XML 数据库模式规范化

曹丽君　张忠平　著

科 学 出 版 社

北　京

内 容 简 介

本书采用路径表达式和树元组的表示方法分析了 XML 数据库模式的设计和 XML 数据规范化设计，增强了 XML 数据的语义表现力，完善了 XML 数据库规范化理论，从基本定义及符号、XML 函数依赖、XML 范式及文档规范化、XML 键约束、XML 多值依赖、XML 函数依赖和多值依赖的成员籍判定、XML 函数依赖和多值依赖并存下的范式及其规范化、XML 文档相似性度量、基于共享路径处理小枝模式的索引方法、基于 ISP 索引的小枝模式处理方法、基于小枝模式查询的灵活匹配等方面进行了深入研究。

本书具有定位准确、取舍合理、体系得当等特点，可作为高等院校计算机专业本科生、研究生 XML 数据库技术课程的参考教材或课外读物，也适合数据库技术人员和管理人员、科研人员等相关专业学者对 XML 数据库模式规范技术的学习和研究。

图书在版编目（CIP）数据

XML 数据库模式规范化/曹丽君，张忠平著. —北京：科学出版社，2015
ISBN 978-7-03-045316-7

I. ①X… Ⅱ. ①曹…②张… III. ①可扩充语言-数据库系统-研究 IV. ①TP312

中国版本图书馆 CIP 数据核字（2015）第 181790 号

责任编辑：陈晓萍 / 责任校对：王万红
责任印制：吕春珉 / 封面设计：子时文化

科学出版社 出版
北京东黄城根北街 16 号
邮政编码：100717
http://www.sciencep.com

印刷

科学出版社发行 各地新华书店经销

*

2015 年 7 月第 一 版 开本：B5（720×1000）
2015 年 7 月第一次印刷 印张：11 3/4
字数：250 000

定价：56.00 元

（如有印装质量问题，我社负责调换〈 〉）

销售部电话 010-62134988 编辑部电话 010-62138978-2009

前　言

随着 XML 成为 Web 上的数据表示和数据交换的标准，需要通过 Web 交换和处理的 XML 数据在大幅度增加，这就对 XML 数据库的模式提出了更高的要求。同关系数据库类似，如果 XML 数据模式设计得不好，同样会引起插入、删除和更新等异常。由于 Web 的开放性，XML 数据异常的危害性要远远大于关系数据异常的危害性。虽然 XML 数据的相关技术研究已有些成果，如 XML 数据的存储与发布技术、XML 数据查询与优化技术等，特别是与关系数据的转换技术已相当成熟。但是，XML 数据已经成为 Internet 上的主流数据，如果仅仅考虑到如何从 XML 文档数据转换到关系数据，且这种转换只是保持了结构信息，而没有从数据库设计的角度来评价 XML 数据库模式，这也必将对以后的 Web 数据处理带来很大的麻烦，势必造成数据的大量冗余和不一致现象。本书从数据库设计的角度出发，对 XML 数据的约束进行深入的研究，直接对 Web 数据进行规范化处理，从而得到良好的 XML 数据库模式，这样不仅完整地保留了 XML 文档数据中的语义和结构信息，满足了数据库设计的要求，一次性地完成了 XML 数据库的设计，避免了现有方法的重复设计，而且减少了数据冗余，保持了 Web 上数据的一致性。因此，XML 数据库模式规范化的研究具有重要的理论意义和实用价值。

本书的主要工作是在已有的 DTD 和 XML-Schema 等规范基础上，采用路径表达式和树元组的表示方法对 XML 数据进行规范化设计与论述。本书共 12 章，分别为绪论、基本定义及符号、XML 函数依赖、XML 范式及文档规范化、XML 键约束、XML 多值依赖、XML 函数依赖和多值依赖的成员籍判定、XML 函数依赖和多值依赖并存下的范式及其规范化、XML 文档相似性度量、基于共享路径处理小枝模式的索引方法、基于 ISP 索引的小枝模式处理方法、基于小枝模式查询的灵活匹配。

第 1 章为绪论。阐述了 XML 数据库模式规范化的背景，分析了国内外 XML 数据库模式规范化的研究现状，论述了 XML 数据库模式规范化的主要内容及理论和实际意义。

第 2 章为基本定义及符号。介绍了形式化的 DTD、XML 文档和路径表

达式等定义，定义了 XML 树元组、结点值相等、树包含等相关符号概念，并结合实例对这些定义和符号进行了说明。

第 3 章为 XML 函数依赖。本章基于路径表达式和树元组给出了 XML 函数依赖的形式化定义，并在此基础上给出了 XML 函数依赖的推理规则和 XML 路径集闭包的概念，并对 XML 函数依赖推理规则的有效性和完备性进行了证明。接着，在 XML 函数依赖和推理规则的基础上引入了 XML 函数依赖的逻辑蕴涵与覆盖的概念，包括无冗余覆盖、规范覆盖以及最小覆盖等，给出了相关的覆盖的求解算法，并对算法的正确性、可终止性和时间复杂性进行了分析和证明。最后，给出 FD_{XML} 集的最优覆盖概念。

第 4 章为 XML 范式及文档规范化。基于 XML 函数依赖形式化定义，给出 XML 范式的定义。提出 XML 文档规范化规则——元素提升规则和元素创建规则，并在此基础上给出 XML 文档规范化算法。对算法的正确性、可终止性和时间复杂性进行证明分析，并实验证明了规范化后的文档在查询时间和存储空间效率上都有明显的改善。

第 5 章为 XML 键约束。本章在分析已有半结构化数据键定义的优劣基础上，结合 FD_{XML} 定义，首先给出了基于路径表达式的 XML 键定义。该定义支持多个元素和属性，表达绝对和相对的现实语义约束，不受限于 XML 文档的任何类型的规范。其次，给出了绝对键和相对键的推理规则，并对其有效性和完备性进行了证明。最后，基于 XML 键及其相关概念和推理规则，给出多项式时间求解 FD_{XML} 集的一个候选键的算法，并对算法的正确性、可终止性和时间复杂性进行了证明和分析。

第 6 章为 XML 多值依赖。本章针对 XML 文档中的多值依赖问题，分析了 XML 文档中由多值依赖而引起的数据冗余和各种操作异常现象，包括插入异常、删除异常、更新异常等。给出了有关 XML 多值依赖的一些基本概念，包括 MVD_{XML} 的逻辑蕴涵、等价与覆盖，路径集闭包，无冗余覆盖，简化的 MVD_{XML} 等。基于基本的概念，给出了 XML 多值依赖有效和完备的推理规则，并对有效性和完备性进行了证明。同时，给出了相应问题的求解算法，包括 MVD_{XML} 成员籍判定算法，左、右部简化的 MVD_{XML} 集算法，无冗余 MVD_{XML} 集检验算法以及求解 MVD_{XML} 无冗余覆盖算法。上述算法从正确性、可终止性和时间复杂性上给出了分析证明。XML 文档中的多值依赖问题的研究更好地表达了现实世界中实体的一对多语义约束关系。

第 7 章为 XML 函数依赖和多值依赖的成员籍判定。本章针对 XML 函数依赖和多值依赖并存情况下的成员籍问题。基于成员籍问题的描述和讨论，给出了求解路径依赖基、成员籍和最小依赖集的算法，并对这三个算法的可终止性、正确性和时间复杂度进行分析和证明。

第 8 章为 XML 函数依赖和多值依赖并存下的范式及其规范化。对 XML 中的冗余和键进行了描述，给出了 3XNF 和 4XNF 定义，在此基础上给出了规范化规则和无冗余判定定理，进一步提出了 XML 文档规范化算法，并对算法的可终止性、正确性及时间复杂度进行分析和证明，最后通过实验证明该算法的有效性。

第 9 章为 XML 文档相似性度量。本章着重讨论了 XML 文档基于路径集合和代价的相似性度量，这对 XML 文档相似性度量，聚类 XML 文档树编辑距离度量提供了更丰富的度量方法，给出了规范化 XML 文档相似性度量方法——集合度量方法、线性度量方法和代价度量方法，并提出基于权重代价的机器学习的相似性度量算法。实验证明该算法扩展了 XML 文档查询范围，提高了文档的查全率和查准率。

第 10 章为基于共享路径处理小枝模式的索引方法。论述了小枝模式查询处理和索引技术的相关研究现状，指出了现有小枝模式查询处理方法中存在的关键问题，结合已经存在的索引思想，给出了一种基于共享路径的索引 ISP，利用该索引技术可以高效地处理小枝模式查询。

第 11 章为基于 ISP 索引的小枝模式处理方法。对基于 ISP 索引的小枝模式处理方法进行了比较深入的分析。首先分析了小枝模式处理的过程，针对其处理过程为每一步骤提出了详细的处理方法或算法，并对基于 ISP 索引的小枝模式处理算法进行了实验分析。

第 12 章为基于小枝模式查询的灵活匹配。分析了小枝模式匹配的国内外研究现状和现有匹配方法的不足，阐述了小枝模式匹配中的基本概念，提出了基于小枝模式匹配的灵活匹配方法，并定义了相关匹配条件，并通过具体的实例对所提出的方法进行了性能分析和说明。

本书由河北科技师范学院学术著作出版基金资助，同时也是数学与信息科技学院网络工程专业改革试点阶段性成果之一，由河北科技师范学院曹丽君统稿并编写第 1～6 章和第 10～12 章（共 20 万字），燕山大学张忠平统稿并编写第 7～9 章（共 5 万字），本书在编写过程中也得到了河北科技师范学

院王海明、马国光、赵立强、李玉香、李密生的帮助与支持，在此一并表示深深的感谢与敬意。

对于本书的编写，作者尽量体现科研思维和成果，虽竭尽全力，但限于能力和水平，书中难免存在疏漏和错误之处，希望广大读者批评指正。

曹丽君

于河北科技师范学院

张忠平

于燕山大学

目　录

第 1 章　绪　　论

1.1　背　　景

互联网（Internet）已经成为新经济时代的标志，它极大地影响了人类的生活方式、商业模式，并将继续对人类社会的进步起着巨大的推动作用。2015 年 2 月 3 日，中国互联网络信息中心（CNNIC）发布了《第 35 次中国互联网络发展状况统计报告》，报告显示：

截至 2014 年 12 月，我国域名总数为 2060 万个，年增长 11.7% 。其中“.CN”域名总数为 1109 万，年增长 2.4%，占中国域名总数比例为 53.8%；“.COM”域名数量为 795 万，占比为 38.6%；“.中国”域名总数达到 28.5 万。中国网站数量为 335 万个，年增长 4.6%。中国网页数量为 1899 亿个，年增长 26.6%。其中，静态网页数量为 1127 亿，占网页总数量的 59.36%；动态网页数量为 772 亿，占网页总量的 40.64%。我国 IPv4 地址数量为 3.32 亿，拥有 IPv6 地址 18797 块/32，年增长 12.8%。国际出口带宽为 4118663Mb/s，年增长 20.9%。

2014 年，台式计算机、笔记本电脑等传统上网设备的使用率保持平稳，移动上网设备的使用率进一步增长，新兴家庭娱乐终端网络电视的使用率达到一定比例。通过台式计算机和笔记本电脑接入互联网的比例分别为 70.8%、43.2%，与 2013 年底基本持平；通过手机接入互联网的比例继续增高，较 2013 年底提高 4.8%；平板电脑的娱乐性和便捷性特点使其成为网民的重要娱乐设备，2014 年底使用率达到 34.8%，并在高学历（本科及以上学历网民使用率 51.0%）、高收入人群（月收入 5000 元以上网民使用率 43.0%）中拥有更高使用率；随着网络技术和宽带技术的发展，网络电视融传统电视和网络为一身，其共享性、智能性和可控性迎合现代家庭娱乐需求，逐渐成为一种新兴的家庭娱乐模式，截至 2014 年 12 月，网络电视使用率已达到 15.6%。

作为互联网最主要应用的 Web 正成为整个世界的窗口，它实现了全球用户和机构信息的共享。Web 已经成为人类社会的主要信息源、媒体和商务的门户，互联网已经成为信息传播的介质和商务运作的基础平台。

随着互联网时代的到来，数据越来越多地开始以网络在线的方式进行发布、交换和集成。1998 年 2 月，万维网协会 W3C（World Wide Web Consortium）推出了可扩展的标记语言 XML（eXtensible Markup Language），将它作为一种互联网进行数据表示和交换的标准。XML 作为一种半结构化数据的表示模型，从提出到现在只不过几年的时间，但它作为一种跨产品、跨界面、跨平台的互联网的标准语言，已经显现出其强大的应用前景，并受到了政府、企业和各大软件厂商的广泛关注。各个行业如金融机构、海关、媒体产业正制定各自行业的 XML 文档类型定义 DTD（Document Type Definition），以利于数据以公认的格式进行交换与集成。随着 XML 数据的增多，相关行业标准 DTD 的制定，人们也开始越来越多地希望以对待数据库的方式来处置和管理 XML 文档。人们关注的首要问题是，用 XML 表示的数据之间有什么联系，有什么约束？表 1-1 中列出了常见的几种 XML 模式语言。

表1-1　常见的几种XML模式语言特性比较

特性	DTD1.0	XML-Schema1.0	XDR 1.0	SOX 2.0	Schematron 1.4	DSD 1.0
模式						
XML 中的语法	否	是	是	是	是	是
命名空间	否	是	是	是	是	否
包含	否	是	否	是	否	是
输入	否	是	否	是	否	否
数据类型						
内置类型	10	37	33	17	0	0
用户定义类型	否	是	否	是	否	是
域约束	否	是	否	部分	是	是
显示空值	否	是	否	是	否	否
属性						
默认值	是	是	是	是	否	是
选择性	否	否	否	否	是	是
可选与必需	是	是	是	是	是	是
域约束	部分	是	部分	部分	是	是
条件定义	否	否	否	否	是	是
元素						
默认值	否	部分	否	否	否	是
内容模型	是	是	是	部分	是	是

续表

特性	DTD1.0	XML-Schema1.0	XDR 1.0	SOX 2.0	Schematron 1.4	DSD 1.0
有序序列	是	是	是	是	是	是
无序序列	否	是	是	否	是	是
选择性	是	是	是	是	是	是
最小和最大次数	部分	是	是	是	是	部分
元素						
开放模型	否	否	是	否	是	否
条件定义	否	否	否	否	是	是
继承						
扩展简单类型	否	否	否	否	否	否
约束简单类型	否	是	否	是	否	否
扩展复杂类型	否	是	否	是	否	否
约束简单类型	否	是	否	否	否	否
唯一性或键						
属性唯一性	是	是	是	是	是	是
非属性唯一性	否	是	部分	否	是	否
属性键	否	是	否	否	是	否
非属性键	否	是	否	否	是	否
属性外键	部分	是	部分	部分	是	是
非属性外键	否	是	否	否	否	是
其他						
动态约束	否	否	否	否	是	否
版本	否	否	否	否	否	是
文件/记录	否	是	否	是	是	是
嵌入 HTML	否	是	否	是	部分	是
自描述性	否	部分	否	否	部分	是

从表 1-1 中可以看出：

从“使用容易”角度看，DTD 是最容易学习的模式语言。Schematron 模式语言描述是相对简单的，但要求用户仍需学习另一种语言 XPath，才能展示更多的功能。DSD 模式比 XML-Schema 和 Schematron 模式倾向于更加详细一些，由于 XML-Schema 和 DSD 支持广泛的性质集，相对而言是比较难学的，但是从 DTD 很容易移植到其他模式语言中。

从“语言”角度看，XML 模式语言可以从多个方面来划分，如基于语法

与基于模式、面向定义与面向有效性、面向结构与面向约束等。DTD，XML-Schema，XDR 和 SOX 属于基于语法分组，而 Schematron 属于基于模式的分组。DSD 介于两者之间，同时支持两种性质。基于语法分组在 XML 查询中有优势，已知模式结构和定义可以帮助用户书写更多的优化查询。另一方面，基于模式语言划分可以在表达中更好地描述约束。

从“数据库”角度看，没有一种语言彻底地满足需要。数据定义语言（Data Definition Language，SQL DDL）描述的不仅仅是关系和属性集的规范，还包括每个属性、完整性约束、每个关系安全性等索引相关的值域信息。XML 模式支持各种固定的域类型，并不能表达关系模式所描述的全部内容。尽管 Schematron 或 DSD 能表达完整性约束，但它们不支持物理索引描述功能。

上述几种典型的模式语言，都是从语言设计角度定义的，缺少对 XML 模式约束的描述，因为约束是数据语义的重要组成部分。然而 XML 文档作为半结构化数据的特例，虽然它很容易表达来自不同数据源的数据，但是由于 DTD 与 XML-Schema 这些模式定义方法对于约束的描述都是有限的，其所能表示的语义信息也是相对有限的。

由于 XML 作为 Internet 上数据标准的出现，XML 数据的相关研究成为热点。如图 1-1 所示，从现实世界中客观事物到其在计算机中的具体表示，实际上经历了三个领域——现实世界、信息世界和机器世界。存在于人们头脑之外的客观世界，称为现实世界。信息世界是现实世界在人脑中的反映，人们把它用文字和符号记载下来。信息世界的信息以某种数据形式存储在机器世界里。本书认为 Web 世界应当与机器世界是同一层次，即它们都位于信息世界下。信息世界的信息被分别存储在这两个世界中。关系模型和面向对象模型都是机器世界中的数据模型。与之相似，XML 则应当被看成 Web 世界中的数据模型，DTD 则应当被看成 XML 数据的模式。

现有的 XML 存储方法，如图 1-1 所示，都是一个从 Web 世界到机器世界直接的转换过程，它们仅仅考虑到如何完整地保留 XML 文档中的结构信息，而没有从数据库设计角度来评价所得到的关系数据库模式，这样必将对后来的数据处理带来很大的麻烦，因为这样的数据库可能存在着插入、删除和更新异常。为了避免这些异常的出现，就必须对得到的关系数据库模式按照数据库设计中的要求进行改进。也就是说，现有的方法看起来简单，实际上要想真正得

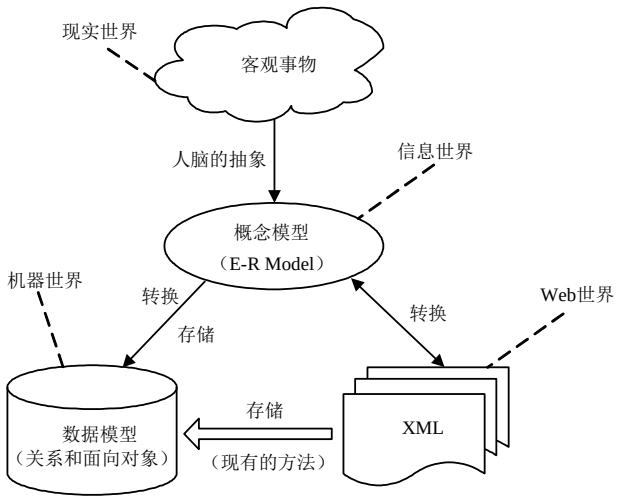

图 1-1　信息的转化过程

到一个优秀的数据库却是非常复杂的。本书的思想不是从 Web 世界到机器世界直接的转换，而是着眼于数据库设计，对 Web 世界中的模式（即 DTD）数据约束进行规范化研究，直接进行 XML 数据库模式的设计，这样不仅完整地保留了 XML 文档中的语义和结构信息，而且满足了数据库设计的要求，一次性地完成了一个优秀数据库的设计，避免了现有方法的数据库重复设计。

随着 XML 应用范围的拓广，为了提高数据质量，作为 XML 文档的主要模式定义方法，一个设计良好的 DTD 是必须的。那么，如何设计良好的 DTD？进一步地，如何将设计不好的 DTD 转化为相应的设计良好的形式就变得很重要。可以从不同的角度来判断 DTD 是否设计良好，一个重要的考虑因素——数据冗余。这是因为，和关系数据库一样，数据冗余往往是导致操作异常产生的根源。数据被重复地多次存储，必然使相应的插入、删除和更新操作变得困难。如果 XML 的模式设计得不好，在 XML 文档中同样也会造成数据冗余和操作异常现象。这必将造成 XML 文档中数据存储空间的浪费，查询优化也会受到影响。对于 XML 文档操作产生的各种异常，直接影响 XML 作为不同应用程序之间数据表示和交换标准的应用。由于 Internet 的开放性，XML 数据操作异常的危害性要远远大于关系数据操作异常的危害性，这就对 XML 数据库模式提出了更高的要求。

XML 是半结构化数据的特例。半结构化数据是介于严格结构化的数据（如关系数据库中的数据）和完全无结构的数据（如声音、图像文件）之间的数据

形式，它具有如下一些特点。

（1）隐含的模式信息。半结构化数据是具有一定的结构，但其结构与数据混在一起，没有显式的模式定义，如 HTML 文件。

（2）不规则的结构。一个数据集合可能有异构的元素组成，例如，学生集合中某些学生有电子邮件地址，而另一些学生则没有，同样的信息可能由不同的类型的数据表示，如某些姓名是字符串，而另一些则是由 first name 和 last name 组成的复杂结构。

（3）没有严格的类型约束。由于没有一个预先定义的模式，以及数据在结构上的不规则性，所以缺乏对数据的严格类型约束。

目前国内外关于半结构化数据的研究主要集中在新的数据模型、查询模式、存储技术以及优化技术等方面，在众多的研究课题中，对半结构化数据结构的研究是一个非常重要的方向。半结构化数据存在一定的结构，但这些结构或者没有被清晰地描述，或者是经常动态变化的，或者过于复杂而不能被传统的模式定义来表现。半结构化数据的模式与传统的关系或面向对象数据的模式不同，主要有如下一些特点。

（1）对半结构化数据来说，是先有数据，后有模式。

（2）半结构化数据的模式是用于描述数据的结构信息，而不是对数据结构进行强制性的约束。

（3）半结构数据的模式是非精确的，它可能只描述数据的一部分结构，也可能根据数据处理的不同阶段的视角而不同。

（4）半结构化数据的模式可能规模很大，甚至超过数据源数据的规模，而且会因数据的不断更新而处于动态的变化过程中。

没有强制性的模式的约束，使得半结构化数据具有很大的灵活性，能够满足网络这种复杂分布环境的需要，但是也给数据的处理带来了很大的困难，使得数据处理的效率低下，很难具有实用性。半结构化数据模式在实际的数据处理中有着很广泛的用途。

（1）用户界面。由于半结构化数据没有明确的模式，给用户查询带来了很大的困难。模式信息有助于用户了解数据的结构，从而提出更精确和有效的查询。

（2）查询优化处理。模式信息有助于查询处理器对查询计划进行优化，大大缩减查询的搜索空间。

（3）改进数据存储。了解模式信息，可以更好地设计数据的物理存储结构以及索引，从而提高查询执行的效率。

（4）异构数据源的集成。了解不同数据源的模式信息，有助于选择适当的集成模式和定义转换规则。

由于认识到半结构化数据模式的重要性，近年来学者们已经在这方面做了很多研究工作，有许多工作目前仍在进行当中。

对于半结构化数据的模式，目前已经提出了多种描述形式，比较有代表性的有基于逻辑的形式和基于图的形式。无论是哪种描述形式，其讨论的基础都是采用带标记的有向图作为半结构化数据模型，最典型的就是 OEM 模型，概括来说目前有两类描述方法。

（1）基于逻辑的描述形式。在已经提出的半结构化数据模式的描述形式中，基于逻辑的描述形式是重要的一类，如一阶逻辑（first-order logic）、描述逻辑（description logic）以及 Datalog 等。它们非常相似，但在表达能力等方面有所差别，这方面比较典型的是基于 Datalog 的模式描述形式。

（2）基于图的描述形式。半结构化数据模式的另一种重要形式是基于图的形式。由于半结构化数据一般采用带标记的有向图来表示，所以这种描述形式的一个显著优点是模式和数据采用同一种数据模型（图模型），给处理带来很大的方便。模式图通常是一个有根、边上带标记的有向图，其边上的标记可以与数据图相同，也可以加以扩充，如允许类似于“name|address”的形式，或采用特定形式的规则（如一元谓词）等。模式图中的结点可以加以一定的注释，表明其代表的语义或其他特定的含义。有许多学者还提出了其他形式的模式图，但本质上基本相同，这里不再一一介绍。

前面已经提到，半结构化数据是先有数据，后有模式。所以模式研究的一个重要方面就是如何从数据中得到模式，即模式的抽取，模式抽取所研究的问题：给定一个数据实例，在没有任何事先知识的情况下，自动地计算数据的相应模式；如果存在多个可能的模式，选择能最好地描述给定数据的模式。目前提出的模式抽取方法主要有 DataGuide、基于 Datalog 规则的抽取以及概念聚类等一些聚类和分类方法。

XML 是面向内容的，它具有更多的结构和更多的语义、良好的可扩展性、简单而易于掌握、自描述等特点，适合于 Web 上的数据交换，目前对 XML 的存储和查询方面的研究正方兴未艾。XML 数据模型与半结构化数据模型

有着很多的相似性，可以说，XML 是 WWW 上的半结构化数据，它既为半结构化数据的研究提供了广阔的应用前景，同时也推动了半结构化数据研究的发展。

XML 图是一种非常灵活的数据模型，它能很容易地构造关系数据和面向对象数据，从另一方面看，数据（包括不规则数据）与 XML 图能很方便地直接映射。XML 图非常适合描述分布的、多态的、动态改变的 Web 数据。在 OEM 模型与 XML 图之间的对应非常简单：OEM 对象对应于 XML 中的元素，OEM 中的子对象关系反映了 XML 中的元素嵌套。它们之间的不同之处在于 XML 的子元素可能是有序的，以及 XML 元素可能包含（属性，值）的列表。为了支持 XML 这两个特点，在 OEM 模型中引入三个新特性，即有序的子对象、（属性，值）列表以及参照边，就可以成为支持 XML 的数据模型了。为了更有效地进行 XML 数据的处理，学者们提出了许多关于其模式描述的方案，最主要的是文档类型定义（DTD）。与半结构化数据的模式相比，DTD 作为模式的优点是它的正则语法支持定义半结构数据。

但 XML 文档标记语言的烙印使 DTD 无法符合数据库的观点为数据提供非常适合的模式。因此，很多研究采用 XML 1.0 标准提出了更适合表达 XML 数据模式的方法，如 XML-Data、XML-Schema、DCD 等。它们的共同点：①要与 XML DTD 兼容并提供更丰富的描述能力；②使用 XML1.0 的语法规范定义自身。众多文献针对六种 XML 模式语言进行了分析比较，但是这些规范的研究还未成熟，在很多方面还未达成共识。相比之下，基于 DTD 规范已被广泛接受。

本书的规范化模式研究就是在这一背景下，基于 DTD 约束提出的。在数据库领域中，规范化是模式设计的基础，它是评价一个模式好坏的主要依据。对于 XML 领域，也要讨论相应的问题。目前 XML 已经成为 Internet 上的数据表示和数据交换的标准，需要通过 Internet 交换和处理的 XML 数据会大幅增加。对于 XML 数据库模式的深入研究将有力地促进企业的信息化、电子商务以及电子政务的发展，因此具有巨大的应用前景和经济效益。

1.2　国内外现状

数据依赖及规范化理论源于数据库领域。在关系数据库里，键和外键是

数据库概念设计的基础，它们提供了如何唯一识别一个元组和如何引用另一个元组的方法。函数依赖理论是关系模式设计的核心部分，是范式研究的基础。基于范式的关系模式，可以消除插入、修改和删除等异常现象，保证数据的完整性和一致性。在关系数据库领域中，对于函数依赖有深入的研究，还包括一些复杂的扩充的模型，如嵌套函数依赖和嵌套关系上的范式。相关文献中还给出了面向对象数据库中路径依赖、局部依赖和全局依赖的定义。此外，在有限的面向对象数据模型上也有关于函数依赖的讨论。它们的研究局限于传统的数据库范畴，XML 所特有的树状结构和路径导航的文档模型显然超出了它们的表述能力。而 XML 数据约束对于研究 XML 数据的查询优化、数据集成以及 XML 数据与其他形式数据库的转换都极为重要。在半结构化数据领域中，由于半结构化数据的树状结构，大多数研究比较集中在路径约束上。由于半结构化模式语言的表达能力的缺陷，造成对键的约束表述能力的缺乏。一些更广泛的约束是基于 Web 的管理进行研究的，显然，这样的研究并不是建立在统一的模型上的。

由于 DTD 规范包含一些基本的结构和域约束，目前关于 XML 上约束的讨论主要是基于 DTD 进行的。例如，下列的 DTD 片段：

```
<!ELEMENT Conference(topic, member+, reference*, sponsor?, property)>
    <!ATTLIST Conference property(internal | international)    #REQUIRED>
```

这个 DTD 片段表达了结构和域值两方面的最基本约束：

1）结构约束

（1）父子关系（祖先后代关系）。例如，元素 *Conference* 和元素 *topic*、*member*、*reference*、*sponsor* 之间就具有父子关系。

（2）基数。DTD 中支持使用符号+、*和?来表示基数，其含义分别为一个以上（含一个），任意多个（可以为零个）和可选（零个或一个）。若不使用+、*和?符号，比如对于元素 *topic*，就表示有且只有一个。

结构约束是平凡的，在进行有关 XML 上的树模式查询和结构化查询时，可以用于查询优化。

2）域约束

Conference 的 *property* 子元素的取值只限于 *internal* 或 *international* 值。

结构和域约束在进行 XML 文档的关系数据库存储时，应考虑到它们的保持性问题。

DTD，XML-Schema 和 XML Data 提供了 XML 上基本的约束定义能力。在 DTD 中，通过为属性标注 ID 和 IDREF 提供了定义键和外键的方法。但是这种机制的表述能力相对有限。首先，被标注了 ID 的属性必须在整个文档内唯一，而不是仅仅在某个特定类型的元素内。这样的要求显然局限性太强。在现实中，很多情况下键值都是采用自然数序列来生成的。如果是这样，那么文档中就不能存在两个这样的键了。例如，SwissProt 数据库由大量的条目组成，每个条目有一个编号，每个条目内又有一系列引用，由 1，2，3，…标注。因此，要想完全地确定一个引用，必须在一个条目内提供两个编号：一个是条目编号，一个是引用编号。其次，ID 或 IDREF 只能被标注在单个属性上，这样在关系数据库中很常见的复合键就没有办法在 XML 中表示了。最后，在 DTD 中，IDREF 既没有类型约束，也没有范围限制，在文档中根本没有任何机制来确保 IDREF 所指向的具体属性。为了解决这些问题，XML Data 提供了新的定义键和外键的能力。在 XML-Schema 中，则更进一步地支持使用 XPath 表达式来定义键和外键；但是 XML-Schema 中所采取的方式也仍然存在着一些问题。首先，XML-Schema 中的“相等”概念是比较模糊的，考虑到 XML 文档中可能具有的元素的复杂构造，对于“相等”概念需要有一个明确的定义。其次，作为一种用来表述键的路径表达式语言，XPath 显得过于复杂了。在 XPath 表达式中，不但可以在文档树上从根到叶，还可以逆向地从叶到根，更不用说其中还允许嵌入函数等。一个很重要的问题就是关于 XPath 表达式的包含和相等目前仍然无法得到有效的判定，这样在规范化研究中非常重要的推理就无法进行了。最后，在所有这些规范中，有关函数依赖的概念都完全没有涉及，更不用说是多值依赖概念了。

在 DTD 规范的基础上，扩展键的定义能力，可以表述相对键。相关文献更进一步地通过分析两类特殊的路径表达式，在文档上讨论了键的逻辑蕴涵问题及其计算复杂度分析。对于“相等”，相关文献中采取了“交不空”的定义方法，而并不是严格意义上的相等。另外它们的讨论仅限于键，并没有引入数据依赖约束的概念。一般来说，键只是数据依赖的一个特例，应该将其归结到数据依赖的体系中去。

形如 $(p,[q_1,q_2,...,q_n \to q_{n+1}])$ 的函数依赖表达式，其中，p,q_i（$i \in [1,n+1]$）均为形如 $\tau.@l$ 的路径表达式，τ 为元素类型。对于 XML 树 T 中的任意两棵子树 T_1、T_2，它们均为由 T 的根结点经由路径 p 到达的子树，如果对每个

$i \in [1,n]$，T_1、T_2 在 q_i 上均相等，则它们在 q_{n+1} 上也相等，此时，称 T 满足函数依赖约束。基于 XML 函数依赖的概念，建立一个 XML 函数依赖的代价模型，来度量 XML 数据库设计中的数据重复因素，通过路径表达式校验 XML 中的函数依赖，给出了实验结果，丰富了 XML 语义，但是没有给出一个用于推理的方法，没有从理论上给出完整的证明，这是判定逻辑蕴涵不可缺少的，且该文并没有讨论 XML 规范化问题。

已有的函数依赖的定义，通过将 XML 文档映射到关系表，讨论了 XML 的范式——XNF（Normal Form for XML），给出了一个将 DTD 转化为符合 XNF 形式的算法。从信息论的角度进一步说明了 XNF 的优越性。目前已有的主要问题是，模型中的“相等”有二义性。对于元素，相等是结点重合，而对于属性，相等则是值相等。文章中所需要的键和相对约束的概念实际上是依托于这个二义性来进行定义的。

半结构化模式图——S3-Graph（Semi-Structured Schema Graph），给出了一个基于半结构化模式图的范式——S3-NF（Normal Form for Semi-Structured Schema Graph）。同时给出两种方法来设计 S3-NF 数据库，即通过模式图分解和 ER 图方法重新构造。前者是分解半结构化模式图为 S3-NF 的规范集，后者是使用 ER 模型消除语义上的异常。但是 S3-Graph 没有区分元素结点和属性结点，没有指定模式的基数。S3-NF 不能够消除部分依赖和路径异常等问题。

基于 DTD、XML 的树模型和路径表达式，对结点的值相等和路径结点集作了定义，并在此基础上，给出了 XML 中函数依赖、逻辑蕴涵和路径闭包的概念，证明了函数依赖在给定 DTD 上的可满足性。提出了一个 XML 上函数依赖的正确和完备的推理规则集，并给出了一个用于计算路径闭包的算法。

可满足性是一个理论性很强的问题，其意义为：给定一组约束，是否存在 XML 文档满足这些约束。在给定 DTD 的情况下，问题就变为是否存在有相应的 XML 文档既符合该 DTD，又满足这些约束。约束主要被限定为键和外键，给出 DTD 和一组包括键和外键的完整性约束定义，是不能判定是否存在同时符合该 DTD 和完整性约束的 XML 文档的。即使进一步将键和外键的定义都限制为简单属性，类似的判定也是 NP 完全的。本书基于有限约束形式，给出在多项式时间内键的满足性和逻辑蕴涵的可判定方法。

众多文献阐述了使用关系数据库来存储 XML 文档以及 XML 文档和关系

数据库模式之间的转换。实际上在 XML 和关系数据库之间进行数据转化时考虑的是两个过程：一个是从数据库模式中产生 DTD，另外一个是根据 DTD 生成数据库模式。

从一个 DTD 中生成一个关系模式的步骤如下。

（1）对每一个元素，产生一个表和一个主键列。

（2）对每一个有混合内容的元素，产生一个独立的表格，用来存储 PCDATA，并通过父表的主键和父表相联。

（3）对元素类型中的每一个单一值的属性，对具有只有 PCDATA 内容的子元素（该子元素按顺序出现），产生一个单独的列，如果子元素类型或者值是可以选择的话，该列就应该可以允许为 NULL 类型。

（4）对有多个值的属性和可以出现多次的子元素（该子元素 PCDATA）的话，需要创建一个单独的表来存储这些值，并通过父表的主键和父表相联。

（5）对每一个包含元素或者混合内容的子元素来说，通过父表的主键把父元素和子元素连接起来。

从一个关系数据库模式构建 DTD 步骤如下。

（1）对每一个表，创建一个元素。

（2）对表中的每一列，创建一个属性或者是一个只有 PCDATA 内容的子元素。

（3）根据表中的每一主键/外键关系，创建该表元素的子元素。

上述的转换过程只是关注了 XML 文档的结构信息，而没有考虑到文档所具有的语义信息，如键、函数依赖等。这样带来的后果是两方面的：一方面是所产生的关系模式往往是零碎和低效率的；另一方面是完全丢失了 XML 文档中固有的约束，损失了语义的重要部分。也就是说，它们没有从数据库设计的角度出发考虑问题。

关系数据库中定义的范式——3NF、BCNF、4NF 和 5NF 或目前已有的嵌套关系 NNF 和 NF-NR 不能直接应用于半结构化数据中，理由如下。

（1）半结构化数据模型比关系数据模型更加丰富和复杂。例如，XML 的基数约束在关系数据模型中是没有的。

（2）半结构化数据的结构和数据是嵌套在一起的。因此，在半结构化数据中没有规则的结构。

（3）关系数据中是严格意义上原子类型值的比较，而半结构化数据中值相等的概念包含结点重合。

在关系模型规范化理论中常用一个关系集来判定给定数据库是否是一个良好的设计。半结构化数据库还没有一个规范化的理论来判定半结构化数据库是否是一个良好的设计。这是半结构化数据库研究领域中的一个重要问题，也是本书重点阐述的一个问题。

1.3　内容及意义

1.3.1　XML 规范化的内容

数据依赖是语义的重要组成部分。在 XML 领域引入函数依赖和多值依赖概念对于语义完整性约束、数据存储、数据集成和查询优化等有非常重要的作用。由于 XML 文档的层次结构远比平面关系数据库复杂，本书基于 XML 文档层次结构中的路径表达式，从数据库设计的角度考虑，展开对 XML 模式规范化的研究。主要研究内容：①基于 XML 函数依赖约束，对 XML 函数依赖、逻辑蕴涵和覆盖等概念进行定义，研究有效和完备的 XML 函数依赖推理规则，并探索多项式时间求解规范覆盖和最小覆盖的算法。②基于 XML 函数依赖形式化定义，定义 XML 不同级别的范式，研究 XML 文档规范化规则，探索 XML 文档规范化算法。③研究 XML 键约束以及绝对键和相对键的推理规则，探索多项式时间求解 XML 候选键的算法。④基于数据约束对 XML 多值依赖、逻辑蕴涵和覆盖进行形式化定义，研究 XML 多值依赖推理规则，探索多项式时间求解 XML 多值依赖无冗余覆盖算法。⑤研究 XML 文档相似性度量方法，探索基于权重代价的相似性度量算法。进一步扩展 XML 文档查询范围，提高文档的查全率和查准率。

本书的主要内容如下。

（1）阐述了 XML 数据库模式规范化的背景，分析了国内外 XML 数据库模式规范化的研究现状，论述了 XML 数据库模式规范化的主要内容及理论和实际意义。

（2）介绍了形式化的 DTD、XML 文档和路径表达式等定义，定义了 XML 树元组、结点值相等、树包含等相关符号概念，并结合实例对这些定义和符号进行了说明。

（3）给出了 XML 函数依赖的形式化定义，并在此基础上给出了 XML 函数依赖的推理规则和 XML 路径集闭包的概念，并对 XML 函数依赖推理规则的有效性和完备性进行了证明。接着，在 XML 函数依赖和推理规则的基础上引入了 XML 函数依赖的逻辑蕴涵与覆盖的概念，包括无冗余覆盖，规范覆盖以及最小覆盖等，给出了相关的覆盖的求解算法，并对算法的正确性、可终止性和时间复杂性进行了分析和证明。最后，给出 FD_{XML} 集的最优覆盖概念。

（4）给出 XML 文档规范化规则——元素提升规则和元素创建规则，并在此基础上给出 XML 文档规范化算法。对算法的正确性、可终止性和时间复杂性进行证明分析，并实验证明了规范化后的文档在查询时间和存储空间效率上都有明显的改善。

（5）首先给出了基于路径表达式的 XML 键定义。该定义支持多个元素和属性，表达绝对和相对的现实语义约束，不受限于 XML 文档的任何类型的规范。其次，给出了绝对键和相对键的推理规则，并对其有效性和完备性进行了证明。最后，基于 XML 键及其相关概念和推理规则，给出多项式时间求解 FD_{XML} 集的一个候选键的算法，并对算法的正确性、可终止性和时间复杂性进行了证明和分析。

（6）给出了有关 XML 多值依赖的一些基本概念，包括 MVD_{XML} 的逻辑蕴涵、等价与覆盖，路径集闭包，无冗余覆盖，简化的 MVD_{XML} 等。基于基本的概念，给出了 XML 多值依赖有效和完备的推理规则，并对有效性和完备性进行了证明。同时，给出了相应问题的求解算法，包括 MVD_{XML} 成员籍判定算法，左、右部简化的 MVD_{XML} 集算法，无冗余 MVD_{XML} 集检验算法以及求解 MVD_{XML} 无冗余覆盖算法。上述算法从正确性、可终止性和时间复杂性上给出了分析证明。XML 文档中的多值依赖问题的研究更好地表达了现实世界中实体的一对多语义约束关系。

（7）给出了求解路径依赖基、成员籍和最小依赖集的算法，并对这三个算法的可终止性、正确性和时间复杂度进行分析和证明。

（8）给出了 3XNF 和 4XNF 定义，在此基础上给出了规范化规则和无冗余判定定理，进一步提出了 XML 文档规范化算法，并对算法的可终止性、正确性及时间复杂度进行分析和证明，最后通过实验证明该算法的有效性。

（9）讨论了 XML 文档基于路径集合和代价的相似性度量，这对 XML 文

档相似性度量，聚类 XML 文档树编辑距离度量提供更丰富的度量方法，给出了规范化 XML 文档相似性度量方法——集合度量方法、线性度量方法和代价度量方法。并提出基于权重代价的机器学习的相似性度量算法。实验证明该算法扩展了 XML 文档查询范围，提高了文档的查全率和查准率。

（10）论述了小枝模式查询处理和索引技术的相关研究现状，指出了现有小枝模式查询处理方法中存在的关键问题，结合已经存在的索引思想，给出了一种基于共享路径的索引 ISP，利用该索引技术可以高效地处理小枝模式查询。

（11）对基于 ISP 索引的小枝模式处理方法进行了比较深入的分析。首先分析了小枝模式处理的过程，针对其处理过程为每一个步骤提出了详细的处理方法或算法，并对基于 ISP 索引的小枝模式处理算法进行了实验分析。

（12）分析了小枝模式匹配的国内外研究现状和现有匹配方法的不足，阐述了小枝模式匹配中的基本概念，提出了基于小枝模式匹配的灵活匹配方法，并定义了相关匹配条件，并通过具体的实例对所提出的方法进行了性能分析和说明。

1.3.2　理论和实际意义

随着互联网时代的到来，数据越来越多地开始以网络在线的方式进行着存储、集成、发布和交换。由于 XML 具有跨平台、简单易用等特性，在很短的时间内就获得了广泛认同，在众多应用领域中，已成为一种被广泛使用的通用数据格式。

由于 XML 的复杂性，本书引入一个比较完整的数据依赖推理规则集以及数据依赖的蕴涵和覆盖概念，以扩充已有的对于约束的定义能力。XML 文档作为半结构化数据的特例，虽然它很容易表达来自不同源的数据，但是其所能表示的语义信息却相对有限。从这一角度看，本书的工作增强了 XML 数据的语义表现力。由于 Web 的开放性，XML 数据操作异常的危害性要远远大于关系数据操作异常的危害性。因此，对于引起 XML 数据操作异常的原因及消除操作异常的方法进行研究具有重要的意义。现有的关于 XML 与关系数据库之间的转换研究都是基于简单的映射方法，这种映射只是保持了结构信息，忽略了语义上的约束保持。另一方面，从关系数据库转换为 XML 数据库，增加了

XML 数据库设计的步骤。XML 数据已经成为 Internet 数据交换和传输标准，从数据库设计的角度考虑，直接设计 XML 数据库模式已成为必要。借鉴关系数据库规范化的研究方法，研究 XML 数据库模式的设计，完善了 XML 数据库规范化理论。在规范化的 XML 数据库上进行存储、集成、发布和传输交换数据，保证了数据在互联网上的一致性，提高数据质量，在存储效率和查询优化上具有重要的实用价值。

第 2 章　基本定义及符号

本章介绍了 XML 基础知识、形式化的 DTD、XML 文档和路径表达式等基本定义，并进一步定义了 XML 文档树、结点值相等和树元组等概念，为后面章节中的讨论作符号上的准备。为了便于 XML 数据库模式规范化的研究，本章从文献中引用来的定义和符号做了形式化上的统一处理。

2.1　XML 简介

2.1.1　XML 与标签

XML 是一种把数据表示为一个文本字符串的语言，这个文本字符串包括用于描述数据分布的“标签（tag）”。使用标签可以把文本和与它的内容或形式相关的信息聚集在一起。

标签用角括号“<”与“>”包含字符（如“<this>”）来区别字符数据（非标签文本）。因此，一个字符串、一个文档由标签和字符数据组成，这些结合起来就形成了元素。一个元素以一个开始标签开始，以一个结束标签结束。结束标签用角括号和一个用来区分开始标签的正斜杠“/”来表示，像“</this>”。标签提供一种机制来给文档添加元内容和结构信息。

第一个通用标记语言 GML（General Markup Language）发明于 1969 年，主要用于支持文档处理应用。1974 年出现了通用标记语言标准 SGML（Standard for General Markup Language），并且从 1978 年到 1980 年它不断被国际化标准组织修改和发展，以满足系统的独立性和国际交换的要求，这些要求使得用于处理当时已有的异构系统的广泛选择成为必要。

谈到 XML，不能不涉及超文本标记语言 HTML（Hyper Text Markup Language）。HTML 是符合 SGML 语法的一种固定格式的超文本标记语言，它是最早应用于网络信息传输的标记语言，也是近几年网上最普及的一种网页制作通用语言。因其格式固定，所以难以扩展，它主要用于显示信息的内容，侧重于 Web 页面表现形式的描述，大大丰富了页面的视觉效果，为推动

WWW 的蓬勃发展、推动信息和知识的网上交流发挥了不可替代的作用。但是，它并没有反映出信息的结构，它自身的特点使它蕴藏了许多危机，随着自身的不断发展，这些危机不但没有减弱，反而越来越突出，甚至成为 HTML 继续发展应用的障碍。HTML 的主要局限性在于：HTML 不允许对页面上的个别元素进行语义性的标注；HTML 只描述文件外貌，但无法涵盖任何实质内容，因此不适合用于明确的查询方式；HTML 无法延伸，因此无法根据特殊规格来定义自己的标签，也无法在文件内部使用格式。HTML 的标记集合完全可以用 XML 来定义。XML 与 HTML 都是用一对互相匹配的标签来标记信息，但 HTML 描述的是数据的显示方式，现在网上通用的文档语言是 HTML，尽管这也是一个存储、显示数据的可行的办法，但是它的效率和能力却非常有限。它至少存在以下三个严重的问题：首先，HTML 的显示方式内嵌于数据之中，如果某个时候需要改用表格来表示这些数据，那么它不得不重新编码所有的 HTML 文件；其次，在数据中寻找信息非常困难，需要编写一个脚本程序；最后，数据自身的逻辑不得不让位于 HTML 的语言规范的逻辑。而 XML 则侧重于描述数据本身，它遵循严格的语法要求，并且便于不同系统间的信息传输，具有良好的保值性。最重要的是，XML 的标签集合不是固定的，XML 不仅允许自定义一套标记，而且这些标记不必仅限于对显示格式的描述。XML 允许根据各种不同的规则来制定标记，用户可以根据自己的需要定义任何一种标签来描述自己文档中的数据元素。因此，XML 作为 Web 上的信息表示和交换的标准，而 HTML 作为在浏览器端显示的工具。从 XML 到 HTML 的转换工具是可扩展样式表语言 XSL（eXtensible Stylesheet Language）。总之，XML 的出现并不是为了取代 HTML，而是为了更好地在 Internet 上进行数据传输与交换。两者的详细比较见表 2-1。

表2-1　XML与HTML比较

内容	HTML	XML
可扩展性	不具有扩展性	是源标记语言，可用于定义新的标记语言
侧重点	侧重于如何表现信息	侧重于如何结构化地描述信息
语法要求	不要求标记的嵌套、配对等，不要求标记之间具有一定的顺序	严格要求嵌套、配对，并遵守 DTD 的属性结构
可读性及可维护性	难以阅读、维护	结构清晰，便于阅读、维护
数据和显示的关系	内容描述与显示方式整合为一体	内容描述与显示方式相分离

续表

内容	HTML	XML
保值性	不具有保值性	具有保值性
编辑及浏览工具	已有大量的编辑、浏览工具	编辑、浏览工具尚不成熟

XML 是 SGML 的一个配置，设计于 20 世纪 90 年代后期，它给 SGML 提供在 Web 环境中的可扩展能力。XML 不完全像 SGML，它不允许自由选择，只支持为 Web 作出的选择，因此它易于学习和使用。

SGML 允许内容和表示的分离。HTML 在同一个框架中混合了内容、表示和处理，这样使得文档难以使用和维护。XML 和与它有关的标准支持内容（作为抽象元素类型）、表示（作为格式对象——一组特定的元素类型）和处理（作为样式表）相分离，每部分都可以独立的发展，不需要在一个统一的框架中折中。

XML 通过 Web 传送。数据使用一个 XML 文档的结构来表示。元素和属性提供有用的元数据（关于数据的数据），并且用一种可以通过 Web 传送、被不同应用使用和被转化为满足不同需求的方式来组织数据。

2.1.2　XML 特性

XML 作为 SGML 的一个配置，最初被设计为通过分离内容和格式来面对大规模的电子出版业的挑战，这种把文档内容和格式分离开来的想法简化了开发和维护。具有不同专门技术的不同人员（或团队）可以独立的操作从文档中抽取不同信息。

XML 可以在 Web 上的应用或应用与用户之间交换数据。XML 本身并不给终端用户提供表示，但是当与样式表一起使用时，容易浏览 XML 文档。XML 文档提供一种方法来获取数据：通过可扩展样式表 XSL 或层叠样式表 CSS（Cascading Style Sheets）（提供把 XML 转化为 HTML 或其他表示的机制）控制 XML 在 Web 浏览器中的表示。

XML 另一个关键的优势就是集成数据和文档的能力。大多数语言或者在表示严格的、绝对的内容和数据结构上设计得好一点，或者在表示灵活的、自由形式的文档文本上设计得好一点，但是 XML 在两方面都做得很好。

XML 用一种灵活的、可扩展的表示来实现内容通信。因此，数据的描述可以反复地被优化为基础域的变化，这使得 XML 对具有一个复杂的、频繁修

订的组织的领域是特别有用的。

与其他数据交换语言相比，XML 具有以下一些特性。

（1）自描述性和可扩展性。首先，XML 允许用户创建自己的 DTD，从而可以产生适合多种应用的“可扩展的”标记集。其次，使用几个附加的标准，用户还可以对 XML 自身进行扩展，向 XML 核心的功能集增加样式、链接以及引用能力。基于这个特点，XML 可作为一个核心标准，为其他标准的产生提供坚实的基础。

（2）可分析性。HTML 主要描述页面的显示形式，因此不能从 HTML 文档中理解所显示内容的实际含义。而 XML 则提供了功能强大、灵活高效地表达数据内容的方法，且其数据内容与具体的应用无关，使得用它表达的数据有很好的使用效率和可重用性。通过结合 DTD 的分析，人们可以理解 XML 文档中各个元素的含义，即 XML 文档具有自解释性。这些特点方便了对网络数据的统计分析，也改变了搜索和组织信息的方式（只需要通过分析标签就可以找到真正相关的信息）。

（3）简单性。XML 文档语法包括一个非常小的规则集，使得开发者可以根据它立刻开始工作。同时根据这种文档的结构，可以创建自己的 DTD 以满足自己的需要；XML 的严格定义和规则集，使人和机器都能很容易地阅读文档，根据它创建 XML 的语法分析器也比较容易。

（4）开放性。XML 标准自身在 Web 上是完全开放的，人们可以对任何一个文档进行语法分析，如果得到了相应的 DTD，还可以校验它。当然，开发者可以以自己的方式进行加密，XML 并不禁止创建自己私有的格式，但是它的开放性是它最大的优点，所以加密者也将失去使用 XML 的不少好处。

由于 XML 的这些特点，XML 被普遍使用的 Web 浏览器所支持，这样就降低了传送的成本。XML 作为 W3C 的推荐技术—— 对比较大的系统的开发具有一定级别的保证，因为较大的软件公司（如 Microsoft，IBM，Sun 和 Oracle）都支持这个技术。这些对数据库来说是特别重要的，因为大的数据库系统一定非常昂贵，并且企业的重要部分可能依赖于它们。

2.1.3　XML 数据库

XML 数据库是一个 XML 文档的集合，这些文档是持久的并且是可以操作的。XML 文档趋向于面向文档处理，或趋向于面向数据处理。面向文档处

理的文档是使用 XML 来获取自然语言的那些文档，侧重于给用户提供信息的最终表示。面向数据处理的文档是 XML 主要用于数据传送的文档，侧重于被应用程序的使用和交换。

XML 数据是一个面向数据处理的文档的集合。如前所述，XML 在数据库与应用程序之间，以及在多对应用程序之间进行数据交换是很有用的。在数据交换中，被交换的数据可能需要保留在一个存档文件中。一个可搜索的 XML 数据库可以作为数据交换的中心部件，并且它也可以与其他使用 XML 的应用程序进行交互，或者获取 Web 服务的摘要信息。

如果数据库把数据保存为 XML，那么在存储粒度上存在几个选择。XML 数据可以被保存为一个文档、被分割为更小的部分并被保存为片段，或者被分解为单独的元素。即使数据不保存为 XML，也可以作为 XML 被输出，提供给其他应用程序或用户。通过解析文档，并把数据保存在数据库结构（如关系）中，数据可以作为 XML 被添加到数据库中。

XML 数据库可分为以下三类。

（1）Native XML database。它专门为存储管理 XML 文档而设计，存储了 XML 模型的所有信息。数据以 XML 文档形式存进数据库，当用户需要数据时，Native XML database 仍然以文档形式输出。但其本身不可能是一个单独的数据库形式存在。文献对它作了具体的介绍。

（2）XML-Enabled database。它的设计是为了存储和取回以数据为中心的数据，同时提供了其他数据形式的接口，并包含了 XML 文档与其本身的数据传输。文献简述了它与 Native XML database 的区别。

（3）Hybrid database。从某种意义上讲，它是 Native XML database 和 XML- Enabled database 的一种混合，它可以粗略看成是 XML 和其他数据的接口。

2.1.4　XML 约束

下面通过几个例子分析一下在 XML 领域中约束的一些相关问题。

例 2.1　某所高校描述课程（*course*）、学生（*student*）和教师（*teacher*）实体之间的关系，采用关系数据库来存储数据，数据库设计的关系模式为

course(**cno**, cname)

student(**sno**, sname)

teacher(**tno**, tname)

courses(**cno**, **sno**, **tno**, cname, sname, tname)

这个数据库模式很直观，包括课程信息（课程号，课程名），学生信息（学号，学生姓名），教师信息（职工号，教师姓名）和选课信息（课程号，学号，职工号，课程名，学生姓名，教师姓名）。cno、sno、tno 和（cno, sno, tno）分别是相应关系的键。另一方面，在选课关系 *courses* 中，cno、sno 和 tno 也是外键，分别引用 *course*、*student* 和 *teacher* 关系中的相应元组。

现在假设该校出于数据交换的考虑，需要将数据库中的数据以 XML 文档的形式来发布，设计的相应 DTD D_0 为

```
<!ELEMENT courses (course*)>
<!ELEMENT course (cname, takenby)>
    <!ATTLIST course
            cno CDATA #REQUIRED>
<!ELEMENT cname (#PCDATA)>
<!ELEMENT takenby (student*)>
<!ELEMENT student (sname, teacher)>
        <!ATTLIST student
            sno CDATA #REQUIRED>
<!ELEMENT sname (#PCDATA)>
<!ELEMENT teacher (tname)>
        <!ATTLIST teacher
            tno CDATA #REQUIRED>
<!ELEMENT tname (#PCDATA)>
```

从语义约束的角度来说，这个 DTD 的定义是不完善的，它丢失了原有的数据库中键和外键的定义，这可能会带来不利的影响。直观来说，在转化的过程中，或许有某个课程的信息会被重复，也可能某个选课信息中会出现根本不存在的课程号或是学号。如果没有完整性约束的定义，相应的 XML 文档就无法避免类似情况的产生。为此，可以为模式定义增加下面形式化的约束表示。

（1）键：

course[cno] →*course*

student[sno] →*student*

teacher[tno] →*teacher*

courses[cno, sno, tno] →*courses*

（2）外键：

courses[cno] ⊆ *course*[cno]

courses[sno] ⊆ *student*[sno]

courses[tno] ⊆ *teacher*[tno]

上面的约束表示的含义：在相应的 XML 文档中，任意两个 *course* 元素的 cno 值必须不同，任意两个 *student* 元素的 sno 值必须不同，任意两个 *teacher* 元素的 tno 值必须不同；*courses* 元素的 cno 值必须在 *course* 元素的 cno 值中出现过，*courses* 元素的 sno 值必须在 *student* 元素的 sno 值中出现过，*courses* 元素的 tno 值必须在 *teacher* 元素的 tno 值中出现过。

但是需要注意的是，即使是这样简单的约束也是无法在 DTD 中表示的。因为按照 DTD 的规范，键只能被定义在属性上。在上面的定义中，即使将 cno，sno 和 tno 都定义为属性，而不是元素，也依然无法表示元素 *courses* 上的键。这是因为在 DTD 规范中，键只能被定义在单个属性上。从这个简单的例子可以看出，DTD 中对于键的定义方法是非常不完善的。

例 2.2　DTD $D_1(E_1, A_1, P_1, R_1, r_1)$描述的是一个学术会议（*Conference*）的信息，表示会议多个主题（*Topic*）、参与讨论各个主题的作者（*Author*）和每个主题的主持人（*question-Master*）实体之间的关系。D_1 定义为

```
<!ELEMENT Conference (Topic*)>
<!ELEMENT Topic (title, discussby)>
  <!ATTLIST Topic
        Tno CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT discussby (Author*)>
<!ELEMENT Author (Aname, question-Master)>
    <!ATTLIST Author
        Ano CDATA #REQUIRED>
<!ELEMENT Aname (#PCDATA)>
<!ELEMENT question-Master (Mname)>
    <!ATTLIST question-Master
        Mno CDATA #REQUIRED>
<!ELEMENT Mname (#PCDATA)>
```

图 2-1 是一个符合 DTD D_1 的 XML 文档树示意图，为了简单起见，图 2-1

中省略了具体的元素和属性值。为了区分元素和属性，在属性前加上“@”。其语义约束为一个学术会议有多个主题，每个主题有一个主持人和多个作者参加讨论，一个主持人可以主持多个主题（在不同的时间里）。

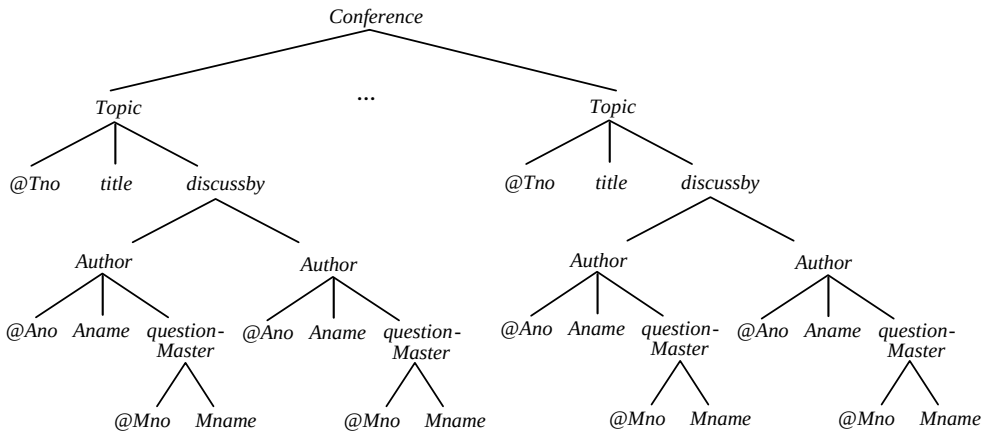

图 2-1　符合 DTD D_1 的 XML 文档树示意图

分析可知，这个 XML 文档中存在的约束如下。

（1）主题号决定主题。

（2）作者号在整个会议中是唯一的，可以决定作者姓名。

（3）主持人编号在整个会议中是唯一的，可以决定主持人姓名。

（4）一个作者可以参加多个主题的讨论。

（5）一个主持人可以主持多个主题。

和前面的例子一样，可以看到在 XML 文档中，数据的依赖关系是存在于文档的树型结构内的。在主题中，主题号是唯一标识，可以决定主题的名称。而在作者信息里，作者号又是作者的标识，可以决定作者的姓名，这些都类似于前面的键的概念。另外，还有一些约束并不是键，如要求每个作者可以参加多个主题的讨论，一个主持人可以主持多个主题。这就存在着一些更为复杂的情况。作者号这个属性，既在整个会议的范围内可以唯一决定作者姓名，又在一个主题内，可以决定主持人的姓名。也就是说，它在两个不同的范围内，起到了两个不同的键的作用。前面一个键是绝对（或全局）的，在整个文档上成立；而后一个键是相对（或局部）的，只在一个具体的主题内成立。

通过对约束的描述，很容易地发现该文档中存在着数据冗余。这在下面的 XML 文档实例中可以看出

```
<Conference>
    <Topic    Tno="T01">
        <title> DB </title>
        <discussby>
            <Author    Ano="A01">
                <Aname> Joe </Aname>
                    <question-Master    Mno="M01">
                        <Mname> John </Mname>
                    </question-Master>
                </Aname>
            </Author>
            <Author    Ano="A02">
                <Aname> Tom </Aname>
                    <question-Master    Mno="M01">
                        <Mname> John </Mname>
                    </question-Master>
                </Aname>
            </Author>
        </discussby>
    </Topic>
    <Topic    Tno="T02">
        <title> DW </title>
        <discussby>
            <Author    Ano="A02">
                <Aname> Tom </Aname>
                    <question-Master    Mno="M01">
                        <Mname> John </Mname>
                    </question-Master>
                </Aname>
            </Author>
            <Author    Ano="A03">
                <Aname> Jan </Aname>
                    <question-Master    Mno="M01">
                        <Mname> John </Mname>
                    </question-Master>
                </Aname>
            </Author>
        </discussby>
    </Topic>
</Conference>
```

根据上面文档的数据约束的分析，可以认定该文档不具有一个设计良好的 DTD。相关的问题就是，如何判定一个 DTD 是否是设计良好的；进一步地，假设某个 DTD 不是设计良好的，那么采用什么样的方式可以找到一个与之表述语义相同而设计良好的 DTD 呢？和关系模式相类似，XML 中存在数据冗余的原因是由于文档内部数据之间存在着某种依赖关系。首先需要一种表示这种依赖关系的方法，引入关系模式中的概念，称其为 XML 上的函数依赖，显然它比平面或是嵌套关系中的函数依赖都要复杂得多，也超出了已有的 DTD 和 XML-Schema 中键可以表达的范畴。由于文档和 DTD 是树型模型，XML 上的函数依赖的描述需要针对这种层次模型来表达。本书是基于路径表达式的方式来描述层次结构。在给出了这样一个函数依赖的定义后，就可以将 XML 上数据约束统一到这个框架上来。

与数据依赖相关的一个问题是逻辑蕴涵，其在 XML 领域中的含义：给定一个数据依赖集 Σ 和一个数据依赖 d，是否可以判定任意使 Σ 成立的 XML 文档 D，必然也使 d 成立；被 D 所逻辑蕴涵的全部数据依赖，就表示所有的约束关系。上述的两个问题对应的是数据依赖的成员籍、闭包问题。如果忽略 D 上的某个数据依赖 d，其他数据依赖仍然能够完全表示 D，对应的就是数据依赖集的等价表示问题，即覆盖问题。推理规则的引入通常是解决上述问题的主要手段，这些都是在 XML 领域中需要讨论的问题。考虑到 XML 自身的复杂性，它的推理规则显然要比关系上的推理规则复杂得多。

在数据依赖和逻辑蕴涵的基础上，就可以形式化地定义 XML 文档中存在的冗余。而要消除文档中存在的数据冗余，归根结底是需要一个设计良好的 DTD。那么接下来的问题就是判定怎样才是一个可以消除对应文档中冗余的 DTD，即规范化的 DTD；并进一步地讨论如何将给定的 DTD 转化为规范化的形式。关系数据库中函数依赖能有效地表达了属性值之间的一对一、多对一联系，它是现实世界中广泛存在也是最基本的数据依赖。但函数依赖还不足以描述现实世界中所有的数据依赖。例如，函数依赖就不能描述属性值之间的一对多联系，同关系数据库一样，本书还将研究 XML 领域的多值依赖来描述元素中部分一对多的联系，包括多值依赖的简化、蕴涵、覆盖以及推理规则等。

2.2　DTD

定义 2.1（文档类型定义，DTD）　DTD 定义为一个六元组 $D=(E, A, P, R, r)$，其中：

- D 为文档类型定义名，它是符号化的元组语义。
- E 表示元素类型的有限集合。
- A 表示属性的有限集合。
- P 表示从 E 到元素类型定义的映射：对于每一个 $\tau \in E$，$P(\tau)=S$ 或 $P(\tau)$ 是如下定义的正则表达式

$$\alpha ::= S \mid \varepsilon \mid \tau \mid \alpha \mid \alpha \mid \alpha,\alpha \mid \alpha^*$$

其中，S 表示字符串类型（元素类型声明中的#PCDATA），ε是空序列（元素类型声明中的 EMPTY），$\tau \in E$，“|”、“,”和“*”分别表示并、连接和 Kleene 闭包。

- R 表示从 E 到 A 的幂集 $\rho(A)$ 的映射，对于任意 $\tau \in E$，如果 $@l \in R(\tau)$，称属性 $@l$ 是 τ 的定义。
- $r \in E$，是根元素类型，不失一般性，对于任意 $\tau \in E$，假定 r 不出现在 $P(\tau)$中。

这里用 *El* 表示元素名集合，*Att* 表示属性名集合，*Str* 表示字符串属性值集合，*Vert* 表示结点标签集合，所有的属性名均以符号@开头。

例 2.3　在例 2.2 给出的 DTD D_1 定义中有

E={*Conference*，*Topic*，*title*，*discussby*，*Author*，*Aname*，*question-Master*，*Mname*}

A={*Tno*，*Ano*，*Mno*}

P(*Conference*)=*Topic**

P(*Topic*)={*title*，*discussby*}

P(*title*)=*P*(*Aname*)=*P*(*Mname*)=*S*

P(*discussby*)=*Author**

P(*Author*)={*Aname*，*question-Master*}

P(*question-Master*)=*Mname*

R(*Conference*)=*R*(*title*)=*R*(*discussby*)=*R*(*Aname*)=*R*(*Mname*)=*Φ*

R(*Topic*)=*Tno*

R(*Author*)=*Ano*

R(*question-Master*)=*Mno*

r=*Conference*

定义 2.2（DTD 路径） 给定一个 DTD $D=(E, A, P, R, r)$，D 中的路径定义为 $p=v_1.\cdots.v_n$，其中：$v_1=r$，$v_i\in P(v_{i-1})$，$i\in[2, n-1]$，$v_n\in P(v_{n-1})$或 $v_n=@l$，$@l\in R(v_{n-1})$，且 $length(p)=n$，$last(p)=v_n$。用 $Paths(D)$表示 D 中所有路径构成的集合；$EPaths(D)=\{p|p\in Paths(D)$且 $last(p)\in E\}$，表示以元素类型结尾的路径构成的集合；$APaths(D)=\{p|p\in Paths(D)$且 $last(p)\in A\}$，表示以属性结尾的路径构成的集合。

例 2.4 在例 2.2 中给出的 DTD D_1 定义中有：*Conference.Topic.@Tno* 和 *Conference.Topic.discussby.Author.Aname* 都是该 DTD 上的路径，前者属于 $APaths(D)$，后者属于 $EPaths(D)$。

2.3 XML 树

定义 2.3（XML 树） 给定 DTD $D=(E, A, P, R, r)$，称七元组 $T=(V, lab, ele, att, val, root)$为满足（或符合）$D$ 的 XML 树，记作 $T\vDash D$，其中：

- T 为 XML 树名，它是符号化的元组语义。
- V 是结点的有限集合。
- lab 是从 V 到 $E\cup A\cup S$ 的映射，对于任意结点 $v\in V$：如果 $lab(v)=\tau$ 且 $\tau\in E$，v 称为元素结点；如果 $lab(v)\in A$，v 称为属性结点；如果 $lab(v)=S$，v 称为文本结点。
- ele 是从 V 到 V^*的部分映射，满足对任意 $v\in V$，都有 $ele(v)=[v_1.\cdots.v_n]$ 且 $lab(v_i)\in P(\tau)$（$i\in[1, n]$）；
- att 是从 V 到 A 的部分映射，满足对任意 $v\in V$ 和$@l\in A$，$att(v, @l)$有定义，当且仅当 $lab(v)=\tau$，$\tau\in E$ 和$@l\in R(\tau)$。
- val 是从 V 到字符串值的部分映射，满足对任意 $v\in V$，$val(v)$有定义，当且仅当 $lab(v)=S$ 或者 $lab(v)\in A$。
- $root$ 是 V 中唯一结点，$lab(root)=r$ 称为 T 的根。

图 2-2 就是符合例 2.2 中定义的 DTD D_1 的一个 XML 文档树实例。

定义 2.4（XML 路径） 给定 DTD $D=(E, A, P, R, r)$和满足 D 的 XML 树 $T=(V, lab, ele, att, val, root)$，$T$ 中的路径定义为 $p=v_1.\cdots.v_n$，其中：$v_1=root$，$v_i\in ele(v_{i-1})$，$i\in[2, n-1]$；如果 $lab(v_n)\in E$，那么 $v_n\in ele(v_{n-1})$，称路径 p 为元素结点类型路径；如果 $lab(v_{n-1})\in A$，那么 $v_n\in att(v_{n-1})$或者 $P(lab(v_{n-1}))=S$，那么 $v_n=S$，称路径 p 为值类型路径。

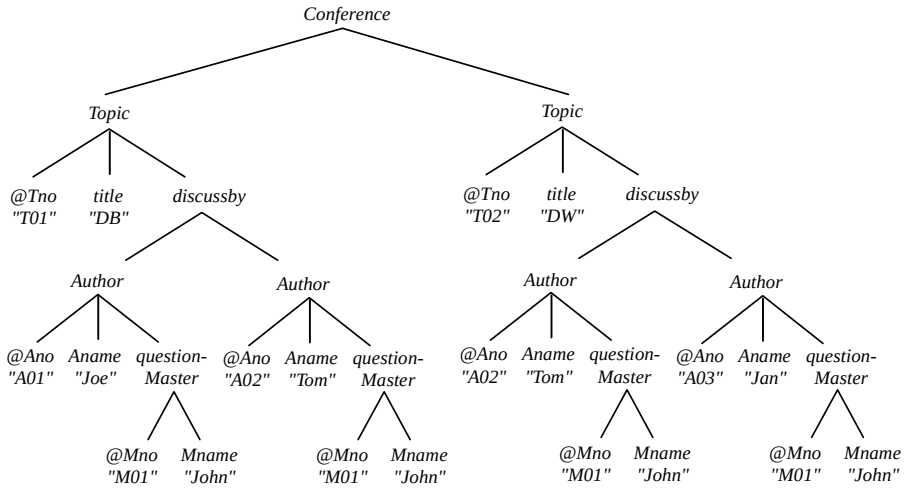

图 2-2　符合例 2.2 中 DTD D_1 的 XML 文档树实例

令 $last(p)=v_n$，表示路径 p 中最后一个结点；$Paths(T)=\{p|p$ 是 T 中的路径$\}$，表示 T 中所有的路径的集合；$EPaths(T)=\{p|p\in Paths(T)$且 $last(p)\in E\}$，表示元素结点类型路径的集合；$VPaths(T)=\{p|p\in Paths(T)$且 $last(p)\in A$ 或 $last(p)\in S\}$，表示值类型路径的集合。事实上，XML 树 $T\vDash D$，T 中的路径也是 D 中相应路径的映射。

定义 2.5（树的包含）　给定两棵 XML 树 $T_1=(V_1, lab_1, ele_1, att_1, val_1, root_1)$ 和 $T_2=(V_2, lab_2, ele_2, att_2, val_2, root_2)$，称 T_2 包含 T_1，记作 $T_1\subseteq T_2$，当且仅当下列条件满足：

- $V_1\subseteq V_2$。
- $root_1=root_2$。
- $\forall v\in V_1$，$lab_1(v)=lab_2(v)$。
- $\forall v\in V_1$，$@l\in A$，$att_1(v, @l)=att_2(v, @l)$。
- 对于所有 $v\in V_1$，$ele_1(v)$为 $ele_2(v)$排列变化后的子列。

如果 $T_1\subseteq T_2$ 且 $T_2\subseteq T_1$，那么 T_1 和 T_2 是等价的，记作 $T_1\equiv T_2$，将以此划分的 T 的等价类记作$[T]$。如果对某个 $T_1\in[T]$，满足 $T_1\vDash D$，记作$[T]\vDash D$。显而易见，对任意的 $T_1\equiv T_2$，有 $Paths(T_1)=Paths(T_2)$。

当且仅当 $Paths(T)\subseteq Paths(D)$时，称 T 与 D 可容（记作 $T\lhd D$）。

定义 2.6（树元组）　给定 DTD $D=(E, A, P, R, r)$和满足 D 的 XML 树 $T=(V, lab, ele, att, val, root)$，树元组 t 定义为 $Paths(T)$到 $V\cup S\cup\{\perp\}$ 的映射，满足：

- 如果 $p\in EPaths(T)$，那么 $t[p]\in V\cup\{\perp\}$，且 $t[p]\neq\perp$。

- 如果 $p \in VPaths(T)$，那么 $t[p] \in S \cup \{\perp\}$。
- 如果 $t[p_1]=t[p_2]$且 $t[p_1] \in V$，那么 $p_1=p_2$。
- 如果 $t[p_1]=\perp$且 p_1是 p_2的前缀，那么 $t[p_2]=\perp$。
- $\{p \in Paths(T) | t[p] \neq \perp\}$是有限的。

其中，S和$\perp$分别表示字符串值（文本）和空值。令 $\mathcal{T}[T]=\{t|t \in T\}$表示树元组的集合，树元组 $t[p]$可记作 $t.p$。

例 2.5　图 2-2 文档树中的部分树元组为

$t[Conference.Topic]=Topic$	元素类型
$t[Conference.Topic.@Tno]=\{T01, T02\}$	属性值类型
$t[Conference.Topic.title]=\{DB, DW\}$	字符串（文本）类型

2.4　结点值相等

关系数据库中的元组相等是严格意义上的值相等。由 XML 树的定义可以看到一个 XML 文档是典型的结点标签树，据此，接下来给出 XML 树上的值相等的定义。

定义 2.7（结点值相等） 给定 DTD $D=(E, A, P, R, r)$和满足 D 的 XML 文档树 $T=(V, lab, ele, att, val, root)$，$n_1$和 n_2是 V 中的结点，两个结点 n_1和 n_2的值相等，记为 $n_1=_v n_2$，当且仅当下列条件是满足的：

- $lab(n_1)=lab(n_2)$。
- 如果 n_1和 n_2是 A 或 S 结点，那么 $val(n_1)=val(n_2)$。
- 如果 n_1和 n_2是 E 结点，那么$\forall a_1 \in att(n_1)$，$\exists a_2 \in att(n_2)$，满足 $a_1=_v a_2$，反之亦然；如果 $ele(n_1)=v_1.\cdots.v_k$，$ele(n_2)=v'_1.\cdots.v'_k$，那么对于所有 $i \in [1, k]$，$v_i=_v v'_i$。

也就是说，$n_1=_v n_2$当且仅当它们的子树是同构的，即在字符串值上是恒等式。

例 2.6　图 2-2 中每个 *Author* 的 *question-Master* 子元素都是值相等的，第二个 *Author* 的 *Aname* 子元素与第三个 *Author* 的 *Aname* 子元素是值相等的。

2.5　其他定义与符号

定义 2.8　给定 DTD D，$\forall p_1, p_2 \in Paths(D)$，如果 p_1是 p_2的前缀，称 $p_1 \leqslant p_2$。当 $p_1 \leqslant p_2$且 $p_1 \neq p_2$时，记 $p_1 < p_2$。

例如，在例 2.2 的 DTD 中就有 *Conference.Topic*＜*Conference.Topic.title*。

定义 2.9　给定 DTD *D*，$p \in Paths(D)$，则 *p* 的祖先路径集由 *p* 的所有前缀组成，表示为 $Anc(p)=\{p_1, p_2, \cdots, p_{n-1}\}$，其中 $p_i \in Paths(D)$，$p_i \leqslant p$。

例如，在例 2.2 中的 DTD 中，Anc(*Conference.Topic.title*)＝{*Conference*, *Conference.Topic*, *Conference.Topic.title*}

定义 2.10　给定 DTD *D*，*D* 上两条路径 p_1, $p_2 \in Paths(D)$，其中，$p_1 = v_1.v_2...v_n.v_1' ...v_n'$，$p_2 = v_1.v_2...v_n.v_1''...v_m''$，$v_1' \neq v_1''$，则 p_1 与 p_2 的最大公共路径 $p_1 \cap p_2 = v_1.v_2...v_n$。

例如，*Conference.Topic.title*∩*Conference.Topic.discussby*＝*Conference.Topic*。

定义 2.11　给定 DTD *D* 以及 *D* 上的路径集 $P=\{p_1, p_2, \cdots, p_n\}$，其中，$p_i \in Paths(D)$，$1 \leqslant i \leqslant n$，如果 $\forall p_i \in \{p_1, p_2, \cdots, p_{n-1}\}$，$p_i \leqslant p_n$ 均成立，则称 $\max(P)=P_n$。如果不存在这样的 P_n，则称 max(*P*)不存在。

例如，P_1＝{*Conference.Topic*, *Conference.Topic.discussby*, *Conference.Topic.discussby.Author*}，则有 $\max(P_1)$=*Conference.Topic.discussby.Author*; P_2={*Conference. Topic.@Tno*, *Conference.Topic.title*, *Conference.Topic.* discussby}，则有 max(P_2)不存在。

定义 2.12　给定 DTD *D*，$p \in EPaths(D)$，$S \subseteq Paths(D)$，*Att*(*p*)表示 *last*(*p*)的属性路径集，*Att*(*S*)={*p.@l*|∀*p*∈*S*, *p*∈*EPaths*(*D*), ∀*@l*∈*A*, *att*(*last*(*p*), *@l*)有定义}。

例如，*Att*(*Conference.Topic*)={*Conference.Topic.@Tno*}

设 *S*={*Conference.Topic*, *Conference.Topic.discussby.Author*}，*Att*(*S*)={*Conference.Topic.@Tno*, *Conference.Topic.discussby.Author.@Ano*}

定义 2.13　给定 DTD *D*，*S*⊆*Paths*(*D*)，*S*.S={*p*.S|∀*p*∈*S*, *p*∈*EPaths*(*D*), *p*.S 在 *P*(*last*(*p*))中}。

例如，设 *S*={*Conference.Topic.title*, *Conference.Topic.discussby. Author. Aname*}，则 *S*.S={*Conference.Topic.title.S*, *Conference.Topic.discussby.Author. Aname*.S}。

2.6　小　　结

本章主要介绍了 XML 基础知识中 XML 与标签、XML 特性、XML 数据库、XML 约束；并进一步定义了形式化的 DTD、XML 文档树、节点值相等和树元组等概念，为后面章节中的论述作符号上的准备。

第 3 章　XML 函数依赖

在第 2 章中给出了 DTD 的形式化定义，但 DTD 语法不足以描述 XML 文档的所有语义信息。本章将介绍 XML 文档函数依赖的概念，进一步扩充 XML 文档语义表达能力。所谓的 XML 文档语义，本质上是指 XML 文档树结点数据之间的约束关系，或者说依赖关系。在关系数据库中，一个数据依赖是关于属性值之间的相关关系的一个命题，它规定了任何一个数据库的合法状态所必须满足的一个语义完整性的约束条件。数据依赖是指数据之间存在着各种内在的联系，比如键就是一种特殊的依赖。函数依赖是最常见、最重要的数据依赖，它是关系数据库设计理论的主要内容之一，是范式研究的基础。同样的语义约束，对于 XML 数据库模式是一样的。本章在回顾传统关系模式上函数依赖概念的基础上，分析 XML 数据库模式上函数依赖的要求，基于路径表达式，给出形式化的 XML 函数依赖定义，并在此基础上给出 XML 部分函数依赖和传递函数依赖的概念，定义了 XML 函数依赖的逻辑蕴涵与覆盖，给出了 XML 函数依赖推理规则，并证明其有效性和完备性，最后给出求解覆盖的算法。

3.1　XML 函数依赖定义

在给出 XML 函数依赖形式化定义之前，简单回顾一下关系数据库中函数依赖的概念。

定义 3.1（函数依赖） 设 $R(W)$是一个关系模式，$X, Y \subseteq W$，关系模式 R 上的一个函数依赖是形如 f：$X \rightarrow Y$ 的一个命题，它的含义：对于 R 的任意一个可能的实例 r，如果对任意 $t_1, t_2 \in r$，$t_1[X]=t_2[X]$，则必有 $t_1[Y]=t_2[Y]$。$X \rightarrow Y$ 读作“X 函数决定 Y”或“Y 函数决定于 X”。

由函数依赖定义可以看到：数据依赖关系是定义在属性上，且属性是原子值。元组相等是定义在关系表上严格意义的值相等。而 XML 半结构化数据是典型的树模型，数据约束比关系数据库要复杂得多，下面通过例子分析 XML 文档内部存在的数据依赖情况。

例 3.1　回顾 2.1.4 节中例 2.1 和例 2.2 中定义的 DTD D_0 和 DTD D_1。

从两个例子中可以看到：在 XML 文档中，数据的依赖关系是存在于文档的树型结构内的，并且 DTD 也并没有完全表达出语义上的约束。具体来说，体现如下。

（1）XML 文档的树型结构决定了 XML 上数据依赖成立的范围比较复杂。函数依赖定义的要素不再是“扁平”的关系模式中的属性，而应结合结构来考虑。

（2）XML 中参与数据依赖的不仅仅包括属性，还包括元素，而元素自身又有其复杂构造的。所以，讨论 XML 函数依赖时，“相等”的概念需要更复杂的考虑。

从数据库角度来看待 XML 文档，将 DTD 看作 XML 数据库模式，符合 XML DTD 定义的 XML 文档看作该模式定义下的数据库实例，以对待数据库的方式来操作 XML 文档。从这一初衷入手，自然希望可以为 XML 提供表述类似约束的能力。引入关系模式中的概念，称其为 XML 中的函数依赖。函数依赖是数据语义的重要组成部分，对于键的定义、完整性约束、模式设计、查询优化和数据集成等有非常重要的作用。与关系数据库理论类似，基于 XML 函数依赖，可以进一步地研究逻辑蕴涵、覆盖，并进行规范化的讨论。接下来，基于第 2 章给出的 DTD，XML 树模型和路径表达式等概念，给出形式化的 XML 上函数依赖（XML Functional Dependency，FD_{XML}）的定义。

定义 3.2（XML 函数依赖，FD_{XML}）　给定 DTD $D=(E, A, P, R, r)$，任意的 XML 文档树 $T \vDash D$，$Paths(D)$为 D 中所有路径的集合，$Paths(T)$为文档树 T 上的路径集合，D 上函数依赖 FD_{XML} 定义为如下形式：

$$\varphi: (H, [p_{x1}, ..., p_{xn} \rightarrow p_{y1}, ..., p_{ym}])$$

其中：

- H 是 XML 函数依赖 FD_{XML} φ 的头部路径，即从 XML 文档的根结点开始的路径，是一个完全限定路径表达式，定义约束保持的范围。
- $p_{x1}, ..., p_{xn}$ 是 XML 函数依赖 FD_{XML} φ 的左部路径，$p_{xi} \in Paths(T)$，每个 p_{xi} 是左部（Left-Hand-Side，LHS）实体类型，即 $last(p_{xi}) \in E \cup A \cup S$。
- $p_{y1}, ..., p_{ym}$ 是 XML 函数依赖 FD_{XML} φ 的右部路径，$p_{yi} \in Paths(T)$，每个 p_{yi} 是右部（Right-Hand-Side，RHS）实体类型，即 $last(p_{yi}) \in E \cup A \cup S$。

对于文档树 T 中任意两个树元组 t_1 和 t_2，在头部路径 H 约束保持范围内，

若存在$t_1[p_{xi}]=t_2[p_{xi}](1\leqslant i\leqslant n)$，一定有$t_1[p_{yj}]=t_2[p_{yj}](1\leqslant j\leqslant m)$。

出于简单方便的考虑，在后面的讨论中将头部路径写入到每个左部实体类型中，使每个左部实体类型都是一个完全限定路径。本书采用的定义与已有文献中的定义有类似的地方，但两个定义有着本质的不同：文献中的左部实体类型为XML 文档元素名或键属性名，本书的定义并没有强调左部必须为键属性，而只要求是 DTD 或文档树中路径即可。

扩展函数依赖（Extended Functional Dependency，EFD），虽然是在 XML 文档模式上定义的，但本书采用的路径表达式与其相比更符合 XML 文档树型结构的特点。本书的定义采用“树元组”，寻找 XML 文档与关系之间的对应关系，从而得到 XML 函数依赖的定义更为直观，更易于后面对 XML 函数依赖推理规则、逻辑蕴涵、函数依赖集闭包等概念的讨论。

例 3.2　考虑图 2-2 所示的 XML 文档，其中部分语义约束的函数依赖形式化表示为

（1）(*Conference.Topic*, [*Conference.Topic.discussby.Author.@Ano*]→
[*Conference.Topic.discussby.Author.question-Master.@Mno*])

（2）(*Conference.Topic*, [*Conference.Topic.@Tno*]→
[*Conference.Topic.discussby.Author.question-Master.@Mno*])

（3）(*Conference.Topic*, [*Conference.Topic.@Tno*]→
[*Conference.Topic.discussby.Author.question-Master.Mname.S*])

（4）(*Conference.Topic*, [*Conference.Topic.discussby.Author. question-Master.*
@Mno]→
[*Conference.Topic.discussby.Author.question-Master.Mname.S*])

定义 3.3（XML 平凡函数依赖） 给定 DTD $D=(E, A, P, R, r)$和 FD_{XML} φ，如果 φ 在任何满足 DTD D 上的文档树 T 都是满足的，即 $T\vDash D\vDash\varphi$，称 φ 是平凡的 XML 函数依赖。

同关系数据库一样，本书在无特殊声明的情况下，只考虑非平凡的 XML 函数依赖。

定义 3.4（XML 函数依赖集） 给定 DTD $D=(E, A, P, R, r)$，任意的 XML 文档树 $T\vDash D$，$Paths(T)$为文档树 T 上的路径集合，D 上函数依赖 FD_{XML} 集定义为如下形式：$\Sigma=\{\varphi|\varphi$ 是满足 DTD D 或 XML 树 T 上的函数依赖 $FD_{XML}\}$，表示所有 XML 函数依赖的集合。

定义 3.5（XML 部分函数依赖集，PFD_{XML}） 给定 DTD $D=(E, A, P, R, r)$，任意的 XML 文档树 $T \vDash D$，$Paths(T)$为文档树 T 上的路径集合，D 上函数依赖 φ：$(H,[p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}])$，如果存在另一个函数依赖 φ'：$(H',[H.p_{z1},...,H.p_{zk} \rightarrow p_{y1},...,p_{ym}])$，满足 $p_{x1},...,p_{xn} \nrightarrow H.p_{z1},...,H.p_{zk}$ 和 $H.p_{z1},...,H.p_{zk} \nrightarrow p_{x1},...,p_{xn}$。$H' \subseteq_{Paths} H \subset_{Paths} P_{xi} \subseteq_{Paths} P_{yl}$，$1 \leqslant i \leqslant n$，$1 \leqslant l \leqslant m$，称 φ 为部分函数依赖，记作 PFD_{XML}。

定义 3.5 的直观解释是：当把语义上非直接联系的元素类型组织成树型结构时，就会有部分函数依赖，这样，在 XML 文档中就会有数据冗余及异常现象。也就是说，$p_{y1},...,p_{ym}$ 是在以 $last(H')$为根的子树内和 $H.p_{z1},...,H.p_{zk} \rightarrow p_{y1},...,p_{ym}$ 有直接的函数依赖关系，即函数依赖 $(H',[H.p_{z1},...,H.p_{zk} \rightarrow p_{y1},...,p_{ym}$ 成立，而和 $p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$ 的函数依赖关系是通过 H 所约束的，也就是通过 DTD 中元素类型之间的树状结构所形成的一种部分函数依赖关系。

例 3.3　图 2-2 所示的 XML 文档中存在下列函数依赖。

φ_1：(*Conference*.*Topic*, [*Conference*.*Topic*.discussby.*Author*.*@Ano*] →
[*Conference*.*Topic*.*discussby*.*Author*.*question-Master*.*@Mno*])

φ_2：*Conference*.*Topic*.*@Tno*→
[*Conference*.*Topic*.*discussby*.*Author*.*question-Master*.*@Mno*])

而 *Conference*.*Topic*.discussby.*Author*.*@Ano*↛*Conference*.*Topic*.*@Tno* 且 *Conference*.*Topic*.*@Tno* ↛*Conference*.*Topic*.discussby.*Author*.*@Ano* 由定义 3.5 可知，φ_1 就是部分函数依赖。

定义 3.6（XML 传递函数依赖集，TFD_{XML}） 给定 DTD $D=(E, A, P, R, r)$，任意的 XML 文档树 $T \vDash D$，$Paths(T)$为文档树 T 上的路径集合，D 上函数依赖 φ：$(H,[p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}])$，如果存在另外两个函数依赖 φ'：$(H,[p_{x1},...,p_{xn} \rightarrow p_{z1},...,p_{zk}])$ 和 φ''：$(H',[p_{z1},...,p_{zk} \rightarrow p_{y1},...,p_{ym}])$，满足 $(H,[p_{z1},...,p_{zk} \nrightarrow p_{x1},...,p_{xn}])$ 和 $p_{y1},...,p_{ym} \nrightarrow p_{z1},...,p_{zk}$。$H' \subseteq_{Paths} H$，称 φ 为传递函数依赖，记作 TFD_{XML}。

例 3.4　图 2-2 所示的 XML 文档中存在下列函数依赖。

φ_3 : *Conference*.*Topic*.*@Tno*→
Conference.*Topic*.*discussby*.*Author*.*question-Master*.*Mname*.*S*

φ_4 : *Conference*.*Topic*.*@Tno*→
Conference.*Topic*.*discussby*.*Author*.*question-Master*.*@Mno*

φ_5 : *Conference.Topic.discussby.Author.question-Master.@Mno*→
Conference.Topic.discussby.Author.question-Master.Mname.S

并且 *Conference.Topic.discussby.Author.question-Master.@Mno*↛*Conference.Topic.@Tno*，那么根据定义 3.6 可知φ_3是传递函数依赖。

由于存在上述的数据依赖关系，造成 XML 文档 D_1 包含冗余信息：主持人“*John*”的信息在同一主题（*Topic*）的每个作者（*Author*）的子元素内都要重复存储；作者“*A02*”的姓名“*Tom*”在他所参加的每一主题中都要重复存储。如果这些信息发生改变，在相应的子元素下面都要同时更新，否则在 XML 文档中就会产生数据不一致，这种现象称为更新异常。这种异常现象正是由于 DTD D_1 在设计上存在问题。为了避免这种异常现象，需要对 DTD 模式进行规范化处理。规范化内容将在下一章进行介绍。

3.2　XML 函数依赖蕴涵问题

由定义 3.5 和定义 3.6 可知，XML 文档数据之间的依赖关系并非是相互独立的。它们之间存在着相互蕴涵关系。下面就给出蕴涵等相关概念。

定义 3.7（FD_{XML} 的逻辑蕴涵） 给定 DTD $D=(E, A, P, R, r)$，Σ 是 DTD 上的一个 FD_{XML} 集，φ是某一个 FD_{XML}，如果每一个满足 Σ 中的全部 FD_{XML} 的文档也都满足φ，即 $SAT(\Sigma)\subseteq SAT(\varphi)$，则称$(D, \Sigma)$逻辑地蕴涵$\varphi$，记作$(D, \Sigma)\models\varphi$，通常简记为$\Sigma\models\varphi$。

定义 3.8（FD_{XML} 集的闭包） 设 Σ 是路径集 *Paths*(*D*)上的一个 FD_{XML} 集，由 Σ 所逻辑蕴涵的所有 FD_{XML} 的集合，称为(D, Σ)的闭包，记作$(D, \Sigma)^+$，通常简记为Σ^+，即$(D, \Sigma)^+=\Sigma^+=\{ p_{x1},..., p_{xn}\rightarrow p_{y1},..., p_{ym} \mid \Sigma\models p_{x1},..., p_{xn}\rightarrow p_{y1},..., p_{ym}\}$。

XML 文档数据之间的非互独立性，导致某些数据依赖可能逻辑地蕴涵其他数据依赖。从例 3.3 和例 3.4 中也能够看出数据之间的依赖关系。实际上，数据之间的蕴涵关系会引出一些新的问题。

（1）已知某个路径集 *Paths*(*D*)及其数据依赖集 Σ，要求找出被 Σ 所逻辑蕴涵的全部数据依赖。

（2）已知某个路径集 *Paths*(*D*)及其数据依赖集 Σ，存在一个数据依赖φ，要求确定是否$\Sigma\models\varphi$。

（3）已知某个数据依赖集 Σ，如果 Σ 中存在某个数据依赖φ使$\{\Sigma-\varphi\}\models\varphi$成立，

那么 Σ–φ 完全能够表示 Σ，这就引出了一个数据依赖的最简等价表示问题。

上述三个问题，通常称为数据依赖的闭包问题、成员籍问题和最小覆盖问题，它们对于数据库设计的研究具有十分重要的意义。

在关系数据库中最早被研究的相关问题称为 Armstrong 推导公理（或简称 FD 公理）。在 XML 数据中同样存在着数据的蕴涵和等价表示等问题，接下来，给出 XML 函数依赖的推理规则。

3.3　XML 函数依赖推理规则

FD_{XML} 推理规则可表述如下。

给定 DTD $D=(E, A, P, R, r)$，任意的 XML 文档树 $T \vDash D$，$\{ p_{x1},..., p_{xn}$，$p_{y1},..., p_{ym}$，$p_{z1},..., p_{zk}$，$p_{w1},..., p_{wl} \} \subseteq Paths(D)$，则

- $FD0_{XML}$（自反规则）：$\{ p_{y1},..., p_{ym} \subseteq p_{x1},..., p_{xn} \subseteq Paths(D)\} \vdash p_{x1},..., p_{xn} \rightarrow p_{y1},..., p_{ym}$；
- $FD1_{XML}$（增广规则）：$\{ p_{z1},..., p_{zk} \subseteq p_{w1},..., p_{wl}$，$p_{x1},..., p_{xn} \rightarrow p_{y1},..., p_{ym} \} \vdash \{ p_{x1},..., p_{xn} \cup p_{w1},..., p_{wl} \} \rightarrow \{ p_{y1},..., p_{ym} \cup p_{z1},..., p_{zk} \}$；
- $FD2_{XML}$（传递规则）：$\{ p_{x1},..., p_{xn} \rightarrow p_{y1},..., p_{ym}$，$p_{y1},..., p_{ym} \rightarrow p_{z1},..., p_{zk} \} \vdash p_{x1},..., p_{xn} \rightarrow p_{z1},..., p_{zk}$；
- $FD3_{XML}$（合并规则）：$\{ p_{x1},..., p_{xn} \rightarrow p_{y1},..., p_{ym}$，$p_{x1},..., p_{xn} \rightarrow p_{z1},..., p_{zk} \} \vdash p_{x1},..., p_{xn} \rightarrow \{ p_{y1},..., p_{ym} \cup p_{z1},..., p_{zk} \}$；
- $FD4_{XML}$（投影规则）：$\{ p_{x1},..., p_{xn} \rightarrow p_{y1},..., p_{ym}$，$p_{z1},..., p_{zk} \subseteq p_{y1},..., p_{ym} \} \vdash p_{x1},..., p_{xn} \rightarrow p_{z1},..., p_{zk}$；
- $FD5_{XML}$（伪传递规则）：$\{ p_{x1},..., p_{xn} \rightarrow p_{y1},..., p_{ym}$，$p_{w1},..., p_{wl} \cup p_{y1}, ..., p_{ym} \rightarrow p_{z1},..., p_{zk} \} \vdash \{ p_{w1},..., p_{wl} \cup p_{x1},..., p_{xn} \} \rightarrow p_{z1},..., p_{zk}$。

为了证明这一推理规则集的有效性和完备性，这里首先给出有关形式化定义。

定义 3.9（FD_{XML} 的有效性与完备性）　设 R 是 FD_{XML} 一个推理规则集，Σ 是一个 XML 函数依赖集，φ 是某个函数依赖，如果规则集 R 可以从 Σ 导出（或证明）φ，记作 $\Sigma \vdash_r \varphi$。如果对于任一 $\Sigma \vdash_r \varphi$ 都逻辑地蕴涵着 $\Sigma \vDash \varphi$，则称推理规则集是有效的。反之，如果对任一 $\Sigma \vDash \varphi$，也必定有 $\Sigma \vdash_r \varphi$，则称推理规则集是完备的。

实际上，所谓的 FD_{XML} 有效性，就是指由推理规则推理出来的函数依赖存在于函数依赖集闭包 Σ^+ 中；FD_{XML} 完备性就是指函数依赖集闭包 Σ^+ 中的函数依赖通过推理规则集可以推理出来。接下来就给出 FD_{XML} 的推理规则的有效性和完备性证明。

3.3.1　推理规则正确性

定理 3.1　$FD0_{XML}$～$FD5_{XML}$ 对于 FD_{XML} 之间的逻辑蕴涵关系的推理是有效的。

（1）用反证法来证明 $FD0_{XML}$ 的有效性。

证明：假设存在 $p_{y1},...,p_{ym} \subseteq p_{x1},...,p_{xn}$，而 $p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$ 不成立，那么由 XML 函数依赖定义可知，存在树元组 t_1，t_2，满足 $t_1[p_{xi}]=t_2[p_{xi}]$（$1 \leqslant i \leqslant n$），一定有 $t_1[p_{yj}] \neq t_2[p_{yj}]$（$1 \leqslant j \leqslant m$）。因为已知 $p_{y1},...,p_{ym} \subseteq p_{x1},...,p_{xn}$，由 $t_1[p_{xi}]=t_2[p_{xi}]$ 可以推出 $t_1[p_{yj}]=t_2[p_{yj}]$，这与假设矛盾，所以假设不成立。$FD0_{XML}$ 的有效性得到证明。

（2）用反证法来证明 $FD1_{XML}$ 的有效性。

证明：假定在某个文档树 $T \vDash D$ 上，$p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$ 成立，而 $\{p_{x1},...,p_{xn} \cup p_{w1},...,p_{wl}\} \rightarrow \{p_{y1},...,p_{ym} \cup p_{z1},...,p_{zk}\}$（其中 $p_{z1},...,p_{zk} \subseteq p_{w1},...,p_{wl}$）不成立，即在 $Paths(T)$ 上存在树元组 t 和 s，$t[p_{x1},...,p_{xn} \cup p_{w1},...,p_{wl}] = s[p_{x1},...,p_{xn} \cup p_{w1},...,p_{wl}]$，而 $t[p_{y1},...,p_{ym} \cup p_{z1},...,p_{zk}] \neq s[p_{y1},...,p_{ym} \cup p_{z1},...,p_{zk}]$，但是，由于 $p_{z1},...,p_{zk} \subseteq p_{w1},...,p_{wl}$，所以 $t[p_{z1},...,p_{zk}]=s[p_{z1},...,p_{zk}]$，故只能 $t[p_{y1},...,p_{ym}] \neq s[p_{y1},...,p_{ym}]$，这与 $p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$ 矛盾。所以，任何满足 $p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$ 的文档，必定满足 $\{p_{x1},...,p_{xn} \cup p_{w1},...,p_{wl}\} \rightarrow \{p_{y1},...,p_{ym} \cup p_{z1},...,p_{zk}\}$。

（3）$FD2_{XML}$ 的有效性证明是直接的。

证明：已知 $p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$ 和 $p_{y1},...,p_{ym} \rightarrow p_{z1},...,p_{zk}$ 成立，根据 FD_{XML} 定义，对于任意的树元组 t 和 s，如果 $t[p_{x1},...,p_{xn}]=s[p_{x1},...,p_{xn}]$，必有 $t[p_{y1},...,p_{ym}]=s[p_{y1},...,p_{ym}]$，又因为 $p_{y1},...,p_{ym} \rightarrow p_{z1},...,p_{zk}$，故有 $t[p_{z1},...,p_{zk}]= s[p_{z1},...,p_{zk}]$。这就是说，对于任意的树元组 t 和 s，如果 $t[p_{x1},...,p_{xn}]=s[p_{x1}, ...,p_{xn}]$，必有 $t[p_{z1},...,p_{zk}]=s[p_{z1},...,p_{zk}]$，即 $p_{x1},...,p_{xn} \rightarrow p_{z1},...,p_{zk}$ 成立。

（4）$FD3_{XML}$ 的有效性证明也是直接的。

证明：已知 $p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$ 和 $p_{x1},...,p_{xn} \rightarrow p_{z1},...,p_{zk}$ 成立，根据 FD_{XML} 定义，对于任意的树元组 t 和 s，如果 $t[p_{x1},...,p_{xn}]=s[p_{x1},...,p_{xn}]$，必有 $t[p_{y1},...,p_{ym}]=s[p_{y1},...,p_{ym}]$，又因为 $p_{x1},...,p_{xn} \rightarrow p_{z1},...,p_{zk}$，故有 $t[p_{z1},...,p_{zk}]=s[p_{z1},...,p_{zk}]$，根据并集必有 $t[p_{y1},...,p_{ym} \cup p_{z1},...,p_{zk}]= s[p_{y1},...,p_{ym} \cup p_{z1},...,p_{zk}]$。这就是说，对于任意的树元组 t 和 s，如果 $t[p_{x1},...,p_{xn}]=s[p_{x1},...,p_{xn}]$，必有 $t[p_{y1},...,p_{ym} \cup p_{z1},...,p_{zk}]= s[p_{y1},...,p_{ym} \cup p_{z1},...,p_{zk}]$，即 $p_{x1},...,p_{xn} \rightarrow \{p_{y1}, ...,p_{ym} \cup p_{z1},...,p_{zk}\}$成立。

（5）$FD4_{XML}$ 的有效性证明。

证明：已知 $p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$ 和 $p_{z1},...,p_{zk} \subseteq p_{y1},...,p_{ym}$，根据 FD_{XML} 定义，对于任意的树元组 t 和 s，如果 $t[p_{x1},...,p_{xn}]=s[p_{x1},...,p_{xn}]$，必有 $t[p_{y1},...,p_{ym}]=s[p_{y1},...,p_{ym}]$，又因为 $p_{z1},...,p_{zk} \subseteq p_{y1},...,p_{ym}$，故有 $t[p_{z1},...,p_{zk}]=s[p_{z1},...,p_{zk}]$。这就是说，对于任意的树元组 t 和 s，如果 $t[p_{x1},...,p_{xn}]=s[p_{x1},...,p_{xn}]$，必有 $t[p_{z1},...,p_{zk}]=s[p_{z1},...,p_{zk}]$，即 $p_{x1},...,p_{xn} \rightarrow p_{z1},...,p_{zk}$ 成立。

（6）$FD5_{XML}$ 的证明可以通过前面的规则直接导出。

证明：

[1] $p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$　（已知）

[2] $p_{w1},...,p_{wl} \cup p_{x1},...,p_{xn} \rightarrow p_{w1},...,p_{wl} \cup p_{y1},...,p_{ym}$　（[1], $FD1_{XML}$）

[3] $p_{w1},...,p_{wl} \cup p_{y1},...,p_{ym} \rightarrow p_{z1},...,p_{zk}$　（已知）

[4] $p_{w1},...,p_{wl} \cup p_{x1},...,p_{xn} \rightarrow p_{z1},...,p_{zk}$　（[2], [3], $FD2_{XML}$）

3.3.2　推理规则完备性

为了证明 FD_{XML} 推理规则的完备性，这里首先给出一个概念——路径集的闭包。

定义 3.10（路径集的闭包） 设 Σ 是路径集 $Paths(D)$ 上的一个 FD_{XML} 集，$\{p_{x1},...,p_{xn}\} \subseteq Paths(D)$，则 $\{p_{x1},...,p_{xn}\}$ 关于 FD_{XML} 集 Σ 的闭包 $\{p_{x1},...,p_{xn}\}^+$ 可定义如下。

$$\{p_{x1},...,p_{xn}\}^+=\{p_y \mid p_{x1},...,p_{xn} \rightarrow p_y \text{ 可用 FDXML 推理规则从 } \Sigma \text{ 中导出}\}$$

引理 3.1　$p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$ 可由 FD_{XML} 推理规则导出当且仅当 $p_{y1},...,p_{ym} \subseteq \{p_{x1},...,p_{xn}\}^+$。

证明：

（充分性） $p_{y1},...,p_{ym} \subseteq \{p_{x1},...,p_{xn}\}^+$，根据路径集闭包定义可知：

$p_{x1},...,p_{xn} \to p_{yi}$ 可以从 FD_{XML} 推理规则推出。再根据 $FD3_{XML}$ 合并规则，可得 $p_{x1},...,p_{xn} \to p_{y1},...,p_{ym}$。

（必要性）设 $p_{x1},...,p_{xn} \to p_{y1},...,p_{ym}$ 能用 FD_{XML} 推理规则推出，根据 $FD4_{XML}$ 投影规则，可知 $p_{x1},...,p_{xn} \to p_{yi}$（$1 \leqslant i \leqslant m$），由路径集闭包定义有 $p_{yi} \subseteq \{p_{x1},...,p_{xn}\}^+$，进而 $p_{y1},...,p_{ym} \subseteq \{p_{x1},...,p_{xn}\}^+$。

证明推理规则的完备性，即对任何 $\Sigma \vDash \varphi$（Σ 是 XML 函数依赖集，φ 是某个函数依赖），都有 $\Sigma \vdash \varphi$。根据逆命题的等价性，这也就是要证明，如果存在某个 FD_{XML} φ：$p_{x1},...,p_{xn} \to p_{y1},...,p_{ym}$ 不能由 Σ 根据 FD_{XML} 推理规则导出，则 φ 一定不为 Σ 所逻辑蕴涵，或者说，至少存在某个 XML 文档树实例 T_r，使 $T_r \in SAT(\Sigma)$，但 $T_r \notin SAT(\varphi)$。

定理 3.2　$FD0_{XML}$～$FD5_{XML}$ 对于 FD_{XML} 之间的逻辑蕴涵关系的推理是完备的。

证明：根据上面的分析，假设某个 FD_{XML} φ：$p_{x1},...,p_{xn} \to p_{y1},...,p_{ym}$ 不能由 Σ 根据 FD_{XML} 推理规则导出，可以证明，至少存在一个 XML 文档树实例 T_r，使 $T_r \in SAT(\Sigma)$ 而 $T_r \notin SAT(\varphi)$。

XML 文档树实例 T_r 采用下列方式构造：它仅有两个树元组 t 和 s 组成，其中 t 在 $Path(T_r)$ 中全部路径上的值均为 1，s 在 $\{p_{x1},...,p_{xn}\}^+$ 中的路径上的值为 1，其他路径上的值均为 0，如图 3-1 所示。

	$\{p_{x1},...,p_{xn}\}^+$ 中的路径	其他路径
t	1　1　1　1…1	1　1　1　1…1
s	1　1　1　1…1	0　0　0　0…0

图 3-1　XML 文档树实例 T_r

（1）首先，证明 XML 文档树实例 $T_r \in SAT(\Sigma)$。

假设存在某个 FD_{XML}：$p_{z1},...,p_{zk} \to p_{w1},...,p_{wl} \in \Sigma$ 在 T_r 中不成立，则显然 $p_{z1},...,p_{zk} \subseteq \{p_{x1},...,p_{xn}\}^+$。因为如果 $p_{z1},...,p_{zk} \notin \{p_{x1},...,p_{xn}\}^+$，则 $t[p_{x1},...,p_{xn}] \neq s[p_{x1},...,p_{xn}]$。既然 T_r 中不存在任何在路径集 $p_{z1},...,p_{zk}$ 上的具有相等值的树元组，那么 $p_{z1},...,p_{zk} \to p_{w1},...,p_{wl}$ 是平凡满足的，这与假设 $p_{z1},...,p_{zk} \to p_{w1},...,p_{wl}$ 不成立相矛盾。

既然 $p_{z1},...,p_{zk} \subseteq \{p_{x1},...,p_{xn}\}^+$，根据 $FD0_{XML}$ 自反规则和 FD_{XML} 路径集闭包的定义，有 $\Sigma \vdash p_{x1},...,p_{xn} \to p_{z1},...,p_{zk}$。又因为 $p_{z1},...,p_{zk} \to p_{w1},...,p_{wl} \in \Sigma$，由

$FD2_{XML}$ 传递规则，有 $p_{x1},...,p_{xn} \rightarrow p_{w1},...,p_{wl}$ 成立，即$\{p_{w1},...,p_{wl}\} \subseteq \{p_{x1},...,p_{xn}\}^+$，从而树元组 $t[p_{w1},...,p_{wl}]=s[p_{w1},...,p_{wl}]$成立，也就是说，对于任意的树元组 t 和 s，如果 $t[p_{z1},...,p_{zk}]=s[p_{z1},...,p_{zk}]$，必有 $t[p_{w1},...,p_{wl}]=s[p_{w1},...,p_{wl}]$成立，即 $p_{x1},...,p_{xn} \rightarrow p_{z1},...,p_{zk}$ 成立，从而原假设不成立。

综上所述，$T_r \in SAT(\Sigma)$。

（2）其次，证明 XML 文档树实例 $T_r \notin SAT$（$p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$）。

因为如果 $T_r \in SAT(\varphi)$，必然 $t[p_{x1},...,p_{xn}]=s[p_{x1},...,p_{xn}]$，$t[p_{y1},...,p_{ym}]=s[p_{y1},...,p_{ym}]$成立。根据 T_r 的构造规则，必定有 $p_{y1},...,p_{ym} \subseteq \{p_{x1},...,p_{xn}\}^+$，进而根据 $FD0_{XML}$ 规则可以推导出 $p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$ 成立。但这与初始假设 $p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$ 不能由 Σ 根据 MVD_{XML} 推理规则导出相矛盾。

这样，就可以构造出一个 XML 文档树实例 T_r，使得 $T_r \in SAT(\Sigma)$且 $T_r \notin SAT(p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym})$，从而 $\Sigma \nvdash p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$。

这就是说，只要 $p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$ 不能由 Σ 通过 FD_{XML} 推理规则导出，则 Σ 必定不能逻辑蕴涵 $p_{x1},...,p_{xn} \rightarrow p_{y1},...,p_{ym}$，所以 FD_{XML} 推理规则是完备的。

3.3.3　推理规则的应用

FD_{XML} 推理规则在 XML 数据库理论中具有重要的应用，包括用它进行某些理论推导，计算路径集闭包，判定 FD_{XML} 的蕴涵问题，求解 FD_{XML} 的冗余和最小覆盖集等。这些内容将在后文中陆续介绍。

我们已经知道，判定某个 FD_{XML} 是否被 FD_{XML} 集 Σ 所蕴涵的问题，实际上就是求解该 FD_{XML} 是否属于 Σ 的成员籍问题。但是由于 Σ^+ 比 Σ 大得多，计算 Σ^+ 是一个相当耗时的工作，一般需要指数级的时间。况且，由于 Σ^+ 中包含大量的冗余信息，计算出全部 Σ^+ 也是没有意义的。

避免指数级时间计算的一个有效办法是利用引理 3.1 和路径集闭包概念。一个计算路径集闭包的算法描述如算法 3.1 所示。

算法 3.1　FD_{XML}_PATHSCLOSURE（求相对于 FD_{XML} 集的路径集闭包）。

输入：DTD D 上的 FD_{XML} 集 Σ，$p_{x1},...,p_{xn} \subseteq Paths(D)$。

输出：$p_{x1},...,p_{xn}$ 相对于 FD_{XML} 集 Σ 的闭包$\{p_{x1},...,p_{xn}\}^+$。

FD_{XML}_PATHSCLOSURE(Σ，$p_{x1},...,p_{xn}$)

（1）初始化：$S:=p_{x1},...,p_{xn}$；$S^+:=S \cup \{root\}$;

（2）for　每一个 $p \in EPaths(D)$　do

$S^+ := S^+ \cup Anc(p) \cup \{p.S\}$;

（3）for　每一个 $p \in S^+$　do

$S^+ := S^+ \cup Att(p)$;

（4）repeat　（直到 S^+ 不再变化）

for　每个 $S_1 \rightarrow S_2 \in \Sigma$（其中 S_1 为形如 $p_1 p_2 ... p_k$ 的路径集）　do

{ if　$S_1 \in S^+$　then　$S_{new} := S_2$;

if　$(\exists S' \subseteq S_2, \forall p \in S_1, q \in S', p \cap q = root)$　then　$S_{new} := S'$;

for　每一个 $p \in S_{new}$　do

{$S^+ := S^+ \cup p \cup \{p.S\}$;

if　$p \in EPaths(D)$　then　$S^+ := S^+ \cup Anc(p) \cup Anc(p).S \cup Att(Anc(p))$;

}

for　每一个 $q_i \in S_2$　do

{ if $((\exists\, p'_1 p'_2 ... p'_k \in S^+$满足 $p_i \cap q \leqslant p'_i \leqslant p_i)$　or

$(\exists q'_i \in S^+$满足 $\max(\{p_1 \cap q, p_2 \cap q, \ldots, p_k \cap q\} \leqslant q'_i \leqslant q_i))$

then

{$S^+ := S^+ \cup q_i$;

if　$q_i \in EPaths(D)$　then　$S^+ := S^+ \cup Anc(q_i) \cup Anc(q_i).S \cup Att(Anc(q_i))$;

}

}

}

（5）return(S^+)

算法 3.1 分析如下。

（1）算法 3.1 FD_{XML}_PATHSCLOSURE 的正确性证明。

算法的第（1）步初始化给定路径集的闭包 S^+，第（2）步又将值类型路径和祖先路径加入闭包中，第（3）步将属性路径加入闭包中，第（4）步利用第（2）和第（3）步反复迭代Σ中的每一个 FD_{XML}，如果每个 FD_{XML} 的左部路径属于闭包，根据 FD_{XML} 推理规则，则闭包将函数决定每个 FD_{XML} 的左部路径，进而函数决定其右部路径，所以每个 FD_{XML} 的右部必将属于闭包，所以整个算法结束后，求得的结果即为路径集的闭包。算法的正确性得证。

（2）算法 3.1 FD_{XML}_PATHSCLOSURE 的终止性证明。

算法 3.1 可终止性主要由第（2）、第（3）、第（4）步中的循环体决定。由于 $EPaths(D)$和给定的路径集 S 中的路径个数是有限的，第（2）、第（3）步中的循环是可终止的，对于算法的第（4）步Σ中的 FD_{XML} 个数也是有限的，

所以第（4）步中 for 循环也是可终止的。因此，整个算法最终是可终止的。

（3）算法 3.1 FD_{XML}_PATHSCLOSURE 的时间复杂性分析。

假设Σ中的 FD_{XML} 个数，每个 FD_{XML} 的左、右部路径的个数和给定路径集中的路径个数均为 n，那么，第（2）、第（3）步的时间复杂性分别为 $O(n)$，第（4）步中有四层嵌套循环，所以时间复杂性为 $O(n^4)$，因上，整个算法的时间复杂性为 $O(n+n+n^4)\doteq O(n^4)$。

算法 3.2　FD_{XML}_MEMBERSHIP（FD_{XML} 成员籍判定）。

输入：DTD D 上的 FD_{XML} 集Σ，一个 FD_{XML}：$p_{x1},...,p_{xn}\rightarrow p_{y1},...,p_{ym}$。

输出：如果 FD_{XML} 集$\Sigma\vDash p_{x1},...,p_{xn}$，输出 true，否则输出 false。

FD_{XML}_MEMBERSHIP(Σ，$p_{x1},...,p_{xn}\rightarrow p_{y1},...,p_{ym}$)

（1）S^+:=FD_{XML}_PATHSCLOSURE(Σ，$p_{x1},...,p_{xn}$);

（2）if　$p_{y1},...,p_{ym}\subseteq S^+$　then　return(true)

　　else　return(false)

算法 3.2 FD_{XML}_MEMBERSHIP 的正确性、可终止性和时间复杂性根据算法 3.1 和定义 3.10 是显而易见的，这里省略证明分析过程。

3.4　XML 函数依赖集的覆盖问题

在 3.2 节中，曾经指出，某些数据依赖可能被其他的数据依赖所蕴涵，因此，在讨论一个 XML 数据库中的数据依赖时，就常常涉及如何尽可能简捷地把它们表示出来的问题。也就是说，对于 XML 函数依赖集Σ需要找到Σ中包含路径数目或函数依赖数目较少的表示，即覆盖问题。这样就可以简化给定的函数依赖集，但不改变它的闭包。由于闭包相同，所以满足简化后的函数依赖集的数据库也一定满足原依赖集，反之亦然。

覆盖问题是数据库理论中的重要问题。而目前，有关 XML 数据模式设计的研究刚刚开始，且均没有考虑到函数依赖集的覆盖问题。本节将为 XML 引入函数依赖集覆盖的概念，并将给出求解函数依赖集覆盖的算法。

3.4.1　等价与覆盖

定义 3.11（XML 函数依赖集覆盖）　给定 DTD $D=(E, A, P, R, r)$，M 和 N 是给定路径集 $Paths(D)$上的两个 FD_{XML} 集，如果 $M^+=N^+$，则称 M 和 N 是等价的，记作 M≡N；如果 FD_{XML} 集 M≡N，则称 M 是 N 的一个覆盖，或 N 是 M 的

一个覆盖。

引理 3.2　设 M 和 N 是 DTD 上的两个 FD_{XML} 集，M≡N 当且仅当 M⊨N 且 N⊨M。

证明：根据引理 3.1、定义 3.10 和定义 3.11，引理 3.2 的证明是显然的。

3.4.2　XML 函数依赖集的无冗余覆盖

为了便于今后的讨论，将不失一般性地假设这里研究的 FD_{XML} 集中的每一个 FD_{XML} 都是右部路径单一化的，因为根据 FD_{XML} 推理规则集中的合并规则，右部路径单一化的 FD_{XML} 可以进行合并，同理，合并后的右部路径根据 FD_{XML} 推理规则集中的投影规则，可以进行右部单一化。

在右部路径单一化的假设前提下，一个 FD_{XML} 集中的信息冗余，无非表现在以下两个方面。

（1）该 FD_{XML} 集中的某些 FD_{XML} 可由其他 FD_{XML} 导出（或逻辑蕴涵）。

（2）该 FD_{XML} 集中的某些 FD_{XML} 的左部路径集中，存在可以略去的路径。

为了求解 XML 覆盖，这里首先给出无冗余覆盖的概念。

定义 3.12（FD_{XML} 集的无冗余覆盖）　给定 DTD $D=(E, A, P, R, r)$，对于某个 FD_{XML} 集 Σ，如果它的任何真子集都和它不等价，则称 Σ 是无冗余 FD_{XML} 集；如果存在 Σ 的某个真子集 $\Sigma' \equiv \Sigma$，则称 Σ 是冗余 FD_{XML} 集；如果 FD_{XML} 集 M 是无冗余 FD_{XML} 集，并且 M 是 Σ 的一个覆盖，则称 M 是 Σ 的一个无冗余覆盖。

算法 3.3　FD_{XML}_REDUNDANCY（冗余 FD_{XML} 集的检验）。

输入：一个 FD_{XML} 集 Σ。

输出：如果 Σ 是冗余 FD_{XML} 集，输出 true，否则输出 false。

FD_{XML}_REDUNDANCY(Σ)

（1）v:=false;

（2）for　每一个 FD_{XML}　$p_{x1},...,p_{xn} \rightarrow p_y \in \Sigma$　do

　　if　FD_{XML}_MEMBERSHIP(Σ-{ $p_{x1},...,p_{xn} \rightarrow p_y$ }，　$p_{x1},...,p_{xn} \rightarrow p_y$)

　　then　{v:=true; return(v); }

（3）return(v)

算法 3.3 分析如下。

（1）算法 3.3 FD_{XML}_REDUNDANCY 的正确性证明。

如果对于 FD_{XML}：$p_{x1},...,p_{xn} \rightarrow p_y$ 能够利用本身之外其他的 FD_{XML} 推导出

来，也就是说，如果 $\Sigma-\{p_{x1},...,p_{xn}\rightarrow p_y\}\vDash p_{x1},...,p_{xn}\rightarrow p_y$，那么 $p_{x1},...,p_{xn}\rightarrow p_y$ 本身在 Σ 中是多余的。算法中调用了 FD_{XML}_MEMBERSHIP($\Sigma-\{p_{x1},...,p_{xn}\rightarrow p_y\}$，$p_{x1},...,p_{xn}\rightarrow p_y$)算法，即判定任一 $p_{x1},...,p_{xn}\rightarrow p_y$ 是否被本身之外的 FD_{XML} 所蕴涵。如果 $p_{x1},...,p_{xn}\rightarrow p_y$ 被其他的 FD_{XML} 所蕴涵，v=true，说明 Σ 是冗余的。算法中 for 循环体对每一个 MVD_{XML} 都进行判定，如不被其他的 FD_{XML} 所蕴涵，算法返回 v 的初始值 false，即说明 Σ 不是冗余的，算法的正确性得证。

（2）算法 3.3 FD_{XML}_REDUNDANCY 的终止性证明。

算法 3.3 中调用了 FD_{XML} 成员籍判定算法 FD_{XML}_MEMBERSHIP，在前面的章节中已经证明是可终止的，又因为Σ中的 FD_{XML} 的路径个数是有限的，所以 for 循环体是可终止的。因此，整个算法最终是可终止的。

（3）算法 3.3 FD_{XML}_REDUNDANCY 的时间复杂性分析。

算法 3.3 的时间复杂性主要由第（2）步中的 for 循环体决定。算法第（2）步调用了时间复杂性为 $O(n^4)$的 FD_{XML} 成员籍判定算法 FD_{XML}_MEMBERSHIP，Σ 中 FD_{XML} 的个数最多为 n 个，所以 for 循环体的时间复杂性为 $O(n\times n^4)=O(n^5)$，所以整个算法的时间复杂性为 $O(n^5)$。

算法 3.4　FD_{XML}_NONREDUNDANCY（求 FD_{XML} 集的无冗余覆盖）。

输入：一个 FD_{XML} 集Σ。

输出：Σ的一个无冗余覆盖Σ'。

FD_{XML}_NONREDUNDANCY(Σ)

（1）初始化：Σ' := Σ;

（2）for　每一个 FD_{XML}：$p_{x1},...,p_{xn}\rightarrow p_y\in\Sigma'$　do

　　if　FD_{XML}_MEMBERSHIP($\Sigma'-\{p_{x1},...,p_{xn}\rightarrow p_y\}$, $\{p_{x1},...,p_{xn}\rightarrow p_y\}$)

　　then　$\Sigma' := \Sigma'-\{p_{x1},...,p_{xn}\rightarrow p_y\}$;

（3）return(Σ')

算法 3.4 分析如下。

（1）算法 3.4 FD_{XML}_NONREDUNDANCY 的正确性证明。

对于Σ中冗余的 FD_{XML}：$p_{x1},...,p_{xn}\rightarrow p_y$，由于它可以利用其他 FD_{XML} 推导出来，所以在Σ中可将其删除，即算法中的$\Sigma' := \Sigma'-\{p_{x1},...,p_{xn}\rightarrow p_y\}$。因为算法中 for 循环体对每个 FD_{XML} 都进行判定，凡是冗余的 FD_{XML} 皆可由$\Sigma' := \Sigma'-\{p_{x1},...,p_{xn}\rightarrow p_y\}$删除。所以算法最终返回的Σ'是无冗余的覆盖。

算法的正确性得证。

（2）算法 3.4 FD_{XML}_NONREDUNDANCY 的终止性证明。

算法 3.4 中第(2)步调用了 FD_{XML} 成员籍判定算法 FD_{XML}_MEMBERSHIP，在前面的章节中已经证明是可终止的，又因为Σ中的 FD_{XML} 的路径个数是有限的，所以 for 循环体是可终止的。因此，整个算法最终是可终止的。

（3）算法 3.4 FD_{XML}_NONREDUNDANCY 的时间复杂性分析。

算法 3.4 的时间复杂性主要由第（2）步中的 for 循环体决定。第（2）步调用了时间复杂性为 $O(n^4)$的 FD_{XML} 成员籍判定算法 FD_{XML}_MEMBERSHIP，Σ中 FD_{XML} 的个数最多为 n 个，所以 for 循环体的时间复杂性为 $O(n\times n^4)=O(n^5)$，因此，整个算法的时间复杂性为 $O(n^5)$。

3.4.3　左部路径冗余与规范覆盖集

对于一个无冗余 FD_{XML} 集，已经不能再通过删除其中的 FD_{XML} 的办法来缩小其表示的规模。但是，我们仍然可以从某些 FD_{XML} 中删去某些路径而不影响其表示能力。

定义 3.13（冗余路径） 设Σ是 *Paths*(*D*)上的一个 FD_{XML} 集，$P_1\to P_2\in\Sigma$，$P_1\to P_2$ 中相对于Σ的冗余路径 p 定义如下。

（1）如果 $p\in P_1$，且$(D,\ \Sigma)$逻辑蕴涵$(\Sigma-\{P_1\to P_2\})\cup(\{P_1-p\}\to P_2)$，则称路径 p 是 $P_1\to P_2$ 左部的冗余路径。

（2）如果 $p\in P_2$，且$(D,\ \Sigma)$逻辑蕴涵$(\Sigma-\{P_1\to P_2\})\cup(P_1\to\{P_2-p\})$，则称路径 p 是 $P_1\to P_2$ 右部的冗余路径。

定义 3.14（简化的 FD_{XML}） 设Σ是 FD_{XML} 集，$P_1\to P_2\in\Sigma$。如果 $P_1\to P_2$ 不包含左部冗余路径，称 $P_1\to P_2$ 是左部简化的。如果 $P_1\to P_2$ 不包含右部冗余路径，称 $P_1\to P_2$ 是右部简化的。如果 $P_1\to P_2$ 同时是左部简化和右部简化的，称 $P_1\to P_2$ 是简化的 FD_{XML}。如果Σ中的每一个 FD_{XML} 都是左部简化的（右部简化的，简化的），称Σ是左部简化的（右部简化的，简化的）FD_{XML} 集。

由于假定被考虑的 FD_{XML} 都是右部路径单一化的，因此，任意一个无冗余 FD_{XML} 集也一定是右部简化的。

定义 3.15（规范 FD_{XML} 集） 一个 FD_{XML} 集Σ如果满足以下条件：

（1）Σ中每一个 FD_{XML} 右部只有单一路径。

（2）Σ是左部简化的并且是无冗余的。

则称Σ是规范化的。

显然，如果Σ是规范 FD_{XML} 集，它也一定是简化的；反之，如果Σ是一个简化的 FD_{XML} 集，则只要将它的每一个 FD_{XML} 右部路径单一化，结果将得到一个规范 FD_{XML} 集。

对于一个任意的 FD_{XML} 集，求它的规范覆盖集的算法描述如算法 3.5 所示。

算法 3.5　FD_{XML}_CANONICAL（求 FD_{XML} 集的规范覆盖）。

输入：一个 FD_{XML} 集Σ。

输出：Σ的一个规范覆盖 C。

FD_{XML}_CANONICAL(Σ)

（1）使Σ中每一个 FD_{XML} 的右部路径单一化，设结果 FD_{XML} 集为Σ'；

（2）for　每一个 FD_{XML}　$P_1 \rightarrow P_2 \in \Sigma'$　do
　　for　每一条路径 $p \in P_1$　do
　　if　FD_{XML}_MEMBERSHIP(Σ', $\{P_1\text{-}p\} \rightarrow P_2$)
　　then　$\Sigma' := (\Sigma' - \{P_1 \rightarrow P_2\}) \cup (\{P_1\text{-}p\} \rightarrow P_2)$

（3）C:=FD_{XML}_NONREDUNDANCY(Σ');

（4）return(C)

算法 3.5 分析如下。

（1）算法 3.5 FD_{XML}_CANONICAL 的正确性证明。

算法 3.5 的第（1）步是右部路径单一化，符合规范 FD_{XML} 集定义中第一个条件；第（2）步调用了 FD_{XML}_MEMBERSHIP 成员籍判定算法，判定左部冗余路径并消除冗余路径；第（3）步调用了 FD_{XML}_NONREDUNDANCY 求解无冗余覆盖算法，后两步符合规范 FD_{XML} 集定义中第二个条件。所以算法求得的 FD_{XML} 集是规范化的。算法的正确性得证。

（2）算法 3.5 FD_{XML}_CANONICAL 的终止性证明。

算法 3.5 中主要是调用了 FD_{XML} 成员籍判定算法 FD_{XML}_MEMBERSHIP，在前面的章节中已经证明是可终止的，又因为Σ和每个 FD_{XML} 中的左部路径个数是有限的，所以 for 循环体是可终止的。因此，整个算法最终是可终止的。

（3）算法 3.5 FD_{XML}_CANONICAL 的时间复杂性分析。

算法 3.5 的时间复杂性主要由第（2）步中的 for 循环体决定。算法第（2）步调用了时间复杂性为 $O(n^4)$的 FD_{XML} 成员籍判定算法 FD_{XML}_MEMBERSHIP，Σ中 FD_{XML} 的个数最多为 n 个，每个 FD_{XML} 中的左部路径个数最多为 n 个，所以 for 循环体的时间复杂性为 $O(n \times n \times n^4)=O(n^6)$，因此，整个算法的时间复

杂性为 $O(n^6)$。

3.4.4　XML 函数依赖集的最小覆盖

定义 3.16（等价路径集）　设Σ是 DTD D 上的 FD_{XML} 集，路径集 P_1, $P_2 \subseteq Paths(D)$，如果 $\Sigma \models P_1 \rightarrow P_2$ 并且 $\Sigma \models P_2 \rightarrow P_1$，则称 P_1 与 P_2 在 FD_{XML} 集Σ下是等价的，记作 $P_1 \leftrightarrow P_2$。

设对于某个 DTD 上的 FD_{XML} 集Σ和路径集 $P \subseteq Paths(D)$，用 $E_\Sigma(P)$ 表示Σ中左部与 P 等价的 FD_{XML} 的集合，用 $\overline{E_\Sigma}$ 表示集合{ $E_\Sigma(P) \mid P \subseteq Paths(D)$且 $E_\Sigma(P) \neq \Phi$}，显然，$\overline{E_\Sigma}$ 形成Σ的一个分割。

定义 3.17（FD_{XML} 集的最小覆盖）　一个简化 FD_{XML} 集 M 如果与某个 FD_{XML} 集Σ等价，并且在与Σ等价的所有 FD_{XML} 集中是 FD_{XML} 个数最少的，称 M 是Σ的一个最小覆盖。

显然，一个 FD_{XML} 集的最小覆盖一定也是一个无冗余 FD_{XML} 集。

算法 3.6　FD_{XML}_MINIMIZECOVER（求 FD_{XML} 集的规范最小覆盖）。

输入：一个 FD_{XML} 集Σ。

输出：Σ的一个最小覆盖 M。

FD_{XML}_MINIMIZECOVER(Σ)

（1）M:=FD_{XML}_CANONICAL(Σ);

（2）计算 M 的分割 $\overline{E_M}$；

（3）for　每一个 $E_M(P) \in \overline{E_M}$　do

　　for　每一对 $P_1 \rightarrow P_2 \neq P_3 \rightarrow P_4 \in E_M(P)$　do

　　if　FD_{XML}_MEMBERSHIP(M- $E_M(P)$, $P_3 \rightarrow P_1$)

　　then　M:=(M-{$P_1 \rightarrow P_2$, $P_3 \rightarrow P_4$}) ∪ ($P_1 \rightarrow P_2P_4$)

（4）return(M)

算法 3.6 分析如下。

（1）算法 3.6 FD_{XML}_MINIMIZECOVER 的正确性证明。

算法 3.6 的第（1）步调用了 FD_{XML}_CANONICAL 算法求解规范 FD_{XML} 集；第（2）步计算 FD_{XML} 集的等价类，即分割；第（3）步对等价类中的 FD_{XML} 利用成员籍判定算法进行合并以减少 FD_{XML} 的数目。合并的思想符合 FD_{XML} 的推理规则，所以算法求得的 FD_{XML} 集是最小的。算法的正确性得证。

（2）算法 3.6 FD_{XML}_MINIMIZECOVER 的终止性证明。

算法 3.6 中主要是调用了 FD_{XML} 成员籍判定算法 FD_{XML}_MEMBERSHIP，

在前面的章节中已经证明是可终止的，又因为Σ和每个 FD_{XML} 中的左部路径个数是有限的，所以 for 循环体是可终止的。因此，整个算法最终是可终止的。

（3）算法 3.6 FD_{XML}_MINIMIZECOVER 的时间复杂性分析。

算法 3.6 的时间复杂性主要由第（3）步中的 for 循环体决定。算法第（3）步调用了时间复杂性为 $O(n^4)$的 FD_{XML} 成员籍判定算法 FD_{XML}_MEMBERSHIP，Σ中 FD_{XML} 的个数最多为 n 个，每个 FD_{XML} 中的左部路径个数最多为 n 个，所以 for 循环体的时间复杂性为 $O(n\times n\times n^4)=O(n^6)$。因此，整个算法的时间复杂性为 $O(n^6)$。

定义 3.18（FD_{XML} 集的最优覆盖） 一个 FD_{XML} 集Σ是最优集，如果在与它等价的 FD_{XML} 集中，它所包含的路径条数是最少的。

3.5　小　　结

本章基于路径表达式和树元组给出了 XML 函数依赖的形式化定义，并在此基础上给出了 XML 函数依赖的推理规则和 XML 路径集闭包的概念，并对 XML 函数依赖推理规则的有效性和完备性进行了证明。接着，在 XML 函数依赖和推理规则的基础上引入了 XML 函数依赖的逻辑蕴涵与覆盖的概念，包括无冗余覆盖，规范覆盖以及最小覆盖等，给出了相关的覆盖的求解算法，并对算法的正确性、可终止性和时间复杂性进行了分析和证明。最后，给出 FD_{XML} 集的最优覆盖概念。这些问题的研究是进行 XML 文档模式设计和规范化的基础。下一章将针对 XML 规范化的问题进行描述。

第 4 章　XML 范式及文档规范化

随着 XML 应用范围的拓广，为了保证数据的一致性，作为 XML 文档的主要模式定义方法，一个设计良好的 DTD 是必须的。同关系数据库类似，设计不好的 XML 数据模式会引起操作异常。判断 DTD 是否设计良好，可以从不同的角度来考虑。一个很重要的因素是，在一个良好设计 DTD 的 XML 文档中，应该尽可能地避免数据冗余的出现。这是因为数据冗余是导致操作异常产生的根源。由于数据依赖的存在，数据被重复地多次存储，必然使相应的插入、删除和更新操作变得困难。随着文档数目的增加，类似问题会显得越来越突出。前面的章节中已经提到，由于 Internet 的开放性，XML 数据操作异常的危害性要远远大于关系数据操作异常的危害性。因此，对引起 XML 数据操作异常的原因及消除异常的方法进行规范化研究具有重要的意义。

在关系数据库领域中，为了区分数据库模式的优劣，常常把数据库模式分为各种不同的等级，称为范式（Normal Form，NF）。在为 XML 引入函数依赖的概念后，就可以基于它进行判定一个 DTD 是否为设计良好的讨论，并对设计不好的 DTD 进行规范化处理。同样地，也可以为 XML 数据库理论引入范式的概念，并在范式的基础上进行 XML 数据库模式的规范化研究。

4.1　XML 范式

在第 3 章中已经给出了 XML 平凡函数依赖、部分函数依赖和传递函数依赖等概念，也就是说，XML 数据之间不是相互独立的，正是由于数据之间的相互约束关系使得数据冗余和操作异常的存在。究其原因是 XML 文档不具有良好的 DTD，那么如何判定一个 DTD 是否是设计良好的呢？同关系数据库一样，为了区分 XML 数据库模式的优劣，本节也给出 XML 数据库模式的不同级别的范式。

4.1.1　XML 范式定义

定义 4.1（XML 范式，XNF）　给定一个 DTD D 和其上的函数依赖集Σ，(D, Σ)是 XNF 当且仅当每个形如 $S \rightarrow p.@l$ 或 $S \rightarrow p.\mathrm{S}$ 的非平凡 $\mathrm{FD}_{\mathrm{XML}}$：$\varphi \in (D, \Sigma)^{+}$，都满足$(D, \Sigma)^{+} \vDash S \rightarrow p$。

直观地，假设 $S \rightarrow p.@l$ 属于$(D, \Sigma)^{+}$。如果 T 是符合 D 且满足Σ的文档树，那么在 T 中对于 S 中元素任意一个值集，仅能找到一个 $p.@l$ 值。因此，S 中每个值集，需要存储 $p.@l$ 值一次。换句话说，(D, Σ)一定蕴涵 $S \rightarrow p$。在这个定义里，强调条件φ是非平凡的 $\mathrm{FD}_{\mathrm{XML}}$，因为平凡的 $\mathrm{FD}_{\mathrm{XML}}$　$p.@l \rightarrow p.@l$ 总是在$(D, \Sigma)^{+}$中，但是通常 $p.@l \rightarrow p \notin (D, \Sigma)^{+}$。不失一般性，本书给出基于 XML 函数依赖的范式级别形式。

4.1.2　XML 范式级别

定义 4.2（XML 第一范式，X1NF）　给定 DTD D 和 D 上的函数依赖集Σ，如果元素结点、属性结点和文本结点的值是不可分解的原子值，那么称 D 是属于 XML 第一范式的，记作 $D \in \mathrm{X1NF}$。如果某个 XML 数据库模式中每个 DTD 模式都是第一范式的，则称该数据库模式是属于第一范式的。

定义 4.3（XML 第二范式，X2NF）　给定 DTD D 和 D 上的函数依赖集Σ，如果 $D \in \mathrm{X1NF}$，并且在$(D, \Sigma)^{+}$中不存在部分函数依赖，则称 D 是属于 XML 第二范式的，如果某个 XML 数据库模式中每个 DTD 模式都是第二范式的，则称该数据库模式是属于第二范式的。

定义 4.4（XML 第三范式，X3NF）　给定 DTD D 和 D 上的函数依赖集Σ，如果 $D \in \mathrm{X1NF}$，并且在$(D, \Sigma)^{+}$中不存在部分函数依赖和传递函数依赖，则称 D 是属于 XML 第二范式的，如果某个 XML 数据库模式中每个 DTD 模式都是第三范式的，则称该数据库模式是属于第三范式的。

4.1.3　模式分解

给定 DTD D，可以用一个关系模式来表示它。其基本思想：根据根结点 r 下的不同元素类型把 D 分解成几个模式，每一个模式的属性包含 r 的属性和叶子结点（元素类型和属性）。具体方法如下。

- 每个元素变成一个表（关系模式）。
- 每个 XML 属性变成列（关系型属性）。

- 一个元素的每个子元素变成表中的列，这个表与子元素指定表之间有一个外键约束。
- 所有同名元素引用一个关系（并且任何属性或子元素关系合并）。

例如，上述方法可得到例 2.2 中 D_1 相应的关系模式为

R(*@Tno, title, @Ano, Aname, @Mno, Mname*)

定义 4.5（DTD 的无损连接分解） DTD 的无损连接分解可以转化为相应的关系模式的无损连接分解来加以简化。也就是说，给定 DTD $D(E, A, P, R, r)$ 及其相应的关系模式是 R_D，另一个 DTD $D'(E', A', P', R', r)$是从 D 经过某种规则转换得到的，其相应的关系模式是$(R_1, ..., R_n)$，如果关系模式$(R_1, ..., R_n)$是对 R_D 的无损连接分解，那么称 D' 是对 D 的无损连接分解。

4.2　规范化规则

4.2.1　元素提升规则

给定 DTD $D(E, A, P, R, r)$，D 上的部分函数依赖φ：$(H,[p_{x1}.\sigma_{x1},...,p_{xn}.\sigma_{xn} \rightarrow p_y.\sigma_y])$，在$(D,\Sigma)^+$中一定存在另一个 $\mathrm{FD}_{\mathrm{XML}}$ φ'：$(H',[H.p_{z1}.\sigma_{z1},...,H.p_{zm}.\sigma_{z1} \rightarrow p_y.\sigma_y])$，其中 $H' \subseteq_{Paths} H \subset_{Paths} P_{xi} \subseteq_{Paths} P_{yl}$，$1 \leqslant i \leqslant n$，$\sigma$ 是属性或者具有形式 $\tau.\mathrm{S}$（τ 是元素类型）。提升 $last(p_y)$结点，使其成为 $last(H)$的子结点，构造一个新的如图 4-1 所示的 DTD $D'(E', A', P', R', r)$，消除部分函数依赖，以减少由部分函数依赖引起的数据冗余现象。

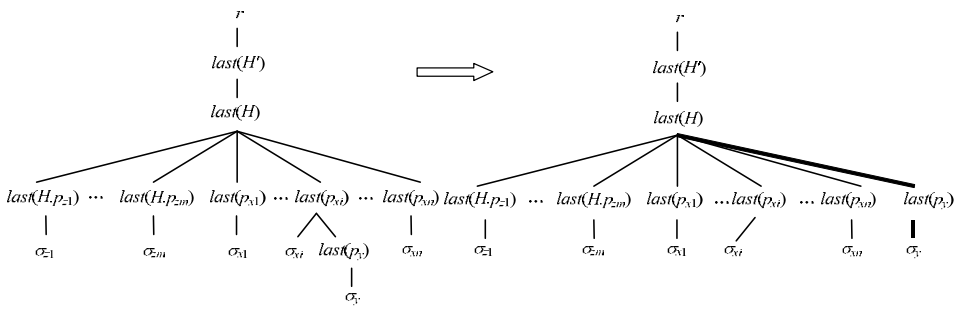

图 4-1　元素提升规则示意图

形式上，DTD $D'(E', A', P', R', r)$，其中，$E'=E$；$A'=A$；$P'(last(H))=\{P(last(H)), last(p_y)\}$；$R'=R$；$P'$的其他定义同 D 中 P 的定义。在Σ中形如 $p_y.\sigma_y$ 形式的函数依

赖在Σ' 中被转换成$H.p_y.\sigma_y$形式。例如，$(H',[H.p_{z1}.\sigma_{z1},...,H.p_{zm}.\sigma_{zm} \to p_y.\sigma_y])$转换成$(H',[H.p_{z1}.\sigma_{z1},...,H.p_{zm}.\sigma_{z1} \to H.p_y.\sigma_y])$。

例 4.1　例 2.2 中 D_1 经元素提升规则可以转化为 DTD D_2，对应的图 2-2 可转换为图 4-2 所示。

<!ELEMENT *Conference* (*Topic**)>
<!ELEMENT *Topic* (*title, discussby,question-Master*)>
<!ATTLIST *Topic*
Tno CDATA #REQUIRED>
<!ELEMENT *title* (#PCDATA)>
<!ELEMENT *discussby* (*Author**)>
<!ELEMENT *Author* (*Aname*)>
<!ATTLIST *Author*
Ano CDATA #REQUIRED>
<!ELEMENT *Aname* (#PCDATA)>
<!ELEMENT *question-Master* (*Mname*)>
<!ATTLIST *question-Master*
Mno CDATA #REQUIRED>
<!ELEMENT *Mname* (#PCDATA)>

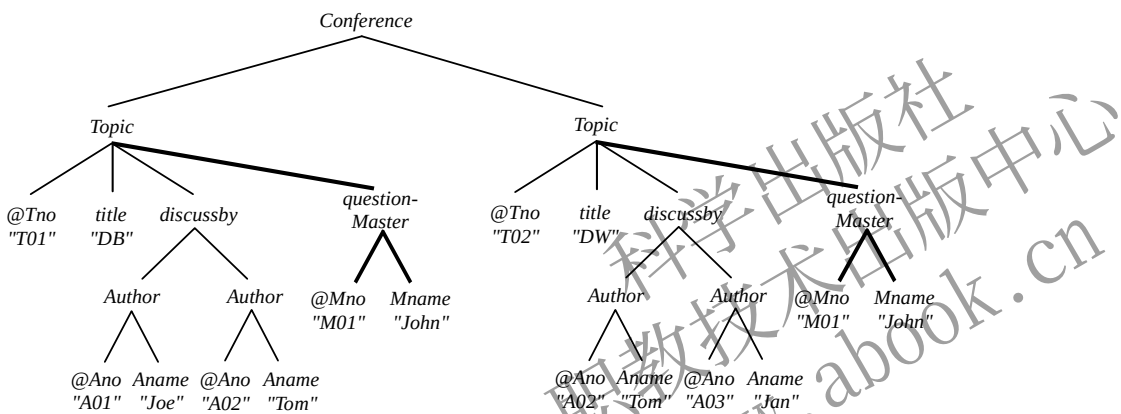

图 4-2　符合 D_2 的一个 XML 文档实例

4.2.2　元素创建规则

给定 DTD $D(E, A, P, R, r)$，D 上的传递函数依赖 φ：$(H,[p_{x1}.\sigma_{x1},..., p_{xn}.\sigma_{xn} \to p_y.\sigma_y])$，在$(D, \Sigma)^+$中一定存在两个 $\mathrm{FD_{XML}}$，φ'：$(H,[p_{x1}.\sigma_{x1},..., p_{xn}.\sigma_{xn} \to$

$p_{z1}.\sigma_{z1}, ..., p_{zm}.\sigma_{zm}]$) 和 φ''：$(H',[p_{z1}.\sigma_{z1},..., p_{zm}.\sigma_{zm} \rightarrow p_y.\sigma_y])$，$H' \subseteq H$，$\sigma$ 是属性或者具有形式 τ.S（τ 是元素类型）。通过创建元素类型 V_{new}，并移动具有传递依赖的结点来构造一个新的如图 4-3 所示的 DTD $D'(E', A', P', R', r)$，消除传递函数依赖，以减少由传递函数依赖引起的数据冗余现象。

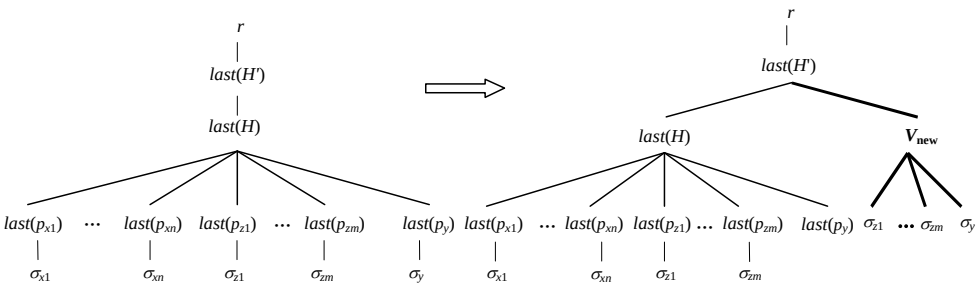

图 4-3　元素创建规则示意图

形式上，DTD $D'(E', A', P', R', r)$，其中：$E'=E \cup \{lab(V_{\text{new}})\}$；$A'=A$；$P'(last(H'))= \{P(last(H)), lab(V_{\text{new}})^*\}$；$P'(lab(V_{\text{new}}))= [\sigma_{z1},...,\sigma_{zm},\sigma_y]$；$P'(last(p_y))=P(last(p_y))\backslash [\sigma_y]$；$R'(lab(V_{\text{new}}))= \{@\sigma_{z1},...,@\sigma_{zm},@\sigma_y\}$；$R'(last(p_y))=R(last(p_y))- \{@\sigma_y\}$；$P'$和 R'的其他定义同 D 中 P 和 R 的定义。在 Σ 中形如 $p_y.\sigma_y$ 形式的函数依赖在 Σ′ 中不再成立，但增加新的函数依赖 φ_{new}：$(H',[H'.V_{\text{new}}.\sigma_{y1},...,H'.V_{\text{new}}.\sigma_{ym}] \rightarrow H'.V_{\text{new}}.\sigma_y)$。

例 4.2　图 4-2 经元素创建规则可转换为图 4-4 所示，其 DTD 为 D_3。

```
<!ELEMENT Conference (Topic*, Masterinfo)>
<!ELEMENT Topic (title, discussby, question-Master)>
    <!ATTLIST Topic
        Tno CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT discussby (Author*)>
<!ELEMENT Author (Aname)>
    <!ATTLIST Author
        Ano CDATA #REQUIRED>
<!ELEMENT Aname (#PCDATA)>
<!ELEMENT question-Master EMPTY>
    <!ATTLIST question-Master
        Mno CDATA #REQUIRED>
```

<!ELEMENT *Masterinfo* (*Mname*)>

<!ATTLIST *Masterinfo*

Mno CDATA #REQUIRED>

<!ELEMENT *Mname* (#PCDATA)>

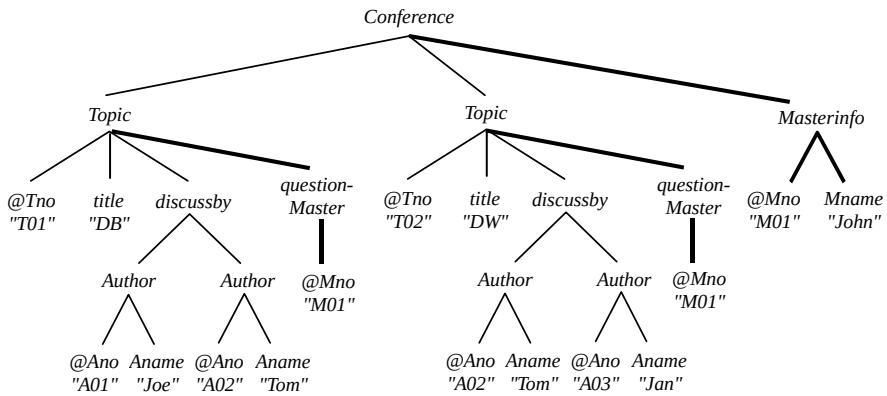

图 4-4　符合 D_3 的一个 XML 文档实例

4.3　规范化算法

根据 4.2 节中的规范化转换规则，本节给出两个 DTD 无损连接分解算法，使其模式分解符合 X2NF 和 X3NF。

4.3.1　无损连接算法

算法 4.1　X2NF-LOSSLESS DECOMPOSITION（DTD 无损连接分解成 X2NF）。

输入：一个规范化的 DTD $D(E, A, P, R, r)$和函数依赖集 Σ。

输出：满足 X2NF 的 D 的无损连接分解 DTD D'。

X2NF-LOSSLESS DECOMPOSITION(D, Σ)

（1）初始化：D':=D，Σ' := Σ；

（2）for　每一个部分函数依赖　φ：$(H,[p_{x1}.\sigma_{x1},...,p_{xn}.\sigma_{xn} \rightarrow p_y.\sigma_y]) \in \Sigma$　do

根据元素提升规则构建新的函数依赖

$\varphi_{\text{new}}:(H,[p_{x1}.\sigma_{x1},...,p_{xn}.\sigma_{xn} \rightarrow H.p_y \quad .\sigma_y])$；　Σ' :=(Σ' -{φ}) ∪ { φ_{new} }；

根据 Σ' 构建新的 DTD D_{new}，D':=D_{new}；

（3）合并 D' 中相同元素类型，消除冗余元素类型；

（4）return(D')

算法 4.2　X3NF-LOSSLESS DECOMPOSITION（DTD 无损连接分解成 X3NF）。

输入：一个规范化的 DTD $D(E, A, P, R, r)$和函数依赖集 Σ。

输出：满足 X3NF 的 D 的无损连接分解 DTD D'。

X3NF-LOSSLESS DECOMPOSITION(D, Σ)

（1）初始化：$D':=D$，$\Sigma':=\Sigma$；

（2）for　每一个传递函数依赖 $\varphi: (H,[p_{x1}.\sigma_{x1},...,p_{xn}.\sigma_{xn} \rightarrow p_y.\sigma_y]) \in \Sigma$　do

根据元素创建规则创建元素类型 V_{new} 构建新的函数依赖

$\varphi_{\text{new}}: (H',[H'.V_{\text{new}}.\sigma_{y1},...,H'.V_{\text{new}}.\sigma_{ym}] \rightarrow H'.V_{\text{new}}.\sigma_y)$；　$\Sigma':=(\Sigma'-\{\varphi\})\cup\{\varphi_{\text{new}}\}$；

根据 Σ' 构建新的 DTD Dnew，D':=Dnew；

（3）合并 D'中相同元素类型，消除冗余元素类型；

（4）return(D')

4.3.2　算法和实验分析

（1）算法 4.1 和算法 4.2 的正确性。

算法 4.1 运用的是元素提升规则，形成所谓的“新元素类型”，并没有改变原文档的元素类型和属性，所得的 DTD D'对应的关系模式同原来的 DTD D 对应的关系模式相等。根据 4.1.3 节的介绍，DTD D_1 对应的关系模式为 R_1(*@Tno, title, @Ano, Aname, @Mno, Mname*)，DTD D_2 对应的关系模式为 R_2(*@Tno, title, @Ano, Aname, @Mno, Mname*)，即 R_1=R_2，且 R_2 属于 X2NF，所以说 R_2 是 R_1 的无损连接分解。算法 4.2 运用的是元素创建规则，形成新元素类型。将形如 $R_1(\sigma_{x1},...,\sigma_{xn},\sigma_{y1},...,\sigma_{ym},\sigma_y,X)$ 形式（其中 X 表示其他的属性集合），转换为包括 $R_{11}(\sigma_{x1},...,\sigma_{xn},\sigma_{y1},...,\sigma_{ym},X)$ 和 $R_{12}(\sigma_{y1},...,\sigma_{ym},\sigma_y)$ 两种形式。由关系模式的无损连接分解易知，R_{11} 和 R_{12} 是对 R_1 的无损连接分解。根据 4.1.3 节的介绍，DTD D_3 相应的关系模式是 R_3(R_{31}, R_{32})，其中 R_{31}(*@Tno, title, @Ano, Aname, @Mno*)和 R_{32}(*@Tno, Mname*)。由定义 4.5 易知，D_3 是 D_1, D_2 的无损连接分解。综上所述，算法 4.1 和算法 4.2 输出的 D'是 D 的无损连接分解。

（2）算法 4.1 和算法 4.2 可终止性。

算法 4.1 重复运用元素提升规则而算法 4.2 重复运用元素创建规则对给定的 DTD D 进行转换，每转换一次，Σ 中部分函数依赖和传递函数依赖的个数就减少一个，由于 Σ 中函数依赖的数目是有限的，所以算法 4.1 和算法 4.2 可

以在有限的时间内终止。

（3）算法 4.1 和算法 4.2 的时间复杂性。

算法 4.1 和算法 4.2 的时间复杂性主要是由 for 循环体和合并消除冗余两步骤决定。for 循环体每执行一步，取 Σ 中一个函数依赖，设最坏情况下 Σ 中所有 FD_{XML} 都是部分函数依赖或传递函数依赖，令 Σ 中函数依赖数目 $n=|\Sigma|$，所以整个 for 循环体要执行 n 步。算法中的第（3）步合并和消除冗余元素类型是检查结点的冗余情况，令 D'中结点数目 $m=|D'|$，检查结点的冗余每次取两个元素，要执行 C_m^2 步（$C_m^2=m(m-1)/2=m^2/2-m/2$），所以合并和消除冗余元素类型的时间复杂性最坏情形下为 $O(m^2)$，所以整个算法的时间复杂性为 $O(m^2+n)$。

（4）实验性能分析。

本节将从查询和存储两个方面进行实验检验规范化算法的有效性。

实验环境：P4 2.4GHz，512MB RAM，Windows 2000 操作系统；

XML 数据库：IPEDO XML Information Hub[①]；

编程语言：Java。

根据本书给出的三种不同的 DTD，我们设计了三种不同的 XML 查询文档 Q_1、Q_2、Q_3，分别对应本书中 D_1、D_2、D_3。

针对每个 XML 文档，在文档中的结点逐渐增加的情况下，不同文档模式设计对于查询 *question-Master* 结点值所需响应时间。在所做的实验中三个文档中的结点数都是从 100 增加到 1000 而得到的结果，实验结果显示如图 4-5 所示。实验表明，规范化的高级别范式的 XML 文档随着结点个数的增加，查询响应时间明显优于低级别范式的 XML 文档的查询响应时间。

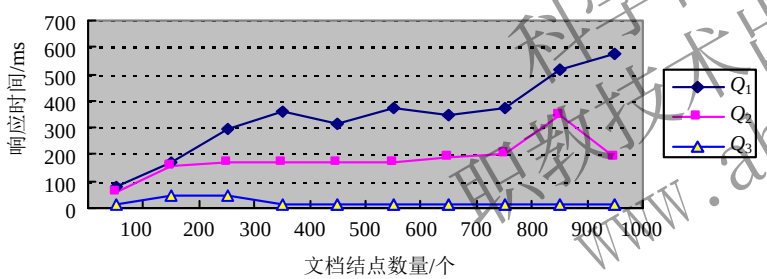

图 4-5　响应时间随文档中结点数量的变化

① IPEDO XML Information Hub 是全球领先的 XML 企业级信息平台，多次获得国际大奖。

在 IPEDO 数据库中，有一个 collection 的概念，就是说一个集合的概念，用户可以将同类的或者相似的或者想放在一起的 XML 文档集中到一个集合中。在实验中，我们应用了这个概念，也就是检验了当一个 collection 中的 XML 文档数量逐渐增加时，在不同设计的 XML 文档中对于查询 *question-Master* 结点值所带来的影响。实验的做法是分别为三种设计创建了三个不同的 collection，然后每个 collection 中的文档数量从 10 增加到 100 而得到的查询响应时间，实验结果如图 4-6 所示。实验表明，规范化的高级别范式的 XML 文档随着文档数量的增加，查询响应时间明显优于低级别范式的 XML 文档的查询响应时间。

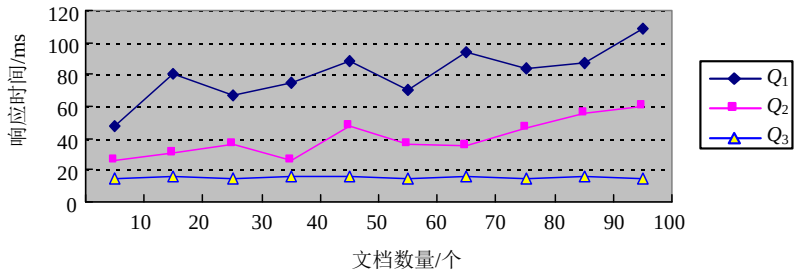

图 4-6　响应时间随文档数量的变化

当文档结点数量增加时，文档的存储容量也随之增加，但是规范化的高级别范式的 XML 文档随着文档结点数量的增加，存储容量明显优于低级别范式的 XML 文档的存储容量，如图 4-7 所示。

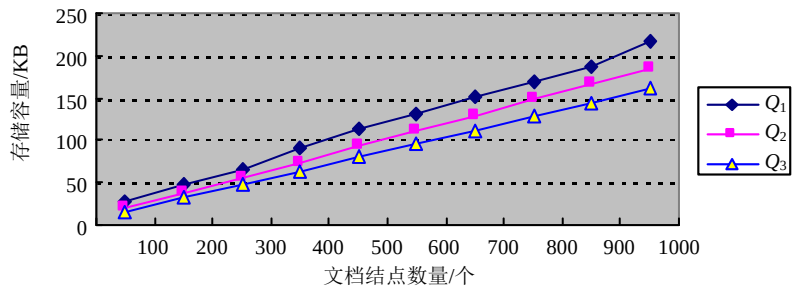

图 4-7　存储容量随文档结点数量的变化

当文档数量增加时，文档的存储容量也随之增加，但是规范化的高级别范式的 XML 文档随着文档数量的增加，存储容量明显优于低级别范式的 XML 文档的存储容量，如图 4-8 所示。

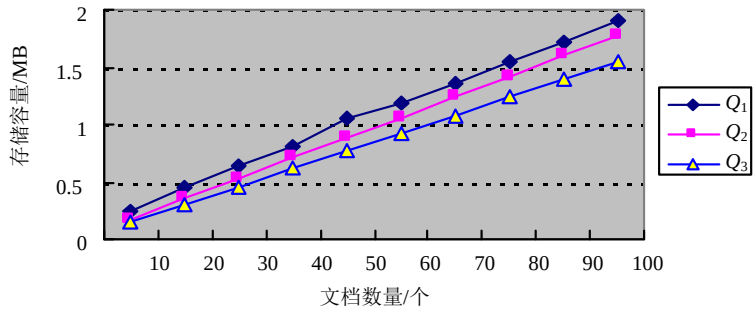

图 4-8　存储容量随文档数量的变化

4.4　小　　结

本章基于 XML 部分函数依赖、传递函数依赖等基本概念给出了 XML 范式等相关定义，给出 XML 模式规范化转换规则，给出规范化的两种算法，对算法的正确性、可终止性和时间复杂性进行证明分析，并实验证明了规范化后的文档在查询时间和存储空间效率上都有明显的改善。