NEXD: NATIVE XML INTERFACE FOR A RELATIONAL DATABASE

Karel Piwko
Masters Degree Programme (2), FIT BUT
E-mail: xpiwko00@stud.fit.vutbr.cz

Supervised by: Petr Chmelař
E-mail: chmelarp@fit.vutbr.cz

ABSTRACT

XML has emerged as leading document format for data exchange, so there is a strong need to store and query information in these documents. However, most companies are still using an RDBMS solutions and some kind of XML to relations transformation with legacy data which must be queried as well. We propose a modified *Hybrid* algorithm to shred documents into relations and we allowed redundancy to make queries faster. Our goal was not to provide an academic solution, but fully working system supporting latest standards which will beat up native XML databases both by performance and vertical scalability.

1 INTRODUCTION

The XML language has emerged as the most commonly used language for data description and information exchange nowadays. It virtually replaced all proprietary solutions used before. Obviously, these amounts of data must be stored and we would like to query them.

Since RDBMS are most commonly used for managing middle size data (up to 10 GiB) within a company, and XML data are often mixed with the legacy ones, we have decided to store XML documents within a RDBMS. However, using REST interface and XQuery language had shown to be much more comfortable to an average user [3].

The main aim of this work is to create such system using existing or slightly modified mappings to beat the performance XML databases on one hand and to provide the same user as XML databases do on the other hand.

2 STORAGE OF XML DATA IN RELATIONAL DATABASES

When storing XML data in a relational database, we can basically divide available methods into three categories, *generic methods*, *schema-driven* methods and *user-defined/driven* methods. Readers interested in further details can follow [2].

We will focus on schema driven methods adapting them for RNG schema language. The methods create a relational database schema from the XML one, joining or splitting elements into relations depending on the occurrence in the document (both amount and position are relevant).

XML documents not conforming to the XML schema can be omitted with required transformation by user, but in our solution we prefer to store their not conforming part(s) in a special table without a big performance hurt.

2.1 MODIFIED HYBRID METHOD

The *Hybrid* method is considered to be the best fixed method [1]. This method decides whether to include child element in parent's table (*inlining*) or to create distinct table and establish a relationship by providing foreign keys. It identifies common elements, which spread among the document definition, storing them in one table. Moreover, it tries to solve recursion which occurs within general definitions by identifying strongly connected components of the schema graph representation. Elements are inlined if their in-degree is more then one but less then three, even if they are shared when reachable from a general node, except recursive descent.

Relations are created from a graph, which is basically a DOM tree of the RNG scheme. The graph is traversed to find out the root element of a scheme without backward references and nodes are processed according to their type (simple, complex) and position recursively. Encoding of nodes is preserved. Once the graph is traversed, metadata are stored in database tables `xcollection`, `xdocument`, `xtable` and `xtext`, further used to query and reconstruct the document.

3 RETRIEVAL OF XML DATA FROM THE PERSISTENCE LAYER

When retrieving data from a persistence storage, operations can be divided into two subsets: The first one, *extraction*, possibly combined with *selection*, to get a fragment of a stored document, so we have to be able to access its elements, attributes or values and then evaluate predicate conditions. The latter one, *querying*, further applies transformation, sorting and aggregation on a result of extraction, or multiple extractions from different documents. Obviously, when XML document is shredded into relations, *extraction* is the most time consuming operation. We allowed data redundancy, to speed it up significantly.

3.1 XPATH PROCESSING

We represent XML document by XPath Data Model (XDM), which is a tree graph where nodes are either atomic or sequence values. XDM is shared among XPath 2.0, XQuery 1.0 and XSLT processors, so it allows us to easily implement XQuery support once the XPath processor is done.

The basic XDM model types can be mapped directly to columns in a relational schema created by Hybrid method. Since SQL datatypes are more general, it could be useful to store information of XSD types during query evaluation. Our XPath parser is written in ANTLR, a Java based LL(*) parser generator framework.

During the query evaluation, we read the metadata and preserve evaluation context, which consists of actual XPath step, table and the intermediate result. NeXD descends from the root element and consecutively calls SQL statements without creating exhaustive join operations. The fragment can be retrieved both as a DOM tree or in its textual form.

The DOM tree is ideal structure for storing intermediate results, because it models the XDM sequence and it can be visitor-flattened, both required by XML:DB API. However, we have to track the identity of nodes with respect to data source to answer queries posed in XPath 2.0 language. Additionally, the DLN [4] indexes are added to model the position of elements with update/delete facilities still supported.

4 EVALUATION OF PERFORMANCE

For performance measuring, we focused on lookup and publish queries because they exercise the use cases expected for NeXD. The same XPath queries were executed on both NeXD and eXist 1.2.5, over a collection of approximately 1000 documents retrieved from [5], measured on Intel Celeron M 1.6 GHz with 2 GiB RAM. An extract of results is shown in table 1.

XPath query	Total results	NeXD [ms]	eXist [ms]
`/weather/head/locale`	1,000	120	146
`//wind`	11,000	145	547
`//part/wind/*`	40,000	141	2,156

Table 1: Sample of performed queries, times are in milliseconds

5 CONCLUSION

In this work we have shown that data-centric XML documents that conform to the RNG schema can be stored in a relational database using modified *Hybrid* algorithm with very satisfiable result. Because we allowed redundancy, we are able speed up queries and preserve processing instructions, comments and other XML related elements.

This work was partially supported by the BUT FIT grant FIT-S-10-2 and the research plan MSM0021630528.

REFERENCES

[1] Jayavel Shanmugasundaram et al. A general technique for querying XML documents using a relational database system. ACM SIGMOD Record, 30(3):20–26, 2001.

[2] Irena Mlýnková and Jaroslav Pokorný. XML in the world of (object-)relational database systems. In 13th International Conference on Information Systems Development Advances in Theory, Practice, and Education, 2005, p. 63–76.

[3] Karel Piwko. Nativní XML databáze. Bachelor's Thesis. FIT BUT in Brno, 2008.

[4] Timo Böhme and Erhard Rahm. Supporting efficient streaming and insertion of XML data in RDBMS, 2004, p. 70-81.

[5] National and Local Weather Forecast, Hurricane, Radar and Report. `http://www.weather.com/`, available in December, 2009.