



TUTORIALES XQuery

e-business designer
Versión 3.3.x

Headquarters

C/Doctor Trueta 50-54, 3º
08005 Barcelona, España
Tel: +34 932 251 070
Fax: +34 932 217 303

Orense 68, 11º
28020 Madrid, España
Tel: +34 917 885 747
Fax: +34 917 885 701

Pza. Alfonso el Magnánimo 1
46003 Valencia, España
Tel: +34 963 534 505
Fax: +34 963 944 127

1401 Brickell Av. # 320
33131 Miami, FL, USA
Tel: +1 305 503 3380
Fax: +1 305 503 3381

Mariano Escobedo, 543
Despacho 301, Rincon del Bosque
CP 11580 México DF
Tel: +52 55 5545 2646

Introducción

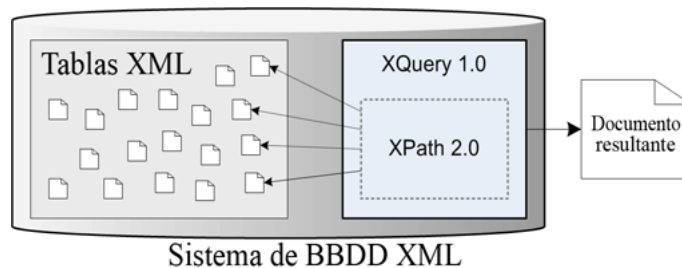
1.1. ¿Qué es XQuery?

XQuery, también conocido como XML Query, es un lenguaje creado para buscar y extraer elementos y atributos de documentos XML. La mejor forma de entender este lenguaje es diciendo que XQuery es para XML lo que SQL es para las bases de datos. 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.

Una consulta XQuery podría resolver, por ejemplo, la siguiente pregunta: "Seleccionar todos los libros con un precio menor a 20 euros de la colección de libros almacenada en un documento XML llamado catalogo.xml".

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 pretende ser el futuro estándar de consultas sobre documentos XML. Actualmente XQuery es un conjunto de borradores en el que trabaja el grupo W3C, por lo que es compatible con XML, Namespaces, XSLT, XPath y XML Schema. Sin embargo, a pesar de no tener una redacción definitiva ya existen o están en proceso numerosas implementaciones de motores y herramientas que lo soportan.

XQuery hace uso de XPath, un lenguaje utilizado para seleccionar partes de XML. De hecho, XQuery 1.0 y XPath 2.0 comparten el mismo modelo de datos y soportan las mismas funciones y operadores.

XQuery en eBD

La sintaxis en eBD es la siguiente:

```
<eBD:XQUERY TABLE="tablename" [NAME="queryname"] [VAR="varname"] [STATIC="1"]>
XQuery
</eBD:XQUERY>
```

De esta manera conseguimos ejecutar una consulta XQuery contra una Tabla vinculada usando el formato XML.

El parámetro **NAME** definirá el nombre con el que se guardará el HANDLER para poder hacer FETCHROW y poder recorrer el resultado de la expresión como una estructura de registros y campos (array de hashes).

El parámetro **VAR** especifica una variable escalar en la que se almacenará el contenido XML de la respuesta de la query, para poder parsear ese XML manualmente con XMLPARSE.

```
<eBD:SET VAR="doc" VALUE="#[eBDDDB.xml_document(nombre_tabla)]"/>
<eBD:XQUERY TABLE="slashdot" NAME="datos" VAR="xml">
FOR \$a IN doc("\$doc")/rss/channel/item
RETURN
<dato>
<nombre>{data(\$a/title)}</nombre>
<enlace>{data(\$a/link)}</enlace>
</dato>
</eBD:XQUERY>
```

Si se especifica el parámetro **STATIC**, eBD no interpretará variables ni expresiones dentro de la XQuery, por lo que se podrán usar los caracteres especiales (\$,%,etc...) sin necesidad de escaparlos.

En XQuery podemos usar <eBD:SET> y de la misma manera que si de una Query standard se tratara.

Consultas en XQuery

3.1. Reglas generales

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

FOR y LET sirven para crear tuplas con las que trabajará el resto de las cláusulas de la consulta, y pueden usarse tantas veces como se desee en una misma consulta, incluso dentro de otras cláusulas. Sin embargo sólo puede declararse una única cláusula WHERE, una única cláusula ORDER y una única cláusula RETURN.

En eBDML hemos de escapar la variable de XQuery, para que eBD no crea que es una variable suya y nos muestre el error:

	TIME	TYPE	MESSAGE
	Thu Jul 3 09:13:34 2008	[ERROR]	line 4: error 205: 'b' Variable does not exists.
	Thu Jul 3 09:13:34 2008	[ERROR]	DBXML error: net.sf.saxon.trans.StaticError: XQuery syntax error in #...sultset>{ for in doc#: expected ")", found "in"

por tanto, la sintaxis correcta será la siguiente:

```
<eBD:SET VAR="doc" VALUE="#[eBDDb.xml_document(Tabla_xml)]"/>
<eBD:XQUERY TABLE="Tabla_xml" NAME="nom" VAR="xml">
FOR \ $a IN doc(' $doc')/*:documento/*:reg
RETURN
<reg>
<iden >{data(\ $a/*:id)}</iden>
< nombre >{data(\ $a/*:nombre)}</ nombre >
</reg>
</eBD:XQUERY>
```

The screenshot displays the eBDML interface. On the left, an XML tree for 'Sample.xml' is shown, containing a root element '<documento>' with five child elements '<reg>'. Each '<reg>' element contains two sub-elements: '<id>' and '<nombre>'. The values for '<id>' are 1, 2, 3, 4, and 5, and the values for '<nombre>' are 'adasda', 'bb', 'cc', 'add', and 'ee'. On the right, the 'Registros' (Records) section shows a table with one record: 'reg'. Below this, the 'Campos Claves' (Key Fields) section shows a table with two columns: 'id' and 'iden'. The 'id' column has a value of 'id', and the 'iden' column has a value of 'iden'. At the bottom, the 'Campos' (Fields) section shows a table with two columns: 'id' and 'nombre'. The 'id' column has a value of 'id', and the 'nombre' column has a value of 'nombre'.

Ninguna de las cláusulas FLWOR (For, Let, Where, Order by, Return) es obligatoria en una consulta XQuery. Por ejemplo, en una expresión XPath como la que se muestra a continuación, es una consulta válida y no contiene ninguna de las cláusulas FLWOR:

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

3.2. 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.

XML que utilizaremos para los ejemplos de XQuery

```
<pelicula fechaEstreno="19/12/2007">
  <titulo>Los Otros </titulo>
  <director>Alejandro Amenabar</director>
</pelicula>
<pelicula fechaEstreno="15/11/1986">
  <titulo>Sangre Facil </titulo>
  <director>Ethan Coen </director>
</pelicula>
<pelicula fechaEstreno="14/06/1990">
  <titulo>No es país para pobres </titulo>
  <director>Ethan Coen </director>
  <director>Joel Coen </director>
</pelicula>
<pelicula fechaEstreno="19/12/2007">
  <titulo> The Lovely Bones </titulo>
  <director> Peter Jackson</director>
</pelicula>
```

3.3. FOR y LET

Usando **FOR**, cada nodo recuperado por la expresión, es vinculado a una única tupla. Por ejemplo,

```
FOR \$b IN doc('informacion.xml')//película/director
RETURN
<directores>\$b</directores>
```

Recuperaríamos todos los nodos <director> de todas las películas del documento:

```
<directores>
  <director>Peter Jackson</director>
</directores>
<directores>
  <director>Alejandro Amenabar</director>
</directores>
<directores>
  <director>Ethan Coen</director>
</directores>
<directores>
  <director>Joel Coen</director>
</directores>
```

Usando **LET** todos los nodos recuperados estarán en una única tupla. Por ejemplo,

```
LET \$b := doc('informacion.xml')//película/director
RETURN
<directores>\$b</directores>
```

Obtendríamos lo siguiente:

```
<directores>
  <director>Peter Jackson</director>
  <director>Alejandro Amenabar</director>
  <director>Ethan Coen</director>
  <director>Joel Coen</director>
</directores>
```

3.4. WHERE

Filtra las tuplas producidas por las cláusulas LET y FOR, recuperando únicamente aquellas que cumplan una expresión. Dicha expresión es evaluada para cada tupla. Si su evaluación es falsa, esa tupla será descartada.

Ejemplo:

```
FOR \$b IN doc('informacion.xml')//pelicula/director
WHERE \$b/..@fechaEstreno="19/12/2007"
RETURN
<directores>\$b</directores>
```

Recuperaría únicamente los directores que hubiesen estrenado película el día 19-12-2007 (suponemos que fechaEstreno es un atributo, por eso ponemos @ delante; el ../ es para subir un nivel en el árbol XML donde está la información):

```
<directores>
  <director>Peter Jackson</director>
  <director>Alejandro Amenabar</director>
</directores>
```

3.5. ORDER

Indica, para un conjunto de información, un orden de salida. Por defecto organiza de manera ascendente según el tipo de datos que se esté evaluando.

Ejemplo:

```
FOR \$b IN doc('informacion.xml')//pelicula/director
ORDER BY \$b/..titulo
RETURN
<directores>\$b</directores>
```

Recuperaría los directores ordenados por el título de sus películas:

```
<directores>
  <director>Alejandro Amenabar</director>
</directores>
<directores>
  <director>Ethan Coen</director>
</directores>
<directores>
  <director>Joel Coen</director>
</directores>
<directores>
  <director>Peter Jackson</director>
```

```
</directores>
```

3.6. RETURN

Formatea la información de salida. Esto nos ofrece muchas posibilidades, ya que podemos convertir la información recuperada a HTML, PDF o cualquier otro tipo.

En los anteriores ejemplos hemos estado utilizando la cláusula RETURN para que nos devolviera los resultados entre los tags <directores>.

3.7. Expresiones condicionales

IF-THEN-ELSE

XQuery también admite expresiones condicionales IF-THEN-ELSE. Esta cláusula es útil para dar un formato diferente a la salida dependiendo de la información de esta, es decir, podemos cambiar la estructura de los nodos en los que recuperamos la información según nos convenga por su contenido.

Si se usa IF, es obligatorio el ELSE, pero puede ser únicamente ELSE ().

Ejemplo:

```
FOR \$b IN doc('informacion.xml')//pelicula/director
RETURN
IF (\$b = "Alenjandro Amenabar") THEN
<espanol>{ \$b }</espanol>
ELSE
<directores>{ \$b }</directores>
```

3.8. Cuantificadores existenciales

EVERY y SOME

XQuery dispone de dos cuantificadores existenciales.

EVERY: recupera aquellas tuplas en las que todos los nodos cumplan la condición.

SOME: recupera aquellas tuplas en las que algún nodo cumpla la condición.

Ejemplo:

```
LET \$b := doc('informacion.xml')//pelicula/titulo
WHERE SOME \$a in \$b../director
SATISFIES (\$a = "Ethan Coen")
RETURN
<titulos>\$b</titulos>
```

Recuperaría los títulos de las películas en las que algún director fuera "Ethan Coen".

```
<titulos>
```



```
<titulo>Sangre fácil</titulo>
<titulo>No es país para pobres</titulo>
</titulos>
```

Igual pero con EVERY:

```
LET $b := doc('informacion.xml')//pelicula/titulo
WHERE EVERY $b in $b/../director
SATISFIES ($b = "Ethan Coen")
RETURN
<titulos>$b</titulos>
```

Recuperaría los títulos de las películas en las que todos los directores (es este caso el único) fueran "Ethan Coen".

```
<titulos>
<titulo>Sangre fácil</titulo>
</titulos>
```

3.9. Operadores y funciones matemáticas

Al igual que otros lenguajes, XQuery tiene una serie de funciones y operadores aritméticos para la recuperación de información

Matemáticos	+, -, *, div(*), idiv(*), mod
Comparación	=, !=, <, >, <=, >=, not()
Secuencia	union (), 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(), exists()

3.10. Comentarios

Los comentarios en XQuery, a diferencia de XML, van encerrados entre caras sonrientes " (: :) ".

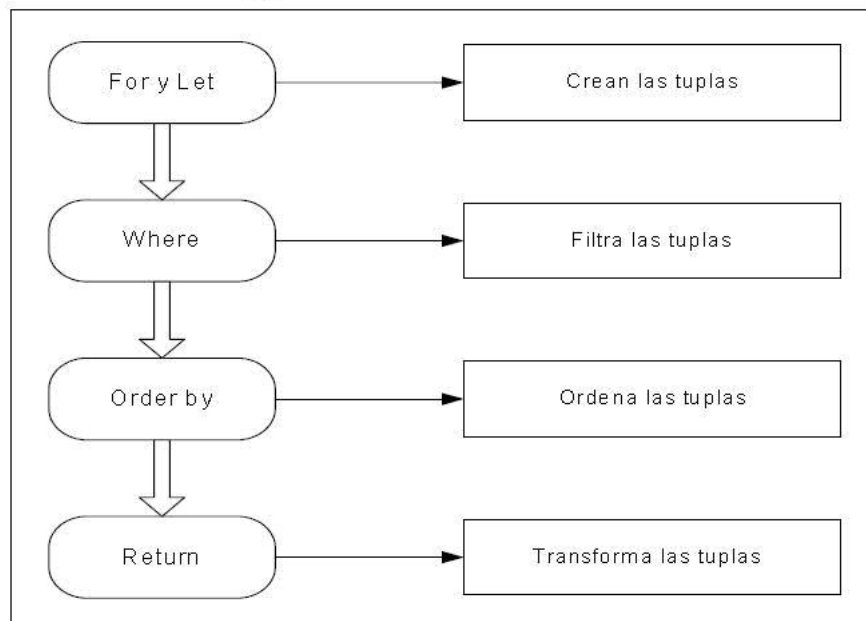
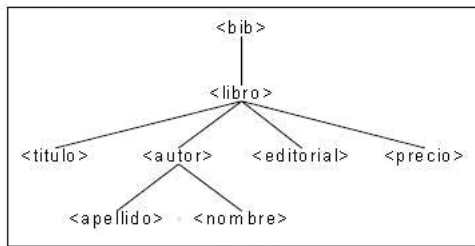
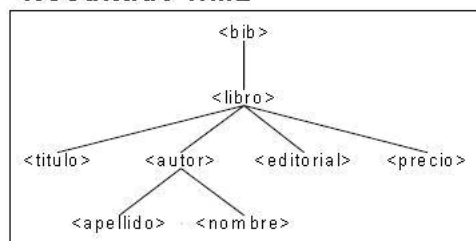
```
<eBD:XQUERY TABLE="Tabla_xml" NAME="nom" VAR="xml">
(: Soy un comentario :)
FOR \$b IN distinct-values(doc('informacion.xml')//pelicula/titulo
Order by \$b
RETURN <titulos>{ \$b }</titulos>
</eBD:XQUERY>
```

La función distinct-values() extrae los valores de una secuencia de nodos y crea una nueva secuencia con valores únicos, eliminando los nodos duplicados.

3.11. Orden de ejecución

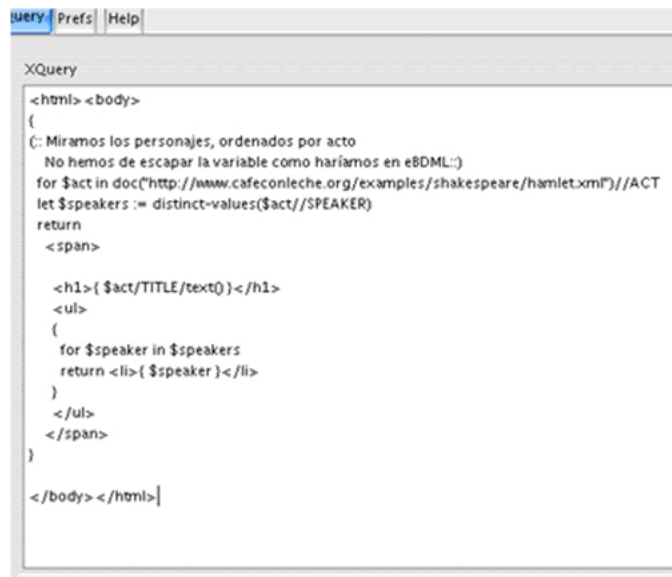
En XQuery, cuando usamos el término tupla, nos estamos refiriendo a cada uno de los valores que toma una variable.

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

Datos XML**Resultado XML**

Utilidades

Podemos usar una extensión para Mozilla Firefox llamada XQuery USE ME (<https://addons.mozilla.org/es-ES/firefox/addon/5515>) para comprobar sentencias XQuery.



```
<?xml version='1.0'>
<html> <body>
(
  (: Miramos los personajes, ordenados por acto
  No hemos de escapar la variable como haríamos en eBDML: :)
  for $act in doc("http://www.cafeconleche.org/examples/shakespeare/hamlet.xml")//ACT
  let $speakers := distinct-values($act//SPEAKER)
  return
  <span>

    <h1>{ $act/TITLE/text() }</h1>
    <ul>
    (
      for $speaker in $speakers
      return <li>{ $speaker }</li>
    )
    </ul>
  </span>
)
</body> </html>
```

Hemos de tener presente que la sintaxis, en este caso, ha de cumplir el estándar del lenguaje, por lo que no deberemos escapar las variables, ni usar el tag <eBD:XQUERY>

Apéndice I. XPath.

XPath es un lenguaje de expresión utilizado para referenciar nodos de información en un conjunto de datos XML. XQuery utiliza este lenguaje para las cláusulas FOR y LET de una consulta.

Las expresiones XPath tienen un comportamiento similar a las expresiones regulares aplicadas al contexto de un conjunto de datos en XML y, más concretamente, al un conjunto de nodos XML en lugar de un conjunto de caracteres.

XPath es muy fácil de entender mediante ejemplos. A continuación mostramos algunas expresiones básicas y lo que se obtiene a partir de ellas:

```
/html/body/h1
```

Selecciona todos los nodos <h1> que son hijos de un nodo <body> que, a su vez, es hijo de un nodo <html> (que es el nodo raíz del árbol XML). El resultado será todos los nodos <h1> que cumplan la anterior condición.

```
//h1
```

Selecciona todos los nodos <h1> que aparezcan en cualquier posición del árbol XML. La doble barra indica cualquier profundidad.

```
count(//libro)
```

Devuelve el número de nodos <libro> que aparecen en el documento en cualquier posición.

```
//libro[autor = "Hunter"]
```

Devuelve todos los nodos <libro> que aparezcan en el documento en cualquier posición y que tengan un nodo hijo <autor> con el valor "Hunter". Los corchetes indican un filtro para seleccionar los resultados que cumplan una determinada condición.

```
//libro[@año > 1999]
```

Devuelve todos los nodos <libro> que tengan un atributo "año" con un valor superior a 1999. La arroba indica que "año" no es un hijo (una etiqueta) sino un atributo de la etiqueta libro.

```
//libro[@paginas]
```

Devuelve todos los nodos <libro> que tengan un atributo "paginas", independientemente del valor de ese atributo.

```
//libro/@paginas
```

Devuelve todos los atributos pagina de los nodos <libro>.

```
(i | b)
```

Devuelve todos los nodos <i> o todos los nodos que encuentre en el nodo contexto. Por defecto el nodo contexto es el nodo raíz del documento.

XPath soporta el uso de predicados, los cuales son condiciones booleanas que permiten seleccionar un subconjunto de los nodos referenciados. La expresión se evalúa por cada nodo encontrado y según sea cierta o falsa, el nodo se selecciona o se descarta.

A continuación se muestra un ejemplo:

```
(//servlet | //servlet-mapping) [servlet-name = $servlet]
```

Devuelve todos los nodos <servlet> o <servlet-mapping> que tengan un hijo llamado <servlet-name> cuyo valor coincida con el valor de la variable \$servlet.

Si el predicado contiene un literal numérico, se trata como si fuera un índice, tal y como muestra el siguiente ejemplo:

```
doc("libros.xml")/bib/libro/autor[1]
```

Esta expresión devuelve sólo el primer nodo autor que encuentre para cada nodo libro.

```
(doc("libros.xml")/bib/libro/autor)[1]
```

Esta expresión es igual a la anterior, excepto por los paréntesis. Estos paréntesis fuerzan a que se evalúe primero el recorrido por los nodos autor de cada libro, y después se aplique el predicado, por lo que el resultado de esta expresión será el primer autor que aparezca en el archivo "libros.xml".

```
//key[. = "Tiempo total"]
```

Devuelve todos los nodos <key> que tengan un valor de "Tiempo total.". El carácter "." representa el nodo contexto, que tiene una función similar al operador "this" en lenguajes como C++ o Java.

```
(//key)[1]/texto
```

Devuelve los nodos <texto> del primer nodo <key> del documento.

Headquarters

C/Doctor Trueta 50-54, 3º
08005 Barcelona, España
Tel: +34 932 251 070
Fax: +34 932 217 303

Orense 68, 11º
28020 Madrid, España
Tel: +34 917 885 747
Fax: +34 917 885 701

Pza. Alfonso el Magnánimo 1
46003 Valencia, España
Tel: +34 963 534 505
Fax: +34 963 944 127

1401 Brickell Av. # 320
33131 Miami, FL, USA
Tel: +1 305 503 3380
Fax: +1 305 503 3381

Mariano Escobedo, 543
Despacho 301, Rincon del Bosque
CP 11580 México DF
Tel: +52 55 5545 2646