5 XQUERY

5.1 INTRODUCCIÓN: NECESIDAD DE XQUERY

Actualmente, XML se ha convertido en una herramienta de uso cotidiano en los entornos de tratamiento de información y en los entornos de programación. Sin embargo, a medida que se emplea en un mayor número de proyectos de complejidad y tamaño crecientes y la cantidad de datos almacenados en XML aumenta, se comprueba que, las herramientas más habituales para manipular desde un programa un árbol con un conjunto de datos en XML, los parsers SAX y DOM, no son prácticas para manejar grandes y complejas colecciones de datos en XML.

El principal problema a la hora de procesar colecciones de datos en XML a bajo nivel mediante parsers DOM y SAX es la necesidad de escribir una gran cantidad de código para realizar el procesamiento. Este código consiste, básicamente, en recorrer un árbol, para un parsers DOM, o detallar una lista de acciones según la etiqueta que se encuentre para parsers SAX.

Otra solución existente en la actualidad es el estándar XSLT que permite definir transformaciones sobre colecciones datos en XML en otros formatos, como HTML o PDF y las herramientas que le dan soporte. Sin embargo XSLT sólo es aplicable en los casos en los que desee obtener una representación distinta de los datos, no cuando se desea localizar una información concreta dentro de dicho conjunto de datos.

Un ejemplo de un escenario donde no es aplicable ninguna de las herramientas comentadas hasta ahora (parsers, XSLT) lo encontramos a la hora de consultar la información en los actuales servidores de bases de datos que ya incorporan funciones para poder obtener los datos en XML o bases de datos XML nativas (Native XML Database). El escenario de uso más común se da cuando tenemos que realizar alguna búsqueda en un documento o conjunto de documentos con gran cantidad de datos en XML, o bien hemos de trabajar sólo sobre un subconjunto de todos los datos XML y queremos obtener dicho subconjunto de una forma sencilla, para evitar trabajar con toda la colección de datos.

A la vista de todo lo comentado hasta ahora, se hace necesaria la aparición de un lenguaje que permita 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. Este lenguaje debe ser, además, declarativo, es decir, independientemente de la forma en que se realice el recorrido o de donde se encuentren los datos. Este lenguaje ya existe y se llama **XQuery**.

A lo largo de este capítulo vamos a mostrar, con multitud de ejemplos, las características principales de este lenguaje y vamos a mostrar como utilizar uno de los motores existentes para incorporar capacidad de procesado de consulta XQuery en nuestras aplicaciones.

5.2 DEFINICIÓN DE XQUERY

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.

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.

5.2.1 REQUERIMIENTOS TÉCNICOS DE XQUERY

El grupo de trabajo en XQuery del W3C ha definido un conjunto de 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 [**1**].
- Debe ser capaz de soportar XML-Schemas y DTDs 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.

[1] Un namespace es un espacio de definición de etiquetas que permite que dos diseñadores puedan utilizar el mismo nombre de etiqueta con dos significados semánticos distintos sin que entren en conflicto. Dos etiquetas con el mismo nombre declaradas en namespaces diferentes se consideran dos etiquetas diferentes.

- 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. Xpath 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.

5.2.2 COLECCIÓN DE DATOS DE EJEMPLO

En los ejemplos de consultas que veremos a lo largo de este capitulo, 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>Advance 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>
```

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)>
```

5.3 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.

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 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 figura 5-1, se muestra gráficamente el orden en que se ejecuta cada cláusula de una consulta y los resultados de cada una:

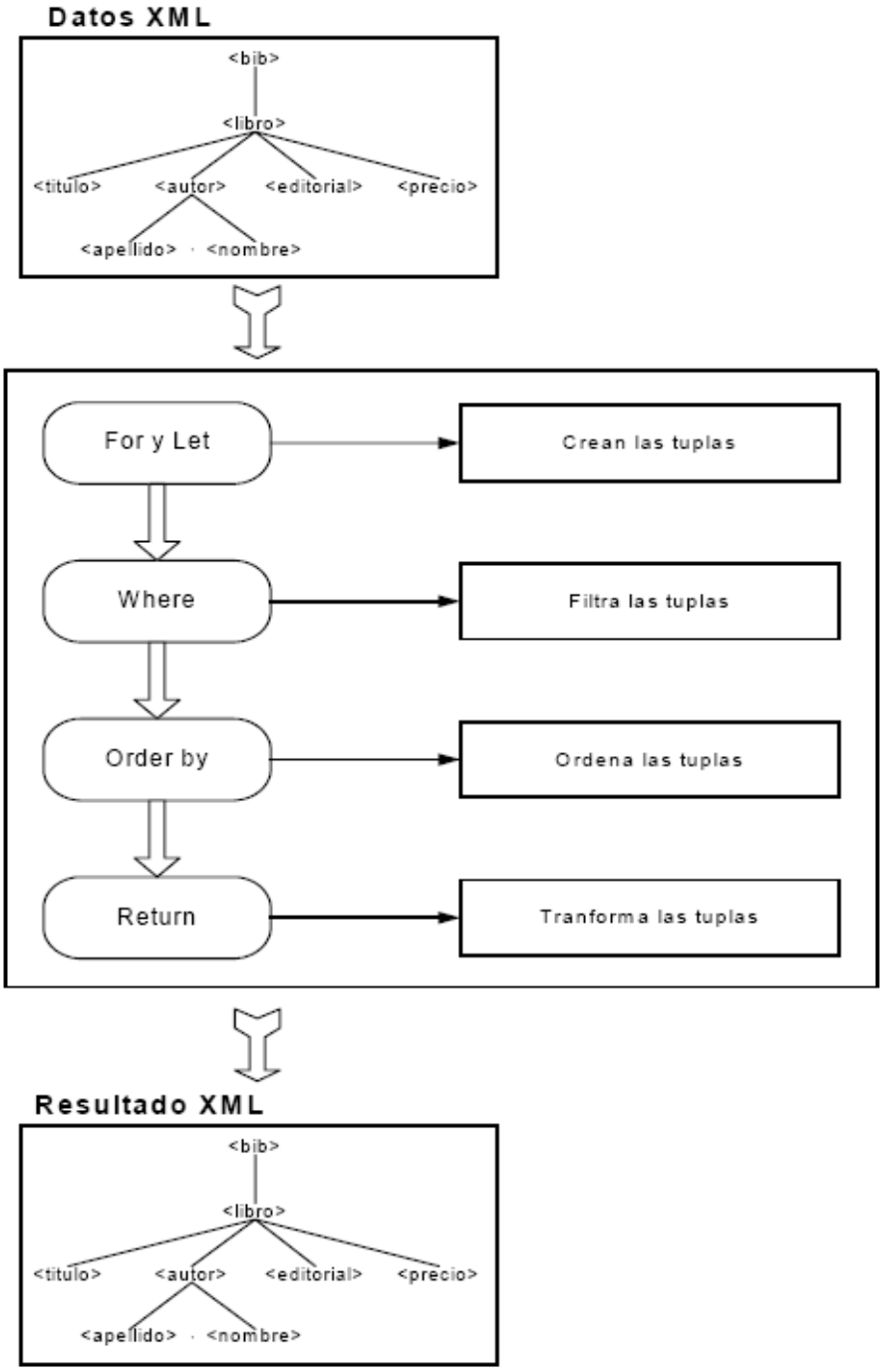

Figura 5-1: Orden de ejecución y resultado de las cinco cláusulas posibles.

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 cláusulas. 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 5-2, se muestra de forma gráfica como se ejecutaría la consulta anterior y los resultados parciales de cada una de sus cláusulas:

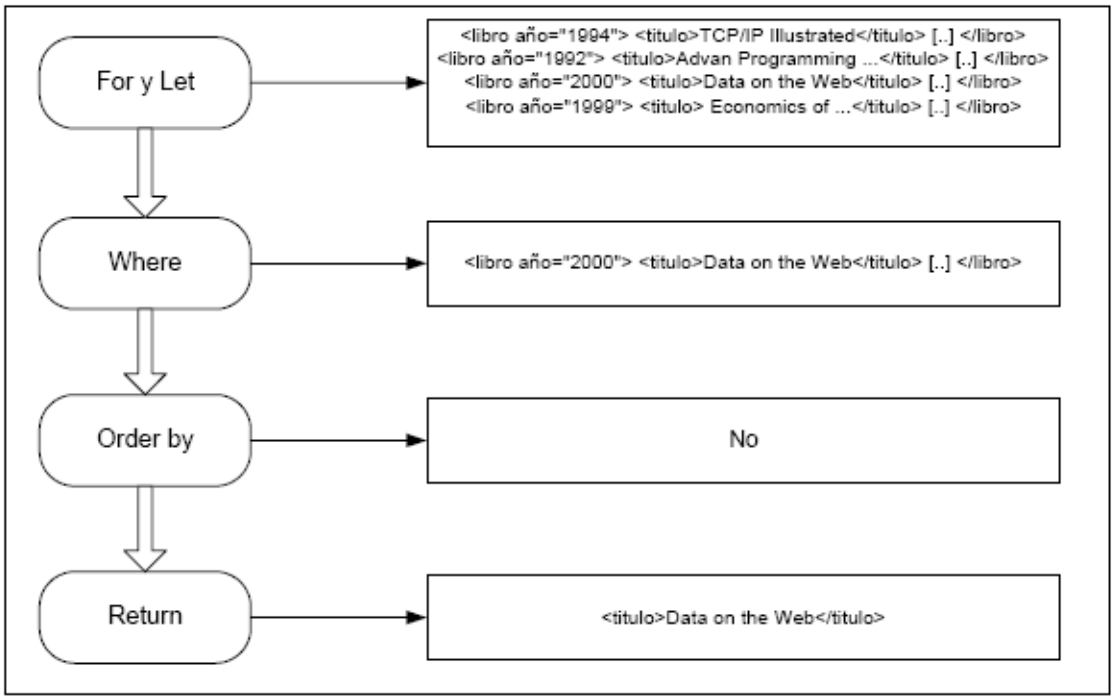

Figura 5-2: Ejecución de una consulta XQuery con los resultados de cada cláusula.

5.3.1 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’.

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.

5.3.2 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>Advance 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>Advance 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”.

```
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>
```

5.3.3 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 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.

5.3.4 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.".

```
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.

5.3.5 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 “programming” 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,"programming"))

return $b/title
```

Otro ejemplo. La siguiente consulta devuelve el título de todos los libros que mencionen “programming” 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,"programming")

return $b/title
```

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

5.3.6 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.

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(), 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. 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>Advance       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.

5.3.7 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 :)
```