Model-Mapping Approaches for Storing and Querying XML Documents in Relational Database: A Survey

Amjad Qtaish, Kamsuriah Ahmad

1School of Computer Science, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, 43600, Bangi, Selangor, Malaysia
dr.qtaish@gmail.com

2School of Computer Science, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, 43600, Bangi, Selangor, Malaysia
kam@fism.ukm.my

ABSTRACT

Extensible Markup Language (XML), which is recommended by the World Wide Web Consortium (W3C), has rapidly become the dominant standard for data interchange and data representation on the web. At present, with the growing use of XML data on the web, the size of this type of data is increasing rapidly, and users issue more complex queries on this data. Therefore, the demand to manage this data in terms of storing and querying has elicited considerable research attention. This demand has prompted numerous researchers to propose various approaches for using relational databases to manage data because of its maturity and excellent scalability. These approaches can be categorized into: structure-mapping and model-mapping approaches. In this paper, we discuss the main characteristics and the challenge of storing XML documents in relational databases to obtain high query performance. We surveyed and compared the most cited model-mapping approaches such as Edge, XRel, XParent and XPEV. This paper presents the advantages and limitations of these approaches in this field. This Paper shows that although XPEV approach overcome on others approaches like Edge, XRel and Xparent in storage space, query response time and the influence of scalability on query performance when data sizes become bigger, it still has not resolve storing and querying XML documents in efficient way. Therefore, the paper paves the way for solutions to improve these approaches in terms of efficient storing and querying of XML documents.

Keywords: XML, Relational database, Model-Mapping, Storing, Querying

1. Introduction

Data interchange between systems has become a challenging issue because of variations in data format and semantics of metadata, which are used to describe data. These challenges led to the development of Extensible Markup Language (XML), which developed by the World Wide Web Consortium (W3C) [1]. XML has rapidly become the prominent standard for data interchange and data representation on the web. It is a semi-structured data language, with a nested, self-describing nature, using simple-to-write and simple-to-parse language to facilitate data interchange in a variety of systems on the web. It provides a platform-independent method for describing data. Therefore, it easily transfers data from one platform to another.

With these features, XML has enabled communication between different computing systems, a feat that had previously been extremely difficult or even impossible. XML thus provides a global structure for the interchange of data regardless of the platforms and data models of the applications [2]. Considerable amounts of data on the web are represented in XML. Therefore, managing XML data in terms of storing and querying has become a research challenge.

Numerous researchers in [3], [4], [5], [6] have introduced different approaches to manage XML data in terms of storing and querying, including file systems, object-oriented database systems, native XML databases systems, and relational database systems. Previous studies like [7] indicated that relational database systems overcome other approaches because of the efficiency of storage. Relational database systems have commercial strength from giant vendors. Furthermore, relational databases are
popular for data management in terms of storing, updating, and searching capabilities via Structured Query Language (SQL).

In view of the maturity of this technology, storing XML data in this technology is mainly conducted by fetching the data in XML documents, convert and store into relational database. The other approach is by mapping XML schema into a relational schema then store into a relational schema. Relational database management systems (RDBMS) will remain the prevalent storage mechanism in managing business data in the foreseeable future due to their powerful data-management services. With this approach, XML document will be represented as a relational database and users can access the document by using the same mechanism being used in relational databases. The user then issues XML queries (such as XPath) over the document which are translated into queries over a relational database (SQL), and the result of the queries will be translated back into XML, where all these processes will be done internally [8].

Several researchers focus on the mapping process from XML documents into relational databases by taking the advantages of the properties from both XML and database. In addition, designing an effective relational storage model to store the XML data to get high query performance has been the central focus for database research.

The main contribution in this study is a comprehensive survey of the topical research development of model-mapping approaches for storing and querying XML data, such as Edge [7], XRel [4], XParent [6], and XPEV [9]. We analyze the most cited model-mapping approaches that concern the basic approaches for any incoming approaches by identifying their advantages and limitations. In addition, we discuss the comparison between them in terms of storage space, query response time, and scalability. We conclude the possible issues for rendering efficient XML storing and querying in the future.

The remainder of this paper is sorted as follows: Section 2 reviews that are model-mapping and structure-mapping, and the differences between them. Section 3 gives some overviews of model-mapping approaches, their advantages and their limitations. Section 4 is a comparison of the model-mapping approaches. Section 5 is the discussion of this paper, which provides certain recommendations for storing and querying XML documents using relational database efficiently. Final section is the conclusion and future work of this paper.

2. Model-mapping and structure-mapping

Numerous different mapping approaches have been introduced to store and query XML documents for reducing the quantity of relational tables, which lead to improved storage space, query performance, and scalability. Therefore, mapping approaches can be categorized into (i) structure-mapping approaches, where the design of relational schema is dependent on the support of DTDs (Document Type Definition) or XML-schema that describes the structure of XML documents, and (ii) model-mapping approach, where a fixed relational schema is designed to store any XML document without DTD assistance [10]. Both approaches perform their mapping process from XML documents into relational databases by shredding (decomposing) XML documents into relations and translate XML queries into SQL queries over these relations.

On the web, XML documents may come with a schema (assistance of DTD or XML-schema to describe the data) or without a schema (without assistance of DTD or XML-schema) that are well-formed. Considering the existence of numerous XML documents in different forms on the web, difficulties present themselves in managing these documents in terms of storing and querying. Model-mapping approaches solve this problem by storing any XML documents in a relational database, and can utilize relational database technology to store and query XML documents.

In this paper, we will study the model-mapping approaches, instead of structure-mapping approaches, for the following reasons: First, they support any modern XML applications that are considered either static (the DTDs are not changed) or dynamic (the DTDs change frequently). By contrast, structure-mapping approaches are unsuitable for storing a large amount of dynamic and structurally variant XML documents. Second, they support well-formed XML applications (without a schema that describes data). This means it supports any XML documents on the web whereas structure-mapping approaches support only XML documents with DTD or XML-schema structure. Finally, they do not require expanding the expressive power of database models to support XML documents.
Conversely, using the structure-mapping approach, the database model does not include constructs to describe the elements in the element content models in DTDs. This means that we cannot find structure-mapping approaches that map data structures of XML documents into the database schema in a natural manner [4].

3. An overview of model-mapping approaches

In this section, we discuss the most cited model-mapping approaches, advantages, and limitations in terms of storage, query response time, and scalability with keeping on query performance in terms of data sizes.

3.1 Edge approach

Edge [7] was one of the first approaches using relational database for storing XML documents. It uses the XML Information Set (Infoset) data model [11]. It is the simplest and the most straightforward approach in terms of loading XML data into a single Edge table. It maps each edge in the XML tree to a tuple in a single relation, which is defined by

\[
\text{Edge (Source, Ordinal, Target, Label, Flag, and Value)}
\]

The Edge table stores the identifier of the source and target nodes of each edge of the graph (tree), the label of the edge, a flag that indicates whether the edge represents an internal node or points to a value (that is, leaf), and an ordinal number reflecting the ordered edges.

The limitations of the Edge approach are that only the edge-label is retained, instead of the label paths. It requires concatenating the edges to create a path. Therefore, multiple join operations are required in comparison with the length of the path expression to check edge-connections for handling user queries, which increases query response time. It requires extreme self-joins for complicated query processing. By storing large amounts of XML data for complicated XML documents, the Edge table will increase the storage space with poor query performance [6]. Therefore, its scalability in terms of increasing the size of XML data in documents will be poor.

3.2 XRel approach

XRel [4] is a path-based mapping approach in the oblivious of schema type. It uses the XPath data model [12]. Therefore, an XML document is represented as a tree structure of nodes. These nodes are stored in relations based on the node type, with distinctive information from the root node to any node. Unlike in Edge, concatenating edges is unnecessary in XRel schema. The distinctive path expressions and the regions of tags within the document are stored in relations. Using a region (the start and end positions of the tag), it handles a containment relationship, which applies not for parent-child relationships, but rather ancestor-descendant relationships. The important feature of XRel is that no node identifiers are required to store XML data graphs. Instead, start and end positions are used. The basic XRel schema contains the following four relational schemas:

\[
\begin{align*}
\text{Path (PathID, Pathexp)} \\
\text{Element (DocID, PathID, Start, End, index, reindex)} \\
\text{Text (DocID, PathID, Strat, End, Value)} \\
\text{Attribute (DocID, PathID, Start, End, Value)}
\end{align*}
\]

The XRel schema stores DocID, PathID, Start, End, and Value, which respectively represent document identifier, simple path expression identifier, start position of a region, end position of a region, and string-value. The limitations of the XRel approach are the high number of joint tables and tuples, which will increase the produced database size [13]. It stores paths for any existing nodes that exist in the XML tree, which causes an increase in storage space requirements and leads to slower search in the path tables. It uses the containment relationship to preserve ancestor-descendant relationships. It uses θ-joins (<, >) operators that have been proven to be expensive in joining its tables.
due to the manner by which an RDBMS processes joins [14]. In XRel, the cost of querying that requires to check the parent-child relationship is high. That means when checking the parent-child relationship of any two nodes from tag positions, confirming that no node appears between the two nodes is important [15]. Its scalability in terms of increasing the size of XML data in documents will be poor because of the \( \Theta \)-joins involved in the translated SQL queries which concern a high query cost.

### 3.3 XParent approach

Xparent [6] is an Edge-oriented method. It supports data-paths and label-paths in two separate tables (DataPath and LabelPath). DataPath provides the interior construction of XML data graphs based on a parent-child relationship. LabelPath preserves all distinctive label paths as tuples. DataPath could be further developed into an Ancestor table to preserve ancestor-descendant relationship. Unlike Edge, it maintains the distinguishing label-paths as data in a table. This is similar to the XRel schema but the difference is that Did (data-path Identifier) in XParent schema is used instead of the start and end positions in XRel. This can be advanced by traditional indexing strategies such as B-tree. It uses an Ancestor table that stores all the ancestors of a specific node in a single table to solve the cost of \( \Theta \)-joins by replacing \( \Theta \)-joins with equijoins over this set of ancestors. It adopts the XPath data model [12] to represent XML documents, which models a document as an ordered tree, then maps that tree into four relational tables as below. To increase the query processing time, an Ancestor table (Did, Ancestor, Level) is proposed, which preserves ancestor-descendant relationships.

**LabelPath** (ID, Len, Path)
**DataPath** (Pid, Cid)
**Element** (PathID, Did, Ordinal)
**Data** (PathID, Did, Ordinal, Value)

In the LabelPath table, the attributes ID, Len, and Path represent a unique label-path identifier, the number of edges of the label-path, and distinctive simple expression path, respectively. In the DataPath table, the attributes Pid and Cid respectively represent the parent-node id and child-node id of an edge. The attributes of the Element and Data tables such as PathId is a foreign key of the ID in the LabelPath table. Did is a data-path identifier. In addition, Ordinal number and the string-value are in those tables.

The limitations of XParent approach are that it is similar with the XRel process, which preserves paths for all existing nodes, resulting in increasing storage space requirements [16]. It retains ParentId and ancestor information for all existing nodes in a document. Therefore, finding an ancestor of a specific node becomes costly because it must join tables with themselves frequently [14]. In addition, query performance and scalability are poor in terms of increasing XML data size in the documents.

### 3.4 XPEV approach

XPEV [9] represents an improvement from Edge and XRel approaches. It stores multiple XML documents in its relational schema and uses three tables schema to store XML documents in a relational database: Path table stores distinctive (root-to-any) paths, Edge table preserves parent-child relationships, and Value table stores all the values of elements and attributes. By using XPEV, any well-formed XML document can be mapped to the relational database schema. It uses XQuery 2.0 and XPath 1.0 Data Model [17], and is an efficient mapping approach for storing XML documents using relational database technology to enhance query processing. In addition, it is easy to realize, and it outperforms other model-mapping approaches. Therefore, it is easier to apply than the XRel approach, and has improved querying performance than those of Edge and XRel. XPEV is characterized by higher scalability querying performance compared with those of Edge and XRel when the size of data sets increase. Its schema consists of the following:

**Path** (Pid, Pathexpress)
**Edge** (Pid, Source, Target, Label, Ordinal, Flag)
**Value** (Pid, Source, Target, Label, Ordinal, Value)
In *Path* table, the attributes *Pid* and *Pathexpress* represent the path identifier and the distinctive simple path expression, respectively. In Edge and Value tables, the attributes *Pid*, *Source*, *Target*, *Label*, *Ordinal*, *Flag*, and *Value* represent a path identifier, the identifier of the source node, the identifier of the target node, the label of the edge, a flag indicates to an internal node or a value (that is, leaf), and an ordinal number of the node among all siblings that share the same parent because the edges are ordered and string-valued, respectively.

The limitations of XPEV approach are that its relational schema remains large, because it stores entire XML documents in two tables, namely *Edge* and *Value*. Similar to XRel, it stores paths for any existing nodes in the XML tree, which cause an increase in storage space requirements, causing slower search in path tables, and affects query performance. Its scalability in terms of data sizes remains insufficiently efficient, which is affected by query performance.

4. **Comparisons of model-mapping approaches**

We discuss the comparisons between the most cited model-mapping on approaches such as Edge, XRel, XParent, and XPEV, in terms of issues in storage space, query response time, and scalability with keeping on query performance in terms of data sizes.

4.1 **Storage Space**

Reducing the storage space of the relational schema is important to produce an efficient model-mapping approach. The comparison between model-mapping approaches based on storage space that require storing their relational schema is shown in Table 1.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Storage Space</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Edge</strong></td>
<td>High because it stores whole XML documents in a single table.</td>
</tr>
<tr>
<td><strong>XRel</strong></td>
<td>High because of the complexity of its relational schema that consists of four tables. In addition, it stores paths for any existing nodes in the document.</td>
</tr>
<tr>
<td><strong>XParent</strong></td>
<td>High because of the complexity of its relational schema that consists of four tables. In addition, it stores every node in the document and stores also every ancestor of a node.</td>
</tr>
<tr>
<td><strong>XPEV</strong></td>
<td>High because of the complexity of its relational schema that consists of three tables. In addition, it stores whole XML documents in two tables (Edge and Value). In addition, it stores paths for any existing nodes in a document.</td>
</tr>
</tbody>
</table>

Table 1 shows the relational schema that is produced by these approaches, which requires high storage space, which increases required query time to retrieve information.

4.2 **Query Response Time**

Reducing the time taken for query response is the most important issue, which makes it a primary objective of several approaches. Therefore, the model-mapping approaches translate the XML query (XPath) into SQL statement depending on their relational schema to attain high query performance. The comparison between the model-mapping approaches following query response time is shown in Table 2.
Table 2 shows that by using model-mapping approaches, the query response time is poor for various reasons.

### 4.3 Scalability

On the web, XML documents change periodically, and increase in size. This leads to numerous approaches to achieve high scalability with keeping on their query performance based on their schemas. The comparison between the model-mapping approaches in terms of achieving scalability with keeping on their query performance is shown in Table 3.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge</td>
<td>Poor because the number of join operations involved in translated SQL is increased.</td>
</tr>
<tr>
<td>XRel</td>
<td>Poor because of the number of O-joins involved in the translated SQL queries is increased.</td>
</tr>
<tr>
<td>XParent</td>
<td>Higher than Edge and XRel</td>
</tr>
<tr>
<td>XPEV</td>
<td>Higher than Edge and XRel</td>
</tr>
</tbody>
</table>

Table 3 shows that the scalability with keeping on query performance in XParent and XPEV are superior to those of Edge and XRel because they use fewer joins than those used by Edge. XParent and XPEV used equijoins, which costs less than the O-joins used in XRel.

## 5. Discussions

As stated, research on mapping XML documents into relational database in terms of storing and querying XML data is not comprehensive. Numerous studies have focused on proposing an efficient approach for storing and querying XML documents in terms of storage space requirements, query
response time, and scalability. However, several challenging issues have not been resolved efficiently and therefore, require extensive research. These approaches failed to find an efficient way for designing relational schema for XML documents based on achieving the least amount of storage space to store relational schema, achieving the least time taken to respond query, and achieving high scalability to preserve query performance even as the XML data increases in size.

These approaches did not consider other challenging issues that may aid in designing their relational databases for storing and querying XML document efficiently. Among those issues are full-text and structural queries, dynamic updates and Relationships between nodes in XML tree.

5.1 Full-text and structural queries

Based on user demand, XML queries can be classified into two forms, full-text query and structural query. In full-text query, users usually do not know the entirety or the exact structure of the XML document. They may simply be aware of names of the element nodes, or values of attribute nodes and leaf nodes. These texts are used as constraints in querying for statistical information. Although, full-text query involves certain keywords or paths, and is simple in most scenarios, the answers provided by the search engine are excessive to retrieve accurate and limited answers. The query can be fulfilled by matching the Path provided on the Path table.

The structural query is more complicated. The user often has a deeper understanding about the content within the XML document. Structural queries include complicating chain and twigging queries. Chain query is defined one node at a time. The node could be a leaf node or an element within inner nodes. The query can be fulfilled by performing substring operations to split the path expression into several nodes so that the name of each node could be obtained. As a result, an efficient model-mapping approach should support both full-text and structural queries.

5.2 Dynamic updates

Considering that XML is a popular language that is adopted for data representation and exchange on the web, this type of data updates as time passes. Thus, an efficient model-mapping approach should support dynamic updates on XML data, i.e., it does not require modification and relabeling of existing data stored in its relational schema when an update occurs, such as inserting a new element in the XML documents.

5.3 Relationships between nodes in XML tree

In order to preserve the relationships between nodes in the XML tree of the XML documents, which increases the performance of Full-text and structural queries in a certain approach, an efficient model-mapping approach should store all relationship information between nodes in its relational schema such as level, siblings, parent-child, and ancestor-descendant relationships.

6. Conclusion and future work

In this study, we discussed the characteristics and limitations of the most cited model-mapping approaches for storing and querying XML documents in relational databases, such as Edge, XRel, XParent, and XPEV. We compare the most cited approaches in terms of storage space requirement, query performance, and scalability in terms of increasing the data sizes. In terms of storage space, we can conclude that model-mapping approaches suffer from increasing storage space for storing their schema, which leads to poor query performance because of the complex designs for their schema. In addition, they store unnecessary data in their schema. Therefore, the demand for designing an efficient relational schema for storing and querying XML documents becomes an urgent issue.

In terms of query performance, we conclude that query performance in XParent and XPEV approaches are better than that in Edge and XRel because they use equijoin operations in translated SQL, which is less costly than the O-joins that are used in XRel; XParent and XPEV also use fewer joins that those used in Edge. However, they continue to attain low query performance due to the complexity of their relational schema. In addition, the translated SQL query remains un-optimized.
Therefore, the need to improve the query performance based on efficient storage has elicited considerable research attention. Based on scalability with keeping on query performance when data sizes become bigger, we conclude that XParent and XPEV have higher scalability than those of Edge and XRel because their query performance is not affected when XML data increases in size. Nevertheless, they do not reach high scalability because the translated SQL query based on these approaches is not optimal.

For future work, we believe that our survey will be useful in the research community and will serve as suitable introductory material for individuals who wish to propose an efficient model mapping approach in storing and querying XML documents using relational database.

7. References