THE IMPLEMENTATION OF AN XPATH QUERY PROCESSOR FOR XREL AND EDGE METHODS

by

Mine MERCAN

February 2004

THE IMPLEMENTATION OF AN XPATH QUERY PROCESSOR FOR XREL AND EDGE METHODS

by

Mine MERCAN

A thesis submitted to

the Graduate Institute of Science and Engineering

of

Fatih University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

February 2004
Istanbul, Turkey

APPROVAL

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Assist Prof. Dr. Atakan KURT
Head of Department

This is to certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assist Prof. Dr. Atakan KURT
Supervisor

Examining Committee Members

Assist Prof. Dr. Atakan KURT

Assist Prof. Dr. Veli HAKKOYMAZ

Assoc. Prof. Dr. Erkan İMAL

It is approved that this thesis has been written in compliance with the formatting rules laid down by the Graduate Institute of Sciences and Engineering.

Assoc. Prof. Dr. Nurullah ARSLAN
Director

20 / 2 / 2004

THE IMPLEMENTATION OF AN XPATH QUERY PROCESSOR FOR EDGE AND XREL METHODS

Mine MERCAN

M.S. Thesis – Computer Engineering
February 2004

Supervisor: Assist. Prof. Atakan Kurt

ABSTRACT

In this thesis, we implemented an XPath query processor for XML documents stored in an RDBMS using two selected methods (XRel and Edge). This study is the complement of the previous work 'An Implementation of Storage and Retrieval Methods for XML Documents in RDBMSs' by Zulal Su. In that implementation MySQL was used as the database management system and PhpMyAdmin 2.4.0 as a web based database administration tool. PhpMyAdmin was extended with *XML collections*. A collection is a set of similar XML documents. In this study we added an XPath query processor to this implementation. A comparative experimental study of a set of XPath queries of these methods for text and data centric documents is also presented.

Keywords: XPath, DBMS, XML Storage Methods, Query Processor

EDGE VE XREL METODLARI İÇİN XPATH SORGU İŞLEMCİSİ UYGULAMASI

Mine MERCAN

Yüksek Lisans Tezi – Bilgisayar Mühendisliği
Şubat 2004

Tez Yöneticisi: Yrd.Doç.Dr. Atakan Kurt

ÖZ

Bu tezde seçtiğimiz iki yöntemi (XRel ve Edge) kullanarak ilişkisel veritabanlarında saklanmış XML dökümanları için sorgulama işlemcisi uygulaması geliştirildi. Bu çalışma Zulal Su tarafından yapılan ' XML dökümanlarını ilişkisel veritabanı yönetim sistemlerinde saklama, veri geri elde etme metodlarının bir uygulaması', çalışmasının devamı niteliğindedir. Uygulamamızda ilişkisel veritabanı yöneticisi olarak MySQL ve web tabanlı veritabanı yönetici programı olarak da PhpMyAdmin kullanıldı. PhpMyAdmin programı 'collection'lar eklenerek geliştirilmiştir. Collection'lar benzer XML dökümanlarının saklandığı depolardır. Bu tezde önceki uygulamaya bir sorgulama işlemcisi eklenmiştir. Aynı zamanda, karşılaştırmalı olarak, bu metodlarda bir grup XPath sorgusu, metin ve veri merkezli dökümanlar için çalıştırılıp, sonuçları sunulmuştur.

Anahtar Kelimeler: XPath, DBMS, XML Dökümanlarını saklama metodları, Sorgulama işlemcisi

To my parents

ACKNOWLEDGEMENT

I express sincere appreciation to Assist. Prof. Dr. Atakan KURT for his guidance and insight throughout the research.

Thanks go to the other faculty members, Assist. Prof. Dr. Veli Hakkoymaz and Assoc. Dr. Erkan IMAL, for their valuable suggestions and comments. The technical assistance of Kerziban MUMCU, Zulal SU, Sevil AKTAN and Tara SALIH are gratefully acknowledged.

I express my thanks and appreciation to my family for their understanding, motivation and patience. Lastly, but in no sense the least, I am thankful to all colleagues and friends who made my stay at the university a memorable and valuable experience.

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURE

CHAPTER 1

INTRODUCTION

1.1 PROBLEM DEFINITION

Extensible Markup Language (XML) has become a standard format for information representation and exchange on the Internet. The advantage over other semi-structured languages is that XML is self-describing. The user can make up new tags for descriptive markup for their own applications. These user defined tags can identify the semantics of data. The relationships between elements can be shown with nested structures. There are lots of applications developed which uses XML format. Therefore many kinds of data will be exchanged in the form of XML documents.

An XML document should be written according to some rules. This XML document is called *well-formed XML* documents.

XML document to be *well formed* (Fan and Libkin, 2001):

- The well-formed XML documents begin with an XML declaration and have only one root element.
- Elements should have 'opening tag' at the begining and 'closing tag' at the end.
- All no empty elements must have an opening and closing tag.
- And empty element is terminated with a forward slash preceding the "less-than" bracket of the tag.
- Nested elements are possible, but tags must not overlap.
- All attributes values must be quoted.
- XML is a case-sensitive language.

The set of rules that define the XML document structure is called Document Type Definition (DTD). If the XML documents DTD is given and the XML document follows that DTD, it is called *valid XML* document.

Since XML became popular, there is a lot of interest for storing XML documents and retrieving information from them. This is one of the problems in the reseach area of XML. There are several approachs to storing XML data. One of the possible approaches of storing semi-structured data is to store them in a relational database system. Relational database systems are mature so that they have the additional advantage. Therefore lots of researchs are made in this area. We can classify these researchs into two categories. One is *integral storage*, the other is *dispersed storage*.

In integral storage methods, original document is kept unparsed and stored in a relational database as a large object (BLOB). Since in this approach parsing is made when accessing to individual elements, it is inefficient. It is only needed for some special cases.

In dispersed storage approach, XML documents are parsed and their components stored in tables. But this time the question is 'How should be designed to get maximum performance'. Speed is a problem when retrieving data from massive XML documents. In general there are two types of solution for this problem. *Dynamic schema mapping approach* and *fixed schema mapping approach* (Yoshikawa and Amagasa, 2001). Dynamic schema mapping is dependent to the structure of XML document which called DTD (Document type descriptor). But fixed schema mapping is independent from the structure of the document.

The advantages of the fixed schema mapping approach are in the following (Yoshikawa and Amagasa, 2001).

- Massive XML documents can be stored without extending database models.
- It supports XML documents that have dynamic structure or static structure and also Well-formed and non-DTD XML documents can be stored.

Su made an implementation to get benefit of fixed schema mapping approach. In her study users can store XML documents to MySQL database by two fixed schema

methods, XRel and Edge. In her method, XML data is mapped into tables. These tables are in XML collections. A collection keeps a set of XML documents which is stored in fixed schema of tables. In our study we give an algorithm to query these XML documents.

There are also several query languages to retrieve data from XML documents. The common feature of these query languages (examples are Lorel, XML-GL, XML-QL, and XQL) is that all of them are based on regular path expressions (Florescu and Kossmann, 1999). XPath is one of the XML query languages.

XPath has been proposed by W3C organization as a node-selecting language from XML documents. The syntax of XPath is seen in Figure 1.2

```
<issue>
   <editor>
      <first>Michael</first>
      <family>Franklin</family>
   </editor>
   <articles>
      <article category="research surveys">
         <title>Comparative Analysis of Six XML Schema Languages</title>
         <authors>
            <author>
               <first>Dongwon</first>
               <family>Lee</family>
            </author>
            <author>
               <first>Wesley</first>
               <middle>W.</middle>
               <family>Chu</family>
            </author>
         </authors>
         <summary>As <keyword>XML</keyword> is emerging ... </summary>
      </article>
   </articles>
</issue>
```

Figure 1.1 An example of an XML Document (Yoshikawa and Amagasa, 2001)

An instance of XPath Query for the document in Figure1.1 is **/issue/articles//author**

This query retrieves all *authors* inside the *articles* which is subelement of the *issue*.

```
Query       ::= PathExpr

PathExpr    ::= RegularExpr
              | '/' RegularExpr
              | '//' RegularExpr
              | BasicExpr Predicate* '/' RegularExpr
              | BasicExpr Predicate* '//' RegularExpr

RegularExpr ::= Step Predicate*
              | RegularExpr '/' Step Predicate*
              | RegularExpr '//' Step Predicate*

Step        ::= NameTest
              | '@' NameTest

Predicate   ::= '[' Comparison ']'

BasicExpr   ::= '(' Comparison ')'
              | Literal
              | Number

Comparison  ::= ArithExpr
              | ArithExpr CompareOp ArithExpr

CompareOp   ::= '='
              | '!='

ArithExpr   ::= BasicExpr Predicate*
              | PathExpr

NameTest    ::= QName
```

Figure 1.2 The syntax of XPath (Yoshikawa and Amagasa, 2001)

1.2 RELATED WORK

Current database systems such as Oracle, Microsoft, IBM have tools to manipulate XML documents. In the following, the storing and retrieving methods that are developed by database vendors, Microsoft, IBM and Oracle, will be explained.

1.2.1 Oracle 9i XDB

Oracle 9i XDB supports various methods to store XML documents. A number of functions added to SQL to handle XML operations.

XMLType is a built in object type that can store an XML document and provides lots of features for managing XML storage. Like any object type, XMLType

can be used as the data type of a column in a table. XMLType data can be stored in two ways; using object relational storage or as a CLOB.

In the following example (Figure 3.1), XML document stored with using XMLType data type.

```
<!ELEMENT FXTRADE (CURRENCY1, CURRENCY2, AMOUNT, SETTLEMENT,
ACCOUNT)>
<!ELEMENT ACCOUNT (BANKCODE,BANKACCT)>
<!ELEMENT BANKCODE  (#PCDATA)>
<!ELEMENT BANKACCT  (#PCDATA)>
<!ELEMENT CURRENCY1  (#PCDATA)>
<!ELEMENT CURRENCY2  (#PCDATA)>
<!ELEMENT AMOUNT    (#PCDATA)>
<!ELEMENT SETTLEMENT (#PCDATA)>
```

(a) DTD of Bank.xml document

```
CREATE TABLE FXTRADE
{

CURRENCY1  CHAR(3),
CURRENCY2  CHAR(3),
AMOUNT     NUMERIC (18,2)
SETTLEMENT DATE,
ACCOUNT  sys.XMLType

}
```

(b) The FXTRADE Table

Figure 3.1 An example of storing XML documents by using XMLType data type in Oracle 9i (Dayen, 2001)

When using object-relational storage, users define the mapping with an XML Schema. Users can create their own mappings or use a default mapping generated by XDB (Bourret, 2003). Further information is at (Oracle XML DB, 2002).

1.2.2 DB2 XML Extender

The XML Extender provides two access and storage methods for using DB2 as an XML repository (Dayen, 2001); *XML column*, stores and retrieves entire XML document as DB2 column data, and *XML Collection*, separates XML documents into

simpler compounds and stores into a collection of relational tables or composes XML documents from a collection of relational tables.

The mapping between the XML Document structure and the database tables is defined by means of a Data Access Definition (DAD) file. DAD refers to a processed document DTD. Further information about XML support in DB2 is at (DB2 XML Extender, 2002).

1.2.3 Access 2002

Access 2002 uses table-based mapping when transferring data from XML documents. Individual data values must be in child elements and table/column names must match element names. Access can create an XML schema document, describing exported data. Further information can be found in (Frank, 2001).

1.2.4 SQL Server 2000

Microsoft SQL Server 2000 extends SQL language for storing and retrieving XML documents. SELECT clause was extended with the FORXML construct to retrieve data in XML format. In order to store XML data a row set function OPENXML is added. Before an XML document stored in tables, "field-element" association should be defined using XPath notation.

A comparison of XML Products is given in Table 1.1 (Dayen, 2001).

Table 1.1 XML Product Comparisons

Vendor	Mapping rules	Single table / Multiple tables	Means of transformation	Symmetrical extraction / storing
Oracle	Implicitly; by constructing object-relational data model	Multiple	Designated Java classes	Symmetrical, if XML document and object-relational model match
IBM	Data Access Definition file	Multiple	Designated stored procedures	Symmetrical
Microsoft	SQL extension; row set function	Multiple for extraction; Single for storing	By using SQL construct FOR XML and row set OPENXML	Asymmetrical

In this thesis, we are going to implement algorithms for translating XPath queries into SQL queries for two approaches to storing XML data in relational database. The first one is Path based storing method (XREL), the other is the Edge approach which stores all attributes in a single table.

1.3 OUTLINE OF THESIS

This thesis is organized as follows: Chapter 2 is an overview of the methods for storing XML documents, more information about XRel and Edge methods is given. Chapter 3 discusses query processing and the implementation of the algorithms, Chapter 4 presents the experimental results. Conclusions are given in Chapter 5.

CHAPTER 2

METHODS FOR STORING XML DOCUMENTS IN RDBMS

In this chapter alternative mapping methods are described that is used to store XML documents in a relational database system. First the data model for XML is considered.

2.1 XML DATA MODEL

The eXtensible Markup Language (XML) has been created by the World Wide Web Consortium (W3C). The goal of XML is to enable the delivery of self-describing data structures of arbitrary depth and complexity to applications that require such structures. (Florescu and Kossmann, 1999) . XML is syntax for semi-structured data. Semi-structured data means that the data keeps its value and its meaning (names and types.) together.

The need for semi-structured data has arisen of the web independently. Varius aspects of managing semi-structured data have been extensively studied in the last years (Florescu and Kossmann, 1999).

Simply an XML document structure is a set of elements. Elements begin with a start-tag and ends with an end-tag. Each element has a name and type. They may include some attributes as well. Elements can have other elements. We called subelements. Below is our instance XML document. We are going to explain our algorithms over this document.

Example: Let's assume that we want to export information of address book. One way of representing this as an XML structure is, each card has a unique identifier *no* attribute. Information about name, *title, address* and *contact* of that person are subelements.

```
<?xml version="1.0"?>

<address_book>
 <card no="1">
        <name>
                <surname>Brown</surname> <given>Paul</given>  <other>Micheal</other>
        </name>
        <title>Prof.Dr</title>
        <address>
                <street>20th Floor, 300 Lakeside</street>
                <city>OAKLAND</city><state>CA</state><zip>98520</zip>
        </address>
        <contact>
                <phone>510 628 3993</phone>
        </contact>
 </card>

 <card no="2">
        <name>
                <surname>Buneman</surname><given>Peter</given>
        </name>
        <title>Instructor</title>
        <address>
                <street>23th Floor, 456 Leiben</street>
                <city>New York</city><state>TX</state><zip>5188</zip>
        </address>
        <contact>
                <phone>4 852 96 0000</phone>
                <phone>4 852 96 0102</phone>
        </contact>
 </card>
</address_book>
```

Figure 2.1 An example of an XML Document, address book.

In the XPath data model, an XML document is modelled as a tree. That is an ordered and labeled directed tree. We took four types of nodes for the simplicity which are root, element, text and attribute but actually there are seven types of nodes. The root node is a virtual node pointing to the root element of the document. Elements in an XML document are represented as a node with the tag name of element. Element nodes can include subelements or text parts as its children. Every text node or an element can only have one parent. Text nodes are leaf nodes which have string values. Text nodes do not have any child nodes. An element node can have a set of attribute nodes or no attributes nodes at all. An attribute node has a name and a value. Attribute nodes do not have any child nodes too. By definition an element node can be a parent of an attribute node but attribute nodes are not the children of the element node. Figure 2.2 shows an XML Data Graph such that a left to right and depth first traversal describes the order of the XML content within our document in Figure 2.1

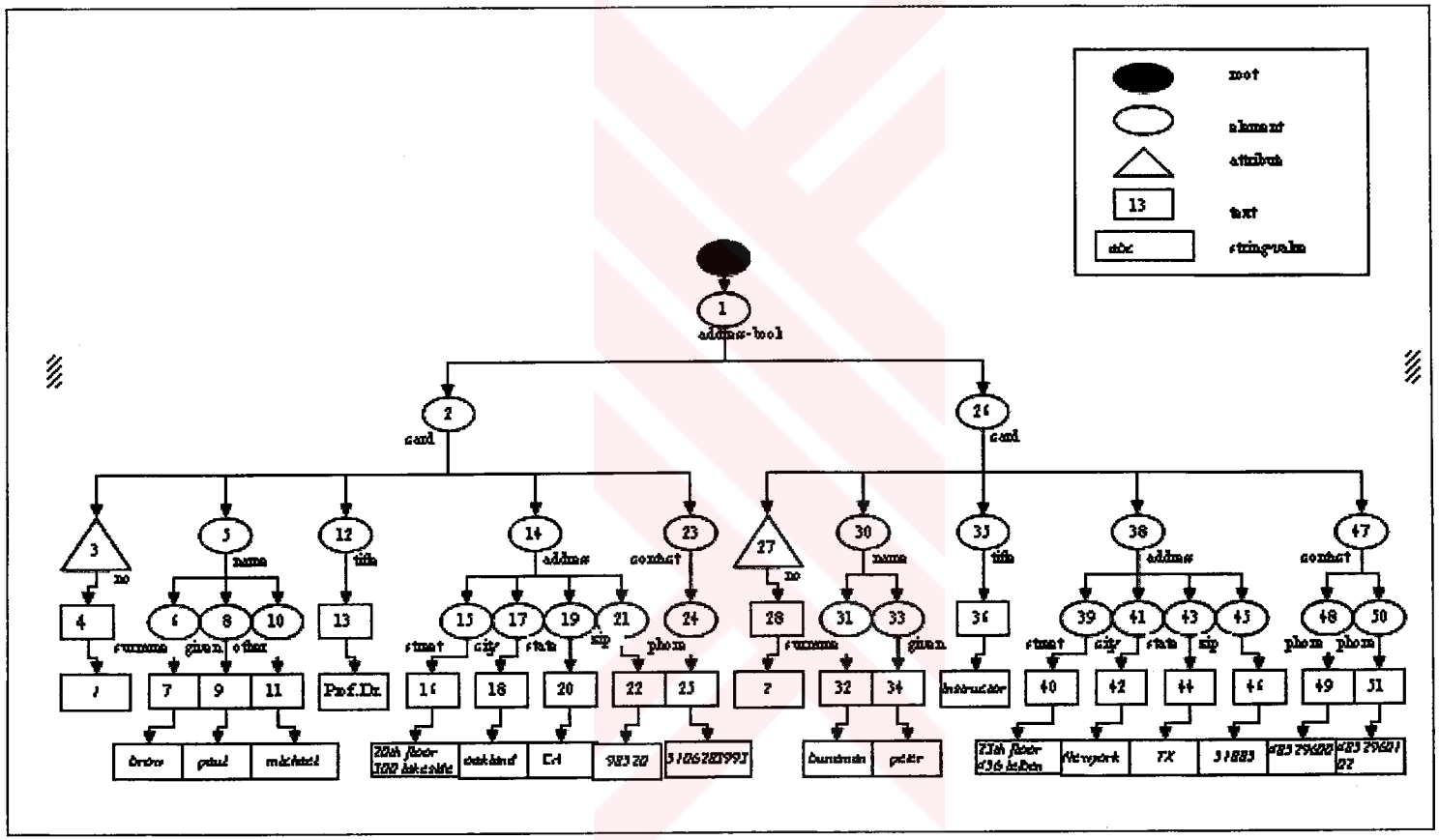

Figure 2.2 The data graph of XML document in Figure 2.1.

2.2 THE EXISTING APPROACHES OF MAPPING

A mapping method describes which tables to create in the database, which columns to build and which tables to store elements, attributes and text values. There are a lot of methods described by researchers but we are going to concentrate on selected three methods.

2.2.1 The Edge approach

This approach was proposed by Florescu, D. and Kossmann, D., (Florescu and Kossmann, 1999). In this method all edges of the graph is stored in a single table named Edge (Florescu and Kossmann, 1999). Edge (Source, Ordinal, Target, Name, Flag, Value). The Edge table for the data graph given in Figure 2.2 is in the Table 2.1.

Table 2.1 Edge table for the XML data graph given in Figure 2.2

Source	Ordinal	Target	Name	Flag	Value
0	1	1	address_book	ref	-
1	1	2	card	ref	-
2	1	3	@no	val	"1"
2	2	5	name	ref	-
5	1	6	surname	val	"Brown"
5	2	8	given	val	"Paul"
5	3	10	other	val	"Michael"
2	3	12	title	val	"Prof.Dr."
2	4	14	address	ref	-
14	1	15	street	val	"20th floor 300 Lakeside"
14	2	17	city	val	"oakland"
14	3	19	state	val	"CA"
14	4	21	zip	val	"98520"
2	5	23	contact	ref	-
23	1	24	phone	val	"510 628 39 93"
1	2	26	card	ref	-
26	1	27	@no	val	"2"
26	1	30	name	ref	-
30	1	31	surname	val	"Buneman"
30	2	33	given	val	"Peter"
26	2	35	title	val	"Instructor"
26	3	38	address	ref	-
38	1	39	street	val	"23th floor 456 Leiben"
38	2	41	city	val	"Newyork"
38	3	43	state	val	"TX"
38	4	45	zip	val	"51885"

26	4	47	contact	ref	-
47	1	49	phone	val	"485 29 600"
47	2	51	phone	val	"485 296 01 02"

In the Edge table, *target* is the node number of that node in the graph, *source* is the parent node of the node, *ordinal* is the order of that node among its siblings, *name* is the node name, *Flag* shows that the node has value or not, *Value* is the text value of the node. As seen in this approach parent-child edges are kept only.

2.2.2 Monet Method

Monet method was proposed by Schmidt, A. (Schmidt et al., 2000) .This approach is a variation of Edge method. In this method XML data is stored in multiple tables. For each unique path there is a table. In other words each table keeps the nodes which has same path. So the number of tables is different for each document. For instance in our data graph (Figure 2.2) we have 34 tables. This method is very efficient in single path queries. We can see the disadvantage of this approach in the complex queries with multiple paths. It needs to conduct joins. Because of this and other disadvantages mentioned in Su's work, the benefits of this approach are limited.

2.2.3 The XRel Approach

This approach was proposed by Yoshikawa, M. and Amagasa, T. (Yoshikawa and Amagasa, 2001). In this method the XML data stored into tables according to the node types and in a different table the unique paths are kept as they are shown in Figure2.3. There are four tables; element, attribute, text and path table. Element table has the document structure. Attribute table has attribute nodes. Text table has text data and path table has the unique paths.

Element (DocID, PathID, Start, End, Index)
Attribute (DocID, PathID, Start, End, Value)
Text (DocID, PathID, Start, End, Value)
Path (PathID, Pathexp)

Figure 2.3 Tables in XRel method.

So the number of tables is fixed. In XRel's element table, unlike Edge table, the nodes are stored with start and end positions that indicate the number of bytes counted from the begining of document. The end position shows us the end of descendents.

In the previous work XRel's tables have changed in two points. Instead of start and end positions, Su put nodeID and EndDesID that shows the last descendent nodeID. The other change is the parentID was added to the element table, attribute table and text table.

The changed tables are shown in Figure2.4.

Element (DocID, NodeID, EndDesID, PathID, ParentID, Ordinal, elname)
Attribute (DocID, NodeID, PathID, ParentID, Value, Name)
Text (DocID, NodeID, PathID, ParentID, Value)
Path (DocID, PathID, Pathexp)

Figure 2.4 The last form of the changed tables in XRel method.

Table 2.2 Tables of XRel

NodeId	EndDescID	PathID	ParentID	Ordinal
1	51	1	0	1
2	25	2	1	1
5	11	4	2	1
6	7	5	5	1
8	9	6	5	1
10	11	7	5	1
12	13	8	2	1
14	22	9	2	1
15	16	10	14	1
17	18	11	14	1
19	20	12	14	1
21	22	13	14	1
23	25	14	2	1
24	25	15	23	1
26	51	2	1	2
30	34	4	26	1
31	32	5	30	1
33	34	6	30	1
35	36	8	26	1
38	46	9	26	1
39	40	10	38	1
41	42	11	38	1
43	44	12	38	1
45	46	13	38	1
47	51	14	26	1
48	49	15	47	1
50	51	15	47	2

(a) Element Table

Table 2.2 Tables of XRel (continued)

PathID	PathExp
1	/address_book
2	/address_book/card
3	/address_book/card/@no
4	/address_book/card/name
5	/address_book/card/name/surname
6	/address_book/card/name/given
7	/address_book/card/name/other
8	/address_book/card/title
9	/address_book/card/address
10	/address_book/card/address/street
11	/address_book/card/address/city
12	/address_book/card/address/state
13	/address_book/card/address/zip
14	/address_book/card/contact
15	/address_book/card/contact/phone

(b) Path Table

NodeId	PathID	ParentID	Value
7	5	6	Brown
9	6	8	Paul
11	7	10	Michael
13	8	12	Prof.Dr.
16	9	15	20th floor 300 Lakeside
18	10	17	Oakland
20	11	19	CA
22	12	21	98520
25	15	24	5106283993
32	5	31	Buneman
34	6	33	Peter
36	8	35	Instrucotor
40	9	39	23th floor 456 Leiben
42	10	41	NewYork
44	11	43	TX
46	12	45	51885
49	15	48	4852960000
51	15	50	4852960102

(c) Text Table

NodeId	PathID	ParentID	Value
3	3	2	1
27	3	26	2

(d) Attribute Table

CHAPTER 3

IMPLEMENTATION

In this section, we will explain query processing algorithms when XML documents are stored into RDBMS according to fixed-schema mapping approaches. The overall approach is to translate an XML query into SQL. RDBMS executes the query and do some processing in order to get the right query result. The most important step is the translation of XPath expression into SQL.

In the previous work, Su implemented two methods to store XML documents to MySQL database which is a free and open source database system (MySQL AB, 2003). The data in a database is organized into tables. And each table is organized into rows and columns. MySQL users can manage data with *insert*, *delete*, *update* and *drop* operations. Su added some commands for XML documents as well. After our work users can query these XML documents. As a result, MySQL will become repository for both XML documents and relational data.

Su's implementation adds "collection" support to the MySQL database. "A collection is a set of XML documents stored in fixed schema of tables. In reality, MySQL database model did not change, but from the user point of view, inside a database there is not only tables but also collections. In addition, the users can not modify or access to the tables of collections directly. Users know the existing collection names and types only. And they can create, drop or browse collections. Also users can insert, browse or delete XML documents from collections." (Su -2003)

In the previous implementation Su embedded her classes to PhpMyAdmin program which is a web based interface between MySQL and users. Su implemented

Edge and XRel methods. She did not implement Monet model and we will not too because of its disadvantages mentioned in chapter 2.

Summarization of new model as follows,

- A database has two types of sub-structures, tables and collections.
- Relational data is stored in tables and XML documents are stored in collections.
- Collections are created using two approaches (XRel and Edge).
- Multiple XML document can be stored in a collection. (Su, 2003)

We applied our model to MySQL 4.1.1-alpha-nt database and we used PhpMyAdmin 2.4.0 program as web interface. In Su's program, user can select and browses inside a collection. When browsing a collection, besides each document there is an XPath button. After clicking this button the Xpath text field comes in a new page. When is submitted Xpath query, one of our main functions are called according the type of that collection. Main functions are "evaluate_XRel" and "evaluate_Edge".

In the following Figures there are three screen shots of extended PhpMyAdmin program. Inside a collection is shown in Figure 3.1, the XPath query screen in Figure 3.2, and the results of XPath query in Figure 3.3.

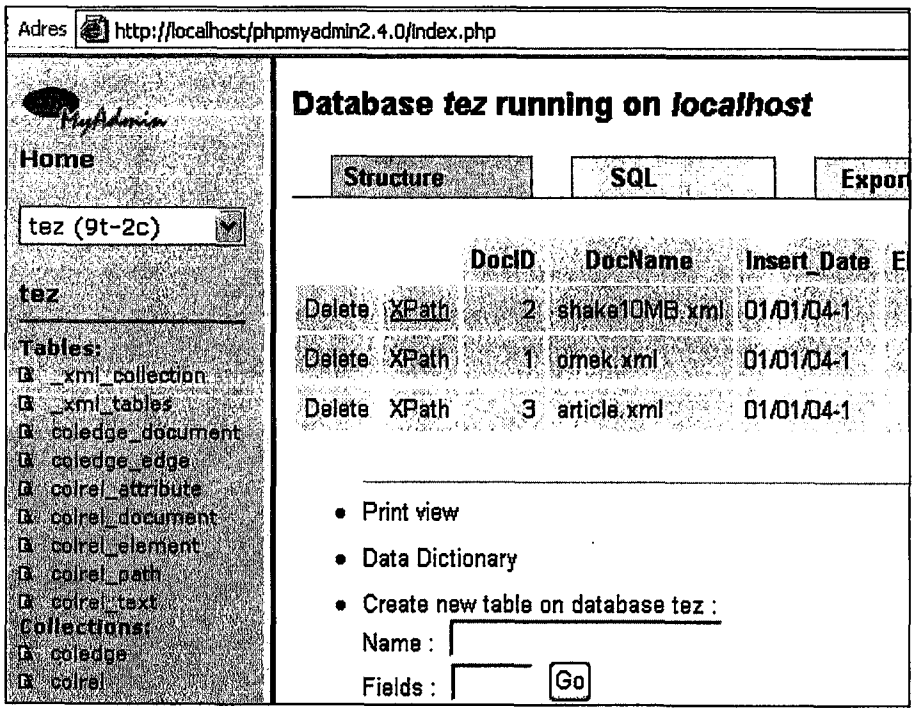

Figure 3.1 Inside a collection

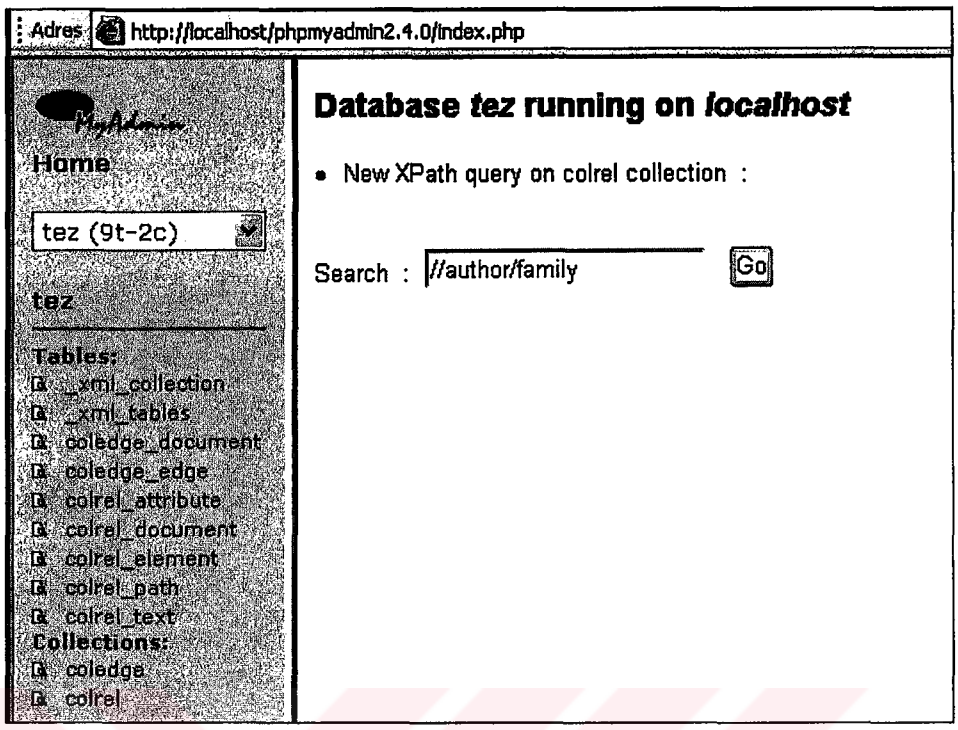

Figure 3.2 Writing XPath query

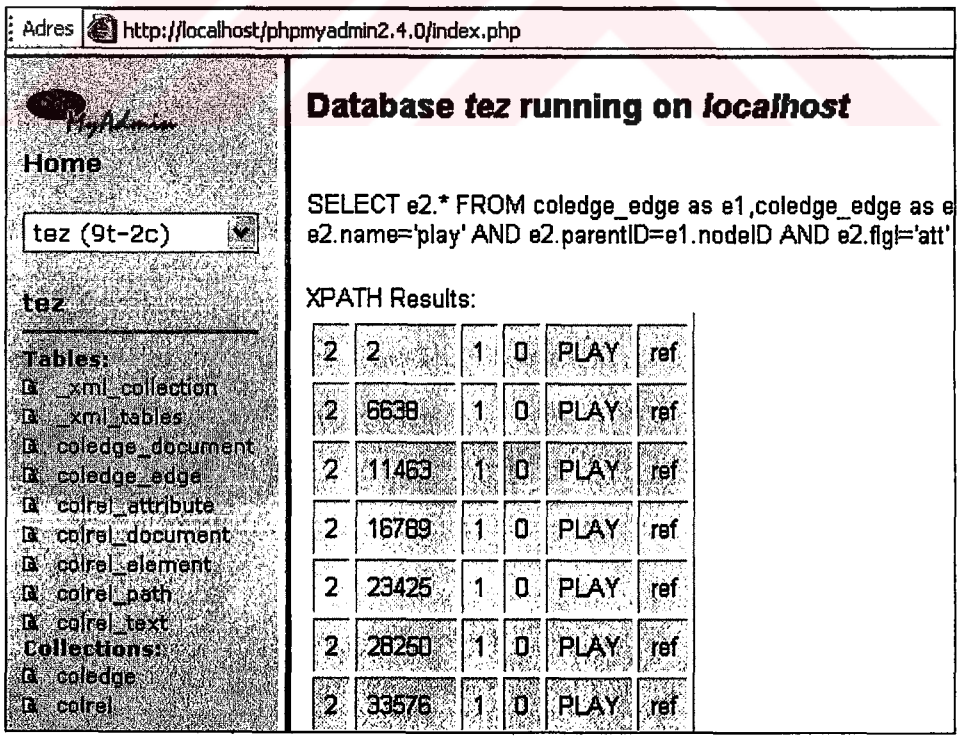

Figure 3.3 The results of an XPath query

3.1 THE QUERY CONVERSION FOR XREL METHOD

The translation from XPath expressions into SQL queries we used algorithm in XRel article (Yoshikawa and Amagasa, 2001). In this algorithm there are two steps.

(1) The XPath graph is created as an intermediate representation of XPath expressions. An XPath expression is divided into simple expressions. Nodes and edges represent path expressions and their relationship, respectively.
(2) SQL queries are generated from XPath graphs.

3.1.1 The translation from XPath expressions into XPath graphs

We take the XPath query. We split the query into fragment parts. In this split operation we consider double slashes and predicates. Firstly we cut the query after predicates and before double slashes. Secondly split predicates.

To explain the method Query 1 used as a sample query,

Query 1://article[summary/keyword = 'XML']//author /family(Yoshikawa and Amagasa, 2001)

This query retrieves all the **familys** inside the **authors** which is subelement of **articles** whose subelement **summary** includes 'XML' **keyword.**

Firstly this query is splitted into two and put in a graph as shown in Figure 3.4.

(1) //article[summary/keyword = 'XML']

(2) //author /family

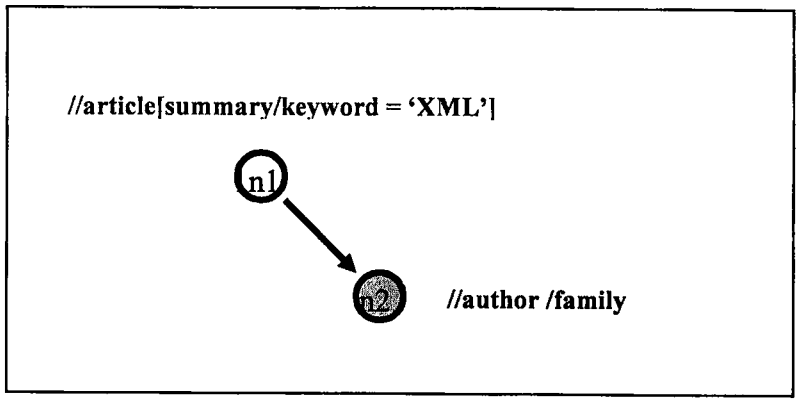

Figure 3.4 How the graph is look like at the beginning.

After the predicates are opened the graph takes the last form, as in Figure 3.5.

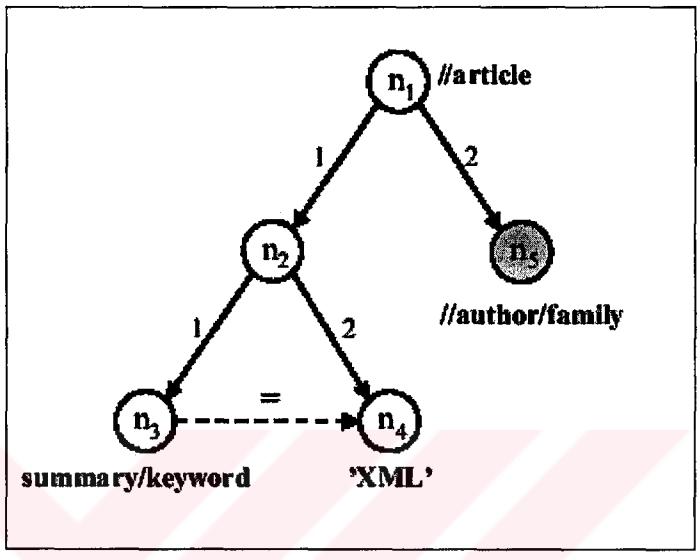

Figure 3.5 The XPath graph of the query (Yoshikawa and Amagasa, 2001)

There is an output node which is the last fragment part in the query. In this query this is n5.

3.1.2 The translation from XPath graph into SQL Query

After creating an XPath graph, we find *path concatenation* of expression nodes in the graph. The concatenated value is expresions along tree edges from the root to a node in an XPath graph to obtain a full path expression.

Example 1: The node **n3** has the concatenated value of **//article/summary/keyword**

In the SQL query the concatenated values are used string matching with the unique paths in the path table. For this match operation we have to make small changes in concatanted values and path table values. The function foo(x) does this change. Takes the concatenated value and returns a character string obtained by replacing every occurrence of '/' in x with '#/'; and every occurrence of '//' in x with '#%/'.

Example2: The concatenated value (Example 1) is translated into the following ,

#%/article#/summary#/keyword

Example 3 : The XPath query in the Query 1 is translated in this SQL query.

```
SELECT e2.* FROM colrel_Path p1, colrel_Element e1, colrel_Path p2,
colrel_Element e2, colrel_Path p4,colrel_Text t4
WHERE p1.pathexp LIKE '#%/article'
AND e1.DocID=3
AND e1.pathID=p1.pathID
AND p2.pathexp LIKE '#%/article#%/author#/family'
AND e2.DocID=3
AND e2.pathID=p2.pathID
AND e1.docID=e2.docID
AND e1.endDesId >=e2.endDesId
AND e1.nodeId <e2.nodeId
AND t4.value='XML'
AND p4.pathexp LIKE '#%/article#/summary#/keyword'
AND t4.pathID=p4.pathID AND e1.docID=t4.docID
AND e1.nodeId <t4.parentId
```

The algorithm is shown below generates SQL queries from XPath graphs. The algorithm in (Yoshikawa and Amagasa, 2001) is implemented.

Evaluate_XRel(Q) :

Input: An XPath query
Output: An SQL query
Algorithm:

(1) For each node *n* of type Expr whose concatenated-value is *p*, do the followings:
—Add "Path p_n" in FROM list; and
—Add "AND p_n.pathexp = *foo*(*p*)" (if *p* does not contain '//'); or
"AND p_n.pathexp LIKE *foo*(*p*)" (otherwise)
in WHERE clause.

(2) For each node *n* which is not connected to a comparison edge, do the followings:

—If the suffix of p, the concatenated-value of n, is an element name, then
—Add "Element e_n" in FROM list; and
—Add "AND e_n.pathID = p_n.pathID" in WHERE clause.
Otherwise (i.e. if the suffix of p is an attribute name),
—Add "Attribute a_n" in FROM list; and
—Add "AND a_n.pathID = p_n.pathID" in WHERE clause.

(3) For each node n which is connected to a comparison edge, do the followings:
—If the suffix of p, the concatenated-value of n, is an element name, then
—Add "Text t_n" in FROM list; and
—Add "AND t_n.pathID = p_n.pathID" in WHERE clause.
Otherwise (i.e. if the suffix of p is an attribute name),
—Add "Attribute a_n" in FROM list; and
—Add "AND a_n.pathID = p_n.pathID" in WHERE clause.

(4) For each index node m of type Number in *Graph*, do the followings:
—Add "AND e_n.index = k" in WHERE clause, where n is the parent element node of m, and k is the value
of m.

(5) For each pair of two nodes m and n in *Graph* such that i) both m and n are a node of type Expr; and ii) n is
the closest ancestor of m, do the followings:
—Add the followings in WHERE clause:
—"AND x_n.nodeID < y_m.nodeID"
—"AND x_n.endDesID >= y_m.endDesID"
—"AND x_n.docID = y_m.docID"

where

x = e (if the suffix of the value of n is an element name, and n is not connected to a comparison node)
t (if the suffix of the value of n is an element name, and n is connected to a comparison node)
a (otherwise, i.e. if the suffix of the value of n is an attribute name)

(similar for y)

(6) For each comparison edge (n, θ,m) in *Graph*,
—Add "AND X_n θ Y_m" in WHERE clause, where

X_n = t_n.value (if the suffix of the value of n is an element name)
a_n.value (if the suffix of the value of n is an attribute name)
the value of the node n (if n is of type `Literal` or `Number`)

(similarly for Y_m)

(7) For the output node n, add:
—e_n.docID, e_n.start, e_n.end (if the suffix of the value of n is an element name)
—a_n.docID, a_n.start, a_n.end (if the suffix of the value of n is an attribute name)
to SELECT clause and ORDER BY clause.
(8) **return** resultant SQL query.

3.2 THE QUERY CONVERSION FOR EDGE METHOD

An XPath query selects a node or a set of nodes from XML graph tree. Therefore an XPath query represents the path of the nodes in the tree.

We assumed that an XPath query consists of one or more steps. Each step begins with a slash. In the evaluate_Edge function, we split XPath query into steps and added necessary clauses for each step.

The algorithm which translates XPath query to SQL query for the Edge method is explained in the following,

Evaluate_Edge (Q, collname, doc)

Input: An XPath Query
Output: An SQL query

Algorithm:

1-Split from slashes into steps and put them in string array **steps**
2-For the first step

if begins with

/*/ : If without predicate

-Add "Edge e1" in FROM list; and
-Add "e1.parentID=0 AND e1.docID=doc"
in WHERE clause.
Else Step_predicate(step,1,0)

/ : If without predicate
-Add "Edge e1" in FROM list; and
-Add "e1.name=step[1] AND e1.parentID=0 AND
e1.docID=doc"
in WHERE clause.

Else Step_predicate(step,1,0)

//* : -Add "Edge e2" in FROM list; and
-Add "e2.docID=doc"
in WHERE clause.

// : if without predicate
-Add "Edge e2" in FROM list; and
-Add "e2.docID=doc AND e2.name=steps[2]"
in WHERE clause.
else
-Step_predicate(step,2,0)

for each step $i whose parent step is $j do the followings $i=3

if step is // : - take next step (means j // i)

-Add "Edge e" in FROM list; and
-Add " AND e$i.nodeID>e$j.nodeID AND e$i.nodeID < ALL (
SELECT min(e.nodeID) FROM e WHERE e.nodeID >
e$j.nodeID AND e.parentID < e$j.nodeID UNION
SELECT max(nodeID) FROM edge) "

if nextstep is
* : -Add "Edge e$i" in FROM list; and
-Add " " in Where clause

/ : If without predicate
-Add "Edge e$i" in FROM list; and
-Add "e$i.name=steps[$i] AND e$i.parentID=0 AND
e1.docID=doc"
in WHERE clause.

Else Step_predicate(step,i,j)

Elseif step is / or * and without predicate
-Add "Edge e$i" in FROM list; and

if /*/

-Add " AND e$i.parentID=e$j.nodeID AND e$i.flg!='att'"
in WHERE clause

else /

-Add " AND e$i.name=steps[$i] AND e$i.parentID=e$j.nodeID
AND e$i.flg!='att'"
in WHERE clause

else Step_predicate(step,$i,$j)

Step_predicate(Step,current,parent)

Input: Step
Output: A SQL query
Algorithm:
`i is the current step,j is the parent step`
`p is the current step of predicate's path`
-split predicate into steps put into array **psteps**
-from this part the algorithm is similar with the evaluate_edge function.

CHAPTER 4

PERFORMANCE STUDY

In this section, we present the performance of a set of XPath queries in XRel and Edge methods. For each query, first the XPath expression is given, then the corresponding SQL statement in XRel and Edge. The charts indicate the response time of each query on three different sizes of two different documents. The experiments were conducted on a Pentium IV 2.40 GHz PC with 1 GB RAM 75 GB hard disk. Windows XP Professional is used. The RDBMS used was MySQL 4.1.1. The test XML documents are two types data-centric and text-centric.

4.1 EXPERIMENT DESIGN

4.1.1 Queries

13 test queries are run on the selected documents during the experiments. Table 4.3 and Table 4.4 show the test queries. When we were selecting queries, we consider choosing different types. The types of queries that we have chosen are below,

- Long XPath (Query 1 and Query 7)
- Short XPath (Query 2 and Query 8)
- Including descendant or self '// ' XPath (Query 3 and Query 9)
- Including '*' wild card XPath. '*' indicates any element. (Query 4 and Query 10)
- Including text predicate XPath (Query 5, Query 11)
- Including range predicate XPath (Query 6 and Query 12)
- Including attribute predicate XPath (Query 13)

The selected queries are below:

Table 4.1 The queries for Shakespeare document

	Query
Query 1	/plays/play/act/scene/title
Query 2	/plays/play
Query 3	//persona
Query 4	/plays/*/act
Query 5	/plays/play/act/scene/speech[speaker='EXTON']
Query 6	/plays/play/act/scene/speech[speaker>='EXTON']

Table 4.2 The queries for Catalog document

	Query
Query 7	/catalog/item/publisher/contact_information/mailing_address/country/name
Query 8	/catalog/item
Query 9	//name
Query 10	/catalog/*/publisher
Query 11	/catalog/item[date_of_release='1932-07-21']
Query 12	/catalog/item[date_of_release>='1932-07-21']
Query 13	/catalog/item[@id='I30']

4.1.2 Documents

In our study we used Catalog.xml (XBench, 2003) as an example of data-centric documents. Data-centric documents have regular structure. Data-centric documents include data from a database. Examples include e-commerce catalogue in XML form. Shakespeare's plays (Bosak, 1999) as an example of text-centric documents. Text-centric documents have irregular structure and include long text data. Examples include book collections in a digital library. The structures of test documents are in Appendix A and B.

A part of each document is given in Appendix C. The characteristics of the test documents are indicated in Table 4.3 and the sizes of the documents when stored in database are shown in Table 4.4.

Table 4.3 The characteristics of the test documents

	Shakespeare	Catalog
Document size	9.99 MB	10.34 MB
Depth	7	8
# of Unique Path	45	67
# of Element node	231477	239527
# of Attribute node	0	14999
# of Text node	191610	147376
Ratio text/document (MB)	0,60	0,42
Ratio element node/total nodes	0,55	0,60

Table 4.4 Table sizes in MySQL

XRel tables	Catalog	Shakespeare
Element	24,077 MB	22,724 MB
Attribute	0,892 MB	0,1 MB
Text	14,697 MB	17,272 MB
Path	0,008 MB	0,006 MB
Total	**39,67 MB**	**40,1 MB**

Edge table	Catalog	Shakespeare
Edge	29,139 MB	30,3 MB

4.1.3 Performance Metrics

The experiments were implemented on one computer with Intel Pentium IV CPU clocked at 2.40 GHz, 1.00 GB of main memory and 75 GB hard disk. The installed operating system was Windows XP Professional. Performance metric could be number of disc read/writes, response time or number of joins. We used response time for our experiments.

4.2 EXPERIMENTAL RESULTS

For each query, first the XPath expression then the corresponding SQL statements for XRel and Edge methods are given. The charts indicate the response times of the average of 5 tests for each query on three different sizes of the two different documents (Shakespeare and Catalog). Document sizes are 5MBs, 7MBs, and 10MBs. The collection names are 'xrelcol' and 'edgecol'. Table indexes are shown in Table 4.5 and Table 4.6.

Table 4.5 The table indexes for XRel method

Tables	***Index 1 (primary key)***	***Index 2***	***Index 3***	***Index 4***
Element	docID+nodeID	parentID	endDesID	pathID
Path	docID+pathID	pathExp		
Text	docID+nodeID	parentID	value	
Attribute	docID+nodeID			

Table 4.6 The Edge table indexes

	Index 1 (primary key)	***Index 2***	***Index 3***	***Index 4***	***Index 5***
Edge table	docID+nodeID	name	textVal	parentID	flg

4.2.1 The Queries and Response Times of the SHAKESPEARE Document

Query 1: /plays/play/act/scene/title

This query retrieves the *titles* inside the *scenes* which are subelement of *act* whose parent is *play* which is subelement of *plays.* For this query in the *Edge* method, produced SQL is shown in the following,

```
SELECT e5.* FROM edgecol_edge as e1,edgecol_edge as e2,edgecol_edge as
e3,edgecol_edge as e4,edgecol_edge as e5 WHERE e1.name='plays' AND
e1.parentID=0 AND e1.docID=3 AND e1.flg!='att' AND e2.name='play' AND
```

e2.parentID=e1.nodeID AND e2.flg!='att'AND e2.docID=3 AND e3.name='act' AND e3.parentID=e2.nodeID AND e3.flg!='att'AND e3.docID=3 AND e4.name='scene' AND e4.parentID=e3.nodeID AND e4.flg!='att'AND e4.docID=3 AND e5.name='title' AND e5.parentID=e4.nodeID AND e5.flg!='att'AND e5.docID=3

For this query in the *XRel* method produced SQL is shown in the following,

SELECT e1.* FROM xrelcol_Path p1, xrelcol_Element e1 WHERE p1.pathexp LIKE '#/plays#/play#/act#/scene#/title' AND e1.DocID=1 AND e1.pathID=p1.pathID AND p1.DocID=1

The query performance for the three sizes of documents is shown in Figure 4.1.

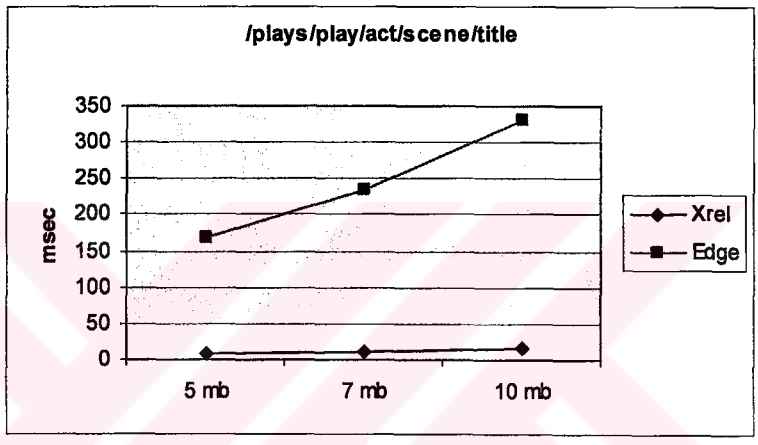

Figure 4.1 The chart of the response times for Query 1.

This is a long XPath query. For long XPath queries in the Edge method, the SQL has many joins. In order to find the children of a group of elements, there is a need to make a join with a new copy of Edge table in the SQL statement for each step in XPath query. So every step results in a join. In this query there are 5 joins. The Edge table is not a small table. Thus these joins in the SQL for the Edge method produce longer response times. But the XRel method uses the Path table to keep track of the ancestors of an element. In the SQL of XRel method, the join is between Path table and Element table. Path is a very small table. Therefore this join doesn't produce a longer response time. As a result XRel is an order of magnitute faster.

Query 2: /plays/play

This query retrieves all the *plays* which is subelement of *play*. For this query in the *Edge* method, produced SQL is shown in the following,

SELECT e2.* FROM edgecol_edge as e1, edgecol_edge as e2 WHERE e1.name='plays' AND e1.parentID=0 AND e1.docID=1 AND e1.flg!='att' AND e2.name='play' AND e2.parentID=e1.nodeID AND e2.flg!='att'AND e2.docID=1

For this query in the *XRel* method produced SQL is shown in the following,

SELECT e1.* FROM xrelcol_Path p1, xrelcol_Element e1 WHERE p1.pathexp LIKE '#/plays#/play' AND e1.DocID=1 AND e1.pathID=p1.pathID AND p1.DocID=1

The query performance for the three sizes of documents is shown in Figure 4.2.

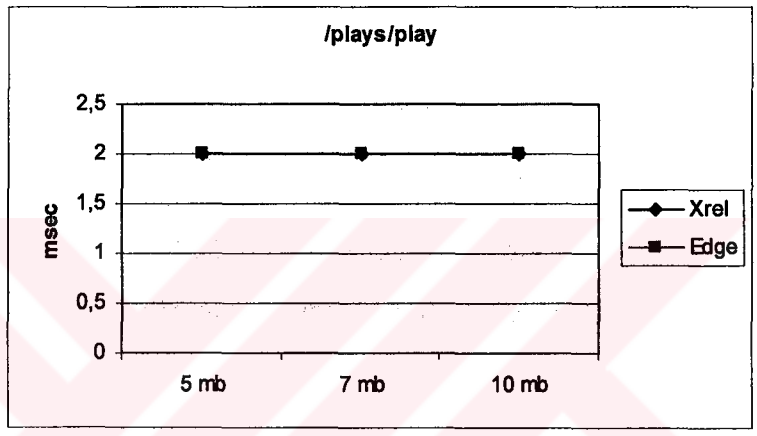

Figure 4.2 The chart of the response times for Query 2.

This query is a short XPath query. So there aren't many joins in the SQL statement for the Edge method. Both methods are fast (the average is 2 secs). The response times are approximately equal.

Query 3: //persona

This query retrieves all the *personas* in the document. For this query in the *Edge* method, produced SQL is shown in the following,

SELECT e2.* FROM edgecol_edge as e2 WHERE e2.name='persona' AND e2.docID=1 AND e2.flg!='att'

For this query in the *XRel* method produced SQL is shown in the following,

SELECT e1.* FROM xrelcol_Path p1, xrelcol_Element e1 WHERE p1.pathexp LIKE '#%/persona' AND e1.DocID=1 AND e1.pathID=p1.pathID AND p1.DocID=1

The query performance for the three sizes of documents is shown in Figure 4.3.

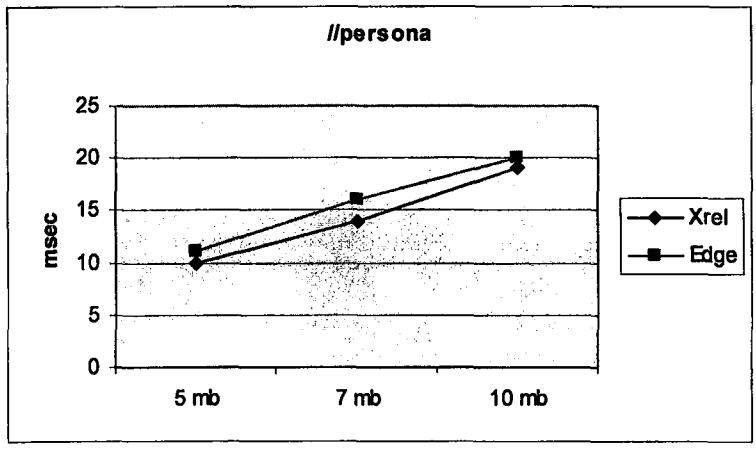

Figure 4.3 The chart of the response times for Query 3.

This is a '//' wild card query. '//' used for self-or-descendant in XPath queries. Although XRel is slightly faster (%10), the response times are very similar for both methods. The SQL statements generated by two methods run in a short time.

Query 4: /plays/*/act

This query retrieves the acts whose parent is subelement of *plays*. For this query in the *Edge* method, produced SQL is shown in the following,

```
SELECT e3.* FROM edgecoll_edge as e1, edgecoll_edge as e2, edgecoll_edge as e3
WHERE e1.name='plays' AND e1.parentID=0 AND e1.docID=3 AND e1.flg!='att'
AND e2.parentID=e1.nodeID AND e2.flg!='att' AND e2.docID=3 AND e3.name='act'
AND e3.parentID=e2.nodeID AND e3.flg!='att'AND e3.docID=3
```

For this query in the *XRel* method produced SQL is shown in the following,

```
SELECT e1.* FROM xrelcol_Path p1,xrelcol_Element e1 WHERE p1.pathexp
REGEXP '^#/plays#/[^/.]+#/act$' AND e1.DocID=1 AND e1.pathID=p1.pathID AND
p1.DocID=1
```

The query performance for the three sizes of documents is shown in Figure 4.4.

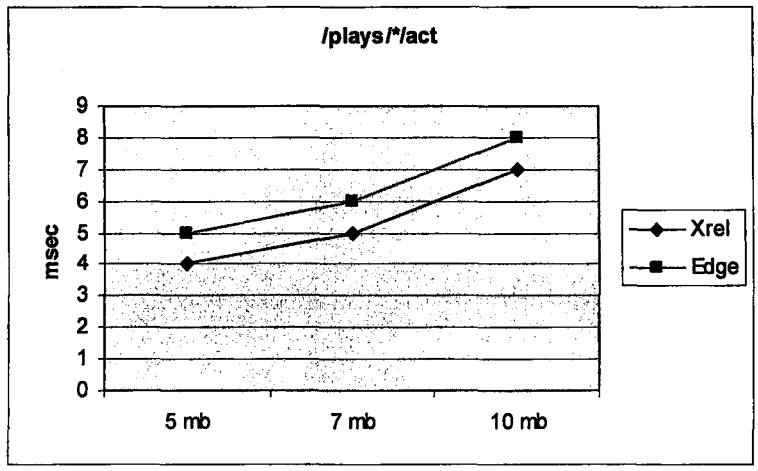

Figure 4.4 The chart of the response times for Query 4.

This is a '*' wild-card-included query. '*' indicates an element but the name is not important. In this query Edge method is slightly slower (%20). The reason could be the number of joins in the SQL for the Edge method. The reason using joins is explained in Query1.

Query 5: /plays/play/act/scene/speech[speaker='EXTON']

This query retrieves the speeches whose subelement speakers' text data equal to 'EXTON' and those speechs are inside the scenes which are subelement of act whose parent is play which is subelement of plays. For this query in the Edge method, produced SQL is shown in the following,

```
SELECT e5.* FROM edgecol_edge as e1,edgecol_edge as e2,edgecol_edge as
e3,edgecol_edge as e4,edgecol_edge as e5,edgecol_edge as ep6 WHERE
e1.name='plays' AND e1.parentID=0 AND e1.docID=1 AND e1.flg!='att' AND
e2.name='play' AND e2.parentID=e1.nodeID AND e2.flg!='att'AND e2.docID=1 AND
e3.name='act' AND e3.parentID=e2.nodeID AND e3.flg!='att'AND e3.docID=1 AND
e4.name='scene' AND e4.parentID=e3.nodeID AND e4.flg!='att'AND e4.docID=1
AND ep6.flg!='att' AND ep6.name='speaker' AND e5.name='speech' AND
e5.parentID=e4.nodeID AND e5.flg!='att' AND ep6.docID=1 AND
ep6.parentID=e5.nodeID AND ep6.TextVal = 'EXTON' AND e5.docID=1
```

For this query in the *XRel* method produced SQL is shown in the following,

```
SELECT e1.* FROM xrelcol_Path p1, xrelcol_Element e1, xrelcol_Path p3,
xrelcol_Text t3 WHERE p1.pathexp LIKE '#/plays#/play#/act#/scene#/speech' AND
e1.DocID=1 AND e1.pathID=p1.pathID AND p1.DocID=1 AND t3.value='EXTON'
```

```
AND p3.pathexp LIKE '#/plays#/play#/act#/scene#/speech#/speaker' AND
t3.pathID=p3.pathID AND p3.DocID=1 AND e1.docID=t3.docID AND e1.nodeId <
t3.parentId AND e1.endDesId >= t3.parentId
```

The query performance for the three sizes of documents is shown in Figure 4.5.

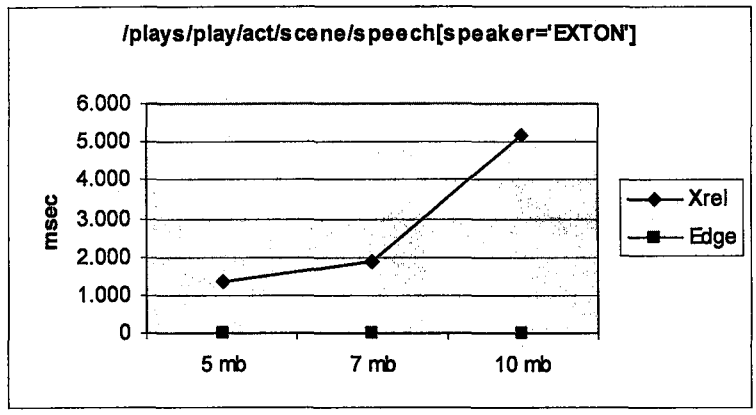

Figure 4.5 The chart of the response times for Query 5.

This text-predicate query didn't produce the results we expected. Although the SQL query for the Edge method has many joins, Edge is much faster than XRel (min 448 times). The comparisons in the SQL query of XRel could be the reason. These comparisons are a part of XRel method. In XRel method in order to find the descendants, the start and end points of an element are kept in element table. Therefore to find the *speakers* which are subelement of *speechs,* nodeID is used for the start of *speech* element and endDesID for the end of the *speech* element. The parentID for text predicate 'EXTON' is *speaker (t3.parentID).* Therefore t3.parentID should be between the nodeID of *speech* and endDesID of *speech.* These two comparisons can make evaluation slower.

Query 6: /plays/play/act/scene/speech[speaker>='EXTON']

This query retrieves the speeches whose subelement speakers' text data greater than or equal to 'EXTON' and those speechs are inside the *scenes* which are subelement of *act* whose parent is *play* which is subelement of *plays.* For this query in the *Edge* method, produced SQL is shown in the following,

```
SELECT e5.* FROM edgecol_edge as e1,edgecol_edge as e2,edgecol_edge as
e3,edgecol_edge as e4,edgecol_edge as e5,edgecol_edge as ep6 WHERE
```

e1.name='plays' AND e1.parentID=0 AND e1.docID=1 AND e1.flg!='att' AND e2.name='play' AND e2.parentID=e1.nodeID AND e2.flg!='att'AND e2.docID=1 AND e3.name='act' AND e3.parentID=e2.nodeID AND e3.flg!='att'AND e3.docID=1 AND e4.name='scene' AND e4.parentID=e3.nodeID AND e4.flg!='att'AND e4.docID=1 AND ep6.flg!='att' AND ep6.name='speaker' AND e5.name='speech' AND e5.parentID=e4.nodeID AND e5.flg!='att' AND ep6.docID=1 AND ep6.parentID=e5.nodeID AND ep6.TextVal >= 'EXTON' AND e5.docID=1

For this query in the *XRel* method produced SQL is shown in the following,

SELECT e1.* FROM xrelcol_Path p1, xrelcol_Element e1, xrelcol_Path p3, xrelcol_Text t3 WHERE p1.pathexp LIKE '#/plays#/play#/act#/scene#/speech' AND e1.DocID=1 AND e1.pathID=p1.pathID AND p1.DocID=1 AND t3.value>='EXTON' AND p3.pathexp LIKE '#/plays#/play#/act#/scene#/speech#/speaker' AND t3.pathID=p3.pathID AND p3.DocID=1 AND e1.docID=t3.docID AND e1.nodeId < t3.parentId AND e1.endDesId >= t3.parentId

The query performance for the three sizes of documents is shown in Figure 4.6.

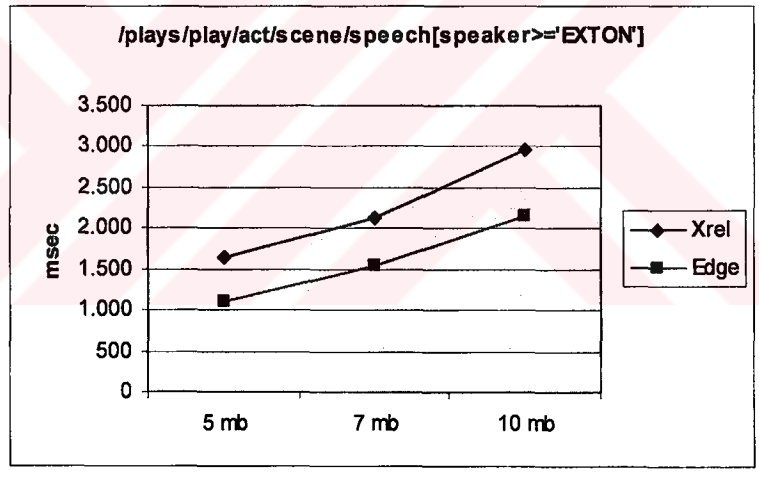

Figure 4.6 The chart of the response times for Query 6.

In this query Edge is faster due to the same reasons as in Query 5. However, Edge is not as fast as in Query 5, because of the increase in the number of records that are returned from this query.

4.2.2 The Queries and Response Times of the CATALOG Document

Query 7: /catalog/item/publisher/contact_information/mailing_address/country/name

This query retrieves the *names* of the *countries* inside the *mailing_addresses* which are subelement of *contact_information* whose parent is *publisher* which is subelement of *items* whose parent is *catalog.* For this query in the *Edge* method, produced SQL is shown in the following,

```
SELECT e7.* FROM edgecol_edge as e1,edgecol_edge as e2,edgecol_edge as
e3,edgecol_edge as e4,edgecol_edge as e5,edgecol_edge as e6,edgecol_edge as e7
WHERE e1.name='catalog' AND e1.parentID=0 AND e1.docID=1 AND e1.flg!='att'
AND e2.name='item' AND e2.parentID=e1.nodeID AND e2.flg!='att'AND e2.docID=1
AND e3.name='publisher' AND e3.parentID=e2.nodeID AND e3.flg!='att'AND
e3.docID=1 AND e4.name='contact_information' AND e4.parentID=e3.nodeID AND
e4.flg!='att'AND e4.docID=1 AND e5.name='mailing_address' AND
e5.parentID=e4.nodeID AND e5.flg!='att'AND e5.docID=1 AND e6.name='country'
AND e6.parentID=e5.nodeID AND e6.flg!='att'AND e6.docID=1 AND
e7.name='name' AND e7.parentID=e6.nodeID AND e7.flg!='att'AND e7.docID=1
```

For this query in the *XRel* method produced SQL is shown in the following,

```
SELECT e1.* FROM xrelcol_Path p1, xrelcol_Element e1 WHERE p1.pathexp LIKE
'#/catalog#/item#/publisher#/contact_information#/mailing_address#/country#/name'
AND e1.DocID=1 AND e1.pathID=p1.pathID AND p1.DocID=1
```

The query performance for the three sizes of documents is shown in Figure 4.7.

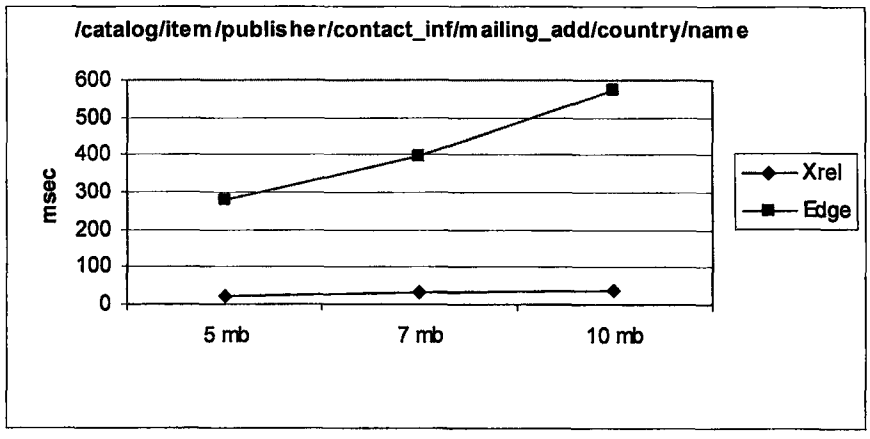

Figure 4.7 The chart of the response times for Query 7.

This is a long XPath query. XRel is an order of magnitute faster. Please see the explanation for Query 5, for the interpretation of the results in this query.

Query 8: /catalog/item

This query retrieves items which are subelement of catalog. For this query in the *Edge* method, produced SQL is shown in the following,

```
SELECT e2.* FROM edgecol_edge as e1, edgecol_edge as e2 WHERE
e1.name='catalog' AND e1.parentID=0 AND e1.docID=1 AND e1.flg!='att' AND
e2.name='item' AND e2.parentID=e1.nodeID AND e2.flg!='att'AND e2.docID=1
```

For this query in the *XRel* method produced SQL is shown in the following,

```
SELECT e1.* FROM xrelcol_Path p1, xrelcol_Element e1 WHERE p1.pathexp LIKE
'#/catalog#/item' AND e1.DocID=1 AND e1.pathID=p1.pathID AND p1.DocID=1
```

The query performance for the three sizes of documents is shown in Figure 4.8.

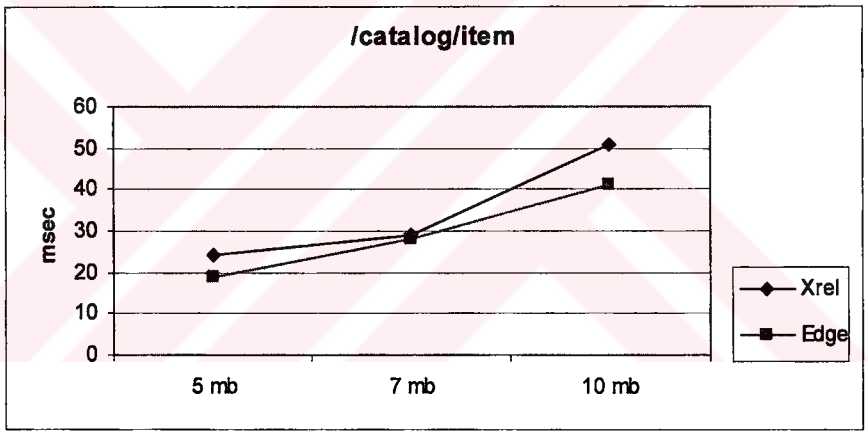

Figure 4.8 The chart of the response times for Query 8.

In this short XPath query response times for both methods are close. Please see the explanation for Query 2, for the interpretation of the results in this query.

Query 9: //name

This query retrieves all the *name* elements of the document. For this query in the *Edge* method, produced SQL is shown in the following,

```
SELECT e2.* FROM edgecol_edge as e2 WHERE e2.name='name' AND e2.docID=1
AND e2.flg!='att'
```

For this query in the *XRel* method produced SQL is shown in the following,

SELECT e1.* FROM xrelcol_Path p1, xrelcol_Element e1 WHERE p1.pathexp LIKE '#%/name' AND e1.DocID=1 AND e1.pathID=p1.pathID AND p1.DocID=1

The query performance for the three sizes of documents is shown in Figure 4.9.

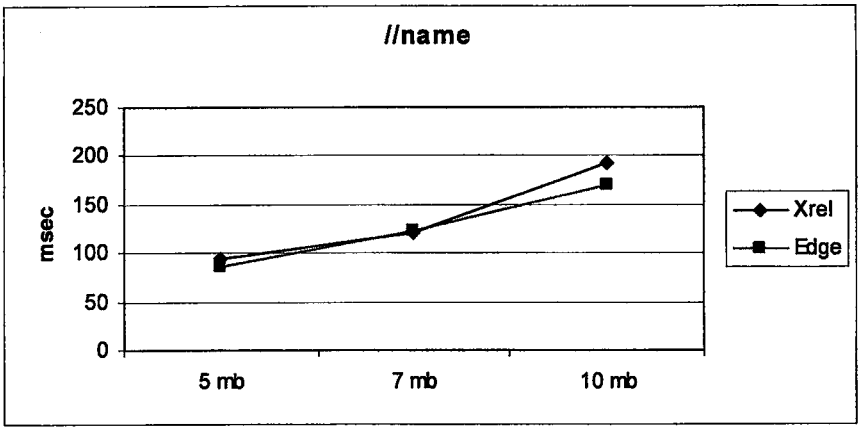

Figure 4.9 The chart of the response times for Query 9.

This short XPath query response times are very close for both methods. Please see the explanation for Query 3, for the interpretation of the results in this query.

Query 10: /catalog/*/publisher

This query retrieves the *publishers* whose parents are a subelement of *catalog.* For this query in the *Edge* method, produced SQL is shown in the following,

SELECT e3.* FROM edgecoll_edge as e1,edgecoll_edge as e2,edgecoll_edge as e3 WHERE e1.name='catalog' AND e1.parentID=0 AND e1.docID=1 AND e1.flg!='att' AND e2.parentID=e1.nodeID AND e2.flg!='att' AND e2.docID=1 AND e3.name='publisher' AND e3.parentID=e2.nodeID AND e3.flg!='att'AND e3.docID=1

For this query in the *XRel* method produced SQL is shown in the following,

SELECT e1.* FROM xrelcol_Path p1, xrelcol_Element e1 WHERE p1.pathexp REGEXP '^#/catalog#/[^/.]+#/publisher$' AND e1.DocID=1 AND e1.pathID=p1.pathID AND p1.DocID=1

The query performance for the three sizes of documents is shown in Figure 4.10.

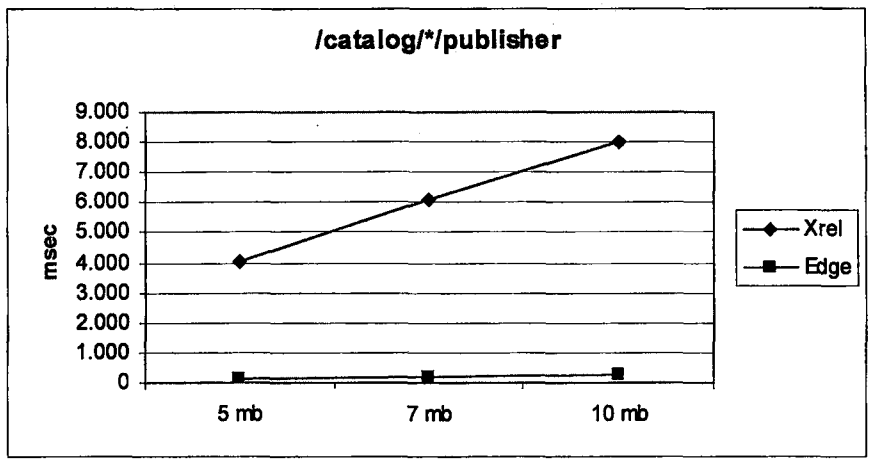

Figure 4.10 The chart of the response times for Query 10.

In this '*' included query XRel is very slower than Edge. This result was not expected.

Query 11: /catalog/item[date_of_release='1932-07-21']

This query retrieves the *items* which are parent of *date_of_release* whose text data is '1932-07-21' and those *items* are the subelement of *catalog*. For this query in the *Edge* method, produced SQL is shown in the following,

SELECT e2.* FROM edgecol_edge as e1, edgecol_edge as e2,edgecol_edge as ep3 WHERE e1.name='catalog' AND e1.parentID=0 AND e1.docID=1 AND e1.flg!='att' AND ep3.flg!='att' AND ep3.name='date_of_release' AND e2.name='item' AND e2.parentID=e1.nodeID AND e2.flg!='att' AND ep3.docID=1 AND ep3.parentID=e2.nodeID AND ep3.TextVal = '1932-07-21' AND e2.docID=1

For this query in the *XRel* method produced SQL is shown in the following,

SELECT e1.* FROM xrelcol_Path p1, xrelcol_Element e1,xrelcol_Path p3,xrelcol_Text t3 WHERE p1.pathexp LIKE '#/catalog#/item' AND e1.DocID=1 AND e1.pathID=p1.pathID AND p1.DocID=1 AND t3.value='1935-12-21' AND p3.pathexp LIKE '#/catalog#/item#/date_of_release' AND t3.pathID=p3.pathID AND p3.DocID=1 AND e1.docID=t3.docID AND e1.nodeId < t3.parentId AND e1.endDesId >= t3.parentId

The query performance for the three sizes of documents is shown in Figure 4.11.

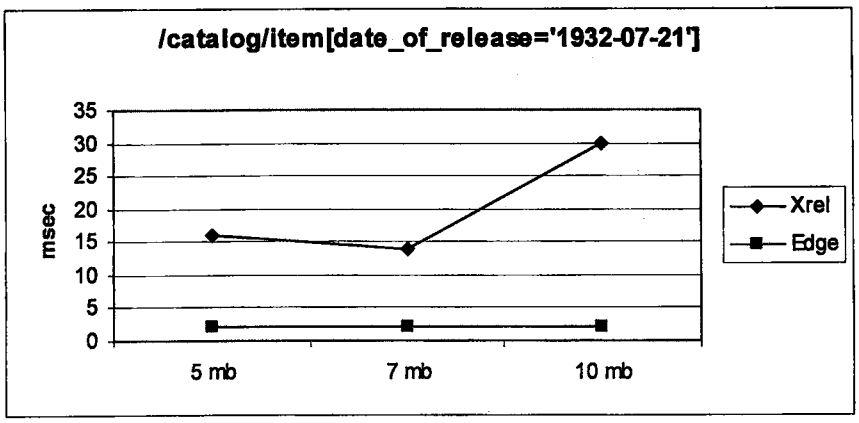

Figure 4.11 The chart of the response times for Query 11.

This text predicate included query Edge is faster. Please see the explanation for Query 5, for the interpretation of the results in this query.

Query 12: /catalog/item[date_of_release>='1935-12-21']

This query retrieves the *items* which are parent of *date_of_release* whose text data is greater than or equal to '1932-07-21' and those *items* are the subelement of *catalog*. For this query in the *Edge* method, produced SQL is shown in the following,

```
SELECT e2.* FROM edgecol_edge as e1, edgecol_edge as e2, edgecol_edge as ep3
WHERE e1.name='catalog' AND e1.parentID=0 AND e1.docID=1 AND e1.flg!='att'
AND ep3.flg!='att' AND ep3.name='date_of_release' AND e2.name='item' AND
e2.parentID=e1.nodeID AND e2.flg!='att' AND ep3.docID=1 AND
ep3.parentID=e2.nodeID AND ep3.TextVal >= '1932-07-21' AND e2.docID=1
```

For this query in the *XRel* method produced SQL is shown in the following,

```
SELECT e1.* FROM xrelcol_Path p1,xrelcol_Element e1,xrelcol_Path p3,xrelcol_Text
t3 WHERE p1.pathexp LIKE '#/catalog#/item' AND e1.DocID=1 AND
e1.pathID=p1.pathID AND p1.DocID=1 AND t3.value>='1932-07-21' AND p3.pathexp
LIKE '#/catalog#/item#/date_of_release' AND t3.pathID=p3.pathID AND p3.DocID=1
AND e1.docID=t3.docID AND e1.nodeId < t3.parentId AND e1.endDesId >=
t3.parentId
```

The query performance for the three sizes of documents is shown in Figure 4.12.

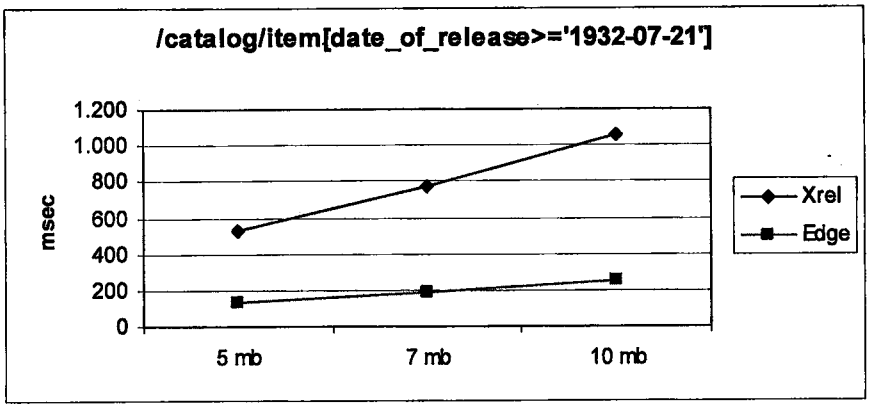

Figure 4.12 The chart of the response times for Query 12.

In this query Edge is faster. Please see the explanation for Query 6, for the interpretation of the results in this query.

Query 13: /catalog/item[@id='I30']

This query retrieves the *items* which are parent of id attribute whose value is 'I30' and those *items* are the subelement of *catalog*. For this query in the *Edge* method, produced SQL is shown in the following,

```
SELECT e2.* FROM edgecol_edge as e1, edgecol_edge as e2, edgecol_edge as ep3
WHERE e1.name='catalog' AND e1.parentID=0 AND e1.docID=1 AND e1.flg!='att'
AND ep3.flg='att' AND ep3.name='id' AND e2.name='item' AND
e2.parentID=e1.nodeID AND e2.flg!='att' AND ep3.docID=1 AND
ep3.parentID=e2.nodeID AND ep3.TextVal = 'I30' AND e2.docID=1
```

For this query in the *XRel* method produced SQL is shown in the following,

```
SELECT e1.* FROM xrelcol_Path p1, xrelcol_Element e1,xrelcol_Path
p3,xrelcol_Attribute a3 WHERE p1.pathexp LIKE '#/catalog#/item' AND e1.DocID=1
AND e1.pathID=p1.pathID AND p1.DocID=1 AND a3.value='I30' AND p3.pathexp
LIKE '#/catalog#/item#/@id' AND a3.pathID=p3.pathID AND p3.DocID=1 AND
e1.docID=a3.docID AND a3.parentID = e1.nodeId
```

The query performance for the three sizes of documents is shown in Figure 4.13.

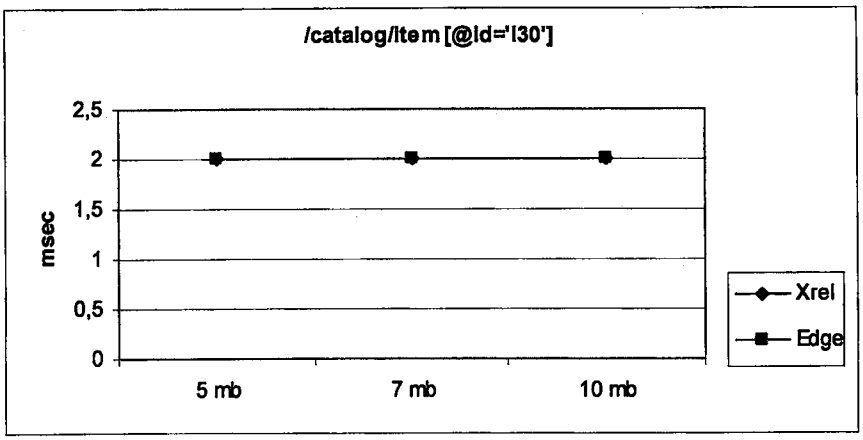

Figure 4.13 The chart of the response times for Query 13.

This is an attribute-predicate query which returns only one record. Because the attributes are indexed in tables, a record is returned in a short response time.

4.2.3 Overall Results

In the Edge method there are more joins then XRel method for the long XPath queries. Therefore we expected Edge to be slower than XRel but as it is shown in results, the difference is not as big as expected and in some queries Edge is better than XRel. The summary of test queries' performance is below,

- In the long XPath queries, *XRel is better than Edge* (Query 1 and Query 7).
- In the short XPath queries, response times for the text-centric document are *the same* but for the data-centric document are *very similar* in both methods. (Query 2 and Query 8).
- In Query 13 which includes attribute predicate, *Edge is faster than XRel.*
- In Query 3 and Query 9 which include a descendent-or-self wild card '// ', their response times are *very similar.*
- In the '*' included queries (Query 4 and Query 10), *XRel is faster.*
- All the queries that include a text predicate, *Edge is faster than XRel* (Query 5, Query 6, Query 11 and Query 12).

As a summary, for data centric and text centric documents, XRel is faster in the long XPath queries and '*' included queries, however, Edge is faster in text-predicate queries.

CHAPTER 5

CONCLUSION

This work is part of a project titled "Storing and querying XML documents in a Relational Database Management System using Middleware" . The XML storage part of the project was implemented by Su, Z. (Su, 2003). The phpMyAdmin 2.4.0 was used as the basis of a middleware implementation. PhpMyAdmin is a web based administration tool for MySQL database. Using the middleware, users can create collections which can be viewed as a XML repository. Users can store XML documents in collections in database and retrieve the whole XML documents from collections. In addition to XML collections, we implemented an XPath query processor. This query processor actually executes the XPath queries by converting to SQL statements.

A set of experiments were designed and conducted to measure the XPath query efficiency of the selected methods (Edge and XRel).

Two types of documents were used, data-centric and text-centric. We used three different sizes of these documents to see how the performance changes according to the size. The results were depicted as charts.

Our initial expectation was that XRel was suppose to be in order of magnitudes, faster than Edge, because it uses the path information. However the results show that this does not seem to be always true. In long XPath queries XRel is faster but in text-predicate queries Edge is faster.

APPENDIX A

THE DTD OF CATALOG DOCUMENTS

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT FAX_number (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT attributes (ISBN, number_of_pages, type_of_book, size_of_book)>
<!ELEMENT author (name, date_of_birth, biography, contact_information)>
<!ELEMENT authors (author+)>
<!ELEMENT biography (#PCDATA)>
<!ELEMENT catalog (item+)>
<!ATTLIST catalog
        xmlns:xsi CDATA #REQUIRED
        xsi:noNamespaceSchemaLocation CDATA #REQUIRED>
<!ELEMENT contact_information (mailing_address, FAX_number?, phone_number, email_address?, web_site?)>
<!ELEMENT cost (#PCDATA)>
<!ATTLIST cost currency CDATA #REQUIRED>
<!ELEMENT country (name, exchange_rate, currency)>
<!ELEMENT currency (#PCDATA)>
<!ELEMENT data (#PCDATA)>
<!ELEMENT date_of_birth (#PCDATA)>
<!ELEMENT date_of_release (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT email_address (#PCDATA)>
<!ELEMENT web_site (#PCDATA)>
<!ELEMENT exchange_rate (#PCDATA)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT height (#PCDATA)>
<!ATTLIST height    unit CDATA #REQUIRED >
<!ELEMENT image (data)>
<!ELEMENT item (title, authors, date_of_release, publisher, subject, description, related_items, media, pricing, attributes)>
<!ATTLIST item  id ID #REQUIRED >
<!ELEMENT item_id (#PCDATA)>
<!ELEMENT last_name (#PCDATA)>
<!ELEMENT length (#PCDATA)>
<!ATTLIST length  unit CDATA #REQUIRED>
<!ELEMENT mailing_address (street_information, name_of_city, name_of_state, zip_code, name_of_country?, country?)>
<!ELEMENT media (thumbnail, image)>
<!ELEMENT middle_name (#PCDATA)>
<!ELEMENT name (#PCDATA | first_name | middle_name | last_name)*>
<!ELEMENT name_of_city (#PCDATA)>
<!ELEMENT name_of_country (#PCDATA)>
<!ELEMENT name_of_state (#PCDATA)>
<!ELEMENT number_of_pages (#PCDATA)>
<!ELEMENT phone_number (#PCDATA)>
<!ELEMENT pricing (suggested_retail_price, cost, when_is_available, quantity_in_stock)>
<!ELEMENT publisher (name, contact_information)>
<!ELEMENT quantity_in_stock (#PCDATA)>
<!ELEMENT related_item (item_id)>
<!ELEMENT related_items (related_item+)>
<!ELEMENT size_of_book (length, width, height)>
<!ELEMENT street_address (#PCDATA)>
<!ELEMENT street_information (street_address+)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT suggested_retail_price (#PCDATA)>
<!ATTLIST suggested_retail_price currency CDATA #REQUIRED >
<!ELEMENT thumbnail (data)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT type_of_book (#PCDATA)>
<!ELEMENT when_is_available (#PCDATA)>
<!ELEMENT width (#PCDATA)>
<!ATTLIST width unit CDATA #REQUIRED >
<!ELEMENT zip_code (#PCDATA)>
```

APPENDIX B

THE DTD OF SHAKESPEARE DOCUMENTS

```
<!ENTITY amp "&#38;#38;">
<!ELEMENT PLAY     (TITLE, FM, PERSONAE, SCNDESCR, PLAYSUBT, INDUCT?, PROLOGUE?, ACT+, EPILOGUE?)>
<!ELEMENT TITLE    (#PCDATA)>
<!ELEMENT FM       (P+)>
<!ELEMENT P        (#PCDATA)>
<!ELEMENT PERSONAE (TITLE, (PERSONA | PGROUP)+)>
<!ELEMENT PGROUP   (PERSONA+, GRPDESCR)>
<!ELEMENT PERSONA  (#PCDATA)>
<!ELEMENT GRPDESCR (#PCDATA)>
<!ELEMENT SCNDESCR (#PCDATA)>
<!ELEMENT PLAYSUBT (#PCDATA)>
<!ELEMENT INDUCT   (TITLE, SUBTITLE*, (SCENE+|(SPEECH|STAGEDIR|SUBHEAD)+))>
<!ELEMENT ACT      (TITLE, SUBTITLE*, PROLOGUE?, SCENE+, EPILOGUE?)>
<!ELEMENT SCENE    (TITLE, SUBTITLE*, (SPEECH | STAGEDIR | SUBHEAD)+)>
<!ELEMENT PROLOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
<!ELEMENT EPILOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
<!ELEMENT SPEECH   (SPEAKER+, (LINE | STAGEDIR | SUBHEAD)+)>
<!ELEMENT SPEAKER  (#PCDATA)>
<!ELEMENT LINE     (#PCDATA | STAGEDIR)*>
<!ELEMENT STAGEDIR (#PCDATA)>
<!ELEMENT SUBTITLE (#PCDATA)>
<!ELEMENT SUBHEAD  (#PCDATA)>
```

APPENDIX C

PARTS OF THE DOCUMENTS

A part of the Shakespeare documents

```
<?xml version="1.0" ?>
- <PLAYS>
- <PLAY>
<TITLE>The Tragedy of Hamlet, Prince of Denmark</TITLE>
- <FM>
<P>ASCII text placed in the public domain by Moby Lexical Tools, 1992.</P>
<P>SGML markup by Jon Bosak, 1992-1994.</P>
<P>XML version by Jon Bosak, 1996-1999.</P>
<P>The XML markup in this version is Copyright © 1999 Jon Bosak. This work
  may freely be distributed on condition that it not be modified or altered in
  any way.</P>
  </FM>
- <PERSONAE>
<TITLE>Dramatis Personae</TITLE>
<PERSONA>CLAUDIUS, king of Denmark.</PERSONA>
<PERSONA>HAMLET, son to the late, and nephew to the present
  king.</PERSONA>
<PERSONA>POLONIUS, lord chamberlain.</PERSONA>
<PERSONA>HORATIO, friend to Hamlet.</PERSONA>
<PERSONA>LAERTES, son to Polonius.</PERSONA>
<PERSONA>LUCIANUS, nephew to the king.</PERSONA>
- <PGROUP>
<PERSONA>VOLTIMAND</PERSONA>
<PERSONA>CORNELIUS</PERSONA>
<PERSONA>ROSENCRANTZ</PERSONA>
<PERSONA>GUILDENSTERN</PERSONA>
<PERSONA>OSRIC</PERSONA>
<GRPDESCR>courtiers.</GRPDESCR>
  </PGROUP>
<PERSONA>A Gentleman</PERSONA>
<PERSONA>A Priest.</PERSONA>
- <PGROUP>
<PERSONA>MARCELLUS</PERSONA>
<PERSONA>BERNARDO</PERSONA>
<GRPDESCR>officers.</GRPDESCR>
  </PGROUP>
<PERSONA>FRANCISCO, a soldier.</PERSONA>
```

A part from the catalog documents used


```
<?xml version="1.0" encoding="UTF-8" ?>
- <!--
 generated by ToXgene Version 1.1a in Mon Jan 20 15:21:20 EET 2003
 -->
- <catalog>
- <item id="I1">
<title>Remarkable,closeclaimsBABABABAOGATOGlearndownthedead,</title>
- <authors>
- <author>
- <name>
<first_name>Sylvie</first_name>
<middle_name>q</middle_name>
<last_name>BABABABAALALIN</last_name>
</name>
<date_of_birth>1844-11-22</date_of_birth>
<biography>dolphins would sleep daringly from the multipliers:somas must
have to nag pains:warthogs should have to thrash closely--ruthless,
dogged somas near the idle, even grouches unwind daring
braids;attainments during the brave, regular ideas boost slyly within the
blithely idle dinos.bold, careful asymptotes must have to sleep thinly;quiet
somas grow dependencies:brave, thin frays hang bravely:bravely ironic
multipliers in place o</biography>
- <contact_information>
- <mailing_address>
- <street_information>
<street_address>8?K!d{KH7D[DEqBY~/r2;2#zXeq6uu6n#G-
[jy{</street_address>
<street_address>OM)9#a#v56y[,MVkqtN+a/acrfPvs</street_address>
</street_information>
<name_of_city>London</name_of_city>
<name_of_state>Indiana</name_of_state>
<zip_code>22821</zip_code>
<name_of_country>Ecuador</name_of_country>
</mailing_address>
<phone_number>+79(623)12528485</phone_number>
<email_address>BABABABAALALINaumd.edu</email_address>
</contact_information>
</author>
- <author>
- <name>
<first_name>Algirdas</first_name>
<middle_name>re</middle_name>
<last_name>BABABABABAALAT</last_name>
</name>
<date_of_birth>1826-11-27</date_of_birth>
```

APPENDIX D

PHP CODE

Collection_Xpath

```
<?php
require('./libraries/grab_globals.lib.php');
$js_to_run = 'functions.js';
require('./header.inc.php');
$err_url = 'tbl_properties.php?' . PMA_generate_common_url($db);
require ('xmlcollection_version2.inc');
require ('evaluate_Edge.php');
require ('evaluate_XRel.php');
?>

<?php
if (isset($submitXPATH)) {

        mysql_select_db($db);
        $xpath=preg_replace("/\\\\/","",$xpath);
        $begin=gettimeofday();

        if ( $ctype =="XRel" ){

                        $sql=evaluateSteps_XRel($xpath,$cname,$dID);
                        echo'<br><b><font color =#FF0000>  XREL </font><br><br><br></b>';

        }
        else {
                        $sql=evaluateSteps_Edge($xpath,$cname,$dID);
                        echo'<br><b><font color =#FF0000>  Edge </font><br><br><br></b>';

        }
        printresult($sql,$begin);

}
else { ?>
<!-- Create a new collection -->
<li>
<form method="post"  ENCTYPE="multipart/form-data" action="<?php echo $PHP_SELF?>">
        <?php echo PMA_generate_common_hidden_inputs($db);
        echo '    <input type="hidden" name="cname" value="'.$collname.'" />'. "\n";

echo '        ' . sprintf('New XPath query on '.htmlspecialchars($collname)) . '
collection  :<br /><br/><br/>' . "\n";
echo '        ' . 'Search ' . ' : ' . "\n";
echo '        ' . '<input type="text" name="xpath" maxlength="160" class="textfield"
value="//article[summary/keyword=\'XML\']//author/family" />' . "\n";
echo '        ' . '<input type="hidden" name="db" value="'.htmlspecialchars($db).'"/>' .
"\n";
echo '        ' . ' <input type="submit"  name="submitXPATH" value="' . $strGo . '"
/>' . "\n";
echo '        '. ' <input type="hidden" name="ctype" value="'.$colltype.'" />'. "\n";
echo '        '. ' <input type="hidden" name="dID" value="'.$documID.'" />'. "\n";

}


?>
</form>
```

Evaluate_Xrel() Function

```
<?php

function splitQuery($term) {

    $separator = '/';
        $resultArr   = array();
        if(((($term[0]=='/') && ($term[1]=='/'))&& !(strpos($term,'['))){
        $resultArr[0]=$term;
                return $resultArr;
        }
    $bracketCounter = 0;
    do {
      $sepLeng =  strlen($separator);
      if (strpos($term, $separator)===FALSE) {
        $resultArr[] = $term;
                break;
      }
      $substituteSep = str_repeat(chr(2), $sepLeng);

      $tmp1 = strpos($term, '(');
      $tmp2 = strpos($term, '[');
          $term2= substr($term,1);
          $tmp3 = strpos($term2,'//');

      if ($tmp1===FALSE) {
        $startAt = (int)$tmp2;
      }
          elseif ($tmp2===FALSE) {
        $startAt = (int)$tmp1;
      }
          if($tmp3===FALSE){
                        $startAt = min($tmp1, $tmp2);
                }
          else {
                        $startAt = min($tmp1, $tmp2);
                        $temp3=(int)$tmp3;
                        $startAt = min($temp3,$startAt)+1;
          }
      $preStr = substr($term, 0, $startAt);
          $postStr = substr($term, $startAt);
      $strLeng = strlen($postStr);

          for ($i=0; $i < $strLeng; $i++) {
        $char = $postStr[$i];
        if ($char=='(' || $char=='[') {
          $bracketCounter++;
                continue;
        }
        elseif ($char==')' || $char==']') {
                    $bracketCounter--;
                }
                if ($bracketCounter == 0) {
                        if ((substr($postStr, $i, $sepLeng) == $separator)&&
                        (substr($postStr,    $i+1, $sepLeng) == $separator)) {

                             for ($j=0; $j<$sepLeng; $j++) {
                                        $postStr[$i+$j] = $substituteSep[$j];
                             }
                         }
                }
     }
        $resultArr = explode($substituteSep, $preStr . $postStr);
    }while (FALSE);

        $resultLeng = count($resultArr);
        for($i=1;$i<$resultLeng;$i++){
                $resultArr[$i]='/'.$resultArr[$i];
         }
return $resultArr;
}
function indexNode($xnode){           // put the index nodes to the Graph
        $Leng=count($xnode);
        $j=$Leng+1;
```

```
        for($i=0;$i<$Leng;$i++){
                $tmp1 = strpos($xnode[$i]['value'], '[');

                if(($tmp1) && ($xnode[$i]['nodeType']!='boolean')){

                        $tmp2 = strpos($xnode[$i]['value'], ']');
                        $a=substr($xnode[$i]['value'], ++$tmp1,$tmp2-$tmp1);

                        if(is_numeric($a)){

                                $xnode[$j-1]['nodeNum']=$j;
                                $xnode[$j-1]['value']= '=';
                                $xnode[$j-1]['parentNum']=$xnode[$i]['nodeNum'];
                                $xnode[$j-1]['nodeType']='index';
                                $xnode[$j-1]['connectComp']=0;

                        // the first part of the comparison
                                $xnode[$j]['value'] ='position()';
                                $xnode[$j]['nodeNum']= $j+1;
                                $xnode[$j]['parentNum']=$j;
                                $xnode[$j]['nodeType']='function';
                                $xnode[$j]['connectComp']=0;

                        // The second part of the comparison
                                $xnode[$j+1]['value'] = $a; //take the second part
                                $xnode[$j+1]['nodeNum']= $j+2;
                                $xnode[$j+1]['parentNum']=$j;
                                $xnode[$j+1]['nodeType']='number';
                                $xnode[$j+1]['connectComp']=0;

                                $xnode[$i]['value']=substr($xnode[$i]['value'],0,--$tmp1);
                                $xnode[$i]['nodeType']='exp';
                        }
                }
        }// end of FOR
        return $xnode;
}
function connectComp($xnode){
 $Leng=count($xnode);
 $j=$Leng+1;
  // find the nodes which connected to boolean nodes
  for($i=0;($i<$Leng);$i++){
        if($xnode[$i]['nodeType']==='boolean') {
            $double=false;
            $tmp1 = strpos($xnode[$i]['value'], '=');
            $tmp2 = strpos($xnode[$i]['value'], '<');
            $tmp3 = strpos($xnode[$i]['value'], '>');

            if($tmp1===false){
                if($tmp2===false){
                        $xnode[$j-1]['value']= '&gt';
                        $pos1=$tmp3;
                        echo'<br>'.$tmp3;
                }
                else{
                        $xnode[$j-1]['value']= '<';
                        $pos1=$tmp2;
                }
             }
             else{
                if($tmp2==$tmp1-1){
                        $xnode[$j-1]['value']= '<=';
                        $pos1=$tmp2;
                        $double=true;
                }
                elseif($tmp3==$tmp1-1){
                                $xnode[$j-1]['value']= '>=';
                                $pos1=$tmp3;
                                $double=true;
                }
                else{
                                $xnode[$j-1]['value']= '=';
                                $pos1=$tmp1;
                }
             }

            $xnode[$j-1]['nodeNum']=$j;
            $xnode[$j-1]['parentNum']=$xnode[$i]['nodeNum'];
            $xnode[$j-1]['nodeType']='boolean';
```

```
            $xnode[$j-1]['connectComp']=0;
            $xnode[$j-1]['concat']='';

            // the first part of the comparison
            $pos2 = strpos($xnode[$i]['value'], '[');
            $xnode[$j]['value'] = substr($xnode[$i]['value'], ++$pos2, $pos1-$pos2);

            $xnode[$j]['nodeNum']= $j+1;
            $xnode[$j]['parentNum']=$j;
            $xnode[$j]['nodeType']='exp';
            $xnode[$j]['connectComp']=1;
            $xnode[$j]['concat']='';

            // The second part of the comparison
            $pos3 = strpos($xnode[$i]['value'], ']');
            if($double==false)
            //take the second part
             $xnode[$j+1]['value'] = substr($xnode[$i]['value'], ++$pos1, $pos3-$pos1);
            else{
             $pos1++;
             // if there is <= or >= second part begins +1
             $xnode[$j+1]['value'] = substr($xnode[$i]['value'], ++$pos1, $pos3-$pos1);
            }
             $xnode[$j+1]['nodeNum']= $j+2;
             $xnode[$j+1]['parentNum']=$j;
             if(is_numeric($xnode[$j+1]['value']))
                $xnode[$j+1]['nodeType']='number';
             else
                $xnode[$j+1]['nodeType']='literal';
             $xnode[$j+1]['connectComp']=1;
             $xnode[$j+1]['concat']='';
             $xnode[$i]['value']=substr($xnode[$i]['value'], 0, --$pos2);
             $xnode[$i]['nodeType']='exp';
             $xnode[$i]['concat']='';
        }
    }
return $xnode;
}

function foo($xnodei){
            $leng=strlen($xnodei);
            for($j=0;$j<$leng;$j++){
               if(($xnodei[$j]=='/') && ($xnodei[$j+1]!='/')){

                        $pre=substr($xnodei,0,$j);
                        $post=substr($xnodei,$j);
                        $xnodei=$pre.'#'.$post;
                        $j++;$leng++;
               }
               if(($xnodei[$j]=='/') && ($xnodei[$j+1]=='/')){

                                $xnodei[$j]='%';
                                $pre=substr($xnodei,0,$j);
                                $post=substr($xnodei,$j);
                                $xnodei=$pre.'#'.$post;
                                $j+=2;
               }

             }

             return $xnodei;
}

// findIndex function finds the place of a node(nodeNum is given) in the array.

function findIndex($nodeNum,$xnode){
            $Leng=count($xnode);
            for ($i=0;$i<$Leng;$i++){
               if($xnode[$i]['nodeNum']==$nodeNum)
                  break;
            }
             return $i;
}


function concatenate($xnode){
        $Leng=count($xnode);
        $j=0;
        $path='';
```

```
        for($i=0;$i<$Leng;$i++){
           if ($xnode[$i]['nodeType']!='exp')
                continue;
           //the concatenated value of an exp node which is connected comparison edge

           if($xnode[$i]['connectComp'])
           // we need an extra '/' for it.
           // like xx[summary/keyword='XML'] to make xx/summary/keyword
               $path='/'.$xnode[$i]['value'];
           else
               $path=$xnode[$i]['value'];
           $j=$xnode[$i]['parentNum'];

           while ($j!=-1){
                      // index of the element which node number is $j
                      $xj=findIndex($j,$xnode);
                      //if it is a boolean type, don't put it in the concatenated value
                      if($xnode[$xj]['nodeType']!='boolean'){
                             $path=$xnode[$xj]['value'].$path;
                      }
                      $j=$xnode[$xj]['parentNum'];


               }
               // store the concatenated value into the node
               $xnode[$i]['concat']=$path;
        }
        return $xnode;
}

function lookStar($x,&$where){

        if ( strpos($x['concat'],'*') === false ){
               $where=$where.' AND p'.$x['nodeNum'].'.pathexp LIKE
               '.'\''.foo($x['concat']).'\'';
        }
        else {
               $a=foo($x['concat']);
               $a='^'.str_replace('*','[^/.]+',$a).'$'      ;
               $where=$where.' AND p'.$x['nodeNum'].'.pathexp REGEXP '.'\''.$a.'\'';
        }

}


function convertSQL($xnode,$output,$collection,$docID){

        $Leng=count($xnode);
        $from=' FROM ';
        $where='';
        $select='SELECT ';
        $orderby=' ORDER BY ';
        for($i=0;$i<$Leng;$i++){
// (1)
          if ($xnode[$i]['nodeType']=='exp'){
                      $from=$from.$collection.'_Path p'.$xnode[$i]['nodeNum'].',';
                      lookStar($xnode[$i],&$where);

           }
// (2)
           if (($xnode[$i]['nodeType']=='exp') && ($xnode[$i]['connectComp']==0)){
               if(!strpos($xnode[$i]['concat'],'@')){
                      $from=$from.$collection.'_Element e'.$xnode[$i]['nodeNum'].',';
                      $where=$where.' AND e'.$xnode[$i]['nodeNum'].'.DocID='.$docID.'
                      AND e'.$xnode[$i]['nodeNum'].'.pathID='.
                      'p'.$xnode[$i]['nodeNum'].'.pathID ';
               }
               else{
                 $from=$from.$collection.'_Attribute a'.$xnode[$i]['nodeNum'].',';
                 $where=$where.' AND a'.$xnode[$i]['nodeNum'].'.pathID='.
                 'p'.$xnode[$i]['nodeNum'].'.pathID ';
                       }
               }


//(3)
           if (($xnode[$i]['connectComp']==1)){
               if(strpos($xnode[$i]['concat'],'@')){
                      $from=$from.$collection.'_Attribute a'.$xnode[$i]['nodeNum'].',';
```

```
                $where=$where.' AND a'.$xnode[$i]['nodeNum'].'.pathID='.
                'p'.$xnode[$i]['nodeNum'].'.pathID';
            }
            elseif($xnode[$i]['nodeType']=='exp') {
                $from=$from.$collection.'_Text t'.$xnode[$i]['nodeNum'].',';
                $where=$where.' ANDt'.$xnode[$i]['nodeNum'].'.pathID='.
                'p'.$xnode[$i]['nodeNum'].'.pathID';
                }
            }
// (4)
            $j=findIndex($xnode[$i]['parentNum'],$xnode);
            if($j<$Leng){
                if (($xnode[$i]['nodeType']=='number') &&
                $xnode[$j]['nodeType']=='index')){
                    $k=$xnode[$i]['value'];
                    $n=findIndex($xnode[$j]['parentNum'],$xnode);
                    $where=$where.' AND e'.$xnode[$n]['nodeNum'].'.index='.$k;

                }
            }
// (5)

    $n=0;$m=0;
    $j=findIndex($xnode[$i]['parentNum'],$xnode);

    if($j<$Leng){


        if(($xnode[$i]['nodeType']=='exp')&&($xnode[$j]['nodeType']=='exp')){

            $m=$xnode[$i]['nodeNum'];
            $n=$xnode[$j]['nodeNum'];
            $h=$j;
        }

        elseif(($xnode[$i]['nodeType']=='exp')&&((($xnode[$j]['nodeType']=='boolean'
        ) or ($xnode[$j]['nodeType']=='index')))) {

                        $h=findIndex($xnode[$j]['parentNum'],$xnode);
                        if($xnode[$h]['nodeType']=='exp'){
                            $n=$xnode[$h]['nodeNum'];
                            $m=$xnode[$i]['nodeNum'];
                        }
        }
    }
    if(($n!=0)&&($m!=0)){
            // for x
            if(($xnode[$h]['nodeType']=='exp') && ($xnode[$h]['connectComp']==0))
                        $x='e';
            if(($xnode[$h]['nodeType']=='exp') && ($xnode[$h]['connectComp']==1))
                        $x='t';
            if(strpos($xnode[$h]['concat'],'@'))
                        $x='a' ;
            // for y

            if(($xnode[$i]['nodeType']=='exp') && ($xnode[$i]['connectComp']==0))
                        $y='e';
            if(($xnode[$i]['nodeType']=='exp') && ($xnode[$i]['connectComp']==1))
                        $y='t';
            if(strpos($xnode[$i]['concat'],'@'))
                        $y='a' ;
            $where=$where.' AND '.$x.$n.'.docID=' .$y.$m.'.docID ';
            if(($x=='e')&&($y=='e'))
                    $where=$where.' AND '.$x.$n. '.endDesId >=' .$y.$m.'.endDesId ';

            if(($x!='t')&& ($y=='a'))
                    $where=$where.' AND '.$y.$m. '.parentID = '.$x.$n.'.nodeId ';
            else
                    if(($x!='t')&& ($y!='t'))
                        $where=$where.' AND '.$x.$n. '.nodeId < '.$y.$m.'.nodeId ';
                    elseif($x=='t') {
                        //parent ID in text table  equal to nodeId in element table
                         $where=$where.' AND '.$x.$n. '.parentId > '.$y.$m.'.nodeId ';
                         $where=$where.' AND '.$x.$n. '.endDesId >= '.$y.$m.'.parentId
';
                    }
                    else if($y=='t'){
```

```
                                $where=$where.'AND'.$x.$n. '.nodeId < '.$y.$m.'.parentId ';
                                $where=$where.'AND'.$x.$n.'.endDesId>='.$y.$m.'.parentId';

                                }
                }

//(6)

        $x='';$y='';
        if($xnode[$i]['nodeType']=='boolean'){
                $s=array();
                $f=0;
                for($k=1;$k<$Leng;$k++){

                        if($xnode[$k]['parentNum']==$xnode[$i]['nodeNum']){
                                $s[$f]=findIndex($xnode[$k]['nodeNum'],$xnode);
                                $f++;
                        }
                }
                        $n=$xnode[$s[0]]['nodeNum'];
                        $m=$xnode[$s[1]]['nodeNum'];
                        if($xnode[$s[0]]['nodeType'] == 'exp' )
                                $x='t'.$n.'.value';
                        if(strpos($xnode[$s[0]]['concat'],'@'))
                                $x='a'.$n.'.value';
                        if(($xnode[$s[0]]['nodeType'] == 'literal' ) or
($xnode[$s[0]]['nodeType'] == 'number' ))
                                $x=$xnode[$s[0]]['value'];
                        if($xnode[$s[1]]['nodeType'] == 'exp' )
                                $y='t'.$m.'.value';
                        if(strpos($xnode[$i]['concat'],'@'))
                                $y='a'.$m.'.value';
                        if(($xnode[$s[1]]['nodeType'] == 'literal' ) or
($xnode[$s[1]]['nodeType'] == 'number' ))
                                $y=$xnode[$s[1]]['value'];

                        $where=$where.' AND '.$x.$xnode[$i]['value'].$y.' ';
        }//end of IF (6)
// (7)
        $x='';$y='';
        if($xnode[$i]['nodeNum']==$output){

                if(strpos($xnode[$i]['concat'],'@')){

                        $select=$select.'a'.$output.'.*';

        $orderby=$orderby.'e'.$output.'.docID,e'.$output.'.nodeId,e'.$output.'.endDesId';
                }
                elseif($xnode[$i]['nodeType']=='exp'){

                        $select=$select.'e'.$output.'.*';

        $orderby=$orderby.'e'.$output.'.docID,e'.$output.'.nodeId,e'.$output.'.endDesId';
                }
        }
  } // end of for

        $from=substr($from,0,-1); // to delete the last coma (,)
        $where=substr($where,5); // to delete first AND
        $where=' WHERE '.$where;
        $sql=$select.$from.$where;
        return $sql;
}
function print_splitted($termArr){
        $Leng=count($termArr);

        for($i=0;$i<$Leng;$i++){
                echo $termArr[$i];
            echo '<br>';
        }
}
function evaluateSteps_XRel ($term, $collection,$docID){

  $node[]= array(
                'nodeNum'       => 0,
                'value'         => '',  // The expression value.
                'parentNum'      => 0,  // The nodeId of parent node
                                        // or -1 if this node is the 'super root'
                'nodeType'      => '',  // expression, comparison, literal or number
```

```
                'connectComp'  => 0,   // if connected to a comparison edge 1
                'concat' => 'cc' );    // the concatenate value of this node

$termArr = splitQuery($term);
        $parent=-1;

        $Leng=count($termArr);
        for($i=0;$i<$Leng;$i++){

                $node[$i]['nodeNum']= $i+1;
                $node[$i]['value']= $termArr[$i];
            $node[$i]['parentNum']=$parent;
            $node[$i]['nodeType']='exp';
            $node[$i]['connectComp']=0;
                $node[$i]['concat']='cc';
                $parent=$i+1;
        }
        $output=$i;

  for($i=0;($i<$Leng);$i++){   // to recognise boolean type nodes

        $comp = FALSE;
        $len=strlen($node[$i]['value']);

         for($j=0;($j<$len);$j++){

                $char = $node[$i]['value'][$j];
                if(($char=='=')or($char=='<')or($char=='>')){
                        $comp = TRUE;
                        break;
                }
         }
         if($comp)
                $node[$i]['nodeType']='boolean';
 }
$node=connectComp($node);
$node=indexNode($node);
$node=concatenate($node);

$sql=convertSQL($node,$output,$collection,$docID);

return $sql;
}

function printresult($sql,$begin){
echo'<br><br>'.$sql.'<br>';

if(!($result= mysql_query($sql))){
                 echo "<p>SQL could not executed: <br> <br>".mysql_error()."<br>";
        }

        else{
                echo'<br>';
                $time=calctime($begin);
                echo'<br><b>TIME </b> <br>'.'<font color =#FF0000>'.$time['sec'].'  sec';
                echo'   '.round($time['milisec']).'  milisec<br></font>';
                echo '<br> <b> number of rows <br> </b><font color =#FF0000>
                '.mysql_num_rows($result).'</font>';

                print_result($result);

        }
}
function calctime($begin){

 $end=gettimeofday();

 if($end['usec']>$begin['usec'])
        $msectime=($end['usec']-$begin['usec'])/1000;
 else {
        $end['sec']--;
        $e=$end['usec']+1000000;
        $msectime=($e-$begin['usec'])/1000;
 }
 $sectime=($end['sec']-$begin['sec']);
 $t=array ( 'sec'=>$sectime,'milisec'=>$msectime);
 return $t;

}
```

```
?>
```

Evaluate_Edge() Function

```
<?php

function evaluateSteps_Edge($xpath,$cname,$docID){

        $where='';
        $from='FROM ';

        $steps = splitSteps('/',$xpath);

        if(!($steps[1]==='')){

                if($steps[1]=='*'){
                        $from=$from.$cname.'_edge as e1,';
                        $where=$where." e1.parentID=0 AND e1.docID=$docID AND
e1.flg!='att'";
                }
                elseif(strpos($steps[1],'[')){ //simple xpath with predicate
                                $i=1;$j=0;

        step_predicate($steps[$i],&$i,&$j,&$from,&$where,$cname,$docID);

                        }
                        else { // simple xpath without predicate
                                $from=$from.$cname.'_edge as e1,';
                                $where=$where." e1.name='$steps[1]' AND e1.parentID=0 AND
e1.docID=$docID AND e1.flg!='att'";
                        }


                $i=2;
        }
        else{    // self or descendents

                if($steps[2]=='*'){
                        $from=$from.$cname."_edge as e2,";
                        $where=$where." e2.docID=$docID AND e2.flg!='att'"; // '//*'
brings whole document
                }
                elseif(strpos($steps[2],'[')){ //absolute xpath with predicate
                                $i=2;$j=0;

        step_predicate($steps[$i],&$i,&$j,&$from,&$where,$cname,$docID);

                                $where=substr($where,4);
                        }
                        else { // absolute xpath without predicate
                                $from=$from.$cname."_edge as e2,";
                                $where=$where." e2.name='$steps[2]' AND e2.docID=$docID
AND e2.flg!='att'";
                        }
                $i=3;
        }  // IF

        for(;$i<count($steps);$i++){

                if($steps[$i]===''){
                        $i++;
                        $j = $i-2;

                        $where=$where." AND e$i.nodeID>e$j.nodeID AND e$i.nodeID < ALL (
                        SELECT min( e.nodeID )
                        FROM ".$cname."_edge AS e
                        WHERE e.nodeID > e$j.nodeID AND e.parentID < e$j.nodeID AND
e.docID=$docID AND e.flg!='att'
                        UNION SELECT max FROM maxnum) ";
                        if($steps[$i]=='*'){
                                $from=$from.$cname."_edge as e$i,";
                                $where=$where." AND e$i.docID=$docID AND e$i.flg!='att'";
                        }
                        else{    //simple xpath  (might have a predicate)
                                if((strpos($steps[$i],'['))){

        step_predicate($steps[$i],&$i,&$j,&$from,&$where,$cname);
                                }
                                else{
```

```
                                        $from=$from.$cname."_edge as e$i,";
                                        $where=$where." AND e$i.name='$steps[$i]' AND
e$i.flg!='att' AND e$i.docID=$docID";
                                }
                        }
                }
                else{
                        if(!(strpos($steps[$i],'['))){    // without predicate
                                $from=$from.$cname."_edge as e$i,";
                                $j=$i-1;
                                if($steps[$i]=='*')
                                        $where=$where." AND e$i.parentID=e$j.nodeID AND
e$i.flg!='att' AND e$i.docID=$docID";
                                else
                                        $where=$where." AND e$i.name='$steps[$i]' AND
e$i.parentID=e$j.nodeID AND e$i.flg!='att'AND e$i.docID=$docID";
                        }
                        else{
                                $j=$i-1;

        step_predicate($steps[$i],&$i,&$j,&$from,&$where,$cname,$docID);

                        }
                }


        }       //  FOR
        $j = $i-1;

        if(!empty($where)) $where=' WHERE '.$where;
        $select = "SELECT e$j.* ";
        $from=substr($from,0,-1);
        $sql=$select.$from.$where;
        return $sql;
}

function step_predicate($step,&$i,&$j,&$from,&$where,$cname,$docID){

                $double=false;
                $tmp1 = strpos($step, '=');
                $tmp2 = strpos($step, '<');
        $tmp3 = strpos($step, '>');

                if($tmp1===false){
                        if($tmp2===false){
                                $operator= '&gt';
                                $pos1=$tmp3;
                        }
                        else{
                                $operator= '<';
                                $pos1=$tmp2;
                        }
                }
                else{
                        if($tmp2==$tmp1-1){
                                $operator= '<=';
                                $pos1=$tmp2;
                                $double=true;
                        }
                        elseif($tmp3==$tmp1-1){
                                        $operator= '>=';
                                        $pos1=$tmp3;
                                        $double=true;
                                }
                                else{
                                        $operator= '=';
                                        $pos1=$tmp1;
                                }
                }
/*////////////////////////////////////////////////////////////////////////////*/
                if(strpos($step,'/')){
                        $pSteps=array();
                        $pos2 = strpos($step, '[');
                        $pos3 = strpos($step, ']');
                        $inside = substr($step, ++$pos2, $pos3-$pos2);
                        $step=substr($step, 0, --$pos2);

                        $pSteps=splitSteps('/',$inside)       ;
```

```
                        if(!(($tmp1===false)and ($tmp1===false)and($tmp1===false))){
                                $last=$pSteps[count($pSteps)-1];
                                $pos2 = 0;
                                $pos3 = strlen($last);
                                $pos1 = strpos($last,'=');

                                if ($double) $pos1--;
                                $a = substr($last, $pos2, $pos1-$pos2);

                                if($double==false)
                                        $b = substr($last, ++$pos1, $pos3-$pos1);
                                else{
                                        $pos1++;
                                        $b = substr($last, ++$pos1, $pos3-$pos1);
                                }
                                $pSteps[count($pSteps)-1]=$a;


        comp_path_pred($pSteps,$step,&$i,&$j,&$where,&$from,$cname,$operator,$b,$docID);

                        }
                        else

        handle_path_pred($pSteps,$step,&$i,&$j,&$where,&$from,$cname,$docID);

                }

                else if(!(($tmp1===false)and ($tmp1===false)and($tmp1===false))){ //
comparisons

                        $pos2 = strpos($step, '[');
                        $a = substr($step, ++$pos2, $pos1-$pos2);// The first part $a

                        $pos3 = strpos($step, ']');
                        if($double==false)
                                $b = substr($step, ++$pos1, $pos3-$pos1); // The second
part $b
                        else{
                                $pos1++;
                                $b = substr($step, ++$pos1, $pos3-$pos1);
                        }
                        $step=substr($step, 0, --$pos2); //inside the step there is only
element now

                        $k=$i+1;    // $k predicate, $i current node

                        $s=strpos($a,'@');

                        if($s!==false){
                                $a=substr($a,1);
                                $where = $where." AND ep$k.flg='att'";
                        }
                        else
                                $where = $where." AND ep$k.flg!='att'";

                        if($a!='*')
                                $where = $where." AND ep$k.name='$a' ";

                        if($step!='*')
                                $where = $where." AND e$i.name='$step' ";


                                // $j parent of current
                                $from=$from.$cname."_edge as e$i,".$cname."_edge as
ep$k,";
                                if(($i-$j)==2)
                                        $where=$where." AND e$i.flg!='att' AND
e$i.docID=$docID
                                        AND ep$k.parentID=e$i.nodeID AND ep$k.TextVal
$operator $b AND ep$k.docID=$docID";
                                 else
                                        $where=$where." AND e$i.parentID=e$j.nodeID AND
e$i.flg!='att' AND ep$k.docID=$docID
                                         AND ep$k.parentID=e$i.nodeID AND ep$k.TextVal
$operator $b AND e$i.docID=$docID";

                }
                else{    // without comparison
                        $pos2 = strpos($step, '[');
```

```
                        $pos3 = strpos($step, ']');
                        $inside = substr($step, ++$pos2, $pos3-$pos2);
                        $step=substr($step, 0, --$pos2);

                        $k=$i+1;  // $k predicate, $i current node
                        // $j parent of current

                        $s=strpos($inside,'@');

                        if($s===false && is_numeric($inside)){   // index node

                                if($step!='*')
                                        $where = $where." AND e$i.name='$step' ";

                                $from=$from.$cname."_edge as e$i,";
                                if(($i-$j)==2)
                                        $where=$where." AND e$i.flg!='att'
                                         AND e$i.ordinal=".intval($inside);
                                 else
                                        $where=$where." AND e$i.flg!='att' AND
e$i.parentID=e$j.nodeID AND e$i.docID=$docID
                                         AND e$i.ordinal=".intval($inside);
                        }
                        else if($s===false){

                                $from = $from.$cname."_edge as e$i,".$cname."_edge as
ep$k,";
                                if($inside!='*')
                                        $where = $where." AND ep$k.name='$inside' ";

                                if($step!='*')
                                        $where = $where." AND e$i.name='$step' ";

                                if(($i-$j)==2)
                                        $where=$where." AND e$i.flg!='att' AND
e$i.docID=$docID
                                        AND ep$k.parentID=e$i.nodeID AND ep$k.flg!='att'
AND ep$k.docID=$docID";
                                 else
                                        $where=$where." AND e$i.parentID=e$j.nodeID AND
e$i.flg!='att' AND e$i.docID=$docID
                                         AND ep$k.parentID=e$i.nodeID AND ep$k.flg!='att'
AND ep$k.docID=$docID  ";

                                }
                                else{   // [@name]  or [@*]
                                        $inside=substr($inside,1);

                                        if($inside!='*')
                                                $where = $where." AND ep$k.name='$inside'
";

                                        if($step!='*')
                                                $where = $where." AND e$i.name='$step' ";


                                        $from=$from.$cname."_edge as e$i,".$cname."_edge as
ep$k,";
                                        if(($i-$j)==2)
                                                $where=$where." AND e$i.flg!='att' AND
e$i.docID=$docID
                                                AND ep$k.parentID=e$i.nodeID AND
ep$k.flg='att' AND ep$k.docID=$docID";
                                         else
                                                        $where=$where." AND
e$i.parentID=e$j.nodeID AND e$i.flg!='att' AND e$i.docID=$docID
                                                         AND ep$k.parentID=e$i.nodeID AND
ep$k.flg='att' AND ep$k.docID=$docID";

                                }
                }
return 0;
}

function comp_path_pred($pSteps,$step,&$i,&$j,&$where,&$from,$cname,$operator,$b,$docID)
{

                        $p=1;
```

```
                        $from=$from.$cname."_edge as e$i,";

                        if($i-$j!=2){

                                if($i==1)
                                        $where=$where." e$i.parentID=0 AND
e$i.docID=$docID";
                                else
                                        $where=$where." AND e$i.parentID=e$j.nodeId AND
e$i.docID=$docID";
                        }
                        if($step!='*')
                                        $where=$where." AND e$i.name= '$step' ";


                $p=1;

                if($pSteps[0]=='' && $pSteps[1]==''){

                                array_shift($pSteps);

                }

                if(!($pSteps[0]=='')){  // inside path is simple

                        if($pSteps[0]=='*'){
                                $from=$from.$cname."_edge as p0,";
                                $where=$where." AND p0.parentID=e$i.nodeId ";
                        }
                        else {
                                $from=$from.$cname.'_edge as p0,';
                                $where=$where." AND p0.name='$pSteps[0]' AND
p0.parentID=e$i.nodeID ";
                        }
                }
                else{

                        $from=$from.$cname."_edge as p1,";

                        if($pSteps[1]!='*'){
                                $where=$where." AND p1.name='$pSteps[1]'";
                        }

                        $where=$where." AND p$p.nodeID>e$i.nodeID AND p$p.nodeID < ALL (
                        SELECT min( e.nodeID )
                        FROM ".$cname."_edge AS e
                        WHERE e.nodeID > e$i.nodeID AND e.parentID < e$i.nodeID AND
e.docID=$docID
                        UNION SELECT max FROM maxnum)";

                $p=2;
        }  // IF

        for(;$p<count($pSteps);$p++){

                if($pSteps[$p]==''){
                        $p++;
                        $q = $p-2;

                        $where=$where." AND p$p.nodeID>p$q.nodeID AND p$p.nodeID < ALL (
                        SELECT min( e.nodeID )
                        FROM ".$cname."_edge AS e
                        WHERE e.docID=$docID AND e.nodeID > p$q.nodeID AND e.parentID <
p$q.nodeID
                        UNION SELECT max FROM maxnum )";

                        $from=$from.$cname."_edge as p$p,";
                        if($pSteps[$p]!='*'){
                                $where=$where." AND p$p.name='$pSteps[$p]' AND
p$p.flg!='att'";
                        }
                }
                else{
                        if(!(strpos($pSteps[$p],'['))){
                                        $from=$from.$cname."_edge as p$p,";
                                        $q=$p-1;
                                        if($pSteps[$p]=='*') $where=$where." AND
p$p.parentID=p$q.nodeID AND p$p.flg!='att'";
```

```
                                        else $where=$where." AND p$p.name='$pSteps[$p]' AND
p$p.parentID=p$q.nodeID AND p$p.flg!='att'";
                        }
                }
        }       //  FOR
                $p--;
                $where=$where."AND p$p.textVal $operator $b AND p$p.docID=$docID"   ;
}
function handle_path_pred($pSteps,$step,&$i,&$j,&$where,&$from,$cname,$docID)
{

                        $p=1;

                        $from=$from.$cname."_edge as e$i,";

                        if($i-$j!=2){

                                if($i==1)
                                        $where=$where." e$i.parentID=0 ";
                                else
                                        $where=$where." AND e$i.parentID=e$j.nodeId ";
                        }
                        if($step!='*')
                                        $where=$where." AND e$i.name= '$step' ";

        $p=1;

                if($pSteps[0]=='' && $pSteps[1]==''){

                                array_shift($pSteps);
                }

                if(!($pSteps[0]=='')){

                        if($pSteps[0]=='*'){ // handle  [/*/AAA]
                                $from=$from.$cname."_edge as p0,";
                                $where=$where." AND p0.parentID=e$i.nodeId ";
                        }
                        else {  // handle [AAA/BBB]
                                $from=$from.$cname.'_edge as p0,';
                                $where=$where." AND p0.name='$pSteps[0]' AND
p0.parentID=e$i.nodeID ";
                        }
                }
                else{   // [//AAA/BBB]

                        $from=$from.$cname."_edge as p1,";

                        if($pSteps[1]!='*'){
                                $where=$where." AND p1.name='$pSteps[1]'";
                        }

                        $where=$where." AND p$p.nodeID>e$i.nodeID AND p$p.nodeID < ALL (
                        SELECT min( e.nodeID )
                        FROM ".$cname."_edge AS e
                        WHERE e.nodeID > e$i.nodeID AND e.parentID < e$i.nodeID
                        UNION SELECT max FROM maxnum) ";

                $p=2;
        }  // IF

        for(;$p<count($pSteps);$p++){

                if($pSteps[$p]==''){
                        $p++;
                        $q = $p-2;

                        $where=$where." AND p$p.nodeID>p$q.nodeID AND p$p.nodeID < ALL (
                        SELECT min( e.nodeID )
                        FROM ".$cname."_edge AS e
                        WHERE e.nodeID > p$q.nodeID AND e.parentID < p$q.nodeID
                        UNION SELECT max FROM maxnum) ";

                        $from=$from.$cname."_edge as p$p,";
                        if($pSteps[$p]!='*'){
                                $where=$where." AND p$p.name='$pSteps[$p]' AND
p$p.flg!='att'";
                        }
                }
```

```
                else{
                        if(!(strpos($pSteps[$p],'['))){
                                        $from=$from.$cname."_edge as p$p,";
                                        $q=$p-1;
                                        if($pSteps[$p]=='*') $where=$where." AND
p$p.parentID=p$q.nodeID AND p$p.flg!='att'";
                                        else $where=$where." AND p$p.name='$pSteps[$p]' AND
p$p.parentID=p$q.nodeID AND p$p.flg!='att'";
                                }
                }
        }       //  FOR


}
function splitSteps($separator,$term){
       $resultArr   = array();
    $bracketCounter = 0;
    do {
      $sepLeng =  strlen($separator);
      if (strpos($term, $separator)===FALSE) {
        $resultArr[] = $term;
        break;
      }
     $substituteSep = str_repeat(chr(2), $sepLeng);

      $tmp1 = strpos($term, '(');
      $tmp2 = strpos($term, '[');
      if ($tmp1===FALSE) {
        $startAt = (int)$tmp2;
      } elseif ($tmp2===FALSE) {
        $startAt = (int)$tmp1;
      } else {
        $startAt = min($tmp1, $tmp2);
      }

      $preStr = substr($term, 0, $startAt);
      $preStr = str_replace($separator, $substituteSep, $preStr);
      $postStr = substr($term, $startAt);
      $strLeng = strlen($postStr);
      for ($i=0; $i < $strLeng; $i++) {
        $char = $postStr[$i];
        if ($char=='(' || $char=='[') {
          $bracketCounter++;
          continue;
        }
        elseif ($char==')' || $char==']') {
          $bracketCounter--;
        }
        if ($bracketCounter == 0) {
            if ((substr($postStr, $i, $sepLeng) == $separator)) {
            for ($j=0; $j<$sepLeng; $j++) {
              $postStr[$i+$j] = $substituteSep[$j];
            }
          }
        }
      }
      $resultArr = explode($substituteSep, $preStr . $postStr);
    } while (FALSE);
    return $resultArr;
}


function print_result($result){
        echo "<p>XPATH Results:";
        if(empty($result)){
        echo"<p>No result from path table</p>";
        return;
        }
         $table_row="";

              $table_header="<table border='1' cellpadding='6'><tr><td
colspan='10'></td>";

                      $table_header.="</tr>"  ;

                while($myrow = mysql_fetch_array($result))
                      {
```

```
                        $table_row.="<tr>

                        <td align='right' valign='top' bgcolor='#DDDDDD'>$myrow[0]</td>
                        <td valign='top' bgcolor='#DDDDDD'>$myrow[1]</td>
                        <td align='right' valign='top' bgcolor='#DDDDDD'>$myrow[2]</td>
                        <td align='right' valign='top' bgcolor='#DDDDDD'>$myrow[3]</td>
                        <td align='right' valign='top' bgcolor='#DDDDDD'>$myrow[4]</td>
                        <td align='right' valign='top' bgcolor='#DDDDDD'>$myrow[5]</td>
                        </tr>";

            }

             $table_display=$table_header.$table_row."</table>" ;
                 echo $table_display;

}
function print_step($myrow){ // prints one node

        echo "printstep";
         $table_row="";

              $table_header="<table border='1' cellpadding='6'><tr><td
colspan='10'></td>";

                      $table_header.="Doc Nod EndDes Prnt Path   elname </tr>"  ;

                       $table_row.="<tr>

                       <td align='right' valign='top' bgcolor='#DDDDDD'>$myrow[0]</td>
                       <td valign='top' bgcolor='#DDDDDD'>$myrow[1]</td>
                       <td align='right' valign='top' bgcolor='#DDDDDD'>$myrow[2]</td>
                       <td align='right' valign='top' bgcolor='#DDDDDD'>$myrow[3]</td>
                       <td align='right' valign='top' bgcolor='#DDDDDD'>$myrow[4]</td>
                       <td align='right' valign='top' bgcolor='#DDDDDD'>$myrow[5]</td>
                       </tr>";

            $table_display=$table_header.$table_row."</table>" ;
                echo $table_display;

}
?>
```

REFERENCES

Bosak, J., The Plays of Shakespeare, 1999, http://www.ibiblio.org/bosak

Bourret, R., XML and Databases, 2002,
http://www.rpbourret.com/xml/XMLandDatabases.htm

Dayen, I., Storing XML in Relational Databases, 2001,
http://www.xml.com/pub/a/2001/06/20/databases.html

DB2 XML Extender, 2002,
http://www.4.ibm.com/software/data/db2/extenders/xmlext.html

Deutsch, A., Fernandez, M. and Suciu, D., "Storing Semistructured Data with STORED", In Proc. of ACM SIGMOD, Philadelphia, PN, 1999

Fan, W., and Libkin, L. "On XML integrity constraints in the presence of DTDs", In Proc. 20 Symp. on Principles of Database Systems, 2001, 114-125

Florescu, D., and Kossmann, D., "Storing and quering xml data using an RDBMS", IEEE Data Engineering Bulletin 22, 3, 27-34, 1999

Frank C., Exploring XML and Access 2002, 2001,
http://msdn.microsoft.com/library/en-us/dnacc2k2/html/odc_acxmllnk.asp

Goldfarb, C., F., Prescod, P., XML Handbook, Prentice Hall 4th Edition, 2001

Jiang, H., Lu, H., "Path Materialization Revisited:An Efficient Storage Model for XML Data",2nd Australian Institute of Computer Ethics Conference, Canberra, 2001

Miloslav Nic, Jiri Jirat "XPath Tutorial", 2000,
http://www.zvon.org/xxl/XPathTutorial/General/examples.html

Nigel Swinson, Sam Blum, Daniel Allen, "A php class for searching an XML document using XPath", 2001, http://sourceforge.net/projects/phpxpath/

Oracle XML DB, 2002, http://otn.oracle.com/tech/xml/xmldb/pdf/xmldb92tfovw.pdf

Shanmugasundaram, J., Zhang, C., Tufte, K., He, G., DeWitt and D., Naughton, J., "Relational Databases for Querying XML Documents: Limitations and Opportunities.", Proceeding of the 25th VLDB Conference, Edinburgh, Scotland, 1999

Shanmugasundaram, J., Tatarinov, I., Shekita, E., Kiernan, J., Viglas, E. and Naughton, J., "A General Technique for Querying XML Documents using a Relational Database System.", SIGMOD, 2001

Schmidt, A., Kersten, M. L., Windhouwer, M., and Wass, F. "Efficient relational storage and retrieval of XML documents.", In WebDB (Informal Proceedings), pages 47-52, 2000

Su, Z., 'An Implementation of Storage and Retrieval Methods for XML Documents in RDBMSs' , 2003

Tatarinov, I., And Iglas, S.,D.,V., "Storing and Quering Ordered XML Using a Relational Database System", ACM SIGMOD,Wisconsin, USA, 2002

XBench, 2003, A Family of Benchmarks for XML DBMSs, http://db.uwaterloo.ca/~ddbms/projects/xbench/

XML Path Language(XPath) 2.0, November 2003, http://www.w3.org/TR/xpath20

Yoshikawa, M. And Amagasa, T., "XRel: A Path -based approach to storage and retrieval of XML documents using relational databases", ACM Transaction on Internet Technology, 2001.