MAPPING XML DATA TO RELATIONAL DATA: A DOM-BASED APPROACH

Mustafa Atay, Yezhou Sun, Dapeng Liu, Shiyong Lu, Farshad Fotouhi
Department of Computer Science
Wayne State University, Detroit, MI 48202
{matay, sunny, dliu, shiyong, fotouhi}@cs.wayne.edu

ABSTRACT
XML has emerged as the standard for representing and exchanging data on the World Wide Web. It is critical to have efficient mechanisms to store and query XML data to exploit the full power of this new technology. Several researchers have proposed to use relational databases to store and query XML data. While several algorithms of schema mapping and query mapping have been proposed, the problem of mapping XML data to relational data, i.e., mapping an XML INSERT statement to a sequence of SQL INSERT statements, has not been addressed thoroughly in the literature. In this paper, we propose an efficient linear algorithm for mapping XML data to relational data. This algorithm is based on our previous proposed inlining algorithm for mapping DTDs to relational schemas and can be easily adapted to other inlining algorithms.

KEY WORDS
XML, schema mapping, data mapping, RDBMS.

1 Introduction

XML is rapidly emerging as the de facto standard for representing and exchanging data over the World Wide Web. The increasing amount of XML documents requires the need to store and query XML documents efficiently. Researchers have proposed using relational databases to store and query XML documents [3][5][8] [1] [2][19]. The main challenge of this approach is that, one needs to resolve the conflict between the hierarchical nature of XML data model and the two-level nature of relational data model. The following problems need to be addressed in order to employ relational databases to store and query XML data:

- *Schema mapping*, which generates the corresponding relational schema from an input DTD. Instead of generating a relational table for each XML element, typically, several XML elements are combined into one table to reduce the number of generated tables and the cost of join operations. Representatives of these algorithms include the shared-inlining algorithm [14] and its variation [9].

- *Data mapping*, which inserts XML data as relational tuples into the target database. Based on the relational schema generated in schema mapping, input XML documents are shredded and composed into relational tuples and inserted into the relational database. This requires that an XML INSERT statement be translated into a sequence of SQL INSERT statements, which are executed against the target database to load the data.

- *Query mapping*, which translates XML queries into SQL queries. Each XML query over XML documents needs to be translated into a sequence of SQL queries to be executed against the relational database.

- *Reverse data mapping*, which publishes XML data from relational data. XML queries are answered by executing the corresponding SQL queries which return relational data. These relational data need to be reformatted into XML data conforming to the structure imposed by the input XML query.

Numerous researchers have addressed the problems of schema mapping [9] [14] [19], query mapping [2] [14] [17] and reverse data mapping [2] [4] [13]. However, the problem of data mapping is mostly ignored in the literature. In this paper, we address the data mapping problem. We propose an efficient linear algorithm to perform data mapping. This algorithm is based on our proposed schema mapping algorithm [9] (referred to by DTDMap afterwards) but can be easily adapted to other inlining algorithms such as the standard shared-inlining algorithm [14]

$Organization.$ The rest of the paper is organized as follows. Section 2 presents an overview of related work. Section 3 gives a brief overview of our schema mapping algorithm DTDMap. Section 4 identifies the data model and describes our proposed data mapping algorithm XInsert. Section 5 presents the experimental results of applying our data mapping algorithm to the two schema mapping algorithms, DTDMap and shared-inlining. Finally, Section 6 concludes the paper and points out some potential future work.

2 Related Work

Different approaches have been proposed for storing and querying XML data. One approach is to develop native XML databases that support XML data model and query languages directly. This includes Software AG's Tamino XML Server [15], IXIA's TEXTML Server [7] and Sonic Software's eXtensible Information Server [16] (formerly eXcelon's XIS). The advantage of this approach is that XML data can be stored and retrieved in their original

formats and no additional mappings or translations are needed. Furthermore, most native XML databases have the ability to perform sophisticated full-text searches including full thesaurus support, word stubbing, and proximity searches. The disadvantage is that, due to the document-centric nature of these databases, complex searches or aggregations might be cumbersome.

The second approach is to use the XML enabled commercial database systems. Currently, most major databases such as SQL Server [10], Oracle [11] and DB2 [6] provide mechanisms to store and query XML data by extending the existing data model with an additional XML data type so that a column of this data type can be defined and used to store XML data. In addition, a set of methods is associated with this new XML data type to process, manipulate and query stored XML data.

The third approach is to use existing mature technologies such as relational DBMSs or object-oriented DBMSs to store and query XML data [3] [9] [14]. The main challenge of this approach is that, one needs to resolve the conflict between XML data model and the target data model. This usually requires various mappings (e.g., schema mapping, data mapping and query mapping) to be performed between the two data models. Therefore, the main issue is to develop efficient algorithms to perform these mappings.

Different approaches have their pros and cons and the choice has to be made based on the requirement of the application at hand and the advancement of these approaches at the time that the choice has to be made. Readers are referred to a recent evaluation study of alternative XML storage strategies [18] for more details.

3 An Overview of Schema Mapping Algorithm DTDMap

The data mapping algorithm proposed in this paper is based on our schema mapping algorithm, DTDMap, proposed in [9]. In this section, we give a brief overview of DTDMap algorithm.

One approach for mapping DTDs to relational schema is mapping each node in the DTD to a table. Although this approach is easy to understand and implement, it has its own drawbacks. This approach results in many tables in the corresponding relational schema. When you query this database, you need to join several tables which causes query processing to be inefficient.

Alternatively, in our DTDMap algorithm, we suggest combining every single child node in a DTD, to its parent node, if it appears in its parent at most once. We call this operation $inlining$. A node is said to be $inlinable$ if it has exactly one parent node, and, its cardinality is not equal to either "*" or "+". An inlinable node is mapped with its parent node into the same table. Hence, we reduce the number of tables and consequently the average number of joins for queries.

DTDMap algorithm takes a DTD as input and produces a relational schema as output. In addition, it outputs mapping functions between XML elements and attributes in the input DTD, and, corresponding tables and relational attributes in the output schema.

```
<!DOCTYPE univ [
  <!ELEMENT univ (colleges, schools?) >
  <!ATTLIST univ uName CDATA #REQUIRED>
  <!ELEMENT colleges (colloge+) >
  <!ELEMENT college (dep* ) >
  <!ATTLIST college cName CDATA #REQUIRED>
  <!ELEMENT schools (school+) >
  <!ELEMENT school (dep*) >
  <!ATTLIST school sName CDATA #REQUIRED>
  <!ELEMENT dep (tel?, fax?, website?) >
  <!ATTLIST dep dName CDATA #REQUIRED>
  <!ELEMENT tel (#PCDATA) >
  <!ELEMENT fax (#PCDATA) >
  <!ELEMENT website (#PCDATA) >
]>
```

Figure 1. Sample XML DTD "univ.dtd"

univ (ID, nodeType, uName)

college (ID, cName)

school (ID, sName)

dep (ID, nodeType, dName, tel, fax, website)

edge (parentID,childID , parentType,childType)

Figure 2. Relational schema for univ.dtd

As an example, given the DTD in Figure 1, DTDMap generates the relational schema shown in Figure 2 and the following three mapping functions σ, θ, and δ:

- σ(e) maps an element type to a corresponding relational table. Therefore, σ(univ) = univ, σ(uName) = univ, etc.
- θ(a) maps an XML attribute to a relational attribute. Therefore, θ(uName) = uName, θ(dName) = dName, etc.
- δ(e) maps a leaf element to a relational attribute. Therefore, δ(tel) = tel, δ(fax) = fax, etc.

In the above examples, some mappings happened to be identity mappings. This is not always the case in practice; they can be general enough and used to resolve name conflict. For example, a mapping θ(date) = BirthDate can avoid the use of "date" as a column name for a table since "date" might be a keyword of the target database and cannot be used as the name of a relational attribute.

4 Data Mapping

The data model we will use for the data mapping algorithm is based on the W3C's Document Object Model [20], but it has some distinctions from DOM specification. In contrast to traditional DOM tree, the XML element DOM tree, which we propose here, does not consider XML values as nodes but consider them as data fields of XML element nodes. This distinction is only for the convenience of the

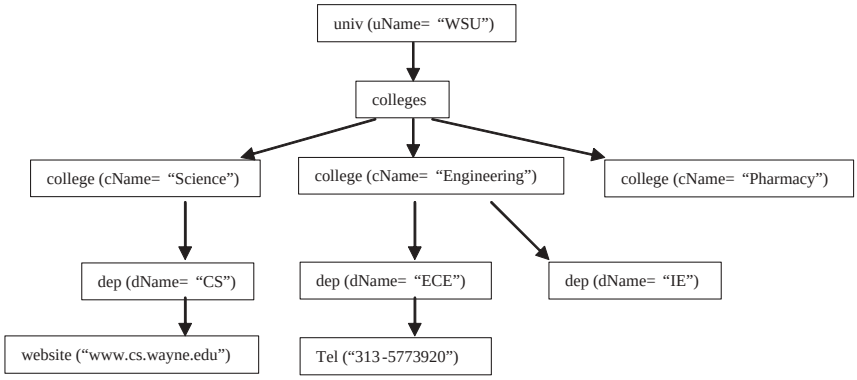

Figure 4. DOMTree for univ.xml

```
<?xml version=”1.0” ?>
<!DOCTYPE univ SYSTEM “univ.dtd”>
<univ uname= “WSU”>
  <colleges>
   <college cname= “Science”>
      <dep dname= “CS”>
     <website>www.cs.wayne.edu</website>
      </dep>
   </college>
   <college cname= “Engineering”>
      <dep dname= “ECE”>
        <tel>313-5773920</tel>
      </dep>
      <dep dname= “IE”></dep>
   </college>
   <college cname= “Pharmacy”></college>
 </colleges>
</univ>
```

Figure 3. Sample XML document univ.xml

presentation; thus the algorithm proposed in this paper can be implemented directly on the standard DOM model. We define the notion of *XML element DOM tree* as follows:

Definition 4.1 (DOMTree) We model an XML element document D as an XML element DOM tree (DOMTree) T, in which nodes represent XML elements and edges represent parent-child relationships between XML elements. For each XML element node e in T, we use the following notations:

- $e.name$, the name of the XML element.
- $e.parent$, the parent node of e, and $e.parent = NULL$ if e is the root node of T.
- $e.children$, the set of children nodes of e, and $e.children = \Phi$ if e is a leaf of T. We also denote the children of e by $e.c_1, \cdots, e.c_m$.
- $e.attributes$, the set of XML attributes of e. We also denote the attributes of e by $e.a_1, \cdots, e.a_n$, and the names and values of these attributes by $e.a_i.name$ and $e.a_i.value$ respectively ($i = 1, \cdots, n$).
- $e.value$, the value of e, and $e.value = NULL$ if e is a non-leaf node.

An XML element DOM tree for the XML document given in Figure 3 is illustrated in Figure 4. Each node e is labeled by $e.name(e.value, e.a_1.name = e.a_1.value, \cdots, e.a_n.name = e.a_n.value)$ and $e.value$ is omitted when e is non-leaf node where $e.value$=NULL.

Our data mapping algorithm XInsert is based on the notion of inlinable elements introduced in the schema mapping phase. However we cannot find out whether a XML element instance is inlinable from a DOMTree itself. We need to refer to the DTD information.

Figure 5 describes the algorithm XInsert which inserts an XML document into the relational database whose schema is previously generated from the input XML DTD.

Given a DTD graph G and nodes n_1 and n_2 in G, $l(n1, n2, G)$ denotes the label of edge between n_1 and n_2. Given an element instance e, $type(e)$ denotes the corresponding element node in G.

We define a field EID which is associated with each element instance e in the algorithm. EID is a unique value and it is generated for each non-inlinable element when it is first visited. We introduce $parentEID$ and $parentNodeType$ fields in the algorithm to keep the parent-child relationship between the elements.

XInsert algorithm is driven by two nested While loops. The outer While loop maintains a Queue, q, to process the non-inlinable XML elements. It obtains the typical information of the tuple, tp, corresponding to a non-inlinable element, e, such as ID, nodeType, XML attribute values and the content(lines 10-14). Finally, it inserts the tuple tp into the table $\sigma(e)$ (line 38). If $type(e)$ is a *-element, then it inserts the tuple, tp_e into the $edge$ table to store the parent-child relationship (lines 39-42).

If e is not a leaf element then the inner While loop is performed to search for inlinable descendants of e. It main-

```
1 Algorithm XInsert
2 Input: DOMTree T, DTDGraph G, Schema mappings σ, θ, δ
3 Output: elements in T are inserted into the relational database
4 Begin
5    Queue q := EmptyQueue(), T.root.EID.value := genID(), q.enqueue(T.root)
6    While q.isNotEmpty() do
7        e := q.dequeue()
8        Table tb:= σ(e)
9        create a tuple tp of table tb with all attributes initialized to NULL
10       tp.ID = e.EID.value
11       If nodeType ∈ tb.AttributeSet then tp.nodetype = e.name End If
12       For each XML attribute e.a_i of e do tp.θ(e.a_i) := e.a_i.value End For
13       If e is a leaf then
14           tp.δ(e) := e.value
15       Else /* e is not a leaf */
16           Queue r := EmptyQueue()
17           For each child e.c_i of e do r.enqueue(e.c_i) End For
18           While r.isNotEmpty() do
19               f := r.dequeue()
20               If f is not inlinable to e then
21                   f.EID.value := genID()
22                   f.parentEID.value := e.EID.value
23                   f.parentNodeType.value := e.name
24                   If l(type(f.parent), type(f), G) <> '*' then
25                       tp.θ(f.EID) := f.EID.value
26                   End If
27                   q.enqueue(f)
28               Else /* f inlinable to e */
29                   for each XML attribute f.a_i of f do tp.θ(f.a_i) := f.a_i.value End For
30                   If f is a leaf then
31                       tp.δ(f) := f.value
32                   Else /* f is not a leaf */
33                       For each child f.c_i of f do r.enqueue(f.c_i) End For
34                   End If
35               End If
36           End While
37       End If
38       Insert tuple tp into table tb
39       If l(type(e.parent), type(e), G) == '*' then
40           insert < e.parentEID.value, e.EID.value, e.parentNodeType.value, e.name >
41           into table edge
42       End If
43   End While
44 End Algorithm
```

Figure 5. The algorithm for mapping XML data to relational data

tains a Queue, r, to process the descendants of e. Firstly, it determines the inlinable descendants of e and retrieve their data to complete the context information for the tuple tp (lines 28-35). Secondly, it keeps the parent-child information of the non-inlinable descendants of e through the fields $parentEID$ and $parentNodeType$ (lines 20-23). Lastly, it introduces a foreign key in the tuple tp if there exists a shared descendant of e (lines 24-26). The algorithm ends when there are no more elements to be processed in queues, q and r.

To analyze the time complexity of algorithm XInsert, we first present some properties of the algorithm in the following lemmas.

Lemma 4.2 *Each non-inlinable element e in DOMTree T is enqueued into Queue q exactly once, and q only contains non-inlinable elements.*

Proof: The operation of enqueue into q is performed only at line 5 and at line 27. Line 5 enqueues the root element which is non-inlinable. Line 27 is in the body of if-statement (line 20) whose condition indicates the element f to be enqueued into q in line 27 is non-inlinable. Therefore, q only contains non-inlinable elements.

To demonstrate that each non-inlinable element e is enqueued into q exactly once, we prove in the following that e is enqueued into q at most once and at least once respectively. First, we notice that for each element e that is dequeued from q, e is non-inlinable as q only contains non-inlinable elements. The While-statement (line 18 to 36) will enqueue each of e's descendant element f exactly once into Queue r, where f satisfies (1) f is e's child (line 17) , or (2) f is inlinable to e (line 33), or (3) f is non-inlinable to e but f's parent is inlinable to e (line 33). The acyclicity of T implies that each non-inlinable element of T can be enqueued into q at most once. In addition, except the root element, the While-statement (line 18 to 36) will ensure that each non-inlinable element will be enqueued into q at least once in line 27. Finally, the root element is enqueued into q exactly once. Therefore, each non-inlinable element e is enqueued into q exactly once.

Lemma 4.3 *Each XML element e, except the root element in DOMTree T is enqueued into Queue r exactly once.*

Table 1. The state of the database after univ.xml is stored

Univ

ID	*nodeType*	*uName*
1	univ	WSU

Edge

parentID	*childID*	*parentType*	*childType*
1	2	univ	college
1	3	univ	college
1	4	univ	college
2	5	college	dep
3	6	college	dep
3	7	college	dep

School

ID	*sName*

Dep

ID	*nodeType*	*dName*	*tel*	*fax*	*website*
5	dep	CS	null	null	www.cs.wayne.edu
6	dep	ECE	313-5773920	null	null
7	dep	IE	null	null	null

College

ID	*sName*
2	Science
3	Engineering
4	Pharmacy

Proof: Lemma 4.2 implies that each non-inlinable element e is dequeued from q exactly once (line 7), and for each such e, the While-statement (line 18 to 36) will enqueue each of e's descendant element f exactly once into Queue r, where f satisfies (1) f is e's child (line 17), or (2) f is inlinable to e (line 33), or (3) f is non-inlinable to e but f's parent is inlinable to e (line 33). Therefore, each element of T except the root element will satisfy one of these three cases for some e and thus will be enqueued into r at least once. The acyclicity of T implies that each element of T can be enqueued into r at most once. Therefore, each XML element in T is enqueued into r exactly once.

The following theorem demonstrates that XInsert is an efficient linear algorithm.

Theorem 4.4 *The time complexity of algorithm XInsert is O(n) where n is the number of XML elements and attributes in DOMTree T.*

Proof: From Lemma 4.2, it follows that the While loop statement in line 6 will be executed for $m1$ times, where $m1$ is the number of non-inlinable elements in T. From Lemma 4.3, it follows that the While loop statement in line 18 will be executed for $m2$ times where $m2 = n - 1$. We have $m1 <= m2 < n$. All the operations involved in those two While loops spend constant amount of time. Hence, it is clear that the XInsert algorithm runs in $O(n)$ time complexity.

Table 1 shows how the XML document given in Figure 3 is mapped into the relational database given in Figure 2 by the data mapping algorithm explained above.

We define the overall mapping process as comprised of schema mapping and data mapping modules. The system architecture is given in Figure 6.

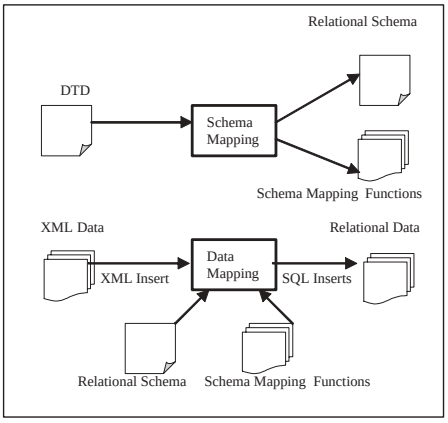

Figure 6. System Architecture

5 Experimental Results

We applied the data mapping algorithm introduced in this paper to two different XML schema mapping schemes. The first XML schema mapping scheme that we use in our experiment is DTDMap algorithm [9] and the second scheme is the shared inlining algorithm [14]. These schemes map the XML elements to relational tables based on the operation of inlining child nodes into parent nodes.

We chose a DTD with document-centric features and another one with data-centric features for our experiment to show the performance of our algorithm on documents with different features. We took the first DTD, which is $catalog.dtd$, from XBench [21] XML Benchmark project and generate our test documents. This DTD shows data-centric features. The second DTD, which is $auction.dtd$, was taken from XMark [12] XML Benchmark project and the test documents were generated. The second DTD has document-centric features besides its data-centric features.

We used a Pentium IV computer with 2.4 GHz processor and 512 MB main memory. We ran our implementation using Java 1.4.1 software development kit. We maintained DOM element tree using W3C's DOM specification and processed it using an on-shelf DOM API.

We generated test documents in five different sizes from 10 MB to 50 MB. Then we applied test runs for each size of test documents for both DTDs using both mapping schemes. We created flat comma-separated text files for each table of the corresponding relational schema. Our performance metric is the time spent to map XML data to relational data. Loading data to the database is not included in this time. In order to see the pure performance of our data mapping algorithm, we did not populate a database

directly.

We minimized the usage of system resources during the experiments to get more realistic spent time values. On the other hand we repeated every experiment for five times and got the mean value of spent time to obtain more accurate results.Table 2 shows the time spent for data mapping in seconds.

Table 2. The time spent for XInsert data mapping algorithm

Algorithm	Document	Document Size				
		10MB	20MB	30MB	40MB	50MB
DTDMap	auction.xml	6.34	11.57	19.15	71.78	483.88
	catalog.xml	6.10	11.53	17.95	31.09	112.09
Shared	auction.xml	10.43	20.19	30.46	53.37	409.93
	catalog.xml	8.72	17.82	25.19	43.28	113.47

We observe from the above table that both mapping schemes for both DTDs make a pick after 40 MB. Document Object Model loads the whole document tree into the main memory and then make the whole tree traversal available. When the document gets larger, it hardly fits into the memory. Then system stores some part of the tree in the disk and starts to come back and forth between the disk and the main memory which causes the increase in time.

We see that processing $auction.xml$ document takes more time than processing $catalog.xml$ document on average. We see the difference more precisely especially at 50 MB for both mapping schemes. The $auction.xml$ document includes document-centric textual elements which are nested recursively and cause DOMTree to be deeper where there is no recursive element in $catalog.xml$.

6 Conclusions

Several algorithms have been proposed for schema mapping by researchers, but the problem of data mapping has not been discussed thoroughly in the literature. In our study, we have addressed the problem of data mapping and defined an efficient and linear algorithm for mapping XML data to relational data. Our algorithm populates a relational database with the input XML documents, according to the relational schema which is generated by the schema mapping algorithm DTDMap. The XInsert data mapping algorithm can be easily adapted to other mapping schemes based on inlining technique.

Ordered nature of XML elements are ignored in this study. We have not dealt with the referential and integrity constraints in our data mapping algorithm. These issues need to be investigated as a potential future work.

References

[1] R. P. Bourret. *XML and Databases*, 2002. http://www.rpbourret.com/xml/XMLAndDatabases.htm.

[2] M. J. Carey, D. Florescu, Z. G. Ives, Y. Lu, J. Shanmugasundaram, E. J. Shekita, and S. N. Subramanian. XPERANTO: Publishing object-relational data as XML. In *WebDB (Informal Proceedings)*, pages 105–110, 2000.

[3] A. Deutsch, M. F. Fernandez, and D. Suciu. Storing semistructured data with STORED. *SIGMOD Conference*, pages 431–442, 1999.

[4] M. F. Fernandez, Y. Kadiyska, D. Suciu, A. Morishima, and W. C. Tan. Silkroute: A framework for publishing relational data in XML. *TODS*, 27(4):438–493, 2002.

[5] D. Florescu and D. Kossmann. Storing and querying XML data using an RDBMS. *IEEE Data Engineering Bulletin*, 22(3):27–34, 1999.

[6] IBM. *DB2 extender for XML*. http://www-4.ibm.com/software/data/db2/extenders/xmlext.html.

[7] IXIASOFT. *TEXTML Server*. http://www.ixiasoft.com/textmlserver/.

[8] G. Kappel, E. Kapsammer, and W. Retschitzegger. XML and relational database systems - a comparison of concepts. In *Int. Conf. on Internet Computing (1)*, pages 199–205, 2001.

[9] S. Lu, Y. Sun, M. Atay, and F. Fotouhi. A new inlining algorithm for mapping XML DTDs to relational schemas. In *Proc. of the First Int. Workshop on XML Schema and Data Management, in conj. with the 22nd ACM Int. Conference on Conceptual Modeling (ER2003)*, Illinois, USA, Oct 2003.

[10] Microsoft. *SQL Server XML support*. http://msdn.microsoft.com/msdnmag/issues/0300/sql/sql/asp.

[11] S. Muench. *Using XML and Relational Database for Internet Applications*. Oracle Corporation. http://otn.oracle.com/tech/xml/htdocs/relational/paper.html.

[12] A. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, , and R. Busse. Xmark: A benchmark for XML data management. In *VLDB*, pages 974–985, 2002.

[13] J. Shanmugasundaram, E. J. Shekita, R. Barr, M. J. Carey, B. G. Lindsay, H. Pirahesh, and B. Reinwald. Efficiently publishing relational data as XML documents. In *VLDB*, pages 65–76, 2000.

[14] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *VLDB*, pages 302–314, 1999.

[15] Software AG. *Tamino XML Server*. Software AG. http://www.softwareag.com/tamino/.

[16] Sonic Software. *eXtensible Information Server*. http://www.sonicsoftware.com/.

[17] 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. In *SIGMOD Conference*, pages 204–215, 2002.

[18] F. Tian, D. J. DeWitt, J. Chen, and C. Zhang. The design and performance evaluation of alternative XML storage strategies. *SIGMOD Record*, 31(1):5–10, 2002.

[19] I. Varlamis and M. Vazirgiannis. Bridging XML-schema and relational databases: a system for generating and manipulating relational databases using valid XML documents. In *Proc. of ACM Symposium on Document Engineering*, Atlanta, USA, Nov 2001.

[20] WWW Consortium. *Document Object Model 2.0*, 2000. http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/.

[21] B. B. Yao, M. T. Ozsu, and J. Keenleyside. XBench - a family of benchmarks for XML DBMSs. In *EEXTT*, pages 162–164, 2002.