CHAPTER 7

Using XML

XML has been around for many years; with the release of Microsoft .NET technology, XML has become even more popular. Microsoft's development tools and technologies have built-in features to support XML. The advantages of using XML and its related technologies are major foundations of both the Internet and .NET.

Our goal in this chapter is to introduce you to the most essential XML concepts and terminology and the most basic techniques for using XML with SQL Server 2005. This will enable you to handle some common programming tasks while writing a software application.

In this chapter, we'll cover the following:

- Defining XML
- Why XML?
- Benefits of storing data as XML
- Understanding XML documents
- Understanding the XML declaration
- Converting relational data to XML
- How to store and retrieve XML documents using the `xml` data type

Defining XML

XML stands for eXtensible Markup Language. XML, which is derived from SGML (Standard Generalized Markup Language), is a metalanguage. A *metalanguage* isn't used for programming but rather for defining other languages, and the languages XML defines are known as *markup languages*. Markup is exactly what it implies: a means of "marking up" something. The XML document is in the form of a text document, and it can be read by both humans and computers.

■Note In essence, each XML document is an instance of a language defined by the XML elements used in the document. The specific language may or may not have been explicitly defined, but professional use of XML demands carefully planning one's XML *vocabulary* and specifying its definition in a *schema* that can be used to validate that documents adhere to both the syntax and semantics of a vocabulary. The XML Schema Definition language (usually referred to as XSD) is the language for defining XML vocabularies.

The World Wide Web Consortium (W3C) developed XML in 1996. Intended to support a wide variety of applications, XML was used by the W3C to create eXtensible HTML (XHTML), an XML vocabulary. Since 1996, the W3C has developed a variety of other XML-oriented technologies, including eXtensible Stylesheet Language (XSL), which provides the same kind of facility for XHTML that Cascading Style Sheets (CSS) does for HTML, and XSL Transformations (XSLT), which is a language for transforming XML documents into other XML documents.

Why XML?

XML is multipurpose, extensible data representation technology. XML increases the possibilities for applications to consume and manipulate data. XML data is different from relational data in that it can be structured, semistructured, or unstructured. XML support in SQL Server 2005 is fully integrated with the relational engine and query optimizer, allowing the retrieval and modification of XML data and even the conversion between XML and relational data representations.

Benefits of Storing Data As XML

XML is a platform-independent, data-representation format that offers certain benefits over a relational format for specific data representation requirements.

Storing data as XML offers many benefits, such as the following:

- Since XML is self-describing, applications can consume XML data without knowing the schema or structure. XML data is always arranged hierarchically in a tree structure form. XML tree structure must always have a root, or parent node, which is known as an *XML document*.

- XML maintains document ordering. Because XML is arranged in tree structure, maintaining node order becomes easy.

- XML Schema is used to define valid XML document structure.

- Because of XML's hierarchical structure, you can search inside the tree structures. XQuery and XPath are the query languages designed to search XML data.
- Data stored as XML is extensible. It is easy to manipulate XML data by inserting, modifying, and deleting nodes.

■**Note** Well-formed XML is an XML document that meets a set of constraints specified by the W3C recommendation for XML 1.0. For example, well-formed XML must contain a root-level element, and any other nested elements must open and close properly without intermixing.

SQL Server 2005 validates some of the constraints of well-formed XML. Some rules such as the requirement for a root-level element are not enforced. For a complete list of requirements for well-formed XML, refer to the W3C recommendations for XML 1.0 at `http://www.w3.org/TR/REC-xml`.

Understanding XML Documents

An XML document could be a physical file on a computer, a data stream over a network (in theory, formatted so a human could read it, but in practice, often in compressed binary form), or just a string in memory. It has to be complete in itself, however, and even without a schema, it must obey certain rules.

The most fundamental rule is that XML documents must be *well formed*. At its simplest, this means that overlapping elements aren't allowed, so you must close all *child* elements before the end tag of their *parent* element. For example, this XML document is well formed:

```
<states>
   <state>
       <name>Delaware</name>
      <city>Dover</city>
      <city>Wilmington</city>
   </state>
</states>
```

It has a *root* (or *document*) element, `states`, delimited by a start tag, `<states>`, and an end tag, `</states>`. The root element is the parent of the `state` element, which is in turn the parent of a `name` element and two `city` elements. An XML document can have only one root element.

Elements may have *attributes*. In the following example, `name` is used as an attribute with the `state` element:

```
<states>
   <state name="Delaware">
      <city>Dover</city>
      <city>Wilmington</city>
   </state>
</states>
```

This retains the same information as the earlier example, replacing the `name` element, which occurs only once, with a `name` attribute and changing the *content* of the original element (`Delaware`) into the *value* of the attribute (`"Delaware"`). An element may have any number of attributes, but it may not have duplicate attributes, so the `city` elements weren't candidates for replacement.

Elements may have content (text data or other elements), or they may be *empty*. For example, just for the sake of argument, if you want to keep track of how many states are in the document, you could use an empty element to do it:

```
<states>
   <controlinfo count="1"/>
   <state name="Delaware">
      <city>Dover</city>
      <city>Wilmington</city>
   </state>
</states>
```

The empty element, `controlinfo`, has one attribute, `count`, but no content. Note that it isn't delimited by start and end tags, but exists within an *empty element tag* (that starts with `<` and ends with `/>`).

An alternative syntax for empty elements, using start and end tags, is also valid:

```
<controlinfo count="1"></controlinfo>
```

Many programs that generate XML use this form.

■Note Though it's easy to design XML documents, designing them well is as much a challenge as designing a database. Many experienced XML designers disagree over the best use of attributes and even whether attributes should be used at all (and without attributes, empty elements have virtually no use). While elements may in some ways map more ideally to relational data, this doesn't mean that attributes have no place in XML design. After all, XML isn't intended to (and in principle can't) conform to the relational model of data. In fact, you'll see that a "pure" element-only design can be more difficult to work with in T-SQL.

Understanding the XML Declaration

In addition to elements and attributes, XML documents can have other parts, but most of them are important only if you really need to delve deeply into XML. Though it is optional, the *XML declaration* is one part that should be included in an XML document to precisely conform to the W3C recommendation. If used, it must occur before the root element in an XML document.

The XML declaration is similar in format to an element, but it has question marks immediately next to the angle brackets. It always has an attribute named `version`; currently, this has two possible values: `"1.0"` and `"1.1"`. (A couple other attributes are defined but aren't required.) So, the simplest form of an XML declaration is

```
<?xml version="1.0" ?>
```

XML has other aspects, but this is all you need to get started. In fact, this may be all you'll ever need to be quite effective. As you'll see, we don't use any XML declarations (or even more important things such as XML schemas and namespaces) for our XML documents, yet our small examples work well, are representative of fundamental XML processing, and could be scaled up to much larger XML documents.

Converting Relational Data to XML

A `SELECT` query returns results as a row set. You can optionally retrieve results of a SQL query as XML by specifying the `FOR XML` clause in the query. SQL Server 2005 enables you to extract relational data into XML form, by using the `FOR XML` clause in the `SELECT` statement. SQL Server 2005 extends the `FOR XML` capabilities, making it easier to represent complex hierarchical structures and add new keywords to modify the resulting XML structure.

■ **Note** In Chapter 13, we'll show how to extract data from a dataset, convert it into XML, and write it to a file with the dataset's `WriteXml` method.

The `FOR XML` clause converts result sets from a query into an XML structure, and it provides four modes of formatting:

- `FOR XML RAW`
- `FOR XML AUTO`

- `FOR XML PATH`
- `FOR XML EXPLICIT`

We'll use the first two in examples to show how to generate XML with a query.

Using FOR XML RAW

The `FOR XML RAW` mode transforms each row in the query result set into an XML element identified as `row` for each row displayed in the result set. Each column name in the `SELECT` statement is added as an attribute to the `row` element while displaying the result set.

By default, each column value in the row set that is not null is mapped to an attribute of the `row` element.

Try It Out: Using FOR XML RAW (Attribute Centric)

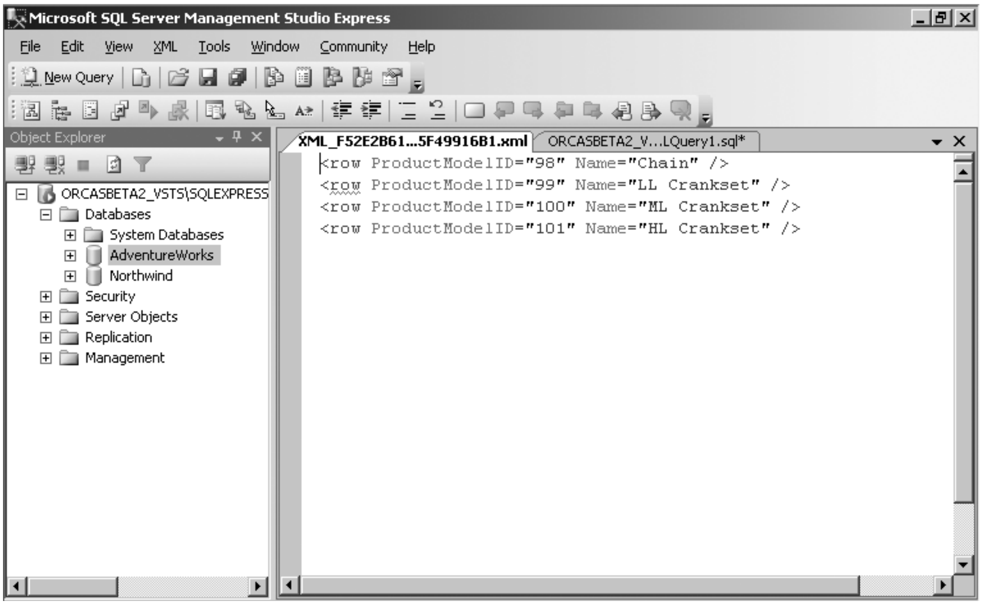
To use `FOR XML RAW` to transform returned rows into XML elements, follow these steps:

1. Open SQL Server Management Studio Express, and in the Connect to Server dialog box select <*ServerName*>\SQLEXPRESS as the server name and click Connect.

2. In Object Explorer, expand the Databases node, select the AdventureWorks database, and click the New Query button. Enter the following query and click Execute:

   ```
   SELECT ProductModelID, Name
   FROM Production.ProductModel
   WHERE ProductModelID  between 98 and 101
   FOR XML RAW
   ```

3. You will see a link in the results pane of the query window. Click the link, and you should see the results shown in Figure 7-1.


Microsoft SQL Server Management Studio Express
File Edit View XML Tools Window Community Help
New Query
Object Explorer
ORCASBETA2_VSTS\SQLEXPRESS
Databases
System Databases
AdventureWorks
Northwind
Security
Server Objects
Replication
Management
XML_F52E2B61...5F49916B1.xml
ORCASBETA2_V...LQuery1.sql*

```
<row ProductModelID="98" Name="Chain" />
<row ProductModelID="99" Name="LL Crankset" />
<row ProductModelID="100" Name="ML Crankset" />
<row ProductModelID="101" Name="HL Crankset" />
```

Figure 7-1. *Using* `FOR XML RAW`

How It Works

`FOR XML RAW` mode produces very “raw” XML. It turns each row in the result set into an XML `row` empty element and uses an attribute for each of the column values, using the alias names you specify in the query as the attribute names. It produces a string composed of all the elements.

`FOR XML RAW` mode doesn’t produce an XML document, since it has as many root elements (`raw`) as there are rows in the result set, and an XML document can have only one root element.

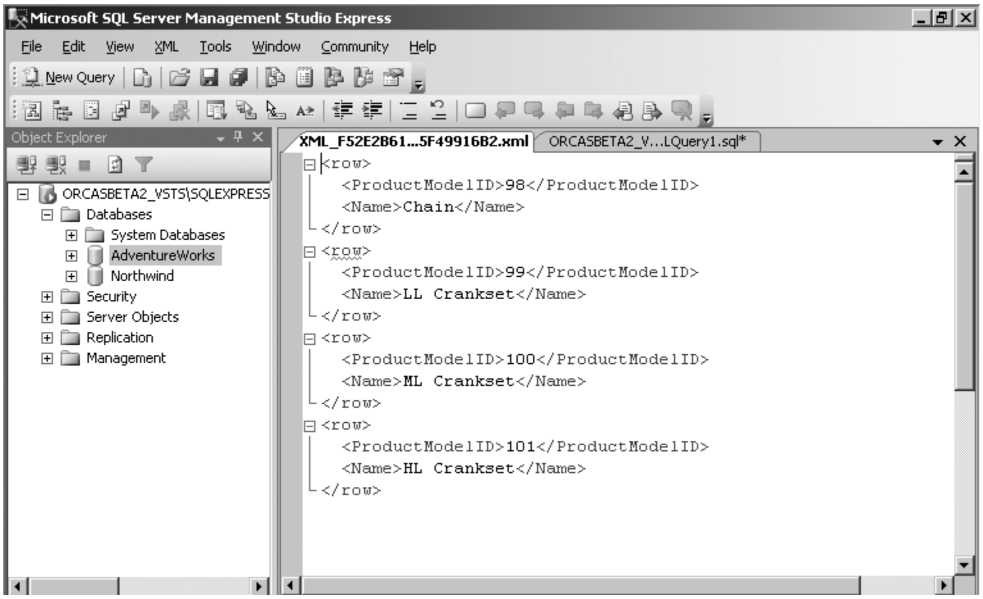
Try It Out: Using FOR XML RAW (Element Centric)

To change the formatting from attribute centric (as shown in the previous example) to element centric, which means that a new element will be created for each column, you need to add the `ELEMENTS` keyword after the `FOR XML RAW` clause as shown in the following example:

1. Replace the existing query in the query window with the following query and click Execute:

```
SELECT ProductModelID, Name
FROM Production.ProductModel
WHERE ProductModelID  between 98 and 101
FOR XML RAW,ELEMENTS
```

2. You will see a link in the results pane of the query window. Click the link, and you should see the results shown in Figure 7-2.


Microsoft SQL Server Management Studio Express
File Edit View XML Tools Window Community Help
New Query
Object Explorer
ORCASBETA2_VSTS\SQLEXPRESS
Databases
System Databases
AdventureWorks
Northwind
Security
Server Objects
Replication
Management
XML_F52E2B61...5F49916B2.xml
ORCASBETA2_V...LQuery1.sql*

```
<row>
  <ProductModelID>98</ProductModelID>
  <Name>Chain</Name>
</row>
<row>
  <ProductModelID>99</ProductModelID>
  <Name>LL Crankset</Name>
</row>
<row>
  <ProductModelID>100</ProductModelID>
  <Name>ML Crankset</Name>
</row>
<row>
  <ProductModelID>101</ProductModelID>
  <Name>HL Crankset</Name>
</row>
```

Figure 7-2. *Using* `FOR XML RAW ELEMENTS`

How It Works

`FOR XML RAW ELEMENTS` mode ode produces very "element-centric" XML. It turns each row in the result set where each column is converted into an attribute.

`FOR XML RAW ELEMENTS` mode also doesn't produce an XML document, since it has as many root elements (`raw`) as there are rows in the result set, and an XML document can have only one root element.

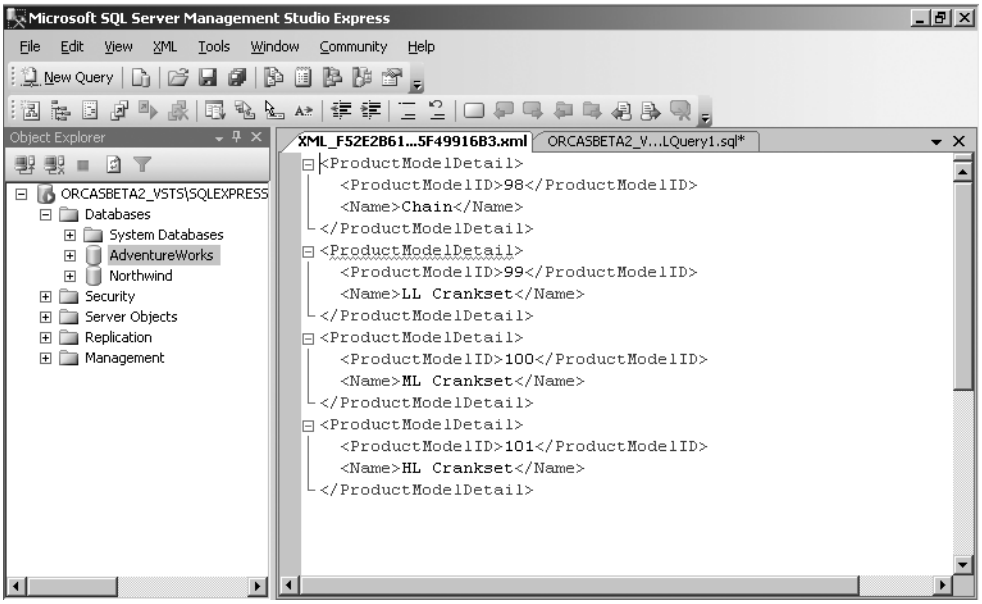
Try It Out: Renaming the row Element

For each row in the result set, the `FOR XML RAW` mode generates a `row` element. You can optionally specify another name for this element by including an optional argument in the `FOR XML RAW` mode, as shown in the following example. To achieve this, you need to add an alias after the `FOR XML RAW` clause, which you'll do now.

1. Replace the existing query in the query window with the following query, and click Execute.

```
SELECT ProductModelID, Name
FROM Production.ProductModel
WHERE ProductModelID  between 98 and 101
FOR XML RAW ('ProductModelDetail'),ELEMENTS
```

2. You will see a link in the results pane of the query window. Click the link, and you should see the results shown in Figure 7-3.


Microsoft SQL Server Management Studio Express
File Edit View XML Tools Window Community Help
New Query
Object Explorer
ORCASBETA2_VSTS\SQLEXPRESS
Databases
System Databases
AdventureWorks
Northwind
Security
Server Objects
Replication
Management
XML_F52E2B61...5F49916B3.xml
ORCASBETA2_V...LQuery1.sql*

```
<ProductModelDetail>
  <ProductModelID>98</ProductModelID>
  <Name>Chain</Name>
</ProductModelDetail>
<ProductModelDetail>
  <ProductModelID>99</ProductModelID>
  <Name>LL Crankset</Name>
</ProductModelDetail>
<ProductModelDetail>
  <ProductModelID>100</ProductModelID>
  <Name>ML Crankset</Name>
</ProductModelDetail>
<ProductModelDetail>
  <ProductModelID>101</ProductModelID>
  <Name>HL Crankset</Name>
</ProductModelDetail>
```

Figure 7-3. *Renaming the* `row` *element*

How It Works

`FOR XML RAW ('alias')` mode produces output where the `row` element is renamed to the alias specified in the query.

Because the `ELEMENTS` directive is added in the query, the result is element centric, and this is why the `row` element is renamed with the alias specified. If you don't add the `ELEMENTS` keyword in the query, the output will be attribute centric, and the `row` element will be renamed to the alias specified in the query.

Observations About FOR XML RAW Formatting

`FOR XML RAW` does not provide a root node, and this is why the XML structure is not a well-formed XML document.

`FOR XML RAW` supports attribute- and element-centric formatting, which means that all the columns must be formatted in the same way. Hence it is not possible to have the XML structure returned with both the XML attributes and XML elements.

`FOR XML RAW` generates a hierarchy in which all the elements in the XML structure are at the same level.

Using FOR XML AUTO

`FOR XML AUTO` mode returns query results as nested XML elements. This does not provide much control over the shape of the XML generated from a query result. `FOR XML AUTO` mode queries are useful if you want to generate simple hierarchies.

Each table in the `FROM` clause, from which at least one column is listed in the `SELECT` clause, is represented as an XML element. The columns listed in the `SELECT` clause are mapped to attributes or subelements.

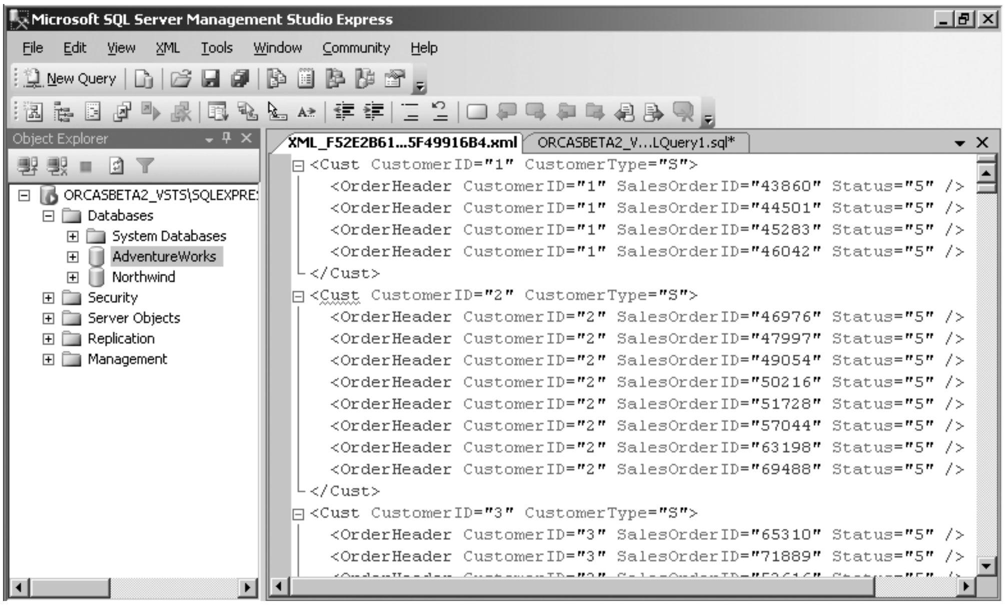
Try It Out: Using FOR XML AUTO

To see how to use `FOR XML AUTO` to format query results as nested XML elements, follow these steps:

1. Replace the existing query in the query window with the following query and click Execute:

```
SELECT Cust.CustomerID,
OrderHeader.CustomerID,
OrderHeader.SalesOrderID,
OrderHeader.Status,
Cust.CustomerType
FROM Sales.Customer Cust, Sales.SalesOrderHeader
OrderHeader
WHERE Cust.CustomerID = OrderHeader.CustomerID
ORDER BY Cust.CustomerID
FOR XML AUTO
```

2. You will see a link in the results pane of the query window. Click the link, and you should see the results shown in Figure 7-4.


Microsoft SQL Server Management Studio Express
File Edit View XML Tools Window Community Help
New Query
Object Explorer
ORCASBETA2_VSTS\SQLEXPRE
Databases
System Databases
AdventureWorks
Northwind
Security
Server Objects
Replication
Management
XML_F52E2B61...5F49916B4.xml
ORCASBETA2_V...LQuery1.sql*

```
<Cust CustomerID="1" CustomerType="S">
  <OrderHeader CustomerID="1" SalesOrderID="43860" Status="5" />
  <OrderHeader CustomerID="1" SalesOrderID="44501" Status="5" />
  <OrderHeader CustomerID="1" SalesOrderID="45283" Status="5" />
  <OrderHeader CustomerID="1" SalesOrderID="46042" Status="5" />
</Cust>
<Cust CustomerID="2" CustomerType="S">
  <OrderHeader CustomerID="2" SalesOrderID="46976" Status="5" />
  <OrderHeader CustomerID="2" SalesOrderID="47997" Status="5" />
  <OrderHeader CustomerID="2" SalesOrderID="49054" Status="5" />
  <OrderHeader CustomerID="2" SalesOrderID="50216" Status="5" />
  <OrderHeader CustomerID="2" SalesOrderID="51728" Status="5" />
  <OrderHeader CustomerID="2" SalesOrderID="57044" Status="5" />
  <OrderHeader CustomerID="2" SalesOrderID="63198" Status="5" />
  <OrderHeader CustomerID="2" SalesOrderID="69488" Status="5" />
</Cust>
<Cust CustomerID="3" CustomerType="S">
  <OrderHeader CustomerID="3" SalesOrderID="65310" Status="5" />
  <OrderHeader CustomerID="3" SalesOrderID="71889" Status="5" />
```

Figure 7-4. *Using* `FOR XML AUTO`

How It Works

The CustomerID references the Cust table. Therefore, a `Cust` element is created and `CustomerID` is added as its attribute.

Next, three columns, OrderHeader.CustomerID, OrderHeader.SaleOrderID, and OrderHeader.Status, reference the OrderHeader table. Therefore, an `OrderHeader` element is added as a subelement of the `Cust` element, and the three columns are added as attributes of `OrderHeader`.

Next, the Cust.CustomerType column again references the Cust table that was already identified by the Cust.CustomerID column. Therefore, no new element is created. Instead, the `CustomerType` attribute is added to the `Cust` element that was previously created.

The query specifies aliases for the table names. These aliases appear as corresponding element names. `ORDER BY` is required to group all children under one parent.

Observations About FOR XML AUTO Formatting

`FOR XML AUTO` does not provide a root node, and this is why the XML structure is not a well-formed XML document.

FOR XML AUTO supports attribute- and element-centric formatting, which means that all the columns must be formatted in the same way. Hence it is not possible to have the XML structure returned with both the XML attributes and XML elements.

FOR XML AUTO does not provide a renaming mechanism the way FOR XML RAW does. However, FOR XML AUTO uses table and column names and aliases if present.

Using the xml Data Type

SQL Server 2005 has a new data type, `xml`, that is designed not only for holding XML documents (which are essentially character strings and can be stored in any character column big enough to hold them), but also for processing XML documents. When we discussed parsing an XML document into a DOM tree, we didn't mention that once it's parsed, the XML document can be updated. You can change element contents and attribute values, and you can add and remove element occurrences to and from the hierarchy.

We won't update XML documents here, but the `xml` data type provides methods to do it. It is a very different kind of SQL Server data type, and describing how to exploit it would take a book of its own—maybe more than one. Our focus here will be on what every database programmer needs to know: how to use the `xml` type to store and retrieve XML documents.

■ **Note** There are so many ways to process XML documents (even in ADO.NET and with SQLXML, a support package for SQL Server 2000) that only time will tell if incorporating such features into a SQL Server data type was worth the effort. Because XML is such an important technology, being able to process XML documents purely in T-SQL does offer many possibilities, but right now it's unclear how much more about the `xml` data type you'll ever need to know. At any rate, this chapter will give you what you need to know to start experimenting with it.

Try It Out: Creating a Table to Store XML

To create a table to hold XML documents, replace the existing query in the query window with the following query and click Execute:

```
create table xmltest
(
   xid  int not null primary key,
   xdoc xml not null
)
```

How It Works

This works in the same way as a `CREATE TABLE` statement is expected to work. Though we've said the `xml` data type is different from other SQL Server data types, columns of `xml` type are defined just like any other columns.

Note The `xml` data type cannot be used in primary keys.

Now, you'll insert your XML documents into `xmltest` and query it to see that they were stored.

Try It Out: Storing and Retrieving XML Documents

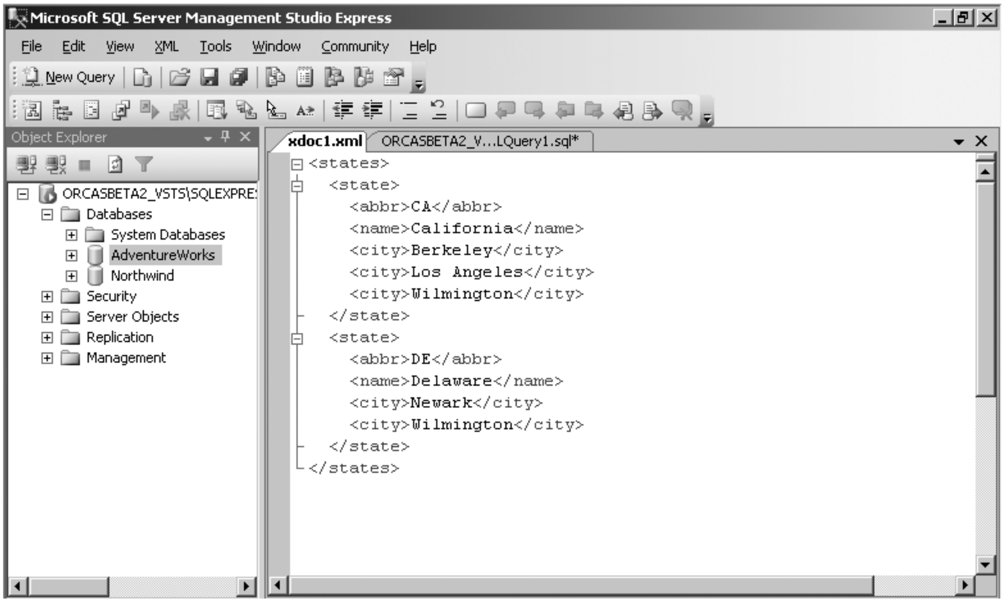
To insert your XML documents, follow these steps:

1. Replace the code in the SQL query window with the following two `INSERT` statements:

```
insert into xmltest
values(
1,
'
<states>
   <state>
      <abbr>CA</abbr>
      <name>California</name>
      <city>Berkeley</city>
      <city>Los Angeles</city>
      <city>Wilmington</city>
   </state>
   <state>
      <abbr>DE</abbr>
      <name>Delaware</name>
      <city>Newark</city>
      <city>Wilmington</city>
   </state>
</states>
'
)
```

```
insert into xmltest
values(
2,
'
<states>
   <state abbr="CA" name="California">
      <city name="Berkeley"/>
      <city name="Los Angeles"/>
      <city name="Wilmington"/>
   </state>
   <state abbr="DE" name="Delaware">
      <city name="Newark"/>
      <city name="Wilmington"/>
   </state>
</states>
'
)
```

2. Run the two `INSERT` statements by clicking Execute, and then display the table with `select * from xmltest`. You see the two rows displayed. Click the xdoc column in the first row, and you should see the XML shown in Figure 7-5.


Microsoft SQL Server Management Studio Express
File Edit View XML Tools Window Community Help
New Query
Object Explorer
ORCASBETA2_VSTS\SQLEXPRE:
Databases
System Databases
AdventureWorks
Northwind
Security
Server Objects
Replication
Management
xdoc1.xml
ORCASBETA2_V...LQuery1.sql*

```
<states>
  <state>
    <abbr>CA</abbr>
    <name>California</name>
    <city>Berkeley</city>
    <city>Los Angeles</city>
    <city>Wilmington</city>
  </state>
  <state>
    <abbr>DE</abbr>
    <name>Delaware</name>
    <city>Newark</city>
    <city>Wilmington</city>
  </state>
</states>
```

Figure 7-5. *Viewing an XML document*

How It Works

This works the same way all `INSERT`s work. You simply provide the primary keys as integers and the XML documents as strings. The query works just as expected, too.

Summary

This chapter covered the fundamentals of XML that every C# programmer needs to know. It also showed you how to use the most frequently used T-SQL features for extracting XML from tables and querying XML documents like tables. Finally, we discussed the `xml` data type and gave you some practice using it.

How much more you need to know about XML or T-SQL and ADO.NET facilities for using XML documents depends on what you need to do. As for many developers, this chapter may be all you ever really need to know and understand. If you do more sophisticated XML processing, you now have a strong foundation for experimenting on your own.

In the next chapter, you will learn about database transactions.