16

Trabajar con datos XML en SQL Server 2000

SQL Server 2000 puede usar protocolos y estándares propios de Internet para comunicarse con otras aplicaciones y otros sistemas. El lenguaje extensible de marcación, o XML (*eXtensible Markup Language*) es el formato estándar de facto para la comunicación entre aplicaciones en Internet.

En este capítulo aprenderemos de qué manera SQL Server usa las tecnologías de la Web y XML para proveer una comunicación independiente de la plataforma. Será solamente una introducción a este tema amplio y muy difundido; para una discusión más completa del funcionamiento de XML en general remitimos al lector a algún libro sobre XML, por ejemplo el de Benoît Marchal, *XML con Ejemplos*. En este capítulo aprenderemos lo siguiente:

- Los conceptos básicos de XML
- Cómo leer datos en formato XML en SQL Server 2000
- El uso de documentos XML en Transact-SQL
- Cómo acceder a SQL Server a través de HTTP
- El uso de *updategrams* XML para modificar, insertar y eliminar datos en SQL Server

Breve introducción a XML

Cada aplicación de computadora usa sus propios tipos de datos y los almacena en un formato interno propio. La comunicación entre aplicaciones es una operación compleja que requiere llegar a un acuerdo acerca del formato de la comunicación. Si las aplicaciones se ejecutan en plataformas de hardware y de sistema operativo diferentes, la situación se hace aún más compleja.

El lenguaje de marcación extensible (XML) alivia esta situación. XML provee un formato autodescriptivo para expresar datos semiestructurados en formato de texto puro, compatible con la mayoría de los sistemas informáticos.

El listado 16.1 muestra un documento XML típico. Para crear un documento XML se puede usar cualquier editor de texto, por ejemplo el Bloc de notas.

Listado 16.1: Un documento XML típico

```
<?xml version='1.0' ?>
<!--De esta forma se escriben los comentarios en un documento XML -->

<Products>
   <Product ProductID="1" ProductName="Chai" UnitPrice="18.0000"/>
   <Product ProductID="2" ProductName="Chang" UnitPrice="19.0000"/>
   <Product ProductID="3" ProductName="Aniseed Syrup" UnitPrice="10.0000"/>
</Products>
```

Internet Explorer 5.5 es capaz de abrir documentos XML. Si guarda el documento del listado 16.1 con la extensión XML y luego lo abre en Internet Explorer, verá una imagen similar a la figura 16.1.

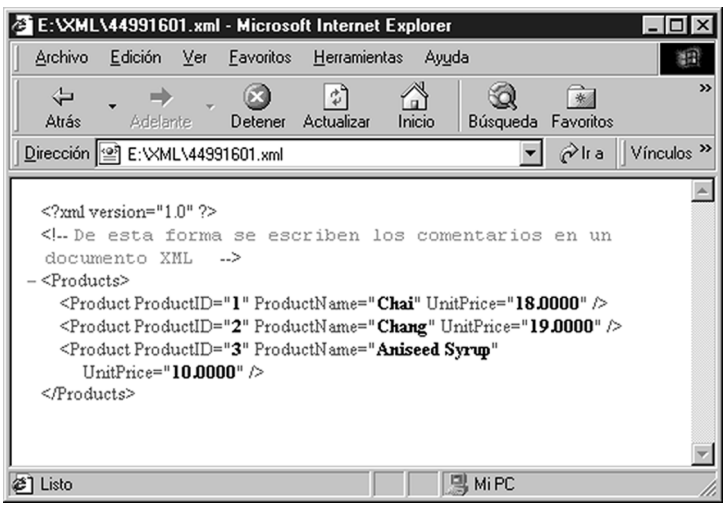

Figura 16.1: *Internet Explorer se puede usar como visor de XML.*

En el documento XML del listado 16.1 podemos ver diferentes secciones. La etiqueta que sigue es una instrucción para el procesador que identifica el documento como documento XML, además de la versión de XML usada:

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

Esto es solamente un comentario dentro del documento:

```
<!--De esta forma se escriben los comentarios en un documento XML -->
```

La etiqueta siguiente identifica el elemento raíz descrito por este documento. Todo documento XML bien formado debe tener un elemento raíz. Este documento describe productos:

```
<Products>
```

Cada una de las líneas siguientes describe un elemento en particular (un producto, en este caso) e indica tres atributos para cada producto: `ProductID`, `ProductName` y `UnitPrice` (el identificador, el nombre y el precio unitario, respectivamente). Para cada atributo, el documento indica el nombre del atributo y el valor que tiene ese atributo para cada producto en particular. El símbolo `/` al final de cada línea representa el final de la definición del elemento individual. En este caso es una simplificación de la sintaxis formal `</Product>`.

```
<Product ProductID="1" ProductName="Chai" UnitPrice="18.0000"/>
<Product ProductID="2" ProductName="Chang" UnitPrice="19.0000"/>
<Product ProductID="3" ProductName="Aniseed Syrup" UnitPrice="10.0000"/>
```

Estas tres líneas también se pueden escribir así:

```
<Product ProductID="1" ProductName="Chai" UnitPrice="18.0000">
</Product>
<Product ProductID="2" ProductName="Chang" UnitPrice="19.0000">
</Product>
<Product ProductID="3" ProductName="Aniseed Syrup" UnitPrice="10.0000">
</Product>
```

Al final del documento XML, debe haber una etiqueta de cierre que marque el fin de la definición del elemento raíz.

```
</Products>
```

PRECAUCIÓN

Observe que en XML se distingue entre mayúsculas y minúsculas. Si la etiqueta de inicio de un elemento y la de cierre no coinciden en el uso de mayúsculas y minúsculas, se producirá un error de sintaxis de XML. Por ejemplo, el listado 16.2 produce el mensaje de error que aparece en la salida incluida. La única diferencia entre el listado 16.1 y el 16.2 es que en este último la etiqueta de cierre `</products>` empieza con minúsculas y la etiqueta de inicio con mayúsculas; la etiqueta de cierre debería haber sido `</Products>`.

Listado 16.2: Este documento produce un error porque el uso de mayúsculas y minúsculas difiere en la etiqueta de cierre y en la etiqueta de inicio

```
<?xml version='1.0' ?>
<!--De esta forma se escriben los comentarios en un documento XML -->

<Products>
   <Product ProductID="1" ProductName="Chai" UnitPrice="18.0000"/>
   <Product ProductID="2" ProductName="Chang" UnitPrice="19.0000"/>
   <Product ProductID="3" ProductName="Aniseed Syrup" UnitPrice="10.0000"/>
</products>
```


```
The XML page cannot be displayed
Cannot view XML input using XSL style sheet.
Please correct the error and then click the Refresh button,
or try again later.
------------------------------------------------------------------------------
End tag 'products' does not match the start tag 'Products'. Line 8, Position 3
</products>
--^
```

En ocasiones es preferible definir los atributos como elementos independientes. Esto puede resultar de utilidad si los datos se deben describir organizados jerárquicamente. El listado 16.3 muestra el mismo documento XML que el listado 16.1, pero en este caso hemos definido cada atributo como un elemento independiente. La figura 16.2 muestra este documento en Internet Explorer. Observe la diferencia entre esta figura y la 16.1.

Listado 16.3: Documento XML típico en el que los atributos están definidos como elementos

```
<?xml version='1.0' ?>
<Products>
 <Product>
   <ProductID>1</ProductID>
   <ProductName>Chai</ProductName>
   <UnitPrice>18.0000</UnitPrice>
 </Product>
 <Product>
   <ProductID>2</ProductID>
   <ProductName>Chang</ProductName>
   <UnitPrice>19.0000</UnitPrice>
 </Product>
 <Product>
   <ProductID>3</ProductID>
   <ProductName>Aniseed Syrup</ProductName>
   <UnitPrice>10.0000</UnitPrice>
 </Product>
</Products>
```

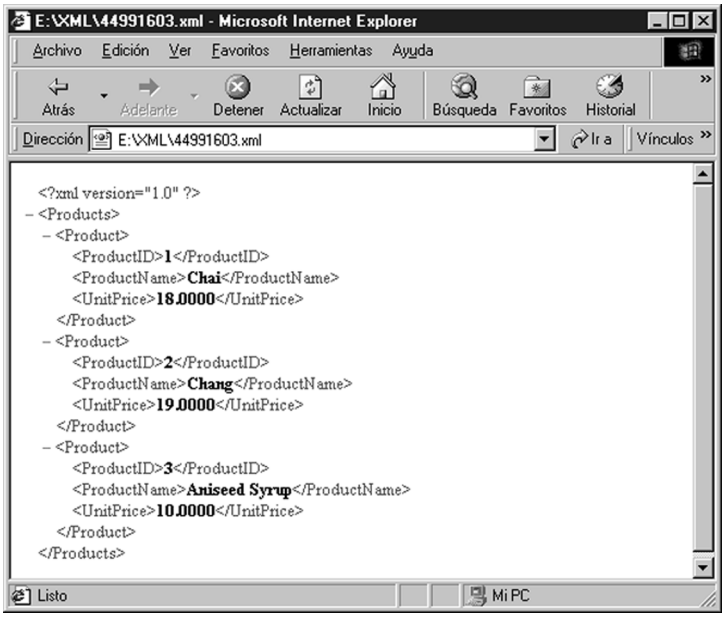

Figura 16.2: *Documento XML organizado en torno a los elementos.*

El listado 16.4 muestra un ejemplo de una estructura jerárquica en XML. La figura 16.3 muestra cómo se ve este documento en Internet Explorer. La figura 16.4 muestra la representación de este documento XML como un árbol en el que a cada elemento le corresponde un nodo en la estructura del árbol.

Listado 16.4: Un documento XML puede representar datos organizados jerárquicamente

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

<Customers>
 <Customer CompanyName="Victuailles en stock">
   <Order Date="1996-07-08">
     <Product Name="Gustafaposs Knackebrod" Price="16.8000" Quantity="6"/>
     <Product Name="Ravioli Angelo" Price="15.6000" Quantity="15"/>
     <Product Name="Louisiana Fiery Hot Pepper Sauce" Price="16.8000"
       Quantity="20"/>
   </Order>
   <Order Date="1996-10-21">
     <Product Name="Filo Mix" Price="5.6000" Quantity="8"/>
     <Product Name="Scottish Longbreads" Price="10.0000" Quantity="10"/>
   </Order>
   <Order Date="1997-02-19">
     <Product Name="Ikura" Price="24.8000" Quantity="20"/>
     <Product Name="Tourtiere" Price="5.9000" Quantity="6"/>
   </Order>
```

Listado 16.4: continuación

```
    <Order Date="1997-02-27">
      <Product Name="Uncle Bobaposs Organic Dried Pears" Price="24.0000"
        Quantity="16"/>
      <Product Name="Spegesild" Price="9.6000" Quantity="20"/>
      <Product Name="Mozzarella di Giovanni" Price="27.8000" Quantity="40"/>
    </Order>
    <Order Date="1997-03-18">
      <Product Name="Ikura" Price="24.8000" Quantity="20"/>
    </Order>
    <Order Date="1997-05-23">
      <Product Name="Uncle Bobaposs Organic Dried Pears" Price="30.0000"
        Quantity="10"/>
      <Product Name="Steeleye Stout" Price="18.0000" Quantity="30"/>
      <Product Name="Tarte au sucre" Price="49.3000" Quantity="40"/>
    </Order>
    <Order Date="1997-12-31">
      <Product Name="Chang" Price="19.0000" Quantity="20"/>
      <Product Name="Louisiana Fiery Hot Pepper Sauce" Price="21.0500"
        Quantity="2"/>
      <Product Name="Longlife Tofu" Price="10.0000" Quantity="15"/>
    </Order>
  </Customer>
  <Customer CompanyName="Vins et alcools Chevalier">
    <Order Date="1996-07-04">
      <Product Name="Queso Cabrales" Price="14.0000" Quantity="12"/>
      <Product Name="Singaporean Hokkien Fried Mee" Price="9.8000"
        Quantity="10"/>
      <Product Name="Mozzarella di Giovanni" Price="34.8000" Quantity="5"/>
    </Order>
    <Order Date="1996-08-06">
      <Product Name="Flotemysost" Price="17.2000" Quantity="20"/>
      <Product Name="Mozzarella di Giovanni" Price="27.8000" Quantity="7"/>
    </Order>
    <Order Date="1996-09-02">
      <Product Name="Gnocchi di nonna Alice" Price="30.4000" Quantity="4"/>
    </Order>
    <Order Date="1997-11-11">
      <Product Name="Konbu" Price="6.0000" Quantity="4"/>
      <Product Name="Jack&amp;apos;s New England Clam Chowder" Price="9.6500"
        Quantity="12"/>
    </Order>
    <Order Date="1997-11-12">
      <Product Name="Inlagd Sill" Price="19.0000" Quantity="6"/>
      <Product Name="Filo Mix" Price="7.0000" Quantity="18"/>
    </Order>
  </Customer>
</Customers>
```

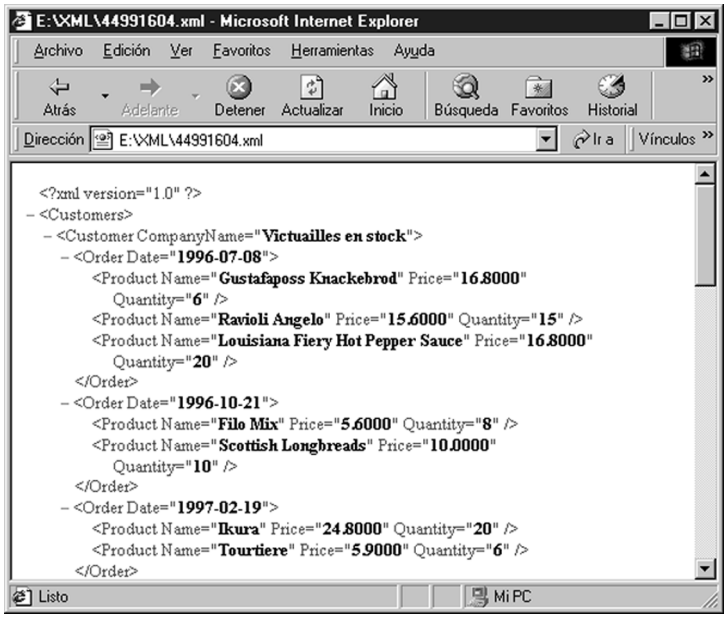

Figura 16.3: *Vista del documento XML organizado jerárquicamente en Internet Explorer.*

Como puede ver en las figuras 16.3 y 16.4, este documento XML contiene varios niveles jerárquicos:

- Un nivel raíz, llamado `Customers`, que define el propósito de los datos de este documento XML (en este caso, la descripción de clientes).
- Un elemento `Customer` para cada cliente. Este elemento contiene un atributo `Name` que en este caso muestra el nombre de la compañía.
- Un elemento `Order` para cada pedido formulado por este cliente en particular. Este elemento muestra como atributo la fecha del pedido.
- Para cada pedido, un elemento `Product` para cada producto incluido en ese pedido en particular. Para cada producto el documento muestra como atributos el nombre del producto, su precio unitario y la cantidad pedida.

Al observar los ejemplos precedentes, el lector podría preguntarse por qué habríamos de usar este formato XML en lugar del formato de salida estándar de SQL Server que devuelven las diversas bibliotecas de base de datos. Las razones principales son la compatibilidad y la versatilidad.

Todos los sistemas informáticos aceptan e interpretan archivos de texto; por ende, todos aceptan documentos XML. Por el contrario, no todos los sistemas pueden interpretar un conjunto de resultados devuelto en el formato puro TDS (*Tabular*

Data Stream) de SQL Server. Y no todos los sistemas tienen bibliotecas de base de datos a su disposición para conectarse a SQL Server e interpretar paquetes TDS.

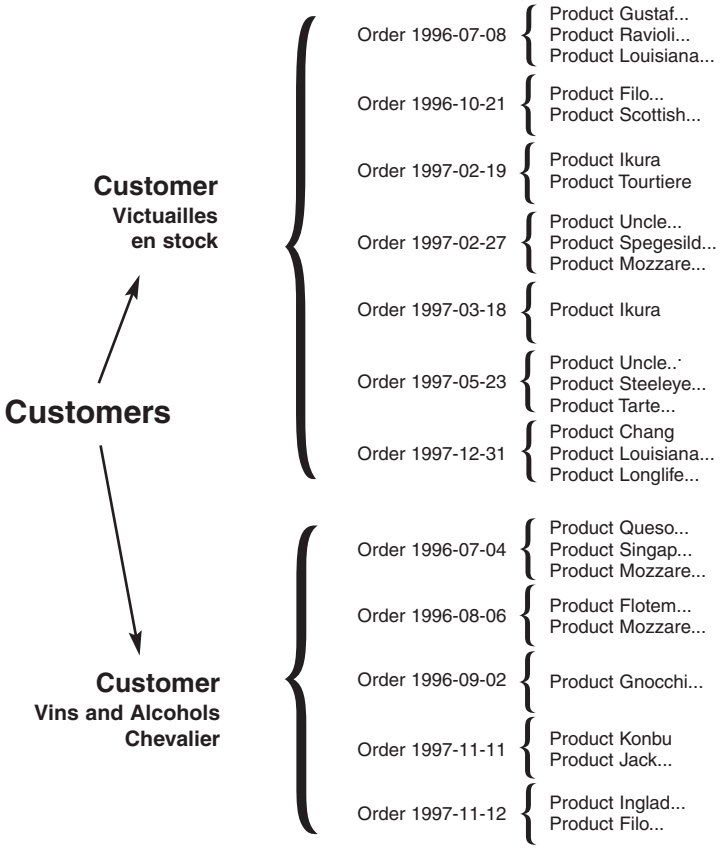

Figura 16.4: *Representación gráfica de un documento XML organizado jerárquicamente.*

Si el lector observa los ejemplos dados, verá que los documentos XML no incluyen nada acerca del formato que se usará para presentarlos. El documento contiene exclusivamente datos y sus definiciones, pero no instrucciones de formato. Éste es un aspecto importante de los documentos XML: son altamente versátiles en lo que respecta al modo que uno elige para su presentación.

Para definir el formato a usar para la presentación de documentos XML, se pueden usar hojas de estilo, que se definen en el lenguaje extensible de hojas de estilo (XSL, *eXtensible Stylesheet Language*).

El listado 16.5 muestra un ejemplo de la forma en que se puede cambiar el formato de presentación de un documento XML suministrando el archivo XSL

apropiado. El listado define una hoja de estilo pensada para aplicarle un formato al documento del listado 16.1. El listado 16.6 muestra el documento del listado 16.1, con una referencia al archivo XSL. La figura 16.5 muestra cómo se ve la salida formateada en Internet Explorer.

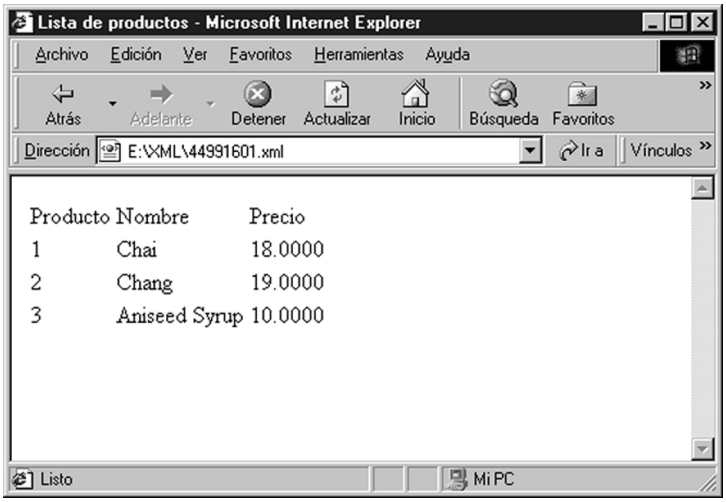

Figura 16.5: *Así se ve un documento XML en Internet Explorer después de aplicarle un sencillo archivo XSL.*

Listado 16.5: Un archivo XSL

```
<xsl:stylesheet xmlns:xsl='http://www.w3.org/TR/WD-xsl'>
<xsl:template match='/'>
 <HTML>
 <Title>Lista de productos</Title>
 <Body>
 <Table border='0'>
   <TR><TD>Producto</TD><TD>Nombre</TD><TD>Precio</TD></TR>
   <xsl:for-each select='Products/Product'>
     <TR>
       <TD><xsl:value-of select='@ProductID'/></TD>
       <TD><xsl:value-of select='@ProductName'/></TD>
       <TD><xsl:value-of select='@UnitPrice'/></TD>
     </TR>
   </xsl:for-each>
 </Table>
 </Body>
 </HTML>
</xsl:template>
</xsl:stylesheet>
```

Listado 16.6: Agregado de una referencia al archivo XSL en el documento XML

```
<?xml version='1.0' ?>
<!--De esta forma se agrega una referencia a la hoja de estilo -->

<?xml-stylesheet type='text/xsl' href='xmlListing5.xsl'?>
<!--De esta forma se escriben los comentarios en un documento XML -->

<Products>
   <Product ProductID="1" ProductName="Chai" UnitPrice="18.0000"/>
   <Product ProductID="2" ProductName="Chang" UnitPrice="19.0000"/>
   <Product ProductID="3" ProductName="Aniseed Syrup" UnitPrice="10.0000"/>
</Products>
```

El principal beneficio de esta técnica es que permite dividir las tareas de desarrollo entre las que tienen que ver con los datos en sí mismos y las que se refieren al formato que se les aplicará. El equipo de diseñadores Web puede concentrarse en cómo darles a las páginas Web una buena apariencia, mientras que el equipo de programación de datos puede concentrarse en la exactitud y la consistencia de los datos, sin preocuparse acerca de su presentación.

El listado 16.7 muestra un archivo XSL diferente, que cambia el formato del documento XML previo. La figura 16.6 muestra cómo se verá en Internet Explorer el documento XML del listado 16.6, después de sólo cambiar la siguiente línea de código para hacer referencia al nuevo archivo XSL:

```
<?xml-stylesheet type='text/xsl' href='xmlListing7.xsl'?>
```

De esta manera se puede cambiar fácilmente la apariencia de un sitio Web por completo, sin afectar la lógica de la programación o los datos disponibles en los archivos XML.

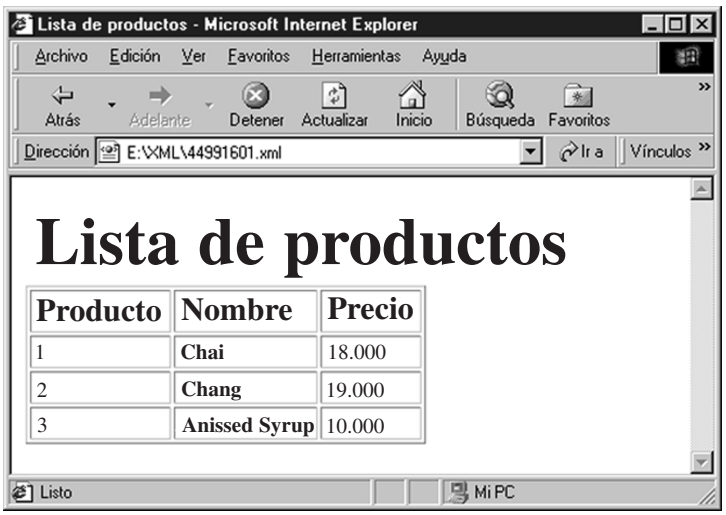

Figura 16.6: *Presentación de un documento XML en Internet Explorer, después de aplicarle un archivo XSL con instrucciones de formato más complejas.*

Listado 16.7: Un archivo XSL con directivas de formato

```
<xsl:stylesheet xmlns:xsl='http://www.w3.org/TR/WD-xsl'>
<xsl:template match='/'>
 <HTML>
 <HEAD>
 <Title>Lista de productos</Title>
 <STYLE>
   .subject {
     font-family: Garamond;
     font-weight: bold;
     font-size: 48;
   }
   .headings {
     font-family: Garamond;
     font-weight: bold;
     font-size: 24;
   }
   .productname {
     font-family: Garamond;
     font-weight: bold;
     font-size: normal;
   }
 </STYLE>
 </HEAD>
 <Body>
   <Table>
     <TR>
     <TD CLASS="subject">Lista de productos</TD>
     </TR>
   </Table>

 <Table border='1'>
   <TR>
     <TD CLASS="headings">Producto</TD>
     <TD CLASS="headings">Nombre</TD>
     <TD CLASS="headings">Precio</TD>
   </TR>
   <xsl:for-each select='Products/Product'>
     <TR>
       <TD><xsl:value-of select='@ProductID'/></TD>
       <TD CLASS="productname"><xsl:value-of select='@ProductName'/></TD>
       <TD><xsl:value-of select='@UnitPrice'/></TD>
     </TR>
   </xsl:for-each>
 </Table>
 </Body>
 </HTML>
```

Listado 16.7: continuación

```
</xsl:template>
</xsl:stylesheet>
```

El estudio de las posibilidades que ofrece este lenguaje poderoso y a la vez simple está más allá del alcance de este libro. Puede aprender más acerca de XML con el libro *XML by Example* (Benoît Marchal, QUE, ISBN 0-7897-2242-9).

Soporte de XML en SQL Server 2000

SQL Server 2000 es un nuevo servidor empresarial .NET, diseñado para proveer servicios *back-end* de datos a otros servidores empresariales .NET y aplicaciones .NET. En esta estrategia le toca un importante papel a SQL Server.

XML se puede usar en SQL Server 2000 de las siguientes formas:

- Obtener datos de SQL Server en formato XML, mediante la nueva extensión de Transact-SQL `FOR XML`. Este formato es conveniente cuando se trata de integrar estos datos con páginas ASP.
- Abrir datos expresados en XML como un conjunto de resultados desde Transact-SQL, mediante la nueva función `OPENXML`.
- Acceder a SQL Server 2000 a través de Internet mediante una conexión HTTP.
- Usar el proveedor mejorado OLE DB de SQL Server y la biblioteca de base de datos ADO con soporte extendido para XML para enviar consultas XML y recibir resultados en XML, mediante el nuevo objeto `Stream`.
- Usar *updategrams* XML para insertar, eliminar o modificar datos de SQL Server.
- Usar el componente de carga masiva XML para importar datos XML a SQL Server 2000. En este capítulo no nos ocuparemos de esta herramienta. Para aprender a usarla puede revisar la documentación instalada como parte del componente de soporte para XML e Internet en SQL Server (*SQL Server XML and Internet Support–Web Release 1*).

Este soporte para XML facilita el uso de SQL Server 2000 en Internet y su integración con otros servidores empresariales .NET, por ejemplo Microsoft Biztalk Server 2000.

El acceso a SQL Server a través de HTTP lo suministra una extensión de Internet Information Server (SQLISAPI.DLL) por medio de un directorio virtual especialmente definido.

El proveedor SQLOLEDB ha sido mejorado para suministrar diversos servicios:

- Traducción de plantillas XML a instrucciones `SELECT ... FOR XML`.
- Traducción de consultas XPath a instrucciones `SELECT ... FOR XML`.

- Traducción de *updategrams* XML a las correspondientes instrucciones Transact-SQL `INSERT`, `DELETE` y `UPDATE`.

En las secciones que siguen aprenderemos acerca del uso de estas nuevas prestaciones. Como ya hemos mencionado, esto no es más que una introducción a esta tecnología extremadamente amplia. Los ejemplos contenidos en este capítulo están pensados para guiar al lector a través de las primeras etapas de su aventura con XML.

Lectura de datos desde SQL Server en formato XML

SQL Server 2000 puede generar conjuntos de resultados directamente en formato XML. Esta opción simplifica el manejo de los datos desde el lado del cliente, sobre todo con páginas ASP.

Cuando SQL Server 2000 tiene que producir datos en formato XML, hay un componente interno que se encarga de traducir el conjunto de filas interno a un flujo de datos XML. El componente que se usará depende del modo de XML seleccionado.

SQL Server envía el flujo XML a la aplicación cliente, y para la presentación de los datos se usan las capacidades de formato que hubiera disponibles en el servidor o en la computadora cliente.

El analizador de consultas no puede interpretar correctamente datos en XML, porque no usa el analizador sintáctico de XML en lo absoluto; muestra los flujos XML como una secuencia de filas de texto, cada una de ellas de hasta 2.033 bytes de largo. El propósito de estas extensiones de Transact-SQL es permitir el acceso a datos provenientes de SQL Server en formato XML desde conexiones HTTP o desde aplicaciones web, y no desde el analizador de consultas.

NOTA

Más adelante en este capítulo, en la sección "Acceso a SQL Server a través de HTTP", aprenderemos a configurar Internet Information Server de modo que provea acceso a SQL Server por medio de HTTP. Puede seguir esas instrucciones para habilitar el acceso HTTP a SQL Server y luego ejecutar los ejercicios de esta sección mediante Internet Explorer, para leer directamente los resultados en XML.

Uso de la cláusula `FOR XML`

Para obtener datos en formato XML se usa la nueva extensión `FOR XML` de la instrucción `SELECT`. El listado 16.8 muestra una sencilla instrucción `SELECT` que obtiene datos en formato XML, y cómo se ve en el analizador de consultas la salida producida.


EJEMPLO

Listado 16.8: Uso de `FOR XML` para producir resultados en formato XML

```
USE Northwind
GO

SELECT CategoryID, CategoryName, Description
FROM Categories
FOR XML AUTO
```

Listado 16.8: continuación

SALIDA

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B
------------------------------------------------------
<Categories CategoryID="1" CategoryName="Beverages"
Description="Soft drinks, coffees, teas, beers, and ales"/>
<Categories CategoryID="2" CategoryName="Condiments"
Description="Sweet and savory sauces, relishes, spreads, and seasonings"/>
<Categories CategoryID="3" CategoryName="Confections"
Description="Desserts, candies, and sweet breads"/>
<Categories CategoryID="4" CategoryName="Dairy
Products"Description="Cheeses"/><Categories CategoryID="5"
CategoryName="Grains/Cereals"
Description="Breads, crackers, pasta, and cereal"/>
<Categories CategoryID="6" CategoryName="Meat/Poultry"
Description="Prepared meats"/><Categories CategoryID="7"
CategoryName="Produce" Description="Dried fruit and bean curd"/>
<Categories CategoryID="8" CategoryName="Seafood"
Description="Seaweed and fish"/>

(8 filas afectadas)
```

PRECAUCIÓN

Ajuste la cantidad máxima de caracteres por columna en el analizador de consultas a un valor mayor que 2.033, o algunos de los ejemplos de esta sección saldrán cortados.

Para ajustar esta opción puede ir a Herramientas, Opciones y entrar a la ficha Resultados.

En el listado 16.8 y en su salida podemos ver ciertos detalles de interés:

- Las únicas palabras clave adicionales que le enviamos a SQL Server son `FOR XML AUTO` al final de una instrucción `SELECT` común.
- Vista en el analizador de consultas, la salida muestra una sola columna con un nombre extravagante: `XML` seguido de una expresión tipo GUID (identificador global único).
- Toda la salida va a una sola línea, pero aun así la cuenta de filas afectadas indica 8 filas. La línea de salida es muy larga e incluye todo el conjunto de resultados.

Copie la línea del panel de resultados del analizador de consultas al bloc de notas y agregue algunos retornos de carro y tabulaciones en los lugares apropiados, para dar a los resultados una apariencia más propia de un documento XML (véase el listado 16.9).

Listado 16.9: La salida de una consulta que usó la cláusula `FOR XML`, luego de formatearla manualmente

```
<Categories
 CategoryID="1"
 CategoryName="Beverages"
 Description="Soft drinks, coffees, teas, beers, and ales"/>
<Categories
 CategoryID="2"
 CategoryName="Condiments"
 Description="Sweet and savory sauces, relishes, spreads, and seasonings"/>
<Categories
 CategoryID="3"
 CategoryName="Confections"
 Description="Desserts, candies, and sweet breads"/>
<Categories
 CategoryID="4"
 CategoryName="Dairy Products"
 Description="Cheeses"/>
<Categories
 CategoryID="5"
 CategoryName="Grains/Cereals"
 Description="Breads, crackers, pasta, and cereal"/>
<Categories
 CategoryID="6"
 CategoryName="Meat/Poultry"
 Description="Prepared meats"/>
<Categories
 CategoryID="7"
 CategoryName="Produce"
 Description="Dried fruit and bean curd"/>
<Categories
 CategoryID="8"
 CategoryName="Seafood"
 Description="Seaweed and fish"/>
```

Guarde este documento en un archivo con el nombre `xmlListing9.xml` y trate de abrirlo en Internet Explorer. Recibirá el mensaje de error de la figura 16.7, porque esta salida XML no es un documento XML bien formado, contiene más de un elemento raíz. Esta clase de documento recibe el nombre de *fragmento de documento XML*.

Vuelva a editar el documento y agregue un nuevo nodo raíz (véase el listado 16.10). Ahora podrá abrir este documento en Internet Explorer sin ningún problema.

Figura 16.7: *La salida directa de la cláusula* `FOR XML` *no es un documento XML bien formado.*

Listado 16.10: Agregarle un nodo raíz convierte la salida XML en un documento XML bien formado

```
<CategoryList>
 <Categories
   CategoryID="1"
   CategoryName="Beverages"
   Description="Soft drinks, coffees, teas, beers, and ales"/>
 <Categories
   CategoryID="2"
   CategoryName="Condiments"
   Description="Sweet and savory sauces, relishes, spreads, and seasonings"/>
 <Categories
   CategoryID="3"
   CategoryName="Confections"
   Description="Desserts, candies, and sweet breads"/>
 <Categories
   CategoryID="4"
   CategoryName="Dairy Products"
   Description="Cheeses"/>
 <Categories
   CategoryID="5"
   CategoryName="Grains/Cereals"
   Description="Breads, crackers, pasta, and cereal"/>
 <Categories
   CategoryID="6"
   CategoryName="Meat/Poultry"
   Description="Prepared meats"/>
```

Listado 16.10: continuación

```
 <Categories
   CategoryID="7"
   CategoryName="Produce"
   Description="Dried fruit and bean curd"/>
 <Categories
   CategoryID="8"
   CategoryName="Seafood"
   Description="Seaweed and fish"/>
</CategoryList>
```

Sin embargo, se pueden usar alias para producir una salida que mejore los nombres de los elementos, como en la consulta del listado 16.11. Una convención de nombres que mejoraría la legibilidad sería usar el nombre de la tabla en singular para denominar cada elemento.

Como ejemplo podemos usar el alias `Category` para la tabla `Categories`, de modo que cada elemento de la salida XML lleve el nombre `Category`. El listado 16.12 muestra la salida, después de agregar el nodo raíz `Categories` para completar el fragmento XML producido por esta consulta.

Listado 16.11: Uso de alias de tabla y de columna para mejorar el uso de nombres en la salida XML

```
USE Northwind
GO

SELECT CategoryID as ID,
CategoryName as Name,
Description
FROM Categories as Category
FOR XML AUTO
```

Listado 16.12: La salida del listado 16.11, después de agregarle el nodo raíz `Categories`

```
<Categories>
 <Category
   ID="1"
   Name="Beverages"
   Description="Soft drinks, coffees, teas, beers, and ales"/>
 <Category
   ID="2"
   Name="Condiments"
   Description="Sweet and savory sauces, relishes, spreads, and seasonings"/>
 <Category
   ID="3"
   Name="Confections"
   Description="Desserts, candies, and sweet breads"/>
 <Category
   ID="4"
```

Listado 16.12: continuación

```
   Name="Dairy Products"
   Description="Cheeses"/>
 <Category
   ID="5"
   Name="Grains/Cereals"
   Description="Breads, crackers, pasta, and cereal"/>
 <Category
   ID="6"
   Name="Meat/Poultry"
   Description="Prepared meats"/>
 <Category
   ID="7"
   Name="Produce"
   Description="Dried fruit and bean curd"/>
 <Category
   ID="8"
   Name="Seafood"
   Description="Seaweed and fish"/>
</Categories>
```

En los ejemplos precedentes, para la salida XML seleccionamos el modo AUTO. SQL Server admite los modos siguientes:

- RAW: produce un flujo XML con elementos independientes para cada fila, donde cada elemento recibe el nombre row (independientemente del nombre de la tabla).
- AUTO: produce un flujo XML con elementos independientes para cada fila, donde cada elemento recibe el nombre de la tabla de la que se extrajeron los datos.
- EXPLICIT: produce una tabla universal que permite un control más estrecho de la forma de organización de los datos XML.

NOTA

El modo EXPLICIT excede los objetivos de este capítulo. Si está interesado en obtener información detallada acerca de este modo de XML puede buscar esa información y ejemplos relacionados en los libros en pantalla o asistir al curso curricular oficial de Microsoft 2091A, "Building XML-Enabled Applications Using Microsoft SQL Server 2000".

El listado 16.13 muestra la misma consulta que el 16.11, pero en modo RAW, con su salida no formateada. Como puede apreciar en la salida, la principal diferencia está en el nombre de los elementos, que en este caso siempre es row.

EJEMPLO

Listado 16.13: Uso de FOR XML RAW para producir resultados en formato XML

```
USE Northwind
GO
```

Listado 16.13: continuación

```
SELECT CategoryID as ID,
CategoryName as Name,
Description
FROM Categories as Category
FOR XML RAW
```

SALIDA

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B
------------------------------------------------
<row ID="1" Name="Beverages"
Description="Soft drinks, coffees, teas, beers, and ales"/>
<row ID="2" Name="Condiments"
Description="Sweet and savory sauces, relishes, spreads, and seasonings"/>
<row ID="3" Name="Confections"
Description="Desserts, candies, and sweet breads"/>
<row ID="4" Name="Dairy Products"
Description="Cheeses"/><row ID="5" Name="Grains/Cereals"
Description="Breads, crackers, pasta, and cereal"/>
<row ID="6" Name="Meat/Poultry" Description="Prepared meats"/>
<row ID="7" Name="Produce" Description="Dried fruit and bean curd"/>
<row ID="8" Name="Seafood" Description="Seaweed and fish"/>

(8 filas afectadas)
```

Sin embargo, si la instrucción SELECT lee datos de más de una tabla hay una gran diferencia entre cómo trabajan los modos RAW y AUTO. El listado 16.14 muestra la misma consulta en ambos formatos, con sus correspondientes salidas.

EJEMPLO

Listado 16.14: Comparación de los modos RAW y AUTO cuando se ejecutan consultas sobre varias tablas

```
USE Northwind
GO

SELECT CategoryName,
ProductName
FROM Categories as Category
JOIN Products as Product
ON Product.CategoryID = Category.CategoryID
WHERE Product.CategoryID < 3
ORDER BY CategoryName, ProductName
FOR XML RAW

SELECT CategoryName,
ProductName
FROM Categories as Category
JOIN Products as Product
ON Product.CategoryID = Category.CategoryID
WHERE Product.CategoryID < 3
ORDER BY CategoryName, ProductName
FOR XML AUTO
```

Listado 16.14: continuación

SALIDA

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B
------------------------------------------------
<row CategoryName="Beverages" ProductName="Chai"/>
<row CategoryName="Beverages" ProductName="Chang"/>
<row CategoryName="Beverages" ProductName="Chartreuse verte"/>
<row CategoryName="Beverages" ProductName="Côte de Blaye"/>
<row CategoryName="Beverages" ProductName="Guaraná Fantástica"/>
<row CategoryName="Beverages" ProductName="Ipoh Coffee"/>
<row CategoryName="Beverages" ProductName="Lakkalikööri"/>
<row CategoryName="Beverages" ProductName="Laughing Lumberjack Lager"/>
<row CategoryName="Beverages" ProductName="Outback Lager"/>
<row CategoryName="Beverages" ProductName="Rhönbräu Klosterbier"/>
<row CategoryName="Beverages" ProductName="Sasquatch Ale"/>
<row CategoryName="Beverages" ProductName="Steeleye Stout"/>
<row CategoryName="Condiments" ProductName="Aniseed Syrup"/>
<row CategoryName="Condiments"
ProductName="Chef Anton&amp;apos;s Cajun Seasoning"/>
<row CategoryName="Condiments" ProductName="Chef Anton&amp;apos;s Gumbo Mix"/>
<row CategoryName="Condiments" ProductName="Genen Shouyu"/>
<row CategoryName="Condiments"
ProductName="Grandma&amp;apos;s Boysenberry Spread"/>
<row CategoryName="Condiments" ProductName="Gula Malacca"/>
<row CategoryName="Condiments"
ProductName="Louisiana Fiery Hot Pepper Sauce"/>
<row CategoryName="Condiments" ProductName="Louisiana Hot Spiced Okra"/>
<row CategoryName="Condiments" ProductName="Northwoods Cranberry Sauce"/>
<row CategoryName="Condiments"
ProductName="Original Frankfurter grüne Soße"/>
<row CategoryName="Condiments" ProductName="Sirop d&amp;apos;érable"/>
<row CategoryName="Condiments" ProductName="Vegie-spread"/>

(24 filas afectadas)

XML_F52E2B61-18A1-11d1-B105-00805F49916B
------------------------------------------------
<Category CategoryName="Beverages">
<Product ProductName="Chai"/>
<Product ProductName="Chang"/>
<Product ProductName="Chartreuse verte"/>
<Product ProductName="Côte de Blaye"/>
<Product ProductName="Guaraná Fantástica"/>
<Product ProductName="Ipoh Coffee"/>
<Product ProductName="Lakkalikööri"/>
<Product ProductName="Laughing Lumberjack Lager"/>
<Product ProductName="Outback Lager"/>
<Product ProductName="Rhönbräu Klosterbier"/>
```

Listado 16.14: continuación

```
<Product ProductName="Sasquatch Ale"/>
<Product ProductName="Steeleye Stout"/>
</Category>
<Category CategoryName="Condiments">
<Product ProductName="Aniseed Syrup"/>
<Product ProductName="Chef Anton&amp;apos;s Cajun Seasoning"/>
<Product ProductName="Chef Anton&amp;apos;s Gumbo Mix"/>
<Product ProductName="Genen Shouyu"/>
<Product ProductName="Grandma&amp;apos;s Boysenberry Spread"/>
<Product ProductName="Gula Malacca"/>
<Product ProductName="Louisiana Fiery Hot Pepper Sauce"/>
<Product ProductName="Louisiana Hot Spiced Okra"/>
<Product ProductName="Northwoods Cranberry Sauce"/>
<Product ProductName="Original Frankfurter grüne Soße"/>
<Product ProductName="Sirop d&amp;apos;érable"/>
<Product ProductName="Vegie-spread"/>
</Category>

(24 filas afectadas)
```

En la salida precedente se puede ver que el modo AUTO muestra automáticamente los datos organizados en forma jerárquica. Esto es adecuado para la mayoría de las aplicaciones, porque de esta manera el usuario puede expandir la jerarquía para examinar niveles más profundos de detalle, si fuera necesario.

Con el modo AUTO se puede usar la opción ELEMENTS para mostrar los atributos como elementos (véase el listado 16.15).

EJEMPLO

Listado 16.15: Uso de la opción ELEMENTS para mostrar los atributos como elementos

```
USE Northwind
GO

SELECT CategoryName,
ProductName
FROM Categories as Category
JOIN Products as Product
ON Product.CategoryID = Category.CategoryID
WHERE Product.CategoryID <3
ORDER BY CategoryName, ProductName
FOR XML AUTO, ELEMENTS
```

SALIDA

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B
------------------------------------------------
<Category>
<CategoryName>Beverages</CategoryName>
<Product>
<ProductName>Chai</ProductName>
```

Listado 16.15: continuación

```
</Product>
<Product>
<ProductName>Chang</ProductName>
</Product>
<Product>
<ProductName>Chartreuse verte</ProductName>
</Product>
<Product>
<ProductName>Côte de Blaye</ProductName>
</Product>
<Product>
<ProductName>Guaraná Fantástica</ProductName>
</Product>
<Product>
<ProductName>Ipoh Coffee</ProductName>
</Product>
<Product>
<ProductName>Lakkalikööri</ProductName>
</Product>
<Product>
<ProductName>Laughing Lumberjack Lager</ProductName>
</Product>
<Product>
<ProductName>Outback Lager</ProductName>
</Product>
<Product>
<ProductName>Rhönbräu Klosterbier</ProductName>
</Product>
<Product>
<ProductName>Sasquatch Ale</ProductName>
</Product>
<Product>
<ProductName>Steeleye Stout</ProductName>
</Product>
</Category>
<Category>
<CategoryName>Condiments</CategoryName>
<Product>
<ProductName>Aniseed Syrup</ProductName>
</Product>
<Product>
<ProductName>Chef Anton&amp;apos;s Cajun Seasoning</ProductName>
</Product>
<Product>
<ProductName>Chef Anton&amp;apos;s Gumbo Mix</ProductName>
</Product>
```

Listado 16.15: continuación

```
<Product>
<ProductName>Genen Shouyu</ProductName>
</Product>
<Product>
<ProductName>Grandma&apos;s Boysenberry Spread</ProductName>
</Product>
<Product>
<ProductName>Gula Malacca</ProductName>
</Product>
<Product>
<ProductName>Louisiana Fiery Hot Pepper Sauce</ProductName>
</Product>
<Product>
<ProductName>Louisiana Hot Spiced Okra</ProductName>
</Product>
<Product>
<ProductName>Northwoods Cranberry Sauce</ProductName>
</Product>
<Product>
<ProductName>Original Frankfurter grüne Soße</ProductName>
</Product>
<Product>
<ProductName>Sirop d&apos;érable</ProductName>
</Product>
<Product>
<ProductName>Vegie-spread</ProductName>
</Product>
</Category>

(24 filas afectadas)
```

Se puede usar la opción XMLDATA para que la salida produzca un esquema XML, como en el listado 16.16.

Listado 16.16: Uso de la opción XMLDATA para incluir un esquema XML en la salida

```
USE Northwind
GO

SELECT CategoryName,
ProductName
FROM Categories as Category
JOIN Products as Product
ON Product.CategoryID = Category.CategoryID
WHERE Product.CategoryID <3
ORDER BY CategoryName, ProductName
FOR XML AUTO, XMLDATA
```

Listado 16.16: continuación

SALIDA

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B
------------------------------------------
<Schema name="Schema2"
xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
<ElementType name="Category"
content="eltOnly" model="closed" order="many">
<element type="Product" maxOccurs="*"/>
<AttributeType name="CategoryName" dt:type="string"/>
<attribute type="CategoryName"/>
</ElementType>
<ElementType name="Product"
content="empty" model="closed">
<AttributeType name="ProductName" dt:type="string"/>
<attribute type="ProductName"/>
</ElementType>
</Schema>
<Category xmlns="x-schema:#Schema2" CategoryName="Beverages">
<Product ProductName="Chai"/>
<Product ProductName="Chang"/>
<Product ProductName="Chartreuse verte"/>
<Product ProductName="Côte de Blaye"/>
<Product ProductName="Guaraná Fantástica"/>
<Product ProductName="Ipoh Coffee"/>
<Product ProductName="Lakkalikööri"/>
<Product ProductName="Laughing Lumberjack Lager"/>
<Product ProductName="Outback Lager"/>
<Product ProductName="Rhönbräu Klosterbier"/>
<Product ProductName="Sasquatch Ale"/>
<Product ProductName="Steeleye Stout"/>
</Category>
<Category xmlns="x-schema:#Schema2" CategoryName="Condiments">
<Product ProductName="Aniseed Syrup"/>
<Product ProductName="Chef Anton&amp;apos;s Cajun Seasoning"/>
<Product ProductName="Chef Anton&amp;apos;s Gumbo Mix"/>
<Product ProductName="Genen Shouyu"/>
<Product ProductName="Grandma&amp;apos;s Boysenberry Spread"/>
<Product ProductName="Gula Malacca"/>
<Product ProductName="Louisiana Fiery Hot Pepper Sauce"/>
<Product ProductName="Louisiana Hot Spiced Okra"/>
<Product ProductName="Northwoods Cranberry Sauce"/>
<Product ProductName="Original Frankfurter grüne Soße"/>
<Product ProductName="Sirop d&amp;apos;érable"/>
<Product ProductName="Vegie-spread"/>
</Category>

(24 filas afectadas)
```

Como puede ver, la salida del listado 16.16 es igual a la de la segunda consulta del listado 16.14. La única diferencia es la presencia de la información del esquema al principio de la salida. Esta información se puede usar para crear un archivo XDR (*XML Data Reduced*, XML simplificado).

PRECAUCIÓN

Si trata de leer las salidas de los listados 16.14 a 16.16 directamente en Internet Explorer se producirán errores de sintaxis de XML, a menos que añada un nodo raíz válido y guarde estas salidas en formato Unicode desde el Bloc de notas.

Lectura de datos en XML de SQL Server desde páginas ASP

Aunque este no es un libro sobre ASP, es interesante mostar al menos un ejemplo de cómo escribir una sencilla aplicación ASP (*Active Server Pages*, páginas activas de servidor) para leer datos de SQL Server en formato XML.

Para una comprensión cabal de la creación de páginas ASP puede leer el libro *Active Server Pages 3.0 con ejemplos*, (Bob Reselman, Prentice Hall, ISBN 987-9460-11-1). Trabajar con ASP requiere conocimiento de *scripting* y cierta comprensión de HTML.

En esta sección aprenderemos a escribir una sencilla página ASP que leerá el documento XML producido por la segunda consulta del listado 16.14.

Para escribir la página ASP puede usar el bloc de notas o cualquier otro editor de texto. El listado 16.17 contiene el código completo de la página ASP SQLXML.ASP. Para que este código funcione, debe guardar el archivo en un directorio que esté bajo la administración de Internet Information Server. En este ejemplo crearemos un directorio con el nombre `XML` dentro de la ruta predeterminada para las páginas web (`c:\InetPub\wwwroot`).

El *script* contiene dos áreas principales:

- *Script de servidor*. Esta sección incluye la parte principal de la lógica de la programación. Aquí nos conectamos a SQL Server para obtener la información requerida. Estas instrucciones se ejecutan en la computadora servidora (Internet Information Server), que no las envía a la computadora cliente.
- *Script de cliente*. Esta sección muestra los resultados en el lugar y la forma que necesitamos. Estas instrucciones se ejecutan en la computadora cliente.

Ambas áreas están convenientemente señaladas en el código con comentarios.

EJEMPLO

Listado 16.17: Una sencilla página ASP que lee una consulta SQL con `FOR XML AUTO`

```
<%@ LANGUAGE = VBScript %>
<% Option Explicit %>
```

Listado 16.17: continuación

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" content="text/html" charset="UTF-8"/>
<TITLE>SQL-XML con ejemplos - SQLXML.asp</TITLE>

<%
 ' ESTE ES EL SCRIPT DE SERVIDOR
 ' Esta parte no será visible en el cliente

 ' Algunas constantes para que el código sea más legible.
 ' Algunas de estas constantes se pueden encontrar en el
 ' archivo adovbs.inc

 const adUseClient = 3
 const adWriteChar = 0
 const adWriteLine = 1
 const adExecuteStream = 1024

 ' Creación de un objeto ADO Connection

 Dim adoConn
 Set adoConn = Server.CreateObject("ADODB.Connection")

 ' Definición de la cadena de conexión para conectarse a una instancia
 ' válida de SQL Server 2000 y a la base de datos Northwind.
 ' Se indica procesamiento en el cliente, que en este caso será
 ' el servidor IIS

 ' Nota: si la instancia predeterminada es SQL Server 7, este código no
 ' funcionará, a menos que tenga instalado el soporte XML para SQL Server 7.0

 Dim sConn
 sConn = "Provider=sqloledb;"
 sConn = sConn & "Data Source=MSSQLFGG\S2K;"
 sConn = sConn & "Initial Catalog=Northwind;"
 sConn = sConn & "User ID=sa"
 adoConn.ConnectionString = sConn
 adoConn.CursorLocation = adUseClient

 ' Abrimos la conexión

 adoConn.Open

 ' Creamos un objeto ADO Command,
 ' para enviar la consulta XML y recibir los resultados en XML
```

Listado 16.17: continuación

```
Dim adoCmd
Set adoCmd = Server.CreateObject("ADODB.Command")
Set adoCmd.ActiveConnection = adoConn

' Comienzo de la petición XML

Dim sQuery
sQuery = "<Catalog xmlns:sql='urn:schemas-microsoft-com:xml-sql'>"
sQuery = sQuery & "<sql:query>"

' Esta es la consulta del listado 16.14

sQuery = sQuery & "SELECT CategoryName, "
sQuery = sQuery & "ProductName "
sQuery = sQuery & "FROM Categories as Category "
sQuery = sQuery & "JOIN Products as Product "
sQuery = sQuery & "ON Product.CategoryID = Category.CategoryID "

' Observe que tenemos que cambiar el símbolo < a &lt;
' porque el carácter < está reservado en XML
' y en este listado, la consulta irá encerrada en un flujo XML

sQuery = sQuery & "WHERE Product.CategoryID &lt; 3 "
sQuery = sQuery & "ORDER BY CategoryName, ProductName "
sQuery = sQuery & "FOR XML AUTO"

' Ya podemos terminar la petición XML

sQuery = sQuery & "</sql:query></Catalog>"

' Creamos y abrimos el objeto Stream

Dim adoStreamQuery
Set adoStreamQuery = Server.CreateObject("ADODB.Stream")
adoStreamQuery.Open

' Escribimos la consulta XML dentro del objeto Stream

adoStreamQuery.WriteText sQuery, adWriteChar
adoStreamQuery.Position = 0

' Seleccionamos el objeto Stream como comando a ejecutar
' Observe el GUID para la propiedad Dialect, que en
' este caso representa el formato MSSQLXML
```

Listado 16.17: continuación

```
 adoCmd.CommandStream = adoStreamQuery
 adoCmd.Dialect = "{5D531CB2-E6Ed-11D2-B252-00C04F681B71}"

 ' Ahora indicaremos que la salida vaya al objeto Response,
 ' que también es un objeto Stream

 adoCmd.Properties("Output Stream") = Response

 ' Definimos la salida XML como una isla de datos con el
 ' nombre CatalogIsle, que se usará para la ejecución
 ' del lado del cliente

 Response.write "<XML ID=CatalogIsle>"
 adoCmd.Execute , , adExecuteStream
 Response.write "</XML>"
%>

<SCRIPT language="VBScript" For="window" Event="onload">

 ' Este es el script de cliente,
 ' que será visible para el navegador cliente

 Dim xmlDoc
 dim OutputXML

 ' Obtenemos una referencia a la isla de datos
 ' que contiene los resultados XML

 Set xmlDoc = CatalogIsle.XMLDocument

 Dim root, CChild, PChild

 ' Obtenemos una referencia al nodo raíz del documento XML

 Set root = xmlDoc.documentElement

 ' Exploramos el primer nivel del árbol, leyendo cada categoría.
 ' Mostramos el atributo CategoryName en negrita

 For each CChild in root.childNodes
   OutputXML = document.all("log").innerHTML
   document.all("log").innerHTML = OutputXML & "<LI><B>" &
     CChild.getAttribute("CategoryName") & "</B></LI>"

   ' Para cada categoría,
   ' exploramos el siguiente nivel del árbol,
```

Listado 16.17: continuación

```
    ' leyendo cada producto perteneciente a la categoría.
    ' Mostramos el nombre del producto en tipografía normal

    For each PChild in CChild.childNodes
      OutputXML = document.all("log").innerHTML
      document.all("log").innerHTML = OutputXML & "<UL>" &
        PChild.getAttribute("ProductName") & "</UL>"
    Next
  Next
</SCRIPT>
</HEAD>
<BODY>
 <H1>SQL-XML con ejemplos</H1>
 <H3>Este es el cat&aacute;logo de productos
     para las categor&iacute;as 1 y 2</H3>
 <UL id=log>
 </UL>
</BODY>
</HTML>
```

Si observa el listado 16.17 encontrará la siguiente línea al principio del archivo:

```
<META HTTP-EQUIV="Content-Type" content="text/html" charset="UTF-8"/>
```

Es importante indicar `UTF-8` (Unicode Transformation Format que codifica cada código UNICODE como una secuencia de uno a cuatro bytes) conjunto de caracteres, ya que los datos que presentaremos en este caso contienen caracteres que no se encuentran en el conjunto de caracteres estándar.

El proceso seguido por esta página ASP es:

1. Creamos un objeto `Connection`, llamado en este ejemplo `adoConn`, y lo conectamos a SQL Server.
2. Creamos un objeto `Stream`, llamado en este ejemplo `AdoStreamQuery`, para definir la consulta XML que enviaremos a SQL Server.
3. Creamos un objeto `Command`, que en este ejemplo se llama `adoCmd`, para enviar a SQL Server la consulta definida por el objeto `Stream` y mandar los resultados directamente al objeto `Response`.
4. Ejecutamos el objeto `Command`, incluyendo los resultados en una isla de datos llamada `CatalogIsle`. De esta forma, el *script* de cliente tendrá un acceso fácil a los datos.

Lo único que tiene el *script* de cliente es un bucle doble para recorrer las categorías y todos los productos de cada categoría.

A lo largo del código hallará gran cantidad de comentarios que describen el propósito de cada instrucción.

Si abre este archivo con Internet Explorer (usando el URL `http://localhost/XML/SQLXML.ASP`) podrá ver la misma salida que en la figura 16.8.

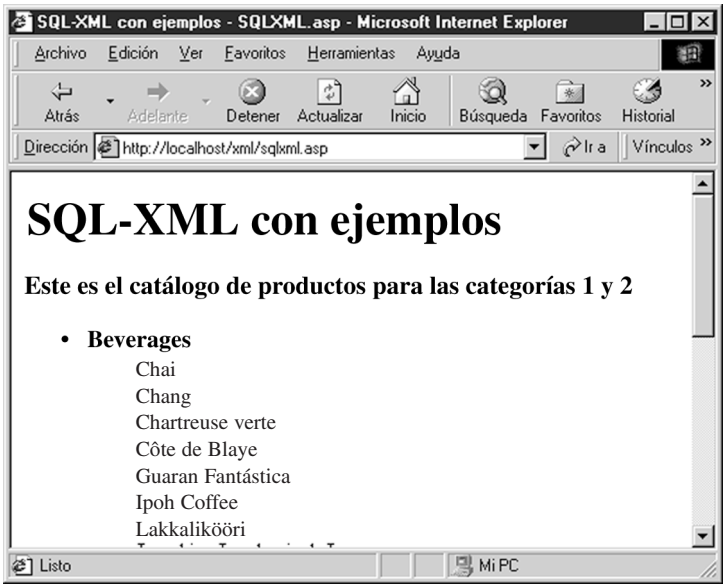

Figura 16.8: *Ejecución de `SQLXML.ASP` en Internet Information Server para leer datos desde SQL Server.*

Ahora, veamos lo que el usuario encuentra si abre el *script* de cliente. En Internet Explorer, vaya al menú Ver y elija Código fuente. Se abrirá el bloc de notas y mostrará el código fuente de la página, que es el que muestra el listado 16.18. Como puede ver, el *script* de servidor (incluyendo la cadena de conexión y la consulta SQL que lee los datos) está oculto desde el punto de vista del cliente.

Observe en el listado 16.18 que la primera parte incluye la isla de datos que contiene el resultado de la consulta en formato XML. Además, en el documento HTML final se han eliminado automáticamente todos los comentarios.

SALIDA

Listado 16.18: Este es el código fuente de la página HTML que recibe el cliente

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" content="text/html" charset="UTF-8"/>
<TITLE>SQL-XML con ejemplos - SQLXML.asp</TITLE>

<XML ID=CatalogIsle>
<Catalog xmlns:sql="urn:schemas-microsoft-com:xml-sql">
<Category CategoryName="Beverages">
```

Listado 16.18: continuación

```
<Product ProductName="Chai"/>
<Product ProductName="Chang"/>
<Product ProductName="Chartreuse verte"/>
<Product ProductName="Côte de Blaye"/>
<Product ProductName="Guaraná Fantástica"/>
<Product ProductName="Ipoh Coffee"/>
<Product ProductName="Lakkalikööri"/>
<Product ProductName="Laughing Lumberjack Lager"/>
<Product ProductName="Outback Lager"/>
<Product ProductName="Rhönbräu Klosterbier"/>
<Product ProductName="Sasquatch Ale"/>
<Product ProductName="Steeleye Stout"/>
</Category>
<Category CategoryName="Condiments">
<Product ProductName="Aniseed Syrup"/>
<Product ProductName="Chef Anton&amp;apos;s Cajun Seasoning"/>
<Product ProductName="Chef Anton&amp;apos;s Gumbo Mix"/>
<Product ProductName="Genen Shouyu"/>
<Product ProductName="Grandma&amp;apos;s Boysenberry Spread"/>
<Product ProductName="Gula Malacca"/>
<Product ProductName="Louisiana Fiery Hot Pepper Sauce"/>
<Product ProductName="Louisiana Hot Spiced Okra"/>
<Product ProductName="Northwoods Cranberry Sauce"/>
<Product ProductName="Original Frankfurter grüne Soße"/>
<Product ProductName="Sirop d&amp;apos;érable"/>
<Product ProductName="Vegie-spread"/>
</Category>
</Catalog>
</XML>

<SCRIPT language="VBScript" For="window" Event="onload">

 Dim xmlDoc
 dim OutputXML
 Set xmlDoc = CatalogIsle.XMLDocument

 Dim root, CChild, PChild
 Set root = xmlDoc.documentElement

 For each CChild in root.childNodes
   OutputXML = document.all("log").innerHTML
   document.all("log").innerHTML = OutputXML & "<LI><B>" &
CChild.getAttribute("CategoryName") & "</B></LI>"

   For each PChild in CChild.childNodes
     OutputXML = document.all("log").innerHTML
```

Listado 16.18: continuación

```
      document.all("log").innerHTML = OutputXML & "<UL>" &
PChild.getAttribute("ProductName") & "</UL>"
   Next
 Next
</SCRIPT>
</HEAD>
<BODY>
 <H1>SQL-XML con ejemplos</H1>
 <H3>Este es el cat&aacute;logo de productos
      para las categor&iacute;as 1 y 2</H3>
 <UL id=log>
 </UL>
</BODY>
</HTML>
```

Ahora puede usar el listado 16.17 como una plantilla para presentar sus propias consultas.

Uso de datos XML desde Transact-SQL

En la sección precedente aprendimos a obtener datos de SQL Server en formato XML. En esta sección aprenderemos a usar datos expresados en XML dentro de SQL Server como parte de cualquier secuencia de comandos, procedimiento almacenado o desencadenador.

SQL Server 2000 implementa una nueva función `OPENXML` que lee documentos XML como conjuntos de resultados, de modo que se los puede usar en la cláusula `FROM` de cualquier instrucción del lenguaje de manipulación de datos (DML).

El proceso de lectura de datos XML implica los siguientes pasos:

1. El proceso recibe el documento XML, por lo general como parámetro de un procedimiento almacenado o como una cadena almacenada en una tabla.
2. El proceso llama al procedimiento almacenado extendido `sp_xml_preparedocument` para crear una representación en memoria del documento XML. SQL Server crea en memoria una estructura en forma de árbol que representa el documento XML y devuelve un entero, que actúa como controlador (*handle*) de la estructura de árbol.
3. Se usa la función `OPENXML` para leer los datos XML en formato de conjunto de resultados, en cualquier instrucción DML de Transact-SQL, como si se tratara de un conjunto de resultados.
4. Se usa el procedimiento almacenado extendido `sp_xml_removedocument` para destruir la representación en memoria del documento XML, cuando ya no es necesaria.

NOTA

Ambos procedimientos, `sp_xml_preparedocument` y `sp_xml_removedocument`, son procedimientos extendidos que llaman a bibliotecas externas. El hecho de que su nombre comience con sp implica que estos procedimientos son globales y que se los puede llamar desde cualquier base de datos, sin indicar `master` como nombre de la base de datos.

El listado 16.19 contiene un ejemplo que usa el documento XML del listado 16.1.

EJEMPLO

Listado 16.19: Uso de la función `OPENXML` para leer un documento XML desde una secuencia de comandos Transact-SQL

```
DECLARE @xml varchar(8000)

-- Este es el documento XML

SET @xml = '<Products>'
SET @xml = @xml + '<Product ProductID="1" '
SET @xml = @xml + 'ProductName="Chai" UnitPrice="18.0000"/>'
SET @xml = @xml + '<Product ProductID="2" '
SET @xml = @xml + 'ProductName="Chang" UnitPrice="19.0000"/>'
SET @xml = @xml + '<Product ProductID="3" '
SET @xml = @xml + 'ProductName="Aniseed Syrup" UnitPrice="10.0000"/>'
SET @xml = @xml + '</Products>'

-- Creamos una estructura en forma de árbol con el documento XML
-- y obtenemos su controlador

DECLARE @iDoc int

EXEC sp_xml_preparedocument @iDoc OUTPUT, @xml

PRINT CHAR(10)
   + 'Se creó el árbol, handle = '
   + CONVERT(varchar(10), @iDoc)
   + CHAR(10)

SELECT *
FROM OPENXML(@iDoc, 'Products', 1)

EXEC sp_xml_removedocument @idoc
```

SALIDA

```
Se creó el árbol, handle = 13

id    parentid nodetype localname    prefix  namespaceuri datatype  prev text
----- -------- -------- ------------ ------  ------------ --------  ---- -----------
0.00  NULL     1.00     Products     NULL    NULL         NULL      NULL NULL
2.00  0.00     1.00     Product      NULL    NULL         NULL      NULL NULL
3.00  2.00     2.00     ProductID    NULL    NULL         NUL       NULL NULL
14.00 3.00     3.00     #text        NULL    NULL         NUL       NULL 1
```

Listado 16.19: continuación

```
4.00  2.00      2.00      ProductName   NULL    NULL          NULL      NULL  NULL
15.00 4.00      3.00      #text         NULL    NULL          NULL      NULL  Chai
5.00  2.00      2.00      UnitPrice     NULL    NULL          NULL      NULL  NULL
16.00 5.00      3.00      #text         NULL    NULL          NULL      NULL  18.0000
6.00  0.00      1.00      Product       NULL    NULL          NULL      2.00  NULL
7.00  6.00      2.00      ProductID     NULL    NULL          NULL      NULL  NULL
17.00 7.00      3.00      #text         NULL    NULL          NULL      NULL  2
8.00  6.00      2.00      ProductName   NULL    NULL          NULL      NULL  NULL
18.00 8.00      3.00      #text         NULL    NULL          NULL      NULL  Chang
9.00  6.00      2.00      UnitPrice     NULL    NULL          NULL      NULL  NULL
19.00 9.00      3.00      #text         NULL    NULL          NULL      NULL  19.0000
10.00 0.00      1.00      Product       NULL    NULL          NULL      6.00  NULL
11.00 10.00     2.00      ProductID     NULL    NULL          NULL      NULL  NULL
20.00 11.00     3.00      #text         NULL    NULL          NULL      NULL  3
12.00 10.00     2.00      ProductName   NULL    NULL          NULL      NULL  NULL
21.00 12.00     3.00      #text         NULL    NULL          NULL      NULL  Aniseed Syrup
13.00 0.00      2.00      UnitPrice     NULL    NUL           NULL      NULL  NULL
22.00 13.00     3.00      #text         NULL    NULL          NULL      NULL  10.0000

(22 filas afectadas)
```

La salida del listado 16.19 no es exactamente lo que queríamos. Contiene demasiados campos y 22 filas. Sin embargo, nuestro documento XML tiene solamente 3 productos con 3 campos cada uno. El conjunto de resultados producido recibe el nombre de *edge table* (tabla irregular) y contiene metadatos que sirven para la construcción de sistemas de gestión de documentos XML personalizados.

Para obtener las filas y columnas deseadas debemos indicar la estructura de la tabla con la opción `WITH` (véase el listado 16.20). La única diferencia con el listado 16.19 está en la instrucción `SELECT`.

Listado 16.20: Uso de la cláusula `WITH` para indicar las columnas deseadas

```
DECLARE @xml varchar(8000)

-- Este es el documento XML

SET @xml = '<Products>'
SET @xml = @xml + '<Product ProductID="1" '
SET @xml = @xml + 'ProductName="Chai" UnitPrice="18.0000"/>'
SET @xml = @xml + '<Product ProductID="2" '
SET @xml = @xml + 'ProductName="Chang" UnitPrice="19.0000"/>'
SET @xml = @xml + '<Product ProductID="3" '
SET @xml = @xml + 'ProductName="Aniseed Syrup" UnitPrice="10.0000"/>'
SET @xml = @xml + '</Products>'
```

Listado 16.20: continuación

```
-- Creamos una estructura de árbol con el documento XML
-- y obtenemos su controlador

DECLARE @iDoc int

EXEC sp_xml_preparedocument @iDoc OUTPUT, @xml

PRINT CHAR(10)
   + 'Se creó el árbol, handle = '
   + CONVERT(varchar(10), @iDoc)
   + CHAR(10)

SELECT *
FROM OPENXML(@iDoc, 'Products/Product', 1)
WITH (ProductID int,
      ProductName nvarchar(40),
      UnitPrice money)

EXEC sp_xml_removedocument @idoc
```

SALIDA

```
Se creó el árbol, handle = 21

ProductID   ProductName                UnitPrice
----------- -------------------------- --------------------
1.00        Chai                       $18.00
2.00        Chang                      $19.00
3.00        Aniseed Syrup              $10.00

(3 filas afectadas)
```

El resultado del listado 16.20 es más legible que el del listado 16.19 y es similar a cualquier otro conjunto de resultados.

PRECAUCIÓN

Si no ejecuta el procedimiento `sp_xml_removedocument`, la estructura en forma de árbol podría permanecer en memoria hasta que se reinicie el servidor. Esto puede causar ciertos problemas de memoria tratándose de documentos XML de gran tamaño.

La función `OPENXML`

El ejemplo del listado 16.20 es muy sencillo: sólo contiene tres elementos de una sola entidad y tres atributos para cada elemento. La función `OPENXML` se puede usar para leer conjuntos de resultados a partir de documentos XML mucho más complejos.

Considere el documento XML del listado 16.4, que contiene información organizada jerárquicamente proveniente de las tablas `Customers`, `Orders` y

Products. Hay diferentes maneras de leer este documento en Transact-SQL mediante la función OPENXML. El listado 16.21 muestra algunos ejemplos de cómo usar la función OPENXML para seleccionar la información que se leerá de un documento XML. En este caso podemos ejecutar consultas que extraigan información sobre los pedidos, los clientes, los productos o el detalle de los pedidos, todo desde el mismo documento XML.

Listado 16.21: Uso de la función OPENXML con la cláusula WITH para leer un documento XML organizado jerárquicamente

```
-- Este es el documento XML

DECLARE @XML varchar(8000)

SET @XML = '
<Customers>
 <Customer CompanyName="Victuailles en stock">
   <Order Date="1996-07-08">
     <Product Name="Gustafaposs Knackebrod" Price="16.8000" Quantity="6"/>
     <Product Name="Ravioli Angelo" Price="15.6000" Quantity="15"/>
     <Product Name="Louisiana Fiery Hot Pepper Sauce"
       Price="16.8000" Quantity="20"/>
   </Order>
   <Order Date="1996-10-21">
     <Product Name="Filo Mix" Price="5.6000" Quantity="8"/>
     <Product Name="Scottish Longbreads" Price="10.0000" Quantity="10"/>
   </Order>
   <Order Date="1997-02-19">
     <Product Name="Ikura" Price="24.8000" Quantity="20"/>
     <Product Name="Tourtiere" Price="5.9000" Quantity="6"/>
   </Order>
   <Order Date="1997-02-27">
     <Product Name="Uncle Bobaposs Organic Dried Pears" Price="24.0000"
       Quantity="16"/>
     <Product Name="Spegesild" Price="9.6000" Quantity="20"/>
     <Product Name="Mozzarella di Giovanni" Price="27.8000" Quantity="40"/>
   </Order>
   <Order Date="1997-03-18">
     <Product Name="Ikura" Price="24.8000" Quantity="20"/>
   </Order>
   <Order Date="1997-05-23">
     <Product Name="Uncle Bobaposs Organic Dried Pears" Price="30.0000"
       Quantity="10"/>
     <Product Name="Steeleye Stout" Price="18.0000" Quantity="30"/>
     <Product Name="Tarte au sucre" Price="49.3000" Quantity="40"/>
   </Order>
   <Order Date="1997-12-31">
     <Product Name="Chang" Price="19.0000" Quantity="20"/>
     <Product Name="Louisiana Fiery Hot Pepper Sauce" Price="21.0500"
```

Listado 16.21: continuación

```
        Quantity="2"/>
      <Product Name="Longlife Tofu" Price="10.0000" Quantity="15"/>
    </Order>
 </Customer>
 <Customer CompanyName="Vins et alcohols Chevalier">
    <Order Date="1996-07-04">
      <Product Name="Queso Cabrales" Price="14.0000" Quantity="12"/>
      <Product Name="Singaporean Hokkien Fried Mee" Price="9.8000"
        Quantity="10"/>
      <Product Name="Mozzarella di Giovanni" Price="34.8000" Quantity="5"/>
    </Order>
    <Order Date="1996-08-06">
      <Product Name="Flotemysost" Price="17.2000" Quantity="20"/>
      <Product Name="Mozzarella di Giovanni" Price="27.8000" Quantity="7"/>
    </Order>
    <Order Date="1996-09-02">
      <Product Name="Gnocchi di nonna Alice" Price="30.4000" Quantity="4"/>
    </Order>
    <Order Date="1997-11-11">
      <Product Name="Konbu" Price="6.0000" Quantity="4"/>
      <Product Name="Jack&apos;s New England Clam Chowder" Price="9.6500"
        Quantity="12"/>
    </Order>
    <Order Date="1997-11-12">
      <Product Name="Inlagd Sill" Price="19.0000" Quantity="6"/>
      <Product Name="Filo Mix" Price="7.0000" Quantity="18"/>
    </Order>
 </Customer>
</Customers>
'

-- Creamos una estructura en forma de árbol con el documento XML
-- y obtenemos su controlador

DECLARE @iDoc int

EXEC sp_xml_preparedocument @iDoc OUTPUT, @xml

PRINT CHAR(10)
   + 'Se creó el árbol, handle = '
   + CONVERT(varchar(10), @iDoc)
   + CHAR(10)

PRINT CHAR(10)
 + 'Consulta 1' + CHAR(10)
 + 'Información sobre productos'
 + CHAR(10)
```

Listado 16.21: continuación

```
SELECT *
FROM OPENXML(@iDoc, 'Customers/Customer/Order/Product', 1)
WITH (Name nvarchar(40),
      Price money,
      Quantity int)

PRINT CHAR(10)
 + 'Consulta 2' + CHAR(10)
 + 'Información sobre pedidos'
 + CHAR(10)

SELECT *
FROM OPENXML(@iDoc, 'Customers/Customer/Order', 1)
WITH (Date smalldatetime)

PRINT CHAR(10)
 + 'Consulta 3' + CHAR(10)
 + 'Información sobre clientes'
 + CHAR(10)

SELECT *
FROM OPENXML(@iDoc, 'Customers/Customer', 1)
WITH (CompanyName nvarchar(40))

PRINT CHAR(10)
 + 'Consulta 4' + CHAR(10)
 + 'Combinación de atributos de diferentes niveles'
 + CHAR(10)

SELECT CompanyName,
      CONVERT(varchar(10), [Date], 102) As Date,
      Name, Price, Quantity
FROM OPENXML(@iDoc, 'Customers/Customer/Order/Product', 1)
WITH (CompanyName nvarchar(40) '../../@CompanyName',
      Date smalldatetime '../@Date',
      Name nvarchar(40),
      Price money,
      Quantity int)

PRINT CHAR(10)
 + 'Consulta 5' + CHAR(10)
 + 'Ejecución de una consulta XPath' + CHAR(10)
 + 'que extrae en orden los productos cuya cantidad > 20'
 + CHAR(10)

SELECT CompanyName,
      CONVERT(varchar(10), [Date], 102) As Date,
```

Listado 16.21: continuación

```
     Name, Price, Quantity
FROM OPENXML(@iDoc, 'Customers/Customer/Order/Product[@Quantity>20]', 1)
WITH (CompanyName nvarchar(40) '../../@CompanyName',
     Date smalldatetime '../@Date',
     Name nvarchar(40),
     Price money,
     Quantity int)

EXEC sp_xml_removedocument @idoc
```

SALIDA

```
Se creó el árbol, handle = 25

Consulta 1
Información sobre productos

Name                                     Price                Quantity
---------------------------------------- -------------------- --------------
Gustafaposs Knackebrod                   $16.80               6.00
Ravioli Angelo                           $15.60               15.00
Louisiana Fiery Hot Pepper Sauce         $16.80               20.00
Filo Mix                                 $5.60                8.00
Scottish Longbreads                      $10.00               10.00
Ikura                                    $24.80               20.00
Tourtiere                                $5.90                6.00
Uncle Bobaposs Organic Dried Pears       $24.00               16.00
Spegesild                                $9.60                20.00
Mozzarella di Giovanni                   $27.80               40.00
Ikura                                    $24.80               20.00
Uncle Bobaposs Organic Dried Pears       $30.00               10.00
Steeleye Stout                           $18.00               30.00
Tarte au sucre                           $49.30               40.00
Chang                                    $19.00               20.00
Louisiana Fiery Hot Pepper Sauce         $21.05               2.00
Longlife Tofu                            $10.00               15.00
Queso Cabrales                           $14.00               12.00
Singaporean Hokkien Fried Mee            $9.80                10.00
Mozzarella di Giovanni                   $34.80               5.00
Flotemysost                              $17.20               20.00
Mozzarella di Giovanni                   $27.80               7.00
Gnocchi di nonna Alice                   $30.40               4.00
Konbu                                    $6.00                4.00
Jack's New England Clam Chowder          $9.65                12.00
Inlagd Sill                              $19.00               6.00
Filo Mix                                 $7.00                18.00

(27 filas afectadas)
```

Listado 16.21: continuación

```
Consulta 2
Información sobre pedidos

Date
-----------------------------------------------------
7/8/1996 12:00:00 AM
10/21/1996 12:00:00 AM
2/19/1997 12:00:00 AM
2/27/1997 12:00:00 AM
3/18/1997 12:00:00 AM
5/23/1997 12:00:00 AM
12/31/1997 12:00:00 AM
7/4/1996 12:00:00 AM
8/6/1996 12:00:00 AM
9/2/1996 12:00:00 AM
11/11/1997 12:00:00 AM
11/12/1997 12:00:00 AM

(12 filas afectadas)

Consulta 3
Información sobre clientes

CompanyName
---------------------------------------
Victuailles en stock
Vins et alcohols Chevalier

(2 filas afectadas)


Consulta 4
Combinación de atributos de diferentes niveles

CompanyName            Date         Name                              Price   Quantity
---------------------- ----------   -------------------------------- ------  ----------
Victuailles en stock   1996.07.08   Gustafaposs Knackebrod           $16.80  6.00
Victuailles en stock   1996.07.08   Ravioli Angelo                   $15.60  15.00
Victuailles en stock   1996.07.08   Louisiana Fiery Hot Pepper       $16.80  20.00
Victuailles en stock   1996.10.21   Filo Mix                         $5.60   8.00
Victuailles en stock   1996.10.21   Scottish Longbreads              $10.00  10.00
Victuailles en stock   1997.02.19   Ikura                            $24.80  20.00
Victuailles en stock   1997.02.19   Tourtiere                        $5.90   6.00
Victuailles en stock   1997.02.27   Uncle Bobaposs Organic Dried     $24.00  16.00
Victuailles en stock   1997.02.27   Spegesild                        $9.60   20.00
Victuailles en stock   1997.02.27   Mozzarella di Giovanni           $27.80  40.00
```

Listado 16.21: continuación

```
Victuailles en stock   1997.03.18   Ikura                          $24.80  20.00
Victuailles en stock   1997.05.23   Uncle Bobaposs Organic Dried   $30.00  10.00
Victuailles en stock   1997.05.23   Steeleye Stout                 $18.00  30.00
Victuailles en stock   1997.05.23   Tarte au sucre                 $49.30  40.00
Victuailles en stock   1997.12.31   Chang                          $19.00  20.00
Victuailles en stock   1997.12.31   Louisiana Fiery Hot Pepper     $21.05  2.00
Victuailles en stock   1997.12.31   Longlife Tofu                  $10.00  15.00
Vins et alcohols       1996.07.04   Queso Cabrales                 $14.00  12.00
Vins et alcohols       1996.07.04   Singaporean Hokkien Fried      $9.80   10.00
Vins et alcohols       1996.07.04   Mozzarella di Giovanni         $34.80  5.00
Vins et alcohols       1996.08.06   Flotemysost                    $17.20  20.00
Vins et alcohols       1996.08.06   Mozzarella di Giovanni         $27.80  7.00
Vins et alcohols       1996.09.02   Gnocchi di nonna Alice         $30.40  4.00
Vins et alcohols       1997.11.11   Konbu                          $6.00   4.00
Vins et alcohols       1997.11.11   Jack's New England Clam        $9.65   12.00
Vins et alcohols       1997.11.12   Inlagd Sill                    $19.00  6.00
Vins et alcohols       1997.11.12   Filo Mix                       $7.00   18.00

(27 filas afectadas)

Consulta 5
Ejecución de una consulta XPath
que extrae en orden los productos cuya cantidad > 20

CompanyName            Date         Name                           Price   Quantity
---------------------- ----------   ------------------------------ ------  ----------
Victuailles en stock   1997.02.27   Mozzarella di Giovanni         $27.80  40.00
Victuailles en stock   1997.05.23   Steeleye Stout                 $18.00  30.00
Victuailles en stock   1997.05.23   Tarte au sucre                 $49.30  40.00

(3 filas afectadas)
```

Examinemos más de cerca los ejemplos del listado 16.21.

El primer ejemplo usa esta consulta:

```
SELECT *
FROM OPENXML(@iDoc, 'Customers/Customer/Order/Product', 1)
WITH (Name nvarchar(40),
      Price money,
      Quantity int)
```

Con esta consulta, sólo podemos obtener datos en el nivel de los productos, ya que en la función `OPENXML` hemos definido la ruta a los datos como `'Customers/Customer/Order/Product'`. La cláusula `WITH` contiene los campos

que se leerán (de acuerdo al *mapping* –o modo de traducción de los valores a XML– predeterminado).

Cuando ejecutamos esta consulta obtenemos 27 filas, una por cada producto contenido en el documento XML.

El segundo ejemplo sólo lee la fecha del pedido, que corresponde al nivel de los pedidos (seleccionado por su ruta, `'Customers/Customer/Order'`):

```
SELECT *
FROM OPENXML(@iDoc, 'Customers/Customer/Order', 1)
WITH (Date smalldatetime)
```

El tercer ejemplo sólo lee nombres de compañías, del nivel de los clientes (seleccionado mediante su ruta, `'Customers/Customer'`).

```
SELECT *
FROM OPENXML(@iDoc, 'Customers/Customer', 1)
WITH (CompanyName nvarchar(40))
```

Hasta ahora fue bastante sencillo. Seleccionamos la ruta para llegar a los datos que queremos leer (en el segundo parámetro de la función `OPENXML`) e indicamos las definiciones de las columnas en la cláusula `WITH`.

El cuarto ejemplo es un poco más complejo, porque queremos leer algo de información de cada uno de los niveles combinando la salida de los tres ejemplos previos:

```
SELECT CompanyName,
     CONVERT(varchar(10), [Date], 102) As Date,
     Name, Price, Quantity
FROM OPENXML(@iDoc, 'Customers/Customer/Order/Product', 1)
WITH (CompanyName nvarchar(40) '../../@CompanyName',
     Date smalldatetime '../@Date',
     Name nvarchar(40),
     Price money,
     Quantity int)
```

El nivel básico sigue siendo `Product`, por eso la ruta es `'Customers/Customer/Order/Product'`. Pero la cláusula `WITH` contiene el campo `Date` del nivel `Order` y el campo `CompanyName` del nivel `Customer`.

Para leer el campo `Date`, que no pertenece al nivel `Product`, tenemos que indicar su ruta `'../@Date'`, que quiere decir "subir un nivel y leer el atributo `Date`".

Para el campo `CompanyName`, la ruta es `'../../@CompanyName'`, que quiere decir "subir dos niveles y leer el atributo `CompanyName`".

El último ejemplo ejecuta una consulta XPath que restringe el conjunto de resultados. En este caso, sólo nos interesan los pedidos que contienen más de 20 unidades de cualquier producto dado. Podemos indicar esta condición en la forma de una ruta `'Customers/Customer/Order/Product[@Quantity>20]'`:

```
SELECT CompanyName,
      CONVERT(varchar(10), [Date], 102) As Date,
      Name, Price, Quantity
FROM OPENXML(@iDoc, 'Customers/Customer/Order/Product[@Quantity>20]', 1)
WITH (CompanyName nvarchar(40) '../../@CompanyName',
      Date smalldatetime '../@Date',
      Name nvarchar(40),
      Price money,
      Quantity int)
```

Microsoft SQL Server 2000 implementa un subconjunto de la especificación de XPath que hace el World Wide Web Consortium (W3C, un consorcio que se dedica a la definición y difusión de estándares para la Web). Puede obtener más información acerca de la especificación XPath en `http://www.w3.org/TR/xpath`.

Los libros en pantalla de SQL Server contienen una descripción completa del subconjunto implementado en esta versión de SQL Server. Puede buscar la sección "Uso de consultas XPath" en los libros en pantalla.

SUGERENCIA

En el ejemplo del listado 16.21 creamos el documento XML en un campo `varchar`, que como tal se limita a 8.000 bytes.

No se pueden crear variables con el tipo de datos `ntext`, pero un procedimiento almacenado puede tener un parámetro con ese tipo de datos, lo que aumenta a casi 2 Gigabytes el tamaño de los documentos XML que se pueden procesar.

Es importante que los parámetros que pasen información en XML estén definidos como `ntext` y no `text`, para poder manejar documentos XML en Unicode.

Quizá se pregunte qué es el tercer parámetro de la función `OPENXML`. Este parámetro selecciona el tipo de valores usados en el documento XML:

0 Por defecto, el *mapping* (la traducción de los valores a XML) se basa en el uso de atributos.

1 Usar *mapping* basado en atributos.

2 Usar *mapping* basado en elementos.

8 Puede combinar *mapping* basado en elementos y en atributos.

Como los ejemplos que usamos en el listado 16.21 sólo contienen atributos, el valor del tercer parámetro siempre es 1.

Combinación de datos XML con datos de SQL Server

Como la función OPENXML devuelve un conjunto de resultados se lo puede combinar con otras tablas o funciones de conjunto de filas, tales como OPENROWSET, OPENDATASOURCE u OPENQUERY.

El listado 16.22 muestra un ejemplo en el que combinamos el conjunto de resultados de OPENXML con las tablas Orders y Order Details.

Listado 16.22: El conjunto de resultados de OPENXML se puede combinar con otras tablas u otros conjuntos de resultados.

```
DECLARE @xml varchar(8000)

-- Este es el documento XML

SET @xml = '<Products>'
SET @xml = @xml + '<Product ProductID="1" '
SET @xml = @xml + 'ProductName="Chai" UnitPrice="18.0000"/>'
SET @xml = @xml + '<Product ProductID="2" '
SET @xml = @xml + 'ProductName="Chang" UnitPrice="19.0000"/>'
SET @xml = @xml + '<Product ProductID="3" '
SET @xml = @xml + 'ProductName="Aniseed Syrup" UnitPrice="10.0000"/>'
SET @xml = @xml + '</Products>'

-- Creamos una estructura en forma de árbol con el documento XML
-- y obtenemos su controlador

DECLARE @iDoc int

EXEC sp_xml_preparedocument @iDoc OUTPUT, @xml

PRINT CHAR(10)
   + ' Se creó el árbol, handle = '
   + CONVERT(varchar(10), @iDoc)
   + CHAR(10)

-- Combinamos el conjunto de resultados de OPENXML
-- con las tablas Orders y [Order Details]

SELECT CONVERT(varchar(10), OrderDate, 102) AS Date,
ProductName, P.UnitPrice, OD.Quantity
FROM OPENXML(@iDoc, 'Products/Product', 1)
WITH (ProductID int,
      ProductName nvarchar(40),
      UnitPrice money) AS P
```

Listado 16.22: continuación

```
JOIN [Order Details] OD
     ON OD.ProductID = P.ProductID
JOIN Orders O
     ON O.OrderID = OD.OrderID
WHERE Quantity > 20
 AND Year(OrderDate) = 1996

EXEC sp_xml_removedocument @idoc
```

SALIDA

```
Se creó el árbol, handle = 11

Date        ProductName                UnitPrice              Quantity
----------  -------------------------- ---------------------- --------
1996.08.20  Chai                       $18.00                 45.00
1996.10.30  Chang                      $19.00                 24.00
1996.12.25  Chang                      $19.00                 25.00
1996.09.05  Chang                      $19.00                 40.00
1996.10.11  Chang                      $19.00                 25.00
1996.07.17  Chang                      $19.00                 50.00
1996.07.24  Chang                      $19.00                 35.00
1996.08.26  Aniseed Syrup              $10.00                 30.00

(8 filas afectadas)
```

Acceso a SQL Server 2000 a través de HTTP

Con SQL Server 2000 se puede usar el protocolo HTTP para acceder a los datos desde cualquier lugar de Internet. Esto es especialmente útil cuando hay que cruzar cortafuegos (*firewalls*), ya que el protocolo HTTP usa el puerto 80, que no presenta ningún problema cuando se trabaja a través de *firewalls*.

SQL Server puede producir páginas web estáticas con el asistente para la Web. Este asistente se puede usar para actualizar la página HTML cada vez que cambien los datos subyacentes. Sin embargo, esta forma de publicar en Internet datos de SQL Server no es muy flexible, ya que se limita a informes específicos.

Con el filtro de la interfaz de programación de aplicaciones de IIS para SQL (SQLISAPI) se puede usar Internet Information Server para enviarle consultas a SQL Server y devolverle los resultados al cliente a través de HTTP.

El proceso, simplificado, funciona así:

1. El cliente se conecta a un directorio virtual especial en IIS, definido para el acceso a SQL Server, e indica en la petición HTTP la consulta que quiere ejecutar.

2. El filtro SQLISAPI envía la petición a SQL Server a través de OLE DB, mediante la cláusula `FOR XML`.
3. SQL Server devuelve los resultados a IIS en formato XML.
4. IIS devuelve los resultados a la aplicación cliente a través de HTTP.

Hay diferentes formas de enviar peticiones a SQL Server a través de HTTP:

- Consultas indicadas en el URL: la consulta va insertada en la invocación misma del URL. Esta opción sólo se debe usar en situaciones de prueba.
- Archivos de plantilla XML : la consulta se define en archivos preparados de antemano, lo que brinda un acceso más seguro que indicar la consulta en el URL.
- Consultas XPath sobre esquemas: para validar consultas que contienen instrucciones del lenguaje XPath.
- El método POST: se puede usar desde cualquier página web.
- *Updategrams* XML: permiten insertar, modificar o eliminar datos en SQL Server.

PRECAUCIÓN

El envío de consultas o plantillas directamente a través de HTTP representa un problema de seguridad, ya que mediante este método los usuarios pueden enviar cualquier consulta Transact-SQL válida (siempre que puedan conectarse al directorio virtual).

En un entorno de Internet de "producción" (destinado al uso real), se recomienda deshabilitar las consultas POST y restringir el acceso HTTP exclusivamente al uso de archivos de plantilla, ya que de esta forma es posible limitar las consultas que los usuarios pueden enviar a través de HTTP.

NOTA

Para poder usar updategrams XML en SQL Server 2000, hay que instalar el soporte de XML para SQL Server (XML for SQL Server Web Release), que se puede descargar de:

```
http://msdn.microsoft.com/downloads/default.asp?URL=/code/sample.asp?url=/
msdn-files/027/001/554/msdncompositedoc.xml
```

Como alternativa, puede descargar la nueva versión SQ:XML 2.0 de la siguiente dirección:

```
http://download.microsoft.com/download/SQLSSVR2000/Install/2.0/W98NT42KMeXP/
EN-US/sqlxml.msi
```

Pero asegúrese de que lee los ficheros que acompañan a esta nueva versión para asegurarse de que mantiene los criterios de compatibilidad con versiones anteriores de esta nueva versión.

En esta sección aprenderemos a crear un directorio virtual en IIS para SQL Server y a enviar consultas a través de HTTP, mediante consultas en URL y plantillas.

Si desea más información acerca de este interesante tema, puede leer la sección "Acceso a SQL Server usando HTTP" en los libros en pantalla y la documentación de XML para SQL que se instala con el soporte de XML para SQL Server 2000 (*XML for SQL Server Web Release*).

Creación de un directorio virtual para SQL Server en IIS

El primer paso para proveer acceso a SQL Server a través de HTTP es configurar un directorio virtual en IIS para SQL Server.

La instalación de SQL Server 2000 deja en el menú de SOL Server un comando llamado Configurar la compatibilidad con SQL XML en IIS. Al presionar en este menú, se inicia la consola administrativa de Microsoft (MMC, *Microsoft Management Console*), con el *applet* de administración de directorios virtuales para SQL Server en IIS.

Seleccione el servidor, expanda su árbol y presione el cursor derecho en el nodo Sitio web predeterminado, para abrir el menú contextual. En el menú contextual, elija Nuevo, Directorio virtual para inicial la utilidad de administración de directorios virtuales de IIS. Escriba el nombre del directorio virtual y la ruta física, que se encuentran en la ficha General, y verá una imagen similar a la figura 16.9. Para seguir los ejemplos de esta sección, el nombre del directorio virtual es SQLXML.

Figura 16.9: *Las propiedades del nuevo directorio virtual.*

Abra la ficha Seguridad para indicar el modo de autentificación de los usuarios. Existen las siguientes opciones:

- Conectarse siempre como, para indicar que todos los accesos a SQL Server a través de HTTP se realicen a través de una sola cuenta. Cuando use esta opción, asegúrese de no usar una cuenta que tenga privilegios administrativos en SQL Server.
- Usar autentificación integrada de Windows, que usa el método de autentificación desafío/respuesta de Microsoft Windows NT (Windows NT 4.0) o autentificación integrada de Windows (Microsoft Windows 2000).
- Usar autentificación básica, para que se solicite un nombre de usuario de SQL Server 2000 y una contraseña.

En este caso, seleccione la opción Usar autentificación integrada de Windows. Se encontrará con la situación presentada en la figura 16.10.

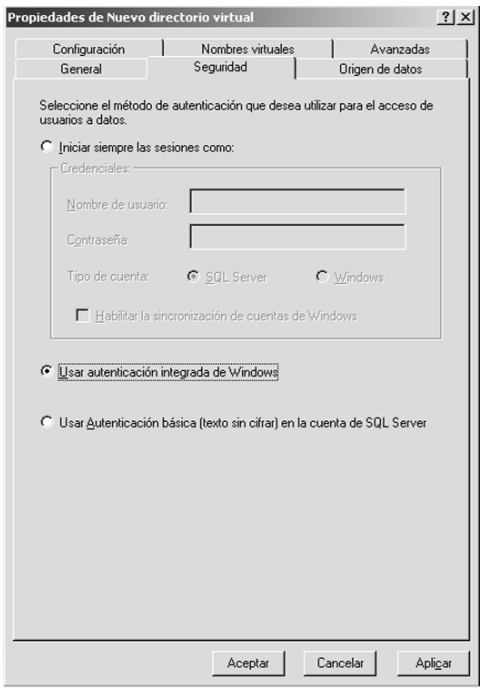

Figura 16.10: *Indicación del modo de autentificación usado para acceder a datos de SQL Server a través de HTTP.*

NOTA

Si quiere proveer diferentes modos de seguridad para el acceso puede crear diferentes directorios virtuales para el mismo servidor. Desde IIS podrá administrar la seguridad de acceso de cada uno de estos directorios virtuales.

Abra la ficha Origen de datos para seleccionar el servidor deseado y la base de datos predeterminada (véase la figura 16.11).

Figura 16.11: *Indicación de la instancia de SQL Server y la base de datos predeterminada.*

NOTA

Para acceder a múltiples servidores conviene crear un directorio virtual para cada uno.

Abra la ficha Configuración para seleccionar el método de acceso. En este caso, seleccione Permitir consultas en URL, Permitir XPath y Permitir consultas con plantilla (véase la figura 16.12).

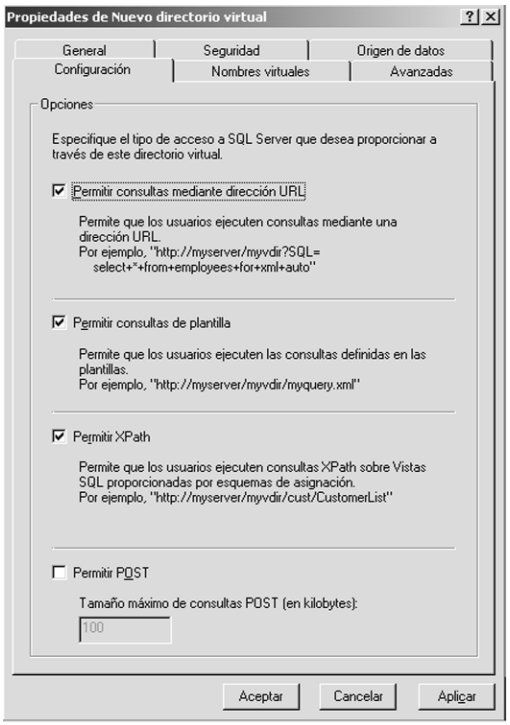

Figura 16.12: *Configuración del método de acceso a los datos de SQL Server.*

Para probar el directorio virtual que acaba de configurar abra Internet Explorer y escriba el siguiente URL en la línea de dirección:

```
http://localhost/sqlxml?sql=SELECT+CategoryName+FROM+Categories+As+Category+FOR+XML+AUTO&root=Categories
```

Verá lo que aparece en la figura 16.13.

NOTA

En consultas indicadas en URL escriba signos + en vez de los espacios.

Uso de datos de SQL Server a través de HTTP

En la sección precedente aprendimos a configurar un directorio virtual para SQL Server en Internet Information Server. Para probar el directorio virtual enviamos una consulta sencilla.

Este directorio virtual se puede usar para enviar todo tipo de consultas a SQL Server. Sin embargo, como podrá imaginar, este método puede ser tedioso, aumenta la probabilidad de cometer errores y puede implicar un riesgo de seguridad, ya que en el URL los usuarios pueden enviar cualquier consulta.

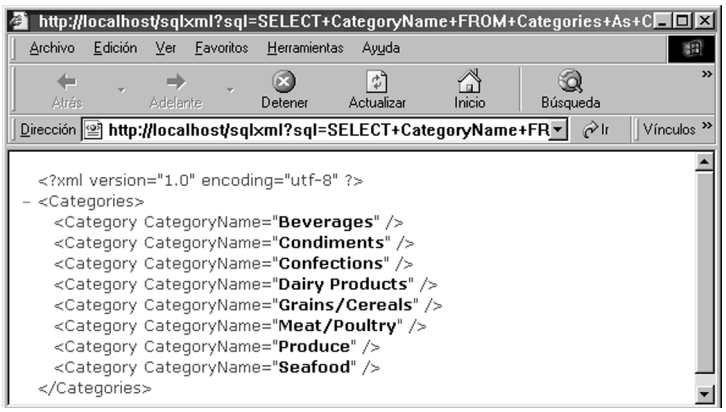

Figura 16.13: *Resultados de una consulta indicada en URL ejecutada a través de un directorio virtual para SQL Server en IIS.*

Como alternativa se pueden crear archivos de plantilla XML con consultas predefinidas. Una plantilla XML es un documento XML que contiene una o más instrucciones SQL y consultas XPath.

El lector puede crear ahora mismo su primera plantilla XML. Cree una carpeta llamada `Plantillas` en cualquier lugar del disco rígido del servidor. Abre el bloc de notas, crea un nuevo archivo con el nombre `Categories.xml` y lo guarda en la ruta física del directorio virtual que creó en la sección precedente. En este archivo escribe las instrucciones que se incluyen en el listado 16.23.

Listado 16.23: Una plantilla XML para mostrar categorías

```
<?xml version="1.0"?>
<Categories xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
 <sql:query>
   SELECT CategoryName
   FROM Categories AS Category
   FOR XML AUTO
 </sql:query>
</Categories>
```

Ahora inicia la ventana de administración de directorios virtuales en IIS y abre el directorio virtual que creó en la sección precedente. Va a la ficha `Nombres virtuales` y crea un nuevo nombre virtual, llamado `Plantillas`, que apunte al directorio `Plantillas` que acaba de crear; indica como tipo del directorio virtual la opción Plantillas. La figura 16.14 muestra la ficha Nombres virtuales luego de crear este nombre virtual.

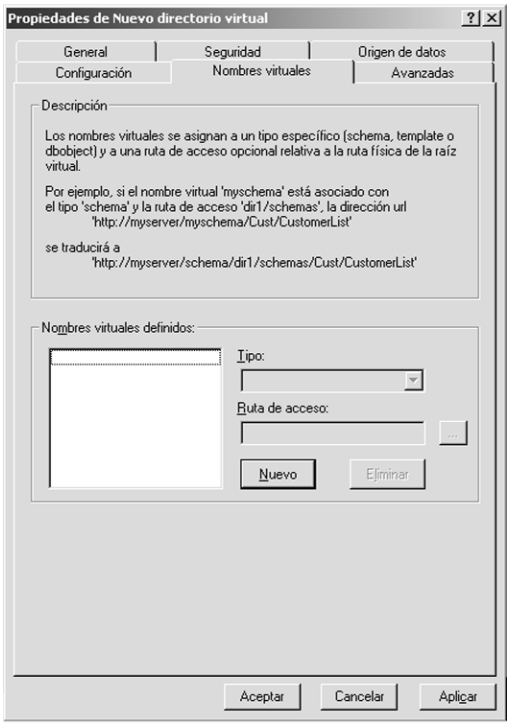

Figura 16.14: *Este directorio virtual contiene un nombre virtual para una plantilla.*

NOTA

No hace falta que la carpeta para las plantillas coincida con el directorio virtual. Sin embargo, si los archivos de plantilla no están guardados en una carpeta con nombre virtual de tipo "Plantillas", IIS no los reconocerá como plantillas XML válidas.

Ahora puede abrir Internet Explorer y entrar al siguiente URL:

```
http://localhost/sqlxml/plantillas/Categories.xml
```

Verá la familiar lista de categorías en formato XML.

De esta forma podemos tener consultas predefinidas guardadas como plantillas y los usuarios no tienen por qué saber cómo están armadas esas consultas.

Para hacer algo más útil puede intentar usar parámetros para seleccionar información específica acerca de los clientes, sus pedidos y los productos incluidos en esos pedidos. Para ello puede escribir las instrucciones que se incluyen en el listado 16.24 y guardarlas en el archivo `CustomersOrders.xml` en el directorio `Plantillas` que creó anteriormente.

Listado 16.24: Una plantilla XML que muestra clientes y sus pedidos

```
<?xml version="1.0"?>
<Customers xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
 <sql:header>
   <sql:param name="Customer">NULL</sql:param>
   <sql:param name="Product">NULL</sql:param>
 </sql:header>
 <sql:query>
   SELECT CompanyName,
     OrderDate,
     ProductName,
     Quantity,
     Item.
     UnitPrice
   FROM Customers AS Customer
     JOIN Orders AS [Order]
       ON [Order].CustomerID = Customer.CustomerID
     JOIN [Order Details] AS Item
       ON Item.OrderID = [Order].OrderID
     JOIN Products AS Product
       ON Product.ProductID = Item.ProductID
   WHERE CompanyName LIKE
     CASE @Customer
       WHEN 'NULL' THEN '%'
       ELSE @Customer + '%'
       END
   AND ProductName LIKE
     CASE @Product
       WHEN 'NULL' THEN '%'
       ELSE @Product + '%'
       END
   ORDER BY CompanyName, OrderDate, ProductName
   FOR XML AUTO
 </sql:query>
</Customers>
```

Si examina el código del listado 16.24 puede identificar la presencia de una nueva sección al principio del archivo:

```
<sql:header>
   <sql:param name="Customer">NULL</sql:param>
   <sql:param name="Product">NULL</sql:param>
</sql:header>
```

Esta sección define los parámetros usados, que funcionan de manera similar a los de los procedimientos almacenados o los de las funciones definidas por el usuario. En este caso hemos declarado dos parámetros: `Customer` y `Product`. Para ambos

parámetros declaramos NULL como valor predeterminado. Observe que en este caso no se trata del valor NULL de SQL, sino de la cadena 'NULL', que SQL Server no trata de la misma manera.

En consecuencia la cláusula WHERE aparece diferente:

```
WHERE CompanyName LIKE
      CASE @Customer
        WHEN 'NULL' THEN '%'
        ELSE @Customer + '%'
        END
    AND ProductName LIKE
      CASE @Product
        WHEN 'NULL' THEN '%'
        ELSE @Product + '%'
        END
```

Para probar esta plantilla puede escribir este URL en Internet Explorer:

```
http://localhost/sqlxml/Plantillas/CustomersOrders.xml
```

Como en este ejemplo no usamos los parámetros, SQL Server devuelve un documento XML de gran tamaño que incluye todos los pedidos hechos por todos los clientes.

Para reducir el alcance de la búsqueda escriba los siguientes URL y vea los resultados en Internet Explorer:

```
http://localhost/sqlxml/plantillas/CustomersOrders.xml?Customer=Alfreds
http://localhost/sqlxml/plantillas/CustomersOrders.xml?Customer=Alf&Product=Ra
http://localhost/sqlxml/plantillas/CustomersOrders.xml?Product=Ravioli
```

Sin embargo, la forma en que el listado 16.24 envía las consultas no es eficiente. El listado 16.25 muestra una secuencia de instrucciones Transact-SQL que es más larga pero más eficiente, ya que SQL Server puede optimizar el uso de los índices según los parámetros enviados. La única diferencia entre estos dos ejemplos es que el listado 16.25 divide la consulta en cuatro consultas individuales:

- Cuando el usuario no indica ninguno de los parámetros (Product y Customer son ambos NULL) la consulta lee información acerca de todos los pedidos.
- Cuando Customer es NULL se lee sólo información acerca de los pedidos relacionados con los productos cuyo nombre comienza con el valor indicado en el parámetro Product.

- Cuando `Product` es `NULL` se lee sólo información acerca de los clientes cuyo nombre comienza con el valor indicado en el parámetro `Customer`.
- En el caso restante se lee sólo información acerca de los productos cuyo nombre comienza con el valor indicado en el parámetro `Product` y clientes cuyo nombre comienza con el valor indicado en el parámetro `Customer`.

Listado 16.25: Una plantilla XML más eficiente para mostrar clientes y sus pedidos

```
<?xml version="1.0"?>
<Customers xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
 <sql:header>
   <sql:param name="Customer">NULL</sql:param>
   <sql:param name="Product">NULL</sql:param>
 </sql:header>
 <sql:query>
   IF @Customer = 'NULL'
     IF @Product = 'NULL'
       SELECT CompanyName,
         OrderDate,
         ProductName,
         Quantity,
         Item.
         UnitPrice
       FROM Customers AS Customer
         JOIN Orders AS [Order]
           ON [Order].CustomerID = Customer.CustomerID
         JOIN [Order Details] AS Item
           ON Item.OrderID = [Order].OrderID
         JOIN Products AS Product
           ON Product.ProductID = Item.ProductID
       ORDER BY CompanyName, OrderDate, ProductName
       FOR XML AUTO
     ELSE
       SELECT CompanyName,
         OrderDate,
         ProductName,
         Quantity,
         Item.
         UnitPrice
       FROM Customers AS Customer
         JOIN Orders AS [Order]
           ON [Order].CustomerID = Customer.CustomerID
         JOIN [Order Details] AS Item
           ON Item.OrderID = [Order].OrderID
         JOIN Products AS Product
           ON Product.ProductID = Item.ProductID
       WHERE ProductName LIKE @Product + '%'
```

Listado 16.25: continuación

```
        ORDER BY CompanyName, OrderDate, ProductName
        FOR XML AUTO
    ELSE
      IF @Product = 'NULL'
        SELECT CompanyName,
          OrderDate,
          ProductName,
          Quantity,
          Item.
          UnitPrice
        FROM Customers AS Customer
          JOIN Orders AS [Order]
            ON [Order].CustomerID = Customer.CustomerID
          JOIN [Order Details] AS Item
            ON Item.OrderID = [Order].OrderID
          JOIN Products AS Product
            ON Product.ProductID = Item.ProductID
        WHERE CompanyName LIKE @Customer + '%'
        ORDER BY CompanyName, OrderDate, ProductName
        FOR XML AUTO
      ELSE
        SELECT CompanyName,
          OrderDate,
          ProductName,
          Quantity,
          Item.
          UnitPrice
        FROM Customers AS Customer
          JOIN Orders AS [Order]
            ON [Order].CustomerID = Customer.CustomerID
          JOIN [Order Details] AS Item
            ON Item.OrderID = [Order].OrderID
          JOIN Products AS Product
            ON Product.ProductID = Item.ProductID
        WHERE ProductName LIKE @Product + '%'
        AND CompanyName LIKE @Customer + '%'
        ORDER BY CompanyName, OrderDate, ProductName
        FOR XML AUTO
 </sql:query>
</Customers>
```

Como puede ver en el listado 16.25, en una plantilla no se está limitado a escribir una sola consulta, sino que puede escribirse un lote completo que incluya varias instrucciones, siempre que en todas las instrucciones `SELECT` se use la cláusula `FOR XML`.

Uso de updategrams para modificar datos en SQL Server 2000

En las secciones precedentes aprendimos a leer datos de SQL Server en formato XML y a usar datos XML dentro de un proceso en Transact-SQL.

La extensión de SQL Server para XML, que como ya hemos explicado se puede bajar de Internet, brinda las siguientes posibilidades adicionales:

- Insertar, eliminar y modificar filas en SQL Server mediante *updategrams* XML.
- Importar datos XML a SQL Server con el componente para carga masiva de XML.

Un *updategram* es una plantilla XML que contiene una imagen de los datos previa y otra imagen posterior.

Como los documentos XML tienen una estructura jerárquica, los cambios a los datos pueden afectar múltiples tablas. Los *updategrams* tienen capacidades transaccionales que permiten considerar un grupo específico de cambios como miembros de la misma transacción, a fin de mantener la consistencia de los datos a lo largo de esas modificaciones. Cada transacción se identifica por medio de un elemento `sync`. El listado 16.26 contiene un sencillo *updategram* que cambia el nombre de contacto de un cliente.

Listado 16.26: Un sencillo *updategram* que cambia el nombre de contacto de un cliente

```
<CustomerUpdate xmlns:updg="urn:schemas-microsoft-com:xml-updategram">
 <updg:sync>
   <updg:before>
     <Customers CustomerID="ALFKI">
       <ContactName>Maria Anders</ContactName>
     </Customers>
   </updg:before>
   <updg:after>
     <Customers CustomerID="ALFKI">
       <ContactName>Stephen Johns</ContactName>
     </Customers>
   </updg:after>
 </updg:sync>
</CustomerUpdate>
```

Use el bloc de notas para crear el ejemplo del listado 16.26 y guárdelo con el nombre `CustomerUpdate.xml` en el directorio `Plantillas` que creó con anterioridad (en la sección "Creación de un directorio virtual para SQL Server en IIS" de este capítulo).

Para ejecutar el *updategram*, abra Internet Explorer y escriba el siguiente URL:

```
http://localhost/sqlxml/plantillas/CustomerUpdate.xml
```

SUGERENCIA

Si recibe un mensaje de error, controle cómo escribió el archivo XML. Recuerde que las etiquetas XML distinguen entre mayúsculas y minúsculas.

Puede abrir una nueva conexión a SQL Server desde el analizador de consultas y verificar que el cliente `'ALFKI'` recibió los cambios definidos en el *updategram*.

Si examina el ejemplo del listado 16.26 puede identificar las siguientes secciones:

```
<CustomerUpdate xmlns:updg="urn:schemas-microsoft-com:xml-updategram">
```

Esta línea, al principio del archivo, define el archivo XML como una plantilla XML que contiene un *updategram*.

```
<updg:sync>
  ...
</updg:sync>
```

Estas dos etiquetas definen los límites de una transacción individual.

```
<updg:before>
   ...
</updg:before>
```

Estas dos etiquetas definen la imagen previa de los datos. En cierto modo, podemos considerar esta sección como la cláusula `WHERE` de una instrucción `UPDATE`, en la que definimos qué filas sufrirán las modificaciones.

```
<updg:after>
   ...
</updg:after>
```

Por último, estas dos etiquetas delimitan los valores a modificar. Esta sección produce el mismo efecto que la cláusula `SET` de la instrucción `UPDATE`.

En este ejemplo usamos la codificación (*mapping*) predeterminada, en la que el código XML muestra los nombres reales de las tablas y las columnas (en este caso, la tabla `Customers` y las columnas `CustomerID` y `ContactName`). En ocasiones puede ser preferible suministrar un archivo de esquema, por ejemplo `CustomerSchema.xml`, en cuyo caso hay que cambiar la línea `<updg:sync>` del archivo del *updategram* y proveer el atributo `mapping-schema`, como en la línea siguiente:

```
<updg:sync mapping-schema="CustomerSchema.xml">
```

NOTA

Si el esquema no estuviera guardado en la misma ruta que el updategram, en el atributo `mapping-schema` se puede indicar la ruta completa.

Usando ADO 2.6 se puede invocar un *updategram* desde una página ASP, como en el ejemplo del listado 16.27.

EJEMPLO

Listado 16.27: Uso de ADO 2.6 en VBScript para ejecutar un *updategram*

```
<%@ LANGUAGE = VBScript %>
<% Option Explicit %>

<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" content="text/html" charset="UTF-8"/>
<TITLE>SQL-XML by Example - SQLXML.asp</TITLE>

<%
 ' ESTE ES EL SCRIPT DE SERVIDOR
 ' Esta parte no será visible en el cliente

 ' Algunas constantes para que el código sea más legible.
 ' Algunas de estas constantes se pueden encontrar en el
 ' archivo adovbs.inc

 const adUseClient = 3
 const adWriteChar = 0
 const adWriteLine = 1
 const adExecuteStream = 1024

 ' Creación de un objeto ADO Connection

 Dim adoConn
 Set adoConn = Server.CreateObject("ADODB.Connection")

 ' Definición de la cadena de conexión para conectarse a una instancia
 ' válida de SQL Server 2000 y a la base de datos Northwind.
 ' Se indica procesamiento en el cliente, que en este caso será
 ' el servidor IIS

 ' Nota: si la instancia predeterminada es SQL Server 7, este código no
 ' funcionará, a menos que tenga instalado el soporte XML para SQL Server 7.0

 Dim sConn
 sConn = "Provider=sqloledb;"
 sConn = sConn & "Data Source=MSSQLFGG\S2K;"
 sConn = sConn & "Initial Catalog=Northwind;"
```

Listado 16.27: continuación

```
sConn = sConn & "User ID=sa"
adoConn.ConnectionString = sConn
adoConn.CursorLocation = adUseClient

' Abrimos la conexión

adoConn.Open

' Creamos un objeto ADO Command,
' para enviar la consulta XML y recibir los resultados en XML

Dim adoCmd
Set adoCmd = Server.CreateObject("ADODB.Command")
Set adoCmd.ActiveConnection = adoConn

' Definimos un updategram XML como el del listado 16.26
' Observe que tenemos que cambiar " por ' porque VBScript
' usa "" para delimitar las constantes

Dim sQuery
sQuery = "<CustomerUpdate "
sQuery = sQuery & "xmlns:updg='urn:schemas-microsoft-com:xml-updategram'>"
sQuery = sQuery & "<updg:sync>"
sQuery = sQuery & "<updg:before>"
sQuery = sQuery & "<Customers CustomerID='ALFKI'>"
sQuery = sQuery & "<ContactName>Maria Anders</ContactName>"
sQuery = sQuery & "</Customers>"
sQuery = sQuery & "</updg:before>"
sQuery = sQuery & "<updg:after>"
sQuery = sQuery & "<Customers CustomerID='ALFKI'>"
sQuery = sQuery & "<ContactName>Stephen Johns</ContactName>"
sQuery = sQuery & "</Customers>"
sQuery = sQuery & "</updg:after>"
sQuery = sQuery & "</updg:sync>"
sQuery = sQuery & "</CustomerUpdate>"

' Creamos y abrimos el objeto Stream

Dim adoStreamQuery
Set adoStreamQuery = Server.CreateObject("ADODB.Stream")
adoStreamQuery.Open

' Escribimos la consulta XML dentro del objeto Stream

adoStreamQuery.WriteText sQuery, adWriteChar
adoStreamQuery.Position = 0
```

Listado 16.27: continuación

```
' Seleccionamos el objeto Stream como comando a ejecutar
 ' Observe el GUID para la propiedad Dialect, que en
 ' este caso representa el formato MSSQLXML

 adoCmd.CommandStream = adoStreamQuery

 ' Ahora indicaremos que la salida vaya al objeto Response,
 ' que también es un objeto Stream

 adoCmd.Properties("Output Stream") = Response

 ' Ya podemos ejecutar el comando con el updategram

 Response.write "Ejecutando Updategram... "
 adoCmd.Execute , , adExecuteStream
 Response.write "Updategram ejecutado"
%>

</HEAD>
<BODY>
 <H1>SQL-XML by Example</H1>
 <H3>Ejecucioacute;n de Updategrams</H3>
 <UL id=log>
 </UL>
</BODY>
</HTML>
```

Como puede ver, el ejemplo del listado 16.27 es básicamente el mismo que el del listado 16.17. Las únicas diferencias son el *updategram* y la ausencia de *script* de cliente, que en este caso no hace falta.

Uso de updategrams para insertar datos en SQL Server

En la sección precedente aprendimos a usar *updategrams* para modificar datos en SQL Server. También se pueden usar *updategrams* para insertar datos en SQL Server, siempre que no contengan una sección `<updg:before>`.

El listado 16.28 contiene un *updategram* que inserta una nueva categoría en la tabla `Categories`.

Listado 16.28: Uso de un *updategram* para insertar una nueva categoría

```
<CategoryAdd xmlns:updg="urn:schemas-microsoft-com:xml-updategram">
 <updg:sync>
   <updg:before>
   </updg:before>
   <updg:after>
```

Listado 16.28: continuación

```
      <Categories>
        <CategoryName>Nueva categ.</CategoryName>
      </Categories>
    </updg:after>
  </updg:sync>
</CategoryAdd>
```

Como puede ver en el listado 16.28, la imagen previa (`before`) está vacía y la ejecución de este *updategram* provocará el agregado de una nueva categoría con el nombre `Nueva categ.`, y con los valores predeterminados (o `NULL`) en todos los demás campos.

La operación de inserción puede ser más compleja, por ejemplo, insertar en la misma operación una nueva categoría y un nuevo producto perteneciente a esa categoría. El problema es que el identificador de categoría (`CategoryID`) es una columna de identidad y necesitamos usar ese nuevo valor en la tabla `Products`. El listado 16.29 muestra el *updategram* que lleva a cabo esta acción.

Listado 16.29: Uso de un *updategram* para insertar una nueva categoría y un nuevo producto perteneciente a esa categoría

```
<CategoryProductAdd xmlns:updg="urn:schemas-microsoft-com:xml-updategram">
 <updg:sync>
   <updg:before>
   </updg:before>
   <updg:after updg:returnid="ID">
     <Categories updg:at-identity="ID">
       <CategoryName>Otra categ.</CategoryName>
     </Categories>
     <Products CategoryID="ID" ProductName="Nuevo producto" Discontinued="1" />
   </updg:after>
 </updg:sync>
</CategoryProductAdd>
```

El ejemplo del listado 16.29 tiene algunos atributos nuevos:

```
<updg:after updg:returnid="ID">
```

Esta directiva indica devolver el valor `IDENTITY` generado automáticamente por SQL Server.

```
<Categories updg:at-identity="ID">
```

En este caso definimos un nombre de variable (`ID`) para la columna de identidad.

```
<Products CategoryID="ID" ProductName="Nuevo producto" Discontinued="1" />
```

Agregamos un nuevo producto mediante `CategoryID` como el valor de identidad recién generado y guardado en la variable `ID`.

Si ejecuta el *updategram* como un URL en Internet Explorer recibirá la respuesta que aparece en el listado 16.30.

Listado 16.30: Respuesta recibida luego de insertar una nueva categoría y un nuevo producto por medio de un *updategram*

```
- <CategoryProductAdd xmlns:updg="urn:schemas-microsoft-com:xml-updategram">
- <returnid>
 <ID>11</ID>
 </returnid>
 </CategoryProductAdd>
```

El valor 11 corresponde al nuevo valor de identidad usado para la nueva categoría.

Uso de updategrams para eliminar datos en SQL Server 2000

En la sección precedente aprendimos a usar *updategrams* sin la imagen previa para insertar nuevas filas en SQL Server. Siguiendo la misma idea, para eliminar filas en SQL Server se pueden usar *updategrams* que no tengan la imagen posterior (`after`).

El listado 16.31 muestra un *updategram* que elimina los productos y las categorías insertados en los listados 16.27 y 16.28.

Listado 16.31: Uso de *updategrams* para eliminar categorías y productos

```
<DeleteCatProducts xmlns:updg="urn:schemas-microsoft-com:xml-updategram">
 <updg:sync>
   <updg:before>
     <Products ProductName="Nuevo producto" />
     <Categories CategoryName="Otra categ." />
     <Categories CategoryName="Nueva categ." />
   </updg:before>
   <updg:after>
   </updg:after>
 </updg:sync>
</DeleteCatProducts>
```

PRECAUCIÓN

Tenga cuidado al definir la imagen `before` del *updategram*. Recuerde que es la cláusula `WHERE` de una instrucción `DELETE`.

¿Qué viene ahora?

La información contenida en este capítulo es apenas un elemento más en el panorama total de SQL Server 2000. El lector puede intentar usar muchas de las posibilidades y ejemplos de los otros capítulos junto con las técnicas de las que hablamos en este.

La programación ASP es un importante componente del desarrollo de aplicaciones con SQL Server y el nuevo paradigma ASP.NET abre nuevos horizontes para este popular entorno de programación.

Con ADO.NET, el nuevo Visual Studio.NET, el marco .NET en general y las nuevas capacidades XML, SQL Server abre una ruta en dirección a un panorama más amplio, que incluye la posibilidad de distribuir datos a más lugares y a diferentes dispositivos, y de interoperar con otros sistemas de base de datos.