International Journal of Fuzzy Logic and Intelligent Systems, vol. 7, no. 4, December 2007 pp. 274-278

On supporting full-text retrievals in XML query

Dong-Kweon Hong

Dept. of Computer Eng., Keimyung University, Daegu, Korea

Abstract

As XML becomes the standard of digital data exchange format we need to manage a lot of XML data effectively. Unlike tables in relational model XML documents are not structural. That makes it difficult to store XML documents as tables in relational model. To solve these problems there have been significant researches in relational database systems. There are two kinds of approaches: 1) One way is to decompose XML documents so that elements of XML match fields of relational tables. 2) The other one stores a whole XML document as a field of relational table. In this paper we adopted the second approach to store XML documents because sometimes it is not easy for us to decompose XML documents and in some cases their element order in documents are very meaningful. We suggest an efficient table schema to store only inverted index as tables to retrieve required data from XML data fields of relational tables and shows SQL translations that correspond to XML full-text retrievals. The functionalities of XML retrieval are based on the W3C XQuery which includes full-text retrievals. In this paper we show the superiority of our method by comparing the performances in terms of a response time and a space to store inverted index. Experiments show our approach uses less space and shows faster response times.

Key Words : XML, XQuery, full-text retrievals

1. Introduction

XML(Extensible Markup language) became a worldwide standard format for digital information on the internet[1]. Traditional database systems face many new difficulties when dealing with XML. During the last several decades relational DBMS have been improved continuously in technical aspects so that it can handle queries on structural relational tables very efficiently[2]. However a query language on XML, XQuery, is much more diverse, complex and powerful than the standard query language on relational DBMS. It even includes full-text retrieval functionality with some structural queries[1][4]. In Jan of 2007 W3C (Worldwide Web Consortium) has published the final recommendation of XQuery full-text which is the essential part of queries on XML[1]. Due to its semi-structured natures of XML documents full-text retrieval capability has received much attention from the XQuery standardization groups. A full-text retrieval for plain text has been studied long mainly from Information Retrieval(IR)[3]. However a full-text retrieval for XML is a new research area that requires the research results of Information Retrieval and database research and needs to consider the element nesting of XML

documents[4][5][6]. Main research directions on full-text retrieval for XML in relational environment are using existing relational model[6][7] and creating new XML data model. In this paper we are focusing on using existing relational model that has been applied widely to many areas.

Manuscript received Aug. 30, 2007; revised Dec. 10, 2007.
This work was supported by the research grant of the Keimyung University in 2007.

Using inverted index is the most popular approach for full-text search in Information Retrieval[2]. It extracts meaningful keyword and phrases from documents and stores them in data structure like B-tree to get the document and the location of a searching keyword. In traditional Information Retrieval user supply a keyword to look for information that are related to searching keyword. However submitting only a keyword sets of keyword to look for information from XML documents is not a proper approach. Keyword itself is not enough to utilize the advantages of XML characteristic. If a search keyword is accompanied with a partial or full path we can search the required information more quickly and precisely.

In order to utilize all the advantages of XML most of full-text queries will use path information with searching keyword. Even the full-text search requires only some part of XML documents when it returns it as a result. Sometimes the results are combined with some other parts of XML documents. These requirements cause us to store much more information than that of a plain text when we deal with XML full-text retrievals[[7][8].

The paper is organized as follows: section 2 gives surveys of several approaches supporting full-text retrieval of XML documents in relational databases. Section 3 describes requirements of full-text retrieval in XML documents based on W3C XQuery documents. By closely looking at XQuery full-text use case document we identify more advanced full-text retrieval functions that are specific to XML and suggest table schema to support XML queries. Section 4 analyzes the table schema that we suggested in section 3 and shows the performance comparisons with previous approaches. Section 5 mentions conclusions and directions for future research.

2. Related Works

One of the most widely used method to support full-text retrieval in Information Retrieval is inverted index approach[7]. It extracts the required information such as the location of a keyword or count of a keyword occurrences and builds indexes such as B-tree or hashing by using the extracted information. A full-text search uses the indexes to look for the location or occurrence count of a search keyword. There are several ways to support full-text retrievals for XML documents. In [Fig. 1] we can see several approaches that implementing full-text retrievals.

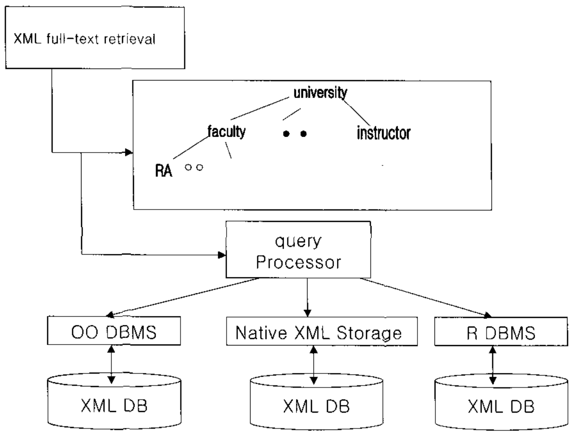


Fig. 1. XML full-text query processing in different DBMS

In the paper of Florescu at el [4] they introduced the way to support full-text retrieval using relational databases. To the best of our knowledge the paper is the first one that seriously studied a method to store XML inverted index in relational tables and introduced many open problems. In their approach they built tables for each pair of keyword and element to support keyword search which is accompanied with the name or paths of elements. For example if we are looking for XML documents that have a keyword "XSL" in <abstract> elements we just need to look through the table named "XSL_abstract" in their approach. This will reduce the search space dramatically and will increase the preciseness of the results a lot. In order to support element traversals for containment queries for XML documents they suggested the concept of binary table. To check the containment relationship for parent, child, and sibling of a element they heavily use join operations among binary tables.

In the research of a containment query [5] they identified the popularity of containment query in XML query and suggested an efficient method to implement it by using the following relational schema.

ELEMENTS(term, docno, begin, end, level)
TEXTS(term, docno, wordno, level)

Here the value of *begin, end* and *wordno* are generated by using a labeling scheme which use the word number of a document. They store all keywords into TEXTS table and all elements into ELEMENTS table. By comparing the value of "wordno", "begin", "end" and "level" they can check the *containment relationship*. Here again join operations are used a lot to check the relationships. There has been many other researches to store and query XML document using relational model [9][10][11]. However their research didn't consider XML full-text queries.

3. XML full-text retrievals

In this section we suggest a method to support XML full-text retrievals using relational database systems.

3.1 Requirements of full-text retrievals

W3C XQuery full-text requirement documents and use_case documents describe many full-text search functions on XML documents[1]. After closely analyzing the documents we identified several key full-text functions and selected several operations as representatives of the requirements.

Table 1. representative operations for XML full-text queries

No.	representative operations	functionalities	results
1	contains(text(), 'XML')	keyword search	docid, elementid
2	TITLE[contains(text(), "ACT")]		docid, elementid
3	//SCENE/*/LINE[contains (text(), 'love')]		docid, elementid
4	/PLAY/TITLE/'The comedy of Errors'	phrase search	docid, elementid
5	//SPEAKER='ACT'		docid, elementid

As we can see in [Table 1] information search in XML usually use a keyword or a phrase with a path in general. Also after we find the required information we have to return *docid* and *elementid* of the results. While processing these kinds search we need to check containment relationships, parent and sibling relationships among elements very often. From now on we only consider these operations for our experimentations.

3.2 Relational schema for inverted index

We store XML documents in the field of the following relational table. In each schema the underlined word(s) represent the primary key for each table.

XML_DOCUMENTS(ID, docname, isidx, XML_contents_id)

For each XML document we extract index informations and assign a sequences of number for each word. Relational table schema for our inverted index are as follows.

```
<Index table schema>
// save keywords
WORDINDEX(TERM, DOCID, POSITION)
/* save element information */
EINDEX(TERM, DOCID, STARTPOS, ENDPOS,
PATHID, DEPTH)
/* save attribute information */
AINDEX(TERM, DOCID, PATHID, VALUE)
/* save value text information */
TINDEX(DOCID, STARTPOS, ENDPOS, PATHID,
DEPTH, VALUE)
/* save path information */
PINDEX(DOCID, PATHID, PATH)
```

```
SAMPLE DATA
<dblp>
        <mastersthesis               mdate="2003-08-10"
key="ms/Brown03">
        <author> Hong Gil Dong</author>
        <title>The   Full   Text   search   using
RDBMS</title>
        <year>2003</year>
        <school>Seoul University</school>
        </mastersthesis>
</dblp>
```

With the previous table schema and the sample XML document we can build the graph representation and table instances as in the following.

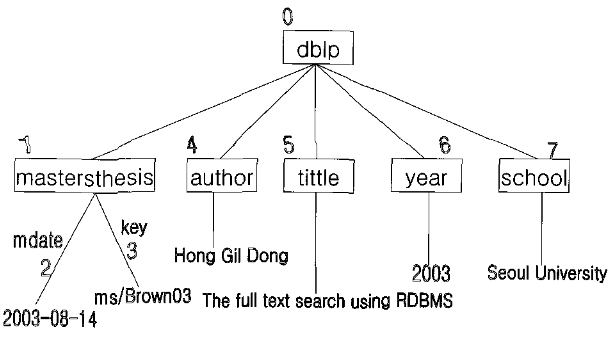


Fig. 2. Graph representation of sample XML document with pathid

In the following tables *POSITION, STARTPOS,* and *ENDPOS* values are assigned with the sequence numbers of word in a documents. For example the word 'full' appears as the 13th word in the document. In the AINDEX table the record ('mdate', 1, 3, '2003-03-10') means *mdate* attribute appears in the document that has the value of *DOCID* 1. And its location in the document is on the path of *PATHID* 3. The record ('author', 1, 7, 10, 4, 1) in the EINDEX says that element *author* starts in the 7th word and finishes 22nd word. And it is on the path of *PATHID* 4 and its distance from the root is 1.

AINDEX

TERM	DOCID	PATHID	VALUE
...	...	...	...
mdate	1	2	2003-08-10
key	1	3	ms/Brown03
...	...	...	...

PINDEX

DOCID	PATHID	PATH
...	...	...
1	0	#/dblp
1	1	#/dblp#/mastersthesis
1	2	#/dblp#/mastersthesis#/@mdate
1	3	#/dblp#/mastersthesis#/@key
1	4	#/dblp#/author
1	5	#/dblp#/title
1	6	#/dblp#/year
1	7	#/dblp#/school
...	...	...

WORDINDEX

TERM	DOCID	POSITION
...	...	...
Hong	1	8
Gil	1	9
Dong	1	10
full	1	13
text	1	14
search	1	15
...	...	...

EINDEX

TERM	DOCID	STARTPOS	ENDPOS	PATHID	DEPTH
...	...	...	...	...	...
dblp	1	1	22	0	0
mastersthesis	1	2	22	1	1
author	1	7	10	4	1
...	...	...	...	...	...

TINDEX

DOCID	STARTPOS	ENDPOS	PATHID	DEPTH	VALUE
...	...	...	...	...	...
1	8	10	4	2	Hong Gil Dong
1	12	17	5	2	The Full Text search using RDBMS
....	...	...	...	...	...

3.3 SQL transformation of full-text retrieval

We can transform the full-text retrievals of Table 1 into the SQL queries by using the relational schema of section 3.2.

```
Query 1: /* contains(text(), 'XML') */
CREATE OR REPLACE VIEW KY_XML AS
          SELECT DOCID, POSITION
          FROM WORDINDEX
          WHERE TERM LIKE '%XML%';
SELECT DISTINCT E.DOCID, E.TERM, E.STARTPOS
FROM EINDEX E, KY_XML KX
WHERE E.DOCID=KX.DOCID AND
            KX.POSITION BETWEEN  E.STARTPOS AND
E.ENDPOS;

Query 2: /* //TITLE[contains(text(), 'ACT'] */
SELECT E.DOCID, E.TERM, E.STARTPOS
FROM EINDEX E, PINDEX P, TINDEX T
WHERE E.DOCID=P.DOCID AND T.DOCID=E.DOCID AND
           P.PATH LIKE '#%/TITLE' E.PATHID=P.PATHID
AND
            P.PATHID=T.PATHID  AND  T.STARTPOS >=
E.STARTPOS AND
           E.ENDPOS >= T.ENDPOS AND T.VALUE LIKE
'%ACT%';

Query 3: /* //SCENE/*/LINE[contains(text(), 'love')] */
SELECT E.DOCID, E.TERM, E.STARTPOS
FROM EINDEX E, PINDEX P, TINDEX T
WHERE E.DOCID=P.DOCID AND P.DOCID=T.DOCID AND
           P.PATH LIKE #%/SCENE#/%#/LINE' AND
                      E.PATHID=P.PATHID          AND
P.PATHID=T.PATHID AND
           T.STARTPOS >= E.STARTPOS AND E.ENDPOS
>= T.ENDPOS AND
           T.VALUE LIKE '%love%';
```

4. Analysis and performance evaluations

We compared our approach of section 3 with the one suggested in the paper [4]. For the naming convenience we named our approach as FTS (full-text with string based approach) and the one in the paper of Florescu [4] as FTJ (full-text with join based approach). FTJ uses a table for each pair of key word and an element. We found in our preliminary experiments the number of table with FTJ might be over several tens of thousands. This is very unrealistic in real situations. Instead we use the following schema that is slight modification of the one used in the paper of Florescu [4].

```
<Table schema of FTJ>
ELEMENTS(ELENAME, DOCID, ELID, START, END)
WORD_TYPE(WORDTYPE,  DOCID,  ELID,  DEPTH,
LOCATION)
BINARY(DOCID, TARGET, SOURCE, VALUE)
```

With the schema above we can transform the XML queries of Table 1 into the following SQL queries in FTJ.

```
Query 1:
SELECT ELENAME, DOCID, ELID
FROM ELEMENTS
WHERE (DOCID, ELID)
          IN
(SELECT DOCID, ELID
          FROM WORD_TYPE
          WHERE WORD_TYPE LIKE '%XML%');
Query 2:
SELECT DISTINCT ELENAME, DOCID, ELID
FROM ELEMENTS
WHERE (DOCID, ELID)
          IN (SELECT DISTINCT DOCID, TARGET
                   FROM TITLE
                   WHERE VALUE LIKE '%ACT%');

Query 3: Not available.
```

When we compare the performance of FTS and FTJ we have used the data that can be found in http://www.oasis-open.org. The nature of the data is explained in Table 2.

Table 2. The characteristics of sample XML data

sample data name	No. of doc	Size	Avg. Depth
Shakespeare's work	37	7.53M	3.77

4.1 Spaces to store inverted indexes

Both approaches maintain several tables to support full-text queries. The size of spaces to maintain tables have been measured as in Table 3. Total space for FTS requires just one fourth of FTJ. When we closely speculate the data in both table schema FTS stores much less data by storing path information as strings.

Table 3. Table spaces for FTS and FTJ

FTS		FTJ	
TABLE NAME	SIZE	TABLE NAME	SIZE
EINDEX	3.719	ELEMENTS	4.700
AINDEX	0.000	WORD_TYPE	91.432
PINDEX	6.975	BINARY	8.271
WORDINDEX	6.851		
TINDEX	7.671		
TOTAL	25.216	TOTAL	104.402

4.2 Response times for keyword searches

In the same computing environment we measured the response times of both approaches for queries Q1, Q2, and Q3 as in Table 9. We haven't compared Q4, Q5, and Q6 because the schema for FTJ have been developed for keyword search and they didn't mention phrase search. Even though the transformed SQL queries for FTS look more complicated than those of FTJ we can see that FTS performs much better than FTJ in [Table 4].

4.3 Phrase search functionality of FTS

Table 4. Response times (sec.) of FTS and FTJ for full-text queries

Query	FTS	FTJ	RESULTS
Q1	0.901	8.662	docid, element id, element name
Q2	0.40	4.286	SPEECH docid, element id
Q3	0.40	X	LINE docid, element id

The schema that we suggested in section 3 is for keyword and phrase search together. We can also transform the phrase search queries in Table 1 into SQL queries with FTS. Unlike the keyword search we use 'TINDEX' for phrase searches. For example query 4 in Table 1 can be transformed into SQL query as follows.

```
Query 4:
/* /PLAY/TITLE/'The comedy of Errors' */
SELECT E.DOCID, E.TERM, E.STARTPOS
FROM EINDEX E, PINDEX O, TINDEX T
WHERE E.DOCID=P.DOCID AND T.DOCID=E.DOCID AND
        P.PATH='#/PLAY#TITLE'          AND
E.PATHID=P.PATHID AND
        P.PATHID=T.PATHID AND E.PATHID=(T.DEPTH
- 1) AND
        T.STARTPOS >= E.STARTPOS AND E.ENDPOS
>= T.ENDPOS AND
        T.VALUE='The Comedy of Errors';
```

References

[1.] http://www.w3c.org

[2] Hector Garcia-Molina, J. Ullman, J. Widom.: *Database Systems: The Complete book.* Prentice Hall, 2002.

[3] Ricardo Baeza-Yates, Berthier Riberiro-Neto,: *Modern Information Retrieval*, Addison Wesley, 1999.

[4] D. Florescu, D. Kossmann, and I. Manolescu, "Integrating keyword search into XML query processing", *WWW9/Computer Networks,* pp.119-135, 2000.

[5] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, G. Lohman."On supporting Containment Queries in Relational Database Management Systems". *ACM SIGMOD,* May Santa Barbara, CA, 2001.

[6] Jan-Macro Bremer, Michael Gertz."XQuery/IR: Integrating XML Document and Data Retrieval", *Proceedings of the 5th International Workshop on the Web and Databases,* pp 1-6, 2002.

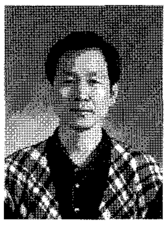
[7] N.Fuhr, K.Grossjohann. "XIRQL: An extension of XQL for information Retrieval", *Proceedings of SIGIR,* 2001.

[8] J. Shanmugasundaram, J. Kienan, E. Shekita, C. Fan, John Funderburk, "Querying XML views of Relational Data", *Proceedings of the 27th VLDB Conference*, 2001.

[9] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B.Lindsay, H. Pirahesh, B. Reinwald, "Efficiently Publishing Relational Data as XML Documents", *Proceedings of the 26th VLDB conference*, 2000.

[10] L. Fegaras, R. Elmasri, "Query Engines for Web-Accessible XML Data", *Proceedings of the 27th VLDB Conference,* 2001.

[11] I. Tatarinov, S. Viglas, K.Bayer, J. Shanmugasundaram, E. Shekita, C. Zhang, "Storing and Querying Ordered XML Using a Relational Database System", *Proceeding of ACM SIGMOD*, 2002.



Dong_Kweon Hong
He received MS. and Ph.D degree from CIS department of University of Florida, Gainesville, USA in 1992 and 1995. From 1997 to present, he is a professor in Computer Engineering department of Keimyung University. His research interests are XML, databases and web technologies.

Phone : 053-580-5281
Fax : 053-580-5165
E-mail : dkhong@kmu.ac.kr