5.3. Manipulación de información en documentos XML

Índice

Objetivos

Con esta unidad perseguimos los siguientes objetivos:

- Conocer cómo podemos **actualizar los datos en fichero XML.**
- Conocer las **herramientas** de gestión de Bases de Datos XML.
- Aprender a realizar consultas con **XQuery.**
- Conocer **XUpdate** y su funcionamiento.
- Conocer **XQuery Update Facility** y su funcionamiento.

Introducción

Conceptos

Tener una base de datos basada en registros XML permite tener gran cantidad de datos **accesibles en cualquier momento**, pero los datos son algo vivo y evolucionan con el tiempo, lo cual nos lleva a la **necesidad de poder modificarlos**.

Como hemos definido anteriormente la modificación de documentos XML se puede realizar de dos formas:

- Usando las API de manipulación SAX y DOM. La primera (SAX) permite recorrer y por lo tanto consultar los datos que se almacenan en el XML. La segunda es (DOM) que permite no solo leer el documento sino insertar información en él. Como DOM trabaja a nivel de árbol en memoria, debe existir un Parser que transforme el árbol en memoria de nuevo a un fichero XML.
- Usando lenguajes de manipulación. Los lenguajes que vamos a revisar en este documento son los que tienen entre sus características la de poder modificar documentos XML. Estos lenguajes son: **XUpdate y XQuery Update Facility.**

Una posibilidad para la recuperación de un documento y su modificación es mediante un programa específico (a través de algún API como SAX ó DOM) y sustituir al documento original; esta solución es muy costosa y nada flexible ya que tendríamos que generar un programa para cada consulta. Esto hace evidente la necesidad de lenguajes que permitan modificar el contenido del mismo de manera declarativa.

Búsqueda de la información (XQUERY)

Técnicas de búsqueda en XML

Cuando trabajamos con bases de datos, la búsqueda es una de las funciones fundamentales. Por lo tanto los sistemas de bases de datos que almacenan documentos XML deben de incluir esa misma funcionalidad. La forma de abordar esa función en bases de datos XML dependerá del tipo de base de datos que sea.

Las bases de datos en XML **son herramientas para almacenar y recuperar la información** cuya base son documentos XML.

- Las bases de datos deben mostrar la información que almacena sobre un tema y en qué documento se encuentra dicha información.
- Estas herramientas proporcionan la referencia, la versión electrónica de la información o bien el propio conjunto de datos.

La problemática existente en estas bases de datos es cómo buscar en ellas. No porque sea especialmente difícil, sino porque no hay un sistema de interrogación común para todas ellas.

Para poder buscar de una forma metódica en una base de datos basada en XML necesitamos poder contar con una serie de herramientas que el sistema gestor implemente.

XQuery

XQuery es un lenguaje diseñado para escribir consultas sobre colecciones de datos expresadas en XML.

- Puede aplicarse tanto a archivos XML, como a 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 de etiquetas XML.
- En este sentido XQuery es independiente del origen de los datos. Este lenguaje se revisa en las siguientes lecciones.

XPath

XPath es un lenguaje estándar y muy importante en XSLT.

- XPath se puede utilizar para navegar a través de elementos y atributos en un documento XML.
- Tiene una sintaxis para definir partes de un documento XML.
- Utiliza expresiones de ruta para navegar en documentos XML.
- Contiene una biblioteca de funciones estándar.
- XPath es una recomendación del W3C.

XQuery

La realización de consultas en bases de datos XML permite acceder a la información a través de un lenguaje común que tiene semejanzas con otros lenguajes de consultas como SQL. A través de dicho lenguaje los elementos y datos que conforman cualquier documento XML bien formado son accesibles.

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, conforme más se utiliza en proyectos complejos y se trabaja con mayor cantidad de datos, se comprueba que las herramientas usadas hasta ahora que pueden manipular los documentos XML con su conjunto de datos, **los parsers SAX y DOM**, no son buenas soluciones para manejar colecciones de datos en XML grandes y complejas. **Estas herramientas permiten a muy bajo nivel manejar los datos pero generan mucho código y no son muy manejables a no ser que sean manejados por un software a medida.**

Otra solución existente es el **estándar XSLT,** que permite definir transformaciones sobre colecciones de 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 se quiera obtener una representación distinta de los datos, no cuando se desea localizar una información concreta dentro de dicho conjunto de datos.

Por todo ello, es necesario pensar en **un lenguaje que defina de forma sencilla 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**, por lo tanto independientemente de la forma en que se realice el recorrido o de dónde se encuentren los datos. **Este lenguaje ya existe y se llama XQuery.**

XQuery es a XML lo que es SQL a las bases de datos relacionales.

XQuery es un lenguaje para encontrar y extraer elementos y atributos de documentos XML.

XQuery 1.0 y XPath 2.0 comparten el mismo modelo de datos y admiten las mismas funciones y operadores.

XQuery, características

Un lenguaje procidemental clásico ejecuta una lista de comandos. XQuery es un **lenguaje funcional,** donde **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 para crear nuevas expresiones más complejas y de mayor potencia semántica.

Características

- **XQuery es un lenguaje declarativo.** Al igual que SQL hay que indicar qué se quiere, no cómo se puede obtener.
- **XQuery es 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 **respetan el modelo de datos XML.**
- Las consultas y los resultados ofrecen **soporte para los namespace.**
- XQuery es capaz de **soportar XML-Schemas y DTDs.**
- **XQuery trabaja con independencia de la estructura del documento**, esto es, sin necesidad de conocerla.
- XQuery soporta tanto **tipos simples;** como enteros y cadenas, y tipos complejos; como un nodo compuesto por varios nodos hijos.
- Las consultan soportan **cuantificadores** universales (para todo) y existenciales (existe).
- Las consultas soportan operaciones sobre **jerarquías de nodos y secuencias de nodos.**
- En una consulta se puede combinar información de **múltiples fuentes.**
- Las consultas son capaces de **manipular** los datos independientemente del origen de estos.
- Mediante XQuery es posible definir consultas que **transformen las estructuras** de información originales y debe ser posible crear nuevas estructuras de datos.
- El lenguaje de consulta **es independiente de la sintaxis**, esto es, debe ser posible que existan varias sintaxis distintas para expresar una misma consulta en XQuery.

XQuery, 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 definen 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.

XQuery, consultas

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.

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, 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.*

Ejemplo consultas XQuery con BaseX

Realicemos un pequeño ejemplo de uso utilizando XQuery. Para realizar las consultas instalaremos el editor BaseX.

Base X
BaseX es muy liviano, fácil de usar y se ejecuta de forma automática.
http://basex.org/download/

Según el siguiente documento XML:

```
<bib>
	<libro año="1994">
		<titulo>Redes TCP/IP</titulo>
		<autor>
			<apellido>Pérez</apellido>
			<nombre>Juan Carlos</nombre>
		</autor>
		<editorial>Addison-Wesley</editorial>
		<precio> 49.80</precio>
	</libro>
	<libro año="1992">
		<titulo>Programación Java</titulo>
		<autor>
			<apellido>Pérez</apellido>
			<nombre>Juan Carlos</nombre>
		</autor>
		<editorial>Addison-Wesley</editorial>
		<precio>49.80</precio>
	</libro>
	<libro año="2000">
		<titulo>Manejo de datos</titulo>
		<autor>
			<apellido>Ramírez</apellido>
			<nombre>Julio</nombre>
		</autor>
		<autor>
			<apellido>Bueno</apellido>
			<nombre>Federico</nombre>
		</autor>
		<autor>
			<apellido>Gálvez</apellido>
			<nombre>Jesús</nombre>
		</autor>
		<editorial>Anaya Mutimedia</editorial>
		<precio>39.95</precio>
	</libro>
	<libro año="1999">
		<titulo> Programación Python</titulo>
		<autor>
			<apellido>Palomo</apellido>
			<nombre>Roberto</nombre>
		</autor>
		<autor>
			<apellido>Otero</apellido>
			<nombre>Julio</nombre>
		</autor>
		<autor>
			<apellido>Alvarez</apellido>
			<nombre>Juan</nombre>
		</autor>
		<editor>
```

```
                <apellido>Marchenado</apellido>
                <nombre>Félix</nombre>
                <afiliacion>CITI</afiliacion>
            </editor>
            <editorial>Editorial S.M.</editorial>
            <precio>129.95</precio>
        </libro>
</bib>
```

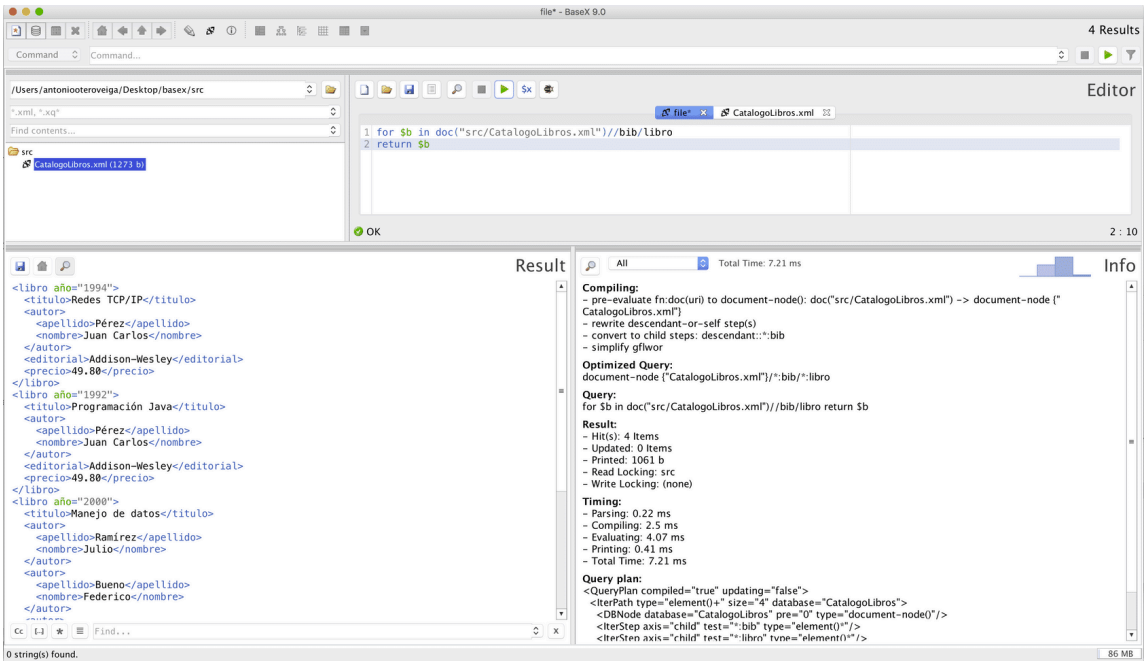
Contenido del archivo de XML de ejemplo

Si quieres probar este ejemplo, solo tienes que instalar BaseX en tu ordenador y guardar en la carpeta *src* del programa, el contenido del XML en un archivo llamado "CatalogoLibros.xml".

En el siguiente ejemplo de cláusula ***for***, la variable ***$b*** tomará como valor cada uno de los nodos *libros* que contenga en el archivo "CatalogoLibros.xml" (mostrado en el ejemplo anterior). Cada uno de esos nodos *libros*, será una tupla vinculada a la variable ***$b***.

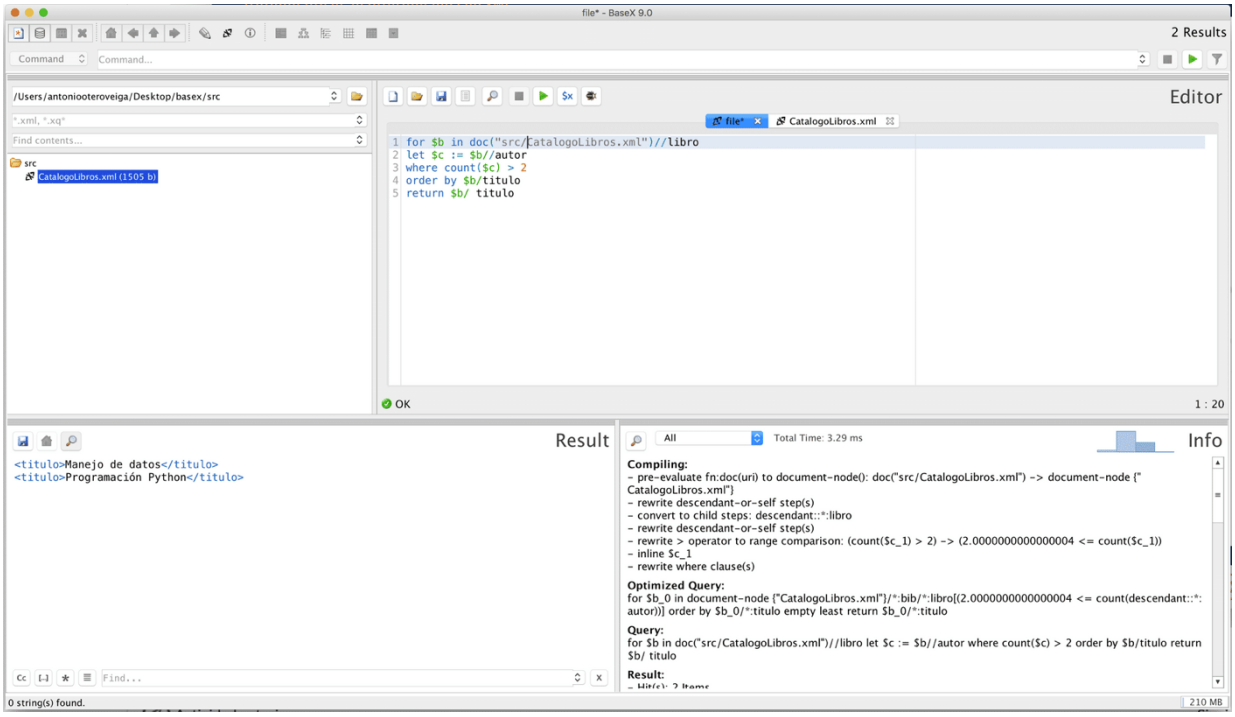
```
for $b in doc("src/CatalogoLibros.xml")//bib/libro
return $b
```

Cláusula *for*



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

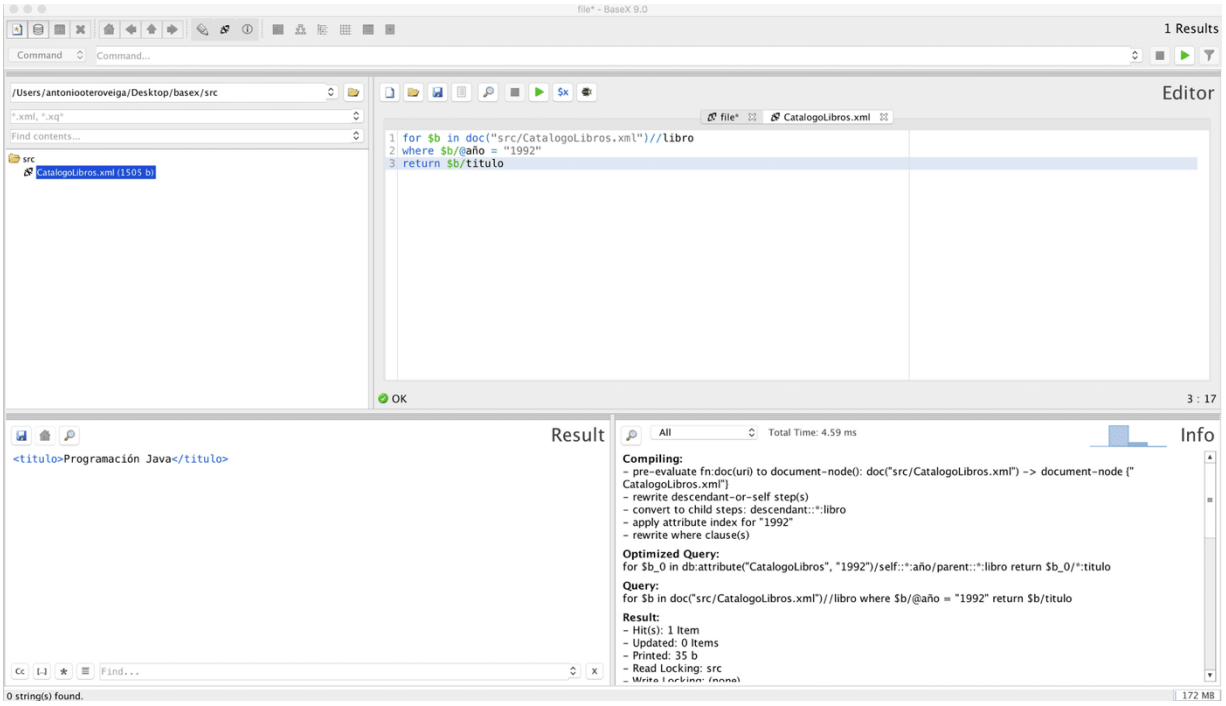
```
for $b in doc("src/CatalogoLibros.xml")//libro
let $c := $b//autor
where count($c) > 2
order by $b/titulo
return $b/ titulo
```



Una expresión *FLWOR* vincula variables a valores con cláusulas *for* y *let* y maneja 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 1992. Como "año" es un atributo y no una etiqueta se le antecede con un carácter "@".

```
for $b in doc("src/CatalogoLibros.xml")//libro
where $b/@año = "1992"
return $b/titulo
```



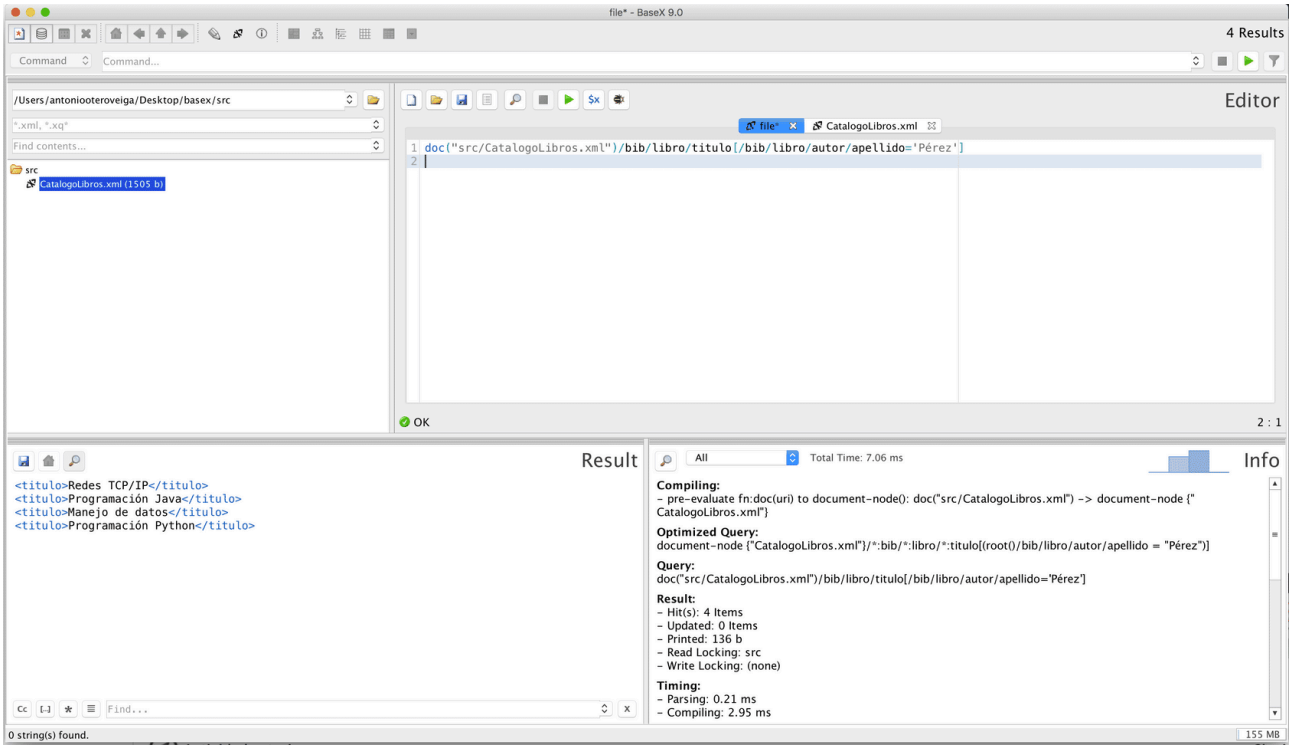
XQuery, reglas

A continuación, vamos a revisar las reglas que son de obligado cumplimiento para conformar 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*.
- Las cláusulas ***FLWOR*** no son obligatorias 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("src/CatalogoLibros.xml")/bib/libro/titulo[/bib/libro/autor/apellido='Pérez']
```

Esta expresión **XPath**, que también es una consulta XQuery válida, devuelve los títulos de los libros que tengan algún autor de apellido "Pérez". Es posible especificar varios criterios de ordenación en la cláusula *order by*, separándolos por comas.



Ejemplo de aplicación de doc*("src/CatalogoLibros.xml")/bib/libro/titulo[/bib/libro/autor/apellido='Pérez']* en BaseX.

Diferencias entre la cláusula *for* y la cláusula *let*

Revisemos la diferencia entre una cláusula *for* y una cláusula *let*. Ejecutamos una consulta que muestre los títulos de todos los libros almacenados en el archivo "CatalogoLibros.xml", primero con una cláusula *for* y, a continuación, con una cláusula *let* y vamos a detallar qué diferencia hay en la información obtenida.

Cláusula *for*

```
for $d in doc("CatalogoLibros.xml")/bib/libro/titulo return { $d }
```

Cláusula *For*

```
<titulos>
    <titulo>Redes TCP/IP</titulo>
</titulos>
<titulos>
    <titulo>Programación Java</titulo>
</titulos>
<titulos>
    <titulo>Manejo de datos</titulo>
</titulos>
<titulos>
    <titulo> Programación Python</titulo>
</titulos>
```

Resultado cláusula *For*

Cláusula *let*

```
let $d := doc("CatalogoLibros.xml")/bib/libro/titulo
return { $d }
```

Cláusula *let*

```
<titulos>
    <titulo>Redes TCP/IP</titulo>
    <titulo>Programación Java</titulo>
    <titulo>Manejo de datos</titulo>
    <titulo> Programación Python</titulo>
</titulos>
```

Resultado cláusula *let*

Examinando el resultado de los dos ejemplos anteriores podemos comprobar que, **aunque afecta a los mismos datos, la composición de los nodos es diferente.**

Tipos de nodos en XQuery

En XQuery, hay siete tipos de nodos: elemento, atributo, texto, espacio de nombres, nodo de instrucción de proceso, comentario y documento (raíz).

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
	<book>
		<title lang="en">Harry Potter</title>
		<author>J K. Rowling</author>
		<year>2005</year>
		<price>29.99</price>
	</book>
</bookstore>
```

Nodos

Al igual que en XML tenemos el nodo documento (raíz), nodos de elemento y nodos de atributo.

Padres

Cada elemento y atributo tiene un padre. El elemento *libro* es el padre de título, autor, año y precio.

Hermanos

Son los nodos que tienen el mismo padre. El título, autor, año y precio son todos hermanos.

Descendientes

Los descendientes de un nodo son los que contiene. Los descendientes de *librería* son el libro, el título, el autor, el año y el precio.

Valores atómicos

Una serie de datos indivisible, Ej.: J K. Rowling.

Hijos

Los nodos pueden tener cero, uno o más hijos. El título, el autor, el año y precio son todos hijos del elemento *libro*.

Antepasados

Los antepasados son el padre de un nodo, padre de los padres, etc. Los antepasados de *título* son el elemento *libro* y el elemento *librería*.

Cuantificadores existenciales

XQuery trabaja con dos tipos de cuantificadores existenciales llamados ***some*** y ***every.***

Some nos permite definir consultas que devuelvan **algún elemento** que satisfaga la condición y ***every*** consultas que devuelvan los elementos en los que **todos sus nodos** satisfagan la condición.

Por ejemplo, la siguiente consulta devuelve los títulos de los libros en los que **al menos uno** de sus autores es Juan Carlos Pérez.

```
for $b in doc("CatalogoLibros.xml")//libro
where some $a in $b/autor
satisfies ($a/last="Pérez" and $a/first="Juan Carlos")
return $b/titulo
```

Consulta usando *some*

El resultado sería:

```
<title>Redes TCP/IP</title>
<title>Programación Java</title>
```

Resultado de la consulta *some*

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 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(), startswith(), ends-with(), substring(), upper-case(), lower-case(), string().
- **Uso general:** distinct-values(), empty(), exits().

Unión

El operador **union (|)** 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("CatalogoLibros.xml")
//(autor | editor)/apellido)
order by $l
return <apellidos>{ $l }</apellidos>
```

Cláusula XQuery

```
<apellidos>Ramírez</apellidos>
<apellidos>Bueno</apellidos>
<apellidos>Marchenado</apellidos>
<apellidos>Pérez</apellidos>
<apellidos>Gálvez</apellidos>
```

Resultado

Except

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("CatalogoLibros.xml")//libro
where $b/titulo = "Redes TCP/IP"
return
<libro>
{ $b/@* }
{ $b/* except $b/precio }
</libro>
```

Cláusula XQuery

```
<libro year = "1994">
      <title>Redes TCP/IP</title>
      <author>
            <apellidos>Pérez</apellidos>
            <first>Juan Carlos</first>
      </author>
      <publisher>Addison-Wesley</publisher>
</libro>
```

Resultado

distinct-values()

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("CatalogoLibros.xml")//autor/apellidos)
return <apellidos>{ $l }</apellidos>
```

Cláusula XQuery

```
<apellidos>Pérez</apellidos>
<apellidos>Ramírez</apellidos>
<apellidos>Bueno</apellidos>
<apellidos>Gálvez</apellidos>
```

Resultado

empty()

La función **empty()** devuelve *true* si la expresión entre paréntesis está vacía.

Por ejemplo, la siguiente consulta devuelve todos los nodos *libro* que tengan al menos un nodo *autor*.

```
for $b in doc("CatalogoLibros.xml")//libro
where not(empty($b/autor))s
return $b
```

Cláusula XQuery

```
<libro año="1994">
	<titulo>Redes TCP/IP</titulo>
	<autor>
		<apellido>Pérez</apellido>
		<nombre>Juan Carlos</nombre>
	</autor>
	<editorial>Addison-Wesley</editorial>
	<precio> 49.80</precio>
</libro>
<libro año="1992">
	<titulo>Programación Java</titulo>
	<autor>
		<apellido>Pérez</apellido>
		<nombre>Juan Carlos</nombre>
	</autor>
	<editorial>Addison-Wesley</editorial>
	<precio>49.80</precio>
</libro>
```

Resultado

Manipulación de la información

XUpdate

XUpdate utiliza la sintaxis XML para realizar las modificaciones en un documento. Es un lenguaje para permitir la actualización de documentos XML.

Los requisitos planteados con este lenguaje son:

- Es **declarativo**, define cómo consultar y actualizar el contenido XML.
- Es simple y directo.
- Es compatible con XML Namespaces, XPath y XPointer.
- Permite extensiones futuras.
- Posee sintaxis XML.
- Es independiente de cualquier *parser* o modelo.
- Puede actualizar una parte o el documento completo.
- Puede actuar específicamente con el API XML:DB.

La estructura básica es

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
xmlns:xupdate="http://www.xmldb.org/xupdate">
...
...
...
</xupdate:modifications>
```

En la sentencia ***xupdate:modifications***, se pueden incluir:

xupdate:insert-before:
Inserta un nodo hermano precediendo al nodo de contexto.

xupdate:append:
Inserta un nodo como hijo del nodo de contexto.

xupdate:insert-after:
Inserta un nodo hermano a continuación del nodo de contexto.

xupdate:update:
Actualiza el contenido de los nodos seleccionados.

xupdate:remove:
Elimina los nodos seleccionados.

xupdate:rename:
Permite cambiar el nombre de un elemento o atributo.

Para acceder a los elementos y atributos se usa XPath.

Documentación
La misión de XUpdate es proporcionar un medio de actualización abiertas y flexibles para modificar los datos en documentos XML. http://xmldb-org.sourceforge.net/xupdate/index.html

Insertar un elemento

Veamos cómo insertar elemento mediante unos ejemplos:

Tenemos el siguiente XML:

```
<alumnos>
	<alumno>
		<nombre>Juan Palomo</nombre>
		<estudios>Grado superior de DAM</estudios>
	</alumno>
	<alumno>
		<nombre>María  Palomita</nombre>
		<estudios>Grado superior de DAW</estudios>
	</alumno>
	<alumno>
		<nombre>Roberto  Pomba</nombre>
		<estudios>Grado superior de DAM</estudios>
	</alumno>
</alumnos>
```

Si queremos insertar un nuevo elemento ***nota*** en el alumno María Palomita, la consulta XUpdate sería:

```
<xupdate:modifications version='1.0'
	xmlns:xupdate="http://www.xmldb.org/xupdate">
	<xupdate:insert-after select="/alumnos/alumno/nombre[.='María Palomita']">
		<xupdate:element name='nota'>Sobresaliente</xupdate:element>
	</xupdate:insert-after>
</xupdate:modifications>
```

Ejemplo de **xupdate:insert-after**

Esta consulta insertará el elemento **<nota>** después (**xupdate:insert-after**) del elemento **<nombre>** que tenga como valor 'María palomita'.

Realicemos una acción similar, pero insertando el elemento **antes (xupdate:insert-before)** del seleccionado con XPath. Para ello vamos insertar un elemento **<matrícula>**, antes del nodo **<nombre>** del elemento que estén realizando los estudios de DAM.

```
<xupdate:modifications version='1.0'
      xmlns:xupdate="http://www.xmldb.org/xupdate">
      <xupdate:insert-before select="/alumnos/alumno[estudios='Grado superior
DAM']/nombre">
            <xupdate:element name='matricula'>DS0001</xupdate:element>
      </xupdate:insert-before>
</xupdate:modifications>
```

Ejemplo de **xupdate:insert-before**

El resultado de estas modificaciones sería:

```
<alumnos>
      <alumno>
            <matricula>DS0001</matricula>
            <!--  Inserción  -->
            <nombre>Juan Palomo</nombre>
            <estudios>Grado superior de DAM</estudios>
      </alumno>
      <alumno>
            <nombre>María Palomita</nombre>
            <nota>Sobresaliente</nota>
            <!--  Inserción  -->
            <estudios>Grado superior de DAW</estudios>
      </alumno>
      <alumno>
            <nombre>Roberto Pomba</nombre>
            <estudios>Grado superior de DAM</estudios>
      </alumno>
</alumnos>
```

Resultado del documento después de aplicar los dos ejemplos planteados

Otra forma de realizar la inserción podía ser utilizando, **update:append** sobre el elemento alumno que insertará el elemento al final como último hermano. Igualmente podemos añadir atributos con **update:attribute**.

Modificar un elemento

Partiendo del ejemplo anterior, veamos cómo modificar un dato del XML con **xupdate:update.** Con la siguiente sentencia vamos a cambiar el nombre de los estudios del alumno "Juan Palomo" por "Graduado Superior en Desarrollo de Aplicaciones Multiplataforma."

```
<xupdate:modifications version='1.0' xmlns:xupdate="http://www.xmldb.org/xupdate">
  <xupdate:update select="/alumnos/alumno[nombre='Juan Palomo']/estudios">
   Graduado Superior en Desarrollo de Aplicaciones Multiplataforma
 </xupdate:update>
</xupdate:modifications>
```

Ejemplo de **xupdate:update**

El resultado sería:

```
<alumnos>
	<alumno>
		<matricula>DS0001</matricula>
		<!--  Inserción  -->
		<nombre>Juan Palomo</nombre>
		<estudios>Graduado Superior en Desarrollo de Aplicaciones
Multiplataforma</estudios>
		<!--  Modificación  -->
	</alumno>
	<alumno>
		<nombre>María Palomita</nombre>
		<nota>Sobresaliente</nota>
		<!--  Inserción  -->
		<estudios>Grado superior de DAW</estudios>
	</alumno>
	<alumno>
		<nombre>Roberto Pomba</nombre>
		<estudios>Grado superior de DAM</estudios>
	</alumno>
</alumnos>
```

Resultado del documento después de modificar el elemento.

Borrar un elemento

Para eliminar elementos podemos usar **xupdate:remove**. Por ejemplo si queremos borrar el primer elemento del XML:

```
<xupdate:modifications version='1.0' xmlns:xupdate="http://www.xmldb.org/xupdate">
     <xupdate:remove select='/alumnos/alumno[1]'/>
</xupdate:modifications>
```

Ejemplos de **xupdate:remove** para borrar un elemento

El resultado sería:

```
<alumnos>
	<alumno>
		<nombre>María Palomita</nombre>
		<nota>Sobresaliente</nota>
		<!--  Inserción  -->
		<estudios>Grado superior de DAW</estudios>
	</alumno>
	<alumno>
		<nombre>Roberto Pomba</nombre>
		<estudios>Grado superior de DAM</estudios>
	</alumno>
</alumnos>
```

Resultado después de eliminar el primer elemento.

Renombrar elementos

En algunos casos, no solo queremos modificar los datos o valores guardados en los elementos (nodos) de nuestros documentos. Sino que queremos cambiar el nombre (la clave) de dicho elemento, para ello podemos usar **xupdate:rename.**

Por ejemplo si queremos cambiar el nombre del nodo **<alumno>** por **<estudiante>**, pondremos

```
<xupdate:modifications version='1.0' xmlns:xupdate="http://www.xmldb.org/xupdate">
      <xupdate:rename select="/alumnos/alumno/"> Escritor </xupdate:rename>
</xupdate:modifications>
```

Ejemplo de **xupdate:rename**

Por lo que el resultado final sería:

```
<alumnos>
       <estudiante>
              <!--  Renombrado del nodo  -->
              <nombre>María Palomita</nombre>
              <nota>Sobresaliente</nota>
              <estudios>Grado superior de DAW</estudios>
       </estudiante>
       <!--  Renombrado del nodo  -->
       <estudiante>
              <!--  Renombrado del nodo  -->
              <nombre>Roberto Pomba</nombre>
              <estudios>Grado superior de DAM</estudios>
       </estudiante>
       <!--  Renombrado del nodo  -->
</alumnos>
```

Resultado de **xupdate:rename** sobre el nodo alumno

XQuery Update Facility

La principal carencia del lenguaje **XQuery** es la falta de definición de sentencias para la actualización de datos. El diseño de XQuery se ha realizado teniendo en cuenta esta necesidad, y **XQuery Update Facility 1.0** es una extensión del estándar para cubrirla.

XQuery Update Facility 1.0 divide las expresiones en tres tipos:

non-updating expression
Expresión XQuery normal.

Basic updating expression
Sentencia *insert*, *delete*, *replace*, *rename* o llamada a función de actualización.

Updating expression

Es una *Basic updating expression* o cualquier otra expresión que contiene una sentencia de actualización (por ejemplo una sentencia *FLWOR* con una sentencia *update* en el *return*).

Documentación

XQuery Update Facility 1.0. https://www.w3.org/TR/xquery-update-10/

Insert

Esta sentencia **inserta** copias de 0 o más nodos especificados en *origen* en las posiciones indicadas en *destino*.

Tanto *origen* como *destino* no pueden ser expresiones de actualización.

```
insert---node----"origen"----after--------------------"destino"
              |-nodes-|          |--before-----------------|
                                                |--------------------into--|
                                                |--as---first---|
                                                |-last--|
```

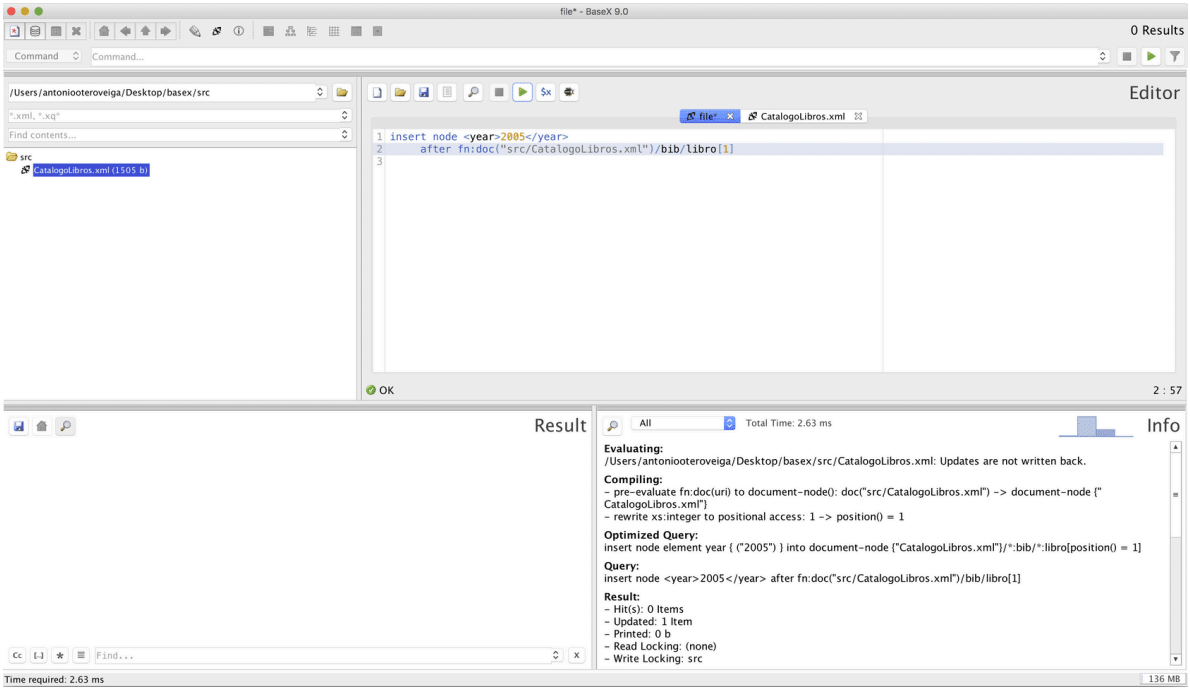
Sintaxis

Origen especifica los datos que vamos a insertar, bien nodos de una expresión bien un literal, *destino* especifica dónde queremos insertar los nodos de origen.

- **Node/nodes** se puede usar indistintamente.
- Con ***before***, el nodo se inserta como *preceding-sibling* del nodo destino, si se insertan varios su orden es mantenido.
- Con ***after***, el nodo se inserta como *following-sibling* del nodo destino, si se insertan varios su orden es mantenido.
- Con ***into*** el nodo es insertado como hijo del nodo destino, si se insertan varios su orden es mantenido.
- La posición de inserción será dependiente de la implementación.
- Si se añade ***as first***, el nodo(s) serán los primeros hijos del nodo destino.
- Si se añade ***as last***, el nodo(s) serán los últimos hijos del nodo destino.

```
insert node <year>2005</year>
     after fn:doc("bib.xml")/books/book[1]/publisher
insert node $new-police-report
     as last into fn:doc("insurance.xml")/policies
        /policy[id = $pid]
        /driver[license = $license]
        /accident[date = $accdate]
        /police-reports
```

Ejemplos insert



Ejemplo de aplicación con BaseX

Delete

Con esta sentencia borramos 0 o más nodos.

```
delete---node-----"destino"
              |-nodes-|
```

Sintaxis

```
delete node fn:doc("bib.xml")/books/book[1]/author[last()]
delete nodes /email/message
                          [fn:currentDate() - date > xs:dayTimeDuration("P365D")]
```

Ejemplos *delete*

Destino debe de ser una expresión de no-actualización y que tenga como resultado una secuencia de nodos.

Replace

Con esta sentencia reemplazamos una secuencia de 0 o más nodos. En este caso debemos tener en cuenta que podemos reemplazar bien el contenido del nodo bien el nodo mismo.

```
replace---node---"destino"--with---"expr.valor"
         |-value of-|
```

Sintaxis

```
replace node fn:doc("bib.xml")/books/book[1]/publisher
with fn:doc("bib.xml")/books/book[2]/publisher
replace value of node fn:doc("bib.xml")/books/book[1]/price
with fn:doc("bib.xml")/books/book[1]/price * 1.
```

Ejemplo *replace*

Tanto *destino* como *exp.valor* no pueden ser expresiones de actualización. Si no especificamos ***value of*** vamos a reemplazar un nodo, por lo que el nuevo nodo va a ocupar su posición en la jerarquía; esto implica que un nodo elemento puede ser reemplazado por otro nodo elemento, o un nodo atributo por un nodo atributo etc. Pero no podemos cambiar el tipo de nodo.

Rename

Se utiliza esta sentencia para cambiar la propiedad *name* con un nuevo valor.

```
rename node "destino" as "nombre"
```

Sintaxis

```
rename node fn:doc("bib.xml")/books/book[1]/author[1] as "principal-author"
```

Ejemplo rename

Despedida

Lecturas recomendas

A lo largo de esta unidad has podido ver algunos ejemplos de cláusulas XQuery y el uso de la herramienta BaseX.

Si quieres ver más ejemplos puedes visitar los siguientes enlaces.

Introducción a XQuery con ejemplos
Uso de BaseX y ejemplos de aplicación paso a paso.
https://www.adictosaltrabajo.com/tutoriales/introduccion-x-query/

Ejercicios prácticos de XQuery
Resolución de casos de búsqueda en XQuery.
https://www.tutorialspoint.com/xquery/index.htm

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

En esta lección hemos visto los conceptos sobre los que se basa el almacenamiento de datos con el uso de XML. Desde sus herramientas a los lenguajes que utilizamos para su manipulación o consulta.

Con XQuery hemos realizado consultas en bases de datos XML, pudiendo filtrar los resultados como hacemos en bases de datos relacionales con SQL. Además gracias a Xupdate, también podemos modificar nuestros documentos, sin tener que utilizar APIs de lenguajes externos.