Chapter 1
XML Native Storage and Query Processing

Ning Zhang
Facebook, USA

M. Tamer Özsu
University of Waterloo, Canada

ABSTRACT

As XML has evolved as a data model for semi-structured data and the de facto standard for data exchange (e.g., Atom, RSS, and XBRL), XML data management has been the subject of extensive research and development in both academia and industry. Among the XML data management issues, storage and query processing are the most critical ones with respect to system performance. Different storage schemes have their own pros and cons. Some storage schemes are more amenable to fast navigation, and some schemes perform better in fragment extraction and document reconstruction. Therefore, based on their own requirements, different systems adopt different storage schemes to tradeoff one set of features over the others. In this chapter, the authors review different native storage formats and query processing techniques that have been developed in both academia and industry. Various XML indexing techniques are also presented since they can be treated as specialized storage and query processing tools.

INTRODUCTION

As XML has evolved as a data model for semi-structured data and the *de facto* standard for data exchange, it is widely adopted as the foundation of many data sharing protocols. For example, XBRL and FIXML defines the XML schemas that are used to describe business and financial information; Atom and RSS are simple yet popular XML formats for publishing Weblogs; and customized XML formats are used by more and more system log files. When the sheer volume of XML data increases, storing all these data in the file system is not a viable solution. Furthermore, users often want to query over large volumes of XML data. A customized and non-optimized query processing system would quickly reach its limits. A more scalable and sustainable solution is to load the XML data into a database system that is specifically designed for storing and updating large volumes

DOI: 10.4018/978-1-61520-727-5.ch001

of data, efficient query processing, and highly concurrent access patterns. In this chapter, we shall introduce some of the database techniques for managing XML data.

There are basically three approaches to storing XML documents in a DBMS: (1) the LOB approach that stores the original XML documents as-is in a LOB (large object) column (Krishnaprasad, Liu, Manikutty, Warner & Arora, 2005; Pal, Cseri, Seeliger, Rys, Schaller, Yu, Tomic, Baras, Berg, Churin & Kogan, 2005), (2) the extended relational approach that shreds XML documents into object-relational (OR) tables and columns (Zhang, Naughton, DeWitt, Luo & Lohman, 2001; Boncz, Grust, van Keulen, Manegold, Rittinger & Teubner, 2006), and (3) the native approach that uses a tree-structured data model, and introduces operators that are optimized for tree navigation, insertion, deletion and update (Fiebig, Helmer, Kanne, Mildenberger, Moerkotte, Schiele, & Westmann, 2002; Nicola, & Van der Linden, 2005; Zhang, Kacholia, & Özsu, 2004). Each approach has its own advantages and disadvantages. For example, the LOB approach is very similar to storing the XML documents in a file system, in that there is minimum transformation from the original format to the storage format. It is the simplest one to implement and support. It provides byte-level fidelity (e.g., it preserves extra white spaces that may be ignored by the OR and the native formats) that could be needed for some digital signature schemes. The LOB approach is also efficient for inserting or extracting the whole documents to or from the database. However it is slow in processing queries due to unavoidable XML parsing at query execution time.

In the extended relational approach, XML documents are converted to object-relational tables, which are stored in relational databases or in object repositories. This approach can be further divided into two categories based on whether or not the XML-to-relational mapping relies on XML Schema. The OR storage format, if designed and mapped correctly, could perform very well in query processing, thanks to many years of research and development in object-relational database systems. However, insertion, fragment extraction, structural update, and document reconstruction require considerable processing in this approach. For schema-based OR storage, applications need to have a well-structured, rigid XML schema whose relational mapping is tuned by a DBA in order to take advantage of this storage model. Loosely structured schemas could lead to unmanageable number of tables and joins. Also, applications requiring schema flexibility and schema evolution are limited by those offered by relational tables and columns. The result is that applications encounter a large gap: if they cannot map well to an object-relational way of life due to tradeoffs mentioned above, they suffer a big drop in performance or capabilities.

Due to these shortcomings of the two approaches, much research has been focusing on the *native* XML storage formats. There is not, and should not be, a single native format for storing XML documents. Native XML storage techniques treat XML trees as first class citizens and develop special purpose storage schemes without relying on the existence of an underlying database system. Since it is designed specifically for XML data model, native XML storage usually provides well-balanced tradeoffs among many criteria. Some storage formats may be designed to focus on one set of criteria, while other formats may emphasize another set. For example, some storage schemes are more amenable to fast navigation, and some schemes perform better in fragment extraction and document reconstruction. Therefore, based on their own requirements, different applications adopt different storage schemes to trade off one set of features over another.

The following are examples of some of the important criteria that many real-world applications consider.