Optimization of XML Data Representation in RDBMS *

Dmitry Barashev, Ekaterina Gorshkova
University of St.-Petersburg, Russia
e-mail: dbarashev@mail.ru, cathy@meta.math.spbu.ru

Abstract
XML is emerging as one of the dominant data formats for data processing on the Internet. In this paper we investigate the problem of representing XML in a relational DBMS and optimization of the relational DB scheme with respect to the given set of queries. We consider different schemes of the storage of XML data in the tables and try to find scheme that brings the best performance of the queries execution.

1. Introduction
XML tags create a logical structure in the plain text. This logical structure not only allows to create a hierarchy of the text parts but it also provides capability to use techniques more efficient than fulltext search to query text data. One of such techniques is to store XML data in the relational database [1].

There are several ways to store XML data in the database [1, 2]. Some of them are quite effective but require additional storage resources. Other are inexpensive from the storage viewpoint but are not effective. Most likely there is no relational representation of XML data that would be optimal in every case. In this paper we focused on the problem of optimization of the XML representation in RDB for the given set of queries that would be performed over the given data set.

2. Description of the problem
We assume that we have an XML data set and a set of queries written in Lorel query language [3] that will be run over this data set. The goal is to build a relational scheme for this data set that would be optimal with respect to some cost function. Queries can have some parameters that affect our choice of the relational representation. These parameters could be, for example, priority of the query or a presumable frequency of the query execution. We can also take into account a statistics of how frequently element tag is found in the document or an information about a regular part of the document which could be obtained from the document type definition (DTD).

3. Possible ways to solve the problem
3.1. Basic relational schemes for XML
It is known that XML document can be represented by a directed graph where elements play the role of vertices and parent-child relationships between elements act as edges. Each edge has a label which is the name of the element it leads to. There are several schemes of the relational representation of XML graph. In our work we use two simple well-known schemes.

In the first approach all XML elements are stored in one table. Each row in this table contains the name of the element, its internal identifier, identifier of the parent element and ordinal number of this element in the set of parent element’s children. This is nothing more than a representation of the directed labeled graph. Attributes are stored in another table with the similar structure.

The second way is to have a separate table for each distinct element and attribute name found in the data set. Here element name becomes a table name and columns store the identifiers of the element, its parent and ordinal number of this element.

*This work was partially supported by INTAS under grant OPEN-97-11309, Russian Foundation for Basic Research (grant 98-01-00436), and Urbansoft Ltd.
Both approaches have advantages and disadvantages. First approach effectively uses disk space but requires a lot of select operations (namely, for each label in the Lored [3] query path we need to perform selection in the big table). Second one allows to avoid select operations but uses disk space ineffectively when there is a few elements with the common name. Furthermore, both approaches require a lot of joins to recover instances of the paths written in the query.

3.2. Avoiding Joins: Access Path Relations

The most complicated and time consuming thing when XML data is stored in RDB is construction of Lored query paths instances. It is performed by a number of joins of the elements table(s). We can make it simpler and avoid joins using special structures called access path relations (APR). In APR we store instances of the particular path in the graph using OIDs. For each label in the path a column in APR is allocated and each instance of this path in the graph is a row in APR. We can also store values in this relation if they appear on the end of the path.

3.3. Optimization of the scheme

Optimization work begins from analysis of the data set and queries. If we have an information that some element tag is found frequently in the document then we can create a separate table for this tag. Similarly, if we know exactly the structure of some element (from it's DTD) we can create access path relations for this element. Furthermore, we can use native indices of the underlying RDBMS to increase performance.

Analysis of the data set provides a space for general optimization. We optimize the storage scheme as a whole not emphasizing its parts. However, it is possible that we will rarely query a certain part of the document. In this case we obviously don't need to optimize this part.

Analysis of the queries allows us to separate critical parts of the scheme. These parts are optimized by the methods given above while not critical parts are stored using one of the base storage schemes. Furthermore, we can use some novel index techniques for semistructured data [4]

The process of the optimal scheme generation is automated and passes through several steps. On the first step we determine and throw away certainly unacceptable schemes. Then we generate some scheme and SQL queries which correspond to the given Lored queries for this scheme. On the third step queries execution time is evaluated and is compared with the current best results (record). If the evaluation results obtained are better than record then scheme then results are kept as a new record and then we generate a next scheme and perform the same actions with it. This way we obtain either optimal scheme or scheme close to optimal if algorithm is euristic.

In the future we are going to create a cost model that will allow us to use in the optimal scheme generation process such methods of discrete optimization as method of branches and edges. We also implement a prototype system that will provide an interface to the relational database, automatically generate different schemes of the relational database and evaluate queries efficiency. Prototype consists of the Java-based client application and Oracle, Microsoft SQL or IBM DB2 database server.

References

[1] Daniela Florescu, Donald Kossmann
A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database
Rapport de Recherche No. 3680 INRIA, Rocquencourt, France, May 1999

Storing Semistructured Data in Relations
Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats, Jerusalem, Israel, 1999

[3] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, Janet L. Wiener
The Lored Query Language for Semistructured Data
Journal on Digital Libraries, volume 1 number 1, 1996