An Analysis of Alternative Methods for Storing Semistructured Data in Relations*

Igor Nekrestyanov, Boris Novikov, and Ekaterina Pavlova

University of St. Petersburg, Russia
{igor,katya}@meta.math.spbu.ru, borisnov@acm.org

Abstract. Although the major source of semistructured data is WWW and therefore the representation of data cannot be controlled by any single site, in many cases the data are replicated and stored locally in a database to support sophisticated query processing.
To date a number of approaches to store data in relations were proposed. In this paper we present an analysis of alternative mapping schemes.

1 Introduction

In recent years there has been an increased interest in managing semistructured data. One of most popular data models for semistructured data is OEM [1], that is based on labeled directed graphs.

The XML is emerging as standard to exchange semistructured data. To retrieve data from semistructured sources a number of novel query languages for semistructured data and XML were proposed [2].

There are several possible approaches to store semi-structured data and to execute queries against them:

Special-purpose DBMS Such systems are particularly tailored to store and retrieve semistructured data, using specially designed structures and indices [8, 10] and particular query optimization techniques [9, 7]. Examples are Strudel, Lore [1].

OODBMS In this approach, the rich data modeling capabilities of object-oriented database systems are exploited. Processing of SGML documents using OODBMS was discussed in [3]. The conclusion was that this is feasible with some extensions to OO query languages.

RDBMS In this approach semistructured data are mapped into tables of a relational scheme and queries in a semistructured query language are translated into SQL queries [14, 4].

It is still unclear which way will find wide-spread acceptance. While special-purpose approach will clearly work it is going to take a long time before such systems will mature. The downside of using semi-structured techniques is that this

* This work was supported in part by RFBR (grant 98-01-00436) and INTAS (grant OPEN-97-11109).

approach turns it back on 20 years work invested in relational and object-oriented database technology. Therefore, usage of relational [14, 6], object-oriented [3, 13] and object-relational [15] databases for storage of semistructured data recently attracted attention.

In this work we focus on relational model. A number of methods to store semistructured data in relations were discussed [4, 6, 14]. The crucial decision here is to select relational scheme to represent semistructured data.

One approach is to use basic, universal mapping schemes of generic directed labeled graph to relations [6]. Alternative is to use schemes which are tailored for semistructured data instance. The scheme extraction as cost-optimization problem was discussed in [4]. It is also possible to type semistructured data [16, 11] and store objects of each type in separate table.

In the frame of this work we evaluate several techniques to store semistructured data in relations. We show that for some access patterns the approaches based on scheme extraction are preferable than universal graph mappings. We also compare several algorithms in terms of quality of extracted scheme.

Rest of the paper is organized as follows: we briefly review methods in the next section and further discuss them in section 3; section 4 contains results of our experiments; we describe related works in section 5 and conclude.

2 Mapping Schemes

2.1 Universal Mapping Schemes

Universal mapping schemes are schemes which can be used to store generic labeled directed graph, i. e. they are applicable to storage of any semistructured data. We will consider only several representatives of universal schemes [6]:

Edge All data are stored in single table of triples $(source, label, target)$ describing edges of the graph.

Attribute A separate table for each existing *label* is created. Table consists of pairs of $(source, target)$ form describing edges.

Wide According to this approach all data are represented by single table whose columns correspond to all labels found in the graph. Conceptually, this table corresponds to result of an outer join of all *Attribute* tables.

2.2 Extracted schemes

Extracted schemes are schemes which are tailored for specific semistructured data instance. Note, that such scheme may not cover all data. However, it is always lossless: parts of the semistructured data that do not fit the scheme are stored in an "overflow" graph that is stored as table of triples $(source, label, target)$ describing graph edges. In this work we consider following approaches:

STORED The technique for generation of relational storage mappings given semistructured data instance was proposed in [4]. Effectiveness of mapping

is determined by several competing goals. These goals are modeled as cost-optimization problem. Considered storage-cost function take into account such parameters as number of tables, disk space, number of nulls, etc.
The problem of computing an optimal storage mapping is *NP-hard* in size of *data*. Therefore, authors consider heuristics, starting from frequent tree patterns in the data discovered by adopted version of WL's data mining algorithm [16]. To reduce complexity only relatively often used paths are considered during data mining step.

Jump An algorithm for deriving approximate type hierarchy from semistructured data was proposed in [11]. Proposed technique is based on the relative importance of some attributes in a larger set that is measured by $Jump$ function. Given a set of labels S it is defined as follows

$$Jump(S) = \frac{|\{o : attributes(o) = S\}|}{|\{o : attributes(o) \supseteq S\}|}$$

where $attributes(o)$ is a set of labels on the outgoing edges at object o.

Datalog Approximate typing of semistructured data using Datalog was described in [12]. Typing is in the form of a monadic datalog program with each intensional predicate defining separate type.
The minimal perfect typing is performed using greatest fixpoint semantics of monadic datalog programs. Adopted k-clustering algorithm is used to collapse 'similar' types. The clustering is based on weighted symmetric difference between the bodies of their rule definitions.

Note, that *Jump* and *Datalog* do not produce relational scheme explicitly. Straightforward solution is to store objects of the same type in the same table.

3 Comparative Analysis

3.1 Comparison of Schema Generation Methods

Comparison (see table 1 for summary) is based on following criteria:

Support for Cycles This is essential for real-world data. Indeed, OEM graph of many Web-documents is cyclic.

Support for Multiple Outgoing Edges with Same Label Multiple edges with same label conceptually are represented by set-valued attributes which are not directly supported by relational databases. Solution is either use several attributes in relation or use nested relation.

Unnesting Objects with similar deep structure are unnested: only values of their leaves need to be stored in relations.

Computational Complexity In this case complexity usually depends on size of *data* n that might be very big.

STORED expects that cycles are cut in some arbitrary, but systematic way. This is not only affects resulted scheme but also causes new untyped references

Criteria	STORED	Jump	Datalog
Cycles in the Data	no	yes	yes
Multiple Outgoing Edges with Dame Label	both	nested tables	nested tables
Unnesting	yes	no	no
Computational Complexity[1]	$O(c^m)$	$O(n^2)$	$O(n^2)$

Table 1. Methods comparison

to be introduced. Note that untyped references cannot be handled in relational systems without proliferation of joins — for each possible reference type [14].

Usage of only nested relations for representation of multiple outgoing edges with label seems to be not so serious limitation because usage of data clustering techniques significantly decreases expenses of extra join.

3.2 Query Optimization

Access Patterns The performance of a data structures depends heavily on access patterns used to process these data, and data representation is always defined with certain access pattern types in mind. Of course, the access patterns may depend on logical structure of dataset.

For any query the amount of extracted data is restricted in two ways: conditions on attribute values, or selection predicates, or logical location of data, or paths. For example, in SQL query the selection predicate is placed in WHERE clause, while path is specified in FROM clause.

An important feature of query languages for semistructured data is that complex path expressions may be included into any part of a query, specifying navigation through the graph. The mapping of such path expressions to any relational representation contains several join operations and therefore may significantly slow down the evaluation of a query.

Completely different type of access pattern can be found for such data sets as, for example, DBLP bibliography collection, which is used as a common test data set for research in semistructured data. In this, relatively large dataset, the graph representing data in semistructured model, is essentially a tree with 3 levels. A typical meaningful query should restrict the result, using predicates on attribute values, rather than complex path expressions.

Obviously, the performance of any particular relational representation depends on which of access patterns dominates in typical queries.

Universal vs Derived from Data Approaches In general, it should be expected that any universal representation is less efficient than one based on derived data structure. Further, we expect that *Edge* representation might be more efficient than *Wide* representation for queries where navigation dominates. We verify these hypotheses in our performance experiments.

Although other research reports *Edge* representation as relatively efficient (e.g. for bulk loads and reconstruction of the whole graph as a document) [6],

	DBLP	DBLP-small	Shakespeare	CyclicDataset
Real-world	yes	yes	yes	no
Have cycles?	no	no	no	yes
Number of edges	1900k	90k	330k	30k
Maximum path length	3	3	7	—

Table 2. Dataset's summary information

it cannot be efficient for more complex or more selective queries, because both navigation and predicate checking require several joins of this table to itself to be performed.

The *Attribute* approach was reported as best [6]. However we feel it may be efficient for navigation but not conditions on attribute values. The reason is that the while size of joined tables is significantly less than in *Edge* representation, but the number of joins is the same.

The *Wide* approach seems especially suitable for selection based on attribute values. The advantage of this representation is that attribute columns may be indexed to ensure very efficient query evaluation by relational engine. The disadvantages are large number of NULL values and huge size of the table.

Deriving scheme from the data allows to reduce the number of NULLs as well as overall size of database. Most likely, the best results can be achieved if both approaches are combined — regular parts of the data are stored according to derived scheme and universal mapping is used for "overflow" (that is, exceptions).

4 Experiments

4.1 Test datasets

DBLP[2] This is the collection of XML-like files with bibliography data. Data is quite irregular: some entries have multiple author's. optional url's, etc.

DBLP-small This is the subset of DBLP in which the numbers of *conf's* and *journal's* entries were limited by 3000. These two kinds of entities are dominating in the DBLP dataset and this cause other kinds of entities to be considered as random irregularity.

Shakespeare[3] This is the set of plays of Shakespeare marked up in XML. Note, that it has deepest tree structure among considered datasets.

CyclicDataset This is synthetic dataset that has cycles in the data. Usage of synthetic data is attractive for our purposes because we are able to measure the effect of various perturbations of the data on the extracted scheme.

4.2 Scheme Quality

We measure scheme quality in terms of number of generated tables, coverage, number of null values, overall size and scheme generation time. Results of experiments for two datasets are summarized in the table 3. For other datasets

	DBLP			DBLP-small		
	STORED	Jump	Datalog	STORED	Jump	Datalog
Number of tables	4	18	40	7	12	25
Coverage	90%	91%	70%	90%	88%	72%
Number of Nulls	70k	196k	65k	10k	19,5k	14k
Overall size (Mb)	67	72	69	3,2	3,5	3,4
Schema generation time	24 h	8 min	40 min	90 min	2 min	15 min

Table 3. Experimental results for DBLP and DBLP-small

the situation is essentially the same with two notes: all methods perform worse on *CyclicDataset* and scheme extraction with STORED for *Shakespeare* dataset took more than 60 hours.

While STORED is specially designed to solve scheme generation problem (unlike both other approaches) it has a number of weak sides. One of the most important is very high complexity of data-mining steps that forces exclusion of rarely used paths from the consideration. While this probably does not reduce coverage too much it does not look like a best idea. For example, for DBLP dataset this implies that only information about *conf* and *journal* entities will be stored in typed tables and other information will be in "overflow". However this information actually consists of fairly regular subsets, such as *books*, and is preserved by both other approaches.

Quality of results *Datalog* approach was not as good as we expected but this seems crucially depend on distance function used on the clustering stage.

It's interesting what simple *Jump* approach shows results comparable to results of much more sophisticated (and computationally expensive) STORED technique. However, we do not feel that this is absolutely best approach to use because it did not perform well in some cases.

4.3 Query Performance

The experiments with relational database engine included several steps. Most of these steps were measured for two datasets: DBLP and DBLP-small.

For each dataset, both *Edge* and *Wide* representations were created in the database. In addition, some of *derived* representations were created as projections of universal representations. Two sets of queries, parametrized with values in selection predicates, were used: simple queries, such as "list table of contents for particular journal issue", and more complex queries involving several joins. All queries return small amount of data (up to few hundreds of rows).

The first measured step was bulk load of data into database. The typical results are shown in the table 4. To improve performance, indices were always built after initial bulk load as a separate step. The time of full scan on these tables ranges from sub-second time for small dataset to 14 and 24 seconds for different representations of the full collection.

Dataset	Mapping Scheme	Dataset size (Mb)	Load time	Table size (Mb)
DBLP-small	*Edge*	2.79	44 sec	5.8
DBLP	*Edge*	60.8	25 min	106
DBLP-small	*Wide*	4.51	77 sec	9.7
DBLP	*Wide*	110.3	39 min	181

Table 4. Typical results of bulkloading experiments

Query	*Wide*	*Edge*	*Attribute*	*Derived*
Simple	1.4 – 2.9	5.2 – 19.8	2.7 – 10.1	0.5 – 1.1
Complex	15.9 – 27.3	5.7 – 23.6	4.1 – 16.8	10.3 – 21.8

Table 5. Typical query performance for DBLP dataset

The time needed for query evaluation varied dramatically depending on physical structure (indices and clustering of tables). Without additional tuning of physical structure, even simple queries took unacceptably long time for any of representations (see table 5). If indices for all columns are built, the *Wide* representation clearly outperforms *Edge* representation.

For complex queries, the performance is slightly better for *Edge* representations, due to smaller size of tables. The *Attribute* and *Derived* representations are better than *Edge* and *Wide*, respectively.

5 Related Work

Usage of several universal mapping schemes of XML data to relational databases was studied in [6]. Performance experiments analyzed the space requirements, the bulkloading times, the running time to reconstruct XML document, the running times of series of queries and update functions for each mapping scheme. Results show that an *Edge* approach is the best overall approach. Our work consider a more sophisticated scheme mappings which are derived from the data.

Virtues and limitation of relational model for processing queries over XML dataset conforming to a scheme were studied in [14]. Results show that it is possible to handle most of queries on XML documents using RDBMS, barring certain type of complex recursion. Authors suggest some extension to relational systems that will improve efficiency of handling XML queries. However, they assumed that all documents conform to some scheme (in form of DTD). Unlike them we consider the situation when nothing about the data is known a-priori.

The problems of dynamic exporting XML data from relational databases are addressed in [5]. Proposed approach is to use virtual XML views on relational data and to automatically rewrite queries on XML data into relational queries. This work is related because they consider evaluation of semistructured queries over relational database. However it is quite different from the focus of our work because it considers relational data with predefined scheme.

6 Conclusion

In this paper we analyzed and compared several different approaches of storing semistructured data in relational database.

The comparison of structure extraction methods shows that sophisticated and time-consuming methods often cannot produce much better structure that more straightforward ones.

Further, the performance of actual representation depends on the contents of data and access patterns. There is no representation which outperforms other for all datasets and/or all access patterns.

For relatively small datasets with path-oriented access, an *Edge* representations are probably best, but for large collections and queries with low selectivity more traditional representation based on derived types provide best performance.

References

1. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *Journal on Digital Libraries*, 1:68–88, Apr. 1997.
2. A. Bonifati and S. Ceri. Comparative Analysis of Five XML Query Languages. *SIGMOD Record*, 29(1), Mar. 2000.
3. V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From Structured Documents to Novel Query Facilities. In *Proc. of the SIGMOD'94*, 1994.
4. A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with STORED. In *Proc. of the ACM SIGMOD'99*, June 1999.
5. M. Fernandez, W.-C. Tan, and D. Suciu. SilkRoute: Trading between Relations and XML. Nov. 1999.
6. D. Florescu and D. Kossmann. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. In *Proc. of the VLDB'99*.
7. D. Florescu, A. Levy, I. Manolescu, and D. Suciu. Query Optimization in the Presence of Limited Access Patterns. In *Proc. of the ACM SIGMOD'99*, 1999.
8. R. Goldman and J. Widom. Approximate DataGuides. In *Proc. of the VLDB'97*.
9. J. McHugh and J. Widom. Query Optimization for XML. In *Proc. of the VLDB'99*, pages 315–326, Sept. 1999.
10. T. Milo and D. Suciu. Index Structures for Path Expressions. In *Proc. of the International Conference on Database Theory*, 1999.
11. S. Nesterov, S. Abiteboul, and R. Motwani. Inferring Structure in Semistructured Data. In *Proc. of the Workshop on Management of Semistructured data*, 1997.
12. S. Nesterov, S. Abiteboul, and R. Motwani. Extracting Schema from Semistructured data. In *Proc. of the ACM SIGMOD'98*, pages 295–306, 1998.
13. D.-Y. Seo, K.-M. Lee, and J.-Y. Lee. Discovery of Schema Information from the forest of Selectively Labeled Ordered Trees. In *Proc. of the Workshop on Management of Semistructured Data*, May 1997.
14. J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proc. of the VLDB'99*, pages 302–314, Sept. 1999.
15. T. Shimura, M. Yoshikawa, and S. Uemura. Storage and Retrieval of XML Documents using Object-Relational Databases. In *Proc. of the DEXA'99*, 1999.
16. K. Wang and H. Lui. Discovering Typical Structures of Documents: A Road Map Approach. In *Proc. of the SIGIR'98*, 1998.