International Journal of the Physical Sciences Vol. 7(25), pp. 4012-4025, 29 June, 2012
Available online at http://www.academicjournals.org/IJPS
DOI: 10.5897/IJPS12.160
ISSN 1992-1950 © 2012 Academic Journals

Full Length Research Paper

Mapping extensible markup language document with relational database management system

Mohammed Adam Ibrahim Fakharaldien, Khalid Edris, Jasni Mohamed Zain and Norrozila Sulaiman

Faculty of Computer System and Software Engineering, University Malaysia Pahang, Malaysia.

Accepted 14 June, 2012

The extensible markup language (XML) is designed for data representation with conciseness, generalization, and usability through the internet, however, this technology requires an appropriate medium for the storage of data. The approach, named XRecursive, is proposed to store and retrieve XML documents using relational database, which is applied to all internal nodes mapped by Schema-oblivious, and all the leaf nodes mapped by Schema-conscious. The XRecursive storage model of XML data querying can effectively reduce the search range and to improve the search speed. The XRecursive storage model for querying data is more efficient than the Edge models, further, it is easy to implement and simpler than the XRel model. XRecursive model is suitable for the storage of XML documents, which is based on a relational database without document type definition (DTD) or XML schema information.

Key words: Extensible markup language (XML), relational database, document type definition (DTD).

INTRODUCTION

XML is rapidly becoming the chief standard for data conversion through the internet. At present, it plays an essential role in data management, interchangeability, and transformation. XML had accomplished the extensive assistance and application in all the principal areas such as software, tools, services, and database suppliers (Grandi et al., 2003). Importantly, XML has become the commonest language by the economic cost of searching, processing, exchanging, and reusing information. XML provides a self-describing and standardized method for representing information in a way that is understandable by humans and effortlessly confirmed, transformed, and distributed. In order to elevate the efficiency of querying and processing data, many researchers always seek the optimal method for storing XML documents (Sainan et al., 2009). On the other hand, to take advantage of the new ubiquitousness of associated software applications, the data can be used for transmission inorder to remote services anywhere using XML-based Web services by the internet.

The openness of XML (Augeriet et al., 2007) permits it to be virtually exchanged between any operating system, hardware, and software (Szlavik et al., 2006) from Queen Mary University, London, whereby they studied the role of automatic summarization on XML retrieval system, and offered the following conclusions: the use of XML summary is very active; searchers will spend more time on reading the summary of the document, rather than reading the full text directly. This means that by viewing the summary, the users decide which part of the document is worth reading (Szlavik et al., 2006). Research on automatic summarization has also attracted the majority of research institutions and enterprises. Document Understanding Conference, which is held annually since 2001, provides an international evaluation platform for researchers. The history of evaluation tasks, automatic abstracting, and information retrieval have widened their perspectives.

*Corresponding author. E-mail: braziliy@yahoo.com.

With the tendency of expanding a number of XML documents on the World Wide Web (WWW), it is necessary to have a powerful and well-organized mechanism for storing and querying XML documents to utilize the technology efficiently. Nowadays XML appears to be a standard for symbolizing data on the WWW when the influential storage instrument for data structure is the relational database (Zafari et al., 2010). The relational databases have been a powerful implementation of the storage, search, and data retrieval from various data collection. The proficiency of mapping XML data in relational database is an obstinate and challenging operation for researchers in the world. Thus, there is an urgent need to enhance an interface and an instrument for efficiently mapping and storing XML data in relational database. Relational databases are used especially for relating separated data records and grouping by types. Researchers can exploit these records as their necessities using Structured Query Language (SQL), and at the same time, introduce one or more records to terminal users. The data storage was transfigured by the relational database pattern for its efficiency, simplicity, and economic costs.

Relational databases have been exploited extensively in huge organizations since the 1980s, and it is probable to preserve the prevailing storage mechanism for business data in the foreseeable future. In spite of these potentials, relational databases still require the flexibility to perfectly integrate with other structures (Reed, 2008). In addition, although relational databases possess many similarities, there are major differences in the major commercial implementations. Thus, it is difficult to develop further applications and integrate with multiple products. The obstacles are distinctive according to data types, various levels of correspondence to the SQL standard, proprietary extensions to SQL, and so on. For storing of an XML document, the problem is to convert its tree structure into tuples of relational tables (Yue et al., 2008).

Nowadays, some data are represented in the form of XML document, the requirement of persistently storing these data has been enlarged promptly in database application. In addition, the native–XML databases normally have some limitations in supporting relational databases. In recent years, with the universality of relational databases (RDB) (Sybase Corporation: 1999; Yoshikawa et al., 2001; Xparent: 2002; Jiang et al., 2002; Kyung-Soo, 2001; Rys, 2000), these RDB-based approaches have properly stored and manipulated XML data as relational tables. There is a need to seamlessly manage XML and relational data with equivalent storage and retrieval capabilities.

Furthermore, Sandeep et al. (2006) proposed a schema-oblivious strategy-- Sucxent++ (Schema Unconscious XML Enabled System), which out-performs existing schema-oblivious methods such as XParent by up to 15 times and 8 times for recursive query execution. Moreover, XML and Relational databases should not be remained independently, since XML has been presented as the general standard format of data for expressing and interchanging information, whereas the majority of existing data are stored in RDBMS and the capability power of data does not allow downgrading. Therefore, in relational databases, an efficient method for storing XML documents is a prerequisite to solving this problem.

A novel efficient method for storing XML document in the relational database is presented in this paper. The objective is to resolve these problems by the combination of Schema-conscious and Schema-oblivious approach. The Storage of XML documents in a relational database is an encouraging solution because relational databases are very common in today's computer world. The advantage is that XML and structured data can survive together in a relational database, which makes it possible to constitute applications involving both data without extra effort. The method does not need a DTD or XML schema. Also, it can be applied as a general solution to any XML data; to all internal nodes mapped by Schema-oblivious; and to all the leaf nodes mapped by Schema-conscious. The steps and algorithms are shown in detail and a description to use the storing structure for storing and querying XML documents in relational database is given. The experimental results are handled on a real database. It shows that the performance of the proposed algorithm obtains a better result in terms of storage size, insertion time, retrieval time, and querying performances, compared with SUCXENT and XParent. The rest of this paper is organized as follows. Reviews of the XML databases are presented. Also, description of the proposed XRecursive algorithm is also discussed. The experimental and comparison results were depicted and finally, the conclusion of this work is given.

LITERATURE REVIEW

Many researchers have introduced a large amount of different procedures for storing XML documents into the database (RDB). These techniques can generally be classified into three tracks: semi-structured database (Goldman et al., 1999), object-oriented database (Bancihon et al., 1988), and relational systems (Shanmugasundaram et al., 1999; Yoshikawa et al., 2001; Jiang et al., 2002; Florescu and Kossman, 1999; Ramanath et al., 2003; Bohannon et al., 2002; Tatarinov et al., 2001; and Zhang et al., 2002). In the above-mentioned methods, the relational storage channel has fascinated noticeable interests with an outlook for leveraging their influential and credible data-management services. The purpose of mapping XML documents into relational database is to exploit relational database power capabilities in indexes, triggers, data integrity and

security and query optimization by SQL in order to store an XML document in a relational database.

First of all, the tree structure of the XML document should be mapped into the schema which is provided with equivalent, flat, and relational characteristics. Additionally, the XML document is represented by means of mapped tables in a shredded and loaded way. Next, the queries of XML are converted into SQL, and then complied into the RDBMS. Finally, the results are re-translated into XML. There are a lot of literatures for proposing the issue of administering XML documents in relational back-ends (Shanmugasundaram et al., 1999; Yoshikawa et al., 2001; Jiang et al., 2002; Florescu and Kossman, 1999; Zhang et al., 2002). These approaches can be classified into two major categories as follows.

Schema-conscious approach

This approach initially produces a relational schema by using the DTD/schema of the XML documents. Firstly, it constituted the cardinality of the relationship with the nodes in the XML document. According to the established information, a relational schema is designed. The structural information of XML data is modeled by using the foreign-key and primary-key, which link to the parent–child relationships of the XML tree in the relational database's model. The approach is applied to Shared-Inlining (Shanmugasundaram et al., 1999), LegoDB (Bohannon et al., 2002; Ramanath et al., 2003). It is clear that this method is a dependent of the existing schema to illustrate the XML data. Additionally, owing to the heterogeneity of XML data, an uncomplicated XML schema/DTD often presents a relational schema with many tables in this approach.

Schema-oblivious approach

This approach supplies a regular schema involving the use of the storage of XML documents. The fundamental concept is to capture the tree structure in an XML document. The method does not involve the existence of an XML schema/DTD and the number of tables settled in the relational schema, also it does not rest on the structural heterogeneity of XML documents. The Schema-oblivious approaches are applied to the Edge approach (Florescu and Kossman, 1999), XRel (Yoshikawa et al., 2001), XParent (Jiang et al., 2002), and SUCXENT (Prakash et al., 2004). In Schema-oblivious approaches, it is obvious that the advantage of this method is its ability to handle the changing XML schema, the unnecessary alteration of the relational schema, and a uniformed querying translation method. On the other hand, Schema-conscious approach has efficient performance of query processing (Tian et al., 2002). Furthermore, for the schema-conscious approach, there are no special needs for relational schema to be designed as it can be generated based on the DTD of the XML document(s).

Edge approach

The commencing point is one or a series of XML documents (Florescu and Kossman, 1999). The researchers suggested parsing and scanning these documents one by one and store all information into relational tables. For downrightness, an XML document can be shown on a labeled and ordered directed graph. In addition, every XML element can be designated by one node in the graph; whereas in the XML object, the node can be labeled with the OID. In the graph, the relationships between element and sub-element are designated by edges and are labeled by the name of the sub-element. In an XML object, in order to design the order of sub-elements, the outgoing edges of a node are also shown in the graph.

The graph shows the leaves' represented values (e.g., strings) of an XML document. In a relational database, six ways are considered to store XML data (that is graphs): three optional methods are designed for storing the edges of a graph, and two alternative ways to store the leaves (that is, values). Consequently, three trials are applied to two varied schemes. Other approaches and variations of these ideas are discussed and described in (Florescu and Kossmann, 1999). Particularly, the researchers proposed and described an approach, which would take advantage of features in an object-relational database system for storing multi-valued attributes (Florescu and Kossmann, 1999).

Using a graph to describe the XML data is a simplification, and some information can be vanished in this process. The interpretation is that the graph model is different between references (i.e., IDREFs) and sub-elements, or between attributes sand XML sub-elements. Thus, the original XML document cannot be restored precisely from the relational data. However, in the relational database, the processes of simplifications can be readily alleviated with additional bookkeeping.

In the graph, the simplest approach to attain the accumulation of all edges which designates an XML document in the edge table is a single table. In the graph, the OIDs of the source and objectives of each edge are recorded by using the Edge table, as well as recorded elements including the label, the edge, a flag that shows whether the edges denote an inter-object reference (that is, an internal node) or point to a value (that is, a leaf), and an ordinal number because the edges are ordered. The mapping methods the author investigated are remarkably simple. Due to the simplified schemes, these will not be the best options, but the experimental results suggest that even with such straightforward mapping

methods, it is feasible to obtain extremely good performance of querying. The only process which had unpleasantly high cost was entirely restructuring an exceptionally large XML document; the more complex the mapping techniques, the weaker would be the performance in this process.

This scheme was only the first stage towards determining the best way for storing XML data. The experimental results can be applied as a source to develop and construct more complex mapping techniques. Additionally, further experimental results with various types of XML data, synthetic and real, are required. Further, other characteristics such as locking activities, and authorization, need to be considered. Furthermore, the performance of experimental results with the storage of OODBMSs and particular-target XML data should also be controlled. From the authors' web pages, XML-QL and SQL queries and the XML documents can be retrieved, which can be applied in the experiments.

XRel

a)The XRel method is applied to store XML data graphs in four tables: Path (PathID, Pathexp); b) Element (DocId, PathID, Start, End, Ordinal); c) Text (DocId, PathID, Start, End, Value); and d) Attribute (DocId, PathID, Start, End, Value) (Yoshikawa, 2001).

The database attributes DocId, PathId, Start, End, and Value denote the documental identifier, simple path representation identifier, starting location of a section, terminal location of a section, and string-value, respectively. In an XML document, the section of a node is the starting and terminal locations of this node. The section, or the pair of starting and terminal locations, indicates a containment connection. '…with sections, the order of documents among attribute nodes with sharing the identical parent element node is left implementation determined in the description of the XPath data model' (Yoshikawa, 2001).

The unique characteristic of XRel is that there is no node identifier as a requirement for storing XML data graphs. As a substitute, the starting and terminal locations are applied. The structures of attribute and text nodes are stored by XRel in the Text table. The method suggested that the retrieval and storage of XML documents are applied to relational databases XRel, on the other hand, enables us to build an XPath interface based on the relational databases.

In the method, the researchers controlled expansion to functions and types, and did not require any particular indexing arrangement for query processing. However, some expansions may be required; for instance, the types of abstract data for the synchronization of querying results would be needed if these are required to implement an XML-QL interface. Furthermore, since the method does not need to apply a specific full-text search pattern, it might not realize high performance on querying retrieval. Therefore, it is essential to propose types of abstract data to enhance performance. For document content, full-text search is essential for data types and XML schemas, and supports for document updates.

XParent

The features of XParent are as follows:

a) XParent is an edge-oriented method. The XParent schema obviously supports data-paths and label-paths in two different tables: datapath and labelpath. DataPath provides the interior construction of XML data graphs based on a relationship of parent and child. LabelPath retains all distinctive label paths as tuples. For DataPath, it can be further emerged into an Ancestor table to establish the relationship between the descendant and the ancestor. Data and elements (attributes or texts) are appended to data-paths. A data-path is recognized by the terminal node identifier (Did for data-path id).
b) In order to differentiate the edge scheme, XParent maintains the distinguishing label-paths as data in a table. By using the information, XParent can simply process standard path queries, for instance, DBGroup.*.Name. For the querying process, XParent can identify all the paths that can match with DBGroup.*.Name in LabelPath. Therefore, in the Data table, XParent can recognize the values assigned to the end of those paths by using the path identifiers.
c) To differentiate the Monet's scheme, XParent arranges datapaths in a single table. To simplify the queries of a single path, this technique is inferior to Monet. However, XParent is predicted to surpass Monet in standard path queries, which is the major influence of XML queries. For instance, an XML query may include two *" paths. With Monet, a *" path matches with a group of tables; the other *" path matches with another group of tables. If there is a junction necessity on the data bound to these two *" paths, every table in a group of tables requires to connect all tables in the other group, additionally, since all label-paths are deliberated as relation names, an additional software module is forced to evaluate the label-paths and recognise relations to be applied.
d) XParent applies the equivalent schema, such as XRel and The Did (data-path id) to replace the start and end pairs used in XRel. XParent can be advanced by the traditional indexing mechanisms like the B-tree. Equijoins will be used for the replacement of μ-joins.

METHODOLOGY

XML document

The data of XML document is the hierarchical structure, including

```
<? Xml version="1.0" encoding="UTF-8"?>
<Personnel>
<Employee type="permanent">
<Name>Seagull</Name>
<Id>3674</Id>
<Age>34</Age>
</Employee>
<Employee type="contract">
<Name>Robin</Name>
<Id>3675</Id>
<Age>25</Age>
</Employee>
<Employee type="permanent">
<Name>Crow</Name>
<Id>3676</Id>
<Age>28</Age>
</Employee>
</Personnel>
```

Figure 1. XML document.

nested structures. The beginning and ending tags strictly mark the elements, and empty-elements tags are used to represent empty elements. The content of the elements is expressed by the Character data between tags. It is a sample of XML document that comprises information about an employee which is shown in Figure 1.

The tree structure representation of XML document

In Figure 2, the structure of the independent mapping method is interpreted with a sample XML document indicated in Figure 1.

XRecursive structure

An XML document is generally expressed as an XML tree. In the XML tree, internal nodes are expressed as element attributes in the XML document, while the leaf nodes are expressed as elements of the property values. The technique of XRecursive is applied to all internal nodes mapped by Schema-oblivious, and all the leaf nodes mapped by Schema-conscious. Internal nodes are used to describe the structure of an XML document in the XML tree, and are only useful for document navigation. Schema-oblivious can provide the completed structure information in an XML document, which can effectively and easily support the document traversal. Since leaf nodes are the values of the elements in an XML document, they are useful in navigating the XML and these leaf nodes can only store the data values of nodes in an XML document. The data type of each leaf node can be effectively described by using the Schema-conscious storing data. In this way, flexible methods are provided for storage according to various requirements.

(a) Storage within the node: In order to speed up the traversal processes in the XML tree, the efficient numbering way is adopted as the nodes coding in the XML tree, and at the same time it is very

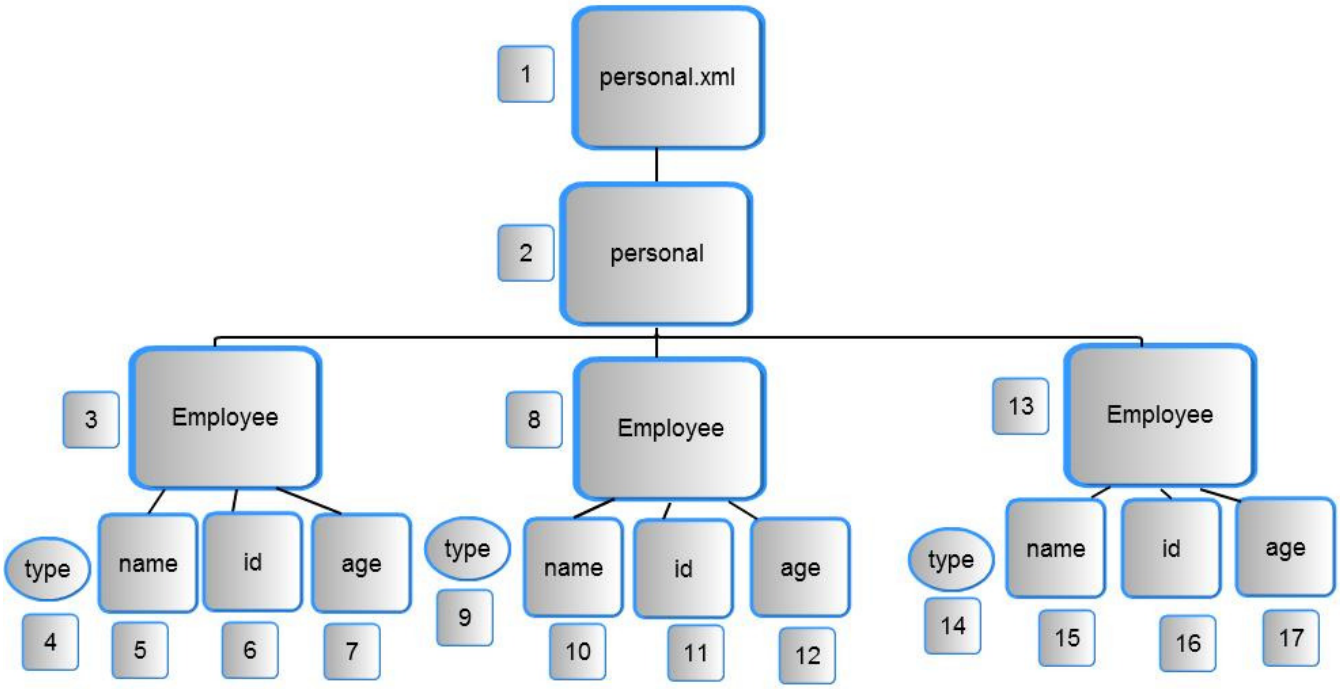

Figure 2. Tree structure of XML document with XRecursive labeling.

important that the parent-child relationship between the elements are rapidly decided based on this encoding scheme. With XRecursive-based XML storage model, it is necessary to capture the coded information of each node mapped to the relevant data model. Note: XRecursive only needs to code for the internal nodes, unlike most Schema-oblivious schemes need to identify all the nodes in the XML.

Tatarinov et al. (2002) proposed three order encoding mode to describe the XML order in the relevant data model. In these methods, the Dewey order method is efficient to query and update performance. Each node is assigned by an ID value, and the value arrays are separated by points which are acquired by the path array from the root to each node. The root node is assigned by a positive integer, the ID of children nodes is expressed as adding a point after the ID of the parents nodes, and then following the serial number of the local node.

The XRecursive method is expressed by ancestor - descendant relationship in the tree structure of Dewey order method in Figure 1. It can be easily identified by the ID values. ID length determines the depth of the tree, but the comparison between the ID strings reduces the query performance, as well as the error of the former generation which is passed to the descendants. O'Neil et al. (2004) proposed ORDPATH method with new level labels to avoid the above defects. ORDPATH provides a notation method by compressing the binary encoding Dewey order. In this strategy, each node ID value is represented by a continuous and variable-length Li/Oi bit stream. Each Li bit stream is the preamble coding mode, which is used to designate the length of Oi binary string in the entire string. Here is an example: Li is expressed as 01, designated the length with 3. Li describes the binary digit of Oi with three-digit, and the binary string (000, 001, 010… 111) can be expressed as the value of Oi with 8 integers (0, 1, 2… 7), for example, 01101 is expressed as ordinal 5 (O'Neil et al., 2004).

XRecursive is designed by using ORDPATH as an internal node coding, and building the document structure information in relational database. The following model is used to store internal nodes:

xpath (xpath id, length, xpathexp)
internalnode (uid, xpath id, nodename, ordpath, parent, grdesc, lid, o id, tablename)
Interpretation of the meaning of the parameters is as follows:

(1) The xpath table is used to record the xpath information in the XML tree. The xpath id and xpathexp are expressed in the xpath identifier and path expressions respectively. The parameter length is used to record number of the edge of xpath.
(2) The internal node table describes each internal node. The u id is used to identify the internal nodes, and the ordpath records the ORDERPATH of this internal node. The lid is a local identifier for the internal nodes, which is used to identify the location of this node in the brother nodes. The OID is used in the query with a predicate, which is the index of nodes appeared multiple times in an XML document. The tablename indicates the table of the leaf nodes' children who stored this node. The parameters parent and grdesc are used to determine the relationship between the ancestors and generations in the two internal nodes.

b) Storage of leaf nodes: In order to identify the data type of each leaf node and to map leaf node in the database by Schema-conscious method, a new mapping mode is used in the proposed method. This mapping model is defined by MPD (the explanation file in mapping process), an XML format. The function of MPD file is similar to an XML format DAD file in DB2, which is used to store the mapping of any XML data in the database. In XRecursive, XML document is considered as a target tree, and then the MPD document is used to map these objects into a relational database.

In this view, a node is generally considered as a class, and the node is mapped to the table.

The proposal of the mapping mode of leaf nodes is much simpler than the existing Schema-conscious. The mapping of an internal node into tables is needed to map attribute information of this node class. It is unnecessary to list the class of relevant internal nodes for this node, such as its parent and child nodes. Note that the relationship between the ancestor and descendant has been mapped to the relevant model by the Schema-oblivious approach.

Every single XML can be described as a XML tree. In Figure 2, the squares are expressed as the elements, and the ovals are expressed as the attributes of the elements. An XML tree has been generated in Figure 2. Every element or attribute is identified by a signature (number).

Definition 1 (XRecursive structure)

XRecursive is a storage structure to store XML documents where each path is identified by its parent from the root node in a recursive manner.

Definition 2 (all nodes)

Let A = {n1, n2... nk} be the set of all nodes (id), B = {np1, np2... npn} be the set of parent notes (pId) and D = {nl1, nl2... nlk} be the set of leaf nodes (tagId) then:

1 - $\forall$ npi $\in$ B: npi $\in$ A

2 - $\forall$ nli $\in$ D: nli $\in$ A

3 - $\forall$ npi $\in$ B: npi $\notin$ D and npi $\in$ A – D

4- $\forall$ nli $\in$ D: nli $\notin$ B and nli $\in$ A – B

Algorithm map-document

To store XML document into relational database, it is essential to map the XML document into a corresponding table in the relational database. The process that maps the XML document into relational database is presented in Figure 3.

Algorithm retrieval-document

Based on the information of tuples, the retrieval document will achieve accurate retrieval of the original document. For every element node the algorithm stores the end tag of element nodes and then output a beginning tag. The retrieval of the XML document from the relational database is presented in Figure 4.

Definition 3 (Node Id)

Let $N = \{n_1, n_2, \cdots, n_k\}$ be a set of all nodes in XML tree such that preorder $n_i <$ preorder n_{i+1}, for $1 \le i \le k$, then node Id is defined as follow:

$$nodeId(n_i) = \begin{cases} 1 & , \quad for\ i = 1 \\ nodeId(n_{i-1}) + 1, & for\ 1 < i \le k \end{cases}$$

Proposition 4 (Parent node Id)

Let $P = \{p_1, p_2, \cdots, p_k\}$ be a set of parent node (pId) in XML tree such that preorder $n_i <$ preorder n_{i+1}, for $1 \le i \le k$, then parent node is defined as follows:

$$parent(Id_i) = \begin{cases} 1 & , \quad for\ i = 1 \\ parent(Id_i) < parent(Id_i), & for\ 1 < j \end{cases}$$

Example 1

In this structure, even a type or an element associates with its signature, it also represents its parent element. In order to add the multiple XML file in the storage, the document name is associated with its id. Figure 2 represents the storage of the XML file in association with its signature. For every element, a signature should be associated with it, as well as a parent's signature in association with it. In Table 1, agName represents the name of the node; id represents the id of the node which is the Primary Key, pId represents the parent id of the node. The document name does not have any parent id so the id parent and the id of the document name are equivalent. It is shown in Figure 2.

Definition 4 (tagId)

Let $T = \{t_1, t_2, \cdots, t_k\}$ be the list of leaf nodes in the XML tree such that preorder $n_i <$ preorder n_{i+1}, for $1 \le i \le k$, then node Id is defined as follows:

$$tag\ Id(n_i) = tag\ Id(n_{i-1}) + 1, \ for\ 1 < i \le k$$

In Table 2, the element or type is represented in association with the value. In XRecursive structure, when the path structure or pathvalue is determined recursively by its parent id, it is not necessary to store it. In Table 1, the name of the tag is represented by tagName, where the parent key is represented by Id. In Table 2, tagId is represented by the id in Table 1, the foreign key is represented by tagId. In this Table, the elements are merely represented by tagId, which contain the value of representing the value column and the 'E' denotes the element and 'A' represents the attribute.

THE XRECURSIVE DOCUMENTS STORAGE ANALYSIS OF MODEL MAPPING

The XRecursive storage model consists of one figure and two tables: the tree structure of XRecursive labeling shows the path information of an XML document in Figure 2. In Table 1, the Tag structure is stored in an

```
00 Algorithm process XMLFile(XML as a document)
01 Begin
02     Let tag_structure represents the list of the nodes of the XML file.
03     Let tag_value represents the list of the values of the nodes.
04     Let filename=null as string and parentid=1 as integer.
05     Let nodecounter=1 as integer
06     filename=read xml document name
07     Add filename, nodecounter, parentid to tag_structure
08
       Recursivexmlnode(XmlDocRootNode,parentid,nodecounter)
09      store tag_structure into database
10 store tag_value into database
11 End Algorithm
00 Procedure Recursivexmlnode(xmlnode, parentid, nodecounter)
01 Begin
02 if xmlnode is not text node then
03 Begin
04 tagstructure(xmlnode, parentid, nodecounter)
05 if xmlnode has childnodes then
06      for each xmlattribute in xmlnode attributes
07      Begin
08      set parentid=nodecounter
09      while xmlnode has childnodes do
10      tagstructure(xmlnode, parentid, nodecounter)
11      End while
12      tagstructure(xmlnode, parentid, nodecounter)
13      tagvalue(nodecounter, value, type)
14 If xmlnode.next=true then go to 09 End
 if
15      End for
```

Figure 3. Mapping XML to RDB algorithm.

```
retrieval xml from RDB Algorithm
01 input: tuples (tag_structure ts, tag_value tv) of table document_name
02 output: XML document
03 Begin
04 for (ts.id=1; ts.id<=n; ts.id++)
05 begin
06 if (ts.id !=tv.tagid)
07 outputfile("<ts.name>")
08 else if (ts.id=tv.tagid and tv.type='A')
09 outputfile("ts.tagname=tv.value")
10 else if (ts.id=tv.tagid)
11 outputfile("<ts.tagname>tv.value<tv.tagname>")
12 end if
13 End for
14 End Algorithm
```

Figure 4. Algorithm of retrieving XML from RDB.

Table 1. Tag_structure.

tagName	Id	pId
Personal.xml	1	1
personal	2	1
Employee	3	2
type	4	3
name	5	3
id	6	3
age	7	3
Employee	8	2
type	9	8
name	10	8
id	11	8
age	12	8
Employee	13	2
type	14	13
name	15	13
id	16	13
age	17	13

Table 2. tag_value.

tagId	Value	Type
4	Permanent	A
5	Seagua11	E
6	3674	E
7	34	E
9	Contract	A
10	Robin	E
11	3675	E
12	25	E
14	Permanent	A
15	Crow	E
16	3676	E
17	28	E

XML document, which does not contain a leaf node. Table 2 shows the Tag value which contains the edge of the leaf nodes for the storage of a document. In the tree structure of XRecursive labeling, each path is corresponded to only one path flag. There are four different paths in Figure 2. In the tables, the Tag structure and value reveals the Tag-name ID, pID, Tag-ID, as well as the destination node's value and type. Using this model, the XML document without schema information can be relatively mapped to relational tables in order to realize the querying of XML data for using the supplied

function in relational database.

The well-known mapping storage models are Edge and XRel. Edge Model is applied to the Infoset data model, and a table describing the XML document content, where the table tuples are expressed as the Target, Source, Lable, Value, Ordera1. This mapping method is simple and easy to implement. XRel model uses XPath the data model, and the XML documents are represented in an ordered tree. XML content of the document describes the model with four tables, namely the path table (Path), the element table (Element), the attribute table (Attribute), and the body of the table (the Text). The path table stores path information. Element table tuples are represented as Doc_ID, path ID, Source, Target, in-dex, re-index, and the information is used to describe all the elements in the document node. Among them, the index and re-index are represented as a serial number and a reverse chronological order serial number for brothers' elements in the document respectively. The attribute table is used to store the value of the node of the document; the properties form the tuple which is Doc-ID, the path-ID, Source, Target, Value. The body of the table stored in the document node value, the tuple form of expression is Doc-ID path_ ID of the Source, Target, Value. In the operation of querying, the speed of querying is improved by the path information and the order of the elements (the index and re-index), which is suitable for querying information of containing inequality (>, <).

The advantage of using the Edge model to describe the contents of an XML document is to take up less storage space. Since XML data query is based on the path, the model does not directly provide the path information. When the query is executed, the model only connects the operating table of each edge linking in order to complete the discrimination of the path information. The query process requires a large number of join operations; so the querying efficiency is not high. In addition, the Edge model is a single XML document model and is not suitable for the storage of multiplied XML documents. XRel model with four tables maps the XML document to a relational database, taking up the storage space.

Because each element in the XML document is marked with a serial number of the elements order and reverse chronological order in the document, the querying process shows a good performance of the inequality conditions. However, most relational databases do not support the indexing based on the range (inequality), so the improving model needs to add some special operations, which does not include the relational database, for instance to identify the elements of the re-index value or to build up the range (inequality) for indexing. This can be achieved specifically using the cumbersome and complex process. In addition, the model of range-based querying does not satisfy the W3C Recommendation of XML data search criteria XQuery.

Therefore, the mapping method can effectively avoid finding the path that is satisfied with the condition, and effectively reduce the number of traverse nodes. In this way, the proposed mapping method can reduce the search range and improve the search speed. Compared with the Edge models, the querying process of the joint operation is significantly reduced. Through the XRecursive value table, a relational database is used to provide an indexing mechanism to further enhance the data querying speed. Moreover, the storage space occupied by the model is smaller than the XRel model.

THE ANALYSIS OF EXPERIMENT

This section presents the experimental results of the proposed method. The evaluated performance includes document's insertion times, storage space requirements, and query performance. The experimental environment is a PC with P4 2.4GHz machine with 1 GB of RAM, 240 GB of hard disk. Its operating system is Windows XP, and the comparison database is SIGMOD, which provides a comprehensive range of XML document types. This section compares XRecursive with Edge (Florescu and Kossman,. 1999), XRel (Yoshikawa et al., 2001) and XParent (Jiang et al., 2002). The experiment is set according to the following three factors:

Extraction time

The extraction time is the total number of time possessed in retrieving the tuples from the database and reconstructing XML document. The extraction time is better than all other approaches because a smaller data set (tag-structure, tag-value) and the optimal relational schema of XRecursive are used, such as Id, tagId columns, which store primary and foreign key information (Figure 5).

Basically, the Edge method is used to handle information of the tree edge, which is reserved in a single relational schema for estimating XPath query. The single table approach, which is adopted in Edge, is uncomplicated, since it only keeps the edge-label, rather than the label Paths, and a vast amount of joints is required to exanimate edge-connections. Therefore, in response to the query with including the path expression, these edges should be concatenated to each other. Additionally, the relational schema of XRecursive method has the similarity with XRel schema. In these processes, XRel method stores path for any existing elements in a XML tree. However, the difference is that the technique of XRecursive is applied to all internal nodes mapped by schema-oblivious and all the leaf nodes mapped by Schema-conscious. Finally, in the XParent method, the Path table has the similarity with XRel process that keeps paths for all existing nodes with resulting in increasing

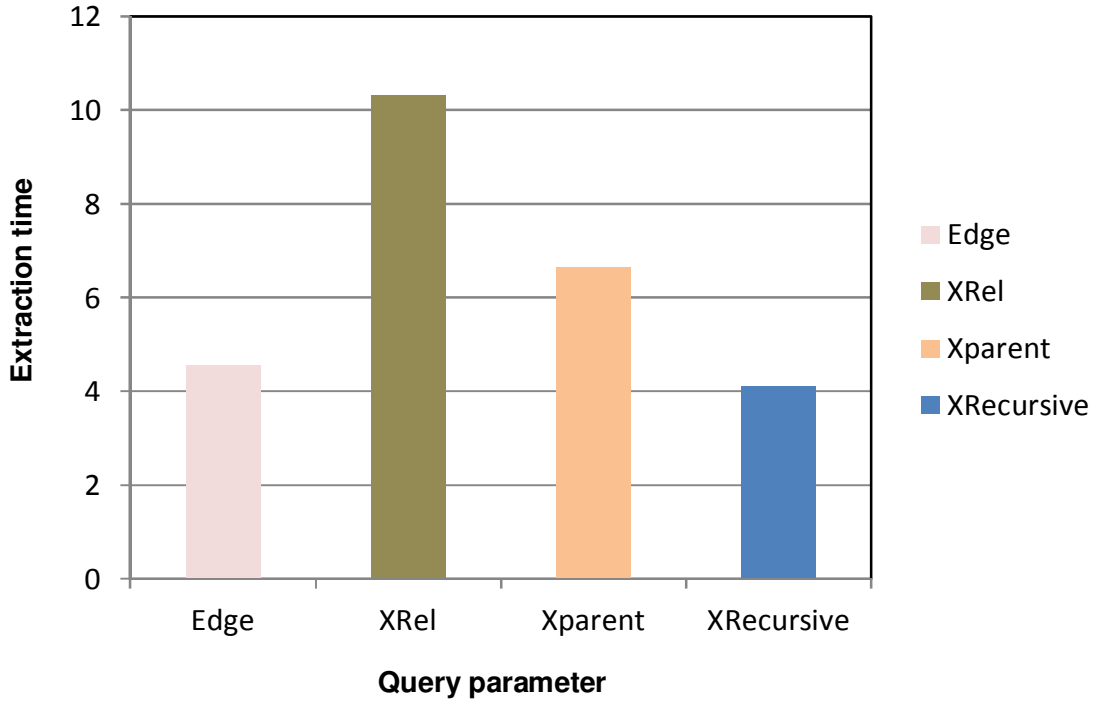

Figure 5. Extraction time.

Table 3. Samples query feature.

Query number	Query feature
Q1	Simple short path
Q2	Simple long path
Q3	Use one axis
Q4	Use two axis

storage space in a document. In XParent, the relational schema does not hold ancestral information in the normal case. However, by using DataPath table, ParentId of all existing nodes within a document is observed. In this way, the discovering process of an ancestor for a special node is extremely costly because it requires to frequently associate tables with themselves. DataPath table, rather than Ancestor table, is used for high capacity of ancestor table.

XRecursive method utilizes the idea of child- parent to solve this problem. The technique of XRecursive is applied to all internal nodes mapped by Schema-oblivious, and all the leaf nodes mapped by Schema-conscious. Internal nodes are used to describe the structure of the XML document in the XML tree, and are only useful for document navigation. Schema-oblivious can provide the completed structure information in an XML document, which can effectively and easily support the document traversal. Since leaf nodes are the values of the elements in an XML document, they are nearly useful for the navigation of XML and it can be considered that these leaf nodes only store the data values of nodes in an XML document. The data type of each leaf node can be effectively described by using Schema-conscious storing data.

Query performance

In the query performance comparison in Table 3, four queries, from Q1 to Q4, are compared by using three methods. With regard to the performance of queries, some points should be taken into account.

Q1: Query b is a quick XPath query. So there are not many joins in the SQL statement for the Edge method. The response time is very intimate as expected.

Q2: Using the XRecursive method, query requires one in association with each step in the XPath query; however in the path table the path information is used in the SUCXENT and XParent. As a result, XRecursive is faster in the long XPath query as expected.

Q3 and Q4: The execution time is comparable.

The proposed system was compared with other systems to evaluate the query performances. In the experiment, the size of the cache was fixed with 256 K. The execution time includes the querying time and the used time in the final results where were sorted according to the nodes

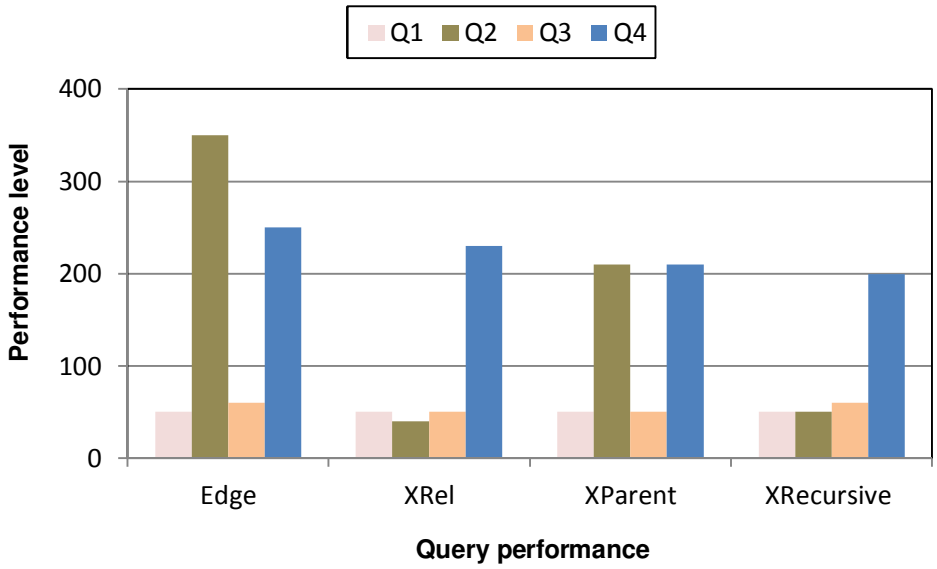

Figure 6. Query performance comparison with other systems.

order. The comparison results were shown in Figure 6. Because the edge approach basically utilizes one simple table to conserve information in XML tree edges, it is indispensable to associate frequently this table with itself. If a path expression should be considered, the produced numerous joints in the final SQL query will result in the poor efficiency. In the XRel system, the advantages are a significant reduction in storage space and faster searching in Path table. In a XML tree, the ratio of all paths in association with the path ended with the leaf will increase exponentially. Nevertheless, XRel requires the containment relationship of ancestor-descendant. In the XParent system, the discovering process of an ancestor for a special node is extremely costly because it requires frequent association of the tables. DataPath table, rather than Ancestor table, is used for high capacity of ancestor table. In the proposed XRecursive, it can provide flexible methods for storage according to various requirements. Furthermore, the technique of XRecursive is applied to all internal nodes mapped by Schema-oblivious, and all the leaf nodes mapped by Schema-conscious. In XRecursive method, child-parent ideal results are used for the reduction of the volume of the path table, which results in increasing the speed of access to nodes in the table, and consequently, it will cause a faster querying process.

Scalability

In this section, the scalability of the proposed XRecursive system was tested in the document. The size of XRecursive document was changed from 10 to 100 M, and the size of the cache is fixed with 256 K, the size of the page is 4 K. The four queries were observed in Figure 7, the results of the implementation can be seen from these results, the physical I/O times and query execution time increased linearly with the document size. Thus the experiment showed a good linear scalability of XRecursive's indexing, storage, and querying architecture.

Conclusion

In this paper, a typical approach to the storage of XML document in a relational database, namely the XRecursive, is proposed. In a relational database, the method implements the model-mapping technique to store the XML document, and the tree structure is decomposed into nodes and all information of nodes is stored recursively according to the types of nodes. Any document can be processed, whether it has fixed schema or not. XRecursive model is a multi-storage model of the XML document, which conforms to the ML data querying specific XPath and XQuery. The proposed mapping mode of leaf nodes is considerably simpler than the existing Schema-conscious. The mapping of an internal node into tables only needs to map attribute information about the node type. It is unnecessary to record the class of relevant internal nodes for this node, such as parents' and children's nodes. It should be noted that relationship of the descendant and ancestor should be mapped to the relevant pattern by Schema-oblivious. In addition, the XRecursive storage model for querying data is more efficient than the Edge models, since it is easy to

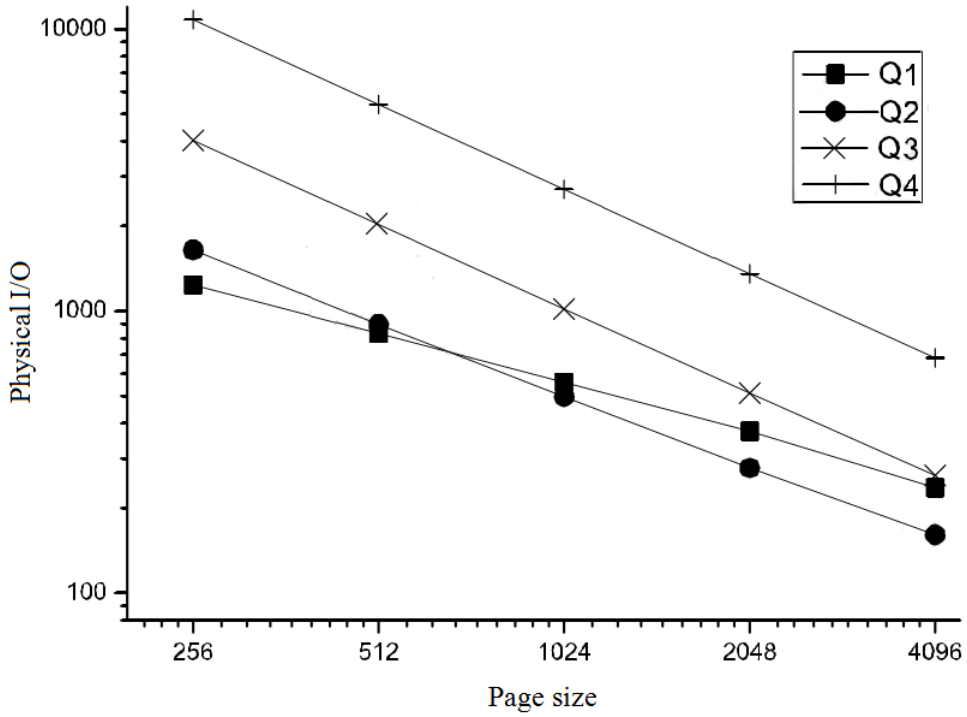

Figure 7. Experimental results: Physical IO vs. page size.

implement than the XRel model. The XRecursive storage model is suitable for the storage of XML documents, which is based on a relational database and without schema information. By utilizing this approach, clearly, the database size is decreased for storing the XML documents into relational database. Furthermore, the proposed method presents the storing approach of XML document into relational database, and its accuracy is determined by using the XML document in the experimental performance section.

REFERENCES

Augeri CJ, Bulutoglu, DA, Mullins BE, Baldwin RO, Baird LC (2007). An analysis of XML compression efficiency. Proc. Workshop Exp. Comput. Sci., pp. 1-12.

Bancihon F, Barbedette G, Benzaken V, Delobel C, Gamerman S, Lécluse C, Pfeffer P, Richard P, Velez F.(1988). The design and implementation of o2, an object-oriented database system. Adv. Object Oriented Database Syst., 6(1): 1-22.

Bohannon P, Freire J, Roy P, Simeon J (2002). From XML schema to relations: a cost-based approach to XML storage. In Proceedings of IEEE Int. Conf. Data Eng., pp. 64-75.

Florescu D, Kossman D. (1999). Storing and querying XML data using an RDBMS. IEEE Data Eng. Bull., 27-34.

Goldman R, McHugh J, Widom J (1999). From semi structured data to XML: migrating the lore data model and query language. In Proceedings of WebDB, 99: 25-30.

Grandi F, Mandreoli F, Tiberio, P, Bergonzini M (2003). A temporal data model and management system for normative texts in XML format. In Proceedings of the 5th ACM Int. Workshop Web Inf. Data Manag., pp. 29-36.

Jiang H, Lu H, Wang W, Yu JX (2002). Path materialization revisited: an efficient storage model for xml data. In Proceedings of the 13th Australasian Database Conference, pp. 85-94.

Kyung-Soo J (2001). A design of middleware components for the connection between xml and rdb. In Proc. IEEE Int. Symp. Ind. Electron., pp. 1753-1756.

O'Neil PE, O'Neil EJ, Pal S, Cseri I, Schaller G, Westbury N (2004). Ordpaths: Insert-friendly xml node labels. In SIGMOD Conf., pp. 903-908.

Prakash S, Bhowmick SS, Mardia S (2004). SUCXENT: An Efficient Path-based Approach to Store and Query XML Documents. Lecture Notes in Computer Science Springer Verlag, 3180: 185-195.

Ramanath M, Freire J, Haritsa J, Roy P (2003). Searching for Efficient XML-to-relational mappings. Lecture Notes in Computer Sciences Springer Verlag, 2824: 19-36.

Reed D (2008).Take a good look. Data Strategy, from Business Source Complete database, 2(4): 24-29.

Rys M (2000). Microsoft sol server 2000 xml enhancements. Microsoft Support Webcast.

Sainan L, Caifeng L, Liming G (2009). Storage Method for XML Document based on Relational Database. In Proceeding of the IEEE Int. Symp. Comput. Sci. Computational Technol., pp. 127-31.

Sandeep P, Sourav SB, Sanjay M (2006). Efficient Recursive XML Query Processing Using Relational Database Systems. In DKE, 58(3).

Shanmugasundaram J, Tufte K, Zhang C, He G, DeWitt DJ, Naughton JF (1999). Relational databases for querying XML documents: limitations and opportunities. In Proceeding of International Conference on Very Large Data Base (VLDB), pp. 302-314.

Sybase Corporation: (1999). Using XML with the Sybase adaptive server sol databases. Technical whitepaper.

Szlavik Z, Tombros A, Lalmas M (2006). The use of summaries in xml retrieval. In Proceedings of the 10th European Conference on Research and Advanced Technology for Digital Libraries, pp. 75-86.

Tatarinov I, Viglas S, Beyer K, Shanmugasundaram J, Shekita E, Zhang

C (2002). Storing and querying ordered XML using a relational database system. Proc. ACM SIGMOD Conf., pp. 204-215.
Tatarinov I, Viglas S, Beyer K, Shanmugasundaram J, Shekita E, Zhang C (2001). Storing and querying ordered XML using a relational database system. In Proc. Proc. 2001 ACM SIGMOD Int. Conf. Manag. data, pp. 204-215.
Tian F, DeWitt D, Chen J, Zhang C (2002). The design and performance evaluation of alternative XML storage strategies. ACM SIGMOD Record, 31(1): 5-10.
Xparent: (2002). An efficient rdbms-based xml database system. In Proc. 18th Int. Conf. Data Eng., pp. 335-336.
Yoshikawa M, Amagasa T, Shimura T, Uemura S (2001). XRel: a path-based approach to storage and retrieval of xml documents using relational databases. ACM Trans. Internet Technol., 1(1): 110-141.
Yue L, Ren J, Qian Y (2008). Storage Method of XML Documents Based-on Pre-order Labeling Schema. In Proceeding of the IEEE Int. Workshop Educ. Technol. Comput. Sci., pp. 50-53.
Zafari H, Hasami K, Shiri ME (2010). Xlight: An Efficient Relational Schema To Store And Query XML Data. In Proceeding of the IEEE Int. Conf. Data Store Data Eng., pp. 254-257.
Zhang C, Naughton J, Dewitt D, Luo Q, Lohmann G (2002). On supporting containment queries in relational database systems. In Proceedings of the Proceedings of the 2001 ACM SIGMOD International Conference on Management of data (ACM SIGMOD), pp. 425-436.

.