1. Introducción.

Extensible Markup Language(más conocido como XML) es un formato para almacenar datos extremadamente versátil, utilizado para representar una gran cantidad distinta de información como, por ejemplo, páginas web (XHTML), mensajes web entre aplicaciones (SOAP), libros y artículos, datos de negocio, representación de bases de datos relacionales, interfaces de usuario (XUL), transacciones financieras y bancarias, partidas de ajedrez, recetas de cocina, trazas del sistema, gráficos vectoriales (SVG), sindicación de contenidos (RSS), archivos de configuración, documentos de texto (OpenOffice.org) y manuscritos en griego clásico.

Un conjunto de datos expresados en XML es una cadena de texto donde cada uno de los datos está delimitado por etiquetas de la forma <T> ... </T> o se incluye como atributo de una etiqueta de la forma <T A="..."> ... </T>. Mediante esta representación es posible expresar también la semántica de cada uno de los datos.

A continuación se muestra un conjunto de datos en XML que contiene información sobre un libro titulado "TCP/IP Illustrated", escrito por "Stevens, W.", editado por "Addison-Wesley" y cuyo precio es 69.95.

```
<bib>
        <libro>
                <titulo>TCP/IP Illustrated</titulo>
                <autor>
                        <apellido>Stevens</apellido>
                        <nombre>W.</nombre>
                </autor>
                <editorial>Addison-Wesley</editorial>
                <precio> 65.95</precio>
        </libro>
</bib>
```

En XML las etiquetas se estructuran en forma de árbol n-ário. En la figura 1 se muestra la estructura en árbol del conjunto de datos referidos al libro.

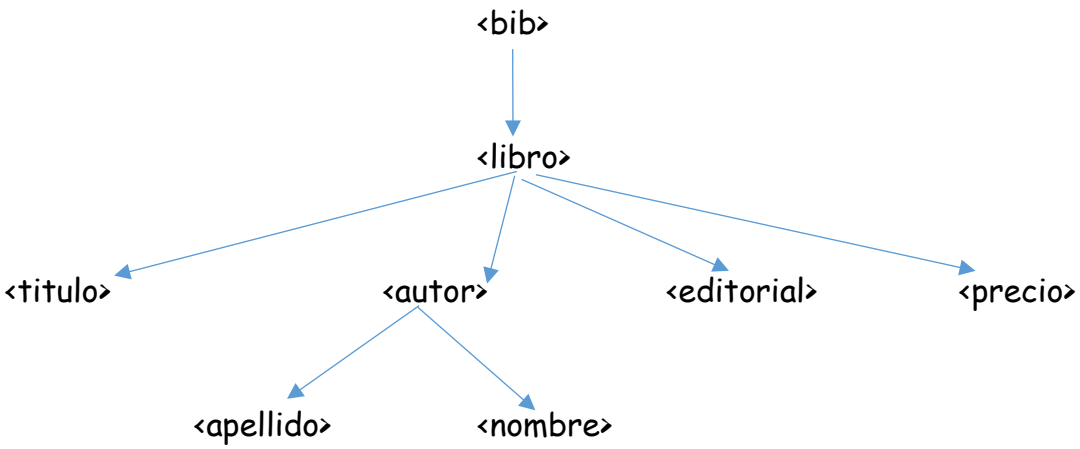

Figura 1. Representación en forma de árbol de un conjunto de etiquetas XML.

1.1. ¿Por qué necesitamos 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 DOM1, 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 parser DOM, o detallar una lista de acciones según la etiqueta que se encuentre para parser SAX.

Posteriormente a los parsers DOM y SAX, ha aparecido una nueva familia de herramientas conocidas genéricamente como bindings2, que, si bien en ciertos aspectos son tremendamente útiles y ahorran una gran cantidad de trabajo, no son útiles en todas las situaciones en las que se puede utilizar XML. Otra solución existente en la actualidad es el estándar XSLT[3] 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, binding y 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.
Dar solución a este escenario de uso mediante código escrito con la ayuda de un parser SAX es rápido y sencillo de desarrollar si la consulta y la estructura de la información no tienen una gran complejidad. A medida que la consulta va ganando en complejidad, es necesario definir comportamientos más complejos a ejecutar cuando se encuentre una etiqueta y se necesita un mayor número de almacenamientos temporales de datos. Otro problema es que el código está fuertemente ligado a la estructura de los datos en XML, por lo que cualquier cambio en la consulta o en la estructura de los datos obliga a volver a escribir el código. Además el código resultante es muy poco parametrizable y reutilizable por lo que para desarrollar una consulta similar habría que volverla a escribir desde el principio, sin poder aprovechar nada de lo escrito. Dar solución a este escenario de uso mediante código escrito con la ayuda un parser DOM añade, a los problemas de reusabilidad, complejidad y fuerte acoplamiento anteriores, la exigencia de tener cargar los datos en memoria antes de comenzar el recorrido, lo cual puede resultar imposible para grandes volúmenes de datos.

Las herramientas de bindings favorecen la sencillez en el proceso de carga en memoria de los datos, pero, aplicadas a este escenario de uso, no facilita la tarea a la hora de escribir una consulta. Primero, muchos bindings necesitan los documentos de definición de los documentos XML (DTDs o XML-Schemas) para poder generar las clases y el código de vinculación, por lo que si no se tienen y la estructura de la información es compleja puede resultar que el tiempo que ganamos con el uso de estas herramientas lo perdamos escribiendo el DTD o XML-Schemas correspondiente.

También existen herramientas que pueden trabajar sin los DTD o XML-Schemas, pero entonces es necesario escribir a mano todas las clases

necesarias para almacenar la colección de datos en XML. Además estas herramientas siguen teniendo el problema de que requieren cargar en memoria todos los datos. Más aún, lo que estamos haciendo es cambiar un conjunto de datos expresados mediante etiquetas XML por un conjunto de datos expresados por los atributos de un conjunto de objetos, por lo que sigue siendo necesario escribir la algoritmia necesaria para esa consulta, algoritmia poco reutilizable en posteriores consultas.

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 artículo vamos a mostrar, con profusión de ejemplos, las características principales de este lenguaje y vamos a mostrar como utilizar uno de los motores open-source existentes para incorporar capacidad de procesado de consulta XQuery en nuestras aplicaciones.

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 nário 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. Actualmente, **XQuery es un conjunto de borradores en el que trabaja el grupo W3C**. Sin embargo, a pesar de no tener una redacción definitiva ya existen o están en proceso numerosas implementaciones de motores y herramientas que lo soportan.

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.

- Cuando se va a analizar un documento XML, se crea un árbol de nodos del mismo.
- Ese árbol tiene un elemento raíz y una serie de hijos. Los hijos del nodo raíz pueden tener más hijos.
- Si repetimos ese proceso llegará un momento en el que el último nodo no tiene ningún hijo, lo que se denomina nodo hoja.
- Los tipos de nodos se puede encontrar en ese recorrido son los siguientes:
    - **Nodo raíz o "1":** Es el primer nodo del documento XML.
    - **Nodo elemento:** Cualquier elemento de un documento XML es un nodo elemento en el árbol. El nodo raíz es un caso especial de Nodo elemento (no tiene padre). Cada nodo elemento posee un padre y puede o no poseer hijos. En el caso que no tenga hijos, sería un nodo hoja.
    - **Nodo texto:** Cualquier elemento del documento que no esté marcado con una etiqueta de la DTD del documento XML.
    - **Nodo atributo:** Un nodo elemento puede tener etiquetas que complementen la información de ese elemento. Eso sería un nodo atributo.

La extracción de la información durante el recorrido del árbol será tan simple como la detección de los nodos a buscar y el procesamiento de la información que se quiere extraer de ese nodo concreto.
Esto que parece tan complicado, se realiza fácilmente con Xpath.
XPath es la herramienta que utiliza XQuery para procesar el árbol de nodos de un documento XML.

XQuery Reglas de sintaxis básica

- XQuery distingue entre mayúsculas y minúsculas
- Elementos de XQuery, atributos y variables deben ser nombres XML válidos
- Un valor de cadena de XQuery puede estar entre comillas simples o dobles
- Una variable de XQuery se define con un $ seguido de un nombre, por ejemplo, $librería
- Los comentarios están delimitados por (: y :), por ejemplo, (: Esto es un comentario :)

Para los ejemplos utilizaremos el fichero **Libros.xml**

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"?>
<Libros>
        <libro año="2000">>
                <Cod_Libro>1</Cod_Libro>
                <Titulo>Don Quijote de la Mancha </Titulo>
                <Autores>
                        <autor>
                                <Cod_Autor>1 </Cod_Autor>
                                <Nombre> Miguel</Nombre>
                                <Apellidos>de Cervantes Saavedra</Apellidos>
                                <Fecha_Nacimiento>29/09/1547</Fecha_Nacimiento>
                        </autor>
                </Autores>
                <Editorial>Juan de la Cuesta</Editorial>
                <Edicion>3</Edicion>
                <ISBN>9788466745840</ISBN>
                <NumPaginas>416</NumPaginas>
```

```
</libro>



<libro año="1995">>
        <Cod_Libro>2</Cod_Libro>
        <Titulo>La Celestina </Titulo>
        <Autores>
                <autor>
                        <Cod_Autor>2</Cod_Autor>
                        <Nombre>Fernando</Nombre>
                        <Apellidos>de Rojas  </Apellidos>
                        <Fecha_Nacimiento>01/01/1470</Fecha_Nacimiento>
                </autor>
        </Autores>
        <Editorial>Maxtor</Editorial>
        <Edicion>1</Edicion>
        <ISBN>9788471664938</ISBN>
        <NumPaginas>320</NumPaginas>
</libro>

<libro año="2001">
        <Cod_Libro>3</Cod_Libro>
        <Titulo>Rimas y Leyendas</Titulo>
        <Autores>
                <autor>
                        <Cod_Autor>3</Cod_Autor>
                        <Nombre>Gustavo Adolfo</Nombre>
                        <Apellidos>Bécquer</Apellidos>
                        <Fecha_Nacimiento>17/02/1836</Fecha_Nacimiento>
                </autor>
        </Autores>
        <Editorial>Cátedra</Editorial>
        <Edicion>4</Edicion>
        <ISBN>9788437620244</ISBN>
        <NumPaginas>217</NumPaginas>
</libro>
```

```
        <libro año="2000">
                <Cod_Libro>4</Cod_Libro>
                <Titulo>Los Misterios del Sol</Titulo>
                <Autores>
                        <autor>
                                <Cod_Autor>4</Cod_Autor>
                                <Nombre>Luis</Nombre>
                                <Apellidos>Negrillo Gallardo</Apellidos>
                                <Fecha_Nacimiento>17/08/1983</Fecha_Nacimiento>
                        </autor>
                        <autor>
                                <Cod_Autor>5</Cod_Autor>
                                <Nombre>Carlos</Nombre>
                                <Apellidos>Negrillo Gallardo</Apellidos>
                                <Fecha_Nacimiento>17/08/1983</Fecha_Nacimiento>
                        </autor>
                </Autores>
                <Editorial>Cátedra</Editorial>
                <Edicion>6</Edicion>
                <ISBN>1118437620244</ISBN>
                <NumPaginas>423</NumPaginas>
        </libro>
</Libros>
```

Lectura de archivos

Con la función doc () se lee un documento XML que se indique como parámetro y devuelve el nodo raíz o los elementos que se indiquen mediante una expresión XPath

```
doc("Libros.xml")
```

La expresión siguiente se utiliza para seleccionar todos los elementos Titulo en el fichero "Libros.xml"

```
doc("Libros.xml")//Libros/libro/Titulo
```

El resultado sería el siguiente:

```
<Titulo>Don Quijote de la Mancha </Titulo>
<Titulo>La Celestina </Titulo>
<Titulo>Rimas y Leyendas </Titulo>
<Titulo>Los Misterios del Sol </Titulo>
```

XQuery utiliza filtros para limitar los datos extraidos de documentos XML.

```
doc("Libros.xml")//Libros/libro[NumPaginas>400]
```

El filtro anterior se utiliza para seleccionar todos los elementos *libro* bajo el elemento *Libros* que tienen un elemento de *NumPaginas* con un valor que es mayor que 400:

El resultado sería el siguiente:

```
<libro año="2000">
        <Cod_Libro>1</Cod_Libro>
        <Titulo>Don Quijote de la Mancha </Titulo>
        <Autores>
                <autor>
                        <Cod_Autor>1  </Cod_Autor>
                        <Nombre> Miguel</Nombre>
                        <Apellidos>de Cervantes Saavedra</Apellidos>
                        <Fecha_Nacimiento>29/09/1547</Fecha_Nacimiento>
                </autor>
        </Autores>
        <Editorial>Juan de la Cuesta</Editorial>
        <Edicion>3</Edicion>
        <ISBN>9788466745840</ISBN>
        <NumPaginas>416</NumPaginas>
    </libro>
    <libro año="2000">
                <Cod_Libro>4</Cod_Libro>
                <Titulo>Los Misterios del Sol</Titulo>
                <Autores>
                        <autor>
                                <Cod_Autor>4</Cod_Autor>
                                <Nombre>Luis</Nombre>
                                <Apellidos>Negrillo Gallardo</Apellidos>
                                <Fecha_Nacimiento>17/08/1983</Fecha_Nacimiento>
                        </autor>
                        <autor>
                                <Cod_Autor>5</Cod_Autor>
                                <Nombre>Carlos</Nombre>
                                <Apellidos>Negrillo Gallardo</Apellidos>
                                <Fecha_Nacimiento>17/08/1983</Fecha_Nacimiento>
                        </autor>
                </Autores>
                <Editorial>Cátedra</Editorial>
                <Edicion>6</Edicion>
                <ISBN>1118437620244</ISBN>
                <NumPaginas>423</NumPaginas>
        </libro>
```

Si sólo queremos los títulos de los libros que tienen menos de 400 páginas, la consulta sería la siguiente:

```
doc("Libros.xml")//Libros/libro[NumPaginas<400]//Titulo
```

El resultado de la consulta:

```
<Titulo>La Celestina</Titulo>
<Titulo>Rimas y Leyendas</Titulo>
```

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: Esta sentencia permite seleccionar los nodos que se quieren consultar, guardándose en la variable

Let: Permite declarar variables a las que se les asignan valores.

Where: Filtra las tuplas eliminando todos los valores que no cumplan las condiciones dadas.

Order by: Ordena las tuplas según el criterio dado.

Return: Devuelve los resultados

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

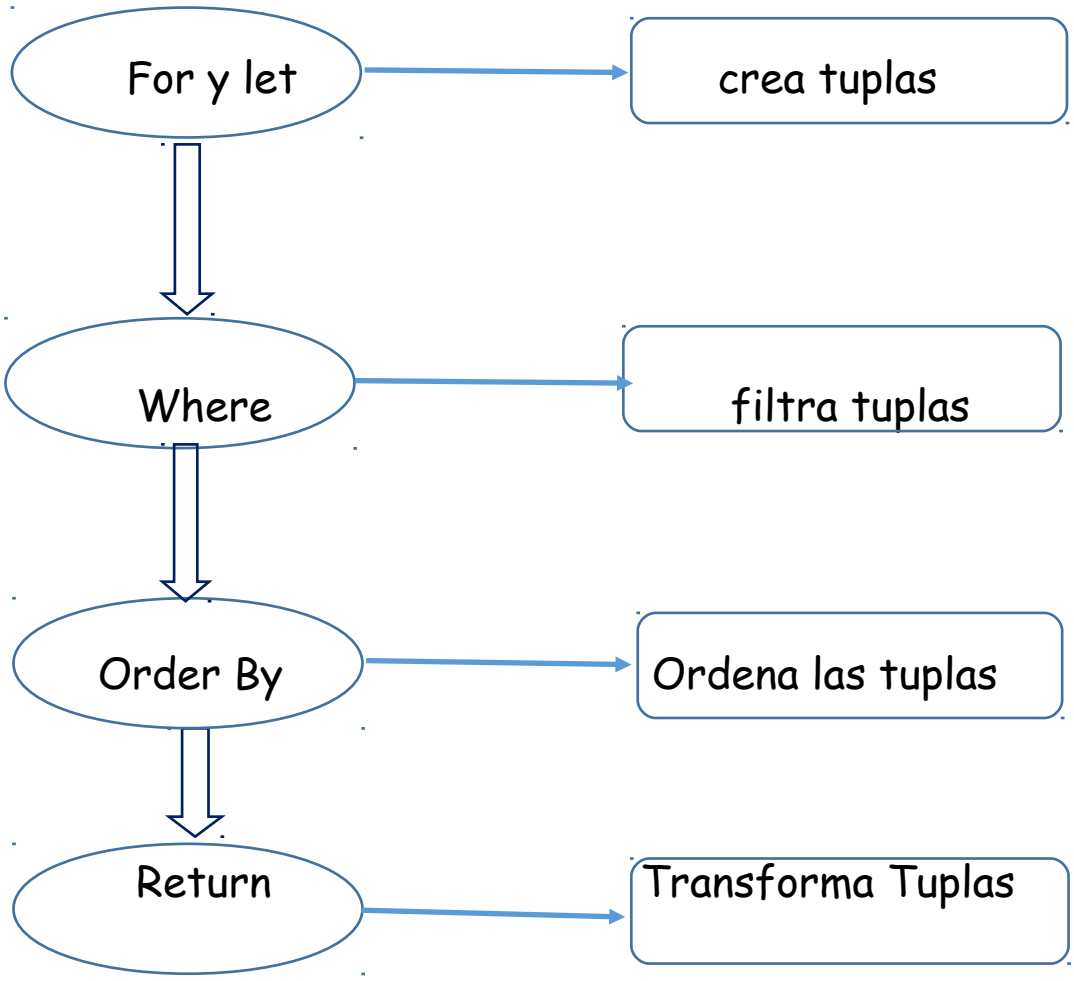

Figura 1. Orden de ejecución y resultado de las cinco cláusulas posibles.

Ejemplos de consultas.

Las siguientes consultas devuelven los mismos resultados

Visualizar todos los libros del fichero libros

- doc("Libros.xml")//Libros/libro
- for $x in doc("Libros.xml")//Libros
  return $x/libro

```
<libro año="2000">>
        <Cod_Libro>1</Cod_Libro>
        <Titulo>Don Quijote de la Mancha </Titulo>
        <Autores>
            <autor>
                <Cod_Autor>1 </Cod_Autor>
                <Nombre> Miguel</Nombre>
                <Apellidos>de Cervantes Saavedra</Apellidos>
                <Fecha_Nacimiento>29/09/1547</Fecha_Nacimiento>
            </autor>
        </Autores>
        <Editorial>Juan de la Cuesta</Editorial>
        <Edicion>3</Edicion>
        <ISBN>9788466745840</ISBN>
        <NumPaginas>416</NumPaginas>
    </libro>

    <libro año="1995">>
        <Cod_Libro>2</Cod_Libro>
        <Titulo>La Celestina </Titulo>
        <Autores>
            <autor>
                <Cod_Autor>2</Cod_Autor>
                <Nombre>Fernando</Nombre>
                <Apellidos>de Rojas  </Apellidos>
                <Fecha_Nacimiento>01/01/1470</Fecha_Nacimiento>
            </autor>
        </Autores>
        <Editorial>Maxtor</Editorial>
        <Edicion>1</Edicion>
        <ISBN>9788471664938</ISBN>
        <NumPaginas>320</NumPaginas>
    </libro>
```

```
<libro año="2001">
	<Cod_Libro>3</Cod_Libro>
	<Titulo>Rimas y Leyendas</Titulo>
	<Autores>
		<autor>
			<Cod_Autor>3</Cod_Autor>
			<Nombre>Gustavo Adolfo</Nombre>
			<Apellidos>Bécquer</Apellidos>
			<Fecha_Nacimiento>17/02/1836</Fecha_Nacimiento>
		</autor>
	</Autores>
	<Editorial>Cátedra</Editorial>
	<Edicion>4</Edicion>
	<ISBN>9788437620244</ISBN>
	<NumPaginas>217</NumPaginas>
</libro>
<libro año="2000">
	<Cod_Libro>4</Cod_Libro>
	<Titulo>Los Misterios del Sol</Titulo>
	<Autores>
		<autor>
			<Cod_Autor>4</Cod_Autor>
			<Nombre>Luis</Nombre>
			<Apellidos>Negrillo Gallardo</Apellidos>
			<Fecha_Nacimiento>17/08/1983</Fecha_Nacimiento>
		</autor>
		<autor>
			<Cod_Autor>5</Cod_Autor>
			<Nombre>Carlos</Nombre>
			<Apellidos>Negrillo Gallardo</Apellidos>
			<Fecha_Nacimiento>17/08/1983</Fecha_Nacimiento>
		</autor>
	</Autores>
	<Editorial>Cátedra</Editorial>
	<Edicion>6</Edicion>
	<ISBN>1118437620244</ISBN>
	<NumPaginas>423</NumPaginas>
</libro>
```

Visualizar el titulo de los libros que contengan menos de 400 páginas

- doc("Libros.xml")//Libros/libro[NumPaginas<400]/Titulo

- for $x in doc("Libros.xml")//Libros/libro
  where $x/NumPaginas<400
  return $x/Titulo

<Titulo>La Celestina </Titulo>

<Titulo>Rimas y Leyendas</Titulo>

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
```

<Titulo>Don Quijote de la Mancha </Titulo>
<Titulo>Los Misterios del Sol</Titulo>

Para devolver más de un resultado:

Ejemplo: Visualizar la editorial y el titulo de todos los libros del fichero libros

```
for $x in doc("Libros.xml")//Libros/libro
let $b:=$x/Editorial
let $c:=$x/Titulo
return<Salida>
            {$b}
            {$c}
        </Salida>
```

```
<Salida>
        <Editorial>Juan de la Cuesta</Editorial>
        <Titulo>Don Quijote de la Mancha </Titulo>
</Salida>
<Salida>
        <Editorial>Maxtor</Editorial>
        <Titulo>La Celestina </Titulo>
</Salida>
<Salida>
        <Editorial>Cátedra</Editorial>
        <Titulo>Rimas y Leyendas</Titulo>
</Salida>
<Salida>
        <Editorial>Cátedra</Editorial>
        <Titulo>Los Misterios del Sol</Titulo>
</Salida>
```

Los resultados de las consultas salen siempre con las etiquetas XML, si queremos obtener un listado sin las etiquetas, utilizaremos la función data().

```
for $x in doc("Libros.xml")//Libros/libro
where $x/NumPaginas<400
return data($x/Titulo)
```

```
La celestina
Rimas y Leyendas
```

Podemos unir dentro de XQuery etiquetas XHTML y expresiones o clausulas FLWOR, pero la parte que se procesa como consulta debe ir obligatoriamente entre { }

Ejemplo:

```
<html>
<body>
    <ul>
        {for $x in doc("Libros.xml")//Libros/libro/Titulo
        order by $x
        return <li>{data($x)}</li>
        }
    </ul>
</body>
</html>
```

La salida sería la siguiente:

```
<html>
<body>
<ul>
    <li>Don Quijote de la Mancha </li>
    <li>La Celestina </li>
    <li>Los Misterios del Sol</li>
    <li>Rimas y Leyendas</li>
</ul>
</body>
</html>
```

Aparte de las consultas, en las bases de datos podemos realizar las siguientes operaciones:

- **Insertar**. La inserción se realizará del siguiente modo:

```
insert node
<libro año="2003">
<Cod_Libro>4</Cod_Libro>
<Titulo>La Familia de Pascual Duarte </Titulo>
<Autores>
   <autor>
        <Cod_Autor>4</Cod_Autor>
        <Nombre>Camilo José</Nombre>
        <Apellidos>Cela  </Apellidos>
        <Fecha_Nacimiento>11/05/1916</Fecha_Nacimiento>
   </autor>
</Autores>
<Editorial>Austral</Editorial>
<Edicion>8</Edicion>
<ISBN>9788471664944</ISBN>
<NumPaginas>267</NumPaginas>
 </libro>
  as last into doc("Libros.xml")//Libros
```

- Sentencia para insertar al principio de la base de datos:

```
as first into doc(Libros.xml)//Libros
```

- Sentencia para insertar al final de la base de datos:

```
as last into doc(Libros.xml)//Libros
```

Ejercicios

1. **Tenemos el siguiente fichero XML (bailes.xml) que almacena información sobre una academia de bailes de salón. De cada uno de los bailes se almacena la siguiente información:**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bailes>
   <baile id="1">
         <nombre>Tango</nombre>
         <precio cuota="mensual" moneda="euro">27</precio>
         <plazas>20</plazas>
         <comienzo>1/1/2012</comienzo>
         <fin>30/6/2012</fin>
         <profesor>Roberto García</profesor>
         <sala>1</sala>
   </baile>
   <baile id="2">
         <nombre>Cha-cha-cha</nombre>
         <precio cuota="trimestral" moneda="euro">80</precio>
         <plazas>18</plazas>
         <comienzo>1/2/2012</comienzo>
         <fin>31/7/2012</fin>
         <profesor>Mirian Gutierrez</profesor>
         <sala>1</sala>
   </baile>
   <baile id="3">
         <nombre>Rock</nombre>
         <precio cuota="mensual" moneda="euro">30</precio>
         <plazas>15</plazas>
         <comienzo>1/1/2012</comienzo>
         <fin>1/12/2012</fin>
         <profesor>Laura Romero</profesor>
         <sala>1</sala>
   </baile>
```

```
    <baile id="4">
        <nombre>Merenge</nombre>
        <precio cuota="trimestral" moneda="dolares">75</precio>
        <plazas>12</plazas>
        <comienzo>1/1/2012</comienzo>
        <fin>1/12/2012</fin>
        <profesor>Jesús Lozano</profesor>
        <sala>2</sala>
    </baile>
    <baile id="5">
        <nombre>Salsa</nombre>
        <precio cuota="mensual" moneda="euro">32</precio>
        <plazas>10</plazas>
        <comienzo>1/1/2012</comienzo>
        <fin>30/6/2012</fin>
        <profesor>Jesús Lozano</profesor>
        <sala>2</sala>
    </baile>
    <baile id="6">
        <nombre>Pasodoble</nombre>
        <precio cuota="anual" moneda="euro">32</precio>
        <plazas>8</plazas>
        <comienzo>1/1/2012</comienzo>
        <fin>30/12/2012</fin>
        <profesor>Mirian Gutierrez</profesor>
        <sala>2</sala>
    </baile>
</bailes>
```

Construir el DTD y el esquema correspondiente.

2. **Realizar las siguientes consultas:**

a) Se necesita saber los nombres de los bailes que se realizan en la sala número 1.
b) Mostrar el mismo resultado sin mostrar las etiquetas <nombre>.
c) Se necesita extraer los nodos de aquellos bailes que se impartan en la sala 2 y cuyo precio sea menor que 35 euros.
d) Se necesita saber el nombre de los profesores que dan clases con cuotas mensuales.
e) Realizar una consulta cuyo resultado sea una tabla XHTML que nos muestre el nombre del baile, el profesor que lo imparte y el número de plazas ofertadas.
   Los mismos resultados ordenados por nombre
f) Queremos realizar la misma consulta anterior pero estableciendo la condición de ser bailes con cuota trimestral. Además se ordenará, de menor a mayor, los bailes según el número de plazas disponibles.
g) Añadir dos nuevos bailes a la base de datos.
h) En uno de los nodos anteriormente añadidos cambiar el nombre del profesor y en número de plazas ofertadas.
i) Borrar el nodo anteriormente añadido.
j) Realizar una copia de seguridad del fichero bailes.xml