

# XML Features in SQL Server 2005 Express

Hua Ma

Suzanne W. Dietrich

Susan D. Urban

Department of Computer Science & Engineering

Arizona State University

Tempe, AZ 85287-8809

November 2004

## ***Introduction***

XML is a platform-independent data representation format. As XML has gained wide acceptance, it leads to the evolution of XML as a data storage format. In SQL Server 2000, there are two options to store the content of structured XML: store the entire document in a text field or convert the content of the document to relational tables. If the first option is applied, the DB server would only be used to store and retrieve the data. Queries against the XML content would occur on the client-side by parsing the retrieved XML document. The second option only works for structured XML data. For better XML data support, SQL Server 2005 introduces XML, a new native data type to the database system.

This document provides an overview of the use of the XML type in Microsoft SQL Server 2005 [Beauchemin, 2004; Brown, 2004; Pal et al., 2004; Rys, 2004]. The examples are presented in the context of the ECLECTIC ONLINE SHOPPING ENTERPRISE described in the ER diagram of Figure 1.1 in [Dietrich and Urban, 2005]. In the ECLECTIC ONLINE SHOPPING application, customer name, address, and other personal information is stored in the customerInformation table. A table named orderInformation contains the order history of all customers. An XML document is created from the customerInformation and orderInformation tables containing customer name and order information. The XML document content is shown as follows:

```

<dataRoot>
  <orderhistory>
    <customerInfo>
      <fName>first1</fName>
      <lName>last1</lName>
    </customerInfo>
    <orderInfo id="57" date="11/25/2000" totalPrice="509.7">
      <productInfo name="Toolbox" unitPrice="17.99" quantity="20" />
      <productInfo name="Toy Car Speedway Center" unitPrice="14.99" quantity="10"/>
    </orderInfo>
    <orderInfo id="58" date="11/25/2000" totalPrice="2909.79">
      <productInfo name="Gyroscope" unitPrice="29.99" quantity="1" />
      <productInfo name="Skooter" unitPrice="274.99" quantity="10" />
      <productInfo name="Doll Baby" unitPrice="12.99" quantity="10" />
    </orderInfo>
    <orderInfo id="59" date="11/25/2000" totalPrice="29.95">
      <productInfo name="Rubiks Cube" unitPrice="5.99" quantity="5" />
    </orderInfo>
  </orderhistory>
</orderHistory>...</orderHistory>
.....
</dataRoot>

```

As seen in the XML document, the customer first name and last name are stored as nested elements of the customerInfo element. Each customer may have many purchase orders so that several orderInfo elements may exist. Each orderInfo element consists of one or many products represented in the nested productInfo element.

## ***XML Type***

The new XML data type, called XML, is a first-class type. Thus it can be used in most of the ways any other SQL Server data type can be used, including as a column in a table, a variable in T-SQL, or a user-defined function parameter. XML values are stored in an internal format as large binary objects (BLOB). An XML type instance must be converted to or from a varchar or nvarchar type. The T-SQL INSERT statement will do automatic conversion from a varchar and nvarchar value used in a VALUES list to an XML data type column value.

The XML data type in SQL Server 2005 implements the ISO SQL-2003 standard XML data type [SQL, 2003]. It can store a well-formed XML document as well as an XML fragment without a root element. An XML data type can be associated with an XML

schema for data validation. Such an XML data is called typed XML; otherwise it is called untyped XML.

## Untyped XML

The untyped XML is an XML data without a bounded XML schema. The following statement creates a table called untypedHistoryTable with an id column (primary key) and an untyped XML column orderHistory.

```
CREATE TABLE untypedHistoryTable (  
id varchar(5) PRIMARY KEY,  
orderHistory XML)
```

As mentioned, SQL Server 2005 automatically converts an input string to an XML data type. Thus the INSERT statement can be used directly to add a new row into the table as follows:

```
INSERT INTO untypedHistoryTable VALUES  
(1,  
'<dataRoot>  
<orderhistory>  
  <customerInfo>  
    <fName>first1</fName>  
    <lName>last1</lName>  
  </customerInfo>  
  <orderInfo id="57" date="11/25/2000" totalPrice="509.7">  
    <productInfo name="Toolbox" unitPrice="17.99" quantity="20" />  
    <productInfo name="Toy Car Speedway Center" unitPrice="14.99" quantity="10"/>  
  </orderInfo>  
  <orderInfo id="58" date="11/25/2000" totalPrice="2909.79">  
    <productInfo name="Gyroscope" unitPrice="29.99" quantity="1" />  
    <productInfo name="Skooter" unitPrice="274.99" quantity="10" />  
    <productInfo name="Doll Baby" unitPrice="12.99" quantity="10" />  
  </orderInfo>  
  <orderInfo id="59" date="11/25/2000" totalPrice="29.95">  
    <productInfo name="Rubiks Cube" unitPrice="5.99" quantity="5" />  
  </orderInfo>  
</orderhistory>  
<orderHistory>...</orderHistory>  
.....  
</dataRoot>')
```

## Typed XML

Typed XML is an XML type data with an associated XML schema. The XML schema is used to validate data and perform type checks. To use XML schemas, an XML schema collection has to be defined first (XML schema collection will be introduced in the next section), and then specify the schema collection to the XML column. The typed XML data can be either an XML document or content fragment. The following example shows how to define a table, named `typedHistoryTable`, with a typed XML document as a table column:

```
CREATE TABLE typedHistoryTable (  
id varchar(5) PRIMARY KEY,  
orderHistory XML(DOCUMENT onlineShoppingCollection))
```

The typed XML column requires the XML instance to specify the target namespace of the XML schema used to validate the XML content as shown in the following statement:

```
INSERT INTO typedHistoryTable VALUES  
(1,  
'<dataRoot xmlns="http://xyz.com/onlineshopping/orderHistory">  
  <orderHistory>  
    <customerInfo>  
      <fName>first1</fName>  
      <lName>last1</lName>  
    </customerInfo>  
    <orderInfo id="57" date="11/25/2000">  
      <productInfo name="Toolbox" quantity="20" />  
      <productInfo name="Toy Car Speedway Center" quantity="10" />  
    </orderInfo>  
.....  
</dataRoot>')
```

Constraints can be added on untyped or typed XML columns except unique, primary key and foreign key constraints because XML instances are not comparable in SQL Server 2005 [Pal et al., 2004]. In order to validate an XML document, an XML query method or an XQuery is used to perform the constraint checking. The following example shows a constraint enforcing the product quantity in an `orderInfo` element has to be greater than 1. The usage of XML query methods and XQuery will be introduced in the section of Querying XML Data.

```
CREATE TABLE untypedHistoryTable2(
id varchar(5) PRIMARY KEY,
orderHistory XML
CONSTRAINT quantity_check CHECK
(orderHistory.value('orderHistory/orderInfo/product[@quantity]',int')>1))
```

## XML Schema Collection

An XML schema collection contains one or many XML schemas. Individual XML schemas within an XML schema collection are identified using their target namespace.

An XML schema collection is created using CREATE XML SCHEMA COLLECTION syntax.

The following example demonstrates adding the order history XML schema to the schema collection called onlineShoppingCollection:

```
CREATE XML SCHEMA COLLECTION onlineShoppingCollection AS
' <?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://xyz.com/onlineShopping/orderHistory"
xmlns="http://xyz.com/onlineShopping/orderHistory"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
<xs:element name="dataRoot">
<xs:complexType>
<xs:sequence>
<xs:element ref="orderhistory"/>
<xs:element ref="orderHistory"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="orderhistory">
<xs:complexType>
<xs:sequence>
<xs:element ref="customerInfo"/>
<xs:element maxOccurs="unbounded" ref="orderInfo"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="orderHistory">
<xs:complexType>
<xs:sequence>
<xs:element ref="customerInfo"/>
<xs:element maxOccurs="unbounded" ref="orderInfo"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="customerInfo">
<xs:complexType>
<xs:sequence>
<xs:element ref="fName"/>
<xs:element ref="lName"/>
```

```

    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="fName" type="xs:NCName"/>
<xs:element name="lName" type="xs:NCName"/>
<xs:element name="orderInfo">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="productInfo"/>
    </xs:sequence>
    <xs:attribute name="date" use="required"/>
    <xs:attribute name="id" use="required" type="xs:integer"/>
    <xs:attribute name="totalPrice" use="required" type="xs:decimal"/>
  </xs:complexType>
</xs:element>
<xs:element name="productInfo">
  <xs:complexType>
    <xs:attribute name="name" use="required"/>
    <xs:attribute name="quantity" use="required" type="xs:integer"/>
    <xs:attribute name="unitPrice" use="required" type="xs:decimal"/>
  </xs:complexType>
</xs:element>
</xs:schema>

```

The ALTER XML SCHEMA COLLECTION statement supports extending an XML schema in an XML schema collection with new schema components and registering new XML schemas. Suppose a customer information XML schema is also available, it can be added to the onlineShoppingCollection as follows:

```

ALTER XML SCHEMA COLLECTION onlineShoppingCollection ADD
'<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://xyz.com/onlineshopping/customerInfo"
xmlns="http://xyz.com/onlineshopping/customerInfo"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
<xs:element name="customerInformation">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="loginName"/>
      <xs:element ref="firstName"/>
      <xs:element ref="lastName"/>
      <xs:element ref="street"/>
      <xs:element ref="city"/>
      <xs:element ref="state"/>
      <xs:element ref="zip"/>
      <xs:element ref="phone"/>
      <xs:element ref="email"/>
      <xs:element ref="password"/>
    </xs:sequence>

```

```
</xs:complexType>
</xs:element>
<xs:element name="loginName" type="xs:string"/>
<xs:element name="firstName" type="xs:string"/>
<xs:element name="lastName" type="xs:string"/>
<xs:element name="street" type="xs:string"/>
<xs:element name="city" type="xs:string"/>
<xs:element name="state" type="xs:string"/>
<xs:element name="zip" type="xs:string"/>
<xs:element name="phone" type="xs:string"/>
<xs:element name="email" type="xs:string"/>
<xs:element name="password" type="xs:string"/>
</xs:schema>
```

## ***Querying XML Data***

XQuery is a language for finding and extracting (querying) data from XML documents. One powerful new feature in XQuery is the FLWOR expression [XQuery, 2004]. FLWOR is an acronym for For-Let-Where-Order by-Return, which are the clauses that make up the expressions.

XML documents can be represented as a tree view of nodes. XQuery includes XPath 2.0 [XPath, 2004] as its navigation language. XPath uses a pattern expression to identify nodes in an XML document. An XPath pattern is a slash-separated list of child element names that describe a path through the XML document. The pattern selects elements that match the path. Some basic XPath expressions include:

A slash (/) at the beginning of the expression represents an absolute path to an element. A double forward slash (//) selects from all descendants of the context node as well as the context node itself.

A double period (..) indicates the parent of the current node. Wildcards (\*) can be used to select unknown XML elements. All attributes are specified by the @ prefix.

In general an XPath expression may refer to more than one node. Each step in a location path may have a predicate that selects specific nodes from the returned node list. The predicate contains a boolean expression, which is tested for each node in the context node list. A predicate is defined using square brackets in an XPath expression. For example, /dataRoot/orderHistory [1] selects the first orderHistory element of the dataRoot element. As

another example, `//customerInfo[!Name='last1']` selects all `customerInfo` elements that have a nested `!Name` element with a value of 'last1'.

In SQL Server 2005, entire XML values or parts of XML instances can be retrieved using the methods associated with the XML data type: `query()`, `value()` and `exist()`. The methods take an XQuery expression as argument. Another method, `modify()`, allows modification of XML data and accepts an XML data modification statement as input.

The `query()` method is useful for extracting parts of an XML instance. The XQuery expression evaluates to a list of XML nodes. The sub-trees rooted at each of these nodes are returned in document order. The result type is untyped XML. The following example illustrates a query returning the first `orderHistory` element from the `untypedHistoryTable`:

```
SELECT id,orderHistory.query('/dataRoot/orderHistory[1]') FROM untypedHistoryTable
```

The `value()` method extracts a scalar value from an XML instance. It returns the value of the node the XQuery expression evaluates to. This value is converted to a T-SQL type specified as the second argument of the method. The following example shows how to get the last name of the customer from the first `orderHistory` element. In the `value()` method, the first argument is the path expression to the specific node. Another function `data()` is also used to extract scalar value from a node. The type value is specified as `nvarchar(20)` in the second `value()` argument. The return type is defined as `nvarchar(20)`.

```
SELECT id,orderHistory.value(
'data(/dataRoot/orderHistory/customerInfo/!Name)[1]','nvarchar(20)')
FROM untypedHistoryTable
```

The `exist()` method is useful for existential checks on an XML instance. It returns 1 if the XQuery expression evaluates to a non-null node list, otherwise it returns 0. The following query example uses the `exist()` method to return all `orderHistory` elements with the product quantity greater than 10.

```
SELECT id, orderHistory.query('/dataRoot/orderHistory')
FROM untypedHistoryTable
WHERE orderInfo.exist('/dataRoot/orderHistory/orderInfo/productInfo[@quantity>10]')=1
```

The modify() method permits modifying parts of an XML instance, such as adding or deleting sub-trees, or replacing scalar values. The following example illustrates the delete function of the modify() method. The example selects the first orderHistory element and assigns the select result to an XML variable, named @order, and then the @order variable calls the modify() method to delete its nested customerInfo element.

```
DECLARE @order xml
SET @order = (SELECT orderHistory.query('/dataRoot/orderHistory[1]') FROM untypedHistoryTable)

-- delete customerInfo element
SET @order.modify(' delete /orderHistory/customerInfo')
```

A subset of XQuery constructs are supported in SQL Server 2005, which include For (iteration over nodes), Where (node check), Return (returning values) and Order by (sorting). The following example demonstrates the use of a SQL Server 2005 FLWOR expression. The query returns the orderInfo elements of the customer whose last name is last1 from the untypedHistoryTable.

```
SELECT orderHistory.query ('For $o in //orderHistory
                           Where $o/customerInfo/IName='last1'
                           Return <result>{$o/orderInfo}</result>')
FROM untypedHistoryTable
```

The FLOWR expression is defined as the input parameter of the query() method. The For clause iterates over all orderHistory elements and binds each element to the variable \$o. The Where ensures that the value of IName element inside \$o is equal to last1. And then the query returns the element orderInfo and wrapped it by a new tag called <result>.

## ***FOR XML Extension***

In Microsoft SQL Server 2000, the FOR XML clause was introduced to the SELECT statement. This clause provides the ability to aggregate the relational rowset returned by the SELECT statement into XML. FOR XML supports three modes: RAW, AUTO, and EXPLICIT. OPENXML decomposes an XML input data to relational tables in SQL Server 2000. The XML composition and decomposition functionality in SQL Server 2005 have been improved. The major change to OPENXML is the support of the XML data type by extending the stored procedure, sp\_xml\_preparedocument, to take an XML type variable as

an input parameter. The existing FOR XML functionality has also been enhanced to support XML data type instances and other SQL Server types such as (n)varchar.

A new TYPE directive has been added to the FOR XML clause to generate the result as an XML type instance. For example:

```
SELECT * FROM customerInformation FOR XML AUTO, TYPE
```

returns the XML content as an XML data type instance, instead of the nvarchar instance that would have been the case without the TYPE directive.

Since FOR XML queries can return an XML type instance, the result of a FOR XML query can be assigned to an XML variable, or inserted into an XML column. As shown in the following example, a table called customerXml with an XML type column is first created. The SELECT ... FOR XML statement returns all tuples in the customerInformation table in XML and assigns the XML content to an XML type variable, called @customerInfo. The value of @customerInfo can be inserted into the customerXml table directly with the associated id.

```
CREATE TABLE customerXml(id varchar(5) PRIMARY KEY, xmlInfo XML)
```

```
DECLARE @customerInfo XML;  
SET @customerInfo = (SELECT * FROM customerInformation FOR XML AUTO, TYPE)  
INSERT INTO customerXml VALUES (1,@customerInfo)
```

A new PATH mode is added to the FOR XML clause allowing a developer to use an XPath-like syntax as a column name, which is mapped into an attribute (e.g., "@attributeName"), element (e.g., "elementName"), nested element structure ("element/sub-element"), element content ("\*"), text node ("text()"), or data value ("data()") in the resulting XML document. An example of the use of the PATH mode is shown as follows:

```
SELECT  
  firstName as '@fName',  
  lastName as '@lName',  
  loginName as 'login/name',  
  password as 'login/password',  
FROM customerInformation FOR XML PATH('customer'),ROOT('dataRoot')
```

In the above example, the PATH mode is used to specify the XML result structure: the firstName and lastName are mapped to attributes fName and lName of the customer element, the

loginName and password columns are enclosed within a login element, the rowset element name is specified as customer, and the root of the XML document is defined as dataRoot. The query result is shown as follows (only showing the first customer element):

```
<dataRoot>
  <customer fName="first1" lName="last1">
    <login>
      <name>11</name>
      <password>111111</password>
    </login>
  </customer>
  ...
</dataRoot>
```

SQL Server 2005 also provides the following new features of the FOR XML clause:

1. The RAW mode can be combined with the ELEMENTS directive, and take a parameter to overwrite the row element name. For example, the following SELECT statement returns an element-based XML representation of the customerInformation table using customer as the enclosing row element:

```
SELECT *
FROM customerInformation
WHERE lastName='last1'
FOR XML RAW('customer'), ELEMENTS
```

The query result is shown as follows:

```
<customer>
  <loginName>hello</loginName>
  <firstName>first1</firstName>
  <lastName>last1</lastName>
  <street>123 E. University</street>
  <city>123</city>
  <state>SC</state>
  <zip>dt</zip>
  <phone>123-456-789</phone>
  <email>123@asu.edu</email>
  <password>he</password>
</customer>
```

- The ELEMENTS directive provides an XSINIL option to map null values to an element with an attribute xsi:nil="true". For example, the following SELECT statement returns tuples that contain null in the column date from the orderInformation table.

```
SELECT *
FROM orderInformation
WHERE [date] is null
FOR XML AUTO,ELEMENTS XSINIL
```

Because of the XSINIL option in the FOR XML clause, columns having a null value are still presented in the resulting XML document. The query result is shown as follows:

```
<orderInformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <id>77</id>
  <type>C</type>
  <orderNo xsi:nil="true" />
  <date xsi:nil="true" />
  <totalPrice>3.3980000000000000e+001</totalPrice>
  <loginName>April</loginName>
</orderInformation>
...
<orderInformation>...</orderInformation>
```

- An XMLSCHEMA inference directive has been added for the RAW and AUTO modes that takes a target namespace URI as an optional argument to generate an inline XML schema associated with the XML document. The following SELECT statement generates an XML representation of the customerInformation table with the associated XML schema.

```
SELECT *
FROM customerInformation
WHERE lastName = 'last1'
FOR XML RAW('customer'), XMLSCHEMA('http://xyz.com/onlineShopping/customer')
```

The query result is shown as follows:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://schemas.microsoft.com/sqlserver/2004/sqltypes">
  <xsd:simpleType name="varchar">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
</xsd:schema>
<xsd:schema targetNamespace="http://xyz.com/onlineshopping/customer"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:sqltypes="http://schemas.microsoft.com/sqlserver/2004/sqltypes" elementFormDefault="qualified">
<xsd:import namespace="http://schemas.microsoft.com/sqlserver/2004/sqltypes"/>
...
</xsd:schema>
<customer xmlns="http://xyz.com/onlineshopping/customer" loginName="hello"
  firstName="first1" lastName="last1" street="123 E. University" city="123"
  state="SC" zip="dt" phone="123-456-789" email="123@asu.edu" password="he"/>
```

As shown in the above example, a XML schema is generated at the top of the result document and the rest is the data from the database table.

## **Summary**

This document describes new features of XML support in Microsoft SQL Server 2005, including XML native type, XQuery and FOR XML clause extensions of the SELECT statement. The features introduced are also applicable to the SQL Server 2005 Express version. All code is tested under SQL Server 2005 Beta 2 and SQL Server Express Beta 1 releases.

## REFERENCES

[Beauchemin, 2004] Bob Beauchemin, “*XML in Yukon: New Version Showcases Native XML Type and Advanced Data Handling*”,

<http://msdn.microsoft.com/msdnmag/issues/04/02/XMLinYukon>, 2004.

[Brown, 2004] Eric Brown, “*XML, T-SQL, and the CLR Create a New World of Database Programming*”,

<http://msdn.microsoft.com/msdnmag/issues/04/02/YukonBasics>, 2004

[Dietrich and Urban, 2005] Suzanne W. Dietrich and Susan D. Urban, *An Advanced Course in Database Systems: Beyond Relational Databases*, Upper Saddle River, NJ: Prentice Hall, 2005.

[Pal et al., 2004] Shankar Pal, Mark Fussell, Irwin Dolobowsky, “*XML Support in Microsoft SQL Server 2005*”,

<http://msdn.microsoft.com/xml/default.aspx?pull=/library/en-us/dnsq190/html/sql2k5xml.asp>, 2004.

[Rys, 2004] Michael Rys, “*What’s New in FOR XML in Microsoft SQL Server 2005*”,

<http://msdn.microsoft.com/library/en-us/dnsq190/html/forxml2k5.asp>, 2004.

[SQL, 2003] International Organization for Standardization, *ISO/IEC 9075-14:2003, Information technology-Database languages-SQL-Part 14: XML-Related Specifications (SQL/XML)*.

[XPath, 2004] World Wide Web Consortium, *W3C Working Draft: XML Path Language (XPath) 2.0*, <http://www.w3.org/TR/xpath20/>, 2004

[XQuery, 2004] World Wide Web Consortium, *W3C Working Draft: XQuery 1.0 and XPath 2.0 Formal Semantics*, <http://www.w3.org/TR/xquery-semantics>, 2004.