XML-to-SQL Query Translation Literature: The State of the Art and Open Problems

Rajasekar Krishnamurthy, Raghav Kaushik, and Jeffrey F. Naughton

University of Wisconsin-Madison
{sekar,raghav,naughton}@cs.wisc.edu

Abstract. Recently, the database research literature has seen an explosion of publications with the goal of using an RDBMS to store and/or query XML data. The problems addressed and solved in this area are diverse. This diversity renders it difficult to know how the various results presented fit together, and even makes it hard to know what open problems remain. As a first step to rectifying this situation, we present a classification of the problem space and discuss how almost 40 papers fit into this classification. As a result of this study, we find that some basic questions are still open. In particular, for the XML publishing of relational data and for "schema-based" shredding of XML documents into relations, there is no published algorithm for translating even simple path expression queries (with the // axis) into SQL when the XML schema is recursive.

1 Introduction

Beginning in 1999, the database research literature has seen an explosion of publications with the goal of using an RDBMS to store and/or query XML data. The problems addressed and solved in this area are diverse. Some publications deal with using an RDBMS to store XML data; others deal with exporting existing relational data in an XML view. The papers use a wide variety of XML query languages, including subsets of XQuery, XML-QL, XPath, and even "one-off" new proposals; they use a wide variety of languages or ad-hoc constructs to map between the relational and XML schema; and they differ widely in what they "push to SQL" and what they evaluate in middleware.

This diversity renders it difficult to know how the various results presented fit together, and even makes it hard to know what if any open problems remain. As a first step to rectifying this situation, we present a classification of the problem space and discuss how almost 40 papers fit into this classification. As a result of this study, we find that some basic questions are still open. In particular, for the XML publishing of relational data and for "schema-based" shredding of XML documents into relations, there is no published algorithm for translating even simple path expression queries (with the // axis) into SQL when the XML schema is recursive. It is our hope that this paper will stimulate others to refine our classification and, more importantly, to improve the state of the art and to address and solve the open problems that the classification reveals.

Technique	Scenario	Subproblems solved	Class of XML Schema considered	Class of XML Queries handled
XPeranto	XP/GAV	VD,QT	tree	XQuery
SilkRoute	XP/GAV	VD,QT	tree	XML-QL
Rolex	XP/GAV	QT	tree	XSLT
[17]	XP/GAV	QT	tree	XSLT
[1]	XP/GAV	VD	recursive	-
Oracle XML DB	XP/GAV, XS/SB	VD,SS,QT	recursive	SQL/XML restricted XPath[1]
SQL Server 2000 SQLXML	XP/GAV, XS/SB	VD,SS,QT	bounded depth recursive	restricted XPath[2]
DB2 XML Extender	XP/GAV, XS/SB	VD,QT	non-recursive	SQL extensions through UDFs
Agora	XP/LAV	QT	non-recursive	XQuery
MARS	XP/GAV + XP/LAV	QT	non-recursive	XQuery
STORED	XS/SO	SS,QT	all	STORED
Edge	XS/SO	SS,QT	all	path expressions
Monet	XS/SO	SS	all	-
XRel	XS/SO	SS,QT	all	path expressions
[35]	XS/SO	SS,QT	all	order-based queries
Dynamic intervals [7]	XS/SO	QT	all	XQuery
[24, 32]	XS/SB	SS	recursive	-
[2, 16, 19, 21, 27]	XS/SB	SS	tree	-

XP/GAV: XML Publishing, Global-as-view XP/LAV: XML Publishing, Local-as-view
XS/SO: XML Storage, schema-oblivious XS/SB: XML Storage, schema-based
QT: Query Translation VD: View Definition SS: Storage scheme
restricted XPath[1]: child and attribute axes
restricted XPath[2]: child, attribute, self and parent axes

Table 1. Summary of various published techniques

The interaction between XML and RDBMS can be broadly classified as shown in Figure 1. The main scenarios are:

1. XML Publishing (XP): here, the goal is to treat existing relational data sets as if they were XML. In other words, an XML view of the relational data set is defined and XML queries are posed over this view.
2. XML Storage (XS): here, by contrast, the goal is to use an RDBMS to store and query existing XML data. In this scenario, there are two main subproblems: (1) a relational schema has to be chosen for storing the XML data, and (2) XML queries have to be translated to SQL for evaluation.

In this paper, we use this classification to characterize almost 40 published solutions to the XML-to-SQL query translation problem.

The results of our survey are summarized in Table 1, where for each technique we identify the scenario solved and the part of the problem handled within that

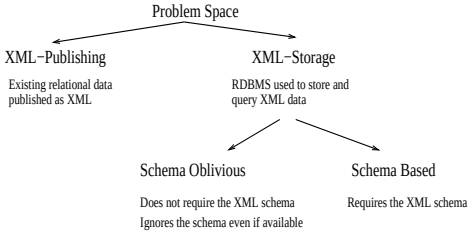

Fig. 1. High-Level Taxonomy

	Tree Schema		Recursive Schema	
Simple Queries (path expressions)	XP	lot	XP	some
	XS/SO	lot	XS/SO	lot
	XS/SB	lot	XS/SB	some
Complex Queries	XP	lot	XP	none
	XS/SO	some	XS/SO	some
	XS/SB	none	XS/SB	none

Fig. 2. Focus of published solutions

scenario. We will look at each of these in more detail in the rest of the paper. In addition to the characteristics from our broad classification, the table also reports, for each solution, the class of schema considered, the class of XML queries handled, whether it uses the "global as view" or "local as view" approach (if the XML publishing problem is addressed), and what subproblems are solved.

The rest of the paper is organized as follows. We survey known algorithms in published literature for XML-publishing, schema-oblivious XML storage and schema-based XML storage in Sections 2, 3, and 4 respectively. For each scenario, we first survey the solutions that have been proposed in published literature, and discuss problems that remain open. When we look at XML support in commercial RDBMS as part of this survey, we will restrict our discussion to those features that are relevant to XML-to-SQL query translation. A full investigation of XML support in commercial RDBMS is beyond the scope of this survey.

2 XML Publishing

The following tasks arise in the context of allowing applications to query existing relational data as if it were XML:

- Defining an XML view of relational data.
- Materializing the XML view.
- Evaluating an XML query by composing it with the view.

In XML query languages like XPath and XQuery, part of the query evaluation may involve reconstructing the subtrees rooted at certain elements, which are identified by other parts of the query. Notice how materializing an XML view is a special case of this situation, where the entire tree (XML document) is reconstructed. In general, solutions to materialize an XML view are used as a subroutine during query evaluation.

2.1 XML View Definition

In XPeranto [29, 30], SilkRoute [12, 13] and Rolex [3], the view definition languages permit definition of tree XML views over the relational data. In [1], XML views corresponding to recursive XML schema (recursive XML view schema) are allowed.

In Oracle XML DB [42] and Microsoft SQL Server 2000 SQLXML [43], an annotated XSD XML schema is used to define the XML view. Recursive XML

views are supported in XML DB. In SQLXML, along with non-recursive views, there is support for a limited number of depths of recursion using the max-depth annotation. In IBM DB2 XML Extender [40], a Document Access Definition (DAD) file is used to define a non-recursive XML view. IBM XML for Tables [44] provides an XML view of relational tables and is based on the Xperanto [30] project.

In the above approaches, the XML view is defined as a view over the relational schema. In a data integration context, Agora [25] uses the local-as-view approach (LAV), where the local source's schema are described as views over the global schema. Toward this purpose, they describe a generic, virtual relational schema closely modeling the generic structure of an XML document. The local relational schema is then defined as views over this generic, virtual schema. Contrast this with the other approaches where the XML view (global schema) is defined as a view over the relational schema (local schema). This is referred to as the global-as-view approach (GAV).

In Mars [9], the authors consider the scenario where both GAV-style and LAV-style views are present. The focus of [9, 25] is on non-recursive XML view schema.

2.2 Materializing the XML View

In XPeranto [30], the XML view is materialized by pushing down a single "outer union" query into the relational engine, whereas in SilkRoute [12], the middleware system issues several SQL queries to materialize the view. In [1], techniques for materializing a recursive XML view schema are discussed. They argue that since SQL supports only linear recursion, the support for recursion in SQL is insufficient for this purpose. Instead, the recursive materialization is performed in middleware by repeatedly unrolling a fixed number of levels at a time. We discuss this in more detail in Section 2.4.

2.3 Evaluating XML Queries

In XPeranto [29], a general framework for processing arbitrarily complex XQuery queries over XML views is presented. They describe their XQGM query representation, an extension of a SQL internal query representation called the Query Graph Model (QGM). The XQuery query is converted to an XQGM representation and composed with the view definition. Rewrite optimizations are performed to eliminate the construction of intermediate XML fragments and to push down predicates. The modified XQGM is translated into a single SQL query to be evaluated inside the relational engine.

In SilkRoute [12], a sound and complete query composition algorithm is presented for evaluating a given XML-QL query over the XML view. An XML-QL query consists of patterns, filters and constructors. Their composition technique evaluates the patterns on the view definition at compile-time to obtain a modified XML view, and the filters and constructors are evaluated at run-time using the modified XML view.

In [17], the authors present an algorithm for translating XSLT programs into efficient SQL queries. The main focus of the paper is on bridging the gap between

XSLT's functional, recursive paradigm, and SQL's declarative paradigm. They also identify a new class of optimizations that need to be done either by the translator or by the relational engine, in order to optimize the kind of SQL queries that result from such a translation. In Rolex [22], a view composition algorithm for composing an XSLT stylesheet with an XML view definition to produce a new XML view definition is presented. They differ from [17] mainly in the following ways: (1) they produce an XML view query rather than an SQL query, (2) they address additional features of XSLT like priority and recursive templates.

As part of the Rainbow system, in [39], the authors discuss processing and optimization of XQuery queries. They describe the XML Algebra Tree (XAT) algebra for modeling XQuery expressions, propose rewriting rules to optimize XQuery queries by canceling operators and describe a cutting algorithm that removes redundant operators and relational columns from the XAT. However, the final XML to SQL query generation is not discussed.

We note here that in Rolex [3], the world view is changed so that a relational system provides a virtual DOM interface to the application. The input in this case is not a single XML query but a series of navigation operations on the DOM tree that needs to be evaluated on the underlying relational data.

The Agora [25] project uses an LAV approach and provides an algorithm for translating XQuery FLWR expressions into SQL. Their algorithm has two main steps — translating the XML query into a SQL query on the generic, virtual relational schema, and rewriting this SQL query into a SQL query over the real relational schema. In the first step, they cross the language gap from XQuery to SQL, and in the second step they use prior work on answering queries using views.

In MARS [9, 10], a technique for translating XQuery queries into SQL is given, when both GAV-style and LAV-style views are present. The basic idea is to compile the queries, views and constraints from XML into the relational framework, producing relational queries and constraints. Then, a Chase and BackChase (C&B) algorithm is used to find all minimal reformulations of the relational queries under the relational integrity constraints. Using a cost-estimator, the optimal query among the minimal reformulations is obtained, which can then be executed. The MARS system also exploits integrity constraints on both the relational and XML data. The system achieves the combined effect of rewriting-with-views, composition-with-views, and query minimization under integrity constraints.

Oracle XML DB [42] provides an implementation of the majority of the operators that will be incorporated into the forthcoming SQL/XML standard [41]. SQL/XML is an extension to SQL, using functions and operators, to include processing of XML data in relational stores. The SQL/XML operators [11] make it possible to query and access XML content as part of normal SQL operations and also provide methods for generating XML from the result of an SQL Select statement. The SQL/XML operators allow XPath expressions to be used to access a subset of the nodes in the XML view. In XML DB, the approach is to

translate the XPath expression into an equivalent SQL query through a query re-write step that uses the XML view definition. In the current release (Oracle9i Release 2), simple path expressions with no wild cards or descendant axes (//) get rewritten. Predicates are supported and get rewritten into SQL predicates. The XPath axes supported are the child and attribute axis.

Microsoft SQL Server 2000 SQLXML [43] supports the evaluation of XPath queries over the annotated XML Schema. The XPath query together with the annotated schema is translated into a *FOR XML* explicit query that only returns the XML data that is required by the query. Here, *FOR XML* is a new SQL select statement extension provided by SQL Server. In the current release (SQLXML 3.0), the attribute, child, parent and self axes are supported, along with predicates and XPath variables.

In IBM DB2 XML Extender [40], powerful user-defined functions (UDFs) are provided to store and retrieve XML documents in XML columns, as well as to extract XML element or attribute values. Since it does not provide support for any XML query languages, we will not discuss XML Extender any further in this paper.

2.4 Open Problems

A number of problems remain open in this area.

1. With the exception of [1, 42], the above work considers only non-recursive XML views of relational data. While Oracle XML DB [42] supports path expression queries with the child and attribute axes over recursive views, it does not support the descendant (//) axis. Translating XML queries (with the // axis) over recursive view schema remains open. In [1], the problem of materializing recursive XML view schema is considered. However, as we have mentioned, that work does not use SQL support for recursion, simulating recursion in middleware instead. The reason for this given by the authors is that the limited form of recursion supported by SQL cannot handle the forms of recursion that arise in with recursive XML schema. We return to this question at the end of this section. The following are open questions in the context of SQL support for recursion:
   - What is the class of queries/view schema for which the current support for recursion in SQL are adequate?
   - If there are cases for which SQL support for recursion is inadequate, how do we best leverage this support? (Instead of completely simulating recursion in middleware.)
2. Any query translation algorithm can be evaluated by two metrics: its functionality, in terms of the class of XML queries handled; and its performance, in terms of the efficiency of the resulting SQL query. Most of the translation algorithms have not been evaluated thoroughly by either metric, which gives rise to a number of open research problems.
   - Functionality: Among the GAV-style approaches, except XPeranto, all the above discussed work deals with languages other than XQuery. Even in the case of XPeranto, the class of XQuery handled is unclear from [29].

It would be interesting to precisely characterize the class of XQuery queries that can be translated by the methods currently in the literature.
- Performance: There has been almost no work comparing the quality of SQL queries generated by various translation algorithms. In particular, we are aware of no published performance study for the query translation problem.

3. GAV vs. LAV: While for the GAV-style approaches, XML-to-SQL query translation corresponds to view composition, for the LAV-style approaches it corresponds to answering queries with views. It is not clear for what class of XML views the equivalent query rewriting problem has published solutions. As pointed out in [25], state-of-the-art query rewriting algorithms for SQL semantics do not efficiently handle arbitrary levels of nesting, grouping etc. Similarly, [9] works under set-semantics and so cannot handle certain classes of XML view schema and aggregation in XML queries. Comparing across the three different approaches — GAV, LAV and GAV+LAV, in terms of both functionality and performance is an open issue.

Recursive XML View Schema and Linear Recursion in SQL In this subsection we return to the problem of recursive XML view schema and whether or not they can be handled by the support for recursion currently provided by SQL.

Consider the problem of materializing a recursive XML view schema. In [1], it is mentioned that even though SQL supports linear recursion, this is not sufficient for materializing a recursive XML view. The reason for this is not elaborated in the paper. The definition of an XML view has two main components to it: the view definition language and the XML schema of the resulting view. Hence, it must be the case that either the XML schema of the view or the view definition language is more complex than what SQL linear recursion can support. Clearly, if the view definition language is complex enough (say the parent-child relationship is defined using non-linear recursion), linear recursion in SQL will not suffice. However, most view definition languages proposed define parent-child relationships through much simpler conditions (such as conjunctive queries). The question arises whether SQL linear recursion is sufficient for these view definition languages, for arbitrary XML schema.

In [6], the notion of linear and non-linear recursive DTDs is introduced. The natural question here is whether the notions of linear recursion in SQL and DTDs correspond. It turns out that the definition of non-linear recursive schema in [6] has nothing to do with the traditional Datalog notion of linear and non-linear recursion. For example, consider a classical part-subpart database. Suppose that the DTD rule for a part element is: part → pname, part*.

According to [6], this is a non-linear recursive rule as a part element can derive multiple part sub-elements. Hence, the entire DTD is non-linear recursive. Indeed, it can be shown that this DTD is not equivalent to any linear-recursive DTD. Now, suppose the underlying relational schema has two relations, Part and Subpart with the columns: (partid,pname) and (partid,subpartid) respectively. Now, the following SQL query extracts all data necessary to materialize the XML view:

```
WITH RECURSIVE AllParts(partid,pname,rtolpath) as (
           select partid,pname,''
           from Part(partid,pname)
        union all
           select P.partid,P.pname,rtolpath+A.partid
           from AllParts A, Subpart S, Part P
           where S.partid = A.partid and S.subpartid = P.partid)
select * from AllParts
```

In the above query, the root-to-leaf path is maintained for each part element through the `rtolpath` column in order to extract the tree structure. Note however that the core SQL query executes the following linear-recursive Datalog program.

AllParts(partid,pname) ← Part(partid,pname)
AllParts(subpartid,subpname) ←
AllParts(partid,pname) Subpart(partid,subpartid) Part(subpartid,subpname)

So, we see that a non-linear recursive rule in the DTD gets translated into a linear recursive Datalog (SQL) rule. This implies that the notion of linear recursion in DTDs and SQL (Datalog) do not have a direct correspondence. Hence, the class of XML view schema/view definition languages for which SQL linear recursion is adequate to materialize the resulting XML views needs to be examined.

3 Schema-Oblivious XML Storage

Recall that in this scenario, the goal is to find a relational schema that works for storing XML documents independent of the presence or absence of a schema. The main problems addressed in this sub-space are:

1. Relational schema design: which generic relational schema for XML should be used?
2. Query translation algorithms: given a decision for the relational schema, how do we translate from XML queries to SQL queries.

3.1 Relational Schema Design

In STORED [8], given a semi-structured database instance, a STORED mapping is generated automatically using data mining techniques — STORED is a declarative query language proposed for this purpose. This mapping has two parts: a relational schema and an overflow graph for the data not conforming to the relational schema. We classify STORED as a schema-oblivious technique since the data since data inserted in the future is not required to conform to the derived schema. Thus, if an XML document with completely different structure is added to the database, the system sticks to the existing relational schema without any modification whatsoever.

In [14], several mapping schemes are proposed. According to the Edge approach, the input XML document is viewed as a graph and each edge of the graph is represented as a tuple in a single table. In a variant known as the Attribute approach, the edge table is horizontally partitioned on the tag name yielding a separate table for each element/attribute. Two other alternatives, the

Universal table approach and the Normalized Universal approach are proposed but shown to be inferior to the other two. Hence, we do not discuss these any further.

The binary association approach [28] is a path-based approach that stores all elements that correspond to a given root-to-leaf path together in a single relation. Parent-child relationships are maintained through parent and child ids.

The XRel approach [37] is another path-based approach. The main difference here is that for each element, the path id corresponding to the root-to-leaf path as well as an interval representing the region covered by the element are stored. The latter is similar to interval-based schemes for representing inverted lists proposed in [23, 38].

In [35], the focus is on supporting order based queries over XML data. The schema assumed is a modified Edge relation where the path id is stored as in [37], and an extra field for order is also stored. Three schemes for supporting order are discussed.

In [7], all XML data is stored in a single table containing a tuple for each element, attribute and text node. For an element, the element name and an interval representing the region covered by the element is stored. Analogous information is stored for attributes and text nodes.

There has been extensive work on using inverted lists to evaluate path expression queries by performing containment joins [5, 18, 23, 26, 33, 36, 38]. In [38], the performance of containment algorithms in an RDBMS and a native XML system are compared. All other strategies are for native XML systems. In order to adapt these inside a relational engine, we would need to add new containment algorithms and novel data structures. The issue of how we extend the relational engine to identify the *use* of these strategies is open. In particular, the question of how the optimizer maps SQL operations into these strategies needs to be addressed.

In [15], a new database index structure called the XPath accelerator is proposed that supports all XPath axes. The preorder and postorder ranks of an element are used to map nodes onto a two-dimensional plane. The evaluation of the XPath axis steps then reduces to processing region queries in this pre/post plane. In [34], the focus is on exploiting additional properties of the pre/post plane to speedup XPath query evaluation and the Staircase join operator is proposed for this purpose. The focus of [15, 34] is on efficiently supporting the basic operations in a path expression and is complementary to the XML-to-SQL query translation issue.

In Oracle XML DB [42] and IBM DB2 XML Extender [40], a schema-oblivious way of storing XML data is provided, where the entire XML document is stored using the CLOB data type. Since evaluating XML queries in this case will be similar to XML query processing in a native XML database and will not involve XML-to-SQL query translation, we do not discuss this approach in this paper.

3.2 Query Translation

In STORED [8], an algorithm is outlined for translating an input STORED query into SQL. The algorithm uses inversion rules to create a single canonical data instance, intuitively corresponding to a schema. The structural component of the STORED query is then evaluated on this instance to obtain a set of results, for each of which a SQL query is generated incorporating the rest of the STORED query.

In [14], a brief overview of how to translate the basic operations in a path expression query to SQL is provided. The operations described are (1) returning an element with its children, (2) selections on values, (3) pattern matching, (4) optional predicates, (5) predicates on attribute names and (6) regular path queries which can be translated into recursive SQL queries.

The binary association method [28] deals with translating OQL-like queries into SQL. The class of queries they consider roughly corresponds to branching path expression queries in XQuery.

In XRel [37], a core part of XPath called XPathCore is identified and a detailed algorithm for translating such queries into SQL is provided. Since with each element, a path id corresponding to the root-to-leaf path is stored, a simple path expression query like book/section/title gets efficiently evaluated. Instead of performing a join for each step of the path expression, all elements with a matching path id are extracted. Similar optimizations are proposed for branching path expression queries exploiting both path ids and the interval encoding. We examine this in more detail in Section 3.3.

In [35], algorithms for translating order based path expression queries into SQL are provided. They provide translation procedures for each axis in XPath, as well as for positional predicates. Given a path expression, the algorithm translates one axis at a time in sequence.

The dynamic intervals approach [7] deals with a larger fragment of XQuery with arbitrarily nested FLWR expressions, element constructors and built-in functions including structural comparisons. The core idea is to begin with static intervals for each element and construct dynamic intervals for XML elements constructed in the query. Several new operators are proposed to efficiently implement the generated SQL queries inside the relational engine. These operators are highly specialized and are similar to operators present in a native XML engine.

3.3 Summary and Open Problems

The various schema-oblivious storage techniques can be broadly classified as:

1. Id-based: each element is associated with a unique id and the tree structure of the XML document is preserved by maintaining a foreign key to the parent.
2. Interval-based: each element is associated with a region representing the subtree under it.
3. Path-based: each element is associated with a path id representing the root-to-leaf path in addition to an interval-based or id-based representation.

We organize the rest of the discussion by considering different classes of queries.

Reconstructing an XML sub-tree This problem is largely solved. In the schema-oblivious scenario, the sub-tree corresponding to an XML element could potentially span all tables in the database. Hence, while solutions that store all the XML data in only one table need to process just that table, other solutions will need to access all tables in the database.

For id-based solutions, a recursive SQL query can be used to reconstruct a sub-tree. For interval-based solutions, a non-recursive query with interval predicates is sufficient.

Simple Path Expression Queries We refer to the class of path expression queries without predicates as simple path expression queries. For interval-based solutions, evaluating simple path expressions entails performing a range join for each step of the path expression. For example the query book/author/name translates into a three-way join. For id-based solutions, each parent-child(/) step translates into an equijoin, whereas recursion in the path expression (through //) requires a recursive SQL query. For path-based solutions, the path id can be used to avoid performing one join per step of the path expression.

Path Expression Queries With Predicates Predicates can be existential path expression predicates, or positional predicates. The latter is dealt with in [35, 37]. We focus on the former for the rest of the section.

For id-based and interval-based solutions, a straightforward method for query translation is to perform one join per step in the path expression [8, 14, 38]. With path ids, however, it is conceivable that certain joins can be skipped, just as they can be skipped for some simple path expressions. A detailed algorithm for doing so is proposed in [37]. That algorithm is correct for nonrecursive data sets — it turns out that it does not give the correct result when the input XML data has an ancestor and descendant element with the same tag name. For that reason, the general problem of translation of path expressions with predicates for the path-based schema-oblivious schemes is still open.

More Complex XQuery queries The only published work that we are aware of that deals with more general XQuery queries is [7]. The main focus of the paper is on issues such as structural equality in FLWR where clauses, full compositionality of XML query expressions (in particular, the possibility of nesting FLWR expressions within functions), and the need for constructed XML documents representing intermediate query results. As mentioned earlier, special purpose relational operators are proposed for better performance. We note that without these operators, the performance of their translation is likely to be inferior even for simple path expressions. As an example, using their technique, the path expression /site/people is translated to an SQL query involving five temporary relations created using the *With* clause in SQL99, three of which involve correlated subqueries. To conclude, excepting [7], all prior work has been on translating path expression queries into SQL. Using the approach proposed by [7], we observe that functionality-wise, a large fragment of XQuery can be handled using dynamic intervals in a schema-oblivious fashion. However, without modifications to the relational engine, its performance may not be acceptable.

4 Schema-Based XML Storage

In this section, we discuss approaches to storing XML in relational systems that make use of a schema for the XML data in order to choose a good relational schema. The main problems to be addressed in this subspace are

1. Relational schema selection — given an XML schema (or DTD), how should we choose a good relational schema and XML-to-relational mapping.
2. Query translation — having chosen an XML-to-relational mapping, how should we translate XML queries into SQL.

4.1 Relational Schema Selection

In [32], three techniques for using a DTD to choose a relational schema are proposed — basic inlining, shared inlining, and hybrid inlining. The main idea is to inline all elements that occur at most once per parent element in the parent relation itself. This is extended to handle recursive DTDs.

In [21], a constraint preserving algorithm for transforming an XML DTD to a relational schema is presented. The authors chose the hybrid inlining algorithm from [32] and showed how semantic constraints can be generated.

In [2], the problem of choosing a good relational schema is viewed as an optimization problem: given an XML schema, an XML query workload, and statistics over the XML data choose the relational schema that maximizes query performance. They give a greedy heuristic for this purpose.

In [16, 24], the theory of regular tree grammars is used to choose a relational schema for a given XML schema.

In [4], a storage mapping that takes into account the key and foreign key constraints present in an XML schema is presented.

There has been some work on using object-relational DBMS to store XML documents. In [19, 27], parts of the XML document are stored using an XML ADT. The focus of these papers is to determine *which* parts of the DTD must be mapped to relations and which parts must be mapped to the XML ADT.

In Oracle XML DB [42], an annotated XML Schema is used to define how the XML data is mapped into relations. If the XML Schema is not annotated, XML DB uses a default algorithm to decide the relational schema based on the XML Schema. This algorithm handles recursive XML schemas.

A similar approach is made in Microsoft SQL Server 2000 SQLXML [43] and IBM DB2 XML Extender [40], but they only handle non-recursive XML schemas.

4.2 Query Translation

In [32], the general approach to translating XML-QL queries into SQL is illustrated through examples without any algorithmic details.

As discussed in [31], it is possible to use techniques from the XML publishing domain in the XML storage domain. To see this, notice that once XML data is shredded into relations, we can view the resulting data as if it were pre-existing relational data. Now by defining a reconstruction view that mirrors

the XML-to-relational mapping used to shred the data, the query translation algorithms in the XML publishing domain are directly applicable. Indeed, this is the approach adopted in [2]. While this approach has the advantage that solutions for XML publishing can be directly applied to the schema-based XML storage scenario, it has one important drawback. In the XML storage scenario, the data in the RDBMS originates from an XML document and there is some semantic information associated with this (like the tree structure of the data and the presence of a unique parent for each element). This semantic information can be used by the XML-to-SQL translation algorithm to generate efficient SQL queries. By using solutions from the XML publishing scenario, we are potentially making the use of this semantic information harder. We discuss this in more detail with an example in Section 4.3.

Note that even the schema-oblivious subspace can be dealt with in an analogous manner as mentioned in [31]. However, in this case, the reconstruction view is fairly complex — for example, the reconstruction view for the Edge approach is an XQuery query involving recursive functions [31]. Since handling recursive XML view schema is open (Section 2.4), this approach for the schema-oblivious scenario needs to be explored further.

In [35], as we mentioned in Section 3.2, the focus is on supporting order-based queries. The authors give an algorithm for the schema-oblivious scenario, and briefly mention how the ideas can be applied with any existing schema-based approach.

In Oracle XML DB [42], Microsoft SQL Server 2000 SQLXML [43] and IBM DB2 XML Extender [40], the XML Publishing and Schema-Based XML Storage scenarios are handled in an identical manner. So, the description of their approaches for the XML Publishing scenario presented in Section 2.3 holds for the Schema-Based XML Storage scenario. To summarize, XML DB supports branching path expression queries with the child and attribute axes, while SQLXML supports the parent and self axes as well. XML Extender does not support any XML query language. Instead, it provides user-defined functions to manipulate XML data.

In [20], the problem of finding optimal relational decompositions for XML workloads is considered in a formal perspective. Using three XML-to-SQL query translation algorithms for path expression queries over a particular family of XML schemas, the interaction between the choice of a good relational decomposition and a good query translation algorithm is studied. The authors showed that the query translation algorithm and the cost model used play a vital role not just in the choice of a good decomposition, but also in the complexity of finding the optimal choice.

4.3 Discussion and Open Problems

There is no published query translation algorithm for the schema-based XML storage scenario. One alternative is to reduce this problem to XML publishing (using reconstruction views). Hence, from a functionality perspective, whatever is open in the XML publishing case is open here also. In particular, the entire

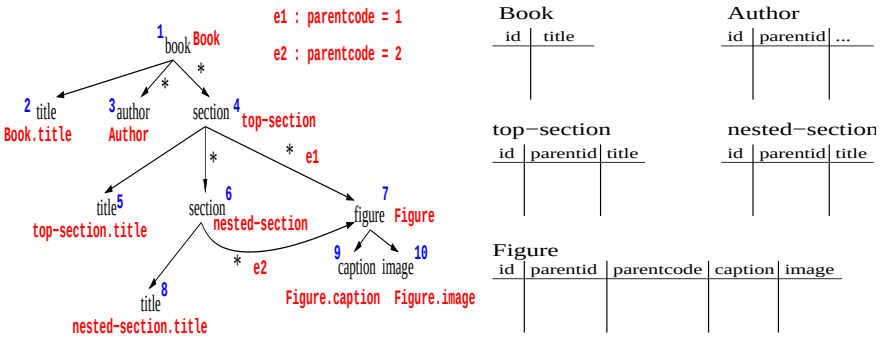

Fig. 3. Sample XML-to-Relational mapping schema

problem is open when the input XML schema is recursive. Even for a non-recursive XML schema, a lot of interesting questions arise when the XML schema is not a tree. For example, if there is recursion in an XPath query through //, the straightforward approach of enumerating all satisfying paths using the schema and handling them one at a time is no longer an efficient approach. If we wish to reduce the problem to XML publishing, the only way to use an existing solution is to unfold the DAG schema into an equivalent tree schema.

We now examine the translation problem from a performance perspective.

Goals of XML-to-SQL Query Translation When an XML document is shredded into relations, there is inherent semantic information associated with the relation instances given that the source is XML. For example, consider the XML schema shown in Figure 3. One candidate relational decomposition is also shown in the figure. The mapping is illustrated through annotations on the XML schema. Each node is annotated with the corresponding relation name. Leaf nodes are annotated with the corresponding relational column as well. Parent-child relationships are represented using id and parentid columns. The figure element has two potential parents in the schema. In order to distinguish between them, a parentcode field is present in the Figure relation. In this case, notice that there is inherent semantics associated with the columns parentid and parentcode given that they represent the manner in which the tree structure of the XML document is preserved.

Given this semantics, when an XML query is posed, there are several equivalent SQL queries, which are not necessarily equivalent without the extra semantics that come from knowing that the relations came from shredding XML. Consider the following query: find captions for all figures in top level sections. This can be posed as an XPath query XQ = /book/section/figure/caption. There are two equivalent ways in which we could translate XQ into SQL. They are shown below.

```
SQ1:
select caption
from figure
where parentcode=1
```

```
SQ2:
select caption
from   figure f, top-section ts, book b
where  f.parentcode=1 and f.parentid=ts.id
       and ts.parentid=b.id
```

While SQ1 merely performs a scan on the figure table, SQ2 roughly performs a join for each step of the path expression. SQ2 is what we would obtain by adapting techniques from XML publishing. Queries SQ1 and SQ2 are equivalent only because of the semantics associated with the parentcode and parentid columns and would not be equivalent otherwise.

Now, since the XML-to-SQL translation algorithm is aware of the semantics of the XML-relational mapping, it is better placed than the relational optimizer to find the best SQL translation. Hence, from a performance perspective, the problem of effectively exploiting the XML schema and the XML-relational mapping during query translation remains open.

Enhancing Schema-Based solutions with Intervals/Path-ids All the schema-based solutions proposed in published literature have been id-based. In the schema-oblivious scenario, it has been shown that using intervals and path-ids can be helpful in XML-to-SQL query translation. The problem of augmenting the schema-based solutions with some sort of intervals and/or path-ids is an interesting open problem. Note that while any id-based storage scheme can be easily augmented by adding either a path-id column or an interval for each element, developing query translation algorithms that use *both* the schema information and the interval/path information is non-trivial.

5 Summary and Conclusions

To conclude, we refer again to the summary in Table 1. From that table, we see that the community has made varying degrees of progress for different subproblems in the XML to SQL query translation domain. We next summarize this progress, in terms of functionality.

- In the XML-Publishing scenario, techniques have been proposed for handling complex query languages like XQuery and XSLT over tree XML view schema. However, handling recursive XML view schema is an open problem. Even for tree XML view schema, the subset of XQuery handled by current solutions is not clear.
- In the schema-oblivious XML storage scenario, excepting [7], the focus has been on path expression queries.
- In the schema-based XML storage scenario, there is no published query translation algorithm. The only approach known to us is through a reduction to the XML publishing scenario.

Our purpose in submitting this paper to this symposium is to begin to clarify what is known and what is not known in the large, growing, and increasingly diverse literature dealing with using relational database systems for storing and/or querying XML. Other classifications of existing work are possible, and certainly our classification can be refined and improved. Furthermore, while we have tried to be complete and have surveyed close to 40 publications, there may be gaps in our coverage. It is our hope that publishing this paper in the symposium proceedings will begin a discussion that will improve the quality of this summary,

and, more importantly, will spur further work to fill the gaps in the state of the art in the XML to SQL translation literature.

Acknowledgement: This work was supported in part by NSF grant ITR-0086002 and a Microsoft fellowship.

References

1. M. Benedikt, C. Y. Chan, W. Fan, R. Rastogi, S. Zheng, and A. Zhou. DTD-Directed Publishing with Attribute Translation Grammars. In *VLDB*, 2002.
2. P. Bohannon, J. Freire, P. Roy, and J. Simeon. From XML schema to relations: A cost-based approach to XML storage. In *ICDE*, 2002.
3. P. Bohannon, H. Korth, P.P.S. Narayan, S. Ganguly, and P. Shenoy. Optimizing view queries in ROLEX to support navigable tree results. In *VLDB*, 2002.
4. Y. Chen, S. B. Davidson, and Y. Zheng. Constraint preserving XML Storage in Relations. In *WebDB*, 2002.
5. S. Chien, Z. Vagena, D. Zhang, V. J. Tsotras, and C. Zaniolo. Efficient Structural Joins on Indexed XML Documents. In *VLDB*, 2002.
6. B. Choi. What Are Real DTDs Like. In *WebDB*, 2002.
7. D. DeHaan, D. Toman, M. P. Consens, and T. Ozsu. A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding. In *SIGMOD*, 2003.
8. A. Deutsch, M. Fernández, and D. Suciu. Storing semistructured data with STORED. In *SIGMOD*, 1999.
9. A. Deutsch and V. Tannen. MARS: A System for Publishing XML from Mixed and Redundant Storage. In *VLDB*, 2003.
10. A. Deutsch and V. Tannen. Reformulation of XML Queries and Constraints. In *ICDT*, 2003.
11. A. Eisenberg and J. Melton. SQL/XML is Making Good Progress. *SIGMOD Record*, 31(2), 2002.
12. M. Fernandez, A. Morishima, and D. Suciu. Efficient Evaluation of XML Middle-ware Queries. In *SIGMOD*, 2002.
13. M. Fernández, D. Suciu, and W.C. Tan. SilkRoute: Trading Between Relations and XML. In *WWW9*, 2000.
14. D. Florescu and D. Kossman. Storing and Querying XML Data using an RDBMS. *Data Engineering Bulletin*, 22(3), 1999.
15. T. Grust. Accelerating XPath location steps. In *SIGMOD*, 2002.
16. S. Hongwei, Z. Shusheng, Z. Jingtao, and W. Jing. Constraints-Preserving Mapping Algorithm from XML-Schema to Relational Schema. In *Engineering and Deployment of Cooperative Information Systems (EDCIS)*, 2002.
17. S. Jain, R. Mahajan, and D. Suciu. Translating XSLT Programs to Efficient SQL Queries. In *WWW*, 2002.
18. H. Jiang, H. Lu, W. Wang, and B. C. Ooi. XR-Tree: Indexing XML Data for Efficient Structural Joins. In *ICDE*, 2003.
19. M. Klettke and H. Meyer. XML and Object-Relational Database Systems - Enhancing Structural Mappings Based on Statistics. In *WebDB*, 2000.
20. R. Krishnamurthy, R. Kaushik, and J. F. Naughton. On the difficulty of finding optimal relational decompositions for xml workloads: A complexity theoretic perspective. In *ICDT*, 2003.

21. D. Lee and W.W. Chu. Constraints-preserving Transformation from XML Document Type Definition to Relational Schema. In *ER*, 2000.
22. C. Li, P. Bohannon, H. Korth, and P.P.S. Narayan. Composing XSL Transformations with XML Publishing Views. In *SIGMOD*, 2003.
23. Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In *Proceedings of VLDB*, 2001.
24. M. Mani and D. Lee. XML to Relational Conversion Using Theory of Regular Tree Grammars. In *Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web (EEXTT)*, 2002.
25. I. Manolescu, D. Florescu, and D. Kossman. Answering XML queries over heterogeneous data sources. In *VLDB*, 2001.
26. N.Bruno, N.Koudas, and D. Srivastava. Holistic Twig Joins: Optimal XML Pattern Matching. In *SIGMOD*, 2002.
27. K. Runapongsa and J. M. Patel. Storing and Querying XML Data in Object-Relational DBMSs. In *EDBT Workshops*, 2002.
28. A. Schmidt, M. Kersten, M. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents. In *WebDB*, 2000.
29. J. Shanmugasundaram, J. Kiernan, E. J. Shekita, C. Fan, and J. Funderburk. Querying XML Views of Relational Data. In *VLDB*, 2001.
30. J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, and B. Reinwald. Efficiently Publishing Relational Data as XML Documents. In *VLDB*, 2000.
31. J. Shanmugasundaram, E. Shekita, J. Kiernan, R. Krishnamurthy, S. D. Viglas, J. Naughton, and I. Tatarinov. A general technique for querying xml documents using a relational database system. *SIGMOD Record*, 30(3), 2001.
32. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *VLDB*, 1999.
33. D. Srivastava, S. Al-Khalifa, H.V. Jagadish, N. Koudas, J.M. Patel, and Y. Wu. Structural Joins: A Primitive For Efficient XML Query Pattern Matching. In *ICDE*, 2002.
34. J. Teubner T. Grust, M. V. Keulen. Staircase Join: Teach a Relational DBMS to Watch its (Axis) Steps. In *VLDB*, 2003.
35. I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang. Storing and querying ordered XML using a relational database system. In *SIGMOD*, 2002.
36. W. Wang, H. Jiang, H. Lu, and J. X. Yu. PBiTree Coding and Efficient Processing of Containment Joins. In *ICDE*, 2003.
37. M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura. XRel: a path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology (TOIT)*, 1(1):110–141, 2001.
38. C. Zhang, J.F. Naughton, D.J. DeWitt, Q. Luo, and G.Lohman. On Supporting Containment Queries in Relational Database Management Systems. In *Proceedings of SIGMOD*, 2001.
39. X. Zhang, B. Pielech, and E. Rundesnteiner. Honey, I Shrunk the XQuery! – An XML Algebra Optimization Approach. In *Workshop on Web Information and Data Management (WIDM)*, 2002.
40. DB2 XML Extender. `http://www-3.ibm.com/software/data/db2/extenders/xmlext/index.html`.
41. INCITS H2.3 Task Group. `http://www.sqlx.org`.
42. Oracle9i XML Database Developer's Guide - Oracle XML DB Release 2 (9.2). `http://otn.oracle.com/tech/xml/xmldb/content.html`.
43. SQLXML and XML Mapping Technologies. `http://msdn.microsoft.com/sqlxml/default.asp`.
44. XML for Tables. `http://http://www.alphaworks.ibm.com/tech/xtable`.