XQuery in Relational Database Systems

Michael **Rys** <mrys@microsoft.com>

Abstract

Relational database systems (and the related standards body ANSI/INCITS H2) are busy adding XML support. One of the main components of such XML extensions will be support for the upcoming XML query language XQuery.

The presentation will outline how XQuery and XML conceptually fit into a relational database environment. It will cover the organization of the XML in the database, how to type it using W3C XML Schema, how to query it both in conjunction with SQL and using top-level XQuery. It will present the concepts, talk about new developments in the ISO/ANSI SQL/XML standards and present some demos of XQuery in the upcoming Microsoft® SQL Server 2005.

Table of Contents

1. Introduction

Relational database systems have embraced XML in many different ways over the last couple of years. First generation XML support in relational database systems mainly concerned themselves with exporting relational data as XML, where the markup was communicating structural and semantic information, and to import relational data in XML markup form back into a relational representation. The first functionality is often called XML Publishing (in the database sense) and the second is often referred to as “shredding”. The main usage scenario supported by these systems is information exchange in contexts where XML is used as the “wire”-format and where the relational and XML schemas are often predefined independently of each other.

While this use case was and still is one of the most important aspects of integrating XML and relational database systems, it is not the only one. With the advent of XML object serializations and last but certainly not least the increased use of XML for representing documents and forms, an increasing number of customers want to store their XML documents inside a database management system without having to manually “shred” them into a relational structure, while they still want to be able to query the data. Often, schema designers also want to use XML to represent so-called semistructured data, that has heterogeneous and potentially fast changing structure that cannot easily be represented using SQL tables. These requirements lead the database systems to add so-called native XML support. Providing native XML support means that the XML is stored as a unit, preserving at least the XML Infoset properties (including order among elements, mixed content etc.), and does not need to be manually shredded. It also means that additional functionality such as support for the upcoming XML query language XQuery is provided to unlock the information hidden inside the XML..

All the major relational database systems (and the related standardization bodies ANSI/INCITS H2 and ISO WG3 SC32) are busy adding native XML support. This paper will outline how XQuery and XML conceptually fit into a relational database environment. It will cover the organization and storage options of the XML in the database, how to type and validate the XML using W3C XML Schema, how to query it both in conjunction with SQL and using top-level XQuery. The paper will also include some of the major new developments in the ISO/ANSI SQL/XML standard[SQLXML 200n]

2. “Natively” Storing and Organizing XML in a Relational Database

All relational database systems that provide for large byte stream data types (commonly referred to as BLOB or CLOB for binary/character large objects) can store XML documents without shredding it. However, such storage is “dumb”, meaning, that it does not guarantee XML properties such as well-formedness and one has to use external code or full-text search capabilities to unlock the information provided in the XML. In order to provide XML semantics, the XML part of the SQL-2003 standard[SQLXML 2003]provides a new data type called`XML`, which all the major database vendors provide in one form or another. Its logical model allows XML documents and content fragments (multiple text or element nodes at the top) and is defined based on the XML Information Set. In the next version of the SQL standard, the`XML`data type will be defined based on the XQuery1.0/XPath 2.0 data model[XQuery DM]and provide for a variety of XQuery type based constraints (for some of them, see below). The following example gives a simple create table statement, that defines a column of type`XML`:

Example 1. Creating a table with an XML column

```
CREATE TABLE Order(id int, orderdate date, PODetail XML)
```

As outlined in[Rys 2003], there is a variety of physical formats that a database system can choose from to store an instance of type`XML`: a binary or character large object, decomposed into tables or a combination of both. Microsoft’s SQL Server 2005 for example stores the data in an internal binary format as a BLOB and then provides a primary XML index, that will shred the XML into a so-called node table, and some secondary indices for improving XQuery performance on the XML (see[PCSSGZ 2004]and[OOPCSW 2004]for more details). The binary BLOB provides the advantage of some space saving (average around 20% compared to the original textual size) and efficient retrieval and reconstruction of the original XML stream while the indices support an efficient query mechanism that can utilize the relational query engine and optimizer (see the query section below for more details). Oracle and IBM DB2 on the other hand provides a shredded physical representation for certain schematized XML and a BLOB for the general case.

Another useful organizational concept for managing XML besides allowing a column of type XML in a relational table is the notion of a collection of XML documents. A collection is similar to a table where the XML column is the only column. Such a concept can either be provided using a storage abstraction such as a generalization of the notion of a table to encompass a named table of type XML, or by a functional abstraction over an XML column in a conventional relational table (see the section on top-level XQuery for more details on the latter approach). The notion of a collection of XML is currently not part of the standardization effort.

The logical and physical model of XML is not the only important aspect. One of the main strengths of integrating XML into a relational database system is its integration with the relational model and mature database management functionality. Thus, it is paramount that the `XML` data type can be used like any other scalar SQL data type and can benefit from backup, recovery and other functionality provided by a mature database management system.

3. Validating and Typing an XML data type

While the SQL-2003 standard only defines a non-validated XML data type, several products and the next version of the standard both add support for W3C XML Schema validation and the XQuery type system to the XML data type.

XML Schemas describe the structure and types of elements and attributes. Each type, element or attribute description is called an XML Schema component. Each component has a name (a so-called qualified name or QName) that consists of a potentially absent namespace URI and the local name part. For example, the XML Schema element for defining elements has a namespace URI of*http://www.w3.org/2001/XMLSchema*and a local name*element*. Schema components are normally described by XML Schema resources describing components of a specific namespace URI (its target namespace) that may import other schemas (and thus namespaces).

In the following, let us look at how SQL Server 2005 manages and utilizes XML Schemas. Relational database management systems have no understanding of URIs for naming and identifying resources. Instead they use their own naming mechanism which is based on names scoped to database schemas and catalogs. This naming mechanism can be used to name a collection of XML Schema components of one or more XML Schema resources. For example, the expression`CREATE XML SCHEMA COLLECTION S1 AS @s`creates a SQL Server XML Schema collection with name`S1`that consists of the XML Schemas (1 or many) provided by the SQL variable @s. Each such collection describes potentially multiple namespaces and stores all the necessary information to perform validation and typing in the database schema's metadata. Using a schema collection has two advantages over just relying on some namespace URI: A database schema can contain different versions of the same URI in different schema collections (for example the different XHTML schemas) and the collection gives a closed, consistent type world for open content sections that helps in statically typing XQuery expressions (see below). A disadvantage is that it requires all schemas to be part of the schema collection and (at least for now) does not support references to externally located schemas. As normal in database management systems, certain forms of schema evolutions are being supported.

SQL Server 2005 allows you to associate a schema collection with an XML data type and to differentiate between an XML type that has to contain a well-formed document or may contain so-called content fragments (more than one element or text node directly under the document node). The additional schema constraint will guarantee that any XML instance inserted into the column will be valid according to the schema and data in the XML will be typed according to the types provided in the schemas. For example, a price element will now be of type`xs:decimal`instead of being untyped if the provided schema provides this type information. The following example shows a table definition that constrains the instances in the PODetail XML column to a well-formed document that is valid according to a PODetailSchema schema collection in the same database:

Example 2. Creating a table with a typed XML column

```
CREATE TABLE Order(id int, orderdate date, PODetail XML(DOCUMENT PODetailSchema))
```

Casting from an untyped XML data type instance to a schema collection constrained XML type will provide validation, such as in`CAST(@x AS XML(S1))`.

How does this compare to the upcoming SQL-200n (where n is 5 or 6) standardization? That standard provides for both an XML Schema identifier and a namespace-URI/location-URI based addressing mechanism. SQL Server 2005's schema collection is basically a form of an XML Schema identifier approach. Also the standard allows for more detailed constraints down to the element declaration level, functionality that could easily be added in a future version of SQL Server. The standard is still in development, but it is very likely to provide a way to type an XML data type, a validation check expression (`IS VALID`) and a casting mechanism (`XMLCAST`).

4. Querying the XML data type

Once you store XML natively, you want to be able to query and transform the XML. While XPath 1.0 can provide some query capabilities and XSLT can do transformations, there are reasons to use XQuery instead of XPath 1.0 or XSLT as the XML query language. XPath 1.0 cannot perform transformations, has some very dynamic, implicit semantics that are hard to optimize, and has only a very rudimentary type system that is not usable for the more data-centric, semistructured data management scenarios. XQuery provides support for performing transforms, supports a richer type model and has cleaner implicit semantics. While XSLT also provides transformation capabilities, it is much harder to optimize an XSLT stylesheet, due to the data-driven nature of match templates. For example, static analysis of templates is much more difficult since templates may be called with different input and thus need to leave more decisions to runtime which will slow down the execution. XQuery on the other hand is more prescriptive in its transformation capabilities, which makes it better suited for optimization and is closer in its processing model to the known SQL processing model.

As outlined in[Rys 2003], you can query XML in (at least) two ways when stored inside a relational database. You can run queries on a per XML data type instance basis within the context of a relational query, or you can run your XQuery at the top, treating a column of type XML as a collection of XML documents. Both SQL Server 2005 and the upcoming SQL standard are focusing on the first capability for now, although I am fairly certain, that the second approach will be added eventually. Let’s look at each of the approaches.

4.1. How to use XQuery on the XML data type

So what does SQL Server provide? SQL Server 2000 offers an XML to rowset mapping function called OpenXML that uses XPath 1.0 expression to extract data from the XML structure into relational form. While this functionality is still available in SQL Server 2005, it less tightly integrated into the native XML processing than the new XQuery functionality. SQL Server 2005 provides XQuery-based query transform, atomic value extraction, existence check, and node-to-row mapping capabilities on the XML data type and some node-level update functionality using a data modification language that is based on XQuery.

SQL Server 2005 provides the query and modification capabilities using the method invocation syntax used also for the newly added CLR user-defined types (even though the XML data type is not a CLR user-defined type). Each method takes a string literal as the query string and potentially other arguments. The XML data type provides the context item for the path expressions and populates the in-scope schema definitions with all the type information provided by the associated schema collection, or if no collection is provided, assumes that the XML is untyped. This means that you do not have to use an explicit call to a document function or variable to bind to the XML to be queried and all schema information that is used to type the data is implicitly provided, thus removing the need for explicitly importing the schemas. Furthermore, the SQL Server 2005 XQuery implementation is statically typed, which will provide you with early path expression typing mistake, type error and cardinality mismatch detection, as well as some additional optimizations. The following gives an overview of the available query methods.

The query method takes an XQuery expression and returns an always untyped XML data type instance that then can be cast to a target schema collection if the data needs to be retyped. In XQuery specification-speak, we have set the construction mode to strip. The following example shows a simple XQuery expression that transforms a complex Customer element that contains a notes section using markup into a summary showing the name and sales leads for Customer elements that have sales leads:

Example 3. The XML data type query() method

```
select doc.query(
   'declare namespace c = "urn:example/customer";
    for $c in /c:doc/c:customer
    where $c//c:saleslead
    return
     <customer id="{$c/@id}">{
        $c/c:name,
        $c//c:saleslead
     }</customer>')
from XMLdoc
```

The above query will be executed for each row in the table XMLdoc and be applied to every doc XML data type instace.

The value method takes an XQuery expression and a SQL type name, extracts a single atomic value from the result of the XQuery expression, and casts its lexical form into the specified SQL type. If the XQuery expression results in a node, the typed value of the node will implicitly be extracted (in XQuery terminology: the node will be atomized). Note that the value method performs a static type check that at most one value is being returned. Since the static type of a path expression often may infer a wider static type, even though the dynamic semantics will only return a single value, we recommend using the positional predicate to retrieve at most one value. The following example shows a simple XQuery expression that counts the order elements in each XML Datatype instance and returns it as a SQL integer value:

Example 4. The XML data type value() method

```
select doc.value(
   'declare namespace c = "urn:example/customer";
    count(/c:doc/c:customer/c:order)', 'int')
from XMLdoc
```

The exist method takes an XQuery expression and returns 1 if the expression results in an non-empty result and 0 otherwise. The following expression retrieves every row for which the document contains at least one customer with a saleslead:

Example 5. The XML data type exist() method

```
select doc
from XMLdoc
where 1 = doc.exist(
  'declare namespace c = "urn:example/customer"; /c:doc/c:customer//c:saleslead')
```

So far, the expressions always map from one XML data type instance to one result value per relational row. Sometimes however, you want to split one XML instance into multiple subtrees where one subtree is in a row of its own for further relational and XQuery processing. This functionality is provided by the nodes method which takes an XQuery expression and generates a single-column row per node that the expression returns. Each cell in the row will contain a reference to a different node. Since the resulting type is a reference type that does not exists in SQL Server outside the context of a single query, any of the query methods have to be applied to materialize the result. The methods will be applied like on any other XML data type with the difference that the context item for the path expressions is not at the document root of the XML data type but at the referenced node. The following example will extract for every customer order in the XML column a row that contains the XML representation of its customer, the name of the customer, the order id and the id of the document that contains the customer:

Example 6. The XML data type nodes() method

```
select
   N1.customer.query('.') as customer,
   N1.customer.value(
      'declare namespace c = "urn:example/customer";
       c:name[1]', 'nvarchar(20)') as CustomerName,
   N2."order".value('@id', 'int') as OrderID,
   N1.customer.value('../@id', 'nvarchar(5)') as DocID
from XMLdoc
cross apply XMLdoc.doc.nodes(
       'declare namespace c = "urn:example/customer";
        /c:doc/c:customer') as N1(customer)
cross apply N1.customer.nodes(
       'declare namespace c = "urn:example/customer";
        ./c:order') as N2("order")
```

Note that this is similar to the OpenXML functionality also available in SQL Server with the difference, that this expression is integrated into the XQuery processing.

Sometimes, one wants to access relational data in the context of the XQuery expression. For that case, SQL Server 2005 has added two special functions called`sql:variable($varname as xs:string)`and`sql:column($colname as xs:string)`that take constant string literals to identify the variable or correlated column value to be used.

How does the database system execute these XQuery expressions? As mentioned earlier, the XML data is stored in an internal binary representation. However in order to execute the XQuery expressions, the XML data type will be transformed into a so called internal node table that a user can prematerialize using the primary XML index. The XQuery expressions are translated into an internal relational algebra that has been extended by some XML specific operations. The resulting algebra tree is grafted into the algebra tree of the relational expression so that in the end, the query execution engine will receive a single execution tree that it will optimize and execute.

The upcoming SQL standard will provide an`XMLQUERY`construct that applies an XQuery expression to a set of variables that bind the XML and relational values. It also allows setting the context item. Unlike the SQL Server 2005 case, the values are bound as XQuery variables instead of using pseudo-functions, which works with the keyword-based approach but is not easily doable with the functional syntax in SQL Server 2005. The following example shows the same query that we used to showcase the query method, but this time using the currently discussed SQL standard syntax setting the implicit context to the doc column:

Example 7. The XMLQUERY expression in SQL-200n

```
select XMLQUERY ('declare namespace c = "urn:example/customer";
   for $c in /c:doc/c:customer
   where $c//c:saleslead
   return
      <customer id="{$c/@id}">{
        $c/c:name,
        $c//c:saleslead
      }</customer>'
PASSING BY REF doc RETURNING CONTENT BY VALUE)
from XMLdoc
```

The standard is also contemplating to add`XMLEXISTS`with the same semantics as the exist method, a way to cast XML values into relational values using`XMLCAST`(similar to the value method), and an XML to rowset mapper called`XMLTABLE`which is very close to the SQL Server functionality of OpenXML and the nodes method. For example, the following statement corresponds to the nodes method example:

Example 8. The XMLTABLE expression in SQL-200n

```
select N1.customer, N1.CustomerName, N2.OrderID, N1.DocID
from XMLdoc,
     XMLTABLE('declare namespace c = "urn:example/customer"; /c:doc/c:customer'
        PASSING doc
        COLUMNS
          customer XML BY REF PATH '.',
          CustomerName nvarchar(20)
             PATH 'declare namespace c = "urn:example/customer"; c:name[1]',
          DocID nvarchar(5) PATH '../@id') AS N1,
     XMLTABLE('declare namespace c = "urn:example/customer"; $c/c:order'
        PASSING N1.customer AS c
        COLUMNS
          OrderID int PATH  '@id') as N2
```

4.2. Top-level XQuery

The previous section showed how to use XQuery together with SQL to operate on one XML instance at a time. Sometimes however, one wants to see a column of type XML as a collection of XML documents and operate on it with an XQuery expression without having to use SQL or have to aggregate the XML documents into a single XML document. This functionality, that is called*top-level XQuery*in[Rys 2003], is currently not provided in SQL Server or the upcoming SQL standard but will very likely be added in future versions.

Top-level XQuery becomes the top-level language that is used as the primary query language at the same level as SQL. It can subsume access to collections of XML documents as well as XML views of relational data. In the following, we will use a keyword XQUERY to indicate a top-level XQuery and place the XQuery expression into curly braces. Such an expression will return an instance of the XML data type. The following example shows how you could use top-level XQuery to query over a collection of XML data type instances stored in the`doc`column of the`XMLdoc`table and further customer information in a relational customer table. The XQuery function`sql:collection`will take the name of the table and column and provide a collection of XML documents, whereas the function`sql:table`provides an XML mapping of the relational data (using one of the mapping options provided by the SQL-2003 standard).

Example 9. Top-level XQuery

```
XQUERY {
   declare namespace c = "urn:example/customer";
   for $xc in sql:collection("XMLdoc.doc")/c:doc/c:customer
   for $rc in sql:table("Customer")
   where $xc//c:saleslead and $rc/id = $xc/@id
   return
     <customer id="{$xc/@id}">{
       $xc/c:name, $rc/address, $xc//c:saleslead
     }</customer>}
```

For more information on top-level XQuery, please see [Rys 2003].

5. Conclusion

Most relational database systems, such as SQL Server 2005, and the SQL standard are busy adding XQuery support to their XML data type. We have taken a look at the XQuery integration that SQL Server 2005 is providing and have seen the major parts of the XQuery support that is being added to the upcoming SQL standard. The already achieved level of integration shows that XQuery fits well into relational databases to manage their XML data. Future possible features such as top-level XQuery provide further, powerful mechanisms to query XML and relational data in an integrated way while being able to take full advantage of the major database management functionalities such as query optimization, concurrency control, replication, access control, backup and recovery.

Bibliography

[Rys 2003] Michael Rys. *XQuery and Relational Database Systems. In "XQuery From the Experts"*. Edited by HowardKatz, Addison-Wesley, 2003.

[PCSSGZ 2004] Shankar Pal, Istvan Cseri, Gideon Schaller, Oliver Seeliger, Leo Giakoumakis, Vasili Zolotov. *Indexing XML Data Stored in a Relational Database*, 1134-1145, VLDB Conference, 2004.

[OOPCSW 2004] Patrick E. O'Neil, Elizabeth J. O'Neil, Shankar Pal, Istvan Cseri, Gideon Schaller, Nigel Westbury. *ORDPATHs: Insert-Friendly XML Node Labels.*, 903-908, SIGMOD Conference, 2004.

[SQLXML 2003] Edited by JimMelton. *ISO/IEC 9075-14:2003, Information technology — Database languages — SQL — Part 14: XML-Related Specifications (SQL/XML)*, 2004.

[SQLXML 200n] Edited by JimMelton. *ISO/IEC 9075-14:200n, Information technology — Database languages — SQL — Part 14: XML-Related Specifications (SQL/XML)*, To be published.

[XQuery DM] Edited by MaryFernández, AshokMalhotra, JonathanMarsh, MartonNagy, and NormanWalsh. *XQuery 1.0 and XPath 2.0 Data Model*, W3C Working Draft 23 July 2004, http://www.w3.org/TR/xpath-datamodel/.

Biography

Michael **Rys**
Program Manager
Microsoft Corporation [http://www.microsoft.com]
Redmond
Washington
United States of America
mrys@microsoft.com

After finishing his Ph.D. at the Swiss Federal Institute of Technology in Zurich in the area of database systems, Michael Rys went to Stanford University for a Postdoc, where he worked on semi-structured databases and distributed heterogeneous information integration. In late 1998, he joined Microsoft Corporation in Redmond where he is now a Program Manager for SQL Server's XML Technologies. Michael is also a member of the W3C XML Query working group, INCITS/ANSI H2 and is a member of ACM and IEEE. He recently co-authored the acclaimed book "Experts on XQuery".