Querying XML Document using Relational Database System: A Survey

Sohel A Rahi 1, G.R Bamnote 2

1 PG Student Department of Computer Science & Engg. Prof..Ram Meghe Institute of Technology and Research Badnera, India
sohelrahi@gmail.com

2 HOD, Department of Computer Science & Engg. Prof..Ram Meghe Institute of Technology and Research Badnera, India
grbamnote@rediffmail.com

ABSTRACT
XML is rapidly emerging being the eminent standard for representing data in the World Wide Web. High-level query engines that allow end users to effectively tap the data stored in XML documents will be crucial to exploiting the full power of XML. While there have been quite a lot of activity recently proposing new semi-structured data models and query languages for this purpose. Paper explore the technique which is used to store XML document into the Relational database or native XML database and to retrieve the data by using Relational database system. We know there is separate query processor required to retrieve the data from the relational database system in this paper we proposed a technique which will enable same query processor to work seamlessly among all the XML file.

Keywords - Native XML, Extensible Markup Language, XML Databases, Query processor, Relational database

I. INTRODUCTION
An XML database is a database software system that permits data to be stored in XML format. This data can then be queried, exported and serialized into the desired format. XML databases are usually associated with document-oriented database. There are two kind of XML database exists discussed below.
1. XML-enabled: these may either map XML to traditional database structures (such as a relational database), accepting XML as input and rendering XML as output, or more recently support native XML types within the traditional database. This term implies that the database processes the XML itself (as opposed to relying on middleware).
2. Native XML (NXD): the internal model of such databases depends on XML and uses XML documents as the fundamental unit of storage, which are, however, not necessarily stored in the form of text files. One reason for the use of XML in databases: the increasingly common use of XML for data transport, which has meant that “data is extracted from databases and put into XML documents and vice-versa”. It may prove more efficient (in terms of conversion costs) and easier to store the data in XML format. The formal definition from the XML: DB initiative states that a native XML database:

Defines a (logical) model for an XML document — as opposed to the data in that document — and stores and retrieves documents according to that model. At a minimum, the model must include elements, attributes, PCDATA, and document order. Examples of such models include the XPath data model, the XML Infoset, and the models implied by the DOM and the events in SAX 1.0. It has an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage. Additionally, many XML databases provide a logical model of grouping documents, called “collections”. Databases can set up and manage many collections at one time. In some implementations, a hierarchy of collections can exist, much in the same way that an operating’s directory-structure works.

All XML databases now support at least one form of querying syntax. Minimally, just about all of them support XPath for performing queries against documents or collections of documents. XPath provides a simple pathing system that allows users to identify nodes that match a particular set of criteria. Many XML databases also support XQuery to perform querying. XQuery includes XPath as a node-selection method, but extends XPath to provide...
transformational capabilities. Traditional RDBMS vendors (who traditionally had SQL-only engines), are now shipping with hybrid SQL and XQuery engines. Hybrid SQL/XQuery engines help to query XML data alongside the relational data, in the same query expression. This approach helps in combining relational and XML data. Relational, Object-Oriented and XML database engines adopt the approach for answering a user query as depicted in Figure 1. As shown in the figure, a user query is first parsed and mapped to its equivalent algebraic representation called logical plan or query plan. This plan is then optimized by applying several optimization techniques and strategies. The output of this phase is an execution plan also known as physical plan. The next phase consists of mapping the execution plan to a sequence of statements which will in turn be processed as a final step towards the generation of results. The logical plan, which has either a tree or a graph structure, consists of a connected sequence of algebraic operators. The set of all operators defined by a database system forms what is called the database’s logical algebra. A clearly and precisely defined logical algebra in any database system is the single mechanism to guarantee the soundness and completeness of any query evaluation. Moreover, optimization in database systems is easier in the presence of a logical algebra. A logical algebra is implemented in a physical algebra where each logical operator is implemented by one or more physical operators. Examples of logical operators are: Select Project, Join, Union, Intersection, etc. Some possible physical implementations of, for instance, the Join operator are Nested loop, Sort-merge, and Hash-based. The difference between these physical operators is in the way they implement the intended functionality of the Join, resulting in a difference in the amount of resources (I/O cost, CPU resources, etc.) consumed by each. One of the tasks of the optimizer is to map, based on some collected statistical data and cost estimation techniques, each logical operator in the plan to one of its corresponding physical implementations such that the execution time of the plan is minimized. Generally speaking, the optimization step is defined as the process by which the optimal or suboptimal plan is chosen for executing the user query. The hardest step in the query execution process is the optimization phase.

Fig. 1 Steps involved in query processing

II. SURVEY DETAILS

Lot of research has been done and many different procedure are introduced for storing XML document into the Relational database System (RDB). These techniques can generally be classified into three tracks: semi-structured database [6], object-oriented database [9], and relational systems [10][11][14]. 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 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 compiled 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 [12]. These approaches can be classified into two major categories as follows.

1. 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-In-lining\(^{(11)}\). 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.

2. 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, XRel, XParent and SUCXENT. 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. 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 documents\(^{(11)}\)\(^{(15)}\).

3. Edge approach

The commencing point is one or a series of XML documents. 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. 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. 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\(^{(15)}\). 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.

4. 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)\(^{(17)}\). 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. 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-SQL interface.

5. 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 datapaths. 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 data paths 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. 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 recognize 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. [16][17]

In the technique discussed above the XML documents are generally shredded and stored into the relational database system i.e. in the form of relation and then SQL query is fired on those relation to retrieve the XML data from the Relational database table.

III. MODIFICATION AND CHANGES

In the technique discussed above in which XML document is first shredded and then stored into the Relational database system, instead the XML document is stored into the native XML database directly. Converting the XML database in the form of relational data involves lot of issues and complexities, hence there is method proposed in which instead of converting the XML data we put XML data as it is in the native XML database and convert the SQL query into the XPath query which will read the XML data from the native XML database and retrieve the XML data.

IV. PROPOSED TECHNIQUE

In the proposed technique, we are creating XML server which will consist of different module. Our XML server will be able to communicate with the client. This server has the capability of storing the XML files into the database. On the stored XML file, SQL query can be fired to manipulate the XML files. The XML file will be stored into the different domain which is maintained by the storage and this storage will be accessible to only the registered user. Figure 2 shows the different component in the proposed system, there is SQL editor to communicate with the server. Also there is XML database to store the XML file and all other processing under the XML server.

A. Component diagram

![Fig. 2 Simple component diagram of the proposed system.](image)
In above figure we showed the component of proposed system there is SQL editor which will be connected to the XML server, this SQL editor will be able to store XML document into the XML database and also able to manipulate XML documents. There is XML database which stores the XML data, this XML data will be in the form of XML file, we can say that for XML database the XML files are the fundamental storage. There is XML server which consist of the different component as shown in figure. SQL client manager will manage the session between the server and the SQL editor. SQL parser will check syntax of SQL statement if it is valid then will generate a parse table. It will parse standard SQL statement such as Select/Insert/Update/Delete. SQL to Xpath will generate Xpath expression according to parse table created by SQL parser. Xpath expression Processor will process the Xpath expression to retrieve data from XML document. It will use java-SAX API for processing Xpath for XML documents.

B. State diagram for the proposed system

![State Diagram](image)

Fig. 3 Diagram representing different states within system

V. CONCLUSION

We now have presented a technique for querying XML documents using a relational database system, which, (a) permits the same query processor to be used with most relational schema generation processes, and (b) allows users to query seamlessly across XML documents. The proposed technique can thus serve as the infrastructure for investigating issues to stored different XML file into the database and carry out the different operation on the file using different query languages. However, based on our prototype implementation in Java, we have found that it takes less to translate SQL queries into XML queries. Most of the time is actually spent in SQL query execution, which typically takes on the order of seconds. For SQL query execution, we use the sorted outer union technique, which has been shown to be both stable and efficient. Also XML server is able to handle the multiple request form the client and process them, and this become possible because of multithreading which force the processor to share its time among the different request by the client. This make our server capable of handling many request at a time. The XML file get store directly into the database which saves the time as well as problems occurs while conversion of XML file into some other format of data. With the XML server we can also maintain the different domain required in the database.

REFERENCES


Conference, Edinburgh, Scotland, September 1999.


