**Captura de XML fuera de la BD**

Las empresas pueden mantener los datos en un sistema relacional y sacar dichos datos en formato XML cuando sea preciso.

**FOR XML**

Es la forma más fácil de coger datos en un formato relacional desde SQL Server.

SELECT column list

FROM table list

WHERE filter criteria

FOR XML RAW | AUTO | EXPLICIT [, XMLDATA] [, ELEMENTS]

[, BINARY BASE64]

**Formas de uso**

Se pueden usar los siguientes modos:

RAW

AUTO

EXPLICIT

Los modos RAW y Auto son los más empleados. La razón de no usarse tanto EXPLICIT es que con los otros modos se cubren normalmente las necesidades.

**Modo RAW**

Devuelve las columnas como atributos y las filas como elementos.

Ejemplo:

USE pubs

GO

SELECT \* FROM Authors FOR XML RAW

Las columnas sin nulos tienen un atributo basado en el elemento.

Si recuperamos datos binarios, necesitaremos especificar BINARY BASE64.

SI RECUPERAMOS UN Schema de datos XML con el XML devuelto, especificaremos XMLDATA.

**Modo AUTO**

La consulta es la misma que en el modo RAW excepto que el nombre del elemento representa la fila como el nombre de la tabla. Ejemplo:

USE pubs

GO

SELECT \* FROM authors FOR XML AUTO

El anidado de los elementos depende del orden en la cláusula SELECT.

No se puede usar GROUP BY, pero podemos usar ORDER BY. La alternativa para un GROUP BY es usar un mandato SELECT anidado para lograr los resultados que queremos, aunque esto tiene algunas implicaciones en el rendimiento.

Al usar JOIN’s, veremos que AUTO anidará los resultados, que probablemente es lo que queremos que ocurra. En el caso de que no lo queramos, necesitaremos usar el modo EXPLICIT.

Ejemplo:

USE pubs

GO

SELECT Publishers.Pub\_Name, Titles.Title, Titles.Price

FROM Titles, Publishers WHERE Publishers.Pub\_ID = Titles.Pub\_ID

FOR XML AUTO

Podemos usar la opción ELEMENTS con FOR XML AUTO.

**Modo EXPLICIT**

Permite controlar completamente la forma en que el XML es generado.

Para ello se usa una tabla universal que describe nuestro documento XML.

Esta tabla consta de una columna para cada valor que queremos que se devuelva así como dos etiquetas adicionales:

* Una que identifica las etiquetas en el XML
* Otra que identifica las relaciones madre-hija.

**Ejemplo de tabla universal**

|  |  |  |  |
| --- | --- | --- | --- |
| Tag | Parent | Directiva columna 1 | Directiva columna 2 |
| 1 | NULL | Valor del dato | Valor del dato |
| 2 | 1 | Valor del dato | Valor del dato |
| 3 | 2 | Valor del dato | Valor del dato |

Se usan las directivas para describir cómo visualizar los datos en la tabla.

Las directivas son comandos especiales que usamos para que SQL Server entienda cómo analizar (parse).

El formato de las directivas es:

Elemento | tag | Atributo | Directiva

Imaginemos que queremos visualizar autores, y deseamos que la columna au\_id sea un atributo y que la columna au\_lname sea un elemento.

SELECT 1 AS Tag, NULL AS Parent,

au\_id AS [Authors!1!au\_id], au\_lname as [Authors!1]

FROM authors FOR XML EXPLICIT

Como se ve, tenemos dos columnas con tag y parent. Necesitamos esas columnas para identificar el tag del elemento actual, que es un entero de 1 a 255, y también el padre del elemento.

En este ejemplo no anidamos nuestros datos (nuestro padre es siempre NULL y nuestro tag es siempre 1), ya que siempre nos referimos al mismo padre.

Usamos la cláusula AS para renombrar nuestros datos para describir el formato XML que queremos hacer.

La nomenclatura de au\_id le dice a SQL Server que queremos usar el elemento authors, un ID de tag 1, y el nombre de nuestro atributo.

Ya que queremos que el resto de los datos sean elementos, los renombramos para que sean el elemento y el nombre del tag.

Si queremos hacer que au\_lname sea devuelto como elemento contenido en sus etiquetas, deberíamos retocar la SELECT anterior:

SELECT 1 as Tag, NULL as Parent, au\_id as [Authors!1!au\_id],

au\_lname as [Authors!1!au\_lname!element]

FROM Authors FOR XML EXPLICIT

**Tabla de directivas para FOR XML EXPLICIT**

|  |  |
| --- | --- |
| **Nombre** | **Descripción** |
| Cdata | Envuelve los datos en esta sección |
| element | Especifica lo que queremos que el elemento decodifique (por ejemplo &gt;) |
| elementxsinil | Si queremos generar elementos generados por valores NULL’s, podemos especificar esta directiva. Creará un elemento con un atributo xsi:nil=TRUE |
| ID | Permite especificar un ID para el elemento. Todas las directivas ID requieren que XMLDATA esté en la cláusula de FOR XML |
| IDREF | Permite a los atributos especificar atributos ID para hacer links dentro del documento |
| IDREFS | Similar a IDREF. Usa estructura para hacerlo. |
| Hide | Oculta el resultado del XML renderizado |
| Xml | Igual que la directiva element, pero no decodifica. |
| Xmltext | Interesante para recuperación de columnas que se añaden al documento. |

Otro ejemplo más complejo:

Supongamos que queremos obtener todos los autores con sus títulos y royalties. Deberemos usar una consulta UNION ALL que combine los datos de tablas dispares y necesitaremos anidar los resultados para que nuestra jerarquía XML aparezca correctamente con authors, royalties, y después titles.

Observemos que definimos un número de relaciones padre-hijo usando las columnas Tag y parent.

Otra observación: usamos ORDER BY para estar seguros que los tags parent estén en el XML antes de que procesemos los tag children.

SELECT 1 AS Tag, NULL AS Parent,

Authors.au\_fname AS [Authors!1!au\_fname!element],

Authors.au\_lname AS [Authors!1!au\_lname!element],

NULL AS [Titleauthor!2!Royaltyper],

NULL AS [Titles!3!Title!element]

FROM Authors

UNION ALL

SELECT 2 AS Tag, 1 AS Parent,

au\_fname,

au\_lname,

royaltyper,

NULL

FROM Authors INNER JOIN Titleauthor

ON Authors.au\_id = Titleauthor.au\_id

UNION ALL

SELECT 3 AS Tag, 2 AS Parent,

au\_fname,

au\_lname,

royaltyper,

title

FROM Authors INNER JOIN Titleauthor ON Authors.au\_id = Titleauthor.au\_id

INNER JOIN Titles ON Titles.title\_id = Titleauthor.title\_id

ORDER BY [Authors!1!au\_fname!element], [Authors!1!au\_lname!element],

[Titleauthor!2!royaltyper], Tag

FOR XML EXPLICIT

**Tabla universal truncada**

|  |  |  |  |
| --- | --- | --- | --- |
| Tag | Parent | Authors!1!au\_fname!element | !au\_lnameelement |
| 1 | NULL | Abraham | Bennet |
| 2 | 1 | Abraham | Bennet |
| 3 | 2 | Abraham | Bennet |
| 1 | NULL | Akiko | Yokomoto |
| 2 | 1 | Akiko | Yokomoto |
| 3 | 2 | Akiko | Yokomoto |
| 1 | NULL | Albert | Ringer |
| 2 | 1 | Albert | Ringer |
| 3 | 2 | Albert | Ringer |
| 2 | 1 | Albert | Ringer |
| 3 | 2 | Albert | Ringer |

Devolución de valores como tipos XML

Con el fin de dar soporte a los valores devueltos usando el tipo XML, FOR XML suministra una directiva extra denominada TYPE.

Pasando esta directiva a su llamada, en lugar de generar el XML y devolverlo como texto, SQL Server devuelve el resultado como un tipo de dato XML.

De esta forma podremos usar XQUERY sobre el valor devuelto para consultar información dentro del conjunto de resultados.

También podemos asignar el resultado a una variable o insertarlo dentro de una columna de tipo XML.

Podemos también anidar los mandatos FOR XML para generar una jerarquía, en lugar de tener que usar XML EXPLICIT.

El código siguiente muestra cómo usar la nueva directiva TYPE y después pasar el resultado a un XQUERY.

SELECT (SELECT \* FROM authors FOR XML AUTO, ELEMENTS, TYPE).query('count(/authors)')

Especificando la localización de las columnas en la jerarquía

El modo PATH permite especificar dónde aparecerá el valor de una columna en la jerarquía XML usando XPATH.

Mostramos a continuación un ejemplo de su uso. Renombramos el elemento root a AuthorsNew y creamos un nuevo tipo complejo denominado NAMES, que almacena el nombre y el apellido:

SELECT au\_fname "Names/FirstName", au\_lname "Names/LastName"

FROM authors FOR XML PATH('AuthorsNew')

Generando un XSD en línea

Otra mejora es el soporte en línea XSD en los modos RAW y AUTO.

Podemos opcionalmente poner la directiva XMLSCHEMA en nuestro código.

El siguiente ejemplo muestra el uso de esta directiva y los resultados devueltos desde el servidor:

<xsd:schema targetNamespace="urn:example.com" 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" schemaLocation="http://schemas.microsoft.com/sqlserver/2004/sqltypes/sqltypes.xsd" />

<xsd:element name="Authors">

<xsd:complexType>

<xsd:attribute name="au\_id" use="required">

<xsd:simpleType sqltypes:sqlTypeAlias="[pubs].[dbo].[id]">

<xsd:restriction base="sqltypes:varchar" sqltypes:localeId="1033" sqltypes:sqlCompareOptions="IgnoreCase IgnoreKanaType IgnoreWidth" sqltypes:sqlSortId="52">

<xsd:maxLength value="11" />

</xsd:restriction>

</xsd:simpleType>

</xsd:attribute>

<xsd:attribute name="au\_lname" use="required">

<xsd:simpleType>

<xsd:restriction base="sqltypes:varchar" sqltypes:localeId="1033" sqltypes:sqlCompareOptions="IgnoreCase IgnoreKanaType IgnoreWidth" sqltypes:sqlSortId="52">

<xsd:maxLength value="40" />

…

<Authors xmlns="urn:example.com" au\_id="172-32-1176" au\_lname="White" au\_fname="Johnson" phone="408 496-7223" address="10932 Bigge Rd." city="Menlo Park" state="CA" zip="94025" contract="1" />

<Authors xmlns="urn:example.com" au\_id="213-46-8915" au\_lname="Green" au\_fname="Marjorie" phone="415 986-7020" address="309 63rd St. #411" city="Oakland" state="CA" zip="94618" contract="1" />

<Authors xmlns="urn:example.com" au\_id="238-95-7766" au\_lname="Carson" au\_fname="Cheryl" phone="415 548-7723" address="589 Darwin Ln." city="Berkeley" state="CA" zip="94705" contract="1" />

<Authors xmlns="urn:example.com" au\_id="267-41-2394" au\_lname="O'Leary" au\_fname="Michael" phone="408 286-2428" address="22 Cleveland Av. #14" city="San Jose" state="CA" zip="95128" contract="1" />

…

<Authors xmlns="urn:example.com" au\_id="899-46-2035" au\_lname="Ringer" au\_fname="Anne" phone="801 826-0752" address="67 Seventh Av." city="Salt Lake City" state="UT" zip="84152" contract="1" />

<Authors xmlns="urn:example.com" au\_id="998-72-3567" au\_lname="Ringer" au\_fname="Albert" phone="801 826-0752" address="67 Seventh Av." city="Salt Lake City" state="UT" zip="84152" contract="1" />

Usando FOR XML desde el lado del cliente

Hasta ahora hemos escrito nuestro código FOR XML para que este sea procesado en el servidor.

Sin embargo, con SQLXML podemos procesar el código FOR XML desde el lado del cliente.

En lugar de enviar los resultados formateados a XML desde el servidor, SQL Server puede enviar los conjuntos de filas a SQLXML, y SQLXML formateará los resultados en el lado del cliente.

Su utilización la veremos en la programación de SQLXML desde .NET

**Plantillas para mejorar el rendimiento**

Podemos usar plantillas para ejecutar consultas contra SQL Server con SQLXML.

Estas plantillas pueden usar consultas SQL o XPath. Necesitamos usar el esquema anotado que creamos para la vista XML con la plantilla. El esquema lo podemos obtener en línea o vía fichero.

Para especificar la plantilla, creamos un fichero que usa el espacio de *urn:schemas-microsoft-com:xml-sql.*

Después podemos introducir nuestras consultas SQL o XPath en la plantilla. SQLXML cahceará las plantillas con el fin de mejorar el rendimiento.

La plantilla siguiente ejecuta una consulta SQL.

<Root><sql:query xmlns:sql=""urn:schemas-microsoft-com:xml-sql"">

SELECT \* FROM Authors FOR XML AUTO</sql:query></Root>

Para usar una consulta XPath, cambiaremos la sintaxis sql:query a sql:xpath-query. El siguiente ejemplo consulta todos los autores:

<Root><sql:xpath-query xmlns:sql=""urn:schemas-microsoft-com:xml-sql"">/Authors

</sql:xpath-query></Root>

Su utilización la veremos en la programación de SQLXML desde .NET

**Trabajando con datos XML**

Las colecciones de esquema XSD en SQL Server 2005 permitían a los usuarios validar datos XML para reforzar la conformidad de los datos XML con una definición de esquema.

Un esquema describe los elementos y atributos permitidos para una estructura de datos XML particular.

La implementación en SQL server 2005 no era una implementación completa de la especificación XML. Era un subconjunto que cubría los escenarios de validación más comunes de XML.

SQL Server 2008 se basa en la implementación anterior pero incluye utilidades adicionales como validación de esquemas, soporte date/time y tipos unión y list.

**Validación para tipos “Any”**

En SQL Server 2005, los usuarios tenían la posibilidad de definir esquemas XSD. Un esquema define las reglas para un documento XML específico, como el tipo de elementos y atributos que pueden aparecer en el documento, el número y el orden de los elementos hijo, etc.

En un esquema XSD podemos definir declaraciones de metacaracteres mediante las palabras clave any, anyAttribute, y anyType.

Esto da al diseñador del esquema la flexibilidad para de forma arbitraria añadir información en línea, sin definirla en tiempo de diseño.

En el siguiente ejemplo, creamos un esquema dentro de un objeto SCHEMA COLLECTION:

CREATE XML SCHEMA COLLECTION EmployeeSC AS

N'<?xml version="1.0"?>

<xsd:schema xmlns="http://schemas.apress.com/AcceleratedSQLServer2008"

xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="Employee">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Name" type="xsd:string"/>

<xsd:element name="DateOfBirth" type="xsd:date"/>

<xsd:element name="Salary" type="xsd:long"/>

<xsd:any namespace="##other" **processContents**="skip" minOccurs="0"

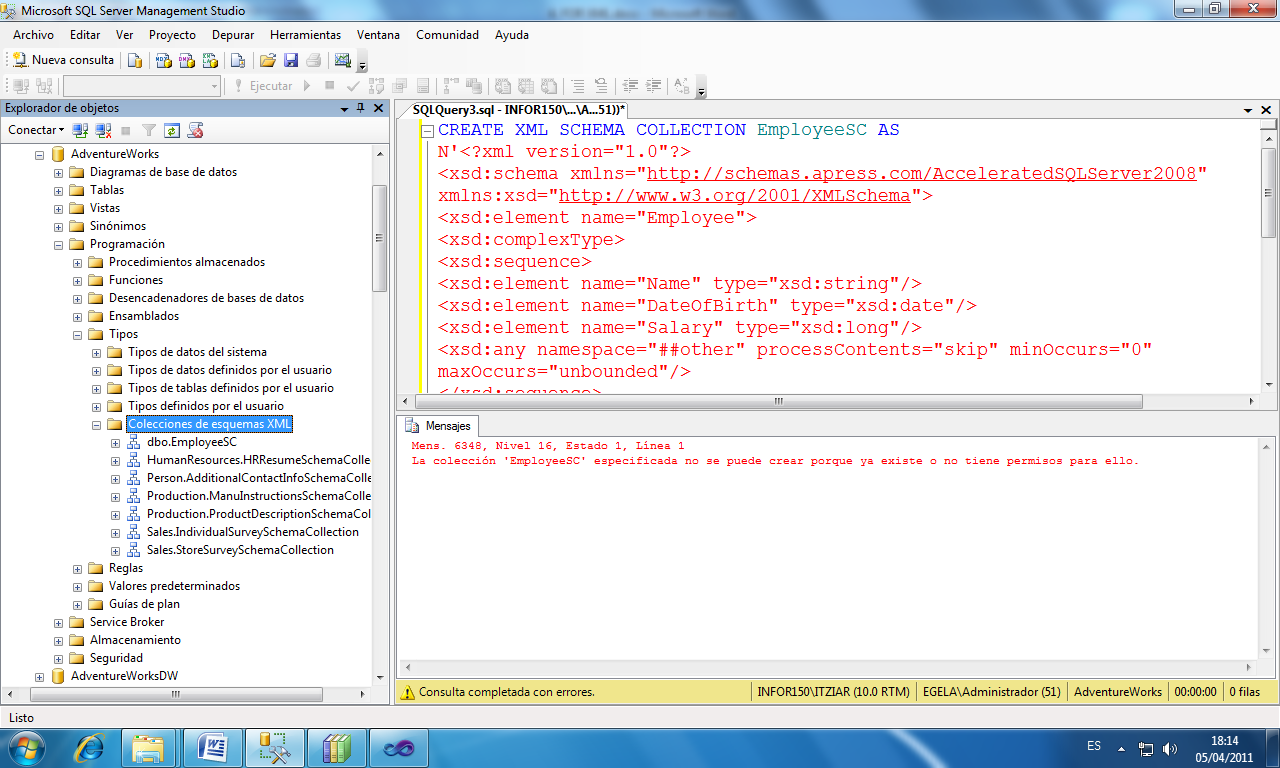
maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:schema>';



Acoplamos este esquema con una table:

CREATE TABLE Employees (emp\_xml XML(EmployeeSC))

En el esquema, definimos los elementos Name, DateOfBirth, y Salary, seguidos por una definición xsd:any.

Le estamos diciendo al procesador que permita cualquier elemento de cualquier espacio de nombres en esta ubicación del documento.

Este documento sería válido:

<Employee xmlns:nsP="http://schemas.apress.com/AcceleratedSQLServer2008">

<Name>John Doe</Name>

<DateOfBirth>1969-03-05</DateOfBirth>

<Salary>34900</Salary>

<nsP:Picture>C:\\employees\\emp1.bmp</nsP:Picture>

</Employee>

Uno de los parámetros disponibles es processContents. Este parámetro le dice al procesador si debe o no ejecutar la validación del elemento. Sus valores pueden ser:

* skip: no lleva a cabo la validación, incluso si el schema collection contiene una declaración para el elemento nsp:Picture del ejemplo anterior.
* strict: el procesador ejecutará la validación fallando si ocurre algún error durante el proceso.
* lax: Nuevo en SQL Server 2008. Fuerza la validación para cualquiera de los elementos que tengan las declaraciones del esquema asociado con ellos, pero ignora a cualquiera de los elementos no definidos en el esquema.

**Soporte Date y Time**

xs:dateTime representa un valor fecha/hora que opcionalmente puede incluir soporte de zona horaria.

En Sql Server 2008 se ha completado mejor con la especificación W3C (<http://www.w3.org/TR/xmlschema11-2/#dateTime>).

Supongamos el siguiente esquema:

<?xml version="1.0"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="StockPurchased" type="xsd:dateTime"/>

</xsd:schema>

En SQL Server 2005, habríamos requerido la información de la zona. Por lo que el siguiente mandato no sería válido, ya que falla en la especificación de la zona.

INSERT INTO DateTimeTest VALUES

('<StockPurchased>2008-01-27T16:11:00.813</StockPurchased >')

En SQL Server 2008, la información de la zona es opcional.

Además, este mandato trabaja correctamente en SQL Server 2008. Si consultamos la tabla DateTimeTest en SQL Server 2008, capturaríamos los siguiente:

<StockPurchased>2008-01-27T16:11:00.813</StockPurchased>

Otro cambio de interés desde SQL Server 2005 es que SQL Server 2008 preserva la información de la zona. Supongamos el siguiente mandato emitido en SQL Server 2005:

INSERT INTO DateTimeTest VALUES

('<StockPurchased>2008-10-10T12:00:00-04:00</StockPurchased>')

Vemos que estamos especificando una hora 4 horas antes del UTC. Si consultamos esta tabla en SQL Server, obtenemos:

<StockPurchased>2008-10-10T16:00:00Z</StockPurchased>

Vemos que el valor almacenado es diferente que el que hemos enviado originalmente al motor de la base de datos ya que internamente este convirtió la hora a un valor basado en el tiempo de zona 0. (Z).

Este comportamiento no ocurre en SQL Server 2008, ya que la misma consulta devolverá el mismo valor que fue introducido.

**Tipos Union y List**

SQL Server 2008 permite declaraciones de esquema para estos tipos.

Como ejemplo, supongamos que nuestra compañía tiene un sitio Web que vende zapatos. La página original fue diseñada para que solo se pudiesen vender zapatos de vestir y zapatillas de tenis.

De repente, un director ha decidido ampliar la línea de productos e incluir zapatillas y otros productos definidos como de pequeño, medio y gran tamaño, que se contraponen a los tamaños tradicionales como 9, 9,5, y 10.

Como desarrolladores XML, tenemos un tipo definido que toma el valor del tamaño del zapato, pero este espera un número y no un string.

Lo ideal sería que nosotros queremos que el esquema acepte tanto valores numéricos como strings, dependiendo de qué zapato sea comprado. Definimos dos listas de valores aceptados: una con valores numéricos y otra con strings.

Nuestro nuevo esquema es:

CREATE XML SCHEMA COLLECTION OrderSchema AS

N'<?xml version="1.0" encoding="UTF-16"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="shoesize" type="shoeSizeType"/>

<xs:simpleType name="shoeSizeType">

<xs:union>

<xs:simpleType>

<xs:list>

<xs:simpleType>

<xs:restriction base="xs:decimal">

<xs:enumeration value="9"/>

<xs:enumeration value="9.5"/>

<xs:enumeration value="10"/>

<xs:enumeration value="10.5"/>

</xs:restriction>

</xs:simpleType>

</xs:list>

</xs:simpleType>

<xs:simpleType>

<xs:list>

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="S"/>

<xs:enumeration value="M"/>

<xs:enumeration value="L"/>

</xs:restriction>

</xs:simpleType>

</xs:list>

< /xs:simpleType>

</xs:union>

</xs:simpleType>

</xs:schema>'

Ahora podemos controlar cualquier tamaño numérico o tamaño string.

**Programando SQLXML desde .NET y COM**

Soporte de Microsoft para este Tema:

<http://technet.microsoft.com/es-es/library/ms171779(SQL.90).aspx>

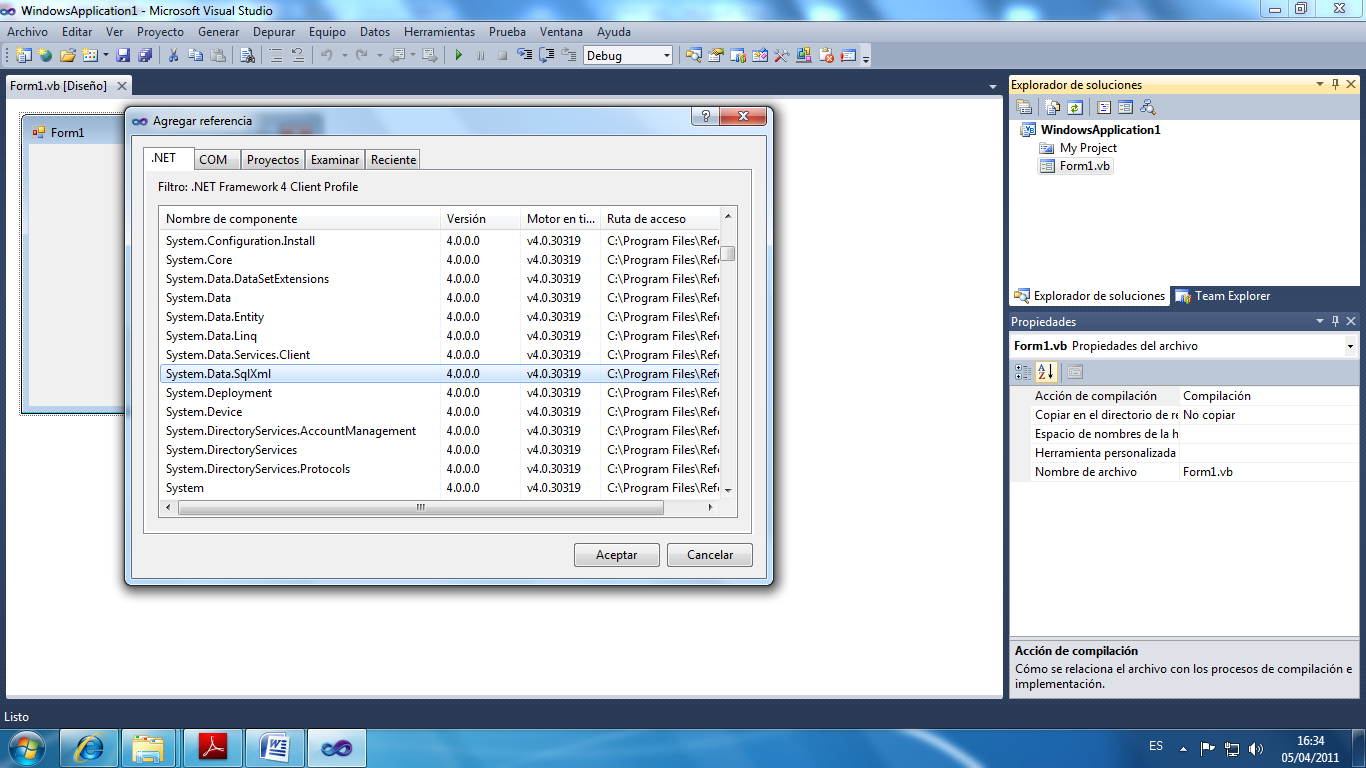
Para poder trabajar elaborando proyectos que utilicen SQLXML en el entorno .NET, debemos descargarnos el Software necesario:

<http://msdn.microsoft.com/es-es/library/cc645615.aspx>

SQLXML 4.0 suministra un modelo de objetos que nos permite programar sus capacidades tanto desde código administrado como no administrado.

Para código no administrado, como ADO, usaremos el driver SQLXMLOLEDB. Este proveedor usa el nuevo SQL Server Native Client.

Para código administrado, podemos añadir una referencia a System.Data.SqlXml en Visual Studio:



El ensamblado SQLXML tiene cuatro clases: SqlXmlCommand, SqlXmlParameter, SqlXmlAdapter,

y SqlXmlException.

Usando estas clases, podemos enviar comandos a SQL Server y procesar los resultados en el lado del cliente, así como renderizar los mandatos FOR XML o ejecutar las plantillas XML.

Veremos cada una de las clases y describiremos algunos ejemplos.

**SQLXML Classes**

SqlXmlCommand es una de las primeras clases con las que interactuamos al usar la funcionalidad SQLXML.

**Tabla de Métodos**

|  |  |
| --- | --- |
| **Name** | **Description** |
| ClearParameters | Elimina todos los parámetros que fueron creados para un objeto command particular |
| CreateParameter | Crear un objeto SqlXmlParameter al que se le puede asignar el nombre y el valor del parámetro |
| ExecuteNonQuery | Ejecuta la consulta pero no devuelve ningún valor. Es interesante si queremos llamar a un updategram que no devuelve un valor |
| ExecuteToStream | Ejecuta la consulta y devuelve los resultados a un objeto Stream que pasamos al método |
| ExecuteStream | Ejecuta la consulta y devuelve los resultados como un nuevo objeto Stream |
| ExecuteXMLReader | Ejecuta la consulta y devuelve los resultados en un objeto XMLReader |

**Tabla de Propiedades**

|  |  |
| --- | --- |
| BasePath | El directorio base Path, que es satisfactorio para asignar paths a XSL, schema mapping, o ficheros de esquema XSD usados en las aplicaciones |
| ClientSideXML | Cuando es true, esta propiedad booleana le dice a SQLXML que convierta los conjuntos de filas a XML en el lado del cliente |
| CommandStream | Permite asignar nuestro commando usando un Stream. Es Bueno si queremos ejecutar un commando desde un fichero |
| CommandText | Coge o asigna el texto del comando que queremos ejecutar |
| CommandType | Permite capturer o asignar el tipo de commando usando los siguientes valores: SQLXMLCommandType.Template, SQLXMLCommandType.TemplateFile, SQLXMLCommandType.XPath, y SQLXMLCommandType.UpdateGram |
| Namespaces | Nos permite especificar espacios de nombres para nuestro XML, en el formato xmlns:x='urn:myschema:Yournamespace'. Cuando usemos consultas XPATH que tienen espacios de nombres cualificados, debemos especificar sus espacios de nombres usando esta propiedad |
| OutputEncoding | Especifica la decodificación de los resultados. Por defecto, la decodificación es UTF-8, pero podríamos especificar ANSI, Unicode, o cualquier otro valor válido de decodificación. |
| RootTag | Especifica la etiqueta raíz del documento XML, si es requerida. Será normalmente el string de la etiqueta raíz. |
| SchemaPath | Especifica la ruta del directorio y el nombre del fichero para el fichero de schema. Si estamos utilizando rutas relativas vía la propiedad BasePath, SQLXML mirará en el directorio base raíz. |
| XslPath | Igual que en SchemaPath pero especificando la ruta del fichero XSLT en lugar del fichero de schema. |

SqlXmlParameter suministra la posibilidad de pasar parámetros al código. Esta clase es muy sencilla ya que tiene solo tres propiedades: Name y Value.

Se especifica el nombre del parámetro y el valor.

Podemos crear un objeto SqlXmlParameter llamando al método del objeto SqlXmlCommand.

El objeto SqlXmlAdapter permite la interoperabilidad entre los datasets .NET y la funcionalidad SQLXML.

El constructor para este objeto tiene tres formatos:

* El primero puede coger un SqlXmlCommand que es poblado con la información necesaria para conectar al origen de datos.
* El segundo es el texto del comando como string, el tipo de comando como SqlXmlCommand y la conexión como string.
* El tercero es el mismo que el segundo, excepto que le pasamos un objeto Stream en lugar de un string para el texto del comando.

Una vez creado el adaptador, tenemos dos métodos sobre el objeto. El método Fill al que le pasamos un dataset

ADO.NET. SQLXML llenará el dataset con datos.

Después podremos modificar el dataset usando las funcionalidades del dataset y llamar posteriormente al método Update, entregándole el dataset como parámetro.

SQLXML usa la modalidad de bloqueo optimista.

El objeto SqlXmlException hereda del objeto SystemException y nos permite pasar excepciones como parte del código. Existe la propiedad ErrorStream que usamos para devolver el error. Este ejemplo usa esta propiedad para sacar los errores:

Catch ex As Microsoft.Data.SqlXml.SqlXmlException

ex.ErrorStream.Position = 0

Dim oSR As New System.IO.StreamReader(ex.ErrorStream)

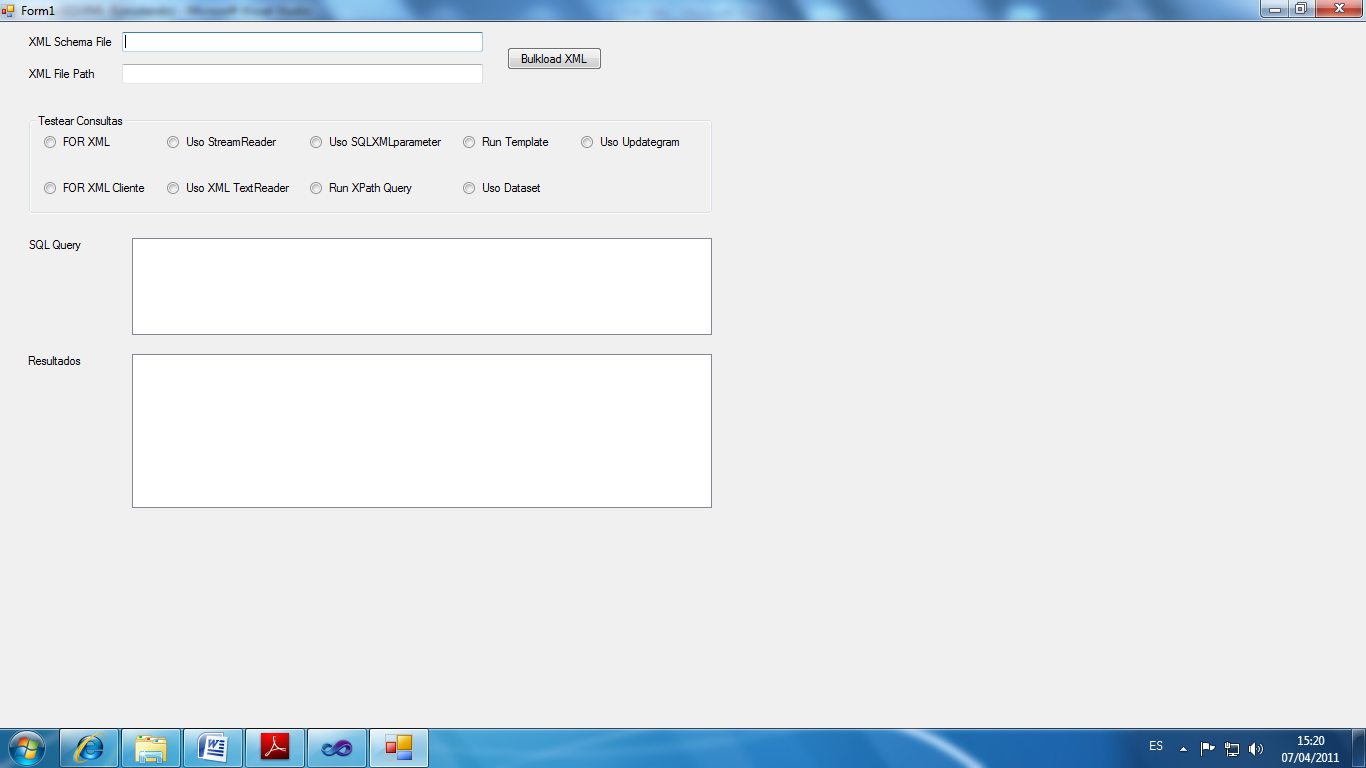
Dim strResult As String = oSR.ReadToEnd()

System.Console.WriteLine(strResult)

End Try

**SQLXML Coding Examples**

Veamos esta aplicación de ejemplo. Este ejemplo nos permite cargar datos XML sobre SQL Server para después trabajar con las distintas funcionalidades vistas anteriormente: FOR XML, integración de datasets, plantillas, uso de updategrams, procesamiento del lado del cliente y XMLTextReaders. Empecemos por el interface de usuario:



Este ejemplo tiene una referencia a SQLXML. Para iniciar el trabajo con datos, necesitamos cargar nuestros datos XML en nuestra base de datos e incluirlos en las columnas de las tablas.

Este código podrías usar el mandato OPENXML pero hemos decidido usar BulkLoad en su lugar.

Para iniciar el uso de BulkLoad, necesitamos añadir una referencia al objeto COM BulkLoad en VisualStudio.

El nombre del componente es Microsoft SQLXML BulkLoad 4.0 Type Library. Next.

Después creamos un objeto BulkLoad en nuestra aplicación:

Dim oXMLBulkLoad As New SQLXMLBULKLOADLib.SQLXMLBulkLoad4Class()

Después necesitamos algunas propiedades para el BulkLoad. No podemos asumir que la tabla que vamos a utilizar en la carga ya existe, por lo que asignamos la propiedad SchemaGen a true. Si la tabla existe, la borramos. El ejemplo asigna a

Entre todas las propiedades la más importante es ConnectionString, ya que le dice a BulkLoad cómo conectar al servidor.

El método execute tiene como parámetros el fichero de mapping y el XML a cargar.

oXMLBulkLoad.ErrorLogFile = "c:\myerrors.log"

oXMLBulkLoad.SchemaGen = True

oXMLBulkLoad.KeepIdentity = False

oXMLBulkLoad.BulkLoad = True

oXMLBulkLoad.SGDropTables = True

oXMLBulkLoad.XMLFragment = True

oXMLBulkLoad.ConnectionString = strConnectionString

oXMLBulkLoad.Execute(txtXMLSchema.Text, txtXMLFile.Text)

FOR XML: Server-Side y procesamiento Client-Side

Una vez cargados los datos, podemos trabajar con ellos. Podemos usar el constructor FOR XML para trabajar.

SQLXML nos permite renderizar el XML tanto en el lado servidor como en el lado cliente.

El ejemplo nos permite seleccionar cualquiera de las dos opciones. El código usa un método común para ejecutar todas las consultas. Este método toma un número diferente de parámetros (SQL, template o diffgram). Un diffgram es un formato XML que se usa para identificar las versiones actuales y originales de las versiones de los datos.

Lo primero que se hace es crear un objeto SqlXmlCommand:

Dim oSQLXMLCommand As New Microsoft.Data.SqlXml.SqlXmlCommand(strConnectionString)

Este asigna el tipo de comando para que sea un tipo apropiado basado en la consulta introducida. Para consultas SQL estándares el tipo es SQL:

oSQLXMLCommand.CommandType = Microsoft.Data.SqlXml.SqlXmlCommandType.Sql

Para enviar nuestra consulta FOR XML al servidor, necesitamos asignar el texto del comando al objeto SqlXmlCommand:

‘Asignamos nuestra Query

oSQLXMLCommand.CommandText = strQuery

Ya que podemos renederizar nuestro FOR XML en el servidor y en el cliente, necesitamos asignar la propiedad ClientSideXml de nuestro objeto a true o false.

El código siguiente usa StreamrReader para capturar los resultados y los deja en nuestra caja de texto. Podemos usar también XmlTextReader:

‘ Miramos si podemos renderizar en lado del cliente

If bUseClientSide = True Then

oSQLXMLCommand.ClientSideXml = True

End If

Dim oStream As System.IO.Stream

oStream = oSQLXMLCommand.ExecuteStream()

oStream.Position = 0

Dim oStreamReader As New System.IO.StreamReader(oStream)

txtResults.Text = oStreamReader.ReadToEnd()

oStreamReader.Close()

Como podemos ver, el uso de FOR XML en SQLXML es muy sencillo. La parte dura es que la consulta FOR XML sea correcta y que devuelva los datos correctos.

Using an XMLTextReader

Hay veces en las que no queremos usar un StreamReader para coger los resultados de nuestras consultas SQLXML, sino que lo queremos es usar un XMLTextReader. El XMLTextReader da un acceso rápido a XML y más flexibilidad en la navegación en el XML que la que consigue un StreamReader.

El XMLTextReader analiza el XML y permite consultar usando XPath. Para usar un XMLTextReader, ynecesitamos cambiar el método de llamada ExecuteStream por el método ExecuteXMLReader en el SqlXmlCommand.

A continuación usaremos los métodos y propiedades de XML reader para navegar por XML.

El código que exponemos ejecuta un XML reader y despliega los resultados al usuario:

'Uso de XMLTextReader

Dim oXMLTextReader As System.Xml.XmlTextReader

oXMLTextReader = oSQLXMLCommand.ExecuteXmlReader()

Dim strXML As String = ""

While oXMLTextReader.Read()

'Nos situamos en un elemento

If oXMLTextReader.NodeType = XmlNodeType.Element Then

strXML += "<" & oXMLTextReader.Name & ""

ElseIf oXMLTextReader.NodeType = XmlNodeType.EndElement Then

strXML += "</" & oXMLTextReader.Name & ">"

End If

'Miramos los atributos

If oXMLTextReader.HasAttributes() Then

Dim i As Integer = 0

Do While (oXMLTextReader.MoveToNextAttribute())

i += 1

strXML += " " & oXMLTextReader.Name & "=" & oXMLTextReader.Value

If oXMLTextReader.AttributeCount = i Then

'Último atributo, y el tag

strXML += " />"

End If

Loop

End If

End While

txtResults.Text = strXML

oXMLTextReader.Close()

Como podemos ver, el uso de XML reader para operaciones simples como visualizar el contenido XML, es excesivo ya que debe analizar el XML para visualizarlo. Sin embargo, si queremos tener más control sobre el XML, como obtener el número de atributos y elementos, o si queremos navegar de una forma más sofisticada, el XML reader es más adecuado.

Uso de Parametros con SQLXML

Para usar parámetros con SQLXML, necesitamos crear un objeto SqlXmlParameter.

Nuestra consulta debe especificar que estamos pasando un parámetro, y que el objeto SqlXmlParameter debe tener sus propiedades asignadas correctamente.

El código nos muestra cómo usar un parámetro en las consultas SQLXML.

strQuery = "SELECT \* FROM " & strTable & " WHERE city = ? FOR XML AUTO, ELEMENTS"

. . .

Dim oSQLXMLParameter As Microsoft.Data.SqlXml.SqlXmlParameter

oSQLXMLParameter = oSQLXMLCommand.CreateParameter()

oSQLXMLParameter.Name = "city"

oSQLXMLParameter.Value = "Oakland"

...

Ejecución de consultas XPath o SQL con plantillas

Con SQLXML, podemos ejecutar consultas XPath o SQL. Las aplicaciones de ejemplo usan una plantilla para ejecutar una consulta SQL y un mandato directo XPath para la consulta XPath.

El ejemplo podría usar una plantilla para la consulta XPath, pero este ejemplo quiere demostrar cómo utilizar el tipo de comando XPath.

Este código establece la plantilla de consulta SQL:

'Carga de nuestra consulta

strQuery = "<Root><sql:query xmlns:sql=""urn:schemas-microsoft-com:xml-sql""> \_"

SELECT \* FROM AuthorsXMLNew FOR XML AUTO</sql:query></Root>"

Después, el ejemplo asigna el tipo de comando para que sea una plantilla para que ejecute el SQL.

El ejemplo especifica también el nodo raíz y la ruta al ficher de esquema XSD.

oSQLXMLCommand.CommandType = Microsoft.Data.SqlXml.SqlXmlCommandType.Template

oSQLXMLCommand.SchemaPath = txtXMLSchema.Text

oSQLXMLCommand.RootTag = "ROOT"

El código usa un StreamReader para renderizar el resultado. Este código ya lo mostramos antes.

Para ejecutar la consulta XPath, de nuevo, ejecutamos el siguiente código:

‘Cargamos nuestra consulta

strQuery = "/AuthorsXMLNew[city='Oakland']"

Ya que estamos usando una consulta XPath de forma directa, necesitamos asignar el tipo de comando para que sea XPath.

Tal como dijimos en el caso de la consulta SQL de plantilla, queremos asignar nuestro nodo raíz y laruta al esquema XSD. Después, volvemos a usar el StreamReader para renderizar nuestros resultados:

oSQLXMLCommand.CommandType = Microsoft.Data.SqlXml.SqlXmlCommandType.XPath

oSQLXMLCommand.SchemaPath = txtXMLSchema.Text

oSQLXMLCommand.RootTag = "ROOT"

Interactuando con el Dataset ADO.NET

SQLXML interactúa con el dataset with the ADO.NET mediante el objeto SqlXmlAdapter.

Podemos usar el SqlXmlAdapter para rellenar nuestro dataset. Después podemos usar el objeto DataSet tal como lo haríamos en ADO.NET.

El código que exponemos, crea una consulta, la ejecuta usando el objeto SqlXmlAdapter, y a continuación llena un dataset con la información.

Para grabar externamente el valor devuelto, el código usa objetos stream.

strQuery = "SELECT \* FROM " & strTable & " WHERE city ="

'oakland' FOR XML AUTO, ELEMENTS"

...

Dim oSQLXMLDataAdapter As New \_

Microsoft.Data.SqlXml.SqlXmlAdapter(oSQLXMLCommand)

Dim oDS As New System.Data.DataSet()

oSQLXMLDataAdapter.Fill(oDS)

'Visualiza el XML subyacente

Dim oMemStream As New System.IO.MemoryStream()

Dim oStreamWriter As New System.IO.StreamWriter(oMemStream)

oDS.WriteXml(oMemStream, System.Data.XmlWriteMode.IgnoreSchema)

oMemStream.Position = 0

Dim oStreamReader As New System.IO.StreamReader(oMemStream)

txtResults.Text = oStreamReader.ReadToEnd()

oMemStream.Close()

Programación de Updategrams

En este caso, analizaremos el uso de updategrams. Updategrams nos permite actualizar nuestro SQL Server usando los documentos XML existentes.

El código crea el updategram usando un objeto StringBuilder. Después, el código asigna el tipo de commando a UpdateGram.

Finalmente, el resto del código es el mismo que el código original para ejecutar el comando; por esta razón, no repetimos el código.

Dim strUpdateGram As New System.Text.StringBuilder()

strUpdateGram.Append("<?xml version='1.0'?><AuthorsXMLNewupdate ")

strUpdateGram.Append("xmlns:updg='urn:schemas-microsoft-com:xml-updategram'>")

strUpdateGram.Append("<updg:sync updg:nullvalue='nothing'>" &

"<updg:before></updg:before>")

strUpdateGram.Append("<updg:after><AuthorsXMLNew au\_id='123-22-1232'")

strUpdateGram.Append(" au\_fname='Tom' state='WA' phone='425-882-8080'/>")

strUpdateGram.Append("</updg:after>")

strUpdateGram.Append("<updg:before><AuthorsXMLNew")

strUpdateGram.Append(" au\_id='267-41-2394'/></updg:before>")

strUpdateGram.Append("<updg:after></updg:after>")

strUpdateGram.Append("<updg:before><AuthorsXMLNew")

strUpdateGram.Append(" au\_id='238-95-7766'/></updg:before>")

strUpdateGram.Append("<updg:after><AuthorsXMLNew")

strUpdateGram.Append(" city='Oakland' phone='212-555-1212'/>")

strUpdateGram.Append("</updg:after></updg:sync></AuthorsXMLNewupdate>")

strQuery = strUpdateGram.ToString()

...

oSQLXMLCommand.CommandType =

Microsoft.Data.SqlXml.SqlXmlCommandType.UpdateGram