Analyzing Performance of Hierarchical Queries in XED

V. Naveen kumar, A. V. H. Sai Prasad, G. Ramesh babu

Abstract—The increase utilization of XML structure for data representation, exchange, and integration has strengthened the need for an efficient storage and retrieval of XML data. Currently, there are two major streams of XML data repositories. The first stream is the Native XML database systems which are built solely to store and manipulate XML data, and equipped with the standard XML query language known as XPath and XQuery. The second stream is the XML-Enabled database systems which are generally existing traditional database systems enhanced with XML storage capabilities. The SQL/XML standard for XML querying is used in these enabled database systems stream. The main specific characteristic of this standard is the fact that XPath and XQuery are embedded within SQL statements. The focus of this paper is to present taxonomy of different hierarchical query patterns in XML-Enabled database environment, and to analyze the performance of the different query structures using the SQL/XML standard.

Keywords—XED, TKPROF Utility, NXD, XPATH, XQuery.

I. INTRODUCTION

To date, there are two well-known streams in this area: the Native XML Database (NXD) and the XML-Enabled Database (XED). The NXD is an XML database system that is built solely to store and manipulate XML data structures, and utilizes pure XQuery as its query language. The XED is generally developed incrementally from another existing database system. XED that has relational database as foundation will support SQL as well as XML query languages such as XPath and XQuery. This extended SQL is widely known as SQL/XML [ISO/IEC, 2003]. This fact has made query performance analysis in XED systems very unique. The analysis has to consider the nature of XML tree as well as the relational-based performance measurement.

Here we compare and analyze the performance of different querying techniques and propose a recommendation to determine which query writing is suitable for various SQL/XML queries. To achieve this, we start by identifying the classification of XML queries based on the nature of the target retrieval and path traversal. We identify several types of query path traversal including: Child, Descendant, Parent, and Ancestor queries, whereby each of them may be represented as sub-tree, element, attribute, or text node. In our SQL/XML query performance experiments; we consider two dimensional aspects whereby each aspect is a measure relevant to SQL and XML respectively. The first aspect of measure is the XML path analysis which is based on above classification of XML query path traversal. The second aspect of measure is the level of hierarchy analysis which is based on the placement of SQL selection and projection at different levels of XML hierarchy. In the experimental testing, we used a major XED product, Oracle 10g Release 2. XML Enabled Databases (XED) can be defined as established databases that support the storage and manipulation of XML data as a document. Most XEDs were developed from relational databases. In terms of XML solution, there is a fundamental difference between the relational database and the XED.

In pure relational database, the XML data will be stored in a table column as a large object or by shredding the tree elements into atomic values. This data type considers the tree structure of XML data and facilitates XML Query languages and standards. The difference between storing XML data in relational database and in XED. Many data management areas for XED are still unexplored. XED requires mechanism to define how a tree or partial tree is stored in the table column. XED also requires special languages to retrieve and manipulate the data as XML Type. SQL/XML standard provides a uniform language for managing data in XED.

However, the standard language still have different syntax structure, which obviously determined by the operations behind each structure. Unlike research work in storing XML data in Relational Database, research in storing and manipulating XML data in XED through special types is still rarely found. Of the few discussed the update operations for XML data in XED through trigger mechanism. We will attempt to investigate this issue. Before we describe these various syntaxes, we show the classification of the query based on the projection and selection target.

II. RELATED WORK

In RDBMS, the XML data will be stored in a table column as a large object or by shredding the tree elements into atomic values. This data type considers the tree structure of XML data and facilitates XML Query languages and standards. Figure 1 depicts the difference between storing XML data in relational database and in XED. Many data management areas for XED are still unexplored. XED requires mechanism to define how a tree or partial tree is stored in the table column. XED also requires special languages to retrieve and manipulate the data as XML Type. SQL/XML standard provides a uniform language for managing data in XED. The standard language still have different syntax structure, which obviously determined by the operations behind each structure. Unlike research work in storing XML data in Relational Database, research in storing and manipulating XML data in XED through special types is still rarely found. Of the few discussed the update operations for XML data in XED through trigger mechanism. We will attempt to investigate this issue. Before we describe these various syntaxes, we show the classification of the query based on the projection and selection target.
III. XML ENABLED DATABASE

XML Enabled Databases (XED) can be defined as established databases that support the storage and manipulation of XML data as a document. Most XEDs were developed from relational databases. In terms of XML solution, there is a fundamental difference between the relational database and the XED [2].

![Figure 1: Storing XML Datas: Relational Databases vs XED](image)

In pure relational database, the XML data will be stored in a table column as a large object or by shredding the tree elements into atomic values. Many researches in the last 10 years have offered this solution. On the other hand, XED stores the XML data as a unique type such as Multiple. This data type considers the tree structure of XML data and facilitates XML Query languages and standards. Figure 1 depicts the difference between storing XML data in relational database and in XED. Many data management areas for XED are still unexplored. XED requires mechanism to define how a tree or partial tree is stored in the table column. XED also requires special languages to retrieve and manipulate the data as XML Type. SQL/XML standard provides a uniform language for managing data in XED. However, the standard language still have different syntax structure, which obviously determined by the operations behind each structure. Unlike research work in storing XML data in Relational Database, research in storing and manipulating XML data in XED through special types is still rarely found. Of the few, discussed the update operations for XML data in XED through trigger mechanism. Provided work on semantic transformation of queries in XED. None of the work has investigated the various SQL/XML syntax structure for managing XML data in XED and how they affect the query performance. We will attempt to investigate this issue. Before we describe these various syntaxes, we show the classification of the query based on the projection and selection target.

IV. TAXONOMY ON XML QUERY

Based on the retrieval target, we classify XML Query into three types:

- **Child/Descendant (C/D) Queries, Parent Queries and Ancestor Queries.** Each of them can be further grouped into the target structure; either it is a sub-tree, an element or an attribute [3].

![Figure 2: XML-Enabled Database Query Classification](image)

V. SUB-TREE QUERIES

These are the queries which involve the retrieval of C/D element(s) based on the selection on the parent, ancestor or sibling element. For example, figure 3 shows a tree representation for C/D retrieval (node D), having selection on the parent element (node B), ancestor element (node A) and sibling element (node E). From the figure, we can see that this query type retrieve node D and its sub-elements. All these queries represented as tree structure can be written using XPath extract Query, XMLTABLE XQuery and XMLQuery XQuery syntax.

VI. TEXT NODE QUERIES

These queries involve with the projection of a text node associated with the C/D element based on the selection on the parent, ancestor or sibling element. For example, figure 3 shows the output of the query is the text associated with element D.In XED, these queries can be performed using SQL function text() in XPath XMLSequence Query and XPath extract Query. The changed syntax of XPath extract Query for Text Node query is given below:

```sql
SELECT _VALUE,'//<projection_element>/text ()')
FROM <table_name> <alias>
WHERE existsNode(<alias>.OBJECT_VALUE,
'//<selection_condition>'
); Attribute Queries

These queries involve in the projection of attributes associated with the C/D element based on the selection on the parent, ancestor or sibling element. For example, Attribute queries use XPath extract Query, XMLTABLE XQuery and XMLQuery XQuery with a little modification. The changed syntax of above mentioned queries in order to
retrieve attribute associated with C/D elements is given below.

XPath extract attribute Query: SELECT extract(<alias>.OBJECT_VALUE, '/<element_having_attribute>/@<projection_attribute>') FROM <table_name> <alias> WHERE existsNode(<alias>.OBJECT_VALUE, '//<selection_condition>');

XMLTABLE attribute XQuery:
SELECT xtab.COLUMN_VALUE FROM <table_name>, XMLTABLE('<!--XQuery Expression using FLOWR Expressions--> for $a in //<element_node> where <selection_predicate[conditional_expression]> RETURN $a/<element_having_attribute>/@<projection_attribute>' PASSING OBJECT_VALUE) xtab;

XMLQuery attribute XQuery:
SELECT XMLQuery('<!--XQuery Expression using FLOWR Expressions--> for $a in //<element_node> where <selection_predicate[conditional_expression]> RETURN $a/<element_having_attribute>/@<projection_attribute>' PASSING OBJECT_VALUE RETURNING CONTENT) FROM <table_name>;

VII. CONCLUSION

The performance of these queries with the aim of finding most optimized query structure for any particular query operations. We applied these queries for two main classifications based on selection and projection predicates, namely child/descendant queries and parent/ancestor queries. The performance of each query by using the statistics generated by the database. The most optimized query structure was identified in terms of cost associated with the execution of the query.

REFERENCES


