Querying XML data: Does One Query Language Fit All?

V. Ramesh, Arijit Sengupta and Bryan Reinicke
venkat@indiana.edu , asengupt@indiana.edu, breinick@indiana.edu
Kelley School of Business, Indiana University, Bloomington, IN 47405

Abstract

In this paper, we describe the characteristics of two different query languages designed to query XML data: DSQL, a declarative SQL like language and XQuery, a procedural language that is fast becoming the defacto language for XML querying. We then describe the design of an experiment aimed at comparing the accuracy and efficiency of the query formulation process when using the two languages. The results from our study should provide some answers to the important question of whether the W3C is justified in pushing a procedural query language like XQuery as the defacto standard for querying XML data. The answer to this question is clearly of importance to both the academic and practitioner communities.

1.0 Introduction

XML is fast becoming the language of choice for exchanging and publishing electronic data on the web. With the increase in the popularity of XML the need for a standard way of retrieving information from such documents is also becoming critical. To address this issue, the World Wide Web Consortium (W3C – http://www.w3.org/) started a working group and tasked it with developing a query language for XML. Before this task group was put in place, approaches to querying XML were taking one of two directions: (i) pattern-based languages based on the tree structure of XML documents such as XPath (Clark and Derose 1999) and XQL (Robie, Lapp and Schach 1998), and (ii) a more logic-oriented approach with conditions and output specifications such as XML-QL (Deutsch, Fernandez, Florescu, Levy and Suciu 1998). Although there are some attempts towards including XML querying support in SQL, including an effort from the International Standards Organization (ISO) (Melton 2001), a common decision among query language designers seems to be to create a completely new language for the purpose of querying XML data. This is apparent in the currently proposed language from the W3C, called XQuery (Chamberlin, Clark, Florescu, Robie, Simeon and Stefanescu 2001). With W3C's approval and increasing support from multiple software vendors, it seems likely that XQuery will become the defacto language for querying XML data. As we will show in this paper, XQuery is inherently a procedural language, and as such may not be suitable for novice users especially those who want to write ad hoc queries.

XML schemas do possess characteristics that might make it possible to write queries using a declarative language. Although XML documents have a complex hierarchical structure, the strong presence of meta-data in XML documents, makes it fairly intuitive to write declarative queries based purely on logical combinations of the properties of the intended results. Document SQL (DSQL; Sengupta and Dalkilic 2002) is an example of a declarative query language (very similar to SQL) that can be used to query XML documents. Declarative query languages such as DSQL, in which the primary focus is on the properties of the result, rather than the process of extracting the result itself, are very suitable for structured data, because they allow the possibility of allowing the system to optimize the queries instead of relying on the users' capabilities for writing an efficient query. In addition, users can take advantage of their existing SQL knowledge when writing these queries. Given all this, would a language like DSQL make it possible for users to write more accurate queries than XQuery? Would it improve the efficiency of query writing? These are some of the questions we hope to answer through the study described in this paper.

2.0 Background: Querying XML documents

The concept of "querying" is fairly new in the domain of documents. The primary means for retrieving data from documents has been in the form of information retrieval through

"searching", in which the main mechanism is based on using boolean combinations of keywords and ranking of documents based on the extent of match. With the standardization of SGML in 1986 and the introduction of marked-up documents, especially on the web, the concept of querying in which searches utilized meta-data as well as data became more prominent. Thus, it is not surprising that with the introduction of XML the need for a query language that could utilize the extensive meta-data available in XML schemas was clearly felt. As we stated earlier, many different approaches to querying XML data have been identified. Below, we will focus on the characteristics of the current defacto standard query language, XQuery.

2.1 XQuery

XQuery is a new query language proposed by W3C as a working draft. XQuery is "designed to be a small, easily implementable language in which queries are concise and easily understood. XQuery is a functional language which allows various kinds of expressions to be nested with full generality. It is also a strongly-typed language in which the operands of various expressions, operators, and functions must conform to designated types." (Chamberlin et al. 2001).

The main building block of XQuery is an expression. Although the full language includes a few different types of expressions, the primary building block of an XQuery query is the FLWR (For – Let – Where – Return) expression. The FOR clause describes the source of the data that is to be queried. The LET clause defines query variables that commonly refer to portions of documents specified in the FOR clause. The WHERE clause describes conditions that the resulting data needs to satisfy, and the RETURN clause shows a template of the output that is to be generated from the FOR clause. To reduce the amount of discussion needed, we show a representative example below. Interested readers are referred to (Chamberlin et al, 2001) for further details.

As a moderately simple example, consider a query from the use case XMP (W3C 2002) – "for each publisher, find the number of distinct books published by them". Although seemingly fairly straight-forward, because of the structure of the data, the query is not simple to write. In this example, the publisher information is underneath the structure for a book requiring that this query be restructured. In XQuery, this is achieved by initially retrieving all the publishers in an outer query, and retrieving all books by that publisher in an inner query and nesting them (see Table 1).

Readers might observe that the query is fairly similar to a conceptual computer program written to perform the above task, in which we can iterate through the publishers in a loop, and for each publisher, each of the books can be checked so as to only retrieve books that are published by that publisher. XQuery, in fact, includes many other features reminiscent of programming languages, such as if-then-else statements, functions including recursion, strong typechecking and other similar features.

2.2 DSQL: An SQL for Querying XML documents

Oddly, a consensus in the research world seems to be that XML needs a completely new and different query language from what we are used to in the database domain. This fact is somewhat puzzling given the popularity of SQL as a query language for retrieving data from relational databases and the fact representing data from relational databases is one of the key uses for XML schemas. SQL has been a standard for sixteen years, and has been in existence for over twenty five years. Multiple research projects have compared SQL with other (procedural) query languages and shown results both in favor and against SQL (see Welty 1990 for a summary). It is however, worth noting that after 25 years, almost none of the "other languages" used in the comparisons are currently in use. Given the popularity of SQL, we developed DSQL, a language for XML (Sengupta and Dalkilic 2002) documents that has the same "flavor" as SQL. A key characteristic of the language is that any query, except for recursive queries, written in XQuery can in fact be written in DSQL. Another important property of DSQL is that it is syntactically

equivalent to SQL for flat structures, which effectively means that every valid SQL query is also a valid DSQL query.

DSQL is based on an algebra (Document Algebra or DA) and a calculus (Document Calculus or DC), along the same lines as the relational algebra (RA) and relational calculus. DA has all the operations of RA, with somewhat different semantics to fit the domain of hierarchically structured documents. DA has some additional operations to create new structures and to traverse hierarchical structures. The DSQL (Document SQL) language has the same SELECT-FROM-WHERE-GROUP BY-HAVING-ORDER BY structure as SQL. In fact, all of the SQL operations have been incorporated in the language in an intuitive way. DSQL uses the concept of simple path expressions (SPEs) in which children are identified using a single period (.) and descendants are identified using a double period (..). Details on the language, path expression constructs and sample queries can be obtained from (Sengupta and Dalkilic 2002). To illustrate with an example, however, consider the DSQL (Table 1) for the same query that was described in the previous subsection. Notice that in this query, the structure is initially flattened (by only selecting the publisher and title for every book) but the GROUP BY operation immediately groups the publishers together, in turn nesting the titles by the same publisher associated with the publisher.

3.0 Theoretical Background and Research Questions

The discussion above illustrates that DSQL represents a viable declarative alternative to the more procedural XQuery language. Hence, the primary question of interest in our study was to understand the circumstances under which it would be more appropriate to use the two languages outlined above. To the best of our knowledge, a study comparing a declarative versus a procedural query language has not been reported in the literature since the early 80's. Welty and Stemple (1981) compared the performance of users when using SQL with a procedural language, TABLET. They found that for simple queries there were no differences between the two languages. They also found that TABLET performed better for complex queries. This suggests that the degree of query complexity is an important factor to be considered in any study of DSQL vs. XQuery.

Another important consideration in a study comparing the two query languages is the "fit" between the characteristics of the representation and the characteristics of the language. XML schemas can be used to represent both flat and tree-like structures. Flat structures are akin to relational database schemas whereas tree structures are more hierarchical in nature possibly containing several levels of nesting. The theory of cognitive fit (Vessey 1991) would suggest that in the case of a flat XML schema an SQL like language would perform better than its procedural counterpart since users will be able to use the same cognitive processes that allow them write successful queries against relational schemas. At the same time, when confronted with a hierarchical structure the procedural constructs in XQuery should prove of benefit since the nature of the representation requires the use of cognitive processes that are similar to that used in writing procedural programs. Given this background we pose the following research questions:

RQ1: Will users be more accurate and efficient when writing queries for **flat XML** schemas using DSQL rather than XQuery?

RQ2: Will users be more accurate and efficient when writing queries for **tree XML** schemas using XQuery rather than DSQL?

4.0 Experimental Design

We conducted an experiment to answer these research questions. Subjects were required to formulate queries in one of the two languages described above. We used a 2 X 2 design. The type of language (DSQL vs. XQuery) was used as a between-subjects factor while the type of schema (flat vs. tree) was a within-subjects factor.

Experimental Task: We adapted two use cases from the W3C Use Case document (W3C 2002) since these documents were created to serve as exemplars of the types of queries that should be supported by an XML query language. Specifically, we used the use case "R" and "Tree" for our

purposes. The use case "R" contains a schema that is a representation of a (multi-table) relational database. While the original use case "R" presents data in tabular format, to avoid any confounds caused by presentation format, we chose to present our data to users in the form a textual XML schema. The use case "Tree" contains a schema for a book wherein it is possible to use elements in a nested fashion, e.g., a section in a book can contain other sections. As with the previous use case, the data was presented in the form of a textual XML schema. We then created a total of 10 queries (5 for each schema) with differing degrees of complexity (for each schema). Due to space limitations we are unable to show the example queries here.

Subjects: Subjects were students from a senior level MIS class on Object-Oriented Design and Programming at a major midwestern state university. They had prior exposure in their curriculum to both database and programming concepts. In particular, they had been exposed to SQL in one course and the C programming language in another. Given their background we believe that these students represented an appropriate surrogate for our target population, moderately knowledgeable users, i.e., those that are likely to write ad hoc queries on XML schemas. Students were randomly assigned to the DSQL or XQuery condition. Students participating in the experiment received extra course credit. However, we still found that a few participating students, for whatever reasons, did not write any usable queries. In the end, useful data was collected from 52 students. Of these, 23 students were in the DSQL condition and 29 students were in the XQuery condition.

Procedure: Students in each group received instruction (through a 35 minute lecture) on how to write queries using DSQL or XQuery. An example scenario (same one for both conditions) different from the ones used in the experimental materials was used for training purposes. The queries presented in the training materials incorporated examples of all the constructs, e.g., GROUP BY, ORDER BY etc., that students would need while performing their assigned task. After the training period was complete students were given the use case descriptions (on paper). Students were allowed to keep the training materials as reference while writing their queries. The use case context was assigned to the students in random order, i.e., they were presented with the 5 queries relating to one of the two use cases (in random order) followed by the 5 queries relating to the other. The order of queries within a use case was however kept constant (progressing from easy to difficult). All of the student queries were entered directly into a PC. Each query was presented to the students through a web based interface. Students were then instructed to enter their query in a text box and to press the submit button when satisfied with their answer. The system recorded the time it took to a student to answer each query.

Dependent Variables: The variables of interest in our study were accuracy and efficiency. Efficiency was measured using the time taken to write each query. To measure accuracy, we adapted the procedure and coding scheme from De, Sinha and Vessey (2001). For each query we identified errors in the following categories: range selection, attribute selection, condition specification, and result display specification. In addition, we also evaluated an overall measure of quality of the query which is a subjective assessment of the extent to which the participants' queries deviates from "ideal" solution for the query in the respective query language.

5.0 Current Status

Two coders are currently in the process of independently coding the data based on the coding scheme described above. As is customary in studies of this nature, we intended to evaluate both raw agreement and the kappa statistic for inter-rater reliability. We should be ready to present the results from our analysis at WITS. The results from our study should provide some answers to the important question of whether the W3C is justified in pushing a procedural query language like XQuery as the defacto standard for querying XML data. The answer to this question is clearly of importance to both the academic and practitioner communities. In addition, by examining the types of errors committed by users when using DSQL and XQuery, we should be able to make suggestions regarding areas where XQuery/DSQL seems to be problematic. Such results can help guide efforts seeking to enhance both XQuery and DSQL languages.

References

D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Simeon and M. Stefanescu, "XQuery 1.0: An XML Query Language," *W3C Working draft*, available from http://www.w3.org/TR/xquery, June 2001.

J. Clark and S. DeRose, "XML Path Language XPath Version 1.0," *W3C working draft*, http://www.w3.org/TR/xpath, November 1999.

P. De, A. Sinha and I. Vessey, "An Empirical Investigation of Factors Influencing Object-Oriented Database Querying," *Information Technology and Management*, 2, 2001, 71-93.

A. Deutsch, M. Fernandez, D. Florescu, A. Levy and D. Suciu, "XML-QL: A Query Language for XML," in online *Proceedings of W3C XML Query Language consortium*, http://www.w3.org/TR/1998/Note-xml-ql-19980819/, September 1998

J. Melton "ISO-ANSI Working Draft on XML-Related Specifications (SQL/XML)," *ISO Document no. WG3-YYJ-012 H2-2001-149*, June, 2001.

J. Robie, J. Lapp and D. Schach, "XML Query Language (XQL)", *In online Proceedings of XML Query Language Consortium*, http:// www.w3.org/TandS/QL/QL98/pp/xql.html, September 1998.

A. Sengupta and M. Dalkilic, "DSQL – an SQL for structured documents," in *Proceedings of CAISE' 02*, Toronto, Canada, June 2002.

I. Vessey, "Cognitive Fit: A Theory-based Analysis of the Graphs versus Tables Literature," *Decision Sciences*, 22(2), 1991, 219-240.

W3C, "XML Query Use Cases," W3C Working Draft, http://www.w3c.org/TR/xmlquery-use-cases, 2002.

C. Welty and David W. Stemple, "Human Factors Comparison of a Procedural and Nonprocedural Query Language," ACM Transactions on Database Systems, 6(4), 1981, 625-649.

C. Welty, "Human Factors Studies of Database Query Languages: SQL As a Metric," *Journal of Database Management*, 1(1), 1990, 2-10.

Table 1 Example queries

Query	DSQL	XQuery
for each publisher, find the number of distinct books published by them	SELECT b.publisher, count(distinct b.title) FROM book b GROUP by b.publisher	**<result> {** **FOR $p IN distinct(book/publisher)** **RETURN** **<publisher>** **{$p/text()}** **</publisher>** **<bookcount>** **{ FOR $b IN book[publisher = $p]** **RETURN { count(distinct-values($b/title)) }** **}** **</bookcount>** **}** **</result>**