

# How to Store and Query XML Data

Silvia Stefanova, UDBL

## Abstract

Due to the fact XML is a universal data-exchange format, all the questions concerning storage and querying XML documents have been lately of great popularity. Query languages, called XPath and XQuery have been developed and standardized in order to query XML data not only in XML files, but anything existing as XML, including databases. XML databases running XPath and XQuery are generally classified as native and relational. The native ones are specialized engines evaluating queries on XML documents, while the relational are built on top of existing database engines.

The intention of this paper is to discuss some of the main problems and research trends from the broad research field devoted to XML storage and querying. The exposition will start with a description of XML Data Management Systems, go through the query languages XPath and XQuery, examine in details the aspects in the development of XML query processing and end up with the latest proposed algorithms for efficient query processing.

## XML Data Management Systems

Existing architectures of XML database management systems will be presented and commented with respect to storing and querying opportunities of XML data.

### 1) Storage of XML documents in relational systems

To store XML data in relational systems has been popular for the last several years and as well implemented by few commercial DBMS. The reason is that it is based on adaptation and reuse of relational technology. However, there are significant differences among the existing methods on how to save and hence query XML data in RDBMS.

#### Shredding XML documents into relations

XML documents are transformed into atomic values and after that stored as relations in relational tables. XQueries are translated to SQL queries to be evaluated by the RDBMS query processor. Several different shredding techniques and query capabilities has been proposed [14, 15]. The advantage of this method is that it does not require big modifications of the existing database engine.

#### XML, stored as unparsed text

XML data is stored in VARCHAR or LOB (large object) columns of relational tables. XML data is queried by a XQuery processor external to the database and invoked as a user-defined function. This approach is used by commercial systems offering XML support (MD2, MS SQL).

This solution is relatively simple but the entire XML document usually has to be first loaded into the memory in order to be processed.

#### Hybrid XML-relational databases

XML documents are stored on disk pages in tree structures matching the XML data model [7, 9, 16]. Hence no mapping is needed between XML and relational structures. The native XML storage is complemented with XML indexes. It is provided XQuery and SQL support.

## 2) Native XML Data Management Systems

These are systems like Niagara [18] and Timber [17] that support only XQuery. By this method, the XML document is broken into nodes and the node information, stored in a B+ tree as all document nodes are stored in order at the leaf level. In Niagara, so called inverted list indexes are created to enable efficient structural joins algorithms.

## XML Query Languages

XPath [4, 19] is a basic XML query language, used to select nodes from XML documents so that the path from the root to each selected node satisfies a specified pattern. A XPath query is specified by a sequence of alternate axes and tags.

The XML queries can be basically divided into two main groups: **path queries** and **twig queries** [3]. Path queries are simple queries against XML documents, which contain path expressions, i.e child axis “/” and descendant axis “//”. An example of a path query is “/Publisher//title”, that returns all books' titles of all publishers. The **tree queries** (also called **twig queries**) are more complex since they contain selection predicates into path expressions. One such example is “/Publisher[@name='Studentlitteratur']/book/title” that returns the titles of all books, published by the Publisher, called “Studentlitteratur”. Generally, multiple predicates might be involved in a tree query.

XQuery [1, 2, 4] is another popular XML query language, which is an extension to XPath. It is a functional language, comprised of FLOWR (For-Let-Where-Return) clauses that can be nested and composed. The following is an example of simple XQuery, defined by a FLOWR constructor that returns all the titles of the books in a bookstore that cost cheaper than 30:

```
for $x in /bookstore/book
  where $x/price > 30
  return $x/title.
```

From this example is visible that XQuery includes two parts: **twig pattern matching**, defined by FLW in the case and **result construction**, defined by Return.

## Processing of XML queries

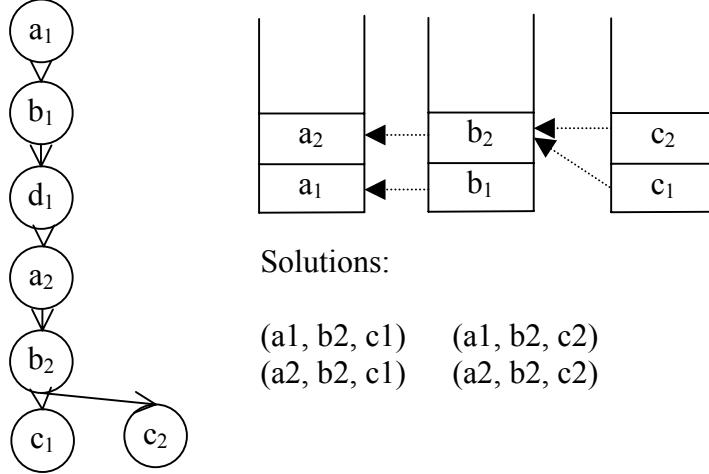
Since 2000 a lot of researchers have been working on XML query processing looking at various methods and algorithms for optimization of twig queries. A short survey of the evolution of XML query processing will be made in respect to the format of the stored XML data.

### XML data stored in flat files

In the cases where XML data is stored in the original way, i.e. in flat files without any redundant data, such as indexes, the XML query processing is actually scan of the entire XML document sequentially element by element. The performance is often very poor since most of the elements that the scan goes through might be irrelevant to the specified queries [20, 21, 22, 23, 24, 25 ].

The existing approaches for single query processing are “the automata” approach and a few stack approaches. The automata [20, 21, 22, 23, 24] approach is in fact forward navigation within the XML document. The idea is to express a XML query as automaton and to run XML documents on this automaton as if they were strings. This simple method has the weakness that it works well for single-node solutions (e.g. a set of A nodes) but has difficulties to derive tuple solutions (a set of (A, B, C) tuples). This has been overcome by a new data structure, called PathStack [13] where one

stack is created for each query node in a path query. So, using a series of linked stacks the scanned data nodes are possible to be tracked. This approach was originally introduced with the native storage of XML. The main concept of this approach for the file stored XML data is illustrated in figure 1. One stack is created for each query node in a path query. When an opening tag is encountered, the corresponding XML element is pushed into its stack, associated with a pointer to the top element in its parent stack and when a closing tag is encountered, the top element in its corresponding stack is popped out.



**Fig.1. The Path Stack approach for processing Xpath query // A// B/ C**

The complement to the PathStack approach is called TwigStack approach and it is capable to answer general twig queries [13] by decomposing twig queries into multiple root-to-leaf path queries. Each path query is processed as in a PathStack, and the query results are finally joined together to get the final result. Since in TwigStack approach the stacks corresponding to the common prefix nodes can be shared, it actually links stacks in the form of a twig instead of in the form of a path as it is with PathStack approach.

### XML data stored in relations

When XML data is loaded into relational databases, XML twig queries need to be transformed to SQL queries over relational data. Then all query processing is pushed into relational query optimizer and no extra processing work is needed [26].

If XML data conform to a schema as DTD [27], the DTD schemas are naturally transformed into relational schemas [28]. In the resulting relational schema there are separate relations created for the root element and for all sub-elements in DTD. Each sub-element's relation has a foreign key to its parent-element table. Once XML data has been shredded into relations, XML queries can be easily translated to SQL queries. By means of this approach a different relational schema is generated for a specific DTD, which capture precisely the structure of XML data. This is in contrast to approaches working with schema less XML. The latter approaches generate the same relational schema for various XML data despite the different structures. All they use positional representation of the elements of the XML data based on the tree structure of XML.

The edge approach is based on edge-labeled XML data trees [29], where all the edges in a data tree are stored in a relational table, Edge and using it, XML queries that do not contain “//” can be transformed to SQL. However, this approach involves many joins and moreover it does not support twig queries with “//” axes, i.e. descendant-descendant queries, because it does not know the number of the tags between the elements of the both sides of “//” [29]. The, so called, Node approach overcomes the latter weakness of the Edge approach, using node-labeled XML data trees

[30]. All internal nodes (element nodes and attribute nodes) in a data tree are stored in a relational table, Node. Similar to the Edge approach, execution of a query goes through two steps 1) finding the candidate-nodes and 2) node joining. The number of the joins is as well very big as it is equal to the number of query nodes in a twig query, which can result in insufficient query processing of large twig queries. In order to reduce the number of the join operations, a Path materialization approach [31] has been proposed. The difference here against the Node approach, is that instead of storing the tag of each node, it is stored the **tag path** from the root to each node (called root path). In such a way it is possible to answer twig queries in units of paths, which is more efficient comparing to units of single nodes. The Path materialization approach supports ascendant-descendant queries but it can not support them efficiently if there are multiple “//” axis in the queries (//A//B/C//D//E). This problem has been solved by the Reversed Path approach [32] in which reversed root paths of data nodes are stored in a table attribute called ReversedPath. An example of how the RP approach answers twig queries with multiple “//” is shown in figure 2. The first step is still twig decomposition (a, b), which in RP continues with further decomposing of path (3), since it includes multiple “//”. So, path (3) is decomposed into path (4) and path (5) both of each include only one “//” and hence it can be used “/F/E%” and “/G%” as search patterns on the ReversedPath attribute to retrieve data nodes of paths (4) and (5).

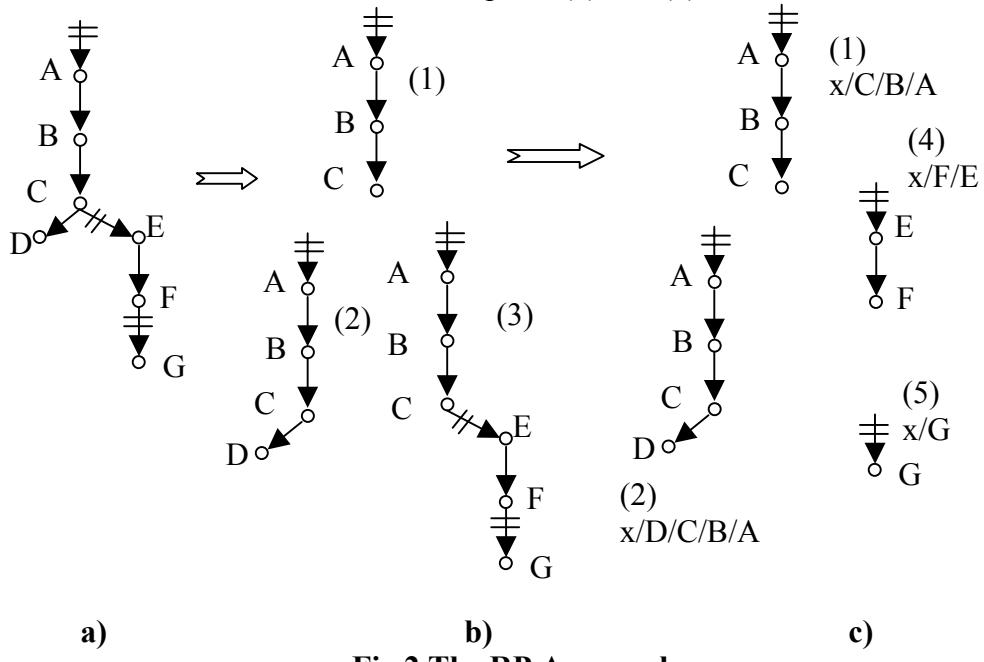


Fig.2 The RP Approach

Further, the intelligent method BLAS (Bi-LAbeling System) [33] has been developed to implement SPM (String Prefix Matching) efficiently. The key idea of BLAS is encoding each ReversedPath string into a number.

In fact the basic Reverse Path approach (without the BLAS extension) has been implemented into Microsoft SQL Server 2005.

### XML data stored in inverted lists (native storage)

In the native approach XML data is stored in **inverted lists**. One inverted list is created for each distinct tag in XML document. It gives positions of all elements of that tag name. Location of an element is expressed using its (**start**, **end**, **level**) numbers as they are stored in increasing order of their start numbers. Some inverted lists of the XML document (figure 3) look like following:

```

<Publishers>
<Publisher>
<Publisher @name='Studentlitteratur'>
  <address>Stockholm</address>
  <book>
    <title>Databases</title>
  </book>
</Publisher>
<Publisher>
  <book>
    <title>XML</title>
    <author>
      <name>Smith</name>
    </author>
  </book>
</Publisher>
<Publishers>

```

**Fig. 3. XML document**

<publisher>	→ (2, 20, 1), (21, 38, 1)
<book>	→ (9, 19, 2), (22, 34, 2)

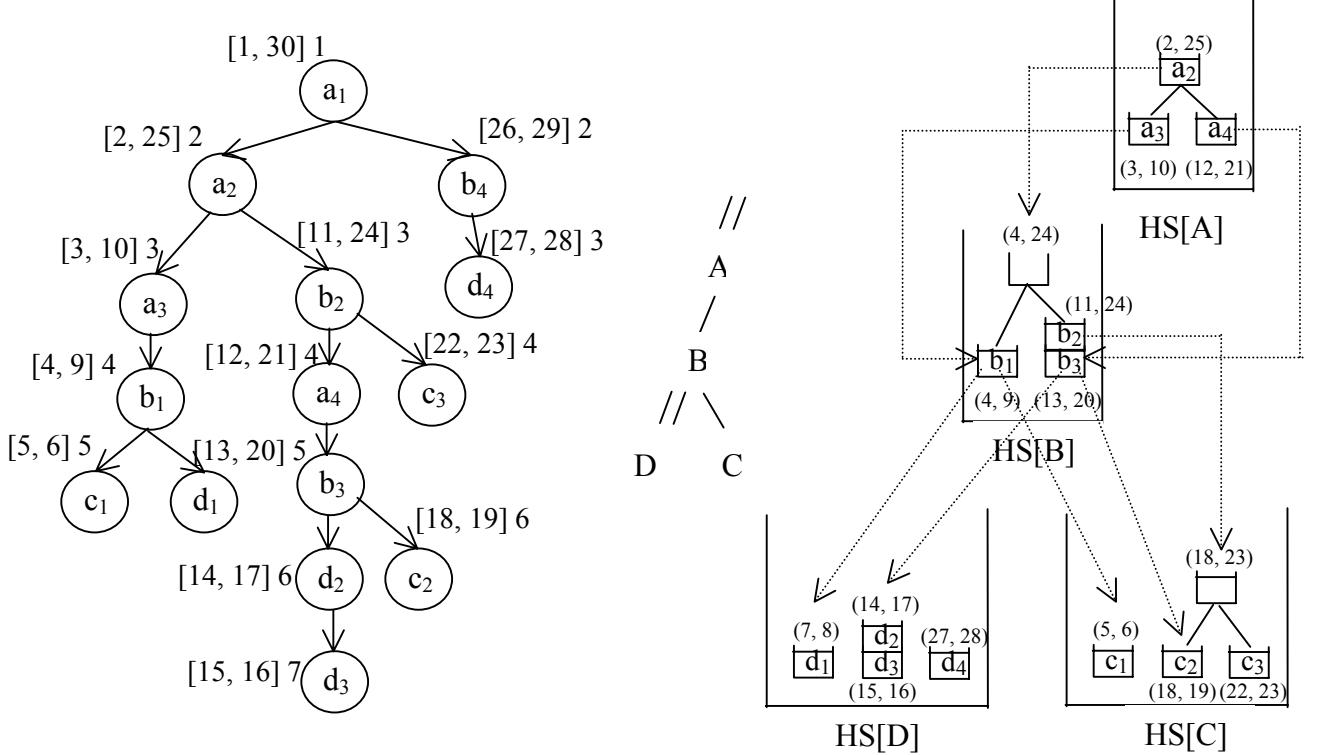
The MPMGJN (Multi-Predicate MerGe JoiN) [12] is a native approach to implement structural joins. The idea is basically the following. In order to answer a query “A/B” or “A//B” two cursors are created on the lists of A and B elements, which are sorted by **start** numbers increasingly. In the beginning the two cursors point to the head of the lists. Later on they are compared with each other and advanced in line to implement merge join. Another native approach, called StackTree [10] has a better efficiency than MPMGJN for queries like “A/B” since it uses a **stack** structure to cache A (parent) elements and so it is not necessary to go through the whole list of B elements in order to find which B:s are children of A. The scenario for the query “A/B” in StackTree is the following having in mind that the XML data is sorted in inverted lists. If it coming an A-tagged

node, it is pushed into the stack. If it is a B-tagged node, StackTree tries to use it to form tuple solution with the existing A-tagged nodes in the stack. Both StackTree and MPMGJN are binary join algorithms, i.e. they join only a pair of inverted lists. However, a complete twig query consists of a series of binary joins. The method for processing such query by decomposing it into binary joins and combining them, is not that effective since it generates a large amount of intermediate results. In order to overcome this, a Holistic approach [13] has been proposed. By it multiple inverted lists are joined at one time so that no intermediate query results are generated. A PathStack algorithm for answering path queries and TwigStack algorithm for answering general twig queries over inverted lists are provided by the Holistic approach. The ideas are the same as with PathStack and TwigStack algorithms used for queries over flat XML files, discussed above but here they are adjusted for native XML data storage. The difference is that the popping out of the stack is triggered by the arrival of nodes with higher start numbers than their end numbers, rather than the arriving of own closing tag.

The Holistic approach has been further extended with a skip technique to avoid visiting nodes that do not form any tuple solutions with other nodes [34] and later this been adjusted to process twig queries with OR predicates. More recently TJFast [35] has been proposed to access only leaf elements by exploiting extended Dewey IDs.

One of the latest query processing algorithm with very good performance, called Twig<sup>2</sup>Stack [8], bottom-up processing, is based on a hierarchical stack encoding scheme to compactly represent the twig results. The main idea, that is the basic difference against the PathStack encoding and actually overcomes its weakness for twig queries with PC edges, is in *organizing the elements matching the same query node in a hierarchical structure* in order to capture their AD relationships. The essence of Twig<sup>2</sup>Stack is sketched in figure 4. A document element **e** is pushed into a hierarchical HS[E] stack if it satisfies the sub-twig queries rooted at this query node E. Only E’s child query nodes N need to be checked due to the fact that all the elements in HS[N] must already have satisfied the sub-twig query rooted at N. Meanwhile, in order to maintain the hierarchical structure of the elements in HS[E], the stack trees in HS[E] are merged and **e** is pushed to the top of the merged stack. When there is no existing stack tree that is the descendant of **e**, a new stack to hold **e**, is created.

The Twig<sup>2</sup>Stack algorithm based on the new hierarchical stack encoding scheme to represent twig results, has shown a better performance in query processing than TwigStack [13] and TJFast[35] but it is also capable to process more complex general twig queries.



**Fig. 4 Hierarchical stack encoding for the showed twig query**

## Conclusion

The evolution of the mechanisms for XML data storage and based on that development of algorithms for XML query processing shows that better performance have been achieved by the native storage approach with native algorithms to process XML twig queries. The main idea is to store XML data in a native tree format and query it by using algorithms, processing efficiently smart constructions (stacks) to extract the consequences of elements within a XML document that correspond to the query pattern.

## References:

1. W3C Group. XQuery 1.0: an XML Query language. <http://www.w3.org/TR/xquery/>, 2007.
2. Introduction to XQuery. [http://www.w3schools.com/xquery/xquery\\_intro.asp](http://www.w3schools.com/xquery/xquery_intro.asp).
3. Queries on XML Documents. <http://www.brics.dk/~amoeller/XML/querying/queries.html>.
4. W3C Group. XQuery 1.0 and XPath2.0 Functions and Operations. <http://www.w3.org/2005/02/xpath-functions/>
5. Galax, <http://www.galaxyquery.org/>
6. Qexo, <http://www.gnu.org/software/qexo/>
7. A. Halverson, V. Josifovski, G. Lohman, H. Pirahesh, M. Mörschel. ROX: Relational Over XML. *Proceedings of the VLDB Conference. Toronto, Canada, 2004.*
8. S. Chen, Hua-Gang Li, J. Tatemura, Hsiung, D. Agrawal, K. Selcuk Candan. Twig<sup>2</sup>Stack: Bottom-up Processing of a Generalized-Tree-Pattern Queries over XML Documents. *VLDB Conference, 2006*
9. P. Boncz, T. Grust, etc. MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. *SIGMOD Conference, 2006.*
10. Shurug Al-Khalifa, H. V. Jagadish, Nick Koudas, Jitnesh M. Patel, Divesh Srivastava, and Yuqing Wu. Structural joins: a primitive for efficient XML query pattern matching. *ICDEConference, 2002.*

11. A query language for XML, <http://www.w3.org/TR/xquery>
12. Chun Zhang, Jeffrey F. Naughton, David J. DeWitt, Qiong Luo, and Guy M. Lohman. On supporting containment queries in relational database management systems. *SIGMOD Conference*, 2001.
13. N. Bruno, N. Koudas, and D. Srivastava. Holistic Twig Joins: Optimal XML Pattern Matching, *SIGMOD Conference*, 2002.
14. Microsoft SQL Server SDK Documentation, Microsoft, <http://www.microsoft.com>.
15. L. Ennser, C. Delpote, M. Oba, K. Sunil: Integrating XML with DB2 XML Extender and DB2 TextExtender, IBM REDbooks, <http://www.redbooks.ibm.com/redbooks/pdfs/sg246130.pdf>
16. M. Nicola, Bert van der Linden, Native XML Support in DB2 Universal Database, *VLDB Conference, 2005*
17. H. V. Jagadish, Shurug Al-Khalifa, Adriane Chapman, Laks V. S. Lakshmanan , Andrew Nierman, Stelios Paparizos, Jignesh M. Patel, Divesh Srivastava, NuweeWiwatwattana, Yuqing Wu, Cong Yu, A Native XML Database, VLDB 2002.
18. J. Naughton, D. DeWitt, D. Meire et al, The Niagara Internet Query System. *IEEE Data Engineering Bulletin, vol. 24, n2, June 2001.*
19. W3C Group. XML path language (XPath) 2.0 <http://www.w3.org/TR/xpath20/>, 2007
20. Rakesh Agrawal, Alexander Borgida, and H. V. Jagadish. Management of transitive relationships in large data and knowledge bases. *SIGMOD Conference*, 1989.
21. Yanlei Diao, Peter M. Fischer, Michael J. Franklin, and Raymond To. YFilter: Efficient and scalable filtering of XML documents. *ICDE Conference*, 2002.
22. Yanlei Diao and Michael J. Franklin. High-performance XML filtering: an overview of YFilter. *IEEE Data Engineering Bulletin, 26:41–48, 2003.*
23. Yanlei Diao, Mehmet Altinel, Michael J. Franklin, Hao Zhang, and Peter M. Fischer. Path sharing and predicate evaluation for high-performance XML filtering. *ACM Transactions on Database Systems (TODS), 28:467–516, 2003.*
24. Nicolas Bruno, Luis Gravano, Nick Koudas, and Divesh Srivastava. Navigation- vs. index-based XML multiquery processing. *ICDE Conference*, 2003.
25. Feng Tian, Berthold Reinwald, Hamid Pirahesh, Tobias Mayr, and Jussi Myllymaki. Implementing a scalable XML publish/subscribe system using a relational database system. *SIGMOD Conference*, 2004.
26. Gang Gou, Rada Chrkova, XML query processing: A survey, 2005
27. W3C Group. Guide to the W3C XML specification (XMLspec) DTD, version 2.1. <http://www.w3.org/XML/1998/06/xmlspec-report.htm>, 2004
28. Jayavel Shanmugasundaram, Eugene J. Shekita, Jerry Kiernan, Rajasekar Krishnamurthy, Stratis Viglas, Jeffrey F. Naughton, and Igor Tatarinov. A general techniques for querying XML documents using a relational database system. *SIGMOD Record, 30:20–26, 2001.*
29. Daniela Florescu and Donald Kossmann. Storing and querying XML data using an RDMBS. *IEEE Data Engineering Bulletin, 22:27–34, 1999.*
30. Chun Zhang, Jeffrey F. Naughton, David J. DeWitt, Qiong Luo, and Guy M. Lohman. On supporting containment queries in relational database management systems. *SIGMOD Conference*, 2001.
31. Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, and Shunsuke Uemura. XRel: a path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology (TOIT), 1:110–141, 2001.*
32. Shankar Pal, Istvan Cseri, Gideon Schaller, Oliver Seeliger, Leo Giakoumakis, and Vasili Vasili Zolotov. Indexing XML data stored in a relational database. *VLDB Conference*, 2004.
33. Yi Chen, Susan B. Davidson, and Yifeng Zheng. BLAS: An efficient XPath processing system. *SIGMOD Conference*, 2004.
34. Haifeng Jiang, Wei Wang, Hongjun Lu, and Jeffrey Xu Yu. Holistic twig joins on indexed XML documents. *VLDB Conference*, 2003
35. J. Lu, T.W. Ling, C. Y. Chan and T. Chen. From Region Encoding To Extended Dewey: On efficient Processing of XML Twig Pattern Matching. *In Proceedings of VLDB, pp. 193-204, 2005.*