Journal of Universal Computer Science, vol. 15, no. 10 (2009), 2058-2077
submitted: 15/1/09, accepted: 5/5/09, appeared: 28/5/09 © J.UCS

SQL/XML Hierarchical Query Performance Analysis in an XML-Enabled Database System

Eric Pardede, J. Wenny Rahayu, Ramanpreet Kauer Aujla
(Department of Computer Science and Computer Engineering, La Trobe University
Melbourne, Australia
{E.Pardede, W.Rahayu}@latrobe.edu.au)

David Taniar
(Clayton School of Information Technology, Monash University, Melbourne, Australia
David.Taniar@infotech.monash.edu.au)

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. To date, most existing work in XML query analysis have been focussing on the first stream of Native XML database systems. The focus of this paper is to present a 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: XML, SQL/XML Query, XML-Enabled Database
Categories: H.2.7

1 Introduction

The distinctiveness of XML data structure has driven the emergence of a new stream of database system known as the XML database systems. To date, there are two well-known streams in this area: the *Native XML Database* (NXD) and the *XML-Enabled Database* (XED).

For the purpose of this paper, we define the main differences between the two streams as follows. The NDX is an XML database system that is built solely to store and manipulate XML data structures, and utilizes pure XQuery [W3C, 2007] as its query language. On the other hand, 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.

In this paper, 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 [Oracle, 2007].

2 Related Work

Some of the existing works used approaches that are based on how the XML is stored in XED. For example, [Zhang, 2001] investigated the query optimization using loosely-coupled information retrieval engine and using DBMS tables with query execution engines. It shows the effect of join algorithm and hardware cache utilization towards containment queries performance. Unfortunately, there is no detail discussion on how to improve the performance, so that, the containment queries can be efficiently processed. In this paper, we use a different approach. We will only focus on the usage of DBMS tables and query execution engine for query optimization. However, we will show how to improve the query performance for a range of SQL/XML query types.

Other works focused on investigating translation from XML standard to XED language. [Wang, 2005] proposed an algebraic approach for order-sensitive XQuery processing over relational databases, which is called XQuery-to-SQL Order-sensitive Translation (XSOT). This work has more focus on direct rewriting from one query family to another query family.

[Zhang, 2002] investigated the algorithm using XML Algebra Tree (XAT) for explicitly represents the semantics of XQuery. Unfortunately, it does not clearly describe how it is applied in XED language such as SQL or SQL/XML. A similar approach is taken by [Grinev, 2005], where the authors proposed a technique to rewrite XQuery by using an XML element constructor. It provided XQuery samples and their optimized result. However, there is no justification for the performance evaluation.

[Le, 2007] proposed semantic transformation algorithms for querying data in XML database. The authors proposed to pre-process XML schema before any retrieval. They also proposed to rewrite the XPath in SQL/XML query languages. In [Le, 2008], they applied the approach for update operations. While the works claimed to have improved the query and update operations, they are only applied to XML data with existing schema. In addition, the semantic path transformation is also limited to certain paths only.

Another research [De Meo, 2004] also investigated the use of semantic concepts to manage XML documents. However, it is more targeted for XML data integration instead of for query optimization or query performance analysis.

Finally [Sun, 2006] used Ontology semantics for rewriting XML Query. They proposed a set of rules that can be used for query optimization. There is no clear guideline on how these rules can be applied in SQL/XML query and XPath in general.

To the best of our knowledge, there has no attempt to solve the query optimization problem from both structural and semantic properties of XML documents. In this paper, we aim to consider both SQL/XML query structure and semantic query classification for performance analysis.

3 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.

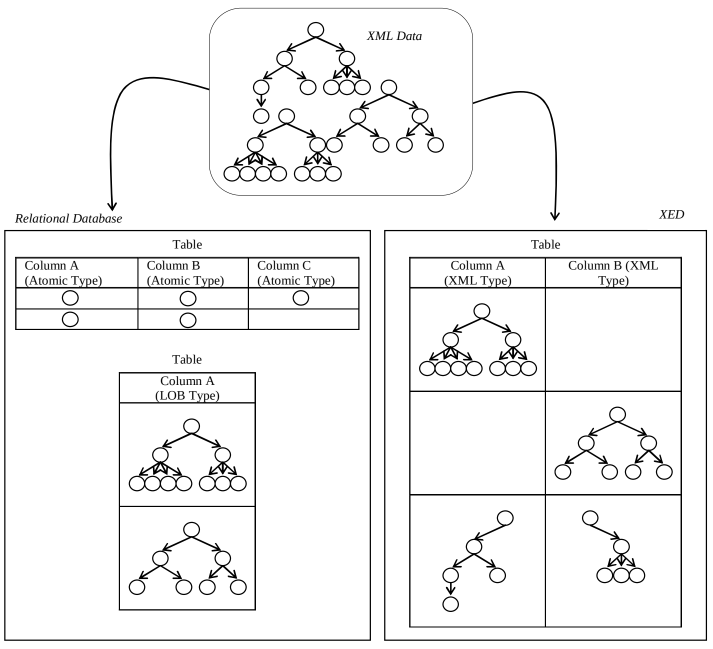

Figure 1: Storing XML Data: Relational Database vs XED

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. Many researches in the last 10 years have offered this solution [Florescu, 1999; Shanmugasundaram, 1999; Pardede, 2004]. On the other hand, XED stores the XML data as a unique type such as XMLType. 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, [Pardede, 2008a] discussed the update operations for XML data in XED through trigger mechanism. [Le, 2007] 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.

4 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* (see figure 2). Each of them can be further grouped into the target structure; either it is a sub-tree, an element or an attribute.

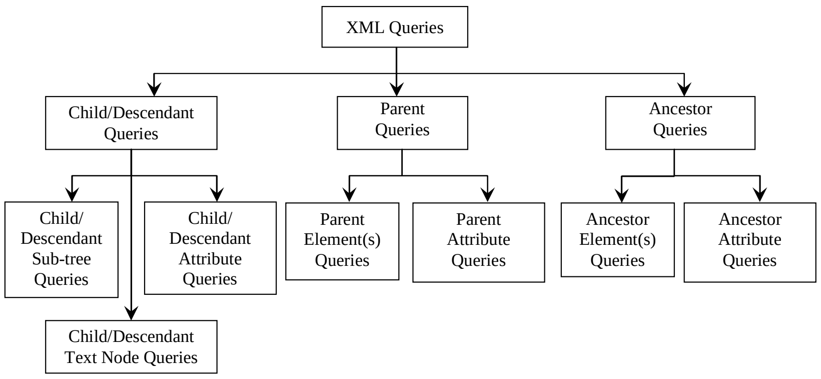
XML Queries
Child/Descendant Queries
Parent Queries
Ancestor Queries
Child/ Descendant Sub-tree Queries
Child/ Descendant Attribute Queries
Parent Element(s) Queries
Parent Attribute Queries
Ancestor Element(s) Queries
Ancestor Attribute Queries
Child/Descendant Text Node Queries

Figure 2: XML-Enabled Database Query Classification

The main idea behind this classification is to check the performance difference for selection at the different levels of hierarchy while fixing the projection at a single node. The query implementation is dependent on the database type. Even in one database family, such as XED, the implementation can be different from one product to another. We discuss about the variation in the next section.

4.1 C/D Queries Classification

C/D queries are further classified into three types: C/D Sub-tree queries, C/D Text Node queries and C/D Attribute queries. For the rest of the paper, we will refer them as sub-tree, text node and attribute query. This sub-division is based on the projection target. Sub-tree queries retrieve whole C/D nodes including text node and attributes associated with it. Text Node queries and Attribute queries retrieve just the text node of an element and attribute respectively.

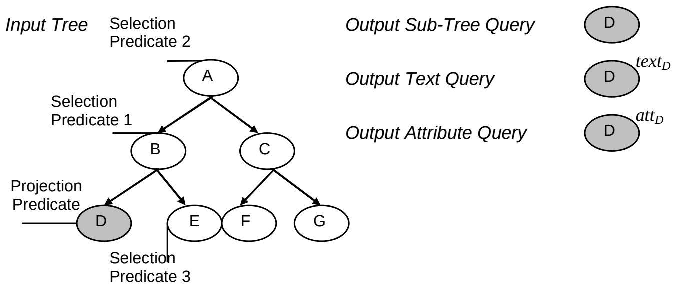

Figure 3: Tree Representation for C/D Queries

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.

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:

```
SELECT extract(<alias>.OBJECT_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, Figure 3 shows the output of the query is the attribute associated with node D. 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>;
```

4.2 Parent and Ancestor Queries Classification

Parent and Ancestor Queries are further classified based on the projection target, into Element(s) Queries and Attribute Queries. Element(s) queries retrieve the whole element node including child elements and attributes associated with it, while Attribute queries retrieve just the attribute associated with the parent/ancestor element.

Parent Element(s) Queries

These are the queries which involve the retrieval of the Parent element(s) based on the selection on the child element. For example, figure 4 shows a tree representation for element retrieval (node *B*), having selection on the child element (node *D*). From the figure, we can see that this query type retrieve node *B* and its sub-elements. All these queries represented as tree structure can be written using XPath extract Query, XMLTABLE XQuery and XMLQuery XQuery syntax.

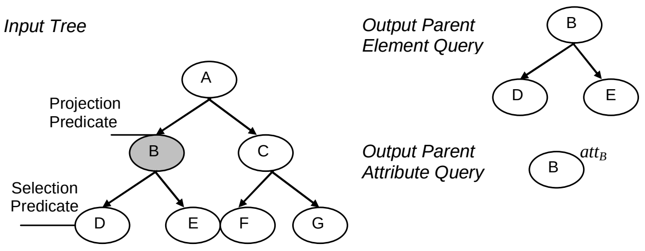

Figure 4: Tree Representation for Parent Queries

Parent Attribute Queries

These queries involve with the projection of attributes associated with the parent element based on the selection on the child element. For example in figure 4, the output of the parent attribute query is the attribute associated with element *B*.

Attribute queries use XPath extract Query, XMLTABLE XQuery and XMLQuery XQuery with a little modification. The changed syntax of the above mentioned queries in order to retrieve attribute associated with Parent element 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>;
```

Ancestor Element(s) Queries

Ancestor Element(s) queries involve with the retrieval of the Ancestor element(s) based on the selection on the descendant element. For example, figure 5 shows the tree representation for element retrieval (node *A*), having selection on the descendant element (node *D*). From the figure, we can see that element *A* is on the top hierarchy. If the descendant node satisfies the selection condition, then the whole input tree will be output tree. All these queries represented as tree structure can be written using XPath extract Query, XMLTABLE XQuery and XMLQuery XQuery syntax.

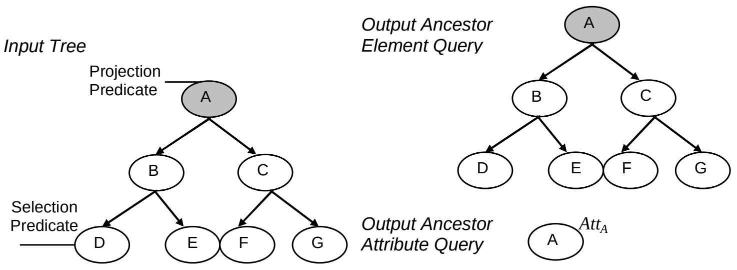

Figure 5: Tree Representation for Ancestor Queries

Ancestor Attribute Queries

Ancestor Attribute queries involve with the projection of attributes associated with an element based on the selection on the descendant element. For example, in figure 5, the retrieval of attribute(s) associated with element *A* having selection predicate on descendant element *D*. This query represented as tree structure can be easily implemented with XPath extract attribute Query, XMLTABLE attribute XQuery and XMLQuery attribute XQuery syntaxes.

5 Queries Performance Experiments

5.1 Experimental Setup and Tool

For all practical purposes, Oracle 10g Release 2 [Oracle, 2007] was used for this paper. The experiments were performed on a Windows workstation having 40 GB of disk storage and 504MB of memory. XML Schema was created based on a publicly available XML document (http://www.dia.uniroma3.it/Araneus/Sigmod/Record/SigmodRecord/SigmodRecord.xml)

The size of the data used for the experiment was 1.01MB. The table was created based on the XML Schema. The XML documents stored contain not only element nodes but also attribute and text nodes. TKPROF utility and Timing utility [Oracle, 2007] were used to do performance analysis. TKPROF utility [Srivastava, 2002], also known as SQL Trace, reports information about each SQL statement executed with the resources it has used, the number of times it was called, and the number of rows which it processed.

Performance analysis of the queries involves the comparison of the statistical information of the queries. The information is based on the following parameters: (i)

CPU Time, refers to time taken to parse, execute, or fetch calls for the statement. Query performance is shown by Execute and Fetch phases; (ii) TKPROF Elapsed Time, refers to the time required for all parsing, executing, or fetching calls for the statement. This is the total time from start to finish of the execution of the query; and (iii) ***Timing Elapsed Time, which is*** the total execution time of the query obtained by timing utility of SQL*Plus.

5.2 Experimentation

Sub-tree Queries Experimentation. *"Retrieve title C/D element when the selection is on parent/ancestor/sibling element."*

The query for this problem can be categorized into six, based on different combinations of projections and selections in the hierarchy [Pardede, 2007]. The projection predicate is fixed at *title* element but the selection predicate is located at different levels of hierarchy. Each category can have four ways of query writing.

Figure 6 represents the selection and the projection predicates for one category (denoted with A) , if the *selection predicate is located at first level ancestor element*. In this case, we *retrieve title descendant element when attribute textValue1 (ancestor element) is 50*. The projection predicate is *title* and the selection predicate is *articles* (textValue1 attribute) located at the fourth level of hierarchy (see figure 6). Queries below represent the possible structures to perform the operation. Queries 1 and 4 are written using XPath extract Query. Query 4 is written using the same query structure as Query 1. However, the way it is written is different. Query 2 is written using XMLTABLE XQuery. Query 3 is written using XMLQuery XQuery.

```
Query 1:
SELECT extract(s.OBJECT_VALUE, '//title')"TITLE"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,
   '//articles[@textValue1="50"]')= 1;

Query 2:
SELECT xtab.COLUMN_VALUE "TITLE"
FROM SIGMOD, XMLTABLE ('for $a in //articles let
      $b:=$a[@textValue1="50"]
     return $b//title' PASSING OBJECT_VALUE)xtab;

Query 3:
SELECT XMLQuery('for $a in //articles return $a//title'
    PASSING OBJECT_VALUE RETURNING CONTENT)"TITLE"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,
   '//articles[@textValue1="50"]')= 1;

Query 4:
SELECT extract(s.OBJECT_VALUE,
      '/SigmodRecord/*//articles/*//title') TITLE"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,'/SigmodRecord/*//articles
      [@textValue1="50"]')= 1;
```

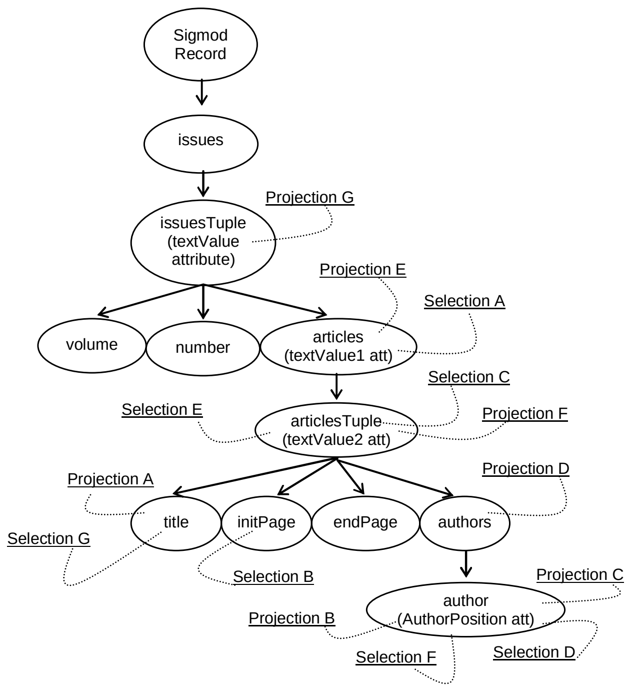

Figure 6: Query Tree Representation

Text Node Queries Experimentation. *"Retrieve Text Node associated with author/volume C/D element when the selection is on a Parent/Ancestor/Sibling element."*

The Text Node Queries are based on projection of two different elements. One of which is the topmost element (*volume*) and the other is the bottom-most element (*author*). Based on the projection and the selection nodes, we can classify this query into 12 categories [Pardede, 2007].

Figure 6 represents selection and projection predicate for one of the category (denoted with B), *if the selection predicate is on element which is a sibling to the parent element of the projection predicate.* In this case, we *retrieve text node of author when element initPage is 114 or 144.* The projection predicate for this query is *author* element and the selection predicate is *initPage* element located at sixth level of hierarchy. The following queries represent the possible structures to perform the operation. Query 5 and Query 7 are written using XPath extract Query with `text()` function. Query 7 is written using the same query structure as Query 5 but the path in Query 7 is written in different way. Query 6 is written using XPath XMLSequence Query. This query takes XPath expression as input.

```
Query 5:
SELECT extract(s.OBJECT_VALUE, '//author/text()') "AUTHORS"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,'//
      articlesTuple[initPage="144" or initPage="114"]')= 1;

Query 6:
SELECT extractValue(value(AUTHORS), '/author')"AUTHORS"
FROM SIGMOD s, table(XMLSequence(extract(s.OBJECT_VALUE,
   '//author')))AUTHORS
WHERE existsNode(s.OBJECT_VALUE,'
   //articlesTuple[initPage="144" or initPage="114"]')= 1;

Query 7:
SELECT extract(s.OBJECT_VALUE, '/SigmodRecord/*//
      articlesTuple/*//author/text()') "AUTHORS"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,'//articlesTuple
      [initPage="144" or initPage="114"]')= 1;
```

Attribute Queries Experimentation. *"Retrieve Attributes associated with C/D element when the selection is on a parent/ancestor element."*

Based on the projection and the selection nodes, we can classify this query into 6 categories [Pardede, 2007]. For our case study, figure 6 represents one category (denoted with C), in which the *selection pred. at the first level ancestor element and the projection predicate is at lowest level of hierarchy.* In this case, we *retrieve the AuthorPosition descendant attribute when the attribute textValue2 (ancestor element) is 51 or 52.*

The following queries represent the possible structures to perform the operation. Query 8 and 11 are written using XPath extract attribute query. Query 9 is written using XMLTABLE attribute XQuery. Query 10 is written using XMLQuery attribute XQuery.

```
Query 8:
SELECT extract(s.OBJECT_VALUE, '//@AuthorPosition')
      "AUTHORPOSITION"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,'//articlesTuple
      [@textValue2="51" or @textValue2="52"]')= 1;

Query 9:
SELECT xtab.COLUMN_VALUE "AUTHORPOSITION"
FROM SIGMOD, XMLTABLE
    ('for $a in //articlesTuple
      where $a/@textValue2 = "51" or $a/@textValue2 = "52"
      return $a//@AuthorPosition'
```

```
       PASSING OBJECT_VALUE)xtab;

Query 10:
SELECT XMLQuery
    ('for $a in //articlesTuple
       return $a//@AuthorPosition'
       PASSING OBJECT_VALUE RETURNING CONTENT)"AUTHORPOSITION"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,'//articlesTuple
      [@textValue2="51" or @textValue2="52"]')= 1;

Query 11:
SELECT extract(s.OBJECT_VALUE, '/SigmodRecord/*//
      articlesTuple/*//@AuthorPosition') "AUTHORPOSITION"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,'/SigmodRecord/*//
      articlesTuple[@textValue2="51" or @textValue2="52"]')=
      1;
```

Parent Element(s) Queries Experimentation. *"Retrieve Parent Element when the selection is on Child Element."*

The query for this problem can be categorized into nine, based on different combinations of projections and selections in the hierarchy [Pardede, 2008b]. For our case study, Figure 6 (denoted with D) represents *the selection predicate at attribute associated with projected element and the projection predicate presents at the bottom-most level of the hierarchy*. In this case, we "*retrieve authors element when its AuthorPosition attribute is located at the lowest level of the tree hierarchy*".

The following queries represent the possible structures to perform the operation. Queries 12 and 15 are written using XPath extract Query. Query 13 is written using XMLTABLE XQuery. Query 14 is written using XMLQuery XQuery.

```
Query 12:
SELECT extract(s.OBJECT_VALUE, '//authors')"AUTHORS"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,
      '//authors[(author/@AuthorPosition="89" and
      author/@AuthorPosition="67") or
      author/@AuthorPosition="100"]')= 1;

Query 13:
SELECT xtab.COLUMN_VALUE "AUTHORS"
FROM SIGMOD, XMLTABLE
    ('for $a in //authors
       where ($a/author/@AuthorPosition="89"
       and $a/author/@AuthorPosition="67")
       or $a/author/@AuthorPosition="100"
       return $a'
       PASSING OBJECT_VALUE)xtab;

Query 14:
SELECT XMLQuery
    ('for $a in //authors
       return $a'
       PASSING OBJECT_VALUE
       RETURNING CONTENT)"AUTHORS"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,'//
       authors[(author/@AuthorPosition="89" and
       author/@AuthorPosition="67") or
       author/@AuthorPosition="100"]')= 1;
Query 15:
SELECT extract(s.OBJECT_VALUE, '/SigmodRecord/*//authors')
       "AUTHORS"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,'/SigmodRecord/*//authors
       [(author/@AuthorPosition="89" and
       author/@AuthorPosition="67") or
       author/@AuthorPosition="100"]')= 1;
```

Parent Attribute Queries Experimentation. *"Retrieve Attribute associated with Parent Element when the selection is on Child Element."*

The query for this problem can be categorized into four, based on different combinations of projections and selections in the hierarchy [Pardede, 2008b]. For our case study, figure 6 (denoted with E) represents *the selection predicate in the attribute associated with child element and the projection predicate present at the middle level of the hierarchy category*. In this case, we "*retrieve textValue1 attribute while the selection predicate is textValue2 attribute*".

The following queries represent the possible structures to perform the operation. The queries include those written using XPath attribute query (Queries 16 and 19), using XMLTABLE (Query 17) and using XMLQuery (Query 18).

```
  Query 16:
  SELECT extract(s.OBJECT_VALUE,
        '//articles/@textValue1')"TEXTVALUE1"
  FROM SIGMOD s
  WHERE existsNode(s.OBJECT_VALUE,'//articles
        [articlesTuple/@textValue2="51"and
        articlesTuple/@textValue2="52"]')= 1;

  Query 17:
  SELECT xtab.COLUMN_VALUE "TEXTVALUE1"
  FROM SIGMOD, XMLTABLE
      ('for $a in //articles
        where $a/articlesTuple/@textValue2="51"
        and $a/articlesTuple/@textValue2="52"
        return $a/@textValue1'
        PASSING OBJECT_VALUE)xtab;

  Query 18:
  SELECT XMLQuery
      ('for $a in //articles
        return $a/@textValue1'
        PASSING OBJECT_VALUE
        RETURNING CONTENT)"TEXTVALUE1"
```

```
FROM SIGMOD
WHERE existsNode(s.OBJECT_VALUE,'//
      articles[articlesTuple/@textValue2="51"and
      articlesTuple/@textValue2="52"]')= 1;

Query 19:
SELECT extract(s.OBJECT_VALUE,
      '/SigmodRecord/*//articles/@textValue1') "TEXTVALUE1"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,'/SigmodRecord/*//
      articles[articlesTuple/@textValue2="51" and
      articlesTuple/@textValue2="52"]')= 1;
```

Ancestor Element(s) Queries Experimentation. *"Retrieve Ancestor Element when the selection is on Descendant Element."*

The query for this problem can be categorized into nine, based on different combinations of projections and selections in the hierarchy [Pardede, 2008b]. For our case study, figure 6 (denoted with F) represents *the selection predicate at the bottom-most descendant element and the projection predicate presents in one level down to the middle level of hierarchy category.* In this case, we " *retrieve textValue2 attribute while the selection predicate is AuthorPosition attribute"*.

The following list shows possible query structures for this category. Like in Parent Element(s) Query, there are four ways to writes this query, using XPath attribute Query, XMLTABLE attribute XQuery and XMLQuery attribute XQuery.

```
Query 20:
SELECT extract(s.OBJECT_VALUE, '//articlesTuple')
      "ARTICLESTUPLE"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,'//articlesTuple
      [(.//@AuthorPosition="67" and
      //@AuthorPosition="89") or
      .//@AuthorPosition="100"]')= 1;

Query 21:
SELECT xtab.COLUMN_VALUE "ARTICLESTUPLE"
FROM SIGMOD, XMLTABLE
    ('for $a in //articlesTuple
      where (($a/authors/author/@AuthorPosition = "67"
      and $a/authors/author/@AuthorPosition = "89")
      or $a/authors/author/@AuthorPosition = "100" )
      return $a'PASSING OBJECT_VALUE) xtab;

Query 22:
SELECT XMLQuery
    ('for $a in //articlesTuple
      return $a'
      PASSING OBJECT_VALUE
      RETURNING CONTENT) "ARTICLESTUPLE"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,'//
      articlesTuple[(.//@AuthorPosition="67" and
      .//@AuthorPosition="89") or
      .//@AuthorPosition="100"]')= 1;

Query 23:
SELECT extract(s.OBJECT_VALUE,
      '/SigmodRecord/*//articlesTuple') "ARTICLESTUPLE"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,'/SigmodRecord/*//
      articlesTuple[(.//@AuthorPosition="67" and
      .//@AuthorPosition="89") or
      .//@AuthorPosition="100"]')= 1;
```

Ancestor Attribute Queries Experimentation. *"Retrieve Attribute associated with Ancestor Element when the selection is on Descendant Element."*

The query for this problem can be categorized into nine, based on different combinations of projections and selections in the hierarchy [Pardede, 2008b]. For our case study, figure 6 (denoted with G) represents *the selection predicate at the bottom-most descendant attribute and the projection predicate presents at top level of the hierarchy category.* In this case, we *"retrieve issuesTuple attribute while the selection predicate is title element"*. We show the possible queries on the list below.

```
Query 24:
SELECT extract(s.OBJECT_VALUE, '//issuesTuple/@textValue')
      "TEXTVALUE"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,'//issuesTuple
      [.//title="XML QUERIES" and
      .//title="Interim Report: ANSI/X3/SPARC Study Group on
            Data Base Management Systems 75-02-08."]')= 1;

Query 25:
SELECT xtab.COLUMN_VALUE "TEXTVALUE"
FROM SIGMOD, XMLTABLE
     ('for $a in //issuesTuple
      where $a//articlesTuple/title = "XML QUERIES"
      and $a//articlesTuple/title = "Interim Report:
          ANSI/X3/SPARC Study Group on Data Base Management
          Systems 75-02-08."
      return $a/@textValue'PASSING OBJECT_VALUE) xtab;

Query 26:
SELECT XMLQuery
     ('for $a in //issuesTuple
      return $a/@textValue'
      PASSING OBJECT_VALUE
      RETURNING CONTENT) "TEXTVALUE"
FROM SIGMOD s
WHERE existsNode(s.OBJECT_VALUE,'//
    issuesTuple[.//title="XML QUERIES" and
    .//title="Interim Report: ANSI/X3/SPARC Study Group on
          Data Base Management Systems 75-02-08."]')= 1;
```

```
Query 27:
SELECT extract(s.OBJECT_VALUE,
      '/SigmodRecord/*//issuesTuple/@textValue') "TEXTVALUE"
FROM SIGMOD s
WHERE
      existsNode(s.OBJECT_VALUE,'/SigmodRecord/*//issuesTuple
      [.//title= "XML QUERIES" and
      .//title="Interim Report: ANSI/X3/SPARC Study Group on
            Data Base Management Systems 75-02-08."]')= 1;
```

5.3 Experimental Results

While in out experiment, we run all queries in possible categories for each query classification, in this section we show the results in seven categories.

Each of the categories can be written in different ways shown in the previous section. The analysis of the Execution Plan of C/D Queries is explained below:

- **Type of operation.** All the execution plans consist of full table access. Execution plan for queries 4, 7 and 11 (in C/D queries) and queries 15, 19, 23 and 27 (in Parent and Ancestor queries) are a subset of all other queries
- **Number of Steps.** The execution plan of queries 4, 7 and 11 (in C/D queries)and queries 15, 19, 23 and 27 (in Parent and Ancestor queries) involve only one step, fewer than for any other execution plan
- **Number of rows.** In queries 4, 7 and 11 (in C/D queries) and queries 15, 19, 23 and 27 (in Parent and Ancestor queries), only one row is being accessed and retrieved in the execution plan. It is less when compared with execution plans of other queries. The execution plan of these three queries involves full table access, whereas in others, several steps had to be performed in order to obtain the required output.

From the analysis of the execution plan, it can be concluded that the execution plan for queries 4, 7 and 11 (in C/D queries) and queries 15, 19, 23 and 27 (in Parent and Ancestor queries) are the most cost effective for its respective query type and category. A comparative analysis of total elapsed time using TKPROF, Total CPU time and total elapsed time using DBMS timing property is graphically represented in figures 7 and 8. We can see how these queries incur the smallest cost for each respected category in each comparative term.

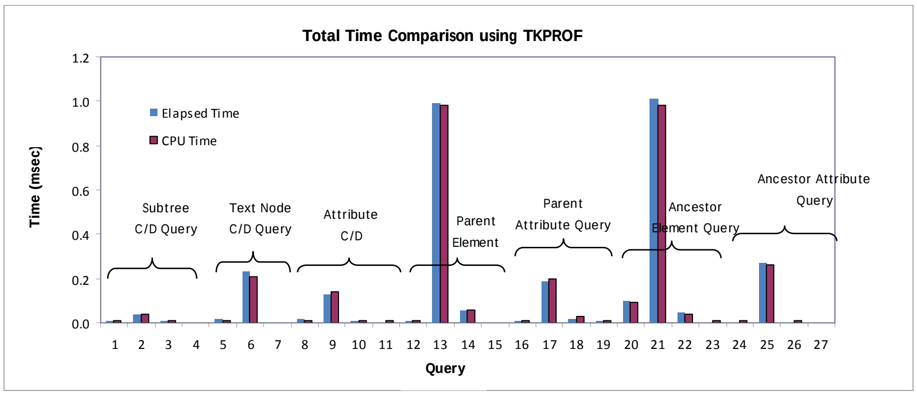

Figure 7: TKPROF Total Time Comparison

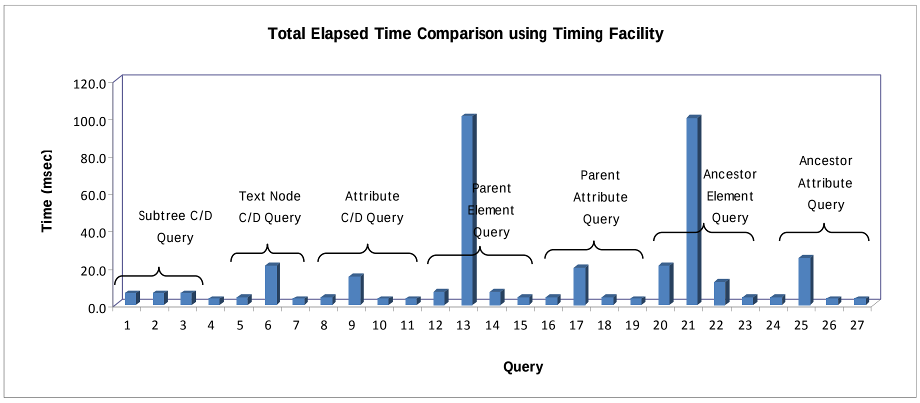

Figure 8: Timing Facility Total Elapsed Time Comparison

Based on the performance experimentations, we can derive some analysis for writing SQL/XML Query in XED.

- From the graphs, we can see XQuery written in terms of XPath Expressions using *extract* SQL function is more cost effective than XQuery written in terms of FLOWR Expressions using *XMLTABLE* and *XMLQuery* SQL Functions. XPath extract query yields good performance even when the selection is at bottom of hierarchy
- In various query types, having the selection predicate at the top level of the hierarchy performs better than having the predicate at the bottom. The performance for the queries having selection at top or bottom can be performed by writing the path //element as */root_node_element/*//element* in both selection and projection
- For all of the queries and different categories, XPath extract query performed better when path in the selection and projection predicates are written as

/root_node_element///projected_or_selected_element*. The performance of the XMLTABLE XQuery and XMLQuery XQuery can also be improved by writing the path as the same way.

6 Conclusion and Future Work

While many popular Relational Database Systems have enabled the storage of XML documents and its management using SQL/XML queries, very few database users aware that difference query structure performs significantly different. SQL/XML supports various structures to perform similar operations. The main structures are queries using (i) XPath extract query, (ii) XMLTABLE XQuery, (iii) XMLQuery XQuery and (iv) XPath XMLSequence query.

In this paper we experiments 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.

We analysed 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.

For the future work, a benchmark study of different XED can also be performed to help database users determine the storage option for their XML data. In addition, the same classification can be used to analyse the most optimised query structure for Native XML Database using XQuery or other proprietary query languages.

Another area that we can embark is the development of query rewriting tool, which performs automatic rewriting of a query structure into the most optimum structures possible.

References

[Bruno, 2002] Bruno, N., Koudas, N., Srivastava, D., Holistic twig joins: optimal XML pattern matching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'02).* ACM Press. 263-274. (2002).

[De Meo, 2004] De Meo, P., Terracina, G., Ursino, D., X-Global: a System for the "Almost Automatic" and Semantic Integration of XML Sources at Various Flexibility Levels. In *Journal of Universal Computer Science 10(9).* 1065-1109. (2004).

[Florescu, 1999] Florescu, D., Kossmann, D., Storing and Querying XML Data using an RDMBS. In *IEEE Data Engineering Bulletin* 22(3). IEEE-CS. 27-34. (1999).

[Grinev, 2005] Grinev, M., Pleshachkov, P., Rewriting-based Optimization for XQuery Transformational Queries. In *Proceedings of The International Database Engineering & Applications Symposium (IDEAS'05).* IEEE-CS. 163-174. (2005).

[ISO/IEC, 2003] Information Technology – Database Languages – SQL – Part 14: XML-Related Specifications (SQL/XML). ISO/IEC 9075-14 (2003).

[Jagadish, 2004] Jagadish, H.V., Lakshmanan, L.V.S., Scannapieco, M., Srivastava, D., Wiwatwattana, N.,. Colorful XML: One Hierarchy Isn't Enough. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'04).* ACM Press. 251-262. (2004).

[Lapis, 05] Lapis, G., *XML and Relational Storage-Are they mutually exclusive*. IBM Corporation. (2005).

[Le, 2007] Le, D.X.T., Bressan, S., Taniar, D., Rahayu, J.W., Semantic XPath Query Transformation: Opportunities and Performance. In *Proceedings of The International Conference on Database Systems for Advanced Applications (DASFAA'07).* Springer. 994-1000. (2007).

[Le, 2009] Le, D.X.T, Pardede, E., On Using Semantic Transformation Algorithms for XML Safe-Update. In *Proceedings of the 8th International Conference on Information Systems Technology and its Applications (ISTA'09),* Springer, 367-378. (2009).

[Oracle, 2007] Oracle, Introduction to Oracle XML DB. Oracle XML DB Developer's Guide, 10g Release 2. Available at http://download-west.oracle.com/docs/cd/B19306_01/appdev.102 /b14259/xdb01int.htm#i1047170. (2007).

[Pardede, 2004] Pardede, P, Rahayu, J.W., Taniar, D., On using collection for aggregation and association relationships in XML object-relational storage. In *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC)*, ACM Press, 703-710. (2004).

[Pardede, 2007] Pardede, E., Rahayu, J.W., Taniar, D., and Aujla, R.K., "Performance Analysis of Child/Descendant Queries in an XML-Enabled Database". In *Proceedings of the International Conference on Computational Science and its Applications (ICCSA'07),* Springer, 749-762. (2007)

[Pardede, 2008a] Pardede, E., Rahayu, J.W., and Taniar, D., "XML Update Management in XML-Enabled Relational Database". In *Journal of Computer and System Sciences,* 74(2), Elsevier Science, 170-195.(2008).

[Pardede, 2008b] Pardede, E., Rahayu, J.W., Taniar, D., and Aujla, R.K., "SQL/XML Performance Analysis of Parent/Ancestor Queries". In *Proceedings of the International Conference on Computational Science and its Applications (ICCSA'08),* Springer, 1258-1273. (2008)

[Shanmugasundaram, 1999] Shanmugasundaram, J., Tufte, K., Zhang, C. He, G., DeWitt, D.J., Naughton, J.F., Relational Databases for Querying XML Documents: Limitations and Opportunities. In Proceedings of International Conference on Very Large Databases (VLDB), Morgan-Kauffman. 302-314 (1999)

[Srivastavas, 2002] Srivastava, P.G., *Performance Evaluation Tools on Oracle,* Department of Computer Science and Computer Engineering, La Trobe University. (2002).

[Sun, 2006] Sun, W., Liu, D., Using Ontologies for Semantic Query Optimization of XML Databases. In *Proceedings of the First International Workshop on Knowledge Discovery from XML Documents (KDXD'06)*, Springer, 64-73. (2006).

[W3C, 2007] World Wide Web Consortium, *XQuery 1.0: An XML Query Language*, W3C Recommendation 23 January 2007, available from:
http://www.w3.org/TR/xquery/.

[Wang, 2003] Wang, G., Liu, M., Yu, J.X., Sun, B., Yu, G., Lv, J., Lu, H., Effective schema-based XML query optimization techniques. In *Proceedings of The International Database Engineering & Applications Symposium (IDEAS'03)*. IEEE-CS. 230-235. (2003).

[Wang, 2005] Wang, L., Wang, S. Murphy, B., Rundensteiner, E., Order-sensitive XML Query Processing over Relational Sources: An Algebraic Approach. In *Proceedings of The International Database Engineering & Applications Symposium (IDEAS'05)*. IEEE-CS. 175-184. (2005).

[Zhang, 2001] Zhang, C., Naughton, J., Dewitt, D., Luo., Q., Lohman, G., OHMAN. On Supporting Containment Queries in Relational Database Management Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'01)*. ACM Press. 425-436. (2001).

[Zhang, 2002] Zhang, X., Pielech, B., Rundesnteiner, E., Honey, I Shrunk the XQuery! — An XML Algebra Optimization Approach. In *ACM International Workshop on Web Information and Data Management (WIDM'02)*. ACM Press. 15-22. (2002).