XML and Relational Database Management Systems: the Inside Story

Michael Rys
Microsoft
mrys@microsoft.com

Don Chamberlin
IBM
chamberlin@almaden.ibm.com

Daniela Florescu
Oracle
dana.florescu@oracle.com

ABSTRACT

As XML has evolved from a document markup language to a widely-used format for exchange of structured and semi-structured data, managing large amounts of XML data has become increasingly important. A number of companies, including both established database vendors and startups, have recently announced new XML database systems or new XML functionality integrated into existing database systems. This tutorial will provide an insight into how XML functionality fits into relational database management systems as seen by three major relational vendors: IBM, Microsoft and Oracle.

1. INTRODUCTION

XML was originally designed as a simplified form of SGML, a document markup language with a simple syntax and an extensible vocabulary. Its use quickly expanded beyond document markup to encompass general data interchange and representation of tree-structured data. A number of domain-specific XML vocabularies such as XBRL [1] and HL7 [2] have been defined. XML has been widely used in e-commerce and serves as the basis for web-services-related languages such as SOAP [3] and WSDL [4]. Increasingly, XML is the format of choice for data that needs to be self-describing because it is sparse or heterogeneous, or for data that has an intrinsic order that carries semantic meaning.

Systems for managing XML data fall into two major categories: specialized systems designed specifically and exclusively for XML, and more general systems designed to manage XML among other data formats. The latter category consists primarily of extended or "universal" relational database systems, and is the subject of this tutorial. These systems promise to provide integrated management of structured and unstructured data, with the capabilities such as concurrency control, backup and recovery, and automatic optimization that relational database users have come to expect. In many cases, they also provide additional XML-related functionality such as management of schema information and validation of XML documents against a designated schema.

Management of XML data has been the subject of work by several industry consortia and standardization bodies. XML itself is a Recommendation of the World Wide Web Consortium (W3C) [5] [6]. Other W3C Recommendations include XML Schema, a type system for XML [7]; XPath, a language for navigating within XML documents [8, 9]; and XSLT, an XML transformation language [10, 11]. W3C is also developing a new general-purpose XML query language called XQuery. XQuery, which is based on the XML Schema type system and includes XPath as a subset, is currently in Last Call as a potential W3C Recommendation [12]. At the same time, the owners of the SQL standard, ANSI/INCITS H2 and ISO WG3 SC32, have published a new set of XML-related functionality as part of the SQL 2003 standard [13].

This tutorial aims to provide insight into how three major relational database vendors, IBM, Microsoft, and Oracle, fit XML and related technologies into a relational database environment. It will cover the organization and storage of XML data, how data is typed and validated using XML Schemas, and how it is accessed and updated using interfaces such as SQL/XML and XQuery.

The tutorial is organized as follows: This introduction provides general background information and introduces some terminology and topics upon which the subsequent vendor-specific parts will expand. In the sections that follow, the individual vendors provide more detailed information on how integration of XML and relational data is handled by their respective systems.

2. STORING XML IN RELATIONAL DATABASE SYSTEMS

The simplest approach to storing XML data in a relational database system is to use a long-character-string datatype, such as CLOB in SQL, to store XML documents or fragments as text in columns of tables. This approach might be said to provide *textual fidelity* because it preserves the original XML at the character-string level. The disadvantage of this approach is that it fails to take advantage of the structural information that is available in the XML markup. A generic string datatype does not provide any specialized support for searching based on semantic content, or for retrieving XML data at a fine level of granularity.

Another method for storing XML documents in relational databases, called *shredding*, distributes the XML information across the columns of one or more tables, preserving both data values and structural relationships. This technique is most commonly used for XML documents whose structure is described by an XML schema. A mapping is defined from the XML schema to a relational database design, typically using a different table for each level of the XML element hierarchy. This approach is called *schema-based shredding*. If a schema is not available, other forms

SIGMOD'05, June 14–16, 2005, Baltimore, Maryland, USA.

of shredding can be used. In general, as the structure of the stored XML documents becomes less regular and predictable, the mapping of these documents into relational tables becomes more generic and less efficient. Shredding can accommodate a certain degree of flexibility in document structure—for example, the children of an element may be optional and may vary in cardinality. However, the shredding technique is not efficient for sparse elements (in which content varies widely from one element to another), or for elements with mixed content (containing both text and child elements). Furthermore, shredding generally fails to preserve some of the XML-centric aspects of stored data, such as document order and processing instructions.

Systems that use shredding to transform XML data into relational data generally provide an inverse transformation, often called *XML publishing*, to reconstruct XML documents from tabular data. XML publishing is also used by pure relational systems to export relational data in XML format, using markup to represent structural and semantic information. New syntax in support of XML publishing was added to the SQL standard in 2003 [13]. In the XML publishing approach, selective retrieval of stored data is accomplished by SQL rather than by an XML query language.

XML storage systems based on shredding and XML publishing are said to provide *relational fidelity*, because the authoritative form of the stored data is relational rather than XML. Examples of systems that use these techniques include Microsoft SQL Server 2000, Oracle 8i, and IBM DB2 Version 8.

While shredding and XML publishing are adequate for many important use cases, a more general storage technique is needed to take advantage of the structural information in unconstrained XML documents. For this purpose, the SQL-2003 standard provides a new datatype called `XML` for storing well-formed XML documents and fragments, based on the XML Information Set [14] and–in an upcoming revision of the standard–on the XQuery Data Model [15]. Since this data model can represent all aspects of a well-formed XML document, systems based on the `XML` datatype are said to provide *XML fidelity*. These systems are able to preserve XML-centric information such as document order and namespace bindings, and to exploit this information using an XML-based query language such as XQuery. For these reasons, this level of support is often referred to as *native XML*.

The logical data model on which the `XML` datatype is based does not specify any particular organization for physical storage. Many physical storage techniques are possible, providing various tradeoffs for time and space efficiency under query and update. In addition, access aids such as indexes may be created to improve query performance. Index creation and maintenance are more complex in a native XML system than in a pure relational system, since the XML data model is less constrained than the relational data model. XML indexes may support access to data at various levels of the element hierarchy, and the objects indexed may vary in cardinality and datatype. In general, native XML databases raise new challenges for all aspects of query optimization, including join planning, index selection, and cost estimation.

The vendor-specific sections that follow will provide more detail on the storage and access techniques used and the level of fidelity provided by their respective implementations, as well as their strategies for indexing XML data and optimizing XML queries.

3. XML TYPING AND VALIDATION

In addition to storing documents, native XML storage systems often provide a way to store XML schemas and to validate stored documents with respect to specific schemas. Facilities may be provided to constrain the contents of a given column to documents that have been validated against a given schema or set of schemas. In addition to guaranteeing the integrity of stored data, the validation process generates type information that can be useful during query optimization and execution.

The operators of XQuery are defined both for untyped data and for data of a known type such as `xs:string` or `xs:decimal`. In general, XQuery operators attempt to coerce untyped data into a type that they understand—for example, numeric operators attempt to cast untyped data into a numeric type. This polymorphic behavior incurs a certain amount of overhead for run-time dispatching of the proper operation, which can be avoided for data that has a known type due to schema validation.

Another potential use for schema type information lies in static type-checking of queries. If schema definitions are available for the elements and attributes referenced in a query, those definitions can be exploited to infer the result-types of various expressions. By static analysis of the query, then, certain kinds of errors can be detected and type information can be extracted that is useful in query optimization. A set of static type inference rules has been defined as an optional feature of the XQuery language [16].

Integration of the XML and relational type systems presents many challenges. The primitive types are different and the mechanisms for deriving new types from existing types are different. The notion of a null value, widely used in relational databases, is missing from XML. The notions of sequences and ordering, central to the XML data model, are missing in relational databases. In order to accommodate XML data, the basic semantic primitives of a relational database system must be expanded and adapted in interesting ways.

The vendor-specific sections that follow will provide more detail on how schemas and type information are stored and exploited by their respective implementations.

4. QUERYING AND MANIPULATING XML DATA

Among the most important aspects of an XML storage facility are the interfaces provided to access and manipulate the stored data. For this purpose, several XML-related languages have been defined. XPath [8, 9] provides a navigation facility within XML documents but does not provide an ability to transform structures or to construct new elements. XSLT [10, 11] supports transformation and construction, but its recursive template-driven nature is not well-suited to optimization or static analysis. XQuery [12] includes XPath as a subset, and provides a complete set of query facilities, including transformation and construction. The syntax and processing model of XQuery are similar enough to SQL to be familiar to relational users and amenable to the kinds of optimization often used by relational systems.

The next version of the SQL/XML standard [13] provides syntax whereby an XQuery can be invoked from within an SQL statement for retrieving data at a fine level of granularity from within a stored XML document. The data returned by XQuery to

the SQL environment can then be processed further using SQL facilities such as grouping and aggregation.

Most SQL/XML implementations also support the inverse of this facility, in which relational data can be accessed from within an XQuery. The next version of the SQL/XML standard will define a facility for binding SQL data to XQuery variables. Some systems also provide implementation-defined XQuery functions that accept SQL queries and return their results to XQuery for further processing. For example, when XML documents are stored in columns of tables, an XQuery might call a function that invokes an SQL query to extract its input documents.

The similarities in syntax and processing model between XQuery and SQL, and the facilities in each language for invoking the other, make the two languages complementary and well-suited for a hybrid system that stores and manages both XML and relational data. With some variations, each of the systems described in the vendor-specific sections that follow is such a hybrid system.

In addition to query facilities, a database management system needs facilities for inserting, deleting, and modifying data. Currently, the XQuery specification does not provide any definition of these facilities. For XML data, the minimum necessary facility must be able to insert documents and document fragments into the data store and to delete them. If the XML data is stored in table columns, this functionality can be provided using the SQL language. A future version of XQuery is expected to provide a syntax for updating stored instances of the XML data model at a finer granularity. A working draft containing requirements for such an update facility has been published by the XML Query Working Group [17]. Until this work is complete, each XQuery implementation is addressing the data manipulation requirement in its own way.

XQuery Version 1.0 is now in "last call" status and may soon become a W3C Recommendation. This version of the language includes only facilities for exact queries—that is, queries with well-defined results. However, the XML Query Working Group has created a task force to study full-text search functionality, in which a query ranks a set of input documents with respect to some criterion to find the most relevant documents. Full-text search takes word proximity into account and uses techniques such as synonyms and stem-matching. Because of the heuristic techniques used, the result of a full-text search is not well-defined. Working drafts containing a proposed syntax for full-text searching in XQuery and a set of example use cases have been published by the task force [18, 19]. This functionality is particularly important for information-retrieval applications where XML is used in its original role as a document markup language.

The vendor-specific sections that follow provide more details about the query interfaces supported by their respective systems and how these systems deal with requirements for data manipulation and full-text search. Consideration is also given to how XML data fits into the general administrative tools and interfaces of the respective systems.

5. REFERENCES

[1] Extensible Business Reporting Language. See http://www.xbrl.org/home.

[2] Health Level 7 XML Special Interest Group (R. H. Dolin and P. V. Biron, editors). *Using XML as a Supplementary Messaging Syntax for HL7 Version 2.3.1.* See http://www.hl7.org/special/committees/sgml/hl7v231xmlFINAL.zip.

[3] W3C XML Protocol Working Group (M. Gudgin et al, editors). *SOAP Version 1.2 Part 1: Messaging Framework.* See http://www.w3.org/TR/soap12.

[4] E. Christensen et al. *Web Services Definition Language (WSDL) 1.1*. W3C Note. See http://www.w3.org/TR/wsdl.

[5] World Wide Web Consortium (W3C). See http://www.w3.org.

[6] T. Bray et al. *Extensible Markup Language (XML) 1.0.* W3C Recommendation. See http://www.w3.org/REC-xml.

[7] W3C Schema Working Group (H. Thompson et al, editors). *XML Schema, Parts 0, 1, and 2*. See http://www.w3.org/TR/xmlschema-0, -1, and -2.

[8] W3C XSL and Linking Working Groups (J. Clark and S. DeRose, editors). *XML Path Language (XPath) Version 1.0.* See http://www.w3.org/TR/xpath.

[9] W3C Query and XSL Working Groups (A. Berglund et al, editors). *XML Path Language (XPath) Version 2.0.* See http://www.w3.org/TR/xpath20.

[10] W3C XSL Working Group (J. Clark, editor). *XSL Transformations (XSLT) Version 1.0.* See http://www.w3.org/TR/xslt.

[11] W3C XSL Working Group (M. Kay, editor). *XSL Transformations (XSLT) Version 2.0.* See http://www.w3.org/TR/xslt20.

[12] W3C Query Working Group (S. Boag et al, editors). *XQuery 1.0: An XML Query Language.* See http://www.w3.org/TR/xquery.

[13] International Organization for Standardization (ISO). *Information Technology—Database Language SQL—Part 14: XML-Related Specifications (SQL/XML).* Standard No. ISO/IEC 9075:2003. Available from American National Standards Institute, New York.

[14] W3C XML Core Working Group (J. Cowan and R. Tobin, editors). *XML Information Set*. See http://www.w3.org/TR/xml-infoset.

[15] W3C Query and XSL Working Groups (M. Fernandez et al, editors). *XQuery 1.0 and XPath 2.0 Data Model*. See http://www.w3.org/TR/xpath-datamodel.

[16] W3C Query and XSL Working Groups (D. Draper et al, editors). *XQuery 1.0 and XPath 2.0 Formal Semantics*. See http://www.w3.org/TR/xquery-semantics.

[17] W3C Query Working Group (D. Chamberlin and J. Robie, editors). *XQuery Update Facility Requirements*. See http://www.w3.org/TR/xquery-update-requirements.

[18] W3C Query and XSL Working Groups (S. Amer-Yahia et al, editors). *XQuery 1.0 and XPath 2.0 Full-Text*. See http://www.w3.org/TR/xquery-full-text.

[19] W3C Query and XSL Working Groups (S. Amer-Yahia and P. Case, editors). *XQuery 1.0 and XPath 2.0 Full-Text Use Cases*. See http://www.w3.org/TR/xquery-full-text-use-cases.