# Integrating XML Data with Relational Databases

Wang-Chien Lee  Gail Mitchell  Xin Zhang*
GTE Laboratories Incorporated
40 Sylvan Road, Waltham, MA 02451, USA
E-Mail: {wlee, gmitchell, xzhang}@gte.com

## Abstract

*XML technology is pushing the world into the* e-*commerce era. Relational database systems, today's dominant data management tool for business, must be able to accommodate the XML data, since collecting, analyzing, mining and managing that data will be tremendously important tasks. In this paper, we investigate the problem of managing XML data in relational database systems. We are specifically concerned with storing and accessing the XML data using relational technology. We propose an algorithm for mapping a DTD to the Entity-Relationship (ER) model (and thus the relational model) and examine some of the issues in loading XML data into the generated model.*

## 1 Introduction

Touted as the ASCII of the future, the Extensible Markup Language (XML) is used to define markups for information modeling and exchange in many industries. By enabling automatic data flow between businesses, XML is pushing the world into the electronic commerce (*e*-commerce) era. We envision that collecting, analyzing, mining, and managing XML data will be tremendously important tasks for the era of *e*-commerce.

Database systems are the traditional tools for managing data. After many years of development, database technology has matured and contributed significantly to the rapid growth of business and industry. Relational database systems are a proven technology for managing business data and are used everywhere by various sizes of companies for their critical business tasks. Commercial relational products embody years of research and development in modeling, storage, retrieval, update, indexing, transaction processing, and concurrency control, and continue to add capabilities to address new kinds of data, such as multimedia. With more and more data flowing in XML formats, it is an obvious step to try to further extend relational systems to accommodate the XML data. Such an approach can avoid re-inventing database technology to suit XML data but, more importantly, takes best advantage of the power of relational database technology and the wealth of experience in optimizing and using the technology.

There are a number of technical issues to overcome in bringing XML data into the relational database for management, including defining a relational schema for the XML data, loading the XML data into a relational database, and transforming XML queries[1] into meaningful SQL queries. The work presented here examines relational schema definition for XML data; our goal is to maintain, as much as possible, the data semantics implied by the XML so that we can more accurately and efficiently manage the data. Thus, we start with XML documents that conform to some DTD and give an algorithm for converting the DTD to an ER model (from whence it could be translated to a relational model using well-known techniques [EN89]). A major contribution of this work is an analysis of the semantic characteristics of XML documents and the mapping to ER derives from this analysis.

The rest of the paper is organized as follows. In Section 2, we briefly introduce XML and Document Type Definitions (DTD). We analyze the characteristics of XML documents and embedded data, and discuss how these are related to the relational model and databases in Section 3. In Section 4, we propose an algorithm to automatically map a DTD into an entity-relationship model. We discuss a number of technical issues , including how to use this result for loading the XML data into a relational database, in Section 5. Section 6 concludes the paper with a summary of related work and

*Xin Zhang is a Ph.D. student of Worcester Polytechnic Institute and an interm at GTE Laboratories.

[1] whether formulated in XSL [Gro99], XML-QL [DFF+99] or other XML query standards.

an outline of our research direction.

## 2 Some Background: XML and DTDs

XML is currently used both for defining document markups (and thus information modeling) and for data exchange. Thus, XML documents have been categorized as *document-centric* and *data-centric* [Bou99]. Document-centric XML documents, like millions of html pages on the WWW, typically have irregular structure and coarse-grained data, and are generally intended for human viewing (e.g., a book). XML documents are also heavily used as containers for data exchange. Data-centric XML documents typically have regular, repetitive data structures and finely-grained data, and are designed mainly for processing by machines (e.g., book orders).

In either case, XML documents are composed of character data and tags used to document the semantics of the embedded text. Tags can be used freely in an XML document (as long as their use conforms to XML specification) or can be used in accordance with *document type definitions* (DTDs) [BPSM98] (to which an XML document declares itself conformant). An XML document that conforms to to a DTD is a *valid* XML document. Logically, each XML document contains one or more *elements*, which are delimited by tags. Each element has a type (identified by its tags) and may have a set of *attributes*.

A DTD can be used to define the kinds and structures of elements that can appear in a valid XML document. Generally speaking, a DTD can include four types of declarations: *element type*, *attribute-list, notation*, and *entity*. An element type declaration names an element and defines its allowable content and structure. An element may contain only other elements (called *element content*) or may contain any mixed of other elements and text (called *mixed content*). An *EMPTY* element type declaration is used to name an element without content (so it can be used, for example, to define a placeholder for attributes). Finally, an element can be declared with content *ANY* meaning the type (content and structure) of the element is arbitrary. Attribute-list declarations define the attributes of an element type. The declaration includes attribute names, types and default values. Two special types of attributes, ID and IDREF, are used to define references between elements. The ID attribute uniquely identifies an element; an IDREF attribute can be used to reference identified element[2]. Entity declarations[3] facilitate flexible organization of XML documents by breaking them into multiple storage units. A notation declaration identifies an external source for an attribute type declaration. In this paper, we assume that readers are familiar with the above terminologies. For more details refer to [BPSM98].

Element and attribute declarations define the structure of compliant XML documents and the relationships among the embedded XML data items. Entity and notation declarations, on the other hand, are used for physical organization of a DTD (similarly to macros and inclusions in many programming languages and word processing documents). Since entity and notation declarations do not provide information pertinent to modeling of the data, they can be substituted or expanded to give an equivalent DTD with only element type and attribute-list declarations. We call such a result a *logical DTD*. For the rest of this paper, we use DTD to refer to a logical DTD. The logical DTD in Example 1 (for books, articles and authors) is used throughout for illustration.

[2]IDREFS can store multiple ID values

[3]XML entities are discussed only in Section 2. For the rest of this paper, *entities* refer to the Entity-Relationship Model.

**Example 1[4]: DTD for Books, Articles, and Authors.**

```
<!ELEMENT book (booktitle, (author* | editor))>
<!ELEMENT booktitle (#PCDATA)>
<!ELEMENT article (title, (author, affiliation?)+,
                   contactauthor?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT contactauthor EMPTY>
  <!ATTLIST contactauthor authorID IDREF #IMPLIED>
<!ELEMENT monograph (title, author, editor)>
<!ELEMENT editor ((book | monograph)*)>
  <!ATTLIST editor name CDATA #REQUIRED>
<!ELEMENT author (name)>
  <!ATTLIST author id ID #REQUIRED>
<!ELEMENT name (firstname?, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT affiliation ANY>
```

## 3 DTD Relationships and the Relational Model

The task of developing a relational schema for an XML document (with associated DTD) requires understanding the components of, and relationships within, the document. We have identified the following properties and relationships embedded in DTDs and their associated XML documents.

[4]This DTD is modified from the book DTD used in [SHT+99]

*Grouping:* Within an element type definition, elements associated within parentheses participate in a *grouping* relationship. This relationship can be further classified as *sequence grouping* or *choice grouping* according to the operator used as delimiters in the grouping (i.e., ',' and '|', respectively). Sequence groups and choice groups (as well as elements) are called *content particles*, and can be placed wherever elements can be used in a DTD. For example, the elements `author` and `editor` have a choice grouping relationship within a `book` element, and the elements `title, author` and `editor` have a sequence grouping relationship in `monograph`.

*Nesting:* An element type definition provides the mechanism for modeling both *aggregation* and *association* relationships among elements. On the one hand, subelements may be *components_of* the defined element (i.e., an aggregation relationship). Alternatively, an element may be a name relating a collection of subelements (i.e., an association). In either case, the relationship can be represented structurally as a hierarchy of elements, so we call both cases a *nesting relationship*[5]. For example, a `monograph` element has nesting relationships with its subelements `title, author` and `editor`.

*Element Referencing:* An attribute-list declaration can also be used to model two functionalities. First, it can be used to define unique element properties which may not fit appropriately in an element type definition. For example, an attribute `pub_kind` might be defined for the element `article` as a place to distinguish a journal article from a conference article (or a workshop article, etc.). Second, *element referencing* relationships are defined by using attributes of type ID and IDREF(s). For example, `contactauthor` has an element reference relationship with `author`.

*Ordering:* We distinguish between the logical ordering among element types that is specified in a DTD and the physical ordering of data elements stored in an XML document. For example, in an XML document compliant to Example 1, a `book` element may be stored as follows:

```
<book> <booktitle>XML RDBMS<booktitle/>
   <author><name><firstname>John<firstname/>
                  <lastname>Smith<lastname/>
           <name/><author/>
   <author><name><firstname>Dave<firstname/>
                  <lastname>Brown<lastname/>
           <name/><author/>
<book/>
```

[5]Thereby choosing the structural view to avoid having to choose a particular semantics for all such relationships.

The DTD specifies that a `booktitle` precedes a list of `authors` (or an `editor`); the XML document itself specifies a particular ordering of the two `author` elements (i.e., John before Dave). To prevent confusion, we use *schema ordering* to refer to the ordering of element types specified in a DTD element type declaration, and *data ordering* to refer to the physical order of data items in an XML document. Note that schema ordering only applies to XML elements; there is no order implied between attributes in XML.

*Existence:* In a DTD, an element type with no content declares the existence of an element with no structure or value for the element type. This kind of virtual element is declared so that attributes can be defined for unique properties of the element or for element referencing relationships.

*Occurrence:* Occurrence indicators (i.e., "?", "*" and "+") indicate optional occurrence or repetition of a content particle in an element definition. For example, the grouping `(book | monograph)` can appear zero or more times in an element `editor`.

These properties and relationships illustrate that a DTD not only defines element types for (conforming) XML documents, but also provides valuable information about relationships between elements. This information can be captured in a relational database, but not all of it will be captured in the relational model. The challenges of mapping a DTD to the relational model arise from the mismatch in types of properties and relationships embedded in DTDs vs. those modeled by relations. For example, a DTD can specify optional elements and repeatability of elements, whereas a relational schema cannot. Also, rows and columns in a relational table are not ordered (although a particular relational implementation may impose an order), but elements in XML documents have to maintain their order. Note that these kinds of properties and relationships are, in effect, constraints on the data (not the model). Thus, we capture this information in metadata (which can be stored in a relational database). In addition, the model mapping between the DTD and relational tables can be tracked as metadata for use in data loading, query optimization, updates, etc. We say more about this metadata later.

## 4 DTD to Relational Mapping

In this section, we present our approach to automatically converting a DTD into a relational schema. Since the entity-relationship (ER) model is well understood[6] and broadly used by professionals in the relational database community, we use it as the target model of our conversion. The subsequent mapping from ER model to relational model is well-studied and can be found in most database textbooks (e.g., [EN89]). To simplify discussion, we assume there exists only one external DTD file for the compliant XML documents and there is no internal DTD in the XML files. This requirement can be achieved by pre-processing XML documents with internal or nested DTDs.

Based on the observations noted in Sections 2 and 3, we developed the following guidelines for mapping a DTD into the ER model:

- Since groups and elements are both content particles, a group can be treated (functionally) as a virtual element. Thus, we can create a new element for each group without changing the meaning of the DTD.
- A nesting relationship in a DTD can be modeled as an ER relationship.
- Elements can be mapped to entities, and element attributes to entity attributes.
- A subelement with type `#PCDATA` that occurs 0 or 1 times in an element type definition can be modeled as an attribute of the element.
- Element references can be modeled as ER relationships.
- A schema ordering relationship can be modeled as an ER relationship between the elements involved.
- Data ordering relationships are not handled in the schema, but can be handled as metadata[7].
- Occurrence and repeatability are properties of content particles so can be saved as metadata (or ignored). Constraints may be defined based on the saved metadata.
- Typically an empty element will have associated attributes, so can be modeled as an entity with attributes.

[6]The ER model is a classic design model for relational databases. We assume readers are familiar with its concepts and terminology.

[7]For example, data ordering could be captured using an *ordering* column in a table to number the data rows.

Based on these guidelines, we give an algorithm (illustrated by Figure 1) for converting a DTD into an ER model. Since we are mainly interested in the schema-related aspects of XML document, we do not address the ordering, occurrence and existence properties of XML documents in this algorithm. This omission does not affect the correctness of our mapping, nor does it have any impact on the feasibility of using relational databases for XML data storage and management. We discuss, in the next section, the use of metadata and constraints to capture these properties.

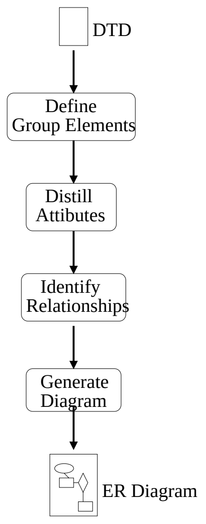

**Figure 1. Algorithm for mapping DTD to ER.**

As illustrated in Figure 1, the algorithm has four sequential steps:

1. **Define Group Elements:** For each group embedded in an element type definition, declare a new element type definition for the group[8] and replace the occurrence of the group with the new element. For example, `<!ELEMENT book (booktitle, (author*|editor))>` is replaced by

```
<!ELEMENT book (booktitle,G1)>
<!ELEMENT G1 (author* | editor)>
```

   This step is repeated until no element contains a group.

2. **Distill Attributes:** If a subelement in an element definition is an element with type "#PCDATA"

[8]The newly defined element is usually an association of the group of subelements.

and it does not occur multiple times, define it as an attribute of the element. If the subelement can occur 0 or 1 times (i.e., with the occurrence indicator ?), the defined attribute is #IMPLIED; otherwise, the attribute is #REQUIRED. For example,

```
<!ELEMENT book (booktitle,G1)>
<!ELEMENT booktitle (#PCDATA)>
```

is replaced by

```
<!ELEMENT book (G1)>
<!ATTLIST book
       booktitle (#PCDATA) #REQUIRED>
```

Note that by moving an element to the attribute list, the ordering relationship among elements is lost. However, this relationship could be maintained as a metadata.

3. **Identify Relationships:** In this step we identify and distinguish between the three types of relationships that will map to E-R: grouping, nesting, and element referencing[9]. By the end of this step, the different types of relationships will all be explicitly denoted (with new tag types) and the only declarations in the DTD will be 'empty' and 'any' elements, attribute lists, and relationships.

   **a)** We deal first with the group elements that were built in step 1, and note that the subelements of a group element have a grouping relationship with each other and a nesting relationship with the element from which the group was defined. For example, in step 1, `(author* | editor)` was extracted from `book` as a group; thus, `author` and `editor` have a grouping relationship with each other (identified as `G1`) and a nesting relationship with `book`. We define a NESTED_GROUP relationship to explicitly identify such elements. For example, the result for a book element now becomes:

```
<!ELEMENT book ()>
<!ATTLIST book
          booktitle (#PCDATA) #REQUIRED>
<!NESTED_GROUP NG1 book (author* | editor)>
```

   Note that group `G1` evolved to nested_group `NG1` and, since the parent of the nesting (i.e., `book`) is identified in the relationship, the element book no longer needs to reference the group.

   b. Once nested groups have been identified, the remaining nesting relationships are extracted from parent elements and explicitly denoted with NESTED tags. As for nested_groups, nested relationships specify the parent element and the subelement, and the subelements are removed from the parent ELEMENT definition. For example,

```
<!ELEMENT monograph ()>
<!ATTLIST monograph
          title (#PCDATA) #REQUIRED>
<!NESTED  Nauthor monograph author>
<!NESTED  Neditor monograph editor>
```

   Note that each subelement has a separate nested relationship with the parent.

   c. Finally, REFERENCE relationships are established between elements with IDREF attributes and those with ID attributes. Each attribute of type IDREF is replaced by a REFERENCE declaration that gives the attribute name, the element type of the IDREF and the element type(s)[10] of the places that can be referenced. For example,

```
<!ATTLIST contactauthor
          authorID IDREF #IMPLIES>
```

   becomes

```
<!REFERENCE authorID
            contactauthor (author)>
```

   Example 2 shows the book-articles-authors DTD after defining group elements, distilling attributes, and identifying relationships.

4. **Generate Diagram:** The converted DTD is mapped in a straightforward way into an E-R diagram. For each element defined in the DTD, an entity is created; attributes of the element are defined as attributes of the entity. Relationships (i.e., nested_group, nesting, and inter-element referencing) are created as follows.

   a. For each nested_group relationship, a relationship node is created; arcs are drawn from the parent element to the relationship node and from the relationship node to each subelements in the nested group. The arcs going out of the relationship node are marked with $\bigcup$ for choice group. In the above mapping,

[9]Recall that we are not handling the ordering relationship at this time, since it is a relationship among the data whereas the relationships we distinguish here all involve XML elements.

[10]Since IDREFs are untyped, an IDREF can reference any element with an ID attribute. Thus, the collection of all element types must be grouped with choice operators (i.e., |)

attributes associated with the nested_group are converted to relationship attributes, accordingly.

b. For each nested relationship, a relationship node is created and the arcs are drawn from the parent element to the relationship node and from the relationship node to the subelement.

c. Finally, for each referencing relationship, a relationship node is created; arcs are drawn from the element with an `IDREF` attribute to the relationship node and from the relationship node to each element with an `ID` attribute. Since the element types with an `ID` attribute is a valid candidate for reference, they form a choice group. Thus, a $\bigcup$ is used to mark the arcs going out of referencing relationships.

### Example 2: Converted DTD.

```
<!ELEMENT book ()>
<!ATTLIST book booktitle (#PCDATA) #REQUIRED>
<!NESTED_GROUP NG1 book (author* | editor)>
<!ELEMENT article ()>
<!ATTLIST article title (#PCDATA) #REQUIRED>
<!NESTED_GROUP NG2 article (author, affiliation?)>
<!NESTED  Ncontactauthor article contactauthor>
<!ELEMENT contactauthor EMPTY>
<!REFERENCE authorID contactauthor (author)>
<!ELEMENT monograph ()>
<!ATTLIST monograph title (#PCDATA) #REQUIRED>
<!NESTED  Nauthor monograph author>
<!NESTED  Neditor monograph editor>
<!ELEMENT editor ()>
  <!ATTLIST editor name CDATA #REQUIRED>
<!NESTED_GROUP NG3 editor (book | monograph)>
<!ELEMENT author ()>
  <!ATTLIST author id ID #REQUIRED>
<!NESTED  Nname author name>
<!ELEMENT name ()>
<!ATTLIST name firstname (#PCDATA) #IMPLIED
               lastname (#PCDATA) #REQUIRED>
<!ELEMENT affiliation ANY>
```

## 5 Next Steps: Technical Issues and Discussion

In addition to the DTD to relational mapping discussed above, there are many technical issues to be overcome in order to bring XML data into a relational database for management. The following are areas where we see efforts need to be put forth:

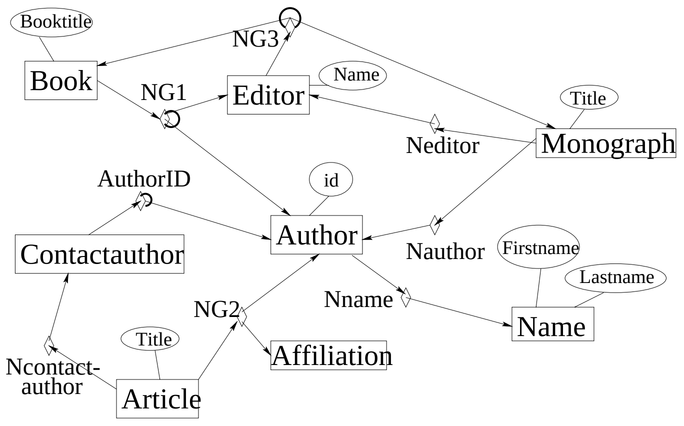

**Figure 2. E-R Diagram for the example DTD.**

**Metadata:** DTDs and XML documents contain relationships and properties which can not be easily captured by the relational model, e.g., ordering relationships and occurrence properties of elements. To compensate the lost of such information, metadata can be collected at the time of DTD to relational mapping and stored as relational tables. The element type and attribute list declarations and embedded relationships in a DTD should be maintained as metadata so they can be used later for data loading and query optimization.

**Data Loading:** There are many tools (e.g., XML parser and XSL) which can be used to facilitate implementation of data loading programs. Also, Document Object Model (DOM) [Gro98] has provided a platform- and language-neutral interface for programs to access a XML document as a tree. Thus, the process of loading the XML data into a relational database can be realized by an algorithm that traverses the DOM tree to download data items (the document contents) into relational table. In addition to functionally mapping elements in XML documents into the corresponding relational tables, the algorithm may use the metadata collected during DTD to relational mapping to establish relationships among relational tables. Also, the metadata may be augmented and updated during the data loading process. There are some properties of the data (e.g., physical ordering relationship) need to be collected and maintained in the metadata tables, which may be useful for later query processing and optimization, e.g., which tables should be joined in order to answer certain queries.

**Query Processing:** W3C is currently working on scripting and querying facilities (e.g., XSL and XQL) to allow users retrieve information stored in XML documents. By storing XML data in relational database, users may use SQL as a replacement of these facilities. However, there are concerns regarding to the impact brought by semantics and structures of XML data to relational database and the subsequent query processing. Is there a need of index structures, different from the traditional ones, for XML data? How is the performance on querying and searching the XML data and documents in relational databases comparing to directly querying the XML documents? Finally, assuming a querying language for XML documents (e.g., XQL or XML-QL) are standardized, how do we transform the XQL or XML-QL queries into "meaningful" SQL queries?

## 6 Related Work and Conclusion

This paper presents our initial steps investigating the problem of managing XML data in relational databases. Other groups working on using database management techniques to manage XML data include [GMP$^+$94, DFS99, SHT$^+$99, Bou99, Wid99]. Of these, [DFS99] and [SHT$^+$99] are most closely related to our work. The STORED [DFS99] project also studies how to store large volumes of XML data in relational databases. A major difference between the STORED work and ours is our use of a DTD for the mapping. Their approach is based on mining over a large number of XML documents of the same type. Also, they use an overflow graph for the parts of the XML documents that do not fit into the relational schema.

Shanmugasundaram et al. [SHT$^+$99] also investigated schema conversion techniques for mapping a DTD to a relational schema. They give (and compare) three alternative methods based on traversal of a DTD tree and creation of element trees. We believe our approach captures more of the structure, properties and embedded relationships among elements in XML documents However, further detailed analysis and performance evaluation are needed to compare the pros and cons of these two approaches.

A major contribution of our work is our analysis of the structure and semantic characteristics of DTDs and XML documents. We proposed an algorithm, based on our findings, to convert a DTD into a corresponding entity-relational model for relational databases. While our proposal results in loss of some information, such as ordering and occurrence, this can be compensated by extending our method to store the additional information as metadata and develop constraints for validation.

We examined XML documents with the goal of managing XML data in relational database systems. Our current focus is on loading the XML data into a relational database in accordance with the model result of this work, and focus on processing queries for XML data. We are designing a relational mechanism for collecting and maintaining critical metadata. Moreover, we need to assess the impact of the semantics and structures of XML data on relational query optimization. For example, is there a need for separate index structures for XML data? We also plan to assess the performance issues in managing XML data in relational database systems.

## References

[Bou99] Ronald Bourret. Xml and databases. http://www.informatik.ty-darmstadt.de/DVS1/staff/bourret/xml/XMLAndDatabases.htm, September 1999.

[BPSM98] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensaible markup language (xml) 1.0. http://www.w3.org/TR/REC-xml, Feburary 1998.

[DFF$^+$99] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for xml. In *Proceedings of the Eighth International World Wide Web Conference (WWW-8)*, 1999.

[DFS99] Alin Deutsch, Mary F. Fernandez, and Dan Suciu. Storing semistructured data with stored. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 431–442, Philadephia, USA, June 1999.

[EN89] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems.* The Benjamin/Cummings Publishing Company, Inc, 1989.

[GMP$^+$94] G.E.Blake, M.P.Consens, P.Kilpelinen, P.-A.Larson, T.Snider, and F.W.Tompa. Text/relational database management systems: Harmonizing sql and sgml. In *Proceedings of the Conference on Applications of Databases (ADB-94)*, pages 267–280, Vadstena, June 1994.

[Gro98] W3C DOM Working Group. Document object model (dom). http://www.w3.org/TR/REC-DOM-Level-1/, October 1998.

[Gro99] W3C XSL Working Group. Extensible stylesheet language (xsl) specification. http://http://www.w3.org/TR/WD-xsl/, April 1999.

[SHT$^+$99] J. Shanmugasundaram, G. He, K. Tufte, C. Zhang, D. DeWitt, and J. Naughton. Relational databases for querying xml documents: Limitations and opportunities. In *Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99)*, pages 302–314, Edinburgh, Scotland, UK, September 1999.

[Wid99] J. Widom. Data management for xml - research directions. *IEEE Data Engineering Bulletin, Special Issue on XML*, 22(3), 1999.