# PACD: A BITMAP-BASED FRAMEWORK FOR PROCESSING XML DATA

Mohammed Al-Badawi[1], Barry Eaglestone[2], Siobhán North[1]
[1] *Department of Computer Science, The University of Sheffield, Sheffield, UK*
*m.badawi@dcs.shef.ac.uk,s.north@dcs.shef.ac.uk*
[2]*Department of Information Studies, The University of Sheffield, Sheffield, UK*
*b.eaglestone@sheffield.ac.uk*

Keywords: XML Processing, XML Update, Mapping, Sparse-Matrix.

Abstract: Current XML/RDBMS storage models and query processing technologies are reviewed in this paper, leading to the identification of query expressiveness and performance limitations. A novel serialized XML query processing framework is proposed to address these. The proposed query processor (called PACD) is based on a bitmap representation for XML's structural relationships. XPath axes, plus their extension (i.e. "next" axis) for accessing the document order, are translated to sparse matrices allowing data compression, query complexity reduction and XML updates relaxation. Experimental results, outlined in this paper, show promising performance improvements over conventional techniques in a wide range of query types.

## 1 INTRODUCTION

Current performance limitations of native XML technology (Abiteboul et al. 2000) determine that XML-encoded data sets are mainly stored and manipulated using the more mature relational technology (e.g Amer-Yahia et al. 2004, DeHaan et al. 2003, and Rys 2005). This poses the challenge of devising methods of utilising relational technology to manage XML data sets, such that query performance is comparable to that achieved for conventional tabular data.

Our survey of storage models and query processing techniques for native and relational representations of XML data identifies current limitations with respect to performance and query expressiveness. To address these, a novel storage model for mapping XML data into a bitmap representation and an associated query processor are proposed. Additionally we implemented and evaluated our framework as a new XML-to-RDBMS mapping technique. Our experiment has demonstrated support for a wider range of XPath/XQuery (Schmidt et al. 2001) queries, greater potential for optimisation through parallel processing of query components, reduced resource overheads, and reduced complexity of XML updates compared to alternatives in the literature.

The rest of this paper is organized as follows. Section 2 reviews XML/SQL related work. The new framework is introduced in Section 3 while Section 4 describes our implementation of the proposed framework as an XML-to-RDBMS mapping technique, which is evaluated in Section 5. Finally, Section 6 concludes the paper and suggests paths for future work.

## 2 RELATED WORK

Technologies for managing XML data sets are either native XML (e.g. Fiebig et al. 2002, and Sipan et al. 2004) or extensions to relational technology (e.g. Amer-Yahia et al. 2004, Rys 2005, and Abdel-Kader et al. 2008). Of the latter, SQL/XML is the mainstream approach, and extends the SQL standard by including XML types, XPath and XQuery capabilities (e.g. Rys 2005), and transformation between XML and relational representations (Amer-Yahia et al. 2004; Abdel-Kader et al. 2008). XML-to-RDBMS mapping inherits the advantages of established relational database engines for storing, querying and indexing XML data; and is widely used because of immaturity and poor performance of native XML technology (Sipan et al 2004). Therefore, relational storage models for XML data (such as in-lining/Edge (Florescu and Kossmann 1999) and ShreX (Amer-Yahia et al. 2004)) with their associated query processors remain a focus for research. There is an ongoing need for more comprehensive performance evaluation of these techniques. Such research must address complexities introduced by storing irregular and deeply structured XML data in the orderly and shallow structures inherent in the relational model. Problems include path navigation, representation of sequence within XML data structures and associated query optimisation.

An approach to support path navigation that resonates with the database approach is to directly model XML structures within the database design. Within the relational model this can be achieved through storage of XML paths in fixed (Jiang et al. 2002; Lau and Ng 2004) or dynamic (Amer-Yahia et al. 2004) pre-defined relations in order to preserve the XML structure (e.g. parent/child and ancestor/descendant relationships). This is an attractive approach, since it avoids overheads incurred by alternative approaches, such as parsing the XML schema and/or data to determine structure (Sipan et al 2004; Rys 2005). However, dynamic pre-defined mapping schemas become insufficient when the XML schema is missing or changes frequently. On the other hand, fixed pre-defined mapping schemas introduce inefficiencies for certain classes of query (e.g. twig-queries (Choi et al. 2003)), particularly when expensive recursive joins must be computed.

The inherent sequence of XML structures poses additional complexity for relational database technology, since relational data is un-sequenced. This is addressed in many models by node labelling approaches; prefix-based and region-based. In the former, the label of a child node contains its parent's label (Tatarrinov et al. 2002; Wong et al 2003), whereas in the later; labels identify each node's descendants (DeHaan et al. 2003; Yoshikawa et al. 2001). However, existing node numbering algorithms suffer from high or unpredictable storage size for node labels, and/or complex label update processes. Solutions, such as interval-labelling (i.e. using gaps) and the use of floating point numbering instead of integers, defer the re-labelling process, but do not avoid it completely ( Yu et al. 2005; Li et al. 2004).

Finally, existing XML-to-RDBMS techniques predominantly translate XPath/XQuery queries into a single SQL-SELECT statement (Yoshikawa et al. 2001; Jiang et al. 2002). However, this is problematic in three respects: queries are restricted to what can be expressed in a single SQL-SELECT; path navigation necessitates a potentially large numbers of self-joins, even for simple path queries; and the query translation process itself can be complex, resulting in long SQL statements which might not be supported by many existing RDBMSs and which present complex optimisation challenges. It is therefore not surprising that recent performance analyses have demonstrated that using only XML types and XQuery or XPath can lead to poor query performance, with exponential query speed deterioration for some classes of query as the database size increases (Krishnamurthy et al. 2004; Abdel-Kader et al. 2008). However, understanding of the trade-offs between different XML and XML/Relational technologies is limited since performance studies, such as (Florescu and Kossmann 1999), have considered only query execution time, while ignoring other resources, such as memory and CPU usage and IO-Read Operations.

In summary, the above review has identified three main problems in existing XML data-storages and query optimization techniques. These are; the large storage requirement for representing XML data or its indexes, the high computational cost during querying and updating XML data, and the restrictions in the query processor domain incurred by the XML data/index representation. The following section hypothesises a novel bitmap framework to overcome these limitations.

## 3 SYSTEM DESIGN

To address the issues identified in Section 2, we propose a framework, called PACD (***P****arent-*
***A****ncestor/ *
***C****hild-*
***D****escendant* relationships) for storing and querying XML databases. PACD can be implemented natively or as a XML-to-RDBMS mapping technique. This section initially motivates the new system and then describes its main components. Sections 4-5 discuss the design of an XML/relational model based on the proposed system.

### 3.1 Motivation

PACD is designed to address issues identified in the above literature by exploitation of the concept of transitive closures, as implemented in XPath axes (Wang et al. 2006). The set of transitive closures (described in Section 3.2) are utilised as indexes to achieve efficient XML querying and manipulation. Firstly, these indexes provide an abstraction of path information that can be compressed using sparse-matrix compression techniques, thus allowing memory-based index processing, which in turn speeds up query execution. Secondly, the PACD data model reduces the high computation/storage cost associated with the document's order while querying and updating. Instead of using the expensive following-sibling/preceding-sibling axes, the document's order is represented by an extended XPath axis (called '*next'* axis) which is encoded in the '*nextOf*' index in the PACD's design. Also, by abstracting out path and sequence information into the matrices, PACD supports low cost operations for updating index structure when the XML data is updated (e.g. adding and deleting nodes). Finally, we propose a query processor for PACD which decomposes the XML query into multiple sub-queries, thus avoiding the complexities associated with executing long and complex XML queries, and providing potential for parallel processing.

A further advantage of PACD is that the number and type of matrices can be configured for specific applications, in order to increase or decrease the XPath/XQuery coverage. For example, for most data-centric (Abiteboul et al. 2000) XML databases, where document order is irrelevant, *childOf* and *descOf* matrices are sufficient to answer XML queries (Pettovell and Fotouhi 2006). The PACD's data model and query processor are described next.

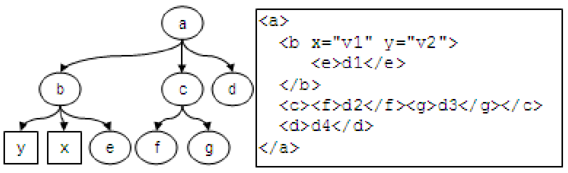
## 3.2 The PACD Data Model

The PACD system consists of two main operations; the *Index Builder* (IB) and its *Query Processor* (QP).

IB initially translates XML data into a list of the nodes, called nodeSet, and translates the structural relationships into a set of sparse matrices. This is done using a single SAX parsing operation (SAX Project 2004). The nodeSet is a set of node desciptions, each comprising a node's name, type, identifier and value. Each matrix represents an XPath axis (W3C 2007), i.e. nextOf, childOf, and descOf, as follows. The matrix is coordinated by node identifiers and each of its elements, <x,y> say, indicates whether the x node is related to the y node by the associated XPath axis relationship. Thus, for example, if y is a child of x, the element <x,y> of the childOf matrix will be 1; otherwise 0.

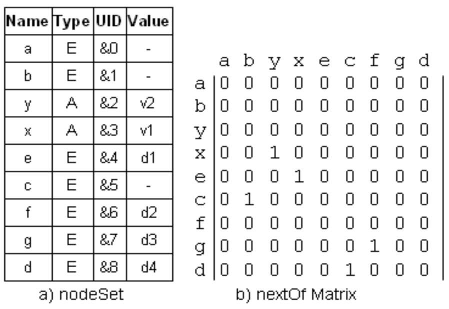
This representation is illustrated in Figure 1 and Figure 2, which respectively depict an example of XML data and its tree structure, and its 'nodeSet' and 'nextOf' matrix representations.

To reduce the storage requirement, the IB incorporates a *Data Compression/De-compression* model (DCD) which compresses the underlying matrices using established sparse-matrix compression techniques so that the compressed data can reside in the main memory. DCD is also used to decompress the stored data during the query processing phrase. The storage-space requirement, without using any DCD model; of the *childOf* and *nextOf* matrices require no more than O(n) storage-unit, whereas the maximum size of the *descOf* matrix is $O(n^2)$.



```
<a>
  <b x="v1" y="v2">
     <e>d1</e>
  </b>
  <c><f>d2</f><g>d3</g></c>
  <d>d4</d>
</a>
```

Fig 1: An Illustrative XML Example



| Name | Type | UID | Value |
|---|---|---|---|
| a | E | &0 | - |
| b | E | &1 | - |
| y | A | &2 | v2 |
| x | A | &3 | v1 |
| e | E | &4 | d1 |
| c | E | &5 | - |
| f | E | &6 | d2 |
| g | E | &7 | d3 |
| d | E | &8 | d4 |

a) nodeSet

$$\begin{array}{c|ccccccccc|} & a & b & y & x & e & c & f & g & d \\ a & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ b & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ e & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ c & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ f & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ g & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ d & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}$$

b) nextOf Matrix

Fig 2: The PACD Sub-representation of Fig 1

## 3.3 Query Processor

During query processing (QP), XML queries are decomposed into sub-queries in order to reduce the complexity of the query evaluation process. This is achieved by following the approach in (Choi et al. 2003; Vagena et al. 2004; Chen et al. 2006) whereby the XPath query is graphically represented as a twig-tree, and twig queries are executed using a Finite-State-Machine (FSM). In the FSM execution; a transition from one state to another returns a subset of the original node-set, while the current state's node-set becomes the input node-set for the following transition(s) to the next state(s). The final result is generated by joining the respective node-sets of all paths in the FSM at the final state.

An advantage of using such an execution plan is that a decision about a NULL result-generation can be made early in the process when a transition from one state to another returns an empty node-set. PACD query processor exploits this feature as part of its query processing optimisation. Also, whereas existing FSM-based query processors use the stack, TwigStack (Vagena et al. 2004) to accumulate intermediate states' results, the PACD processor frees its stack at every state transition. This allows more free memory resource for other computations.

## 4 PACD MAPPING TECHNIQUE

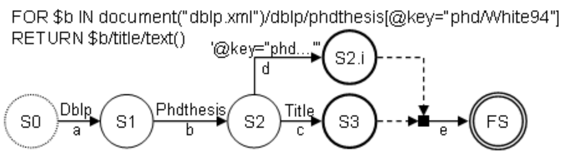
The previous two sections have overviewed the XML data representation and query processing strategy deployed by PACD. Here we demonstrate their application using a worked example, in which PACD is implemented using an RDBMS.

The mapping schema consists of four base relations, each of which corresponds to an IB's component (underlining denotes the use of RDBMS indexes):

nodeSet(<u>UID</u>, Typel, <u>Name</u>, <u>Value</u>)
childOf(<u>childID</u>, <u>parentID</u>)
descOf(<u>descID</u>, <u>anceID</u>)
nextOf(<u>nextID</u>, <u>prevID</u>)

This relational implementation of IB applies the simplest form of sparse-matrix compression technique, i.e., *Zero-element Removal* algorithm (Bell and McKenzie 1998). Based on this, the PACD's relations include one record per each non-zero element of structure-based matrices. The QP then translates an XML query into multiple SQL-SELECT statements.

An illustrative example is provided in Figure 3 and Table 1, which respectively present the example Xquery and associated FSM, and the resulting set of SQL sub-queries.



```
FOR $b IN document("dblp.xml")/dblp/phdthesis[@key="phd/White94"]
RETURN $b/title/text()
```

Fig 3: An XQuery Example and its FSM Execution Plan.

Tab 1: An SQL Translation of XQuery in Fig 3.

```
a:: SELECT childID AS dblp FROM childOf c, nodeSet n
      WHERE (c.parentID=-1) & (c.childID=n.uid) &
                   (n.name="dblp") &  (n.type="E")
b:: SELECT childID AS phdthesis FROM childOf c, s1,
      nodeSet n  WHERE (c.parentID=s1.dblp) &
      (c.childID=n.uid) & (n.name="phdthesis") & (n.type="E")
                                   …
e:: SELECT n.value FROM nodeSet n, s2i, s3 WHERE
(s2i.title=s3.title) & (n.uid=s3.title)
```

Using the DBLP database, the XQuery example displays the title of a PhD thesis identified by 'phd/white94'. The execution plan starts at *S0* and proceeds to finish at *FS* whenever there is a transition from a state $S_i$ to $S_{i+1}$ via the marked labels (e.g. 'dblp' and 'title'). A separate SQL statement is generated for every transition. (Table 1 lists the first (a), second (b) and last SQL (e) of the generated SQL sub-queries.

## 5 A PERFORMANCE STUDY

### 5.1 Experiment Setup

The comparative performance of PACD has been evaluated against the Edge (Florescu and Kossmann 1999) and XParent (Jiang et al. 2002) techniques. The latter two techniques were selected bacause; a) both are schema-oblivious, like PACD, b) Edge is a single-relation mapping schema while XParent represents multi-relations mapping techniques, and c) unlike PACD, Edge and XParent use node-labelling algorithms to support XML order.

We considered eight database functionalities (extracted from the XMark project (Schmidt et al. 2001)), listed in Table2.

Tab 2: Query-set.

| |
|---|
| *Q1: Shallow Exact Matching→* Return the name of the item with ID 'item?????' registered in North America? |
| *Q1B: Deep Exact Matching→* Return all text elements for the category with ID 'category0'? |
| *Q2: Order Access→*Return the initial increase of all open auctions? |
| *Q3: Regular Path Expression using "*"→* How many items are listed on all continents? |
| *Q3B: Regular Path Expression using "//"→* How many text elements are listed under all items from all continents? |
| *Q4: Joins on Values→* How many emails sent about each item in Asia and Europe at every date? |
| *Q5: Path Traversal→* Print the text in annotations of closed auctions? |
| *Q6: Missing Elements→* Which person does not have a homepage? |
| *Q7: Sorting→* Give an alphabetically ordered list of all items along with their location? |
| *Q8: Aggregation→* Group customers by their incomes and output the cardinality of each group? |

The query-sample tests critical aspects of processing XML databases over relational databases, in line with the XMark benchmark, and other performance studies (e.g Li et al. 2004; DeHaan et al. 2003). Table 3 summarises features of the XML databases used. Our experiment was designed to determine the impacts of the depth and breadth of XML structure on system performance. Four performance variables were measured; Execution Time, Memory Usage, CPU Usage, and IO-Read

Tab 3: Characteristics of the Tested Dataset

| DB Name | Nodes | Value Nodes | Level | Size (MB) |
|---|---|---|---|---|
| XMARK | 2,437,669 | 1,547,337 | 10 | 153.627 |
| DBLP | 2,439,294 | 2,166,223 | 6 | 85.344 |
| TREEBANK | 2,437,667 | 1,392,833 | 36 | 84.065 |

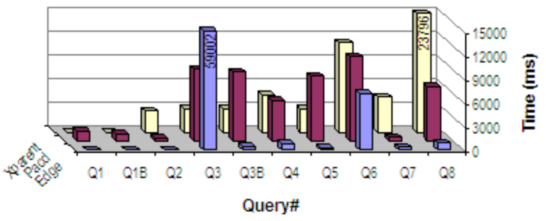


Fig 4: Execution Time over DBLP Database

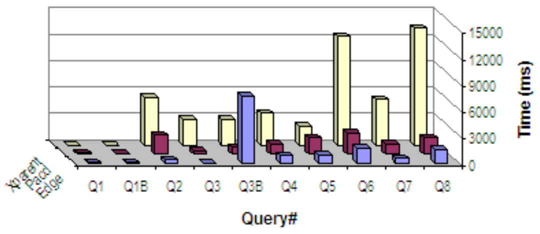


Fig 5: Execution Time over XMARK Database

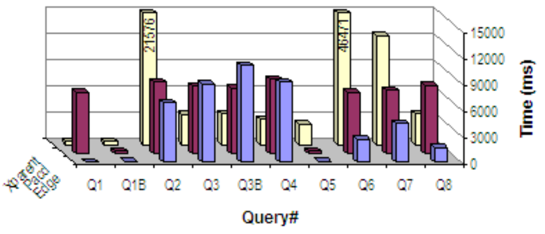


Fig 6: Execution Time over TREEBANK Database

Tab 4: Result Summary (Execution Time)

| | Q1 | Q1B | Q2 | Q3 | Q3B* | Q4 | Q5 | Q6 | Q7 | Q8* |
|---|---|---|---|---|---|---|---|---|---|---|
| **PACD vs. Edge** | - | - | - | B, D | A, D | D | - | - | - | - |
| **PACD vs. XParent** | - | D | B, A, D | A | A | A | A, D | B, A, D | B, A, D | B, A |

[*D=Depth-oriented, B=Breadth-oriented, A=Average-oriented*], * Edge and/or XParent's fail to satisfy this query; multiple-SQL were used instead

Operations. Due to space limitation, only the execution-time is discussed in this paper. Ninety queries were executed by the FoxPro®9.0 database engine installed on a stand-alone PC (Pentium®4, 3.60GHz dual processor, 1GB physical memory, 160GB HD) running Windows XP SP2.0.

## 5.2 Results Discussion

Experimental results are represented in Figures 4-6 and summarised in Table 4.

***PACD vs. Edge:*** Results shown that PACD outperforms Edge for queries; Q3 (TreeBank & DBLP), Q3B (TreeBank & XMark), and Q4 (TreeBank). Comparing to Edge, PACD was ≈0.16 and ≈5.5 times faster in the 1st case; ≈0.5 and ≈8.5 times faster in the 2nd case; and ≈0.07 times faster in the 3rd case. The superiority of PACD against Edge is valid for 10%, 33.4% and 10% of the query-sample over the DBLP, TreeBank and XMark databases respectively. These low-percentages, in terms of execution-time, are ameliorated by a better performance in terms of other measurements such as memory-usage. On the negative side, PACD was extremely slow for Q1 over the DBLP and TreeBank databases and for Q3 over the XMark database.

***PACD vs. XParent:*** PACD outperformed XParent for queries; Q1B-Q8 (XMark≡80%), Q1B-Q2 and Q5-Q7 (TreeBank≡50%), and Q2 and Q6-Q8 (DBLP≡40%). The best performance in the 1st case was for Q3 and Q8 where PACD was more than 6 times faster than XParent; while in the 2nd case, it was ≈7.5 times faster for Q5, and in the 3rd case was ≈8.6 times faster for Q7. In contrast, the worst performance of PACD against XParent was in Q1 (≈140 times slower) and Q1B (≈58.5 times slower) over the DBLP databases.

Our experiment shows that, Edge and XParent failed to execute Q3B and Q8 using single-SQL translations. Overall, PACD is more suitable for depth-oriented XML databases for most XQuery types.

## 6 CONCLUSION

In this paper we have introduced a novel XML processing framework which is based on a bitmap data representation and serialized query processing. The new framework aims to overcome three major problems identified in the existing XML storage-models and optimization techniques.

The large storage-size problem is addressed by employing sparse-matrix compression techniques while the query-domain is increased by using a special serialized query processor that acts on the proposed XML data representation. In terms of order-access, the storage and update problems are solved by extending XPath axes to include the "next" axis as described in Section 3.

We have evaluated our framework as an XML-to-RDBMS mapping technique (also called PACD). Our preliminary results show several enhancements in terms of query-coverage, query processing and storage management. Table 4 summaries our experimental results.

On the other hand we are less successful in handling complex ordering-queries and achieving optimum performance for some query types. Our ongoing research investigates optimizing the use of the 'next' axis to address the first issue and the use parallelism to address the second. We are also aiming to achieve better storage performance by testing more sophisticated data compression techniques.

## REFERENCES

Abdel Kader, Y., Eaglestone, B., and North, S. (2008) 'An Analysis of Relational Storage Strategies for Partially Structured XML', *WebIST'08*, Madeira, Portugal, pp 165-170.

Abiteboul, S., Buneman, P., and Suci. D. (2000) *Data on the Web: From Relations to Semistructured Data and XML,* California: Morgan Kaufmann Publishers.

Amer-Yahia, S., Du, F., and Freire, J. (2004) 'A Comprehensive Solution to the XML-to-Relational Mapping Problem', In *Proceedings of the 6th annual ACM/IWIDM'04*, Washington DC, USA, pp 31-38.

Bell, T., and McKenzie, B. (1998) 'Compression of Sparse Matrices by Arithmetic Coding', *ICDC'98,* pp 23-32.

Chen, S., Li, H., Tatemura, J., Hsiung, W., Agrawal D., and Candan, K. (2006) 'Twig2Stack: Bottom-up Processing of Generalized-Tree-Pattern Queries over XML Documents', *VLDB'06*, Seoul, Korea, pp 283-294.

Choi, B., Mahoui, M., and Wood, D. (2003) 'On the Optimality of Holistic Algorithms for Twig Queries', *LNCS 2736*, pp 28-37.

DeHaan, D., Toman, D., Consens, M., and Ozsu, M. (2003) 'A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding', *ACM/SIGMOD'03, San Diego, CA, USA,* pp 623-634.

Fiebig, T., Helmer, S., Kanne, C., Moerkotte, G., Neumann, J., and Weld, R.. (2002) 'Anatomy of a native XML base management system'. *VLDB Journal, 11(4)*, pp 292-314.

Florescu, D., and Kossmann, D. (1999) 'A Performance Evaluation of alternative Mapping Schemas for Storing XML Data in a Relational Database', *TR:3680, May 1999, INRIA, Rocquencourt, France.*

Jiang, H., Lu, H., Wang, W., and Yu, J. (2002) 'XParent: An Efficient RDBMS-Based XML Database System', *ICDE'02, CA, USA,* pp 1-2.

Krishnamurthy, R., Kaushik, R., and Naughton, J. (2004) 'Efficient XML-to-SQL Query Translation: Where to Add the Intelligence?', *VLDB'04, Toronto, Canada,* pp 144-155.

Lau, H., and Ng, V. (2004) 'INode*: An Effective Approach for Storing XML using Relational Database'. *Int'l Journal of WET, 1(3),* pp 338-352.

Li, H., Lee, M., Hsu, W., and Chen, C. (2004) 'An Evaluation of XML Indexes for Structural Join', *ACM/SIGMOD, 33(3),* pp 28-33.

Pettovello, P., and Fotouhi, F. (2006) 'MTree: An XML XPath Graph Index', *ACM/Sym. on Applied computing'06*, *Dijon, France,* pp 474-481.

Rys, M. (2005) 'XML and Relational Database Management Systems: Inside Microsoft SQL Server 2005', *ACM/SIGMOD'05*, Baltimore, Maryland, pp 958-962.

SAX Project. (2004) Simple API for XML (SAX). [Online] Avail: http://sourceforge.net/projects/sax/ [20/09/2008].

Schmidt, A., Waas, F., Kersten, M., Florescu, D., Manolescu, I., Carey, M., and Busse. R. (2001) 'The XML Benchmark Project', *INS-R0103 Apr30*, pp 1-18.

Sipan, S., Verma, K., Miller, J., and Aleman-Meza, B. (2004) 'Designing a high-performance database engine for the 'Db4XML' native XML database system', *The Journal of Systems and Software-69*, pp 87-104.

Tatarrinov, I., Viglas, S., Beyer, K., Shanmugasundaram, J., Shekita, E., and Zhang, C. (2002), 'Storing and Querying Ordered XML Using a Relational Database System', *ACM/SIGMOD'02*, Madison, Wisconsin, pp 204-215.

Vagena, Z., Moro, M., and Tsotras, V. (2004) 'Twig Query Processing over Graph-Structured XML Data', *7th int'l workshop on the Web & Data.*, Paris, France, pp 43-48.

W3C. (2007) XML Path Language (XPath) 2.0, [Online] Avail: http://www.w3.org/TR/xpath20/ [30/10/2008].

Wang, H., He, H., Yang, J., Yu, P., and J Yu. (2006) 'Dual Labelling: Answering Graph Reachability Queries in Constant Time', *ICDE'06,* pp 75-86.

Wong, W., Jiang, H., Lu, H., and Yu, J. (2003) 'PBiTree Coding and Efficient Processing of Containment Joins', *ICDE'03, Boston, USA,* pp 391-402.

Yoshikawa, M., Amagasa, T., Shimura, T., and Uemura, S. (2001) 'XRel: A Path-based Approach to Storage and Retrieval of XML Documents using Relational Databases', *ACM/IT., 1(1), NY, USA,* pp 110-141.

Yu, J., Luo, D., Meng, X., and Lu, H. (2005) 'Dynamically Updating XML Data: Numbering Scheme Revisited', *WWW, 8(1)*, pp 5-25.