XQuery es el lenguaje que permite definir de forma rápida y compacta, consultas o recorridos complejos sobre colecciones de datos en XML los cuales devuelvan todos los nodos que cumplan ciertas condiciones.

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

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>
```

Consultas en 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.

Ya hemos visto que **XPath** es un lenguaje declarativo para la localización de nodos y fragmentos de información en árboles **XML**. Puesto que XQuery ha sido construido sobre la base de XPath y realiza la selección de información y la iteración a través del conjunto de datos basándose en dicho lenguaje, **toda expresión XPath también es una consulta XQuery válida**.

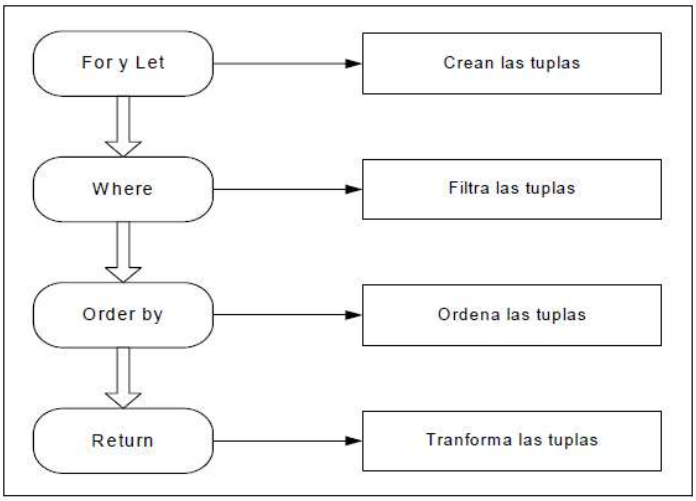
Los **comentarios** en XQuery están limitados entre caras sonrientes, es decir: (: Esto es un comentario XQuery :).

En un documento XQuery **los caracteres { } delimitan las expresiones que son evaluadas** para crear un documento nuevo.

XQuery admite expresiones condicionales del tipo **if-then-else** con la misma semántica que tienen en los lenguajes de programación habituales.

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 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 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 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
```

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

```
for $b in doc("C:/Program Files (x86)/BaseX/repo/librosXQuery.xml")//libro
let $c := $b//autor
where count($c) > 2
order by $b/titulo
return $b/ titulo
```

La misma sentencia desde la base de datos creada desde el fichero:

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

El resultado de estas consultas se muestra a continuación.

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

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 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 //libro
where $b/@año = "2000"
return $b/titulo
```

El resultado de la consulta anterior se muestra a continuación.

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

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.

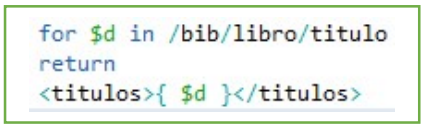
```
doc("libros.xml")/bib/libro/titulo[/bib/libro/autor/apellido='Stevens']
```

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

Diferencias entre las cláusulas for y let

La consulta con una cláusula for se muestra a continuación:

```
for $d in /bib/libro/titulo
return
<titulos>{ $d }</titulos>
```

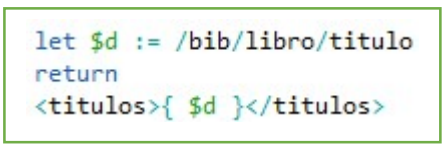


El resultado de esta consulta se muestra a continuación:

```
<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 una cláusula let.

```
let $d := /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 //libro
let $c := $b/autor
return
<libro>{ $b/titulo, <autores>{ count($c) }</autores>}</libro>
```


```
for $b in //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:

```
<libro>
  <titulo>TCP/IP Illustrated</titulo>
  <autores>1</autores>
</libro>
<libro>
  <titulo>Advan Programming for Unix environment</titulo>
  <autores>1</autores>
</libro>
<libro>
  <titulo>Data on the Web</titulo>
  <autores>3</autores>
</libro>
<libro>
  <titulo>Economics of Technology for Digital TV</titulo>
  <autores>0</autores>
</libro>
```

Expresiones condicionales

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 "y más…".

```
for $b in //libro
return
      <libro>
            { $b/titulo }
            {
                  for $a at $i in $b/autor
                  where $i <= 2
                  return <autor>{string($a/apellido), ", ",
                  string($a/nombre)}</autor>
            }
            {
                  if (count($b/autor) > 2)
                  then <autor>y más…</autor>
                  else ()
            }
      </libro>
```

Resultado:

```
for $b in //libro
return
<libro>
{ $b/titulo }
{
for $a at $i in $b/autor
where $i <= 2
return <autor>{string($a/apellido), ", ",
string($a/nombre)}</autor>
}
{
if (count($b/autor) > 2)
then <autor>y más…</autor>
else ()
}
</libro>
```

```
<libro>
  <titulo>TCP/IP Illustrated</titulo>
  <autor>Stevens ,  W.</autor>
</libro>
<libro>
  <titulo>Advan Programming for Unix environment</titulo>
  <autor>Stevens ,  W.</autor>
</libro>
<libro>
  <titulo>Data on the Web</titulo>
  <autor>Abiteboul ,  Serge</autor>
  <autor>Buneman ,  Peter</autor>
  <autor>y más…</autor>
</libro>
<libro>
  <titulo>Economics of Technology for Digital TV</titulo>
</libro>
```

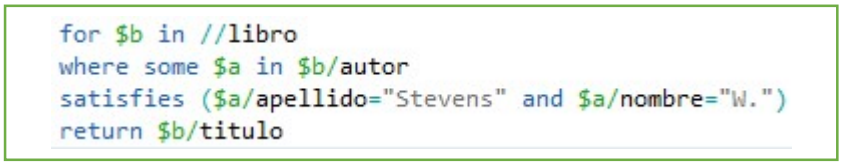
La cláusula where de una consulta permite filtrar las tuplas que aparecerán en el resultado, mientras que una expresión condicional nos permite crear una u otra estructura de nodos en el resultado que dependa de los valores de las tuplas filtradas.

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

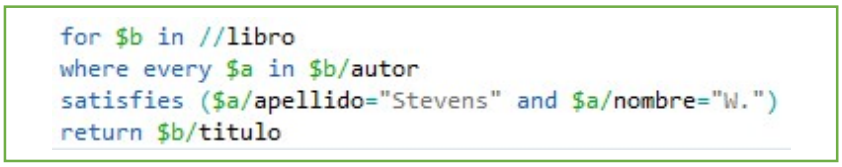


El resultado de esta consulta se muestra a continuación

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

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



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

```
<titulo>TCP/IP Illustrated</titulo>
<titulo>Advan Programming for Unix environment</titulo>
<titulo>Economics of Technology 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 //libro
where some $p in $b //titulo
satisfies (contains($p,"Unix") and contains($p,"Programming"))
return $b/titulo
```

```
<titulo>Advan Programming for Unix environment</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 //libro
where every $p in $b //titulo
satisfies (contains($p,"on"))
return $b/titulo
```

Esta consulta es distinta de la anterior ya que no es suficiente que "on" 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".

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

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 son:

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(), starts-with(), 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.

Operadores

- **Comparación de valores:** Comparan dos valores escalares y produce un error si alguno de los operandos es una secuencia de longitud mayor de 1. Estos operadores son:
  - **eq**, igual.
  - **ne**, no igual.
  - **lt**, menor que.
  - **le**, menor o igual que.
  - **gt**, mayor que.
  - **ge**, mayor o igual que.

```
for $b in //libro
where every $p in $b/@año
satisfies ($p gt "1995")
return $b/titulo

<titulo>Data on the Web</titulo>
<titulo>Economics of Technology for Digital TV</titulo>
```

- **Comparación general:** Permiten comparar operandos que sean secuencias.
  - **=**, igual.
  - **!=**, distinto.
  - **>**, mayor que.
  - **>=**, mayor o igual que.
  - **<,** menor que.
  - **<=**, menor o igual que.
- **Comparación de nodos:** Comparan la identidad de dos nodos.
  - **is**, devuelve true si las dos variables que actúan de operandos están ligadas al mismo nodo.
  - **is not**, devuelve true si las dos variables no están ligadas al mismo nodo.
- **Comparación de órdenes de los nodos: <<**, compara la posición de dos nodos. Devuelve "**true**" si el nodo ligado al primer operando ocurre primero en el orden del documento que el nodo ligado al segundo.
- **Lógicos: and** y **or** Se emplean para combinar condiciones lógicas dentro de un predicado.
- **Secuencias de nodos:** Devuelven secuencias de nodos en el orden del documento y eliminan duplicados de las secuencias resultado.
  - **Unión**, devuelve una secuencia que contiene todos los nodos que aparecen en alguno de los dos operandos que recibe.
  - **Intersect**, devuelve una secuencia que contiene todos los nodos que aparecen en los dos operandos que recibe.
  - **Except**, devuelve una secuencia que contiene todos los nodos que aparecen en el primer operando que recibe y que no aparecen en el segundo.
- **Aritméticos**: **+, -, *, div y mod**, devuelven respectivamente la suma, diferencia, producto, cociente y resto de operar dos números dados.

Funciones

- **Funciones numéricas**
  - **floor(),** que devuelve el valor numérico inferior más próximo al dado**.**
  - **ceiling(),** que devuelve el valor numérico superior más próximo al dado**.**
  - **round(),** que redondea el valor dado al más próximo.
  - **count(),** determina el número de ítems en una colección.
  - **min()** o **max(),** devuelven respectivamente el mínimo y el máximo de los valores de los nodos dados.
  - **avg(),** calcula el valor medio de los valores dados.
  - **sum(),** calcula la suma total de una cantidad de ítems dados.

- **Funciones de cadenas de texto**
  - **concat(),** devuelve una cadena construida por la unión de dos cadenas dadas.
  - **string-length(),** devuelve la cantidad de caracteres que forman una cadena.
  - **startswith(), ends-with(),** determinan si una cadena dada comienza o termina, respectivamente, con otra cadena dada.
  - **upper-case(), lower-case(),** devuelve la cadena dada en mayúsculas o minúsculas respectivamente.

- **Funciones de uso general**
  - **empty(),** devuelve "**true**" cuando la secuencia dada no contiene ningún elemento.
  - **exits(),** devuelve "**true**" cuando una secuencia contiene, al menos, un elemento.
  - **distinct-values(),** extrae los valores de una secuencia de nodos y crea una nueva secuencia con valores únicos, eliminando los nodos duplicados.

- **Cuantificadores existenciales:**
  - **some, every**, permiten definir consultas que devuelven algún, o todos los elementos, que verifiquen la condición dada.

Además de estas funciones que están definidas en el lenguaje, XQuery permite al desarrollador construir sus propias funciones:

```
declare nombre_funcion($param1 as tipo_dato1, $param2 as
tipo_dato2,… $paramN as tipo_datoN)
as tipo_dato_devuelto
{
...CÓDIGO DE LA FUNCIÓN...
}
```

En el siguiente ejemplo puedes ver la definición y llamada a una función escrita por el usuario que nos calcula el precio de un libro una vez que se le ha aplicado el descuento.

```
declare function local:minPrice($p as xs:decimal?,$d as
xs:decimal?)
as xs:string?
{
let $disc := ($p * $d) div 100
return (string($p - $disc))
};

for $libro in //libro
return <precioMin>{local:minPrice($libro/precio,5)}</precioMin>
```

```
declare function local:minPrice($p as xs:decimal?,$d as xs:decimal?)
as xs:string?
{
let $disc := ($p * $d) div 100
return (string($p - $disc))
};

for $libro in //libro
return <precioMin>{local:minPrice($libro/precio,5)}</precioMin>
```

El resultado obtenido será:

```
<precioMin>62.6525</precioMin>
<precioMin>62.6525</precioMin>
<precioMin>37.9525</precioMin>
<precioMin>123.4525</precioMin>
```

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(//(autor | editor)/apellido)
order by $l
return <apellidos>{ $l }</apellidos>
```

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

El resultado de esta consulta es:

```
<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 //libro
where $b/titulo = "TCP/IP Illustrated"
return
<libro>
{ $b/@* }
{ $b/* except $b/precio }
</libro>
```

```
for $b in //libro
where $b/titulo = "TCP/IP Illustrated"
return
<libro>
{ $b/@* }
{ $b/* except $b/precio }
</libro>
```

El resultado de esta consulta se muestra a continuación:

Antes

```
<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>
```

Después

```
<libro año="1994">
  <titulo>TCP/IP Illustrated</titulo>
  <autor>
    <apellido>Stevens</apellido>
    <nombre>W.</nombre>
  </autor>
  <editorial>Addison-Wesley</editorial>
</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. Existen autores con el mismo apellido:

```
for $l in //autor/apellido/text()
return <apellidos>{ $l }</apellidos>
```

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

La siguiente consulta devuelve todos los apellidos distintos de los autores.

```
for $l in distinct-values(//autor/apellido)
return <apellidos>{ $l }</apellidos>
```

El resultado es:

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

La función empty() devuelve cierto valor 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 $b in //libro
where (empty($b/autor))
return $b
```


```
for $b in //libro
where (empty($b/autor))
return $b
```


El resultado de esta consulta se muestra a continuación:

```
<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>
```

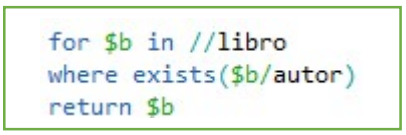
```
for $b in //libro
where not(empty($b/autor))
return $b
```

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>
```

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 $b in //libro
where exists($b/autor)
return $b
```



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