Mapping XML to Inverted Indexed Circular Linked Lists

Teng Lv[1], Ping Yan[2], and Weimin He[3]
[1]School of Information Engineering, Anhui Xinhua University, China
[2]School of Science, Anhui Agricultural University, China
[3]Department of Computing and New Media Technologies, University of Wisconsin-Stevens Point, USA

Abstract: *Extensible Markup Language (XML) has become the de facto standard for data exchange on the World Wide Web and is widely used in many fields, so it is urgent to develop some efficient methods to manage, store, and query XML data. Traditional methods use relational databases to store XML data which take advantage of mature technologies of relational databases. But it needs to map XML schemas to relational schemas, then rewrite XML queries to SQL queries, and finally, transform returned SQL-style results to XML-style results again. One possible solution to this is to store XML data directly and query it directly by XML query languages. In this paper, we research the problem of how to map XML data so that storing and querying it can be efficient. We propose the following framework to gain the goal: Firstly, we map a given XML data tree to a set of inverted indexed circular list, in which the relationships between parent and child nodes (and also ancestor and descendent nodes) are preserved. Then, an XML schema tree is used to guide and improve the efficiency of querying the corresponding XML data tree, which is generated from the given XML data tree. Finally, an efficient algorithm is given to query the XML data tree by using the corresponding set of inverted indexed circular list and its schema. The algorithms analysis and experiments prove the efficiency of our method over naïve method.*

Keywords: *XML, mapping, query, schema.*

Received August 12, 2014; accepted December 23, 2014

1. Introduction

Extensible Markup Language (XML) has become the de facto standard for data exchange on the World Wide Web and is widely used in many fields. With the increase of XML data on the World Wide Web, it is urgent to develop some efficient methods to manage, store, and query XML data. There are two main methods of storing and managing XML data today. The first method uses native XML databases to store and manage XML data directly, such as [15, 16]. The second method proposed in [5, 7, 14, 17] use relational databases to store XML data which takes advantage of the mature technology of relational databases. For the latter case, it needs to research some efficient methods to map XML schemas to relational schemas [12]. Furthermore, to query XML data, it is needed to transform XML queries to SQL queries, which is an additional cost of querying XML data and may decrease the querying efficiency. So it is significant to store XML data directly and query it directly by XML query languages such as XPath.

In this paper, we research the problem of how to map XML data so that storing and querying XML data can be efficient. We gain the goal by the following steps: Firstly, we map a given XML data tree to a set of inverted indexed circular list, in which the relationships between parent and child nodes (and also ancestor and descendent nodes) are preserved. Then, an XML schema tree is generated from the given XML data tree. The XML schema tree can be used to guide and improve the efficiency of querying the corresponding XML data tree. Finally, an efficient algorithm is given to query an XML data tree by using the corresponding set of inverted indexed circular list and its schema produced in the previous step.

2. Related Works

Our work is concentrated on mapping XML data efficiently in order to support efficient querying XML data directly. For XML labelling, [9] proposed a labelling scheme based on the concept of the complete tree whose space requirement of labelling scheme is superior to others in most cases. But we use e a simple label for each node to represent its path in the XML data tree and a set of inverted indexed circular list to represent the XML data tree. Our method can support upgrade easily as the nodes are linked by links in the circular lists. Also, many normalization forms [2, 10, 11, 18, 19] for XML are proposed based on different methods to remove data redundancies, but they cannot efficiently support querying XML documents as there are many joins when normalization are applied, especially in very small and sparse XML segments. So schema-based approach, such as [1, 6], uses schema to minimize tree patterns to improve query efficiency. Although these methods can simplify query patterns in

the sense that some irrelevant elements can be avoided to access in the query compilation, but they cannot avoid other irrelevant elements in the case that irrelevant data are stored continuously. [3, 4, 8, 13] storage XML based on various partition method. They can be used to improve XML queries efficiency because only relevant XML data need to be accessed when queries are simplified to specific paths. Our method in the paper use set of inverted indexed circular linked list to store XML data and set of indexed circular linked list for its schema, which can avoid both of these two types of irrelevant data in the process of querying XML data.

2.1. Main Contributions

In this paper, we study the problem of mapping and querying XML data, the main contributions are listed as follows:

1. A new mapping framework to map XML data tree to a set of inverted indexed circular linked list, which is efficient both in mapping time and storage space.
2. An efficient algorithm to directly query XML data by a set of inverted indexed circular linked lists with their schemas. The algorithm is a two stages process: the first stage is fully directed by the schemas inferred from the XML data tree, and the second stage is fully applied the inter-relationship between nodes in the inverted indexed circular linked list of the XML data tree.

2.2. Organization

The paper is organized as follows. Section 2 gives the algorithm to map XML data tree to a set of inverted indexed circular linked lists. Section 3 gives the algorithm to generate XML schema from a given XML data tree. Section 4 gives the algorithm of querying XML data tree by a set of inverted indexed circular linked lists with their schemas. Section 5 gives some experiments to verify our method. And finally, section 6 concludes the paper and points to the future direction of the work.

3. Mapping XML Data Tree to Inverted Indexed Circular Linked Lists

An XML document is depicted as a labelled XML tree such as Figure 1. we use uppercase letter to denote element type, and uppercase letter followed by a number to indicate a specific element node of this element type. For example, A, B, C, D, and E are all element types, and B1, B2, and B3 are all element nodes with element type B. Each node is labelled by its path. The root node is labelled as /1, and the other node labelled orderly as its occurrence under its parent node prefixed by its parent's label recursively. For example, node B1 is the first child node of root node A1, so its label is /1/1 as its parent node, i.e., root node A's label is /1. Another example, D1 is the third child node of B1, so its label is /1/1/3 as its parent node's label is /1/1.

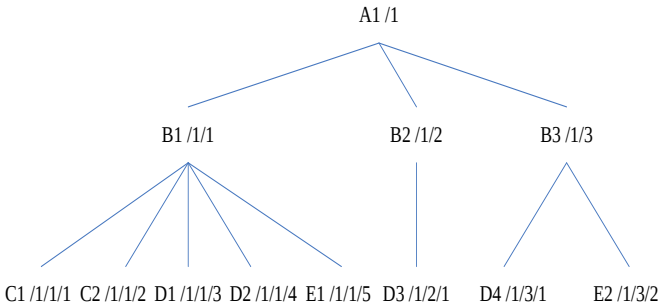


Figure 1. An XML tree.

For an XML tree such as Figure 1, we can store it as circular linked lists for efficiently accessed and queried. For each non-leaf node, we construct a circular linked list, in which each node has the following form: {ParentLink, NodeName, Path, ChildLink}, where ParentLink except for the head node points to its parent node, NodeName is the element type of the node, Path indicates the node's path from the root node to itself, ChildLink except for the tail node points to the next child node of the head node of the circular linked list. For tail node, i.e., the last child node, its ChildLink points to "NULL" indicating that there is no more child node of the head node. For example, root node A1 of Figure 1 has the following circular linked list as illustrate in Figure 2, which says that A1 has 3 child nodes, such as B1, B2, and B3, respectively, each child node points to its parent node by a pre-link.

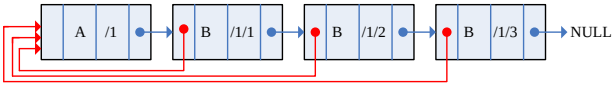


Figure 2. A circular linked list of root node A1 of Figure 1.

For leaf node, there is no necessary to construct a circular linked list as non-leaf nodes do, because leaf nodes are directly linked by their parent nodes so far. So their connectivity structure has been captured by the circular linked lists of their parent nodes.

To improve query and search efficiency, we can construct inverted index on each head node of circular linked list based on what child nodes contained in each head node. For example, Figure 3 is the inverted indexed circular linked lists for Figure 1, it means that containing child element type B is head node A(/1), containing child element type C is head node B(/1/1), containing child element type D are head nodes B(/1/1) and B(/1/2), etc.

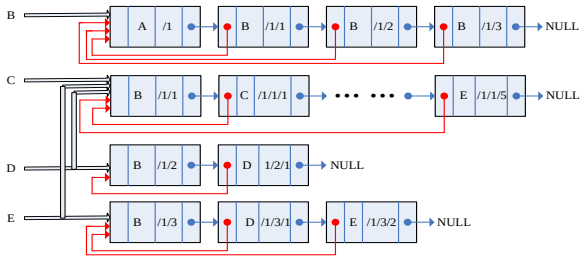


Figure 3. The inverted indexed circular linked lists for Figure 1.

An XML data tree can be mapped to inverted indexed circular linked lists by Algorithm 1.

Algorithm 1: Mapping XML data tree T to inverted indexed circular linked lists.

Traverse from the root node r of T by Breadth First Search (BFS), for each non-leaf node n, construct a circular linked list l:

1. *The head node H is {null, Element(n), Path(n), ChildLink(n)}, where n is the element type of the node, Element( ) is a function to get the element type of the node, Path( ) is a function to assign a path for each node as described before, and ChildLink(n) points to NULL and if the head node has child node, it will points to the new constructed node in the succeeding step* 2.
2. *Construct the next node N_i if there is a child node of the head node. Update ChildLink(n)* → *N_i.*
3. *Let N_i ={H, Element(N_i), Path(N_i), ChildLink(N_i)}, where ChildLink(N_i) points to NULL and if the head node has another child node, it will point to the new constructed node in step 3. Construct the next node N_i if there is a child node of the head node and update ChildLink(N_i)* → *N_i.*
4. *Go to step 3 and until there is no child node of node n.*
5. *Add the list l into the inverted indexed list according to its child element by function AddIndex(l).*
6. *Terminate until all the non-leaf nodes are processed.*

- *Complexity Analysis*: Suppose there are *n* nodes in the XML tree, and the fraction of the non-leaf nodes is *f* (where *f*<1 is a constant number), so the number of non-leaf node is *f*•*n*. And suppose each head node has *c* child nodes on average (where *c* is a very small constant number comparing to *n*), so Algorithm 1 generates *f*•*c*•*n* nodes in inverted indexed circular linked lists. Each node in the list can be processed in *O*(1) time, so the complexity of Algorithm 1 is *O*(*f*•*c*•*n*), i.e., *O*(*n*).
- *Improvement*: To improve the storage efficiency, for each inverted indexed circular linked list, only the head node stores the whole path (i.e., absolute path) in its item "Path", and the other node only stores the relative path in its item "Path", as we can construct the absolute path from its relative path plus its parent node path. For example, root node A1 of Figure 1 has the following circular linked list as illustrated in Figure 4.

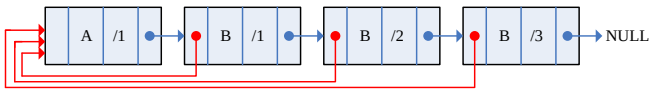


Figure 4. An improved circular linked list of root node A1 of Figure 1.

To obtain the absolute path of B(/1/1), we just use its parent node path plus its own relative path, i.e., B.Path = "/1" + "/1" = "/1/1".

4. Generating XML Schema From XML Data Tree

The XML schema tree of an XML tree is a labeled tree that summarizes all paths in XML documents. All nodes that have the same root-to-node path are mapped into a single node in the XML schema tree. So each distinctly labelled path appears exactly once in the corresponding XML schema tree. Furthermore, each node in XML schema tree is annotated by one of the following symbols which indicates the number of occurrences of this schema node under its parent in the XML documents:!: exactly once and only once;?: zero or once;*: zero or more times; and +: once or more times. Figure 5 is an XML schema tree for XML tree of Figure 1.

We give Algorithm 2 to generate an XML schema tree from an given XML tree.

Algorithm 2: Generating XML schema tree from an XML tree.

1. *Convert the root node r to Element(r).*
2. *Traverse from the root node r of T by BFS. For each node n, and for each child node c_i of n, generate Element(c_i) as the child of Element(n). Merge all child nodes if their Element(c_i) are equal by function Merge(Element(c_i)). Assign one of the symbols "!", "?", "*", and "+" according to their occurrences by function AssignSymbol (Element(c_i)).*
3. *Terminate if all the nodes are processed in the way of step 2.*

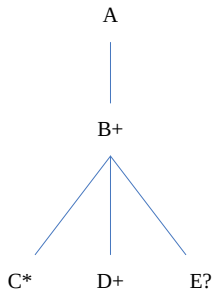


Figure 5. The XML schema tree of Figure 1.

- *Complexity Analysis*: Suppose there are n nodes in the XML tree. Each node in the tree can be processed in O(1) time by functions Element(), Merge(), and AssignSymbol(). So the complexity of Algorithm 2 is O(n).

For XML schema tree, we can also store it in inverted circular linked lists for efficiently accessed and queried. For each non-leaf node, we construct a circular linked list, in which each node has the following form: {ParentLink, ElementName, OccurrenceSymbol, ChildLink}, where ParentLink (except that of the head node) points to its parent element, ElementName is the element type name of the

element node's name in the XML schema tree, OccurrenceSymbol can be one of the symbols "!", "?", "*", and "+" indicating the possible number of occurrences of this schema node under its parent in the XML tree, ChildLink (except that of the tail node) points to the next child node of the head node of the circular linked list. For tail node, i.e., the last child node, its ChildLink points to "NULL" indicating that there is no more child node of the head node. For example, root node element A of Figure 5 has the following circular linked list which says that element A may have one ore more element B as its child node, and B's parent is A.

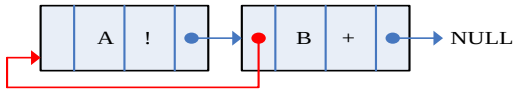


Figure 6. A circular linked list for element A in Figure 4.

The algorithm of how to map an XML schema tree to circular linked lists is similar to Algorithm 1, so we do not give the details here considering the space and clarity.

5. Querying XML Data Tree by Inverted Indexed Circular Linked Lists with Schema

In this section, we study how to query XML data tree represented in inverted indexed circular linked lists with its schema represented in circular linked lists. We consider the query in XPath form. The process can be finished by two stages.

1. Simplify the query against the schema tree by the following rules. As the schema provides some path information, a query may be simplified by the schema.
2. Using the simplified query to query the XML data tree represented in inverted indexed circular linked lists. In summary, the whole process is a two stages process: the first stage is a top-down stage to process the query by the schema tree and the second stage is a bottom-up stage to process the query by the XML data tree.

- *Simplifying Rules*: From the characteristics of path, we give some simplifying rules to simplify a given query to improve query efficiency:
  - *Rule 1*. For a part of path /A/B, if B is never adjacent to A, i.e., wherever there is an element A, there is no element B, then no result satisfies the query.
  - *Rule 2*. For a part of path /A//B, if B is not a descendent of A, i.e., wherever there is an element A, there is no descendent element B, then no result satisfies the query.
  - *Rule 3*. For a part of path /A/[./B], if B is always adjacent to A, i.e., wherever there is an element A, there is always an element B, then it can be simplified as /A.

We now give an algorithm to process a query given in XPath form by using the above process as illustrated in Algorithm 3.

Algorithm 3: Query an XML data tree.

Given an XPath query Q, the inverted indexed circular linked lists L of an XML data tree T, and circular linked lists of XML schema tree T_S.

- *(Top-Down Stage) Simplify the query Q as Q_S against the schema tree by the above rules. From the left of query Q, process each part of Q until the end of Q:*
  1. *If it is the form /A/B, traverse element A's circular linked list of schema T_{SA} to determine whether it is a legitimate path in T_S. If it is not, return "no result" by Rule 1.*
  2. *If it is the form /A//B, traverse from element A's circular linked list of schema T_{SA}, then A's child elements' circular linked list of schema one by one recursively to determine whether it is a legitimate path in T_S. If it is not, return "no result" by Rule 2.*
  3. *If it is the form /A[./B], traverse element A's circular linked list of schema T_{SA} to determine whether B is always adjacent to A. If it is, then /A[./B] is simplified as /A by Rule 3.*
- *(Bottom-Up Stage) Query Q_S against the XML data tree. From the right element of the simplified query Q_S obtained in Step 1, process each element of Q_S until the head of Q_S:*
  1. *For each element E, search the "E" index L(E) of the inverted indexed circular linked lists L and locate the satisfied nodes set N.*
  2. *For each node n in N, find its parent node n_p.*
  3. *If n_p has a condition path of the following form \Element(n_p)\[\.E_c], then determine whether "Element(n_p)" index L(Element(n_p)) of the inverted indexed circular linked list L has such a node. If it is not the case, process the next node n in N, go to step 2.*
  4. *Traverse n_p's parent node recursively until the left element of query Q_S and return the satisfied result nodes. Go to step i.*
- *Complexity Analysis*:

1. *Step 1*. Top-Down Stage: suppose there are n elements in query Q. In step 1, at most (n-1) parts of the query should be processed against circular linked lists of XML schema tree T_S in the worst case, and each part can be processed in O(1) time, so the overall time is O(n).
2. *Step 2*. Bottom-Up Stage: suppose there are m elements in query Q_S and the right element of query Q_S has c nodes to be processed, where c is related to the specific query and XML data tree. In step 2, at most cm nodes of the inverted indexed circular linked lists L should be processed in the worst case, and each node can be processed in O(1) time, so the overall time is O(cm). So the complexity of Algorithm 3 is O(n).

We give some examples to illustrate the queries of XML data tree of Figure 1 by Algorithm 3:

- *Example 1*. Consider the query Q1: /A/D.
  By searching the schema of circular linked lists (such as Figure 5), we known that there is no such path as /A/D, we confirm that no answers can satisfy such query in the given XML data tree by rule 1. So there is no need to search the XML data tree.
- *Example 2*. Consider the query Q2: /A/B/[./D]/E.
  1. From the schema of circular linked lists of Figure 5, we know that wherever there is an element B, there is always an element D, so the query Q2 can be simplified as /A/B/E by rule 3.
  2. From the inverted indexed circular linked lists (Figure 3), the last element of query Q2 is E, so from the index "E" of Figure 3, we know that E(/1/3/2) and E(/1/1/5) satisfy the query. From E(/1/3/2), we can get its parent B(/1/3). From B(/1/3) and index "B", we can get its parent A(/1). So we get a result "/1/3/2", i.e., E2 of Figure 1. Similarly, from E(/1/1/5), we can get another result "/1/1/5", i.e., E1 of Figure 1.
- *Example 3*. Consider the query Q3: /A/B/[./C]/E.
  1. From the schema of circular linked lists of Figure 6, query Q3 can not be simplified.
  2. From the inverted indexed circular linked lists (Figure 3), the last element of query Q3 is E, so from the index "E" of Figure 3, we know that E(/1/3/2) and E(/1/1/5) satisfy the query. From E(/1/3/2), we can get its parent B(/1/3). But B(/1/3) does not have a child element C from circular linked list of B(/1/3). So E(/1/3/2) does not satisfy query Q3. From E(/1/1/5), we can get its parent B(/1/1). As B(/1/1) has a child element C from circular linked list of B(/1/1), we continue search B(/1/1) by index "B", we can get its parent A(/1). So we get a result "/1/1/5", i.e., E1 of Figure 1.

6. Experimental Results

We implemented an XML data generator generate 10 XML data files. For each XML data file, a root element "A" is generated. Then, 1-4 elements are generated randomly for each element. For each new element, one of the 4 occurrence symbols "!", "?", "*" and "+" is assigned to it to indicate its occurrence. For element with "!" symbol, exactly one node with this type is generated as sub-element of its parent element. For element with "?" symbol, none or one node with this type is generated as sub-element of its parent element. For element with "*" symbol, 0-9 node(s) with this type are generated as sub-element of its parent element. For element with "+" symbol, 1-10 node(s) with this type are generated as sub-element of its parent element. From the root node, there are 6 levels in the final XML data file to be generated.

Element names are named as "#level" followed by a letter alphabetically, such as "1B", "1C", "2B", etc. Table 1 is the 10 selected XML data files generated by this method.

Table 1. 10 XML data files.

ID	#nodes (K)
#1	5
#2	6
#3	9
#4	12
#5	16
#6	18
#7	21
#8	23
#9	27
#10	30

We implemented Algorithms 1, 2, and 3 and the algorithm of mapping an XML schema tree to a set of circular linked list. To evaluate our method, we also implemented a naïve algorithm without any index information which just searches an XML data tree by BFS. All the experiments are performed on a computer with Windows Server 2008, Intel Core 2 CPU 2.5GHz, and 6G RAM.

For each XML data file, we design 3 types of queries: query type 1 cannot be simplified; query type 2 returns no results by rule 1 or rule 2; and query type 3 can be simplified by rule 3. The 3 queries of each type of XML data file #1 are shown in Table 2.

Table 2. 3 queries of each type of XML data file #1.

Query ID	Query Expression
Q11	/A/1B/3B/3C/4F/5B
Q12	/A//4H
Q13	/A/1B/[./2B]/2C/3F

In Table 2, Q11 is type 1 query which can not be simplified by our method, Q12 is type 2 query which returns no results just by examining the corresponding XML schema tree, and Q13 can be simplified as/A/1B/2C/3F by rule 3. For other queries of file #2-#10, we omit here considering the space. Figure 7 is query time of File #1. For Figure 7, we can see that: for type 1 query Q11, our method presented limited improvements over naïve method, because type 1 query cannot be simplified in stage 1 of Algorithm 3. But in stage 2 of Algorithm, our method can compensate for the loss of time in stage 1. For type 2 query Q12, as there are no results satisfying the query, our method can immediately return "no result" just by stage 1 of Algorithm 3. So our method has many improvements over the naïve method. Similarly, for type 3 query Q13, as it can be simplified in some degree, our method has pretty improvements over the naïve method.

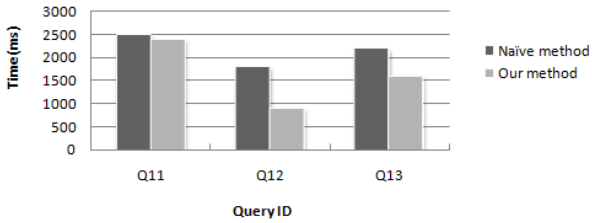


Figure 7. Query execution time of file#1.

We got similar results considering files #2-10 except for type 1 query in file #4. The reason behind this is: our method's improvement in stage 2 cannot compensate for the extra cost in stage 1 for this query.

In one word, for type 1 queries, our method has some improvements over the naïve method in most cases. For type 2 queries, our method has many improvements over the naïve method in all cases. And for type 3 queries, our method has pretty improvements over the naïve method in all cases.

7. Conclusions and Future Work

In this paper, we proposed a new mapping approach to map XML data so that store and query XML data can be efficient. The main reason behind this is that the new mapping can preserve the relationships between parent and child nodes (and also ancestor and descendent nodes) of the original XML data tree. And also, an XML schema tree produced from the original XML data tree can be used to guide and improve the efficiency of querying the corresponding XML data tree. To demonstrate this, we give an efficient query algorithm to query the XML data tree by using the corresponding set of inverted indexed circular list and its schema.

For future work, we think that map only related parts of the XML data tree according to the given query to improve the storage and query efficiency further. The key problem is how to decide the proper parts of the XML data tree to satisfy the given query without loss of information. More challenge work is to design an automatic mechanism to finish the parts selection within acceptable time.

Acknowledgements

The work is supported by Anhui Province Colleges Nature Science Research Project(No.KJ2015A325), Anhui Province Quality Engineering (2015zy073), Introduction of Talents Foundation and Academic Leader Foundation of Anhui Xinhua University (2014XXK06), National Nature Science Foundation of China (No.11201002), and Natural Science Foundation of Anhui Province (No.1208085MF110).

References

[1] Amer-Yahia S., Cho S., Lakshmanan L., and Srivastava D., "Tree Pattern Query Minimization," *The International Journal on Very Large Data Bases*, vol. 11, no. 4, pp. 315-331, 2002.

[2] Arenas M. and Libkin L., "A Normal Form for XML Documents," *ACM Transactions on Database Systems*, vol. 29, no. 1, pp. 195-232, 2004.

[3] Arion A., Bonifati A., Manolescu I., and Pugliese A., "Path Summaries and Path Partitioning in Modern XML Databases," *World Wide Web*, vol. 11, no. 1, pp. 117-151, 2008.

[4] Beyer K., Cochrane R., Josifovski V., Kleewein J., Lapis G., Lohman G., Lyle R., Özcan F., Pirahesh H., Seemann N., Truong T., Linden B., Vickery B., and Zhang C., "System RX: One Part Relational, One Part XML," *in Proceeding of the ACM SIGMOD International Conference on Management of Data*, Maryland, pp. 347-358, 2005.

[5] Chen L., Bernstein P., Carlin P., Filipovic D., Rys M., Shamgunov N., Terwilliger J., Todic M., Tomasevic S., and Tomic D., "Mapping XML To A Wide Sparse Table," *in Proceeding of IEEE 28th International Conference on Data Engineering*, Washington, pp. 630-641, 2012.

[6] Chen Z., Jagadish H., Lakshanan L., and Paparizos S., "From Tree Patterns to Generalized Tree Patterns: on Efficient Evaluation of Xquery," *in Proceeding of 29th International Conference on Very Large Data Bases*, Berlin, pp. 237-248, 2003.

[7] Deutsch A., Fernandez M., and Suciu D., "Storing Semistructured Data With STORED," *in Proceeding of ACM SIGMOD International Conference on Management of Data*, Pennsylvania, pp. 431-442, 1999.

[8] Georgiadis H. and Vassalos V., "Xpath on Steroids: Exploiting Relational Engines for Xpath Performance," *in Proceeding of the ACM SIGMOD International Conference on Management of Data*, Beijing, pp. 317-328, 2007.

[9] Lin R., Chang Y., and Chao K., "A Compact and Efficient Labeling Scheme for XML Documents," *in Proceeding of 18th International Conference on Database Systems for Advanced Applications*, Wuhan, pp. 269-283, 2013.

[10] Lin X., Wang N., Zeng X., and Sun Y., "XML Normalization Based on Entity Segments," *Information Sciences*, vol. 239, pp. 85-95, 2013.

[11] Lv T. and Yan P., "A Framework of Summarizing XML Documents with Schemas," *The International Arab Journal of Information Technology*, vol. 10, no. 1, pp. 18-27, 2013.

[12] Lv T. and Yan P., "Mapping Dtds to Relational Schemas with Semantic Constraints," *Information and Software Technology*, vol. 48, no. 4, pp. 245-252, 2006.

[13] Murthy R., Liu Z., Krishnaprasad M., Chandrasekar S., Tran A., Sedlar E., Florescu D., Kotsovolos S., Agarwal N., Arora V., and Krishnamurthy V., "Towards an Enterprise XML Architecture," *in Proceeding of the ACM SIGMOD International Conference on Management of Data*, Baltimore, pp. 953-957, 2005.

[14] Schmidt A., Keysten M., Windhouwer M., and Wass F., "Efficient Relational Storage and Retrieval of XML Documents," *in Proceeding of the Third International Workshop on the Web and Databases*, Dallas, pp. 137-150, 2000.

[15] Software AG., http://www1.softwareag.com/corporate/products/tamino/default.asp, Last Visited 2014.

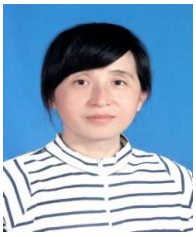
[16] Sonic Software Corporation. http://www.sonicsoftware.com/products/sonic_xml_server/index.ssp, Last Visited 2014.

[17] Tatarinov I., Viglas S., Beyer K., Shanmugasundaram J., Shekita E., and Zhang C., "Storing and Querying Ordered XML Using a Relational Database System," *in Proceeding of the ACM SIGMOD International Conference On Management of Data*, Madison, pp. 204-215, 2002.

[18] Teng L., Ning G., and Ping Y., "Normal Forms for XML Documents," *Information and Software Technology*, vol. 46, no. 12, pp. 839-846, 2004.

[19] Yu C. and Jagadish H., "XML Schema Refinement through Redundancy Detection and Normalization," *The VLDB Journal*, vol. 17, no. 2, pp. 203-223, 2008.

Teng Lv received his PhD degree from Fudan University, China. His research interests include database and data management. He is the author or coauthor of more than 70 journal papers or reviewed conference papers. He is the reviewers or PC members of several journals and conferences both at home and abroad.



Ping Yan received her PhD degree from Fudan University, China. Her research interests include partial differential equations and their applications in neural network and epidemic diseases, and data management.

Weimin He revieved his PhD degree from University of Texas at Arlington, USA. His research interests include XML data management, information retrieval and Peer-to-Peer computing data management.