Sinteza
Međunarodna naučna konferencija Univerziteta Singidunum

Impact of Internet on Business Activities in Serbia and Worldwide
Uticaj Interneta na poslovanje u Srbiji i svetu

DOI: 10.15308/SINTEZA-2014-878-881

# MANAGING XML TREES USING XPATH, XQUERY, CLUSTERING AND TREE TUPLES OVER SEDNA XML DATABASE

**Hasham Elzentani**
Singidunum University, Belgrade, Serbia

**Abstract:**
XML is a markup language that defines a set of rules for encoding semi-structured data in a readable format for both humans and machines, in other words the increase of various XML data as information source has raised a number of issues about how to represent and manage this data. Find out an ontology knowledge to derive semantics of XML document has become a major challenge in semi-structured data management. XPath and XQuery languages are used for selecting nodes and compute values from XML document, tuple the XML tree and address the problem of XML data clustering according to structure with ontology knowledge. Experiment on real XML database gives evidence that our proposed approach is highly effective in tupling and clustering of XML tree. So this paper will propose Sedna, which is a free native XML database that provides flexible XML processing facilities include W3C XQuery implementation. Finally we will discuss the results of No. of replayed nodes and time taken by XPath or XQuery.

**Key words:**
Sedna,
XML clustering,
XPath,
XML trees,
XML tuples,
XQuery.

## INTRODUCTION

XML is touted as the driving-force for representing and exchanging data on the Web. Indeed, the semi-structured and self-describing physiognomy of XML makes it feasible to model a broad variety of data as XML documents, in order to fulfill the promises of the next generation Web. In such a context, a challenge is inferring semantics from XML documents according to the available syntactic information, namely structure and content features. This has several interesting application domains, such as integration of data sources and query processing, that can be seamlessly generalized to any kind of semi-structured data. As a fundamental exploratory data mining task, clustering represents the natural solution to discover common characteristics and specific facets exhibited by XML documents. However, the complexity intrinsic to semi-structured data requires non-trivial effort to define an effective clustering framework. Extracting significant features, modeling document structures and contents, defining an appropriate notion of homogeneity between documents are only some of the issues to be addressed [7].

## XML TREES AND PATHS

A *tree T* is a tuple $T = <r_T, N_T, E_T, \lambda_T>$, where $N_T \subseteq N$ denotes the set of nodes, $r_T \in N_T$ is the distinguished root of $T$, $E_T \subseteq N_T \times N_T$ denotes the (acyclic) set of edges, and $\lambda_T : N_T \mapsto \Sigma$ is a function associating a node with a label in the alphabet $\Sigma$. Let Tag, *Att*, and *Str* be alphabets of tag names, attribute names and strings, respectively. An *XML tree XT* is a pair $XT = <T, \delta>$, such that: (1) $T$ is a tree defined on the alphabet $\Sigma = Tag \cup Att \cup \{S\}$, where symbol $S \notin Tag \cup Att$ is used to denote the ***#PCDATA*** content model, (2) given $n \in N_T$, $\lambda_T(n) \in Att \cup \{S\} \Leftrightarrow n \in Leaves(T)$, (3) $\delta$: $Leaves(T) \mapsto Str$ is a function associating a string to a leaf node of $T$.

An *XML path p* is a sequence $p = s_1.s_2.\ldots .s_m$ of symbols in $Tag \cup Att \cup \{S\}$. Symbol $s_1$ corresponds to the tag name of the document root element. An XML path can be of two types: *tag path*, if $s_m \in Tag$, or *complete path*, if $s_m \in Att \cup \{S\}$. We denote as $P_{XT}$ the set of complete paths in $XT$.

Let $XT = <T, \delta>$ be an XML tree, and $p = s_1.s_2.\ldots .s_m$ be an XML path. The application of $p$ to $XT$ identifies a set of nodes $p(XT) = \{n_1, \ldots, n_h\}$ such that, for each $i \in [1..h]$, there exists a sequence of nodes, or *node path*, $np_i^p = [n_{i_1}, \ldots, n_{i_m}]$ with the following properties: (1) $n_{i_1} = r_T$ and $n_{i_m} = n_i$, (2) $n_{i_{j+1}}$ is a child of $n_{i_j}$, for each $j \in [1..m\text{-}1]$, (3) $\lambda(n_{i_j}) = s_j$, for each $j \in [1..m]$.

Moreover, we say that the application of a path to an XML tree yields an *answer*, which is defined depending on the type of path. In case of a tag path $p$, the answer of $p$ on $XT$ is exactly the set of node identifiers $p(XT)$, that is $A_{XT}(p) \equiv p(XT)$. For a complete path $p$, the answer of $p$ on $XT$ is defined as the set of string values associated to the leaf nodes identified by $p$, that is $A_{XT}(p) = \{\delta_T(n) \mid n \in p(XT)\}$ [7][9].

## XML TREE TUPLES

Tree tuples resemble the notion of tuples in relational databases and have been proposed to extend functional dependencies to the XML setting [8][9]. In a relational database, a tuple is a function assigning each attribute with a value from the corresponding domain. According to [8], we provide the following definition:

Given an XML tree *XT* , a tree tuple T is a maximal subtree of *XT* such that, for each (*tag or complete*) path *p* in *XT*, the answer $A_T(p)$ contains at most one element. We denote as $T_{XT}$ the set of tree tuples from *XT*. Intuitively, a tree tuple is a (sub) tree representation of a complete set of distinct concepts that are correlated according to the structure semantics of the original tree. Moreover, tree tuples extracted from the same tree maintain identical structure while reflect different ways of associating content with structure as they can be naturally inferred from the original tree. For example consider the XML tree shown in figure 1, which represents two journal articles from the DBLP (year 2013) archive. Any internal node has a unique label denoting a tag name, whereas each leaf node is labeled with either name and value of an attribute, or symbol ***S*** and a string corresponding to the ***#PCDATA*** content model. Path answers can be easily computed: for example, path ***dblp.article.title*** yields the set of node identifiers $\{n_7, n_{28}\}$, whereas path ***dblp.article.author.S*** yields the set of strings {'E. F. Codd', 'C. J. Date'}. Three tree tuples can be extracted from the example tree (see figure 2). One tree tuple is extracted starting from the left subtree rooted in the *dblp* element. Two tree tuples are instead extracted starting from the right subtree rooted in *dblp*, as in this subtree there are two paths ***dblp.article.author***, each of which yields a distinct path answer corresponding to the articles' author [7][9].

```
<dblp>
 <article mdate="2002-01-03" key="persons/Codd71a">
  <author>E. F. Codd</author>
  <title>Further Normalization of the Data Base Relational Model.</title>
  <journal>IBM Research Report, San Jose, California</journal>
  <volume>RJ909</volume>
  <month>August</month>
  <year>1971</year>
  <cdrom>ibmTR/rj909.pdf</cdrom>
  <ee>db/labs/ibm/RJ909.html</ee>
 </article>
<article mdate="2002-01-03" key="persons/CoddD74">
  <author>E. F. Codd</author>
  <author>C. J. Date</author>
  <title>Interactive Support for Non-Programmers: The Relational and Network Approaches</title>
  <journal>IBM Research Report, San Jose, California</journal>
  <volume>RJ1400</volume>
  <month>June</month>
  <year>1974</year>
 </article>
</dblp>
```

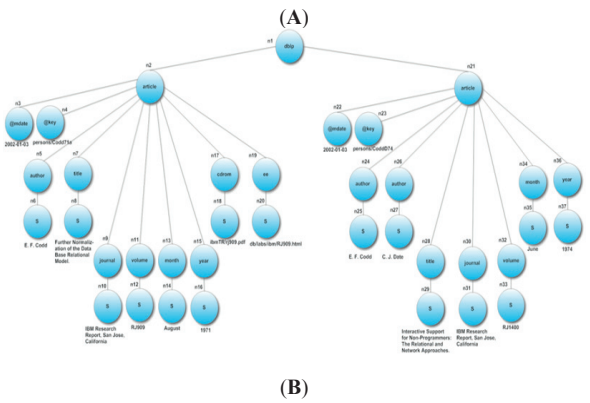



Fig. 1. Example of DBLP XML document and its tree.

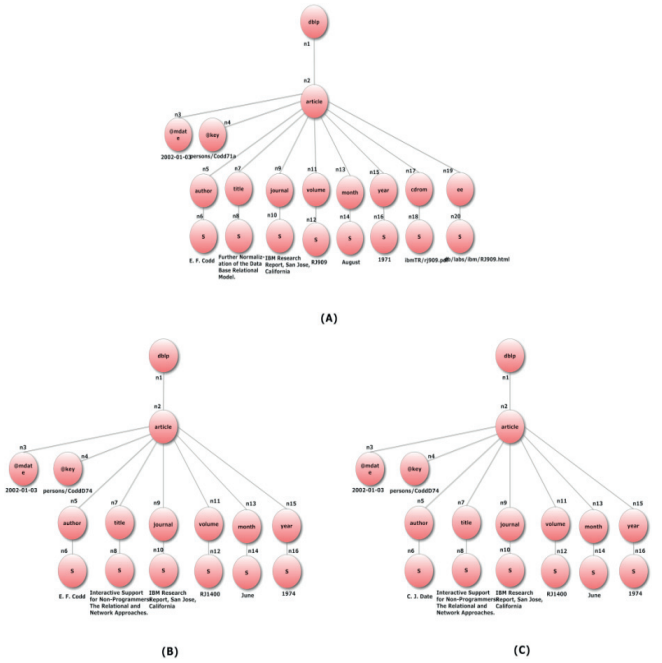



Fig. 2. The tree tuples extracted from the XML tree of figure 1-B.

## XPATH

The XML Path Language, is a query language for selecting nodes from an XML document. In addition, XPath may be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML document. XPath was defined by W3C. The XPath language is based on a tree representation of the XML document, and provides the ability to navigate around the tree, selecting nodes by a variety of criteria. For example, the XPath expression ***/book/chapter/section*** navigates from the root of a document (designated by the leading slash "/") through the top-level "*book*" nodes, to their "*chapter*" child nodes, and on to their child nodes named "*section*". The result of the evaluation of the entire expression is the set of all the "*section*" nodes that can be reached in this manner. Furthermore, at each step in the navigation the selected nodes can be filtered using qualifiers. A qualifier is a boolean expression between brackets that can test the existence or absence of paths. So if we ask for ***/book/chapter/section[citation]*** then the result is all "*section*" elements that have a least one child element named "*citation*" [3][4][5].

## XQUERY

XQuery is a query language designed by the W3C, it's a functional programming language that is designed to query and transform collections of structured and unstructured data, usually in the form of XML. XQuery contains a superset of XPath expression syntax to address specific parts of an XML document. It supplements this with a SQL-like "FLWOR expression" for performing joins. A FLWOR expression is constructed from the five clauses after which it is named: FOR, LET, WHERE, ORDER BY, RETURN (for instance see figure 3)[2].

XQuery has a rich set of features that allow many different types of operations on XML data and documents, including: selecting information based on specific criteria, filtering out unwanted information, searching for information within a document or set of documents, joining data from multiple documents or collections of documents, sorting, grouping, and aggregating data, transforming and restructuring XML data into another XML vocabulary or structure, performing arithmetic calculations on numbers and dates and manipulating strings to reformat text [1].

```
<textbooks>
    {
      for $book in doc("bookstore.xml")//book
      where $book/@cat="textbook"
      return <textbook isbn="$book/@isbn">{$book/title}</textbook>
    }
</textbooks>
```

Fig.3. Example of XQuery.

## SEDNA

Sedna is a free native XML database which provides a full range of core database services persistent storage, ACID transactions, security, indices, hot backup. Flexible XML processing facilities include W3C XQuery implementation, tight integration of XQuery with full-text search facilities and a node-level update language. The Sedna client application programming interfaces (APIs) provides programmatic access to Sedna from client applications developed in host programming languages. The Java API provides programmatic access to XML data from the Java programming language. Using the Java API, applications written in the Java can access one or more databases of the Sedna DBMS and manipulate database data using the database languages (XPath or XQuery) [6].

## EXPERIMENT

In our experiment, a laptop with specification in table I has been used, 50 MB XML database has been loaded to Sedna server and Sedna's java API which provides programmatic access has been used to access XML data from NetBeans IDE (7.4 Beta).

Table. 1. System specifications.

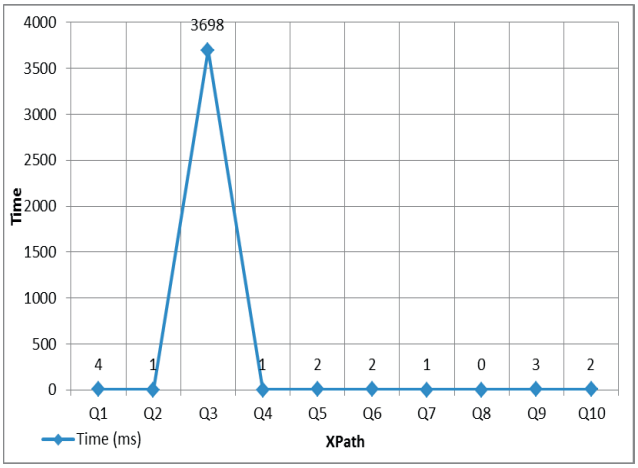
| Specifications | Details |
|---|---|
| Processor | Intel ® Core ™ i5-2430 M CPU @ 2.40 GHz |
| RAM | 6 GB |
| Hard Disk | 700 GB |
| Operating System | Windows 7 Home Premium, Service Pack 1 (64-bit) |
| Sedna | Version 3.5 |

## XPaths

XPath is the XML Path Language that used for selecting or navigating nodes from an XML document using queries. Ten different types of XPath queries has been used (see Table II) to measure query's time and number of replayed nodes. From result in table 2 and figure 4, we find out that the time has taken to answer query is depending on the number of replayed nodes, position of node in tree and the query itself, as example $Q_3$ it takes much more time than the others, because $Q_3$ has to select all descendent nodes of the current node that match the selection and satisfy the condition.

Table. 2. XPath result.

| Query | XPath Expression | Query answer | |
|---|---|---|---|
| | | Number of Replayed Nodes | Time (ms) |
| $Q_1$ | /dblp/article/author[3] | 44862 | 4.0 |
| $Q_2$ | /dblp/article[author[3]]/pages[last()] | 42133 | 1.0 |
| $Q_3$ | //phdthesis[@mdate='2012-04-18'] | 6020 | 3698.0 |
| $Q_4$ | /dblp/phdthesis/url | 3615 | 1.0 |
| $Q_5$ | /dblp/inproceedings/* | 16 | 2.0 |
| $Q_6$ | /dblp/inproceedings[year>=1974] | 2 | 2.0 |
| $Q_7$ | /descendant::book | 1 | 1.0 |
| $Q_8$ | //book/isbn | 1 | 0.0 |
| $Q_9$ | /dblp/www[author and year] | 7 | 3.0 |
| $Q_{10}$ | /dblp/www[author or year] | 17 | 2.0 |




XPath Queries and replayed time.

## XQueries

XQuery is a functional programming language that is used to query a tuple (group) of XML tree. XQuery uses XPath syntax for addressing different parts of an XML document. In this part, XQueries have used the XPath queries which illustrated in *section A* to cluster and tuple XML tree. From result in Table III and figure 5, we find out a proportional relationship between the time taken to answer the query and the size of resulted subtree (depending on the tree tuples of XML tree and how it's been clustered).

Table. 3. XQuery results.

| Query | XQuery Expression | Time (ms) |
|---|---|---|
| $Q_1$ | <dblp> { let $author := doc('article')//dblp/article/author[3] return <article>{$author}</article>} </dblp> | 507.0 |
| $Q_2$ | <dblp> { for $pages in doc('article')//dblp/article[author[3]]/pages[last()] return $pages} </dblp> | 461.0 |
| $Q_3$ | <dblp> { for $phdthesis in doc('article')//phdthesis where $phdthesis[@mdate='2012-04-18']return $phdthesis} </dblp> | 303.0 |
| $Q_4$ | <dblp> { for $url in doc('article')//dblp/phdthesis/url return $url} </dblp> | 21.0 |
| $Q_5$ | <dblp> { for $inproceedings in doc('article')//dblp/inproceedings/* return $inproceedings} </dblp> | 7.0 |
| $Q_6$ | <dblp> { for $inproceedings in doc('article')//inproceedings, $year in $inproceedings/year where $inproceedings/$year/(text()>=1974) return <inproceedings>{$inproceedings/*}</inproceedings>} </dblp> | 5.0 |
| $Q_7$ | <dblp> { for $book in doc('article')//descendant::book return $book} </dblp> | 6.0 |
| $Q_8$ | <dblp> { for $book in doc('article')//book return <book>{$book/isbn}</book>} </dblp> | 2.0 |
| $Q_9$ | <dblp> { for $www in doc('article')//www where $www[author and year]return <www>{$www/*}</www>} </dblp> | 4.0 |
| $Q_{10}$ | <dblp> { for $www in doc('article')//www where $www[author or year]return <www>{$www/*}</www>} </dblp> | 5.0 |

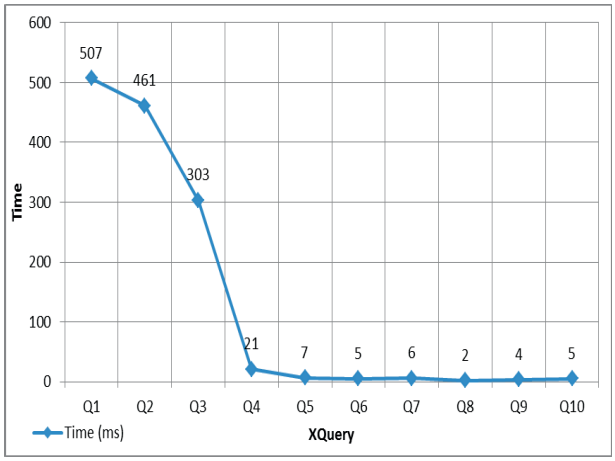



Fig. 5. XQueries and replayed time.

## CONCLUSION

The ontology knowledge to derive semantics of XML document has become a major challenge in semi-structured data management. XPath and XQuery languages used for selecting nodes and compute values from XML document, tuple the XML tree and address the problem of XML data clustering according to structure with ontology knowledge. So this paper proposed tree tuples using XPath and XQuery over Sedna XML database.

XPath is a regular expression, a filter for an XML tree and it is the transformational component of XSLT. XQuery is a programming language used to select several nodes from an XML document for the purpose of processing using different queries, XQuery uses XPath syntax for addressing different parts of an XML document. The joins are performed using the FLWOR expression. This expression has five clauses, namely, WHERE, ORDER BY, FOR, LET, and RETURN.

From *section A* we found that the time taken to answer query was depended on the number of replayed nodes, position of node in the tree and the query expression itself (structure of expression). From *section B* we found a proportional relationship between the time taken to answer the query and the size of resulted subtree (depended on the tree tuples of XML tree and how it's clustered).

## REFERENCES

[1] Priscilla Walmsley, "XQuery", April 2007.

[2] W3C, "XQuery 3.0: An XML Query Language", October 2013, http://www.w3.org/TR/xquery-30/

[3] Bergeron, Randy, "XPath-Retrieving Nodes from an XML Document", October 31, 2000.

[4] Pierre Geneves, "Course- The XPath Language", October 2012.

[5] Pierre Geneves, "Logics for XML", December 2006.

[6] Sedna, http://www.sedna.org/

[7] Andrea Tagarelli, Sergio Greco, "Toward Semantic XML Clustering".

[8] M. Arenas and L. Libkin, "A Normal Form for XML Documents", ACM Trans. Database Systems, 29(1):195–232, 2004.

[9] S. Flesca, F. Furfaro, S. Greco, and E. Zumpano, "Repairs and Consistent Answers for XML Data with Functional Dependencies", In Proc. Int. XML Database Sym-posium (XSym), pages 238–253, 2003.