Formación XQuery

13/05/2010

FOR-100513-XQuery

Autor

Miguel García (mgarcia@optaresolutions.com)

Leandro Galli (lgalli@optaresolutions.com)

Confidencialidad

El contenido de este documento es propiedad de Optare Solutions, S.L. y debe tratarse con estricta confidencialidad. El uso de la información contenida en este documento debe contar con la autorización expresa de Optare Solutions, S.L.

El receptor de este documento se compromete a no copiar ni divulgar, cualesquiera que sean los medios o técnicas empleados, este documento ni ninguna de sus partes sin el consentimiento previo y explícito de Optare Solutions, S.L.

Tabla de contenido

[1 Historial de versiones 5](#_Toc262716288)

[2 Introducción 6](#_Toc262716289)

[3 XPATH 7](#_Toc262716290)

[4 Requerimientos técnicos de XQuery 8](#_Toc262716291)

[5 Consultas XQuery 9](#_Toc262716292)

[6 Reglas generales 13](#_Toc262716293)

[7 Diferencias entre las cláusulas for y let 14](#_Toc262716294)

[8 Funciones de entrada 17](#_Toc262716295)

[9 Expresiones condicionales 18](#_Toc262716296)

[10 Cuantificadores existenciales 20](#_Toc262716297)

[11 Operadores y funciones principales 22](#_Toc262716298)

[12 Operadores XPath 23](#_Toc262716299)

[13 Definición de funciones en XQuery 28](#_Toc262716300)

[13.1 28](#_Toc262716301)

[13.2 Ejemplo 28](#_Toc262716302)

[14 Comentarios 30](#_Toc262716303)

[15 Ejemplos 31](#_Toc262716304)

[15.1 Clausula for 33](#_Toc262716305)

[15.2 Cláusula let 35](#_Toc262716306)

[15.3 Añadir etiquetas HTML y texto 36](#_Toc262716307)

[15.4 CON SQL: 37](#_Toc262716308)

[16 Referencias 39](#_Toc262716309)

[17 Ejemplos de estilos para copy & paste 40](#_Toc262716310)

[17.1 Titulo 2 40](#_Toc262716311)

[17.1.1 Titulo 3 40](#_Toc262716312)

[17.2 Estilo de Tabla 41](#_Toc262716313)

# Historial de versiones

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev** | **Fecha** | **Autor(es)** | **Comentarios** |
| 0.1 | 12/05/2010 | Miguel García | Versión previa del contenido del documento, incluyendo formación de XQuery |
| 1.0 | 13/05/2010 | Leandro Galli | Separación entre formación XQRepository y formación XQuery |

# Introducción

De manera rápida podemos definir XQuery como un símil en el que XQuery es a XML lo mismo que SQL es a las bases de datos relacionales.

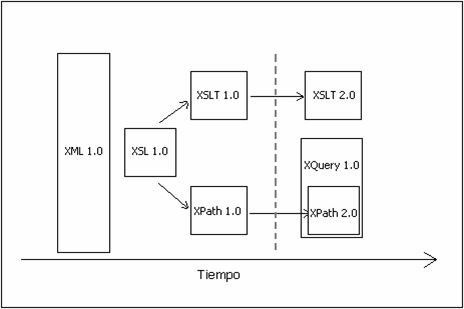
XQuery es un lenguaje de consulta diseñado para escribir consultas sobre colecciones de datos expresadas en XML. Abarca desde archivos XML hasta bases de datos relacionales con funciones de conversión de registros a XML. Su principal función es extraer información de un conjunto de datos organizados como un árbol n-ario de etiquetas XML. En este sentido XQuery es independiente del origen de los datos.

XQuery es un lenguaje funcional, lo que significa que, en vez de ejecutar una lista de comandos como un lenguaje procedimental clásico, cada consulta es una expresión que es evaluada y devuelve un resultado, al igual que en SQL. Diversas expresiones pueden combinarse de una manera muy flexible con otras expresiones para crear nuevas expresiones más complejas y de mayor potencia semántica.

XQuery está llamado a ser el futuro estándar de consultas sobre documentos XML.

# XPATH

XQuery hace un uso intensivo de XPath (un lenguaje utilizado para seleccionar porciones de XML); de hecho algunos ven a XQuery como un superconjunto de XPath. En el gráfico que se muestra en la Ilustración 6 se puede visualizar algunas de las especificaciones del W3C, ubicadas por orden de aparición. XPath en un principio fue parte de XSL 1.0 y luego se desarrolló como una especificación separada. La nueva versión de XPath (XPath 2.0) está siendo desarrollada de manera conjunta a XQuery, por el mismo grupo de trabajo.



Ilustración

# Requerimientos técnicos de XQuery

Requerimientos técnicos para este lenguaje. Los más importantes se detallan a continuación.

* XQuery debe ser un lenguaje declarativo. Al igual que SQL hay que indicar que se quiere, no la manera de obtenerlo.
* XQuery debe ser independiente del protocolo de acceso a la colección de datos. Una consulta en XQuery debe funcionar igual al consultar un archivo local que al consultar un servidor de bases de datos que al consultar un archivo XML en un servidor web.
* Las consultas y los resultados deben respetar el modelo de datos XML.
* Las consultas y los resultados deben ofrecer soporte para los ***namespace***.
* Debe ser capaz de soportar XML-Schemas y DTDs (Document Type Definition) y también debe ser capaz de trabajar sin ninguno de ellos.
* XQuery debe poder trabajar con independencia de la estructura del documento, esto es, sin necesidad de conocerla.
* XQuery debe soportar tipos simples, como enteros y cadenas, y tipos complejos, como un nodo compuesto por varios nodos hijos.
* Las consultan deben soportar cuantificadores universales (para todo) y existenciales (existe).
* Las consultas deben soportar operaciones sobre jerarquías de nodos y secuencias de nodos.
* Debe ser posible en una consulta combinar información de múltiples fuentes.
* Las consultas deben ser capaces de manipular los datos independientemente del origen de estos.
* Mediante XQuery debe ser posible definir consultas que transformen las estructuras de información originales y debe ser posible crear nuevas estructuras de datos.
* El lenguaje de consulta debe ser independiente de la sintaxis, esto es, debe ser posible que existan varias sintaxis distintas para expresar una misma consulta en XQuery.

Aunque XQuery y SQL puedan considerarse similares en casi la totalidad de sus aspectos, el modelo de datos sobre el que se sustenta XQuery es muy distinto del modelo de datos relacional sobre el que sustenta SQL, ya que XML incluye conceptos como jerarquía y orden de los datos que no están presentes en el modelo relacional. Por ejemplo, a diferencia de SQL, en XQuery el orden es 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 (que pueden estar anidadas dentro de una etiqueta <A> o fuera).

# Consultas XQuery

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.

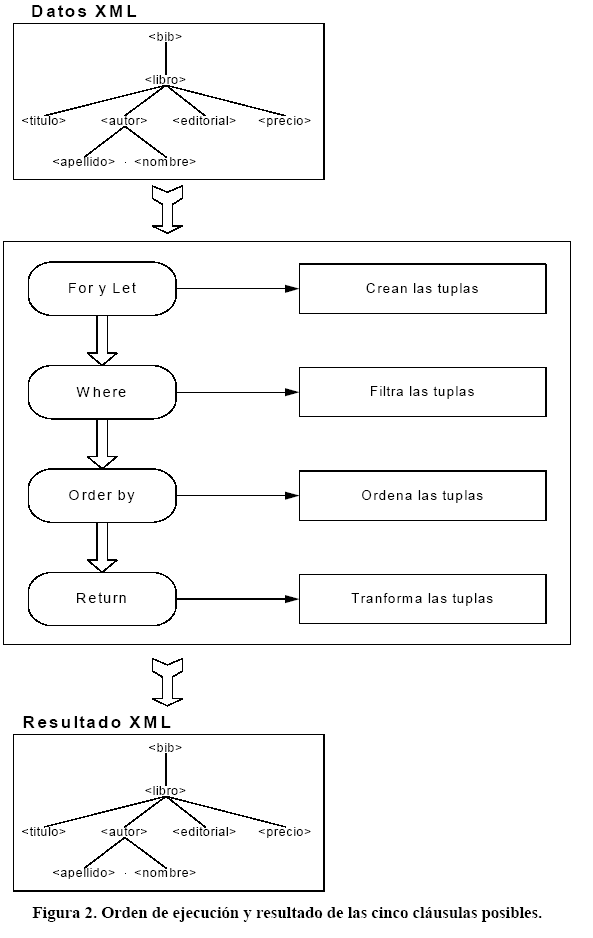
Un detalle importante es que, a diferencia de lo que sucede en SQL, en XQuery las expresiones y los valores que devuelven son dependientes del contexto. Por ejemplo los nodos que aparecerán en el resultado dependen de los namespaces, de la posición donde aparezca la etiqueta raíz del nodo (dentro de 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. A continuación, se describe la función de cada bloque:

* **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.
* **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 resultado de la consulta para una tupla dada, después de haber sido filtrada por la cláusula where y ordenada por la cláusula order by.

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

A continuación, en la ilustración 7, se muestra gráficamente el orden en que se ejecuta cada cláusula de una consulta y los resultados de cada una:



Ilustración

(Según XML de ejemplo, ver apéndice A)

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.

1. for $b in document("libros.xml”)//bib/libro

A continuación se muestra un ejemplo de una consulta donde aparecen las 5 cláusulas. La siguiente consulta devuelve los títulos de los libros que tengan más de dos autores ordenados por su título.

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

El resultado de esta consulta se muestra a continuación.

1. <title>Data on the Web</title>

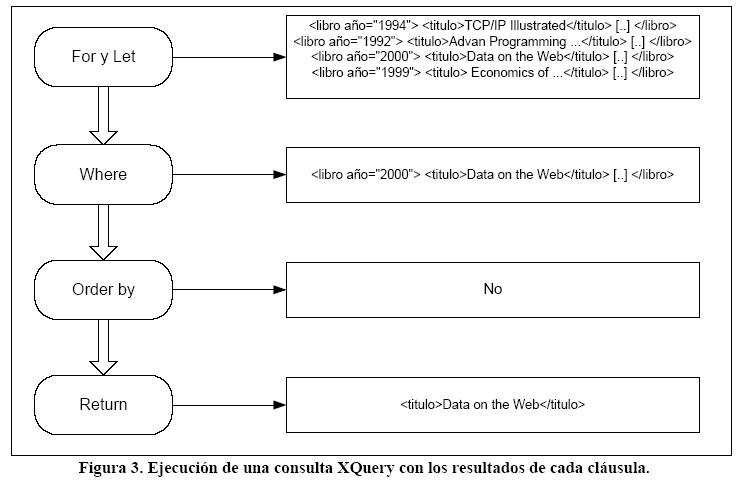
Las barras: “//” no indican comentarios, sino que son parte de la expresión XPath que indica la localización de los valores que tomará la variable $b. En esta consulta la función count() hace la misma función que en SQL, contar el número de elementos, nodos en este caso, referenciados por la variable $c. La diferencia entre la cláusula for y la cláusula let se explica con más detalle en un punto posterior.

Una expresión FLWOR vincula variables a valores con cláusulas for y let y utiliza esos vínculos para crear nuevas estructuras de datos XML.

A continuación se muestra otro ejemplo de consulta XQuery. La siguiente consulta devuelve los títulos de los libros del año 2.000. Como “año” es un atributo y no una etiqueta se le antecede con un carácter “@”.

1. for $b in doc("libros.xml")//libro
2. where $b/@año = "2000"
3. return $b/titulo

A continuación, en la ilustración 8, se muestra de forma gráfica como se ejecutaría la consulta anterior y los resultados parciales de cada una de sus cláusulas:



Ilustración

# Reglas generales

A continuación, enunciamos una serie de reglas que debe cumplir cualquier consulta escrita en XQuery:

* For y let sirven para crear las tuplas con las que trabajará el resto de las cláusulas de la consulta y pueden usarse tantas veces como se desee en una consulta, incluso dentro de otras cláusulas. Sin embargo solo puede declararse una única cláusula where, una única cláusula order by y una única cláusula return.
* Ninguna de las cláusulas FLWOR es obligatoria en una consulta XQuery. Por ejemplo, una expresión XPath, como la que se muestra a continuación, es una consulta válida y no contiene ninguna de las cláusulas FLWOR.

1. doc("libros.xml")/bib/libro/titulo[/bib/libro/autor/apellido='Stevens']

Esta expresión XPath, que también es una consulta XQuery válida, devuelve los títulos de los libros que tengan algún autor de apellido ‘Stevens’.

Es posible especificar varios criterios de ordenación en la cláusula order by, separándolos por comas.

Los criterios de ordenación se aplican por orden de izquierda a derecha.

# Diferencias entre las cláusulas for y let

Para ver claramente la diferencia entre una cláusula for y una cláusula let vamos a comenzar estudiando una misma consulta que muestre los títulos de todos los libros almacenados en el archivo “libros.xml”, primero con una cláusula for y, a continuación, con una cláusula let y vamos a detallar que diferencia hay en la información obtenida.

La consulta con una cláusula for se muestra a continuación.

1. for $d in doc("libros.xml")/bib/libro/titulo
2. return
3. <titulos>{ $d }</titulos>

El resultado de esta consulta se muestra a continuación:

1. <titulos>
2. <titulo>TCP/IP Illustrated</titulo>
3. </titulos>
4. <titulos>
5. <titulo>Advan Programming for Unix environment</titulo>
6. </titulos>
7. <titulos>
8. <titulo>Data on the Web</titulo>
9. </titulos>
10. <titulos>
11. <titulo> Economics of Technology for Digital TV</titulo>
12. </titulos>

A continuación repetimos la misma consulta sustituyendo la cláusula for una cláusula let.

1. let $d := doc("libros.xml")/bib/libro/titulo
2. return
3. <titulos>{ $d }</titulos>

El resultado de esta consulta se muestra a continuación.

1. <titulos>
2. <titulo>TCP/IP Illustrated</titulo>
3. <titulo>Advan Programming for Unix environment</titulo>
4. <titulo>Data on the Web</titulo>
5. <titulo> Economics of Technology for Digital TV</titulo>
6. </titulos>

Como se puede ver comparando los resultados obtenidos por ambas consultas, la cláusula for vincula una variable con cada nodo que encuentre en la colección de datos.

En este ejemplo la variable $d va vinculándose a cada uno de los títulos de todos los libros del archivo "libros.xml", creando una tupla por cada título. Por este motivo aparece repetido el par de etiquetas <titulos>...</titulos> para cada título. La cláusula let, en cambio, vincula una variable con todo el resultado de una expresión. En este ejemplo, la variable $d se vincula a todos los títulos de todos los libros del archivo "libros.xml", creando una única tupla con todos esos títulos. Por este motivo solo aparece el par de etiquetas <titulos>...</titulos> una única vez.

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. Un ejemplo se muestra en la siguiente consulta:

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

Esta consulta devuelve el título de cada uno de los libros de archivo "libros.xml" junto con el número de autores de cada libro. El resultado de esta consulta se muestra a continuación:

1. <libro>
2. <titulo>TCP/IP Illustrated</titulo>
3. <autores>1</autores>
4. </libro>
5. <libro>
6. <titulo>Advanced Programming in the UNIX Environment</titulo>
7. <autores>1</autores>
8. </libro>
9. <libro>
10. <titulo>Data on the Web</titulo>
11. <autores>3</autores>
12. </libro>
13. <libro>
14. <titulo>The Economics of Technology and Content for
15. Digital TV</titulo>
16. <autores>0</autores>
17. </libro>

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, es decir, las tuplas generadas cubren todas las posibles combinaciones de los nodos de dichas variables. Un ejemplo se muestra en la siguiente consulta, la cual devuelve 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”.

1. for $t in doc("books.xml")//titulo,
2. $e in doc("comentarios.xml")//entrada
3. where $t = $e/titulo
4. return <comentario>{ $t, $e/comentario }</comentario>

El resultado de esta consulta se muestra a continuación.

1. <comentario>
2. <titulo>Data on the Web</titulo>
3. <comentario>Un libro muy bueno sobre bases de
4. datos.</comentario>
5. </comentario>
6. <comentario>
7. <titulo>Advanced Programming in the Unix
8. environment</titulo>
9. <comentario>Un libro claro y detallado de programación en
10. UNIX.</comentario>
11. </comentario>
12. <comentario>
13. <titulo>TCP/IP Illustrated</titulo>
14. <comentario>Uno de los mejores libros de
15. TCP/IP.</comentario>
16. </comentario>

# Funciones de entrada

XQuery utiliza las funciones de entrada en las cláusulas for o let o en expresiones XPath para identificar el origen de los datos. Actualmente el borrador de XPath y XQuery define dos funciones de entrada distintas, doc(URI) y collection(URI).

La función doc(URI) devuelve el nodo documento, o nodo raíz, del documento referenciado por un identificador universal de recursos (URI). Esta es la función más habitual para acceder a la información almacenada en archivos.

La función collection(URI) devuelve una secuencia de nodos referenciados por una URI, sin necesidad de que exista un nodo documento o nodo raíz. Esta es la función más habitual para acceder a la información almacenada en una base de datos que tenga capacidad para crear estructuras de datos XML.

# 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..). Por ejemplo, la siguiente consulta devuelve 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.".

1. for $b in doc("libros.xml")//libro
2. return
3. <libro>
4. { $b/titulo }
5. {
6. for $a at $i in $b/autor
7. where $i <= 2
8. return <autor>{string($a/last), ", ",
9. string($a/first)}</autor>
10. }
11. {
12. if (count($b/autor) > 2)
13. then <autor>et al.</autor>
14. else ()
15. }
16. </libro>

El resultado de esta consulta se muestra a continuación.

1. <libro>
2. <titulo>TCP/IP Illustrated</titulo>
3. <autor>Stevens, W.</autor>
4. </libro>
5. <libro>
6. <titulo>Advanced Programming in the Unix
7. Environment</titulo>
8. <autor>Stevens, W.</autor>
9. </libro>
10. <libro>
11. <titulo>Data on the Web</titulo>
12. <autor>Abiteboul, Serge</autor>
13. <autor>Buneman, Peter</autor>
14. <autor>et al.</autor>
15. </libro>

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.

A diferencia de la mayoría de los lenguajes, *la cláusula* ***else*** *es obligatoria y debe aparecer siempre en la expresión condicional*. El motivo de esto es que toda expresión en XQuery debe devolver un valor. Si no existe ningún valor a devolver al no cumplirse la cláusula if, devolvemos una secuencia vacía con ‘else ()’, tal y como se muestra en el ejemplo anterior.

# Cuantificadores existenciales

XQuery soporta dos cuantificadores existenciales llamados “some” y “every”, de tal manera que nos permite definir consultas que devuelva 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, la siguiente consulta devuelve 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/last="Stevens" and $a/first="W.")  return $b/titulo |

El resultado de esta consulta se muestra a continuación

|  |
| --- |
| <title>TCP/IP Illustrated</title>  <title>Advanced Programming in the Unix Environment</title> |

La siguiente consulta devuelve 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/last="Stevens" and $a/first="W.")  return $b/titulo |

El resultado de esta consulta se muestra en el siguiente párrafo.

|  |
| --- |
| <titulo>TCP/IP Illustrated</titulo>  <titulo>Advanced Programming in the Unix Environment</titulo>  <titulo>The Economics of Technology and Content for Digital TV</titulo> |

El último título devuelto como resultado de la consulta es un libro que no tiene autores. Cuando un cuantificador universal se aplica sobre un nodo vacío, siempre devuelve cierto, como se ve en el ejemplo anterior.

Otro ejemplo. La siguiente consulta devuelve los títulos de los libros que mencionen “Unix” y “programing” en el mismo párrafo. Si el libro tiene más de un párrafo solo es necesario que aparezca en, al menos, uno de ellos. Esto lo indicamos con la palabra reservada “some” justo a continuación de where.

|  |
| --- |
| for $b in doc("bib.xml")//libro  where some $p in $b//parrafo satisfies  (contains($p,"Unix") AND contains($p,"programing"))  return $b/title |

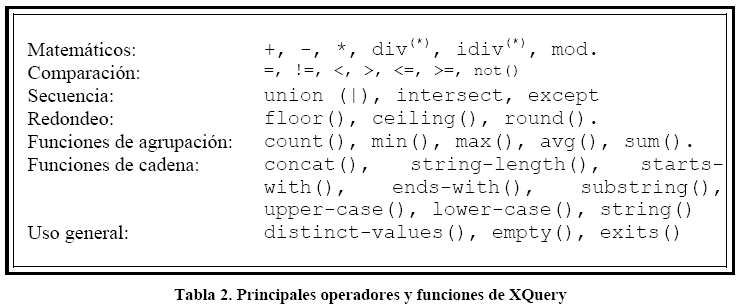
Otro ejemplo. La siguiente consulta devuelve el título de todos los libros que mencionen “programing” en cada uno de los párrafos de los libros almacenados en “bib.xml”.

|  |
| --- |
| for $b in doc("bib.xml")//libro  where every $p in $b// parrafo satisfies  contains($p,"programing")  return $b/title |

Esta consulta es distinta de la anterior ya que no es suficiente que “programing” aparezca en al menos uno de los párrafos, sino que debe aparecer en todos los párrafos que existan. Esto lo indicamos con la palabra reservada “every” justo a continuación de “where”.

# Operadores y funciones principales

XQuery soporta operadores y funciones matemáticas, de cadenas, para el tratamiento de expresiones regulares, comparaciones de fechas y horas, manipulación de nodos XML, manipulación de secuencias, comprobación y conversión de tipos y lógica booleana. Además permite definir funciones propias y funciones dependientes del entorno de ejecución del motor XQuery. Los operadores y funciones más importantes se muestran en la tabla 2 y se detallan a lo largo de este apartado.



Ilustración

# Operadores XPath

Lista de operadores que se pueden usar en expresiones XPath:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Return value** |
| | | Computes two node-sets | //book | //cd | Returns a node-set with all book and cd elements |
| + | Addition | 6 + 4 | 10 |
| - | Subtraction | 6 - 4 | 2 |
| \* | Multiplication | 6 \* 4 | 24 |
| div | Division | 8 div 4 | 2 |
| = | Equal | price=9.80 | true if price is 9.80 false if price is 9.90 |
| != | Not equal | price!=9.80 | true if price is 9.90 false if price is 9.80 |
| < | Less than | price<9.80 | true if price is 9.00 false if price is 9.80 |
| <= | Less than or equal to | price<=9.80 | true if price is 9.00 false if price is 9.90 |
| > | Greater than | price>9.80 | true if price is 9.90 false if price is 9.80 |
| >= | Greater than or equal to | price>=9.80 | true if price is 9.90 false if price is 9.70 |
| or | or | price=9.80 or price=9.70 | true if price is 9.80 false if price is 9.50 |
| and | and | price>9.00 and price<9.90 | true if price is 9.80 false if price is 8.50 |
| mod | Modulus (division remainder) | 5 mod 2 | 1 |

**(**[**http://www.w3schools.com/xpath/xpath\_functions.asp**](http://www.w3schools.com/xpath/xpath_functions.asp)**)**

(\*) 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.

El resultado de un operador aritmético en el que uno, o ambos, de los operandos sea una cadena vacía es una cadena vacía. Como regla general el funcionamiento de las cadenas vacías en XQuery es análogo al funcionamiento de los valores nulos en SQL.

El operador unión recibe dos secuencias de nodos y devuelve una secuencia con todos los nodos existentes en las dos secuencias originales. A continuación se muestra una consulta que usa el operador unión para 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> |

El resultado de esta consulta se muestra en el siguiente párrafo:

|  |
| --- |
| <apellidos>Abiteboul</apellidos>  <apellidos>Buneman</apellidos>  <apellidos>Gerbarg</apellidos>  <apellidos>Stevens</apellidos>  <apellidos>Suciu</apellidos> |

El operador de intersección recibe dos secuencias de nodos como operandos y devuelve una secuencia conteniendo todos los nodos que aparezcan en ambos operandos.

El operador de sustracción (except) recibe dos secuencias de nodos como operandos y devuelve una secuencia conteniendo todos los nodos del primer operando que no aparezcan en el segundo operando. A continuación se muestra una consulta que usa el operador sustracción para 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> |

El resultado de esta consulta se muestra a continuación.

|  |
| --- |
| <libro year = "1994">  <title>TCP/IP Illustrated</title>  <author>  <apellidos>Stevens</apellidos>  <first>W.</first>  </author>  <publisher>Addison-Wesley</publisher>  </libro> |

La función distinct-values() extrae los valores de una secuencia de nodos y crea una nueva secuencia con valores únicos, eliminando los nodos duplicados. Por ejemplo, la siguiente consulta devuelve todos los apellidos distintos de los autores.

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

El resultado de esta consulta se muestra en el siguiente párrafo.

|  |
| --- |
| <apellidos>Stevens</apellidos>  <apellidos>Abiteboul</apellidos>  <apellidos>Buneman</apellidos>  <apellidos>Suciu</apellidos> |

La función empty() devuelve cierto cuando la expresión entre paréntesis está vacía. Por ejemplo, la siguiente consulta devuelve todos los nodos libro que tengan al menos un nodo autor.

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

El resultado de esta consulta se muestra a continuación.

|  |
| --- |
| <libro año="1994">  <titulo>TCP/IP Illustrated</titulo>  <autor>  <apellido>Stevens</apellido>  <nombre>W.</nombre>  </autor>  <editorial>Addison-Wesley</editorial>  <precio> 65.95</precio>  </libro>  <libro año="1992">  <titulo>Advan Programming for Unix environment</titulo>  <autor>  <apellido>Stevens</apellido>  <nombre>W.</nombre>  </autor>  <editorial>Addison-Wesley</editorial>  <precio>65.95</precio>  </libro>  <libro año="2000">  <titulo>Data on the Web</titulo>  <autor>  <apellido>Abiteboul</apellido>  <nombre>Serge</nombre>  </autor>  <autor>  <apellido>Buneman</apellido>  <nombre>Peter</nombre>  </autor>  <autor>  <apellido>Suciu</apellido>  <nombre>Dan</nombre>  </autor>  <editorial>Morgan Kaufmann editorials</editorial>  <precio>39.95</precio>  </libro> |

La función opuesta a empty() es exists(), la cual devuelve cierto cuando una secuencia contiene, al menos, un elemento. Por ejemplo, como la consulta anterior tiene una cláusula where que comprueba una negación sobre empty(), podemos rescribirla usando la función exists() tal y como se muestra a continuación:

|  |
| --- |
| for $b in doc("libros.xml")//libro  where exists($b/autor)  return $b |

El resultado de esta consulta es el mismo que el resultado de la consulta anterior.

# Definición de funciones en XQuery

**(**[**http://www.w3schools.com/xpath/xpath\_functions.asp**](http://www.w3schools.com/xpath/xpath_functions.asp)**)**

Si no puedes encontrar la función que necesitas, puedes escribirla tu mismo. Se pueden definir funciones dentro de la query o en una librería separada.

Pasos:

1. Usar la palabra clave **declare function**, al inicio de la función.
2. El nombre de la function debe tener un prefijo (ej: local:nombre)
3. El tipo de dato de los parametros muchas veces son del mismo tipo que los definidos en XML Schema
4. El cuerpo de la función debe ir dentro de los {}.

|  |
| --- |
| declare function *prefix:function\_name*($*parameter* AS *datatype*)  AS *returnDatatype*  {  (: ...function code here... :)  }; |

## Ejemplo

|  |
| --- |
| declare function local:minPrice(  $price as xs:decimal?,  $discount as xs:decimal?)  AS xs:decimal?  {  let $disc := ($price \* $discount) div 100  return ($price - $disc)  };  (: Below is an example of how to call the function above :)  <minPrice>{local:minPrice($book/price, $book/discount)}</minPrice> |

# 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, y parece muy feliz :) |

# Ejemplos

**LIBROS.XML:**

|  |
| --- |
| <?xml version="1.0" encoding="ISO-8859-1"?>  <bib>  <libro año="1994">  <titulo>TCP/IP Illustrated</titulo>  <autor>  <apellido>Stevens</apellido>  <nombre>W.</nombre>  </autor>  <editorial>Addison-Wesley</editorial>  <precio> 65.95</precio>  </libro>  <libro año="1992">  <titulo>Advan Programming for Unix environment</titulo>  <autor>  <apellido>Stevens</apellido>  <nombre>W.</nombre>  </autor>  <editorial>Addison-Wesley</editorial>  <precio>65.95</precio>  </libro>  <libro año="2000">  <titulo>Data on the Web</titulo>  <autor>  <apellido>Abiteboul</apellido>  <nombre>Serge</nombre>  </autor>  <autor>  <apellido>Buneman</apellido>  <nombre>Peter</nombre>  </autor>  <autor>  <apellido>Suciu</apellido>  <nombre>Dan</nombre>  </autor>  <editorial>Morgan Kaufmann editorials</editorial>  <precio>39.95</precio>  </libro>  <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>  </bib> |

**COMENTARIOS.XML**

|  |
| --- |
| <?xml version="1.0" encoding="ISO-8859-1"?>  <comentarios>  <entrada>  <titulo>Data on the Web</titulo>  <precio>34.95</precio>  <comentario>  Un libro muy bueno sobre bases de datos.  </comentario>  </entrada>  <entrada>  <titulo>Advanced Programming in the Unix environment</titulo>  <precio>65.95</precio>  <comentario>  Un libro claro y detallado de programación en UNIX.  </comentario>  </entrada>  <entrada>  <titulo>TCP/IP Illustrated</titulo>  <precio>65.95</precio>  <comentario>  Uno de los mejores libros de TCP/IP  </comentario>  </entrada>  </comentarios> |

## Clausula for

La cláusula **for** une una variable para cada elemento devuelto por la expresión. Para especificar un número determinado de veces en una cláusula **for**, puede usar la palabra clave **to**:

|  |
| --- |
| for $x in (1 to 5)  return <test>{$x}</test> |

Resultado:

|  |
| --- |
| <test>1</test>  <test>2</test>  <test>3</test>  <test>4</test>  <test>5</test> |

La palabra clave **at** puede usarse para contar el número de iteraciones:

|  |
| --- |
| for $x at $i in doc("books.xml")/bookstore/book/title  return <book>{$i}. {data($x)}</book> |

Resultado:

|  |
| --- |
| <book>1. Everyday Italian</book>  <book>2. Harry Potter</book>  <book>3. XQuery Kick Start</book>  <book>4. Learning XML</book> |

Puede existir más de una expression dentro de la misma claúsula **for** separadas por comas:

|  |
| --- |
| for $x in (10,20), $y in (100,200)  return <test>x={$x} and y={$y}</test> |

Resultado:

|  |
| --- |
| <test>x=10 and y=100</test>  <test>x=10 and y=200</test>  <test>x=20 and y=100</test>  <test>x=20 and y=200</test> |

## Cláusula let

La cláusula **let** permite asignar una variable N veces a una misma expresión.

|  |
| --- |
| let $x := (1 to 5)  return <test>{$x}</test> |

Resultado:

|  |
| --- |
| <test>1 2 3 4 5</test> |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | /ProductDescription/Summary | Selecciona todos los elementos **<Summary>** que son hijos del elemento **<ProductDescription>**, que es el elemento raíz del documento. | | //Summary | Selecciona todos los elementos **<Summary>** que se encuentran dentro del documento. La doble barra indica profundidad arbitraria. | | count(//Summary) | Retorna el número de elementos **<Summary>** que aparecen en el documento. | | //Picture[Size = "small"] | Retorna todos los elementos **<Picture>**, de profundidad arbitraria, que tienen un hijo cuyo valor es "**small**". | | //ProductDescription[@ProductModelID=19] | Retorna todos los elementos que contienen un atributo **ProductModelID** y su valor es **19**. El símbolo **@** indica que **ProductModelID** es un atributo. Verás estos atributos en la primera línea del código XML que se lista arriba. | | //ProductDescription[@ProductModelID] | Retorna todos los elementos que contienen un atributo, independientemente del valor que contengan. | | //ProductDescription/@ProductModelID | Retorna los valores del atributo **ProductModelID**. | | //Size[1] | Retorna el primer nodo **<Size>** que encuentra. | |

Ahora queremos eliminar el elemento title y mostrar solo el dato dentro de él.

|  |
| --- |
| <ul>  {  for $x in doc("books.xml")/bookstore/book/title  order by $x  return <li>{data($x)}</li>  }  </ul> |

Resultado:

|  |
| --- |
| <ul>  <li>Everyday Italian</li>  <li>Harry Potter</li>  <li>Learning XML</li>  <li>XQuery Kick Start</li>  </ul> |

## Añadir etiquetas HTML y texto

Ahora, queremos añadir algunas etiquetas HTML en el resultado. Mostraremos el resultado en una lista HTML:

|  |
| --- |
| <html>  <body>  <h1>Bookstore</h1>  <ul>  {  for $x in doc("books.xml")/bookstore/book  order by $x/title  return <li>{data($x/title)}. Category: {data($x/@category)}</li>  }  </ul>  </body>  </html> |

Resultado:

|  |
| --- |
| <html>  <body>  <h1>Bookstore</h1>  <ul>  <li>Everyday Italian. Category: COOKING</li>  <li>Harry Potter. Category: CHILDREN</li>  <li>Learning XML. Category: WEB</li>  <li>XQuery Kick Start. Category: WEB</li>  </ul>  </body>  </html> |

## CON SQL:

Otra forma de realizar consultas XQuery al motor es incrustándolas dentro de un SQL-Select. Veamos la siguiente consulta:

1. Seleccionar la fecha de modificación (**ModifiedDate**, de tipo **DateTime**) y los elementos "*Step*" (pasos) que se encuentran dentro de las instrucciones (*Instructions*, de tipo XML).
2. Consulta:

|  |
| --- |
| SELECT ModifiedDate, Instructions::query('  namespace AWMI="http://schemas.adventure-works.com/ManufInstructions/ByProdModel"  for $Step in //AWMI:WorkCenter[1]/AWMI:step  return  string($Step)') as Result FROM ProductPlan Where ProductModelID=7 |

SELECT ModifiedDate, Instructions::query('  
 namespace AWMI="http://schemas.adventure-works.com/ManufInstructions/ByProdModel"  
 for $Step in //AWMI:WorkCenter[1]/AWMI:step  
 return  
 string($Step)') as Result  
FROM ProductPlan  
Where ProductModelID=7

# Referencias

<http://es.wikipedia.org/wiki/XQuery>

<http://www.w3schools.com/xquery/default.asp>

Formación interna OPTARE I-FOR-070419 - XQuery

# Ejemplos de estilos para copy & paste

Texto del título 1

## Titulo 2

Texto del título 2

Si hay que incluir código en el documento usar el formato “sourceCode” incluido en la lista de estilos rápidos:

1. var snow=new Array()
2. var marginbottom
3. var marginright
4. var timer
5. var i\_snow=0
6. var x\_mv=new Array();
7. var crds=new Array();
8. var lftrght=new Array();
9. var browserinfos=navigator.userAgent
10. var ns6=document.getElementById&&!document.all
11. var opera=browserinfos.match(/Opera/)
12. var browserok=ie5||ns6||opera
13. function randommaker(range) {
14. rand=Math.floor(range\*Math.random())
15. return rand
16. }

Lo mejor para usarlo es pegar el código en texto sin formato, seleccionarlo y aplicar el estilo.

### Titulo 3

Texto del título 3:

* Punto 1
* Punto 2
* Punto 3

#### Titulo 4

Texto del título 4

##### Título 5

Texto del título 5

## Estilo de Tabla

|  |  |  |
| --- | --- | --- |
| Encabezado | Encabezado | Encabezado |
| Contenido | Contenido | Contenido |
| Contenido | Contenido | Contenido |
| Contenido | Contenido | Contenido |
| Contenido | Contenido | Contenido |

|  |  |  |
| --- | --- | --- |
| Encabezado | Contenido | Contenido |
| Encabezado | Contenido | Contenido |
| Encabezado | Contenido | Contenido |
| Encabezado | Contenido | Contenido |
| Encabezado | Contenido | Contenido |