Pattern-based Extensible Index Technique for XML documents

Jungsuk Song[†], Tackgon Kim*, Woosaeng Kim*
[†]R&D Institute, Hanwha S&C — *Dept. of Computer Science, Kwangwoon University
Janggyo-dong, Joong-gu, Seoul, Korea — Wolkye-dong, Nowon-gu, Seoul, Korea

Abstract: - Recently, a lot of index techniques for storing and querying XML document have been studied so far and many researches of them used coordinate-based methods. But update operation and query processing to express structural relations among elements, attributes and texts make a large burden. In this paper, we propose an efficient extensible index technique based on pattern information. It supports containment queries and pattern queries and it does not cause serious performance degradations even if there are frequent update operations. Management of XML Schema's pattern information can reduce the number of nodes participating in the containment relationship query processing among each element. Overall, the performance could be improved by reduction of the number of times for traversing nodes.

Key-Words: - Extensible Index, XML Indexing, XML Schema, Containment Query, Pattern Query

1 Introduction

The existing multimedia applications use for the limited area of the simple play for the image, audio and video. But multimedia applications have been expanded into services through the comprehensive uses of multimedia resources with increases of the explosive internet popularization and various demands of internet users these days. XML [1], which is derived from SGML [2] through recommendations of W3C (World Wide Web Consortium), comes up as a standard language, that can exchange the data through Internet and still can keep the interoperability, in recent years.

There are many applications with abundant representation ability of XML language. It is the trend to use XML for expression of all the data in business application program. Therefore, there are steady efforts for the study to store and retrieve the XML documents in the database efficiently and it needs to provide any methods to preserve not only the structural contents also positional information in the XML documents.

In this paper, we propose an extensible index technique that expresses structural pattern information based on XML schema structure using relational database. We will show pattern-based method that can efficiently perform storing and querying XML document through the proposed method.

The pattern-based method uses pattern information that is based on the appearance of child nodes per each parent node. It analyzes and indices the type of child nodes based on XML schema that each node contains, and then it can express structural pattern information of XML documents. Through this index, each node of XML tree can be expressed as pattern information. And it can be performed better than coordinate-based index method because it can easily process the relation information between elements in query processing.

This paper is organized as follows. In chapter 2, we review related works about index techniques of XML documents. In chapter 3, we introduce an efficient extensible index technique based on pattern information. In chapter 4, we present operations for extensible index technique, and in chapter 5, we explain query examples for containment queries and pattern query. In chapter 6, we show the performance evaluation between the proposed technique and existing coordinate-based technique. Finally, we make a conclusion in chapter 7.

2 Related Works

In position-based indexing, queries are processed by manipulating the range of offsets of words, elements or attributes. GCL position-based model was proposed in [3]. GCL is based on a data structure called a concordance list, which consists of text intervals called extents. Each extent is described by a start position and length. It is possible to process the query to express containment relation, thanks to the description for the range of a position.

In path-based indexing, the location of words is expressed as structural elements and the paths in tree

* The present research has been conducted by the Research Grant of Kwangwoon University in 2005.

structures are used for the processing of query. In order to determine the position of a word within a document, it is necessary to construct an encoding of the path of the element names from the root of the document to the leaf node containing the word. And then, for each word occurrence, the inverted list includes a representation of the path to that word.

In [4], data structure for indexing XML documents based on relative region coordinates is used. Region coordinates describe the location of content data in XML documents. They refer to start and end points of text sequences in XML documents. Region coordinates are adjusted by offsets relative to the corresponding region coordinate of the parent node in the index structure.

In [5], new technique based on bitmap indexing was introduced. XML documents are represented and indexed using a bitmap indexing technique. They define the similarity and popularity of the available operations in bitmap indices and propose a method for partitioning a XML document set. 2-dimensional bitmap index is extended to a 3-dimensional bitmap index, called BitCube. They define statistical measurements and correlation coefficient. BitCube proved eminent performance in performance assessment with systems such as existent XQEngine, XYZFind already through the fast search speed.

[6] uses numbering method that made grasp hierarchical relationship between composition elements using the number given suitable number to each composition element on XML document.

[7] explains about index graph that changes structure to find a fast route that is used often using data mining method. It has a problem that it must update the index graph every time the query processes fundamentally inaccurate index query.

3 Pattern-based Extensible Index

In this section, we introduce our proposed index model, and how to manage pattern information for index.

3.1 Table Schemas for Index

We use the table schema in figure 1 in order to store positional and pattern information of nodes.

Schemas for storing positional information consist of six tables: Element, Attribute, Path, and Text, Pattern, PatternIndex.

The Element table indicates information about element nodes with docID, pathID, and parentAddr which is positional information, and header, offset.

Element (docID, pathID, parentAddr, header, offset)
Attribute (docID, pathID, attrName, attrValue)
Text (docID, pathID, textValue)
Path (pathID, pathExp)
Pattern (patternID, curNode, childNodes)
PatternIndex (patternID, docID, pathID)

Fig. 1 Schemas for extensible index based on pattern

The Attribute and Text table stores each identification and positional information about attribute and text contents of elements.

The Path table keeps information about simple path. Fields are pathID that is path identifier and pathExp that is simple path of each node's.

The Pattern Table contains pattern information for child nodes that have each node and relevant node. Field that compose table is consisted of patternID for pattern identifier, curNode for describing pattern information to child node for one node, childNodes field etc.

The PatternIndex table stores occurrence availability of pattern using structure of inverted index about each XML document, and is consisted of docID, pathID field to express patternID and actual position in XML document that is pattern identifier field that composes table expression pattern of each node.

3.2 Description of extensible index technique

To define extensible index technique, we consider an XML document D rooted at r. Let's denote P as a parent node of the tree and C_1, C_2, …, C_n are child nodes of P node.

Definition 1. (Extensible index) Extensible index describes an address of node in an XML document. Address of the child node among sibling nodes C_1, C_2, …, C_n of an XML document are combinations of numbers (A_1, A_2), where A_1 is address of parent node, A_2 is offset for child node. C_1 and C_2 are expressed as bit string expression. Consecutive bit string of A_1 and A_2 means extensible index for a child node.

$$AddressOfChild=AddressOfParent+OffsetOfChild \qquad (1)$$

Definition 2. (Offset) Offset among child nodes are given to an order with a binary bit form according to a sequence order like A = {000, 001, 010, 011, 100, …, n-1} as described in figure 2.

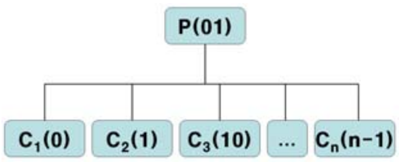


Fig. 2 An example tree and offset list

Definition 3. (Header) In order to store the number of bit string for processing offset of child nodes, initial value of directory header is assigned. Formula for determining initial value of header is as follows:

$$Header=ceiling\ (log_2 NumberOfChildNode) \qquad (2)$$

3.3 Description of pattern index technique

Suppose as following to define pattern index techniques based on XML schema.

XML document is known as D and root node is known as r. And P means node of tree, and C_1, C_2, …, C_n are child nodes of each P. And set of child nodes of P node is expressed into the brackets such as '[child]'.

Definition 4. (Pattern index) Pattern Index expresses pattern of node structure in XML document. Define pattern for number of cases that child node of each P node can be calculated through the information of XML schema and build the index. And expression of pattern can be denoted with pair of P nodes and child nodes. Following is the notation of pattern index.

$$Pattern{:}\ P/[\ subset\{C_1, C_2, \ldots, C_n\ \}]$$

We use appearance information of possible child nodes about each node P to express pattern. So it can express structural information reflecting pattern information about each node in extension index, and can easily apply the technique to use relative pattern information because we only consider one node and its child nodes.

3.3.1 Extraction of pattern information

Element's appearance is decided by Order Indicators and these are defined as, Sequence, Choice, and Arbitrary type in XML schema.

Definition 5. (Sequence pattern) It defines the sequence of elements, when the sequence of element is defined in <xsd:Sequence> tags.

Figure 3 shows the sequence pattern in XML Schema and four patterns {NULL, 'A', 'B', 'A, B'} can be achieved.

```
<xsd:Sequence>
        <xsd:element name="A" type="xsd:string" />
        <xsd:element name="B" type="xsd:string" />
</xsd:Sequence>
```

Fig. 3 An Example of Sequence order identifier

Definition 6. (Choice pattern) In case of selecting one element among several elements, the number of element is defined in <xsd:Choice> tags.

Figure 4 shows choice pattern for XML schema. Result pattern {'A', 'B'} can be generated.

```
<xsd:Choice>
        <xsd:element name="A" type="xsd:string" />
        <xsd:element name="B" type="xsd:string" />
</xsd:Choice>
```

Fig. 4 An Example of Choice order identifier

Definition 7. (Arbitrary pattern) It means pattern that is described without any rules such as elements in the <xsd:complexType> tags.

Figure 5 shows arbitrary pattern in XML Schema. Five pattern of result set {NULL, 'A', 'B', 'A, B', 'B, A'} can be possible.

```
<xsd:ComplexType>
        <xsd:element name="A" type="xsd:string" />
        <xsd:element name="B" type="xsd:string" />
</xsd:ComplexType>
```

Fig. 5 An Example of Arbitrary order identifier

3.3.2 Saving the Patterns

After analyzing type of pattern through XML Schema, these contents are stored on Pattern table and processed such a way as figure 6.

```
1. Analyze the XML scheme
1.1 switch(Order Identifier)
1.1.1 case complexType :get types of pattern from execution equation 4
1.1.2 case Sequence : get types of pattern from execution equation 5
1.1.3 case Choice : get types of pattern from the number of elements
2. Save the pattern on Pattern Table
2.1 Number the patterns and save the patternID field on Pattern Table
2.2 Save the parent element to the curNode field on Pattern Table
2.3 Save the patterns to the childNodes field on Pattern Table
2.4 if curNode is a leaf node and pattern is nothing then childNodes
    field's value is 'NULL'
```

Fig. 6 Pseudo algorithm for saving the patterns

Example 1. Example figure 7 will be stored into relational table like figure 8. Result patterns are 'A/[NULL]', 'A/[B] ',A/[C], 'A/[B, C]', 'B/[NULL]'.

```
<xsd:element name="A" type="xsd:string">
<xsd:Sequence>
            <xsd:element name="A" type="xsd:string" />
            <xsd:element name="B" type="xsd:string" />
</xsd:Sequence>
</xsd:element>
```

Fig. 7 A sample of XML schema

patternID	curNode	childNodes
1	A	A, B
2	A	A
3	A	B
4	A	NULL
5	B	NULL

Fig. 8 A Result of Pattern table from figure 7

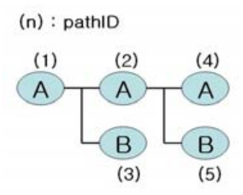
3.3.3 Building the PatternIndex Table

First, we get pattern information and store it on PatternIndex table using algorithm of figure 9. It constructs index for each element through XML document.

```
1. Analyze the XML tree
2. per each element
2.1 get pattern
2.2 match the pattern with the Pattern Table
2.3 get patternID from the Pattern Table
3. get the path expression
4. Save the PatternIndex Table
4.1 save the patternID to PatternID field on PatternIndex Table
4.2 save the docID field on PatternIndex Table from document ID
4.3 save the pathID to pathID field on PatternIndex Table
```

Fig. 9 Pseudo algorithm to build PatternIndex

Example 2. In algorithm of figure 6, we saved pattern information. We bring patternID value that corresponds to each pattern by analyzing pattern from Pattern table that each node has, and then stores on PatternIndex table. Next we bring pathID value from Path table about each path. figure 10(b) shows PatternIndex table with figure 10(a)'s XML tree.



pathID	docID	patternID
1	1	1
2	1	1
3	1	4
4	1	5
5	1	5

(a) **(b)**

Fig. 10 An example of (a) XML tree and (b) PatternIndex table

4 Operations for extensible index

4.1 Access of nodes

Access of nodes by extensible index is for search of XML document that is possessing relevant node ultimately. This paper examined method that can search various containment relations between nodes using simple query language through chapter 5.

4.2 Update of nodes

Because it is stored the field value as the value of node in table of relational database in extensible index technique, the existing value performs through process that replace existent value by new value simply at update. Figure 11 is an example of XML document to explain the process of update operation.

```
<document>
   <report>
      <author>Video database</author>
      <date>June 12, 2000</date>
   </report>
   <paper>
     <title>XML query data model</title>
      <author>Don Robie</author>
      <source>W3C, June 2000</source>
   </paper>
</document>
```

Fig. 11 An example of XML Document

Figure 12 shows the address information of three different index techniques. And figure 13 depicts the result address after update operation that lower left node changes from 'Video database' to 'Multimedia database' completed

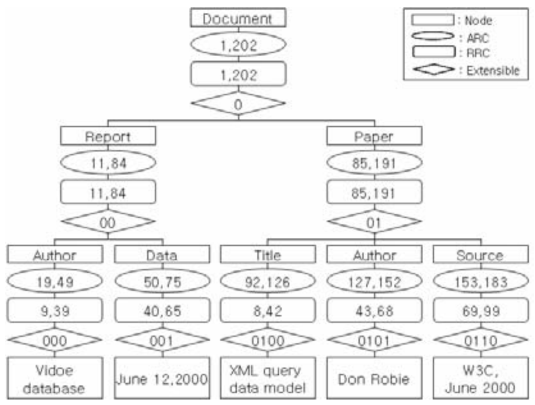


Fig. 12 Three index methods

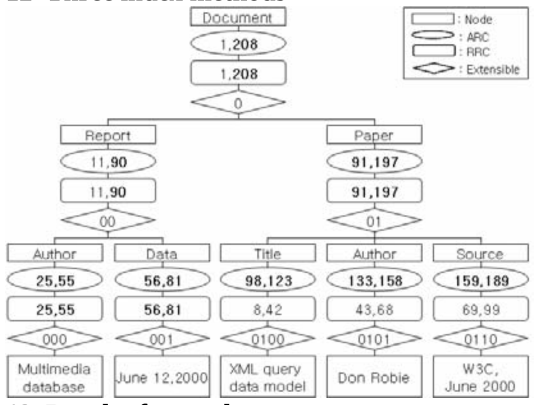


Fig. 13 Result after update

Bold faced values are changed values. All of the position information for absolute-based method changed. In case of relative–based method, position information for some nodes was not changed. However, for the case of the proposed extensible index techniques, there is no change of the position information by updating.

4.3 Insertion of a node

Figure 14 shows pseudo algorithm for changing index structure from the insertion of a node.

```
1. Calculate the location to insert a new node into the sibling nodes.
2. Compare and operate through the header and offset of previous node.
3. If (offset of previous node < $2^{\text{header of previous node}}$ -1)
3.1. NewOffset=offset of previous node + 1
3.2. NewHeader=header of previous node
4. If (offset of previous node  >= $2^{\text{header of previous node}}$ -1)
4.1. Header of new previous node = header of old previous node + 1
4.2. Offset of new previous node = offset of old previous node << 1
4.3. Header of new inserted node = header of old previous node + 1
4.4. Offset of new inserted node = (offset of old previous node << 1)+1
```

Fig. 14 Pseudo algorithm for insertion operation

Example 3. If C_4 node is inserted into the table in figure 15, it is to be inserted into the sufficient free space. New header value and new offset value are calculated according to the above pseudo algorithm.

Node	Address of P	Header	Offset
C_1	01	2	00
C_2	01	2	01
C_3	01	2	10
C_4	**01**	**2**	**11**

Fig. 15 Table after insertion of node C_4

Example 4. Unlike C_4, consecutive insertion of node C_5 after example 1 makes different result. In this situation, there is no sufficient free space for the new offset according to the old header, so inserted node should split with previous node. Figure 16 shows the table after insertion of node C_5.

Node	Address of P	Header	Offset
C_1	01	2	00
C_2	01	2	01
C_3	01	2	10
C_4	**01**	**3**	**110**
C_5	**01**	**3**	**111**

Fig. 16 Table after insertion of node C_5

5 Query Processing

In this section, we use containment relationship queries[8][9] and pattern query to evaluate extensible index based on pattern information technique such as direct containment query, indirect containment query, perfect containment query, pattern query.

5.1 Direct containment query

Definition 8. (Direct containment) Direct containment relationship query indicates the query that is consisted of direct containment relationship between elements, attributes, and texts. The symbol '/' indicates direct containment (i.e., parent-child relationship).

$$\textit{Parent (parentAddr+offset) =Child (parentAddr)} \qquad (3)$$

5.2 Indirect containment query

Definition 9. (Indirect containment) Indirect containment relationship query indicates the query that is consisted of indirect containment relationship between elements, attributes, and texts. The symbol '//' indicates indirect containment relation (i.e., ancestor-descendent relationship).

$$\textit{Ancestor(parentAddr+offset)} \subset \textit{Descendant(parentAddr)} \qquad (4)$$

5.3 Complete containment query

Definition 10. (Complete containment) Complete containment relationship query indicates the query that is consisted of complete containment relationship between elements, attributes, and texts. It can be executed to compare the text value of a node and the parameter in the query condition.

5.4 Pattern Query

Definition 11. (Pattern query) It means the query that search patterns having arbitrary node including specified child nodes, and it searches pattern of XML schema and can be used for searching the document including relevant pattern.

Pattern: P/ [C_1, C2, …, Cn]
Where 'P' = PatternIndex.curNode and
'C_1, C2, …, Cn' = PatternIndex.childNode

It also can query using wild card characters to support query that find structure including specified pattern. '*' expresses to find the case of the pattern is 0 or more and ‘+’ expresses to find that it is 1 or more.

And ‘?’ is the case of 0 or 1 and ‘^’ is the case of excluding the pattern.

Example 5. Figure 17 shows example of pattern query. And it shows query that find patterns having nodes of ' title', ' star' as the child nodes under of 'Movie' node.

```
Query : movie/[title, star]
SELECT   …
FROM     …
WHERE    Pattern.curNode = 'movie'
         and Pattern.childNodes = concat('title', 'star') and …
```

Fig. 17 Pattern query

Example 6. Figure 18 shows example of query that find patterns including specified pattern. It shows the query that finds the patterns that includes one or more other patterns and has the ‘title’ node under 'Movie' node as a child node.

```
Query : movie/[title+]
SELECT   …
FROM     …
WHERE    Pattern.curNode = 'movie'
         and Pattern.childNodes like '%title%' …
```

Fig. 18 Pattern query with filter character

6 Performance Evaluation

In this chapter, we show the comparison between conventional index technique and extensible index based on pattern technique. Proposed algorithm reduces the workload to perform the comparison and update operation when it retrieves and indicates data in the relational tables for these indices.

6.1 The number of updated nodes

In the XML tree, according to growing the depth and the width, the number of nodes is to be increased in a geometrical progression. In the situation that update of an offset data affects other node by the insertion of a node, in the worst case, all of the position data should be reconstructed and it is caused lots of cost for the operation.

Let XML tree is balanced binary tree and depth is from 1 to k, in case using coordinate-based index and extensible index, the accumulative number of the node of which offset data is changed from the insertion and update operations are as follows.

<u>**Insert operation on leaf nodes**</u>

Absolute coordinate = $2^{k-2} \times \left(\sum_{m=1}^{k} \prod_{j=1}^{m} 2 + k + 1 \right)$ (5)

Relative coordinate = $2^{k-2} \times (3k-1)$ (6)

Extensible index = 2^{k-1} (7)

<u>**Update operation on leaf nodes**</u>

Absolute coordinate = $2^{k-2} \times \left(\sum_{m=1}^{k} \prod_{j=1}^{m} 2 + k + 1 \right)$ (8)

Relative coordinate = $2^{k-2} \times (3k-1)$ (9)

Extensible index = Constant (10)

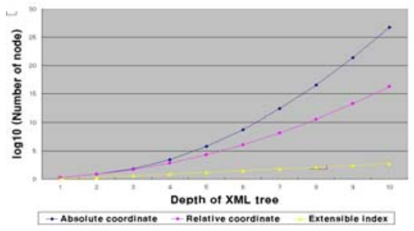


Fig. 19 Insert operation on leaf nodes

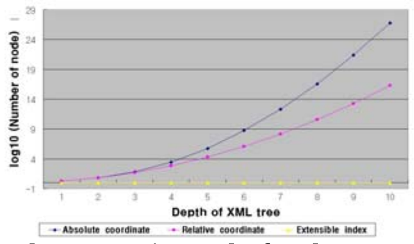


Fig. 20 Update operation on leaf nodes

6.2 Pattern-based extensible index technique

To have a simple comparison between conventional coordinate-based index technique and proposed technique, we choose a situation when the XML tree T is balanced.

Therefore, the number of child node is C^{D-1}, as the average number of child nodes is C and depth is D. And D is from 1, 2, …k (depth of tree).

We consider two queries that pattern query and direct containment query. Let’s denote RPq() the workload of searched nodes on pattern query, and RCq() the workload of searched nodes on containment query. Figure 21, figure 22, figure 23 and figure 24 shows the comparison result.

<u>**In case of Pattern Query**</u>

Proposition 1. $RPq_{\text{coordinate-based}} = \sum_{n=1}^{k-1} c^n * \frac{c}{2}$ (11)

Proposition 2. $RPq_{\text{extensible}} = \log_2 \sum_{n=1}^{k-1} c^n + 1$ (12)

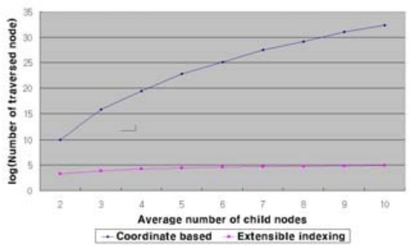


Fig. 21 Evaluation of pattern query according to increase the number of child nodes

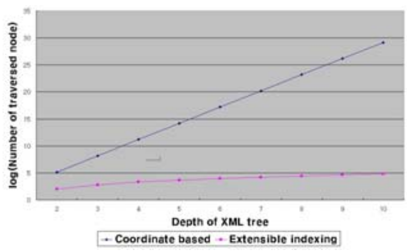


Fig. 22 Evaluation of pattern query according to increase the depth

<u>In case of Direct Containment Query</u>

Proposition 3. $RCq_{\text{coordinate-based}} = \sum_{n=1}^{k-1} c^n * \frac{c}{2} * (l-1)$ (13)

Proposition 4. $RCq_{\text{extensible}} = \log_2(\sum_{n=1}^{k-1} c^n) + 1\} * (2l-3)$ (14)

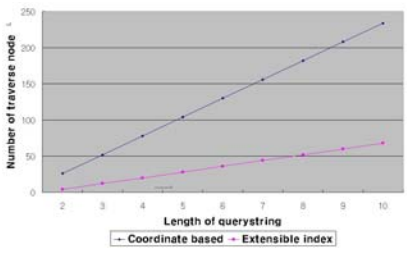


Fig. 23 Evaluation of direct containment query according to increase query length

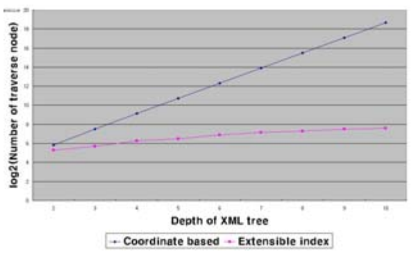


Fig. 24 Evaluation of direct containment query according to increase depth

7 Conclusion

We proposed an extensible index technique to express position information between nodes in a XML document based on XML Schema. It is an efficient index technique that simplifies the comparative object applied to a search query and minimizes the reconstruction of index structure by update operation.

In addition, proposed index technique keeps to support not only conventional direct containment query, indirect containment query, compete containment query, proximity query but also specific pattern query by simple SQL statement widely used in general.

Moreover, the technique can minimize reconfiguration process of index structure by update operation, and can simplify comparison target that is applied on search query. It improves performance better than conventional coordinated-based index techniques.

References:

[1] T. Bray, et al, “Extensible Markup Language (XML) 1.0 (Second Edition),” http://www.w3.org/TR/2000/REC-xml-20001006

[2] ISO, “Information Processing - Text and Office System - Standard Generalized Markup Language (SGML),” ISO/IEC 8879, Oct. 15, 1986.

[3] R. Davis, T. Dao, J. Thom, J. Zobel, "Indexing documents for queries on structure, content and attributes," DMIB'97, Nov. 1997.

[4] C. Clarke, G. Cormack, F. Burkowski, "An algebra for structured text search and a framework for its implementation," The Computer Journal, 1995.

[5] Jong P. Yoon, Vijay Reghavan, Venu Chakilan, “BitCube: a three-dimensional bitmap indexing for XML documents,” Proceedings, Thirteenth International Conference on Scientific ans Statistical Database Management (SDBM’2001), July 2001.

[6] D. Kha, M. Yoshikawa, S. Uemura, "An XML indexing structure with relative region coordinate," ICDE'2001, April 2001.

[7] C. Chung, J. Min, K. Shim, “APEX: An Adaptive Path Index for XML Data”, In Proceedings of the ACM SIGMOD International Conference on the Management of Data, 2002:121-132

[8] C. Cheng, J. Naughton, D. DeWitt, Q.Luo, and G. Lohman, "On supporting containment queries in relational database management systems," ACM SIGMOD, 2001.

[9] C. Seu, S. Lee, H. Kim, "An Efficient Inverted Index Technique based on RDBMS for XML Documents", KICS:Database Vol 30. No 1. 2003.2

[10] J. Song, W. Kim, "Extensible index technique for storing and retrieving XML documents," CIT’2004, pp. 280-287, Sep., 2004.