韓國컴퓨터情報學會 論文誌
第14卷 第3號, 2009. 3.

2009-14-3-1-6

X2RD: XPath를 이용한 XML 데이터의 관계형 데이터베이스로의 저장과 질의

오 상 윤*

X2RD: Storing and Querying XML Data Using XPath To Relational Database

Sangyoon Oh *

요 약

XML은 웹 환경 정보의 표준으로 자리 잡고 있으며, 웹 서비스, 시멘틱 웹 등의 출현으로 XML을 이용한 정보교환은 더욱 확산될 것으로 예상되고 있다. 대부분의 데이터들은 관계형 데이터베이스에 저장되어 있으므로 XML 데이터의 저장과 질의에 관계형 데이터베이스를 이용하려는 연구가 최근 주목을 받고 있으며, 특별히 XPath, XQuery 들과 같은 XML 관련규약들을 지원하는 방식에 대한 시도가 이루어져 왔다. 본 논문에서는 기존에 제안된 XML을 관계형 데이터베이스에 저장하고 질의를 수행하는 구조들의 특성들을 분석하고, 관계형 데이터베이스를 이용한 새로운 XML 저장 및 질의 방식을 제안한다. 제안된 방식은 XML 데이터를 분할 (Shred) 하여 관계로 표현하며, XQuery의 기본이 되는 XPath를 이용한 Query를 SQL로 변환하여 적용하는 구조를 가진다. 본 제안 방법론을 이용하여 Query Processor를 구현하고 실제 RDBMS를 연동하고 실험한 결과, XML 데이터를 효과적으로 RDBMS에 효과적으로 저장하고 질의할 수 있는 것을 확인할 수 있었다.

Abstract

XML has become a de facto standard for structured document and data on the Web. An XML data deluge over the network will be more, since XML based standards such as Web Service and Semantic Web gets popular. There are efforts to store and query XML documents in a relational database system and recent efforts focus on how to provide such operations using XPath and XQuery. In this paper, we present study about those research efforts and we propose a new scheme to store and query XML documents in a relational database using XPath query. The scheme uses a 'shred' method to store and translates XPath queries to SQL. We also present our empirical experiments using a RDBMS.

▸ Keyword : XML, 관계형 데이터베이스(relational database), XPath 질의(XPath Query)

• 제1저자 : 오상윤
• 투고일 : 2009. 2. 25, 심사일 : 2009. 3. 9, 게재확정일 : 2009. 3. 23.

* 아주대학교 정보및컴퓨터공학부 조교수

※ 본 연구는 지식경제부 및 정보통신연구진흥원의 대학IT연구센터지원사업의 연구결과로 수행되었음 (IITA-2009-C1090-0902-0003)

Ⅰ. INTRODUCTION

XML (eXtensive Markup Language) has become a de facto standard for structured documents and data on the Internet [1,2]. We are already experiencing an XML data deluge over the network and there will be more, since XML based technologies such as Web service technology get popular. Web service enables a language and platform independent communication between distributed services. Thus, more messages (e.g. SOAP[2], WSDL[3], and RDF[4]) on the network are in the form of XML.

In general, there are options to store and query XML data. We can store XML data in a flat file system or Native XML databases such as Xindice of Apache [3] and Ozone Database [4]. Object-oriented databases can also be used to store and query XML data. There are advantages and disadvantages to use each system to store and query XML data. Even though a flat file system gives us maximum easiness to store XML data, however it provides us almost no way to query the XML data. Native XML database has many advantages including processing complex queries to large databases, however it seems the performance and the maturity of the systems are not there yet.

Thus, it becomes more important to store XML documents in a relational database. This is mainly because a relational database system has better performance than native XML databases which store XML documents in raw format. For such reason, there are efforts to decompose XML documents into relations and store XML in a relational database among major industrial companies and academic institutes. Those efforts naturally include the efforts to query to XML contents in a relational database.

In this paper, we propose a new scheme to store and query XML documents XPath (XML Path Language) [5] to a relational database. Our proposal includes two major issues. A proposal of method for mapping a raw XML document in a relational database system would be one, and a scheme to implement XPath queries to the XML data in a relational database would be the other.

In order to demonstrate the potential of our proposed scheme, we implement the scheme and it will be described in the following sections, which focuses on traversing capability of XPath. We organize this paper as follows. In section 2, we discuss background works. In section 3, we present our study about XPath Query Mapping to store and query XML data on a relational database. Section 4 shows the implementation for verifying the proposed scheme. We discuss about the future work and conclude in section 5.

Ⅱ. RELATED WORKS

As we pointed in the previous section, there are some research efforts to store and query XML documents in a relational database system [6~9, 15]. Researchers have proposed options to store and query XML documents by shredding them into relations and translate XML queries into SQL queries. Options are vary from very simple one which asks a user input how to store XML elements in relational tables [10]. It is simple and easy, however it requires human interruptions in the process and error introduced by human can cause fatal malfunctions. We discuss three notable works in following subsections.

1. A simple ad-hoc scheme: Florescu and Kossmann

Florescu and Kossmann [6] analyze no user input. They scan and parse XML documents one by one and store all the information into tables on a relational database. There are assumptions on their study: first, they process an XML document as of an ordered and labeled directed graph. Each XML element is represented by a node in the graph. Relationships between elements and subelement are

[2] Formerly known as Simple Object Access Protocol
[3] Web Service Description Language
[4] Resource Description Framework

represented by edges in the graph and labeled by the name of the subelement. Values of the XML documents are represented as leaves in the graph. This approach inspires us in the way they used to store XML documents by shredding them.

2. Supporting the ordered XML data model: Shanmugasundaram et al. (IBM Almaden RC)

Shanmugasundaram and colleagues at IBM Almaden Research Center show how ordered XML data model can be efficiently supported by the relational data model [8]. They proposed three ordering methods (i.e. Global Order, Local Order, and Dewey Order) for representing XML order in the relational data model. The proposal also includes algorithms to translate an XPath query into a SQL query for each encoding methods. This approach resembles with our work in shredding XML documents to store and using XPath to query. However, the focuses of works are different: their main interest is to show that XML ordered data model can be efficiently stored and queried using relational database. Our interest is to show an alternative option to store and query and verify the option through implementation.

3. Oracle XML DB: Oracle Database 11*g*

Oracle XML DB is a feature of Oracle Database 11*g* [11]. It provides a high-performance, native XML storage and retrieval technology. It fully absorbs the W3C XML data model into the Oracle11g Database, and provides new standard access methods for navigating and querying XML. For instance, it supports a new data type, 'XMLType'. This gives user the flexibility to store XML in a column, in a row, or as a whole file, like object. Oracle XML DB user can query to XML in two ways. First, it supports SQL-style queries, in which users can use XPath notation to 'step into' XML documents. This style of querying, known as SQL/XML, is being standardized by the SQLX group [12]. SQL/XML is an extension to SQL to include processing of XML data in relational stores. When data is mixed of structured and semi-structured, SQL/XML is the preferred way to query. Oracle11g Database also supports XQuery [13], which is a prototype implementation in Java. XQuery, which is also known as XML Query in conjunction with XPath, is a standard way to access pure-XML documents. It is the most relevant when the data is in XML structure.

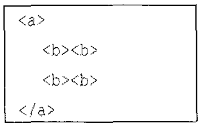
Works discussed above provide schemes to store and query XML documents. In this paper, we propose and implement yet another one which is simple and ad-hoc. Section 3 shows our proposed scheme.

III. PROPOSED SCHEME

In this section, we describe our proposed scheme to map an XML DOM into a single ternary relation. Thus, XML documents in relational database are stored and queried. We also describe our simple language to query onto the stored document which is XPath compatible.

1. Simple Mapping

There are several techniques to map an XML document in the relational database as we presented in section 2. We used rather simple form of mapping by using ternary relationship. In our scheme, XML DOM is parsed down and each XML element is stored as a single ternary relation (i.e. index of the node, parent, and the name of the node). For instance, let's map a size 2 document A (Fig 1.). The parse tree of the document is depicted in Fig 2.



```
<a>
    <b><b>
    <b><b>
</a>
```

그림 1. 크기2인 XML 문서
Fig. 1. A size 2 XML Document A

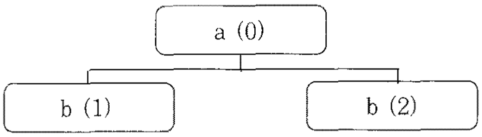


그림 2. Indexing된 XML 문서 A의 트리
Fig. 2. Parse tree of the Document A with index

To map a parsed XML DOM tree to a ternary relation, we introduce a Mapping function P and we define it as follows:

```
P:dom → num X num X string
```

Thus, if we put input A (i.e. XML Document A) to the function, it will be

```
P(A)={<x.index,x.parent_index, x. node_name>|x
is a member of DOM}
```

and the relation is presented in Table 1.

In this mapping, we define all nodes as the same type to make the mapping process simple. If you consider the retrieval of an XML document from the relational database, it is more complex task than storing and guaranteeing an identical XML document is another level of challenge, since some information can be lost when you store. In this paper, we focus on to show the alternative scheme to store and query using XPath. Thus, we believe that our simple scheme approach is enough.

표 1. XML 문서 A의 관계형 테이블
Table 1. XML Document A to a relational table

Index	Parent	Name
0	0	a
1	0	b
2	0	b

For the same reason, the attribute of an element becomes a child node of the element. Also the value of an element becomes a child. That is `<a id=1234>` and `<a>1234</a>` modified to `<a><id><1234></1234></id></a>` and `<a><1234></1234></a>`, respectively.

2. Query Language

The language we use in our proposed scheme is subset of the W3C XPath language. We implemented three XPath axes and a node test. The XPath is syntax for defining parts of an XML document and it uses path expression to navigate in XML documents. We choose to use XPath because XQuery is designed to query XML data (i.e. SQL for XML data) and XPath is a major element in it. XQuery and XPath share the same data model and support the same functions and operations. Thus, extending our query language to implement the full XQuery specification is matter of engineering issue.

Terms in the XPath specification (e.g. children, parent, descendant, and ancestor) are semantically shared with XSLP, XQuery and XML and there is no need to re-define terms for nodes. However, XPath Axes are not in the XML and needs to be explained. An axis, which is one of expressions in XPath, is a node-set to the current node. Examples from the XPath recommendation of the W3C show how we can address parts of an XML documents. Examples of axes with a node test operator are follows:

- `child::*`
  : selects all element children of the current node
- `descendant::para`
  : selects the para element descendants of the current node
- `ancestor::div`
  : selects all div ancestors of the current node

Our scheme supports all examples above. However, we defined axis, and node test symbol syntactically little different from W3C recommendation. We defined P, C, D as a name of parent axe, a child axe, and a descendant axe, in that order. And the node test symbol '::' from W3C recommendation is replaced by ';'. Thus, our proposed language expression 'C;para' is equivalent to the 'child::para' of W3C XPath recommendation.

표 2. W3C 규약과 제안된 방식의 비교
Table 2. Comparison between W3C recommendation and proposed scheme

항목	값
parent	P
child	C
descendant	D
:: (node test)	; (node test)

IV. EVALUATION

To verify the feasibility, we implement our proposed scheme with the PostgreSQL relational database management system [12] which is under BSD-style license and freeware. In this evaluation, we made few assumptions to make our problem domain small enough but not miss any crucial issues for verification. Our implementation is for per schema. It is the option which Oracle 8*i* and Florescu and Kossmann used in their work. A structure of the XML document is given by the user before we make a query to the database. The second assumption we made is that a XPath expression which user input has no syntactical error in it. And the third and final assumption is our implementation gives no node manipulating capability.

1. Components of X2RD

Our implementation, X2RD (XML to Relational Database), is designed to support the XPath query implementation to an XML document in the relational database. As listed in table 2, X2RD is consisting of two components - JDBC module, which is integrated with Java Swing GUI (Graphical User Interface) and 'plpgsql' function that make a query to the PostgreSQL RDBMS. In the system, JDBC (Java Database Connectivity) defines the interface to the Postgres database in order to provide methods querying and updating data in the database from GUI.

2. Event flow of X2RD

Events in the X2RD generates from the GUI (Graphical User Interface) to the Postgre ORDBMS using JDBC and PostgreSQL. On the GUI, user inputs a well-formed XPath expression and an expression is possibly consisting of multiple location steps. A whole XPath expression is parsed into each location step and put into the Java Vector object. The JDBC module integrated with GUI make a query pl/sql statement for each location step. An implementation issue we've faced is where we are going to store intermediate result. X2RD do not bring the intermediate result into main memory. Rather, a result of each location step is written back to the database by pl/sql function. The 'temp' table is a temporary table where pl/sql functions write an intermediate result to. Since pl/sql function iterate through the result (node set) from the previous stage, we can avoid a possible memory overflow that can cause by a huge size of intermediate result set. After the query processing, the final state of 'temp' table is displayed on the GUI. The display area in GUI is a scrollable pane so that there is no missing data.

In detail, the each axe has a corresponding pl/sql function, except D (descendant) which has two designated functions. Each pl/sql program processes a location step. JDBC class makes queries the Prepared Statement with a node name to be tested according to the axes. The descriptions of Java classes and pl/sql functions are following.

2.1 pl/sql

Except ancestor_descendant.sql, all the pl/sql functions loops over the previous node set in 'temp' table with cursors. They select the node with the combination of index and parent column value. For instance, the result set of node for C (Child) axe are selected by setting '*where*' clause as *parent* = *ind* and *index* <> *ind* where *ind* is the index of node where the cursor locate currently. Then, there is going to be a condition more of '*where*' clause to do the node test, such as name = name of node. In the case of node name '*', pl/sql function does not perform the node test, however returns all node of the axe result set.

- descendant.sql

: function that selects descendant of current context nodes (which are stored in 'temp table) and do the node test over selected set of nodes.

- parent.sql

: function that selects parent of current context nodes (which are stored in 'temp' table) and do the node test over selected set of nodes.

- child.sql

: function that selects child of current context nodes (which are stored in 'temp' table) and do the node test over selected set of nodes.

- ancestor_descendant.sql

: function that returns a table that contains the descendant of given node. This recursive function traverse down until function reaches down to the leaves of the sub-tree of current node.

2.2 Client Modules

X2RD client consists of X2RD and User GUI. The descriptions for each module are as follows:

- X2RD

: this is main module that create user frame initially. It also manages all the JDBC issues. It loads up org.postgresql.Driver and prepares each statement for the sql call. It provides the method to initiate 'temp' table and to display the final state of it.

- GUI

: It takes the user input XPath query and parses it to each of location step. It loops for each location step processing until the final location step is processed.

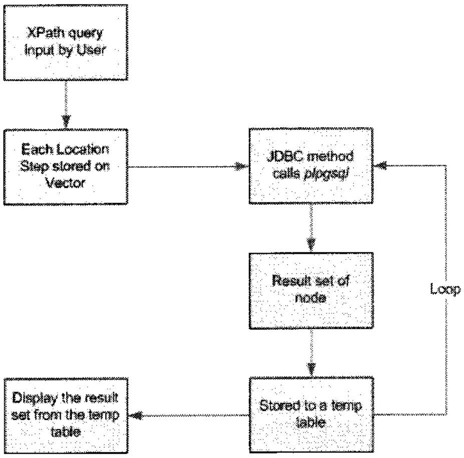

XPath query Input by User
Each Location Step stored on Vector
JDBC method calls plpgsql
Result set of node
Loop
Stored to a temp table
Display the result set from the temp table

그림 3. X2RD 이벤트 흐름
Fig. 3. Event flow in X2RD

표 3. 시스템 환경
Table 3. System Environment

CPU	1.6GHz
Memory	1 GB
OS	Linux Redhat
RDBMS	PostgreSQL 8.2

3. Experimental Results

We test X2RD with XML documents with various size and complexity. All experiments were run on 1.6GHz Intel Centrino duo processor with 1GB memory running a Linux Redhat operating system. System environments are summrized in table 3. We used PostgreSQL version 8.2 for our experiment. One of test XML data, students.xml is shown in figure 4. It is mapped into a single table as shown in figure 5 by using the mapping rule we've proposed in section 3.1. For mapping, we used our proposed mapping function, which 'shreds' XML documents (i.e. student.xml). The test cases are from level 1, which has one location step expression to level 3. The test queries are shown in table 4. The X2RD query processor visits each location step and sends SQL queries to the RDBMS to get all the right set of node to axes and node test from the table.

표 4. 테스트 쿼리
Table 4. Tested Queries

Level 1	C:* D:* D:first C:student
Level 2	C:student/C:* C:student/D:John D:name/P:*
Level 3	C:student/D:CrsCode/C:CS308 C:student/D:CrsCode/P:*
Level 4	C:student/D:CrsCode/P:*/C:Semester

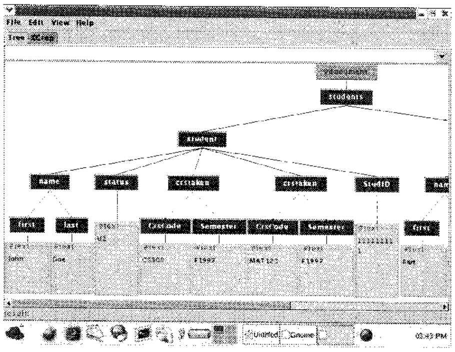
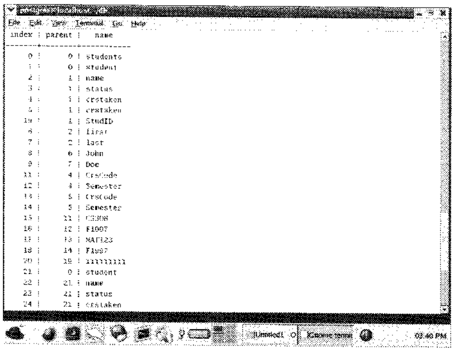
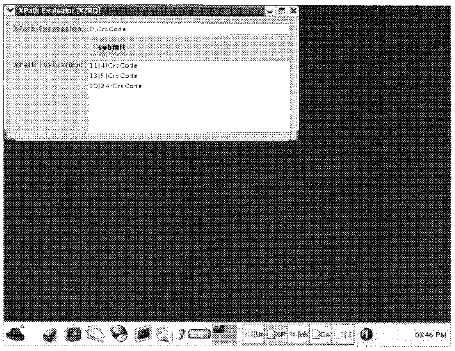


그림 4. 테스트용 students.xml with xviewer
Fig. 4. test XML (students.xml) with xviewer



```
index | parent |   name
-------+--------+---------
     0 |      0 | students
     1 |      0 | student
     2 |      1 | name
     3 |      1 | status
     4 |      1 | crstaken
     5 |      1 | crstaken
    10 |      1 | StudID
     6 |      2 | first
     7 |      2 | last
     8 |      6 | John
     9 |      7 | Doe
    11 |      4 | CrsCode
    12 |      4 | Semester
    13 |      5 | CrsCode
    14 |      5 | Semester
    15 |     11 | CS308
    16 |     12 | F1997
    17 |     13 | MAT123
    18 |     14 | F1997
    20 |     10 | 111111111
    21 |      0 | student
    22 |     21 | name
    23 |     21 | status
    24 |     21 | crstaken
```

그림 5. 매핑된 테스트 XML
Fig. 5. Mapped XML in a relational table



XPath Expression: D:CrsCode

XPath Evaluation:
11|4|CrsCode
13|5|CrsCode
30|24|CrsCode

그림 6. X2RD의 사용자 인터페이스
Fig. 6. User Interface of X2RD

V. CONCLUSION

Storing and querying XML documents in a relational database become increasingly important because XML is everywhere in these days. As Web service technology and Semantic Web get popular, more and more XML data will be around on the Web and there are many efforts from research institutes and venders (including IBM, MS SQL, ORACLE, and MySQL) who put efforts on supporting XML data.

This paper has addressed some key issues to store and query XML data in a relational database including shredding XML to relations, providing mapping functions, and implementing a query processor to translate XPath expression into SQL query on the relational database.

In this paper, our primary goal is to propose an alternative option to store and query XML and we show that XML documents can be efficiently supported by a relational database. Our scheme is simple; however it supports most of issues in storing/querying XML documents in RDBMS except reconstructing and guaranteeing an identical XML document. Supporting a full set of XPath syntax is also a work will be done in the future.

In order to verify the potential of our proposed scheme, we implement it using general RDMBS (i.e. PostgreSQL) and our implementation, X2RD successfully performs XPath axes operation and others on XML documents stored in the RDBMS. Empirical results show that our scheme can be efficiently implemented.

REFERENCE

[1] 박종현, 강지훈, "디지털 방송을 위한 Set-Top Box 기반 TV-Anytime 메타데이터 관리," 한국컴퓨터정보학회논문지, 제13권 제4호, 2008년 7월

[2] 안후영, 박영호, "관계형 데이터베이스 기반 온톨로지 기법을 활용한 모바일 패션 정보 제안," 한국컴퓨터정보학회논문지, 제12권, 제6호, 2007년 12월

[3] The Apache XML Project, Apache Xindice, *http://xml.apache.org/xindice/*

[4] The Ozone Database Project, ozone, *http://www.ozone-db.org/frames/home/what.html*

[5] The World Wide Web Consortium (W3C), "*XML Path Language (XPath) version 1.0*," W3C Recommendation, Nov. 1999.

[6] D. Florescu and D. Kossmann, "Storing and Querying XML Data using an RDBMS," IEEE Data Engineering Bulletin, Vol. 22, No. 1, pp. 27-34, Mar. 1999.

[7] A. Deutsch, M. Fernandez, and D. Suciu, "Storing semistructured data with STORED," ACM SIGMOD International Conference on Management of Data, pp. 431-442, Philadelphia, Pennsylvania, USA, June 1999.

[8] I. Tatarinov et al., "Storing and Querying Ordered XML Using a Relational Database System," ACM SIGMOD International Conference on Management of Data, Madison, pp. 204-215, Wisconsin, USA, June 2002.

[9] M. Rys, "XML and relational database management system: inside Microsoft® SQL ServerTM 2005," ACM SIGMOD International Conference on Management of Data, pp. 958-962, Baltimore, Maryland, USA, June 2005.

[10] Oracle Corporation, ORACLE 8i Database, *http://tahiti.oracle.com/pls/tahiti/tahiti.homepage.*

[11] Oracle Corporation, ORACLE Database 11g, *http://www.oracle.com/technology/products/database/oracle11g/index.html.*

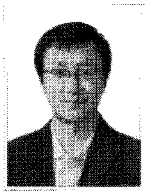
[12] SQLX Group, SQLX, *http://sqlx.org/*

[13] XML Query Working Group of The W3C, "*XQuery 1.0: An XML Query Language*," W3C Recommendation, Jan. 2007.

[14] PostgreSQL Global Development Group, PostgreSQL, *http://www.postgresql.org*

[15] 안동찬, 변찬우, "안전한 XML 접근제어에서 효율적인 질의 재작성 기법," 한국컴퓨터정보학회논문지 제11권 제5호, 2006년 11월.

저 자 소 개



오 상 윤 (Sangyoon Oh)

2006년 미 인디애나대 Computer Science 박사

2006~ 2007년 SK 텔레콤

2008~ 현재 아주대학교 정보통신대학 정보및컴퓨터공학과 조교수

관심분야: 웹/분산 시스템, SOA, Ad-hoc/P2P 시스템