Acceso a Datos

UT7 – BASES DE DATOS XML.

LENGUAJE XQUERY

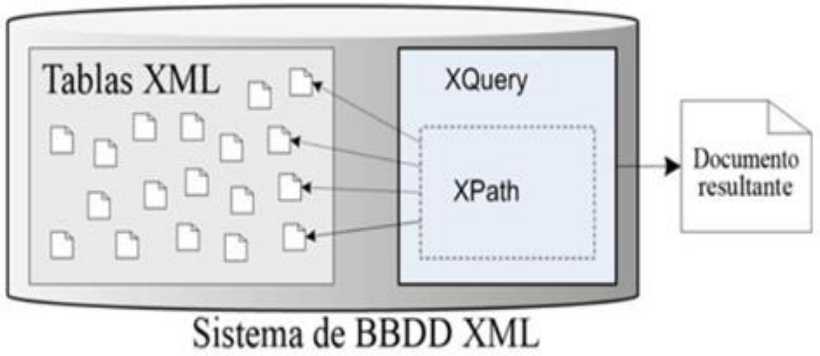
1. XQuery

XQuery, también conocido como XML Query, es un lenguaje creado para buscar y extraer elementos y atributos de documentos XML.

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.

1. XQuery

XQuery hace uso de XPath, por lo tanto, comparten el mismo modelo de datos y soportan las mismas funciones y operadores.



(Para realizar las consultas XQuery utilizaremos el XML *libros.xml*)

2. Consultas FLWOR

FLWOR son las siglas de For, Let, Where, Order by y Return (en ingles se lee flower), y son una serie de cláusulas que se usan para construir expresiones XQuery.

- **FOR**: Indica qué nodos se van a seleccionar desde la base de datos XML o desde un documento XML.
- **LET**: Permite declarar variables a las que se le asignan valores.
- **WHERE**: Permite introducir condiciones que deben cumplir los nodos seleccionados por la cláusula "for".
- **ORDER BY**: Permite ordenar los nodos antes de su visualización.
- **RETURN**: Devuelve los resultados. Es la única cláusula obligatoria.

2.1. Cláusula "for" y "return"

Con la cláusula "**for**" recuperaremos una serie de elementos (habitualmente desde un documento XML de partida utilizando XPATH) y los introduciremos en una variable para poder utilizarla en la cláusula "**return**". Hay que señalar que la cláusula "return" se ejecutará una vez por cada nodo que devuelva la cláusula "for".
En el return se pueden añadir condicionales usando *if-then-else* y así tener más versatilidad en la salida.
Hay que tener en cuenta que la cláusula *else* es obligatoria y debe aparecer siempre en la expresión condicional, se debe a que toda expresión XQuery debe devolver un valor. Si no existe ningún valor a devolver al no cumplirse la cláusula *if*, devolvemos una secuencia vacía con else().
Si añadimos etiquetas los datos a visualizar los encerramos entre llaves { }.

2.1. Cláusula "for" y "return"

Ejemplos:

- Se genera la secuencia de números del 1 al 5 y se asigna iterativamente a la variable $x:

```
for $x in (1 to 5)
return <numero>{$x}</numero>
```

- Mostrar el titulo de todos los libros:

```
for $book in /bib/book
return $book/title
```

2.1. Cláusula "for" y "return"

- Si queremos numerar los resultados que se van procesando utilizaremos la claúsula **at**.

```
for $book at $i in /bib/book
return <libro>{$i}.{$book/title/string()}</libro>
```

- Si quisiéramos englobar todas las etiquetas anteriores en una superior, tendríamos que agregar el contenido que se calcula cuando se ejecuta la consulta entre llaves { }. Esto se llama una expresión cerrada:

```
<biblioteca>
{
  for $book in /bib/book
  return <titulo>{$book/title/text()}</titulo>
}
</biblioteca>
```

2.1. Cláusula "for" y "return"

- Mostrar todos los autores de libros existentes en el XML. Para no mostrar resultados repetidos utilizaremos la función **distinct-values().**

```
for $autores in distinct-values(/bib/book/author)
return $autores
```

- Mostrar el número de autores por cada uno de los libros

```
for $libros in /bib/book
let $autores:=count($libros/author)
return<libro>{$libros/title/string()}--Autores:
{$autores}</libro>
```

2.1. Cláusula "for" y "return"

- Si queremos mostrar los títulos de los libros según su temática Web o Redes:

```
for $book in /bib/book
return if ($book/title[contains(lower-case(.),"web")]) then
<Web>{$book/title/string()}</Web>
else <Redes>{$book/title/string()}</Redes>
```

2.2. Cláusula “let”.

La cláusula "let" nos va a permitir crear variables con cierto contenido. La diferencia con "for" es que ésta sólo se ejecutaría una sola vez con la cláusula "return". La cláusula "let" asigna las variables mediante los caracteres ":=".

Se puede declarar de dos maneras:

a)

```
let $x:=7, $y:= 3
return 10*$x+$y
```

b)

```
let $x:=7
let $y := 3
return 10*$x+$y
```

2.2. Cláusula “let”.

Ejemplos:

- Si el ejemplo anterior de libros, lo realizáramos con "let":

```
let $book := /bib/book
return <titulo>{$book/title}</titulo>
```

- Buscar el año más alto que exista mediante la función "max" para ver el último libro que se ha escrito:

```
let $max := max(/bib/book/year)
return <last_year>{$max}</last_year>
```

2.2. Cláusula “let”.

- Si queremos mostrar los libros con etiquetas XML que identifiquen el año sería:

```
for $libro in /bib/book
let $year:=$libro/year
return element {concat('libros',$year)}{$libro}
```

Y si quisiéramos incluir el año como atributo sería:

```
for $libro in /bib/book
let $year:=$libro/year
return <libros year="{$year}">{$libro}</libros>
```

2.2. Cláusula “let”.

- Si queremos mostrar los libros con etiquetas XML que identifiquen el autor sería:

```
for $author in distinct-values(/bib/book/author)
let $book:=/bib/book[author=$author]
return element {concat('author',$author)}{$book}
```

(No podríamos hacerlo de la siguiente forma porque hay libros que pueden tener más de un autor y nos daría error:

```
for $book in /bib/book
let $author:=$book/author
return element {concat('author',$author)}{$book}
```

2.3. Cláusula “where”.

Con la cláusula "where" podemos filtrar los nodos que se seleccionan en la cláusula "for", para ello también podemos utilizar los mismos operadores y funciones que en el lenguaje XPath. Muy importante, la cláusula "where" NO filtraría los nodos si los estamos obteniedo con "let".

Ejemplos

- Mostrar el último libro publicado

```
let $anyo:=max(/bib/book/year)
for $libro in /bib/book
where $libro/year=$anyo
return <libro>{$libro/title}{$libro/author}</libro>
```

2.3. Cláusula “where”.

- Mostrar los títulos de un determinado autor:

```
for $book in /bib/book
where $book/author = "Stevens"
return $book/title
```

- Mostrar todos los títulos cuyo autor empiece por la letra “S”:

```
for $book in /bib/book
where $book/author[starts-with(.,"S")]
return $book/title
```

2.4. Cláusula “order by”.

Con la cláusula "order by" podemos ordenar los nodos antes de que empiece a ejecutarse la cláusula "return", ya que como sabemos, la salida será la misma que el orden que tengan los nodos en el documento o base de datos XML. Por defecto se ordenan de forma ascendente (ascending) si queremos hacerlo de forma descendente añadiremos la claúsula descending.

Ejemplos

- Ordenar descendentement los libros por fecha de publicación:

```
for $book in /bib/book
order by $book/year descending
return $book
```

2.4. Cláusula “order by”.

Con la cláusula "order by" podemos ordenar los nodos antes de que empiece a ejecutarse la cláusula "return", ya que como sabemos, la salida será la misma que el orden que tengan los nodos en el documento o base de datos XML. Por defecto se ordenan de forma ascendente (ascending) si queremos hacerlo de forma descendente añadiremos la claúsula descending.

Ejemplos

- Ordenar descendentement los libros por fecha de publicación:

```
for $book in /bib/book
order by $book/year descending
return $book
```

3. Consultas complejas con XQUERY

Dentro de las consultas XQuery podremos trabajar con varios documentos xml para extraer su información, podremos incluir tantas sentencias *for* como se deseen, incluso dentro del *return*. Además, podremos añadir, borrar e incluso modificar elementos, bien generando un documento xml nuevo.

3.1. Joins de documentos

Ejemplos

- Visualizar por cada empleado del documento empleados.xml, su apellido, su número de departamento y el nombre del departamento que se encuentra en el documento departamentos.xml.

```
for $emp  in /EMPLEADOS/EMPLEADO
let  $apellido:=$emp/APELLIDO
let $dep:=$emp/DEPT_NO
let $nom-dep:=(/DEPARTAMENTOS/DEPARTAMENTO[DEPT_NO=$dep]/DNOMBRE)
return <EMPLEADO>{$apellido} {$nom-dep}</EMPLEADO>
```

3.1. Joins de documentos

- Utilizando los documentos departamentos.xml y empleados.xml, obtener por cada departamento, el nombre de departamento y la media de salario.

```
for $dep in /DEPARTAMENTOS/DEPARTAMENTO
let $dep-no:=$dep/DEPT_NO
let $media:=floor(avg(/EMPLEADOS/EMPLEADO[DEPT_NO=$dep-no]
                                                /SALARIO))
return <DEPARTAMENTO>{$dep/DNOMBRE/string()}, Salario base:
{$media}€</DEPARTAMENTO>
```

3.1. Joins de documentos

- Convertir la salida de la consulta anterior, de manera que el total salario, sea atributo de cada departamento. Hacemos que la salida que se cree sea una concatenación (concatenamos con comillas simples) de los datos a obtener:

```
for $dep in /DEPARTAMENTOS/DEPARTAMENTO
let $dep-no:=$dep/DEPT_NO
let $media:=floor(avg(/EMPLEADOS/EMPLEADO[DEPT_NO=$dep-no]
                                                /SALARIO))
return concat('<departamento media="',$media ,'">',
data($dep/DNOMBRE), '</departamento>')
```

3.2. Utilización de varios *for*.

Ejemplos:

- Visualiza por cada departamento del documento universidad.xml, el número de empleados que hay en cada puesto de trabajo. Utilizamos un *for* para obtener los nodos departamento y el segundo *for* para obtener los distintos puestos de cada departamento.

```
for $dep in /universidad/departamento
for $pue in distinct-values($dep/empleado/puesto)
let $cu:=count($dep/empleado[puesto=$pue])

return
<depart>{data($dep/nombre)}<puesto>{$pue}</puesto><profes>{$c
u}</profes></depart>
```

3.2. Utilización de varios *for*.

- Visualiza por cada departamento del documento universidad.xml, el salario máximo y el empleado que tiene ese salario. El primer *for* obtiene los nodos departamento y el segundo *for* los empleados de cada departamento. Para sacar el máximo en la salida preguntamos si el salario es el máximo.

```
for $dep in /universidad/departamento
for $emp in $dep/empleado
let $emple:= $emp/nombre
let $sal:= $emp/@salario
return if ($sal = $dep/max(empleado/@salario)) then
<depart>{data($dep/nombre)}      <salamax>{data($sal)}</salamax>
<empleado>{data ($emple) }</empleado></depart>
else ()
```

3.2. Utilización de varios *for*.

- A partir del XML productos.xml y zonas.xml, obtén la zona que más productos tenga:

```
let $max:=(for $zona in (/zonas/zona)
let $count:=count(/productos/produc[cod_zona=$zona/cod_zona])
order by $count descending
return $count)[1]

for $zonas in (/zonas/zona)
let $productos:=count(/productos/produc[cod_zona=$zonas/cod_zona])
return if ($max=$productos) then
$zonas
 else()
```

4. Inserción, borrado y modificación

RENOMBRADO DE NODOS

Es posible renombrar un elemento o grupo de elementos determinados. Se le puede cambiar el nombre a un nodo con la sentencia **rename**. Por ejemplo, podemos cambiar la primera etiqueta <nif> de una persona por la etiqueta <cif> con la siguiente sentencia:

```
rename node /personas/persona[1]/nif as "cif"
```

Rename funciona nodo a nodo. Si se quiere cambiar todos los nif de personas por cif se utilizaría el siguiente código:

```
for $nif in /personas/persona/nif
return
rename node $nif as "cif"
```

4. Inserción, borrado y modificación

INSERCIÓN DE NODOS

Se puede añadir un nuevo nodo a un XML con la sentencia INSERT. Si quisiéramos añadir una nueva persona en la base de datos se realizaría de la siguiente manera:

```
insert node
  <persona>
  <nif>nifnuevo</nif>
  <nombre>nombrenuevo</nombre>
  <direccion>calleNuevo</direccion>
  <poblacion>puebloNuevo</poblacion>
  <telefono>nuevo</telefono>
  <gastos>
  <gasto>
  <fecha>2012-01-01</fecha>
  <valor>100</valor>
  </gasto>
  </gastos>
  </persona>
before /personas/persona[1]
```

Este código inserta el nodo indicado en fichero "personas.xml". El nodo se insertará antes del primer nodo de la base de datos (por la clausula "before"). Si se quiere insertar después, solo cambiaría "before" por "after".

4. Inserción, borrado y modificación

Una sentencia equivalente para insertar al principio del fichero, sin tener que referenciar a nodo alguno de la base de datos, sería sustituyendo la última línea por esta otra:

```
as first into /personas
```

Cada vez que se pulsa una ejecución introduce un nuevo elemento repetido, pasa exactamente lo mismo que con la sentencia insert en una base de datos.

Si, por el contrario, lo que se desea es insertar al final del XML, sin referenciar a ningún nodo, se realizaría de la siguiente manera:

```
as last into /personas
```

4. Inserción, borrado y modificación

MODIFICACIÓN DE VALORES

En el apartado anterior hemos insertado una nueva persona, ahora vamos a cambiar el nif del primero cambiando nifnuevo por nif-n1. La modificación se realizaría de la siguiente manera:

```
replace value of node /personas/persona[1]/nif
with "nif-n1"
```

Para modificar varios nodos es necesario usar un bucle for. Por ejemplo, fijar todas las poblaciones a Plasencia:

```
for $poblacion in /personas/persona/poblacion
return
replace value of node $poblacion with "Plasencia"
```

4. Inserción, borrado y modificación

BORRADO DE NODOS

Después de añadir y modificar elementos, podemos borrarlos. Por ejemplo, se van a borrar la primera y la segunda persona:

```
delete node /personas/persona[1],
delete node /personas/persona[2]
```

Cuidado con los borrados múltiples que pueden dejar el fichero vacío.

4. Inserción, borrado y modificación

Delete permite borrar listas de nodos, por lo que es posible ejecutar el siguiente código para borrar todos los clientes de pueblo1:

```
for $persona in /personas/persona[poblacion =lower-
case("pueblo1")]
return
delete node $persona
```

Para más información sobre XQuery: http://www.w3.org/TR/xquery/

Dudas y preguntas