MAXDOR: Mapping XML Document into Relational Database

Ibrahim Dweib*, Ayman Awadi and Joan Lu

School of Computing and Engineering, University of Huddersfield, UK

Abstract: The eXtensible Markup Language (XML) is used for representing and exchanging data through the Internet, but this technology needs a suitable medium for storing these data. At present, three common technologies can be used to store and retrieve XML documents, i.e., native XML database, Object oriented Database (OODB) and Relational Database (RDB). This paper describes a general method for mapping XML documents to relational database. The method does not need a DTD or XML schema. It uses global label approach for identifying each token in XML document. Three label are added to each token; parent labels, left sibling and right sibling; to facilitate insertion and update process and makes this cost constant, in contrast of previous approaches that need to relabelled following or descendants tokens. The method can also be used for data-centric and document-centric documents. Experiments on this method show its ability to maintain document structure at a low cost price and building of the original document is straight forward.

Keywords: XML, labelling XML, schema less, relational database.

1. INTRODUCTION

The World Wide Web (WWW) nowadays is one of the important media used by most of the human beings in their daily life activities (i.e.; e-business, e-mail, e-management, e-learning, and e-library). Many enterprises collaborate with other enterprises in long-running read-write workflows through XML-based data exchange technologies such as web services. A large amount of data is needed to be exchanged through the web (i.e. XML format) and stored somewhere as a digital copy.

Storing the huge amount of web services data is an attractive area of research for the researchers and database vendors. But the important issue is how to retrieve and query these data in an efficient manner. At present, three common technologies can be used to store and retrieve XML documents, i.e., native XML database [1, 2] Object Oriented Database [3] and Relational Database [4-9].

The most important factor in choosing the target database is the type of XML documents to be stored, data-centric (e.g., bank transaction, airlines transactions) or document-centric (e.g., emails, books, manual).

The use of XML for data exchanging and representation and Relational Database Management System (RDBMS) for storing and querying together represents a sophisticated hybrid approach to solving most of the data problems (e.g., integrity, multi-user access, retrieving, exchanging, concurrency control, crash recovery, indexing, security, storing semi-structure data, and reliability). Following this track, the key challenges in previous studies with fixed shredding is that there is loss of information from the original XML documents, the reconstruction of the original XML documents is very difficult and the size of generated RDB is huge due to inlining of XML elements on the relational tables.

Existing Mapping techniques from XML-to-relational can generally be classified into two tracks. The first one is the structured-centric technique, which depends on the XML document structure to guide the mapping process [5, 9-13]. The second track is the schema–centric, which makes the use of schema information such as DTD or XML schema to derive an efficient relational storage for XML documents [4, 7, 8, 14-17].

In this research we will focus on a method for mapping XML documents to RDB. The method does not need a DTD or XML schema to simplify the mapping process since many applications deal with highly flexible XML documents from different sources, which make it difficult to define their structure by a fixed schema or a DTD, or sometimes the XML schema or DTD is not available at all. Therefore, it is necessary to look at ways to deal with such XML documents. In this method, a global label method is used to identify each token (i.e., element or attribute) in the document. Three other labels are given to each token, parent label, left sibling label and right sibling label to facilitate future insertion and relocating of a given token, and make insertion cost constant for this process since left and right sibling are just needed to be updated.

The method aims to overcome the challenges faced due to fixed shredding, i.e.;

1) No loss of information while shredding.
2) Reconstruction of original XML documents is easier and faster.
3) Maintaining document structure.
4) Preserve the ordering nature of XML data.
5) Ability to perform semantic search.

The rest of the paper is organized as follows: section 2 discusses related works, section 3 discusses mapping XML documents into relational database method, section 4 shows the system implementation, section 5 presents the experi-

*Address correspondence to this author at the School of Computing and Engineering, University of Huddersfield, UK;
E-mail: ibrahim_thweib@yahoo.com

ment and analysis, section 6 draws the conclusions and future works.

2. RELATED WORKS

There have been a number of different techniques for storing XML documents in a relational database (RDB). These techniques can generally be classified into two tracks. The first one is the structured-centric technique or schema less-centric approach, which depends on the XML document structure to lead the mapping process [5, 9-13]. The second track is the schema–centric approach, which makes use of schema information such as DTD or XML schema to develop a relational storage schema for XML documents [4, 7, 8, 14-17]. Unfortunately, relational storages constructed from schema-centric approach need database reconstruction when there is any change in the XML schema, which is very expensive. Each approach introduced some solutions for the mapping process but failed to solve other.

The aim of mapping XML documents into relational database is to utilize relational database power capabilities in indexes, triggers, data integrity, security, multi-user access, query optimization by SQL query language, and not just for backup. Most studies in this track take care of this issue, and they work to translate users XML queries in XPath expression [18] or W3C's recommendation XQuery expression [19] into SQL queries or statements. Using of XQuery gives the used method more power since XQuery comprises XPath, gives access to multi documents and it is recommended by W3C, while XPath is not. Another issue which mapping XML documents to relational database approaches should also take care of is the ability to reconstruct the stored XML document without loss of information and retrieve it in acceptable time.

Table **1** shows a summary of some works in mapping of XML documents into relational database in both tracks (i.e. schema-less and schema-based), while Table **2** gives a brief comparison between labelling methods for XML tree (i.e. document) nodes. A discussion of most related works is given after that.

One of the issues of mapping XML to RDB is the loss of information due to shredding XML documents and inlining the shreds in RDB tables [4]. To preserve the original XML document information and to solve the problem of the document size limitation, a querying approach for XML documents by dynamic shredding was proposed in [5]. In this approach, the user's involvement is needed to typically first "shred" their documents by isolating what they predict to be meaningful fragments, then store the individual fragments according to some relational schema, and later translate each XML query (expressed in XQuery) to SQL queries

Table 1. A Summary of XML to RDB Related Works

Technique	Schema/ Schema Less	No. of Tables	Cost-Based	Preserve Order	Preserve Constraints	Recursive Consideration	XML Query XPath/XQuery
(Shanmugasundaram *et al.* 1999) [4]	Schema	> 2	yes	no	yes	no	XPath
XRel (Yoshikawa *et al.* 2001) [10]	Schema less	4	No	Yes	No	no	XPath
Dewey (Tatarinov *et al.* 2002) [12]	Schema less	4	no	Yes	No	yes	XPath
XParent (Jiang *et al.* 2002) [11]	Schema less	4	No	Yes	Yes	no	N/A
(Zhang & Tompa, 2004) [5]	Schema less	> 2	no	yes	yes	no	XQuery
ORDPATH (O'Neil *et al.* 2004) [6]	Schema less	2	no	Yes	Yes	No	XPath
ShreX (Yahia *et al.* 2004) [15]	Schema	> 2	No	Yes	No	no	Partial XPath
RELAXML (Knudsen *et al.* 2005) [17]	Schema	> 2	Yes	yes	no	no	N/A
SPIDER (Fujimoto *et al.* 2005) [7]	Schema	4	No	Yes	yes	no	XPath
(Atay *et al.* 2007) [14]	Schema	> 2	yes	yes	yes	yes	XPath
LegoDB & FleXMap (Ramanath, 2006) [20]	Schema	>2	Yes	No	No	yes	XPath
XShreX (Lee *et al.* 2006) [16]	Schema	> 2	yes	Yes	Yes	yes	XPath
(Soltan and Rahgozar, 2006) [13]	Schema less	2	no	Yes	Yes	No	N/A
Oracle interMedia Text, 2006 [21]	Schema less /Schema	1	No	Yes	yes	-	XPath, XQuery
DB2 Text Extender, 2006 [22]	Schema less /Schema	1	No	Yes	No	-	N/A
XTRON (Min *et al.* 2008) [23]	Schema less	6	No	Yes	Yes	No	Partial XQuery

Table 2. A Summary of XML Labelling Methods

Technique	Name	Description	Advantages	Disadvantages
(Li and Moon, 2001) [24]	Interval encoding based on the number of words	It identifies any node by 4 attributes, DocID, StartPos, EndPos and LevelNum. It keeps space for future insertion	Partially solve dynamic update problem	Relabelling of many nodes is needed in case of inserted data size exceed reserved space
(Tatarinov *et al.* 2002) [12]	Global order label	Each node is given a number staring from 1, which identifies the absolute position for a node in the document	It can help in answering XPath queries such as following and following-sibling	All nodes of higher label than inserted node must be relabelled. It is difficult to answer ancestor-descendant relationship
(Tatarinov *et al.* 2002) [12]	Local order label	Each node is given a number that identifies its relative position among its siblings	Only the following siblings of the inserted node need to be relabelled	Just Sibling nodes following inserted node must be relabelled. Maintain parent-child relation is very difficult
(Tatarinov *et al.* 2002) [12]	Dewey order label	It is based on Dewey Decimal Classification. Each node is given a vector that identifies its path from the document's root to the node itself. So, Each part of its path identifies the local order of an ancestor node. An example, 1.2.5: node 5 in level 3 whose parent in level 2 is node #2 and it ancestor the document's root	It is easy to maintain parent-child and ancestor-descendant relation	All sibling nodes right to the inserted node and their descendant must be relabelled
(O'Neil *et al*. 2004) [6]	ORDPATH	It is based on Dewey ordering label, but it keep a gap between the labels for future insertion, odd integer numbers for initial label, and even and negative numbers for future insertion	It provides an ability for nodes insertion without a cost to relabel any existing node. Also it reserved parent-child relation	Many node need to be relabelled after the reserved space is used up. It fails to perform semantic search or path search
(Wu *et al.* 2004) [25]	Prime number labelling	Each node is given a prime number, and its label is the product of self-label from the root to the node	It is easy to identify ancestor-descendant relationship depends on whether their labels are divisible or not. Also insertion of new node and giving it prime number is easy	Large space size since each node label is the product of self-labels from the root to the node
(Soltan and Rahgozar, 2006) [13]	Cluster based order	It is similar to Dewey labelling, but the label is given to a group of sibling elements instead to a label for each element, all sibling nodes are stored in one relational record	It is easy to maintain parent-child and ancestor-descendant relation. Also it decreases the # of records in the table	All sibling cluster right to the inserted cluster and their descendant must be relabelled
(Chung and Yun, 2008) [3]	Dynamic interval-based labelling	It used the ideal of nested tree structure based on the interval based labelling	Parent-child and ancestor-descendant relationship are reserved. It solved partially insertion and updating issue	Some nodes need to be relabelled if no space available at the position of insertion. Querying process becomes high when the label is too long

expressed against the shredded documents. The main idea in this approach is to keep the original document untouched, so there is no need to reconstruct it again. But not saving the XML document in the relational database will make it impossible to connect with the data already existing in the relational database. Also there is a need for query translation for every XQuery query with a support of appropriate structured text operators.

XRel approach [10] and XParent approach [11] are used to store XML documents in RDB. Both approaches used predefined fixed relational schema to store the XML tree information. Figs. (**1** and **2**) shows the relational schemas used in both respectively. In XRel, elements, attributes and text are stored in different tables (element, text and attributes tables), while the fourth table is used as a path table for document paths, where the path is the sequence of elements from the root to the element. In XParent, element table stores each element in the document, and data table stores attributes and text values. While LabelPath table stores all paths and the length of the path, and in DataPath table all parent-child relations are stored.

```
Path (PathID, PathExp)
Element (DocId, PathID, start, End, Index, Reindex)
Text (DocID, PathID, Start, End, Value)
Attribute (DocID, PathID, start, End, Value)
```

Fig. (1). XRel relational Schema.

```
LabelPath (ID, Len, Path)
DataPath(PID, CID)
Element (PathID, DID, Ordinal)
Data (PathID, DID, Ordinal, Value)
```

Fig. (2). XParent relational Schema.

Both approaches, XRel and XParent, assign one code for each element which increases the number of storage records in large XML documents, and increases the number of path joins to process the query.

Labelling or encoding of XML tree (i.e. document) contents is a big issue for researchers and database vendors. Since labelling method plays a main role in facilitating of retrieving and updating the document contents. This issue becomes more important in case of storing document's contents in relational database. Relational database consists of tables of two dimensions forming rows and columns, where rows identify object or person and columns identify this object attributes (i.e. ID, name). Order of these rows and columns are not essential in relational database, while order is necessary for document centric documents such as e-book, e-journal, and email. Many methods and techniques were proposed in to solve this issue. Table **2** gives a brief comparison between these methods. The comparison shows that they are a good ideas presented to enhance data query and retrieval such as [3, 12, 13, 24]. But insertion of new node needs to relabel many nodes in the documents.

Global, Local and Dewey labelling were proposed [12] for labelling XML tree. In Global label, each node is assigned a number that represents the node's absolute position in the document. In this label, dynamic update is very difficult since all the nodes after the inserted node need to be relabelled and extracting the parent-child and ancestor-descendant relationship are also impossible. In Local label, each node is assigned a number that represents its relative position among its siblings. In this label, a combination of a node's position with that of its ancestors as a path vector identifies the absolute position of the node within the document. An update in Local label has less overhead than Global label because only the following siblings of the new node need to be renumbered. But extracting the parent-child and ancestor-descendant relationships is still very difficult. While in Dewey order label, each node is given a label based on Dewey Decimal Classification. Each node is given a vector that identifies its path from the document's root to the node itself. So, each part of its path identifies the local order of an ancestor node. An example, label 1.2.5 means that: node 5 in level 3 whose parent in level 2 is node #2 and it ancestor the document's root. By using this way, it gives an easy way to extract node labels from its ancestors. But in case of inserting new node, all sibling nodes right to the inserted node and their descendant must be relabelled.

Pre-order label and node size together are used to identify the node [24]. It uses 4 attributes to identify any node, DocID, StartPos, EndPos and LevelNum. It keeps space for future insertion. An arbitrary integer large than the node size is considered for future insertion. Parent-child and ancestor-descendant relationship are achieved, and insertion issue is solved partially. But, many nodes need to be relabelled when the data size exceeds the reserved space.

ORDPATH, a hierarchical labelling schema implemented in Microsoft SQL Server 2005, was introduced [6]. It is used to label nodes of an XML tree without requiring a schema. It used two tables to store XML data. Fig. (**3**) shows ORDPATH relational schema.

```
Node (OrdPathCode, Tag, NodeType, Value, PathID)
Path (PathID, PathExp)
```

Fig. (3). ORDPATH relational Schema [6].

In contrast of Dewey Labelling method, which suffers from the problem of dynamic updating after the insertion of new node, i.e. many nodes should be relabelled. ORDPATH can support insertion of new nodes at arbitrary positions in the XML tree without updating the labels of old nodes since it only used positive odd integers to be assigned to nodes during initial loading and reserved even-numbered and negative integer values for later insertions into the existing tree. Fig. (**8**) shows ORDPATH labelling for an XML document. The advantages of ORDPATH label are no overhead deserved for updates and it reserves the structure of XML document. But, it fails to perform semantic search or path search.

Prime number labelling [25] is another way for labelling XML node. Each node in the document is given a prime number, and its label is the product of self-label from the root to the node itself. It is easy to identify ancestor-descendant relationship depends on whether their labels are

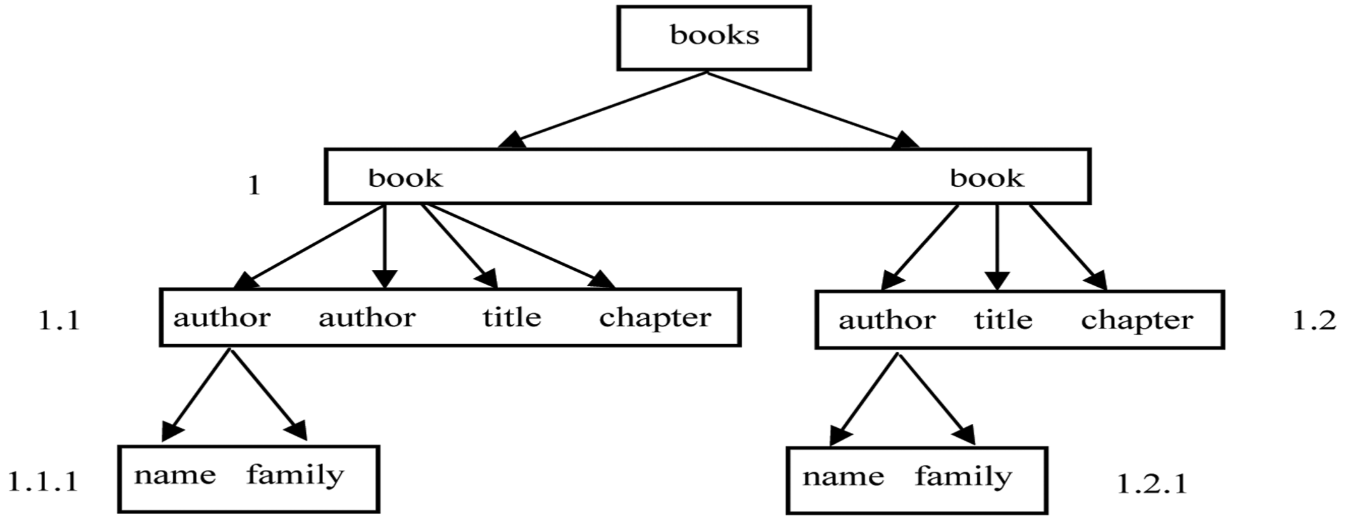


Fig. (4). Clustered labels for XML Tree (Soltan and Rahgozar, 2006) [13].

divisible or not. Also insertion of new node is possible and giving it prime number is easy. But, large space size is created since each node label is the product of self-labels from the root to the node itself.

Variable Length Endless Insertable (VLEI Code) [26] makes use of Dewey order method to construct the node label as a sequence of bits separated by ".", or octal number separated by "9". For example 1.1.10, its parent is 1.1 and ancestor 1. VLEI code suppose that 10<1<11<110<11<111. Or 19397 as a node label, means that 193 is its parent and 1 is its ancestor. Parent-child and ancestor-descendant relationship are reserved by this method. It reduced insertion cost since relabelling it not needed. Using octal number with "9" delimiter reduce the space needed for labelling. Using octal and "9" delimiter instead of "." as character reduce the space but increase the time for relabelling since it as Dewey without space between label for future insertion.

A clustering-based scheme for labelling XML trees was proposed in [13]. In this scheme, a group of elements is labelled instead of a single element. Elements are separated into various groups, putting all sibling elements in one group, and assigning one label to this group instead of one label to each element and stored them in one relational record. Figs. (**4** and **5**) show clustered labelling method for an XML tree and the relational schema for it respectively.

```
Node (ClusteredCode, Tags, NodeType, Value, PathID)
Path (PathID, PathExp)
```

Fig. (5). Clustered relational Schema [13].

A clustering-based scheme will reduce the size of the database needed to store the XML tree by reducing the number of records generated from the mapping process, since it uses one label for a group of elements (a cluster) which is stored in one relational record, in contrast of other labelling methods that need a label for each node. Also, it reduces the number of path joins needed to process the query, and makes the reconstruction of XML document from RDMB faster. But this method suffers from the problem of dynamic updating after the insertion of new node, i.e. many nodes should be relabelled. And also, it fails to perform semantic search or path search.

Dynamic interval-based labelling [3] used the ideal of nested tree structure based on the interval based labelling as in [24]. Parent-child and ancestor-descendant relationship are reserved. It also solved partially insertion and updating issue. But, still some nodes need to be relabelled if no space available at the position of insertion. Also, extra space is needed for identifying each element and querying process becomes high when the label is too long.

Schema less Approach [9] used a global label method to label XML document contents, and used string field to reserve document structure. This approach is efficient in reconstructing and retrieving parts of XML document for small size document. Since document structure is stored in text field, and sequential search is done for reconstructing and retrieving part of the document. But, the performance of this approach decreases if the size of the document becomes larger.

3. MAPPING XML DOCUMENTS INTO RELATIONAL DATABASE METHOD

1. The goals of the method applied in this paper are:
2. Utilize the advantages of XML in representing and exchanging data, and relational database in querying, security, multi-user access, data integrity.
3. Maintain document structure with no loss of information while shredding.

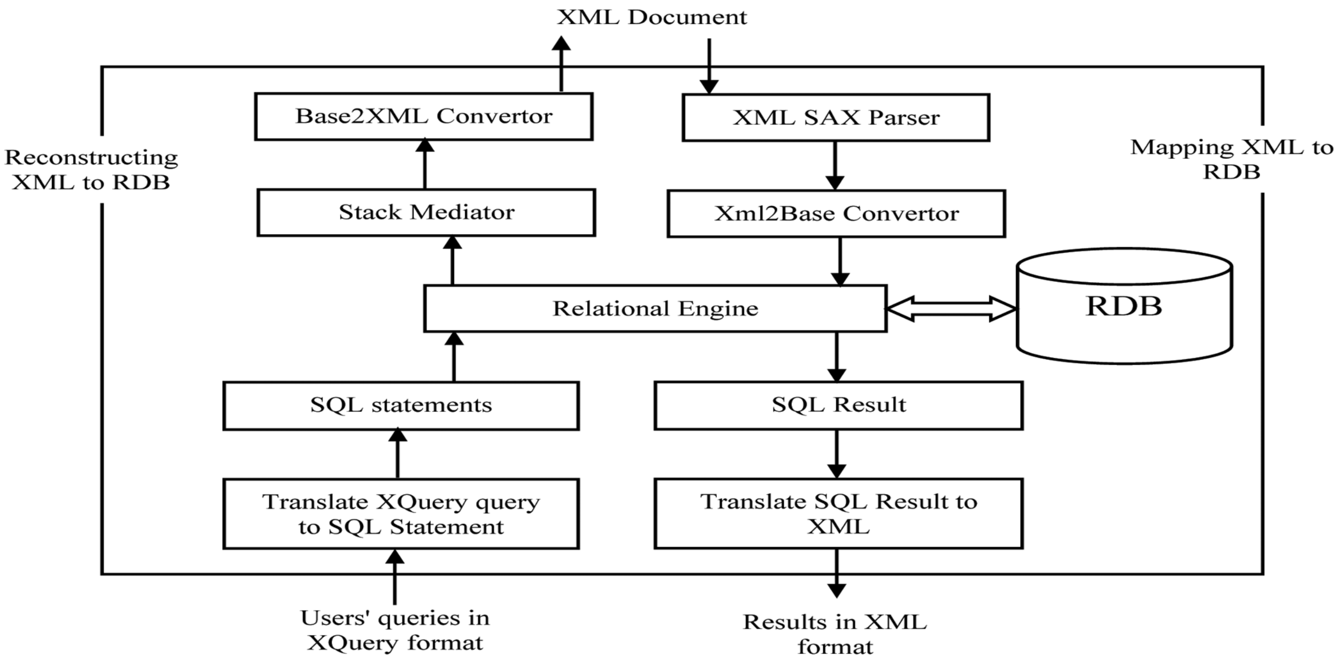


Fig. (6). MAXDOR architecture.

4. Ease of process, transforming a fresh document should be an easy task, and updating an already transformed document should also be apple with constant cost.
5. Ability to reconstruct the XML document from relational database without loss of information.
6. Ability to perform semantic search.

3.1. MAXDOR Architecture

Fig. (**6**) shows MAXDOR system architecture, the system consists of four parts. Part one maps XML document into RDB, part two reconstruct XML document from RDB, part three translate users XQuery queries into SQL statements, and part four translate SQL statements result into XML format.

In part one, the system loads the XML document and parses it by XML SAX parser shreds the document content into tokens, and stores these tokens into predefined relational schema, more details in the relational schema is given in section 3.3.

While part two of the system goes through the relational tables and reconstructs the XML document. It gives the facility for the user to insert, delete, and update the content of the document and store it again to the database.

In part three, the user XQuery queries are translated to SQL statements and fired against the database engine to get the results. And these results are translated from relational table format to XML format hierarchical format and return back to the user. Full details of MAXDOR system contents are given in the following sections.

3.2. Theory Guidance

The main mathematical concepts that are used in this method are presented in this section.

Definition 1: Composite Relation: If f is a parent-child relation between X and Y as

f: $X \rightarrow Y$ and g is a parent-child relation between Y and Z as g: $X \rightarrow Y$. Then we can say that h: $g \circ h$ is ancestor-descendant relation between X and Z as

h: $X \rightarrow Z$, [27]. Fig. (**7**) shows this relation.

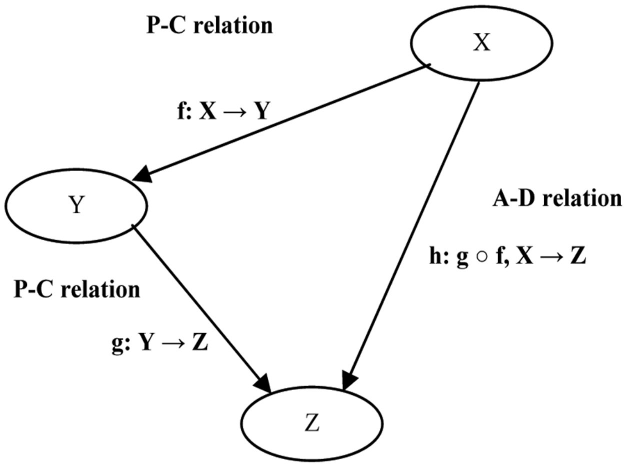


Fig. (7). Composite parent-child relations.

Definition 2: Associative Relation: If f is a parent-child relation between X and Y as

f: $X \rightarrow Y$, g is a parent-child relation between Y and Z as g: $X \rightarrow Y$, and h: is a parent-child relation between Z and W as h: $Z \rightarrow W$. Then i: $g \circ f$ is ancestor-descendant relation between X and Z, j: $h \circ g$ is ancestor-descendant relation between Y and W, and K: $(h \circ g) \circ f = h \circ (g \circ f)$ is also ancestor relation between X and W, [27]. Fig. (**8**) shows this relation.

Definition 3: XML tree is composed of many subtrees of different levels; it can be defined as the following [14]:

$T = \sum_{i=1}^{n} (E_i, A_i, X_i, r_{i-1})$; i=1, 2 … n, represent the levels of XML tree, 0 represents the root.

Where:

E_i is a finite set of elements in the level i.

A_i is a finite set of attributes in the level i.

X_i is a finite set of texts in the level i.

r_{i-1} is the root of the subtree of level i.

Definition 4: A dynamic fragment (shred) $df(i)$ is defined to be the attributes and texts (child leaves) of the subtree i of the XML tree plus its root r_{i-1}, as follows:

$df(i) = (A_i, X_i, r_{i-1})$

Where:

A_i is a finite set of attributes in the level i.

X_i is a finite set of texts in the level i.

r_{i-1} is the root of the subtree of level i.

Definition 5: The root of the fragment (shred) is the node which has an out-degree more than one.

3.3. Design Framework

A Four dimensional labels (FDLS) is used to label the XML document contents. FDLS uses a global label approach to give a label to the XML elements and attributes. The label is unique for each element or attributes. But it is not required to be in a sequence as in [12, 13]. An initial pre-order traversing for the XML document is applied. No re-labelling for XML document tokens (i.e. elements and attributes) are needed if new element or subtree is added in contrast with [12, 13, 28]; all labels following the inserted token or tokens need to be relabelled. In FDLS, each node is assigned four labels, as follows:

Node (NodeID, left-sibling, parent, right-sibling).

- NodeID is a unique label given to identify each node.
- LS (Left-sibling) label is the node's preceding sibling NodeID.
- PID (Parent) label is the parent NodeID.
- RS (Right-sibling) label is the node's following sibling NodeID.

A fixed relational schema consists of two tables. The "documents" table keeps the required information of the

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[rr, bend right, "g \circ f"'] \arrow[rrr, bend right=40, "(h \circ g) \circ f = h \circ (g \circ f)"'] & Y \arrow[r, "g"] \arrow[rr, bend left, "h \circ g"] & Z \arrow[r, "h"] & W
\end{tikzcd}$$

Fig. (8). Associative ancestor-descendant relations.

XML documents structure, and "tokens" table keeps the detailed contents of the XML documents. The following subsections give more details about the approach.

SAX (Simple Application Interface for XML) parser will be used instead of DOM (Document Object Model) to solve the issue of large XML document size. Since SAX parses XML document information as a sequence of events in contrast of DOM which needs to represent the whole document as tree in the memory first and then parses it.

The transformation methodology should satisfy many requirements; the significance of each requirement is to a certain level application dependant. In some applications it is extremely important to maintain order of nodes as in document –centric documents while in others as in data-centric documents, order is insignificant. Among these requirements that should be met, are the follows:

1. Maintain document structure.
2. Transform a fresh document should be an easy task, and update an already transformed document should be done with a constant cost.
3. Ability to reconstruct the XML document from relational database.
4. Ability to perform semantic search.

The method is an enhancing for our previous work [9], but it is different in that it uses left-sibling, right-sibling and parent labels to reserve the document structure and tokens order in XML document while the previous one uses a string field in the document table to reserved the document structure. This update helps to solve the big text field issue for large XML document.

3.3.1. FDLS Relational Schema

A description for the relational schema to be used in FDLS is given bellow:

1. A master table for documents is needed. It is called "documents", this table will keep information about documents themselves, at minimum it will have the following structure:

Documents(documentID, documentName, docElement, totalTokens, schemaInfo)

Additional fields may be added to keep all information about the document itself such as date created, statistics, types … etc.

a. DocID is a unique id generated per document to identify documents.

b. DocName is the external name for XML document.

c. DocElement represents the document's root.

d. totalTokens represents the number of elements and attributes in the document (i.e. number of token). It helps in future insertion. Since new inserted node is given new number following the last token number in the document.

e. schemaInfo keeps the document's schema information if exist for documentation purpose.

2. A second table to store the actual contents for all documents is also needed. Documents will be shredded into pieces of data that will be called "tokens", each document element, tag, or property will be considered a token, the tokens table will the following structure:

Tokens(documentId, tokenId, LS, Par, Rs, tokenLevel, tokenName, tokenValue, tokenType).

a. TokenId field is the primary generated id for each token.

b. DocumentId is the foreign key linking the tokens table to the documents table.

c. LS (left-sibling) field keeps the id of the left sibling token of current node. It is used to preserve the document's structure and tokens' order.

d. Par keeps the id of node's parent. It is used to reserve parent-child and ancestor descendant relations.

e. RS (Right-sibling) field keeps the id of the right sibling token of current node. It is used to preserve the document's structure and tokens' order.

f. tokenLevel reserved the token level in the document or tree. It is starting from 0 for document element.

g. TokenName is the tag name or the property name as found in the original XML document.

h. TokenValue is the text value of the XML tag property.

i. TokenType is used to differentiate between elements and attributes. (1 = element, 2 = attribute).

So, the relational schema for this method has two tables as shown in Fig. (**9**).

- Documents(documentID, documentName, docElement, totalTokens, schemaInfo)
- Tokens(documentID, tokenID, LS, Par, RS, tokenLevel, tokenName, tokenValue, tokenType)

Fig. (9). Relational Schema.

3.3.2. Insertion of New Token or Subtree

Insertion of new token or subtree in any location or level in the XML tree (i.e. document) can be done with constant cost. This insertion follows the following rules:

a. Insertion of a new token between two siblings:

1) The new token T takes a label tokenID following to the last token in the document.

2) RS(T) = RS(PrecT)

3) LS(T) = RS(FolT)

4) RS(PrecT) = tokenID

5) LS(FolT) = tokenID

6) Par(T) = Par(FolT)= Par(PrecT)

b. Insertion of a new token to the left of a subtree:

1) The new token T takes a label tokenID following to the last token in the document.

2) RS(T) = FolT

3) LS(T) = Null

4) LS(FolT) = tokenID

5) Par(T) = Par(FolT)

c. Insertion of a new token to the right of a subtree:

1) The new token T takes a label tokenID following to the last token in the document.

2) LS(T) = PrecT

3) RS(T) = Null

4) RS(PrecT) = tokenID

5) Par(T) = Par(PrecT)

d. Insertion of a new token as a parent subtree:

1) The new token T takes a label tokenID following to the last token in the document.

2) Par(T) = Par(childT)

3) Par(childT) = TokenID

4) RS(T) = LS(T) = Null

3.3.3. Deletion of a Token or Subtree

Deletion of existing token or subtree in any location or level in the XML document can be done also with constant cost. This deletion follows the following rules:

a. Deletion of a token between two siblings:

1) RS(PrecT) = RS(T)

2) LS(FolT) = LS(T)

3) The TotalTokens content will not change (i.e. not decremented). Since no relabelling for the tokens within the document will done.

b. Deletion of a token from the left of a subtree:

1) LS(FolT) = Null.

2) The TotalTokens will not change (i.e. not decremented). Since no relabelling for the tokens within the document will done.

c. Deletion of a token from the right of a subtree:

1) RS(PrecT) = Null.

2) The TotalTokens will not change (i.e. not decremented). Since no relabelling for the tokens within the document will done.

d. Deletion of a subtree:

Deletion of a subtree can be handled as a single token by one of the previous three cases.

3.3.4. Re-Allocating or Moving Simple or Complex Element within XML Document

a. Re-allocating or moving simple element within XML document can be handled as follows:

1) Deleting it's pointers from original location as in section 4.3.1.2.5.

2) Inserting this element in its new location as in section 4.3.1.2.4. But, there is no need to assign new TokenID for this element and its follows if it has attributes and childs, since it owned ones before.

b. Re-allocating or moving complex element (i.e. subtree) within XML document can be handled as moving of simple element since just the pointers of the subtree root is managed and there is no change on other contents of the subtree. That's mean the cost of moving a complex element is the same the cost of moving a simple element since there are no assignment of new IDs for complex element contents and no relabeling is needed.

3.3.5. Mapping XML to RDB Algorithm

The data model used for the mapping algorithm uses the W3C's Simple Application Program Interface for XML (SAX parsing); it also uses a variable data structure array to traverse the XML document by pushing the children of each node onto stack in order to reserve and identify nodes order and parent child relationship. SAX fires actions on a lot of events, i.e. document start, document end, element start, element end, characters, element attributes, and processing instruction. These events help in striping XML document into relational database. Each token; element or attribute; is given a unique general ID that identify this token. Three more labels are added to token description, its parent ID, left sibling ID, and right sibling ID. Left and right sibling IDs are given to make the time needed for future insertion in the document constant since these IDs are needed to be updated if new node or sub tree is added or relocating in the document.

3.3.6. Reconstructing XML Document from RDB Algorithm

The reconstructing algorithm uses SAX writer for building XML documents from relational database. SAX writer makes building of XML document easier since it contains a lot of properties that can help in creating XML document, such as documentStart, documentEnd, elementStart, elementEnd, and attributes properties. Fig. (**10**) shows base2XML algorithm with XML document id and relational database tables as input and XML document as output.

In Fig. (**10**), line 4 initiates the document, lines 4 and 5 open and XML file as an output. The loop, from line 10 to line 50 is used to build the document. Lines 11 to 13 reserve element contents in an array for future comparison, in order to write the element's end tag of element child or sibling. Lines 14 to 23 checks if the element is not a document ele-

```
1. base2XML Algorithm
2. Input: DocID.
3. Output: XML Document.
4. saxwrt.startDocument
5. iHandle = FreeFile
6. Open App.Path & "\rebuilt\" & lDocId & ".xml" For Output As iHandle
7. ' Loop and build
8. bEnd = False
9. mRow = 0
10. Do While True
11. pElement(mRow).TokenName = rsTokens.TheDataSource!TokenName
12. pElement(mRow).TokenValue = rsTokens.TheDataSource!TokenValue
13. pElement(mRow).TokenLevel = rsTokens.TheDataSource!TokenLevel
14. If mRow > 0 Then
15. Do While pElement(mRow).TokenLevel <= pElement(mRow - 1)
16. saxwrt.endElement "", "", pElement(mRow - 1).TokenName
17. ' move down
18. pElement(mRow - 1).TokenName = pElement(mRow).TokenName
19. pElement(mRow - 1).TokenValue = pElement(mRow).TokenValue
20. pElement(mRow - 1).TokenLevel = pElement(mRow).TokenLevel
21. mRow = mRow - 1
22. Loop
23. End If
24. ' Checks for element attributes
25. Do While True
26. If Not rsTokens.FindRow("PrevToken = " & rsTokens.TheDataSource!TokenId) Then
27. ' End of document
28. bEnd = True
29. Exit Do
30. End If
31. If rsTokens.TheDataSource!TokenType = TokenTypes.xbTag Then
32. ' Go to outer-Loop and continue the build process
33. Exit Do
34. End If
35. atrib.addAttribute "", "", rsTokens.TheDataSource!TokenName, "", rsTokens.TheDataSource!TokenValue
36. Loop
37. saxwrt.startElement "", "", pElement(mRow).TokenName, atrib
38. saxwrt.characters pElement(mRow).TokenValue ' writing element value
39. ' clear atrib object
40. atrib.Clear
41. mRow = mRow + 1
42. ' If end of doc
43. If bEnd Then
44. Do While mRow > 0 ' close tags for root
45. saxwrt.endElement "", "", pElement(mRow - 1).TokenName
46. mRow = mRow - 1
47. Loop
48. Exit Do
49. End If
50. Loop
51. saxwrt.endDocument
52. Print #iHandle, wrt.output
53. Close #iHandle
```

Fig. (10). Mapping XML to relational data algorithm.

ment, and the loop writes all the end tag of sibling element and its last child. Lines 25 to 36 check for element attributes if any, and check for end of document to terminate the loop. Line 37 writes element starting tag and element attributes if excites. Line 38 writes element value if it has one. Where line 40 clear attributes of current element. Lines 43 to 49 check if end of document is achieved, end tags of remaining elements in the array including document element are written to the document. Line 52 writes the output to the free file on the disk. While line 53 closes the file.

3.3.7. Theory Implementation on Simple Case Study

In this subsection, an example is given to illustrate the application of the mapping method described in Subsection 3.3.3. Consider the XML document in Fig. (**11**).

```
<books>
 <book id="11210" >
 <author id="a1" >M. John</author>
 <name>CS 101</name>
 </book>
 <book id="11211">
 <subject>Math</subject >
 <name> Math 102</name>
 </book>
</books>
```

Fig. (11). XML document.

Any XML document can be represented as a rooted, labelled Tree. Fig. (**12**) presents an XML tree for the XML document in Fig. (**11**). In this method, each node is given a generated label in pre-order traversal. This label is unique since it identifies each token in the document. The token represents an element or attribute.

After transformation, this document will be represented by a single record in the documents table with documentId for example = 1, as in Fig. (**13**). And the tokens table will be containing the records for the document contents as shown in Fig. (**14**).

4. SYSTEM IMPLEMENTATION

To test the performance of the proposed technique, a system is implemented of four main components, as shown in Fig. **6**. 1) Mapping XML documents into relational database. 2) Constructing XML documents from relational database. 3) Translating users XQuery queries into SQL statements. 4) Translating SQL statements result into XML format. Fig. (**15**) shows the GUI of MAXDOR. By clicking the new icon from the tool bar, the user can create new XML document and assign a name for it. Then, to select the XML document from a local storage to be mapped to relational database, the user can click mapping icon from the tools bar as show in Fig. (**16**). For reconstruct the desired document from relational database, user can select the document to be constructed from the list of documents and click on reconstructing icon.

5. EXPERIMENT AND ANALYSIS

In this section, a comparison of the performance of the MAXDOR system with other mapping methods [9] will be performed. The performance metrics to be measured are the time needed to:

1. Mapping XML document to relational database with scalability.
2. Reconstruct XML document from relational database.

5.1. Experiment Environment

An Intel Core2Duo 2GHz CPU, 1GB RAMS and 256MB shared Cache and running Windows Vista is used for the experimental tests. Visual Basic 6 is used as software development kit with Microsoft Access 2003 as target relational database. Six XML documents with different sizes are used in the experiment. The experiment is repeated five times and the mean value of those times is reported to obtain a realistic and accurate result. The data is taken from the XML data repository that are available at the web site of the School of Computer Science and Engineering, University of Washington [29].

5.2. Performance Analysis of Various Mapping Methods

5.2.1. Mapping XML Document to Relational Database

MAXDOR performance is compared with schema less approach [9], which is named as string field, it uses string field to reserve document structure. The experiment is done in two faces as follows:

a. Face 1, scalability test: supplier document from [29] was taken and its size is update to be 300K, 600KB, 900KB, 1200KB, 1500KB and 1800KB. In this experiment, MAXDOR shows good performance in linear and scalable manner in comparison with string field method as document size is increasing. The mapping performance over different sizes of the same document is shown in Fig. (**17**), i.e. documents of the same complexity and levels.

b. Face 2, effectiveness test: five documents of the same size (300KB) but have different structure and different numbers of elements are taken in this experiment. Table **3** shows these documents properties and there mapping and reconstructing time for both methods. In Fig. (**18**), MAXDOR has better performance than String field, but the time needed for mapping process increases for both methods when the number of elements becomes larger.

5.2.2. Reconstructing XML Document from Relational Database

Also, the experiment is done in three faces as follows:

a. Face 1, scalability test: the supplier documents mapped before will be constructed in this experiment to see that there is no loss of information and scalability of MAXDOR in reconstructing XML document from relational database. Fig. (**19**) shows that both methods are closed for small documents sizes and the gap increase when the document size becomes larger. MAXDOR gives better results than String field since the later uses the string field becomes larger for large documents.

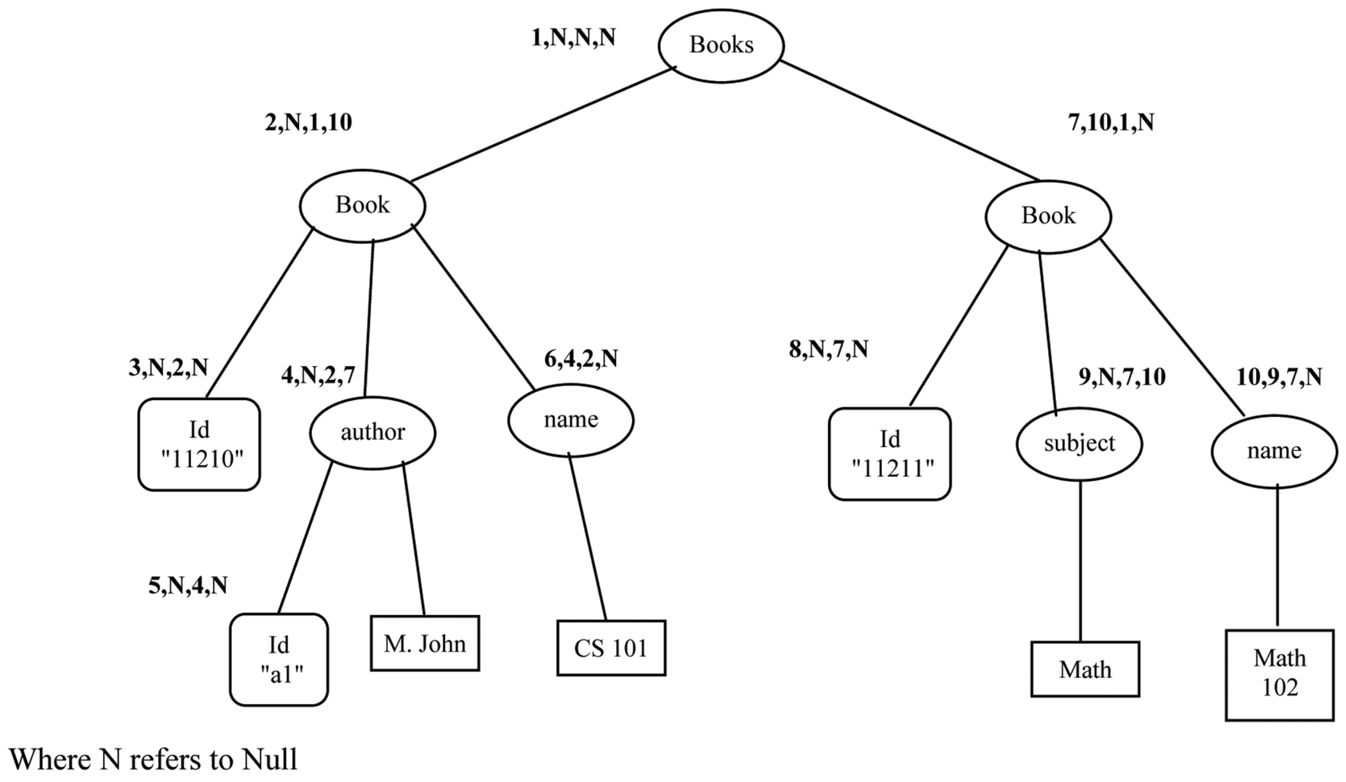


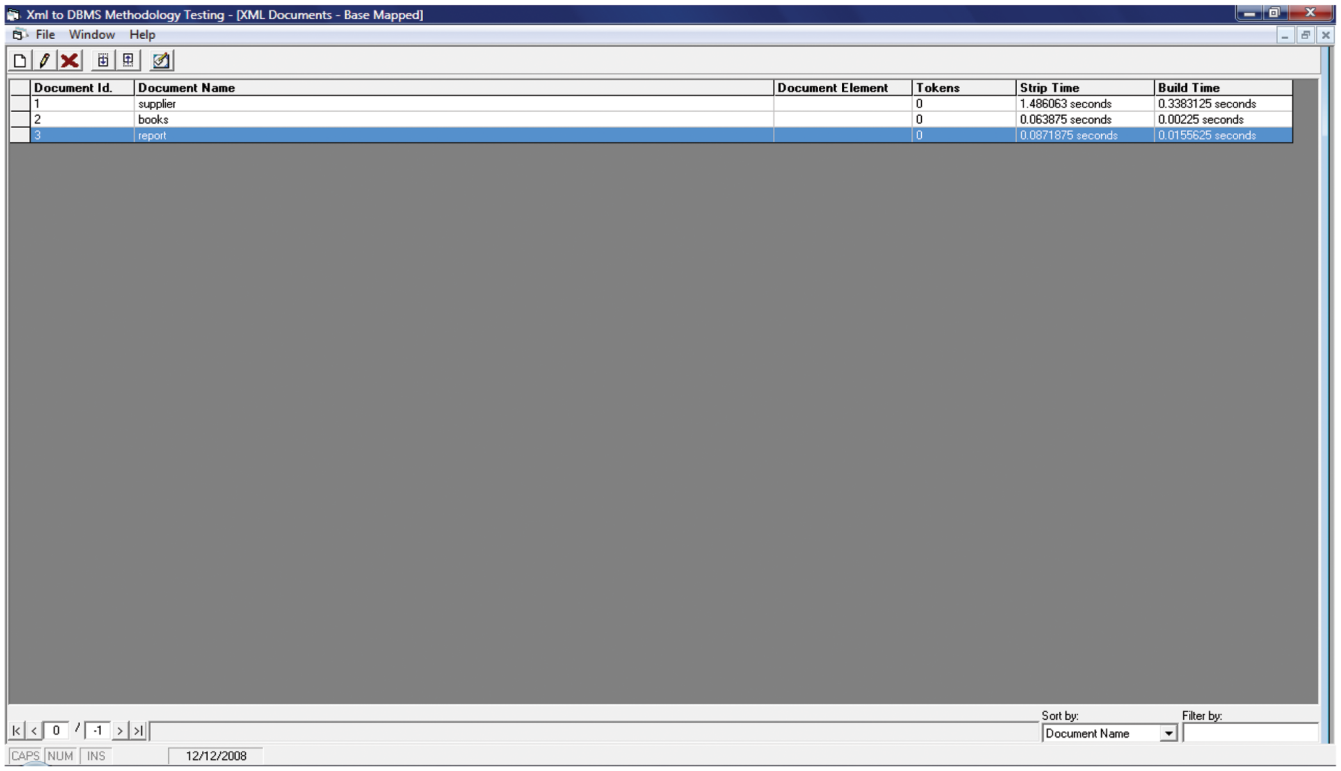
Fig. (12). A tree representation for XML document.

Documents			
documentId	documentName	docElement	totalTokens
1	Biography	Books	10

Fig. (13). Documents table.

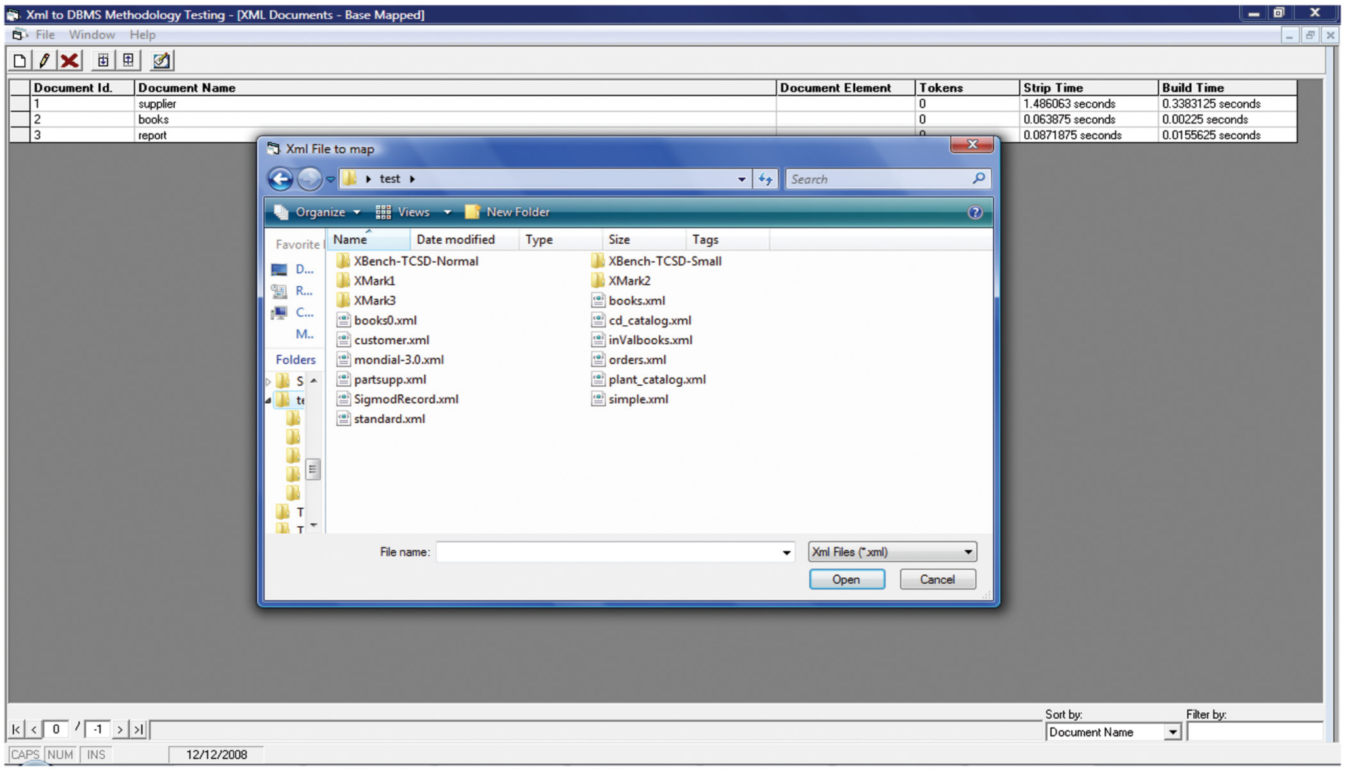
Token								
Document Id	**Token Id**	**LS**	**Par**	**Rs**	**Token Level**	**Token Name**	**Token Value**	**Token Type**
1	1	N	N	N	0	books	Null	1
1	2	N	1	7	1	book	Null	1
1	3	N	2	N	2	id	11210	2
1	4	N	2	6	2	author	M. John	1
1	5	N	4	N	3	id	a1	2
1	6	4	2	N	2	name	CS101	1
1	7	2	1	N	1	book	Null	1
1	8	N	7	N	2	id	11211	2
1	9	N	7	10	2	subject	Math	1
1	10	9	7	N	2	name	Math 102	1

Fig. (14). Tokens table.



Document Id.	Document Name	Document Element	Tokens	Strip Time	Build Time
1	supplier		0	1.486063 seconds	0.3383125 seconds
2	books		0	0.063875 seconds	0.00225 seconds
3	report		0	0.0871875 seconds	0.0155625 seconds

Fig. (15). The GUI of MAXDOR.



Document Id.	Document Name	Document Element	Tokens	Strip Time	Build Time
1	supplier		0	1.486063 seconds	0.3383125 seconds
2	books		0	0.063875 seconds	0.00225 seconds
3	report		0	0.0871875 seconds	0.0155625 seconds

XBench-TCSD-Normal	XBench-TCSD-Small
XMark1	XMark2
XMark3	books.xml
books0.xml	cd_catalog.xml
customer.xml	inValbooks.xml
mondial-3.0.xml	orders.xml
partsupp.xml	plant_catalog.xml
SigmodRecord.xml	simple.xml
standard.xml	

Fig. (16). Select XML document to be mapped.

Table 3. Mapping XML Document to Relational Database, Documents of Equal Size, but of Different Number of Elements (Time in Seconds)

No of Element	String Field	MAXDOR
113	0.0308125	0.2491875
8030	38.03191	13.96078
8402	44.095	14.3305
9952	60.02904	17.19003
10891	69.2325	18.70478

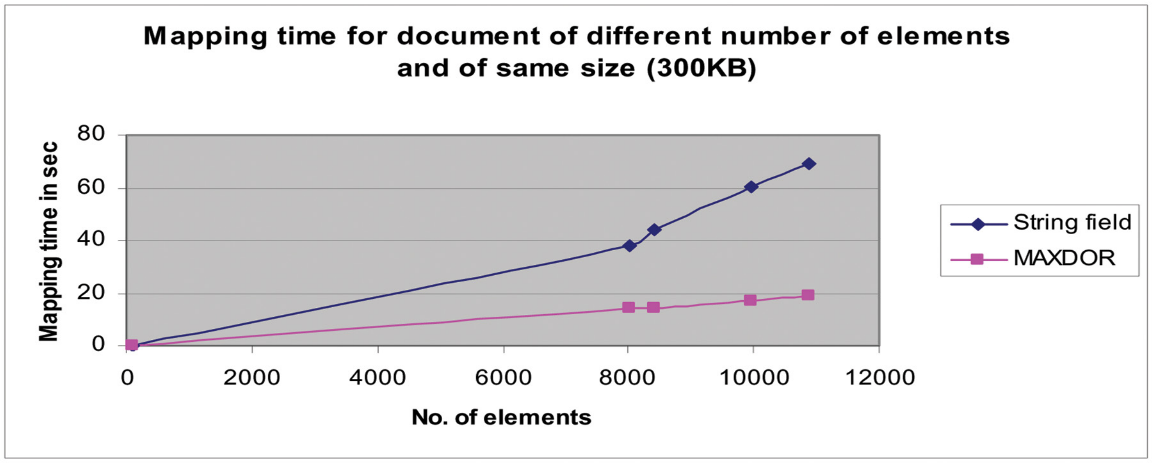


Fig. (17). Mapping time for both MAXDOR and String field method over different sizes of the same document.

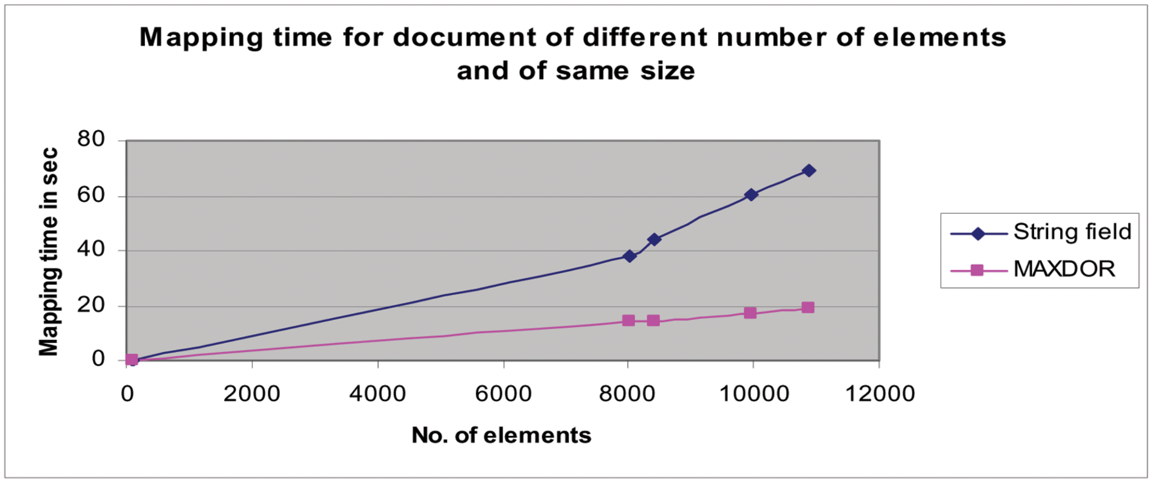


Fig. (18). Mapping time for both MAXDOR and String field method over documents of same size but of different number of elements.

b. Face 2, effectiveness test: in this face reconstructing of document-centric document and data-centric document are done to see the ability of MAXDOR to with different XML document types. Fig. (**20**) show that MAXDOR has better performance than String field for larger number of elements since the later one uses string field to reserve document structure and this field becomes larger for documents of larger number of elements.

6. CONCLUSION AND FUTURE RESEARCH

In this paper, MAXDOR approach is presented. It is a schema less approach for mapping XML documents to relational database (i.e. no need for XML schema or DTD). MAXDOR extracts structural information efficiently and makes use of it for the mapping process.

In MAXDOR, the content of XML documents is stripped in to tokens as elements and element attributes. A global

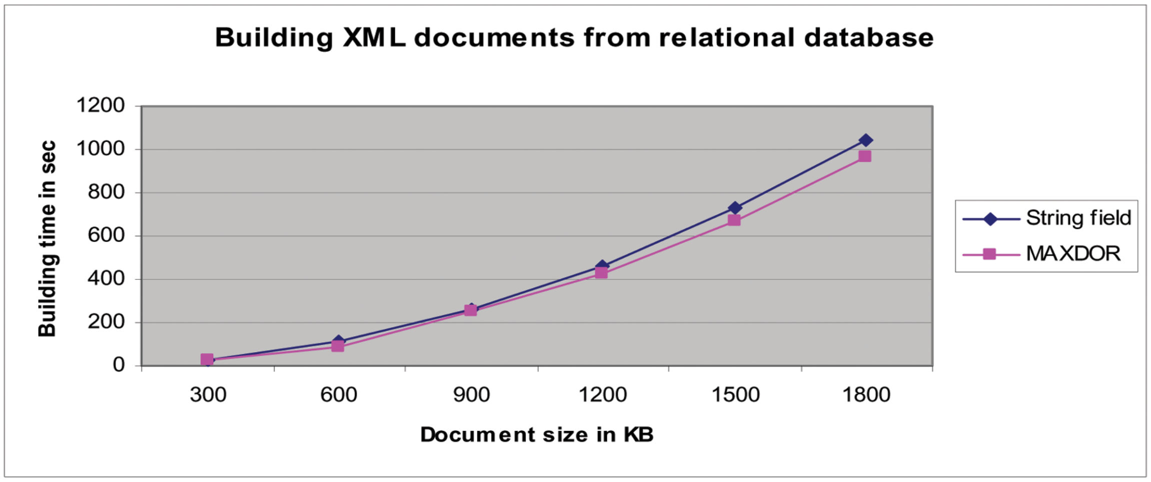


Fig. (19). Building time for both MAXDOR and String field method over different sizes of the same document.

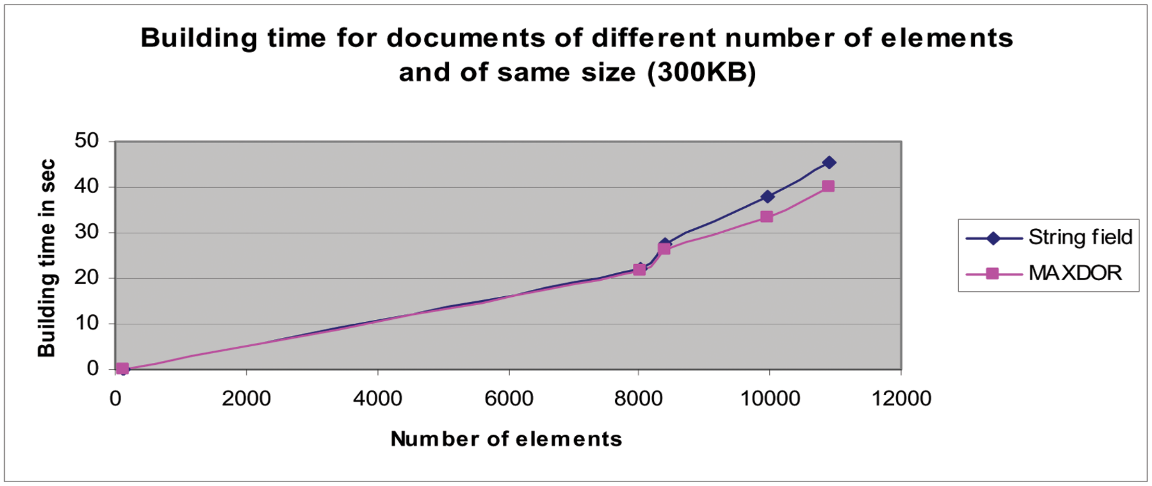


Fig. (20). Building time for both MAXDOR and String field method over documents of same size but of different number of elements.

label method is used to label this content. Each token has a unique ID, and another three labels are added to the token information that are parent ID, left sibling ID and right sibling ID. These labels are used to reserve the document structure, token order, and parent-child and ancestor-descendant relationship. Also using of left sibling and right sibling makes insertion or relocating cost of elements or attributes constant in MAXOR since the right sibling of previous token and left sibling follow token are needed to be updated. This is an advantage for MAXDOR since in other approaches there is a need to relabel all following token or all ancestors token.

An implementation for MAXDOR is conducted and an intensive experimental study for both real-life and synthetic data sets. The experimental results show that MAXDOR gives acceptable results for both mapping and reconstructing of XML documents to and from relational database when compared with other methods.

Currently, two faces of MAXDOR are being implemented that are query and update faces. After that, experiments will be done on these faces to show MAXDOR performance. Also, comparisons with other approaches in the literature will be done to see MAXDOR performance.

REFERENCES

[1] H. V. Jagadish, S. Al-khalifa, A. Chapman, L. V. S. Lakshmanan, A. Nierman, S. Paparizos, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu, "TIMBER: A Native XML Database," in *SIGMOD*, San Diego, CA, 2003.

[2] A. Fomichev, M. Grinev, and S. Kuznetsov, "Sedna: A Native XML DBMS," in *SIGMOD*, San Diego, CA, 2003.

[3] S. M. Chung, and S. B. Jesurajaiah, "Schemaless XML document management in object-oriented databases," in the *International Conference on Information Technology: Coding and Computing, ITCC 2005*, 2005, pp. 261-266.

[4] 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*, 1999, pp. 302-314.

[5] H. Zhang, and F. W. Tompa, "Querying XML Documents by Dynamic Shredding," in *DocEng'04*, Milwaukee: Wisconsin: USA, 2004.

[6] O. N. Patrick, O. N. Elizabeth, P. Shankar, C. Istvan, S. Gideon, and W. Nigel, "ORDPATHs: insert-friendly XML node labels," in *Proc. of the 2004 ACM SIGMOD international conference on Management of data*, ACM: Paris, France 2004.

[7] K. Fujimoto, T. Yoshikawa, D. D. Kha, M. Yashikawa, and T. Amagasa, "A Mapping Scheme of XML Documents into Relational Databases Using Schema-based Path Identifiers," in *International Workshop on Challenges in Web Information and Integration, (WIRI'05)*, 2005, pp. 82-90.

[8] G. Xing, Z. Xia, and D. Ayers, "X2R: a system for managing XML documents and key constraints using RDBMS," in *Proc. of the 45th annual southeast regional conference*, Winston-Salem, ACM: North Carolina 2007.

[9] I. Dweib, A. Awadi, S. E. F. Alrahman, and J. Lu, "Schemaless approach of mapping XML document into Relational Database," in *Proc. of the 8th IEEE International Conference on Computer and Information Technology, CIT 2008*, Sydney, Australia, 2008, pp. 167-172.

[10] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura, "XRel: A Path-Based Approach to Storage and Retrieval of XML documents using Relational Databases," *ACM Transactions on Internet Technology*, vol. 1, issue 1, pp. 110-141, 2001.

[11] H. Jiang, H. Lu, W. Wang, and J. X. Yu, "XParent: An Efficient RDBMS-Based XML Database System," in *ICDE*, 2002, pp. 335-336.

[12] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang, "Storing and Querying Ordered XML using a Relational Database System," in *SIGMOD*, 2002, pp. 204-215.

[13] S. Soltan, and M. Rahgozar, "A Clustering-based Scheme for Labeling XML Trees," *IJCSNS International Journal of Computer Science and Network Security*, vol. 6, pp. 84-89, 2006.

[14] M. Atay, A. Chebotko, D. Liu, S. Lu, and F. Fotouhi, "Efficient schema-based XML-to-Relational data mapping," *Information Systems*, vol. 32, pp. 458-476, 2007.

[15] S. Amer-Yahia, F. Du, and J. Freire, A comprehensive Solution to the XML-to-Relational Mapping Problem, in *WIDM'04* Washington, DC, USA, 2004.

[16] Q. Lee, S. Bressan, and W. Rahayu, XShreX: Maintaining Integrity Constraints in the Mapping of XML Schema to Relational, in *the 17th International Conference on Database and Expert Systems Applications (DEXA'06)*, Krakow, 2006.

[17] S. U. Knudsen, T. B. Pedersen, C. Thomsen, and K. Torp, RelaXML: Bidirectional Transfer between Relational and XML Data, in *the 9th International Database Engineering &#38 (IDEAS'05)*, 2005, pp. 151-162.

[18] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon, "XML Path Language (XPath) 2.0," Internet: http://www.w3.org/TR/xpath20/, 2007 [Feb. 15, 2008].

[19] X. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon, "XQuery 1.0: An XML Query Language," Internet: http://www.w3.org/TR/2007/REC-xquery-20070123/, 2007 [Feb. 15, 2008].

[20] M. Ramanath, "*Schema-based Statistics and Storage for XML*," Doctor of Philosophy Thesis, Indian Institute Of Science, Bangalore, India, 2006.

[21] "Oracle XML DB Developer's Guide 10g," Internet: http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14259/toc.htm, [Oct. 10, 2006].

[22] "DB2 XML Extender," Internet: http://www-306.ibm.com/software/data/db2/extenders/xmlext/index.html [Oct. 10, 2006].

[23] J.-K. Min, C.-H. Lee, and C.-W. Chung, "XTRON: An XML data management system using relational database," *Information and Software Technology*, vol. 50, pp. 462-479, 2008.

[24] Q. Li, and B. Moon, Indexing and Querying XML Data for Regular Path Expressions, in *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001, pp. 361-370.

[25] X. Wu, M. L. Lee, and W. Hsu, A prime number labeling scheme for dynamic ordered XML trees, in *Proceedings of the 20th International Conference on Data Engineering*, 2004, pp. 66-78.

[26] K. Kobayashi, L. Wenxin, D. Kobayashi, A. Watanabe, and H. Yokota, VLEI Code: An Efficient Labeling Method for Handling XML Documents in an RDB, in *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005*, 2005, pp. 386-387.

[27] Jaap van Oosten, "Basic Category Theory," Department of Mathematics, Utrecht University, The Netherlands. Internet: www.math.uu.nl/people/jvoosten/syllabi/catsmoeder.pdf Jul. 2002 [Aug. 15, 2008].

[28] Grust Torsten, "Accelerating XPath location steps," in *Proc. of the 2002 ACM SIGMOD international conference on Management of data*, Madison, Wisconsin: ACM, 2002.

[29] U. Washington, Computer Science & Engineering Research. "XMLData Repository," Internet: http://www.cs.washington.edu/research/xmldatasets/, 2002 [Oct. 15, 2008].

Received: February 15, 2009 Revised: May 25, 2009 Accepted: June 03, 2009