

LENGUAJES DE MARCAS Y SISTEMAS DE GESTIÓN DE INFORMACIÓN

1º Desarrollo de Aplicaciones Multiplataforma.

UND 7: XQuery



Contenido

INTRODUCCIÓN.	2
Consultas en XQuery.	2
Reglas generales.	6
Diferencias entre las cláusulas for y let.	6
Funciones de entrada.	10
Expresiones condicionales.	10
Cuantificadores existenciales.	11
Operadores y funciones principales.	13
Comentarios.	16
Ejemplos de consultas.	17
Consulta 1.	17
Consulta 2.	18
Consulta 3.	18
Consulta 4.	19
Consulta 5.	20
Consulta 6 (XML a HTML)	21
Uso de BaseX para procesamiento de consultas XQuery.	22

INTRODUCCIÓN.

Se podría decir 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.

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. Las expresiones pueden combinarse con otras expresiones para crear nuevas expresiones más complejas y potentes.

Consultas en XQuery.

Como se deduce de lo anterior, XQuery necesita interpretar un XML como una base de datos, en la que una tupla es una ocurrencia de una etiqueta del documento XML, la cual se asigna a una variable.

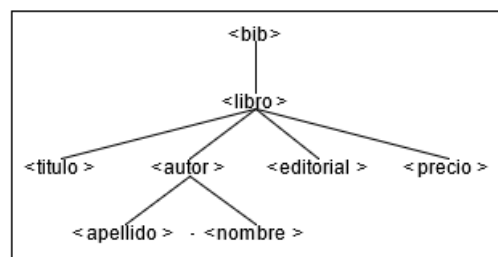
Una consulta en **XQuery** es una expresión que lee una secuencia de datos en XML y devuelve como resultado otra secuencia de datos en XML.

A diferencia de lo que sucede en SQL, en XQuery las expresiones y los valores que devuelven son dependientes del contexto. Es decir, los nodos que aparecerán en el resultado dependen de los espacios de nombres, 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:

Clausula	Definición
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.

Usando un documento XML con la siguiente estructura:



Un ejemplo de consulta muy simple sería:

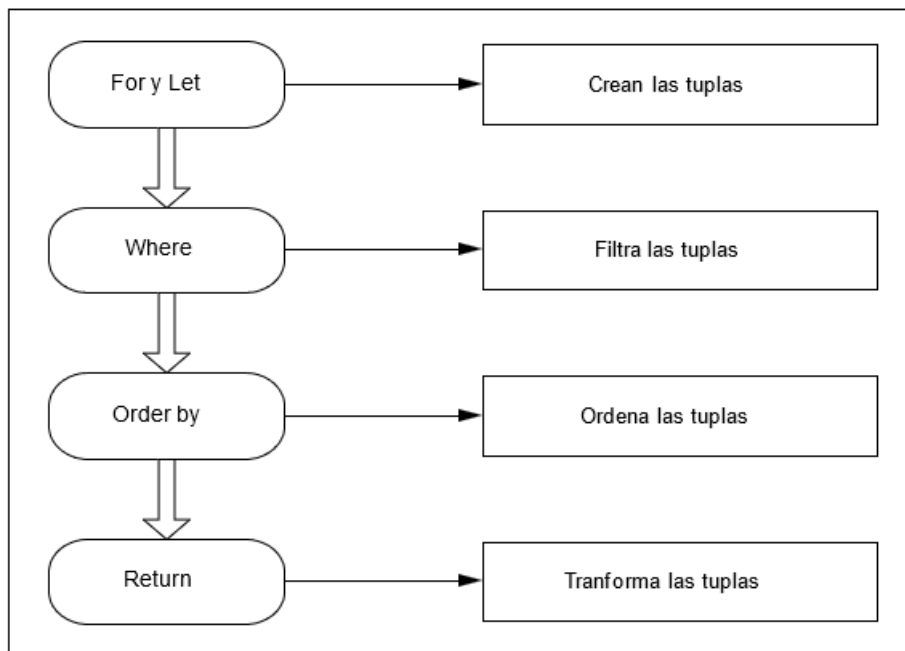
```
for $b in document("libros.xml")//bib/libro
```

Usando la 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**.

Esta otra consulta contiene las cinco clausulas:

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

Este es el orden en el que se ejecuta una consulta XQuery:



El documento XML que vamos a usar para los ejemplos es el siguiente:

```

<?xml version="1.0" encoding="UTF-8"?>
<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>
  </libro>
</bib>
  
```

```

        <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>

```

A continuación, se puede ver 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 “@”:

```

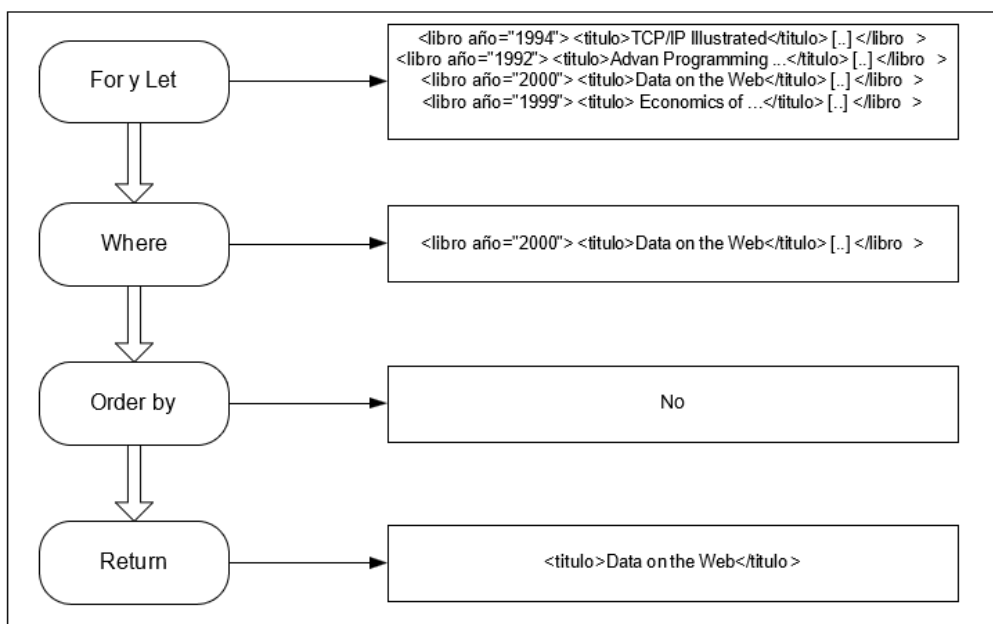
for $b in doc("libros.xml")//libro
where $b/@año = "2000"
return $b/titulo

```

Y el resultado sería:

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

En la siguiente figura se puede ver el proceso de ejecución de la consulta:



Reglas generales.

Cualquier consulta escrita en XQuery debe cumplir las siguientes reglas:

- **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.

```
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 qué diferencia hay en la información obtenida.

La consulta con una cláusula **for** se muestra a continuación:

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

El resultado de esta consulta se muestra a continuación:

```
<titulos>
  <titulo>TCP/IP Illustrated</titulo>
</titulos>
<titulos>
  <titulo>Advan Programming for Unix environment</titulo>
</titulos>
  <titulos>
<titulo>Data on the Web</titulo>
  </titulos>
<titulos>
  <titulo>Economics of Technology for Digital TV</titulo>
</titulos>
```

A continuación, repetimos la misma consulta sustituyendo la cláusula **for** por una cláusula **let**:

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

El resultado de esta consulta se muestra a continuación.

```
<titulos>
  <titulo>TCP/IP Illustrated</titulo>
  <titulo>Advan Programming for Unix environment</titulo>
  <titulo>Data on the Web</titulo>
  <titulo>Economics of Technology for Digital TV</titulo>
</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:

```
for $b in doc("libros.xml")//libro
let $c := $b/autor
return <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:

```
<libro>
  <titulo>TCP/IP Illustrated</titulo>
  <autores>1</autores>
</libro>
<libro>
  <titulo>Advanced Programming in the UNIX Environment</titulo>
  <autores>1</autores>
</libro>
<libro>
  <titulo>Data on the Web</titulo>
  <autores>3</autores>
</libro>
<libro>
  <titulo>The Economics of Technology and Content for Digital
  TV</titulo>
  <autores>0</autores>
</libro>
```

Si en la consulta aparece más de una cláusula **for** (o más de una variable en una cláusula **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**”.

El archivo “**comentarios.xml**” tiene el siguiente código:

```
<?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>
```

La consulta es la siguiente:

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

El resultado de esta consulta se muestra a continuación.

```
<comentario>
  <titulo>Data on the Web</titulo>
  <comentario>Un libro muy bueno sobre bases de datos.</comentario>
</comentario>
<comentario>
  <titulo>Advanced Programming in the Unix environment</titulo>
  <comentario>Un libro claro y detallado de programación en
  UNIX.</comentario>
</comentario>
<comentario>
  <titulo>TCP/IP Illustrated</titulo>
  <comentario>Uno de los mejores libros de TCP/IP.</comentario>
</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.".

```
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/last), ", ",
string($a/first)}</autor>
}
{
  if (count($b/autor) > 2)
  then <autor>et al.</autor>
  else ()
}
</libro>
```

El resultado de esta consulta se muestra a continuación.

```
<libro>
  <titulo>TCP/IP Illustrated</titulo>
  <autor>Stevens, W.</autor>
</libro>
<libro>
  <titulo>Advanced Programming in the Unix Environment</titulo>
  <autor>Stevens, W.</autor>
</libro>
<libro>
  <titulo>Data on the Web</titulo>
  <autor>Abiteboul, Serge</autor>
  <autor>Buneman, Peter</autor>
  <autor>et al.</autor>
</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 son 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 título. Si el libro tiene más de un título 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 $b//titulo satisfies (contains($p,"Unix") AND
contains($p,"programing"))
return $b/titulo
```

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 $b//titulo satisfies contains($p,"programing")
return $b/titulo
```

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

Operadores y funciones principales.

El conjunto de funciones y operadores soportado por XQuery 1.0 es el mismo conjunto de funciones y operadores utilizado en XPath 2.0 y XSLT 2.0.

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.

Tipos	Operadores y Funciones
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()

(*) 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 $1 in distinct-values(doc("libros.xml")// (autor | editor)/apellido)
order by $1
return <apellidos>{$1}</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.

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 :)
```

La sintaxis de comentario usada en XML no es aplicable a XQuery salvo que la consulta se escriba con la sintaxis XQueryX, como se verá más adelante.

Ejemplos de consultas

Como ya hemos visto, la consulta más sencilla posible en XQuery es una expresión en XPath, por ejemplo:

```
document("libros.xml")//libro[titulo="Ricotta Pie"]
```

La expresión anterior devuelve todos los nodos **<libro>** que tengan un hijo **<titulo>** con el valor “Ricotta Pie” del conjunto de datos almacenado en el archivo “**libros.xml**”. El orden de los libros será, por defecto, el mismo orden en que aparecen en el documento “**libros.xml**”.

Partiendo del conjunto de datos que venimos usando en el texto, vamos a escribir una serie de consultas y a mostrar cuales serían los resultados obtenidos.

Consulta 1

Escribir una consulta que obtenga el nombre y el año de todos los libros publicados por Addison-Wesley después de 1991. El código de la consulta será:

```
<bib> {  
  for $b in doc("libros.xml")/bib/libro  
  where $b/editorial = "Addison-Wesley" and $b/@año > 1991  
  return  
    <libro año="{ $b/@año }">  
      { $b/titulo }  
    </libro>  
}  
</bib>
```

El resultado de esta consulta se muestra a continuación:

```
<bib>  
  <libro año="1994">  
    <titulo>TCP/IP Illustrated</titulo>  
  </libro>  
  <libro año="1992">  
    <titulo>Advanced Programming in the Unix environment  
    </titulo>  
  </libro>  
</bib>
```

Consulta 2

Escribir una consulta que obtenga el título de los libros cuyo precio esté por debajo de 50.00€:

```
for $b in doc("libros.xml")//libro
where $b/precio < 50.00
return $b/titulo
```

El resultado de esta consulta se muestra a continuación.

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

Consulta 3

Escribir una consulta que, por cada libro almacenado en el archivo "**libros.xml**" devuelva el título del libro, el precio con que consta dicho libro en el archivo "**libros.xml**" y el precio con que consta ese libro en el archivo "**comentarios.xml**".

```
<libros-con-precios>
{
  for $b in doc("libros.xml")//libro,
    $a in doc("comentarios.xml")//entrada
  where $b/titulo = $a/titulo
  return
    <libro-con-precios>
      { $b/titulo }
      <precio-btienda2>{ $a/precio/text() }
    </precio-btienda2>
      <precio-btienda1>{ $b/precio/text() }
    </precio-btienda1>
  </libro-con-precios>
}
</libros-con-precios>
```

Los resultados de esta consulta se muestran a continuación.

```
<libros-con-precios>
  <libro-con-precios>
    <titulo>TCP/IP Illustrated</titulo>
```

```

        <precio-btienda2>65.95</precio-btienda2>
        <precio-btienda1>65.95</precio-btienda1>
    </libro-con-precios>
</libro-con-precios>
    <titulo>Advanced Programming in the Unix environment</titulo>
    <precio-btienda2>65.95</precio-btienda2>
    <precio-btienda1>65.95</precio-btienda1>
</libro-con-precios>
</libro-con-precios>
    <titulo>Data on the Web</titulo>
    <precio-btienda2>34.95</precio-btienda2>
    <precio-btienda1>39.95</precio-btienda1>
</libro-con-precios>
</libros-con-precios>

```

Consulta 4

Escribir una consulta que, por cada libro con autores, devuelva el título del libro y sus autores. Si el libro no tiene autores, pero sí editor, la consulta devolverá el título del libro y la afiliación del editor.

```

<bib> {
  for $b in doc("libros.xml")//libro[autor]
  return <libro> { $b/titulo } { $b/autor }
    </libro>
  }
  {
    for $b in doc("libros.xml")//libro[editor]
    return <referencia> { $b/titulo } { $b/editor/afiliacion }
      </referencia>
    }
  }
</bib>

```

Los resultados de esta consulta se muestran a continuación:

```

<bib>
  <libro>
    <titulo>TCP/IP Illustrated</titulo>
    <autor>
      <apellidos>Stevens</apellidos>
      <nombre>W.</nombre>
    </autor>
  </libro>
  <libro>
    <titulo>Advanced Programming in the Unix environment
    </titulo>

```

```

        <autor>
            <apellidos>Stevens</apellidos>
            <nombre>W.</nombre>
        </autor>
    </libro>
    <libro>
        <titulo>Data on the Web</titulo>
        <autor>
            <apellidos>Abiteboul</apellidos>
            <nombre>Serge</nombre>
        </autor>
        <autor>
            <apellidos>Buneman</apellidos>
            <nombre>Peter</nombre>
        </autor>
        <autor>
            <apellidos>Suciu</apellidos>
            <nombre>Dan</nombre>
        </autor>
    </libro>
    <referencia>
        <titulo>The Economics of Technology and Content for
        Digital TV </titulo>
        <afiliacion>CITI</afiliacion>
    </referencia>
</bib>

```

Consulta 5

Mostrar los pares de títulos que sean distintos, pero tengan el mismo autor o grupo de autores. Hay que tener en cuenta que el orden de aparición de los autores puede variar de un libro a otro.

```

<bib> {
  for $libro1
    in doc("libros.xml")//libro, $libro2
    in doc("libros.xml")//libro
  let $aut1 := for $a in $libro1/autor
  order by $a/apellidos, $a/nombre
  return $a
  let $aut2 := for $a in $libro2/autor
  order by $a/apellidos, $a/nombre
  return $a
  where $libro1 << $libro2
    and not($libro1/titulo = $libro2/titulo)
    and deep-equal($aut1, $aut2)
  return
    <libro-par>
      { $libro1/titulo }
      { $libro2/titulo }
    </libro-par>
}
</bib>

```

Los resultados de esta consulta se muestran a continuación:

```
<bib>
  <libro-par>
    <titulo>TCP/IP Illustrated</titulo>
    <titulo>Advanced Programming in the Unix environment</titulo>
  </libro-par>
</bib>
```

Esta consulta usa la función **deep-equal()** encargada de comparar secuencias de nodos. Para la función **deep-equal()** dos consultas son iguales si todos los nodos de la primera secuencia aparecen en la segunda secuencia en la misma posición que en la primera secuencia.

Consulta 6 (XML a HTML)

También es posible utilizar XQuery para transformar datos XML en otros formatos, como HTML, convirtiéndose XQuery en una alternativa más sencilla y rápida de usar que XSLT. A continuación, como ejemplo, se muestra una consulta que crea una tabla HTML con los títulos de todos los libros contenidos en el archivo **"libros.xml"**.

```
<html>
  <head>
    <title>
    </title>
  </head>
  <body>
    <table>
      { for $b in doc("libros.xml")/bib/libro
        return
          <tr> <td> <I> { string( $b/titulo ) } </I> </td> </tr>
      }
    </table>
  </body>
</html>
```


El resultado de la consulta anterior se muestra a continuación:

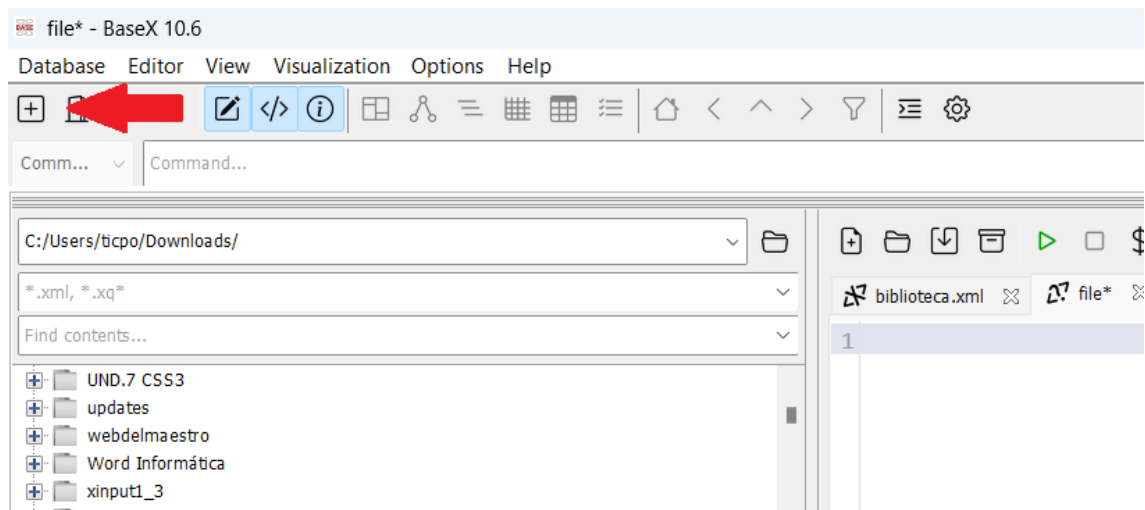
```
<html>
  <head>
    <title>
    </title>
  </head>
  <body>
    <table>
      <tr>
        <td><I>TCP/IP Illustrated</I></td>
      </tr>
      <tr>
        <td><I>Advan Programming for Unix
environment</I></td>
      </tr>
      <tr>
        <td><I>Data on the Web</I></td>
      </tr>
      <tr>
        <td><I>Economics of Technology for Digital
TV</I></td>
      </tr>
    </table>
  </body>
</html>
```

Uso de BaseX para procesamiento de consultas XQuery

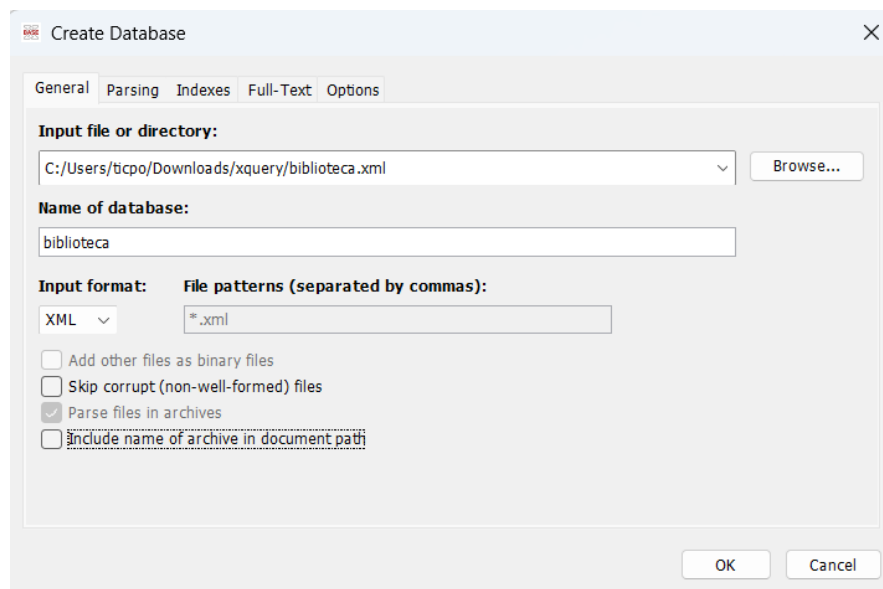
Para el procesamiento de consultas XQuery podemos usar el programa BaseX que se puede encontrar en este enlace:

<https://basex.org/download/>

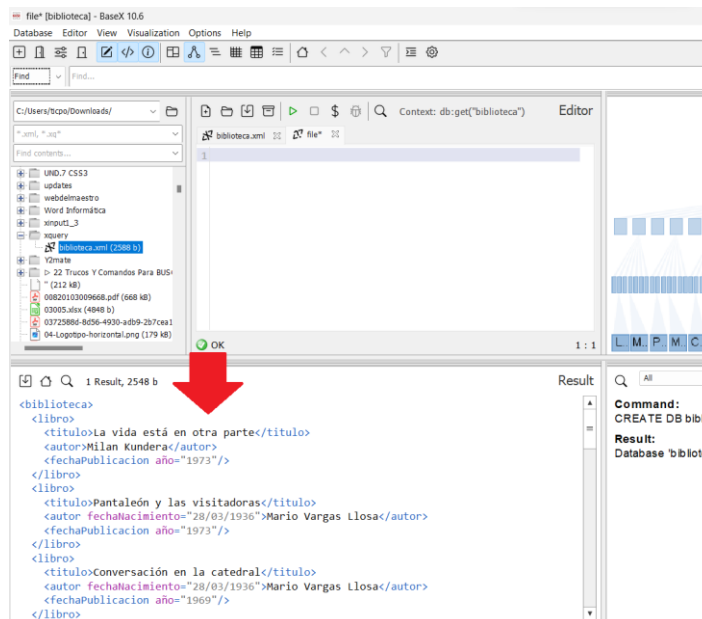
Esta aplicación se utiliza para la manipulación de bases de datos. En nuestro caso lo que vamos a hacer es generar bases de datos a partir de un documento XML. Para ello hacemos clic sobre el símbolo  que encontramos arriba a la derecha en la aplicación y que está marcado con una flecha roja en la siguiente imagen:



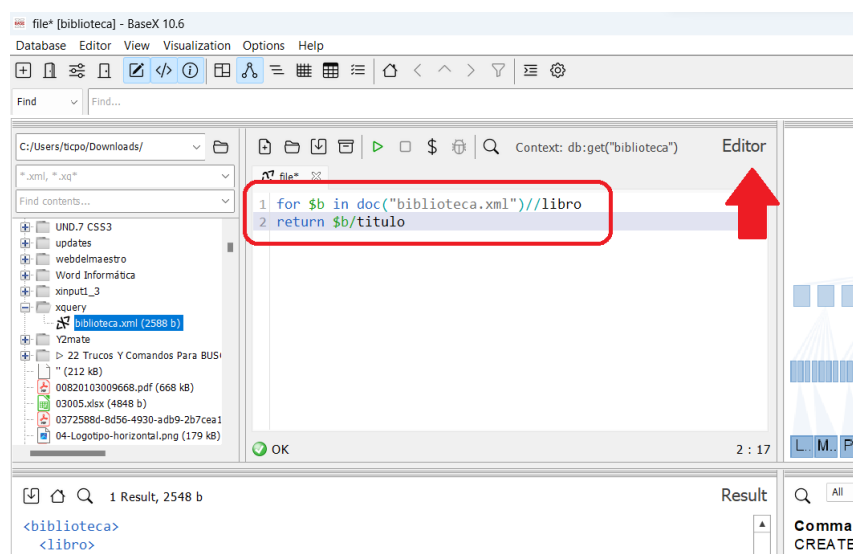
Se nos abrirá una ventana donde podremos buscar el documento XML dentro de nuestro equipo y convertirlo en base de datos. Hay que asegurarse que está seleccionado XML como formato de entrada (Input format).




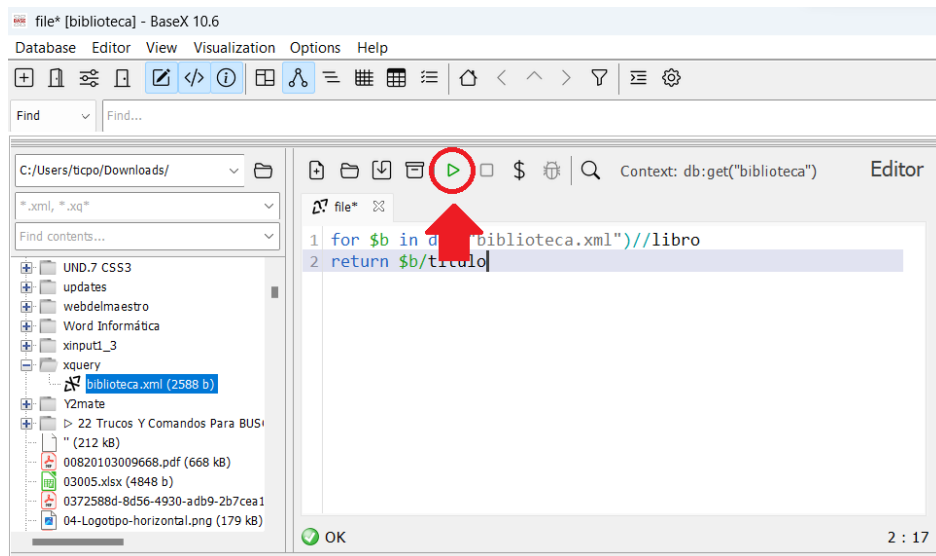
Ahora en la pantalla principal del programa, en la ventana Result, nos aparecerá el contenido del documento XML:



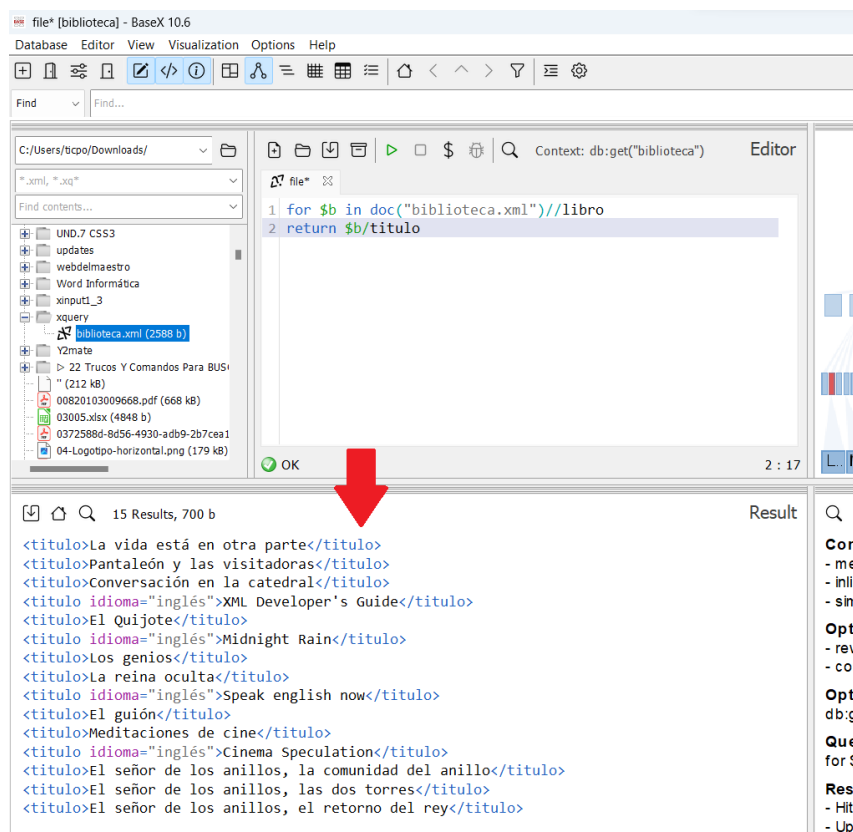
Ahora podremos escribir la consulta XQuery en la ventana editor:




Hacemos clic en el icono ejecutar  que hay en esa misma ventana en el lugar en que aparece marcado en la siguiente imagen:



Y el resultado aparecerá en la ventana Result:



También podemos guardar la consulta XQuery con la extensión xq haciendo clic en Save as... que nos aparecerá en el icono Guardar  de la ventana Editor.

