Query Processing and Evaluation for XML Databases

Asmita P. Asre
PG Student, Department of Computer Sc. & Engg of Prof.Ram Meghe Institute of Tech & Research, Badnera

Prof. Dr. M.S.Ali
Principal, Prof. Ram Meghe College of Engineering & Management, Badnera.

ABSTRACT

While the information published in the form of XML-compliant documents keeps fast mounting up, efficient and effective query processing and evaluation for XML have now become more important than ever. The query processor of a database system is the most critical component when it comes to performance and scalability. Structural join operations are central to evaluating queries against XML data, and are typically responsible for consuming a lion's share of the query processing time. Thus, structural join order selection is at the heart of query optimization in an XML database, just as (value-based) join order selection is central to relational query optimization. XML is an emerging standard for data representation and exchange on the World-Wide Web. Due to the nature of information on the Web and the inherent flexibility of XML, much of the data encoded in XML will be semistructured, the data may be irregular or incomplete, and its structure may change rapidly or unpredictably. Our contribution can be understood as a roadmap that reveals desirable information and a theoretical perspective for an XML query processing, evaluation and query optimization.

Keywords: XML Database, Query Processing, Query Evaluation, Query Optimization.

1. INTRODUCTION:

After relational, network-based, hierarchical, object-oriented, object-relational, and deductive database systems, academic research and businesses increase their attention to the database-driven processing of XML documents, resulting in a new kind of information system, namely the *(native) XML database system* (XDBS). This development is reasonable, because the *eXtensible Markup Language* nowadays plays an important role in various key technologies like content management systems, electronic data interchange, and data integration techniques. Furthermore, for the management of a possibly large collection of XML documents, the classical advantages of dedicated database systems over file systems still hold: Convenient use of XML data through a standardized application programming interface (API); Transactional warranties for all operations on XML data; Processing of large volumes of data, measured in number of documents as well as document size. Further advantages of database systems like scalability with respect to the current transactional load, high availability and fault tolerance, as well as data and application independence shall be mentioned for completeness, though they are not XML specific.

Level of Abstraction	XDBS	RDBS
Language Model	XQuery	SQL
Logical Access Model	XML Query Algebra	Relational Algebra
Physical Access Model	Physical XML Query Algebra	Physical DB-Operators
Storage Model	XTC, Natrix, Shredded Documents	Record-oriented DB-Interface

Table 1. XML Query processing abstraction levels

As XML [1] has gained prevalence in recent years, the storage and querying of XML data have become an important issue. Effective query optimization is crucial to obtaining good performance from an XML database given a declarative query specification. A join is frequently the most expensive physical operation in evaluating a relational query. Thus, selection of join order is a key task for a relational query optimizer.

This observation is true for an XML query optimizer as well, but with significant twists. Perhaps the most important of these is the prevalence of *structural joins* in XML. Structural join order selection is a critical component of an XML query optimizer. A join in the relational context is usually a value-based equi-join, which involves two tables and is based on the values of two columns, one in each table. In the XML context, even though there are value-based joins, *structural joins* occur much more frequently. A *structural join* focuses on the containment (ancestor-descendant or parent-child) relationship of the XML elements to be joined. The join condition is specified not on the value of the XML elements, but on their relative positions in the XML document. In short, queries on XML data have some features that are different from queries in the traditional relational context. Therefore, the set of alternative plans, and their relative costs, in the XML context are also quite different.

2. LEVELS OF ABSTRACTION IN XML QUERY PROCESSING:

To handle the complexity of query processing, several levels of abstraction between a declarative query expression and its procedural evaluation using a set of low-level operations can be identified. These levels are depicted in Table 1.To facilitate comprehension, the new XML-related concepts are compared to their well-known counterparts of relational query processing. The most abstract view of a query is its formulation in a way that only describes the desired result in a certain declarative language. The same query may be represented using an algebra expression, whose operators express the query in the *Logical Access Model*. Optimization techniques at this level only rely on the expression itself, but do not cope with system-specific information. In general, this is the task of the layer below—the *Physical Access Model*

Finally, the bottom layer accomplishing the storage of XML documents plays also an important role, because the efficiency of operations is critically dependent on the chosen storage structure. An explicit separation of this abstraction level helps to cope with mapping requirements when multiple heterogeneous storage models are present. [2]

3. XML QUERY PROCESSING:

An XML query language defines more comprehensible and structurized construct for conducting operation on an XML document or various XML documents. For processing the query, an XML query engine or processor translates the syntaxes and executing the operations hinted by the query. Output is returned after process and processing time is projected to be minimum thus alluding efficient processing.

A query processor extracts the high level abstraction of declarative query and its procedural evaluation into a set of low-level operations [3]. Analogous to SQL processor, SQL query is translated at logical access model and then the logical access prior to accessing and returning the physical storage model.

XDBS denotes XML database management system and RDBMS are Relational Database Management System. The language model is designed to meet the demands of [4] which are reflected in the language ability to perform search functionality and document-order awareness hence document-centric characteristics and later on the data-centric characteristics which is associated with powerful selection and transformation. The semantic processing should then be able to analyze the query and transform it into an international representation to be used throughout subsequent optimization steps.

Logical access model should implement algebraic and non-algebraic procedure to optimize the internal representation of the query. Non-algebraic optimization minimizes intermediary results by restructuring the query and executing most selective operations as early as possible. Algebraic optimization will transform the internal expression into a more optimized expression in a semantics-preserving manner.

Physical access model is related to system specific issue. At this level, each logical algebra operator will be decomposed into corresponding physical operators. The goal of this step of optimization is a query executing plan (QEP) which is arranged of chosen physical operators and their sequences of execution. Finally, the storage model affects the rate of QEP. For optimized query processing, appropriate storage model should be deployed in order to minimize I/O costs, CPU costs, storage costs for intermediary results, and communication costs. Currently used storage models comprise LOBs (Large Objects), certain XML-to-relational mappings (shredded documents), or native storage formats like Niagara [5] and Timber [6]. The relational XML data model and native storage model attract more attentions indicated by various proposals for respective overlying query processors. Referring to the abstraction levels, various XML query processors have been proposed for more optimized query processing.

4. XML QUERY EVALUATION:

When querying data in XML, the query is read by the Query Parser. It manipulates it and passes an internal data structure to the Query Optimizer. Then, the Query Optimizer, using information from the Metadata Manager, changes the structure and passes it to the Query Evaluator. The Query Evaluator uses the Data Manager and the Index Manager to evaluate the query. Again, both the Data Manager and the Index Manager use the Storage Manager to get actual data. Overall, the evaluator takes in a query evaluation tree and substitutes each of the nodes with an appropriate access method. The evaluator uses both the Index Manager and the Data Manager to get results of the query and passes them to the Query Output API.

```
<library>
<book id = “B001-05”>
<category title= “XML Concepts”>
<author>Korth </author>
<author>Navathe </author>
<publisher>McGraw Hill</publisher>
</category>
</book>
<book id= ”B002-02”>
<category title= “Introduction to XML”/>
</book>
<journal id= “J001-05” >
<author>Kimball </author>
</journal> </library>
```

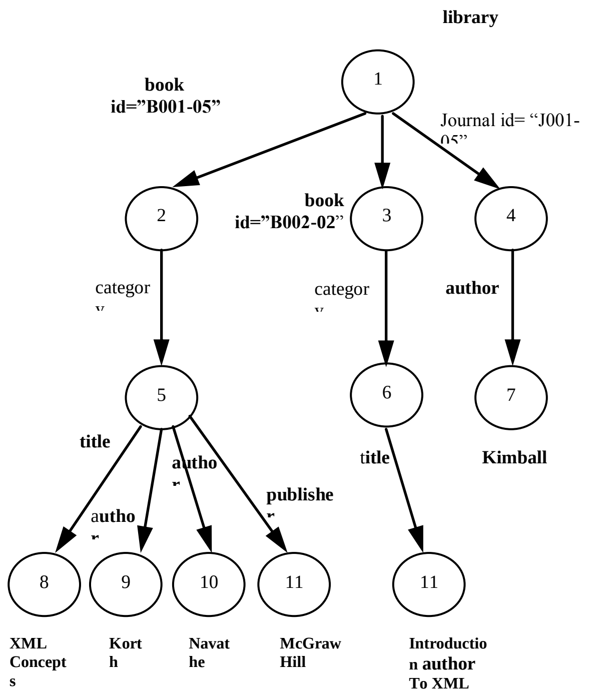

Figure. 1: A small XML document and its OEM representation

The main challenge of using RDBMS as instance storage is that, one need to resolve the conflict between the hierarchical nature of XML data model and the two-level nature (row and column) of relational data model [11]. There are two ways of doing so: (1) using the graph-based approach and (2) by inferring schema from Document Type Definition (DTD) [12].

Instead of generating relational tables for each XML element, several XML elements are combined into a single table named *edge* to reduce the number of join operations incurred while querying the data. The *edge* table stores the object identifiers (oids) of the source and target objects of each edge, the label of the edge, the ordering of the edge and a flag to shows whether the edge is a leaf or non-leaf node. If the node is a leaf, then a corresponding record will be stored in the *value* table. This table has the field of vid (storing oids of values) and value of the string.

Another way to map XML into RDBMS is to derive a relational schema from either a XML schema or DTD. This technique is not applicable for XML without a schema. This limitation, however, has been overcome [13].

A DTD graph (Figure 2) is created based on XML document in Fig. 1. Traversing down from library on the left-side, we have '+' edge follows by book element. Traversing further down, we reach category element before reaching to title, author and publisher element. All elements and attributes nested within category occur at most once, except the author elements. Hence, we can store category, title, publisher in the same relation as book. In each relation, the *ID* field serves as the primary key, while the *parentID* field serves as the foreign key to match with the values in the primary key. For example, in the relation journal, it has *journal.parentID* that joins journal with library.

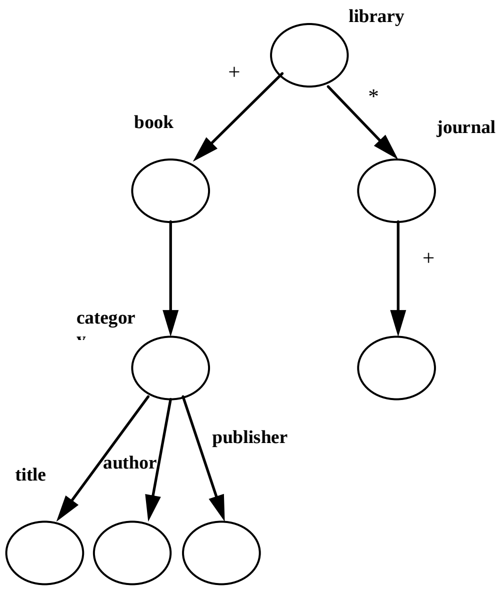

Figure.2: DTD graph derived from Figure. 1

5. QUERY OPTIMIZATION

Query optimization in the context of XML databases is extremely challenging. The main reason for this is the high complexity of the XML data model, as compared with other data models, e.g., relational models. This high complexity renders a much enlarged search space for XML query optimization. Furthermore, XML applications are typically Web-hooked and have many simultaneous, interactive users. This dynamic nature requires highly efficient XML query processing and optimization. The classical cost-based and heuristic-based approaches yield unacceptably low efficiency when applied to XML data—query optimization itself becomes very time-consuming because of the huge search space for optimization caused by the high complexity of the XML data model. Lots of work related to XML query processing has been done, but the majority is focused on investigation for efficient supporting algorithms [7], [8,] and indexing schemes [9].

There are many query optimization techniques discovered and implemented by researchers. Among them are path traversal, use of indexes, use of materialized views, pipeline evaluation, structured join order selection, schema-based optimization, and reformulation of XML constraints, duplicate removal, tree pattern matching and labeling scheme.

A problem with path traversal methods is that traversing is only possible in the constrained set of path. However, for the structure summary indexing, most of it has the problem of large index size growth in the worst case and not supporting partial queries path matching. Labeling scheme allow quick determination of the relationships among the element nodes and reduce the index size, but fails to support dynamic XML data. Towards the later year, most researchers proposed hybrid-indexing techniques. Hybrid system opens the possibility of covering each technology's weaknesses by its strengths. [10]

6. CONCLUSION

While the information published in the form of XML-compliant documents keeps fast mounting up, efficient and effective query processing and optimization for XML have now become more important than ever. XML query processing is important, irrespective of how XML data is stored: in a native XML database, after mapping to a relational database, or after some other mapping, such as to an object-oriented database. In this paper, an attempt is made to provide a roadmap that reveal desirable information and a theoretical perspective for an XML query processing, evaluation and query optimization. Based on the special features of XML data and XML queries, different techniques for performing query optimization in the XML framework can be the optimistic plan ahead for the researchers.

7. REFERENCES

[1] T. Bray, J. Paoli, C. M. Sperberg- McQueen."Extensible Markup Language (XML) 1.0".W3C Recommendation. Available at http://www.w3.org/ TR/1998/REC-xml-19980210, Feb. 1998.

[2] Christian Mathis, Theo Härder-"A Query Processing Approach for XML Database Systems". www.lgis.informatik.unikl.de/cms/fileadmin/users/mathis/.../gws2.pdf

[3] C. Mathis and T. Harder. "A Query Processing Approach for XML Database Systems". 2005.

[4] D. Maier. "Database Desire data for XML Query Language".http://www.w3.org/TandS/QL/QL98/pp /maier.html

[5] J. Naughton et al. The Niagara Internet Query System. In IEEE Data Engineering Bulletin vol 24 issue 2. 2001

[6] H. V. Jagadish et al. "A Native XML Database". In International Conference of VLDB. 2002

[7] Dunren Che, Karl Aberer, M. Tamer Özsu. "Query optimization in XML structured-document databases". The VLDB Journal (2006) DOI 10.1007/s00778-005-0172-6. Published online: 28 April 2006 _c Springer-Verlag 2005

[8] Gottlob, G., Koch, C., Pichler, R.: "Efficient algorithms for processing XPath queries. In: Proceedings of VLDB, Hongkong, China (2002).

[9] Chan, C.-Y., Felber, P., Garofalakis, M., Rastogi, R.: "Efficient filtering of XML documents with XPath expressions". In: Proceedings of International Conference on Data Engineering, San Jose, California, February 2002, pp. 235–244 (2002).

[10] Su Cheng Haw, and G. S. V. Radha Krishna Rao. "Query Optimization Techniques for XML Database" World Academy of Science, Engineering and Technology 22 2006.

[11] M. Atay, Y. Sun, D. Liu, S. Lu, F. Fotouhi, "Mapping XML Data To Relational Data: A DOM-Based Approach", Proc. Of the 8th IASTED International Conference on Internet and Multimedia Systems and Applications, 2004, pp. 59-64.

[12] T.S. Chung, H-J Kim, "Techniques for the evaluation of XML queries: a survey", ACM Data and Knowledge Engineering 46, 2003, pp. 225-246.

[13] M. Garafalakis, A. Gionis, R. Rastpgo, S. Seshadri, K. Shim, "XTRACT: a system for extracting document type descriptors from XML documents", Proceeding of the ACM SIGMOD Int. Conference on the Management of Data, 2000, pp. 165-176.