- 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).

XQuery ha sido construido sobre la base de Xpath[3]. Xpath[5] es un lenguaje declarativo para la localización de nodos y fragmentos de información en árboles XML. XQuery se basa en este lenguaje para realizar la selección de información y la iteración a través del conjunto de datos.

2.2. Colección de datos de ejemplo.

En los ejemplos de consultas que veremos a lo largo de este trabajo, vamos a trabajar con un conjunto de datos compuesto por fichas de varios libros almacenadas en un archivo local llamado “libros.xml”, cuyo contenido se muestra a continuación.

```
<?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>
```

[5] Para más información sobre XPath consultar el apéndice III y el apéndice IV.

```
            <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>
```

A continuación se muestra el contenido del DTD correspondiente al archivo "libros.xml".

```
<!ELEMENT bib (libro* )>
<!ELEMENT libro (titulo,(autor+ | editor+ ),editorial, precio )>
<!ATTLIST libro year CDATA #REQUIRED >
<!ELEMENT autor (apellido, nombre )>
<!ELEMENT editor (apellido, nombre, afiliacion )>
<!ELEMENT titulo (#PCDATA )>
<!ELEMENT apellido (#PCDATA )>
<!ELEMENT nombre (#PCDATA )>
<!ELEMENT afiliacion (#PCDATA )>
<!ELEMENT editorial (#PCDATA )>
```

```
<!ELEMENT precio (#PCDATA )>
```

En algunas consultas también necesitaremos combinar la información contenida en el archivo "libros.xml" con la información contenida en el archivo "comentarios.xml". El contenido de este archivo se muestra a continuación.

```
<?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>
```

A continuación se muestra el contenido del DTD correspondiente al archivo "comentarios.xml".

```
<!ELEMENT comentarios (entrada*)>
<!ELEMENT entrada   (titulo, precio, comentario)>
<!ELEMENT titulo   (#PCDATA)>
<!ELEMENT precio   (#PCDATA)>
<!ELEMENT comentario  (#PCDATA)>
```

3. Consultas en XQuery.

Una consulta en XQuery[4][5] 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, en la tabla 1, 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.

Tabla 1. Posibles cláusulas en una consulta XQuery.

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

A continuación, en la figura 2, se muestra gráficamente el orden en que se ejecuta cada cláusula de una consulta y los resultados de cada una:

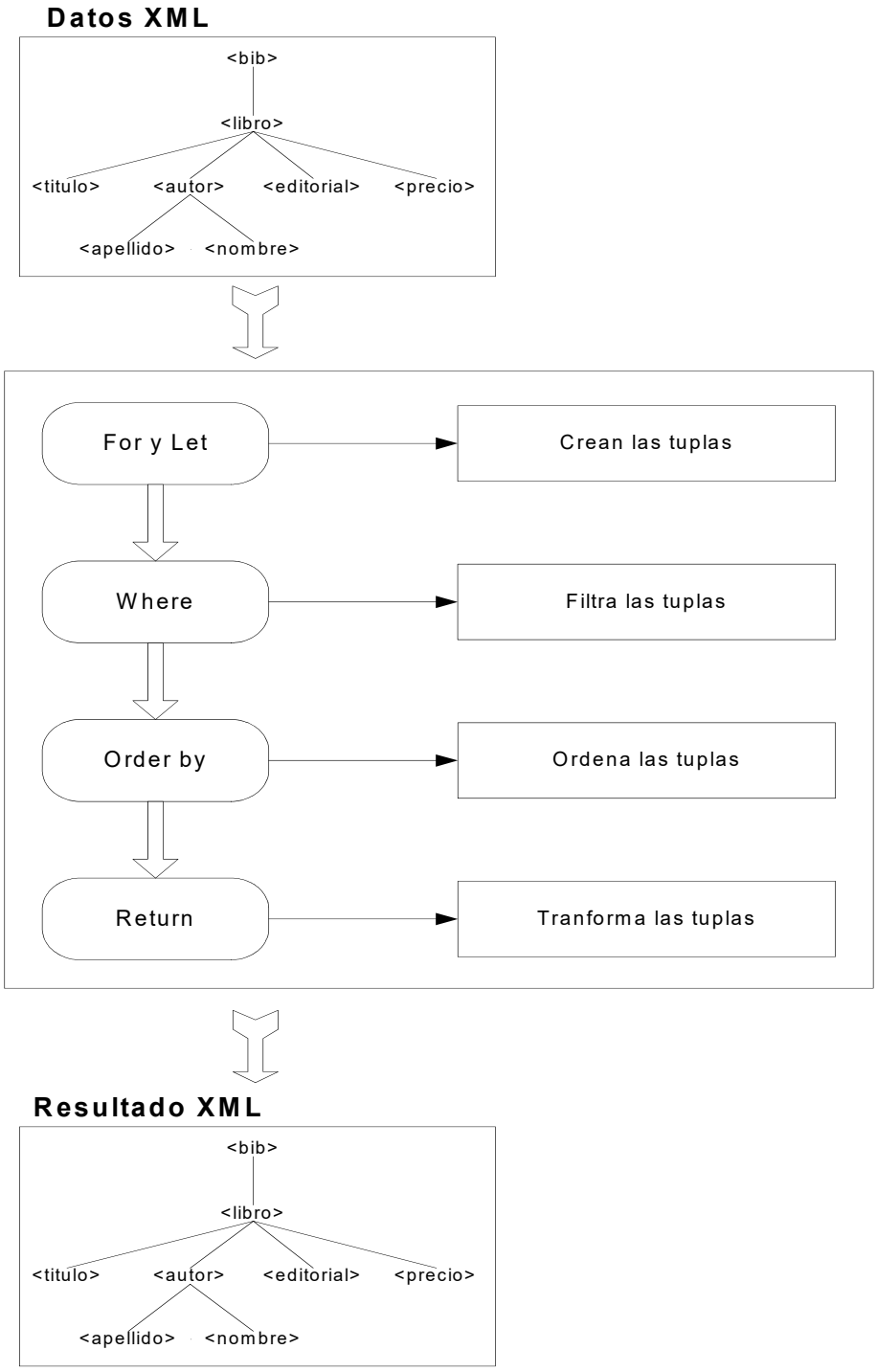

Figura 2. Orden de ejecución y resultado de las cinco cláusulas posibles.

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

El resultado de esta consulta se muestra a continuación.

```
<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 "@".

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

El resultado de la consulta anterior se muestra a continuación.

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

A continuación, en la figura 3, se muestra de forma gráfica como se ejecutaría la consulta anterior y los resultados parciales de cada una de sus cláusulas:

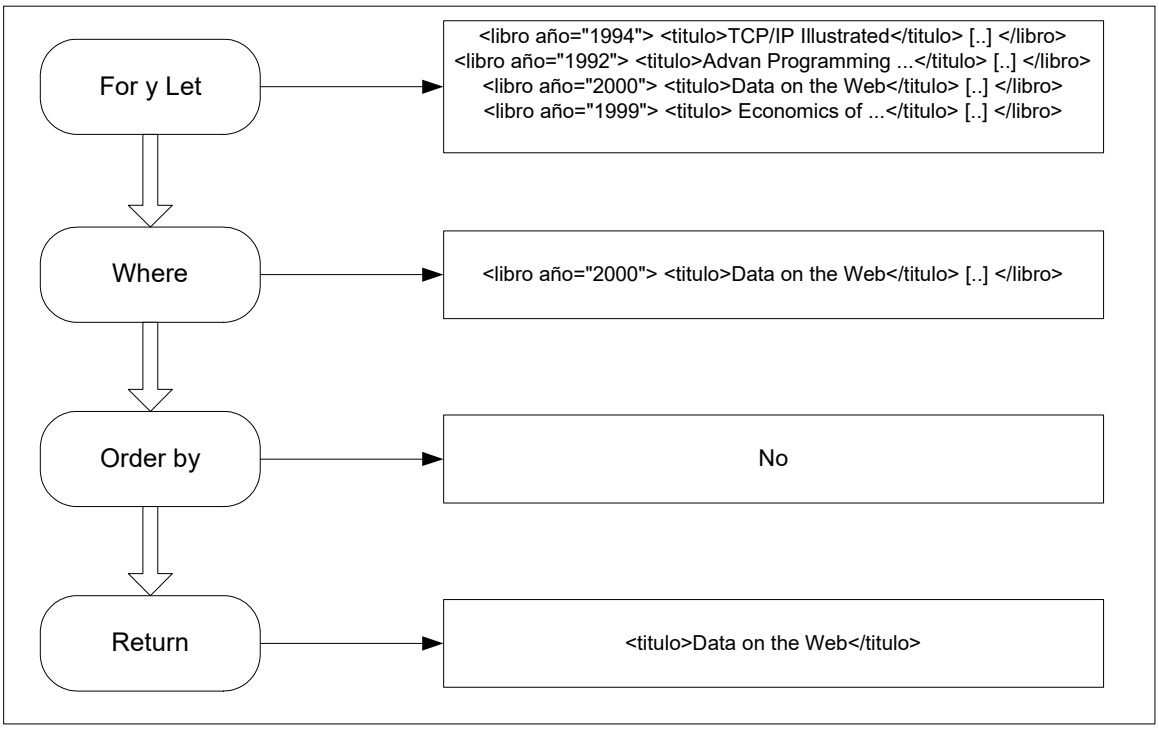

Figura 3. Ejecución de una consulta XQuery con los resultados de cada cláusula.

3.1. Reglas generales.

A continuación, enunciamos una serie de reglas que debe cumplir cualquier consulta escrita en XQuery[4][5]:

- 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='St
evens']
```

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.

3.2. 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.

```
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 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 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”.

```
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>
```

3.3. 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.

3.4. 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.".