A Cost-Based Approach for Converting Relational Schemas to XML University of Iowa Technical Report 05-02

Ramon Lawrence

IDEA Lab, University of Iowa
Iowa City, IA, USA, 52242

Abstract

Converting relational database schemas to XML schemas is important as applications originally written to manipulate relational data are converted to XML-enabled applications. Most research has focused on the challenging problem of efficiently extracting relational data through XML views. However, another important challenge is determining good XML schemas from existing relational schemas, so that applications may be migrated to use XML. In this paper, we propose a cost measure based on the space efficiency of the XML encoding as a metric for measuring the XML schema's desirability. Using this cost metric, we define an algorithm that returns the optimal XML encoding exhibiting no redundancy. Further, the algorithm can calculate the most space efficient encoding with respect to user constraints on the XML schema, which has previously not been possible without forcing the user to specify a complete mapping. The result is a fully automatic system for migrating relational schemas to XML schemas while respecting conversion constraints.

Key words: XML, schema conversion, schema mapping, space efficiency, application migration

1 Introduction

As XML becomes the standard for data exchange, existing applications are being migrated from using relational databases to XML. One of the challenges of this migration process is re-designing a new XML schema from the existing

Email address: `ramon-lawrence@uiowa.edu` (Ramon Lawrence).
URL: `http://www.cs.uiowa.edu/~rlawrenc` (Ramon Lawrence).

relational schema. Since considerable effort was invested in designing a good relational schema, the goal is to preserve this investment while at the same time exploit the modeling advantages inherent in XML. The major modeling advantage of XML over the relational model is the ability to define nested schemas. This allows data to be displayed more naturally and organized more efficiently. A result is that the number of foreign keys and joins necessary to relate data in XML is reduced.

Although converting a relational schema into an XML schema is trivial if a flat translation is used (no nesting), flat translations do not take advantage of XML's hierarchical nature. By exploiting nesting in an XML document, it is possible to define XML encodings of the relational data that are more natural, easier to read, faster to query, and are more space efficient. The focus of this work is to develop a cost measure for evaluating when an XML schema represents the relational data efficiently in terms of space. [1]

The contributions of this work are:

- A fully automated relational to XML schema conversion algorithm that optimizes the space efficiency of the resulting XML schema without requiring the user annotate the relational schema or map to an intermediate model.
- Unlike other approaches, the algorithm incorporates user preferences during the mapping if they are present. Thus, the mapping can be completed with no user involvement, or as much involvement as the user desires. The algorithm then determines the optimal mapping based on the user's constraints on their desired form of the XML schema.
- Empirical results demonstrating how XML nesting improves on flat translation by creating more space efficient schemas without introducing redundancy.

We begin the discussion with a short motivation (Section 2) of the importance of relational to XML translation. Section 3 covers background work and explains the current translation algorithms. In Section 4, we discuss the advantages that nesting in XML has over flat relations in the relational model, and how we can exploit this hierarchical nature to improve readability and space efficiency. Nesting allows more efficient XML encoding of the same data than a flat translation, and we develop a cost metric to quantify this advantage. An algorithm that uses the cost metric to develop the optimal space efficient mapping from relational to XML is given in Section 5. An advantage of this algorithm is that user input in the form of constraints on the desired form of the XML schema can be quantified as costs, and the algorithm will compute the optimal solution given the constraints. Section 6 provides ex-

[1] Since the only schema property we exploit in this paper is hierarchical structure, DTDs are sufficient for the discussion. However, all results also apply to XML-Schema [26].

perimental results of testing the algorithm on various databases. Discussion and contributions are in Section 7, and the paper closes with future work and conclusions.

2 Motivation

Converting relational databases to XML is increasingly important as data stored in relational databases is accessed and exchanged using XML. Research has been performed on the query aspects of how to build XML documents efficiently when queries are generated through an XML view of a relational database [5,10,11,24]. Although efficient query generation and execution is a major challenge, another important consideration is the ability to automatically generate entire XML schemas from existing relational schemas.

To rapidly migrate existing relational database applications to XML applications, the relational schema must be mapped to an XML schema. Ideally, this mapping should be automatic and preserve the considerable effort required to develop the relational schema. It is clearly undesirable for the XML schema to be totally redesigned if the application domain is mostly unchanged. Rather, an automatic algorithm that converts the relational schema into a good XML schema is desired. This conversion algorithm should require minimal user input, preserve the normalization present in the relational schema, and take advantage of the nesting in XML schemas to improve readability and storage efficiency. Automatic tools that generate good XML schemas from relational schemas with minimal user input would save considerable time during such software conversions.

Selecting space efficient XML representations is important as encoding data in XML has high overhead which affects storage space and query performance. Even when exploiting XML compression [25], it is desirable to develop an XML schema that encodes the information space efficiently as this reduces the number of data elements to be encoded and the size of the original file before compression. It is useful to have a tool that indicates the desirability of a particular XML encoding with respect to others. Although there are multiple possible metrics [16] for desirability, including redundancy avoidance (normalization), readability, and query efficiency, the metric used in this paper is space efficiency. A cost-based approach has been used for the reverse mapping of XML documents to relational models [4].

Many commercial vendors have translation modules [2,14,23] that convert from relational to XML. Most of these systems rely on either simple flat translation or extraction queries. Simple flat translation is easy to implement, but not necessarily the best encoding, as it does not exploit the hierarchical nature of XML to improve any of the desirable properties listed previously. Specifying queries requires extensions to the base SQL language or new query languages [10] and is a lot of work. The extraction queries convert relational data to XML, but they do not migrate entire relational schemas to XML. The goal is to produce a schema translation algorithm capable of performing a translation that is more space efficient than flat translation (by exploiting nesting) and requires the user to only input the constraints on the XML schema that they require.

3 Background

Since XML has features in common with both the hierarchical/network models [8] and the nested relation model [22], initially one would expect that the relational to XML conversion is a well-studied problem with solutions that are immediately applicable from prior work. In fact, there are several aspects of the problem that make it unique. First, from a historical perspective, translating from the relational model to the hierarchical model is a relatively under-published topic. This may be partly due to the fact that the conversion was typically from network/hierarchical databases to relational and not vice versa. Conversion to relation hierarchies from the Universal Relation was studied in [18].

It is also important to note at this point that the focus of this work is not determining a normal form for XML or studying XML normalization. XML normalization is very similar to normalization of nested relations [19,21,22], and specific XML normalization results are available [1,9]. The assumption in this work is that the relational schema has already been normalized to the degree determined appropriate by its designer (typically third or fourth normal form). The conversion algorithm utilizes the normalized relations and must ensure redundancies do not get re-introduced during conversion, but its goal is not to improve on the current normalization of the relational schema. The translation algorithm only has available the schema information that can be extracted such as primary and foreign keys and does not have available functional and multi-valued dependencies that was the foundation of most early translation methods.

One of the challenges unique to relational to XML translation is that schema overhead is as big a factor as data redundancy. Thus, the ability to avoid encoding data items has a double benefit: the data item itself is not encoded and its accompanying tags are not encoded. Further, the translation should be easily expressible and preferably require minimum user input. The common weakness with most approaches is that the relational model is mapped to a different data model before conversion to XML and this procedure requires human involvement. Then, once the user has constructed the intermediate model, they have limited impact on the final result as the mapping is performed by translation rules. This work is unique in that it does not use intermediate models for conversion and allows the user to specify constraints on the mapping that are respected by the algorithm.

We will categorize previous translation methods into four categories:

- **Flat Translation** - converts relations to XML without using nesting.
- **Query-based Translation** - conversion occurs by using a query extraction language that is an extension of SQL or a new XML query language.
- **Model-based Translation** - converts the relational schema to an intermediate model which then is mapped to XML using conversion rules.
- **Dependency-based Translation** - converts the relational schema using dependency information.

The simplest method for translating relational data to XML is flat translation [17] where each relation is translated to an element E, and each attribute of the relation is either translated to a subelement (element approach) or attribute (attribute approach) of E.

Methods for defining an XML document by specifying a query or extraction rules allow user control of the conversion process, but require complete specification of the mapping. That is, the entire target XML schema is defined during the mapping. SilkRoute [10] and XML publishing using a relational database engine [24] are examples of this approach. Thus the focus is on converting relational query results to XML, not the conversion and migration of entire relational schemas to XML. Graphical extraction tools [20] have also been developed. In many cases, it is desirable for the user to only specify the constraints on the form of the target XML schema, and have the system determine the best target schema that respects those constraints.

Model-based translation methods convert the relational schema into an intermediate model which is then mapped to XML. Most of these methods are more concerned with designing good XML modeling languages than performing the translation, although translation can be performed as long as the relational model can be transformed into the intermediate model. One approach [3] converted schemas expressed using Object Role Modelling [13] into XML-Schemas. Another approach defined XNF [9] as a normal form for XML documents and provided an algorithm for translating from a conceptual schema to an XML DTD. Conversion was also done using ORA-SS [6]. This approach requires annotation to convert from the relational model to ORA-SS, and the conversion rules did not handle user constraints. Challenges with these methods include that conceptual models are not always the starting point for conversion to XML, and the user performing the translation does not want to learn another model. More typically, relational models are to be converted to XML. [2] Second, none of the approaches specifically developed a cost metric to evaluate good DTDs beyond avoiding redundancy. Further, the algorithms proposed do not allow the user to enter constraints on the final form of the DTD that are respected by the algorithm.

Dependency-based translation uses functional, multi-valued, and inclusion dependencies for translation. Most early translation systems such as [18] and the work on nested relations [21] assume the algorithms have access to the functional and multi-valued dependencies. Given the dependencies as an input set, the algorithms produce a normalized nested relation. There are two factors that make these results not directly applicable to this problem. First, the assumption is that the conversion algorithm only has available dependencies in the form that can be extracted from the relational schema. Using standard technologies such as JDBC, this limits the class of dependencies to foreign key dependencies. Second, none of the algorithms handle user input by allowing the user to direct the mapping by imposing constraints on the desired form of the output result.

Recently, dependency-based translations using nesting have been proposed [17]. Two algorithms, NeT [17] and CoT [17], are defined. NeT uses the nest operator [15] to determine optimal nestings of attributes from a single table, while CoT is used to determine nestings of multiple tables. Although not explicitly denoted as such in [17], CoT is a schema-level nesting algorithm (that uses inclusion dependencies), and NeT is a data-level nesting algorithm that scans the data to determine when nesting is appropriate. In most cases, the payoff for schema-level nesting is much greater than data-level nesting as most relational databases are highly normalized, and data-level nesting is very costly.

For illustration, this paper will use two example databases for conversion to XML. The first example is the TPC-H v1.3.0 benchmark [3], and the second example is a modified version of a university database similar to that presented in [17]. The schemas and inclusion dependencies for TPC-H and the university database are in Figures 1 and 2 respectively.

The input to the CoT algorithm is a relational database schema S which consists of a set of relational schemas $\{R_1, R_2, ..., R_n\}$ and a set of integrity constraints IC. In terms of integrity constraints, the only constraints of interest are foreign key constraints of the form $R_i[A] \subseteq R_j[B]$ where $i, j <= n$, A and B are subsets of the attributes of R_i and R_j respectively, and the set of attributes A is non-nullable. The output of the algorithm is an XML DTD for

[2] Although it is possible to convert from relational to conceptual model, then to XML.

[3] http://www.tpc.org/tpch/spec/h130.pdf

part(p_partkey, p_name, p_mfgr, p_brand, p_type, p_size, p_container, p_retailprice, p_comment)
supplier(s_suppkey, s_name, s_address, s_nationkey, s_phone, s_acctbal, s_comment)
partsupp(ps_partkey, ps_suppkey, ps_availqty, ps_supplycost, ps_comment)
customer(c_custkey,c_name,c_address,c_nationkey,c_phone,c_acctbal,c_mktsegment,c_comment)
orders(o_orderkey, o_custkey, o_orderstatus, o_totalprice, o_orderdate, o_orderpriority, o_clerk, o_shippriority, o_comment)
lineitem(l_orderkey, l_partkey, l_suppkey, l_linenumber, l_quantity, l_extendedprice, l_discount, l_tax, l_returnflag, l_linestatus, l_shipdate, l_commitdate, l_receiptdate, l_shipinstruct, l_shipmode, l_comment)
nation(n_nationkey, n_name, n_regionkey, n_comment)
region(r_regionkey, r_name, r_comment)

lineitem[l_partkey] ⊆ partsupp[ps_partkey] ⊆ part[p_partkey]
lineitem[l_suppkey] ⊆ partsupp[ps_suppkey] ⊆ supplier[s_suppkey]
lineitem[l_partkey,l_suppkey] ⊆ partsupp[ps_partkey,ps_suppkey]
orders[o_custkey] ⊆ customer[c_custkey]
customer[c_nationkey] ⊆ nation[n_nationkey], supplier[s_nationkey] ⊆ nation[n_nationkey]
lineitem[l_orderkey] ⊆ orders[o_orderkey]
nation[n_regionkey] ⊆ region[r_regionkey]

Fig. 1. TPC-H Database Schema with Inclusion Dependencies

course(cid, ctitle)	
department(did, dname, dmgr)	department[dmgr] ⊆ professor[eid]
student(sid, sname, advisor)	student[advisor] ⊆ professor[eid]
project(pid, pname, pmgr)	project[pmgr] ⊆ professor[eid]
professor(eid, ename, ecid, pid, did)	professor[ecid] ⊆ course[cid], professor[pid] ⊆ project[pid], professor[did] ⊆ department[did]
enroll(cid, sid, grade)	enroll[cid] ⊆ course[cid], enroll[sid] ⊆ student[sid]

Fig. 2. University Database Schema with Inclusion Dependencies

representing the relational data.

Given S as input, the CoT algorithm produces an IND-Graph $G = (V, E)$ using the non-nullable foreign key constraints specified in S. For every relation $R_i \in S$ there is a node $V_i \in V$. For every non-nullable foreign key constraint of the form $R_i[A] \subseteq R_j[B]$, there is a directed edge $e = (V_j, V_i) \in E$ which goes from V_j to V_i. IND-Graphs for the two databases are in Figures 3 and 4.[4]

Given an IND-Graph, an XML DTD can be constructed by determining the top-level nodes, and then performing a breadth-first search (BFS) to build nesting trees. Top-level nodes [17] are defined as nodes that cannot be subelements of other nodes. Top-level nodes are represented in the figures as greyed boxes.

[4] In Figure 4, two edges have dashed lines because they represent valid inclusion dependencies that should be in IND-Graph but are not shown in [17].

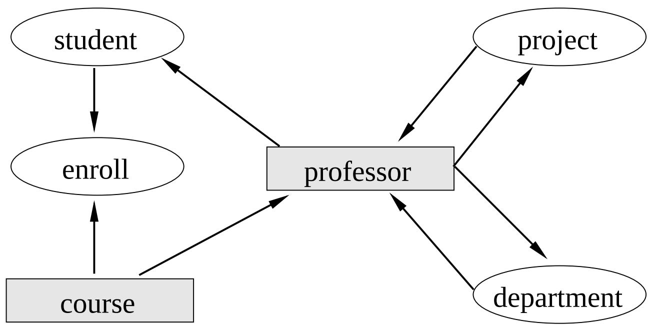

Fig. 3. IND-Graph for University Database

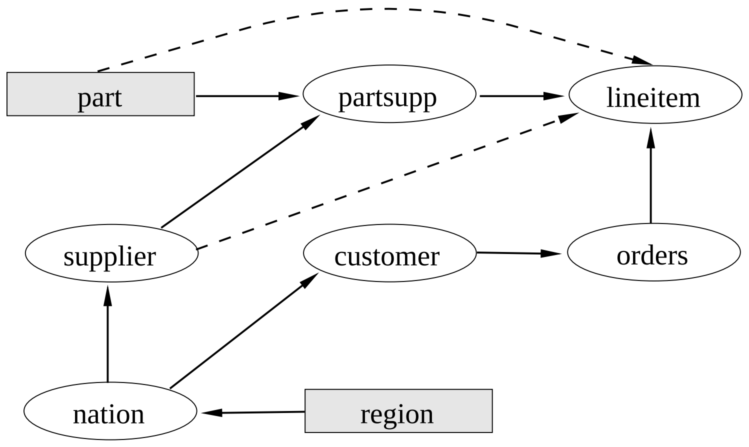

Fig. 4. IND-Graph for TPC-H Database

There are two problems with this approach. First, when cycles are present, the selection of the top-level nodes are arbitrarily made by the system. In the university example, `course` is a top-level node because it has no incoming edges, but `professor` was arbitrarily chosen as a top-level node to break the cycles. `Department` or `Project` could also have been selected. Second, there is no deterministic way for the system to perform the nesting given the top-level nodes. The nesting result is different depending on which top-level node the BFS is started from first. The user has no mechanism for indicating to the nesting algorithm which result is more desirable beyond selecting top-level nodes, and the order of their evaluation, which does not always provide the best result.

This paper contributes by defining a fully automatic relational to XML schema translation algorithm that uses only foreign key dependencies extracted using standard technologies to produce an optimal, space-efficient XML schema. Unlike all previous approaches, the algorithm has the ability to incorporate user constraints on the form of the XML schema desired and produce the optimal encoding given those constraints.

4 Nesting and Efficiency

Exploiting nesting in XML documents has three major advantages:

- Improves readability by grouping related concepts together.
- Improves space efficiency by avoiding the repetition and encoding of foreign keys that can be implicitly defined by the hierarchical structure.
- Improves query efficiency by clustering concepts together and avoiding joins.

If nesting is used indiscriminately, it is possible to re-introduce redundancy that was eliminated by normalization during the transformation from the conceptual to the relational model. The focus of this work is only on nesting that does not introduce redundancy. Note however that controlled introduction of redundancy may have numerous benefits for the efficiency of XML querying because updates are rare, and redundancy via nesting is similar to pre-materialization of joins. Such nesting introduces a tradeoff between query performance versus space efficiency and update cost.

4.1 Space Efficiency Metric

The metric chosen for this work is space efficiency. The space savings of nesting results from the fact that foreign keys originally necessary in the document to link concepts, can now be omitted as their relationship is encapsulated by the hierarchical nesting of elements. For example, in TPC-H, order information can be nested under customer information, and the foreign key *orders.o_custkey* no longer needs to be represented explicitly:

```
<!ELEMENT customer(c_custkey, c_name, ..., orders*)>
<!ELEMENT orders(o_orderkey, o_orderdate, ...)>
```

In this case, although the reverse nesting of customer under order is technically possible, it introduces redundancy, and would not be considered.

One challenge with nesting is that the foreign key must be non-nullable. In the previous example, if *orders.o_custkey* could be `null`, then orders without a customer would not be mapped into the final document. This could be resolved by introducing a dummy customer record with value of *custkey* =`null`, but that is often not desirable. Initially, we will consider only non-nullable foreign keys, and then in Section 5.1 we will discuss how to handle `null` foreign keys.

It is possible to quantify space savings via nesting on foreign keys. The space saved via nesting is directly proportional to the size of the foreign key and the number of tuples nested. Given two relations R_i and R_j, we will define a function that returns the amount of space saved by nesting R_i under R_j.

Let $FK_{R_i}(R_j)$ denote the attribute(s) in R_i that constitute the foreign key to the primary key of R_j (PK_{R_j}). Define the function $nest(R_i, R_j, K)$ as the nesting of relational schema R_i under R_j on the foreign key K. K will often be omitted ($nest(R_i, R_j)$) if there is only one possible key for nesting R_i under R_j. Let $|R_i|$ denote the number of tuples of R_i.

Definition 1 *The space savings of nesting relational schema R_i under R_j on key K is denoted by $savings(R_i, R_j, K)$ or just $savings(R_i, R_j)$ if K is understood, and can be approximated by: $savings(R_i, R_j, K) = sizeOf(K) * |R_i|$ where $sizeOf(K)$ is the maximum schema size of the foreign key K of R_i.*

To be even more precise, the exact savings of nesting in XML depends on the size of the data value of the foreign key, the tag size for the foreign key, and the XML overhead of tag specification. For attributes, the XML overhead is 4 characters (2 for quotation marks, 1 for "=", and 1 for the space separating the attribute tag name). For elements, the XML overhead is 5 characters (2 for "<", 2 for ">", and 1 for "/"). The tag size we will assume to be the length of the attribute name for the foreign key in the database. For element encoding, the tag name is used twice. Estimating the character size of a data item is difficult. In the worst case, it is the maximum schema size of the element [5], but in practice is mostly likely around half that size. This leads to the following estimates for attribute and element space savings of nesting as given in Figure 5. K is the foreign key in R_i, and $len(K)$ is the length of the attribute name for K in the schema.

Encoding Type	**Space Savings Formula**
Attribute Encoding	$(1/2 * sizeOf(K) + len(K) + 4) * \|R_i\|$
Element Encoding	$(1/2 * sizeOf(K) + 2 * len(K) + 5) * \|R_i\|$

Fig. 5. Calculating Space Savings using Attribute and Element Encodings

In practice the difference between attribute and element encoding is less important as the major savings is $O(|R_i|)$, and the rest of the formula is simply a constant factor. Savings in terms of data values not in the XML document (without worrying about their model representation) is always $|R_i|$. [6]

[5] For numeric values, the maximum schema size is assumed to be given with respect to a character encoding (e.g. decimal(9,2)).

[6] More precisely, it is $|R_i|$*(# of foreign key attributes).

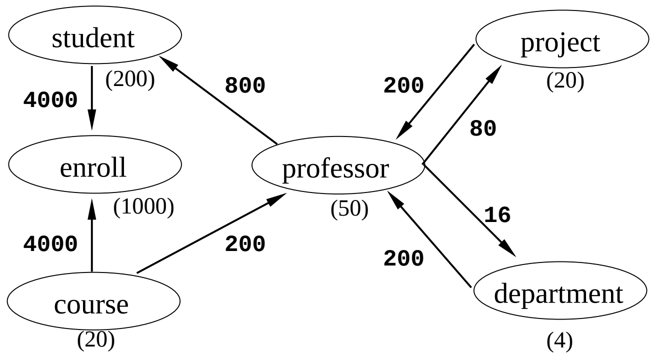

Fig. 6. Nest-Graph for University Database

4.2 Nesting Graph

Using the *savings* function and extracting database schema information on primary and foreign keys, it is possible to build a graph representing the nesting possibilities and their desirability. All information necessary for calculating the *savings* function can be extracted from the schema and queries counting the number of tuples in each relation. Thus, the algorithm does not need to know the functional and multivalued dependencies present in the schema, and the extraction process is completely automatic.

Definition 2 *A* Nest-Graph $G = (V, E)$ *is a directed, weighted graph consisting of a set of nodes* V *and a directed edge set* E*, such that for each relation* R_i *in the database schema, there exists a node* $V_i \in V$*, and for each non-nullable foreign key* $FK_{R_i}(R_j)$ *there exists an edge* $e = (V_j, V_i, w) \in E$ *where* $w = savings(R_i, R_j, FK_{R_i}(R_j))$*.*

The Nest-Graphs for the example databases are in Figures 6 and 7 respectively. For simplicity, all primary and foreign keys are defined to occupy 4 characters of space, and we will use the simplest form of $savings(R_i, R_j, K) = sizeOf(K) * |R_i|$. Relation sizes are given in parentheses under their respective nodes in the graphs.

5 Mapping from Relational to XML

The mapping from the relational model to an XML DTD has these general steps:

- Extract relational schema information including foreign key constraints and relation sizes from the database.
- Use the extracted information to build a Nest-Graph G.
- Use the classical algorithm by Edmonds [7] to calculate the maximum weight

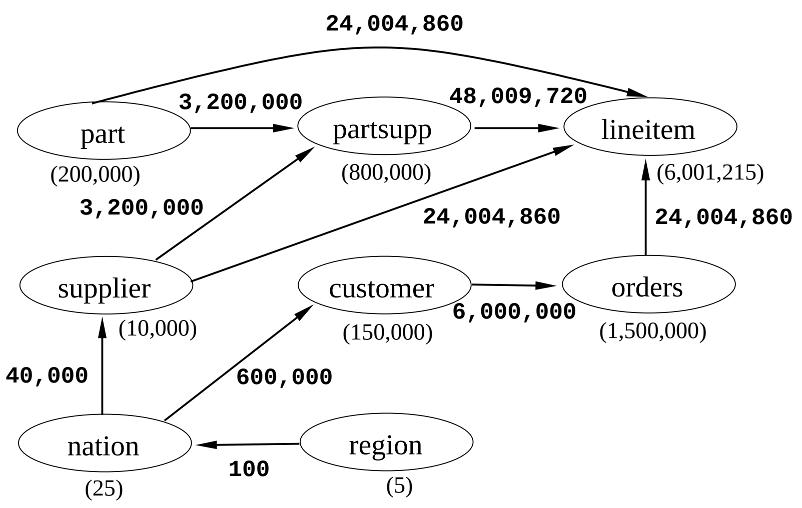

Fig. 7. Nest-Graph for TPC-H Database

arborescence T of G.

- The maximum weight arborescence is a maximal spanning tree (MST) T. Use T to generate the resulting nesting in the output XML DTD. Given an edge $e = (V_j, V_i, w)$ of T, the two corresponding relations are $R_j(A_1, A_2, ..., A_m)$ and $R_i(B_1, B_2, ..., B_n)$ respectively. Let $A_1 = PK_{R_j}$, $B_1 = PK_{R_i}$, and $B_k = FK_{R_i}(R_j)$. For illustration, the primary keys and foreign keys are encoded as XML attributes, and all other relational attributes are encoded as XML elements.
  - For each edge $e = (V_j, V_i, w)$ of T, we will nest R_i under R_j (and omit B_k) in the XML DTD such as:

```
<!ELEMENT R_j(A_2, ..., A_m, R_i*>
   <!ATTLIST R_j A_1 ID>
<!ELEMENT R_i(B_2, B_3,..., B_k-1, B_k+1,...B_n>
   <!ATTLIST R_i B_1 ID>
```

  - For each edge $e = (V_j, V_i, w)$ where $e \in G \wedge e \notin T$, this relationship will be captured using ID/IDREF. Under the element R_i will be a IDREF attribute of the form:

```
<!ELEMENT R_i(B_2, B_3,..., B_k-1, B_k+1,...B_n>
<!ATTLIST R_i B_k IDREF>
```

Extraction of the schema information is performed using standardized access technologies such as JDBC. The Nest-Graph is represented using an adjacency list structure for representing graphs. The algorithm to calculate the maximum weight arborescence T of a graph G deserves some explanation as it is not a commonly used algorithm.

Edmonds' algorithm [7] is used to solve the maximum cost arborescence problem which involves finding the maximum cost spanning tree (MST) T in a directed, weighted graph G with a given root node r. Note that the most efficient known algorithm by Gabow *et al.* [12] for this problem has cost

$O(nlogn + m)$,where n is the number of nodes and m is the number of edges. The basic algorithm is as follows:

- Discard any incoming edges to the root r. For all other nodes, select the entering edge with maximum cost. Let the selected $n - 1$ edges be the set S.
- If $G(V, S)$ contains a cycle
  - · Contract the nodes in the cycle into a pseudonode k. For each edge $e = (i, j, w_{ij})$ where i is a node outside the cycle, and j is a node inside the cycle, let $w_{ik} = w_{ij} - (w_{i(j),j} - \delta)$, where $w_{i(j),j}$ is the weight of edge $f \in S$ currently directed into j and δ is the minimum weight edge in the cycle.
  - · Recursively repeat the maximum edge selection/condensing procedure for the condensed graph until no cycles are present. Let S' be the set of edges of the MST for the condensed graph with no cycle.
  - · Replace each pseudonode with the entering edge with maximum weight (in the opposite order to which they are formed) and add it to S'. After all pseudonode expansions are complete, S' is the set of edges of the maximum spanning tree.

Given a directed, weighted graph G the algorithm will return the maximum spanning tree T of G if one exists. For our space optimization problem, the algorithm will select the nestings (edges) of maximum benefit. The tree constructed mimics the hierarchical model construct available in XML. Edmonds' algorithm cannot be directly applied to a Nest-Graph as it only returns a MST if one exists. A MST in a Nest-Graph will only exist if there is only one top-level node, and all other nodes are reachable from that top-level node. To get around this issue, we introduce a new node r' to G, and a set of edges E' where each $e = (r', V_i, 0) \in E'$ for all $i = 1..n$. The introduction of this node and edges connecting it to all others nodes with zero weight will guarantee that a MST will always be found, and the edges of zero weight will have no effect on the calculation of the optimal nestings.

Application of Edmonds' algorithm to the Nest-Graph for TPC-H produces the MST T given in Figure 8. By summing up the edges in T, we can calculate the space savings by introducing nesting in our DTD when mapping from the relational model. For TPC-H at scale factor 1, the space savings is 57,849,820 characters or 14,462,455 data values. Equivalently, the savings is 43.5% of all foreign keys or 12.4% of all data values. The DTD for TPC-H is given in Figure 9.

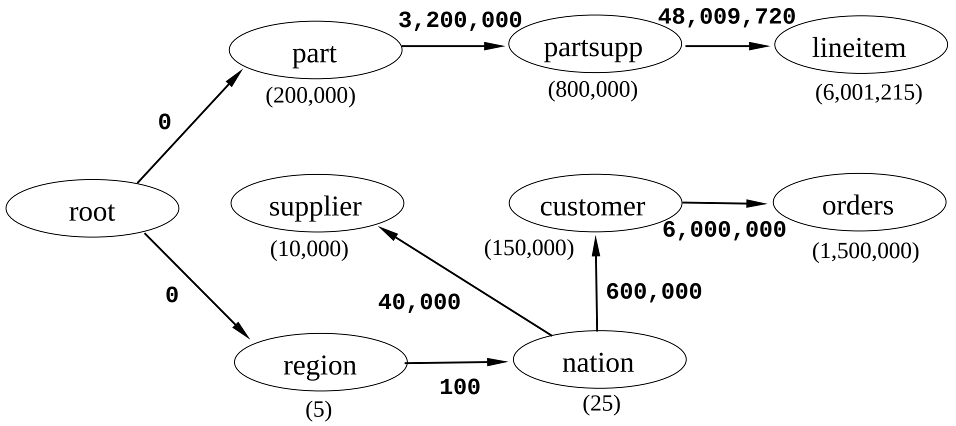

Fig. 8. Maximum Spanning Tree for TPC-H Database

```
<!ELEMENT root(part*,region*)>
<!ELEMENT part(P_NAME,P_MFGR,P_BRAND,P_TYPE,P_SIZE,P_CONTAINER,
       P_RETAILPRICE,P_COMMENT,partsupp*)>
    <!ATTLIST part P_PARTKEY ID>
<!ELEMENT region(R_NAME,R_COMMENT,nation*)>
    <!ATTLIST region R_REGIONKEY ID>
<!ELEMENT partsupp(PS_AVAILQTY,PS_SUPPLYCOST,PS_COMMENT,lineitem*)>
    <!ATTLIST partsupp PS_SUPPKEY IDREF>
<!ELEMENT nation(N_NAME,N_COMMENT,customer*,supplier*)>
    <!ATTLIST nation N_NATIONKEY ID>
<!ELEMENT lineitem(L_LINENUMBER,L_QUANTITY,L_EXTENDEDPRICE,L_DISCOUNT,L_TAX,
       L_RETURNFLAG, L_LINESTATUS,L_SHIPDATE,L_COMMITDATE,L_RECEIPTDATE,
       L_SHIPINSTRUCT,L_SHIPMODE,L_COMMENT)>
    <!ATTLIST lineitem L_ORDERKEY IDREF>
<!ELEMENT customer(C_NAME,C_ADDRESS,C_PHONE,C_ACCTBAL,C_MKTSEGMENT,
       C_COMMENT,orders*)>
    <!ATTLIST customer C_CUSTKEY ID>
<!ELEMENT supplier(S_NAME,S_ADDRESS,S_PHONE,S_ACCTBAL,S_COMMENT)>
    <!ATTLIST supplier S_SUPPKEY ID>
<!ELEMENT orders(O_ORDERSTATUS,O_TOTALPRICE,O_ORDERDATE,O_ORDERPRIORITY,
       O_CLERK,O_SHIPPRIORITY,O_COMMENT)>
    <!ATTLIST orders O_ORDERKEY ID>
```

Fig. 9. Condensed DTD for TPC-H Database (omitting #PCDATA declarations)

5.1 User-Directed Mapping

Exploiting Nest-Graphs and Edmonds' algorithm provides a mechanism for determining the optimal nesting of relational schemas into an XML DTD given no constraints. However, the optimal nesting in terms of space efficiency is not always the optimal nesting in terms of usability, readability, or other metrics. For this reason, it is useful for the user to be able to specify constraints on the XML DTD that should be satisfied without having to specify a total mapping to the system. In this case, the system must construct the optimal, space efficient XML DTD that respects the user constraints. For instance, the user may be able to improve readability by specifying different top-level nodes and certain nestings that should be utilized. This way the user can be involved to whatever degree is required in the actual conversion process.

Constraints can be incorporated into the mapping algorithm by appropriate modifications to the Nest-Graph. We examine four different types of constraints:

- Specification of top-level nodes
- Specification of edges (nestings) that must be present in the XML DTD
- Specification of edges (nestings) that should not be present in the XML DTD
- Handling nullable foreign keys

To specify such constraints, the user could potentially manipulate a graphical view of the Nest-Graph, but how the constraints are specified is not our focus.

The first constraint type is the specification of top-level nodes. By allowing a user to specify that a node is a top-level node, we are forcing the system not to nest the node (relation). The user can specify that any node V_i is a top-level node, even nodes that have incoming edges. If V_i has incoming edges, then the incoming edges of V_i are removed. This will cause V_i to be a top-level node when the Edmonds' algorithm is run, and R_i will not be nested in the XML DTD.

Constraints on edges are valuable because they allow the user to control the nesting process at a fine degree of precision. In many cases, the user is not interested in all the nestings, but some nestings may be especially important to be present or not to be present in the final DTD. For example, although there is a minor savings by nesting `customer` and `supplier` information under `nation` and `region` in TPC-H, this savings may not be enough to justify organizing the information in this way. Another example is that it is almost always more preferable for `lineitem` information to be nested under the corresponding `order`, even though more savings is achieved by nesting under `partsupp`.

Edges (nestings) that must be present in the final DTD can be guaranteed by setting the edge weights to $+\infty$. Edges that must not be present in the final DTD are insured by removing the edges from the graph. The algorithm will then determine the optimal space efficient mapping given the user specified constraints. Thus, the major benefit of this mapping method is that the user can specify as many or as few constraints on the final XML DTD, and the system will compute the optimal solution given the constraints.

Finally, foreign keys that may be null can be handled by giving the user an option on their encoding. The user can select if nullable foreign keys should be always encoded (edges are present in Nest-Graph) or not encoded (edges are not present in Nest-Graph). If a foreign key may be null and it is nested, then when data translation is performed the system must insert a dummy null record in the parent element.

5.2 General Query Translation

The algorithm presented automatically creates the optimal, space-efficient XML schema given an input relational schema. Converting relational schemas to XML schemas is valuable when coverting relational database applications to use XML storage. It is also useful when sharing entire databases as the data can be extracted from the server into XML, transported to its final destination, and then reloaded into another system. A related problem to converting an entire schema is the problem of converting the result of a query over a relational schema into XML.

There is a significant advantage in nesting a relation formed by querying over several base relations, as the query result will contain redundant data. Space savings can be achieved by taking the de-normalized query result and applying some normalization by exploiting nesting. Note that this is a different problem than converting a set of relational schemas into an XML schema as discussed previously. When encoding an entire database, the relations are normalized and nesting is used to save space by eliminating foreign keys. In the problem of translating query results, the result is a single, non-normalized relation, and the goal is to use nesting to encode the data more efficiently. For example, consider the query in Figure 10 on TPC-H. This query would return over 6 million records (the size of the `lineitem` relation). If this query result was encoded as a flat relation in XML, then there would be an enormous amount of wasted space encoding duplicate values from the attributes in the `nation`, `customer`, and `orders` relations. For instance, since the cardinality of `nation` is 25, the XML encoding would redundantly encode approximately 6 million nation names. Obviously, a huge space efficiency improvement can be obtained by re-normalizing the result using nesting. We will not argue in this work whether re-normalizing using nesting is applicable to all query results or even desirable from an end-user perspective. Rather, we will demonstrate how space savings can be achieved for certain types of queries if the user is willing to accept a nested result.

```
SELECT n_name, c_name, o_orderdate, o_orderkey, l_partkey, l_quantity, s_name
FROM customer as C, orders as O, lineitem as LI, supplier as S, nation as CN
WHERE C.c_custkey = O.o_custkey AND LI.l_orderkey = O.o_orderkey and
      LI.l_suppkey = S.s_suppkey AND C.c_nationkey = CN.n_nationkey
```

Fig. 10. Natural Join Query Example

A common type of query involves selection, projection, and natural joins between relations on primary and foreign keys. These queries are naturally modeled using Nest-Graphs. Given a query Q and database Nest-Graph G, a Nest-Graph G' for the query is a subset of G that contains only the nodes (relations) and edges (primary key-foreign key pairs joined on) in Q. The weight of each node N_i is the expected cardinality of the relation $|R_i|$ in the result. For each edge entering a node R_i, the edge weight is $(N - |R_i|) * A$ where N is the number of tuples in the query result and A is the number of attributes from R_i in the result. The graph produced contains edges $e = (R_i, R_j, w)$ where the weight w is the number of attribute values that do not need to be encoded by nesting R_j under R_i. A root node is added to G' with outgoing edges to all nodes with no incoming edges. An example graph for the query in Figure 10 is in Figure 11.

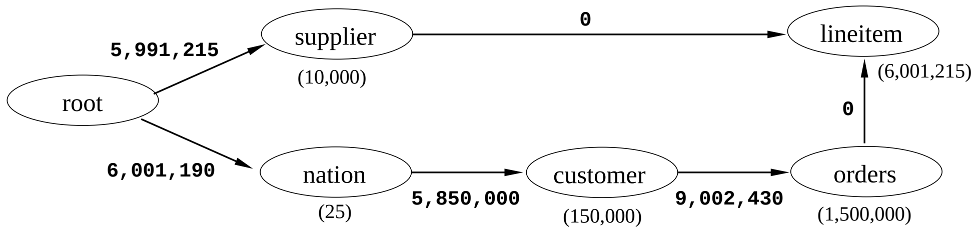

Fig. 11. Nest-Graph for Example Query

An estimated optimal nesting can be produced using Edmonds' algorithm. However, this nesting may require ID/IDREF which is probably undesirable for a query result. To determine a single nested tree, perform a separate breadth-first search starting from each distinct edge from the root node. Then, for each search, sum the weight of all edges traversed. Select the root node edge (and resulting BFS tree) with greatest weight. This produces the optimal nested tree for the query result. For any edges in G' not in the BFS, these attributes will be redundantly encoded. The optimal nesting for the example query is given as a DTD in Figure 12.

```
<!ELEMENT root(nation*)>
<!ELEMENT nation(N_NAME, customer*)>
<!ELEMENT customer(C_NAME, orders*)>
<!ELEMENT orders(O_ORDERKEY, O_ORDERDATE, lineitem*)>
<!ELEMENT lineitem(L_PARTKEY, L_QUANTITY, S_NAME)>
```

Fig. 12. DTD for Example Query (omitting #PCDATA declarations)

This algorithm works for queries involving selection, projection, and natural joins on foreign keys. Projecting attributes in the query does not affect the algorithm, but does change the edge weights (constant A) and the absolute amount of savings (since fewer attributes will be redundantly encoded.) Selection criteria changes the node weight *estimates* based on the criteria and initial relation cardinalities. If the estimates are very inaccurate, this may affect if the best nesting is chosen. Parsing the query to determine reliable cardinality estimates is beyond the scope of this work.

Group by and aggregate queries can sometimes be nested with great benefit. This occurs when there is at least 2 group by attributes from different relations and the set of group by attributes includes keys from two (or more) relations connected using inclusion dependencies. An example is query #18 in the TPC-H benchmark that groups on `c_name`, `o_orderkey`, `o_orderdate`, and `o_totalprice`. Using the inclusion dependencies in the database Nest-Graph, the system can determine that nesting order attributes under the customer attributes is always the most space efficient. Note that without the presence of keys (example grouping on `c_name`,`o_date`) the optimal nesting may be data dependent and cannot be determined using schema information alone. In this case, the optimal nesting depends on if there are more orders per day or more orders per customer.

Other types of queries are generally more complicated to translate and are not the focus of this work. Queries involving general join criteria (such as `customer` $\bowtie_{cname>oclerk}$ `orders`) cannot have nestings determined for them using inclusion dependencies because they no longer give accurate estimates of the cardinalities of the relationships. For example, a customer may have many orders (1:N relationship), so nesting order under customer is usually preferable. However, a general join condition like above causes a M:N relationship between customer and order tuples, and thus it is difficult to determine based only on schema information which nesting (if any) should be applied. In these cases, data-level nesting is the only alternative.

6 Experimental Results

A Java program that uses JDBC to extract schema information including foreign keys was constructed to test the conversion algorithm. The program connects to a database using JDBC, extracts schema information, gathers relation size statistics, then uses that information to build a Nest-Graph. The Nest-Graph is transformed into a form compatible with Edmonds' algorithm, and the algorithm is run to find a MST. Then, the MST is converted into an XML DTD. The savings with respect to the number of data values not encoded are given in Figure 13. [7]

As can be seen in the results of encoding entire databases in XML, a significant number (40-50%) of foreign keys do not have to be encoded. When migrating large database instances to XML, this results in a significant savings. Elimination of all foreign keys is only possible if the Nest-Graph is a tree. Even when considering the total number of attributes, eliminating between 10 to 30% of all attributes using nesting is significant. The nesting has the added advantage of improving the readability of the data and reducing the number of joins required to connect related data.

Some experiments were also performed on converting query results over relational schema into XML. In these experiments, each query was used to define

[7] The results on TPC-H in [17] are identical to the results here except those results were obtained by human optimization of the translation. The algorithm in [17] had no mechanism for computing the optimal translation without human involvement.

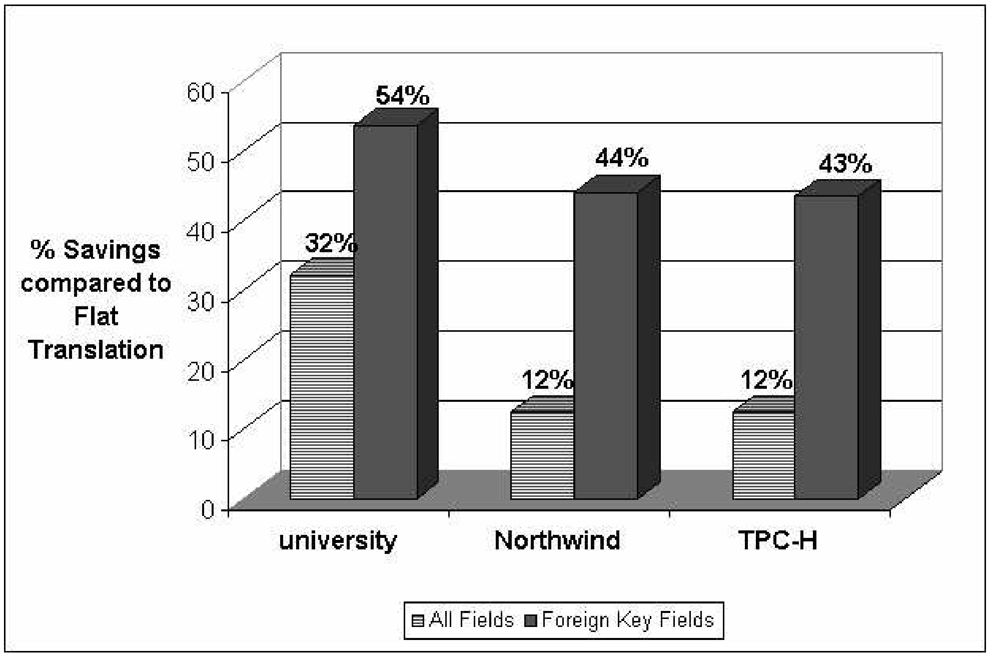

Fig. 13. Percentage of Data Values not Encoded with Optimal DTDs

a subset of the database Nest-Graph, and the query result was normalized using nesting without ID/IDREF (as a single tree). Although several of the queries involved selection conditions which changes the node weights, the node weights used were the base relation cardinalities. Even without changing the node weights according to the query, the optimal nesting was determined in the three test cases. The three test queries were two queries from the TPC-H standard (#2 and #18) and the example query in Figure 10. Query #2 is a natural join query over 5 relations, and query #18 is a group by query with 5 group by attributes.

As can be seen in Figure 14, significant savings can be achieved by normalizing the query result using nesting. The normalization can be determined before query execution using inclusion dependencies for many common queries (those involving natural joins). For the example query, nesting the query result reduces the number of data values in the XML document by 50% which corresponds to 20,904,839 fewer values. Even when utilizing compression techniques, this is significant as it results in smaller files to compress/decompress. Future work involves a more detailed investigation and experimental validation on the types of queries that can be normalized using only inclusion dependencies and the amount of space savings achieved.

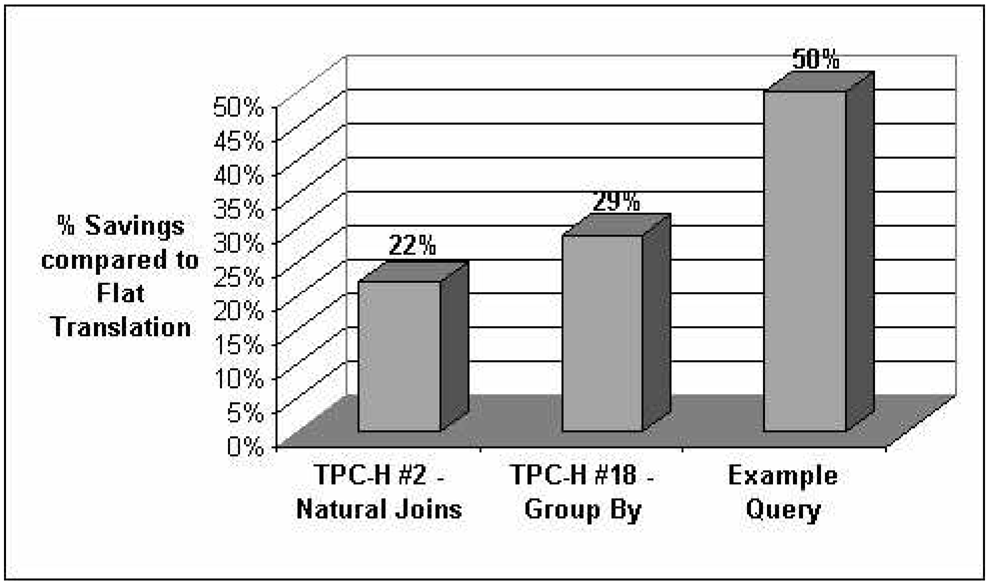

Fig. 14. Percentage of Data Values not Encoded with Nesting

7 Discussion and Contributions

The contribution of this work is a method for translating relational schemas to XML without requiring user input or intermediate data models. The translation process can be user-directed to varying degrees, and results in space efficient XML documents. The space efficiency cost-metric proposed is useful, although other metrics are possible including those that factor in query costs as well. Using Edmonds' algorithm to minimize the cost-metric in translation is unique. The algorithm can process user constraints and calculate the most efficient schema under those constraints. This is the first mapping method to incorporate both automatic cost-based evaluation and user constraints.

There are some issues not tackled by the current algorithm. First, although the algorithm can be modified to handle non-nullable foreign keys, it may be desirable to consider the impact of nulls in the cost formula directly. Further, the XML DTD produced is optimal in the sense of schema-level nesting only based on the current schema normalization. If the database schema is not fully normalized, it may be possible for improvements to be made by data-level nesting using NeT [17]. It may be useful to use data mining techniques to determine if further normalization is possible during the translation.

It may also be interesting to look at heuristics to improve the readability of the XML schema without user specified constraints. For example, grouping `lineitem` information under `order` is desirable, but not guaranteed in the base algorithm. However, a heuristic that favors nesting of elements when the cardinality of their relationship is low may be beneficial. In this case, that would favor `lineitem` information under `order` as the cardinality of the relationship is on average 1:7 in comparison to 1:20 for `partsupp`.

We did not examine the case of multiple foreign keys from one relation to another. An example would be an order relation having three foreign keys to a company relation. Each foreign key represents a different role: shipper, supplier, buyer, etc. In this case, it is probably preferable not to nest because the likelihood of one company record playing multiple roles is a distinct possibility and nesting would result in introduction of redundancy.

Even more interesting is the possibility of introducing controlled redundancy to improve query efficiency. It is possible to introduce negative edges in the graph representing space inefficiencies by using nestings that result in redundant data. By itself, this is useful to determine when the cost of replicating information is not prohibitive. In combination with a query cost metric, it would be possible to determine the optimal XML schema in terms of some balance between space and query efficiency.

8 Future Work and Conclusions

Given the growing importance of representing data in XML format and its common preexistence in relational databases, it is important to have an efficient and user-friendly tool for automating construction of XML schemas from relational database schemas. This work describes a mapping tool that uses a space efficiency cost metric and user constraints to generate optimal schemas. It is an improvement over current approaches which either do not have formal methods for capturing user constraints (CoT [17]) or require the user to exactly specify the entire XML schema in the form of an extraction query [10].

Future work involves expanding the algorithm to consider nestings that introduce redundancy to increase query performance at the sacrifice of space efficiency.

References

[1] M. Arenas and L. Libkin. A Normal Form for XML Documents. In *Proceedings of the 21st ACM Symposium on Principles of Database Systems (PODS 2002)*, pages 85–96, 2002.

[2] S. Banerjee, V. Krishnamurthy, M. Krishaprasad, and R. Murthy. Oracle 8i - The XML Enabled Data Management System. In *Proceedings of the 16th International Conference on Data Engineering*, 2000.

[3] L. Bird, A. Goodchild, and T. Halpin. Object Role Modelling and XML-Schema. In *Proceedings of the 19th International Conference on Conceptual Modeling (ER 2000)*, pages 309–322, 2000.

[4] P. Bohannon, J. Freire, P. Roy, and J. Simeon. From XML Schema to Relations: A Cost-Based Approach to XML Storage. In *Proceedings of the 18th International Conference on Data Engineering (ICDE 2002)*, pages 64–76, 2002.

[5] M. Carey, J. Kiernan, J. Shanmugasundaram, E. Shekita, and S. Subramanian. XPERANTO: A Middleware for Publishing Object-Relational Data as XML Documents. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB 2000)*, pages 646–648, 2000.

[6] W. Du, M. Lee, and T. Ling. XML Structures for Relational Data. In *Proceedings of the 2nd International Conference on Web Information Systems Engineering (WISE'01)*, pages 151–160, 2001.

[7] J. Edmonds. Optimum Branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

[8] R. Elmasri and S. Navathe. *Fundamentals of Database Systems.* Addison-Wesley, 2000.

[9] D. Embley and W. Mok. Developing XML Documents with Guaranteed 'Good' Properties. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER 2001)*, pages 426–441, 2001.

[10] M. Fernandez, Y. Kadiyska, D. Suciu, A. Morishima, and W. Tan. SilkRoute: A Framework for Publishing Relational Data in XML. *ACM Transactions on Database Systems*, 27(4):438–493, 2002.

[11] M. Fernandez, A. Morishima, and D. Suciu. Efficient Evaluation of XML Middle-ware Queries. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 103–114, 2001.

[12] H. Gabow, Z. Galil, T. Spence, and R. Tarjan. Efficient Algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122, 1986.

[13] T. Halpin. *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design.* Morgan-Kaufmann, 2001.

[14] IBM. IBM DB2 XML Extender, (*http://www-3.ibm.com/software/data/db2/extenders/xmlext/*), 2002.

[15] G. Jaeschke and H.-J. Schek. Remarks on the Algebra of Non First Normal Form Relations. In *Proceedings of the 1st ACM Symposium on Principles of Database Systems (PODS 1982)*, 1982.

[16] M. Klettke, L. Schneider, and A. Heuer. Metrics for XML Document Collections. In *EDBT 2002 Workshops*, pages 15–28, 2002.

[17] D. Lee, M. Mani, F. Chiu, and W. Chu. NeT & CoT: Translating relational schemas to XML schemas using semantic constraints. In *Proceedings of the 11th CIKM*, pages 282–291, 2002.

[18] Y. Lien. Hierarchical Schemata for Relational Databases. *ACM Transactions on Database Systems*, 6(1):48–69, 1981.

[19] W. Mok, Y-K Ng, and D. Embley. A Normal Form for Precisely Characterizing Redundancy in Nested Relations. *ACM Transactions on Database Systems*, 21(1):77–106, 1996.

[20] R. Orsini and M. Pagotto. Visual SQL-X: A Graphical Tool for Producing XML Documents from Relational Databases. In *WWW Posters*, 2001.

[21] M. Roth and H. Korth. The Design of 1NF Relational Databases into Nested Normal Form. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, pages 143–159, 1987.

[22] M. Roth, H. Korth, and A. Silberschatz. Extended Algebra and Calculus for Nested Relational Databases. *ACM Transactions on Database Systems*, 13(4):389–417, 1988.

[23] M. Rys. State-of-the-Art XML Support in RDBMS: Microsoft SQL Server's XML Features. *IEEE Data Engineering Bulletin*, 24(2):3–11, 2001.

[24] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, and B. Reinwald. Efficiently Publishing Relational Data as XML Documents. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB 2000)*, pages 65–76, 2000.

[25] P. Tolani and J. Haritsa. XGRIND: A query-friendly XML compressor. In *Proceedings of the 18th International Conference on Data Engineering (ICDE 2002)*, 2002.

[26] W3C. XML-Schema Part 0: Primer (*http://www.w3.org/TR/xmlschema-0/*), 2000.