

# A DOM-Based Approach of Storage and Retrieval of XML Documents Using Relational Databases

Hosam F. El-Sofany<sup>1</sup>, Samir A. El-Seoud<sup>2</sup>, Fayed F. M. Ghaleb<sup>3</sup>,  
Jihad M. Al Ja'am<sup>1</sup>, Sameh S. Daoud<sup>3</sup>, and Ahmad M. Hasnah<sup>1</sup>

<sup>1</sup>Department of Computer Science and Engineering, College of Engineering, Qatar University.

<sup>2</sup>Department of Computer Science, Princess Sumaya University for Technology

<sup>3</sup>Department of Mathematics, Faculty of Science, Ain Shams University.

**Abstract:** *This paper describes a novel approach of storage and retrieval of XML documents using relational databases. In this approach, an XML document is decomposed into nodes based on its tree structure, and stored into relational tables according to the nodes types. Our approach enables us to store XML documents using a fixed relational schema without any information about XML schema, and DTD. For the processing of XML documents, we propose two algorithms denoted by XR and RX, where the first one is for converting XML data to relational data, and the second one for extracting data from a database and insert it into XML document. The application does not impose any extension of relational databases for storage and retrieval of XML documents. We show the effectiveness of this approach through several experiments using different XML documents.*

**Keywords:** *XML documents, Data Mapping, Query Processing, Algorithms, Information Storage and Retrieval.*

**Received:** January 10, 2007 | **Revised:** June 10, 2007 | **Accepted:** August 20, 2007

## 1. Introduction

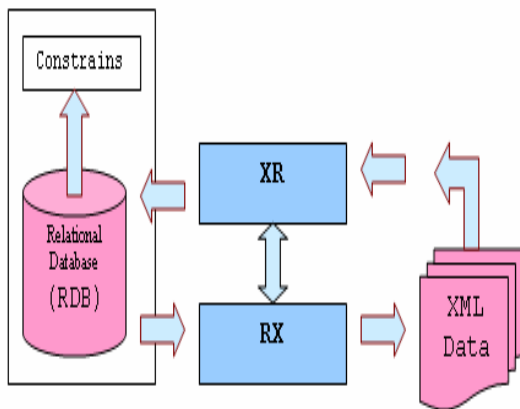
XML (eXtensible Markup Language) is emerging as a standard format for data and documents on the Internet [1]. Various kinds of applications that use the XML format have been developed and deployed in the net. As a result, many kinds of data will be exchanged in the form of XML documents or XML data. In addition, storage devices are begun to be miniaturized while their capacity is being enlarged. Thus, it is expected that not only organizations but also individuals will use a large quantity of XML documents in the near future. Development of techniques for storing massive XML documents and retrieving information needed from them is one of the core problems regarding the point of contact between the research area of database and XML [13,14].

One of the important features of XML documents is that we can perform operations based on their logical structures [12]. Based on this feature, databases that manage XML documents have to support queries on their logical structures and on their contents. Considering access based on a logical unit and reusability, it is appropriate to decompose and store

XML documents according to their tree structures. They are then stored in appropriate databases. In order to retrieve XML documents from such databases, we use a dynamic application based on a W3C's Document Object Model (DOM), and database queries (typically in SQL).

In this work, we describe a novel approach of translating XML data to relational data, and also to extracting data from relational database tables and insert it into XML documents (see Figure 1). The design goals of our approach are as follows:

- (1) No restriction should be imposed on XML documents being stored. Thus, any *valid* or *well-formed* XML documents should be stored and queried.
- (2) Storage and manipulation of XML documents should be made possible using currently available relational databases.
- (3) No extension of data model, query expressive power, or index structures of relational database systems should be assumed
- (4) The data model used for the data mapping algorithms is based on the W3C's DOM [18].



**Figure 1.** Overview of our approach: Two algorithms, XR and RX, are used independently or jointly.

The paper is organized as follows: in section two we describe a number of important issues (approaches) through the related works. In section three we will give a brief overview about the XML documents and DOM model. In section four we introduce the XR algorithm, used for converting XML data to relational data. In section five, we introduce the RX algorithm, used for extracting data from a database and insert it into XML document, while in section six we describe the experimental results. The paper is finally concluded in section seven.

## 2. Related Works

Different approaches have been proposed for storing and querying XML data [2,9,16]. 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 [10], IXIA's TEXTML Server [5] and Sonic Software's eXtensible Information Server [11]. 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 other approach is to use the *XML enabled database* systems, such as SQL Server [7], Oracle [8], and DB2 [4] which 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. This usually requires various mappings (e.g., *schema mapping*, *data mapping* and *query mapping*) to be performed between the two data

models [3,6]. Therefore, the main issue is to develop efficient algorithms to perform these mappings.

## 3. An Overview of XML Documents

An *XML document* consists of three parts: an XML declaration, a DTD (Document Type Definition) or XML Schema, and an XML instance (XML document data). An XML declaration and Schemas are not mandatory for an XML document. An *XML declaration* specifies the version and the encoding of XML being used. A *DTD* or *XML Schema* is a schema that constrains the structure of XML instances, and corresponds to an extended context-free grammar. An *XML instance* is a tagged document. We omit concrete descriptions of an XML declaration, and a DTD.

An XML instance is a hierarchy of elements, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements, by empty-element tags. Character data between start-tags and end-tags are the content of the element. Figure 2 shows an example of an XML instance. A start-tag is the token that encloses an element type with < and >, and an end-tag is the token that encloses an element type with </ and >. Elements can nest properly within each other, and the nesting represents logical structure. Within start-tags, attribute names and attribute values can be specified.

```

<?xml version="1.0"?>
<books>
  <book>
    <author> H.M. Deitel and P.J.Deitel </author>
    <title> C++ How To Program </title>
    <publisher> Prentice Hall Publishing Co. </ publisher>
  </book>
  <book>
    <author>Jack Herrington</author>
    <title>PHP Hacks</title>
    <publisher>O'Reilly</publisher>
  </book>
</books>
  
```

**Figure 2.** An XML instance (books.xml).

XML documents have two levels of conformance: *valid* and *well-formed*. A *well-formed* XML document follows tagging rules prescribed in XML. An XML document is *valid* if it is well-formed and if the document complies with the constraints expressed in an associated schema.

**Definition 3.1 (XMLTree):** An *XML document* can be viewed as a *tree*, where leaf nodes correspond to data values (text) and internal nodes correspond to XML elements [18].

### 3.1 The DOM Model For XML Documents

The Document Object Model is an object model to represent XML documents and an interface to work with that model. It gives a *tree* overview of all XML elements and how they relate to one another. It is a good model for handling XML documents because it takes the tree like model of XML as its core idea and makes no presumptions about the structure of the document. This also presents a good starting point for creating mappings to other tree based structures.

**Definition 3.2 (XDTree):** We model an XML document  $D$  as an XML DOM document tree (XDTree)  $T$ , 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  $e.parent = \text{NULL}$  if  $e$  is the root node of  $T$ .
- $e.children$ : the set of children nodes of  $e$ , and  $e.children = \emptyset$  if  $e$  is a leaf of  $T$ . We also denote the children of  $e$  by:  $e.c_1, \dots, e.c_m$ .
- $e.attributes$ : the set of XML attributes of  $e$ . We also denote the attributes of  $e$  by:  $e.a_1, \dots, e.a_n$ . The names and values of these attributes denoted by  $e.a_i.name$  and  $e.a_i.value$  respectively ( $i = 1, \dots, n$ ).
- $e.value$ : the value of  $e$ , and  $e.value = \text{NULL}$  if  $e$  is a non-leaf node.

An XML DOM tree for the XML document given in Figure 2 is illustrated in Figure 3. The Figure shows that, the *books* node at the top of the tree has two child *book* tags. Within each *book*, there are *author*, *publisher*, and *title* nodes. The *author*, *publisher*, and *title* nodes each have child *text* nodes that contain the text.

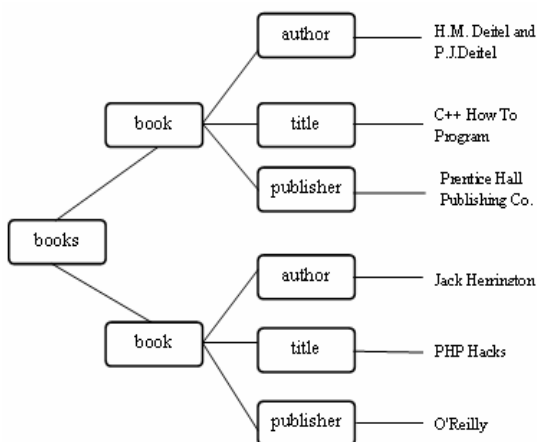


Figure 3. XML DOM tree for the `books.xml`.

## 4. Converting XML Data to Relational Data

In this section, we describe the relational database schemas used to store XML documents. In our

implementation phase, we used MySQL Server 5.0 as a Relational Database Management System (RDBMS), PHP 5.0, and Apache Server.

### 4.1 Proposed relational schema

When trying to store XML document into a RDBMS one cannot use the DTD or XML Schema to generate a mapping, because if at all present with all documents, it might vary wildly between them. Therefore another approach is needed, something that maps the structure of XML itself. Note that to be useful for document storage, the document order needs to be preserved. Hence, we have mapped the XML DOM model onto the following relational database schema:

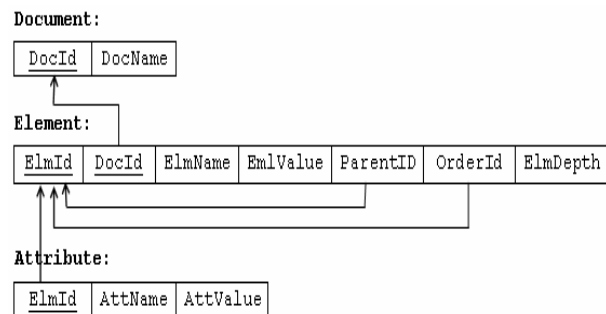


Figure 4. The basic XR schema for mapping the XML document to relational tables

The core of the mapping is the **Element** entity, that contains the following attributes: **ElmId**: A unique identifier for each element, **DocId**: The unique key of the associated document, **ElmName**: The name of this element, **ElmValue**: The text or data of this element (NULL for none), **ParentID**: The element which is parent to this element (NULL for the root element), **OrderID**: Element id of the next element in scope order (NULL for last element), and **ElmDepth**: The scope depth of this element.

The **Element** entity is supplemented with two other entities. The first one is the **Document** entity, that contains the following attributes: **DocId**: A unique identifier for each document, and **DocName**: The name of this document. The second entity is the **Attribute** entity, that contains information about each attribute; such as, **ElmId**: The unique key of the associated element, **AttName**: The name of this attribute, and **AttValue**: The value of this attribute.

### 4.2 XR Algorithm: For Mapping XML Document to RDB Tables

The *XR* algorithm takes the XML document file as input and store its contents in a relational database (RDB) tables.

**Algorithm** : *XR*

**Input**: XML document file ( $D = \text{"XMLFile.xml"}$ ), as an XML DOM tree  $T$ .

**Output:** Insert the XML document into relational database tables.

```

1. Begin
2. xml_file = "XMLFile.xml" // XML file
3. T = xmldocfile($xml_file) // parse it
4. root = T->root() // get the root node
5. children = get_children(root) // get its children
6. elementCount = 1 // starting with 1 to include document element
7. read_dom_tree(children) // read the XML elements Dom Tree
8. insert_dom_tree_into_db(1)
9. // this recursive function accepts an array of nodes as argument, iterates
   // through it and store a list for each element found.
10. function read_dom_tree(nodeCollection) {
11.   global elementCount // iterate through array
12.   for (i=0; i<sizeof(nodeCollection); i++) {
13.     elementCount++
14.     e.name = nodeCollection[i]->tagname
15.     e.value = nodeCollection[i]-> item(0) -
   >nodeValue
16.     e.parent = nodeCollection[i]->parentID
17.     e.order_id = nodeCollection[i]->orderID
18.     e.depth = nodeCollection[i]->depth
19.     insert_dom_tree_into_db(2)
20.     if ( e.attributes != Null )
21.       { e.attname = nodeCollection[i]->att_tagname
22.         e.attvalue = nodeCollection[i]->att_value
23.         insert_dom_tree_into_db(3) }
24.     // go to the next level of the tree
25.     nextCollection = get_children(nodeCollection[i])
26.     read_dom_tree(nextCollection)
27.   }
28. // this function return an array of children, given a parent node
29. function get_children(node)
30. { temp = node->children()
31.   collection = array()
32.   for (j=0; j<sizeof(temp); j++) // iterate through children array
33.     // filter out all nodes except elements and create a new array
34.     if (temp[j]->type == XML_ELEMENT_NODE)
35.       { collection[] = $temp[j] }
36.   }
37.   return collection; // return array containing child nodes
38. }
39. function insert_dom_tree_into_db (flage)
40. { dbname = 'XR'
41.   connection = mysql_connect(dbhost,'root', 'hosam')
42.   mysql_select_db(dbname)
43.   if (flage == 1)
44.     { query = "insert into Document (docName) values (xml_file)"
45.     else if ( flage == 2)
46.       query="insert into Element values (e.name,e.value, e.parent,
         e.order_id, e.depth)"
47.     else
48.       query = "insert into Attribute (e.attname, e.attvalue)"
49.     }
50.   mysql_query(query)
51.   mysql_close(connection)
52. }
53. End

```

**Figure 5.** XR : A pseudo-code algorithm for storing XML document into relational tables

Figure 6 shows a database instance of the XR schema, that explains how the XML document given in Figure 2 is mapped into the relational database schema given in Figure 4, by using the XR algorithm explained in Figure 5.

### 4.3 Mapping Properties for the XR Algorithm

We analyze the performance and time complexity of the XR algorithm, by considering the following properties for the proposed data mapping:

(1) *Speed*: The performance of our model is equal to the *Edge* model [17], because it is a variation of it.

The main differences between this model and the *Edge* model are:

- *Edge* uses a single table to store all attributes for mapping an XML document to a relational database.
- *Edge* uses numbers to preserve ordering. This makes inserting a cumbersome process because inserting between two numbers requires a renumbering of all the following elements.
- *Edge* does not include explicit depth information. Depth information accelerates queries on descendants of nodes.

To increase the performance of XQuery and XPath [15,19] queries, a *Path* table and *Element.PathId* column may be added. This *Path* table holds the names of the nodes above the current elements. Queries which specify a node where to start their search benefit from this because they can locate the node with only one lookup. Without this extra information they would have to 'walk' down the tree. So the *basic XR Schema* for mapping the XML document to a relational tables can be modified to the following database schema:

```

Document (DocId, DocName)
Element (ElmId, DocId, ElmName, ElmValue,
ParentID, OrderId, ElmDepth, PathId)
Attribute (ElmId, AttName, AttValue)
Path (ElmId, Path)

```

**Document table:**

DocId	DocName
1	books.xml

**Element table:**

ElmId	DocId	ElmName	ElmValue	ParentID	OrderId	ElmDepth
1	1	xml	Null	Null	2	0
2	1	books	Null	1	3	0
3	1	book	Null	2	7	0
4	1	author	H.M. Deitel and P.J.Deitel	3	5	1
5	1	title	C++ How To Program	3	6	1
6	1	publisher	Prentice Hall Publishing Co.	3	Null	1
7	1	book	Null	2	Null	0
8	1	author	Jack Herrington	7	9	1
9	1	title	PHP Hacks	7	10	1
10	1	publisher	O'Reill	7	Null	1

**Attribute table:**

ElmId	AttName	AttValue
1	version	"1.0"

**Figure 6.** A database instance of the XR schema storing the XML document in Figure 2.

- (2) *Scalability*: This item is for a large part determined by the scalability of the chosen RDBMS backend. Our approach is very reasonable, in all size of XML documents.
- (3) *Accuracy*: Excluding the XML schemas scales (like DTD and XML Schema), our approach granted to store the XML document, and all the relationships between it elements.

(4) *XML Query capability*: XPath and XQuery queries are accelerated by use of the proposed `Path` table extension in property (1). This table could also be implemented as an extra column in the `Element` table. Depending on your RDBMS solution that might actually be faster. Because no XML Schema information is explicitly stored, XPath operators like `<`, `>`, `<=`, `>=` are hard to map. One can choose to cast values to numeric types when these operators occur, or just use the default string compare.

**Theorem 4.1** *The time complexity of algorithm XR is  $O(n)$  where  $n$  is the number of XML elements and attributes in XML DOM Tree  $T$ .*

**Proof:** It is clear that, the first `for` loop statement in line 12 will be executed for  $n_1$  times, where  $n_1$  is the number of elements in  $T$ , including the document element. Also, the second `for` loop statement in line 32 will be executed for  $n_2$  times where  $n_2 = n - 1$ . We have  $n_1 \leq n_2 < n$ . Therefore, all the operations involved in those two `for` loops spend constant amount of time. Hence, it is clear that the *XR* algorithm runs in  $O(n)$  time complexity.

## 5 Creating XML Document From Relational Database

One of the most popular uses of XML is as a data storage facility. Because XML is based on a hierarchical data format, it's suitable to store almost any information, and it has the capability to mimic some of the capabilities found in RDBMSs. However, since XML is stored in a text format, data cannot be retrieved as quickly or stored as efficiently when compared to the binary formats used by modern RDBMSs.

XML is often used to transfer data between disparate systems. A large number of systems today exchange data stored in XML even though they were never originally designed to do so. An example of such a system is an *e-commerce business*. Sometimes, orders may be collected by one company and the merchandise is shipped by another company. In this type of business arrangement, the companies could use XML to exchange order information. Usually, the order data is stored in an underlying database, so the key for this type of transaction is to convert order data stored in a database into an XML document.

The goal of our research is to extract data from a relational database tables and create an XML DOM document corresponding to this data.

### 5.1 RX Algorithm: For Mapping RDB Tables to XML Document

In this section, we describe how to extract data from a database and insert it into XML document, using the *RX* algorithm. We can summarize this process as follows:

- step1: Connects to the database and performs a Select query to get the relational data
- step2: Creating a new XML DOM document  $T$ , in which the data will transfer to it
- step3: The first element we create in the XML document is called the "root" element
- step4: For each *row*: add a new element to the XML document, using the table name, then insert it into the document as a child of the root element.
- step5: Loop through each column in the current *row*, and insert the *field* name, and corresponding value
- step6: Create a new element for the *field* and then insert it as a *child* to the current database row.
- step7: Add the *field* value as a text node, then insert it as a child element to the current field node
- step8: Repeat from Step 4: These loops do not terminate until they have processed every column of every row which has been retrieved from the database
- step9: Returns the completed XML document as a string
- step10: Insert the results into the "`xmlfile.xml`", and display it on the client browser
- step11: End.

The pseudo-code for the *RX* algorithm that used for extracting data from a database and insert it into DOM XML document is described in Figure 7.

#### Algorithm: *RX*

**Input:** Relational database table(s).

**Output:** XML document file, as an XML element DOM tree  $T$ .

```

1. Begin
2. connection = mysql_connect( 'localhost','root', 'hosam' )
3. mysql_select_db( 'company' )
4. table_id = 'table_name'
5. query = "SELECT * FROM table_id"
6. dbresult = mysql_query(query, connection )
7. T = new DomDocument('1.0') // create a new XML document
8. root = T->createElement('root') // create root node
9. root = T->appendChild(root)
10. while(row = mysql_fetch_assoc(dbresult)) { // process one row at a
    time
11.     occ = T->createElement(table_id) // add node for each row
12.     occ = root->appendChild(occ)
13.     foreach (row as fieldname => fieldvalue) { // add a child node
        for each field
14.         child = T->createElement(fieldname)
15.         child = occ->appendChild(child)
16.         value = T->createTextNode(fieldvalue)
17.         value = child->appendChild(value)
18.     } // foreach
19. } // while
20. xml_string = T->saveXML() // get completed xml document
21. echo xml_string
22. End.
```

**Figure 7.** *RX*: A pseudo-code algorithm for extracting data from a database and insert it into XML document

The above code will produce an XML file with the following structure:

```

<?xml version="1.0" ?>
<root>
  <table1>
    <column1> value1 </column1>
    <column2> value2 </column2>
    .....
    <columnn> valuen < columnn>
  </table1>
  .....
  <tablen>
  .....
</tablen>
</root>

```

Note that: we can extend the SQL statement in the *RX* algorithm (line 4 and 5) to include a one-to-many relationship between the database tables, such that the XML string will contain data from two (or many) tables arranged in a one-to-many (or parent-to-child) relationship.

As an example if we replace the SQL statement in the *RX* algorithm (line 4 and 5), by the following statement:

```

select d.DocName, e.ElmName, e.ElmValue, p.Elmname,
       p.Elmvalue
from Document d, Element e, Attribute a, Element p
where d.DocId = e.DocId and
       e.ElmId = a.ElmId and
       e.ElmId = p.ElmId

```

This modification will produce the XML document in Figure 2, with the following structure:

```

<?xml version="1.0" ?>
<root>
  <books>
    <book>
      <author> H.M. Deitel and P.J.Deitel </author>
      <title> C++ How To Program </title>
      <publisher> Prentice Hall Publishing Co. </ publisher>
    </book>
    <book>
      <author>Jack Herrington</author>
      <title>PHP Hacks</title>
      <publisher>0'Reilly</publisher>
    </book>
  </books>
</root>

```

This has the structure `<root>` to `<parent_table>` to `<child_table>`. The `<parent_table>` has child nodes which are its column values as well as multiple occurrences of `<child_table>`.

## 6 Experimental Results

We have implemented the *XR* and *RX* algorithms for data mapping between relational databases and XML documents, and carried out a series of performance experiments in order to check the effectiveness of our approach. In this section, we present the outlines of the implementation and the experimental results.

We applied the data mapping algorithms introduced in this paper to three different XML documents to show the performance of our algorithms on documents with different features. The first one has a DTD with document-centric features, the second one with data-centric features, and the third one hasn't a related XML schema.

We took the first XML document `catalog.xml`, from Xbench project [20] and generate our results. This XML document and its DTD shows data centric features. The second document is `auction.xml`, was taken from XMark project [21] and the test results were generated. The second document and its DTD has document-centric features besides its data-centric features. The third document `books.xml`, is taken from [15] without XML schema.

All experiments were executed on Pentium IV computer with 3.06 GHz processor and 512MB main memory. We ran our implementation using PHP 5.0, MySQL Server 5.0. and Apache Server. We maintained DOM element tree using W3C's DOM specification.

Our performance metric is the time spent to map XML document 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 many times and got the mean value of spent time to obtain more accurate results. Table 1 shows the time spent for data mapping in seconds.

**Table 1.** The time spent for *XR* and *RX* data mapping algorithms

Algorithm	Document	Document Size				
		10 MB	20 MB	30 MB	40 MB	50 MB
<i>XR</i>	<code>auction.xml</code>	6.25	10.90	18.16	70.50	481.55
	<code>catalog.xml</code>	6.06	11.53	17.85	31.08	112.04
	<code>books.xml</code>	5.80	10.75	15.40	29.30	110.44
<i>RX</i>	<code>auction.xml</code>	3.13	5.40	9.08	35.20	240.70
	<code>catalog.xml</code>	3.05	5.75	9.80	15.53	56.02
	<code>books.xml</code>	2.80	5.35	7.60	14.60	55.20

We observe from the above table that, all data mapping for the XML documents make a pick after 40 MB. The DOM 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` and `books.xml` documents on average. We see the difference more precisely especially at 50 MB for both mapping algorithms. The `auction.xml` document includes document-centric textual elements which are nested recursively and cause DOM tree to be deeper where there is no recursive element in `catalog.xml` and `books.xml` documents. Finally, we conclude that, the time spent to insert XML document into database tables (using our proposed relational database schema) is approximately *twice*

the time spent to extracting the same data from a database and insert it into XML document, using the *RX* algorithm.

## 7 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 this paper we described a novel approach of storage and retrieval of XML documents using relational databases. Our approach enables us to store XML documents using a fixed relational schema without any information about XML schema, and DTD. For the processing of XML documents, we proposed *RX* algorithm, for converting XML data to relational data, and *XR* algorithm for extracting data from a database and insert it into XML document. The implementation of the two algorithms does not impose any extension of relational databases for storage and retrieval of XML documents. We show the effectiveness of this approach through several experiments using different XML documents.

## References

- [1] World Wide Web Consortium. 1998. Extensible Markup Language (XML) 1.0. <http://www.w3.org/TR/1998/REC-xml-19980210.W3CRecommendation10-February-1998>.
- [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] 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.
- [4] IBM. *DB2 extender for XML*. <http://www-4.ibm.com/software/data/db2/extenders/xmlext.html>.
- [5] IXIASOFT. *TEXTML Server*. <http://www.ixiasoft.com/textmlserver/>.
- [6] 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.
- [7] Microsoft. *SQL Server XML Support*. <http://msdn.microsoft.com/msdnmag/issues/0300/sql/sql/asp>.
- [8] S. Muench. *Using XML and Relational Database for Internet Applications*. Oracle Corporation. <http://otn.oracle.com/tech/xml/htdocs/relational/paper.html>.
- [9] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. De-Witt, and J. F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *VLDB*, pages 302–314, 1999.
- [10] Software AG. *Tamino XML Server*. Software AG. <http://www.softwareag.com/tamino/>.
- [11] Sonic Software. *eXtensible Information Server*. <http://www.sonicsoftware.com/>.
- [12] 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.
- [13] 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.
- [14] 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.
- [15] XQuery 1.0: An XML Query Language. <http://www.w3.org/XML/Query>. W3C Candidate Recommendation 3 Nov. 2005.
- [16] Florescu, D. And Kossmann, D. 1999a. A performance evaluation of alternative mapping schemes for storing xml data in a relational database. Technical Report 3680 (May), INRIA. <http://rodin.inria.fr/dataFiles/FK99.ps>.
- [17] Florescu, D. And Kossmann, D. 1999b. Storing and querying xml data using an RDBMS. *IEEE Data Engineering Bulletin* 22, 3 (Sept.), 27–34.
- [18] WWW Consortium. Document Object Model 2.0, 2000. <http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/>. W3C Recommendation 13 November, 2000.
- [19] XML Path Language (XPath). <http://www.w3.org/TR/xpath>.

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



**Hosam F. El-Sofany** received his M.Sc. degree in Computer Science from Ain Shams University, Cairo, Egypt. He is currently an Instructor at the Department of Engineering and Computer Science at Qatar University. Since 2004, he is working towards his

Ph.D. in Computer Science from Faculty of Science, Ain Shams University. His research is focused on E-Learning, XML Databases, Relational Databases Systems, and Semantic Web Applications.



**Fayed Ghaleb** received a Ph.D. degree in Mathematics from Moscow State University, Moscow USSR, June, 1978. He is currently a Professor E. at the Mathematics Department, Faculty of Science, Ain Shams Univ., Egypt. His research interests

include Applied Functional Analysis and Spectral Theory, Graph Theory, Data Mining, and E-Learning. Dr Fayed is a board member at the Egyptian Mathematical Science Research Centre, he is also the Treasurer of the Egyptian Mathematical Society (ETMS).



**Professor Samir Abou El-Seoud** received his BSc degree in Physics, Electronics and Mathematics from Cairo University in 1967, his Higher Diplom in Computing from Technical University of Darmstadt (TUD) /Germany in 1975 and his

Doctor of Science from the same University (TUD) in 1979. Professor El-Seoud holds different academic positions at TUD Germany. Latest Full-Professor in 1987. Outside Germany Professor El-Seoud spent different years as a Full-Professor of Computer Science at SQU – Oman and Qatar University and acted as a Head of Computer Science for many years. At industrial institutions, Professor El-Seoud worked as Scientific Advisor and Consultant for the GTZ in Germany and was

responsible for establishing a postgraduate program leading to M.Sc. degree in Computations at Colombo University / Sri-Lanka (2001 – 2003). He also worked as Application Consultant at Automatic Data Processing Inc., Division Network Services in Frankfurt/Germany (1979 – 1980). Professor El-Seoud joined PSUT in 2004.

**Ahmad M. Hasnah:** Associate Professor of Computer Science.

**Jihad Mohamad ALJA'AM:** Full Professor of Computer Science.