JOURNAL OF KOREA MULTIMEDIA SOCIETY, VOL. 10, NO. 12, DECEMBER 2007(pp. 1539-1550)

# XML Type vs Inlined Shredding into Tables for Storing XML Documents in RDBMS

Min Jin[†], Minjun Seo[††]

## ABSTRACT

As XML is increasingly used for representing and exchanging data, relational database systems have been trying extend their features to handle XML documents. XML documents can be stored in a column with XML data type like primitive types. The shredding method, which is one of the traditional methods for storing and managing XML documents in RDBMS, is still useful and viable although it has some drawbacks due to the structural discrepancy between XML and relational databases. This method may be suitable for data-centric XML documents with simple schema. This paper presents the extended version of the *Association inlining* method that is based on inlined shredding and compares the performance of querying processing to that of XML type method of conventional relational database systems. The experiments showed that in most cases our method resulted in better performance than the other method based on XML data type. This is due to the fact that our shredding method keeps and uses the order and path information of XML documents. The *path* table has the information of the corresponding table and column for each distinct path and the structure information of the XML document is extracted and stored in data tables.

**Keywords:** Inlined Shredding, Extended Association Inlining, XML Type, XQuery

## 1. NTRODUCTION

XML was originally developed as a simplified form of SGML, a mark up language with simple syntax and extendible vocabulary. It has become a de facto standard for representing and exchanging data on the Internet and in business applications. In recent years, several approaches have been developed for managing XML data. These can be classified into two groups: native XML data management systems such as Timber, Niagara, and Natix, and conventional relational database systems[1]. The latter group is within the scope of this paper. There are three ways for storing XML data in relational database systems. First, XML data is stored as a CLOB(character large object) supporting text fidelity[1]. The original structure of the XML document is maintained in this scheme. It allows fast insertion and retrieval of full XML documents, but suffers from poor search of the documents and retrieval of partial documents due to the fact that parsing is required at every query execution.

Second, XML documents are shredded into relational tables supporting relational fidelity. The hierarchical structure of the document is broken into flat tables. The most significant disadvantage of shredding is that the characteristics of the original XML document such as hierarchy and order can be lost in the process of shredding due to the structural discrepancy between XML and relational database. In fact, there is no published algorithm

※ Corresponding Author : Min Jin, Address : (631-701) 449 Wolyoung-dong, Masan, Kyungnam, S. Korea, TEL : +82-55-249-2653, FAX : +82-55-248-2554,
E-mail : mjin@kyungnam.ac.kr
Receipt date : May. 31, 2007, Approval date : Oct. 10, 2007
[†] Div. of Computer Science and Engineering, Kyungnam University
[††] Div. of Computer Science and Engineering, Kyungnam University
(E-mail : mjseo@kyungnam.ac.kr)
※ This work was supported by Kyungnam University Foundation Grant, 2006.

for translating simple path expression queries into SQL when the XML schema is recursive. It becomes worse when the XML schema is very large and complex, and tables may have many null values when the XML document is sparse. The second disadvantage is that the insertion of XML documents takes a long time when shredding XML documents into relational tables due to the costly parsing that includes multitable inserts. The third disadvantage is the difficulty of reconstruction of the original full XML document without special facilities. It is also difficult to extract subparts of the document in some cases. The fourth disadvantage is that it is difficult to support schema evolution since data are scattered among tables.

The shredding method has some advantages. The first advantage is that modifications to the conventional relational engine are not required and the existing features of the engine can be used. No extensions to existing facilities are needed for managing XML documents. Once XML documents are broken into flat tables, conventional SQL can be used in querying the data. The second one is that some queries could be supported conveniently by using relational tables. Relational database users feel comfortable with this method since they are accustomed to using flat tables and SQL. The third one is that most application tools such as data mining and business intelligence are developed based on relational tables. These programs can be applied to this method without reprogramming.

Third, a native data type is defined for XML documents in the relational database supporting XML fidelity. They are stored in a column with XML data type and treated the same as primitive data types such as integer, char, and decimal. The SQL-2003 standard provides a new data type called XML for storing XML documents and fragments. SQL/XML is an extension of SQL that is part of SQL 2003[2]. Recently relational database vendors such as IBM, Oracle, and Microsoft have provided extended facilities for interacting and managing XML documents[1,3-8]. These database systems store XML documents in XML data types. An index can be created to expedite query processing. SQL Server 2005 provides a mechanism for indexing the XML data[6,9,10]. It contains the structural information including the hierarchical relationships among XML nodes and the document order. The size of the index necessary for providing smooth access to the data is several times that of the original XML document. Relational database vendors continue to extend their facilities to support two standards for managing XML documents: SQL/XML and XQuery. SQL/XML is SQL-centric while XQuery is XML-centric.

In sum, the shredding method is still useful and viable although it has some drawbacks compared to the XML data type based approach. This method might be good for data-centric XML documents with simple schema. Hence, this paper presents a comparison of query processing between the shredding method and the XML data type based method in conventional database systems. The experiment is conducted on possible patterns of queries. The shredding method used in this paper is the extended version of the association inlining, which combines the join reduction properties of hybrid inlining and sharing features of shared inlining[11]. It leads to the reduction of both relational fragments and excessive joins compared to both inlining methods.

The rest of this paper is organized as follows. Section 2 reviews related work concerning the storage of XML documents in relational databases. Section 3 describes the extended association inlining method. Section 4 shows the result of experiments on possible query patterns. Section 5 offers conclusions.

## 2. RELATED WORK

There are two scenarios in XML and relational

database systems; XML publishing and XML storage. The former aims to publish data stored in RDBMS in XML format, treating the data as XML documents and the latter aims to store XML documents in RDBMS[11-17]. There are three ways of storing XML documents in RDBMS; text fidelity, relational fidelity, and XML fidelity.

Shredding, which provides relational fidelity, can be classified into two approaches: the Model mapping approach and the Structure mapping approach[18]. The Model mapping approach deals with the storage of XML documents without structural information, such as DTD and XML schema. Relational schemas are defined regardless of the structural information of the given XML document. In contrast to the Model mapping approach, the Structure mapping approach deals with storing XML documents with structural information such as a DTD or an XML schema. The relational tables are generated based on the structural information extracted from the DTD or the XML schema. There are three methods for this, called basic inlining, shared inlining, and hybrid inlining[11]. The association inlining method combines the join reduction properties of hybrid inlining with the sharing features of shared inlining[19]. The most important characteristic of the association inlining method is the path table. The path table, which contains all possible paths from the root to each node, is constructed and used in both populating the data into the tables and processing queries. The experiment showed that association inlining had 92.21% lower joins than that of shared inlining. The number of subqueries per query is 88.05% lower than that of hybrid inlining. These reduction rates depend on the characteristics of the document.

As XML is increasingly being used in data processing environments, relational database systems such as DB2, Oracle, and SQL Server have been trying to extend their features to handle XML documents[20-22]. The SQL-2003 standard provides a new data type called XML for storing well-formed XML documents and fragments based on the XML Infoset and XQuery data model. Columns, variables and parameters can have XML types like primitive types.

SQL/XML and XQuery are two standards that use declarative queries to access data to be returned in XML format. SQL/XML extensions allow publishing relational data in XML format at an arbitrary level of XML documents, by providing functions that operate like XPath and XQuery, inside SQL statements[2]. This is being supported by Oracle and IBM. SQL Server provides vendor-specific methods for this purpose[3].

SQL Server 2005 stores XML documents in internal binary format as BLOBs. A primary index can be created to avoid parsing the XML BLOBs at each query time. And additional secondary indexes can be created for improving processing performance of path-based queries, property-based queries, and value-based queries[9,10,20]. SQL Server 2005 provides query and modification capabilities using query methods on the XML data type that accept the XQuery[10]. It also provides XML schema collection as a mechanism for managing XML schema documents as metadata.

DB2 introduces a native XML storage format to avoid parsing at every query time. It stores XML documents as instances of the XQuery data model, in a structured type-annotated tree. Unlike SQL server 2005, XML parsing is not required at query time. Indexes can be defined on specific paths. As every node of an XML document has type information, it could easily support schema evolution[1,8].

Oracle treats the XML data type as a logical abstraction over a variety of physical storage forms; CLOB, hybrid, shredding, and binary XML format[4,22]. In binary XML type representation, the data is encoded in its native typed format. The XML tree can be broken into disjoint fragments in a native fashion via the notion of section

references. The XML data represented in binary format can be stored in one or more tables with BLOB columns. This is different from shredding, in that the table schema is decoupled from XML schema. Hence, there are many-to-one mappings of XML tree fragments to binary XML tables.

## 3. EXTENDED ASSOCIATION INLINING

### 3.1 A Path Table and Data Tables

We use an extended version of Association inlining as a shredding method. The main characteristics of the method are as follows.

First, the structural information of the XML document such as the order of the nodes and hierarchical relationship among nodes is represented in the table. In the stage of shredding XML documents into relational tables, the order of nodes and the hierarchical relationships among nodes are revealed. This information is captured and kept in the data tables and reflects the fidelity of the XML document structure. Second, the path table is also extended. It contains the type of the rightmost node of the path from the root to the node. Third, the data tables and path tables are more tightly integrated in the population of dada and query processing. The example XML document is shown in Fig. 1 and the order of nodes is shown in Fig. 2.

```
<BOOK ISBN="1-55860-438-3">
<SECTION>
<TITLE>Bad Bugs</TITLE>
        Nobody loves bad bugs.
<FIGURE CAPTION="Sample bug"/>
</SECTION>
<SECTION>
<TITLE>Tree Frogs</TITLE>
All right-thinking people
<BOLD>love</BOLD>tree frogs.
</SECTION>
</BOOK>
```

Fig. 1. XML Data.

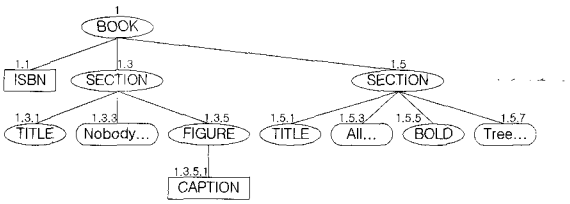


Fig. 2. ORDPATH Node Label used in[9].

Fig. 3 shows the path table, which contains paths from every node to the root. This information is mostly based on the XML schema information. However, for recursive cases, it also depends on XML document occurrences, since there is no published mechanism for translating XML to SQL. The delimiter '#' is added to use the Like clause of SQL. Table and column indicate the relational table and column that correspond to the last node on the path expression from the root to the node.

| pathID | pathExp | nodeType | table | column | parentCode | lastnode |
|---|---|---|---|---|---|---|
| 1 | #/BOOK | element | book | NULL | NULL | BOOK |
| 2 | #/BOOK#/@ISBN | attribute | book | ISBN | NULL | ISBN |
| 3 | #/BOOK#/SECTION | element | section | NULL | book | SECTION |
| 4 | #/BOOK#/SECTION#/ | text | section.text | text | section | SECTION |
| 5 | #/BOOK#/SECTION#/TITLE | element | section | title | book | TITLE |
| 6 | #/BOOK#/SECTION#/FIGURE | element | NULL | NULL | NULL | FIGURE |
| 7 | #/BOOK#/SECTION#/FIGURE#/@CAPTION | attribute | section | caption | book | CAPTION |
| 8 | #/BOOK#/SECTION#/BOLD | element | section | bold | book | BOLD |

Fig. 3. Path table.

**Book**

| ID | docID | parentID | parentCode | ISBN | elementOrd | order | nested | pathID |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | NULL | NULL | 1-55860-438-3 | 1.1 | 1 | NULL | 1 |

**Section**

| ID | docID | parentID | parentCode | title | caption | bold | elementOrd | order | nested | pathID |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | book | Bad Bugs | Sample bug | NULL | 1.3 | 1 | NULL | 3 |
| 2 | 1 | 1 | book | Tree Frogs | NULL | love | 1.5 | 2 | NULL | 3 |

**Section.text**

| ID | docID | parentID | parentCode | text | elementOrd | order | nested | pathID |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | section | Nobody loves bad bugs. | 1.3 | 1 | NULL | 4 |
| 2 | 1 | 2 | section | All right-thinking people | 1.5 | 1 | NULL | 4 |
| 3 | 1 | 2 | section | tree frogs. | 1.5 | 2 | NULL | 4 |

Fig. 4. Tables for storing the XML document in Fig. 1.

The *nodeType* represents the node type of the last node of the path from the root. *ParentCode* indicates the parent table in the hierarchy of relational tables for accommodating the hierarchical relationships between nodes. *Lastnode* indicates the last node on the path expression from the root to the node. It is indexed to expedite getting table information for path expressions. When the table and column columns are both not NULL, the path is final; there are no more nodes to move along the path.

The data tables are shown in Fig. 4. *Book* and *Section* elements are represented as separate tables. The text for the *section* element is represented as a separate table, under the name *Section.text*. Some elements such as *title* and *bold* are represented as columns of a table. The *figure* element appears as neither a table nor a column. Each column except the meta-columns for the structural information has the corresponding order number represented in the Dewey Order Encoding method[16,23]. The order information of a node is represented in the *elementOrd* column of the table. The order column indicates the occurrence order of the same element, attribute, or text within an element. The hierarchical relationship among nodes is represented via *parentCode* and *parentID*. The *parentCode* column indicates the parent table of the row and *parentID* indicates the parent row in the parent table. The nested column indicates whether or not the row is recursively created. The *pathID* column represents the path which causes a row to be generated in the table. This value helps disambiguate between rows that are created in the same table, but their paths may be different. This value is brought from the *pathID* column of the path table. Here, it is represented as a concatenation of numbers which mean the corresponding nodes separated by '#' in reverse order.

Some nodes such as element-only elements and elements with inlinable sub-nodes don't appear in the data table. Hence the corresponding order may be missing. In order to keep the complete order of an XML document, the missing value should be collected and stored. Fig. 5 shows the missing order of the XML document.

| docID | path | pathID | order |
|---|---|---|---|
| 1 | #/book#/section#/figure | 6 | 1.3.5 |

Fig. 5. Missing order of the XML document in Fig. 2.

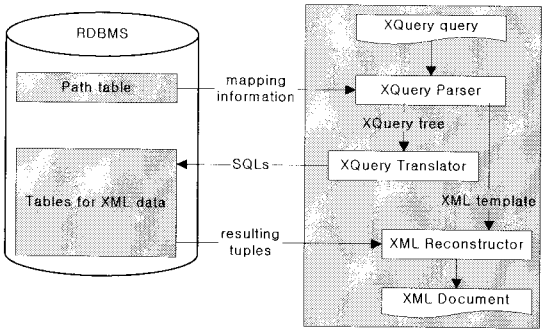


Fig. 6. Architecture for processing of XQuery query.

### 3.2 Translating XQuery into SQL

The overview of our system for translating XQuery into SQL and reconstructing XML documents in XML format is shown in Fig. 6.

An XQuery query such as the one in Fig. 7 is parsed and decomposed into an XQuery Tree, which is translated into SQL and an XML template.

The mapping information in the Path table is exploited in creating the XQuery tree and the XML template. The outline of the algorithm for getting the mapping information from the Path table is described in Fig. 8.

```
For $b in /BOOK/[@ISBN="1-55860-438-3"]
Where $b/SECTION/TITLE/text()="Bad Bugs"
Return
  <BOOK>
    <SECTION>$b/SECTION/text()</SECTION>
  </BOOK>
```

Fig 7. An XQuery query.

The XQuery translator takes an XML query tree as input and translates it into SQL. The outline of the algorithm for translating XQuery queries into SQL is described in Figure 9. Figure 10 shows the SQL query translated from the XQuery query of Fig. 7.

## 4. EXPERIMENTS

We executed some typical types of queries on the XMark data set[24] by using both the shredding method and the XML type in SQL Server, DB2, and Oracle. The size of the original XML document used is 115,775KB for scale factor 1.0. We used the extended association inlining as a shredding method. The number of generated tables

```
GetTable(exp, table, column, pathID) {
// Input: exp, which is a fragment of path expressions
// Output: table, column, pathID
select table, column, pathID
from Path
where pathExp = exp
if (table is null and column is null ) {
  // When the element is not mapped to a table, it is an element-only element
    select pathExp
    from Path
    where pathExp = exp + '#%'
    for each exp
      GetTable(exp, table, column, pathID)
  }
}
```

Fig. 8. The the algorithm for getting the mapping information from the Path table.

```
Algorithm tranlating XQuery into SQL
Input:
N: Nodes of XQuery tree
Output: SQL query sql
Begin
   Let N be the root node in XQuery Tree.
   parentTables=""
   SelectWhere="": FromClause="": WhereClause="": OrderByClause=""
   XQuerytoSQL( N, parentTables )
   sql="SELECT "+SelectClause+" FROM "+FromClause+" WHERE "+WhereClause
   Return sql
End
Procedure XQuerytoSQL(Node of XQuery Tree N, parentTables)
Begin
   If N_type is "For" or "Let" then
      If N is root node then
         FromClause=FromClause∨N_table
         WhereClause=WhereClause∨N_table+".pathID"+" IN ("+N_pathID+")"
         parentTables=parentTables∨N_table
      End If
   End If
   If N_type is "Predicate" or "Where" then
      WhereClause=WhereClause∨N_table.N_column N_perdicate
      If N_table isn't the element of parenTables then
         If N_parentTable is the element of parentTables then
            FromClause=FromClause∨N_table
            WhereClause=WhereClause∨N_parentTable+".ID="+N_table+".parentID"
            WhereClause=WhereClause∨N_table+".pathID"+" IN ("+N_pathID+")"
            parentTables=parentTables∨N_table
         Else
            FromClause="(SELECT elementOrd,)+SelectClause+" FROM "+FromClause+" WHERE "+WhereClause+")
                        "+Alias","+N_table
            WhereClause=N_table+".elementOrd BETWEEN "+Alias+".elementOrd AND Descendants("+Alias+".elementOrd)"
            parentTables=parentTables∨N_table
         End If
      End If
   End If
   If N_type is "OrderBy" then
      OrderByClause=OrderByClause∨"N_table.N_column"
   End If
   If N_type="Return" then
      If N_table is the element of parenTables then
         SelectClause=SelectClause∧N_table.N_column
      Else If N_parentTable is the element of parentTables then
         FromClause=FromClause∨N_table
         WhereClause=WhereClause∨N_parentTable+".ID="+N_table+".parentID"
         WhereClause=WhereClause∨N_table+".pathID"+" IN ("+N_pathID+")"
         parentTables=parentTables∨N_table
      Else
         FromClause="(SELECT elementOrd,)+SelectClause+" FROM "+FromClause+" WHERE "+WhereClause+") "+Alias","+N_table
         WhereClause=N_table+".elementOrd BETWEEN "+Alias+".elementOrd AND Descendants("+Alias+".elementOrd)"
         parentTables=parentTables∨N_table
      End If
   End If
   For each child node of N do
      XQuerytoSQL( N, parentTable )
   End For
End
```

Fig 9. The outline of the algorithm for translating XQuery queries into SQL.

```
SELECT section.text.text
FROM book, section, section.text
WHERE book.id=section.parentid AND book.pathid=1 AND section.id=section.parentid AND section.pathid=3
    AND book.ISBN="1-55860-438-3" AND section.title="Bad Bugs"
```

Fig. 10. SQL query corresponding to XQuery query of Fig. 7.

was 18, the total number of rows was 906,443, and the total amount of tables was 364,192KB. We used an 1800MHz AMD Athlon XP processor with 1GB of main memory running Windows 2003. The database platform used was Microsoft SQL Server 2005. Some of the queries which were used in our experiments were taken from [24]. The classification of queries is not complete and not pairwise disjoint, they can overlap. Constructing complex results and reconstructing fragments queries are not included in the experiment since these are not fully supported in our scheme.

### 4.1 Queries

(1) Exact match

Q1: for $b in /site/people/person[@id="person111"] return $b/name/text()

The translated SQL of Q1 in our scheme is as follows:

```
SELECT name FROM Person WHERE id = 'person111' AND pathID = 413
```

Q2: for $b in/site/closed_auctions/closed_auction/annotation/description/parlist/listitem/text/keyword return <keyword>{$b/text()}</keyword>

(2) Selection

Q3: for $b in /site/people/person where $b/profile/age > 40 return $b/name/text()

(3) Containment

Q4: for $b in //item[@id="item0"] return $b//text/text()

The following is the corresponding SQL.

```
SELECT Text.text
FROM ( SELECT elementOrd FROM Item
        WHERE id = 'item0' AND pathID IN
        ( 4, 65, 126, 187, 248, 309 ) ) A, Text
WHERE Text.elementOrd BETWEEN
        A.elementOrd
AND dbo.Descendants( A.elementOrd )
```

Q5: for $b in //keyword return <keyword>{$b/text()}</keyword>

Q6: for $b in /site/regions/*/item return <item id="{$b/@id}" />

(4) Ordered access

Q7: for $b in /site/open_auctions/open_auction return <increase>{$b/bidder[1]/increase/text()}</increase>

The corresponding SQL of Q5 in our scheme is as follows.

```
SELECT Bidder.increase
FROM (
        SELECT * FROM Open_auction
        WHERE docID = '1' AND pathID = 438
        ) A, Bidder
WHERE Bidder.[order] = 1 AND A.id = Bidder.
        parentID
AND Bidder.parentCode = 'open_auction'
```

(5) Path traversal

Q8: for $b in /site/regions/australia/item return <item name= "{$b/name/text()}"> </item>

(6) Number of occurrences

Q9: for $b in /site/regions return count($b//item)

```
SELECT COUNT(*) FROM Item WHERE
```

```
        pathID IN ( 4, 65, 126, 187, 248, 309 )
```

Q10: for $b in /site/regions/australia return count($b/item)

(7) Join on ID

Q11: for $b in //closed_auction/buyer, $c in //person
where $b/@person = $c/@id
return <name>{$c/name/text()}</name>

(8) Join on values

Q12: for $b in //person, $c in //open_auction
where $b/profile/@income < $c/current
return <person name="{fn:distinct-values ($b/name/text())}" />

The translated SQL is as follows.

```
SELECT DISTINCT A.name
FROM ( SELECT * FROM Person WHERE
        pathID = 413 ) A,
     ( SELECT * FROM Open_auction
       WHERE pathID = 438 ) B
WHERE A.income < B.current
```

(9) Missing elements/attributes

Q13: for $b in //person
where empty($b/homepage/text())
return <person name="{$b/name/text()}" />

### 4.2 Results of Experiments

The storage amount of the XML document in SQL Server is 172,172KB. The size of primary index and three secondary indexes is 404,188 KB and 612,141 KB respectively for the scale factor 1.0. The amount of the primary index is usually three times that of the XML data in the table[20]. XQuery is used for querying over XML documents stored using the XML data type. In general, shredding is obviously not a feasible option when the XML schema is very large and complex, resulting in thousands of tables and not complete and sound mapping. However, it is a competitive alternative when the schema is simple and the XML document is data-centric. Table 1 and Table 2 show the time of processing queries in both the extended inlined shredding and XML type. The cost of parsing and building index in XML type and constructing path table in our method is not taken into consideration. The experiments demonstrate that the performance

Table 1. Query performance(ms) for XMark queries for scale factor 0.1

| Query | Extended Inlined Shredding | SQL Server | | | DB2 | Oracle |
|---|---|---|---|---|---|---|
| | | Primary and secondary Index | Primary index | No index | Index | No ndex |
| Q1 | 166 | 808 | 3,218 | 1,178 | 78 | 66,230 |
| Q2 | 101 | 400 | 269 | 1,052 | 211 | 24,660 |
| Q3 | 15 | 3,599 | 375 | 1,208 | 172 | 28,920 |
| Q4 | 61 | 1,195 | 769 | 2,647 | 531 | 62,200 |
| Q5 | 371 | 43,125 | 917 | 1,779 | 253 | 42,530 |
| Q6 | 97 | 32,148 | 800 | 637 | 234 | 50,480 |
| Q7 | 57 | 206 | 368 | 1,488 | 1,063 | 84,070 |
| Q8 | 4 | 172 | 254 | 463 | 188 | 22,840 |
| Q9 | 8 | 66 | 276 | 468 | 172 | 37,120 |
| Q10 | 5 | 37 | 220 | 359 | 15 | 22,390 |
| Q11 | 166 | 8,335 | 545,488 | 508,510 | 824,219 | 46,430 |
| Q12 | 602 | 3,074,942 | 1,202,125 | 1,777,277 | 2,210,687 | >1h |
| Q13 | 117 | 6,602 | 606 | 1,032 | 1,297 | 38,170 |

Table 2. Query performance(ms) for XMark queries for scale factor 1.0

| Query | Extended Inlined Shredding | SQL Server | | | DB2 |
|---|---|---|---|---|---|
| | | Primary and secondary Index | Primary index | No index | Index |
| Q1 | 56 | 3,955 | 19,343 | 6,702 | 1,125 |
| Q2 | 338 | 17,388 | 1,968 | 9,999 | >1h |
| Q3 | 198 | 180,375 | 3,518 | 11,609 | 2,172 |
| Q4 | 7,322 | 9,015 | 6,819 | 41,926 | 18,562 |
| Q5 | 2,615 | 2,670,115 | 7,878 | 17,103 | >1h |
| Q6 | 315 | 1,309,335 | 5,5521 | 5,604 | >1h |
| Q7 | 4,724 | 5,089 | 4,101 | 12,888 | 11,406 |
| Q8 | 104 | 11,026 | 3,233 | 3,183 | 1,766 |
| Q9 | 21 | 80 | 2,656 | 4,223 | 2,313 |
| Q10 | 23 | 56 | 5,239 | 2,999 | 15 |
| Q11 | 688 | 431,660 | >1h | >1h | >1h |
| Q12 | 37,590 | >1h | >1h | >1h | >1h |
| Q13 | 485 | 175,107 | 10,801 | 9,567 | 13,641 |

of processing queries of the shredding method is better than that of XML type in all cases in the experiment. The performance improvement of the extended inlined shredding in query processing is originated from the reduction of joins. By storing and exploiting the order information of elements as mentioned in section 3, the number of joins deceases in processing queries such as A//E. However, our scheme is inferior to the XML type method for queries that extract fragments of elements or construct complex results. There are differences in the performance among SQL Server, DB2, and Oracle. These are due to the fact that representation and indexing mechanism of XML type are different depending on the system. The performance of Oracle for the scale factor 1.0 is left out in the Table 1 since it can not be measured within one hour for the given queries.

## 5. CONCLUSION

We presented a comparison of the performance of processing queries between shredding and XML type for storing XML documents in relational databases. Relational DBMS such as SQL Server, IBM DB2, and Oracle have been extending their facilities to accommodate storing and managing XML documents. XML documents can be stored like a primitive type in a table. Index structures are provided to expedite access to the documents. The shredding method which is one of the conventional methods of storing XML documents in RDBMS is still a viable alternative even though it has some drawbacks due to the structural discrepancy between them. The experiments showed that in most cases the extended version of the Association inlining method had better performance than the other method based on XML type. This is due to the fact that our shredding method keeps and uses the order and path information.

The results of the experiment lead to a hybrid method in which both shredding and XML type are used together for storing XML documents in the RDBMS. Shredding is used on the part of XML documents with simple XML schema without recursion, while the XML data type is used for XML documents with complex and complicated schema. Translation of queries with XQuery into SQL is

still an ongoing issue in the shredding method. The support of schema evolution is also required for the shredding method to be a competitive alternative in storing XML documents in relational databases.

## REFERENCES


[1] K.S. Beyer, R. Cochrane, M. Hvizdos, V. Josifovski, J. Kleewein, G. Lapis, G.M. Lohman, R. Lyle, M. Nicola, F. Özcan, H. Pirahesh, N. Seemann, A. Singh, T.C. Truong, R.C. Van der Linden, B. Vickery, C. Zhang, and G. Zhang, "DB2 goes hybrid: Integrating native XML and XQuery with relational data and SQL," *IBM Systems Journal*, Vol.45, No. 2. pp. 271-298, 2006.

[2] A. Eisenberg and J. Melton, "Advancements in SQL/XML.," *SIGMOD Record*, Vol.33, No.3. pp. 79-86, 2004.

[3] S. Klein, *Professional SQL Server 2005 XML*, Wrox Press, 2006.

[4] R. Murthy, Z.H. Liu, M. Krishnaprasad, S. Chandrasekar, A. Tran, E. Sedlar, D. Florescu, S. Kotsovolos, N. Agarwal, V. Arora, and V. Krishnamurthyet, "Towards an Enterprise XML Architecture," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 953-957, 2005.

[5] M. Nicola and B.V. Linden, "Native XML Support in DB2 Universal Database," *Proceedings of the 31st VLDB Conference*, pp. 164-1174, 2005.

[6] M. Rys, "XML and Relational Database Management Systems: Inside Microsoft SQL Server 2005," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 958-962, 2005.

[7] M. Rys, D.D. Chamberlin, and D. Florescu, "XML and Relational Database Management Systems: The Inside Story," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 945-947, 2005.

[8] P.G. Selinger, "Information Integration and XML in IBM's DB2," *Proceedings of the 28th VLDB Conference*, pp. 906-907, 2002.

[9] S. Pal, I. Cseri, G. Schaller, O. Seeliger, L. Giakoumakis, and V.V. Zolotov, "Indexing XML Data Stored in a Relational Database," *Proceedings of the 30th VLDB Conference*, pp. 1134-1145, 2004.

[10] S. Pal, I. Cseri, O. Seeliger, M. Rys, G. Schaller, W. Yu, D. Tomic, A. Baras, B. Berg, D. Churin, and E. Kogan, "XQuery Implementation in a Relational Database System," *Proceedings of the 31st VLDB Conference*, pp. 1175-1186, 2005.

[11] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. Dewitt, and J. Naughton, " Relational Databases for Querying XML Documents: Limitations and Opportunities," *Proceedings of the 25th VLDB Conference*, pp. 302-314, 1999.

[12] M. Fernandez, Y. Kadiyska, A. Morishima, D. Suciu, and W.C. Tan, "SilkRoute: A Framework for Publishing Relational Data in XML," *ACM Transactions on Database Systems* pp. 438-493, 2002.

[13] R. Krishnamurthy, R. Kaushik, and J.F. Naughton, "XML-to-SQL Query Translation Literature: The State of the Art and Open Problems," *The 1st International XML Database Symposium*, pp. 1-18, 2003.

[14] J. Shanmugasundaram, J. Kiernan, E. Shekita, C. Fan, and J. Funderburk, "Querying XML Views of Relational Data," *Proceedings of the 27th VLDB Conference*, pp. 261-270, 2001.

[15] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, and B. Reinwald, "Efficiently Publishing Relational Data as XML Documents," *Proceedings of the 26th VLDB Conference*, pp. 65-76, 2000.

[16] I. Tatarinov, S. Viglas, K.S. Beyer, J. Shanmugasundaram, E.J. Shekita, and C. Zhang, "Storing and Querying Ordered XML

Using a Relational Database System," *ACM SIGMOD Conference*, pp. 204-215, 2002.

[17] C. Zhang, J.F. Naughton, D.J. DeWitt, Q. Luo, and G.M. Lohman, "On Supporting Containment Queries in Relational Database Management Systems," *ACM SIGMOD Conference*, pp. 425-436, 2001.

[18] M. Yoshikawa and T. Amagasa, "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases," *ACM Transactions on Internet Technology*, Vol.1, No.1. pp. 110-141, 2001.

[19] B.J. Shin and M. Jin, "Association Inlining for Mapping XML DTDs to Relational Tables," *Proceedings of the 2004 International Conference on Computational Science and its Applications*, pp. 849-858, 2004.

[20] Microsoft SQL Server 2005. In http://www.microsoft.com.

[21] IBM DB2. In http://www.ibm.com.

[22] Oracle Database. In http://www.oracle.com.

[23] P.E. O'Neil, E.J. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury, "ORDPATHs: Insert-Friendly XML Node Labels," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 903-908, 2004.

[24] A. Schmidt, F. Waas, M.L. Kersten, M.J. Carey, I. Manolescu, and R. Busse, "XMark: A Benchmark for XML Data Management," *Proceedings of the 28th VLDB Conference*, pp. 974-985, 2002.

**Min Jin**

He received the B.S degree in computer science and statistics from Seoul National University in 1982, and the M.S degree in computer science from KAIST in 1984, and the Ph.D. degree in computer science and engineering from the University of Connecticut in 1997. He has been working at Kyungnam University since 1985. He is currently a professor in the division of computer science and engineering. His research interests include data modeling, object-oriented database, XML storage and processing, and data mining.

**Minjun Seo**

He received the B.S degree in computer engineering from Kyungnam University in 2006. He is a Master student at Kyungnam University. His research interests are database, data modeling, and XML.