Chapter 11
On Efficient Evaluation of XML Queries

Sherif Sakr
University of New South Wales, Australia

ABSTRACT
Recently, the use of XML continues to grow in popularity, large repositories of XML documents are going to emerge, and users are likely to pose increasingly more complex queries on these data sets. In 2001 XQuery is decided by the World Wide Web Consortium (W3C) as the standard XML query language. In this article, we describe the design and implementation of an efficient and scalable purely relational XQuery processor which translates expressions of the XQuery language into their equivalent SQL evaluation scripts. The experiments of this article demonstrated the efficiency and scalability of our purely relational approach in comparison to the native XML/XQuery functionality supported by conventional RDBMSs and has shown that our purely relational approach for implementing XQuery processor deserves to be pursued further.

INTRODUCTION
The eXtensible Markup Language (XML) (Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 2006) has been introduced by the end of the 1990’s in order to create a standard data-format for the World Wide Web which can be easily handled by computers as well as by humans. In recent years, XML has found practical application in numerous domains including data interchange, streaming data and data storage. The semi structured nature of XML allows data to be represented in a considerably more flexible nature than in the traditional relational paradigm. However, the tree-based data model underlying XML poses many challenges especially with regard to the problem of performing efficient query evaluations.

DOI: 10.4018/978-1-60960-521-6.ch011
As XML continues to grow in popularity, large repositories of XML documents are going to emerge, and users are likely to pose increasingly more complex queries on these data sets. Consequently, there is a great demand for efficient XML data management systems for managing complex queries over large volumes of the XML data. In 2001 XQuery is decided by the World Wide Web Consortium (W3C) as the standard XML query language (Boag et al., 2006).

XQuery is based on a hierarchical and ordered document model which supports a wide variety of constructs and use cases. The language addresses a wide range of requirements, thus incorporating a rich set of features.

The work of this article was developed within the Pathfinder project (Pathfinder, 2003). The aim of the Pathfinder project is to implement XQuery as a query language that can be used to query XML data stored on relational database systems. The architecture of Pathfinder is designed in a front-end/back-end fashion. Pathfinder receives an XQuery expression, which is parsed, normalized, and translated into XQuery Core. The Core expression is then simplified, type checked optimized, and translated into an intermediate algebraic plan. Initially, Pathfinder used the MonetDB main memory RDBMS as its target back-end. In this development branch (Boncz et al., 2006), the Pathfinder intermediate algebraic plan is translated into MIL (Monet Interpreter Language) code (Boncz & Kersten, 1999) which is then executed by the kernel of MonetDB. The MIL code generated by the Pathfinder compiler relies on some extensions added to the MonetDB back-end such as the staircase join algorithm (Grust, Keulen, & Teubner, 2003) which is designed as an efficient algorithm to evaluate XPath expressions. Although the approach of Pathfinder/MonetDB XQuery processor has been shown to be highly efficient and scalable, it is very tightly bound to the Monet DBMS and thus cannot be used with any other relational back end. Another disadvantage of using MonetDB is that it requires huge main memory sizes to store large XML documents. The limitation of this approach was the main motivation behind our purely relational approach for implementing XQuery processor described in this article.

In this article, we describe the design and implementation of an efficient and scalable purely relational XQuery processor which translates expressions of the XQuery language into their equivalent SQL evaluation scripts. The proposed XQuery processor is enhanced with an accurate algebraic-based cost model (Sakr, 2007) which facilitates the processor’s ability to generate enhanced cardinality aware SQL translation scripts. Figure 1 illustrates the different alternative back-ends for the Pathfinder XQuery compiler. In particular, the main contribution of the work of this article is that it describes the design and implementation of an efficient and scalable purely relational XQuery processor. The proposed relational XQuery processor stores source XML documents in a relational repository using a tree aware relational encoding scheme and translates the XQuery expressions into SQL evaluation scripts.