6. Almacenamiento de la información

Autor	Ⓧ Xerach Casanova
Clase	Lenguajes de Marcas
Fecha	@Mar 11, 2021 7:16 AM

1. Utilización de XML para almacenamiento de la información

Podemos pensar en XML como una base de datos, una colección de documentos XML, cada uno de ellos representa un registro de la base de datos.

La estructura de un documento XML suele seguir un mismo esquema XML, aunque no es necesario que sea así. Cada archivo se puede configurar de forma estructurada e independiente, pero fácilmente accesible.

La ventaja principal de bases de datos XML es que proporcionan gran flexibilidad, lo que conlleva a crear aplicaciones que usen bases de datos XML.

1.1. Ámbitos de aplicación

Los documentos y requerimientos de almacenamiento de datos XL se pueden agrupar en:

- **Sistemas centrados en los datos.** Cuando los XML tienen una estructura bien definida y contienen datos que pueden ser actualizados y usados de diversos modos. Es apropiado para ítems con contenidos de periódicos, artículos, publicidad, facturas, órdenes de compra y algunos documentos menos estructurados.
- **Sistemas centrados en los documentos.** Cuando tienden a ser más impredecibles de tamaño y contenido. Presentan más tipos de datos, con reglas flexibles para campos y opcionales para el contenido.

La mayoría de los productos se enfocan en servir uno de esos dos formatos de datos mejor que el otro. Las bases de datos relacionales tradicionales tratan mejor requerimientos centrados en los datos, mientras que los sistemas de administración de contenido y documentos son mejores para almacenar datos centrados en el documento.

Los sistemas de bases de datos deben ser capaces de exponer datos relaciones y documentos XML, así como almacenar un documento XML

2. Sistemas de almacenamiento de la información

XML permite integrar información hasta ahora separados:

- **Sistemas de información basados en documentos** (ficheros), tienen estructura irregular, utilizan tipos de datos relativamente simples y dan gran importancia al orden.
- **Sistemas de información estructurados** (bases de datos relacionales), son relativamente planos, utilizan tipos de datos relativamente complejos y dan poca importancia al orden.

Podemos establecer las siguientes semejanzas entre bbdd y un fichero xml con su esquema asociado:

- La tecnología XML usa uno o más documentos para almacenar la información.
- Define esquemas sobre la información.

- Tiene lenguajes de consulta específicos para recuperar la información requerida.
- Dispone de APIs (SAX, DOM).

Debido a que no es una base de datos, XML carece de almacenamiento y actualización eficientes como índices, seguridad, transacciones, integridad de datos, acceso concurrente, disparadores, etc...

3. XML y BD relacionales

Las BD relacionales se basan en relaciones como medio para representar los datos del mundo real y están asociadas a SQL.

Las bases de datos relacionales suponen una posibilidad para el almacenamiento de datos XML, pero no están bien preparadas para almacenar estructuras de tipo jerárquico como lo son XML. Algunas causas:

- Las bd relacionales tienen estructura regular frente al carácter heterogéneo de los documentos XML.
- Los documentos XML suelen contener muchos niveles de anidamiento y los datos relacionales son planos.
- Los documentos XML tienen un orden intrínseco, los relacionales no son ordenados.
- Los datos relacionales son generalmente densos (cada columna tiene valor). Los datos XML son dispersos, pueden representar carencia de información mediante la ausencia del elemento.

Las razones para usar los tipos de BD relacionales y los productos de bd existentes para almacenar XML, aunque no sea de forma nativa son:

- Las bd relacionadas y orientadas a objetos son conocidas, mientras que las bd XML nativas son nuevas.
- como resultado, La familiaridad con las bd relacionales y orientadas a objetos, los usuarios se inclinan a ellas por el rendimiento.

3.1. De BD relacional a XML.

El proceso de traducción se descompone en los siguientes pasos.

- **Crear el esquema XML** con un elemento para cada tabla y los atributos correspondientes para cada columna no clave. Las columnas que no permiten valores nulos se marcan como requeridos y las no requeridas se marcan como

opcionales en el esquema XML. Las columnas pueden ser también anidadas como elementos, pero pueden surgir problemas cuando el mismo nombre de columna se usa en más de una tabla.

- **Crear las claves primarias en el esquema XML.** Se puede agregar un atributo para la columna clave, con un ID agregado al nombre de la columna. Este atributo puede necesitar ser definido en el esquema XML como tipo ID. Para resolver el problema de posibles colisiones por coincidencia de nombres en las tablas de la bd relacional, se puede agregar el nombre del elemento (nombre de la tabla), al valor de la clave primaria (valor del atributo).
- **Establecer las relaciones de clave migrada.** Esto se logra mediante anidamiento de elementos bajo el elemento padre. un ID de esquema XML puede ser usado para apuntar a una estructura XML correspondiente conteniendo un IDREF.

Pueden existir muchas variaciones de esquemas XML para una BD relacional.

4. XML y BD orientadas a objetos

Las bases de datos orientadas a objetos, soportan un modelo de objetos puro y no están basados en extensiones de otros modelos como el relacional.

- Están influenciados por lenguajes POO.
- Pueden verse como un intento de añadir funcionalidad SGBD a un lenguaje de programación.
- Son una alternativa para el almacenamiento y gestión de XML.

Componentes estándar de Orientación a Objetos:

- **Modelo de objetos:** Concebido para proporcionar un modelo de objetos estándar para BDOO. Es el modelo en que se basan los demás componentes.
- **Lenguajes de especificación de objetos (ODL)** para definir objetos.
- **Lenguaje de consulta de objetos (OQL)** para realizar consultas a objetos.
- **Bindings para C++, Java y Smalltalk.** Definen un lenguaje de manipulación de objetos OML, que extiende el lenguaje de programación para soportar objetos persistentes. Además incluyen soporte para OQL, navegación y transacciones.

Cuando se transforma el XML en objetos, son gestionados por SGBDOO, y se consulta la información mediante OQL, los mecanismos de indexación, optimización y procesamiento de consultas son del propio SGBDOO, no suelen ser específicos para el modelo XML.

5. BD XML nativas

Las bases de datos XML nativas son bases de datos y soportan transacciones, acceso multi-usuario, lenguajes de consulta, etc... y están diseñadas para almacenar documentos xml.

Se caracterizan por:

- Almacenar documentos en colecciones. Estas colecciones son como las tablas en BD relacionales.
- Validación de los documentos.
- Consultas. la mayoría de las BD XML nativas soportan uno o más lenguajes de consulta. El más popular es XQuery.
- Indexación XML. Se permite la creación de índices que aceleran consultas.
- Creación de identificadores únicos. A cada documento se le asocia una ID única.
- Actualizaciones y borrado.

Según el tipo de almacenamiento se divide en dos grupos.

- **Almacenamiento basado en texto.** Se almacena el documento en texto y proporciona alguna funcionalidad de base de datos para acceder a él. Dos posibilidades.
  - Posibilidad 1. se almacena como un BLOB en una bd relacional, mediante un fichero y proporciona algunos índices sobre el documento que aceleren el acceso a la info.
  - Posibilidad 2. Almacenar un documento en un almacén adecuado con índices, soporte para transacciones, etc...
- **Almacenamiento basado en el modelo.** Almacena un modelo binario del documento (por ejemplo, DOM), en un almacén existente o bien específico.
  - Posibilidad 1: traducir el DOM a tablas relacionales como elementos, atributos, etc.
  - Posibilidad 2: traducir el DOM a objetos en un BDOO.
  - Posibilidad 3: utilizar un almacén creado para esta finalidad.

6. XQuery

XQuery es un lenguaje diseñado para escribir consultas sobre colecciones de datos expresadas en XML. Puede aplicarse a archivos XML y a bases de datos relacionales con funciones de conversión a registros XML. Su principal función es extraer información de un conjunto de datos organizados como un árbol de etiquetas XML. XQuery es independiente del origen de datos.

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

Los requerimientos técnicos más importantes son:

- Debe ser un lenguaje declarativo.
- Debe ser independiente del protocolo de acceso a la colección de datos. XQuery funciona igual al consultar un archivo local, que al consultar una base de datos o un XML en una web.
- Las consultas y resultados deben respetar el modelo de datos XML.
- Las consultas y los resultados deben ofrecer soporte para los namespaces.
- Debe soportar XML-Schemas y DTD's, y también debe ser capaz de trabajar sin ellos.
- Ha de ser independiente de la estructura del documento, funciona sin conocerla.
- Debe soportar tipos simples, como enteros y cadenas, o tipos complejos como un nodo compuesto.
- Las consultas deben soportar cuantificaciones universales (para todo) y existenciales (existe).
- las consultas deben soportar operaciones jerárquicas de nodos y secuencias de nodos.
- Debe ser posible combinar información de múltiples fuentes en una consulta.
- Las consultas deben ser capaces de manipular datos independientemente del origen de estos.
- El lenguaje de consulta debe ser independiente de la sintaxis. Existen varias sintaxis distintas en una misma consulta XQuery.

6.1. Aplicaciones

- Recuperar información a partir de datos XML.
- Transformar una estructura de datos XML en otras estructuras que organizan la información de forma distinta.

- Ofrecer una alternativa a XSLT para realizar transformaciones.

Los motores XQuery de código abierto más relevantes son:

- BaseX. Open Source y disponible para Linux, Windows y Mac.
- Qexo. Escrito en Java y con licecnia GPL que se distribuye dentro del paquete Kawa.
- Saxon. Escrito en Java y distribuido en múltiples paquetes, algunos de ellos open-source.

6.2. Modelo de datos

XQuery y SQL se pueden considerar similares, pero el modelo de datos de XQuery es distinto al de SQL. XML incluye conceptos como el de jerarquía y orden de datos.

A diferencia de SQL el orden en el que se encuentran los datos en XQuery es importante. No es lo mismo buscar una etiqueta <B> dentro de <A>, que todas las etiquetas <B>.

La entrada y salida de una consulta se define en términos de un modelo de datos, el cual proporciona una representación abstracta de uno o más XML.

Sus características son:

- Se basa en la definición de secuencia, como colección ordenada de cero o más ítems, pueden ser heterogéneas) contener varios nodos y valores atómicos), pero nunca puede ser un ítem de otra secuencia.
- Orden del documento. Corresponde al orden en que los nodos aparecerían si la jerarquía fuese representada en XML, si el primer carácter de un nodo ocurre antes que el primero de otro, lo precede también en el orden del documento.
- Contempla un valor especial llamado "error value", que es el resultado de evaluar una expresión que da error.

6.3. Expresiones

Una consulta XQuery es una expresión que lee una secuencia de datos XML y devuelve como resultado otra secuencia de datos en XML.

La mayoría de expresiones están compuestas por la combinación de expresiones más simples unidas por operadores y palabras reservadas.

XQuery se ha construdio sobre la base de XPath, toda expresión XPath es una consulta XQuery válida.

Los comentarios de delimitan entre (: y :).

En un XQuery los caracteres {} delimitan expresiones que son evaluadas para crear un nuevo documento.

XQuery admite condicionales tipo if-then-else, con la misma semántica de los lenguajes habituales.

Pueden estar formadas por cinco tipos de cláusulas distintas. Siguen la norma FLWOR (se pronuncia flower). Estas cláusulas son los bloques principales de XQuery que equivalen a select, from, where, group by, having, order by y limit de SQL:

En una sentencia FLWOR al menos debe existir un FOR o un LET, el resto si existen deben respetar el orden dado por FLWOR.

Con estas sentencias se consigue buena parte de la funcionalidad que diferencia XQuery de Xpath. Entre otras cosas permite construir el documento que será la salida de la sentencia.

Una consulta XQuery está formada por dos partes:

- Prólogo. Donde se declaran nombres, funciones, variables....
- Expresión, consulta propiamente dicha.

6.4. Cláusulas

- **FOR:** asocia una o más variables con cada nodo que encuentre en la colección de datos. 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.
- **LET:** vincula las variables al resultado de una expresión. Si esta cláusula aparece en una sentencia en la que ya hay al menos una cláusula FOR, los valores de la variable vinculada por la cláusula LET se añaden a cada una de las tuplas generadas por la cláusula FOR.
- **WHERE:** filtra tuplas producidas por las cláusulas FOR y LET, quedando solo aquellas que cumplen con la condición.
- **ORDER BY:** ordena las tuplas generadas por FOR Y LET después de que han sido filtradas por la cláusula WHERE. Por defecto el orden es ascendiente, pero se puede usar el modificado DESCENDING para cambiar el sentido del orden.

- **RETURN:** construye el resultado de la expresión FLWOR para una tupla dada.

6.5. Ejemplos XQuery

Los siguientes ejemplos están realizados a partir del siguiente código.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!-- Fichero: libros.xml -->
<biblioteca>
  <libros>
    <libro publicacion="2003" edicion="2">
      <titulo>Learning XML</titulo
      <autor>
        <apellido>Ray</apellido>
        <nombre>Erik T.</nombre>
      </autor>
      <editorial>O'Reilly</editorial>
      <paginas>16</paginas>
      <versionElectronica/>
    </libro>
    <libro publicacion="2003" edicion="2">
      <titulo>XML Imprescindible</titulo>
      <autor>
        <apellido>Harold</apellido>
        <nombre>Elliot Rusty</nombre>
      </autor>
      <autor>
        <apellido>Means</apellido>
        <nombre>W. Scott</nombre>
      </autor>
      <editorial>O'Reilly</editorial>
      <paginas>832</paginas>
    </libro>
    <libro publicacion="2002">
      <titulo>XML Schema</titulo>
      <autor>
        <apellido>van der Vlist</apellido>
        <nombre>Eric</nombre>
      </autor>
      <editorial>O'Reilly</editorial>
      <paginas>400</paginas>
    </libro>
    <libro publicacion="2002">
      <titulo>XPath Essentials</titulo>
      <autor>
        <apellido>Watt</apellido>
        <nombre>Adrew</nombre>
      </autor>
      <editorial>Wiley</editorial>
      <paginas>516</paginas>
      <versionElectronica/>
    </libro>
    <libro publicacion="2005">
      <titulo> Beginning XSLT 2.0: Form Novice to Professional</titulo>
      <autor>
        <apellido>Tennison</apellido>
        <nombre>Jeni</nombre>
```

```
      </autor>
      <editorial>Apress</editorial>
      <paginas>797</paginas>
    </libro>
    <libro publicacion="2007">
      <titulo> XQuery</titulo>
      <autor>
        <apellido>Walmsley</apellido>
        <nombre>Priscilla</nombre></autor>
      <editorial>O'Reilly</editorial>
      <paginas>491</paginas>
    </libro>
  </libros>
</biblioteca>
```

- El elemento raíz es **biblioteca**. Contiene un elemento libros.
- Dentro de libros, hay varios elementos libro.
- Los elementos **libro** tienen atributos **publicacion** y **edicion** (opcional). También tienen elementos titulo, autor (puede haber más de uno), **editorial**, **paginas** y un elemento opcional para indicar si hay edición electrónica, **edicionElectronica**.

Ejercicio 1

Título y editorial de todos los libros. Para devolver varios campos, los envolvemos en un elemento.

```
for $x in doc("libros.xml")/biblioteca/libros/libro
return <libro>{$x/titulo, $x/editorial}</libro>
```

```
<libro>
  <titulo>Learning XML</titulo>
  <editorial>O'Reilly</editorial>
</libro>
<libro>
  <titulo>XML Imprescindible</titulo>
  <editorial>O'Reilly</editorial>
</libro>
<libro>
  <titulo>XML Schema</titulo>
  <editorial>O'Reilly</editorial>
</libro>
<libro>
  <titulo>XPath Essentials</titulo>
  <editorial>Wiley</editorial>
</libro>
<libro>
  <titulo>Beginning XSLT 2.0: Form Novice to Professional</titulo>
  <editorial>Apress</editorial>
</libro>
```

```
<libro>
  <titulo>XQuery</titulo>
  <editorial>O'Reilly</editorial>
</libro>
```

Ejercicio 2.

El título (sin etiquetas) de todos los libros de menos de 100 páginas.Para hacer comparaciones con números, lo mejor es convertir los datos con la función number para evitar problemas de tipo de dato o que los compare como cadenas.

```
for $x in doc("libros.xml")/biblioteca/libros/libro
where number($x/paginas) < 100
return data($x/titulo)
```

```
Learning XML
```

Ejercicio 3.

```
El número de  libros de menos de 100 páginas. Utilizamos la función count().
```

```
for $x in doc("libros.xml")/biblioteca/libros
let $y := $x/libro[number(paginas) < 100]
return count($y)
```

```
1
```

Ejercicio 4.

Una lista HTML con el título de los libros de la editorial “O'Reilly” ordenados por título. Podemos mezclar etiquetas HTML y XQuery y obtener HTML como resultado de una consulta.

```
<ul>
{
for $x in doc("libros.xml")/biblioteca/libros/libro
where $x/editorial = "O'Reilly"
order by $x/titulo
return <li>{data($x/titulo)}</li>
}
</ul>
```

```
<ul>
  <li>Learning XML</li>
  <li>XML Imprescindible</li>
  <li>XML Schema</li>
  <li>XQuery</li>
</ul>
```

Ejercicio 5

Título y editorial de los libros de 2002.

```
for $x in doc("libros.xml")/biblioteca/libros/libro
where $x[@publicacion=2002]
return <libro>{$x/titulo, $x/editorial}</libro>
```

```
<libro>
  <titulo>XML Schema</titulo>
  <editorial>O'Reilly</editorial>
</libro>
<libro>
  <titulo>XPath Essentials</titulo>
  <editorial>Wiley</editorial>
</libro>
```

Ejercicio 6

Título y editorial de los libros con más de un autor.

```
for $x in doc("libros.xml")/biblioteca/libros/libro
where count($x/autor)>1
return <libro>{$x/titulo, $x/editorial}</libro>
```

```
<libro>
  <titulo>XML Imprescindible</titulo>
  <editorial>O'Reilly</editorial>
</libro>
```

Ejercicio 7

Título y editorial de los libros que tienen versión electrónica.

```
for $x in doc("libros.xml")/biblioteca/libros/libro
where $x/versionElectronica
return <libro>{$x/titulo, $x/editorial}</libro>
```

```
<libro>
  <titulo>Learning XML</titulo>
  <editorial>O'Reilly</editorial>
</libro>
<libro>
  <titulo>XPath Essentials</titulo>
  <editorial>Wiley</editorial>
</libro>
```

Ejercicio 8

Título de los libros que no tienen versión electrónica.

```
for $x in doc("libros.xml")/biblioteca/libros/libro
where not($x/versionElectronica)
return$x/titulo
```

```
<titulo>XML Imprescindible</titulo>
<titulo>XML Schema</titulo>
<titulo>Beginning XSLT 2.0: Form Novice to Professional</titulo>
<titulo>XQuery</titulo>
```

6.6. Funciones

Las funciones que soporta XQuery son matemáticas, de cadenas, tratamiento de expresiones regulares, comparaciones de fechas y horas, manipulación de nodos, manipulación de secuencias, comprobación y conversión de tipos y lógica booleana. Algunas de ellas son:

- **Funciones numéricas**
  - floor() que devuelve el valor numérico inferior más próximo al dado**.**
  - ceiling()**,** que devuelve el valor numérico superior más próximo al dado**.**
  - round()**,** que redondea el valor dado al más próximo.
  - count(), determina el número de ítems en una colección.
  - min() o max() **,** devuelven respectivamente el mínimo y el máximo de los valores de los nodos dados.
  - avg() **,** calcula el valor medio de los valores dados.
  - sum(**,** calcula la suma total de una cantidad de ítems dados.

Funciones de cadenas de texto

- concat(), devuelve una cadena construida por la unión de dos cadenas dadas.
- string-length(), devuelve la cantidad de caracteres que forman una cadena.
- startswith(), ends-with(), determinan si una cadena dada comienza o termina, respectivamente, con otra cadena dada.
- upper-case(), lower-case(), devuelve la cadena dada en mayúsculas o minúsculas respectivamente.
- **Funciones de uso general**
  - empty(), devuelve “true” cuando la secuencia dada no contiene ningún elemento.
  - exits(), devuelve “true” cuando una secuencia contiene, al menos, un elemento.
  - distinct-values(), extrae los valores de una secuencia de nodos y crea una nueva secuencia con valores únicos, eliminando los nodos duplicados.
  - data(), devuelve el valor de los elementos que recibe como argumentos, es decir, sin etiquetas.
- Cuantificadores existenciales:
  - some, every, permiten definir consultas que devuelven algún, o todos los elementos, que verifiquen la condición dada.

Ejemplo de la función data:

Esta consulta devuelve todos los títulos, incluyendo las etiquetas:

```
for $x in doc("libros.xml")/biblioteca/libros/libro/titulo
return $x
```

Resultado:

```
<titulo>Learning XML</titulo>
<titulo>XML Imprescindible</titulo>
<titulo>XML Schema</titulo>
<titulo>XPath Essentials</titulo>
<titulo>Beginning XSLT 2.0: Form Novice to Professional</titulo>
<titulo>XQuery</titulo>
```

Utilizando data:

```
for $x in doc("libros.xml")/biblioteca/libros/libro/titulo
return data($x)
```

se obtiene:

```
Learning XML
XML Imprescindible
XML Schema
XPath Essentials
Beginning XSLT 2.0: Form Novice to Professional
XQuery
```

También se pueden crear funciones propias:

```
declare nombre_funcion($param1 as tipo_dato1, $param2 as tipo_dato2,… $paramN as tipo_datoN)
as tipo_dato_devuelto
{
...CÓDIGO DE LA FUNCIÓN...
}
```

6.7. Ejemplo: definición y llamada a una función.

En este ejemplo puedes ver la definición y llamada a una función escrita por el usuario que nos calcula el precio de un libro una vez que se le ha aplicado el descuento.

```
declare function minPrice($p as xs:decimal?,$d as xs:decimal?) as xs:decimal?
  {
  let $disc := ($p * $d) div 100
  return ($p - $disc)
  }
```

Un ejemplo de cómo invocar a la función desde la consulta es:

```
<minPrice>{minPrice($libros/precio,$libros/descuento)}</minPrice>
```

6.8. Operadores

Agrupados por funcionalidad.

- **Comprobación de valores:** Comparan dos valores escalares y produce un error si alguno de los operandos es una secuencia de longitud mayor de 1.

Estos operadores son:

- **eq:** igual.
- **nee:** no igual
- **lt:** menor qué
- **le:** menor o igual qué
- **gt:** mayor qué
- **ge:** mayor o igual qué

- **Comparación generales:** permiten comparar operandos que sean secuencias.
  - **=** igual
  - **!=** distinto
  - **>** mayor qué
  - >=, mayor o igual que.
  - <**,** menor que.
  - <=, menor o igual que.
- **Comparación de nodos:** Comparan la identidad de dos nodos.
  - **is**, devuelve true si las dos variables que actúan de operandos están ligadas al mismo nodo.
  - **is not**, devuelve true si las dos variables no están ligadas al mismo nodo.
- **Comparación de órdenes de los nodos:** <<, compara la posición de dos nodos. Devuelve "true" si el nodo ligado al primer operando ocurre primero en el orden del documento que el nodo ligado al segundo.
- **Lógicos:** and y or Se emplean para combinar condiciones lógicas dentro de un predicado.
- **Secuencias de nodos:** Devuelven secuencias de nodos en el orden del documento y eliminan duplicados de las secuencias resultado.
  - **Unión**, devuelve una secuencia que contiene todos los nodos que aparecen en alguno de los dos operandos que recibe.
  - **Intersect**, devuelve una secuencia que contiene todos los nodos que aparecen en los dos operandos que recibe.
  - **Except**, devuelve una secuencia que contiene todos los nodos que aparecen en el primer operando que recibe y que no aparecen en el

segundo.

- **Aritméticos**: **+, -, *, div y mod**, devuelven respectivamente la suma, diferencia, producto, cociente y resto de operar dos números dados.

Mapa conceptual