# SQL Server XML y XQuery

#### Introducción

La convergencia es un concepto de gran actualidad. Los teléfonos celulares integran funcionalidades de PDA’s y de cámaras. Las PDA’s parecen teléfonos celulares.

La convergencia también ocurre en el mundo de los datos. Los clientes no quieren tratar con múltiples sistemas para tratar sus datos desestructurados, semiestructurados y estructurados.

Los clientes actualmente necesitan que sus bases de datos relacionales administren no solo sus datos estructurados sino los semiestructurados.

Existen razones por las que elegir un sistema de base de datos relacional en lugar de un sistema basado en ficheros del sistema.

Una de ellas es que probablemente queramos cruzar información entre datos relacionales y no relacionales. Por ejemplo, podemos querer mostrar las ventas de un cliente, en donde sus datos de ventas están almacenados en un sistema relacional y los datos de los clientes en formato XML.

Para este caso, si queremos cruzar entre un fichero del sistema y un sistema relacional, deberemos escribir bastante código para tratarnos entre las dos tecnologías.

Para dar soporte a estos nuevos escenarios que requieren datos XML y relacionales, Microsoft da soporte nativo XML en SQL Server.

Con SQL Server, trabajamos con datos no relacionales de la misma forma que con los relacionales. SQL Server soporta de forma nativa el almacenamiento, las consultas y las actualizaciones de los datos XML.

#### Usando el tipo de dato XML

SQL Server tiene el tipo de datos XML que se puede usar para almacenar información en formato nativo en sus bases de datos. Esto se permite desde SQL Server 2005.

Otro beneficio del uso nativo de este tipo de datos es que los datos en formato XML son diferentes de los datos relacionales tradicionales. Los datos XML tienen una estructura jerárquica que puede ser recursiva y soportan un lenguaje de consulta diferente.

Existen muchos escenarios en donde el uso del modelo relacional tiene ventajas con respecto al XML y viceversa.

Por ejemplo, si tenemos datos que están muy interrelacionados, como clientes, sus pedidos, los productos pedidos y los almacenes que suministran los productos, podríamos implementar una solución con XML, pero sería un gran desafío:

* ¿Cómo estructurar la jerarquía?
* ¿Cómo escribir una consulta que devuelva todos los clientes que nos han pedido al menos 5 veces, con al menos un importe de 300€ …?
* Otro problema es la repetición de los datos a través de la jerarquía (nombres de productos, precios, etc.
* Además, si queremos borrar un cliente pero no los productos o los pedidos del cliente, no lo podremos hacer, ya que los productos son hijos bajo el nodo cliente.

Actualmente, sin embargo, existen escenarios en donde el XML es muy interesante.

XML es más flexible que el modelo relacional. Si necesitamos un formato flexible para almacenar los datos, XML puede ser una buena elección:

* XML se describe a sí mismo y es fácilmente transportable entre aplicaciones y plataformas.
* Si los datos tienen entradas limitadas o necesitan una rica funcionalidad multivalor.
* Si tenemos documentos de Office, podemos querer almacenar esta información en XML, ya que los documentos de Office carecen de estructuras rígidas; XML suministra la capacidad para almacenar y consultar la información en un formato enriquecido.

Si elegimos XML como el formato para nuestros datos, necesitaremos decidir entre usar el tipo de datos XML o almacenar el XML en columnas de tipo varchar(max) o varbinary(max).

Si queremos usar XPath y XQuery lo correcto sería usar el tipo de datos XML en la columna.

Si necesitamos preservar los datos XML tal como fueron creados, almacenaremos el XML en una columna varchar(max) o varbinary(max). Algunos escenarios como los documentos legales pueden requerir este formato.

SQL Server puede soportar un modelo híbrido: cierta información puede estar en formato relacional, otra en formato varchar y otra en formato XML.

# Almacenar XML en SQL Server

#### Características

* XML no es almacenado como texto.
* El formato en que se almacena es en XML binario
* Con el tipo de datos XML la redundancia de los nombres de las etiquetas de los elementos y de los atributos es eliminada, estando la información más comprimida que en el caso de los textos planos.
* Este tipo de datos se implementa usando el tipo varbinary(max).

La recomendación es usar tipos XML para almacenar este tipo de información que tipos varchar por dos razones:

* La compresión suele alcanzar porcentajes como media de un 20% (depende del número y tamaño de etiquetas)
* Las recuperaciones son más rápidas ya que no se necesita convertir los datos

Si queremos comprender el tema de la compresión, podemos usar la función DATALENGTH. El ejemplo siguiente compara ambos usos:

select DATALENGTH(N'<?xml version="1.0" standalone="yes"?>

<people>

<person>

<name>

<givenName>Martin</givenName>

<familyName>Smith</familyName>

</name>

</person>

</people>

') as NVARLEN,

DATALENGTH(CAST(N'<?xml version="1.0" standalone="yes"?>

<people>

<person>

<name>

<givenName>Martin</givenName>

<familyName>Smith</familyName>

</name>

</person>

</people>

' AS XML)) as XMLLEN

Como podemos ver, se ahorra en este caso más de tres veces de espacio usando el tipo XML.

#### Columnas XML

**Creación**

Este código crea una nueva tabla que contiene una columna con restricción PK y una columna de tipo XML.

CREATE TABLE DVDList (pk INT IDENTITY PRIMARY KEY, xmlColumna XML not null)

**Características**

Podemos tener múltiples columnas XML en una tabla.

En este ejemplo no tenemos esquemas asociados con la columna XML.

SQL Server sabemos que soporta ambos tipos (“typed” y “untyped”):

* Las columnas “Untyped” no tienen esquema asociado con ellas.
* Las columnas “Typed” tienen esquemas XML asociados con los que la información a ser almacenada debe ser contrastada.

Es mejor asociar esquemas XML ya que SQL Server validará los datos, optimizando mejor las consultas y modificaciones de los datos, mejorando los chequeos y optimizando su almacenamiento.

#### Esquemas

* Si la colección schema no existe, el código crea la colección schema en SQL Server.
* Estas colecciones tienen el ámbito de base de datos.
* No se pueden expandir a otras bases de datos o instancias, por lo que si los queremos utilizar en otros lugares, deberemos recrearlos en dichas bases de datos o instancias.
* Una **advertencia** con los esquemas es que una vez creado un esquema, no podemos modificarlo o borrarlo hasta que todas las referencias a él hayan sido eliminadas del sistema.
* El esquema es cargado en los metadatos de SQL Server y se pueden visualizar usando la vista de metadatos sys.xml\_schema\_collections.

DECLARE @x XML

SELECT @x = s

FROM OPENROWSET (

BULK 'F:\XML\XML1\DVDListSchema.xsd',SINGLE\_BLOB) AS TEMP(s)

select @x

-- Nos aseguramos de que el esquema no existe

IF EXISTS(select \* from sys.xml\_schema\_collections where name='DVDList')

DROP XML SCHEMA COLLECTION DVDList2

-- Creamos el esquema en la colección SCHEMA de la base de datos

CREATE XML SCHEMA COLLECTION DVDList2 AS @x

GO

Primero,necesitamos cargar nuestro esquema XML en SQL Server.

El código usa OPENROWSET para abrir el fichero de Schema almacenado en el sistema de ficheros.

El código asigna el esquema a la variable x.

Después, eliminamos el esquema si existe.

**Recuperar esquema**

Si queremos recuperar el esquema después de cargarlo en el sistema, necesitaremos usar la función xml\_schema\_namespace.

Esta función toma dos parámetros:

* El primero es el esquema relacional en el que schema XML fue creado
* El segundo es el nombre del esquema que queremos recuperar

Este código recupera el esquema autores creado en el ejemplo anterior:

--Devuelve el schema

USE xmldb

go

SELECT xml\_schema\_namespace(N'dbo',N'autores')

go

#### Definición de columnas XML *Typed*

Para crear una columna XML Typed, necesitamos cargar nuestro esquema en SQL Server y después asociarlo con la columna.

Una vez realizada la operación, solo se permitirán altas de documentos asociados al esquema.

Podemos tener uno o varios esquemas asociados con una columna XML.

Este código crea una nueva tabla que usa un esquema para el tipo de dato XML de la columna:

CREATE TABLE DVDList2 (pk INT IDENTITY PRIMARY KEY,

xmlColumna XML (CONTENT DVDList2))

GO

# Inserción de datos en columnas XML

Además de dicha posibilidad existen tres posibilidades más de hacerlo:

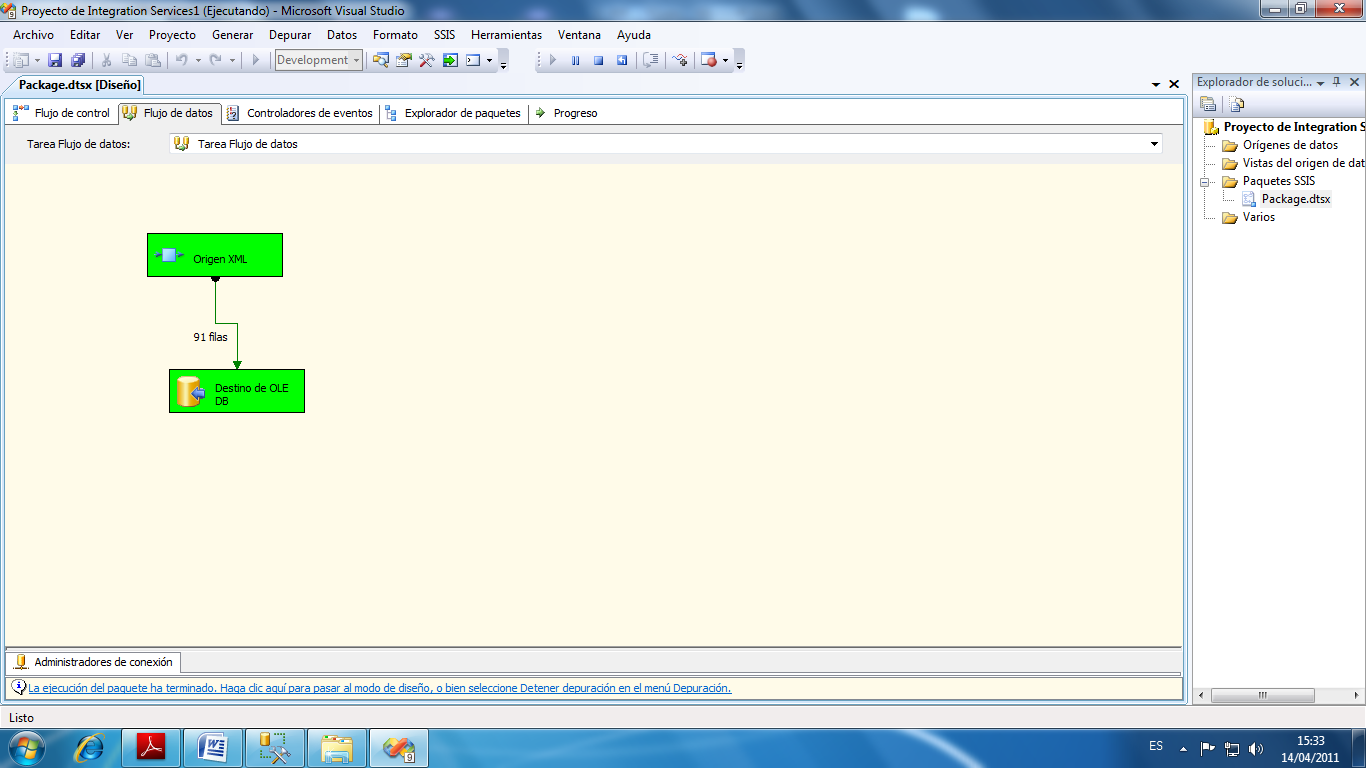
* SSIS – SQL Server Integration Services
* Carga masiva con OPENROWSET y bcp
* Escritura de consultas o aplicaciones.

#### SSIS con XML

Esta utilidad nos facilita la labor de mover datos entre distintas fuentes de datos.

SSIS admite múltiples formas de mover datos entre XML y el modelo relacional.

Observemos el siguiente mapa:



Un componente interesante de SSIS es la Tarea XML. Esta tarea permite coger varios documentos XML y combinarlos o bien transformar el documento usando XSLT.

Los resultados se pueden dejar en un fichero o en una variable SSIS que puede ser usada por el resto de componentes de flujo SSIS.

XQuery no es una opción en la tarea XML. En su lugar, si queremos ejecutar un XQuery, necesitaremos usar la utilidad de tareas Execute SQL y usar los métodos use XQuery sobre un tipo de dato XML almacenado en la Base de datos de SQL Server.

#### Bulk-Load XML

Podemos cargar los datos XML en SQL Server usando la palabra clave OPENROWSET. Este ejemplo captura los datos de un fichero del sistema (XML) y los inserta en una columna de una tabla de SQL Server:

INSERT INTO DVDList (xmlColumna)

SELECT \*

FROM OPENROWSET (

BULK 'F:\XML\XML1\DVDList.xml',

SINGLE\_BLOB) as x;

Usar un proveedor bulk-load dentro de OPENROWSET es similar a usar el mandato BULK INSERT, pero no necesitamos enviar la salida a una tabla en el caso del proveedor bulk-load.

#### Aplicaciones que graban los datos consultados

Otra forma de almacenar los datos dentro de columnas XML es escribir aplicaciones que inserten los datos.

La aplicación podría ser un script T-SQL ejecutado en Management Studio o una aplicación cliente que inserta los datos usando ADO.NET o ADO.

Cualquiera que sea el método utilizado, insertaremos XML usando el mandato INSERT INTO.

Este es un ejemplo de uso de un script:

INSERT INTO DVDList(xmlColumna)

VALUES ( '<?xml version="1.0" encoding="UTF-8"?>

<!--Creado @ 05/06/2013 12:45:37-->

<DvdList><DVD Category="Science Fiction" ID="1"><Title>The Matrix</Title><Director>Sr.</Director><Price>18.74</Price>-<Starring><Star>Keanu Reeves</Star><Star>Laurence Fishburne</Star></Starring></DVD>-<DVD Category="Drama" ID="2"><Title>Forrest Gump</Title><Director>Robert Zemeckis</Director><Price>23.99</Price>-<Starring><Star>Tom Hanks</Star><Star>Robin Wright</Star></Starring></DVD></DvdList>')

)

En las aplicaciones, podemos usar SQL dinámico para añadir las filas o crear procedimientos almacenados para ello.

# Consultando datos XML

Una vez tenemos los datos en el servidor, necesitamos cogerlos.

El tipo de datos XML suministra cuatro métodos para llevarlo a cabo: query(), value(), exist(), and nodes() y modify()

Estos métodos están basados en el uso del lenguaje XQuery.

#### Métodos de consulta básicos de XML

Ayuda SQL Server

ms-help://MS.SQLCC.v10/MS.SQLSVR.v10.es/s10de\_6tsql/html/d112b9c9-be9f-435c-a9e6-d21b65778fb7.htm

Vamos a ver cuatro métodos para consultar tipos de datos XML:

* query(): Devuelve el XML que se empareja con la consulta.
* value(): Devuelve un valor escalar de XML
* exist(): Chequea la existencia del XML especificado en el tipo de dato XML.

#### query()

Este método recoge un mandato y devuelve el XML emparejado con la consulta.

El XML es recogido como un XML no tipado y puede ser fuertemente analizado si es necesario.

Para ver cómo trabaja, creamos una nueva tabla, xmltabla, para almacenar el XML. Esta tabla la vamos a usar para aprender a consultar y modificar el XML.

-- Creamos una nueva tabla

CREATE TABLE xmltabla (pk INT IDENTITY PRIMARY KEY, gente XML)

GO

--Insertamos datos en la tabla

INSERT INTO xmltabla (gente)

SELECT \*

FROM OPENROWSET (

BULK 'C:\gente.xml',

SINGLE\_BLOB) AS TEMP

GO

Este ejemplo usa el método query() para buscar a las personas que tienen un elemento edad, y después devuelve el XML que identifica a cada persona por el nombre.

SELECT gente.query

(

'

for $p in (/gente)

where $p//edad

return

<persona>

<identifica>{$p//nombre}</identifica>

</persona>

'

)

FROM xmltabla

#### value()

Este método devuelve un valor escalar desde la instancia XML. Este método recibe dos argumentos: el primero es la consulta XQuery que queremos utilizar para capturar el valor, y el Segundo es el tipo T-SQL al que queremos que el valor sea convertido.

No podemos convertir a tipos timestamp, NTEXT, TEXT, o IMAGE. Tampoco podemos convertir a tipos de datos XML o sql\_variant.

Queremos, lógicamente emparejar tipos de datos XML y SQL. Este código captura la edad de la segunda persona en el documento XML y la devuelve como un entero:

SELECT gente.value('/gente[1]/persona[2]/edad[1][text()]', 'integer')

as edad FROM dbo.xmltablanueva

where pk=1

#### exist()

Este método es usado para chequear la existencia de un tipo de datos XML. Devuelve 1 si la expresión XQuery devuelve un resultado no nulo y 0 en el caso contrario.

En este ejemplo devolvemos las columnas en el mandato SELECT en donde el elemento nombre de la persona almacenada en el documento XML es igual a Laida:

SELECT pk, gente FROM xmltabla

WHERE gente.exist('/gente/persona/identifica/nombre[.="Laida"]') = 1

#### XQuery

Los datos XML están estructurados de forma distinta que los relacionales, y su tratamiento supone un cierto problema de adaptación para los programadores relacionales.

SQL Server soporta XPath y XQuery. XQuery es el “gran hermano” de XPath.

Vamos a trabajar con el siguiente ejemplo de documento XML:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<gente>

<persona>

<identifica>

<nombre>Martin</nombre>

<apellido>Martinez</apellido>

</identifica>

<edad>33</edad>

<peso>ligero</peso>

</persona>

<persona>

<identifica>

<nombre>Laida</nombre>

<apellido>Eceiza</apellido>

</identifica>

<edad>40</edad>

<peso>ligero</peso>

</persona>

<persona>

<identifica>

<nombre>Tomas</nombre>

<apellido>Alfaro</apellido>

</identifica>

<edad>30</edad>

<peso>medio</peso>

</persona>

</gente>

Para recuperar los nombres de todas las personas, ejecutamos el siguiente mandato:

/gente//identifica

Para trasladar esta consulta XPath a XQuery, encerraremos la consulta entre paréntesis, lo que advertirá al procesador de que la consulta es una expresión XQuery y no un literal string.

#### FLWOR

* Son las siglas de For-Let-Where-Order By-Return.
* Es similar a los mandatos T-SQL’s SELECT, FROM, y WHERE, en que pueden devolver información condicional basada en el criterio que asignemos.
* En lugar de devolver datos relacionales como es el caso de T-SQL, XQuery devuelve datos XML.

Las partes de FLOWR son:

* **For** permite especificar el XML sobre el que deseamos iteractuar. El XML lo especificamos usando la expresión XPATH. Normalmente son etiquetas XML de inicio o de final. Esto dependerá de en qué nodos nos interese iteractuar, y del documento XML devuelto.
* **Let**: permite asignar una variable a una expresión, pero no iteractúa por los valores.
* **Where** es similar a la cláusula WHERE de T-SQL. Esta expresión evalúa un valor booleano. Cualquier valor que devuelva True es admitido y si devuelve False es rechazado.
* **Order By** es similar a la de la SELECT de T-SQL. Permite ordenar los resultados usando el orden que especifiquemos.
* **Return** especifica el contenido que debería ser devuelto.

El ejemplo iteractúa sobre el XML contenido en una tabla y devuelve solo el nombre de las personas que tienen elemento edad en el documento XML.

SELECT xmlColumna.query(

'for $p in //DVD

where $p//Title

return

<persona>

<identifica>{$p//Title}</identifica>

</persona>

'

)

FROM DVDList

Esta consulta XQuery cuenta el número de elementos persona que hay en un documento XML. También muestra el uso del método de consulta del tipo de dato con XQuery.

SELECT xmlColConSchema.query(

'count(//persona)

')

FROM XMLtablanueva

Go

Aparte de FLWOR, XQuery soporta funciones como avg, min, max, ceiling, floor, and round.

El siguiente ejemplo muestra cómo calcular la edad media de las personas de un documento XML.

SELECT xmlColConSchema.query(

'round(avg(//edad))

')

FROM XMLtablanueva

Go

# Modificando datos XML

SQL Server tiene un método modify() en XQuery. Este método permite modificar partes de los datos XML almacenados.

Podemos añadir o eliminar partes, sustituir valores, etc.

Este método incluye todos los comandos DML.

Al hacer estas operaciones SQL Server ejecuta solo las operaciones necesarias sobre el XML en lugar de sustituir el texto entero.

#### Inserción de un elemento

Con el comando insertar podemos insertar XML como el primer elemento o como el último elemento.

También podemos insertar antes, después, o como descendiente directo de un elemento XML existente.

También podemos insertar atributos.

Podemos además usar variables T-SQL escalares de tipo XML.

Por ejemplo, imaginemos la siguiente lista de la compra:

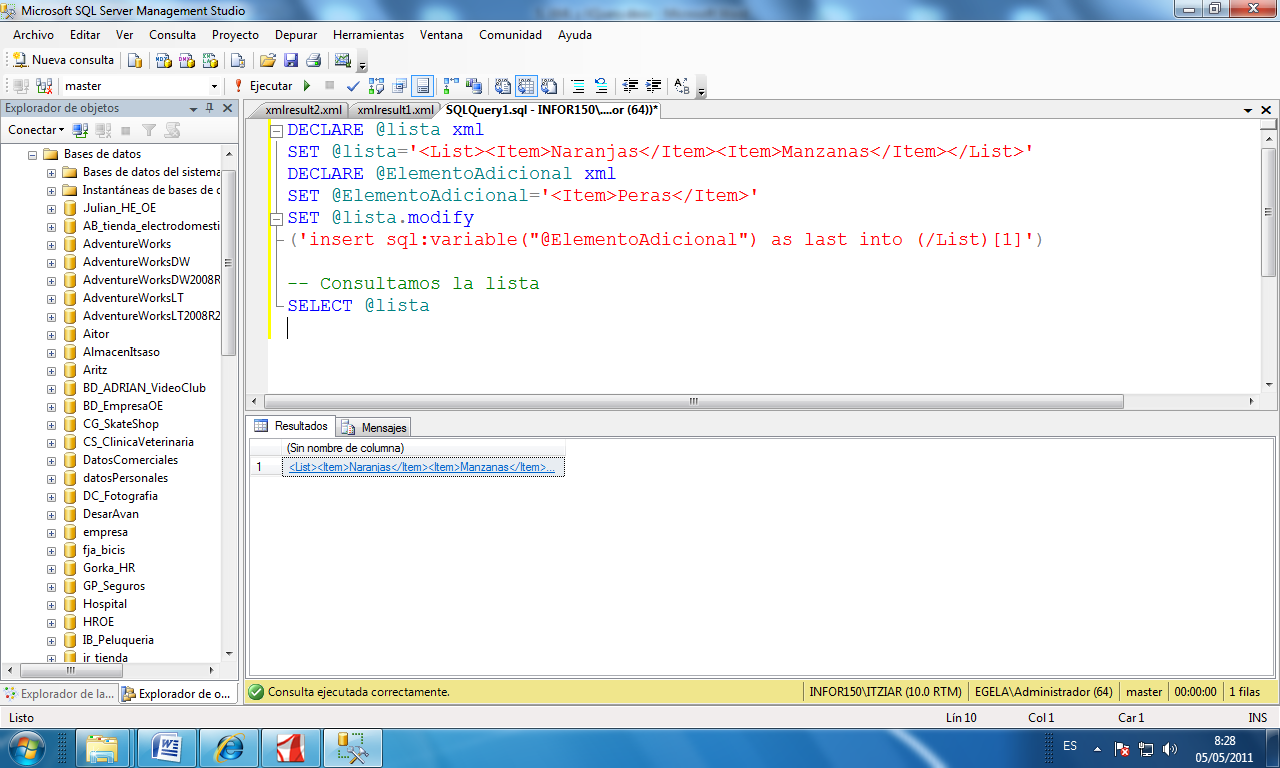
<List>

<Item>Bananas</Item>

<Item>Apples</Item>

</List>

Supongamos que queremos insertar un valor almacenado en una variable T-SQL dentro de la lista como otro elemento.



The result of the @GroceryList variable is as follows:

<List>

<Item>Naranjas</Item>

<Item>Manzanas</Item>

<Item>Peras</Item>

</List>

#### Borrado de un elemento

Con el comando delete, podemos eliminar elementos XML o atributos del documento.

Por ejemplo:

DECLARE @lista xml

SET @lista='<List><Item>Naranjas</Item><Item>Manzanas</Item></List>'

DECLARE @ElementoAdicional xml

SET @ElementoAdicional='<Item>Peras</Item>'

-- Insertamos un elemento al final

SET @lista.modify

('insert sql:variable("@ElementoAdicional") as last into (/List)[1]')

-- Eliminamos el elemento que ocupa la posición 2 (Manzanas)

SET @lista.modify ('delete /List/Item[2]')

-- Consultamos la lista

SELECT @lista

#### Cambiar el valor de un nodo

El comando replace value of nos permite sustituir un nodo con un nuevo valor que especifiquemos. No se pueden seleccionar varios nodos, solo se puede sustituir de uno en uno.

Ejemplo:

--Consultamos los datos

SELECT \* FROM xmltabla

GO

--Insertamos un nuevo valor

UPDATE xmltabla SET gente.modify(

'insert <ColorFavorito>Verde</ColorFavorito>

as last into (/gente/persona[3])[1]')

WHERE pk=1

GO

--Consultamos los datos

SELECT \* FROM xmltabla

GO

--Modificamos el valor

UPDATE xmltabla SET gente.modify(

'replace value of (/gente/persona[3]/ColorFavorito[1]/text())[1]

with "Azul"')

WHERE pk=1

GO

--Consultamos los datos

SELECT \* FROM xmltabla

GO

--Eliminamos el valor

UPDATE xmltabla SET gente.modify(

'delete /gente/persona[3]/ColorFavorito')

WHERE pk=1

GO

--Consultamos los datos

SELECT \* FROM xmltabla

GO