# Path Materialization Revisited: An Efficient Storage Model for XML Data

**Haifeng Jiang**[†] **Hongjun Lu**[†] **Wei Wang**[†] **Jeffrey Xu Yu**[‡]

[†]Dept. of Computer Science
The Hong Kong University of Science and Technology,
Clear Water Bay, Hong Kong, China
Email: {jianghf, luhj, fervvac}@cs.ust.hk

[‡]Dept. of Systems Engineering and Engineering Management
The Chinese University of Hong Kong, Hong Kong, China
Email: yu@se.cuhk.edu.hk

## Abstract

XML is emerging as a new major standard for representing data on the world wide web. Several XML storage models have been proposed to store XML data in different database management systems. The unique feature of model-mapping-based approaches is that no DTD information is required for XML data storage. In this paper, we present a new model-mapping-based storage model, called XParent. Unlike the existing work on model-mapping-based approaches that emphasized on converting XML documents to/from database schema and translation of XML queries into SQL queries, in this paper, we focus ourselves on the effectiveness of storage models in terms of query processing. We study the key issues that affect query performance, namely, storage schema design (storing XML data across multiple tables) and path materialization (storing path information in databases). We show that similar but different storage models significantly affect query performance. A performance study is conducted using three data sets and query sets. The experimental results are presented.

*Keywords:* Semistructured data, XML database

## 1 Introduction

The Extensible Markup Language (XML) is an emerging standard for data representation and exchange on the Internet. There is a consensus among the researchers that XML is as an easy-to-write, easy-to-parse language to exchange data in a variety of applications on the Internet. Several XML query languages were proposed including Lorel [Abiteboul et al., 1997], XML_QL [Deutsch et al., 1999a], XML-GL [Ceri et al., 1999], Quilt [Chamberlin et al., 2000], YATL [Cluet et al., 1998], XPath [Clark and DeRose, 1999] and XQuery [Chamberlin et al., 2001]. The surveys on XML schema languages and query languages can be found in [Bonifati and Ceri, 2000, Lee and Chu, 2000a].

In order to facilitate the task of querying XML documents, efficient storage models for storing XML documents in database management systems were studied [McHugh et al., 1997, Fernandez et al., 1998, Kanne and Moerkotte, 2000, Florescu and Kossmann, 1999, Schmidt et al., 2000, Deutsch et al., 1999b, Deutsch et al., 1999c, Shanmugasundaram et al., 1999, Kappel et al., 2000, Lee and Chu, 2000b]. When XML documents are stored in off-the-shelf database management systems, the problem of storage model design for storing XML data becomes a database schema design problem. In [YoshiKawa and Amagasa, 2001], the authors categorize such database schemas into two categories: *structure-mapping approach* and *model-mapping approach*. In the former, the design of database schema is based on the understanding of DTD (Document Type Descriptor) that describes the structure of XML documents. Examples include [Shanmugasundaram et al., 1999, Kappel et al., 2000, Lee and Chu, 2000b]. In the latter a fixed database schema is used to store any XML documents without assistance of DTD, such as [Kanne and Moerkotte, 2000, Florescu and Kossmann, 1999, Schmidt et al., 2000, YoshiKawa and Amagasa, 2001]. As indicated in [YoshiKawa and Amagasa, 2001], the main advantages of the model-mapping approaches are, i) it is capable of supporting any sophisticated XML applications that are considered either as static (the DTDs are not changed) or dynamic (the DTDs vary from time to time); ii) it is capable of supporting well-formed but non-DTD XML applications; and iii) it does not require extending the expressive power of database models, in order to support XML documents. Therefore, it is possible to store large XML documents in off-the-self database management systems.

In this paper, as a new model-mapping approach, we propose a database schema, called XParent. The unique feature of XParent is: a) it is a four table schema; b) it explicitly maintains both label-paths (sequences of element tags) and data-paths (sequences of elements) in two separate but inter-related tables. The label-paths give a global view on the XML documents stored in the database management system. As for the data-paths, two formats are considered. The first keeps parent-child relationships. The second further materializes it by maintaining ancestor-descendant relationships. We call them path materialization. The rest of the paper is organized as follows. Section 2 discusses an XML data model. Section 3 briefly introduces three existing model-mapping approaches with discussions. Section 4 introduces a new approach called XParent. In Section 5, we discuss query processing issues. The outline of our prototype system is given in Section 6. In Section 7, some results from our performance studies are presented. We conclude the paper in Section 8.

## 2 An XML Data Model

We adopt the data model of XPath to represent XML documents [Clark and DeRose, 1999]. The

XPath data model models XML documents as an ordered tree using 7 types of nodes, namely, root, element, text, attribute, namespace, processing-instruction and comment. The full specification can be found in [Chamberlin et al., 2001]. In this paper, without loss of generality, we emphasize on four types: root, element, text and attribute. A simple example in Figure 1 illustrates the four node types, using similar symbols as those used in [YoshiKawa and Amagasa, 2001]. The root node is a virtual node pointing to the root element of an XML document. Elements in an XML document are represented as an element node with an expanded-name (the element-type name specified in the tag). Element nodes can have $n$ ($n \geq 0$) other elements or text as its children. A text node or an element can only have one parent. Text nodes are string-valued leaf-nodes. Text nodes do not have any child nodes. An element node can have a set of attribute nodes. An attribute node has an attribute-name and an attribute-value. Attribute nodes do not have any children. It is interesting to know that, despite of the fact that an element node is the parent of a set of attribute nodes associated with, the attribute nodes are not children of the element node by definition. The IDREF attribute nodes are used for intra-document references.

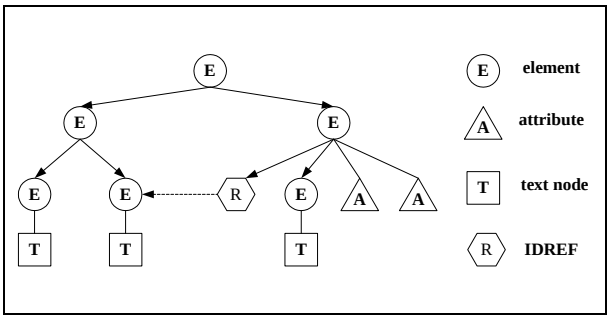



Figure 1: The Four Widely-used XPath Node Types

In this paper, we call such a data representation as an *XML data graph*. Figure 2 shows an XML data graph for the following XML document.

```
<DBGroup>
   <Member>
      <Name> Fervvac </Name> <Age> 23 </Age>
      <Office> CSD 4212 </Office>
   </Member>
   <Member>
      <Name> Daniel </Name>
      <Office>
         <Building> CS </Building> <Room> 4215 </Room>
      </Office>
   </Member>
   <Member Project=105>
      <Name> Ryan </Name> <Age> 24 </Age>
      <Office> CSD 4212 </Office>
   </Member>
   <Project id = 105> <Title> XML </Title> </Project>
   <Project> <Title> Cube </Title> </Project>
</DBGroup>
```

In Figure 2, the root node of the XML data graph has an edge, with an edge-label `DBGroup`, pointing to the element (&1) which is the real root element of the XML document. All the oval nodes are element nodes. An element node has an element-type name, which is represented as an edge-label of the incoming edge pointing to the element node itself. The text nodes are rectangle nodes. The element node (&4) has an IDREF attribute (`@Project`) pointing to the element node (&5) whose ID attribute (`@Id`) is 105.

We omit the formal definition of the XML data graph. In brief, we introduce the terms we will use

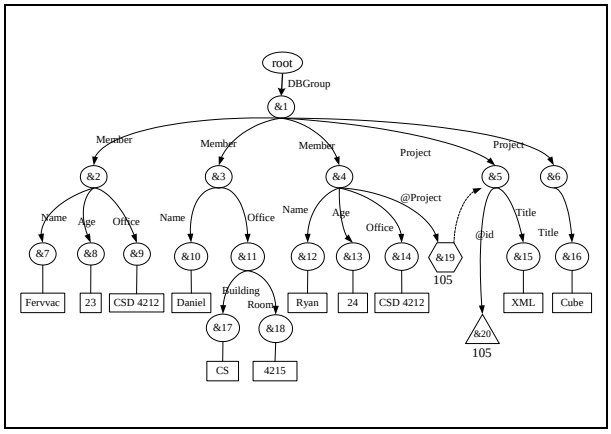



Figure 2: A Data Graph for a Small XML Document

in the following discussions. Because elements of an XML document are ordered, in an XML data graph, the *ordinal* of an element is the order of this element among all siblings that share the same parent. In Figure 2, the ordinals for elements &3 and &4 are 2 and 3, respectively. A *label-path* in an XML data graph is a dot-separated sequence of edge-labels (element-type names). In a similar fashion, a *data-path* is a dot-separated alternating sequence of element nodes. As an example, in Figure 2, the two data-paths, &1.&2.&7 and &1.&3.&10, share the same label-path `DBGroup.Member.Name`. Here, &2 and &3 are the elements with the same edge-label `Member`, and &7 and &10 are the elements with the same edge-label `Name`.

## 3 Three Existing Approaches

Edge [Florescu and Kossmann, 1999] and XRel [YoshiKawa and Amagasa, 2001] are two representatives of the model-mapping approaches that use a fixed database schema to store the structure of XML documents in relational databases. For an XML data graph, let $e_0.e_1.e_2.\cdots.e_n$, where $e_0$ is the virtual root and $e_i$ is the id of an element, be a data-path ($e_0$ is sometimes omitted when we refer to a data-path, for simplicity), and let $l_1.l_2.\cdots.l_n$ be the label path of the data-path. The Edge approach stores pairs of label edge and data edge (a pair of element ids), in a single table: $(l_i, e_{i-1}, e_i)$. A variation of the Edge approach, called Monet, partitions the schema by label-paths [Schmidt et al., 2000]. For a label-path, $l_1.l_2.\cdots.l_n$, Monet creates $n+1$ tables. The first $n$ tables are $l_1$, $l_1.l_2$, $\cdots$, $l_1.l_2.\cdots.l_n$. In a table $l_1.\cdots.l_i$, it keeps all data edges, $(e_{i-1}, e_i)$. The last table to be created is for text or attribute CDATA, $l_1.l_2.\cdots.l_n$.CDATA. A label-path is maintained as a set of table names. With Monet, all the tables are smaller. But the number of tables is huge, because it needs to create a table for each distinctive label-path. Unlike Monet, XRel explicitly keeps all such distinctive label-paths as tuples in a table. XRel does not explicitly store edges, $(e_{i-1}, e_i)$, for data-paths, $e_0.e_1.e_2.\cdots.e_n$. Instead, XRel records containment relationships using the notion of *region*. A region is specified by the start and end positions of a node. If $n_i$ is in the tree rooted at $n_j$, then the region of $n_j$ covers $n_i$. The details of these database schemas are given below.

### 3.1 Edge

The Edge approach [Florescu and Kossmann, 1999] stores XML data graphs (a directed graph) in a table named Edge.

| Src | Ord | Tgt | Label | Flag | Value |
|---|---|---|---|---|---|
| &0 | 1 | &1 | "DBGroup" | ref | – |
| &1 | 1 | &2 | "Member" | ref | – |
| &2 | 1 | &7 | "Name" | val | "Fervvac" |
| &2 | 1 | &8 | "Age" | val | "23" |
| &2 | 1 | &9 | "Office" | val | "CSD 4212" |
| &1 | 2 | &5 | "Member" | ref | – |
| &3 | 1 | &10 | "Name" | val | "Daniel" |
| &3 | 1 | &11 | "Office" | ref | – |
| &11 | 1 | &17 | "Building" | val | "CS" |
| &11 | 1 | &18 | "Room" | val | "4215" |
| &1 | 3 | &4 | "Member" | ref | – |
| &4 | 1 | &12 | "Name" | val | "Ryan" |
| &4 | 1 | &13 | "Age" | val | "24" |
| &4 | 1 | &14 | "Office" | val | "CSD 4212" |
| &4 | 1 | &19 | "@Project" | val | "105" |
| &1 | 1 | &5 | "Project" | ref | – |
| &5 | 1 | &20 | "@id" | val | "105" |
| &5 | 1 | &15 | "Title" | val | "XML" |
| &1 | 2 | &6 | "Project" | ref | – |
| &6 | 1 | &18 | "Title" | val | "Cube" |

Table 1: Edge Table for the Data Graph in Figure 2

```
Edge(Source, Ordinal, Target, Label, Flag, Value)
```

The Edge table keeps all edges of XML data graphs. An edge is specified by two node identifiers, namely, `Source` and `Target`. The `Label` attribute keeps the edge-label of an edge. The `Ordinal` attribute records the ordinal of the edge among its siblings. A `Flag` value indicates whether the target node is an inter-object reference (ref) or points to a value (val). Table 1 shows the Edge table for the XML data graph given in Figure 2. For example, here, the tuple, (&2, 1, &7, "Name", val, "Fervvac"), describes the edge from the element node (&2) to the element node (&7). The edge has an edge-label ("Name" and a value ("Fervvac"). Its ordinal is 1 (the first outgoing edge from the element node &2).

### 3.2 Monet

As a variation of the Edge approach, Monet [Schmidt et al., 2000] stores XML data graphs in multiple tables. In other words, Monet partitions the Edge table according to all possible label-paths. For each unique path, Monet creates a table. For example, the leftmost path in Figure 2 needs to be stored in three tables, namely, `DBGroup.Member`, `DBGroup.Member.Name` and `DBGroup.Member.Name.CDATA`. The first two are for the element nodes and the last one is for the text node. For element nodes, the corresponding element table has three attributes: `Source`, `Target` and `Ordinal`. The `Source` and `Target` attribute together specify a unique edge in an XML data graph. For texts (or attributes), Monet creates a table with two attributes, `Id` and `Value`. Unlike the Edge approach, there is no corresponding `Label` and `Flag` attributes. The table name implicitly specifies its labels and type (flag). The number of tables equals to the number of distinct label-paths. For the XML data graph in Figure 2, the corresponding edge table shown in Table 1 could be partitioned into 18 tables in Monet approach.

### 3.3 XRel

The XRel approach [Kha et al., 2001, YoshiKawa and Amagasa, 2001] stores XML data graphs in four tables. They are `Path`, `Element`, `Text`, and `Attribute`.

```
Path(PathID, Pathexp)
Element(DocId, PathID, Start, End, Ordinal)
Text(DocId, PathID, Start, End, Value)
Attribute(DocId, PathID, Start, End, Value)
```

The database attributes `DocId`, `PathId`, `Start`, `End` and `Value` represent document identifier, simple path expression identifier, start position of a region, end position of a region, and string-value, respectively. The region of a node is the start and end positions of this node in an XML document. The region (or the pair of start and end positions) implies a containment relationship. As given in [YoshiKawa and Amagasa, 2001], with regions, the document order among attribute nodes sharing the same parent element node are left implementation dependent in the specification of the XPath data model. The unique feature of XRel is that no node identifiers are needed to store XML data graphs. Instead, start and end positions are used.

Because the schema for both `Attribute` and `Text` are the same, physically, the XRel stores both text nodes and attribute nodes in the `Text` table. The `Attribute` table is not maintained. The database for the XML data graph (Figure 2) is shown in Table 2. Here, attribute nodes are stored in the `Text` table. We omitted `DocId` attributes in Table 2.

## 4 XParent

In this section, we categorize the above three model-mapping-based approaches, namely, Edge, Monet and XRel, into two categories, edge-oriented and node-oriented. Some observations on the three existing approaches are summarized below.

- **Edge-Oriented Approaches (Edge, Monet)**: The edge-oriented approach maintains edges individually. Therefore, it needs to concatenate the edges to form a path for processing user queries. The single table approach, adopted in Edge, is simple. Because it only keeps edge-label, rather than the label paths, a large number of joins is needed to check edge-connections. Monet creates a large number of tables. A user query is efficiently processed, if it is a simple path query dedicated to a single label-path. For complex queries with multiple label-paths, it needs to conduct joins. Besides, Monet cannot handle regular path expressions involving "*" operator well, due to the nature of the Monet schema. It is also difficult to support XML applications which change its structure by adding or removing paths. Therefore, the benefits of this approach are limited.
- **A Node-Oriented Approach (XRel)**: The node-oriented approach maintains nodes rather than edges. There is no edge information explicitly maintained in this schema. With a concept called region, it maintains a containment relationship. A region is a pair of start and end points of a node. A node, $n_i$, is reachable from another node, $n_j$, if the region of $n_i$ is included in the region of $n_j$. The containment relationship is not for parent-child relationships, but rather ancestor-descendent relationships. The advantage of this approach is that it can easily identify whether a node is in a path from another node by using $\theta$-joins (to test the containment-relationship using $>$, $<$). But $\theta$-joins are considered as more costly than equijoins. In addition, off-the-self database management systems usually do not have a special index mechanism to support containments.

XParent is a four table database schema, `LabelPath`, `DataPath`, `Element` and `Data` as follows. The differences between XParent and others will be discussed later.

| PathID | Start | End | Value |
|---|---|---|---|
| 3 | 4 | 5 | "Fervvac" |
| 4 | 8 | 9 | "23" |
| 5 | 12 | 13 | "CSD 4212" |
| 3 | 18 | 19 | "Daniel" |
| 6 | 23 | 24 | "CS" |
| 7 | 27 | 28 | "4215" |
| 8 | 34 | 35 | "105" |
| 3 | 38 | 39 | "Ryan" |
| 4 | 42 | 43 | "24" |
| 5 | 46 | 47 | "CSD 4212" |
| 10 | 52 | 53 | "105" |
| 11 | 56 | 57 | "XML" |
| 11 | 62 | 63 | "Cube" |

(a) Text Table

| PathID | Start | End | Ordinal |
|---|---|---|---|
| 3 | 3 | 6 | 1 |
| 4 | 7 | 10 | 1 |
| 5 | 11 | 14 | 1 |
| 2 | 2 | 15 | 1 |
| 3 | 17 | 20 | 1 |
| 6 | 22 | 25 | 1 |
| 7 | 26 | 29 | 1 |
| 5 | 21 | 30 | 1 |
| 2 | 16 | 31 | 2 |
| 8 | 33 | 36 | 1 |
| 3 | 37 | 40 | 1 |
| 4 | 41 | 44 | 1 |
| 5 | 45 | 48 | 1 |
| 2 | 32 | 49 | 3 |
| 10 | 51 | 54 | 1 |
| 11 | 55 | 58 | 1 |
| 9 | 50 | 59 | 1 |
| 11 | 61 | 64 | 1 |
| 9 | 60 | 65 | 2 |
| 1 | 1 | 66 | 1 |

(b) Element Table

| PathID | Pathexp |
|---|---|
| 1 | #/DBGroup |
| 2 | #/DBGroup#/Member |
| 3 | #/DBGroup#/Member#/Name |
| 4 | #/DBGroup#/Member#/Age |
| 5 | #/DBGroup#/Member#/Office |
| 6 | #/DBGroup#/Member#/Office#/Building |
| 7 | #/DBGroup#/Member#/Office#/Room |
| 8 | #/DBGroup#/Member#/@Project |
| 9 | #/DBGroup#/Project |
| 10 | #/DBGroup#/Project#/@Id |
| 11 | #/DBGroup#/Project#/Title |

(c) Path Table

Table 2: The XRel schema for the XML data graph in Figure 2. Both attribute nodes and text nodes are kept in the text table as indicated in XRel. We omit the DocId attribute in the text and element tables.

```
LabelPath(ID, Len, Path)
DataPath(Pid, Cid)
Element(PathID, Did, Ordinal)
Data(PathID, Did, Ordinal, Value)
```

The table `LabelPath` stores label-paths. Each label-path, identified by a unique `ID`, is stored in the attribute `Path`. The number of edges of the label-path is recorded in the attribute `Len`. Because data-paths usually vary in length, and can be very long, we store pairs of node identifiers in the table `DataPath`. Here, `Pid` and `Cid` are parent-node id and child-node id of an edge in the data-path. Such a data-path table is indeed a `Parent` table. In `Element` and `Data` tables, `PathID` is a foreign key of the `ID` in table `LabelPath`, which identifies the label-path of a node. The `Did` (for data-path id) is a node identifier which also serves as unique data-path identifier for the data-path ended at the node itself. `DataPath` keeps parent-child relationships. Therefore, it needs joins in order to check edge connections. In order to speed up such processing, in addition, an `Ancestor table` is proposed which maintains ancestor-descendant relationship.

```
Ancestor(Did, Ancestor, Level)
```

where, `Did` is a data-path identifier which identifies a data path. `Ancestor` and `Level` maintains ancestor-descendant relationships. For example, a tuple, (&9, &1, 2), indicates that the grandfather of the node &9 is the node &1.

Table 3 shows the tables that store the XML data graph shown in Figure 2. Like XRel [YoshiKawa and Amagasa, 2001], the reason we use './' to separate labels is to facilitate regular path evaluation with SQL correctly. The `Ancestor` table is shown in Table 4.

The `Parent` table and `Ancestor` table are two alternatives to store the data-paths of XML data. Although the `Ancestor` alternative is supposed to speed up the query processing time due to its capability of quickly determining ancestor-descendant relationship between elements, it will need much more disk space to store the data-paths than the `Parent` method. Another problem with `Ancestor` table is that it introduces overhead for updates because redundant information is kept in the table. The choice between `Parent` and `Ancestor` depends on different applications. In this paper, we will focus on the `Parent` method. More detailed analysis on `Ancestor` alternative will be our future work.

The features of XParent are as follows.

1. XParent is an edge-oriented approach. The XParent schema explicitly supports label-paths and data-paths in two different tables, `LabelPath` and `DataPath`. The `LabelPath` keeps all distinctive label paths as tuples. The `DataPath` keeps the core structure of XML data graphs based on a parent-child relationship. It can be further materialized into an `Ancestor` table to support ancestor-descendant relationships. Elements and data (texts or attributes) are attached to data-paths. A data-path is identified by the end node identifier (`Did` for data-path id).

2. Unlike Edge, XParent keeps the distinctive label-paths as data in a table. With this information, XParent can easily process regular path queries, for example, `DBGroup.*.Name`. For this query, XParent can efficiently identify all the path identifiers that match `DBGroup.*.Name` in the `LabelPath` table. Then, by utilizing the path identifiers, XParent can identify the values attached to the end of those paths in the `Data` table.

| Id | Len | Path |
|---|---|---|
| 1 | 1 | ./DBGroup |
| 2 | 2 | ./DBGroup./Member |
| 3 | 3 | ./DBGroup./Member./Name |
| 4 | 3 | ./DBGroup./Member./Age |
| 5 | 3 | ./DBGroup./Member./Office |
| 6 | 4 | ./DBGroup./Member./Office./Building |
| 7 | 4 | ./DBGroup./Member./Office./Room |
| 8 | 3 | ./DBGroup./Member./@Project |
| 9 | 2 | ./DBGroup./Project |
| 10 | 3 | ./DBGroup./Project./@id |
| 11 | 3 | ./DBGroup./Project./Title |

(a) LabelPath table

| PathID | Did | Ordinal | Value |
|---|---|---|---|
| 3 | &7 | 1 | "Fervvac" |
| 4 | &8 | 1 | "23" |
| 5 | &9 | 1 | "CSD 4212" |
| 3 | &10 | 1 | "Daniel" |
| 6 | &17 | 1 | "CS" |
| 7 | &18 | 1 | "4215" |
| 8 | &19 | 1 | "105" |
| 3 | &12 | 1 | "Ryan" |
| 4 | &13 | 1 | "24" |
| 5 | &14 | 1 | "CSD 4212" |
| 10 | &20 | 1 | "105" |
| 11 | &15 | 1 | "XML" |
| 11 | &16 | 1 | "Cube" |

(b) Data table

| PathID | Ordinal | Did |
|---|---|---|
| 1 | 1 | &1 |
| 2 | 1 | &2 |
| 2 | 2 | &3 |
| 2 | 3 | &4 |
| 3 | 1 | &7 |
| 3 | 1 | &10 |
| 3 | 1 | &12 |
| 4 | 1 | &8 |
| 4 | 1 | &13 |
| 5 | 1 | &9 |
| 5 | 1 | &11 |
| 5 | 1 | &14 |
| 6 | 1 | &17 |
| 7 | 1 | &18 |
| 8 | 1 | &19 |
| 9 | 1 | &5 |
| 10 | 1 | &20 |
| 11 | 1 | &15 |
| 9 | 2 | &6 |
| 11 | 1 | &16 |

(c) Element table

| Pid | Cid |
|---|---|
| &1 | &2 |
| &1 | &3 |
| &1 | &4 |
| &1 | &5 |
| &1 | &6 |
| &2 | &7 |
| &2 | &8 |
| &2 | &9 |
| &3 | &10 |
| &3 | &11 |
| &4 | &19 |
| &4 | &12 |
| &4 | &13 |
| &4 | &14 |
| &5 | &20 |
| &5 | &15 |
| &6 | &16 |
| &11 | &17 |
| &11 | &18 |

(d) Data-Path table

Table 3: The XParent Schema for the XML Data Graph in Figure 2.

| Did | Ancestor | Level | Did | Ancestor | Level |
|---|---|---|---|---|---|
| &2 | &1 | 1 | &12 | &1 | 2 |
| &3 | &1 | 1 | &13 | &4 | 1 |
| &4 | &1 | 1 | &13 | &1 | 2 |
| &5 | &1 | 1 | &14 | &4 | 1 |
| &6 | &1 | 1 | &14 | &1 | 2 |
| &7 | &2 | 1 | &20 | &5 | 1 |
| &7 | &1 | 2 | &20 | &1 | 2 |
| &8 | &2 | 1 | &15 | &5 | 1 |
| &8 | &1 | 2 | &15 | &1 | 2 |
| &9 | &2 | 1 | &16 | &6 | 1 |
| &9 | &1 | 2 | &16 | &1 | 2 |
| &10 | &3 | 1 | &17 | &11 | 1 |
| &10 | &1 | 2 | &17 | &3 | 2 |
| &11 | &3 | 1 | &17 | &1 | 3 |
| &11 | &1 | 2 | &18 | &11 | 1 |
| &19 | &4 | 1 | &18 | &3 | 2 |
| &19 | &1 | 2 | &18 | &1 | 3 |
| &12 | &4 | 1 | | | |

Table 4: The Ancestor Table for the XML Data Graph in Figure 2

3. Third, unlike Monet, XParent manages data-paths in a single table. For simple single path queries, this approach is inferior to Monet. However, XParent is expected to outperform Monet for regular path queries, which are the main force of XML queries. For example, an XML query may involve two "*" paths. With Monet, a "*" path matches a set of tables. The other "*" path matches another set of tables. If there is a join condition on the data bound to these two "*" paths, every table in a set of tables needs to join all tables in the other set. In addition, because all label-paths are treated as relation names, an addition software module is required to analyze the label-paths and identify relations to be used.
4. XParent uses the similar schema like XRel. The `Did` (data-path id) replaces the start and end pairs used in XRel. XParent can be efficiently supported using the conventional index mechanisms like B-tree. Equijoins will be used instead of $\theta$-joins. The query processing issues will be discussed in the following section.

## 5 Query Processing

In this section, we will have a brief discussion on how XML queries are translated into SQL statements for different mapping schemas and the operations involved in the SQL statements. An XML query is given below using the XQuery syntax [Chamberlin et al., 2001], for simplicity.

**Example 1** *Given the XML data graph in Figure 2. Select the names of all members whose ages are greater than 20.*

```
Q1:     /DBGroup/Member[Age>20]/Name
```

Here, `[]` specifies a condition. The symbol "/" represents direct containment (parent-child relationship). In particular, the first "/" indicates that `DBGroup` is a root element. Two label-paths are involved in this example, `DBGroup.Member.Age` and `DBGroup.Member.Name`. The condition is on the former label-path, and the results (member names) are from the latter label-path (Refer to Figure 2). The result set contains names of those members whose ages are greater than 20. Because all data are stored in tables in a relational database management system, for this query, the system needs to test all (`Age, Name`) pairs. First the pairs must be for the same `Member`. Second, the `Age` value is greater than 20. However,

due to differences in database schemas, in order to process this query, Edge uses 6 selections and 3 equijoins. Monet uses 1 selection and 4 joins. XRel uses 4 selections and 7 joins (3 equijoins and 4 $\theta$-joins). XParent uses 3 selections and 5 equijoins.

**SQL 1** A translated SQL query for the XML query `Q1` using Edge.

```
select name.Value
from Edge dbgroup, Edge member,
     Edge age, Edge name
where dbgroup.Label = 'DBGroup'
     and member.Label = 'Member'
     and age.Label = 'Age'
     and name.Label = 'Name'
     and dbgroup.Source = 0
     and dbgroup.Target = member.Source
     and member.Target = age.Source
     and member.Target = name.Source
     and cast(age.Value as int) > 20
```

SQL 1 shows the translated SQL query using the Edge schema. It needs to join the Edge table repeatedly, in order to ensure the edge connections. The 3 equijoins are for the following edge connections: `DBGroup.Member`, `Member.Name`, and `Member.Age`. The 5 selections are for the distinctive label edges involved in the above edge connections, `DBGroup`, `Member`, `Name` and `Age`. A selection is for checking age values. The number of joins and selections are determined by the length of label paths (the number of edge connections).

**SQL 2** A translated SQL query for the XML query `Q1` using Monet.

```
select cn.Value
from DBGroup.Member.Name n,
     DBGroup.Member.Age a,
     DBGroup.Member m,
     DBGroup.Member.Name.CDATA cn,
     DBGroup.Member.Age.CDATA ca
where m.Target = n.Source
     and m.Target = a.Source
     and a.Target = ca.Id
     and n.Target = cn.Id
     and cast(ca.Value as int) > 20
```

SQL 2 shows the translated SQL query using the Monet schema. Monet uses a large number of small relations. It is important to know that, even for simple path queries, the number of joins is not small. In this simple example, Monet keeps all names and ages in two tables, `DBGroup.Member.Name.CDATA` and `DBGroup.Member.Age.CDATA`. However, it needs to join `DBGroup.Member`, `DBGroup.Member.Age` and `DBGroup.Member.Name` where `DBGroup.Member` is the least upper bound of the two label-paths, `DBGroup.Member.Age` and `DBGroup.Member.Name`. Obviously, if the least upper bound ancestor is far away from the corresponding CDATA tables, more equijoins need to be used.

SQL 3 shows the translated SQL query using the XRel schema. It first finds path identifiers for the following three label paths: `#/DBGroup#/Member`, `#/DBGroup#/Member#/Age`, and `#/DBGroup#/Member#/Name` using 3 equijoins. With these label-path identifiers, it then finds the elements (or data) for each label-path identifier. Finally, it uses the start and end positions to check the containment-relationship. Four $\theta$-joins are used for checking `Member.Age` and `Member.Name` connections. Totally, 7 joins are used. The advantage of

**SQL 3** A translated SQL query for the XML query `Q1` using XRel.

```
select v2.Value
from Element e1, Path p1, Path p2,
     Path p3, Text v1, Text v2
where p1.Pathexp = '#/DBGroup#/Member'
     and p2.Pathexp =
          '#/DBGroup#/Member#/Age'
     and p3.Pathexp =
          '#/DBGroup#/Member#/Name'
     and e1.PathID = p1.PathID
     and v1.PathID = p2.PathID
     and v2.PathID = p3.PathID
     and e1.Start < v1.Start
     and e1.End > v1.End
     and e1.Start < v2.Start
     and e1.End > v2.End
     and cast(v1.Value as int) > 20
```

XRel approach is that it, sometime, does not need to check all edge connections one-by-one from $e_1$ to $e_n$ for $e_1.e_2.\cdots.e_n$, if it only needs to check whether $e_n$ is reachable from $e_1$. However, when multiple branches are involved and multiple conditions on different branches are specified, XRel needs to use more $\theta$-joins to check edge-connections. Also, some of the containment information can be determined based on label-paths, which makes the start and end pairs somehow less important as they should be.

SQL 4 shows the translated SQL query using the XParent schema. In a similar fashion, like SQL 3, SQL 4 first uses three joins to identify the three path identifiers. Then, it uses three joins to check edge connections. Totally 5 equijoins are used.

**SQL 4** A translated SQL query for the XML query `Q1` using XParent.

```
select d2.Value
from Data d1, Data d2, Element e1,
     LabelPath lp1, LabelPath lp2,
     DataPath p1, DataPath p2
where lp1.Path = './DBGroup./Member./Age'
     and lp2.Path =
          './DBGroup./Member./Name'
     and cast(d1.Value as int) > 20
     and d1.PathID = lp1.Id
     and d2.PathID = lp2.Id
     and d1.Did = p1.Cid
     and d2.Did = p2.Cid
     and p1.Pid = p2.Pid
```

As our future work, we will further study the effectiveness of those database schemas by analyzing different queries, for example, lengths of paths, complexity of regular path queries, etc. In this paper, as our preliminary study, we have implemented a prototype system. We have conducted extensive performance studies using three different data sets and query sets. In the following section, in brief, we give the outline of our prototype system followed by a performance study.

## 6 The System Outline

Figure 3 outlines the architecture of our XML management system. In addition to the underlying RDBMS, there are three main components: XML Data Loader, XML Query Interface and XML Query Translator. The XML Data Loader takes an XML

document, parses it and temporarily stores its corresponding XML data graphs into four files. A file corresponds to a single table following our XParent schema. Then these four files are bulk-loaded into the RDBMS. With the XML Query Interface, users can browse structures of XML documents and issue XML queries to query the XML documents. The Query Translator translates an XML query into a corresponding SQL query.

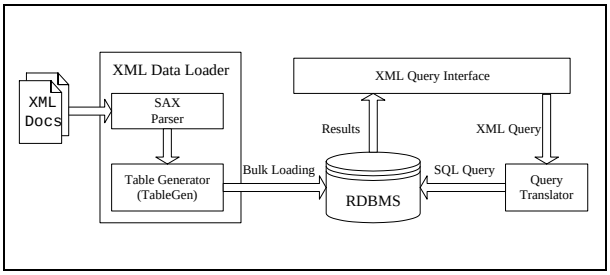



Figure 3: The Prototype of the XML Management System

The XML Data Loader component, as shown in Figure 3, consists of two modules: an SAX parser and a table generator module (TableGen). The SAX parser (the Simple API for XML) is a standard interface for event-based XML parsing. The SAX parser is available in different programming languages. We used *Xerces-C SAX*[1] as the event-based XML parser. The *Xerces-C SAX* is an implementation of the SAX 1.0/2.0 specification using the C programming language. It parses an XML document on the fly and returns event identifiers such as `START_ELEMENT`, `END_ELEMENT`, `ATTRIBUTE` and `CHARACTERS`. The table generator module, TableGen, generates XParent data files based on SAX events. Our current implementation of Query Translator supports the core of XQuery. We use XQuery because it is declarative, "relationally complete" and simple (in terms of parsing). Our system can be easily extended to support other languages. We use a query translation technique similar to [YoshiKawa and Amagasa, 2001].

## 7 A Performance Study

To assess the effectiveness of the XParent approach, extensive experimental studies have been conducted. In this section, we study Edge, XRel and XParent, and report some of the results.

All the experiments were conducted on a 450MHz PC with 256M RAM, 30G hard disk. The RDBMS used was Microsoft SQL Server 7.0. We conducted our experiments using three different sets of XML data and queries. They are summarized below.

- **DBLP**: The first data set is DBLP bibliography files.[2] Three different sizes of the data set are used, namely, **DBLP1** (5.1MB), **DBLP2** (10.4MB) and **DBLP3** (21.1MB). The data size of **DBLP2** is double of **DBLP1**, and the data size of **DBLP3** is double of **DBLP2**.
- **SHAKES**: The second data set is the Bosak Shakespeare collection.[3].
- **BENCH**: The third data set is from the XML benchmark project [Schmidt et al., 2001]. We generated the benchmark data with different scale factors. Two different sizes of data are used: the whole BENCH data set (indicated as **BENCH1.0**), and 40% of the original BENCH data set (indicated as **BENCH0.4**).

Table 5 shows some detailed information about the data sets used.

| Name | Description | Size(MB) | #Query |
|---|---|---|---|
| SHAKES | Shakespeare | 7.5 | 8 |
| DBLP1 | DBLP Set 1 | 5.1 | 8 |
| DBLP2 | DBLP Set 2 | 10.4 | 8 |
| DBLP3 | DBLP Set 3 | 21.1 | 8 |
| BENCH1.0 | sf=1.0 | 113.8 | 10 |
| BENCH0.4 | sf=0.4 | 46.4 | 10 |

Table 5: Details of the Data Sets

Queries were carefully selected for the experiment datasets. Those queries cover different aspects of XML queries for accessing XML data. For testing DBLP data, we used eight queries following the work in [Tian et al., 2000]. The same set of queries was selected from XRel [YoshiKawa and Amagasa, 2001] for testing the Shakespeare data set. For the Benchmark data set, 10 queries were selected for testing. We code each query with "QXN", where 'X' is one of 'D'(DBLP), 'S'(Shakespeare), or 'B'(Benchmark Data), and 'N' is the query number within the respective dataset.

Indexes on relational tables were also properly built to improve query processing as follows:

- For Edge, we created indexes as proposed in [Florescu and Kossmann, 1999].
- In XParent, a clustered index was built on `Element.PathId` and `Data.PathId` to efficiently select tuples with particular PathIds. $B^+$-tree indexes were created on `DataPath.Pid` and `DataPath.Cid` to speed up self-joins of the `DataPath` table. Another $B^+$-tree index was created on `Data.Value`.
- For XRel, a clustered composite index was built on `Element(PathId,Start,End)` and `Data.(PathId, Start, End)`. A $B^+$-tree index was created on `Text.Value`

### 7.1 Edge-Oriented vs Node-Oriented

In the section, we studied two edge-oriented approaches, Edge and XParent, in comparison with XRel. We did not compare with Monet, because Monet did not use a fixed schema.

In [YoshiKawa and Amagasa, 2001], the authors compared XRel with a variation of the Edge approach called "binary tables with inlining" (the same as shown in Table 1). The binary tables with inlining approach is reported to outperform other variations of the Edge approach [Florescu and Kossmann, 1999]. The experiment results in [YoshiKawa and Amagasa, 2001] showed that XRel is more effective than "binary table with inlining" in most cases, especially when path expressions are long or contain wildcats. We conducted an experimental study using the same data set (SHAKES) and the same queries used in [YoshiKawa and Amagasa, 2001]. The eight queries are listed below.

- **Query1** `/PLAY/ACT`
- **Query2** `/PLAY/ACT/SCENE/SPEECH/LINE/STAGEDIR`
- **Query3** `//SCENE/TITLE`
- **Query4** `//ACT//TITLE`
- **Query5** `/PLAY/ACT[2]`

[1] It is available from `http://xml.apache.org/xerces-c/api.html`

[2] It is available at `ftp://ftp.informatik.uni-trier.de/pub/users/Ley/bib/records.tar.gz`

[3] It is available at `http://metalab.unc.edu/bosak/xml/eg/shaks200.zip`

- **Query6** `(/PLAY/ACT)[2]/TITLE`
- **Query7** `/PLAY/ACT/SCENE/SPEECH[SPEAKER = 'CURIO']`
- **Query8** `/PLAY/ACT/SCENE[//SPEAKER='Steward']/TITLE`

The query elapsed times are shown in Figure 4 (Log-scale). For queries that only contain short, simple paths, such as QS1, QS3[4] and QS5-7, Edge performs similarly to XParent. For other queries, XParent outperforms Edge significantly. QS2 is a simple path query but the path is very long and Edge has to use a lot of joins to trace edge connections. QS4 and QS8 are regular path queries. Nearly the whole XML data tree needs to be traversed in Edge while XParent uses stored label-paths to limit search space. Comparing XParent with XRel, we found that for only one-path queries (QS1-QS5), XParent and XRel have similar performance. For Q7, XParent significantly outperforms XRel (up to 20 times faster).

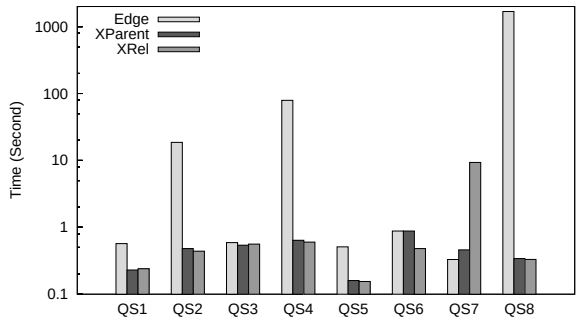



Figure 4: Query Elapsed Time: Edge, XParent and XRel Using SHAKES

We further tested XParent vs XRel using BENCH0.4. The 10 queries used are listed below.

- **Query1** Return the name of the person 'person0'.
- **Query2** Return the initial increases of all open auctions.
- **Query4** List the reserves of those open auctions where a certain person issued a bid before another person.
- **Query8** List the names of persons and the number of items they bought.
- **Query9** List the names of persons and the names of the items they bought in Europe.
- **Query11** For each person, list the number of items currently on sale whose price does not exceed 0.02% of the person's income.
- **Query12** For each person with an income of more than 50000, list the number of items currently on sale whose price does not exceed 0.02% of the person's income.
- **Query17** Which persons don't have a homepage?
- **Query19** Give an alphabetically ordered list of all items along with their location.

Figure 5 shows the query elapsed times, in log-scale. XParent outperforms XRel for all queries except QB16, by an advantage ratio up to 16 (QB2).

To explain the experiment results, we studied SQL query plans used by the relational database management system for XParent and XRel queries. We found that SQL Server used reasonable physical execution plans for almost all the queries of XParent and XRel by starting from most selective branches with its estimation based on statistics. The performance differentiation is mainly caused by equijoins and $\theta$-joins adopted by XParent and XRel respectively. Equijoins in the translated queries were evaluated more efficiently by using proper join algorithms than $\theta$-joins in XRel queries.

XRel slightly outperforms XParent with Q16. In this case, XParent needs 14 equijoins, while XRel needs only 3 equijoins and 4 $\theta$-joins.

As a conclusion, XParent outperforms both Edge and XRel in most cases. Compared with Edge, XParent wins because it keeps distinctive label-paths as data in a table. With this information, XParent can easily process regular path queries. Although XRel also keeps label-paths, it performs much worse than XParent because it uses $\theta$ joins, e.g. $(>, <)$, to determine ancestor-descendant relationships between elements while XParent only needs equijoins for query processing, as shown in section 5. As we know, relational database management systems are well tuned for equijoins, but not for $\theta$-joins.

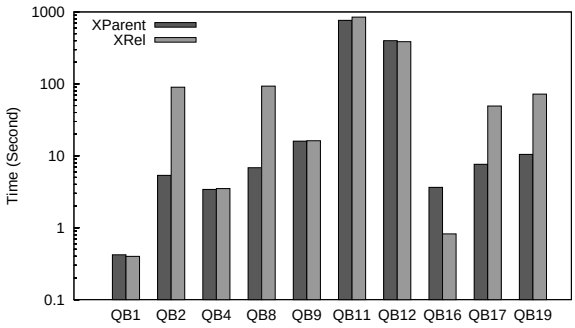



Figure 5: Query Elapsed Time: XRel vs XParent Using BENCH0.4

### 7.2 Scalability Test: XParent vs XRel

In this section, we investigate the scalability of XParent in comparison with XRel using DBLP1, DBLP2 and DBLP3 as listed in Table 5. Detail of the eight selected queries are listed below.

- **Query 1** Select conference papers published in year 2000 on XML.
- **Query 2** Select papers written by Michael Stonebraker.
- **Query 3** Select papers written by Michael Stonebraker or Jim Gray.
- **Query 4** Select papers published between 1990 and 1994, with titles starting with "database".
- **Query 5** Select journal papers that have an cite entry whose label is CARE84.
- **Query 6** Select journal papers longer than 40 pages.
- **Query 7** Select papers by Michael Stonebraker quoted by papers published in year 1994.
- **Query 8** Select papers by Jim Gray that are quoted by Michael Stonebraker.

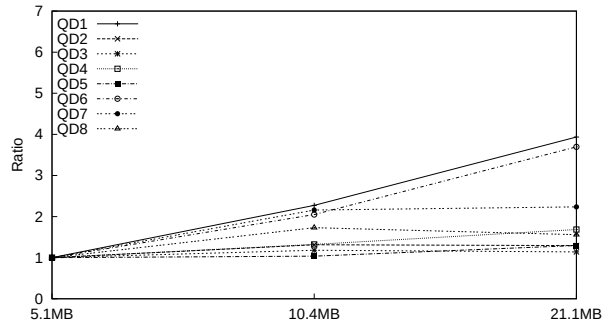
Figure 6 shows the elapsed time ratios for XParent and XRel respectively, where the x-axis shows the data size for DBLP1 (5.1MB), DBLP2 (10.4MB) and DBLP3 (21.1MB). The details for XParent and XRel using DBLP3 data set are given in Figure 7.

The query elapsed time ratios is defined as follows. Suppose the elapsed time of a query using DBLP1 is treated as scale $t_a$, and suppose the elapsed time of the same query is $t_b$, using either DBLP2 or DBLP3. The elapsed time ratio is $t_b/t_a$. One finding is that the query elapsed times for a few queries even go down when data sizes get larger, such as QD3 and QD8 for XParent and QD5 and QD6 for XRel. This is caused by a shift to relatively "better" physical query plans by the query optimizer. The scalability of XParent, in terms of data sizes, is superior to XRel, which is mainly due to the bad scalability of the $\theta$-joins involved in the translated SQL queries for XRel.

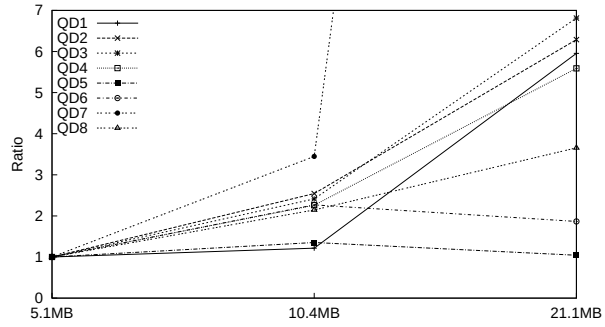
## 8 Conclusion and Future Work

In this paper, we proposed a new model-mapping approach called XParent, which is a four table schema. The unique feature is that it explicitly maintains both label-paths and data-paths in two tables. The label-paths give a global view on the XML documents stored in the database management system. Two

[4]QS3 is processed as a simple path query in Edge because the wildcast is at leading place.




(a) Query Elapsed Time Ratio for XParent




(b) Query Elapsed Time Ratio for XRel

Figure 6: Scalability Test with XParent and XRel Using DBLP1, DBLP2 and DBLP3

forms of data-paths were proposed for parent-child and ancestor-descendant relationships, respectively. As our first experimental study, we studied XParent (based on parent-child relationships), in comparison with Edge and XRel, using three different data sets and query sets. XParent outperforms XRel and Edge in most cases.

We are currently working on developing query translation techniques to translate XML queries into corresponding SQL statements for the XParent schema. Such query translation techniques should also work for other model-mapping-based schemas due to their similarities.

As our future work, we will investigate the efficiency of our ancestor-descendant relationships. We did not compare XParent with structure-mapping-based (or DTD-based) approaches in the performance

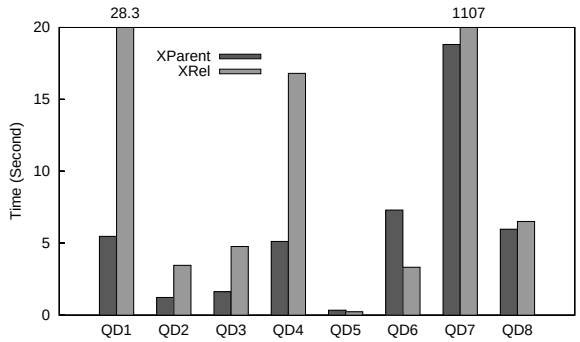



Figure 7: Query Elapsed Time: XParent vs XRel Using DBLP3

study since we focus ourselves on model-mapping-based approaches in this paper. We will conduct analytical studies and investigate the optimal storage models for a well-designed class of XML queries.

## References

[Abiteboul et al., 1997] Abiteboul, S., Quass, D., McHugh, J., Widom, J., and Wiener, J. L. (1997). The lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88.

[Bonifati and Ceri, 2000] Bonifati, A. and Ceri, S. (2000). Comparative analysis of five XML query languages. *SIGMOD Record*, 29(1):68–79.

[Ceri et al., 1999] Ceri, S., Comai, S., Damiani, E., and Fraternali, P. (1999). Xml-gl: a graphical language for querying and restructuring xml documents 1. In *WWW8*.

[Chamberlin et al., 2001] Chamberlin, D., Florescu, D., and et al, J. R. (2001). XQuery: A query language for XML. In *W3C Working Draft*, http://www.w3.org/TR/xquery.

[Chamberlin et al., 2000] Chamberlin, D. D., Robie, J., and Florescu, D. (2000). Quilt: An XML query language for heterogeneous data sources. In *WebDB (Informal Proceedings)*, pages 53–62.

[Clark and DeRose, 1999] Clark, J. and DeRose, S. (1999). XML path language (XPath). In *W3C Recommendation 16 November 1999*, http://www.w3.org/TR/xpath.

[Cluet et al., 1998] Cluet, S., Delobel, C., Simeon, J., and Smaga, K. (1998). Your mediators need data conversion! In *Proceedings of the 24th ACM SIGMOD International Conference on Management of Data*.

[Deutsch et al., 1999a] Deutsch, A., Fernandez, M., and Florescu, D. (1999a). A query language for XML. In *Proceedings of the 8th International World Wide Web Conference*.

[Deutsch et al., 1999b] Deutsch, A., Fernandez, M., and Suciu, D. (1999b). Storing semistructured data in relations. In *Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*.

[Deutsch et al., 1999c] Deutsch, A., Fernandez, M. F., and Suciu, D. (1999c). Storing semistructured data with stored. In *Proceedings of the 25th ACM SIGMOD International Conference on Management of Data*.

[Fernandez et al., 1998] Fernandez, M. F., Florescu, D., Kang, J., Levy, A. Y., and Suciu, D. (1998). Catching the boat with strudel: Experiences with a web-site management system. In *Proceedings of the 24th ACM SIGMOD International Conference on Management of Data*.

[Florescu and Kossmann, 1999] Florescu, D. and Kossmann, D. (1999). A performance evaluation of alternative mapping schemes for storing xml data in a relational database. Technical report.

[Kanne and Moerkotte, 2000] Kanne, C.-C. and Moerkotte, G. (2000). Efficient storage of XML data. In *Proceedings of the International Conference on Data Engineering*.

[Kappel et al., 2000] Kappel, G., Kapsammer, E., Rausch-Schott, S., and Retschitzegger, W. (2000). X-ray - towards integrating xml and relational database systems. In *Proceedings of the International Conference on Conceptual Modeling*.

[Kha et al., 2001] Kha, D. D., Yoshikawa, M., and Uemura, S. (2001). An XML indexing structure with relative region coordinate. pages 313–320.

[Lee and Chu, 2000a] Lee, D. and Chu, W. W. (2000a). Comparative analysis of six XML schema languages. *ACM SIGMOD Record*, (3).

[Lee and Chu, 2000b] Lee, D. and Chu, W. W. (2000b). Constraints-preserving transformation from xml document type definition to relational schema. In *Proceedings of the International Conference on Conceptual Modeling*, pages 323–338.

[McHugh et al., 1997] McHugh, J., Abiteboul, S., Goldman, R., Quass, D., and Widom, J. (1997). Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66.

[Schmidt et al., 2000] Schmidt, A., Kersten, M. L., Windhouwer, M., and Waas, F. (2000). Efficient relational storage and retrieval of XML documents. In *WebDB (Informal Proceedings)*, pages 47–52.

[Schmidt et al., 2001] Schmidt, A. R., Waas, F., Kersten, M. L., Florescu, D., Manolescu, I., Carey, M. J., and Busse, R. (2001). The XML benchmark project. Technical report, CWI, Amsterdam, The Netherlands.

[Shanmugasundaram et al., 1999] Shanmugasundaram, J., Tufte, K., Zhang, C., Gang, H., DeWitt, D. J., and Naughton, J. F. (1999). Relational databases for querying xml documents: Limitations and opportunities. In *Proceedings of the 25th International Conference on Very Large Data Bases*.

[Tian et al., 2000] Tian, F., DeWitt, D. J., Chen, J., and Zhang, C. (2000). The design and performance evaluation of altervative xml storage strategies. Technical report, Computer Science Department, University of Wisconsin-Madison.

[YoshiKawa and Amagasa, 2001] YoshiKawa, M. and Amagasa, T. (2001). XRel: A path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology*, 1(1).