SQL 2005: Enter XML

SQL 2005: Enter XML

When SQL Server 2000 was released, it brought with it built-in T-SQL support for XML. SQL Server 2000 XML support is cumbersome to work with and more than a little kludgy to use.

SQL Server 2005 "kicks it up a notch" with a built-in `XML` data type, XPath and XQuery support, and streamlined improvements to legacy XML tools. This article is an introduction to the centerpiece of SQL Server 2005 XML support: the new `XML` data type.

XML Documents and Fragments

The SQL Server 2005 `XML` data type can hold well-formed XML documents or XML fragments. A well-formed XML document must satisfy the following criteria:

1. It must match the W3C "document" production,
2. It must contain at least one element,
3. It must have exactly one top-level element, known as the root element.

[This definition is from the W3C XML Standard 1.0 at http://www.w3.org/TR/REC-xml/#NT-document.]

An XML fragment is defined by the W3C to mean "part of an XML document..." XML fragments do not have to be well-formed. [See the W3C XML Fragment Interchange working draft at http://www.w3.org/1999/06/WD-xml-fragment-19990630.html for more information.]

For this example, I'm going to make one big assumption: that you're as big a fan of *Law & Order* as I am. (Even if you're not, the sample will still get the point across.) The sample in Listing 1 creates an `XML` variable and loads it with a well-formed XML document containing *L&O* episode summaries. It also loads a second `XML` variable with an XML fragment containing a short list of characters from the series. The XML is implicitly cast from `nvarchar` data to untyped `XML` instances in both cases.

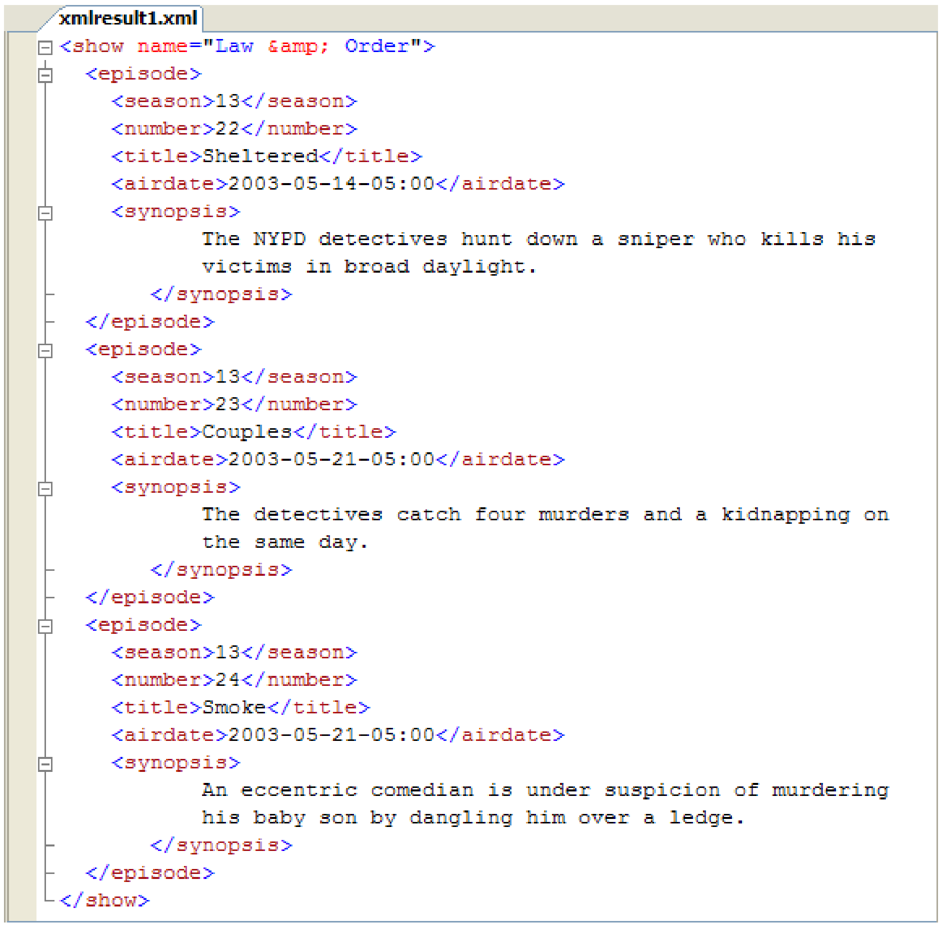
Listing 1. XML document vs. XML fragment

```
/* Will hold a well-formed XML document */
DECLARE @doc XML;

/* Populate the XML document */
SELECT @doc = N'<?XML version = "1.0" encoding = "UTF-16"?>
<show name = "Law &amp; Order">
   <episode>
       <season>13</season>
       <number>22</number>
       <title>Sheltered</title>
       <airdate>2003-05-14-05:00</airdate>
       <synopsis>
           The NYPD detectives hunt down a sniper who kills his
           victims in broad daylight.
       </synopsis>
   </episode>
   <episode>
       <season>13</season>
       <number>23</number>
       <title>Couples</title>
       <airdate>2003-05-21-05:00</airdate>
       <synopsis>
           The detectives catch four murders and a kidnapping on
           the same day.
       </synopsis>
   </episode>
   <episode>
       <season>13</season>
       <number>24</number>
       <title>Smoke</title>
       <airdate>2003-05-21-05:00</airdate>
       <synopsis>
           An eccentric comedian is under suspicion of murdering
           his baby son by dangling him over a ledge.
       </synopsis>
   </episode>
</show>';

/* Select the XML document */
SELECT @doc;

/* Will hold an XML fragment */
DECLARE @frag XML;

/* Populate the document fragment */
SELECT @frag = N'
   <role>
       <actor>Benjamin Bratt</actor>
       <character>Det. Ray Curtis</character>
       <seasons>6 through 9</seasons>
   </role>
   <role>
       <actor>Jerry Orbach</actor>
       <character>Det. Lennie Briscoe</character>
       <seasons>3 through 14</seasons>
   </role>
   <role>
       <actor>Sam Waterson</actor>
       <character>Exec. A.D.A. Jack McCoy</character>
       <seasons>5 through 17</seasons>
   </role>';

/* Select the XML fragment */
SELECT @frag;
```

NOTE: You might notice the `airdate` element in the well-formed XML document contains a date in the format "`2003-05-21`**`-05:00`**". This is a standard date format defined by ISO 8601. SQL Server's XML implementation requires that date and time data (`xs:date`, `xs:time`, `xs:dateTime` types) in a typed XML document must have a time zone attached (we'll discuss typed versus untyped XML later on). The timezone "Z" is the "Zero Meridian", or UTC time. In the example, the time zone "**`-05:00`**" is UTC minus 5 hours, or U.S. Eastern Standard Time.

The results of the sample in Listing 1 are shown in Figures 1 and 2 below.

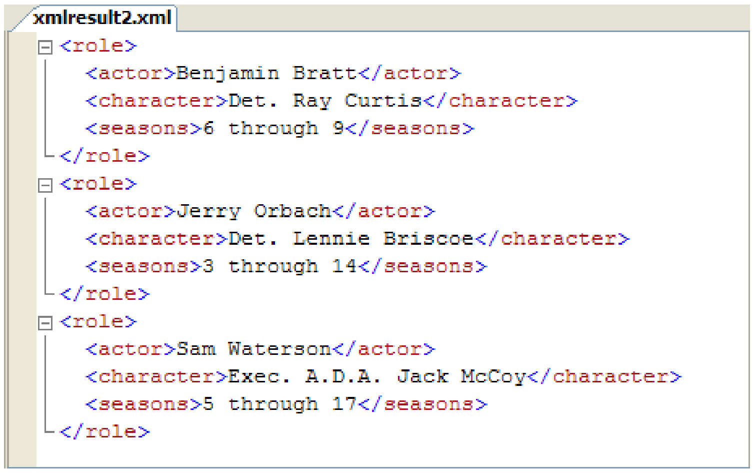
Figure 1. Law & Order Episodes in XML format



xmlresult1.xml

```
<show name="Law &amp; Order">
  <episode>
    <season>13</season>
    <number>22</number>
    <title>Sheltered</title>
    <airdate>2003-05-14-05:00</airdate>
    <synopsis>
           The NYPD detectives hunt down a sniper who kills his
           victims in broad daylight.
       </synopsis>
  </episode>
  <episode>
    <season>13</season>
    <number>23</number>
    <title>Couples</title>
    <airdate>2003-05-21-05:00</airdate>
    <synopsis>
           The detectives catch four murders and a kidnapping on
           the same day.
       </synopsis>
  </episode>
  <episode>
    <season>13</season>
    <number>24</number>
    <title>Smoke</title>
    <airdate>2003-05-21-05:00</airdate>
    <synopsis>
           An eccentric comedian is under suspicion of murdering
           his baby son by dangling him over a ledge.
       </synopsis>
  </episode>
</show>
```

Figure 2. The cast members in XML format



xmlresult2.xml

```
<role>
  <actor>Benjamin Bratt</actor>
  <character>Det. Ray Curtis</character>
  <seasons>6 through 9</seasons>
</role>
<role>
  <actor>Jerry Orbach</actor>
  <character>Det. Lennie Briscoe</character>
  <seasons>3 through 14</seasons>
</role>
<role>
  <actor>Sam Waterson</actor>
  <character>Exec. A.D.A. Jack McCoy</character>
  <seasons>5 through 17</seasons>
</role>
```

To and From XML

The `XML` data type can be used anywhere that other SQL data types are used. You can, for instance:

- declare an `XML` variable (as we did above),
- declare a column of `XML` type in a table,
- declare `XML` parameters for stored procedures and user-defined functions,
- define the return value of a scalar user-defined function as `XML`,
- implicitly or explicitly cast to and from XML, according to the following rules you can:
    - implicitly cast values from `char`, `varchar`, `nchar`, `nvarchar`, `binary`, `varbinary`, `ntext`, or `text` to the

XML data type.
  - explicitly cast XML values to char, varchar, nchar, nvarchar, binary, or varbinary data types using the CAST function.
  - explicitly cast an XML instance to another XML instance.
  - implicitly cast an untyped XML instance to another untyped XML instance, a typed XML instance to an untyped instance, or an untyped XML instance to typed instance.

Most of your conversions will probably be casts from varchar, nvarchar, and varbinary data to an XML instance, as we did in the sample code in Listing 1; but the other casting options are there when (and if) you need them.

If you cast from varchar to XML, the XML is encoded with one-byte characters. If you declare an XML encoding in the prolog of the XML, that encoding will be used. This example prolog declares a UTF-8 encoding for your varchar source:

```
<?XML encoding = "UTF-8" ?>
```

If not explicitly specified in the prolog, the code page of the source string will be used (if there is one associated). If you're casting nvarchar to XML, UTF-16 encoding is used. You can include an encoding in your source prolog like this (if you choose to):

```
<?XML encoding = "UTF-16" ?>
```

But it's not required.

If you're casting from varbinary your XML needs to contain the byte-order mark (BOM) or the UTF-16 encoding declaration (or both) if your data is Unicode. Otherwise you can use a UTF-8 encoding declaration or just leave the encoding off.

So What Is "Typed" XML?

We'll start this section by describing "untyped" XML. Untyped XML is standard XML; that is to say take any old data, entitize it (see below), slap some tags around it and voila -- instant XML (in reality I can only hope you put more planning than that into your *real* XML, but who's to say?) Untyped XML is not associated with an XML Schema, and as the name indicates it has no type information associated with it. The examples in Listing 1 were untyped XML.

NOTE: "Entitizing" your XML is the process of converting certain special characters to XML entities. Characters like "<", "&", etc., have special meaning in XML and have to be converted to entities like "&lt;" and "&gt;".

Typed XML is XML associated with an XML Schema collection. An XML Schema collection is a group of XML Schema definitions, each definition containing type information for the elements of an XML document. Typed XML provides advantages over untyped XML including:

- validation and typing of your XML data,
- more efficient querying of your XML data,
- the ability to perform data-type specific operations on your XML data, like math operations on numeric data.

One disadvantage is that you lose some of the flexibility of untyped XML, where you can stick data of any type in any element. Consider the following simple untyped XML document:

```
<numbers>
        <number>6.00</number>
        <number>six</number>
        <number>VI</number>
</numbers>
```

Of course, most client applications wouldn't be equipped to deal with alternating Arabic numerals, English words, and Roman numerals to represent numbers in an XML document. But, for those times when your business rules absolutely demand it, untyped XML can do the job. In Listing 2, we'll create an XML Schema collection for the well-formed XML document we created in Listing 1.

Listing 2. Creating an XML Schema collection

```
CREATE XML SCHEMA COLLECTION ShowSchema AS N'<?XML version="1.0" encoding="UTF-16"?>
<xs:schema xmlns="http://schemas.sqlservercentral.com/ShowSchema"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="show">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="episode" minOccurs="1" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="season" minOccurs="1" maxOccurs="1" type="xs:integer"/>
                            <xs:element name="number" minOccurs="1" maxOccurs="1" type="xs:integer"/>
                            <xs:element name="title" minOccurs="1" maxOccurs="1" type="xs:string"/>
                            <xs:element name="airdate" minOccurs="1" maxOccurs="1" type="xs:date"/>
                            <xs:element name="synopsis" minOccurs="1" maxOccurs="1" type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
</xs:schema>';
GO
```

A full description of the intricacies of XML Schema could fill an entire book, and is well beyond the scope of this introductory article. Suffice it to say the XML schema here is used to validate the elements of the document, ensuring that they:

- appear the correct number of times,
- are nested properly and occur in the proper sequence,
- have required attributes, and
- are of the proper type (integer, string, date, etc.)

NOTE: The XML Schema datatypes are described in detail on the W3C website, in the *XML Schema Part 2: Datatypes Second Edition* specification, at http://www.w3.org/TR/xmlschema-2/.

Once the XML Schema has been registered with the XML Schema collection in SQL Server you can use it to create typed XML instances. Listing 3 uses the XML document and XML Schema collection from the previous examples to create a typed XML instance.

Listing 3. Creating a typed XML instance

```
/* Will hold a well-formed XML document */
DECLARE @doc XML (DOCUMENT ShowSchema);

/* Populate the XML document */
SELECT @doc = N'<?XML version="1.0" encoding="UTF-16"?>
<show name = "Law &amp; Order">
<episode>
   <season>13</season>
   <number>22</number>
   <title>Sheltered</title>
   <airdate>2003-05-14-05:00</airdate>
   <synopsis>
       The NYPD detectives hunt down a sniper who kills his
       victims in broad daylight.
   </synopsis>
</episode>
<episode>
   <season>13</season>
   <number>23</number>
   <title>Couples</title>
   <airdate>2003-05-21-05:00</airdate>
   <synopsis>
       The detectives catch four murders and a kidnapping on
       the same day.
   </synopsis>
</episode>
<episode>
```

```
    <season>13</season>
    <number>24</number>
    <title>Smoke</title>
    <airdate>2003-05-21-05:00</airdate>
    <synopsis>
        An eccentric comedian is under suspicion of murdering his
        baby son by dangling him over a ledge.
    </synopsis>
</episode>
</show>';

/* Select the XML document */
SELECT @doc;
```

The typed XML document has to conform to the content rules specified in the XML Schema. Some of the content rules we specified in the example include:

- `show` is the root element of the XML document, and it has a required attribute name,
- the `show` element can contain one or more episode elements,
- each `episode` element must contain exactly one of each of the following elements, in this order:
    - `season` which must contain an `xs:integer` value,
    - `number` which must contain an `xs:integer` value,
    - `title` which contains an `xs:string` value,
    - `airdate` which must contain an `xs:date` value, and
    - `synopsis` which contains an `xs:string` value

An XML document that complies with an XML Schema is a "valid" XML document. You can change the XML document to make it invalid. Try changing the contents of a `number` element to a non-integer value or move some of the elements out of order. SQL Server will reject the XML if it does not conform to the XML Schema in the collection.

Facets

Observant readers might have noticed the following line in Listing 3:

```
DECLARE @doc XML (DOCUMENT ShowSchema);
```

This line declares an XML variable and associates it with the XML Schema collection we named `ShowSchema`. You might have also noticed the word `DOCUMENT` in parentheses right before the XML Schema collection name. The `DOCUMENT` keyword indicates a "facet", which tells SQL Server what type of XML the variable (or column) can hold. The available facets are `DOCUMENT`, which indicates that the XML instance must be a well-formed document (only one top-level, or root, element), and `CONTENT`, which indicates that the XML can be an XML fragment. The default facet is `CONTENT`.

Tune in Next Time

This article introduced the new SQL Server 2005 `XML` data type. In the next article we'll take a look at the `XML` data type's `query()` method and SQL Server 2005's XQuery support.