A Novel Approach for XML Query Optimization

Shilpa P. Mene(PG student), Prof. S.M.Kamalapur KKWIEER Nasik, Prof. N.L. Bhale MCOERC Nasik

Abstract
Extensible Markup Language (XML) has gained importance in web and middleware development from the end of last millennium. It is write-it-yourself markup language that one uses to describe data and it allows for more precise structuring of that data than is possible with more rigid markup language. XML is the product of the world wide web consortium, W3C. As XML is involved in the access of information, the intermediate processing on XML should not be the bottleneck to deteriorate the performance of the data access. The choice data structure to represent the XML document is a tree. Twig pattern matching on XML trees is core operation for optimal evaluation of XML queries. But optimality of any pattern matching algorithm depends on labeling scheme applied to the logical tree of XML document on which twig pattern is to be matched. Most of existing labeling schemes computes the labels by traversing the logical tree of XML document in some order which is considered for prediction of relationship such as P-C and A-D between the elements.Proposed labeling scheme is designed to reduce the space requirement without compromising the speed of execution.

Key terms
Encoding, Labeling, Pattern Matching, Semi-structured Data, Twig, XPath

1. Introduction

XML query optimization is need of the time as the use of XML is inevitable for majority of web accesses, e-commerce, cloud computing and information retrieval in distributed environment that is Service Oriented Architectures. The various functional units within an enterprize connected through multiple media, and each comprising of several computing systems are linked together. As Business Logic (Middleware) is playing the crucial role in information exchange and XML is inseparable part of communication contents, it is worth to understand, study and apply the optimizing solutions to cope up the latency issue in distributed environment. There were the attempts of researchers to address various aspects to improve query response time. Only fast access solely con not qualify for exactness of the intent of the query. So there is a need to design the solution so that time efficiency and relevance should go hand in hand. The proposed work is an attempt to use an appropriate intermixes of optimal labeling and twig pattern matching algorithm and further enhancing it with the usage of indexing for speedy search. Majority of labeling schemes are rigid, that is not flexible to update in XML document, if update is required whole process has to be repeated with previous efforts wasted. Many labeling schemes need large space that may grow dynamically leading to increased disk accesses and hence defeating the purpose of minimizing response time. Consequently the pattern matching algorithm is affected with the drawbacks of chosen labeling scheme. Varying size of data in question can lead to varying performances due to matching cross. The organization of paper starts with literature survey and system requirement specification followed by mathematical foundation then the adequate set of results on different data sets are provided to support the claim of optimization.

2. Related Work

The eXtensible Markup Language (XML) is the universal format for structured documents and data on the Web. Main Advantages of XML are it is human and machine readable, it is more flexible than HTML as well as not so complicated as SGML and unlike relational table, XML can describe tree and graph structural data. An XML document is commonly modeled as a rooted, ordered tree, as given in an example

```
<book year="1967">
   <title>The politics of experience</title>
<author>
    <firstname>Ronald</firstname>
     <lastname>Laing</lastname>
</author>
</book>
```

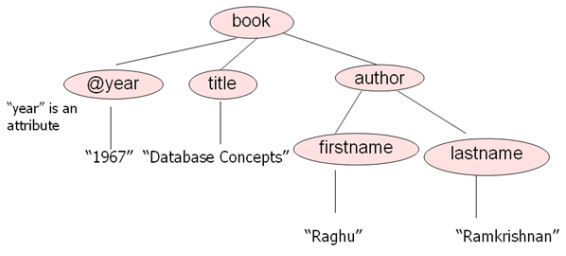
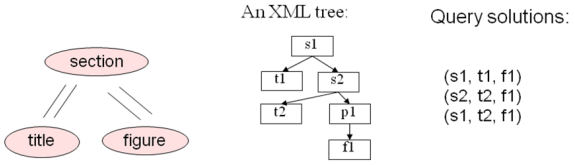


Figure 1: An XML document as rooted ordered tree

The Major standards for querying XML data are

XPath and XQuery, which includes two parts, namely value match (keyword-based search) and structure match (complex queries specified in a tree-like structure). A keyword search is similar to content retrieval in information retrieval technology. However, supporting structural query basically means that we need an effective way to match (i) the query node tag and (ii) the structural relationships between query nodes. There are three main types of relationships, namely, Parent-Child (P-C), Ancestor-Descendant (ANCESTOR-DESCENDANT) or siblings. Hence XML Twig pattern match is core operation in XQuery and XPath. An XML twig pattern is a small tree whose nodes are tags, attributes or text values; and edges are either parent-child (P-C) or ancestor-descendant (ANCESTOR-DESCENDANT) relationships.

Consider the following twig pattern and document



The approach involves two steps, namely labeling and computing. Labeling scheme assigns each element in the XML document tree an integer label to capture the structural information of documents. Computing involves use labels to answer the twig pattern without traversing the original document.

3. LABELING SCHEME

A labeling scheme for a document tree D is a decentralized structural summary of a specific set of tree relations in D. Each node in D is assigned a typically unique node label, so that any of these relations between the nodes in D can be inferred from their labels, without access to remote parts of D or to a global representation of the entire document tree. Several labeling schemes (also known as numbering scheme or encoding scheme) have been introduced, based on the following observation. Edges in XML data trees represent structural relationships between data nodes. To answer XML queries, structural relationships, or more specifically reachability between any pair of nodes in XML data trees is compared.

These labeling schemes can be classified, based on the fundamental principles underlying the different labeling schemes. Classification highlights each technique's main characteristics, advantages and drawbacks. Most node labeling schemes are based on the node-labeled data model. (2)shows an example XML document, which is used throughout. In a node-labeled data tree, there are two main objects, namely, nodes and edges. Nodes can be further classified into (1) Element Node (2) Attribute Node and (3) Value Node. Element Nodes correspond to the tags in the XML document.

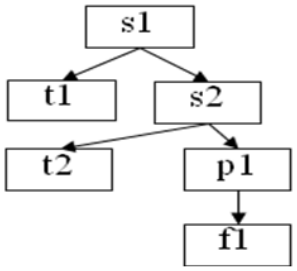


Figure 2: An Example XML tree

There has been a great diversity of labeling schemes, since the emergence of the XML. Generally, they can be broadly classified into three main categories, viz Subtree labeling, Prefix-based labeling and Multiplicative labeling as shown in Figure

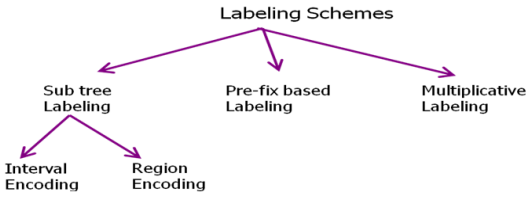


Figure 3: Classification of labeling schemes

4. SUBTREE LABELING

This category is the simplest, where the label of a given document node v in D encodes the position and the extent of the subtree Dv of D that is rooted in v, by means of offsets in the sequence of nodes resulting from traversing (at least a part of) the document tree in a specific order. While the exact representation of the subtrees varies accordingly, for the given nodes v, w in D, their ANCESTOR-DESCENDANT and P-C relationships are

always determined by testing whether Dv contains Dw. The label of a node is usually concise in this group of labeling scheme. Performance degrades in an update intensive environment, as the labels usually need to be regenerated. The subtree labeling can be further broken down into two subclasses: interval encoding and region encoding.

4.1. Interval Encoding

This is the earliest labeling scheme proposed, where the initial objective was to accelerate routing in communication networks. It has then inspired research in labeling semistructured and structured data. Tree Traversal Order: introduced the first XML numbering scheme based on tree traversal order. In this scheme, each node is labeled with a pair of unique integers consisting of preorder and postorder traversal sequences, as shown in (4). His proposition is: for two given nodes v and w of a tree D, v is an ancestor of w, if and only if v occurs before w in the preorder traversal of D and after w in the postorder traversal. By using this approach, we can determine the ANCESTOR-DESCENDANT relationship easily. Nevertheless, the P-C relationship could not be determined directly. As such, this method is inefficient for a dynamic XML document, because whenever a new node is inserted or deleted, the preorder and postorder values need to be recomputed.

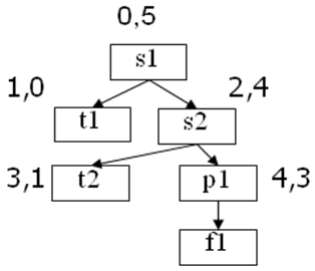


Figure 4: XML tree with Pre-order Label

Extended Preorder Traversal: IS a numbering scheme integrated with indexing mechanisms, which enables efficient search by value and structure. It is designed based on the notion of extended preorder traversal, to accommodate future insertion gracefully. Here, each node in the XML tree is labeled with a pair of numbers ¡order, size¿, as shown in (5). To enable future insertions gracefully, size(v) can be an arbitrary integer larger than the total number of current descendant of v. However, a global reordering is necessary when all the reserved spaces have been consumed. Moreover, it is not clear how one can assign a large enough value for "size", based on the three propositions.

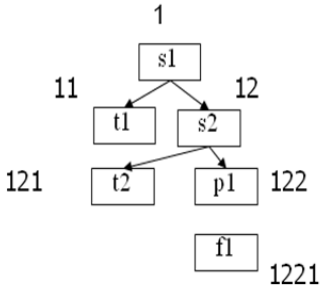


Figure 5: XML tree with Extended Preorder Label

4.2. Region Encoding

The region encoding (also known as range encoding) schemes are originally designed for structured text databases. A node in the document tree corresponds to a substring of the entire string of the XML document. Such substrings can be naturally identified by region coordinate, which is interpreted as a pair of integers (start position, end position) of the substring counting from the beginning of the XML document using inverted lists to process containment query of XML data. Containment queries are a class of queries based on the relationships among elements, attributes and their contents. To support processing of semi structured XML document, the inverted index is extended to text index (T-index) and element index (E-index). T-index is similar to the traditional index in information retrieval systems, while E-index maps element to inverted lists. Each inverted list records the occurrences of a word or an element known as term. Each occurrence is indexed by its document number, position and depth within the document, which is denoted as (docno, begin: end, level) for E-index and (docno, wordno, level) for T-index. The position (begin, end, wordno) are generated by counting the word numbers in the XML document, based on depth-first traversal.

5. PREFIX BASED LABELING

In the prefix labeling scheme (also known as path-based labeling scheme), the label of a given node v encodes

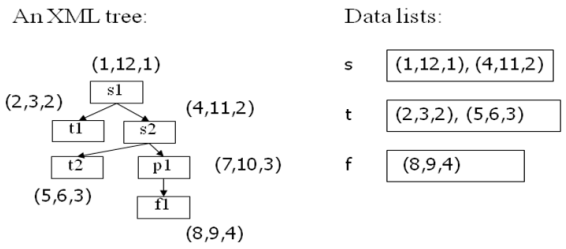


Figure 6: XML tree with Region Encoding

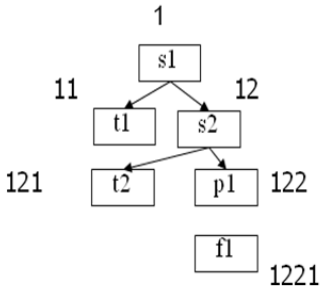


Figure 7: XML tree with Tree Location Address Label

the nodes on the path from the document root down to v, as a sequence to uniquely denote an ancestor of v on that path. Thus, given a node v and its ancestor u, their relationship could be determined precisely, i.e. u is an ancestor of v if label(u) is a prefix of label(v). However, this labeling scheme has some limitations in terms of space consumption and efficiency. The size of the label grows with the length of the encoded path. In the worst case, its size is O(n). Hence, the path encoding takes up more space as compared to subtree encodings, whose label size is O(log n). This further affects the efficiency during the query evaluation process, as more time is needed to process the longer encoded path.

5.1. Tree Location Address

Kimber [4] proposed an approach using "tree location address" to locate a node in a tree by selecting an ancestor node at each level of the tree. According to this approach, each identifier of an ancestor node is a prefix of its descendant. A node id (nid) is the concatenation of the nid through the path from the root to the respective node. Using this method requires variable space to store the identifiers. Thus, the time to determine the ANCESTOR-DESCENDANT relationship is not constant, as it depends on the length of identifiers. As a result, this method may not be practical for large databases.

5.2. Simple Prefix

Cohen et al. [5] proposed a simple prefix labeling scheme, wherein each label inherits its parent's label as prefix. The root is labeled with an empty string (" "). The first child of the root is labeled with "0", the second child with "10", followed by the third and fourth with "110" and "1110" respectively. For any node L(v) denoting the label of v, the first child of v is labeled with L(v)"0", the second child of v is labeled with L(v)"10" and the ith child with L(v)"(1..1)i-10". Referring to (8), for example, the node chapter is an ancestor of caption, for "0110" is a prefix of "011010100". This labeling scheme does not need to be regenerated for any arbitrarily heavy update such as deletion or addition of nodes or subtree to each right side of a subtree. The limitation of this technique is that the size of simple prefix is often too huge.

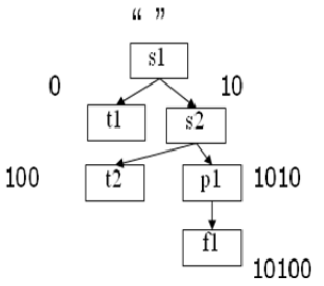


Figure 8: XML tree with Simple prefix Label

5.3. Dewey ID

Dewey ID [6] is based on the Dewey Decimal Classification System , which is widely used by librarians. The Dewey ID labeling is very similar to tree location address except that dot separators are present in Dewey ID labeling to differentiate each label inherited from each level of their ancestors. Using this labeling scheme, structural relationships between elements can be determined efficiently. It is said that element u is an ancestor of element v if and only if label(u) is a prefix of label(v). To test whether any two nodes is in P-C relationship, the number of integer in label(u) is one more than that of label(v). In Dewey ID, the size of the node label at each level is exactly one byte and thus the maximum label size (in

bytes) depends only on the maximum depth of the tree. As a matter of fact, Dewey IDs may become quite long, especially in trees with large depth and fan-out values, due to the large size of labels and delimiters (dot separators), which incurs high storage overhead. Therefore, serious effort is needed to develop a more practical solution for them. Nevertheless, for fan-out degrees greater than 10, larger alphabets such as Unicode Character Set could be used to label each node instead.

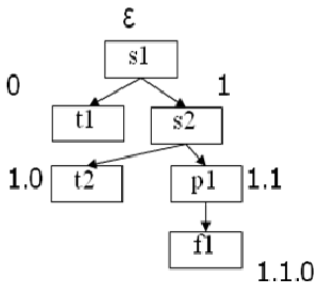


Figure 9: XML tree with Dewey labels

5.4. ORDPATH

It is similar conceptually to the Dewey ID. ORDPATH encodes the P-C relationship by extending the parent's ORDPATH label with a component for the child. The main difference between ORDPATH and Dewey ID is that even numbers are reserved for further node insertions in ORDPATH. An example of tree labeling using ORDPATH is depicted in Figure 2.2.4. During the initial labeling, ORDPATH assigns only positive and odd integers. Even and negative integer values are reserved for later insertions. For instance, if the newly inserted node is to be added to the right of all the existing children, its label is generated by adding +2 to the last ordinal of the last child. If the newly inserted node is to be added to the left of all the existing children, its label is generated by adding ?2 to the last ordinal of the first child. However, this approach is not suitable for deep trees. As can be seen in Figure 8, the size of the ORDPATH label may become quite huge, especially in trees with large depth and fan-out values. To cope with such trees, ORDPATH uses labels that do not reflect ancestry and thereby loses some of its expressivity.

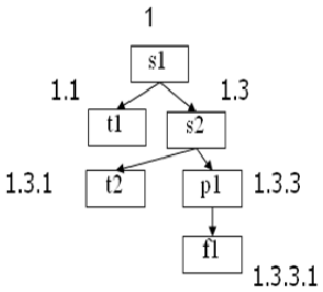


Figure 10: XML tree with ORDPATH

6. MULTIPLICATIVE LABELING

This category of labeling scheme uses atomic numbers to identify nodes. Relationships between nodes can be computed, based on some arithmetic properties of the node labels. The idea is to find a mapping from a given irregularly structured document tree D to a regular tree D', such that some of the arithmetic properties in D' carry over to D. The main limitation is that the computation of multiplicative labeling is very expensive. Hence, it is unsuitable for labeling a large-scale XML document.

6.1. Unique Identifier (UID)

It enumerates the node using K-ary tree. Few calculations are needed to determine parent of a node but label computation is computationally expensive and recursive lookup is necessary to determine A-D P-C relationships.Here, each internal node is supposed to have the same number of fan-out k. Thus, virtual nodes are created to balance the number of fan- outs. Starting from each level, each node is assigned a label starting with integer 1 from top to bottom and from left to right. The virtual nodes created are shown only until level 3, due to space constraint. As opposed to other labeling schemes, which can only com- pare two already known identifiers in order to determine the parent-child relationship, the UID technique has an interesting property whereby the parent node can be determined, based on the identifier of the child node.

6.2. rUnique Identifier (rUid)

It is an extention to UID and recursive by nature. It clusters the node into connected subsgrph. It supports update/ inserts but as UID the label computation caused is an overhead and requires recursive lookup.

6.3. Prime Number Based

In this scheme prime numbers are used for labeling by using top down and bottomup approach. With this, the root will be labeled with the first prime number 1. Each non-leaf node will be given a unique prime number. The label of each node is the product of its parent nodes' label (parent-label) and its own assigned number (self-label).It is very efficient on update but the label generation is computationally expensive This approach is good for dynamic updates. When a new node is inserted, an unassigned self-label prime number is allocated. Thus, no re-labeling is required. However, one disadvantage of the prime number labeling scheme is that each prime number can only be used once. Thus, the size of the label increases as it reaches the bottom of the tree. The size storage of the label has direct impact on the performance of XML query processing. It is, therefore, not suitable for a tree with many levels. The authors also suggest some optimization techniques such as (1) Assign small prime number to the root and level right after the root, so that the value inherited will be smallest possible (2) Use number 2 and its sequence. the only even prime number to label the self-labels of the leaf nodes and the odd numbers for the non-leaf nodes (3) Combine those paths, which occur multiple times, to reduce redundancy and further decrease the size of the labels.

Labeling schemes are the fundamental building block for many different structural joins algorithms and a valuable complement to structural indexes. Choosing a correct labeling scheme is crucial. There are several factors to be considered, when choosing a suitable labeling scheme. These are listed below:

1. Storage: How much storage space is available? Is there enough space to store each node label on disk or in memory?

2. Nature of the data: How large is the XML document? How frequently does it change?

3. Query type: Is it a structural query or a full-text query that needs to be supported?

4. Efficiency: What is the construction cost of each labeling technique? How fast is the manipulation of node labels during query evaluation?

7. Proposed method

The proposed method focuses on the development of an optimal labeling scheme to enhance the process of XML query optimization. Further the algorithm will be devised to work in tandem with the labeling scheme. The trade off will be used to exploit the benefits of labeling scheme and algorithm to satisfy the objective. The chosen duo of labeling scheme and the algorithm must complement each other. When tested on varying sizes of XML data sets and possible XML queries the claimed enhancement must be observed. Thus the labeling scheme and the algorithm work together to help optimize the XML query. XMark benchmark is used to generate synthetic data set for experiments. Different XML documents can be given as an input to system for which with the help of logical XML tree corresponding binary trees are obtained for applying labeling scheme. On labeled binary tree pattern matching algorithm is applied to find out match for twig query given. While doing this computation, performance of system is measured in terms of time and space required for computation which is to be used to compare the performance of proposed scheme with other algorithms used for same computations.
Possible Research outcome:

- A novel technique to optimize XML query execution
- Enhancement in the performance of XML data access
- Quantization of the performance of the proposed technique
- Comparative assessment of various existing node labeling and search techniques with proposed technique

7.0.1. Proposed algorithm for Labeling

1. Input: Logical XML tree mapped into equivalent binary tree
2. Output: Labeled logical XML tree
3. Start from root node and assign '1' as label (Lp) to it.
4. Start pre-order traversal from root
5. if(child is left child)
   assign label as Lc=2(Lp) and mark as visited

else
assign label as Lc=2(Lp)+1 and mark as visited

6. Set Lp=Lc

7. Repeat step 5 and 6 until all nodes are marked as visited.

8. Results and Discussion

As many of the XML documents specially with respect to business data exchange not undergo frequent and major updates , the proposed scheme can give pattern matching result in optimal way in terms of time and space as label computation time is get reduced as range of start and end value of label can be passed for faster computation. If we compare it with ORDPATH labeling in which label composed of combination of document Id, prefix order: postfix order and Level values which requires twice computations for calculating preorder, postoerder and level value of every node. In comparison with Dewey labeling the space requirement is very less in proposed labeling.

9. Conclusion

Motivated by the need to support XML data updates in dynamic XML environments, a new XML labeling scheme is proposed. The aim of all versions of the labeling algorithm is to keep label size and required relabeling with inserts and deletes at minimum while providing several relationships. As the labeling scheme has wide range of values to be assigned to the labels the insert and delete operations can be performed without significant re-computations. The search is optimized due to numeric notation of the label. Experimental study of this project compares the performance of the proposed labeling scheme with Dewey and Extended Dewey labeling schemes. The tests are performed with 4 different cases; labeling the XML data, space requirement, query performance and update performance. The results of the performance tests confirmed that 1. The proposed labeling scheme is better in terms of space requirement. 2. The labeling process requires less time for label computation in comparison with string based labeling. 3. The twig pattern matching performs better when proposed labeling scheme is used.

References

[1] J. Lu, T. W. Ling,Z. Bao and C. Wang *Extended XML Tree Pattern Matching: Theories and Algorithms*, IEEE TKDE Journal 2011

[2] Haw Su-Cheng and Lee Chien-Sing *Node Labelling Schemes In XML Query Optimization : A Survey and Trends*, IETE technical review — Volume 26—Issue 2— Mar-Apr 2009

[3] A. Berglund, S. Boag, and D. Chamberlin, XML Path Language (XPath) 2.0, W3C Recommendation, http://www.w3.org/TR/ xpath20/, Jan. 2007.

[4] A. Deutsch, M. Fernandez, and D. Suciu. *Storing Semistructured Data with STORED* , In Proc. ACM SIGMOD Int. Conf. on Management of Data, pages 431-442, 1999.

[5] C.-C. Kanne and G. Moerkotte. *Efficient Storage of XML Data* In Proc. 16th Int.Conf. on Data Engineering, pages 198-209, 2000.

[6] C. Koch. *Efficient Processing of Expressive Node-Selecting Queries on XML Data inSecondary Storage: A Tree Automata -based Approac* . In Proc. 29th Int. Conf. on Very Large Data Bases, pages 249-260, 2003.

[7] J. Lu, T. Chen, and T.W. Ling, *Efficient Processing of XML Twig Patterns with Parent Child Edges: A Look-Ahead Approach* Proc. 13th ACM Int'l Conf. Information and Knowledge Management (CIKM), pp. 533-542, 2004.

[8] S. Al-Khalifa, H.V. Jagadish, J.M. Patel, Y. Wu, N. Koudas, and D. Srivastava, *Structural Joins: A Primitive for Efficient XML Query Pattern Matching* Proc. 18th Int'l Conf. Data Eng. (ICDE), pp. 141152, 2002.

[9] N. Bruno, D. Srivastava, and N. Koudas, *Holistic Twig Joins: Optimal XML Pattern Matching* Proc. ACM SIGMOD, pp. 310321, 2002.

[10] Q. Li, and B. Moon, *Indexing and Querying XML Data for Regular Path Expressions*, in Proceedings of the VLDB, pp. 361-70, 2001.

[11] M. Shalem and Z. Bar-Yossef, *The Space Complexity of Processing XML Twig Queries over Indexed Documents* Proc. 24th Int'l Conf. Data Eng. (ICDE), 2008.

[12] J. Lu, T.W. Ling, C. Chany, and T. Chen, *ŞFrom Region Encoding to Extended Dewey: On Efficient Processing of XML Twig Pattern Matching,Ť* Proc. IntŠl Conf. Very Large Data Bases (VLDB), pp. 193-204, 2005.

[13] M. Duong, and Y. Zhang, *LSDX: A New Labeling Scheme for Dynamically Updating XML Data* , in Proceedings of the 16th Australasian Database Conference, pp. 185-93, 2005.

[14] P O'Neil, E. O'Neil, S. Pal, L. Cseri, G. Schaller, and N. Westbury, *ORDPATHS: Insert-Friendly XML Node Labels* , in Proceedings of the ACM SIGMOD, pp. 903-8 ,2004.

[15] X. Wu, M.L. Lee, and W. Hsu, *A Prime Number Labeling Scheme for Dynamic Ordered XML Tree* , in Proceedings of the ICDE, pp. 66-78, 2004.

[16] M. Brantner, S. Helmer, C.-C. Kanne, and G. Moerkotte. *Full-fledged AlgebraicXPath Processing in Natix* ,In Proc. 21st Int. Conf. on Data Engineering, pages 705-716, 2005.