RC22923 (W0309-187) September 30, 2003
Computer Science

IBM Research Report

Checking XML Constraints Using Relational Technology

Yuan Wang
Department of Computer Science
University of Wisconsin
Madison, WI

George A. Mihaila, Sriram Padmanabhan
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Checking XML Constraints Using Relational Technology

Yuan Wang George A. Mihaila Sriram Padmanabhan

Department of Computer Science
University of Wisconsin, Madison
yuanwang@cs.wisc.edu

IBM T.J. Watson Research Center
Hawthorne, New York 10532
{mihaila, srp}@us.ibm.com

Abstract

As XML becomes the prevalent format for data interchange between B2B applications, efficiently checking the validity of XML documents with respect to a given schema is going to be critical. While this can be currently achieved using a standard validating parser, we argue that this is not always the most efficient way. Since most of the XML data today resides in relational database systems, it makes sense to rely on the capabilities of database systems for checking the constraints imposed by particular schemas. In this paper we examine how the various types of XML constraints can be expressed as SQL queries. We give an algorithm that, given a mapping between an XML schema and a relational schema, automatically translates XML constraints into equivalent database queries. We also report on our experimental evaluation of this method's performance.

1 Introduction

With the recent advent of Web services and protocols such as BizTalk and SOAP, XML is quickly becoming the de facto standard format for data exchange. In this scenario, an application invokes a service from a remote application by sending an XML formatted request and receives the response also as an XML document. In order for this process to work correctly, the schemas of all incoming and outgoing documents are negotiated in advance by the parties involved in this conversation.

These schemas specify a set of recognized element names and prescribe a document structure (ie. element nesting hierarchy, element ordering, element count, etc). In order to check all these constraints, validating parsers have to maintain internal data structures and perform complex checks that in some cases depend on the entire contents of the document (e.g. referential integrity). Since database systems have to deal with similar types of constraints for large volumes of data, a natural idea would be to take advantage of these capabilities when checking XML schema conformance. This seems especially useful considering that today's B2B applications produce and consume XML messages at their interface, while using a relational database system as backend. In this scenario both the outgoing and incoming XML documents are ultimately mapped to some collection of tuples from the underlying database tables. Therefore, instead of checking the schema conformance at the XML level, we propose translating XML constraints to relational constraints or queries, evaluating these in the DBMS and interpreting the results of these checks into reports of XML schema violations. Our argument is that the data can be checked more efficiently in the DBMS, before being converted into XML. Aditionally, this technique provides a natural method to transfer XML schema constraints to a rational database system whenever such a mapping is possible.

Validating parsers are also limited in their ability of checking individual constraints in isolation. For example, there is no direct method to instruct the parser to check just a key uniqueness constraint. Also, there is no support for incremental constraint checking: when new data is inserted to an existing large document, the whole document needs to be checked again. In our approach, the new data can be bulk-loaded into temporary tables, checked for constraint violations and then appended to the production tables.

The objective of this paper is to prove that in addition to providing reliable, scalable storage for large volumes of XML data, relational technology can be successfully leveraged for checking XML data integrity. To summarize, our contributions are as follows:

- we discuss how XML schema constraints can be checked using relational queries for several of the XML-relational mappings that have been proposed in the literature (summarized in Section 2);
- we analyze the complexity of constraint checking queries for the various mapping schemes and propose an extension to an existing mapping that eliminates expensive joins from these queries;
- we define a formalism describing XML-relational

mappings and present algorithms for automatic constraint translation for any given mapping;

- we study the performance of our method through experimental validation and compare it against a validating parser.

The paper is organized as follows: Section 2 discusses related work; Section 3 shows how XML constraints can be translated into relational queries given various types of XML-relational mappings; Section 4 presents our experimental performance evaluation; and Section 5 summarizes the work.

2 Related Work

In recent years, we have witnessed an increasing interest in the research community for the efficient storage and querying of XML data. There are essentially two main directions: designing native XML databases (eg. Natix [KM00], Lore [GMW00], Tamino [Sch01], Niagara [NDM+01]) and using off-the-shelf database systems. While we do recognize the potential performance benefits of native storage systems, in this work we focus on the relational storage route.

Because of the differences in data models, relational database engines cannot directly store XML (except as BLOBs, which is not very interesting), therefore some sort of data reorganization is needed. Here again there are two main categories of efforts: converting XML data to a format suitable for relational storage and publishing existing relational data as XML.

In the first category, Florescu and Kosmann [FK99] propose a number of mappings derived from the basic *edge table* approach that stores all parent-child edges in one table. While these mappings do not require a priori knowledge of a schema, they exhibit poor performance for document reconstruction and querying. An alternate method, that exploits schema information, is proposed by Shanmugasundaram et. al. [STZ+99]. They propose two mappings, *hybrid inlining* and *shared inlining* which cluster related elements together in tables, for better performance.

In the second category, that of publishing existing relational data as XML, the XPERANTO system [SSB+00, CKS+00] provides an XMLQuery-based mechanism for defining virtual XML views over a relational schema and translates XML queries to SQL. A similar system, SilkRoute [FMS01], translates XML queries to an intermediate query language, RXL. The XML Access Server (XAS) [LCPC01] uses annotated DTDs for defining XML views over multiple data sources. Also, all major commercial DBMS products (eg. Oracle 8i, IBM DB2 and Microsoft SQLServer) support some form of XML view definition over relational data.

In this paper we are not trying to define yet another mapping from XML to relational schemas. Instead, we discuss how XML constraints can be translated relational queries for some of the existing mapping schemes.

The first types of integrity constraints that have been defined for XML are *key* and *key reference* (also known as *foreign keys* in database literature). These constraints can be stated in Document Type Definitions (DTDs) using special attributes ID and IDREF. These concepts have been refined in XMLSchema (`www.w3c.org/xmlschema`), which allows any XML attribute or element to be declared as a key or key reference. Additionally, XMLSchema introduces other constraints, such as occurence constraints, string patterns, etc.

Integrity constraints in XML have also been examined from a more theoretical point of view. Thus, Fan and Libkin [FL01] study the interaction between DTDs and XML integrity constraints. They show that, in general, deciding whether a DTD with key and foreign key definitions can have conforming documents is an NP-complete problem. Buneman et. al. [BDF+01] examine keys in XML, and argue for the usefulness of *relative keys*, definable using path expressions.

3 Translating Constraints

3.1 Motivation

XML Schema has much more expressive power than DTD does and provides users the ability to define many types of data constraints. Some constraints, such as value pattern constraints, can be validated by checking the single element on which they are defined. But some other constraints force a validating parser to look beyond the target element to validate them. For instance, to validate a key constraint defined on an element E, a validating parser has to maintain some kind of data structure to record values of all E elements. We call this type of constraints `Non-local` constraints. Among other non-local constraints we can mention *Uniqueness* and *Keyref* constraints. Due to the space limit, we only discuss *Key* constraints in this paper.

According to our experiments on major validating parsers, we found that non-local constraint validation is the bottleneck in parsing XML documents. To aleviate this problem, we propose using SQL queries for constraint validation. We have studied three mapping schemes, namely `manual mapping`, `edge table` [FK99], and `shared inlining` [STZ+99]. Due to the space limit, we first show some example key constraints which we provide SQL queries for only `shared inlining`. Then, in Section 3.4, we describe an algorithm for automatic translation of constraints.

3.2 Translating Constraints for the Shared Inlining Scheme

Shanmugasundaram et al. [STZ+99] discuss several schema conversion techniques. In this paper, we evaluate our method over one of techniques they proposed, the `Shared Inlining` technique.

Let us consider the following key constraint in a bookstore schema.

```
<?xml version="1.0"?>
<schema>
 <element name="Books"><complexType><sequence>
   <element name="Book" maxOccurs="unbounded">
    <complexType><sequence>
     <element name="Title" type="string"/>
     <element name="ISBN" type="isbn_type"/>
     <element name="Publisher" type="string"/>
     <element name="Authors" maxoccurs="unbounded">
        ...
     </element>
    </sequence></complexType>
   </element>
  </sequence></complexType>
  <key name="KEY_ISBN">
   <selector xpath="Book"/>
   <field xpath="ISBN"/>
  </key>
 </element>
</schema>
```

The following tables will be generated by the shared inlining method.

```
Books (ID)
Book (ID, parentID, Title, ISBN, Publisher, ...)
Authors (...)
...
```

The following two queries validate the uniqueness and existence parts of the key constraint respectively.

```
SELECT ISBN              SELECT *
FROM   Book              FROM   Book
GROUP  BY ISBN           WHERE  ISBN = null
HAVING count(*) > 1
```

3.3 An Enhanced Shared Inlining Scheme

Consider the schema tree (a) in Figure 1. If element E_i is the effective range element defined in the key constraint, ie., E_{Key}s should be unique in E_i, we need $n-i-1$ join clauses in the SQL query to validate the key constraint, "$E_{i+1}.ID = E_{i+2}.parentID$ AND ... AND $E_{n-1}.ID = E_n.parentID$". Note, the join of "$E_i.ID = E_{i+1}.parentID$" can be saved by using "GROUP BY $E_{i+1}.parentID$".

Also consider the schema tree (b) in Figure 1, no matter which element is defined as the effective range element, if key elements are only E_{Key}s that are under E_i, we need at least $n-i-1$ join predicates to locate all those E_{Key} elements: $E_{i+1}.ID = E_{i+2}.parentID$ AND ... AND $E_{n-1}.ID = E_n.parentID$.

In both examples, we join the tables bottom-up from the table that contains the key element. Since join operations can be expensive in executing those queries, we propose an improved shared inlining method which eliminates join operations in our validating SQL queries.

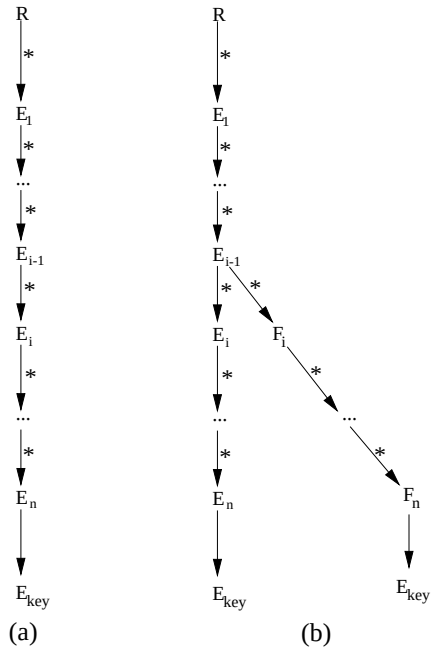

Figure 1: Schema Examples

The intuition is, since the reason for joining the tables is to locate the effective range element either for uniqueness level or effective branch, we can add that information in the bottom table to avoid the joins. We add two attributes, *uniqueness range* and *selective range*, in the table that contains the key element. [1]

- *Uniqueness range* contains element ids under which key elements are required to be unique;
- *Selective range* contains element ids that are the top elements in the `selector` field(s) of the key constraint.

So the validating queries for schemas (a) and (b) in Figure 1 based on this improved shared inlining method are

```
SELECT   E_Key                  SELECT   E_Key
FROM     E_N                    FROM     E_N
GROUP BY E_uniqueID, E_Key      WHERE    E_selectID <> null
HAVING   count(*) > 1           GROUP BY E_uniqueID, E_Key
                                HAVING   count(*) > 1
```

3.4 Automatic Translation of Constraints

In this section we describe an algorithm for automatic translation of constraints. Our algorithm takes as input a mapping from an XML schema to a relational schema and a list of XML constraints and outputs the equivalent relational queries for the constraints.

[1] In general, if there are multiple key constrains defined on an element, multiple *uniqueness range* and *selective range* attributes should be added in the table.

First, let us introduce some preliminary notions.

Definition 3.1 An *XML schema graph* is a labeled directed graph $S = (V, E, \lambda_V, \lambda_E)$ where V is a set of vertices corresponding to element, attribute and text node definitions, E is a set of child edges, $\lambda_V : V \rightarrow \Sigma^*$ maps each element and attribute node to a name and each text node to the empty string ϵ, and $\lambda_E : E \rightarrow N^2$ specifies for each edge, the minimum and maximum number of occurences (in an instance document): for any edge $e \in E$, $\lambda_E(e) = (\lambda_E^{min}(e), \lambda_E^{max}(e))$; the edge e is called *simple* if $\lambda_E^{max}(e) = 1$ and *multiple* otherwise.

Definition 3.2 An *XML instance graph* is a labeled directed tree $I = (W, F, \lambda_W)$ where W is a set of vertices corresponding to elements, attributes, and text nodes in an XML document, F is a set of child edges, and $\lambda_W : W \rightarrow \Sigma^*$ maps each vertex to its name (for element and attribute nodes) or content (for text nodes).

The XML instance graph I is said to *conform* to an XML schema graph S if there exists a function $f : W \rightarrow V$ with the following two properties:

- **node conformance:** for every instance node w, if $\lambda_V(f(w)) \neq \epsilon$ (it's an element or attribute node), then $\lambda_W(w) = \lambda_V(f(w))$;
- **edge conformance:** for every instance edge $(w, w') \in F$, its image through f is a schema edge: $(f(w), f(w')) \in E$;
- **edge multiplicity:** for every schema edge $(v, v') \in E$ and for every instance vertex w such that $f(w) = v$, $\lambda_E^{min}(v, v') \leq |\{(w, w') | f(w') = v'\}| \leq \lambda_E^{max}(v, v')$.

The first property ensures that every element or attribute node in the instance graph has the same label as its corresponding schema node. The second property ensures that child edges in the instance graph correspond to edges in the schema. The third property ensures that the number of edges emerging from any node falls between the minimum and maximum number of edges allowed for each type of child node.

Definition 3.3 Consider an XML schema graph $S = (V, E, \lambda_V, \lambda_E)$ and a relational schema $R = (R_1, \ldots, R_n)$, where each relation R_i has attributes $att(R_i) = \{a_1^i, \ldots, a_{k_i}^i\}$. Denote by A the set of all attributes in schema R. An *XML-Relational Mapping* is a function $\mu : A \rightarrow V$ satisfying the following property:

- **node surjectivity:** for every vertex v there exists at least one attribute a such that $\mu(a) = v$;
- **edge surjectivity:** for every edge $(v, v') \in E$ there exists at least one relation R_i containing two attributes $a, a' \in att(R_i)$ such that $\mu(a) = v$ and $\mu(a') = v'$.

The first property simply requires that every schema node is covered by a relational attribute. The second property ensures that each parent-child relationship in the schema is captured in some relation. Note that a schema node can be covered by more than one attribute from different relations. We call such attributes μ-equivalent. For a given relation R_i denote by $\mu(R_i)$ the subgraph of S induced by the set of nodes $\mu(att(R_i))$. A subgraph S' of S for which there exists a relation R_i such that $S' \subseteq \mu(R_i)$ is called μ-subgraph.

Note that Definition 3.3 captures a large class of possible XML-to-relational mappings (including the manual "semantically rich" mappings and all the inlining mappings), but it does not include the *edge table* approach directly. However, an edge table mapping can be transformed into a mapping expressible into this formalism by horizontally partitioning the edge table by element type.

Definition 3.4 Given an XML instance graph $I = (W, F, \lambda_W)$, a relational schema R and an XML-relational mapping μ, a database instance D is the *relational representation through* μ of the instance graph I if:

- for every instance node $w \in W$ and every relational attribute $a \in \mu^{-1}(f(w))$ there is exactly one tuple t such that $t.a = w$ if w is an element or attribute node and $t.a = \lambda_W(w)$ if w is a text node; we say that w is the instance node represented by the attribute a of tuple t and we write $w = \mu^I(t, a)$ (the schema mapping μ induces an instance mapping $\mu^I : D \times A \rightarrow W$);
- for every pair of attributes a and a' in some relation R_i such that $\mu(a')$ is a child of $\mu(a)$ (in the schema graph) and every tuple t in R_i, the node $\mu^I(t, a')$ is a child of the node $\mu^I(t, a)$ (in the instance graph).
- for every pair of attributes a and a' in some relation R_i such that $\mu(a')$ is reachable from $\mu(a)$ (in the schema graph) and every tuple t in R_i, the node $\mu^I(t, a')$ is reachable from the node $\mu^I(t, a)$ (in the instance graph).

The first property ensures that each instance node is mapped to a an attribute according to its corresponding schema node. The second property ensures that whenever two attributes of a relation are mapped to the endpoints of a schema edge, for every tuple of that relation these two attributes are mapped to the endpoints of an instance edge. Finally, the third property is similar to the second, but refers to paths instead of edges. The induced mapping μ^I specifies a *correspondence* relationship between the components of each tuple and the nodes of the instance graph. The actual value stored in the tuple components is a node ID for element and attribute nodes and the text value for text nodes.

We are now ready to present our algorithms for the constraints translation.

Uniqueness Constraint. Consider the following uniqueness constraint: "nodes of type q reachable from nodes of type v through the context path $C = (v = v_1, \ldots, v_m = q \in V)$ must have unique combinations of values for children $f_1, \ldots, f_k$ of q within the scope of each node of type v." Algorithm 1 generates an SQL query Q that checks this constraint.

Algorithm 1 Uniqueness Constraint Translation

Require: $S = (V, E, \lambda_V, \lambda_E)$, an XML schema graph;
$R = (R_1, \ldots, R_n)$, a relational schema;
$\mu : R \rightarrow S$, an XML-relational mapping;
$U = (C, F)$, a uniqueness constraint where:
$C = (v = v_1, \ldots, v_m = q \in V)$, is a *context path* and
$F = \{f_1, \ldots f_k\} \subset V$, are *field nodes*, all children of q;

1: Split the context path C into μ-subgraphs $C_1, \ldots, C_t$
2: **for** $i = 1, t$ **do**
3: Let R^i be a relation containing attributes for all the nodes in C_i
4: Let a_i be such that $\mu(a_i)$ is the first node in C_i
5: Let b_i be such that $\mu(b_i)$ is the last node in C_i
6: **end for**
7: **for** $j = 1, k$ **do**
8: Let attribute c_j of R^t be such that $\mu(c_j) = f_j$
9: **end for**
10: Q = SELECT $c_1, \ldots, c_k$
FROM $R^1, \ldots, R^t$
WHERE $b_1 = a_2$ AND $b_2 = a_3 \cdots$ AND $b_{t-1} = a_t$
GROUP BY $a_1, c_1, \ldots, c_k$
HAVING count(*)> 1

Note that it is always possible to split a path C into μ-subgraphs because of the *edge surjectivity* property from Definition 3.3: in fact, this can be done in linear time (in the length of the path), by examining the edges of the path in sequence and selecting for each edge (v_i, v_{i+1}) a relation R_i containing two attributes a_i, b_i such that $\mu(a_i) = v_i$ and $\mu(b_i) = v_{i+1}$. As an optimization heuristic, in order to minimize the number of joins in the generated query, one can select at each step the relation that covers the largest number of consecutive edges.

Key Constraint. The key constraint is very similar to the uniqueness constraint. The only difference is that in addition to being unique, all the fields specified by the constraint must have non-null values. Therefore, to check this constraint, the previous algorithm can be modified to generate the following existence query Q' to be executed just before the uniqueness query:

Q' =SELECT $c_1, \ldots, c_k, b_t$
FROM $R^1, \ldots, R^t$
WHERE $b_1 = a_2 \cdots$ AND $b_{t-1} = a_t$
AND (c_1 = NULL OR... c_k = NULL);

4 Performance Evaluation

In this section we evaluate our relational solutions for validating XML schema constraints. We compare our techniques with a general XML validating parser. We also evaluate the enhanced shared inlining method that we proposed in Section 3.3.

We use a synthetic dataset that is designed to be the cases shown in Section 3.3 ($n = 3$ in case(a), $n = 2, i = 1$ in case(b)). The number of key elements in each XML document varies from 50 to 20,000.

When we parse an XML document, we measure the execution time in two different trials. First, we remove the key constraint from the schema. Then, we validate the XML document against the full schema. So we can get a breakdown of the execution time of validating a specific key constraint.

When we evaluate our relational solutions, we load XML documents into relations through a mapping program. We do not construct any indexes on any relations. The query execution time is messured to compare with the validating parser.

We use Apache's Xerces-J 1.4.1 SAX parser as the validating parser. We run SQL queries on IBM DB2 UDB 7.2. All experiments are done on a 500 MHz Pentium III PC with 256 MB memory, running Windows 2000.

Figure 2 shows the difference in key constraint validation between the XML parser and SQL queries generated in the shared inlining scheme. The experiments clearly show that SQL solutions outperform the validating parser by several orders of magnitudes.

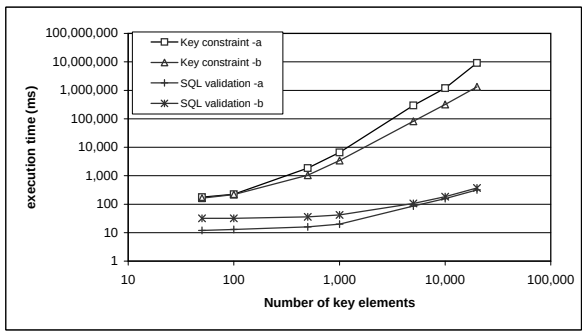

Figure 2: SQL vs. Validating Parser

In Section 3.3, we noticed that the validating SQL queries may contain additional join operations and/or selection predicates for some complicated key constraints. In order to avoid those joins and selection predicates, we proposed an enhanced shared inlining method for mapping XML documents to relational tables. Figure 3 shows the execution time of the queries on the enhanced shared inlining method, along with queries of the original shared inlining method. In both

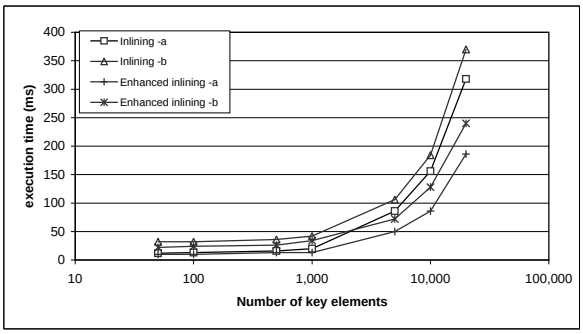

Figure 3: Enhanced Inlining vs. Shared Inlining

scenarios, whose key constraints are still simple, we achieved 30 to 40 percent performance gain.

5 Summary and Future Work

We addressed the problem of validating the constraints defined by an XML schema specification. We first examined the performance of current validating parsers when checking XML constraints. Thus, we identified a class of constraints for which the parsers exhibit a particularly poor performance (exponential increase in validation time with the size of the document).

Since most of today's XML data is stored in relational databases through some XML-to-relational mapping, we proposed using the capabilities of the database engine to check constraints. We analyzed the constraint translation problem in one of the most widely used mapping scenarios: the inlining approach. Thus, we have shown how key constraints can be expressed as queries against the relational tables. We then analyzed the performance implications of this method by examining the complexity of the resulting queries (ie. the number of joins involved). We discussed the performance gains that can be achieved by reducing the number of joins needed if the target relational schema is slightly extended. We then generalized from the particular translation techniques that we have identified for the inlining mappings, and we proposed algorithms for the automatic translation of constraints for a large class of mappings.

Finally, we validated our proposed techniques by experimentally comparing their performance against a validating parser. Thus, we were able to show a dramatic amelioration in validation time in all cases. Also, our experiments clearly demonstrate the scalability of our methods: while the running time for the parsers increases exponentially, the execution time for the equivalent queries grows linearly, even in the absence of additional indexes. Of course, indexes would further speed up the execution.

One important direction for future research is the automatic analysis of several constraint translation alternatives, perhaps taking into account data statistics, in order to pick the most efficient translation for each constraint. Another direction we are considering is extending the automatic translation algorithms to other types of mappings such as the edge table approach, path based mappings and arbitrary annotation-based manual mappings.

References

[BDF+01] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. In *WWW'10*, 2001.

[CKS+00] M.J. Carey, J. Kiernan, J. Shanmugasundaram, E. J. Shekita, and S. N. Subramanian. XPERANTO: Middleware for publishing object-relational data as XML documents. In *VLDB 2000*, pages 646–648, 2000.

[FK99] D. Florescu and D. Kossmann. Storing and querying XML data using an rdmbs. *IEEE Data Engineering Bulletin*, 22(3):27–34, 1999.

[FL01] W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. In *PODS*, 2001.

[FMS01] M. Fernandez, A. Morishima, and D. Suciu. Efficient evaluation of XML middleware queries. In *SIGMOD 2001*, pages 103–114, May 2001.

[GMW00] R. Goldman, J. McHugh, and J. Widom. Lore: A database management system for XML. *Dr. Dobb's Journal of Software Tools*, 25(4):76–80, April 2000.

[KM00] C-C. Kanne and G. Moerkotte. Efficient storage of XML data. In *ICDE 2000*, pages 198–200, San Diego, California, USA, March 2000. IEEE.

[LCPC01] M-L. Lo, S-K. Chen, S. Padmanabhan, and J-Y. Chung. XAS: A system for accessing componentized, virtual XML documents. In *ICSE 2001*, pages 493–502, may 2001.

[NDM+01] J.F. Naughton, D.J. DeWitt, D. Maier et. al. The Niagara Internet query system. *IEEE Data Engineering Bulletin*, 24(2), 2001.

[Sch01] H. Schöning. Tamino - A DBMS designed for XML. In *ICDE' 01*, pages 149–154, Heidelberg, Germany, April 2001. IEEE.

[SSB+00] J. Shanmugasundaram, E. J. Shekita, R. Barr, M. J. Carey, B. G. Lindsay, H. Pirahesh, and B. Reinwald. Efficiently publishing relational data as XML documents. In *VLDB 2000.*, pages 65–76, 2000.

[STZ+99] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *VLDB'99*, pages 302–314. Morgan Kaufmann, 1999.