XML Support in PostgreSQL *

© Nikolay Samokhvalov

Moscow Institute of Physics and Technology
nikolay@samokhvalov.com
Ph.D. Advisor S. D. Kuznetsov

Abstract

This paper presents the roadmap for the development of native XML type support in open source relational database system PostgreSQL [15]. Started five years ago as a "contrib" module ("contrib/xml2") by John Gray, implementing a set of basic XML-oriented features, it is now in the process of complete reworking and integration into the DBMS core, in accordance with SQL/XML standard [17, 4]. Initial XML type support will be included into the next release of PostgreSQL v.8.3.

We describe the current accomplishments illustrated by usage examples, overview new XLABEL contrib module developed to increase performance of XPath expressions evaluation using GiST-based structures, and discuss other aspects of XML support such as full-text indexes. Finally, we present a brief description of our plans for futher development of PostgreSQL XML type.

1 Introduction

Today XML becomes fullfledged, mature technology based upon the group of specifications including XML 1.1 [11], XPath 2.0 [12], XQuery 1.0 [13], XPath 2.0 and XQuery 1.0 Data Model [14]. It is a recognized standard for information representation and exchange on the Internet. On the other hand, relational database management systems (RDBMS) remain to be mainstream technology to store, process and work with data. Native XML database systems are rapidly developed, but it does not eliminate the needs in capabilities for working with XML data in relational databases. Since relational database systems are widely used it appears attractive to extend them with XML storing and manipulation facilities.

What are the reasons for keeping XML data in databases? In some cases it is necessary to store original XML documents without any changes of the format. Typical examples are legal documents, financial reports, and book sources in DocBook XML format. In other cases the data structures are very sophisticated and may change frequently, making too expensive to maintain database schema if only relational data model is used (good example for such data sets is metadata in astronomical heterogeneous data archives [9]).

In this paper we present an overview of the XML support in the upcoming version (8.3) of the open source RDBMS PostgreSQL [15] and describe some key implementation concepts like using Generalized Search Tree (GiST [5]) to increase the performance of queries to XML data. Also we provide usage examples of dealing with the new XML data type and describe the roadmap of the XML support in PostgreSQL.

The rest of this paper is organized as follows. After pointing to related work in section 2, we provide an overview of new capabilities for dealing with XML data in PostgreSQL version 8.3 and then describe which parts of the system have been changed to provide XML support, in section 3. In section 4 we present several usage examples of dealing with XML and relational data by means of XPath functions and standard SQL/XML routines. In section 5 we discuss performance issues and describe different types of XML indexes. Then we briefly overview current and future work in section 6 and, finally, conclude with a summary in section 7.

2 Related Work

Nowadays every major commercial database system provides more or less powerful XML support, allowing to store, manipulate, search and retrieve XML data. There are three basic approaches to implement such support. The first, LOB-based, usually allows fast insert and retrieval of entire XML documents but often demonstrates poor performance in tasks of searching and manipulation with fragments of documents. These problems can be partially solved by indexes of different types (including additional structures, redundantly representing entire documents). Nevertheless, this approach is *a priori* limited with characteristic features of relational physical storage.

The second approach is shredding XML to relational tables. Though it needs additional overhead during insertion time, this method promises high performance since queries can be transformed to plain SQL over relational data. However, high performance can be achieved only under certain circumstances. For example, one of drawbacks of the method is the fact that more complex XML schema is, more tables must be created in corresponding relational schema. High number of tables leads to at least two problems. First, even simple queries to XML

* This work is partially supported by Google Inc. (Google Summer of Code 2006).

Proceedings of the Spring Young Researcher's Colloquium on Database and Information Systems, Moscow, Russia, 2007

data might require very complex and expensive SQL queries to the corresponding relational data (in other words, these queries might involve a lot of joins to reconstruct original XML data; therefore, it might turn out so that simple retrieval of entire document is very expensive). Second, after relational schema is built and the data is inserted it might be unfeasible to introduce even simple changes to XML schema.

The third approach is the most promising one, but it needs affecting almost all the subsystems of DBMS so its implementation becomes very difficult. This approach is known as "native XML storage": XML data is stored in a special format (optimized for operations on trees) and the database manages both relational and XML native storage in an integrated manner. This allows fast processing of structural queries; XML value indexes and full text indexes help to increase performance of queries with value predicates. XML support in the new IBM's DB2 Universal Database [7] implements this approach.

Microsoft SQL Server 2005 stores XML data as byte sequences in BLOB columns. So called primary index can be defined to avoid parsing at the query time. It is based on the prefix numbering scheme known as ORDPATH [8]. In addition, secondary indexes of different types can be defined to increase query performance.

Nowadays open source relational database systems are lacking powerful XML support. For example, MySQL 5.1 has only a couple of simple functions, ExtractValue() and UpdateXML(), dealing with VARCHAR values. Weakness of open source software in this area could be explained by the fact that implementation of XML support belongs to the class of integration issues, where commercial development companies seem to be stronger. However, needs in capabilities of dealing with XML data in existing open source database systems are obvious. This motivates our work on implementation of the XML type support in one of the most popular and most advanced open source RDBMS, PostgreSQL [15]. Our approach to index XML data described in section 5 is similar to ORDPATHs in general, but uses more efficient numbering scheme to define special GiST-based data type.

Finally, Oracle 10g, Microsoft SQL Server 2005 and DB2 Universal Database provide advanced capabilities to deal with XML data. Some of these features (e.g., full text index for XML data, XML to the relational mapping with annotated schemes) are subject to our future work as described in section 6.

3 The Development of XML Support for PostgreSQL

Since 2001, PostgreSQL provides a set of basic XPath and XSLT functions for VARCHAR data type via contrib module "xml2" by John Gray. With release of the new version of the SQL standard (SQL/XML [10, 3]) it became clear that PostgreSQL lacks of support of a special data type to deal with XML data, XML type. On Spring 2006 work on the implementation of native XML type was started. First task was to create XML type at the logical level, taking VARCHAR/TEXT type as a basis, and then to integrate the reworked XPath and XSLT functions with that data type. One of the major requirements to the new data type was conformance to the SQL/XML standard. This allows to develop unified XML type implementation (and simplifies migrations from and to other database systems) and, moreover, helps to achieve high level of integration of XML and relational types to provide ability to work with both types of data in an integrated manner.

Before describing some aspects of the XML type implementation in PostgreSQL we briefly discuss a set of database features representing the concept of what is called "XML support".

3.1 What Is XML Support in RDBMS?

XML does not conform to the relational model. To enable operations on XML documents, database system has to be extended with special abilities. Usually, needs in XML support appear in the processes of integration of several systems, e.g. during development of B2B relationships between two or more companies. Therefore, the primary goal for XML support is implementation of data export and import facilities.

XML type defined in the new SQL/XML standard [10] is considered as integrated to the system of relational types. The standard defines the concepts of data models integration, where XQuery data model plays key role. Additionally, so called SQL/XML functions provide facilities to publish existing relational data in XML format. The same functions can be used for working with XML data inside the relational database – one can use them to create XML values from relational data.

Also, mapping from XML to relational tables might be useful under certain circumstances. In some cases there is no need in storing entire XML documents. In others, mapping to relations might be required to make already developed system work.

3.2 Integrating XML Support to PostgreSQL System

During our work, most SQL/XML publishing functions have been implemented, that required to introduce multiple changes to PostgreSQL's internals (such as parser and executor) due to several reasons (e.g., static keyword `NAME` in the `XMLELEMENT` function). The system parts that have been changed to support XML type and SQL/XML functions are marked with the asterisk ("*") mark in Figure 1. PostgreSQL internals are described in [6] and [16].

XML type based on VARCHAR was defined. The value being inserted is parsed with the XML parser and checked for well-formedness. PostgreSQL XML type allows not only entire XML documents but also fragments of them, providing subset of possible values defined in the XQuery Data Model (XDM) [14]. Special technique for variable length values (known as TOAST – The Oversized-Attribute Storage Technique [16]) provides efficient storage method for XML data due to its flexibility and high compression rate. The maximum size of the XML value is 1 GB.

Besides SQL/XML publishing functions, PostgreSQL's XML manipulation capabilities include XPath expression evaluation and XSLT support. Several examples demonstrating how to manipulate XML values are

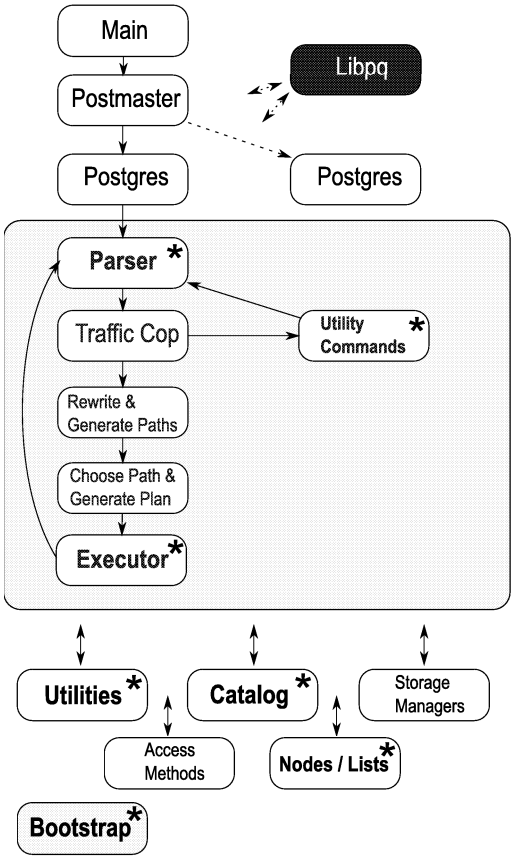

Figure 1: PostgreSQL internals. The system parts that have been changed to support XML are marked with the asterisk mark.

provided in the following section.

4 Working with XML Type

Being a part of SQL:200x standard, SQL/XML paper [10] provides comprehensive description of the XML type integrated into relational type system. PostgreSQL's XML support has been developed in accordance with the statements from the standard.

4.1 SQL/XML

PostgreSQL 8.3 supports standard SQL/XML routines to work with date of XML type (`XMLPARSE`, `XMLSERIALIZE`) as long as publishing functions (`XMLELEMENT` with `XMLATTRIBUTES`, `XMLFOREST`, `XMLAGG`, `XMLCONCAT`, `XMLCOMMENT`, `XMLPI`, `XMLROOT`). SQL/XML publishing functions transform relational data into XML format, producing values of XML type. Then, these values can be published (one of examples is simple creation of RSS channel) or used as other values of XML type (e.g., be stored in a table with XML column). This integration is possible because XML type in PostgreSQL partially implements XQuery data model [13].

The following example demonstrates how an RSS channel representing PostgreSQL database activity can be easily created, just with one `SELECT` statement:

```
SELECT
  XMLROOT(
    XMLCONCAT(
      XMLCOMMENT('Database activity'),
      XMLELEMENT(
        NAME "rss",
        XMLATTRIBUTES(
          '2.0' AS "version",
'http://purl.org/dc/elements/1.1/'
            AS "xmlns:dc"
        ),
        XMLELEMENT(
          NAME "channel",
          XMLFOREST(
            'Sample Database Activity'
              AS "title",
            'Generated by SQL/XML funcs'
               AS "description",
            'http://example.com'
              AS "link"
          ),
          XMLELEMENT(
            NAME "item",
            XMLAGG(
              XMLFOREST(
                usename || ' at ' || datname ||
                '(procpid:' || procpid || ')'
                  AS "title",
                'http://example.com'
                  AS "link",
                query_start
                  AS "pubDate",
                current_query
                  AS "description"
              )
            )
          )
        )
      )
    ),
    VERSION '1.0'
  )
FROM
   pg_stat_activity
GROUP BY
   procpid;
```

4.2 XPath

Following examples are based upon this simple table:

```
xmltest=# CREATE TABLE table1(
  id INTEGER PRIMARY KEY,
  created TIMESTAMP NOT NULL
    DEFAULT CURRENT_TIMESTAMP,
  xdata XML
);
CREATE TABLE

xmltest=# INSERT INTO
  table1(id, xdata)
  VALUES(
    1,
    '<dept
```

```
xmlns:smpl="http://example.com"
smpl:did="DPT011-IT">
       <name>IT</name>
       <persons>
         <person smpl:pid="111">
           <name>John Smith</name>
           <age>24</age>
         </person>
         <person smpl:pid="112">
           <name>Michael Black</name>
           <age>28</age>
         </person>
       </persons>
     </dept>'
   );
INSERT 0 1
```

PostgreSQL 8.3 provides `xpath_array` function, which allows XPath expressions evaluation with namespaces bindings. The type of returned data is XML array (`xml[]`). Here is a primitive example:

```
xmltest=#
  SELECT xpath_array(
    '//person/name/text()',
    xdata
    )
  FROM table1;
            xpath_array
-----------------------------------
 {"John Smith","Michael Black"}
(1 row)
```

The same function is extended with capabilities to define namespace mappings for XPath expression:

```
xmltest=#
    SELECT xpath_array(
      '//person/@smpl:pid',
      xdata,
      ARRAY[ARRAY['smpl'],
        ARRAY['http://example.com']]
    )
  FROM table1;
 xpath_array
-------------
 {111,112}
(1 row)
```

5 XML Type and Performance

Since PostgreSQL stores XML in its text representation, selecting entire documents is extremely fast, while search and work with document parts are not. To increase the performance we introduce several types of indexes and additional structures.

First, the most simple type of indexes is functional B-tree index defined over XPath functions:

```
CREATE INDEX i_table1_xdata
  ON table1
  USING btree(
    xpath_array(xdata, '//person/@name')
  );
```

It allows to avoid XML parsing and XPath expression evaluation at query time. This is the best choice for cases when an application produces large amount of similar queries with `xpath_array` function call, where XPath expression itself is static.

5.1 XLABEL: Extending XML Storage With GiST

The Generalized Search Tree (GiST) [5] is an extensible data structure, which allows to develop indexes over any kind of data, supporting any lookup over that data.

The GiST is a balanced tree structure, containing {`key, pointer`} pairs, where keys are members of a user-defined class. The rules for tree balancing are defined by the developer, who is responsible for the type creation. There are several structures implemented with GiST support: B-tree, R-tree (for indexing spatial data), RD-tree (indexes for sets of items; used, for example, in the PostgreSQL full text engine), etc.

To increase the performance of XPath expressions evaluation we have developed special module, XLABEL. The general concept is similar to ORDPATHs [8] but we use more efficient prefix numbering scheme, SLS [1], which serves as a basis for GiST-based XLABEL data type. Being modifications-friendly (there is no need in key recalculation for any document node that has not been affected by a partial document modification), this scheme provides more compact way to store keys for nodes of the XML document.

The basic idea behind the XLABEL module is as follows. LOB-based XML type has great performance on tasks where retrieval of entire documents is needed. But evaluation of arbitrary XPath query needs XML processing at query time, dramatically decreasing the performance. To allow fast XPath expression evaluation, the XLABEL module extends the XML storage with additional tables, known as `xnames` and `xdata`. The first one is the database-wide table where XML element and attribute names for all XML values in the database are stored. It allows to enumerate all element names to save space. A table of the second type is created for every XML column, which is registered in XLABEL module (there might be many such tables in the database).

Table 1 and 2 demonstrate examples of, respectively, `xnames` table and `xdata` table for `table1` table defined in section 4.2. For simplicity reasons namespaces information is not shown. Also, numbering scheme keys represented in the table corresponds to the simplest prefix scheme. [1] provides comprehensive description of SLS numbering scheme, which is essentially the next step in development of prefix schemes.

Table 1: `xnames`

xname_id	**xname_name**
1	person
2	dept
3	name
4	did
5	persons
...	...

Table 2: `table1_xdata`

tid	xlabel	node_type	xname_id	value
1	a	1 (elem.)	2	NULL
1	a.b	2 (attr.)	4	DPT011-IT
1	a.c	1 (elem.)	3	NULL
1	a.c.a	NULL	NULL	IT
...	...	...	...	...
1	a.d.a.b	1 (elem.)	3	NULL
1	a.d.a.b.a	NULL	NULL	John Smith
...	...	...	...	...

Using shredding transformation the XLABEL module fills the `xdata` table with rows each representing one node in the original XML document. Hierarchical information is encoded by means of the numbering scheme. Then, GiST index is created over the column of `xlabel` type. Additional indexes (including composite indexes) might be created to increase query performance.

An XPath expression is translated by the XLABEL module to pure SQL query to the main table which contains the XML type column and additional tables described above. This SQL statement uses special GiST-powered operators over `xlabel` column. There are various approaches for further performance increase, which are beyond the scope of this paper.

5.2 XML Full-Text Indexes

Full-text search is a technique widely used in document and content-centric XML applications. PostgreSQL existing full-text search engine, tsearch2 [2] (available as a separate GiST-based module before version 8.3 and integrated to the core in 8.3), provides a powerful set of capabilities such as advanced parsing techniques based on ispell dictionaries, stemming methods, stopword lists, synonym dictionaries and thesaurus support.

PostgreSQL full-text search engine can be easily adopted to the needs of XML support. For example, using XPath expressions it is possible to point out, what parts of documents must be indexed. Moreover, it is possible to define different weights to different parts of the document and then use the weights in ordering (ranking) of result set.

6 Future Work

XML support in relational DBMS is a huge task. There are completely different directions of work on the support, including XQuery data model [14] and XML Schema types support, performance improvements using advanced index types and data structures, tools and applications for work with XML data in RDBMS, XML type support in API of various programming languages (such as Java, Perl, PHP) and procedural languages (such as PL/Perl, PL/Python).

Our nearest plans include following items:

- XLABEL module enhancements;
- better integration of full-text engine with XML type;
- support of XML values to relations decomposition based on annotated schemas (some reasons to implement this technique in the database with already implemented XML type were given in section 3.1);
- XMLQUERY (XQuery integrated with SQL, per SQL/XML standard) support.

One of prioritized direction is development of GiST-based indexes (now as a part of XLABEL module) suitable for use in other database systems. It is possible, because there exist several separate implementations of GiST engine, and due to the fact that the code of PostgreSQL and adjacent projects is shipped under liberal BSD license.

7 Summary

XML support in open source relational database systems only starts developing. PostgreSQL is a good platform to develop complicated data structures and powerful solutions, what makes it attractive to develop support of work with XML data.

PostgreSQL version 8.3 was enhanced with integrated XML type, what includes implementation of standard SQL/XML functions and support of XPath expressions. To allow efficient queries containing XPath expressions we have developed XLABEL module which provides an advanced method to work with XML data. This module uses GiST-based structures to index the data in XML documents and allows XPath expressions be transformed to plain SQL with special operator over `xlabel` type.

Additionally, we presented several directions of our work on XML support, including full-text indexes, XQuery support and decomposition with annotated schemas.

Aknowledgements

The author recognizes developers participated in XML type support imlementation: P. Eisentraut, P. Stehule and J. Gray & T. Dyson; and thanks I. Chilingarian, D. Morozov and I. Zolotukhin for valuable discussions on PostgreSQL XML type support.

References

[1] N. A. Aznauryan, S. D. Kuznetsov, L. G. Novak, and M. N. Grinev. SLS: A numbering scheme for large XML documents. *Program. Comput. Softw.*, 32(1):8–18, 2006.

[2] O. Bartunov and T. Sigaev. Full-Text Search in PostgreSQL. A Gentle Introduction. http://mira.sai.msu.su/∼megera/pgsql/ftsdoc/.

[3] A. Eisenberg and J. Melton. Advancements in SQL/XML. *SIGMOD Rec.*, 33(3):79–86, 2004.

[4] P. Eisentraut. XML Support in PostgreSQL. Specifications and Development (Talk at PostgreSQL Anniversary Summit), 2006. http://developer.postgresql.org/∼petere/xml.pdf.

[5] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer. Generalized Search Trees for Database Systems. In *VLDB*, pages 562–573, Trondheim, Norway, 1995.

[6] B. Momjian. PostgreSQL Internals Through Pictures, 2005. momjian.us/main/writings/pgsql/internalpics.pdf.

[7] M. Nicola and B. van der Linden. Native XML Support in DB2 Universal Database. In *VLDB*, pages 1164–1174, Trondheim, Norway, 2005.

[8] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: insert-friendly XML node labels. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 903–908, New York, NY, USA, 2004. ACM Press.

[9] I. Zolotukhin, N. Samokhvalov, F. Bonnarel, and I. Chilingarian. Comprehensive Metadata Query Interface for Heterogeneous Data Archives Based on Open Source PostgreSQL ORDBMS. In *Astronomical Data Analysis Software & Systems XVI*, page P3.23, Tucson, Arizona, USA, 2006.

[10] SQL:2006, 2006, Part 14: XML-Related Specifications (SQL/XML), ISO/IEC JTC 1/SC 32, CD 9075-14:200x(E). International Standard, ISO, ANSI.

[11] Extensible Markup Language (XML) 1.1. W3C Recommendation REC-xml11-20060816, 2006. http://www.w3.org/TR/2006/REC-xml11-20060816/.

[12] XML Path Language (XPath) 2.0. W3C Recommendation REC-xpath20-20070123, 2007. http://www.w3.org/TR/2007/REC-xpath20-20070123/.

[13] XQuery 1.0: An XML Query Language. W3C Recommendation REC-xquery-20070123, 2007. http://www.w3.org/TR/2007/REC-xquery-20070123/.

[14] XQuery 1.0 and XPath 2.0 Data Model (XDM). W3C Recommendation REC-xpath-datamodel-20070123, 2007. http://www.w3.org/TR/2007/REC-xpath-datamodel-20070123/.

[15] PostgreSQL Database Management System. http://postgresql.org.

[16] *PostgreSQL Manuals. PostgreSQL 8.2 Documentation.* http://postgresql.org/docs/8.2/static/.

[17] XML Support in PostgreSQL (Summer of Code Project), 2006. http://chernowiki.ru/Dev/Pgxmltype.