XQuery. Un lenguaje de consulta para XML.

Nieves Carralero Colmenar.
IES Pedro Mercedes. Junta de Comunidades de Castilla-La Mancha. España.

Resumen

En este artículo se muestra una introducción práctica al lenguaje XQuery. XQuery es un lenguaje de consulta y procesamiento de datos XML propuesto por la W3C. Surge como un equivalente natural de SQL pero para datos XML. *XQuery* es una extensión de *XPath 2.0* que se especifica con nueva funcionalidad, potenciando sus posibilidades: Por ello, cualquier sentencia *XPath 2.0* (y por tanto *XPath 1.0* también) es una sentencia *XQuery 1.0* válida y debe devolver los mismos resultados

Lenguaje de consulta para XML: XQuery (Xml Query Language).

XQuery es un lenguaje de consulta y procesamiento de datos XML propuesto por la W3C. Surge como un equivalente natural de SQL pero para datos XML. *XQuery* es una extensión de *XPath 2.0* que se especifica con nueva funcionalidad, potenciando sus posibilidades: Por ello, cualquier sentencia *XPath 2.0* (y por tanto *XPath 1.0* también) es una sentencia *XQuery 1.0* válida y debe devolver los mismos resultados.

Al igual que ocurre con *XPath 2.0*, la definición completa del lenguaje *XQuery* es muy extensa y su especificación está formada por varios documentos, por ello en esta sección sólo se comentarán las más destacadas con el fin de hacer únicamente una introducción. Siempre que no se diga lo contrario, las consultas ejecutadas se harán sobre el documento siguiente:

```
<!-- Base de datos de libros en Castellano -->
<Libros>
  <Libro>
    <Autor>Nikolai Gogol</Autor>
    <Titulo>El Capote</Titulo>
  </Libro>
  <Libro>
    <Autor>Gonzalo Giner</Autor>
    <Titulo>El Sanador de Caballos</Titulo>
  </Libro>
  <Libro>
    <Autor>Umberto Eco</Autor>
    <Titulo>El Nombre de la Rosa</Titulo>
  </Libro>
</Libros>
```

FLWOR (de For Let Where Order Return).

Es una sentencia que permite la unión de variables sobre conjuntos de nodos y la iteración sobre el resultado. FLWOR. Esta sentencia tiene la siguiente estructura:

- La clausula FOR vincula una o más variables a expresiones escritas en *XPath*, creando un flujo de tuplas en el que cada tupla está vinculada a una de las variables.

- La clausula LET vincula una variable al resultado completo de una expresión añadiendo esos vínculos a las tuplas generadas por una cláusula FOR o, si no existe ninguna cláusula FOR, creando una única tupla que contenga esos vínculos.

- La clausula WHERE permite establecer las condiciones que deben cumplir las tuplas devueltas como resultado.

- La clausula ORDER BY se utiliza para ordenar las tuplas devueltas según un criterio dado.

- La clausula RETURN muestra la estructura con la que se devolverá el resultado. Construye el resultado de la consulta para una tupla dada, después de haber sido filtrada por la cláusula WHERE y ordenada por la cláusula ORDER BY.

FOR.

El objetivo de la sentencia FOR es obtener una secuencia de tuplas; se puede asimilar el concepto de tupla a un valor o nodo del árbol del documento de origen. El formato de la sentencia FOR es similar al formato en *XPath 2.0* y consta de:

• *variable*: variable sobre la que se almacenan los nodos

• *secuencia enlazada*: nodos que se van a tratar

for 'variable' in 'secuencia enlazada'

Con esta clausula se asocia una secuencia de tuplas a la variable. Para cada ocurrencia que se consiga en '*secuencia enlazada"* se obtiene una tupla. Se pueden especificar varias variables en la misma sentencia; si se hace así se obtiene una tupla para el producto cartesiano de todas ellas.

La siguiente consulta obtiene todos los autores dentro de etiquetas <*MisAutores*>. Para ello, primero la consulta recupera todos los nodos *//Libros/Libro/Autor* que identifican en la variable *$a*. Seguidamente, cada una de las tuplas almacenadas en *$a* se colocan entre etiquetas <*MisAutores*>

```
for $a in //Libros/Libro/Autor
return <MisAutores>{$a}</MisAutores>
```

El resultado sería:

```
<MisAutores>
   <Autor>Nikolai Gogol</Autor>
</MisAutores>
<MisAutores>
   <Autor>Gonzalo Giner</Autor>
</MisAutores>
<MisAutores>
   <Autor>Umberto Eco</Autor>
</MisAutores>
```

La siguiente consulta, sin tener mucho sentido práctico, muestra el uso de dos variables y el producto cartesiano que se produce entre las tuplas que cada una alberga.

```
for $a in //Libros/Libro/Autor,$b in //Libros/Libro/Titulo
return <TituloAutor>{$a,$b}</TituloAutor>
```

El resultado son 9 elementos <TituloAutor> (3x3):

```
<TituloAutor>
   <Autor>Nikolai Gogol</Autor>
   <Titulo>El Capote</Titulo>
</TituloAutor>
<TituloAutor>
   <Autor>Nikolai Gogol</Autor>
   <Titulo>El Sanador de Caballos</Titulo>
</TituloAutor>
<TituloAutor>
   <Autor>Nikolai Gogol</Autor>
   <Titulo>El Nombre de la Rosa</Titulo>
</TituloAutor>
<TituloAutor>
   <Autor>Gonzalo Giner</Autor>
   <Titulo>El Capote</Titulo>
</TituloAutor>
<TituloAutor>
   <Autor>Gonzalo Giner</Autor>
   <Titulo>El Sanador de Caballos</Titulo>
</TituloAutor>
```

```
<TituloAutor>
   <Autor>Gonzalo Giner</Autor>
   <Titulo>El Nombre de la Rosa</Titulo>
</TituloAutor>
<TituloAutor>
   <Autor>Umberto Eco</Autor>
   <Titulo>El Capote</Titulo>
</TituloAutor>
<TituloAutor>
   <Autor>Umberto Eco</Autor>
   <Titulo>El Sanador de Caballos</Titulo>
</TituloAutor>
<TituloAutor>
   <Autor>Umberto Eco</Autor>
   <Titulo>El Nombre de la Rosa</Titulo>
</TituloAutor>
```

Debido a que en los documentos XML no sólo son relevantes los datos que contienen sino también la posición que ocupan, la clausula FOR permite utilizar variables de posición que recuperan el lugar que ocupan cada elemento en el documento (*at*). Por ejemplo, la siguiente consulta devuelve los autores, poniendo como atributo del elemento *<MisLibros>* un atributo *posición* con la posición que ocupa el elemento. La estructura *$a at $p* asigna a *$p* las posiciones de las tuplas que tenga *$a* en la consulta.

```
for $a at $p in //Libros/Libro
return <MisLibros posicion="{$p}">
        {$a/Autor}
        </MisLibros>
```

El resultado sería:

```
<MisLibros posicion="1">
   <Autor>Nikolai Gogol</Autor>
</MisLibros>
<MisLibros posicion="2">
   <Autor>Gonzalo Giner</Autor>
</MisLibros>
<MisLibros posicion="3">
   <Autor>Umberto Eco</Autor>
</MisLibros>
```

LET

El objetivo de la sentencia LET es el mismo de la sentencia FOR, obtener una secuencia de tuplas, pero la forma de hacerlo es distinta. El formato de la sentencia es el siguiente (observar que la asignación entre la variable y la secuencia enlazada se hace con *:=* y no con *in* como ocurre en FOR):

let 'variable' := *'secuencia enlazada'*

A diferencia de FOR, LET obtiene un única tupla y no varias. Por ejemplo, la primera consulta ejecutada hecha con un LET quedaría:

```
let  $a := //Libros/Libro/Autor
return <MisAutores>{$a}</MisAutores> <MisAutores>
```

El resultado sería los autores en un único *<MisAutores>:*

```
<MisAutores>
   <Autor>Nikolai Gogol</Autor>
   <Autor>Gonzalo Giner</Autor>
   <Autor>Umberto Eco</Autor>
</MisAutores>
```

WHERE

La cláusula WHERE es una cláusula opcional que sirve para seleccionar determinadas tuplas del flujo generado por las sentencias FOR y LET. La sintaxis de la sentencia es

WHERE 'expresión'

Donde '*expresión*' es evaluada a booleano para cada tupla del flujo, tratándola si el resultado es *true*, descartándola si es *false*. La forma de obtener un booleano de la expresión es igual que en la sentencia condicional de *XPath 2.0*.

Por ejemplo la siguiente consulta recupera todos los <*Titulo*> de los libros cuyo <*Autor*> tenga entre sus primeros 7 caracteres la cadena "NIKOLAI". Para ello en *$a* se recuperan todos los elementos <*Titulo*> (3 en total) y sobre ellos aplica la condición del WHERE (que solo uno cumple). Para la condición se ha usado una función *substring()* que devuelve una sub-cadena dada entre dos posiciones (en nuestro caso 1 y 7) y *upper-case()* que convierte en mayúsculas un texto. Si el resultado de aplicar esas dos funciones es una cadena igual a 'NIKOLAI', entonces la tupla se incluye como resultado y se le aplica la clausula RESULT.

```
for $a in //Libros/Libro/Autor
where upper-case(substring($a,1,7)) ='NIKOLAI'
return
<LibrosNikolai>{$a/ancestor::Libro/Titulo}</LibrosNikolai>
```

La consulta, además, utiliza *ancestor::* para hacer referencia al padre de *$a* y así poder acceder al *Título* del libro, ya que es lo que se quiere recuperar. El resultado sería:

```
<LibrosNikolai>
   <Titulo>El Capote</Titulo>
</LibrosNikolai>
```

ORDER BY

La sentencia ORDER BY se utiliza para ordenar el resultado. Se evalúa antes de la sentencia RETURN y después del WHERE. La forma de ordenar la secuencia de tuplas viene dada por ascendiente (*ascending*) o descendiente (*descending*).

Debido a que XML no tiene porqué tener una estructura fija, es posible que haya nodos que cumplan la condición del WHERE pero que no contengan el nodo por el que se ordenan (vacios). En ese caso *XQuery* ofrece dos opciones: los vacios se ponen con mayor prioridad (*empty greatest*) o con menor prioridad (*empty least*) en la salida.

Por ejemplo la siguiente consulta obtiene los autores de los libros cuya posición es mayor que 1 y menor que 4. El resultado es ordenado por la posición descendiente.

```
for $a at $p in //Libros/Libro
where $p>1 and $p<4
order by $a descending
return <MisLibros posicion="{$p}">{$a/Autor} </MisLibros>
```

El resultado sería:

```
<MisLibros posicion="3">
   <Autor>Umberto Eco</Autor>
</MisLibros>
<MisLibros posicion="2">
   <Autor>Gonzalo Giner</Autor>
</MisLibros>
```

RETURN

Como se puede extraer de los ejemplos anteriores, la sentencia RETURN es evaluada una vez para cada tupla dentro de la secuencia de tuplas obtenida con las sentencias FOR, LET, WHERE y en el orden especificado por ORDER BY. La potencia de esta sentencia radica en que permite la creación de nuevos elementos y por tanto poder dar una salida con una estructura distinta a la los datos originales. Esta funcionalidad de *XQuery* da muchas posibilidades para su integración en aplicaciones basadas en tecnología XML.

La creación de nodos en la salida puede hacerse como en los ejemplos anteriores, poniendo la variables entre llaves {}, o usando palabras reservadas para indicar el tipo de nodo que se desea crear: *element nombre { }* para elementos y *attribute nombre { }* para atributos. Aplicando estas palabras reservadas a una consulta similar a la anterior pero con una condición diferente el resultado sería:

```
for $a at $p in //Libros/Libro
where $p=1
order by $a descending
return element MisLibros{
    attribute posicion {$p},
    attribute titulo {$a/Titulo}
}
```

La consulta devolvería:

```
<MisLibros posicion="1" titulo="El Capote"/>
```

Lo mostrado en estos ejemplos es solo la punta del iceberg de todo lo que *XQuery* ofrece para consultar datos XML: hacer reuniones, declarar variables, definir funciones son algunas de esas funciones.

SOCIEDAD DE LA INFORMACION

www.sociedadelainformacion.com

Edita:

Director: José Ángel Ruiz Felipe

Jefe de publicaciones: Antero Soria Luján

D.L.: AB 293-2001

ISSN: 1578-326x