International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:1, 2007

# Approaches and Schemes for Storing DTD-Independent XML Data in Relational Databases

Mehdi Emadi, Masoud Rahgozar, Adel Ardalan, Alireza Kazerani, and Mohammad Mahdi Ariyan

***Abstract*—**The volume of XML data exchange is explosively increasing, and the need for efficient mechanisms of XML data management is vital. Many XML storage models have been proposed for storing XML DTD-independent documents in relational database systems. Benchmarking is the best way to highlight pros and cons of different approaches. In this study, we use a common benchmarking scheme, known as XMark to compare the most cited and newly proposed DTD-independent methods in terms of logical reads, physical I/O, CPU time and duration. We show the effect of Label Path, extracting values and storing in another table and type of join needed for each method's query answering.

***Keywords*—**XML Data Management, XPath, DTD-Independent XML Data.

## I. Introduction

EXTENSIBLE Markup Language (XML) is the "lingua franca" for data exchange between inter-enterprise applications [1]. It is a simple and very flexible text format derived from SGML (ISO 8879). XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web. It was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996 [2].

Several XML query languages are proposed including Lorel [12], XML-QL [13], Quilt [14], XPath [15] and XQuery [21].

Among these proposals, XPath is being commonly used and accepted as a standard XML query language. Lee and Chu [17] and Bonifati and Ceri [16] present recent surveys on XML schema and query languages. In order to facilitate the task of querying XML documents, many storage models for XML documents are proposed. Some of those models are: lore [12], Edge [6], Monet [18], XRel [7], XParent [8], ORDPATH [3], ORDPATH+ [11] and DLN [4].

In this work, we have tried to benchmark five DTD-independent approaches including Edge, Edge-Value [6], XRel [7], XParent [8] and ORDPATH+ [11] on a commonly used relational database system. We omit ORDPATH[3] because it shows a poor performance and wrecks the comparisons.

The early works on DTD-independent relational storage of XML data used to apply an approach known as Parent-Child method. This method results in poor performance for determining ancestor-descendant relationships necessary for document reconstructions. The new approach called ORDPATH [3] claims to resolve this problem through some numbering scheme which represents the order of XML nodes and keeps track of sub-tree access paths. This new approach has not yet been evaluated in recent benchmarks [5],[8],[16], [17], [19]. In this work we would present this evaluation.

In this article, first, the current literature is reviewed briefly. Then the methodology and implementations are described and afterward, the benchmark basis is developed and the experimental results are reported. Finally, our conclusions and future works are presented.

## II. Related Work

Recently, efforts in designing XML benchmarks have been reported. Böhme and Rahm [19] proposed the XMach benchmark as a scalable multi-user benchmark for evaluating the performance of XML data management systems. It was also designed to test single/multiple DTD(s) for all documents. Different from XMach, where XML data in the benchmark is document-oriented, the XML Benchmark Project (XMark) [20] proposed a benchmark where the data model represents an auction Web Site. That is, it was designed to concentrate on the core ingredient of the XML benchmark: the query processor and its interaction with the data store. Other XML benchmarks include the XOO7, Michigan and etc.

In addition to design XML benchmarks, some reported studies examine the performance of XML systems systematically. Florescu and Kossmann [6] reported their experimental results using eight XML-QL [13] queries to access XML data stored in a relational DBMS with five different schemas. However, the eight XML-QL queries are rather simple, and no detailed information about the RDBMS is reported.

The study reported in [5] presents a good benchmark on a number of storage methods. It compares three categories of XML databases: native, document-dependent-in-relational and document-independent-in-relational. In our previous work [22], those DTD-independent-in-relational approaches are compared with a new approach in this category, ORDPATH. In this work, we study the effects of path table and compare

Mehdi Emadi, Adel Ardalan, Aliraza Kazerani and Mohammad Mahdi Ariyan are with the Database Research Group, Faculty of Electrical and Computer Engineering, University of Tehran, Tehran, Iran (phone: +98-21-88003323; e-mail: {m.emadi, a.ardalan, a.kazerani, m.ariyan}@ece.ut.ac.ir).

Masoud Rahgozar is with the Database Research Group, Faculty of Electrical and Computer Engineering, University of Tehran, Tehran, Iran, (phone: +98-21-88003323; e-mail: rahgozar@ut.ac.ir).

this new method, called ORDPATH+, with other approaches. Here, a commonly accepted benchmarking scheme, XMark is used. We compare document-independent methods for storing XML data without DTD.

## III. Necessity of this Benchmark

In this work, a number of XML storage methods which have been proposed in recent literature are implemented. Although some benchmarks have reported such comparisons ([5], [8]), we develop ours with the new approach which has been announced in[3].

In [11], the proposed method mainly improves the insertion operation of new XML elements into an existing document. No work has been reported subject to comparing retrieval of data in older methods and this new type of methods concerned with labeling of elements. We try to compare the efficiency of retrievals with different types of queries in these methods.

Additionally, we implement another method, ORDPATH+, which is a development of ORDPATH using path indexes. This improves some types of queries with a polynomial factor. In all these methods, the objective is to store the document structure in one or more relations of a RDBMS. We briefly introduce these methods in the following sections.

## IV. Overview of Different XML Storage Methods

XML documents' data model is a tree-based model, considering elements, attributes and values as nodes and containment relationships as edges.

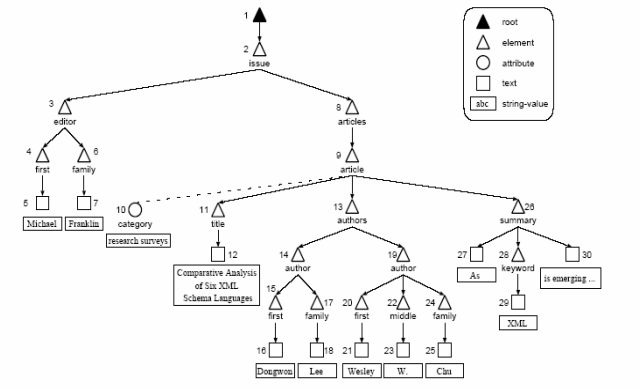


Fig. 1 An XML Tree[7]. All XML file has a tree data model

### A. Edge and Edge-Value Methods

The schemas of the relations [4] used for storing XML data are as shown below.

***Edge (A Single Table Schema)***

Edge(SourceID, TargetID, Ordinal, Label, Flag, Value)

***Edge-Value (A Separate Table For Values)***

Edge(SourceID, TargetID, Ordinal, Label, Flag)

EdgeValue(TargetID, Value)

### B. XRel Method

The schema of the tables [7] used in this method is as shown below.

Path(PthID, PathExp)

Element(DID, PthID, Start, End, Ord)

Text(DID, PthID, Start, End, Value)

Attribute(DID, PthID, Start, End, Value)

This method tries to reduce multiple joins cost using a "Path Index Table" (named Path in the schema). The region of a node is the start and end positions of this node in an XML document. The region implies a containment relationship.

### C. XParent Method

The schema of the tables [8] used in this method is as shown below.

LabelPath(PthID, Len, PathExp)

DataPath(ParentID, ChildID)

Element(PthID, DID, Ord)

Data(PthID, DID, Ord, Value)

Like XRel, XParent uses the Path Index Table, but it uses the Edge and Edge-Value approach to store the parent-child relationship. This relationship is maintained in a separate table (DataPath), again, to reduce the join cost.

### D. ORDPATH+ Method

The schema of the tables [11] used in this method is as shown below.

Node(ORDPATH, Tag, NodeType, Value, PthID)

Path(PthID, PathExp)

ORDPATH+ is just like ORDPATH plus a path index table used to track paths in the original document. The use of path index is like XRel and XParent methods and improves the performance of ORDPATH in some query types.

We need a function to deduce the parent code of each node from its ORDPATH code. As may be seen, it uses string functions multiple times and this leads to a poor performance which is a direct result of the coding schema nature of ORDPATH.

## V. Implementation

We've implemented data loaders for these methods using SAX parser. The benchmark data is then loaded into SQL Server 2000 Personal Edition database.

The test data used in this work is XMark benchmark scheme test data. We use a 127 MB test data file. We use the test data generator available at (monetdb.cwi.nl/xml) with scaling factor of 1.1.

For XRel and XParent, we've implemented XPath-to-SQL translator using Java and ANTLR (parser generator) with XPathCore graph, introduced in [7]. After parsing the XPath query, XPathCore graph which is a general, intermediate representation of the query and is independent of the storage method, is generated. Then for each method, this intermediate representation is translated into relevant method-specific SQL

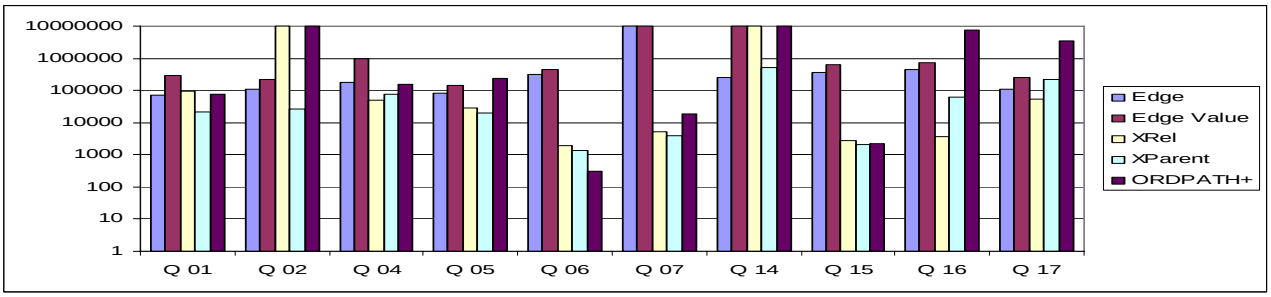


Fig. 2 Read Counts (This diagram shows the number of logical reads or accesses are needed for answering each query in each schema)

query. But in this work for another method, we translate the query manually.

## VI. Benchmark Queries

After loading the data into the tables, we choose ten basic XPath queries for comparing the methods from XPathMark. Then we adopt these queries to the target document.

| | |
|---|---|
| 1 | /site/regions/namerica/item[@id="item20748"]/name/ |
| 2 | /site/open_auctions/open_auction/bidder [1]/increase/ |
| 4 | /site/open_auctions/open_auction[bidder[personref/@person="person18829"]/following-sibling::bidder[personref/ @person="person10487"]]/reserve/ |
| 5 | count(/site/closed_auctions/closed_auction[price >= 40]) |
| 6 | count(/site/regions//item) |
| 7 | count(/site//description\|/site//annotation\|/site//email) |
| 14 | /site/regions/*/item[contains(description,'gold')]/name/ |
| 15 | /site/closed_auctions/closed_auction/annotation/description/parlist/listitem/parlist/listitem/text/keyword/emph/ |
| 16 | /site/closed_auctions/closed_auction[annotation/description/parlist/listitem/parlist/listitem/text/emph /keyword/]/seller/@person |
| 17 | /site/people/person[not(homepage/text())]/name/ |

## VII. Results

We need multiple joins of Edge table in Edge and Edge-Value methods. Because of smaller size of Edge table, Edge-Value does better in these queries. XRel, XParent and ORDPATH generally do better than Edge and Edge-Value methods because of path indexes, which prevent nested join operations.

Fig. 2 shows the final results of our experiments. We discuss about them on a query-based approach. Fig. 3 presents the physical I/O count of different methods on different queries. CPU time results are presented in Fig. 4 and Fig. 5 shows the duration of each query for various methods applied to.

We discuss the read count results in a query-by-query basis. Other metric results are presented for the sake of comparison.

**Q1**- In XParent, join condition is a simple equivalence, but in XRel and ORDPATH it's more complex. For XRel, we use range checking on two fields for join condition. In ORDPATH, string operations (e.g. SUBSTRING) are used for join condition and this leads to inefficiency of using indexes for enhancing the join operation. We examined our experiments with and without indexes and got the (nearly) same results. Although we have these limitations about ORDPATH, because of the narrowing condition in the query ([@id="item20748"]), one of the tables is restricted to a small subset of main table tuples and this leads to an appropriate read count.

Edge method functionality is so similar to XParent as it is an extension of Edge method which uses path index. Restricting the Edge table with conditions of low frequency, the performance of Edge method is better than XRel for this query.

**Q4**- This query is an example of XPath axis functions usage. Implementation of "following-sibling" is straight forward in all these methods. As the condition expression of this query has two levels and there are two such conditions, XParent needs to join two more tables in order to find the result. Instead, XRel has a similar work as for Q1 and this leads to better performance of XRel. ORDPATH, again because of using string functions, shows a worse response time comparing with XRel and XParent.

**Q7**- For "descendant-or-self" conditional expressions ("//"), Edge and Edge-Value need an enormous number of unconditional joins and thus their performance are terrible.

**Q15**- Generally, for queries like Q15 which ask about a simple path, XRel, XParent and ORDPATH get the result with less reads because of using a path index. ORDPATH returns the results with better performance than the XRel, because it stores values in the same table as keys and path identifiers. As a result of range checking for revealing relationships in XRel, XParent works better then XRel. Because of numerous joins in Edge and Edge-Value, their read count is higher than others.

**Q16**- As in ORDPATH we join three tables including values, XRel and XParent work better because they don't store values in join tables (element tables). This is obvious in terms of

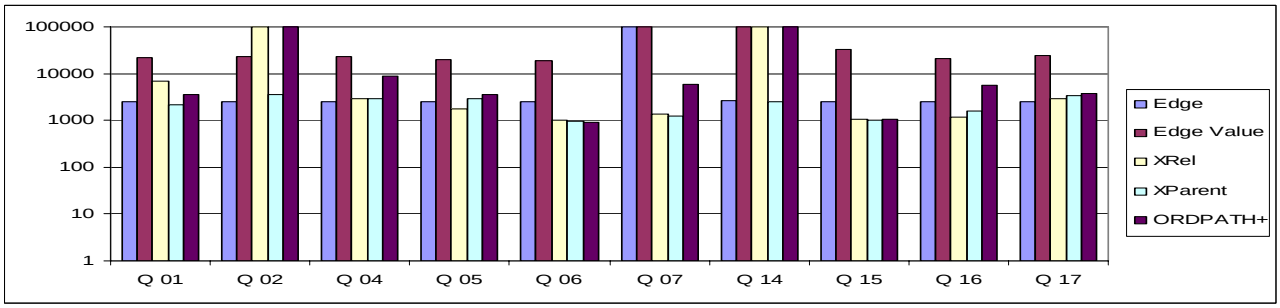


Fig. 3 Physical I/O. (This Diagram shows the number of physical block access needed for executing query in the selected RDBMS)

physical I/O (Fig 3). Edge and Edge-Value have the same problem as Q15.

**Q17-** Same conclusions as Q16 could be applied to this one as a result of high-level of similarity.

## VIII. Conclusion and Future Works

The newly proposed method, ORDPATH, does not seem to improve the read I/O performance in comparison with earlier methods, but ORDPATH+ has the same performance as XRel and XParent methods. After all, it should improve the insert operations rather than read operations. Because of the high cost of substring comparison join operations for the purpose of key matching (prefixing) in order to discover the ancestor/descendant relationships, overall performance of ORDPATH+ is at the same level with XRel and XParent methods. We have used variable character field for ORDPATH code in this benchmark. We are studying the efficiency of using indexes on such type of fields in more details.

Our future work is to compare these methods with DTD-dependant XML data storage approaches. We believe that, enhancing the performance of current methods would need more innovative and intelligent techniques.

Our objective is to yield a new method for storing XML data in relational databases based on the results of our current benchmarking studies.

## Appendix

Here is an example for translating XPath to SQL for all methods in this study:

**Q1: XPath**

/site/regions/namerica/item[@id="item20748"]/name/text().

**SQL for all Methods:**

**Edge:**

```
select t5.value
from EdgeTable t1, EdgeTable t2, EdgeTable t3,
    EdgeTable t4, EdgeTable t5, EdgeTable t6
where t1.TargetID = t2.SourceID
   and t2.TargetID = t3.sourceID
   and t3.TargetID = t4.sourceID
   and t4.TargetID = t5.sourceID
   and t1.tagName = N'site'
   and t2.tagName = N'regions'
   and t3.tagName = N'namerica'
   and t4.tagName = N'item'
   and t5.tagName = N'name'

   and t6.tagName = N'@id'
   and t4.TargetID = t6.sourceID
   and t6.value = 'item20748'
```

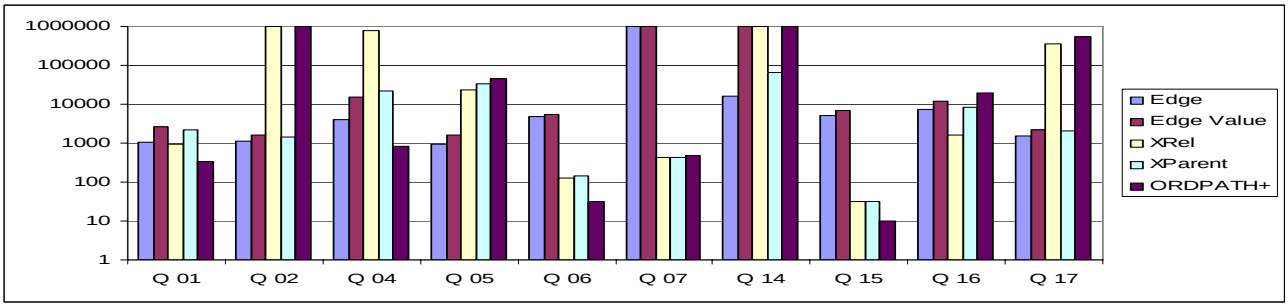


Fig. 4 CPU Time in msec (Amount of CPU time witch is assigned to run process of each query)

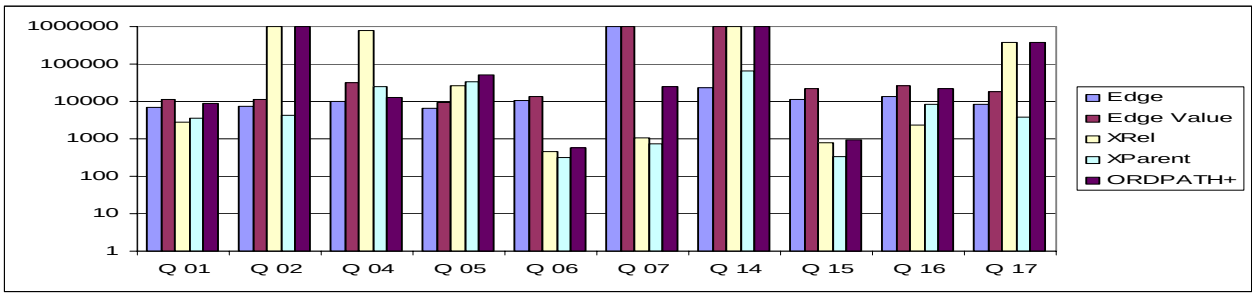


Fig. 5 Duration Time in msec ( Total time of execution of each query)

**XParent :**

```
SELECT distinct d2.value
FROM XParent_ElementTable e1, XParent_DataTable d1,
XParent_DataPathTable dp1,
XParent_ElementTable e2, XParent_DataTable d2,
XParent_DataPathTable dp2, XParent_ElementTable e3
WHERE
p0.pathExpression = './site./regions./namerica./item'
AND e1.pathId_fk = p0.pathid
AND p0_0_0.pathExpression =
'./site./regions./namerica./item/@id'
AND a0_0_0.pathId_fk = p0_0_0.pathid
AND e2.pathId_fk = 249
AND e2.[id] = dp1.elementId_fk
AND e1.[id] = dp1.parentId_fk

AND d1.elementId_fk = e2.[id]
AND d1.value = 'item20748'
AND p1.pathExpression =
'./site./regions./namerica./item./name'
AND e1.pathId_fk = p1.pathid
AND e3.pathId_fk = 253
AND e3.[id] = dp2.elementId_fk
AND e1.[id] = dp2.parentId_fk
AND d2.elementId_fk = e3.[id]
```

**Edje Value:**

```
select v2.value
from EdgeValueEdgeTable t1, EdgeValueEdgeTable t2,
EdgeValueEdgeTable t3,
EdgeValueEdgeTable t4, EdgeValueEdgeTable t5,
EdgeValueEdgeTable t6,
EdgeValueValueTable v1, EdgeValueValueTable v2
where t1.TargetID = t2.SourceID
and t2.TargetID = t3.sourceID
and t3.TargetID = t4.sourceID
and t4.TargetID = t5.sourceID
and t1.tagName = N'site'
and t2.tagName = N'regions'
and t3.tagName = N'namerica'
and t4.tagName = N'item'
and t5.tagName = N'name'
and t6.tagName = N'@id'
and t4.TargetID = t6.sourceID
and v1.nodeid = t6.TargetID
and v1.value = 'item20748'

and v2.nodeid = t5.TargetID
```

**XRel:**

```
SELECT distinct t1.value
FROM XRel_ElementTable e0, XRel_PathTable p0_0_0,
XRel_AttributeTable a0_0_0, XRel_ElementTable e1
,XRel_textTable t1
WHERE
p0.pathExpression = '#/site#/regions#/namerica#/item'
AND e0.pathId_fk = p0.pathid
AND p0_0_0.pathExpression =
'#/site#/regions#/namerica#/item/@id'
AND a0_0_0.pathId_fk = p0_0_0.pathid
AND e0.docid = a0_0_0.docid
AND e0.start <= a0_0_0.start
AND e0.[end] >= a0_0_0.[end]
AND a0_0_0.value = 'item20748'
AND e0.docid = e1.docid
AND e0.start < e1.start
AND e0.[end] > e1.[end]
AND p1.pathExpression =
'#/site#/regions#/namerica#/item#/name'
AND e1.pathId_fk = p1.pathid
AND t1.start > e1.start
AND t1.[end] < e1.[end]
```

**ORDPATH:**

```
select ord3.*
from ORDPATHTable ord1, ORDPATHTable ord2,
ORDPATHTable ord3,
ORDPATHPathTable pt1, ORDPATHPathTable pt2,
ORDPATHPathTable pt3
where --ord1.tag = 'sigmodrecord'
= ord1.ordpathcode
```

```
ord1.pathId = pt1.pathId
and pt1.pathExpression like
 '#/site#/regions#/namerica#/item'
and ord2.pathId = pt2.pathId
and pt2.pathExpression like
 '#/site#/regions#/namerica#/item/@id'
and ord2.nodetype = 2
and ord2.value = 'item20748'
and ord3.pathId = pt3.pathId
and pt3.pathExpression like
 '#/site#/regions#/namerica#/item#/name'
and substring(ord2.ordpathcode, 1, len(ord1.ordpathcode))
= ord1.ordpathcode
and substring(ord3.ordpathcode, 1, len(ord1.ordpathcode))
= ord1.ordpathcode
order by ord3.ordpathcode
```

## ACKNOWLEDGMENT

This work is supported by grants from the TAKFA (National Information and Communication Technology Agenda; Iran).

## REFERENCES

[1] M. Fernndez, Y. Kadiyska, D. Suciu, A. Morishima and W. C. Tan, "Silkroute: A framework for publishing relational data in xml,". ACM Trans. Database Syst. Vol .27, No. 4, pp. 438-493, 2002.

[2] F. Yergeau, T. Bray, J. Paoli, C. M. Sperberg-McQueen and E. Maler, "Extendible Markup Language (XML),". W3C Recommendation, 2004. Available at http://www.w3.org/XML/

[3] P. O'Neil, E. O'Neil, S. Pal, I. Cseri and G. Schaller, "ORDPATHs: Insert-Friendly XML Node Labels,". Proc. ACM SIGMOD. France pp. 903-908, 2004.

[4] T. Böhme and E. Rahm, "Supporting Efficient Streaming and Insertion of XML Data in RDBMS,". Proc. DIWeb. Latvia, pp. 70-81, 2004.

[5] H. Lu, J. Xu Yu, G. Wang, S. Zheng, H. Jiang, G. Yu and A. Zhou, "What makes the differences: benchmarking XML database implementations,". ACM Trans. Internet Techn. Vol. 5, No. 1, pp 154-194, 2005.

[6] D. Florescu and D. Kossmann, "Storing and Querying XML Data using an RDMBS,". IEEE Data Eng. Bull. Vol. 22, No. 3, pp. 27-34, 1999.

[7] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura, "XRel: a path-based approach to storage and retrieval of XML documents using relational databases,". ACM Trans. Internet Techn. Vol. 1, No. 1, pp. 110-141, 2001.

[8] H. Jiang, H. Lu, W. Wang and J. Xu Yu, "Path Materialization Revisited: An Efficient Storage Model for XML Data,". Proc. of ACM Australasian Database Tech. Conf. Australia, pp. 85-94, 2002.

[9] H. Jiang, H. Lu, W. Wang and J. Xu Yu, "XParent: An Efficient RDBMS-Based XML Database System,". Proc. of IEEE ICDE. USA, pp. 335-336, 2002.

[10] E. Cohen, H. Kaplan and T. Milo, "Labeling Dynamic XML Trees,". Proc. of ACM PODS. USA, pp. 271-281, 2002.

[11] S. Pal, I. Cseri, O. Seeliger, G. Schaller, L. Giakoumakis and V. Zolotov, "Indexing XML Data Stored in a Relational Database,". Proc. of VLDB. Canada, pp. 1134-1145, 2004.

[12] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. Wiener, "The Lorel Query Language for Semistructured Data,". Int. J. on Digital Libraries. Vol. 1, No. 1, pp. 68-88, 1997.

[13] A. Deutsch, M. Fernandez, D. Florescu, A. Levy and D. Suciu "A Query Language for XML,". Proc. of WWW. Canada, pp. 1155-1169, 1999.

[14] D. Chamberlin, J. Robie and D. Florescu, "Quilt: An XML Query Language for Heterogeneous Data Sources,". Proc of WebDB (LNCS). USA, pp. 53-62, 2000.

[15] J. Clark and S. DeRose, "XML Path Language (XPath) Version 1.0,". W3C Recommendation, 1999. Available at http://www.w3.org/TR/xpath.

[16] A. Bonifati and S. Ceri, "Comparative Analysis of Five XML Query Languages,". ACM SIGMOD Record. Vol. 29, No. 1, pp. 68-79, 2000.

[17] D. Lee and W. W. Chu, "Comparative Analysis of Six XML Schema Languages,". ACM SIGMOD Record. Vol 29, No. 3, pp. 76-87, 2000.

[18] A. Schmidt, M. Kersten, M. Windhouwer and F. Waas, "Efficient Relational Storage and Retrieval of XML Documents,". Proc. of WebDB (LNCS). USA, pp. 137-150, 2000.

[19] T. Böhme and E. Rahm, "Multi-User Evaluation of XML Data Management Systems with XMach-1,". Proc. of EEXTT (LNCS). Germany, pp. 148-159, 2003.

[20] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, and R. Busse, "Xmark: A Benchmark for XML Data Management,". Proc. of VLDB China, pp. 974-985, 2002.

[21] S. Boag, D. Chamberlin, M. F. Fernández , D. Florescu, J. Robie, J. Siméon, "W3C XML Query (XQuery),". W3C Candidate Recommendation. 2005. Available at http://www.w3.org/XML/Query/

[22] M. Emadi, M. Rahgozar, A. Ardalan, A. Kazerani and M.M. Arian, "A Comparative Study of DTD-Independent XML Data Storage Approaches,". 11th International CSI Computer Conference (CSICC'06). Iran, pp. 624-628, 2006.