XQuery

Yolanda García Ruiz (UCM)

January 19, 2011

XQuery - Introducción

- XQuery es a XML lo mismo que SQL es a las bases de datos relacionales
- Lenguaje que **permite definir** de forma rápida y compacta, **consultas** o recorridos complejos **sobre colecciones de datos en XML** los cuales devuelven todos los nodos que cumplan ciertas condiciones
- Este lenguaje es **declarativo**, es decir, independiente de la forma en que se realice el recorrido o de donde se encuentren los datos
- Es **independiente de la fuente de datos**: Esta fuente de datos puede ser **archivos XML** hasta **bases de datos relacionales** con funciones de conversión de registros a XML

XQuery - Introducción

- XQuery y SQL puedan considerarse similares (con ciertos matices). El modelo de datos sobre el que se sustenta XQuery es muy distinto del modelo de datos relacional sobre el que sustenta SQL
- XML incluye conceptos como **jerarquía** y **orden de los datos** que no están presentes en el modelo relacional
- En XQuery el orden en el que se encuentren los datos es importante y determinante, ya que no es lo mismo buscar una etiqueta $< B >$ dentro de una etiqueta $< A >$ que todas las etiquetas $< B >$ del documento

XQuery - Introducción

- XQuery ha sido construido sobre la base de Xpath
- Xpath es un lenguaje declarativo para la localización de nodos y fragmentos de información en árboles XML
- XQuery se basa en este lenguaje para realizar la selección de información y la iteración a través del conjunto de datos

Consultas

- Una consulta en XQuery es una expresión que lee una secuencia de datos en XML y devuelve como resultado otra secuencia de datos en XML
- En XQuery las expresiones y los valores que devuelven son **dependientes del contexto**. Por ejemplo, los nodos del resultado dependen de los namespaces, de la posición donde aparezca la etiqueta raíz del nodo (dentro de otra, por ejemplo), etc.
- En XQuery las consultas pueden estar compuestas por cláusulas de hasta cinco tipos distintos
- Las consultas siguen la norma **FLWOR** (leído como flower), siendo FLWOR las siglas de **For**, **Let**, **Where**, **Order** y **Return**

FLWOR

- **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 variable.

FLWOR

- **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 variable.
- **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.

FLWOR

- **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 variable.
- **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.
- **Where:** Filtra las tuplas eliminando todos los valores que no cumplan las condiciones dadas

FLWOR

- **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 variable.
- **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.
- **Where:** Filtra las tuplas eliminando todos los valores que no cumplan las condiciones dadas
- **Order by:** Ordena las tuplas según el criterio dado

FLWOR

- **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 variable.
- **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.
- **Where:** Filtra las tuplas eliminando todos los valores que no cumplan las condiciones dadas
- **Order by:** Ordena las tuplas según el criterio dado
- **Return:** Construye el árbol XML resultado de la consulta para una tupla dada, después de haber sido filtrada por la cláusula where y ordenada por la cláusula order by

Nota: En XQuery, cuando usamos el térmico tupla, nos estamos refiriendo a cada uno de los valores que toma una variable

Ejemplo

- En el siguiente ejemplo de cláusula `for`, la variable `$b` tomará como valor cada uno de los nodos libros que contenga en archivo "libros.xml".
- Cada uno de esos nodos libros, será una tupla vinculada a la variable `$b`.

```
for $b in doc("libros.xml)//bib/libro
```

Ejemplo

La siguiente consulta devuelve los títulos de los libros del año 2.000

```
for $b in doc("libros.xml")//libro
where $b/@año = "2000"
order by  $b/titulo
return $b/titulo
```

cuyo resultado es:

```
<titulo>Data on the Web</titulo>
<titulo>Millenium</titulo>
```

Ejemplo

La siguiente consulta devuelve los títulos de los libros que tengan más de dos autores ordenados por su título

```
for $b in doc("libros.xml")//libro
let $c := $b//autor
where count($c) > 2
order by $b/titulo
return $b/ titulo
```

cuyo resultado es:

```
<title>Data on the Web</title>
```

- **For y Let:**
  - Sirven para crear las tuplas con las que trabajará el resto de las cláusulas de la consulta
  - Pueden usarse tantas veces como se desee en una consulta, incluso dentro de otras cláusulas
- **Where:**
  - Filtra las tuplas. Solo puede declararse una única cláusula Where
- **Order by:**
  - Ordena las tuplas. Solo puede declararse una única cláusula Order by
  - Es posible especificar varios criterios de ordenación separándolos por comas
- **Return:**
  - Transforma las tuplas. Solo puede declararse una única cláusula Return

- No es necesario que aparezca ninguna de las cinco cláusulas FLWOR en una consulta.
- Como XQuery está construido sobre XPath, una expresión XPath es una consulta válida y no tiene ninguna de las 5 cláusulas vistas

Funcionamiento de una consulta

La siguiente consulta devuelve los títulos de los libros del año 2000

```
for $b in doc("libros.xml")//libro
where $b/@año = "2000"
order by  $b/titulo
return $b/titulo
```

1. La cláusula **for** asigna a la variable $b cada uno de los nodos <libro> .... <\libro> que aparecen en cualquier lugar del documento XML.

```
<libro año="1994"> <titulo>TCP/IP Illustrated ... </libro>
<libro año="2000"> <titulo>Millenium... </libro>
<libro año="1992"> <titulo>Advan Programming  ...</libro>
<libro año="2000"> <titulo>Data on the Web ... </libro>
<libro año="1999"> <titulo>Economics of ...</libro>
```

Funcionamiento de una consulta

La siguiente consulta devuelve los títulos de los libros del año 2000

```
for $b in doc("libros.xml")//libro
where $b/@año = "2000"
order by  $b/titulo
return $b/titulo
```

2. La cláusula **where** hace una criba de los valores de la variable $b comprobando para cada valor de $b si el atributo año es igual a 2000. Si es así, ese valor se incluye en la solución

```
<libro año="2000"> <titulo>Millenium... </libro>
<libro año="2000"> <titulo>Data on the Web ... </libro>
```

Funcionamiento de una consulta

La siguiente consulta devuelve los títulos de los libros del año 2000

```
for $b in doc("libros.xml")//libro
where $b/@año = "2000"
order by  $b/titulo
return $b/titulo
```

3. La cláusula **order by** ordena los nodos título que han pasado la criba por el orden alfabético de su contenido

```
<libro año="2000"> <titulo>Data on the Web ... </libro>
<libro año="2000"> <titulo>Millenium... </libro>
```

Funcionamiento de una consulta

La siguiente consulta devuelve los títulos de los libros del año 2000

```
for $b in doc("libros.xml")//libro
where $b/@año = "2000"
order by  $b/titulo
return $b/titulo
```

4. La cláusula **return** devuleve como resultado de la consulta los nodos que hayan pasado la criba

```
<titulo>Data on the Web</titulo>
<titulo>Millenium</titulo>
```

Diferencias entre las cláusulas for y let

- A primera vista la cláusula **for** y **let** pueden parecer iguales pero, aunque su objetivo es el mismo, trabajan de diferente forma
- La cláusula **for** vincula una variable con cada nodo que encuentra en la colección de datos
- Para la consulta:

```
for $b in doc("libros.xml")/bib/libro/titulo
where $b/@año = '2000'
return <titulos>{ $b }</titulos>
```

- La cláusula **for** asigna a la variable $b cada uno de los libros del año 2000 que aparecen en cualquier lugar del documento XML. En el resultado se repiten 2 veces la etiqueta `<titulos>`

```
<titulos> <titulo>Millenium</titulo> </titulos>
<titulos> <titulo>Data on the Web</titulo> </titulos>
```

Diferencias entre las cláusulas for y let

- La cláusula **let** vincula una variable con todo el resultado de una expresión
- Sustituyendo **for** por **let** en la consulta anterior:
- Para la consulta:

```
let $b in doc("libros.xml")/bib/libro/titulo
where $b/@año = '2000'
return <titulos>{ $b }</titulos>
```

- Obtenemos el siguiente resultado:

```
<titulos>
   <titulo>Millenium</titulo>
   <titulo>Data on the Web</titulo>
</titulos>
```

- La variable $b se vincula una sola vez a todos los títulos de todos los librosdel año 2000. Aparece las etiqueta `<titulos>` una sola vez

Ejemplo

- Si una cláusula **let** aparece en una consulta que ya posee una o más cláusulas **for**, los valores de la variable vinculada por la cláusula **let** se añaden a cada una de las tuplas generadas por la cláusula **for**.

```
for $b in doc("libros.xml")//libro
let $c := $b/autor
return <libro>{ $b/titulo,
                <autores>{ count($c) }</autores>}
       </libro>
```

- Obtenemos el siguiente resultado:

```
...
```

Ejemplo

- Obtenemos el siguiente resultado:

```
<libro>
  <titulo>TCP/IP Illustrated</titulo>
  <autores>1</autores>
</libro>
<libro>
  <titulo>Advan Programming for Unix environment</titulo>
  <autores>1</autores>
</libro>
<libro>
  <titulo>Millenium</titulo>
  <autores>1</autores>
</libro>
 ...
  <titulo> Economics of Technology for Digital TV</titulo>
  <autores>0</autores>
</libro>
```

Ejemplo

- Si en la consulta aparece más de una cláusula **for** (o más de una variable en una clásula **for**), el resultado es el producto cartesiano de dichas variables variables.
- Seleccionar los títulos de todos los libros contenidos en el archivo "libros.xml" y todos los comentarios de cada libro contenidos en el archivo "comentarios.xml"

```
for $t in doc("libros.xml")//titulo,
    $e in doc("comentarios.xml")//entrada
where $t = $e/titulo
return
   <comentario>{ $t, $e/comentario }</comentario>
```

- Obtenemos el siguiente resultado:

```
...
```

Ejemplo

- Obtenemos el siguiente resultado:

```
<comentario>
  <titulo>TCP/IP Illustrated</titulo>
  <comentario>Uno de los mejores libros de TCP/IP</comentario>
</comentario>
<comentario>
  <titulo>Data on the Web</titulo>
  <comentario>Un libro muy bueno sobre bases de datos</comentario>
</comentario>
```

Expresiones condicionales

- Además de la cláusula **where**, XQuery también soporta expresiones condicionales del tipo `if-then-else` con la misma semántica que en los lenguajes de programación más habituales (C, Java, Delphi, etc..)
- La cláusula **where** de una consulta permite filtrar las tuplas que aparecerán en el resultado, mientras que una expresión condicional nos permite crear una u otra estructura de nodos en el resultado que dependa de los valores de las tuplas filtradas

Expresiones condicionales: Ejemplo

- Seleccionar los títulos de todos los libros almacenados en el archivo "libros.xml" y sus dos primeros autores.
- En el caso de que existan más de dos autores para un libro, se añade un tercer autor "et al.".

```
for $b in doc("libros.xml")//libro
return
 <libro>
  { $b/titulo }
  {
  for $a at $i in $b/autor
  where $i <= 2
  return <autor>{string($a/apellido), ", ", string($a/nombre)}</aut
  }
  {
  if (count($b/autor) > 2)  then <autor>et al.</autor>
  else ()
  }
 </libro>
```

Ejemplo

- Obtenemos el siguiente resultado:

```
<libro>
    <titulo>TCP/IP Illustrated</titulo>
    <autor>Stevens, W.</autor>
</libro>
<libro>
    ...
<libro>
    <titulo>Millenium</titulo>
    <autor>Falk, Lombardo</autor>
</libro>
<libro>
    <titulo>Data on the Web</titulo>
    <autor>Abiteboul, Serge</autor>
    <autor>Buneman, Peter</autor>
    <autor>et al.</autor>
</libro>
<libro>
<titulo> Economics of Technology for Digital TV</titulo>
```

Cuantificadores existenciales

- XQuery soporta dos cuantificadores existenciales:

  `some` y `every`

- Estos cuantificadores nos permiten definir consultas que devuelvan algún elemento que satisfaga la condición `some` o consultas que devuelvan los elementos en los que todos sus nodos satisfagan la condición `every`.
- Por ejemplo, seleccionar los títulos de los libros en los que al menos uno de sus autores es W. Stevens

```
for $b in doc("libros.xml")//libro
where some $a in $b/autor
      satisfies ($a/apellido="Stevens" and $a/nombre="W.")
return $b/titulo
```

Ejemplo

- Obtenemos el siguiente resultado:

```
<titulo>TCP/IP Illustrated</titulo>
<titulo>Advan Programming for Unix environment</titulo>
```

Cuantificadores existenciales

- Seleccionar todos los títulos de los libros en los que todos los autores de cada libro es W. Stevens.

```
for $b in doc("libros.xml")//libro
where every $a in $b/autor
satisfies ($a/apellido="Stevens" and $a/nombre="W.")
return $b/titulo
```

Ejemplo

- Obtenemos el siguiente resultado:

```
<titulo>TCP/IP Illustrated</titulo>
<titulo>Advan Programming for Unix environment</titulo>
<titulo>Economics of Technology for Digital TV</titulo>
```

- El último libro no tiene autores

Ejemplo

- La siguiente consulta devuelve el título de todos los libros que mencionen 'programing' en cada uno de los párrafos de los libros almacenados en `libros.xml`

```
for $b in doc("libros.xml")//libro
where every $p in $b//titulo
      satisfies contains($p,"Programming")
return $b/titulo
```

- Obtenemos el siguiente resultado:

```
<titulo>Advan Programming for Unix environment</titulo>
```

- XQuery soporta operadores y funciones:
  - matemáticas,
  - de cadenas,
  - para el tratamiento de expresiones regulares,
  - comparaciones de fechas y horas,
  - manipulación de secuencias,
  - manipulación de nodos XML,
  - comprobación y conversión de tipos y lógica booleana
- Además permite definir funciones propias y funciones dependientes del entorno de ejecución del motor XQuery

- XQuery soporta operadores y funciones:
  - **Matemáticas:** $+, -, *, div(*), idiv(*), mod.$
  - **Comparación:** $=, !=, <, >, <=, >=, not()$
  - **Secuencia:** unión (|), intersect, except
  - **Redondeo:** floor(), ceiling(), round().
  - **Funciones de agrupación:** count(), min(), max(), avg(), sum().
  - **Funciones de cadena:** concat(), string-length(), startswith(), ends-with(), substring(), upper-case(), lower-case(), string()
  - **Uso general:** distinct-values(), empty(), exits()
- La división se indica con el operador `div` ya que el símbolo / es necesario para indicar caminos.
- El operador `idiv` es para divisiones con enteros en las que se ignora el resto.

Ejemplo

- Obtener una lista ordenada de apellidos de todos los autores y editores

```
for $l in distinct-values(doc("libros.xml")
                //(autor|editor)/apellido)
order by $l
return <apellidos>{ $l }</apellidos>
```

- Obtenemos el siguiente resultado:

```
<apellidos>Abiteboul</apellidos>
<apellidos>Buneman</apellidos>
<apellidos>Falk</apellidos>
<apellidos>Gerbarg</apellidos>
<apellidos>Stevens</apellidos>
<apellidos>Suciu</apellidos>
```

Ejemplo

- Obtener un nodo libro con todos sus nodos hijos salvo el nodo precio.

```
for $b in doc("libros.xml")//libro
where $b/titulo = "TCP/IP Illustrated"
return
  <libro>
     { $b/@* }
     { $b/* except $b/precio }
  </libro>
```

- Obtenemos el siguiente resultado:

```
<libro año="1994">
   <titulo>TCP/IP Illustrated</titulo>
   <autor>
      <apellido>Stevens</apellido>
      <nombre>W.</nombre>
   </autor>
   <editorial>Addison-Wesley</editorial>
 </libro>
```

Ejemplo

- Obtener los nodos libro que tengan al menos un nodo autor

```
for $b in doc("libros.xml")//libro
where not(empty($b/autor))
return $b
```

- Obtenemos el siguiente resultado:

```
<libro año="1999">
<titulo> Economics of Technology for Digital TV</titulo>
  <editor>
     <apellido>Gerbarg</apellido>
     <nombre>Darcy</nombre>
  <afiliacion>CITI</afiliacion>
  </editor>
  <editorial>Kluwer Academic editorials</editorial>
  <precio>129.95</precio>
</libro>
```

Comentarios

- Los comentarios en XQuery, a diferencia de XML, van encerrados entre caras sonrientes, tal y como se muestra a continuación.

```
(: esto es un comentario :)
```

Aplicaciones de XQuery

- Hemos visto cómo utilizar XQuery para escribir consultas sobre colecciones de datos XML.
- Sin embargo es posible utilizar el lenguaje para otros usos:
  - **Transformación de Jerarquías:** transformar unas estructuras de datos XML en otras estructuras que organicen la información de forma diferente.
  - **XQuery como alternativa a XLST:** permite realizar transformaciones de datos en XML a otro tipo de representaciones, como HTML o PDF

Bibliografía:

- J. J. Gutiérrez, M. J. Escalona, M. Mejías, J. Torres, D.Villadiego. *XQuery*. Universidad de Sevilla. 2005. http://www.lsi.us.es/docs/informes/LSI-2005-02.pdf