ISSN No. 0976-5697

Volume 3, No. 3, May-June 2012

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Theory and Practices in Xml Query Optimization

[1]Asmita P. Asre
PG Student, Department of Computer Sc. & Engg.
Prof. Ram Meghe Institute of Tech. & Research, Badnera.
asmitaasre@gmail.com

[2]Prof. M.S.Ali
Principal, Prof. Ram Meghe College
Of Engineering & Management, Badnera.
softalis@hotmail.com

Abstract: As computers and technology continue to become more commonplace and essential to everyday life, more data is captured, stored, and analyzed by a variety of institutions. As this amount of data grows, so does the need for efficient methodologies and tools used to store, retrieve, and transform the data. A common method used to store this schemaless, semi-structured data is through the Extensible Markup Language, XML. In this way, an XML document is viewed as a database. With this sizable amount of data stored in a common format, one problem is how to efficiently query XML documents. While relational database management systems contain built-in query optimizers, no such framework exists for XML databases. A multitude of document shapes, query shapes, index structures, and query techniques exist for XML databases, but the implications of these choices and their effects on query processing have not been investigated in a common framework. This paper focuses on the theory and practices applied to solve the problem of query optimization in XML databases.

Keywords: Query Optimization, XML Databases, Cost Model, Query Optimization issues.

I. INTRODUCTION

As computers and technology become more commonplace and essential to everyday life, more and more data is captured, stored, and analyzed by a variety of institutions. As this amount of available data grows, so does the need for efficient methodologies and tools used to store, retrieve, and perform operations on the data. The relational model was first proposed by Codd in 1970 [**1**] as a way of describing data using only its natural structure. Specifically, the natural structure of the data refers to the relations between data elements. It is based on the notions of set theory and first order predicate logic and has, at its core, the idea of a mathematical relation as the basic building block.

Data in the relational model must conform to a global schema. A relational schema is typically developed by a database administrator before data is loaded into the system. As the relational model gained popularity, it inspired many end-user database management systems (DBMS) to be created using it as a theoretical backbone. Since relational algebra (the mathematical notation used to manipulate relational data) can be complex, a higher-level query language was developed to ease user interaction with the DBMS. The Structured Query Language (SQL) was standardized by the American National Standards Institute (ANSI) and the International Standards Organization (ISO) in 1986.

This version of SQL was revised and expanded in 1992 and is commonly referred to as SQL-92. While SQL allows complex queries to be written and executed, it does not optimize queries to improve performance and query return times. In order to improve query return time, commercial DBMS packages currently include query optimization techniques built-in to the software. These types of optimizations fall into two categories: logical and physical (Figure 1). When a SQL query is presented to the database, the first step is logical optimization. The high-level SQL query is converted to a corresponding relational algebra tree. Transformations are then performed on the tree in order to optimize the query, i.e., reduce the data retrieved and operated on. The goal of logical optimization is to rewrite the user query into an equivalent form that is more efficient to execute.

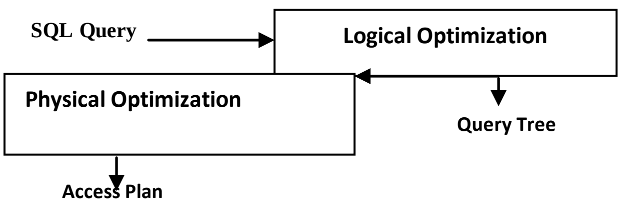


Figure 1. Traditional Query Optimization

The result of logical optimization is an equivalent query tree, and this tree is then passed on for physical optimization. Physical optimization takes into account file organization and auxiliary access and mechanisms. How the data is stored on disk and the indexes or other access methods available to the database are crucial in retrieving the requested data quickly. The DBMS is aware of the physical storage and auxiliary access methods available to the system. Since there is always a cost to access the data on disk, choosing an efficient access plan among all possible choices is referred to as cost-based optimization. The relational model and associated optimization techniques are mature technologies. When data is highly-structured and uses a well-defined schema, relational databases are an excellent choice for storing and accessing data. However, with the growth of the Internet in the past decade, new ways of structuring and describing data have become available. One such data model, XML, is discussed below. These new types of data present challenges for traditional query processing and optimization techniques.

A. XML and OEM:

Most data on the web is said to be semistructured or loosely-structured data as well as schemaless or self-describing. In other words, unlike data in the relational model, there exists little or no metadata [ABS00] separate from the data itself. The Extensible Markup Language (XML) is a new standard for data exchange on the Internet and between different processing platforms. An open-standard specification for XML is kept by the W3C [xml].

While XML is syntactically similar to HTML, it does more than simply specify the appearance of text on a page. Data represented in XML is self-describing, i.e., it contains embedded descriptive information, and generally does not require an outside schema.

A brief example of an XML document is shown in Figure 1.1. Information is represented both in the text and the tags around the text. The two main methods to represent data are as elements or attributes. An example of an element if shown in line 3 of Figure 1.1. The element identifier is name and the corresponding element value is Chili's. Information can also be represented as

```
<FoodDrink>
<restaurant id=''R001''>
<name>Chili's</name>
 <phone>671-1102</phone>
 <owner>G. Haldiram</owner> </restaurant>
 <restaurant id=''R002''>
<name>Maggi's</name>
 <owner>G. Peppard</owner>
<manager>Crow</manager>
 </restaurant>
 <bar id=''B001''>
<name>Crow</name>
<style>Indian</style>
</bar>
</FoodDrink>
```

Figure1.1: XML Example

An attribute of an element (*as shown in line 2*). The element restaurant has an attribute of R001. The nesting of XML elements gives it a tree (or graph) structure, and this yields information about hierarchical relationships (such as parent-child or ancestor-descendant) in the data. While XML is robust and highly-adaptable (attributes, elements, and element tags can be dynamically specified and defined by the user), it can be somewhat daunting to read and understand. The Object Exchange Model (OEM) was proposed in 1995 [PGMW95], and it serves as a diagrammatical representation for XML documents. Data represented in OEM is self-describing and therefore does not require additional schema definitions. An object in OEM is defined as the quadruple (label, oid, type, value). The variable label gives a character label to the object, oid provides the object's unique identifier, and type can be either an atomic value or complex. If type is an atomic value, then the object is an atomic object and value is an atomic value of the corresponding type. Otherwise, if type is complex, then the object is a complex object and value is a list of object identifiers (oids) [**2**]. The OEM retains the simplicity of relational models but allows some of the flexibility given by object-oriented models [**2**] for specifying nested objects. OEM is one example of a graphical convention used to display an XML document. It is important because the document has an inherent structure, data labels, and data that are readily visible to the reader.

B. XPath and XQuery:

The simplest type of query in XML is an XPath expression [3]. XPath expressions resemble the UNIX directory structure with some extensions. The slash (/) and double-slash (//) retain their UNIX interpretations (parent-child and ancestor-descendent relationship, respectively), and the text in brackets ([ ]) acts as a filter on the data to be returned. An example of a simple XPath expression is given by /FoodDrink/Restaurant [owner='G.Haldiram'] and corresponds to the XML document shown in Figure 1.1.

This expression results in a positive match to two restaurant nodes, one with id equal to R001 and the other with id equal to R002. The single slash represents a strict parent-child relationship. The expression //[style='Indian'] matches only one node, the bar node with id 6 equal to B001. The double-slash represents an ancestor-descendant relationship. In this case, we are only interested in nodes that, at some point in their list of descendants, have a style of Indian.

X Query is a query language for XML designed to be broadly applicable across many types of XML sources. Designed to meet the requirements identified by the World Wide Web Consortium (**W3C**), XQuery operates on the logical structure of an XML document, and it has both human-readable syntax and XML-based syntax. A grammar for XQuery is defined by the W3C [**3**]. While XQuery can successfully extract information from XML documents, there are no built-in optimization techniques that relate to the relational optimization techniques. The current version of XQuery (1.0) is an extension of XPath 2.0.

C. Native and non-native techniques:

There currently exist two broad methodologies, native and non-native techniques, used to query XML documents. Native techniques implement XML queries on XML documents. The original document, while perhaps slightly transformed, maintains the inherent properties of an XML document.

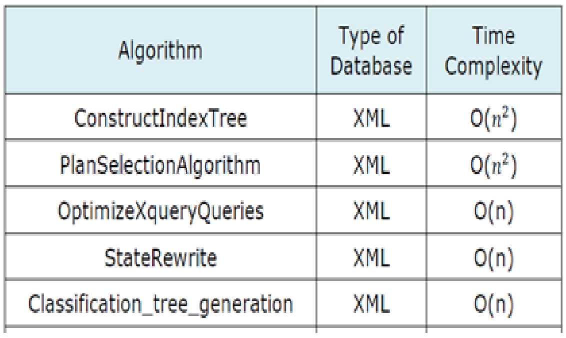
This means that the document is tree shaped, has both depth and breadth, and is constructed by linking individual nodes (elements) together. In contrast, non-native techniques transform the original XML document into another format that is not XML. An example of a non-native technique is to take an XML document, flatten it, and store the contents in a relational database. Some of these techniques allow standard XPath expressions to be executed over the transformed data, but the underlying document is no longer an XML file.

II. XML PERSPECTIVE

In the last few years, XML became the de-facto standard for exchanging structured and semi-structured data in business as well as in research. The database research community took this development into account by proposing among others—native *XML database management systems* (XDBMSs) for efficient and transactional processing of XML documents. As in the relational world, the quality of query optimizers plays an important role for the acceptance of database systems by a wide range of users, especially in business scenarios where longer-than-necessary running queries can cause high costs. One of the main tasks in query evaluation is plan generation, where physical operators are arranged in such a way, that the given optimization goal (e. g., maximum throughput) is satisfied while the semantics of the query is still preserved. In recent years, several join operators for the evaluation of structural relationships like *child* or *descendant* have been proposed. All of them belong to one of the major classes of XML join operators: *Structural Joins* [**4**] and *Holistic Twig Joins* [**5**]. Being binary join operators, SJ operators decompose tree-structured query patterns, which are also called twig query patterns, into binary relationships and evaluate each of them separately, before they merge intermediate results to get the final query result. On the other hand, HTJ operators are able to evaluate twigs holistically. A precondition for the efficient evaluation of SJ and HTJ operators is a node labeling scheme [**6**] that assigns to each node in an XML document a unique identifier that (1) allows to decide, without accessing the document, for two given nodes whether they are structurally related to each other and (2) that does not require re-labeling even after modifications to the document. Besides SJ and HTJ operators, several approaches for indexing XML documents were proposed.

These approaches can be classified into a hierarchy of access methods w. r. t. their availability in a native XDBMS. *Primary access paths* provide input for navigational primitives as well as for SJ and HTJ operators. The most important representative of this class is a *document index* that indexes a document using the unique node labels as keys. *Secondary access paths1* provide more efficient access to specific element nodes using *element indexes*. They are absolutely necessary for efficient evaluation of structural predicates by SJ or HTJ operators. *Tertiary access path)* like *path indexes* [**7**] employ structural summaries such as *Data* **[8]** for providing efficient access to nodes satisfying structural relationships like *child* or *descendant*. *Content indexes* [**9**] support efficient access to text nodes or attribute-value nodes. Finally, *hybrid indexes* (Wang, Park, Fan, & Yu, 2003), which are also called *content-and-structure (CAS) indexes*, are a promising approach for indexing content and structure at a time. Compared to PAPs, which are available per default in a native XDBMS, SAPs and TAPs have to be manually created by the database administrator. Furthermore, TAPs like path indexes or CAS indexes can replace complete trees of SJ and HTJ operators and become—if available—first-class citizens for query evaluation. As maintenance and updates of TAPs can cause substantial overhead, they will only be created by the database administrator in rare cases, e.g., for frequently queried sub trees. **[9]**

Table I. Comparison of Algorithms



Algorithm	Type of Database	Time Complexity
ConstructIndexTree	XML	$O(n^2)$
PlanSelectionAlgorithm	XML	$O(n^2)$
OptimizeXqueryQueries	XML	O(n)
StateRewrite	XML	O(n)
Classification_tree_generation	XML	O(n)

Table I. Focuses and provides the different approaches and the baseline provided earlier to solve the problem of query optimization for XML databases.

III. CONCLUSION

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 and indexing schemes. Our approach of minimizing time to execute a query, will certainly add to the available solutions in the context of query optimization in XML databases.

IV. REFERENCES

[1]. E. F. Codd. A relational model of data for large shared data banks. Communications of the ACM, 13(6):377– 387, 1970.

[2]. Serge Abiteboul, Peter Buneman, and Dan Suciu. Data on the Web. Morgan Kaufmann Publishers Inc., 2000.

[3]. R. G. G. Cattell, Douglas K. Barry, Dirk Bartels, Mark Berler, Jeff Eastman, Sophie Gamerman, David Jordan, Adam Springer, Henry Strickland,and Drew Wade. The Object Database Standard: ODMG 2.0. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[4]. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/2005/CR-xquery-20051103/, July 30 2009.

[5]. Zhang; Al-Khalifa. Open and Novel Issues in XML Database Applications: books.google.co.in/books?isbn=160566308 5

[6]. N. Bruno, N. Koudas, and D. Srivastava B+ Tree Based Indexing Scheme for FLOWR Queries on XML www.csjournals.com/IJCSC/PDF1-2/18..pdf

[7]. Harder. Haustein, Mathis, & Wagner. A Framework for Cost-Based Query Optimization in Native XML. www.irma int ernational.org/vie wtitle/41504

[8]. Tova Milo, Dan Suciu. Typechecking for XML transformers Xml path language dl.acm.org/citation.cfm?id=33516 8.33 5171

[9]. Roy Goldman, Dallan Quass, Jennifer Widom. Lore: A database management system for XML citeseer.ist.psu.edu/viewdoc/summary?doi = 10.1.1.41.3062

[10]. Jason Mchugh and Jennifer Widom Query Optimization for XML citeseerx. ist. psu. edu/ viewdoc/ summary? do i=10.1.1.1.2777