A Middleware Approach to Storing and Querying XML Documents in Relational Databases

Zülfü Şevkili, Mine Mercan, Atakan Kurt

Dept. of Computer Eng.
Fatih University
Istanbul, Turkey
{zsevkili, minemercan, akurt}@fatih.edu.tr

Abstract. In this paper we present a middleware for storing and retrieving XML documents in relational databases. To store XML documents in RDBMS, several mapping approaches can be used. We chose structure independent approach. This approach stores XML documents in fixed-schema tables and does not require a direct extension of SQL. So the middleware can be used with any RDBMS with minor changes in the interface. The middleware offers two alternative methods—namely XRel and Edge—for storing XML in the database. The Edge method is a straightforward method, while XRel utilizes path summary information for faster query processing. We present a comparative experimental study on the performance of insertion and retrieval of two types of XML documents along with a set of XPath queries executed through the XPath query processor which is a part of the middleware.

1 Introduction

XML—eXtensible Markup Language—[3] emerged as a major standard for representing data on the World Wide Web. Despite the excitement XML has produced, there are important problems to tackle with regarding the management of large amounts of XML documents efficiently in a database. Recently native-XML databases such as Tamino have emerged in the market; however native-XML databases usually have limited support for relational data.

Taking up emerging requirements, database vendors such as IBM, Oracle and Microsoft are enabling their products for XML. XML-Enabled databases have mature and proven techniques for relational data processing but XML-extensions have not been mature enough yet. Since there are many relational databases systems on the market, a middleware approach [1] can be used until most relational systems have XML-support. The key to middleware approach is to store XML documents in a relational database through a user interface and with an XPath query processor.

There are various approaches for effective, automatic conversion and storage of XML documents into relational databases [7], [9], [11], [17], [15], [20], [22]. Generally, we can classify all studies on XML storage in two categories as in [10]. The first one is integral storage, which normally keeps the unparsed document intact and stores it as a BLOB in a database or a file in the file system. With this approach,
access to individual elements is realized by parsing, and therefore it is inefficient. This approach is appropriate for retrieving documents, but requires specific index structures for query processing. The second one is dispersed storage; in which the documents are shredded and shredded items are stored in tables. When XML documents are stored via dispersed storage, a problem occurs about schema design. The problem of a storage model design becomes a database schema design problem. In [11], the authors categorize such database schemas into two categories: structure dependent mapping approach and structure-independent-mapping approach. In the former, the design of database schema is based on the understanding of DTD (Document Type Descriptor). In the latter, a fixed database schema is used to store all XML documents regardless of DTDs.

Vendors such as IBM, Oracle, and Microsoft either use structure dependent mapping approach or store unparsed documents in a CLOB field. For details about XML support in DB2, Oracle and SQL Server, please refer to [4, 7, and 15].

In this study, we present a middleware that takes advantage of the structure independent mapping approach. We adapted PhpMyAdmin 2.4.0 web base interface for storing and retrieving XML documents in MySQL open source database. With this implementation, users can store XML documents regardless of structure of documents. The application gives users two alternatives (XRel and Edge) for database schema design. We compared with storage and retrieval time of a set of XML documents. We also present an experiment with a set of XPath queries. The results seem to be satisfactory in most cases, very good in some cases, and acceptable in other cases.

The remainder of this paper is organized as follows. Section 2 presents structure independent mapping approaches. We describe the middleware in Section 3. Experimental results are discussed in Section 4. The conclusions are drawn in Section 5.

2 Structure Independent Mapping Approach

In this section, the structure independent mapping approach is explained with a sample XML document shown in Figure 1. In this paper, we employ the data model of XPath [5] to represent XML documents. In the XPath data model, XML documents are modeled as an ordered and directed labeled tree. There are seven types of nodes. In this paper, we consider only the following four types of nodes for the sake of simplicity: root, element, text and attribute. The root node is a virtual node pointing to the root element of an XML document. Elements in an XML document are represented as an element node. Element nodes can have other elements or text as its children. Text nodes are string-valued leaf-nodes. An element node can have a set of attribute nodes. An attribute node has an attribute-name and an attribute-value. Figure 2 shows an XML tree such that a left-to-right and depth-first traversal describes the order of the XML content within our document in Figure 1.
2.1 The Edge Method

The Edge method stores all edges of an XML tree in a single-table named Edge [9]. The table schema is given below:

```
Edge(Source, Ordinal, Target, Name, Flag, Value)
```

An edge is specified by two node identifiers, namely Source and Target. The Name attribute stores the element or attribute name. The Ordinal attribute records the ordinal of the node among its siblings. A Flag value indicates whether the target node is an inter-object reference (ref) or points to a value (val). Table 1 shows the Edge table for the XML document in Figure 2. The tuple (5, 2, 8, given, val, "Paul"), means the element with name "given" and value "Paul". The parent node of this element is 5 and its ordinal is 2. This approach is quite simple. It stores parent-child relationship. It may be required during retrieval of the original document from the records stored in the table if the tree is not shallow. The same can be said for processing long XPath queries on data stored with this method.

Fig. 1. Sample XML

As a variation of the Edge approach, the XML tree can be stored in multiple tables in [16]. That method partitions the Edge table according to all possible label-paths. For each unique path, the method creates a table. This method is also a structure independent approach, but the number of tables is fixed. So it is difficult to support database schema for dynamic XML documents.

Fig. 2. The XML Tree the sample XML

<table>
<thead>
<tr>
<th>Table 1. Edge Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
### 2.2 The XRel Method

This approach [22] stores XML documents in four different tables: **Element** table stores only the document structure. **Text** table holds only text data. **Attribute** table stores attribute values. **Path** table keeps unique paths in XML documents. The key to this method is the path table and regions associated with the inner nodes in the tree.

- **Element** (DocID, PathID, Start, End, Index)
- **Attribute** (DocID, PathID, Start, End, Value)
- **Text** (DocID, PathID, Start, End, Value)
- **Path** (PathID, PathExp)

Each node is associated with start and end positions. A region (or the pair of start and end positions) implies a containment between elements with regards to the ancestor-descendant and parent-child relationships. For example, a node, \( n_i \), is reachable from another node \( n_j \) if the region of \( n_j \) is included in the region of \( n_i \).

In our study, we modified some attributes of tables in XRel approach: we added a **ParentID** column to Element, Attribute, and Text tables to find parent nodes easily. We put NodeID and last descendant node id (EndDescID) attributes instead of start and end columns in Element table. The XRel tables can be seen in Table 2 for the XML document given in Figure 1:

<table>
<thead>
<tr>
<th>PathID</th>
<th>PathExp</th>
<th>NodeID</th>
<th>EndDescID</th>
<th>PathID</th>
<th>ParentID</th>
<th>Ordinal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/address_book</td>
<td>1</td>
<td>51</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>/address_book/</td>
<td>2</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>/address_book/card</td>
<td>3</td>
<td>11</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>/address_book/card/name</td>
<td>4</td>
<td>11</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>/address_book/card/name/surname</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>/address_book/card/name/given</td>
<td>6</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>/address_book/card/name/other</td>
<td>7</td>
<td>11</td>
<td>7</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>/address_book/card/title</td>
<td>8</td>
<td>13</td>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>/address_book/card/address</td>
<td>9</td>
<td>13</td>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>/address_book/card/address/street</td>
<td>10</td>
<td>18</td>
<td>11</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>/address_book/card/address/city</td>
<td>11</td>
<td>20</td>
<td>12</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>/address_book/card/address/state</td>
<td>12</td>
<td>22</td>
<td>13</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>/address_book/card/address/zip</td>
<td>13</td>
<td>25</td>
<td>14</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>/address_book/card/contact</td>
<td>14</td>
<td>25</td>
<td>15</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>/address_book/card/contact/phone</td>
<td>15</td>
<td>25</td>
<td>15</td>
<td>23</td>
<td>1</td>
</tr>
</tbody>
</table>

(a) Path Table

(b) Element Table

---

<table>
<thead>
<tr>
<th>Title</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>Prof.Dr</td>
</tr>
<tr>
<td>address</td>
<td>20th floor 300 Lakeside</td>
</tr>
<tr>
<td>city</td>
<td>Oakland</td>
</tr>
<tr>
<td>state</td>
<td>CA</td>
</tr>
<tr>
<td>zip</td>
<td>98520</td>
</tr>
<tr>
<td>phone</td>
<td>510 628 39 93</td>
</tr>
</tbody>
</table>
3 Implementation: Middleware for XML Support on MySQL

We used MySQL as the DBMS for storing and retrieving XML documents using structure independent mapping approaches. To realize this, we develop a middleware and embed it into PhpMyAdmin program which is a web based interface between MySQL and users.

In Section 2, we discussed three different methods that we can use in structure independent mapping approach. There are variations of these methods as well. We chose the Edge and XRel methods, because Edge is a simple method to implement, and XRel is more complex with the path summary for faster XPath query processing. So our initial expectation was that XRel should be several times faster than Edge. However the results are somewhat unexpected. In some cases Edge seems to do better, in some other cases it displays performance comparable to XRel.

Figure 3 outlines the architecture of the middleware which adds XML support to the MySQL database system. The three main classes (Collection, Document, XPath) can be used to adapt the same interface to other databases systems.
Our implementation adds collection support to XML into the MySQL database. A collection is a set of similar XML documents stored in fixed schema of tables, sometimes referred to as XML repository. In reality, MySQL database model does not change, but from the user point of view, a database can contain tables and collections. Fig. 4 shows a screenshot during collection creation. Users can not modify or access the tables behind collections directly. Users can manipulate XML data through collections only. A collection can store document using either Edge or XRel method. Users can create, drop or browse collections, insert, delete, or browse XML documents in a collection.

![Database test running on localhost](image)

**Fig. 4.** Creating collection in PhpMyAdmin

4 Experimental Result

This section discusses the experimental results of storage and retrieval time of XML documents and a set of XPath queries using XRel and Edge methods. All the experiments were conducted on a Pentium IV 350 MHz PC with 1 GB RAM 300GB hard disk. We used Windows XP, MySQL 4.0.13, Php 4, Apache 1.3.29, and PhpMyAdmin 2.4.0 for the experiments.

Generally, we can classify XML documents in two categories as in [21]: data-centric and text-centric. Text-centric documents have more irregular structure. They can contain long and variable-length text and the order of siblings is always important. Data-centric documents are usually used for XML-encoding relational data. We used Catalog.xml [21] as a sample of data-centric documents and Shakespeare’s plays [2] documents as sample of text-centric documents. Although Shakespeare’s plays are well-structured, they contain variable-length strings like dictionary file in [21]. So they can be considered text-centric documents. Tables were indexed on their key fields and the fields used in join. Insertion and retrieval results are given below, for details please refer to [19].
4.1 XML Insertion

The insertion time of both types of document using EDGE and XREL methods are given in Figure 5. As seen in the figure, the Edge method is at least twice as fast as the XRel method. The reason could be that the data is stored in one table only in the Edge method, while the XRel method has to store path summary and end_descendant_id which requires more complex processing.

4.2 XML Reconstruction

The retrieval times are given in Figure 6. The results are close. Edge seems to be a little faster than XRel. The retrieval time is roughly proportional to file size. We think that the results are affected by the number of inner node as well. The XRel method needs to join 3 different tables, while Edge table joins with itself.

![Fig. 5. Insertion time (mm:ss) of docs](image1)

![Fig. 6. Reconstruction time (mm:ss) of docs](image2)

4.3 Query Results

A set of 13 queries executed on Catalog and Shakespeare documents. Below we only present 6 queries on Shakespeare documents and skip queries on catalog documents because of space limitations. Please refer to [14] for a full presentation. The results are shown in Figure 7 for both methods. The XPath queries were automatically converted to SQL by the query processor. The execution times do not include sorting nodes according to document order, and conversion to text with XML tags. Time is given in milliseconds.

Query a is a long XPath query. Query b represents a short XPath query. Query c is a (/) descendant-or-self query. Query d has a wild card (*) which represents any single element. Query e and f contains equality and range predicates on text-values.
Fig. 5. Retrieval time (milliseconds) of XML documents.

Query a: Query a requires one join in each step (5 steps total) in the XPath query using the Edge method, however the XRel uses the path information in the Path table, as a result XRel is faster in the long XPath query as expected. The SQL statements for Edge and XRel methods respectively generated by the query processor are given below:
A Middleware Approach to Storing and Querying XML Documents in Relational Databases

SELECT e5.* FROM edgecol_edge as e1,edgecol_edge as e2,edgecol_edge as e3,edgecol_edge as e4,edgecol_edge as e5 WHERE e1.name=’plays’ AND e1.parentID=0 AND e1.docID=3 AND e1oggled=’att’ AND e2.name=’play’ AND e2.parentID=e1.nodeID AND e2.oggled=’att’ AND e2.docID=3 AND e3.name=’act’ AND e3.parentID=e2.nodeID AND e3.oggled=’att’ AND e3.docID=3 AND e4.name=’scene’ AND e4.parentID=e3.nodeID AND e4.oggled=’att’ AND e4.docID=3 AND e5.name=’title’ AND e5.parentID=e4.nodeID AND e5.oggled=’att’ AND e5.docID=3

Query b: Query b is a short XPath query. So there aren’t many joins in the SQL statement for the Edge method. The response times are very close as expected.

Query c and d: Query c is a ‘//’ descendant-or-self query, while Query d is a ‘*/’ wild card query. Although XRel seems to be slightly faster (%10) in the first query, this difference is trivial. Since both queries are short XPath queries, the execution times are comparable.

Query e and f: Both queries contain predicates on text values. The objective is to see how fast the query processing is with text not with structure as with previous queries. Edge seems to do better than XRel on these queries.

We think that the execution times for insertion, retrieval and queries processing are adequate and comparable to commercial applications and databases. Experiments conducted on commercial databases support this observation [12], [13].

5 Conclusion

In this work we studied a middleware approach to storing XML documents in relational databases. There are 2 main approaches to support XML in databases (i) Native XML databases (ii) XML-enabled databases. Since there are several relational databases on the market, a middleware approach can be used as an affordable and quick solution until XML data processing matures. The key to middleware approach is storing XML documents in a relational database, providing a user interface for XML manipulation, and adding an XPath query processor for XML querying.

The middleware implemented in this study can be used with any other database management system as it doesn’t require any modification to the DBMS itself. It provides collections or XML repositories to store XML documents in a database.

The experiments yield promising results. Overall the performance of Edge and XRel methods is comparable in most cases with the exception of long XPath queries where XRel is definitely faster. We can say that the Edge method can certainly be considered for query processing in most cases. We think that the execution times for insertion, retrieval and queries processing are adequate and comparable to commercial applications and databases.
References


