Using a Relational Database Management System to Implement XML-QL

Steven M. Beitzel, Eric C. Jensen, David A. Grossman
Information Retrieval Laboratory
Department of Computer Science
Illinois Institute of Technology
10 W. 31st Street
Chicago, IL 60616
{steve,ej,grossman}@ir.iit.edu

Frederick J. Ingham, Tim Lewis
Bitsystems, Inc.
{inghamf,lewist}@bit-sys.com

Abstract

We present an innovative method for implementing XML-QL using a relational database back-end and ad hoc query translation to SQL. Our system guarantees support for XML documents of any schema, or any combination of schemas. Furthermore, we ensure that any changes to the schema of XML source documents will not require a change to the underlying schema of the relational database. We have developed a prototype system which accepts XML-QL queries and translates them to a set of equivalent SQL queries. These queries are then posed to the relational back-end, and results are obtained. We will discuss the architecture and implementation details of our prototype, and give a detailed description of the algorithms used in query translation.

1 Introduction

As more and more structured information is migrated to the World Wide Web, it is becoming necessary to publish it in such a way that it can be retrieved and viewed by all parties. As a solution to this problem, the World Wide Web Consortium (W3C) has created the eXtensible Markup Language (XML) standard [10, 11]. This markup language, inherited from SGML, allows documents written in it to specify their own schema. This allows for tremendous flexibility in designing semi-structured collections, while solving the problem of platform and application dependence. Application-defined schemas also allow XML documents to maintain hierarchical continuity over the data they represent, which clearly has tremendous advantages when one is working with data of a hierarchical nature.

Because XML is a relatively new standard, methods and tools for searching collections of it are still being researched and developed. There are a large number of proposed query languages for XML data, but a standard has not yet been agreed upon [12, 17, 19]. We chose to implement XML-QL [14, 15, 13], one of the most mature and full featured languages being considered for standardization. Our prototype is able to combine the proven technology of relational databases and SQL with an XML-specific query language in XML-QL to provide a reliable and scalable method for storing and searching a corpus of XML documents.

Section 2 will give background information on the problems at hand. Section 3 will discuss the relational architecture of our prototype, closely examining our schema. Section 4 will discuss our approach to translating XML-QL queries to SQL. Section 5 will give a summary and directions for future work.

2 Background

The development of our prototype has been chiefly driven by two main problems that are faced by the community of XML users, the selection of an appropriate query language, and the selection of a platform. We will give background information on the state of each problem here.

2.1 Query Languages

In order for the XML standard to become practically usable as a document format, there must be a way to search collections of XML data in a concise, effective manner. XML is a *semi-structured* data format [18, 20], which is to say that it can be used to add a dimension of structure to data that is primarily text-based. As such, the query languages that are currently most popular with structured and unstructured data are not suitable, as they do not fully realize the capabilities provided by XML. Keyword Search, which is most often used to express queries on unstructured data, is too imprecise to express a query on data containing semi-structural elements. Similarly, SQL (Structured Query Language), which is the most widely used query language for structured data, is difficult to use for searching semi-structured data, as it imposes highly strin-

gent structural elements on the user. Additionally, neither natural language nor SQL are very well-suited to expressing hierarchical queries. To transcend these restrictions, a large number of XML query languages have been developed [16, 19], and the W3C is currently reviewing several of them for standardization.

2.2 Storage Platforms

In addition to the problem of an appropriate query language, there are varying opinions on how to best store hierarchical data, and also on the best methods for indexing it. Some have tried to build tree structures to hold the data, but this lacks portability and cannot take advantage of the many features present in database systems [8]. One approach is to build a database system specifically tailored to storing semi-structured data [21, 22]. This approach is advantageous in that, it allows indexing structures and query optimization to be specifically tailored towards semi-structured data. Unfortunately, relational databases are inherently flat in structure, and it is difficult to map hierarchical data onto them. This forces most developers of semi-structured databases to base their approach on the less popular object-oriented database design. While this approach clearly has its roots in the database community, others approach the problem with traditional Information Retrieval techniques, choosing to build inverted index structures using the values of nodes in an XML document [23, 24]. This allows many common IR techniques to be used for evaluating the similarity of a query, and is very well suited to performing free-text searches on the data. A drawback of this approach is that it becomes difficult to query for documents which contain a certain range of values, and the structure of an inverted index, like that of a database, is flat, making the storage of hierarchical data a concern. In order to achieve the benefits of both of these strategies, while still preserving data hierarchy, much research has focused on storing both the XML data and it's hierarchy in a relational database [2, 3, 4, 7]. A common method for doing this involves extracting an appropriate relational schema from the schemas and DTD's of XML documents [13, 6]. We extend on this approach by developing a query processing system with an *unchanging* relational schema, and support for XML documents of any schema.

3 Relational Architecture

The architecture of our prototype uses a relational database as the back-end storage mechanism for the source collection of XML documents. In order to efficiently store XML documents, we have developed a relational schema with two critical properties: it can support XML documents of any schema without modification, and it allows for the preservation of hierarchy in a document that is entered into the system. The latter property of the schema solves the most limiting factor imposed by flat relational structures.

Our prototype contains an indexing module, which accepts the source XML documents and builds an index of them in the database. The index structure is stored in the relations contained in our schema. Each incoming document is given a unique ID, and has the entirety of its content stored as a TEXT field in one relation. As the documents are entered into the database, they are parsed with XERCES, a DOM-compliant XML parser, and all unique paths are extracted. A relation exists which contains each unique path found

```
WHERE
  <Set of Path Conditions>
  <Set of Predicate Conditions>
IN <collection>
CONSTRUCT
  <requested output specification>
```

Figure 1: Simple XML-QL Query

in the collection, along with a unique ID. A similar operation is performed for all unique tags and all unique attributes, each of which has its own unique ID and is stored in its respective relation.

Once all of the unique tags, paths, and attributes in a document have been discovered, an index of the document is built in a master relation which joins across the tag, path, and attribute relations. The records in this relation record the proper foreign keys for the current path, tag name, and attribute name, and they also contain the element type (tag or attribute), and element value, along with a unique ID for each record, and the ID of the document that each record is built from. Clearly, our index relation will contain a large number of records, even for small documents.

4 Query Translation

In order to query the index relations created in our back-end database, we developed a method to translate queries expressed in XML-QL to a set of SQL queries which produce results that are equivalent to those of the original XML-QL query. A single SQL query is generated for each distinct XML-QL query block (nested queries demand their own distinct SQL queries), so we can take full advantage of the database's query optimization techniques. In order to parse the XML-QL queries, we utilize the grammar and parser provided by the AT&T Research Laboratories, the initial developers of the XML-QL query language [15]. We used their query parser to create a parse tree for the posed queries, and implemented a visitor that traverses that parse tree and builds the appropriate SQL. In order to discuss the specifics of the algorithms used, some background on what comprises an XML-QL query is needed.

4.1 XML-QL Query Structure

An XML-QL Query, on a very general level, has structure similar to the one shown in Figure 1. Each WHERE clause in a query can contain a number of conditions, where conditions are either a tagpattern that the document must match, or a predicate, which imposes a set of limiting conditions on a value in the documents.

4.2 Tag-Path Condition Translation

In order to construct a SQL query equivalent to a Tag-Path condition, we must gather the full path for this condition and all bound variables which appear inside it. When the visitor encounters a Tag-Path condition in the parse tree, its children are visited in sequence. Once the processing of *TagPattern* and its children is complete, we can use the gathered information to build our SQL queries for this condition. This is done by selecting all documents from

our database which contain all of the paths discovered by the visitor, and ensuring that all element and attribute values specified in the query occur at the proper paths. This process requires a series of self-joins, one for each distinct path element located in the Tag-Path condition. These joins are limited by any literal values that were provided in the paths. They are selected if and only if a variable is bound to their path and used in the CONSTRUCT statement. The bound variables values are not selected if they are not needed to construct output.

4.3 Predicate Condition Translation

Once the appropriate SQL for implementing the conditions imposed by the *TagPattern* conditions in the query has been generated, we focus our attention on the Predicate conditions. Predicate conditions are most often used to pose some restriction on the result set based on the value, or a range of possible values, for some element in the source collection. As such, they are the most common method for limiting the joins performed on the tag paths. For each predicate, the visitor processes the lvalue, relational operator, and rvalue out of the parse tree and converts them to their SQL equivalents. This expression is returned to the higher level visitor calls where it is appended to the single SQL query's WHERE condition to limit the join corresponding to the tag path(s) which bound this variable(s).

4.4 Result Construction

Results are constructed by replacing variable values returned from the SELECT statement into the result template provided by the user in the CONSTRUCT statement. This template can have embedded conditions facilitated by nested XML-QL queries for added flexibility, in which case those XML-QL queries are converted to SQL and executed to enable replacement of their results into the higher level template. A significant feature of our system is that it does not do any processing of the results beyond this text replacement. All query processing is carried out in the SQL query, and results are directly extracted from the results of that.

5 Summary and Future Work

We have presented a prototype system for efficient storage and querying of large collections of XML data. Others have examined the question of the relational structure from the standpoint of performance [1]. We will extend on the high-performance structures presented to enable full XML-QL queries to be generated for them using our algorithms.

We are also working to generalize the storage and retrieval system to be a record-based rather than a document-based system. This approach will allow the system to query large single XML documents that may contain many records, and return each matching record, thus giving the query system a much finer granularity. Research into the performance effects of breaking large documents into smaller chunks for context indexing will also be performed.

There is also work to be done in possibly adding a free-text extension to XML-QL, which would allow for easy integration of our system as part of a web-based portal or mediator.

References

[1] D. Florescu, D. Kossmann Storing and Querying XML Data using an RDMBS *IEEE Data Engineering Bulletin, 22(3):27-34, 1999.*

[2] P. Boncz, A. Wilschut, M. Kersten Flattening an Object Algebra to Provide Performance *IEEE 14th International Conference on Data Engineering, 568-577, 1998.*

[3] R. Zwol, P. Apers, A. Wilschut Modelling and querying semistructured data with Moa *proceedings of Workshop on Query Processing for Semistructured Data and Non-standard Data Formats, 1999.*

[4] A. Schmidt, M. Kersten, M. Windhouwer, F. Waas Efficient Relational Storage and Retrieval of XML Documents *Proceedings of the Third International Workshop on the Web and Databases, 47-52, 2000.*

[5] A. Deutsch, M. Fernandez, D. Suciu Storing Semistructured Data with STORED *SIGMOD Conference, 431-442, 1999.*

[6] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, J. Naughton Relational Databases for Querying XML Documents: Limitations and Opportunities *Proceedings of the 25th VLDB Conference, 1999.*

[7] T. Shimura, M. Yoshikawa, S. Uemura Storage and retrieval of xml documents using object-relational databases *Proc. of DEXA, Florence, Italy. Lecture Notes in Computer Science, 1677:206–217, 1999.*

[8] C. Kanne, G. Moerkotte Efficient storage of XML data *Proc. Of the 16th Int. Conf. On Data Engineering, 2000.*

[9] S. Abiteboul Querying semi-structured data *ICDT, volume 6, pages 1–18, 1997.*

[10] The Extensible Markup Language *(XML)* http://www.w3.org/XML/

[11] Extensible Markup Language *(XML)* Second Edition http://www.w3.org/TR/2000/REC-xml-20001006

[12] The Query Languages Workshop http://www.w3.org/TandS/QL/QL98/

[13] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu A query language for XML *International World Wide Web Conference, 1999.*

[14] XML-QL: A Query Language for XML http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/

[15] XML-QL: A Query Language for XML http://www.research.att.com/sw/tools/xmlql/

[16] XML Query Languages: Experiences and Exemplars http://www-db.research.bell-labs.com/user/simeon/xquery.html

[17] J. Robie. The design of XQL, 1999. http://www.texcel.no/whitepapers/xql-design.html

[18] P. Buneman. Tutorial: Semistructured data. *In Proceedings of ACM Symposium on Principles of Database Systems, pages 117–121, 1997.*

[19] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener The Lorel query language for semistructured data. *International Journal on Digital Libraries, 1(1):68–88, April 1997.*

[20] R. Goldman, J. McHugh, and J. Widom From semistructured data to XML: Migrating the Lore data model and query language. *In Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99), Philadelphia, Pennsylvania, June 1999.*

[21] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom Lore: A Database Management System for Semistructured Data. *SIGMOD Record, 26(3):54-66, September 1997.*

[22] D. Quass, J. Widom. R. Goldman, K. Haas, Q. Luo, J. McHugh, S. Nestorov, A. Rajaraman, H. Rivero, S. Abiteboul, J. Ullman, and J. Wiener LORE: A Lightweight Object REpository for Semistructured Data. *Proceedings of the ACM SIGMOD International Conference on Management of Data, Montreal, Canada, June 1996. Demonstration description.*

[23] Dongwook Shin BUS: An Effective Indexing and Retrieval Scheme in Structured Documents. *Proceedings of Digital Libraries '98.*

[24] Dongwook Shin Structured querying, indexing, and retrieval for SGML/XML documents. *Proceedings of SGML/XML Japan '98, pp. 199-214.*