无效。

3.关系变量SCT也等于SC、CT、TS的连接的某个子集, 并且那个连接反过来是C{CNO}、S{SNO}和T{TNO}的连接(实际上是笛卡儿积)的某个子集。

尤其是因为第2点，SC、CT、TS可以去掉，仿佛它们确实是在原来版本的例子中的。但是，假设现在不删除它们。那么就有明显的冗余。例如，根据图15-7的示例值，命题学生S1参加课程C1，既由（a）关系变量SC中的一个明确的元组表示，又表示为（b）如下命题的组合之一，该命题由关系变量SCT中的一个明确的元组所表示：

学生S1参加课程C1且

课程C1由导师T1辅导且

导师T1辅导学生S1且

导师T1在课程C1上辅导学生S1。

即使删除关系变量SC，我还是声称有冗余！例如，同一命题学生S1参加课程C1既表示为上述复合命题之一，又表示为如下（也是复合的）命题的组合之一：

学生S1参加课程C1且

课程C1由导师T2辅导且

导师T2辅导学生S1且

导师T2在课程C1上辅导学生S1。

这两个复合命题都由关系变量SCT中的明确的元组表示。因此，虽然我以前对这个例子中的冗余的描述也许有些误导，但我声明，无论如何冗余确实存在。更重要的是，如果你同意这个立场，我想你一定也同意，使用代理（见本章前面对当前例子（例9）的讨论）或关系值属性（见本章前面对例10的讨论）也没有有什么区别！也就是说，即使有代理和RVA，在一般情况下，仍然存在多次表示某些命题这种情况。现在，我承认，我的后一个声明可能有点争议。但是，如果你不同意它，那么我认为你需要相当仔细地证明你的立场，特别是我认为你需要对我所提出的冗余的“最终”定义给出一个替代定义（其实是一种改善）。

作为上述所有内容的一种附录，让我补充一点，我相信类似的分析适用于本书前面的其他一些例子。例如，考虑第9章和第12章的关系变量CTXD及其属性CNO、TNO、XNO和DAYS。当我第一次介绍这个例子中，我曾说过，谓词是教师TNO用教科书XNO花费DAYS天在课程CNO上。不过，这个谓词更准确的说法如下：

课程CNO可以由教师TNO讲授且

课程CNO使用教科书XNO且

教师TNO用教科书XNO花费DAYS天在课程CNO上。

同样，考虑来自第9章和第10章的关系变量SPJ及其属性SNO、PNO和JNO。当我第一次介绍这个例子时，我曾说过，谓词是供应商SNO供应零件PNO给工程JNO。不过，我认为这个谓词更准确的说法如下：

供应商SNO供应零件PNO且

零件PNO被提供给工程JNO且

工程JNO是由供应商SNO供应的且

供应商SNO供应零件PNO给工程JNO。

让我们对这最后一个例子继续研究片刻：我们从第10章知道，SPJ不属于5NF，并因此建议把它分解成分别具有标题{SNO,PNO}、{PNO,JNO}和{JNO,SNO}的“投影”关系变量SP、PJ和JS。但这三个关系变量的谓词分别是什么呢？答案取决于我在第9章（一个脚注中）提到的这个情况的完整语义。例如，考虑关系变量SP。如果一个元组(s,p)可以出现在SP中，而不需要有供应商s

的一个元组出现在JS或零件p的一个元组出现在PJ中，那么SP的谓词就只是供应商SNO供应零件PNO。但是，如果元组(s,p)在SP中的出现意味着，必须既有供应商s的一个元组在JS中又有零件p的一个元组在PJ中，那么SP的谓词是供应商SNO供应零件PNO给某个（未指定的）工程JNO。由于第二种可能性比第一种有更多的限制，在我看来，假设第一种解释将是审慎的做法。

### 例10

请参见上述例9的讨论。

### 例11

没有提供进一步讨论。

### 例12

设C1是一个客户，并设客户C1支付的总和（比如说）是\$10 000。那么这个命题——客户C1的付款总和是\$10 000——既由关系变量TOTALS中的客户C1的一个元组的出现显式表示，又由关系变量PAYMENTS中同一个客户的元组集合的出现隐式表示。

[1] 尤其是有关CT的谓词缺乏量词的内容，你可能想要再看看3.1节。

## 15.17 结束语

鉴于上一节的讨论，让我们同意为简单起见，我们感兴趣的仅有的命题是那些非存在性量化的。那么，我在本章中声称，如果一个数据库包含同一个命题的两个不同的表示方式，那么它肯定包含冗余。特别是，我们不希望同样的元组出现在两个不同的地方，如果这两次出现代表相同的命题。（很显然，我们想禁止这样的重复命题，然而，遗憾的是，DBMS不理解这样的命题。）但相同的元组出现两次是可以的，如果这两次出现并不代表同一个命题——并且无论如何，即使根本没有任何元组出现两次，我们也会有冗余，正如我们已经看到的那样。

目前，规范化和正交似乎是我们解决冗余问题的所有科学的攻略。遗憾的是，我们已经看到了，规范化和正交不能解决整个问题，在一般情况下，它们可以减少冗余，但它们不能完全消除它。具体而言，我们已经看到了几个完全符合规范化和正交的原则也呈现一些冗余的设计例子，并且这种讨论必定远远不够详尽。我们需要更多的科学！（现在我已经告诉过你这点至少三次了，而我告诉你三次的东西是真实的。）

鉴于上述状况，似乎在大多数数据库中，肯



定会在冗余。如果确实是这样的话：

■它至少应该加以控制，在这个意义上，DBMS应负起责任，保证它不会导致不一致。

■如果它不能得到控制，那么至少应该声明适当的约束，并由系统强制执行，（再一次）以确保它不会导致不一致。

■如果它不能加以控制且约束不能由系统执行（或也许甚至不能正式地声明），那么你能靠自己，如果你犯任何错误，你就会有麻烦了。

可悲的是，根据当今的商业化实现的状态，最后一种情况是最有可能在实践中碰到的。

## 习题

15.1 我在本章正文中声明，如果命题 $p$ 不包含存在量词并且数据库 $DB$ （显式或隐式地）包含 $p$ 的两个或两个以上不同的表示方式，那么 $DB$ 包含某种冗余。你能想到一个数据库（显式或隐式地）不包含任何此类命题的两个或两个以上不同的表示方式，但在你看来，仍显示一些冗余吗？

## 第六部分 附录

### 附录A 主键是良好的，但不是必需的

Life is rather like a tin of sardines——we're all of us looking for the key.

——Alan Bennett: *Beyond the Fringe*

注意：本附录的一部分最初曾以有所不同的形式出现在我的书《Relational Database Writings 1991-1994》（Addison-Wesley, 1995）中。

回顾来自第1章的这段文字：

我说过，通常要选择一个主键。事实上，这是通常的，但不是100%必要的。如果只有一个候选键，那么我们别无选择并且没有什么问题，但如果有两个或多个候选键，那么选择其中一个并把它作为主键，似乎有一点点随意性，至少对我来说是这样的。（当然在有的情况下，似乎没有任何充分的理由作出这样的选择，甚至有可能有很好的理由不这么做。附录A[即当前附录]详细阐述了有关事宜。）

这段摘录文字阐述的这种立场显然有点违背

传统的智慧，据说甚至可能会与广泛接受的一些关系模型戒律或一般的关系理论相抵触。具体来说：

■来自一个给定的关系变量所拥有的键（必须是非空的）集合，最初定义的关系模型把一种主要的作用归咎于那个集合中任意选择的一个称为主键的成员。

■关系的设计方法（虽然本身没有关系模型）倾向于建议，再次有点随意地，一个给定的“实体”应当在整个数据库中用相同的（主键）值确定，并在表示它的任何地方引用。

然而，正如所指出的，这些建议（有些人甚至会称它们为规定）都涉及一定程度的随意性。特别是第一条一直是造成关系的倡导者（包括我自己）有一些轻微的尴尬的源头。赞成关系模型最有力的论据之一是并且一直是其要求严密的逻辑基础。然而，尽管这种说法在大多数情况下显然是有道理的，主键和备用键之间的区别<sup>[1]</sup>

（即，从平等的集合中选择一个成员并使它在某种程度上“比别人更平等”的想法）似乎一直取决于那些得不到相同程度的理论重视的立场。当然对于这个区别也似乎没有任何正式的理由；它似乎更是一种教条，而不是逻辑，这就是为什么

（我说）我发现情况令人尴尬的原因。因为我自己对正统的关系立场在这些问题上似乎缺乏确凿的理由日益感到不满，所以我写了本附录。（如我的一个朋友曾经指出的，这些都是在现场演示中“你一带而过，希望没有人会注意到”的领域。）

更重要的是，对于主键与备用键的区别不仅似乎确实没有正式的理由，而且似乎也没有任何正式的方式可以作出这种选择。事实上，Codd在自己的记录上说：“[用来作出选择]的正常的的基础是简单的，但这一方面超越关系模型范围的”。[\[2\]](#)但是，为什么首先作出选择应该是必要的呢？即，为什么在这种情况下，一个真正的选择是存在的？是否有必要，或值得引入这样一个随意性元素呢？

此外，最初定义的关系模型坚持，在数据库中的任何地方，通过外键对给定的关系变量（元组）的所有引用必须始终专门通过关系变量的主键，从来不要通过某个备用键。因此，我们看到，首次决定基本上是任意的（从各个键中选择主键）会导致对随后作出的决定产生武断的限制，也就是说，它可能会以首次决定（即决定主键）时可能无法预见的方式，限制什么可以是而什么不能是一个合法的外键的决定。

因此，我声明，在如前所述的关系模型中，应对主键和备用键加以区别（以下简称为PK：AK的区别）的想法，为否则是一个正式地定义的（即，关系模型本身）系统引入了一个具有随意性、人为性、尴尬性和非对称性的不和谐音符。我进一步声明，它也可以导致引入一个具有随意性、人为性、尴尬性的和非对称性的不和谐程度到数据库中。我更进一步声明，它也可以导致一个不可取的和不必要的基础关系变量和派生关系变量的区别，如我将要显示的。

正因为如此，PK：AK区别真能是合理的吗？本附录提供了我认为有力的论据来支持这个问题的答案必须是“否”的立场。

## A.1 为PK：AK区别辩护的论据

在详细考虑PK：AK的区别的后果之前，我首先要研究为其辩护的论据。由于在历史上我自己是这种区别的一名捍卫者，[\[3\]](#)也许我应该从总结和反思我自己的观点开始！主要的几点，如下所示：

- 1.删除PK：AK的区别将意味着，除其他事项外，实体完整性规则将扩大到适用于所有候选键（至少是基础关系变量中的所有候选键）。

正如我希望你知道的，实体完整性规则的大意是基础关系变量的主键的组成属性不允许有空值。现在，我已经为此争论了很长一段时间，无论如何这条规则应该去掉，部分原因是因为它与空值（我断然拒绝的一个概念）有关，另一部分原因是因为它描绘了基础关系变量与其他关系变量之间的一个区别，从而违反了基础关系变量和视图的互换性原则。（如果你不熟悉这后一个原则，它基本上只是说，基础关系变量和视图之间不应该有任何不必要的区别，对用户来说，视图应该在“外观”上与基础关系变量是一样的）。因此，我现在发现第一个赞成PK：AK区别的论据是与它不相关的。

2.在每处引用的地方，都使用相同的符号来确定一个给定实体的原则使系统能够识别这些引用确实都指代同样事情的事实。

到目前为止，这个论点显然是有效的，但我现在觉得所称的这个原则应该被视为一个非正式的指导方针，而不是一个硬性的要求。请参阅本附录后面的讨论，特别是申请人及员工的例子，了解示例情况，在实践中不遵循这样的指导方针可能是可取的。在任何情况下，所涉及的指导方针真正与设计有关（换句话说，如何在一些特定的情况下应用关系模型），而不是与如前所述的

关系模型有关；特别是，因此，它与如前所述的关系模型是否应坚持主键没有任何关系。当我原本提出这种说法时，我一定有点困惑。

3.如果实体在不同的地方以不同的方式确定，“元数据查询”（*metaquery*，即，对目录的查询）可能更难以编写。例如，如果有时用员工编号而有时用社会保障号码指代员工，考虑编写元数据查询“哪些关系变量是指员工呢？”要涉及什么。

这里的想法基本上是根据上述第2点提到的原则对用户以及系统可能是有益的。但是，再次，在我看来，我们真正在谈论的是非正式的准则，而不是绝对的要求。

4.我的下一个观点并不完全赞成PK：AK之间的区别，而是对一个反对它的论据的批评。后一个论据如下：假设出于安全考虑，要防止某个用户看到某个主键，则该用户需要用某个备用键访问数据，那么为什么要首先区别PK：AK？

我至今还没有发现“后一种论据”非常有说服力，但当然批评反对某个立场的论据并不能证明相反的立场是正确的！



5.我的最后一个观点是诉诸“奥卡姆剃刀”（“不应该超过必要地扩大概念”）。实际上，我曾认为，平等地对待所有候选键对关系模型的元组级解决方案会造成不必要的复杂化。但人们也很可能认为（且现在我会认为）奥卡姆剃刀适用于其他相反的方式，且主键和备用键的概念是不必要的！即，我们真正需要的是候选键，或换句话说只是键。

概括地说，上述论据对我来说似乎不再非常令人信服，唯一一个似乎仍然具有一些说服力的是第2点和第3点的归纳，（如我说过的），无论如何它不是在如前所述的关系模型中区别PK:AK的一个真正的论据。我也说过，我现在觉得应该可以看到由特定的论据支持的立场更应该作为指导，而不是一个不可违反的规则（再次，见后面的例子来证明这个立场）。

虽然，我也注意到在我原来的论文中有些模棱两可.....这里是另一个摘录（我这里只是对它稍作改写）：

需要注意的是，如果我们现在同意保留PK:AK的区别，如果在未来的某个时间需要的话，总是有可能消除这种区别。而且，注意，这种说法反过来并不适用：一旦我们承诺将平等地对待

所有候选键，一个需要不同主键的系统将永远是非标准的。

虽然那时我并没有说太多话，但这个引文有效地构成谨慎设计的原则的要求，我确实仍然坚信的一个原则。<sup>[4]</sup>事实上，在我看来，我能够改变我对于PK：AK区别的立场（这确实是我正在做的）这个事实，可以视为对这一原则的平反。

在结束本节之前，我评论，Codd也在他说选择主键没有正式的基础的同一篇文章中，记载了他自己是坚持PK：AK的区别的一名捍卫者（这并不奇怪，因为正是他提出了这个区别）：

会出现严重的问题.....，如果允许任何关系变量有多个主键[原文].....允许多个主键所产生的后果.....对单个基础关系变量[将是]灾难性的。

（我已经随意地在这个摘录的片断中把术语“关系”用术语“关系变量”替换了两次）。他接着举了一个承担“几个不同的责任”（项目管理、部门管理、库存管理等）的员工的例子。然后，他说：

比较多个标识符的平等性.....目的是确定一

个员工和相同的员工与.....有关如果取决于选择哪对员工识别[属性]用于比较，员工标识符的类型可以不同，这个目标就会受到严重的打击。

我想你可以看到这种说法基本上与以上第2点和第3点是相同的，它（a）正如我已经指出的一样，有点令人困惑，且（b）如我们将在本附录后面看到的，无论如何并没有在密切关注下完全站得住脚。

[1]术语备用键（alternate key）在第1章中定义，但为方便起见，我在这里重复该定义：设关系变量R有两个或两个以上的键，并设其中之一被选为主键，那么其他的键是备用键。在实践中，这个术语用得不多，但在本附录中我确实需要使用它。

[2]这个引文来自Codd的论文“Domains,Keys,and Referential Integrity in Relational Databases”（InfoDB 3, No.1, Spring 1988）。

[3]在《Relational Database: Selected Writings》（Addison-Wesley, 1986）中的“Why Every Relation[sic]Should Have Exactly One Primary Key”；在《Relational Database Writings 1985-1989》（Addison-Wesley, 1990）中的“Referential Integrity and Foreign Keys”和其他地方。

[4]谨慎设计的原则指出：假定一个设计要在选项A和B之间做出选择，其中A是向上兼容B的且B的全面影响目前还不得而知，谨慎的决定是选择A。

## A.2 具有多个键的关系变量

现在，让我们考虑具有多个键的关系变量的一些切合实际的例子。首先考虑一个关系变量 EXAM，它具有属性 S（学生）、J（科目）和 P（名次），以及谓词学生 S 参加了科目 J 的考试并在班级的名单中获得名次 P。为了这个例子的目的，让我们假设没有并列名次（即，没有两个学生在同一科目获得相同的名次）。那么，显然，给定一个学生和科目，正好有一个相应的名次，同样，给定一个科目和名次，正好有一个相应的学生。因此， $FD\{S,J\} \rightarrow \{P\}$  和  $\{J,P\} \rightarrow \{S\}$  都存在，且  $\{S,J\}$  和  $\{J,P\}$  都是键（或候选键，如果你喜欢）：

---

```
EXAM{S,J,P}  
KEY{S,J}  
KEY{J,P}
```

---

下面是另一个例子（它基本上是第13章的习题13.6）：假设我们有一个表示婚姻的关系变量，它具有属性 A、B、C 和谓词人 A 在日期 C 与人 B 结婚。假设没有多配偶制，并假设也没有两个人彼此结婚不止一次，这里的每一对属性都是一个键：

MARRIAGE{A,B,C}  
KEY{A,B}  
KEY{B,C}  
KEY{C,A}

---

这里还有另一个基于一个简单的航空公司应用程序的例子（谓词是飞行员PILOT在日DAY小时HOUR从门GATE出发开始一次飞行）：

---

ROSTER{DAY,HOUR,GATE,PILOT}  
KEY{DAY,HOUR,GATE}  
KEY{DAY,HOUR,PILOT}

---

在这样的情况下我们如何选择主键？有什么理由选择一个键而不是另一个呢？Codd关于“简单”的标准似乎对此并没有帮助。还请注意，无论你怎么选择，我们都会因不愉快的不对称性而紧张。例如，在婚姻的例子中，我们可能会发现自己对配偶之一的处理“比另一个更平等”（从而肯定得罪某人）。我们为什么要被迫实行这种不对称性？不对称性通常不是一个好主意。在这里，重申来自第15章的波利亚的引言：“努力对称地处理对称的东西，且不要大肆破坏任何自然的对称性。”

现在，在上述的所有例子中的键不仅是复合

的，而且它们重叠（即，它们有一个共同的属性）。为了避免被认为只有当键重叠时，才有可能存在选择主键的困难，因此，让我举一个反例。假设我们有一个关系变量ELEMENT（元素）表示周期表（即化学元素表）。[\[1\]](#)那么每一种元素都有一个唯一的名称（如铅），一个唯一的符号（例如，铅的符号是Pb），以及一个唯一的原子序数（例如，铅的原子序数为82）。关系变量从而明确有三个不同的键，所有这三个键都是简单的（即，只涉及一个属性），并显然根本没有任何重叠。我们有什么理由在这三个键中选择一个作为主键呢？在我看来，一个很好的办法可以是视情况而定，把它们中的任何一个当作主键。

下面是另一个熟悉的（也许是太熟悉了）例子，其中包含一个有多个键的关系变量，所有这些键都是简单的：

---

```
TAX_BRACKET{LOW,HIGH,PERCENTAGE}  
KEY{LOW}  
KEY{HIGH}  
KEY{PERCENTAGE}
```

---

当然，我在这里假设，没有任何两个应税收入范围（从低到高）（LOW到HIGH）适用相同

的税率。

我可以举出更多的例子，但现在我的观点大概是清楚的：不仅没有选择一个键而不是另一个的正式标准（在有选择的情况下），而且有时也不会出现任何非正式标准。因此，坚持必须始终作出这样的选择似乎确实不恰当，即使在某些情况下（甚至可能是大多数情况下），这么做是恰当的。

还有另外一个要点需要提到，比我到目前为止已经说明的大多数观点更正式的一个观点。在过去40年左右的时间里，人们已经对依赖理论和进一步规范化、视图的更新、优化（特别是包括语义优化）、可用性和许多其他事项进行了大量的研究。且在所有的研究中，起到至关重要作用的是候选键，而不是主键。（事实上，这是必然的，正是因为所涉及的研究是正式的，正式定义候选键的概念，而没有正式定义主键的概念）。既然是这样的话，正式地坚持PK: AK的区别似乎确实是不恰当的，但是，重申一句，非正式地建议这种做法可能是恰当的。

然而我想提出的另一点是，PK: AK的区别导致基础关系变量和其他关系变量之间一个不可取和不必要的区别。这是因为，根据Codd的理



论，关系模型：

- 基础关系变量需要主键；

- 视图和快照允许但不需要它们；且

- 认为“完全没有必要（completely unnecessary）为任何其他关系变量（斜体字为原文）声明或演绎主键”。

这些陈述是意译（但只是一点点）自Codd写的论文，其中提到主键的选择没有正式的基础（见本附录前面部分）。事实上，该论文到目前为止表明，基础关系变量以外的其他关系变量甚至可能无法拥有一个主键，如果这一建议属实，这个概念首先肯定会引起严重的问题——请记住（基础关系变量和视图的）互换性原则。即便如此，在这些问题上，我的立场是截然不同的。具体而言，我会说：

- 首先，每一个关系变量，基础或派生的，确实至少有一个键（当然，因为没有关系曾经允许重复的元组，更不用说没有关系变量曾经允许重复的元组）。

- 其次，每一个基础关系变量必须至少有一

个显式声明的键。当然，所有这些键最好都应该显式声明。

■通常情况下，基础关系变量会专门有一个显式声明的主键，但我不坚持把这种状况作为一种硬性要求。

■由于《SQL与关系数据库理论》详细说明的原因，我相信系统应该能够从派生关系变量推导出键。

■尽管有先前的要点，我相信针对派生的关系变量（特别是视图和快照）也可以声明键。再次，请参阅《SQL与关系数据库理论》的进一步讨论。

[1]类似第13章中的PLUS的例子（请参阅），ELEMENT实际上可能是一个关系常量，而不是一个关系变量，但它仍然有键。

## A.3 发票和发货例子

现在我想把我的注意力转到一个更详细的例子上。这个例子（顺便说一句，它是在现实世界的一个应用程序的基础上产生的）涉及发票和发货，这两个实体类型之间有一个一对一的关系：每次发货都正好有一张发票，每一张发票都正好对应一次发货。在这里，然后是“明显的”数据库设计（为了这个例子的目的，我使用了一个明确区分主键和备用键的假设性的语法）：[\[1\]](#)

---

```
INVOICE{INVNO,SHIPNO,INV_DETAILS}  
PRIMARY KEY{INVNO}  
ALTERNATE KEY{SHIPNO}  
FOREIGN KEY{SHIPNO}REFERENCES SHIPMENT  
SHIPMENT{SHIPNO,INVNO,SHIP_DETAILS}  
PRIMARY KEY{SHIPNO}  
ALTERNATE KEY{INVNO}  
FOREIGN KEY{INVNO}REFERENCES INVOICE
```

---

因此，数据库的结构如图A-1所示（请注意，图A-1中的箭头不同于本书中其他图中的箭头，它代表外键引用，而不代表函数依赖）：

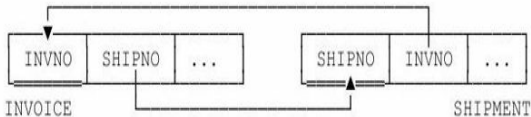


图 A-1 发票和发货数据库

现在，在这个例子中的每个关系变量实际上都有两个键： $\{INVNO\}$ 和 $\{SHIPNO\}$ 。但是，我假设为了论点我们可以同意，即INVOICE（发票）的“自然”的主键是 $\{INVNO\}$ ，而SHIPMENT（发货）的“自然”的主键是 $\{SHIPNO\}$ ，那么在INVOICE中的 $\{SHIPNO\}$ 和在SHIPMENT中的 $\{INVNO\}$ 都是备用键。此外，当然，这些备用键也都是引用其他关系变量的主键的外键（如图A-1所示）。

上述的设计存在的一个问题如下所示。显然，数据库服从如下约束，（实际上它是一个相等依赖，我把它叫做C），如果INVOICE关系变量显示发票i对应发货s，那么SHIPMENT关系变量必须显示与发票i相对应的发货s（反之亦然）：[\[2\]](#)

---

CONSTRAINT C

INVOICE $\{INVNO, SHIPNO\}$ =SHIPMENT $\{INVNO, SHIPNO\}$

换句话说，当且仅当元组  $(s,i, \dots)$  出现在SHIPMENT中时，元组  $(i,s, \dots)$  出现在INVOICE中。但图A-1的设计不捕获或强制执行这个约束（例如，图A-2所示的配置值是这个设计允许的，但违反了那个约束）。因此，这个约束需要单独地说明（如上述），并单独执行。

INVOICE			SHIPMENT		
INVNO	SHIPNO	...	SHIPNO	INVNO	...
i1	s1	...	s1	i2	
i2	s2	...	s2	i1	

图 A-2 违反约束C的“合法”的INVOICE和SHIPMENT值

旁白：有人可能会认为，如果我们假装每个关系变量的主键是组合  $\{INVNO, SHIPNO\}$ ，并且如果我们进一步将那些假冒的“主键”定义为引用另一个关系变量的主键的外键，那么将自动处理约束C。[\[3\]](#)但是，关系模型需要主键（事实上，一般的键）是不可约的，这意味着，它们必须不包含任何与唯一标识的目的不相关的属性（并且要求这么做也有很好的理由，正如我们从第4章

知道的)。换句话说，{INVNO,SHIPNO}其实不是每个关系变量的一个键（所以它肯定不能是主键），而如果我们告诉系统并非如此，那我们是在撒谎。事实上，如果{INVNO,SHIPNO}是真正的键，那么发票和发货之间的关系应该是多对多的，但它不是。结束旁白。

正因为约束C存在，图A-1的设计显然包含某种冗余：出现在任一关系变量中的每对{INVNO,SHIPNO}值也必须出现在另一个关系变量中。那么，我们能够通过把这两个关系变量组合为下面的一个关系变量来避免冗余：

---

---

```
INV_SHIP{INVNO,SHIPNO,INV_DETAILS,SHIP_DETAILS  
PRIMARY KEY{INVNO}  
ALTERNATE KEY{SHIPNO}
```

---

---

通过用这种方式消除冗余，我们也不再需要说明和强制约束C。此外，我们现在可以把原来的INVOICE和SHIPMENT关系变量定义为INV\_SHIP的视图，具体地说，投影视图，从而使用户仍把发票和发货作为不同的实体。<sup>[4]</sup>因此，这个修改后的设计确实比“明显的”版本具有一定的优势。

另一方面，修改后的设计也有一些缺点。观

察：首先，我们不得不再次做出一个不对称的决定，从{INVNO}和{SHIPNO}中任意选择前一个作为关系变量INV\_SHIP的主键。其次，进一步假设发货有一定的辅助信息而发票没有，例如，假设发货使用集装箱，每次发货都涉及几个集装箱。那么需要一个新的CONTAINER（集装箱）关系变量，如下所示：

---

```
CONTAINER{CONTNO,SHIPNO, .....}  
PRIMARY KEY {CONTNO}  
FOREIGN                                KEY {SHIPNO}REFERENCES  
INV_SHIP {SHIPNO}
```

---

现在，我们有一个引用一个备用键的外键！这是最初定义的关系模型所禁止的，正如我们所知道的。

那么，我们能避免这种明显违反原始模型的设计吗？当然，答案是肯定的。有各种不同的方法可能会完成这个任务：

- 1.我们可以重新回到两关系变量的设计上（但是，这会重新引入数据冗余和对额外的约束的需要）。

- 2.我们可以把CONTAINER关系变量中的

SHIPNO替换成INVNO。然而，这种做法似乎很不自然（集装箱本身与发票没什么关系），而且在设计中引入了一层不愉快的间接性（对于一个给定集装箱的发货，只能通过相应的发票来访问）。

3.我们可按原样保留CONTAINER关系变量，但把外键规格说明替换为一个显式的声明，其大意是在CONTAINER内出现的每个SHIPNO值也必须出现在INV\_SHIP中（使用类似SQL或Tutorial D这样的允许定义任意复杂约束的语言）。但要用这样一个迂回的方式处理一个如此类似于“真正的”外键约束的一个约束确实似乎很可惜，的确，可能会认为它的效果是再次引入一个不良的不对称性：引用主键的外键以一种方式处理，而引用备用键的“外键”以相当不同的另一种方式处理。

4.我们可以为INV\_SHIP引入一个代理主键（比如说，{ISNO}），并用它作为CONTAINER表的外键，这仍然会包含一层如上文第2段中的间接性，但至少会重新引入当我们任意选择{INVNO}作为INV\_SHIP的主键而丢失了的对称性。

总结：这四个“变通”方法似乎没有一个完全



使人满意。这样的例子似乎表明，如果我们希望避免冗余性、随意性、人为性、不对称性和间接性，那么我们就需要能够将主键和备用键平等地看待，而且我们需要能够拥有引用备用键的外键。换句话说，我们需要忽略主键和备用键之间的差异，并简单地认为它们只是键。请注意，但是，我不是说在这个例子中违反原本关系模型的戒律的明显需要是无法避免的，我想说的是，我既没有看到能避免它的一种良好方式，也没有一个很好的理由采用一种坏的方式。因此，我想建议把所涉及的戒律视为强(?)的指导方针，但不作为不可违反的规则。

[1] 一个审校者问，为什么一个包含三个关系变量（发票、发货和它们之间的关联）的设计并不是“明显”的设计？嗯，这可能是一个更好的设计，并且它可能是明显的一个。但是，关联关系变量仍然有两个键（INVNO和SHIPNO），并且这种说法导致的主要的结论（即，需要平等地处理这两个键）仍然存在。

[2] 注意，因此，此设计违反了正交性（见第14章）。

[3] 实际上，我见过这样的伎俩被人明确推荐，这些人本来应该更有见识的。

[4] 当然，根据当今的商业化产品的状态，要更新这种视图可能会有一些困难。但是，这是一个单

独的问题，超出了本附录（及这本书）的范围。

## A.4 是否每个实体类型一个主键

现在我想谈谈本附录引言中提到的两个问题中的第二个：即，一个给定类型的实体都应该在数据库中的任何地方以完全相同的方式确定。不严格地说，这一点的意思是通常存在：

- 相关实体类型的一个“锚”关系变量，具有某个特定的主键；

- 提供有关实体类型的进一步信息的零个或多个附属关系变量，每个都有一个引用那个锚关系变量的主键的外键。

（这种状况提醒你第15章讨论的RM/T原则了吗？）但随之出现了下面几个明显的问题。

- 对于一个可能对应实体类型的不同“角色”（见下一节）的给定实体类型，是否可能没有很好的理由有多个锚关系变量呢？

- 如果有多个这样的锚关系变量，是否可能没有很好的理由在不同的锚关系变量中有不同的主键——从而，这意味着相同的实体在不同的上下文中可能会以不同的方式确定吗？

■因此，在不同的关系变量中定义不同的外键，只是为了在不同的上下文中以不同方式标识同一个实体（再次重申），这样做的理由是否不够充分？

■最后，是否可能甚至没有良好的理由，对在同一关系变量中的同一实体有几个权重相等的不同标识符呢？

我们已经在本附录（A.2节）看到了几个例子，其中对这些问题中的最后一个的答案显然是肯定的。为了研究其他问题，让我们来考虑另一个例子。

## A.5 申请人及员工的实例

这个例子（它类似于发票和发货的例子，基于一个真实的应用程序）涉及申请在某企业的工作。关系变量APPLICANT（申请人）用于保存这类申请人的记录，如下所示：

---

---

```
APPLICANT{ANO,NAME,ADDR, .....}  
PRIMARY KEY{ANO}
```

---

---

申请人编号（ANO）是在申请人申请工作的那个时间分配的，它对申请人是唯一的，{ANO}从而构成明显的主键（事实上，它是唯一的键）。

接下来，多个进一步的关系变量用来保存申请人附属信息（以前的工作、证明人列表和相关人列表等）。我在这里只考虑其中的一个，即，“以前的工作”关系变量（APPLICANT\_JOBS）：

---

---

```
APPLICANT_JOBS{ANO,EMPLOYER,JOB,START,END,  
.....}  
PRIMARY KEY{ANO,START}  
ALTERNATE KEY{ANO,END}  
FOREIGN KEY{ANO}REFERENCES APPLICANT
```

---

---

---

注意，顺便说一下，我们似乎要再次面临对主键的一个任意选择，但那不是在这里要研究的关键问题。

现在，当求职者求职成功时，给他或她指派员工编号（ENO，对员工是唯一的），以及有关新员工的信息（职位、部门编号、电话号码等）记录在EMP关系变量中：

---

```
EMP {ENO,JOB,DNO,PHONENO, .....}  
PRIMARY KEY {ENO}
```

---

现在，我们有两个不同的锚关系变量，APPLICANT和EMP，完全相同的实体（即，一个成功的申请人）由这两个关系变量中的一个的ANO值确定，并由另一个的ENO值确定。当然，两个关系变量确实代表不同的角色，（在APPLICANT中的元组代表一个人属于申请人角色，而在EMP（如果有的话）中的相应元组代表同样的人属于员工角色），但事实仍然是，所涉及的只是单个实体。

上述内容不是故事的结局。显然，关系变量EMP需要以某种方式引用关系变量

APPLICANT（为简单起见，我假设，每一位员工都曾经是申请人，虽然这个假设可能有点不切实际）。因此，我们需要添加ANO属性到EMP关系变量中，并相应地定义一个外键：

---

```
EMP {ENO,ANO,JOB,DNO,PHONENO, .....}  
PRIMARY KEY {ENO}  
ALTERNATE KEY {ANO}  
FOREIGN KEY {ANO} REFERENCES APPLICANT
```

---

现在再次有两个候选键！即，{ENO}和{ANO}。这一点不久后将是有重大意义的，但是，现在我会忽略它。

接下来，当然，需要更多的关系变量来保存附属信息（员工薪资历史、福利详情等）。这里是薪酬历史关系变量：

---

```
SAL_HIST {ENO,DATE,SALARY, .....}  
PRIMARY KEY {ENO,DATE}  
FOREIGN KEY {ENO} REFERENCES EMP
```

---

现在，一个ENO值和ANO值不仅确定，而且引用同一个实体，这里的ENO是一个关系变量（SAL\_HIST）中的而ANO值是在其他关系变量（APPLICANT\_JOBS与EMP）中的ANO。换句话

说，数据库的结构如图A-3所示。

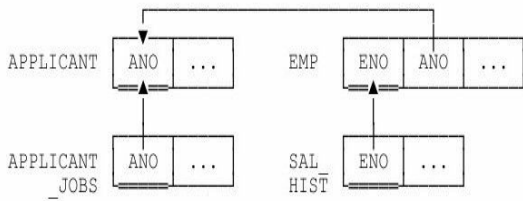


图 A-3 申请人和员工数据库

现在，我们通过把EMP当作APPLICANT的一个子类型，可能会避免对相同的实体类型有两个不同的标识符（ANO和ENO）的明显需要，毕竟，每一个员工都是一个申请人（不严格地说），而反之则不然。在这种方式中，我们可以使用{ANO}作为EMP的主键，用{ENO}作为一个备用键（或甚至完全删除它），并把ENO替换为在SAL\_HIST关系变量中的ANO。现在的数据库结构如图A-4所示：



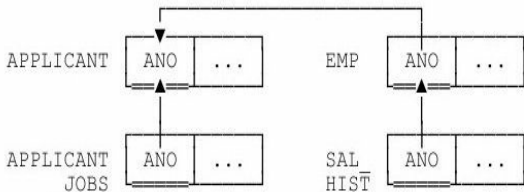


图 A-4 用{ANO}作为EMP的主键

但是，请注意这种状况的影响：它不仅造成数据库设计的改变，而且造成企业必须改变操作的方式。（一开始，它现在必须用员工的申请人编号，而不是员工编号来确定员工。）企业为什么要只是因为关系教条（“每个实体类型有一个主键”）而改变它做生意的方式呢？具体地说，即使申请人及员工都是人，且每一个员工确实都是（或曾经是）申请人，为什么不应该允许用申请人编号确定申请人而用员工编号来确定员工呢？

旁白：另一种可能性是引入一个PERSON关系变量，然后把APPLICANT和EMPLOYEE都视为PERSON的子类型。我把它的详细内容作为一个习题留给读者，我只是说，即使我们发明了一种“人编号”（PNO）并把{PNO}作为PERSON的主键，这种方法基本上也不会解决任何问题。另

一方面，当“相同的”主键在任何地方都涉及时（例如，如果我们要处理员工、程序员、系统程序员和应用程序员等，所有这些都根据员工编号确定），我肯定会推荐这种超类型/子类型的方法。请参阅在第15章中的例4。结束旁白。

总而言之：上述例子充分表明可能在有的场合中以下方法确实是首选的（a）相同的实体类型有几个不同的锚关系变量；（b）那些锚关系变量中的每个都具有一个不同的主键；且（c）在不同附属关系变量中有引用这些不同主键的不同外键。请再次注意，我不是说这里违反“每个实体类型有一个主键”的规定的明显需要是无法避免的，我想说的是，我既没有看到一种能避免它的好方式，我也没有看到一个很好的理由来采用一种坏方式。所以，我想建议，把“每个实体类型有一个主键”戒律视为一个强大的（？）的指导方针，而不是作为一个不可违反的规则。

## A.6 结束语

本附录我为下面几点提出了一些务实的论据：

■放宽普遍接受的“每个基础关系变量有一个称为主键的与众不同的键”的规则；

■放宽（可能不太普遍接受的）“每一个外键专门引用一个主键，而不是一个备用键”的规则；<sup>[1]</sup>

■放宽普遍接受的“每个实体类型正好有一个锚关系变量”的规则。

当然，我也知道，如果我们放宽这些规则，那么我们为糟糕的设计创造了可能性。这就是为什么我建议保留如“每个实体类型有一个主键”的建议作为经验法则，或不错的设计指导方针。换句话说，仅当这样做有一些很好的理由时，才应该违反所涉及的规则。但我已经竭力在本附录中显示，有时这种好的理由确实存在。

<sup>[1]</sup>我说不太普遍接受的，因为（值得称道的是）至少SQL标准，确实允许外键引用任何候选键。

## 附录B 冗余回顾

Nothing is certain but the unforeseen.

——19th century proverb

第13章讨论了我称之为RFNF（无冗余范式）的一个范式。不过，我也曾在那一章中提到完全相同的名字早已被Millist W.Vincent在一篇论文中用来表示相当不同东西的事实。[\[1\]](#)本附录将简单地介绍一下Vincent的RFNF。

考虑我们一贯的供应商关系变量S及其FD{CITY}→{STATUS}和如图1-1所示的示例值。那个关系变量中的供应商S1的元组具有城市伦敦和状态20，因此，供应商S4的元组，其中也有城市伦敦，必须有状态20，否则将违反FD{CITY}→{STATUS}。因此，从某种意义上说，出现在供应商S4的元组中的状态值20是多余的，因为没有别的可能，它是一个逻辑后果，完全由出现在关系中其他地方的值确定，这个关系是在所涉及的时间那个关系变量的当前值。

类似上述的例子为下面这个直观上吸引人的定义提供了动力（由Vincent提出的，但这里相当大幅度地转述）。

■定义：设关系 $r$ 是关系变量 $R$ 的一个值，设 $t$ 是 $R$ 中的一个元组，并设 $v$ 是 $t$ 中的属性值。那么，当且仅当把那个 $v$ 的出现替换为 $v'$ 的出现（ $v' \neq v$ ），而其他所有内容都保持不变，会导致违反 $R$ 的某个FD或JD时， $t$ 中 $v$ 的那次出现在 $r$ 中是冗余的，且 $R$ 存在冗余。

换句话说，如果所涉及的属性值的出现必须是 $v$ 而不是其他内容，那么存在冗余。

旁白：虽然我说过上述定义在直观上是有吸引力的（我认为是），我想我还应该指出的是，至少在一个方面，它也有点怪。再次考虑启发的例子，在关系变量 $S$ 中，因为供应商 $S1$ 的元组有状态值20，供应商 $S4$ 的元组必须有状态值20。注意相反的论点同样存在：因为供应商 $S4$ 的元组有状态值20，供应商 $S1$ 的元组的必须有状态值20！现在，说那两个20都是冗余的肯定是没有意义的（不是这样吗？），但是，我们认为两个中的哪一个是冗余的是任意的，这个事实似乎确实有些奇怪，至少对我来说是这样的。结束旁白。

即使如此，Vincent继续定义了一个新的范式，如下：[\[2\]](#)

■定义： $n$ 当且仅当关系变量 $R$ 不存在刚才定

义的冗余时，它属于（Vincent的）无冗余范式，RFNF。

那么，很明显，根据上述定义，一个不属于BCNF（或其实是4NF）的关系变量也不属于Vincent的RFNF（再次参见关系变量S的一个例子）。但是，一个属于“我们的”RFNF的关系变量会怎么样呢？[\[3\]](#)回顾下面来自第13章的例子。设我们有一个关系变量SPJ，它具有属性SNO（供应商编号）、PNO（零件编号）和JNO（工程编号），以及谓词供应商SNO提供零件PNO给工程JNO。此外，下面的依赖成立：

---

---

$$\{SNO, PNO\} \rightarrow \{JNO\}$$
$$\star \{\{SNO, PNO\}, \{PNO, JNO\}, \{JNO, SNO\}\}$$

---

---

那么，我们曾在第13章看到，如果关系变量包含以下三个元组：

---

---

$$\begin{aligned} t1 &= s1 \ p1 \ j2 \\ t2 &= s1 \ p2 \ j1 \\ t3 &= s2 \ p1 \ j1 \end{aligned}$$

---

---

那么下面的元组必须也出现：

---

---

但是， $\{SNO, PNO\}$ 是一个键，因为元组 $t1$ 和 $t4$ 具有相同的键值，所以它们实际上是同一个（并且因此， $j1=j2$ ）。因此，SPJ既不存在FD冗余，也不存在JD冗余（因为我在第13章定义了这些术语），因此，此关系变量属于“我们的”RFNF。

但是，现在注意，根据Vincent的定义，元组 $t1$ 中的 $j2$ 必须等于 $j1$ 的事实表示关系变量存在冗余！因此我们与Vincent的这两个RFNF在逻辑上是不同的，其实，从一个关系变量可以属于我们定义的RFNF而不属于文森特定义的RFNF，而相反的情况不成立这个意义上，Vincent的定义比我们的定义更严格。（当然，进一步可得，这两个RFNF真的需要不同的名字）。事实上，我们可以作出如下更强的陈述。

■定理：5NF蕴含SKNF；SKNF蕴含Vincent的RFNF；Vincent的RFNF蕴含我们的RFNF；而我们的RFNF蕴含4NF。相反的蕴含不成立。

Vincent也证明了以下有用的结果。

■定理：当且仅当关系变量 $R$ 属于BCNF，并

且对于在R中存在的每个JD J，都有J中那些是R的超键的分量的并集等于R的标题时，它属于 Vincent的RFNF。[4]

举例来说，再次考虑关系变量SPJ'。正如我们所知道的，在这个关系变量中存在下面的JD：

---

☼{{SNO, PNO}, {PNO, JNO}, {JNO, SNO}}

---

这个JD中仅有的一个是超键的分量是 {SNO,PNO}，所有超键分量的并集肯定不等于标题，因此，这个关系变量不属于 Vincent的 RFNF。

所以SPJ'是一个属于我们的RFNF但不属于 Vincent的RFNF的一个关系变量的例子。但是，我们可以找到一个属于 Vincent的RFNF但不属于 SKNF的一个关系变量的例子吗？请注意，这样的关系变量一定存在，因为 Vincent的RFNF严格地位于我们的RFNF和SKNF之间（即SKNF蕴含 Vincent的RFNF, Vincent的RFNF蕴含我们的 RFNF）。那么，如前面所讨论的，通过变量 SPJ'及其两个依赖，我们可以很容易构造一个这样的关系变量的例子：

---



$$\{SNO, PNO\} \rightarrow \{JNO\}$$

---

$$\odot \{\{SNO, PNO\}, \{PNO, JNO\}, \{JNO, SNO\}\}$$

并添加另一个FD:

---

$$\{PNO, JNO\} \rightarrow \{SNO\}$$

---

这个额外的依赖（当然，这意味着， $\{PNO, JNO\}$ 是另一个键）对应一个额外的业务规则：

■任何给定的零件p最多由一个供应商s供应给一个给定的工程j。

现在注意，关系变量SPJ的这个修订版本满足Vincent的定理的条件：具体而言，在 $JD \odot \{\{SNO, PNO\}, \{PNO, JNO\}, \{JNO, SNO\}\}$ 中的超键分量是 $\{SNO, PNO\}$ 和 $\{PNO, JNO\}$ ，它们的并集等于整个标题，所以此关系变量属于Vincent的RFNF。<sup>[5]</sup>同时，它不属于SKNF，因为同一个JD包含一个不是一个超键的分量 $\{JNO, SNO\}$ 。所以，我们在这里有一个属于Vincent的RFNF但不属于SKNF的一个关系变量的例子。

现在，我们已经看到了（a）属于Vincent的RFNF，但不属于SKNF和（b）属于我们的RFNF，但不属于Vincent的RFNF的关系变量的例子。但是，这些不同范式之间的“语法”差异可能有点难记，在这里总结它们可能会有所帮助：

■当且仅当关系变量 $R$ 属于BCNF，且对于在 $R$ 中存在的每个JD  $J$ ,  $J$ 的某个分量是 $R$ 的一个超键时，它属于我们的RFNF。

■当且仅当关系变量 $R$ 属于BCNF，且对于在 $R$ 中存在的每个JD  $J$ ,  $J$ 中是 $R$ 的超键的分量的并集等于标题 $R$ 时，它属于Vincent的RFNF。

■当且仅当对于关系变量 $R$ 中存在的每一个不可约JD  $J$ ,  $J$ 的每个分量都是 $R$ 的一个超键时， $R$ 属于SKNF。

现在，让我提醒你，来自第15章的冗余的定义如下：

■定义：设 $D$ 是一个数据库设计，设 $DB$ 是一个符合 $D$ 的数据库值（即在 $D$ 中提到的关系变量的值集合），设 $p$ 是一个不涉及任何存在量化的命题。如果 $DB$ 包含两个或多个不同的 $p$ 表示，那么 $DB$ 包含冗余，且 $D$ 允许冗余。

再次考虑关系变量SPJ（原始版本，没有额外的FD{PNO,JNO} $\rightarrow$ {SNO}）。我现在声称，根据这个定义，此关系变量的这个设计确实允许冗余。具体地说（并参照前面所示的示例元组），未量化的命题供应商s1提供零件p1给工程j1既由元组t1（或元组t4）的出现显式表示，又作为JD和下列命题的一个合乎逻辑的结果隐式地表示：

■供应商s1供应零件p1（给某个工程）：由元组t1表示

■零件p1被提供给工程j1（由某个供应商）：由元组t3表示

■工程j1由供应商s1供应（某个零件）：由元组t2表示

因此，我声明，Vincent关于冗余的专门定义与我在第15章中为一般的冗余给出的（“最终版本”）定义是一致的，并且可以看作后者的一个特例。

我想通过指出下面这一点结束本附录，即，一个关系变量可以属于“Vincent的RFNF”（甚至属于5NF），但仍然与Vincent的定义不完全相同，但肯定是非常相似的冗余。回想一下这个例子

（在第12章的一个注脚中简要讨论的）：设我们有一个关系变量SPJQ，它具有属性SNO、PNO、JNO、QTY（只有这些属性）和谓词“供应商SNO供应数量为QTY的零件PNO给工程JNO”。此关系变量的唯一键是{SNO,PNO,JNO}，因此它属于BCNF。请注意，在此关系变量中不存在JD $\bowtie$ {{SNO,PNO},{PNO,JNO},{JNO,SNO}}，但是，它确实存在于此关系变量在{SNO,PNO,JNO}上的投影中（换句话说，它是一个嵌入式依赖，如果你需要刷新你关于这个概念的记忆，请参阅第12章）。现在假设此关系变量包含下列元组（只包含这些元组）：

---

```
s1 p1 j2 100  
s1 p2 j1 200  
s2 p1 j1 300  
s1 p1 j3 400
```

---

然后，通过嵌入式依赖，我们必定有j3=j1。因此，如前所述，关系变量SPJQ肯定存在冗余，更重要的是，它存在的冗余与Vincent定义的那种冗余是非常相似的。然而，这个关系变量属于Vincent的RFNF！问题是，Vincent的冗余是关于所涉及的关系变量的FD和JD（只有这两种）定义的，它对任何其他也可能碰巧存在的限制，如嵌入式依赖，没有什么意义。

[1] Millist W.Vincent: “Redundancy Elimination and a New Normal Form for Relational Database Design,” in B.Thalheim and L.Libkin (eds.), Semantics in Databases.Berlin,FDR: Springer-Verlag (1998) .

[2] 请注意，刚才提到的随意性并没有影响这个定义（也许是幸运的）。

[3] 如第13章中指出的，“我们的”RFNF已经更名为ETNF（“必需的元组范式”）。如需进一步讨论，请参阅附录C中的“最新消息”。

[4] Vincent实际上是怎么做的：他定义，当且仅当关系变量R满足规定的条件时（即，当且仅当它属于BCNF，并对于在R中存在的每一个 $JD = J, J$ 中那些是R的超键的分量的并集等于R的标题时，R属于他称之为KCNF（键完全范式）的另外一个范式。然后，他继续证明，KCNF和他的RFNF在逻辑上是等价的。

[5] 我在此处依靠如下事实，即这个特定JD是存在的仅有的非平凡的JD。

## 附录C 重要论文回顾

History is not what you thought. It is what you can remember.

——W.C.Sellar and R.J.Yeatman: 1066 and All That

本附录提供了设计理论领域中具有深远影响的一些科研出版物中的一个简要而客观的概括。所涉及的出版物大体上按时间顺序排列。

如前所述的关系模型起源于Codd的两篇具有里程碑意义的论文。

■E.F.Codd: “Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks,” IBM Research Report RJ599 (August 19th, 1969) 和其他地方。

■E.F.Codd: “A Relational Model of Data for Large Shared Data Banks,” CACM 13, No.6 (June 1970); 重新发表在 Milestones of Research—Selected Papers 1958-1982 (CACM 25th Anniversary Issue), CACM 26, No.1 (January 1983) 和其他地方。

这些论文中的第一篇对设计本身没说什么。但是，第二篇论文有一节的标题是“范式”，它包括诱人的评论：“一类规范化的进一步操作是可能的。这些都不在本文中讨论。”这些评论在一个显示了如何消除关系值属性的例子（见附录D中习题12.8的答案）后面出现，这就是为什么Codd使用短语“进一步操作”的原因。

顺便说一句，上述论文的第二篇是术语连接陷阱（connection trap）（见第9章）的来源。

如前所述的设计理论开始于Codd在下面论文中提出的FD、2NF和3NF：

■E.F.Codd: “Further Normalization of the Data Base Relational Model, ”in Randall J.Rustin,ed., Data Base System: Courant Computer Science Symposia Series 6 (Prentice Hall, 1972) .

这里有两个简短的评论：首先，标题是误导性的，进一步规范化不是对关系模型做的某些东西，而是对关系变量做的某些东西，或者更确切地说，是对关系变量设计做的。（重复附录D中习题1.1的答案，如前所述的关系模型并不关心数据库碰巧是如何进行设计的，只要所涉及的设计不违反任何关系模型的戒律即可）其次，本文中

的一些材料的初始版本，可以在Codd早期的两篇论文中发现。第一篇是“The Second and Third Normal Forms for the Relational Model”，发表于1970年10月6日的IBM technical memo。第二篇是“Normalized Data Base Structure: A Brief Tutorial,”Proc.1971 ACM SIGFIDET Workshop on Data Description,Access,and Control,San Diego,Calif., (November 11th-12th, 1971)。[\[1\]](#)

希思定理实际上在Codd 1972年发表的规范化论文前就出现了。

■I.J.Heath: “Unacceptable File Operations in a Relational Data Base,”Proc.1971 ACM SIGFIDET Workshop on Data Description,Access and Control,San Diego,Calif. (November 11th-12th, 1971) .

BCNF是在以下论文中定义的（虽然它在那里称为第三范式）。

■E.F.Codd: “Recent Investigations into Relational Data Base Systems,”Proc.IFIP Congress,Stockholm,Sweden (North-Holland, 1974) 和其他地方。



阿姆斯特朗的FD公理的第一个陈述也在同样的IFIP会议上提出。

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

MVD和4NF，以及第12章提到的费金定理都是在下面的论文[\[2\]](#)中定义的。

■Ronald Fagin: “Multivalued Dependencies and a New Normal Form for Relational Databases, ”ACM TODS 2, No.3 (September 1977) .

MVD的公理化在以下论文中报道。

■Catriel Beeri,Ronald Fagin,and John H.Howard: “A Complete Axiomatization for Functional and Multivalued Dependencies, ”Proc.1977 ACM SIGMOD International Conference on Management of Data,Toronto,Canada (August 1977) .

依赖保持的理论起源于：

■Jorma Rissanen: “Independent Components of Relations,”ACM TODS 2, No.4 (December 1977) .

通常称赞下面的论文首次指出，不等于其任何两个投影的连接，但等于其三个或多个投影的连接的关系变量是可以存在的，（尽管事实上，正如第9章提到的，Codd实际上在他1969年的论文中提出了同样的观点）。这篇论文也是追逐算法的源，但是，它只考虑了FD和MVD，而不是一般的JD。

■A.V.Aho,C.Beerli,and J.D.Ullman: “The Theory of Joins in Relational Databases,”ACM TODS 4, No.3 (September 1979) ; previously published in Proc.19th IEEE Symposium on Foundations of Computer Science (October 1977) .

如前所述的JD首次定义在以下论文中。

■Jorma Rissanen: “Theory of Relations for Databases—A Tutorial Survey,”Proc.7th Symposium on Mathematical Foundations of Computer Science,Springer-Verlag Lecture Notes in Computer Science 64 (Springer-Verlag, 1979) .

下一篇论文提出了投影连接范式

(PJ/NF)，也称为5NF的概念。它可以被视为所谓的“经典”规范化理论（即基于投影的无损分解的分解操作符和以自然连接作为相应的重构操作符的理论，以及BCNF，4NF，5NF这些范式）的明确声明。

■Ronald Fagin: “Normal Forms and Relational Database Operators,”Proc.1979 ACM SIGMOD International Conference on Management of Data,Boston,Mass. (May/June 1979) .

下一篇论文提出了关于包含依赖 (INclusion Dependency,IND) 的一套健全和完善的推理规则，换句话说，一个公理化。注意：我不知道任何地方有对相等依赖 (EQD) 的任何正规的处理，这是一个重要的，但在某些方面相当平凡的特例。

■Marco A.Casanova,Ronald Fagin,and Christos H.Papadimitriou: “Inclusion Dependencies and Their Interaction with Functional Dependencies,”Proc.1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems,Los Angeles,Calif. (March 1982) .

域-键范式定义于以下论文中。

■Ronald Fagin: “A Normal Form for Relational Databases That Is Based on Domains and Keys, ”ACM TODS 6, No.3 (September 1981) .

6NF定义于以下论文中。

■C.J.Date,Hugh Darwen,and Nikos A.Lorentzos: Temporal Data and the Relational Model (Morgan Kaufmann, 2003) .

至于正交，它的概念在下面文章中第一次讨论（虽然叫的不是这个名字）。

■C.J.Date and David McGoveran: “A New Database Design Principle, ”in C.J.Date,Relational Database Writings 1991-1994 (Addison-Wesley, 1995) ; previously published in Database Programming& Design 7, No.7 (July 1994) .

然而，请注意，本书描述的正交性与在上述论文讨论的版本是显著不同的。（我为这种状况承担全部责任，虽然这一概念最初由David McGoveran提出，但我写了大量引用的论文，我现在才意识到，我当时这样做时是相当困惑

的。)

最新消息：“我们的”无冗余范式（请参阅第13章和附录B）最近已更名为必需元组范式（ETNF）。它在以下论文中描述。

■Hugh Darwen, C.J. Date, and Ronald Fagin: “A Normal Form for Preventing Redundant Tuples in Relational Databases,” Proc. 15th International Conference on Database Theory, Berlin, Germany (March 26th-29th, 2012).

这篇论文证明了第13章讨论的所有有关“我们”的RFNF（现在称为ETNF）的定理、结果等，但有些术语已修订。请让我把这篇论文中的术语与第13章中的术语联系起来。首先，设 $R$ 是关系变量，设 $r$ 是 $R$ 的一个值，设 $t$ 是 $r$ 中的一个元组。那么当且仅当 $t$ 是部分冗余或完全冗余时， $t$ 在 $r$ 中是冗余的。这篇论文显示，（a）当且仅当 $R$ 不属于BCNF时（即，当且仅当 $R$ 是FD冗余的，见第13章），该元组存在且是部分冗余的；（b）当且仅当在 $R$ 中存在一个元组强迫JD时（即，当且仅当 $R$ 是JD冗余的，请再次参阅第13章），这样的元组存在且是完全冗余的。因此，请注意，这个“完全冗余”并不是“部分冗余”的一个特例，事实上，一个元组可以部分冗余但不完全冗余，或

相反。最后，当且仅当元组 $t$ 在 $R$ 中不是冗余的时， $t$ 在 $r$ 中是必需的。如果 $R$ 属于ETNF，那么每一个是 $R$ 的一个合法值的关系 $r$ ，都使得在 $R$ 中的每个元组都是必需的。注意：我们在这种情况下选择使用术语必需的，是受到Codd的必要性概念的影响，它在下面的论文中提出：E.F.Codd and C.J.Date: “Interactive Support for Nonprogrammers: The Relational and Network Approaches,” in Randall J.Rustin (ed.), Proc.ACM SIGMOD Workshop on Data Description, Access, and Control—Data Models: Data-Structure-Set versus Relational (Ann Arbor, Michigan, May 1st-3rd, 1974)。[3]简单地说，根据Codd的论述，说一些数据结构是必需的就是说其缺失会造成信息的损失。正如已经指出的，从这个意义上讲，在每个是一个ETNF关系变量的一个可能值的关系中的每一个元组都是必需的。

[1]第二篇论文对2NF和3NF本身没有太多的涉及，因为它的观点是，关系可以代表其他的数据结构（层次结构、网络结构等）能代表的任何东西。它确实很简要地讨论了2NF和3NF，但其对这些主题的覆盖范围基本上限于在每一种情况下给出一种相当非正式的例子。

[2]事实上，还有大量计算文献的理论结果，而不只是在如前所述的设计理论领域中，这些都可以名正言顺地称为“费金定理”。

[3]再次发表于C.J.Date,Relational Database: Selected Writings (Addison-Wesley, 1986) .

## 附录D 习题答案

Science offers the best answers.

——Richard Dawkins: Break the Science  
Barrier

注意：本附录中的所有错误都是故意的＜开玩笑＞。

### 第1章

1.1 正确，它不要求。良好的设计有利于用户，并在一定程度上对DBMS也是如此，但这里所说的关系模型并不关心数据库恰巧是如何进行设计的，只要它不违反任何关系模型的戒律即可，尤其是，只要它处理的对象确实是关系，而不是别的东西即可（遗憾的是，在SQL中往往存在这些别的东西）。

1.2 参见第15章。

1.3 参见第4章及第5章。

1.4 正确（请参见第4章及第5章）。

1.5 不正确（其实，甚至连“每个二元关系



变量都属于2NF”都不是不正确的。见习题4.6）。

1.6 不正确（参见第9章及第10章）。

1.7 更不可能正确，参见习题1.5答案。

1.8 不正确（参见第13章）。

1.9 不正确（参见第13章）。

1.10 参见第10章。

1.11 不正确（参见第9章及第15章）。

1.12 参见第8章。

1.13 参见第5章。

1.14 参见第14章。

1.15 参见第11章。

1.16 参见第7章。

1.17 参见第11章。

1.18 参见第13章及附录B。



## 第2章

2.1 信息原则 (The Information Principle) 是支撑整个关系模型的基本原则。它可以表示为如下：

■定义：信息原则，说明在关系数据库中允许的唯一一种变量是关系变量。等价地，整个信息数据库的内容在任何给定的时间都以一种方式并且只能以一种方式表示，即，作为关系的元组中的属性位置的值。

需要注意的是，涉及从左到右的列顺序，或包含重复的行或空值的SQL表（至少，数据库中的SQL表），全都违反了信息原则（参见下一个习题的答案）。然而，有趣的是，具有匿名列或非唯一名称的列的SQL表，显然不违反原则。其原因是，此原则明确声明适用于数据库中的关系或关系变量。而虽然一般的SQL表中可以有匿名的列或非唯一名称的列，但这样的表不可能是数据库的一部分。这种状况强烈地表明，信息原则还可以更严格一点。

2.2 a.正确。b.正确。c.正确。d.正确。e.正确。f.正确。g.正确。h.不正确。（但是，它“几乎是”正确的，有两个小例外，出于当前的目

的，我将对它们只是稍微做简化。第一个例外是，如果关系 $r$ 的类型是 $T$ ，那么 $r$ 的任何属性本身的类型都不可以是 $T$ 。第二个例外是，数据库中的关系都不可以具有任何指针类型的属性）。i. 正确。附加习题：如果把原先的问题陈述改为用SQL表而不是关系或关系变量的形式来问，这些答案是否有任何改变呢？（答案：是的，除a和h外，其余所有问题的答案都会改变。此外，对于问题h的情况，答案应该也真的改变了，从“不正确”变为“不正确，但更是这样。”产生这种状况的原因之一，而不是唯一原因——是SQL并没有表类型的真正概念，因此SQL列更不可能是这种类型）。

2.3 a.正确。b.正确。c.正确。注意：为了备案，也许我应该说明，按照通常的做法，我在这本书中将“B是A的子集”形式的表达式当作包含“B和A是相等的”的可能性。因此，例如，每个标题都是它本身的一个子集，每个正文也是如此，且每一个元组也是如此。当我想排除这种可能性时，我将明确地用术语“真子集”来说明。例如，我们平时的供应商关系的正文肯定是其本身的一个子集，但不是其本身的一个真子集（任何集合都不是其本身的一个真子集）。更重要的是，上述言论同样适用于超集，加以必要的变通，例如，我们平时的供应商关系的正文是其本

身的一个超集，但不是其本身的一个真超集。更加术语化的：一个集合包含（include）它的子集。顺便说一句，不要将它与含有（containment）混淆，一个集合包含它的子集，但含有它的元素。

2.4 本章正文中没有提到“域”这个术语的原因是，它只是类型（type）的一个代名词。（早期的关系理论著作，包括我自己的在内，往往使用它，但较近期的著作都使用类型代替它，因为它更短<sup>[1]</sup>，并至少在计算领域中无论如何有一个更广泛的血统。）因此，一个域是一个命名的、有限的值集合，其所有可能的值都属于一些特定的种类：例如，所有可能的整数，或者所有可能的字符串，或所有可能的三角形，或者所有可能的XML文档，或所有可能带有特定标题的关系（等等）。顺便说一下，不要把关系领域中理解的域与在SQL中相同名称的结构混淆，后者最好被当作一种非常弱的类型（参见《SQL与关系数据库理论》的解释）。

2.5 参见本章的正文。

2.6 关系变量S：供应商SNO的名称为SNAME，并位于城市CITY，该城市的状态为STATUS。关系变量P：零件PNO的名称为

PNAME，具有颜色COLOR和重量WEIGHT，并存储在城市CITY。关系变量SP：供应商SNO提供数量为QTY的零件PNO。

## 2.7 没有提供答案。

2.8 封闭世界假设（The Closed World Assumption）表明，不严格地说，在数据库中明示或暗示的一切都是真实的，其他一切都是假的。[\[2\]](#)而开放世界假设（The Open World Assumption）——是的，有这样的东西——表明，在数据库中明示或暗示的一切都是真实的，其他一切都是未知的。那么，它的含义是什么？首先让我们同意把封闭世界假设和开放世界假设分别简称为CWA和OWA。现在考虑查询“供应商S6在罗马吗？”（更精确地说，它的意思是，“关系变量S的供应商S6中是否有一个CITY值等于罗马的元组呢？”）。它的Tutorial D表达方式如下：

---

---

```
(S WHERE SNO='S6' AND CITY='Rome') {}
```

---

---

正如在《SQL与关系数据库理论》中解释的，这个表达式的值要么是TABLE\_DEE，要么是TABLE\_DUM（其中TABLE\_DEE和

TABLE\_DUM是仅有的度为0的关系；其中TABLE\_DEE只包含1个元组，而TABLE\_DUM根本不包含任何元组）。此外，根据CWA，如果结果是TABLE\_DEE，它指的答案是“是，供应商S6确实存在并在罗马”，如果结果是TABLE\_DUM，它指的答案是“否，供应商S6存在并在罗马这种情况不成立”。相比之下，根据OWA，TABLE\_DEE仍然意味着“是”，但TABLE\_DUM意味着“不知道供应商S6是否存在并在罗马”。

现在考虑查询“如果供应商S6存在，它是在罗马的供应商吗？”（请注意这个查询与上面讨论的查询的逻辑区别）。注意，如果关系变量S表示“供应商S6是存在的，但在罗马以外的某个城市”，那么无论我们在谈论CWA或OWA，此查询的答案都是“否”。[\[3\]](#)因此，它的Tutorial D表达方式如下：

---

```
TABLE_DEE MINUS ( (S WHERE SNO='S6' AND CITY≠'Rome') {} )
```

---

请注意，因此，如果这个表达式的判断结果为TABLE\_DUM，这个TABLE\_DUM必须表示“否”，即使根据OWA，也是如此。因此，OWA存在一种固有的二义性：有时TABLE\_DUM的意

思是“不知道”，有时它意味着“否”，当然我们不能（一般地）说何时适用哪个解释。

要点在于：在关系领域中，TABLE\_DEE和TABLE\_DUM的意思确实分别是“是”和“否”，而且在关系领域中不存在度为零的“第三个关系”来代表OWA基本要求的“第三个真值”。因此，OWA和关系模型从根本上是互不兼容的。

## 2.9 精确的定义在第5章中给出。

2.10 任何种类的两个值相等，当且仅当它们是相同的值（更不用说，这意味着，它们必须是同一类型的）。特别是：（a）两个元组 $t$ 和 $t'$ 是相等的，当且仅当它们有相同的属性 $A_1, \dots, A_n$ ，并且对所有的 $i$ （ $i=1, \dots, n$ ）， $t$ 中的 $A_i$ 的值和 $t'$ 中的 $A_i$ 的值是相等的；（b）两个关系 $r$ 和 $t'$ 是相等的，当且仅当它们有相同的标题和相同的正文（即，它们的标题是相等的，它们的正文也是相等的）。请特别注意，因此，两个“空关系”（即没有任何元组的关系），当且仅当其标题相等时，它们是相等的。

2.11 是的！然而，我们当然会希望这样的操作总是产生一个有效的元组作为结果（即，我们会希望封闭（closure）这些操作，就像我们有



关系操作闭包，参见下面习题2.16的答案）。例如，对于元组并操作，我们会希望输入的元组是这样的情况：同名的属性具有相同的值（因此也是同一类型的，这就更不用说了）。举例来说，设t1和t2分别是一个供应商元组和一个发货元组，并且设t1和t2具有相同的SNO值。那么t1和t2的并， $\text{UNION}\{t1, t2\}$ 是——使用Tutorial D语法——一个类型为 $\text{TUPLE}\{\text{SNO} \quad \text{CHAR}, \text{SNAME} \quad \text{CHAR}, \text{STATUS} \quad \text{INTEGER}, \text{CITY} \quad \text{CHAR}, \text{PNO} \quad \text{CHAR}, \text{QTY} \quad \text{INTEGER}\}$ 的元组，具有与t1或t2中或两者中相同（如适用）的属性值。例如，如果t1是（S1, Smith, 20, London），而t2是（S1, P1, 300），使用本章正文中介绍的元组速记符号，那么它们的并集是元组（S1, Smith, 20, London, P1, 300）。注意：此操作可能合理地称为“元组连接”，而不是“元组并”。

顺便说一下，不只是一般的集合操作符可以改写为适用于元组，某些众所周知的关系操作符也可以，（其实我只是建议）。一个重要的例子是元组的投影操作，它是关系的投影操作符的一个简单改写。例如，设t是一个供应商的元组，那么t在属性{SNO,CITY}上的投影 $t\{SNO,CITY\}$ 是只包含t中SNO和CITY分量的t的子元组。（当然，子元组其本身就是一个元组）。同样， $t\{CITY\}$ 是只包含t中的CITY分量的t的子元组，而

$t\{\}$ 是完全不包含分量的 $t$ 的子元组（换句话说，它是0-元组，参见下一个习题）。事实上，这是值得明确注意的，每个元组在属性空集上都有一个投影，准确地说，这个投影的值为0-元组。

2.12 空的元组（注意，只有一个这样的元组，等价地，所有的空元组都是彼此相等的）与前一个习题的答案中提到的0-元组是一回事。对于这样一个元组的用途，我只想说的是，至少在概念上，这样的元组确实存在这一事实在许多方面非常重要。特别是空的元组是特殊关系TABLE\_DEE中唯一的元组，这在习题2.8的答案中已经提到了。

2.13 如果说关系变量 $R$ 有一个空的键，即是说 $R$ 永远不可以包含多个元组。为什么呢？因为对于属性空集，每一个元组具有相同的值，即，空的元组（参见前两个习题的答案），因此，如果 $R$ 有一个空的键，并且如果 $R$ 包含两个或多个元组，我们就会违反键的唯一性。是的，限制 $R$ 永远不会包含多个元组肯定是有意义的。我将把发现这种情况的一个例子作为附加习题。

2.14 一个参数为空集的谓词是一个命题。（换句话说，命题是退化的谓词，所有命题都是谓词，但“大部分”谓词不是命题）。

2.15 投影和连接的定义在第5章中给出，但这里有一个RENAME（重命名）的定义。

■定义：设 $r_1$ 是一个关系，设 $A$ 是 $r_1$ 的一个属性，并设 $r_1$ 没有名为 $B$ 的属性，那么重命名 $r_1$   
 $\text{RENAME}\{A \text{ AS } B\}$ 是一个关系 $r_2$ ，（a）它的标题与 $r_1$ 的相同，除了标题中的属性 $A$ 改名为 $B$ ，并且（b）它的正文与 $r_1$ 的相同，除了正文（更确切地说，在该正文的元组中）中对 $A$ 的所有引用被替换为对 $B$ 的引用。注意：Tutorial D另外还支持一种允许两个或多个单独的重命名并行地执行（“多个RENAME”）的RENAME形式。这种形式的例子在第14章中给出。

2.16 关系代数由（说起来很不严格）允许我们可以从“旧”的关系推导出“新”的关系的操作符组成。每个这样的操作符需要以一个或多个关系作为输入，并产生另一种关系作为输出（例如，差操作符需要以两个关系作为输入，并且以从其中一个“减去”另一个来派生另一个关系作为输出），而这就是封闭性。请注意，这个属性（除其他事项外），允许我们编写嵌套关系表达式，因为每一个操作的输出都是与输入同一类的东西，所以一个操作的输出就可以成为另一个操作的输入。例如，我们可以求关系 $r_1$ 和 $r_2$ （按照这个顺序）之间的差，将此计算结果作为输入与

某个关系 $r_3$ 执行一个并操作，再把并操作的结果作为输入输入到一个投影或限制等。

[1]英文单词type比domain短，中文刚好相反。

[2]为了说明我在这里说的“明示或暗示”的意思是什么，请考虑如图1-1所示的发货元组（S1，P1，300）。元组明示命题“供应商S1供应数量为300的零件P1。”不过，这也暗示着其他几个命题，例如，命题“供应商S1至少供应数量为300的一种零件。”

[3]反之，如果关系变量S中没有供应商S6的元组（在逻辑上，如果p是假的，那么“如果p，那么q”是真的，——再说一遍，无论我们谈论的是CWA还是OWA），答案必须是“是”。

## 第3章

3.1 参考第1章（图1-2）所示的关系变量STP的示例值，除非供应商S5提供某些零件，否则我们不能插入供应商S5具有状态30的事实；我们不能在不失去供应商S3的状态是30的事实的条件下删除供应商S3的发货；我们也不能改变一个给定的供应商（比如说，供应商S1）的一个元组中的状态，而不改变它所有元组中的状态。最明显的分解方法是将其关系变量分解为标题分别为{SNO,STATUS}和{SNO,PNO,QTY}的两个关系变量，这也是明显的，这个分解避免了异常。注意：值得指出的是，造成插入和删除异常的原因是设计在逻辑上不正确，而修改异常则是由它是冗余的这一事实所造成的（见习题3.3）。

3.2 设把 $r$ 的标题划分成属性集合 $X$ 、 $Y$ 和 $Z$ ，并且设 $r_1$ 和 $r_2$ 分别是 $\{X,Y\}$ 和 $\{Y,Z\}$ 上的投影。（按照定义， $X$ 、 $Y$ 和 $Z$ 是不相交的。）现在，设 $(x,y,z)$ 是 $r$ 的一个元组，那么 $(x,y)$ 和 $(y,z)$ 分别是 $r_1$ 和 $r_2$ 的元组，而且因此 $(x,y,z)$ 是 $r_1$ 和 $r_2$ 的连接中的一个元组。附加习题：如果上述证明中的集合 $Y$ 是空的，会发生什么情况？

3.3 本章正文解释了这两个目的（校正不正确的设计和减少冗余）。至于你是否认为该点得

到广泛理解：只有你能回答这个问题，但如果让我自己来说，我必须说，我不认为该点得到广泛理解。

## 第4章

4.1 FD的完整集合——这正式地称为闭包（closure），尽管它与关系代数的封闭性属性没有什么关系——对于关系变量SP，共包含31个FD。它们的清单如下：

---

$\{SNO, PNO, QTY\} \rightarrow \{SNO, PNO, QTY\}$

$\{SNO, PNO, QTY\} \rightarrow \{SNO, PNO\}$

$\{SNO, PNO, QTY\} \rightarrow \{SNO, QTY\}$

$\{SNO, PNO, QTY\} \rightarrow \{PNO, QTY\}$

$\{SNO, PNO, QTY\} \rightarrow \{SNO\}$

$\{SNO, PNO, QTY\} \rightarrow \{PNO\}$

$\{SNO, PNO, QTY\} \rightarrow \{QTY\}$

$\{SNO, PNO, QTY\} \rightarrow \{\}$

$\{SNO, PNO\} \rightarrow \{SNO, PNO, QTY\}$

$\{SNO, PNO\} \rightarrow \{SNO, PNO\}$

$\{SNO, PNO\} \rightarrow \{SNO, QTY\}$

$\{SNO, PNO\} \rightarrow \{PNO, QTY\}$

$\{SNO, PNO\} \rightarrow \{SNO\}$

$\{SNO, PNO\} \rightarrow \{PNO\}$

$\{SNO, PNO\} \rightarrow \{QTY\}$

$\{SNO, PNO\} \rightarrow \{\}$

$\{SNO, QTY\} \rightarrow \{SNO, QTY\}$

$\{SNO, QTY\} \rightarrow \{SNO\}$

$\{SNO, QTY\} \rightarrow \{QTY\}$

$\{SNO, QTY\} \rightarrow \{\}$

$\{PNO, QTY\} \rightarrow \{PNO, QTY\}$

$\{PNO, QTY\} \rightarrow \{PNO\}$

$\{PNO, QTY\} \rightarrow \{QTY\}$

$\{PNO, QTY\} \rightarrow \{\}$

$\{SNO\} \rightarrow \{SNO\}$   
 $\{SNO\} \rightarrow \{\}$   
 $\{PNO\} \rightarrow \{PNO\}$   
 $\{PNO\} \rightarrow \{\}$   
 $\{QTY\} \rightarrow \{QTY\}$   
 $\{QTY\} \rightarrow \{\}$   
 $\{\} \rightarrow \{\}$

---

其中非平凡的FD只有下面4个：

$\{SNO, PNO\} \rightarrow \{SNO, PNO, QTY\}$   
 $\{SNO, PNO\} \rightarrow \{SNO, QTY\}$   
 $\{SNO, PNO\} \rightarrow \{PNO, QTY\}$   
 $\{SNO, PNO\} \rightarrow \{QTY\}$

---

其中不可约的FD有下面11个：

$\{SNO, PNO\} \rightarrow \{SNO, PNO, QTY\}$   
 $\{SNO, PNO\} \rightarrow \{SNO, PNO\}$   
 $\{SNO, PNO\} \rightarrow \{SNO, QTY\}$   
 $\{SNO, PNO\} \rightarrow \{PNO, QTY\}$   
 $\{SNO, PNO\} \rightarrow \{QTY\}$   
 $\{SNO, QTY\} \rightarrow \{SNO, QTY\}$   
 $\{PNO, QTY\} \rightarrow \{PNO, QTY\}$   
 $\{SNO\} \rightarrow \{SNO\}$   
 $\{PNO\} \rightarrow \{PNO\}$   
 $\{QTY\} \rightarrow \{QTY\}$   
 $\{\} \rightarrow \{\}$

---



4.2 是，它是真的。（“每当两个元组在X上是一致的，它们在Y上也是一致的”意味着，所涉及的元组在属性X和Y上的投影之间的比较）

4.3 没有提供答案。

4.4 首先，这里有两个定义，为了便于后面引用，我将它们进行了编号：

1.当且仅当对于关系变量R的每一个键K和R的每个非键属性A,FD  $K \rightarrow \{A\}$ 都是不可约的时，R属于2NF。

2.当且仅当对于关系变量R存在的每一个非平凡的FD  $X \rightarrow Y$ ，以下各点中都至少有一个为真时：（a）X是一个超键，（b）Y是一个子键，（c）X不是一个子键，R属于2NF。

现在，定义2表明当且仅当在R中存在一个非平凡的FD  $X \rightarrow Y$ ，其中X不是超键且Y不是一个子键且X是一个子键时，R不属于2NF。但是，如果X是一个子键，而不是一个超键，它一定是一个真子键，因此，定义2相当于说，当且仅当在R中存在一个非平凡的FD  $X \rightarrow Y$ ，其中X是一个真子键且Y不是一个子键时，R不属于2NF。因此，设X作为一个真子键包含于键K，那么R中存在的FD

$K \rightarrow Y$ 是可约的，因此，对于 $Y$ 中包含的每个属性 $A$ ,  $R$ 中存在的FD  $K \rightarrow \{A\}$ 都是可约的。所以定义2意味着定义1。按照这种说法，反过来同样表明，定义1也意味着定义2。

#### 4.5 考虑以下说法。

设关系变量 $R$ 不属于2NF。那么就一定有 $R$ 的某个键 $K$ 和 $R$ 的某个非键属性 $A$ , FD  $K \rightarrow \{A\}$ （必须是在 $R$ 中存在的）是可约的，这意味着，一些属性可以从 $K$ 中删除，得到比如 $K'$ ，使得FD  $K' \rightarrow \{A\}$ 仍存在。因此 $K$ 必须是复合键。

这种说法似乎表明，本题的答案一定是“正确”，即，如果关系变量不属于2NF，那么它一定有一个复合键。但该说法是不正确的！下面就是一个反例。设USA是一个属性为COUNTRY（国家）和STATE（州）的二元关系变量，谓词是“州是国家的一部分”，但每一个元组的国家都是美国。现在， $\{STATE\}$ 是此关系变量唯一的键，而FD  $\{STATE\} \rightarrow \{COUNTRY\}$ 因此一定存在。然而，FD  $\{\} \rightarrow \{COUNTRY\}$ 也显然存在（见下面习题4.10的答案）；FD  $\{STATE\} \rightarrow \{COUNTRY\}$ 因此是可约的，并因此关系变量不属于2NF，但键 $\{STATE\}$ 也不是复合键。

4.6 不正确！下面举一个反例，考虑前面一道习题的答案的关系变量USA。此关系变量遵守 $FD\{ \} \rightarrow \{ COUNTRY \}$ ，这个函数依赖既不是平凡的，也不是一个从超键出来的箭头，所以此关系变量不属于BCNF。（当然，事实上，它甚至不属于2NF，如我们在前面的习题的答案中所见。）由此可见，此关系变量可以无损地分解成分别在 $\{ COUNTRY \}$ 和 $\{ STATE \}$ 上的两个一元投影。（请注意，重建原始的关系变量需要的相应连接其实是一个笛卡儿积）。

4.7 正确。如果根本不存在不平凡的FD——对于一个“全键”关系变量的情况，这是肯定的——那么对于决定因素不是一个超键的情况，不存在不平凡的FD，所以此关系变量属于BCNF。

## 4.8

---

```
CONSTRAINT.....  
COUNT ( SNP { SNO, SNAME } ) = COUNT ( SNP { SNO } ) ;  
CONSTRAINT.....  
COUNT ( SNP { SNO, SNAME } )  
= COUNT ( SNP { SNAME } ) ;
```

---

顺便说一下，这种规定一个FD的方式——即

指出，两个投影具有相同的基数——确实做了这项工作，如在第2章中指出的，但是，它很难做到非常优雅，因为这个原因，在那一章中，我使用AND、RENAME和JOIN展示了一个不同的方法。另外，Hugh Darwen和我提出了[\[1\]](#)Tutorial D应该支持的另一种形式的CONSTRAINT（约束）语句，在这种形式中，通常的布尔表达式被替换为关系表达式的组合和一个或多个键的规格说明。使用这个语法，上述约束可以写成这样：

---

```
CONSTRAINT.....  
SNP{SNO,SNAME}KEY{SNO}KEY{SNAME};
```

---

说明：把如下关系表达式——本例中的SNP{SNO,SNAME}——想象为定义某个临时关系变量（也许是一个视图）；那么键规格说明——在上面的例子中是KEY{SNO}和KEY{SNAME}——表示该指定的属性构成关系变量的键。

顺便说一句，我注意到，Darwen和我也建议允许外键约束用同种表达方式指定。

4.9 设R中存在FD  $X \rightarrow Y$ 。通过定义，X和Y是R的标题的子集。由于一个包含n个元素的集合

具有 $2^n$ 个可能的子集，因此，每个X和Y都具有 $2^n$ 个可能的值，因此在R中可能存在的FD的数量上限是 $2^{2n}$ 。例如，如果R的度是5，可能存在的FD的最大数量为1 024个（其中243个是平凡的）。附加习题：（a）243这个数字从何而来？（b）假设这1 024个FD都实际存在。这种情况下，关于R我们可以得出什么结论？（答案：它必须具有少于2的基数，原因是，在这样的情况下，存在的一个FD是 $\{\} \rightarrow H$ ，其中H是标题，因此 $\{\}$ 是一个键，所以约束R最多只能包含一个元组，如下一个习题的答案中所解释的。）

至于R可以有多少个键：设m是大于或等于 $n/2$ 的最小整数。如果（a）m个属性的每个不同的集合都是一个键，或（b）m为奇数且 $(m-1)$ 个属性的每个不同的集合都是一个键，R将有最大可能数量的键。无论哪种方式，都得出键的最大数量为 $n! / (m! * (n-m)!)$ 。[\[2\]](#)例如，一个度为5的关系变量至多可以有10个不同的键。

**4.10** 设关系变量R中存在指定的FD  $X \rightarrow Y$ 。现在，每一个元组（不管它是否是R的一个元组）对于该元组在属性的空集上的投影具有相同的值——即0元组。如果Y是空的，因此，FD  $X \rightarrow Y$ 适用于R的属性的所有可能的集合X，事实

上，它是一个平凡的FD（所以它不是很有趣），因为空集是所有集合的一个子集，并且因此在这种情况下，Y肯定是X的一个子集。另一方面，如果X是空的，那么在任何给定的时间内，FD  $X \rightarrow Y$  表示的是，R的所有元组具有相同的Y值（因为它们当然都具有相同的X值）。更重要的是，如果Y反过来是R的标题（换句话说，如果X是一个超键），则限制R至多含有一个元组。注意：在后一种情况下，X正好不是一个超键，而是一个键，因为它当然是不可约的。更重要的是，它是唯一的键，因为标题的所有其他子集都把它作为一个真子集来包含。

4.11 考虑数据库目录中的一个关系变量，其目的是记录数据库中的各个关系变量存在的FD，由于FD是一种形式为 $X \rightarrow Y$ 的表达式，其中X和Y是属性名集合（见第5章），该目录关系变量的合理设计是具有属性R（关系变量名）、X（决定因素）和Y（依赖因素），并具有谓词“R中存在 $X \rightarrow Y$ ”，那么，对于任何给定的R值，X和Y的值是度为1的关系，其元组包含名为R的关系变量的属性名。

对于另一个涉及“用户关系变量”而不是目录中的一个关系变量的例子，你可能想考虑一下以下问题：

我决定举办一次聚会，所以我起草了我想邀请的人的一份名单，并做了一些初步调查。大家对此聚会反应良好，但有几个人，他们将根据某些其他被邀请者的接受情况来决定是否出席。例如，Bob和Cal都表示，如果Amy来了，那么他们会来；Fay说，如果Don和Eve都来了，那么她会来；Guy说无论如何他要来；Hal说如果Bob和Amy都来了，那么他会来，等等。设计一个数据库来显示哪些人在哪些人接受邀请的基础上接受邀请。（感谢Hugh Darwen）。

在我看来，一个合理的设计会涉及一个带有两个属性X和Y的关系变量，两个都是关系值，而谓词是，集合X中的人都将出席，当且仅当集合Y中的人都将出席。附加习题：对这样的设计你能想到任何你可能进行的改进吗？（提示：当且仅当Bob将出席时，Bob会出席，这是真的？）

4.12 我不知道你的结论是什么，但我知道我做的东西。我的结论之一是，我们应始终防范最新的时尚对我们的诱惑。关于这个问题我可以说不多，但我不认为这个附录是这样做的适当地方。

4.13 有两种情况需要考虑：（a）所描绘的关系是某个关系变量R的一个示例值；（b）所描

绘的关系是某个关系表达式 $rx$ 的一个示例值，这里 $rx$ 不是一个简单的关系变量引用（关系变量引用基本上只是相关的关系变量名称），而是其他的东西。在情况（a）中，双下划线简单地表示已经为 $R$ 声明了一个主键 $PK$ ，并且相关的属性是 $PK$ 的一部分。在情况（b）中，你可以认为 $rx$ 是为一些临时关系变量 $R$ 定义的表达式（如果你喜欢，可以认为它是一个视图定义表达式而 $R$ 为相应的视图），那么双下划线表示原则上可以为 $R$ 声明主键 $PK$ 且相关的属性是 $PK$ 的一部分。

4.14 为了这个习题和下一个习题的目的，我假定如图4-1所示的关系是一个关系变量 $SPQ$ 的一个示例值。那么在这里是两个查询的Tutorial D表达方式（不是唯一的）：

a.

---

---

$$\left( \left( SPQ \text{ WHERE } SNO='S2' \right) \text{ UNGROUP } (PQ) \right) \{PNO\}$$

---

---

b.

---

---

$$\left( \left( SPQ \text{ UNGROUP } (PQ) \right) \text{ WHERE } PNO='P2' \right) \{SNO\}$$

---

---

观察，这些表达式中的第一个包括后跟一个



取消分组的限制，而第二个包括一个后跟一个限制（存在不对称性）的取消分组。注意：在此之前，**UNGROUP**（取消分组）操作还没有讨论过，但是，从前面的实例看，其语义应该是显而易见的。基本上，它是用来把一个带有**RVA**的关系映射到一个没有这样的属性的关系。（从“相反的方向”，也有**GROUP**操作，也就是说，把一个没有**RVA**的关系映射到一个有**RVA**的关系）有关的进一步讨论，请参见《SQL与关系数据库理论》。

4.15 在这里，我想为<relation assign>给出Tutorial D的部分语法，可能会有所帮助，这是在Tutorial D中的基本关系更新操作符。（语法类别的名称意味着要在直观上不言自明。）

---

---

```
<relation assign>
: =<relvar name> : =<relation exp>
|<insert>|<delete>|<update>
<insert>
: =INSERT<relvar name><relation exp>
<delete>
: =DELETE<relvar name><relation exp>
|DELETE<relvar name>[WHERE<boolean exp>]
<update>
: =UPDATE<relvar name>[WHERE<boolean exp>]:
{<attribute assign commalist>}
```

---

---

而一个<attribute assign>，如果所涉及的属性是关系值的，那么它基本上是一个<relation assign>（除了在那个<relation assign>中出现在目标<relvar name>的位置的相关<attribute name>），而这也正是我们进入的地方。那么这里是所需的更新的Tutorial D语句：

a.

---

---

```
INSERT SP RELATION{TUPLE{'S2', 'P5', 500}};
```

---

---

b.

---

---

```
UPDATE SPQ WHERE SNO='S2':  
{INSERT PQ RELATION{TUPLE{PNO'P5', QTY 500}}};
```

---

---

4.16 没有提供答案！但要注意，早在1995年，我正像其他人一样困惑，（但至少我在关于这个主题的书的后续版本中纠正了我的错误。）

[1]在我们的书《Database Explorations: Essays on The Third Manifesto and Related Topics》（Trafford, 2010）和其他地方。

[2]符号 $r!$ 读作“ $r$ 阶乘”并表示 $r \times (r-1) \times \dots \times 2 \times 1$ 的积。



## 第5章

5.1 对一个单独的关系的连接 $\text{JOIN}\{r\}$ ，就是 $r$ ，完全没有任何关系的连接 $\text{JOIN}\{\}$ ，是 $\text{TABLE\_DEE}$ （唯一的度为0，基数为1的关系）。进一步说明，请参见《SQL与关系数据库理论》。

5.2 请参见本章正文。

5.3 只有FD  $c$ 和 $d$ 是平凡的。所有8个FD  $a \sim h$ 都满足关系变量 $S$ 的当前值。除 $h$ 以外的全部FD都在关系变量 $S$ 中存在。FD  $a$ 、 $c$ 、 $e$ 和 $g$ 对于关系变量 $S$ 是不可约的，而FD  $b$ 、 $d$ 和 $f$ 是可约的。（对于 $h$ ，并不会出现不可约的问题，因为这个FD在关系变量中不存在。如果你不能马上掌握这一点，请检查FD不可约性的定义）。

5.4 希思定理（原始版本）指出，如果（a）关系 $r$ 具有标题 $H$ ，（b） $X$ 、 $Y$ 和 $Z$ 是 $H$ 的子集且它们的并集等于 $H$ ，且（c） $r$ 满足FD  $X \rightarrow Y$ ，那么（d） $r$ 等于其在 $XY$ 和 $XZ$ （其中 $XY$ 表示 $X$ 和 $Y$ 的并集，对于 $XZ$ 也类似）上的投影的连接。在下文中，我将用详尽的细节展示这个定理的证明。注意：“ $t \in r$ ”可以解读为“元组 $t$ 在关系 $r$ 中出现。”

首先，考虑可能最简单的情况，其中X、Y和Z是单例集合（即，每个集合只包含一个属性）。设所涉及的属性分别为A、B和C。现在，我们从习题3.2的答案中知道，通过分别在XY（= $\{A,B\}$ ）上取投影 $r_1$ 和在XZ（= $\{A,C\}$ ）上取投影 $r_2$ ，然后将 $r_1$ 和 $r_2$ 重新连接在一起， $r$ 的元组没有丢失。我现在反过来说明，此连接的每一个元组的确是 $r$ 的一个元组（换句话说，连接不会产生任何“假”元组）。设 $(a,b,c) \in \text{JOIN}\{r_1, r_2\}$ 。为了在连接中产生这样的一个元组，我们必须有 $(a,b) \in r_1$ 和 $(a,c) \in r_2$ 。因此，对于某个 $b'$ 和某个 $c'$ ，一定存在元组 $(a,b,c') \in r$ 和 $(a,b', c) \in r$ 。但因为 $r$ 满足 $\{A\} \rightarrow \{B\}$ ，所以 $b=b'$ ，并且 $(a,b,c) \in r$ 。

次简单的情况是，其中X、Y和Z不一定是单例集合，但它们两两不相交。在这种情况下，我们可以把构成X的属性作为单个复合属性（对于Y和Z也类似地处理），然后直接把上一段落的论证应用于此。

我们现在需要考虑如果X、Y和Z不是两两不相交，会发生什么情况。有三种情况需要考虑：X和Y相交，X和Z相交，以及Y和Z相交。

那么，首先设X和Y是相交的，但设X和Z是

不相交的，并设 $Y$ 和 $Z$ 是不相交的（因此 $Z=H-XY$ ）。现在回顾一下，如果满足 $X \rightarrow Y$ ，那么对于 $Y$ 的所有子集 $Y^-$ ，也满足 $X^+ \rightarrow Y^-$ ，因此，满足FD  $X \rightarrow Y-X$ 。但由前面的结果， $X$ 和 $Y-X$ 是不相交的，因此， $r$ 等于其在（a）在 $X$ 和 $Y-X$ 的并集和（b） $XZ$ 上的投影的连接。但是，（再次因为） $X$ 和 $Y-X$ 是不相交的，所以它们的连接等于 $XY$ 。因此，该定理也适用于这种情况，不失一般性，我们可以（并且我会）在证明的其余部分中假设 $X$ 和 $Y$ 是不相交的。

现在，设 $X$ 和 $Z$ 是相交的，但设 $Y$ 和 $Z$ 是不相交的。那么，根据前面的结果， $r$ 等于其在（a） $XY$ 和（b） $X$ 和 $Z-X$ 的并集上的投影的连接。但 $X$ 和 $Z-X$ 的并集等于 $XZ$ 。因此，该定理也适用于这种情况，不失一般性，我们可以（并且我会）在证明的其余部分中假设 $X$ 和 $Z$ 是不相交的。

现在，设 $Y$ 和 $Z$ 是相交的。设 $W=Z-Y$ 。因为 $r$ 满足FD  $X \rightarrow Y$ ，那么，它也满足FD  $X \rightarrow Y-W$ ，且 $Y-W$ 和 $Z$ 是不相交的。因此，根据前面的结果， $r$ 等于其在（a）在 $X$ 和 $Y-W$ 的并集和（b） $XZ$ 上的投影的连接。我在此得出一个很容易证明的引理（见下文），大意是，如果（a） $r_1$ 和 $r_2$ 分别为 $r$ 的投影，使得 $\text{JOIN}\{r_1, r_2\}=r$ ，（b） $H'$ 是 $H$ 的一个子集，但同时也是 $r_1$ 的标题的一个超集，并且

(c)  $r'$ 是 $r$ 在 $H'$ 上的投影, 那么 (d)  $\text{JOIN}\{r', r_2\}=r$ ; 换句话说, 不严格地说,  $r_1$ 可以用 $r_2$ 的属性任意扩展, 而不改变该连接的结果。从这个引理, 立即可以得出,  $r$ 等于其在 $XY$ 和 $XZ$ 上的投影的连接; 所以该引理也适用于这种情况。结论: 希思定理在所有可能的情况下都是有效的。

引理: 设 $r$ 具有标题 $H$ , 并设 $H$ 被划分为 $A$ 、 $B$ 、 $C$ 和 $D$ , 为简单起见, 假设这4个子集都不是空的。(附加习题: 扩充这个证明以覆盖这种假设不成立的情况。) 不失一般性, 我们可以把 $A$ 、 $B$ 、 $C$ 和 $D$ 都看作单独的属性。因此, 设 $r_1=r\{A,B\}$ 且 $r_2=r\{B,C,D\}$ , 并设 $(a,b) \in r_1$ 且 $(b,c,d) \in r_2$ 。由于 $r=\text{JOIN}\{r_1, r_2\}$ , 因此可以得出 $(a,b,c,d) \in r$ , 因此 $(a,b,c) \in r\{A,B,C\}$ 且 $(b,c,d) \in r\{B,C,D\}$ , 因此 $(a,b,c,d) \in \text{JOIN}\{r\{A,B,C\}, r\{B,C,D\}\}$ 。所需的结果如下—— $r\{B,C,D\}$ 是 $r_2$ , 并且 $r\{A,B,C\}$ 可以当作 $r'$ , 它具有 $H'=\{A,B,C\}$ 。引理结束。

希思定理的逆命题可以表述为, 如果关系 $r$ 等于其在 $XY$ 和 $XZ$ 上的投影的连接, 那么 $r$ 满足 $\text{FD } X \rightarrow Y$ 。此逆命题是假的。展示出反例足以说明这一点。所以考虑一个关系变量 $\text{CTX}$ , 它具有属性 $\text{CNO}$  (课程)、 $\text{TNO}$  (教师) 和 $\text{XNO}$  (教科书), 且谓词是课程 $\text{CNO}$ 可以由教师 $\text{TNO}$ 讲授

并使用教科书XNO。下面是这个关系变量的一个示例值：

CNO	TNO	XNO
C1	T1	X1
C1	T1	X2
C1	T2	X1
C1	T2	X2

此示例值等于其在 $\{CNO, TNO\}$ 和 $\{CNO, XNO\}$ 上的投影的连接，但它显然不能满足 $FD\{CNO\} \rightarrow \{TNO\}$ （或 $FD\{CNO\} \rightarrow \{XNO\}$ ，事实上）。注意：第12章详细地讨论了这个特殊的例子。

5.5 请参阅本章正文。

5.6 假设我们以一个属性为D、P、S、L、T、C的关系变量开始，其中这些属性以明显的方式（即英文单词首字母）对应谓词中的参数。那么在此关系变量中存在下面平凡的FD：

---


---

$$\begin{aligned}\{L\} &\rightarrow \{D, P, C, T\} \\ \{D, P, C\} &\rightarrow \{L, T\} \\ \{D, P, T\} &\rightarrow \{L, C\}\end{aligned}$$



$\{D,P,S\} \rightarrow \{L,C,T\}$

---

一个可能的BCNF关系变量集合是： 

---

SCHEDULE{L,D,P,C,T}

KEY{L}

KEY{D,P,C}

KEY{D,P,T}

STUDYING{S,L}

KEY{S,L}

---

需要注意的是，在这种分解中， $FD\{D,P,S\} \rightarrow \{L,C,T\}$ “丢失”了（见第6章）。

5.7 最简单的设计是：

---

EMP{ENO,ENAME,SALARY}

KEY{ENO}

PGMR{ENO,LANG}

KEY{ENO}

FOREIGN KEY{ENO}REFERENCES EMP

---

每一个员工都在EMP中有一个元组（并且EMP没有其他元组）。另外，碰巧是程序员的员工都在PGMR有一个元组（并且PGMR没有其他元组）。需要注意的是，EMP和PGMR的连

接给出了完整信息——员工数量、姓名、工资和语言技能（只适用于程序员）。

如果程序员可以有任意数量的语言技能，那么唯一显著的区别是关系变量PGMR将是“全键”（即，它的唯一键将会是{ENO,LANG}）。

5.8 是的，它们（当然）等价。

[\[1\]](#)附加习题：这些关系变量的谓词分别是什么？

## 第6章

### 6.1

---

```
CONSTRAINT.....  
COUNT (JOIN{TJ,TS}) =  
COUNT ( (JOIN{TJ,TS}) {S,J} ) ;
```

---

或者，使用习题4.8答案中的另一种约束风格来描述：

---

```
CONSTRAINT.....JOIN{TJ,TS}KEY {S,J};
```

---

6.2 设LT和CT分别是RX2A'在{CLASS,STATUS}和{CITY,STATUS}上的投影。则(a){CLASS}和{CITY}将成为RX2B'中的外键，分别引用LT和CT，并且(b)还将存在以下多关系变量约束：

---

```
CONSTRAINT.....WITH (LTX: =LT  RENAME{STATUS  
AS X},  
CTY: =CT RENAME{STATUS AS Y}) :  
AND (JOIN{RX2B', LTX,CTY}, X=Y) ;
```

---

### 6.3 给定FD中的第一个指

$\{\text{STREET}, \text{CITY}, \text{STATE}\}$  是一个键；第二个指此关系变量不属于BCNF。但是，如果我们用希思定理把它分解（在 $\text{FD}\{\text{ZIP}\} \rightarrow \{\text{CITY}, \text{STATE}\}$ 的基础上）为如下BCNF投影：

---

$\text{ZCT}\{\text{ZIP}, \text{CITY}, \text{STATE}\}$   
 $\text{KEY}\{\text{ZIP}\}$   
 $\text{ZR}\{\text{ZIP}, \text{STREET}\}$   
 $\text{KEY}\{\text{ZIP}, \text{STREET}\}$

---

那么我们就丢失了 $\text{FD}\{\text{STREET}, \text{CITY}, \text{STATE}\} \rightarrow \{\text{ZIP}\}$ 。其结果是，关系变量ZCT和ZR不能独立地更新。（附加习题：为ZCT和ZR设计一些示例值来说明这一点。）当然，如果我们不执行此分解，就会存在一定的冗余；具体地说，一个给定的邮政编码对应一个特定的城市和州这一事实将出现多次。但是，这种冗余会引起问题吗？由于一个城市和州的邮政编码不经常改变，答案是“有可能，但不是很频繁。”（另一方面，说邮政编码永远不会改变是不正确的。）

6.4 下面是RX1的一个不可约覆盖：

---

$\{\text{SNO}, \text{PNO}\} \rightarrow \{\text{QTY}\}$   
 $\{\text{SNO}\} \rightarrow \{\text{CITY}\}$   
 $\{\text{CITY}\} \rightarrow \{\text{STATUS}\}$

---

---

3NF过程得到{SNO,PNO,QTY}、  
{SNO,CITY}和{CITY,STATUS}。

接下来，RX3的一个不可约覆盖如下：

---

---

$\{SNO\} \rightarrow \{CLASS\}$   
 $\{CLASS\} \rightarrow \{CITY\}$   
 $\{CITY\} \rightarrow \{STATUS\}$

---

---

3NF过程得到{SNO,CLASS}、  
{CLASS,CITY}和{CITY,STATUS}。

最后RX2的一个不可约覆盖如下：

---

---

$\{SNO\} \rightarrow \{CLASS\}$   
 $\{SNO\} \rightarrow \{CITY\}$   
 $\{CLASS\} \rightarrow \{STATUS\}$   
 $\{CITY\} \rightarrow \{STATUS\}$

---

---

3NF过程得到{SNO CLASS,CITY}、  
{CLASS,STATUS}和{CITY,STATUS}。关于这个  
例子的有趣之处在于（如在本章正文中显示  
的），如果我们在FD{SNO}→{CLASS,CITY}的  
基础上分解，我们会得到{SNO,CLASS,CITY}和  
{CLASS,CITY,STATUS}作为3NF投影的标题，而

这不是我们从3NF过程中得到的。事实上，3NF过程的结果需要保持下面相当复杂的多关系变量约束：

---

---

CONSTRAINT.....JOIN{SLC,LT}=JOIN{SLC,CT};

---

---

（“对一个指定的供应商，类的状态=城市的状态”；这里的SLC、LT和CT分别是指RX2在{SNO,CLASS,CITY}{CLASS,STATUS}和{CITY,STATUS}上的投影）。

因此，这个例子说明了这一点，即虽然3NF过程肯定保证产生3NF的投影并且不会失去任何FD，但还是不应该过于盲目地遵循它。

注意：假设我们把关系变量LT和CT的状态属性命名成不同的，比如：

---

---

LT{CLASS,CLASS\_STATUS}  
CT{CITY,CITY\_STATUS}

---

---

那么“对于任何给定的供应商两个状态值必须是相等”的约束，可能是这样描述的：

---

---

CONSTRAINT.....IS\_EMPTY ( (JOIN{SLC,LT,CT}) )

WHERE CLASS\_STATUS $\neq$ CITY\_STATUS) ;

---

（如果关系 $r$ 是空的，Tutorial D表达式 $I_{S\_EMPTY}(r)$ 返回TRUE，否则返回FALSE）。  
另一种描述方法：

---

CONSTRAINT.....  
AND (JOIN{SLC,LT,CT},  
CLASS\_STATUS=CITY\_STATUS) ;

---

这个例子中的总体信息可能可以这样表达：  
特别是丢失或保持函数依赖这整件事，真的只是一个更为普遍的现象的一个特例。事实上，这应该是显而易见的，在一般情况下，如果我们从某个设计DBD1开始，并将它映射到某个逻辑上与它等价的设计DBD2，那么这个过程必然会涉及约束以及关系变量的一些重组。

6.5 假设：在任何一部给定的电影中，任何明星都不会扮演多个角色，任何电影都不会有超过一名导演。（这些假设是合理的吗？）FD：

---

$\{S,M\} \rightarrow \{R\}$   
 $\{M\} \rightarrow \{D,Y\}$   
 $\{S\} \rightarrow \{B\}$   
 $\{B\} \rightarrow \{Z,C\}$

$$\{Z,C\} \rightarrow \{H\}$$

---

$\{S,M\}$  是一个键。BCNF过程得到 $\{S,M,R\}$ 、 $\{M,D,Y\}$ 、 $\{S \ B\}$ 、 $\{B,C,Z\}$ 和 $\{Z,C,H\}$ 。没有丢失任何FD。



## 第7章

7.1 请参阅本章正文。

7.2 FD集合 $F$ 的闭包 $F^+$ 是 $F$ 中的FD蕴含的所有FD的集合。在发货关系变量 $SP$ 中存在的FD的集合的闭包在习题4.1的答案中给出。

7.3 当且仅当任意两个在 $X$ 上一致的元组，它们也在 $Y$ 上一致时，FD  $X \rightarrow Y$ 得到满足。（我故意用一种相当不严密的形式给出这个定义）。所以：

■如果两个元组在 $X$ 上一致，那么它们当然也在 $X$ 的每个子集 $Y$ 上一致，所以，自反律是合理的。

■如果两个元组在 $XZ$ 上一致，那么它们当然在 $Z$ 上一致，它们当然也在 $X$ 上一致，并且因此，如果满足 $X \rightarrow Y$ ，那么它们也在 $Y$ 上一致，因此它们在 $YZ$ 上一致，所以，增广律是合理的。

■如果两个元组在 $X$ 上一致且满足 $X \rightarrow Y$ ，那么它们在 $Y$ 上一致。如果它们在 $Y$ 上一致且满足 $Y \rightarrow Z$ ，那么它们也在 $Z$ 上一致，所以传递律是合理的。

7.4 请参阅本章正文。

7.5 设 $U$ 表示 $Z$ 和 $Y$ 的交集，并设 $V$ 表示 $Z$ 和 $Y$ 之间（按此顺序）的差集 $Z-Y$ 。那么：

1.  $X \rightarrow Y$ （已知）

2.  $Z \rightarrow W$ （已知）

3.  $X \rightarrow U$ （对1应用分解律）

4.  $XV \rightarrow UV$ （对3应用增广律）

5.  $XV \rightarrow Z$ （简化4）

6.  $XV \rightarrow W$ （对5和2应用传递律）

7.  $XV \rightarrow YW$ （对1和6应用复合律，证毕）

在这个证明中使用的规则已经在注释中列出了。下面的规则都是Darwen定理的特例：合并律、传递律和增广律。下面这个有用的规则同样也是：

■如果 $X \rightarrow Y$ 且 $XY \rightarrow Z$ ，则 $X \rightarrow Z$ 。

注意：此后者是有时称为伪传递性

(pseudotransitivity) 规则的一个特例，其一般形式看起来是这样的：

■如果 $X \rightarrow Y$ 且 $YW \rightarrow Z$ ，则 $XW \rightarrow Z$ 。

7.6 第一步是重写给定的FD集合，使得每一个FD都有一个单例的右侧：

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蕴含6，所以我们可以去掉6。

■根据增广律，8蕴含 $CF \rightarrow BC$ ，这与3一起根据传递律可得 $CF \rightarrow D$ ，所以我们可以去掉9。

■根据增广律，8蕴含 $ACF \rightarrow AB$ ，且根据增广律，11蕴含 $ACD \rightarrow ACF$ ，所以根据传递律可得 $ACD \rightarrow AB$ ，再根据分解律可得 $ACD \rightarrow B$ ，所以我们可以去掉4。

其余FD不可能进一步削减了，所以我们就得到以下不可约覆盖：

$AB \rightarrow C$

$C \rightarrow A$

$BC \rightarrow D$

$BE \rightarrow C$

$CE \rightarrow F$

$$CF \rightarrow B$$

$$D \rightarrow E$$

$$D \rightarrow F$$

另一种方法：

■2蕴含6，所以我们可以去掉6（同前）。

■根据增广律，2蕴含 $CD \rightarrow AD$ ，再次根据增广律，这蕴含 $CD \rightarrow ACD$ ，根据传递律，这与4一起蕴含 $CD \rightarrow B$ ，所以我们可以把4替换成 $CD \rightarrow B$ 。

■根据组合律，2和9蕴含 $CF \rightarrow AD$ ，根据增广律，这蕴含 $CF \rightarrow ADC$ ，根据传递律，这与（原始的）4一起蕴含 $CF \rightarrow B$ ，所以我们可以去掉8。

其余FD不可能进一步削减了，所以我们就得到了以下不可约覆盖：

$$AB \rightarrow C$$

$$C \rightarrow A$$

$$BC \rightarrow D$$

$CD \rightarrow B$

$BE \rightarrow C$

$CE \rightarrow F$

$CF \rightarrow D$

$D \rightarrow E$

$D \rightarrow F$

注意，因此，原来的FD集（至少）有两个不同的不可约覆盖。还要注意的，这两个覆盖有不同的基数。

7.7 是的，它是。证明这一结果最简单的方法是计算ACF集合的闭包 $ACF^+$ ，它原来是整个集合ABCDEFG。另外，我们也可以应用阿姆斯特朗公理和本章正文讨论的其他规则，如下：

1.  $A \rightarrow B$ （已知）

2.  $ACF \rightarrow BCF$ （对1应用增广律）

3.  $BC \rightarrow E$ （已知）

4.  $BCF \rightarrow EF$  (对3应用增广律)

5.  $ACF \rightarrow EF$  (对2和4应用传递律)

6.  $ACF \rightarrow AEF$  (对5应用增广律)

7.  $AEF \rightarrow G$  (已知)

8.  $ACF \rightarrow G$  (对6和7应用传递律)

9.  $BC \rightarrow DE$  (已知)

10.  $BC \rightarrow D$  (对9应用分解律)

11.  $BCF \rightarrow DF$  (对10应用增广律)

12.  $BCF \rightarrow D$  (对11应用分解律)

13.  $ACF \rightarrow D$  (对2和12应用传递律)

14.  $ACF \rightarrow DG$  (对7和13应用复合律)

7.8 设第一个集合的FD的编号如下:

1.  $A \rightarrow B$

2.  $AB \rightarrow C$

$$3.D \rightarrow AC$$

$$4.D \rightarrow E$$

现在，3可以替换为：

$$3.D \rightarrow A \text{ 和 } D \rightarrow C$$

接下来，1和2一起蕴含（参见习题7.5的答案接近最后部分“有用的规则”中提到的内容），于是2可以被替换为：

$$2.A \rightarrow C$$

但是，现在我们有 $D \rightarrow A$ 和 $A \rightarrow C$ ，所以根据传递律可得，蕴含 $D \rightarrow C$ ，它可以去掉，因此得到：

$$3.D \rightarrow A$$

因此第一个FD集合等价于下列的不可约覆盖：

$$A \rightarrow B$$

$$A \rightarrow C$$



$D \rightarrow A$

$D \rightarrow E$

第二组给定的FD集合

$A \rightarrow BC$

$D \rightarrow AE$

显然也等价于此相同的不可约覆盖。因此这两个给定的集合是等价的。

7.9 这个集合明显是可约的，因为 $C \rightarrow J$ 和 $CJ \rightarrow I$ 一起蕴含 $C \rightarrow I$ 。至于键：一个很明显的超键是 $ABCDGJ$ （给定FD的左侧提到的所有属性的组合）。因为 $C \rightarrow J$ ，所以我们可以从这个集合去掉 $J$ ，并且因为 $AB \rightarrow G$ ，所以我们可以去掉 $G$ 。由于 $A$ 、 $B$ 、 $C$ 、 $D$ 中的任意一个都不会出现在任何给定的FD的右侧，因此， $ABCD$ 是一个键。

## 第8章

8.1 这个习题（故意）用不同的词重复习题4.6。这种说法是不正确的，正如先前习题的答案所示。

8.2 下面是我自己对这些观点的一些意见。

■“如果数据是反规范化的，许多查询会更容易理解”：我怀疑这里的理解实际上应该是编写（例如，理解查询“获取所有供应商的详细信息”与数据库是如何设计的完全无关）。如果我的怀疑是正确的，那么这个说法可能是有效的。但相反的说法也是有效的！如果数据没有反规范化，许多查询会更容易编写，正如本章正文阐述的。

■采访者提出，非规范化可能会导致完整性问题，并且会降低支持未预料到的查询的灵活性。我同意这些看法。

■“规范化和它强调的消除冗余存储，是一个纯粹的事务处理问题”：规范化是减少冗余，而不是减少冗余存储，但我想顾问混淆了两者可能是可以原谅的，根据当今最广泛可用的实现。但它肯定不是“一个事务处理问题”！正如我在第1

章说过的，在一般情况下，当我们做数据库设计以及特别是当我们做规范化时，我们主要关注数据是什么，而不是它将如何使用。

■“当用户查看数据时，他们以一个冗余的形式看到它”：有时他们这样做，但有时他们不这样做。但是，即使他们这样做，这也不是一个非规范化设计的论据；例如，通过常规视图机制的手段，可以给用户提供一个感觉上是非规范化的数据，而底层的数据库仍然是规范化的。

■“为了将数据转换成一种对用户有用的形式……”：这只是一个有偏见的说法。

■“[连接]实质上是为了更好地利用数据而将其动态反规范化的一种方式”，用户可能会想要进行动态的连接，但他们一般没有不能静态地做连接（即提前做连接）的理由。而且我相信，假如有一个架构良好的DBMS，他们往往会那样做。[\[1\]](#)一个连接的结果必须始终反规范化的建议也是不正确的。“更强的易用性”是另一种有偏见的言论。

■“[用户]无法容忍连接的时间和成本”：连接不一定是费时或昂贵的。再次强调，这取决于实现。

■“为了解决这个问题，许多公司在越来越多的决策支持数据库中复制数据，它代表数据的反规范化视图”：这可能是真实的，但如果它是真实的，那么它是对当前实现的控诉，而不是一个赞成非规范化的论据。

8.3 首先，代理键和元组标识符是不一样的东西。一方面（不言自明），代理键标识的是实体而元组标识符标识的是元组，且实体和元组之间一定没有一一对应之类的东西。（尤其是考虑到派生元组，例如，某些查询结果中的元组。事实上，无论如何，派生的元组是否将有元组标识符是完全不明确的。）此外，元组标识通常有性能的含义，但代理键没有这种含义（通常认为通过一个元组的元组标识符访问它是快速的，但这种看法不适用于代理键）。此外，元组标识通常是对用户隐藏的，但根据信息原则（见习题2.1），代理键绝对不能对用户隐藏，换句话说，可能（且最好）不能够在一个数据库关系变量中存储元组标识符，同时它肯定（且最好）能够在一个数据库关系变量中存储代理键。一言以蔽之：代理键和逻辑设计有关而元组标识符和物理设计有关。

代理键是一个好主意吗？首先请注意，关系模型在这个问题上没什么可说的，其实，正如一

般的设计业务，是否使用代理键与如何应用关系模型有关，而与关系模型无关。

这就是说，我也不得不说代理键是好还是坏这个问题一点都不简单。双方都有强有力的论据：实际上，非常多，在这里我不可能公平地判断它们（虽然我在第15章总结了其中一些）。有关详细信息，请参阅我的书《Relational Database Writings 1989-1991》（Addison-Wesley, 1992）中的论文“Composite Keys”（“复合键”）。注：该论文称为“复合键”是因为在实践中，特别是，在已有键和相应的外键是复合键的情况下，代理键最有可能有用。

8.4 我用SQL显示解决方案，只是为了做出一个改变。用第二个设计的方式定义第一个设计（概要）：

---

```
SELECT DISTINCT EX.ENO,  
  (SELECT PAY  
   FROM EMP AS EY  
   WHERE EY.ENO=EX.ENO  
   AND MONTH='Jan') AS JAN_PAY,  
.....  
  (SELECT PAY  
   FROM EMP AS EY  
   WHERE EY.ENO=EX.ENO  
   AND MONTH='Dec') AS DEC_PAY
```

用第一个设计的方式定义第二个设计（再次以概要的方式显示）：

---

```
SELECT ENO, 'Jan'AS MONTH,JAN_PAY AS PAY FROM  
EMP  
UNION  
.....  
UNION  
SELECT ENO, 'Dec'AS MONTH,DEC_PAY AS PAY FROM  
EMP
```

---

[1]我在这里首先想到的是，一个使用 TransRelational™模型的设施实现的DBMS（并且类似的言论适用于整本书中所有使用“良好架构”这句话的地方）。你可以在我的书《数据库系统导论（原书第8版）》找到该模型的一个初步的（不完整）说明，并在我的书《Go Faster! The TransRelational™Approach to DBMS Implementation》（Ventus, 2002, 2011）找到一个更全面的论述。

## 第9章

9.1 连接SP和PJ在本章正文中讨论过了。连接PJ和JS产生假的元组 (S2, P2, J1)，然后消除它，因为SP中没有 (S2, P2) 元组。连接JS和SP产生假的元组 (S2, P2, J2)，然后消除它，因为在PJ中没有 (P2, J2) 元组。

### 9.2

---

```
CONSTRAINT.....SPJ=JOIN{SPJ{SNO,PNO},  
SPJ{PNO,JNO},  
SPJ{JNO,SNO}};
```

---

9.3 首先，我们将大概需要三个关系变量来分别表示销售代表、销售区域和产品，如下所示：

---

```
R{RNO, .....}KEY{RNO}  
A{ANO, .....}KEY{ANO}  
P{PNO, .....}KEY{PNO}
```

---

现在，如果代表r负责区域a，且产品p在区域a出售，且代表r销售产品p，那么代表r在区域a销售产品p。这是一个3路循环规则。因此，如果我

们有一个看起来像这样的关系变量RAP:

---

$$\text{RAP}\{\text{RNO},\text{ANO},\text{PNO}\}\text{KEY}\{\text{RNO},\text{ANO},\text{PNO}\}$$

---

（带有明显的谓词）——那么在该关系变量中将存在下面的JD:

---

$$\odot\{\{\text{RNO},\text{ANO}\}, \{\text{ANO},\text{PNO}\}, \{\text{PNO},\text{RNO}\}\}$$

---

因此，该关系变量存在冗余。所以，让我们用三个二元投影来替代它:

---

$$\begin{aligned}\text{RA}\{\text{RNO},\text{ANO}\}\text{KEY}\{\text{RNO},\text{ANO}\} \\ \text{AP}\{\text{ANO},\text{PNO}\}\text{KEY}\{\text{ANO},\text{PNO}\} \\ \text{PR}\{\text{PNO},\text{RNO}\}\text{KEY}\{\text{PNO},\text{RNO}\}\end{aligned}$$

---

（现在有几个相等的依赖关系需要说明和执行，如，投影 $R\{\text{RNO}\}$ 、 $RA\{\text{RNO}\}$ 和 $PR\{\text{RNO}\}$ 必须始终是相等的，但这些细节很简单，这里省略它们。）

接下来，每名代表负责在一个或多个区域销售，每个区域有一名或多名负责销售的代表。但此信息已包含在关系变量RA中，所以它不是必要



的。同样，关系变量AP处理每个区域有一个或多个产品出售且每个产品在一个或多个区域出售的事实，并且关系变量PR处理每个产品都有一名或多名负责销售的代表和每名代表负责一个或多个产品的销售的事实。但是，请注意，不需要告知用户，RA、AP和PR的连接不包括任何“连接陷阱”（即，存在3路循环规则）。让我们来探讨这一点。首先，RA、AP和PR的谓词如下。

■RA：代表RNO负责区域ANO的销售。

■AP：产品PNO在区域ANO出售。

■PR：产品PNO是代表RNO销售的。

请注意，顺便说一句，一个架构良好的DBMS（遗憾的是，据我所知，当今市场上还没有）将使得设计人员能够告诉它这些谓词。注意：当然，告诉DBMS这个谓词将也等于告诉用户。所不同的是，后者可以用非形式的方法做（事实上，在当今的系统中，只能用非形式的方法做），但前者，如果完全能做到它，将必须用形式的方法做（见第15章）。

回到3路规则。显然，设计师不能只告诉用户关系变量RA、AP和PR的连接等于关系变量

RAP，因为分解后关系变量RAP不再存在。但是，我们可以把那个连接定义为一个视图（或“虚拟关系变量”）：

---

```
VAR RAP VIRTUAL (JOIN{RA,AP,PR})  
KEY {RNO,ANO,PNO};
```

---

然后同样架构良好的DBMS将可以推断出以下内容作为视图RAP的一个谓词：

代表RNO负责区域ANO的销售且产品PNO在区域ANO销售且产品PNO是代表RNO销售的。

但是，这个谓词没有完全表达真相（它没有捕获3路循环规则）。因此，在理想情况下，设计师应该有一个方法来告诉DBMS（以及用户），谓词实际上如下所示<sup>[1]</sup>：

代表RNO负责区域ANO的销售且产品PNO在区域ANO销售且产品PNO是代表RNO销售的。

且

代表RNO在区域ANO销售产品PNO。

请注意，这后一个谓词强于前一个谓词，如

果某个 (RNO,PNO,ANO) 三元组满足后者, 那么它一定满足前者。

#### 9.4 没有提供答案。

[1]因此, 这是用户 (或在这个案例中, 设计者) 肯定比系统知道更多东西的那些情况之一。

## 第10章

10.1 a.不正确（见本章正文中在讨论关系变量SPJ时举的一个反例）。b.不正确（实际上，如习题4.6的答案所示，二元关系变量甚至不一定属于BCNF或2NF）。c.不正确（参阅第13章）。d.不正确（也参阅第13章）。e.见本章正文。f.不正确（请参阅第9章中关系变量CTXD的一个反例，还可参阅第15章）。

10.2 请参阅本章正文。

10.3 首先，我假设，任何JD都没有任何重复的分量，因为否则JD的数量从字面上看是无限的。其次，假设关系变量SP属于5NF，其实还属于6NF，虽然我们还没有讨论6NF（见第13章），但我至少可以说，如果某个关系变量属于6NF，那么在该关系变量中存在的所有JD都将是平凡的。所以，问题就变成了：在关系变量SP中有多少平凡的JD？那么，所有这些JD都具有 $\{H, X_1, \dots, X_n\}$ 的形式，其中H表示整个标题且 $\{X_1, \dots, X_n\}$ 是H的真子集构成的一个集合（可能是空集）。由于H的度为3，它有8个子集，其中除一个外都是真子集。其元素是规定的七种元素的一个集合的某个子集的不同集合共有多少个呢？根本没有任何元素的这种集合，有1

个；只有一个元素的这种集合，有7个；更一般地，具有*i*个元素（*i*=0, 1, ....., 7）这样的集合，有“7选*i*”个。<sup>[1]</sup>因此，H的真子集（构成的）集合的总个数=（7选0）+（7选1）+（7选2）+ .....+（7选7）=1+7+21+35+35+21+7+1=128。因此，在关系变量SP中存在128个平凡的JD。注意：在这128个JD中，有64个包含一个空分量，可以合理地忽略它，例如， $JD \odot \{H, \{\}\}$ 与 $\odot \{H\}$ 显然是等效的，<sup>[2]</sup>从而总计数降低到了64。

#### 10.4 请参阅本章正文。

10.5 平凡的JD的定义，请参阅本章正文。由于一个FD不是一个JD，但仅仅蕴含一个JD，因此一个平凡的FD不是一个平凡的JD的特例。不过，反过来，被一个平凡的FD所蕴含的JD本身的确是平凡的。例如，在供应商关系变量S中存在平凡的FD $\{CITY, STATUS\} \rightarrow \{STATUS\}$ 。因此，应用希思定理，我们看到了在S中存在平凡的 $JD \odot \{AB, AC\}$ ，这里A是 $\{CITY, STATUS\}$ ，B是 $\{STATUS\}$ ，并且C是 $\{SNO, SNAME\}$ （并因此AC是整个标题）。

10.6 一个元组强迫的JD的一个例子，请参阅在本章正文的SPJ例子。至于不是元组强迫的JD的例子，考虑，例如，在关系变量S中存在的

JD $\bowtie$ { {SNO,SNAME,CITY}, {CITY,STATUS} },  
(注意, 它之所以不是元组强迫的JD, 正是因为它有一个分量是相关关系变量的一个超键)。

10.7 可以通过对在本章正文中给出的关系变量SPJ进行系统地替换得到这些例子。替换方法是把供应商的号码替换为RNO值, 零件号码替换为ANO值, 工程号码替换为PNO值。没有提供进一步的答案。

10.8 很明显, 我不知道你是否有任何看法, 但我肯定有。不过, 我认为在这里宣扬我的看法是不礼貌的, 所以我不会把它们写出来。

[1] 在一般情况下, 表达式“n选r”表示从一个包含n个元素的集合中挑选r个元素的方法的种数。

[2] 证明: 对于所有的关系r,  $\text{JOIN}\{r\{H\}, r\{\}\} = \text{JOIN}\{r\} = r$ 。

## 第11章

11.1 哪些JD是平凡的？没有。哪些包括无关的分量呢？i、k和l。哪些蕴含其他JD呢？a蕴含b；d蕴含g和h；e蕴含g；f蕴含h和i；j蕴含k。哪些JD对是等价的？没有一个。图1-1中的示例值满足哪些呢？a、b、c、d、e、g和l。哪些在关系变量P中存在？a、b、c、e和l。哪些是不可约的？a、b、c和e。

### 11.2

a.根据希思定理，答案显然是肯定的（设X为A,Y为BC,Z为D）。但是让我们看看我们是否能用追逐算法证明这个结果。前提元组如下：

---

---

x1	y12	y13	x4
x1	x2	x3	y24

---

---

从FD  $A \rightarrow B$ ，我们得出 $y12=x2$ ，从FD  $A \rightarrow C$ ，我们得出 $y13=x3$ 。进行替换：

---

---

x1	x2	x3	x4
x1	x2	x3	y24

---

---

现在，我们有一个全部由x组成的元组，这就得出了所需的结果。给定的JD确实由给定的FD得出。

b.前提元组如下：

---

x1 x2 y13 y14  
y21 x2 x3 y24  
y31 y32 x3 x4

---

FD蕴含 $y_{24}=x_4$ 和 $y_{13}=x_3$ 。进行替换：

---

x1 x2 x3 y14  
y21 x2 x3 x4  
y31 y32 x3 x4

---

FD  $C \rightarrow D$ 现在蕴含 $y_{14}=x_4$ ；进行替换为我们提供了一个全部由x组成的元组，因此得出结果如下：给定的JD确实由给定的FD得出。需要注意的是，在这个例子中，我们不得不在追逐中使用其中一个FD两次。也要注意，我们可以应用希思定理两次得到相同的结果：FD  $C \rightarrow D$ 蕴含 $JD \odot \{CD, CAB\}$ ，由于FD  $B \rightarrow C$ ，这反过来又蕴含 $JD \odot \{CD, BC, BA\}$ 。

c.我把它留给你来展现这里的答案是否定



的。

d.前提元组如下：

---

---

x1 x2 y13 y14  
y21 x2 x3 y24  
y31 y32 x3 x4

---

---

对具有一个共同的B值（即x2）的元组应用  
 $JD \otimes \{BC, ABD\}$ 产生下列元组：

---

---

x1 x2 x3 y24  
y21 x2 y13 y14

---

---

我们没有得到一个全部由x组成的元组，因此这个“目标”JD并非由给定的JD得出，事实上，我们现在有一个满足后一个JD但不满足前一个JD的示例关系（由5个元组组成）。

11.3 下面首先是（b）部分的扩展定理的一个证明：

1.  $X \rightarrow Y$ （已知）

2.  $XZ \rightarrow YZ$ （增广律）

3.  $XZ \rightarrow XZ$  (自含律)

4.  $XZ \rightarrow XYZ$  (对2和3应用合并律)

因此,  $XZ$ 是 $R$ 的一个超键。

至于逆命题, 假设关系变量 $R$ 包含下列元组:

---

x	y1	z1
x	y2	z2

---

由于 $JD \bowtie \{XY, XZ\}$ , 下面的元组必须也出现:

---

x	y1	z2
x	y2	z1

---

但 $XZ$ 是一个超键, 所以 $XZ \rightarrow Y$ 成立, 所以 $y1=y2$ , 因此 $X \rightarrow Y$ 成立。

11.4 这个习题将在第14章中进一步讨论, 但我在这里先给出一个初步的讨论。首先, 假设进行了这样的分解(即, 在第二个 $JD$ 的基础上)。设这样得到的投影以明显的方式标记为

SNC和CTN。那么SNC和CTN在{SNAME,CITY}上的投影显然是相等的，也就是说，下面的相等依赖（见第13章）存在：

---

---

CONSTRAINT.....

$SNC\{SNAME,CITY\}=CTN\{SNAME,CITY\};$

---

---

且因此关系变量SNC和CTN存在冗余。

进一步观察， $FD\{CITY\} \rightarrow \{STATUS\}$ 在CTN中存在。因此，根据希思定理，我们可以把CTN分解成其分别在{CITY,STATUS}和{CITY,SNAME}上的投影CT和CN。因此，JD

---

---

$\bowtie\{\{SNO,SNAME,CITY\}, \{CITY,STATUS,SNAME\}\}$

---

---

蕴含JD

---

---

$\bowtie\{\{SNO,SNAME,CITY\}, \{CITY,STATUS\}, \{CITY,SNAME\}\}$

---

---

然而，在后一个JD中，{CITY,SNAME}分量显然是无关的，因为它是{SNO,SNAME,CITY}分量的一个真子集，因此它可以去掉，而不造成显

著损失。（当然，事实上，后一个JD与原来的习题中给出的那两个JD中的第一个相同。）

## 第12章

12.1 当然，（a）本章正文中的关系变量CTX就是一个例子，但如果你能从自己的工作环境中想出一个例子将会更好。（b）设C是一个特定的集合，设关系变量 $R\{A,B\}$ 当且仅当a和b两者都是C的成员时，元组（a,b）出现在R中。那么R等于其投影 $R\{A\}$ 和 $R\{B\}$ 的笛卡儿积，因此，它服从 $JD \bowtie \{A,B\}$ ，等价地，它服从以下MVD：

---

$$\{\} \twoheadrightarrow A|B$$

---

这些MVD不是平凡的，因为它们肯定不会在所有的二元关系变量中存在，而且它们也不被超键蕴含（R中唯一的键是整个标题）。因此R不属于4NF。然而，它肯定属于BCNF，因为它是“全键”。

### 12.2 可能的写法：

---

a.CONSTRAINT.....CTX=JOIN{CTX{CNO, TNO},  
CTX{CNO,XNO}};  
b.CONSTRAINT.....CTXD{CNO, TNO, XNO}=  
JOIN{CTXD{CNO,TNO}, CTXD{CNO,XNO}};

---

12.3 (a) 假设CTX的当前值如图12-1所示。那么在所示的4个元组中，没有一个可以孤立地删除：这是删除异常。(b) 假设CTX的当前值只包含如图12-1所示的“前两个”元组。那么，所示的“第三个”和“第四个”元组都不可以孤立地插入：这是插入异常。

12.4 来自第9章的关系变量SPJ就是一个例子（在此关系变量中根本不存在除了平凡的MVD以外的MVD，所以此关系变量肯定属于4NF）。

12.5 可能会认为以下对这样明显的一点的证明是很重大的灾难：设所涉及的投影是 $R'$ 。那么当且仅当 $R'$ 的每两个元组 $t_1'$ 与 $t_2'$ ，只要具有相同的 $X$ 值，它们也具有相同的 $Y$ 值时，FD  $X \rightarrow Y$ 在 $R'$ 中存在。设 $T_1$ 和 $T_2$ 分别是派生出 $t_1'$ 的 $R$ 中的元组集合和派生出 $t_2'$ 的 $R$ 中的元组集合。根据投影的定义， $T_1$ 中的每一个元组 $t_1$ 具有与 $t_1'$ 相同的 $X$ 和 $Y$ 值，同样地， $T_2$ 中的每一个元组 $t_2$ 具有与 $t_2'$ 相同的 $X$ 和 $Y$ 值。因此， $R$ 的元组 $t_1$ 和 $t_2$ ，只要具有相同的 $X$ 值，它们也具有相同的 $Y$ 值；所以FD  $X \rightarrow Y$ 在 $R$ 中存在，由此进一步可得，当且仅当 $X \rightarrow Y$ 在 $R$ 中存在时，它在 $R'$ 中存在。

12.6 可以立即从希思定理得出这个结果：如果 $R$ 服从FD  $X \rightarrow Y$ ，它也服从JD  $\bowtie \{XY, XZ\}$ ，其

中Z是R的“其他”属性，并因此它服从MVD  $X \twoheadrightarrow Y|Z$ 。

12.7 当且仅当 $XY=H$ 或 $XZ=H$ 时， $JD \bowtie \{XY, XZ\}$ 是平凡的。如果 $XY=H$ ，那么我们得出情况（b）。如果 $XZ=H$ ，则 $Z=H-X$ ，而根据定义， $Z=H-X-Y$ ，所以Y是X的一个子集，那么我们得出情况（a）。

12.8 此规则等于说：如果我们一开始的一个关系变量具有两个或两个以上独立的关系值属性（RVA），并且我们要消除它们，我们通常想要这么做但不总是这样（见习题4.11的答案）——那么我们应该做的第一件事是把这些RVA分离出来。使用这个习题的符号，这一步我们将得到分别具有标题XY和XZ的关系变量。接下来的事情，我们要做的是对这些关系变量中的每个取消分组RVA。假设在Y和Z中的关系分别具有标题A和B，那么从取消分组得到的那些关系变量将分别具有标题XA和XB。<sup>[1]</sup>现在用通常的方法规范化这些关系变量，把它们替换为BCNF投影。那么，这些BCNF投影会“自动”属于4NF。换言之，如果遵循前述过程，导致关系变量不属于4NF的MVD应该不会在实践中出现。

顺便说一下，有个有趣的事情值得注意，

Codd在他1970年著名的论文（见附录C）中举了一个例子，他实际上在其中遵循了上述过程，并且他在次年另一篇论文（“Normalized Data Base Structure: A Brief Tutorial,”Proc.1971 ACM SIGFIDET Workshop on Data Description,Access,and Control,San Diego,Calif., November 11th-12th, 1971，再次见附录C）中再次简要地论及它，但我不认为他曾经再次提到它，至少不会以书面形式（也许是因为它如此显而易见）。

注意：如果你发现前面的讨论过于抽象，那么设R是一个具有标题{CNO,T,X}的关系变量，其中T和X是关系值并分别包含标题{TNO}和{XNO}的关系。我们分离这些RVA可得到分别具有标题{CNO,T}和{CNO,X}的关系变量。我们随后取消分组可得到分别具有标题{CNO,TNO}和{CNO,XNO}的关系变量，当然，在CTX例子中这恰恰是我们所希望得到的。

12.9 首先，我们将大概需要三个关系变量来表示代表、区域和产品，它们分别为：

---

R{RNO, .....}KEY{RNO}  
A{ANO, .....}KEY{ANO}  
P{PNO, .....}KEY{PNO}

---



---

接下来，我们可以用下列关系变量表达  
(a) 销售代表和销售区域之间的关系，及 (b)  
销售代表和产品之间的关系：

---

$RA \{RNO, ANO\} \text{KEY} \{RNO, ANO\}$   
 $RP \{RNO, PNO\} \text{KEY} \{RNO, PNO\}$

---

每一件产品都在每个区域销售。因此，如果我们引入一个关系变量

---

$AP \{ANO, PNO\} \text{KEY} \{ANO, PNO\}$

---

代表区域和产品之间的关系，那么我们有以下约束：

---

$\text{CONSTRAINT C1 } AP = \text{JOIN} \{A \{ANO\}, P \{PNO\}\};$

---

(这里的连接实际上是一个笛卡儿积)。注意，这个约束意味着AP不属于4NF。事实上，AP没有给我们任何我们无法从其他关系变量获得的信息。确切地说，以下EQD存在：

---

$AP \{ANO\} = A \{ANO\}$

$$AP\{PNO\}=P\{PNO\}$$

---

但是，此刻让我们假设，关系变量AP无论如何在我们的设计中。

在同一地区没有两个代表销售相同的产品。换句话说，给定一个{ANO,PNO}组合，只有一个负责的销售代表，RNO，并且因此我们可以引入一个关系变量：

---

$$APR\{ANO,PNO,RNO\}KEY\{ANO,PNO\}$$

---

其中（来明确地说明FD），

---

$$\{ANO,PNO\}\rightarrow\{RNO\}$$

---

（{ANO,PNO}是一个键这一规格说明足以表达这个FD）。然而，关系变量RA、RP和AP现在全是多余的，因为它们都是APR的投影，因此，它们可以去掉。我们现在需要用约束C2来代替约束C1：

---

CONSTRAINT C2 APR{ANO,PNO}=JOIN{A{ANO},  
P{PNO}};

---

此约束必须明确和单独说明，因为它不是“由键蕴含的”。

此外，由于每一个代表在他负责的所有地区销售他负责的所有产品，因此我们在关系变量APR上有额外的约束C3：

---

---

$$\{RNO\} \rightarrow \rightarrow \{ANO\} || \{PNO\}$$

---

---

（这些MVD是非平凡的，且不是由键蕴含的，因此关系变量APR并不属于4NF）。[\[2\]](#)

同样，这个约束必须明确和单独说明。

因此，最终的设计包括关系变量R、A、P和APR，以及约束C2和C3（事实上这两者都再次是相等依赖）：

---

---

CONSTRAINT	C2	$APR\{ANO,PNO\} = JOIN\{A\{ANO\}, P\{PNO\}\};$
CONSTRAINT	C3	$APR = JOIN\{APR\{RNO,ANO\}, APR\{RNO,PNO\}\};$

---

---

（从APR到其他三个关系变量也有一些外键约束，但它们在细节上很简单，我在这里忽略它

们。)

这个习题很好地说明了一点，即在一般情况下，规范化可能会足以代表一个给定的问题的一些语义方面（基本上是由键蕴含的FD、MVD和JD），但其他方面的附加约束可能需要明确地陈述。这也说明了另一点，规范化可能并非“在所有情况下”总是可取的（关系变量APR属于BCNF，但不属于4NF）。

作为一个附加习题，你可能要考虑对于当前的问题，一个包括RVA的设计是否可能是适当的。这样的设计是否可能意味着上一段落的某些意见不再适用？

12.10 这里要注意的第一点是，MVD  $A \twoheadrightarrow B|C$  和  $A \twoheadrightarrow C|D$  分别没有提及属性D和B。但难道我没有说过，对于给定的MVD  $X \twoheadrightarrow Y|Z$  的通用对，X、Y和Z的并集必须等于标题吗？嗯，是的，我说过，但我现在必须解释说，我们也允许使用一个特定的速记符号，正如本习题中所示。为了定性，让我们专注于表达式  $A \twoheadrightarrow B|C$ 。根据定义，这意味着  $A \twoheadrightarrow B$  和  $A \twoheadrightarrow C$ ；以及  $A \twoheadrightarrow B$  蕴含  $A \twoheadrightarrow CD$ ，以及  $A \twoheadrightarrow C$  蕴含  $A \twoheadrightarrow BD$ 。此外，由于A、B、C和D是单个属性，因此它们不相交，MVD的分解律允

许我们可以无论从 $A \twoheadrightarrow CD$ 还是从 $A \twoheadrightarrow BD$ 都能推断出 $A \twoheadrightarrow D$ 。总而言之，我们可以看到， $A \twoheadrightarrow B|C$ 是 $A \twoheadrightarrow B|CD$ 和 $A \twoheadrightarrow BD|C$ 之一或两者的一个速记。而且，鉴于这种状况，我们采用一个速记，根据它， $A \twoheadrightarrow B|CD$ 和 $A \twoheadrightarrow BD|C$ 都可以写成这样： $A \twoheadrightarrow B|C|D$ ——且反过来也可以认为后面的一个表达式是 $A \twoheadrightarrow B, A \twoheadrightarrow C$ 和 $A \twoheadrightarrow D$ 这三个MVD的结合的简写。

现在，让我们尝试追逐。这里是 $A \twoheadrightarrow C|D$ 的前提元组，正如我们已经看到的，这个MVD等价于 $A \twoheadrightarrow BC|D$ ：

---

---

x1	x2	x3	y14
x1	y22	y23	x4

---

---

应用 $A \twoheadrightarrow B|CD$ 生成：

---

---

x1	x2	y23	x4
x1	y22	x3	y14

---

---

应用 $B \rightarrow D$ 得出了 $y14=x4$ 。进行替换：

---

---

x1	x2	x3	x4
x1	y22	y23	x4

x1 x2 y23 x4

x1 y22 x3 x4

---

现在，我们有一个全部由x组成的元组，所以给定的依赖确实蕴含着目标JD。

[1]如果在A或B中有任何属性与X的某个属性具有相同名称，我们可能会首先做一些属性重命名操作。

[2]因此，注意，关系变量APR揭穿了另一种流行的误解：即，由一个单独的键和一个单独的非键属性组成的关系变量必然属于4NF。也参见本附录后面的习题13.10的答案。

## 第13章

13.1 参见图13-1。

13.2 请参阅本章正文。

13.3 键和FD的不可约性，以及与2NF相关的FD不可约性，都在第4章中讨论；FD不可约性在第5章中进一步讨论。不可约覆盖在第6章中讨论。不可约JD在第11章中讨论。不可约（即，6NF）关系变量和相关的“不可约的事实”的概念在本章（即，第13章）正文进行了讨论。

13.4 请参阅本章正文。

13.5 我想到的重点是，拥有某种“主”关系变量，其主要目的只是记录目前在数据库中表示的所有零件的零件编号，这样可能是不错的。如果我们称为关系变量P，那么在关系变量P和关系变量PN、PL、PW和PC中的每个在{PNO}上的投影之间就会存在EQD（而不是PN在{PNO}上的投影和PL、PW和PC在{PNO}上的投影之间随意存在的EQD，事实上，拥有主关系变量的一个好处正是它避免了轻微的随意性）。

此外，假设每一个零件总是有一个已知的名

称和重量，但并不一定有一个已知的颜色或城市。那么，我们可以把关系变量P、PN、PW组合起来，用该组合（我还是会称为P）作为主关系变量，并把以前需要的EQD替换为从PL和PC到主关系变量上的外键约束。（一个没有已知颜色的零件将在P中表示，但不在PL中表示；同样，一个没有已知城市的零件将在P中表示，但不在PC中表示。）

顺便说一句，另一种赞成包括主关系变量P的说法与发货关系变量SP有关：如果有主关系变量，那么我们可以保持传统的从SP到P的外键约束；如果没有它，事情就会变得更加混乱。

13.6 每对属性都是一个键。指定的JD不存在，因为下面所示的无疑是此关系变量的一个合法值：

---

a1	b1	c2
b1	a1	c2
a1	b2	c1
b2	a1	c1
a2	b1	c1
b1	a2	c1

---

( $a1 \neq a2$ ,  $b1 \neq b2$ ,  $c1 \neq c2$ )；也就是说，元



组  $(a1, b1, c2)$ 、 $(a1, b2, c1)$  以及  $(a2, b1, c1)$  当然不强迫元组  $(a1, b1, c1)$  出现 (!)。此关系变量属于6NF。但是请注意，它服从一个特定的对称约束，具体地说，当且仅当元组  $(b,a\ c)$  出现时，元组  $(a,b,c)$  出现（查看上图的示例值以说明这一点）。因此，此关系变量也存在一定的插入和删除异常，且它不属于DK/NF。

13.7 设关系 $r$ 具有标题 $H$ ，如果 $r$ 的基数为1或0，那么 $r$ 一定会满足所有可以在 $H$ 上定义的可能FD和JD。因此，依赖关系（FD和JD）的所有可能的集合都是一致的（虽然可能有一些这样的集合具有这样的含义，满足它们的任何关系只可以有至多为一的基数）。

13.8 以下肯定是关系变量SPJ'的一个合法值，

---

s1	p1	j1
s2	p1	j1

---

$(s1 \neq s2)$ ，所以 $\{PNO, JNO\}$ 不是一个键。同样，下面所示的也是SPJ'的一个合法值，

---

s1 p1 j1  
s1 p2 j1

---

( $p1 \neq p2$ )，所以{JNO,SNO}也不是一个键。

13.9 在这里要做的事情是把已经举行的比赛和那些没有举行的比赛分开：

---

```
PAST_MATCHES{DATE,OPPONENT,GOALS_FOR,GOALS_
.....}
KEY{DATE}
FUTURE_MATCHES{DATE,OPPONENT, .....}
KEY{DATE}
```

---

这些关系变量都属于5NF。尤其是PAST\_MATCHES可能不应该被替换为6NF投影。13.10能，我们可以把它定义为一个视图：

---

```
VAR          SCP          VIRTUAL ( (JOIN{S,SP,P})
{SNO,PNO,CITY}) ;
```

---

在这个视图中存在下面的FD：

---

$\{SNO,PNO\} \rightarrow \{CITY\}$

---

（事实上， $\{SNO, PNO\}$ 是一个键）。下面的（非平凡的）MVD也存在：

---

---

$$\{CITY\} \twoheadrightarrow \{SNO\} \parallel \{PNO\}$$

---

---

因为这些MVD的存在，所以关系变量SCP不属于4NF，虽然它属于BCNF。至于“传统的看法”，这个例子揭穿了另一种流行的误解：即，由单个键和单个非键属性组成的一个关系变量必然属于6NF，或至少属于5NF（见习题1.8）。

13.11 有关的定义，请参阅本章正文。至于一个例子，假设关系变量SP服从一个约束，其大意是编号为奇数的零件仅可以由编号为奇数的供应商提供且编号为偶数的零件仅可以由编号为偶数的供应商提供，（这个例子当然是很做作的，但对于当前的目的它是足够的）。那么这个约束显然不是被关系变量SP中存在的域约束和键约束蕴含的，所以关系变量不属于DK/NF，但它肯定属于6NF。

13.12 它们当然是有区别的，因为过强的PJ/NF蕴含5NF而5NF蕴含SKNF，且不存在反向的蕴含。但很容易将两者混淆，因为以下两个表面上类似的观察结果都是正确的（注意粗体字部

分)：

■当且仅当对于在关系变量R中存在的每一个不可约 $JD \bowtie \{X_1, \dots, X_n\}$ ，每个 $X_i$  ( $i=1, \dots, n$ ) 都包括R的某个键K时，R属于SKNF。

■当且仅当对于在关系变量R中存在的每一个不可约 $JD \bowtie \{X_1, \dots, X_n\}$ ，每个 $X_i$  ( $i=1, \dots, n$ ) 都包括R的相同键K时，R属于过强的PJ/NF。

13.13 如果你认为下面这些定义有点姗姗来迟，我为此道歉。

■定义：设r是关系 $\langle H, h \rangle$ ，并设c是一个布尔表达式，其中的每个属性都引用r的某个属性，并且没有任何关系变量的引用。那么c是一个限制条件 (restriction condition)，并且根据c的限制 ( $r \text{ WHERE } c$ )，r是关系 $\langle H, x \rangle$ ，其中x是r中c的判断结果为TRUE的所有元组的集合。

■定义：设关系 $r_1, \dots, r_n$  ( $n \geq 0$ ) 都有相同的标题H。那么 $r_1, \dots, r_n$ 的合并 ( $\text{UNION}\{r_1, \dots, r_n\}$ ) 是一个关系，它的标题为H且正文是使得t出现在 $r_1, r_2, \dots, r_n$ 中至少一个中的所有元组t的集合。（这里没有显

示，如果 $n=0$ ，需要一些句法机制来指定相关的标题 $H$ ，其结果是唯一具有那个标题的空关系。）注意，这里定义的合并是一个 $n$ 元操作符，而不仅仅是一个二元操作。

13.14 粗略地讲，谓词的析取是两个或两个以上其他谓词的“或”（OR）。如果某个关系变量 $R$ 具有一个析取关系变量谓词，那么“或”在一起的每个单独的谓词必须具有相同的参数（因为满足它们的元组都必须具有相同的类型）。把这样的谓词简化为简单的谓词将涉及通过限制而不是投影实现的关系变量分解（通过合并而不是连接实现重构）。事实上，这也正是在本章正文最后一节的所有内容。本书第四部分给出了更多的细节。

**[1]** 所以你觉得此关系变量存在冗余吗？证明你的答案！

## 第14章

14.1 请参阅本章正文。

14.2 没有提供答案。

14.3 “否”，事实并非如此。参见第15章对这类例子的进一步说明。

14.4 这个设计不违反正交性，但有其他几个地方存在错误。例如，你将如何表达“获取供应商S1的城市”这个查询？（有两种情况需要考虑：第一种情况，你至少确实知道存在什么样的供应商城市，第二种情况，你不知道供应商城市是什么。在后一种情况下，你可能还要考虑这个查询：“供应商S1在数据库里存在吗？”）此外， $FD\{CITY\} \rightarrow \{STATUS\}$ 是什么状况？关系变量SP中的外键{SNO}又是怎么回事？（再次提醒，实际上也有两种情况需要考虑——与刚才相同的两个情况。）

接下来，如果我们确实在关系变量LS、PS等中保持CITY属性，那么这些关系变量中的每个都将存在 $FD\{\} \rightarrow \{CITY\}$ 。因为这个FD不是“从一个键出来的箭头”，所以这些关系变量不属于BCNF。（见习题4.6的答案，它讨论了一个基本

上类似的例子。)

更重要的是，假设 $FD\{CITY\} \rightarrow \{STATUS\}$ 在原始的供应商关系变量S中存在，当然，它仍然在关系变量LS、PS等中存在（假设，也就是说，我们在那些关系变量中保持CITY属性），再次得出那些关系变量不属于BCNF。

总之，无论我们是否保持CITY属性，这些关系变量中的每个都存在 $FD\{\} \rightarrow \{STATUS\}$ ，再次得出关系变量不属于BCNF。因此，如果 $FD\{CITY\} \rightarrow \{STATUS\}$ 真的存在（仅在保留CITY属性时存在），根据建议的水平分解，它其实是可约的。

本书由“行行”整理，如果你不知道读什么书或者想获得更多免费电子书请加小编微信或QQ：491256034 小编也和结交一些喜欢读书的朋友 或者关注小编个人微信公众号id：d716-716 为了方便书友朋友找书和看书，小编自己做了一个电子书下载网站，网址：[www.ireadweek.com](http://www.ireadweek.com) QQ群：550338315

## 第15章

15.1 如果关于这个习题你有一个很好的答案，请通过“PO Box 1000, Healdsburg, CA 95448, USA”把它寄给我（请仅使用普通邮件）。



# Table of Contents

[题献](#)

[O'Reilly Media, Inc. 介绍](#)

[业界评论](#)

[译者序](#)

[作者简介](#)

[前言](#)

[先决条件](#)

[逻辑与物理设计的对比](#)

[致谢](#)

[第一部分 设置环境](#)

[第1章 篇首语](#)

[1.1 从文献摘录的一些引用](#)

[1.2 关于术语的说明](#)

[1.3 正在运行的示例](#)

[1.4 键](#)

[1.5 设计理论的地位](#)

[1.6 本书的目的](#)

[1.7 结束语](#)

[习题](#)

[第2章 预备知识](#)

[2.1 概览](#)

[2.2 关系及关系变量](#)

[2.3 谓词和命题](#)

## 2.4 更多的供应商和零件

### 习题

## 第二部分 函数依赖、BOYCE/CODD范式及相关事宜

### 第3章 规范化：一些通则

#### 3.1 规范化用于两个目的

#### 3.2 更新异常

#### 3.3 范式层次结构

#### 3.4 规范化和约束

#### 3.5 结束语

### 习题

### 第4章 函数依赖和BCNF（非正式的）

#### 4.1 第一范式

#### 4.2 函数依赖

#### 4.3 键的重新审视

#### 4.4 第二范式

#### 4.5 第三范式

#### 4.6 Boyce/Codd范式

### 习题

### 第5章 函数依赖和BCNF（正式的）

#### 5.1 初步定义

#### 5.2 函数依赖

#### 5.3 Boyce/Codd范式

#### 5.4 希思定理

### 习题

## 第6章 保持函数依赖

6.1 遗憾的冲突

6.2 第二个例子

6.3 第三个例子

6.4 第四个例子

6.5 一个能够工作的过程

6.6 恒等分解

6.7 关于冲突的更多内容

6.8 独立投影

习题

## 第7章 FD公理化

7.1 阿姆斯特朗公理

7.2 附加规则

7.3 证明附加规则

7.4 另一种闭包

习题

## 第8章 反规范化

8.1 “反规范化是为了性能”吗

8.2 反规范化是什么意思

8.3 什么不是反规范化（I）

8.4 什么不是反规范化（II）

8.5 反规范化是有害的（I）

8.6 反规范化是有害的（II）

8.7 结束语

习题

### 第三部分 连接依赖、第五范式及其他相关事项

#### 第9章 连接依赖及5NF（非正式的）

##### 9.1 连接依赖的基本思路

##### 9.2 一个属于BCNF但不属于5NF的关系变量

##### 9.3 循环规则

##### 9.4 结束语

##### 习题

#### 第10章 连接依赖及5NF（正式的）

##### 10.1 连接依赖

##### 10.2 第五范式

##### 10.3 被键蕴含的JD

##### 10.4 一个有用的定理

##### 10.5 FD不是JD

##### 10.6 更新异常再探

##### 习题

#### 第11章 隐式依赖关系

##### 11.1 无关的分量

##### 11.2 结合分量

##### 11.3 不可约的JD

##### 11.4 小结

##### 11.5 追逐算法

##### 11.6 结束语

##### 习题

#### 第12章 多值依赖和4NF

[12.1 一个介绍性的例子](#)

[12.2 多值依赖（非正式的）](#)

[12.3 多值依赖（正式的）](#)

[12.4 第四范式](#)

[12.5 公理化](#)

[12.6 嵌入式依赖](#)

[习题](#)

## [第13章 额外的范式](#)

[13.1 相等依赖](#)

[13.2 第六范式](#)

[13.3 超键范式](#)

[13.4 无冗余范式](#)

[13.5 域-键范式](#)

[13.6 结束语](#)

[13.6.1 基本键范式  
\(EKNF\)](#)

[13.6.2 过强的PJ/NF](#)

[13.6.3 “限制-合并”范式](#)

[习题](#)

## [第四部分 正交](#)

### [第14章 正交设计原则](#)

[14.1 规范化的两个欢呼声](#)

[14.2 一个启发性的例子](#)

[14.3 一个更简单的例子](#)

[14.4 元组与命题](#)

[14.5 第一个例子再探](#)

[14.6 第二个例子再探](#)

[14.7 最终版本](#)

[14.8 澄清](#)

[14.9 结束语](#)

[习题](#)

## [第五部分 冗余](#)

### [第15章 我们需要更多的科学](#)

[15.1 一点历史](#)

[15.2 数据库设计是谓词设计](#)

[15.3 例1](#)

[15.4 例2](#)

[15.5 例3](#)

[15.6 例4](#)

[15.7 例5](#)

[15.8 例6](#)

[15.9 例7](#)

[15.10 例8](#)

[15.11 例9](#)

[15.12 例10](#)

[15.13 例11](#)

[15.14 例12](#)

[15.15 管理冗余](#)

[1.只用原始设计](#)

[2.声明约束](#)

[3.使用视图](#)

[4.使用快照](#)

[15.16 改善定义](#)

[15.17 结束语](#)

[习题](#)

## [第六部分 附录](#)

[附录A 主键是良好的，但不是必需的](#)

[A.1 为PK：AK区别辩护的论据](#)

[A.2 具有多个键的关系变量](#)

[A.3 发票和发货例子](#)

[A.4 是否每个实体类型一个主键](#)

[A.5 申请人及员工的实例](#)

[A.6 结束语](#)

[附录B 冗余回顾](#)

[附录C 重要论文回顾](#)

[附录D 习题答案](#)

[第1章](#)

[第2章](#)

[第3章](#)

[第4章](#)

[第5章](#)

[第6章](#)

[第7章](#)

[第8章](#)

[第9章](#)

[第10章](#)

[第11章](#)

[第12章](#)

[第13章](#)

[第14章](#)

[第15章](#)



如果你不知道读什么书，  
就关注这个微信号。



公众号名称：幸福的味道

公众号ID：d716-716

小编：行行：微信号：491256034

为了方便书友朋友找书和看书，小编自己做了一个电子书下载网站，网址：[www.ireadweek.com](http://www.ireadweek.com)  
QQ群：550338315 小编也和结交一些喜欢读书的朋友  
“幸福的味道”已提供120个不同类型的书单

- 1、 25岁前一定要读的25本书
  - 2、 20世纪最优秀的100部中文小说
  - 3、 10部豆瓣高评分的温情治愈系小说
  - 4、 有生之年，你一定要看的25部外国纯文学名著
  - 5、 有生之年，你一定要看的20部中国现当代名著
  - 6、 美国亚马逊编辑推荐的一生必读书单100本
  - 7、 30个领域30本不容错过的入门书
  - 8、 这20本书，是各领域的巅峰之作
  - 9、 这7本书，教你如何高效读书
  - 10、 80万书虫力荐的“给五星都不够”的30本书
- .....

关注“幸福的味道”微信公众号，即可查看对应书单

如果你不知道读什么书，就关注这个微信号。