XML 在关系型数据库的存储实现*
XML Storage in Relatonal Database

陈 挺** 徐立臻
(东南大学 南京 210096)

摘要:从介绍 XML 和 XML Schema 出发,在分析已有的 XML 文档在关系型数据库存储方法上,给出一种新的 XML 文档在关系型数据库存储的模式。

关键词:XML;关系数据库
中图分类号:TP311.132 **文献标识码**:B **文章编号**:1672-4550(2005)02-0032-04

1 XML 简介

XML 是可扩展标记语言 (XML: Extensible Markup Language 的缩写), HTML 是大部分人都熟悉的一种标记语言, 但 HTML 只用来显示数据, 标记少且能力有限。而 XML 可使用自定义的结构来存储专有的数据, 实际上, XML 脱胎于通用标记语言标准 (Standard for General Markup Language)。但 SGML 非常复杂以致于难以得到大规模的应用。作为 SGML 的子集, XML 由于取消了一些复杂性, 在易用性方面得到很大改革, 尤其是在 Web 应用上, 各种基于 XML 的应用和服务大量出现, 也推动了 XML 规范的制定和完善。XML 规范现在是由 W3C (World Wide Web Consortium) 来负责制定发布。

在文献 [1] 中, Steven 总结了 XML 的三个优点:

(1) 易于数据交换;
(2) 可定制标记语言;
(3) 自描述的数据。

2 XML 和关系型数据库

XML 为数据库处理提供一系列的接口, 例如 DTD, XML Schema, XQuery, XPath, XQL 等等。对于编程来说同样有 SAX (Simple API for XML) 和 DOM (Document Object Model)。但对于 XML 文档来说, 最关键的问题是缺乏高效的存储、索引, 事务处理和数据整合, 还有多用户使用多个 XML 文档, 在多个 XML 文档之间查询等等问题。XML 文档着重于提供统一的数据格式而不是数据库方面的特色, 因此, 数据库在 XML 应用程序中的角色和传统应用程序中的角色是一样的。

朱亮在文献 [2] 中提出了一个观点: XML 文档的树形结构是不同于关系数据库中的二维结构, 这就导致了两种截然不同的存储 XML 文档的数据库类型, XML 化 (XML - enabled) 数据库 (XED) 和原生 XML (Native XML) 数据库 (NXD)。

3 存取 XML 文件的数据和内容

在文献 [3]、[4] 中, Ronald 提供了两种把 XML 文档中数据存储在关系数据库中的办法。

3.1 基于表格的映射办法

基于表格的映射办法是把 XML 文件看成一个表或一组表的结合, 如下所示:

```
<A>
  <B> <C> C111 </C> <D> D111 </D> <E> E111 </E> </B>
  <B> <C> C222 </C> <D> D222 </D> <D> D222 </D> </B>
  ……
</A>
```

可以看成如下的表格形式:

* [收稿日期] 2004-09-22; [修订日期] 2004-10-27
** [作者简介] 陈 挺 (1976—), 男, 东南大学硕士研究生, 研究方向为软件工程。

Table A		
.....		
C	D	E
C111	D111	E111
C222	D222	E222
.....		

3.2 对象 - 关系映射方法（Object - Relational Mapping）

这个映射模型把 XML 文件中数据看成对象树。在此模型中，元素类型，元素属性，元素内容都被看待为类似于面向对象编程语言中的类（classes），譬如元素类型为 PCDATA 的元素内容都被看成标量属性。对于此模型可以使用传统的关系 - 对象映射办法把数据存储到数据库中去，类可以映射成一组表格，标量属性可以映射成表里的行记录，而对象之间的关系可以用表格之间的内 - 外键关系来处理。以下说明这种处理办法的基本结构：

```
DTD:
    <! ELEMENT A (B, C) >
    <! ELEMENT B (#PCDATA) >
    <! ELEMENT C (D, E) >
    <! ELEMENT D (#PCDATA) >
    <! ELEMENT C (#PCDATA) >
CLASS:
    CLASS A {
        String B;
        C    c;
    }
    CLASS C {
        String D;
        String E;
    }
```

进而可以看成表格：

```
Table A
    Column A_ PK
    Column B
    Column C_ FK
Table C
    Column C_ PK
    Column D
    Column E
```

如果 B 和 C 对于元素 A 是可以选择的，那么字段 b 和 c_ fk 的值可以为空。如果 A 至少有一个元素 B，那么数据库映射时需要增加一个表 B 来补充数据描述，这样表 A 就会是（a_ pk，c），表 B 将是（a_ fk，b）。

这种方法有大量的规则来处理从 DTD 到数据库表模式之间的映射。基本方法是：

（1）对于每一个复杂元素类型（有子元素），建立一个表和主键。

（2）对于每个有混合内容的元素（PCDATA），建立一个单独的表存储 PCDATA 的值，通过内外键与其父元素之间建立联系。

（3）对于有单值属性的元素和其简单子元素（无后继元素），建立一个单独的表来描述，并且此表的数据类型可以和元素的数据类型一致。如果数据库没有对应数据类型，那就设置字段类型为数据库中预定义的类型，如 CLOB 或者 VARCHAR（255）等等。当属性或简单子元素对于父元素是可选时，设置字段属性可以为空（nullable）。

（4）如果元素有多值属性并且其子元素出现次数可为多次，那么建立单独的表去描述属性值和子元素。

（5）对于复杂元素，描述父元素和子元素的表之间通过内外键进行联系。

对于方向的数据处理——从数据库的记录恢复 XML 文档，那么建立类似的处理规则即可处理。

3.3 DTD 无关的存储方法

在文献［5］中，Tomas 给出了一种在关系数据库中存储 XML 文件的较为灵活的方法，其基本思想是把 XML 文档看成是有方向，有顺序，并且是标记好的图。子元素和元素属性用一个节点延伸出来有顺序的边来表示。属性，子元素，应用都可看作是边。基本的数据库表结构如下：

```
Edge (Sourceid, targeted, leafed, attrid, docid, type, name, depth)
Leaf (id, value)
Attr (id, value)
Doc (id, value)
```

XML 文档的内容由 leaf 和 attr 两个表来存储。它们和表 edge 的联系是通过 leafid 和 atrrid 来实现的。如果把文档看作树的话，那么每条边都有源节点和目标节点。对于此方法来讲，为了实现处理多文档（由表 doc 来实现），每条边都被设置一个 docid，字段 type 是用来区分边的类型，可以是 leaf，attribute 或者引用。字段 name 用来存储元素名，字段 depth 是用来查询数据库产生 XML 文档。

如果用这个办法来存储 XML 文档，初始化数据库必然要用到 SAX 来解析 XML 文档。作者 Tomas 给出了示例来说明这个方法，如下所示：

```
<tree>
<person age = " 36".
  <name>Peter</name>
  <address>
    <street. Main Road 4</street>
    <zip>04236</zip>
    <city>Leipzig</city>
  </address>
  </person>
  </tree>
```

Edge

Sourceid	targetid	leafid	attrid	docId	name	type	depth
0	1	NULL	NULL	1	tree	ref	3
1	2	NULL	NULL	1	person	ref	2
2	3	NULL	1	1	age	attr	0
2	4	1	NULL	1	name	leaf	0
2	5	NULL	NULL	1	address	ref	1
5	6	2	NULL	1	street	leaf	0
5	7	3	NULL	1	zip	leaf	0
5	8	4	NULL	1	city	leaf	0

Leaf

id	Value
1	Peter
2	Main Road 4
3	04236
4	Leipzig

Attr

id	Value
1	36

Doc

doc	url
1	sample.xml

图 1　示例表格

3.4　几种方法的比较

上述几种方法，对于结构比较规范的 XML 文档来说，处理较为方便，例如销售记录，图书或电影记录等等。描述事物需要的信息量少，即一个事物的属性不是很多，但对于描述项多的事物，采用第（1）、第（2）种方法时，表结构中的空字段增多，冗余数据存取较多，并且由于表间通过内外键联系，存取路径和代价开销较大，不适合采用。并且对于嵌套的元素，不能很好地表达 XML 文档的数据结构，如果应用程序需要改变 DTD 文件，那么数据库表结构就需要作相应修改。这样如果应用程序是基于存储的 XML 文档时，问题随之而来。如何才能保证 DTD 改变而数据库模式不变，第（3）种方法给出了很好的思路，不去专门描述 DTD，直接对每一个在 XML 文档中出现的元素给以类似定义的描述和存储。实际上，笔者认为这样做是很浪费存储空间的，DTD 的定义内容远少于 XML 文档的大小，XML 文档里的元素肯定是 DTD 里定义的元素反复出现，没有必要反复存储元素的定义信息。

3.5　对 XML 文件在关系数据库中存储的改进方法

```
<! ELEMENT root (A1 | A2 | A3 * >
<! ELEMENT A1 (B1?, B2 *, A2 *) >
<! AT TLIST A1 name ID #REQUIRED
        version CDATA #REQUIRED
        release CDATA #REQUIRED >
<! ELEMENT A2 (C1 *, C2 *, A3?) >
<! ELEMENT A3 (B1 *, C1?) >
<! ELEMENT B1 (#PCDATA) >
<! ELEMENT B2 (#PCDATA) >
<! ELEMENT C1 (#PCDATA) >
<! ELEMENT C2 (#PCDATA) >
```

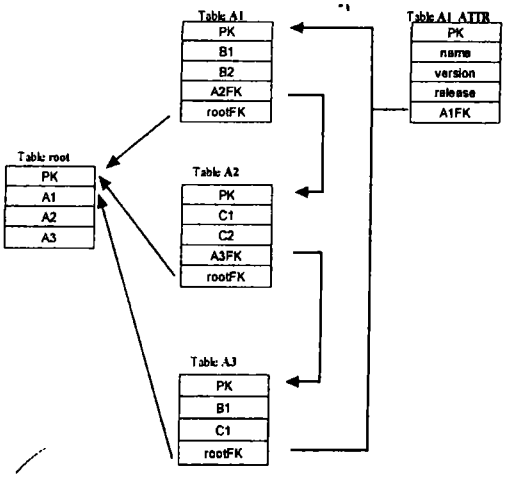

图 2　采用表格映射的办法

从图 2 可以看出，如果要重建 XML 文档，那么需要很多的 join 查询，并且从存储效率上来说，A1，A2，A3 是选择出现，极端情况下在表 root 中会出现大量空值。对于第（3）种方法提出的记录多文档的手段是 docid，对于一个文档来说，这个记录值重复次数和文档中节点个数是一样的。

笔者的方法基于 DOM 概念，还是以 3.5 节的 DTD 为例，转化成树形结构如图 3 所示。

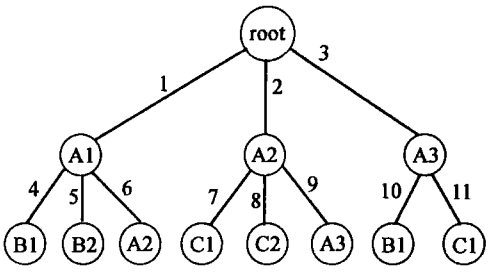

图 3　DTD 的树形结构

考虑单个 XML 文档的情况，无论 XML 文档

的内容如何，父元素和子元素的固定关系均可在图 3 中找到，即节点间的关系是固定，root 节点后面的的关系只可能是 1，2，3。

首先用三个基本表描述 DTD，然后基于 DOM 中节点的概念用另外两个表来描述存储实际的 XML 文档。数据库表结构如图 4 所示。

虚线左边存储 DTD 描述内容，表 element 存储 XML 文档中所有可能出现的元素的基本信息，表 attribute 存储元素的属性名和属性值类型信息，不存储属性值。表 Relationship 存储元素之间的关系，即父/子元素之间的关系，虚线右边的表格存储 XML 文档里的节点，XML 文档里的节点类似于左边元素表里的表项的实例化，并且前后节点间的关系在表 relationship 都能找到，下面以一个以 3.5 中 DTD 为基础的 XML 文档为例说明：

```
<? XML version = " 1.0" encoding = " UTF - 8"? >
<! DOCTYPE root SYSTEM " Untitled2. dtd" >
<root >
<A1 name = " test" version = " 1" release = " 8" >
    <B1 > B1 - 1 </B1 > <B2 > B2 - 1 </B2 > <B2
> B2 - 2 </B2 >
    <A2 >
      <C1 > C1 - 1 </C1 > <C1 > C1 - 2 </C1 > <
C2 > C2 - 1 </C2 > <C2 > C2 - 2 </C2 >
    <A3 >
      <B1 > B1 - 2 </B1 > <B1 > B1 - 3 </B1 > <
C1 > C1 - 3 </C1 >
    </A3 >
  </A2 >
</A1 >
</root >
```

实际的存储树如图 5 所示，可以看出在实际的树中，前后关系不会超出图 3 中的范围。

4 结论

XML 在关系数据库的存储是一个比较实际的课题，本人在爱尔兰爱立信软件园做项目期间正好碰到这样一个问题需要解决，并且没有使用商业化的大型数据库软件来实现，实现效果和项目评价较好。项目的要求用大型商业数据库 XML 存储功能也不能满足。XML 的规范还在不断修订之中，而关系型数据库从工程角度讲还有相当长的寿命，在关系数据库的基础上还有相当大的空间给新技术整合。

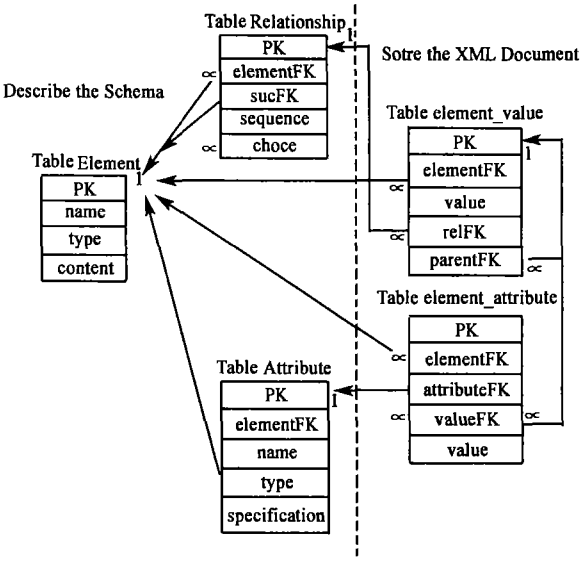

图 4 数据库表结构

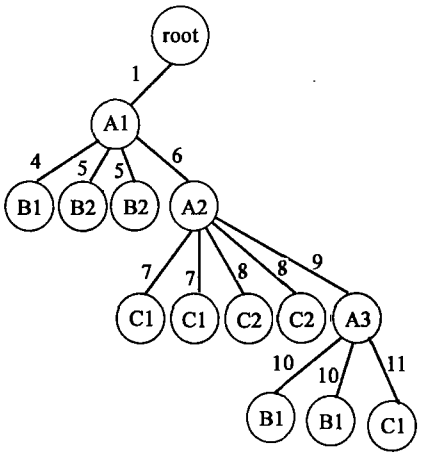

图 5 示例 XML 文档的 DOM 图

参 考 文 献

[1] Steven Holzner. Sams Tech Yourselc XML in 21 Days. Third Edition (Sams Publishing 2003) chapter 1

[2] Steven Holzner. Real World XML (New Riders Publishing 2003) chapter

[3] Ronald Bourret. XML and Datbase. http: //www. rpbourret. com/xml/XMLAndDatabase. htm

[4] Ronald Bourret. Mapping DTD To Database Schema. http: //www. xml. com/pub/a/2001/05/09/dtdtodbs. html

[5] Thomas Kurdrass. Management of XML Documents with - out Schema in Relationship Database System. Information and Software Technology, 44 (2002) p269 - p275. http: //www. elsevier. com/locate/infsof