XML 数据库存储策略综述

门爱华[1] 冯建华[2] 周立柱[2]

(赤峰学院计算机科学与技术系 内蒙古 024000)[1] (清华大学计算机科学与技术系 北京 100084)[2]

摘 要 XML是SGML一个子集,本质上是一种特殊的SGML标记语言。XML已经成为Internet上数据表示和数据交换的新标准,被认为是最有前途的一种半结构化数据组织方式。XML的重点是管理信息的数据本身,而不是数据的样式。XML这种明确的分工导致的将是更高效的Web程序设计,更快的搜索引擎、更统一的数据表示和更方便的数据交流的出现。因为底层的存储表达对上层的查询处理和优化有着重要的性能影响,所以如何存储XML文档才是最好的方式已经成为一个重要问题。本文介绍了几种XML数据库的存储策略,并对每种存储策略进行了描述、分析,然后对几种存储策略进行了性能和优缺点的比较。

关键词 XML,半结构化数据,数据库,存储策略,结点,簇集,优化

A Survey on Storage Strategies of XML Database

MEN Ai-Hua[1] FENG Jian-Hua[2] ZHOU Li-Zhu[2]

(Department of Computer Science and Technology, Chifeng College, NeiMongolia 024000)[1]

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084)[2]

Abstract XML is a subset of SGML, and is a specific SGML markup language in its essence. XML has become the new criteria for the data expression and data exchange on Internet and is regarded as the most promising formation way for semi-structured data. The main function of XML is to manage the information data rather than the style of the data. The clear labor division of XML will lead to the appearance of more efficient Web programming, quicker explorer, more consistent data expression and more convenient data exchange. The lower layer storage design has great influence on the upper layer inquiry processing and optimization, so which is the best way to store XML files has become an important issue. The article introduces several storage strategies for XML database, providing careful description and analysis for each of them, and compares their functions and their advantages and disadvantages.

Keywords XML, Semi-structured data, Database, Storage strategies, Node, Clustering, Optimization

1 引言

Web的发展首先应该是HTML标记语言的发展,HTML技术的发展几乎伴随整个Internet的发展,但是HTML技术本身存在诸多缺陷。首先HTML是一种样式语言,在目前Internet上它只是充当了数据表示的主要角色,随着Internet上信息量的增多,HTML变得越来越难以胜任,另外HTML浏览器的过度依赖性也形成了HTML标准的严重不统一,从而导致许多信息表示只能由某种特定的浏览器来解释。HTML的这些不足导致人们重新思考HTML在Internet上的角色并开始研究一门新的语言来弥补HTML的缺陷,XML的产生正是这种重新思考的结果[1]。XML已经成为Internet上数据表示和数据交换的新标准,被认为是最有前途的一种半结构化数据组织方式。XML最明显的特点在于它可以创建标记和文法结构,以便于结构化地描述特定领域的信息,从而提供一种处理数据的最佳方式。无论在数据表示和存储方面,还是在数据的传输和处理方面,XML都有独特的优势[1]。

2 XML数据库类型

所谓数据库就是一组相互有关联的数据集合,而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操作的[2]。目前XML数据库有下面三种类型[3]。

2.1 XEDB

XEDB是能处理XML的数据库(XML Enabled Database)。其特点是在原有的数据库系统上扩充对XML数据的处理功能,使之能适应XML数据存储和查询的需要。一般的做法是在数据库系统之上增加XML映射层,这可以由数据库供应商提供,也可以由第三方厂商提供。映射层管理XML数据的存储和检索,但原始的XML元数据和结构可能会丢失,而且数据检索的结果不保证是原始的XML形式。XEDB的基本存储单位与具体的实现紧密相关。

2.2 NXD

NXD即纯XML数据库(Native XML Database)。其特点是以自然的方式处理XML数据,以XML文档作为基本的逻辑存储单位。针对XML数据存储和查询特点专门设计适用的数据模型和方法。

2.3 HXD

HXD即混合XML数据库(Hybrid XML Database)。根据应用的需求,可以视其为XEDB或NXD的数据库,比较典型的例子是Ozone[4]。

3 XML数据库的存储策略

因为底层的存储表达对上层的查询处理和优化有着重要的性能影响[5],所以如何存储XML文档才是最好的方式已经成为一个重要问题。根据已有的文[6~9],XML数据库的存储策略主要有以下四种:利用文件系统的平面文件、利用成熟的RDBMS(Relational Database Management System,关系数据库管理系统)、利用对象管理器或OODBMS(Object-Oriented Database Management System,面向对象数据库管理系

门爱华 副教授;冯建华 副教授;周立柱 教授,博士生导师。

统)、采用全新的 Native XML 数据库管理系统。

3.1 文件系统的平面文件方法

3.1.1 *平面文件与平面文件数据库* XML 文档本质上是序列化数据,序列化数据通常采用平面文件的形式,即将每一个 XML 文档分别存储在一个文本文件里[6]。这些文件可以采用许多形式中的任何一种,最常用的是重复记录方法。在这些序列化表示中,每条记录都被定义成具有一定数目的字符,并且需要一个独立的文档来描述每条记录的内容[10]。

表 1 平面文件结构描述

起始位置	长度	名称	格式	描述
1	30	姓名	String	客户姓名,长度不足30时右边用空格补齐
31	10	余额	Numeric(10,2)	客户余额,长度不足10时左边用零填充
41	6	到期日	Date	客户帐单的到期日期,MMDDYY

表 2 平面文件示例

FENG Jianhua 0000010023011504Xing Chunxiao 0000150089121203

3.1.2 *利用平面文件存储 XML 文档* 将每一个 XML 文档存储成一个文本文件,并且实现一个查询引擎,当查询被执行的时候,XML 文件被解析成驻留在内存的一棵树。只要查询计算还需要树中的结点,这棵树就必须驻留在内存里。一般来说,解析的时间左右着查询计算的时间,而且这种方法慢得令人无法接受。为了提高这种方法的可用性,必须建立以下索引。利用 XML 元素在文本文件中的偏移量作为 id,建立从标记 tag(parent_offset, tag)映射到子偏移量 child_offset 的路径索引以及从标记 tag(child_offset, tag)映射到父偏移量 parent_offset 的反向路径索引。这两个索引非常有利于遍历 XML 文档或在 XML 文档中进行导航。而其他的索引建立从标记值(tagname, value)或属性值(attribute_name, attribute_value)到元素偏移量 element_offset 的映射,这些索引有助于计算查询中的选择条件(谓词)。查询引擎可以利用这些索引检索与查询相关的 XML 文件的片段,极大地减少了解析的时间[5]。

3.2 传统的关系数据库系统方法

除了平面文件以外,还可以将 XML 文档存储在传统的关系数据库系统里。为了下面叙述方便,表 3 给出了示例所使用的 XML 文档"Dept. xml"和 DTD 文件"Dept. dtd"。并采用如图 1 所示的 XML 文档的树型结构来说明每一种方法是如何实际存储 XML 数据的。XML 文档可以模型化为有向图,图中的结点表示 XML 元素、属性和文本,图中的有向边表示父子关系,其中矩形框表示元素结点,而椭圆形表示属性或文本结点[5]。

表 3 Dept. xml 和 Dept. dtd

```
〈? xml version="1.0"?〉
〈! DOCTYPE Dept SYSTEM "Dept. dtd"〉
〈Dept dept_id="dept1"〉
    〈Student student_id="123"〉
        〈Name〉St1〈/Name〉
        〈Enroll〉CS10〈/Enroll〉
        〈Enroll〉CS20〈/Enroll〉
    〈/Student〉
    〈Student student_id="124"〉
        〈Name〉St2〈/Name〉
    〈/Student〉
〈/Dept〉
```

```
〈? xml?〉
〈! ELEMENT Dept(Student * )〉
〈! ATTLIST Dept dept_id ID #REQUIRED〉
〈! ELEMENT Student(Name, Enroll * )〉
〈! ATTLIST Student student_id ID #REQUIRED〉
〈! ELEMENT Name #PCDATA〉
〈! ELEMENT Enroll #PCDATA〉
```

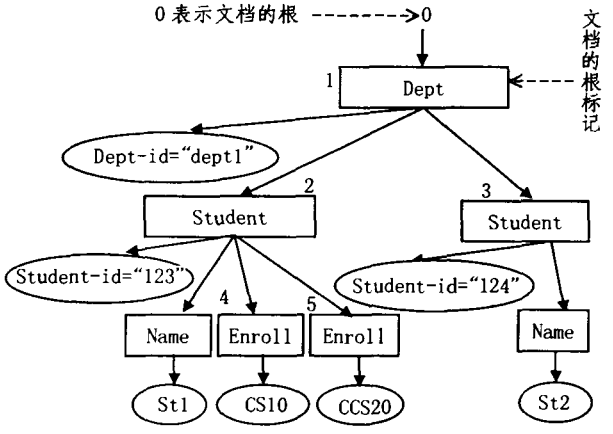

图 1 Dept. xml 文档的树型结构表示

3.2.1 *利用关系的 DTD 方法* 这种方法是文[9]建议的共享内嵌法,而且需要相应 DTD 的支持。用一个独立的表来捕获具有相同标记的元素与其子元素集之间的包含关系。表中的每个元组都被赋予一个 ID,而且该表还包含一个 ParentID 列以标识该元组的父亲结点。一个元素在描述它的元素表中只能出现一次,而且它的父结点内嵌为表的一列以便表示该元素的父亲。如果描述 XML 文档模式的 DTD 图中包含环(如 Dept(Student *)),则必须用一个独立的表来打破这个环。根据 Dept 的 DTD 生成的关系模式如下所示[5]。

表 4 Dept 表

ParentID	ID	Dept_id
0	1	dept1

表 5 Student 表

ParentID	ID	Student_id	Name
1	2	123	St1
1	3	124	St2

表 6 Enroll 表

ParentID	ID	Enroll
2	4	CS10
2	5	CS20

在这种情况下,XML 数据库的索引可以完全建立在上述各个关系表上。

3.2.2 *利用关系的边的方法* 文[7]描述了边的方法,将 XML 文档的有向图存储在单个 Edge 表中。按照广度优

先的顺序赋予有向图中每个结点一个惟一的 id。Edge 表中的每个元组对应有向图里的一条边，并且包含该条边所连接的两个结点的 id 值、目标结点的标记(tag)值以及一个顺序号，该顺序号表示同一结点与其多个子结点对应边的顺序编码。当元素只有一个文本子结点时，文本内容就内嵌到以该元素结点为目标结点的边所对应的元组里[5]。

表 7 Edge 表

SourceID	Tag	Ordinal	TargetID	Data
0	Dept	1	1	NULL
1	dept_id	0	0	dept1
1	Student	1	2	NULL
1	Student	2	3	NULL
2	student_id	0	0	123
2	Name	1	0	St1
2	Enroll	2	0	CS10
2	Enroll	3	0	CS20
3	student_id	0	0	124
3	Name	1	0	St2

在 Edge 表上运用簇集策略对查询的性能也会有重要的影响。一般都会选择按照 Edge 表上的 Tag 字段进行簇集，使得具有相同 Tag 的元素存储在一起。当然也可以选择按照 SourceID 字段进行簇集(表 7 的实际情况就是如此，实际上也是按照边的广度优先顺序)。

3.2.3 *利用关系的属性方法* 文[7]还建议了另外一种所谓的"属性"方法。它是按照 Edge 的 Tag 字段对边表进行水平划分，不同标记(指 Tag 字段的值)的元组存储在各自不同的表里。这种方法以牺牲 Edge 表的非常重要的属性 Tag 为代价而节省了存储空间(不用存储 Tag 字段)。对于属性方法来说，查询处理器需要 DTD 来决定哪些表包含子元素，因为子元素的标记没有存储在表中。需要注意的是，对于有很多 XML 文档的一个大集，属性方法可能导致大量的表[5]。

3.3 对象管理器方法

表 8 保存 Dept.xml 文档对象

Offset	Record(lw_object)
0	Length=40, Dept, parent=nil, prev=nil, next=nil, first_child=40, last_child=140, Attr(dept_id="dept1")
40	Length=40, Student, parent=0, prev=nil, next=140, first_child=80, last_child=120, Attr(student_id="123")
80	Length=20, Name, parent=40, prev=nil, next=100, no children, no attribute, #PCDATA="St1"
100	Length=20, Enroll, parent=40, prev=80, next=120, no children, no attribute, #PCDATA="CS10"
120	Length=20, Enroll, parent=40, prev=100, next=nil, no children, no attribute, #PCDATA="CS20"
140	Length=40, Student, parent=0, prev=40, next=nil, first_children=180, last_child=180, Attr(student_id="124")
180	Length=20, Name, parent=140, prev=nil, next=nil, no children, no attribute, #PCDATA="St2"

(注意：表示一个记录的兄弟之间关系的 prev 和 next 列是按照广度优先的顺序指定的；(? 是否可以保留所有的 child? child 的个数；以及列偏移数组))

在对象管理器里存储 XML 文档的明显方法就是把每一个 XML 元素存储成一个独立的对象[8]。但是，由于 XML 元素通常都非常的小，因此这种方法的空间开销高的令人不敢问津。取而代之，把 XML 文档的所有元素存储在一个单独的对象里，而 XML 元素本身就变成了这个对象里的轻量级对象[5]。同时，文[5]用 lw_object，即表 8 中的 Record 来表示轻量级对象，而用 file_object 表示代表整个 XML 文档的对象。

一个记录，即轻量级对象 lw_object 的格式如表 9 所示。

表 9 轻量级对象的数据格式

EID?	Length	Flag	Tag	Parent	Prev	Next	Opt_child	Opt_attr	Opt_text
NodeID?	√	?	√	√ Page:Slot	兄弟关系 √ 顺序 Page:Slot		定长部分? First/Last:All? Page:Slot	变长部分 String	变长部分 String

轻量级对象(lw_object)在文件对象(file_object)中的偏移量(offset)被用作该对象的标识(lw_oid)。在上面的记录格式中，Length 字段记录轻量级对象的总长度；Flag 字段包含的位用来表示这个对象是否有 Opt_child、Opt_attr 或 Opt_text 字段。Tag 字段表示 XML 元素的标记名；Parent 字段记录 XML 元素父结点的 lw_oid；如果轻量级对象有子结点，则 Opt_child 字段记录 XML 元素的第一个和最后一个子结点的 lw_oid；通过 Prev 和 Next 字段将结点的兄弟链表实现成一个双向链表；Opt_attr 字段记录 XML 元素的每一个属性的(name,value)对；如果文本是 XML 元素的惟一子结点，那么文本数据就内嵌 Opt_tex 字段中；否则，就要把文本数据作为一个单独的轻量级对象来处理(我们的记录格式设计在文本处理上与此有差别)。在文件对象上建立从(Tag, Opt_text)和(attr_name, attr_value)映射到 lw_oid 的 B-树索引。即使 Opt_text 字段是空的，XML 元素也要进入这个索引，以便可以利用这个索引来检索具有特定标记名的所有 XML 元素[5]。当然，文[5]也建立了从(parent_id, tag)映射到子 lw_oid 的路径索引。

4 不同存储方法的性能比较与分析

上面实际上介绍了存储 XML 文档的 5 种方法，本文将通过实验数据比较这 5 种方法的性能。实验使用两个数据集：第 1 个数据集包含 250 个 XML 文档，总共 114MB[11]；第 2 个数据集是 ODP，总共 140MB[12]。实验所使用的计算机的 CPU 为 800MHz 的 PⅢ处理器，有 256MB 内存，所运行的操作系统为 Linux 2.2；实验所用的关系数据库为 DB2 V7.1，对象方法的底层利用系统 Shore[13] 来实现；DB2 和 Shore 都配置为使用 30MB 的内存缓冲池。文本方法中没有缓冲池，其查询处理器使用所有可用的 256MB 的物理内存；文本方法中的索引利用 Berkley DB[14] 来实现。在利用关系数据库的 DTD、Edge 和属性方法中，将用 XQuery 表达的查询手工翻译成 DB2 可执行的 SQL 查询语句[5]。

表 10 显示了每种方法所使用的索引。

表 10 各种方法所使用的索引列表

方法	索引
文本(TEXT)	路径索引、反向路径索引;(tag,data)或(attrname,attrvalue)到元素的偏移量
共享内嵌(DTD)	在包含 XML 数值的每一列上建立了索引;在 ParentID 和 ID 列上也建立了索引
边(Edge)	(Tag,Data)、(SourceID,Ordinal)、(TargetID)
属性(ATTR)	(SourceID)、(TargetID)、(Data)
对象(Object)	(Tag,Data)、(attr_name,attr_value)、路径索引

4.1 存储比

表 11 显示了各种方法所占用的存储空间的多少。

表 11 存储空间比较(除了"存储比"以外单位为:MB)

数据集	比较项目	TEXT	DTD	Edge	ATTR	Object
数据集 1	数据	114	69.7	223	165	104
	存储比	1.000	0.611	1.956	1.447	0.912
	索引	206	29.3	167	130	164
数据集 2	数据	145	126	222	187	160
	存储比	1.036	0.9	1.586	1.336	1.143
	索引	212	132	190	181	192

实验结果表明基本上 DTD 方法是最节省存空间的[5]。

4.2 重构原始的 XML 文档

这个实验测量在当前的存储状态下重构原始数据集的 XML 文档所需要的时间。由于 TEXT 方法把 XML 文档存储在文件系统里,因此对 TEXT 方法来说没有重构时间。其他方法的重构时间按从小到大排列顺序如下:Object<DTD<Edge<ATTR[5]。

DTD 和 Edge 方法按照标记名簇集存储 XML 元素。因此,表中元组的顺序不再反映元素在 XML 文档的原始顺序,这给重构工作招来了很多随机 I/O。在 Edge 的方法中,一条 SQL 语句就可以检索到某元素的所有子元素的元素 ID;而在 ATTR 方法中,必须利用 DTD 信息来决定哪些表包含子元素。找到所有子元素的 SQL 查询语句的个数与可能的标记个数相等[5]。

4.3 不同存储方法的优缺点比较

平面文件是存储 XML 文档的最简单的机制,但一般不支持索引查询,也不容易修改文档[2]。关系型或面向对象数据库按照一定的粒度来存储 XML 文档,这使得对 XML 文档的访问比较容易也比较灵活,同时也提高了查询和修改 XML 文档的效率,而且还可以方便地建立和维护各种索引。

在存储 XML 文档时,一般有三种不同的簇集策略:第一种是对应于现实世界的同一对象的元素簇集存储,例如,将 Student 的 ID 和 Name 存储在一起;第二种是将相同种类的元素一起簇集存储,例如,把所有的 Student 元素存储在一起;第三种是按照和原始 XML 文本文件一样的顺序(深度优先),将元素簇集存储。

关系-DTD 方法积极地使用了策略 1 和策略 2,DTD 信息有助于产生更加紧凑的数据表示,但它的缺点是不能处理没有 DTD 的 XML 文档。利用关系数据库系统还有其他优势:可移植性和可扩展性。另外,一个非常重要的因素是当前 Web 上的数据都驻留在关系数据库系统中,因此使用关系 DBMS 来存储 XML 文档使得查询只有一个系统的数据类型和一种查询语言成为可能[5]。边方法和属性方法都采用了簇集策略 2。当查询必须对相关的几个子元素应用谓词(条件)时,或者是在构造结果文档时,簇集策略 2 都导致了非常糟糕的性能。XML 元素之间的父子关系是由 SQL 的连接(JOIN)运算捕获的。对复杂的路径表达式,这种策略产生了涉及到十几个连接运算的非常复杂的 SQL 查询,使得关系数据库的查询优化器很难产生正确的查询执行计划。连接运算的数目也表明了这两种方法对路径表达式复杂性的敏感度。属性方法比边方法的数据表示更加紧凑。另一方面,为了重构一个元素,属性方法必须要 DTD 的信息。而且其重构的代价比较高,原因就是需要更多的 SQL 查询来获取所有的子元素[5]。

对象方法使用簇集策略 3。由于在原始的 XML 文档中,对应于一个现实世界对象的所有元素常被簇集在一起,所以策略 3 共享了策略 1 的某些好处。当重构查询结果时,策略 3 提供了非常好的性能;而与 DTD 方法比较起来,对象方法中相似的对象(具有相同标记名的元素)没有被簇集在一起的事实显著增加了查询处理的开销。另外,在面向对象数据库基础上实现 XML 数据的存储和查询,而面向对象数据库在查询优化上存在的问题制约了 XML 数据的查询分解和优化[5]。

5 已有的 NXDBMS 存储方法介绍

5.1 OrientX

OrientX 是中国人民大学最近 3 年开发的一个纯 XML 数据库管理系统。存储系统建立在操作系统的文件系统之上,申请若干定长的文件,这些文件属于特定的数据集,数据集用 SetID 来标识;在文件上划分逻辑物理块,物理块定长,是操作系统物理块整数倍,物理块用 LpNo 来标识。给定一对(SetID,LpNo),能马上找到对应文件的相应的偏移量。存储管理以记录为单位,不同的存储粒度,记录的物理含义不同,可能是一棵子树,也可能是一个结点。虽然 OrientX 采用了 DEB、CEB、DSB 和 CSB 四种不同的存储方法,但对上层查询提供的接口都是一样的,这样可以提高数据的独立性。

5.2 Tamino

德国的 Software AG 于 1999 年发布的 Native XML 服务器的第一个版本 Tamino,可以处理面向数据的文档和面向文档的文档,也可以存储其它类型的数据,如图像、HTML 文件等。Tamino 数据库由很多所谓的数据集(collections)组成,数据集仅仅是一组文档的包装器,每个文档只能存储在一个集里。每个集都有相关的 W3C 的 XML Schema 描述,在模式里,doctype 是个特殊的概念,它标识了在 XML Schema 中作为根元素声明的所有元素中的一个。在集里,每个文档都只能作为一个 doctype 的成员被存储。文档的根元素标识了 doctype,因此,在一个集里 doctype 和根元素类型是 1∶1 的关系。如果文档存储在一个集里,而没有 doctype 和文档的根元素相对应,这时就会动态地创建一个 doctype。在这种情况下,就没有相关的用户定义的模式。万一集里有一个相关的用户定义的模式存在,Tamino 就对照这个模式验证输入文档。Tamino 的元数据也以 XML 文档的形式存储在系统集里的系统 doctypes 中。

5.3 Timber

Timber 是密歇根(Michigan)大学于 1999 年开始开发的一个 NXDBMS。Timber 有数据存储、索引存储和元数据存

储三种存储类型。Timber 基于非常流行后端存储系统 Shore[15]，Shore 负责磁盘存储管理、缓冲和并发控制。Timber 中的结点是广义结点，将所有的属性集成为一个子结点，将元素的文本内容单独作为一个子结点。Timber 将与元素相关的子元素簇集在一起，即按照文档的顺序(或者是深度优先的前序遍历顺序)簇集存储结点，等价的方法就是按照 NodeID 的 S(Start)值来簇集存储结点，实际上就是 DEB 的存储方法。Timber 目前只实现了单结点索引，索引可以在属性值上或者元素内容上(当元素内容可以识别为一个数时)构造。当然，当元素内容为大文本时，也可以构造基于关键字的倒排索引。还可以在标记名 Tag Name 上建立索引(到相应元素的映射)。也就是说，给定一个标记名 Tag Name 就可以返回具有特定标记的所有元素。在 XML 数据库管理系统中，很多时候，元数据不是很显眼。元数据信息包括属性类型、数据集大小和索引结构等，而用于代价估计目的的直方图则是新的元数据。

5.4 Natix

德国曼海姆(Mannheim)大学开发了 Natix，目的是用来处理树结构的大对象，特别是 XML 文档。与传统的大对象的管理相比，Natix 不是在任意字节位置分割对象，而是要考虑 XML 文档的基本树结构的语义，将包含相关结点的一棵子树看作是一条记录。因此 Natix 采用是 DSB 和 CSB 之间的一种存储方法：划分子树时要考虑文档树结构的语义，但是按照深度优先的顺序存储子树对应的记录。Natix 的特点在于其强大的数据存储能力，是子树级的中粒度存储方法的代表。虽然在划分子树时考虑了文档树结构的语义，但总体上还是根据物理块大小而非逻辑意义来划分子树，因此主体上仍属于 DSB 方法的范畴。

5.5 Lore

Lore 是 Stanford 大学数据库研究组为美国空军开发的一个轻量级的半结构化数据的数据库管理系统。Lore 将对象安排在物理的磁盘页上，每个页都有许多槽，每个槽里都有单独的一个对象，所以 Lore 采用的是分槽页结构来组织数据。由于对象是变长的，所以 Lore 根据首次适合(first-fit)算法来放置对象，并且提供对象转向机制来处理由于增长变大而不能在本页继续存在的对象。另外 Lore 支持可以跨越多个页的大对象，这样的大对象对多媒体类型，以及具有非常宽的扇出的复杂的对象来说是有用的。在页里对象按深度优先的顺序簇集在一起，这主要是因为 Lore 的基于 scan 的计划是按照深度优先的顺序遍历数据库的。Lore 采用的是狭义结点、DEB 的存储方法。

讨论 初步的实验结果表明要想获得好的性能和紧凑的数据表示，相关 XML 文档的 DTD 信息是至关重要的。当 DTD 信息可利用时，DTD 方法的数据表示就很紧凑，而且该方法对不同的数据集和不同的查询均表现出优秀的性能。问题是有很多的 XML 文档无 DTD 或者是 XML 文档被用作标记语言。目前，NXD 既能存储合法的 XML 文档(有 DTD 或 XML Schema 支持)，又能存储良构的 XML 文档(无 DTD 和 Schema 支持)。NXDBMS 从存储、索引、查询等方面，都围绕着 XML 的特点和标准进行设计，它没有像关系数据库方法那样繁杂的数据转换。当然 NXDBMS 在理论和应用上都远远没有关系数据库系统成熟，因此，对 NXDBMS 的研究还是一个长期而艰巨的任务。

参 考 文 献

1 栗松涛编著. XML 程序设计. 清华大学出版社，2001

2 (美)Mark Graves 著，尹志军，等译. XML 数据库设计. 机械工业出版社，2002

3 http://www.xmldb.org/faqs.html

4 http://www.ozone-db.org/frames/home/what.html

5 Tian Feng, DeWittD J, et al. The Design and Performance Evaluation of Alternative XML Storage Strategies. SIGMOD Record, March 2002, 31(1)

6 Abiteboul S, Cluet S, et al. Querying and updating the file. In: Proc. of 19th International Conference on Very Large Data Bases, Dublin, Ireland 1993.

7 Florescu D, KossmanD. Storing and Querying XML Data using an RDBMS. IEEE Data Engineering Bulletin, September 1999, 22(3)

8 Kanne C, Moerkotte G. Efficient storage of XML data. In: Proc. of the 16th International Conference on Data Engineering, 28 February - 3 March, 2000, San Diego, California, USA. IEEE Computer Society 2000

9 Relational Databases for Querying XML Documents: Limitations and Opportunities, In: Proc. of 25th Intl. Conf. on Very Large Data Bases, Edinburgh, Scotland, UK 1999

10 Williams K. 用于数据的 XML：重用它还是丢弃它——企业中的 XML 重用. http://www-900.ibm.com/developerWorks/cn/xml/x-xdreuse/index.shtml, 2003

11 Carey M, DeWitt D, et al. The BUCKY Object-Relational Benchmark. In: Proc. of ACM SIGMOD Intl. Conf. on Management of Data, May 13-15, 1997, Tucson, Arizona, USA. SIGMOD Record, June 1997, 2(12)

12 Open Directory Project, http://www.dmoz.org/

13 Carey M, DeWitt D, et al. Shoring Up Persistent Applications. In: Proc. of ACM SIGMOD Intl. Conf. on Management of Data, May 24-27, 1994, Minneapolis, Minnesota, USA. SIGMOD Record, June 1994, 23(2)

14 Berkeley DB toolkit, http://www.sleepycat.com/

15 http://www.w3.org/MarkUp/SGML/