Query Optimization against XML Data

Nicoleta Liviana TUDOR

Petroleum-Gas University of Ploiesti,
Department of Informatics, Information Technology, Mathematics and Physics,
39, Bucuresti Blvd., Ploiesti, 100680, Romania,
Tudor.Liviana@gmail.com

Abstract: Web services allow middleware access to a relational database and require data representation in XML format. The XML views obtained from relational databases can be accessed by using XPath queries. This article proposes an optimization model for XML data processing based on a heuristic algorithm to extract data from XPath views. To this end, the author uses various XPath query classes temporarily stored in cache, as XPath views. For each view selected from cache, a compensation query can be found and composed with in order to solve an XML data query. Experimental results reveal the effectiveness of the heuristic method used to solve queries on XML documents.

Keywords: cache, heuristic algorithm, relational databases, query processing, XML data.

1. Introduction

The distributed systems use business object or XML web services to share data between applications, ensuring the platform and programming language independence. The Web services architecture involves the existence of a few layers, protocols and related technologies like XML, SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language).

Various XML data storage approaches for relational databases recommend the use of the generic relational structure, including XML document mapping. Kossman [1] has represented XML documents using graphs.

The ideas of memorizing information related to each node of the tree in an XML document has been developed by Yoshikawa, Amagasa and others [2]. The algorithm for XML data translation proposed by Yoshikawa and Amagasa is only appropriate for nonrecursive data and fails to obtain accurate results if the XML data has ascendants with the same label in the tree representation.

The relational model allows the elaboration of translation algorithms for the XQuery queries into SQL queries, where XQuery is a standard XML data interrogation language. For example, Oracle allows the creation of XML views for relational data, and the interrogation of XML views can use the XPath language.

Multiple queries optimization has been expressed in several contexts in the recent past including transient views [3], view maintenance [4], XML query optimization and continuous query optimization.

Tudor proposes a cache pattern with multiqueries and describes the multi-query optimization with scheduling, caching and pipelining. A set of cache patterns is derived from a set of class of multiqueries that are loaded into the cache [5].

A semantic cache memorizing XML views can be used to optimize business objects. To avoid repeated connections to a backend database, the views stored in cache are interrogated. This type of middle-tier cache has become very popular for Web applications with relational databases. Semantic cache uses the views' semantics to determine if the queries can be solved with the cache information entirely or partially [6], [7].

The contributions to this article can be summarized as follows:

1. a method to optimize access to XML data has been identified and it is based on the extraction of XPath views from a semantic cache.
2. a new solving technique has been proposed for the XPath queries. Thus, this paper offers:
   a. a definition for the XPath query classes for which the XPath view heuristic extraction algorithm is evaluated;
   b. an effective method to select an XPath view from cache based on the constraint satisfaction verification for the XPath expressions;

3. the heuristic solving techniques for the queries described above have been implemented for relational Oracle databases. The complexity analysis stands as proof for the performance of the proposed algorithm. The time complexity will be evaluated according to the size of the input space represented by the set of XPath views. The experimental results prove the feasibility and effectiveness of the newly proposed heuristic algorithm.

The article is organized as follows. Section 2 describes the way to process XML queries and the conversion of XQuery queries in relational databases. Section 3 describes the issue of XML data rewriting using XPath views in Oracle databases. The author presents the composition of XPath queries and the creation of an XPath views' cache. Section 4 describes the HSelectXP heuristic algorithm for special XPath query classes. Section 5 describes a complexity evaluation for the heuristic algorithm. Section 6 presents the experimental results, comparisons against other known algorithms and the way to process queries using the heuristic algorithm. The experimental study emphasizes the performance of the heuristic algorithm in processing XML query data. Section 7 draws-up the conclusions regarding the effectiveness of a semantic cache and the heuristic algorithm in optimizing access to XML data.

2. Processing XML Data

Generally speaking, processing XML queries involves compiling XML documents and creating an XML DOM (Document Object Model) tree with nodes memorizing elements, attributes and text. The compiler generates a view based on the transmitted parameters, mapping the XML documents' nodes in the lines of the created view.

The XML views obtained from the relational data use XDR (XML Data Reduced) schemes and can be accessed using XPath queries. XPath language (XML Path Language) allows the interrogation of data in an XML view (Figure 1).

An XPath query selects a set of nodes in the XML DOM graph associated to the XML document, using access control operators and rules [8], [9].

In some cases, there is a compatibility issue between the language used to update XQuery views and the SQL language, for relational databases. Some XQuery processors don't allow for XQuery instruction conversion to SQL, during data transmission to the client application or the conversion of XML documents.

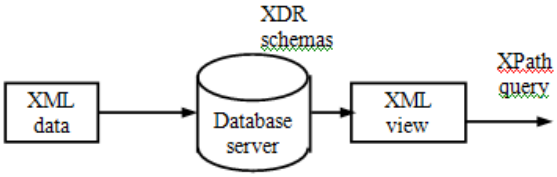

Figure 1. Xpath queries

In the following paragraphs we will analyze the possibility to optimize data native XML data access from Oracle databases, using a semantic cache for XPath views.

The XQuery optimizations can be divided into two broad areas. Logical optimizations are transformation of the XQuery into equivalent SQL query modeling XQuery semantics. Physical optimizations are transformation of the XPath operators, into equivalent operations directly on the underlying internal storage and index tables.

In the following paragraphs we will analyze the possibility to optimize data native XML data access from Oracle databases, using a semantic cache for XPath views.

3. Semantic Cache For XPath Views

Query optimization at the level of the server database, but also at the level of the client application performing semantic caching determines the occurrence of the rewrite issue for the XPath queries using XPath views.

In the following paragraphs, we will present an optimization method for XPath queries through the use of a heuristic algorithm which extracts XPath views from a semantic cache. To avoid repeated connections to the database, the views stored in cache can be interrogated.

Mandhani and Suciu proposed a method to create a semantic cache which memorizes frequently used XPath views for query processing [10]. XQuery queries contain XPath fragments, and the views materialized in cache contain XPath expressions. For the execution of a query, the system first checks whether or not it can return the result of the query from the cache.

Mandhani and Suciu have stored the view cache in relational tables, demonstrating the effectiveness of the view selection technique.

The semantic cache proposed by Lee and Wesley Chu [11] consists of a Hash table with type data (key, value), where key is a semantic description of the query performed, and the value contains the results for the query associated with the key. In this type of cache, semantic views only use conjunctive predicates, the queries being turned into conjunctive components.

Processing queries using an XPath view semantic cache involves finding a query which, by composition with the view in cache, will return the result of a query.

3.1 Rewriting XPath views

The XPath expression is evaluated against the XML document without ever constructing the XML document in memory. This optimization is called XPath rewrite [12]. For example, Oracle XML DB can optimize queries that use XPath (XQuery) expressions. The result of the queries was stored in cache as materialized views.

XML data allow rewriting equivalent to using XPath view internal structure navigation [13]. XML data can be represented as a non-oriented tree with a U set of edges, an X set of nodes, an r root and an f function labelling the tree nodes.

3.2 The creation of an XPath view cache

Let's consider a materialized V view and a Q XPath query which needs processing. The composition of the queries involves the existence of a C query which, by composition with a V view from cache, will return the result of a Q query observing relation

$$C \circ V = Q \quad (1)$$

In the context of the composition of a query

using the XPath view cache, we will describe the operation of inserting an XP view in cache, the selection of a view from cache and we will present the situation in which a query cannot be solved from cache, even if the results are stored in cache.

To insert a view in cache, we will consider the XPath views Vi, $1 \le i \le n$ and the filtration and validation constraints for the XPath expressions p(1), p(2), and so on. The description of a semantic cache with n XPath views can be the following [14]:

$$semantic\ cache := \{Vi\ /Vi:/a/b[p(j)]\ [p(k)]\}, \quad (2)$$

where a and b are XML elements of view, $1 \le i \le n$; $j, k \ge 1$.

Inclusion of XPath expressions $p(i) \le p(j)$, $i, j \ge 1$, shows that the set of nodes resulted from the evaluation of p(i) is included in the set of nodes selected by the p(j) expression. An XPath expression is considered invalid when the set of nodes evaluated is always void.

Lee and Wesley Chu [11] have stored the semantic cache in Hash tables with input data of type (key, value), where the key is a semantic description based on the precedent queries and the value contains results of queries associated to the key.

In the following paragraphs we will introduce a heuristic method to effectively solve XML data queries.

4. Heuristic Selection Algorithm For XPath Views In Cache

Suppose the system has cached the results of a very large number of XML views. When the system needs to evaluate a new XPath query Q, one possibility is to iterate over the views one by one and use a query-answering algorithm.

To optimize XML data queries from Oracle databases, we will create a semantic cache for XPath views. We will propose a *HSelectXP* heuristic algorithm which selects a materialized V XPath view in cache in order to process an XPath query. For each view selected from cache, a C compensation query can be found, composing the view according to the relationship: $C \circ V = Q$. The algorithm will be checked for two classes of XPath queries which will be hereafter defined.

4.1 XPath query classes

XPath queries can contain child nodes marked with the symbol / and nodes representing descendant subtrees marked with the symbol //. The predicates used for data filtering will be marked with [ ], and the symbol * will be used to substitute a descendant node. Thus, it can be safely assumed that the tree formalization of an XP (XPath) query uses the representation XP{/,//,*,[]}.

Two classes of XP queries shall be taken into account: XP{/,//,[]}, XP{/,*,[]}. The models XP{/,//,*,[]} can be defined to represent the trees associated to some XPath query classes, for a set of constraints of the XPath expressions. Constraints of XPath expressions p(1), p(2), and so on, can be used for selection of nodes of XML trees needed for rewriting XPath views [14].

The existence of a morphism on the group of XP models offers the possibility of equivalent rewriting of XPath views, using transformations of models of trees associated to XP queries from cache, under filtering constraints of XPath expressions, in XP{/,//,*, []} representation [15].

4.2 Heuristic algorithm description

The *HSelectXP* heuristic algorithm selects an XPath view from cache to quickly process an XPath query. For each view selected by the heuristic algorithm, a compensation query can be found, to which it is composed to supply the results of a data query. The query can be formed by the difference of the set of constraints of the initial query and the selected view.

```
procedure HSelectXP(V, n)
   Input: Q, a XPath query
   Output: V, a XP view from cache if
   exist; otherwise null
   let be: ViXP views, 1 ≤ i ≤ n
   let be: Y, the set of constraints of
   query Q
   for i = 1, n
       call Choose(V, i, X)
       call Possible(X, sw)
       if sw = 1 then store(V, i)
       endif
   repeat
return(V)
```

The *HSelectXP* heuristic algorithm works as follows. In the procedure *HSelectXP*, the *n* view set from cache is used. The *Choose* procedure supplies the X set (the set of constraints for the Vi view in cache). The *Possible* procedure returns in the variable sw the value 1 if X is a possible solution (if X is included in Y) and returns 0 otherwise.

There may be cases in which a query cannot be processed using XPath views from cache. As an example, let's consider a *Q* query, two XPath views, V_1 and V_2 stored in cache and the XPath expression validation constraints p(i), where i>0:

Q: /a/b [p(1)] [p(2)]/c

V1: /a/b[p(3)]
V2: //b[p(4)]

The results of the *Q* query contain attributes included in the views V_1 and V_2, but the constraints [p(1)] and [p(2)] (for node b) do not correspond to the views V_1 and V_2.

5. Complexity Analysis

In order to evaluate the complexity of the *HSelectXP* heuristic algorithm, two criteria could be taken into account: (a) the time required to run the algorithm (given by the number of elementary operations required to select an Xpath view from cache) and (b) the amount of memory that the algorithm requires. The time complexity has no connection to the specifications of the machine which is running the algorithm and represents the main criterion for the analysis.

n is given as the number of input data for the *HSelectXP* heuristic algorithm. We note that *n* equals the number of memory locations required to store the XP views from cache, flagged as V_i, i=1,…, n. We note $T_V(n)$ as the time required by the algorithm for a *V* set of n input data and *T(n)* as the time required by the algorithm in the worst case scenario, thus:

T(n) = sup{$T_V(n)$, where V is a set of input data with the size n}

We will study the behaviour of T(n) where n has higher values. We will determine the upper end of the running time using the *O* order of an *f* function which increases at least as slow as *T*, compared to the *n* size of the input data so that:

$$T(n) \sim f(n) \Leftrightarrow \lim_{n \to \infty} \frac{T(n)}{f(n)} = const > 0 \qquad (3)$$

and

$$T(n) = O(f(n)) \Leftrightarrow \exists C > 0, \; n_0 \in N, \; T(n) \le C f(n), \; \forall n \ge n_0$$

The evaluation of T(n) for the *HSelectXP*heuristic algorithm is based on the n times repetition of the call for the procedures *Choose* and *Possible*. The procedure *Choose* required a constant amount of time. The running time for the procedure *Possible*

depends on the complexity of the XPath query for each i step and on its comparison to the initial query. The average number of comparisons is

$$T(n)=\sum_{i=1}^{n} p_i nr_i \quad (4)$$

where p_i represents the probability that the V_i XPath view is chosen and processed, and nr_i represents the number of constraints associated with the query of V_i. Data uncertainty is frequently modeled as a probability distribution over possible data values [15].

The probability for an XPath view to be looked up and processed can be expressed in the following manner [16]:

$$p_i=\frac{number\ of\ searches\ of\ i\ view}{total\ number\ of\ searches} \quad (5)$$

We are dealing with the case of a successful search: if we assume that the search probability of a view is the same as the search probability of other views, then the average number of comparisons is:

$$T(n)=\sum_{i=1}^{n} \frac{i}{n}\cdot nr_i \leq \leq \frac{nr(1+2+\ldots+n)}{n}=\frac{nr(n+1)}{2} \quad (6)$$

where $nr = sup\{ nr_i / nr_i$ represents the number of constraints associated with the query of V_i, $i=1,..., n\}$. Should we choose a polynomial function f of 1^{st} degree. It means that T increases as slow as the function f, where $f(n) = n$. Thus the time complexity of the heuristic algorithm is $O(n)$.

6. Experimental Evaluation and Comparisons

The experimental study is performed on an AMD Athlon 1.9Ghz processor with 2GB RAM. The database is implemented using Oracle Database 10g Express Edition. The use of cache in an Oracle database is hereafter described. In the first stage, an XML type data table is created and an XML data benchmark is inserted in it, having a size of 462Kb. Two classes of frequently used XP queries shall be taken into account: XP{/,//,[]}, XP{/,*,[]}. Then, a cache of views corresponding to the stored query classes is created.

Heuristic cache shall hereafter represent the cache of XPath views implemented in the Oracle database which uses a cursor type temporary table and memorizes the constraints of the XPath expressions applied to the XML data table. The heuristic algorithm which selects an XPath view from cache is implemented in the PL/SQL Oracle programming language.

To evaluate the performance of the XML view cache, a comparative analysis of the execution time shall be performed for the queries done with the SELECT command and the queries done using two types of caches (heuristic cache and naive cache).

6.1 Comparisons to other algorithms

We will compare our heuristic cache with semantic cache, naive semantic cache and with algorithm without cache. A naive semantic cache (called the Naive Cache in our experiments) is based on matching of query strings.

A semantic cache of XQuery views provides solutions for the query and view matching problem [17]. For XQuery views, it will obtain smaller cached results and rewritten queries, which will increase cache hits.

An effective semantic cache based algorithm uses a Greedy approximation to look up the result of a query in cache and describes heuristics to improve the selection of a view from cache [10]. The complexity of the algorithm based on query caching and view selection depends on the operations required to generate a potential query, for the operation of selecting a view based on the template of a XPath query.

The method for maintaining a semantic cache of material-ized XPath view describes some heuristics to improve the view selection, and obtain a higher cache hit rate.

Answerability checking based on matching of query strings requires comparing operations between the tree patterns of the query and view.

6.2 XML data benchmark

The Benchmark sets of data offer the possibility of performing standard tests for real applications. The large sizes of the Benchmark sets of data represent a reason to study the performance parameters in the database operations.

The experimental study uses a benchmark which stores information on the scientific papers registered in the SIGMOD database. The document "SIGMON.xml" has a structure presented in a tree form in Figure 2.

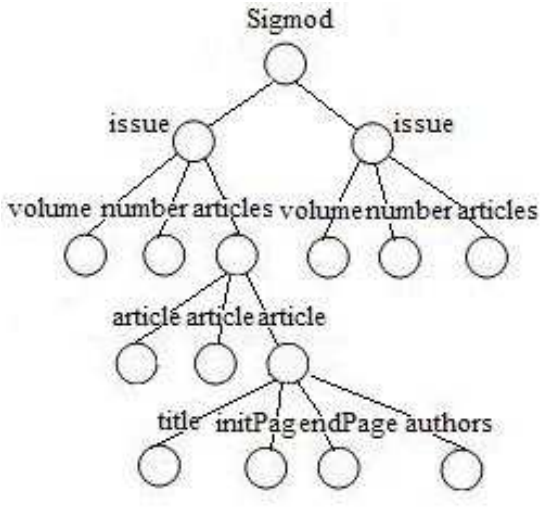

Figure 2. SIGMOD structure

6.3 Selection of XPath view from cache

Let's consider four types of XPath query for the SIGMOD.xml document and the corresponding views memorizing query results in cache.

V1: //SigmodRecord/issue/volume
V2: //SigmodRecord/issue
[volume=12]/articles/article/title
V3://SigmodRecord/issue[volume=11]
[number=3]/articles/article/*
V4: //SigmodRecord/issue[volume=14]
[number=1]/articles
V5://SigmodRecord/issue[volume=11]/
/number,SigmodRecord/issue[volume=11]
//articles

Let's also consider a workload with XPath queries mentioned as Qi (i>1), which are checked for solvability using a view in cache. If a query cannot be processed using a view in cache, then a new view is added to the cache.

An example of query which can be solved using views in cache is the next one:

Q1: SELECT extract(OBJECT_VALUE,
'/SigmodRecord/issue[volume=11]
[number=1]//articles') FROM references;

Query Q_1 is related to views V_3 and V_5 (results of query Q are included in views V_3 and V_5), but the constraint *[number=3]* does not contain *[number=1]*. Therefore only view V_5 returns results for query Q, and query C, to whom it composes is:

C://[number=1]/articles

This is a demonstration that to process the query Q, the cache of XPath views may be used, because there is a query C and a view V5 materialized in cache thus CoV5 = Q.

6.4 Heuristic algorithm performance

We will consider a cache comprised by XPath views corresponding to the query classes defined above. To study the performance of the heuristic cache the hit rate and average processing duration for XPath queries parameters are used.

The average query time for the XPath views is presented in Table 1. For each XPath query, the heuristic algorithm implemented in PL/SQL Oracle selects a view which is appropriate for processing. Table 2 presents the average

Table 1. Query duration for views in cache

XPath views	V1	V2	V3	V4	V5
The average processing duration (5 queries) in seconds	0.135	0.146	0.122	0.236	0.230

Table 2. The average processing duration for XPath queries

Processing method	**The average query processing duration in seconds (10 executions)**						
	Q1	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**
no cache	0.17	0.2	0.12	0.18	0.15	0.13	0.131
heuristic algorithm	0.02	0.07	0.01	0.06	0.02	0.01	0.019
the view selected by the algorithm	V5	V2	V1	V1	V5	V4	V1
the average duration	0.03	0.01	0.01	0.01	0.01	0.01	0.02

processing duration resulted from the composition of the cache view selection operation and processing of the selected view.

The comparative analysis of the XPath queries' processing methods emphasizes the effectiveness of the proposed heuristic algorithm, as compared to the solving method for the queries without using cache, by using a database (Figure 3).

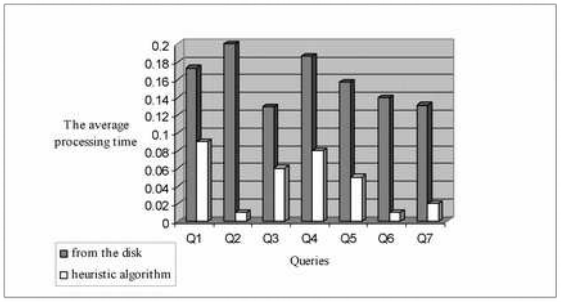

Figure 3. The comparative analysis of the average processing duration for XPath queries.

6.5 Effectiveness of the heuristic cache

Cache effectiveness can be statistically determined, by calculating the hit rate (of the performance), which is dependent on the view cache implementing method and represents the percentage of queries satisfied by the cache. For example, the H hit rate is calculated as a percent as follows: from 100 accesses, how many times the data is found in cache.

We will compare our heuristic cache with a naive semantic cache. We will consider two workloads with sizes of 7 and 15 XPath queries and the hit rate for each type of cache studied shall be determined (Figure 4). Figure 5 presents the query processing time using three methods: with the proposed heuristic cache, with the naive cache and with no cache.

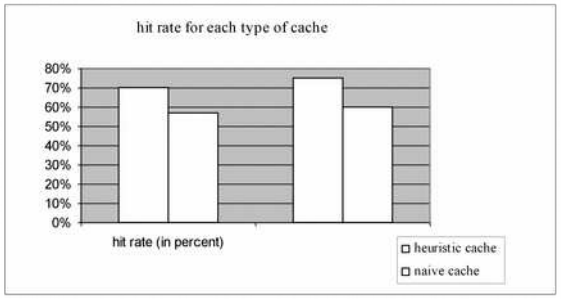

Figure 4. Hit rate for two types of cache.

The effectiveness of the heuristic cache proposed is emphasized by comparing the values for hit rate and average processing duration for the queries obtained in the case of the heuristic algorithm, with the ones obtained in the case of the naive cache and, respectively, with the values obtained with no cache.

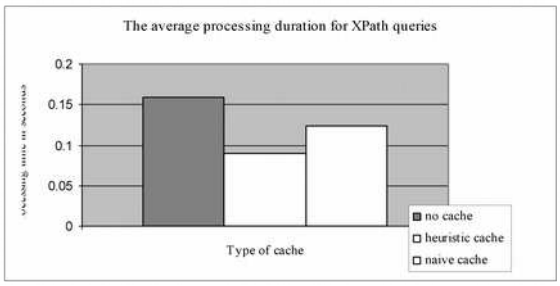

Figure 5. The average processing duration for XPath queries.

7. Conclusions

In this article, an optimization method has been defined for XML data access, which is based on a heuristic algorithm for selecting an XPath view from cache. The use of cache stored XPath views allows avoiding repeated access to database data and optimizes XPath query processing. The author describes a heuristic algorithm for XP{/,//,*,[]} query classes which select an XPath view stored in cache to solve a very large XML data query. For each XPath view, a compensation query can be found, and it helps to supply the result of an XPath query. The complexity analysis stands as proof for the performance of the proposed algorithm.

The experimental study uses a Benchmark with XML data extracted from the SIGMOD database. The XPath view cache is stored in an Oracle database as temporary table. To study the performance of the heuristic cache the hit rate and average processing duration for XPath queries parameters are used. The comparative analysis of the two types of cache (naive cache and heuristic cache) emphasizes the effectiveness of the heuristic algorithm in processing XML data queries.

REFERENCES

1. KOSSMAN, D., D. FLORESCU, **Storing and Querying XML Data using an RDBMS**, IEEE Data Eng. Bulletin, 1999.
2. YOSHIKAWA, M., T. AMAGASA, T. SHIMURA, S. S. UEMURA, **Xrel: A Path-based Approach to Storage and Retrieval of XML Documents using Relational Databases**, ACM Trans. on Internet Tech., no. 1, 2001, pp. 110-141.

3. SUBRAMANIAN, S. N., S. VENKATARAMAN, **Cost based Optimization of Decision Support Queries using Transient Views**, In ACM SIGMOD International Conference Management of Data, Seattle, WA, 1998.

4. MISTRY, H., P. ROY, S. SUDARSHAN, K. RAMAMRITHAM, **Materialized View Selection and Maintenance using Multi-query Optimization**, In ACM SIGMOD International Conference on Management of Data, 2001.

5. TUDOR, N. L., **Cache Pattern with Multi-Queries**, Advances in Electrical and Computer Engineering, Faculty of El. Engineering and Computer Science, Stefan cel Mare University of Suceava, Romania**,** vol. 10, nr. 2, 2010, pp. 84-88.

6. ARINBJARNAR, M., B. PORSSON, B. P. JONSSON, **Performance of Semantic caching Revisited**, Technical Report RUTR–CS06002, Reikjavik University Iceland, 2006.

7. LUO, Q., S. KRISHNAMURTHY, C. MOHAN, H. PIRAHESH, H. WOO, B. LINDSAY, J. NAUGHTON, **Middle-tier Database Caching for e-Business**, In Proceedings of the ACM SIGMOD International conference on Management of data, 2002, pp 600 - 611.

8. DAMIANI, E., S. D. C. DI VIMERCATI, S. PARABOSCHI, P. SAMARATI, **Design and Implementation of an Access Control Processor for XML Documents**, Computer Networks, Amsterdam, Netherlands, 1999, vol. 33(1–6), 2000, pp. 59-75.

9. FUNDULAKI, I., M. MARX, **Specifying Access Control Policies for XML Documents with XPath**, In The ACM Symposium on Access Control Models and Technologies (SACMAT), ACM Press, 2004, pp 61-69.

10. MANDHANI, B., D. SUCIU, **Query Caching and View Selection for XML Databases**, Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005.

11. LEE, D., C. W. WESLEY, **Semantic Caching Via Query Matching for Web Sources**, In ACM Proceedings of the 8th International Conference on Information and Knowledge Management, USA, 1999.

12. TANG, J., S. ZHOU, **A Theoretic Framework for Answering XPath Queries using Views**, Third International XML Database Symposium, XSym, Trondheim, Norway, 2005.

13. Balmin, A., F. Özcan, K. S. Beyer, R. Cochrane, H. Pirahesh, **A Framework for using Materialized XPath Views in XML Query Processing**, In VLDB, 2004.

14. TUDOR**,** N. L., **Models XP for Rewriting XPath Queries**, Studies in Informatics and Control, Bucharest, Romania, vol. 20, no. 2, 2011, pp. 121-128.

15. VASILE, S. L., **Query Optimization on Random Databases**, Studies in Informatics and Control, vol. 23(3), 2014, pp. 257-265.

16. TREMBLAY, J. P., P. G. SORENSON, **An Introduction to Data Structures with Applications**, second edition, McGraw-Hill, New-York, 1984.

17. CHEN, L., S. WANG, E. RUNDENSTEINER, **Replacement Strategies for XQuerycaching Systems**, Data & Knowledge Engineering, vol. 49, no. 2, 2004, pp. 145-175.