TEMA 6: ALMACENAMIENTO DE INFORMACIÓN

1. UTILIZACIÓN DE XML PARA ALMACENAMIENTO DE INFORMACIÓN

Como ya hemos visto, XML, es un estándar potente y de amplia aceptación para guardar y comunicar información acerca de objetos. Permite la codificación de información separada de la forma en la que se debe presentar al usuario. Para encontrar un fragmento específico de información en los contenidos de un nodo o atributo XML, habrá que procesar completamente el archivo XML.

Una base de datos XML se puede ver como una colección de documentos XML, en la que cada documento representa un registro.

Cada documento XML es un archivo en el sistema de archivos y contiene una cadena válida XML.

La estructura de un documento XML suele seguir un mismo esquema XML, aunque no es necesario que sea así. Este es uno de los beneficios de las bases de datos XML, ya que cada archivo se puede configurar de forma estructurada por lo que es independiente, pero fácilmente accesible.

El tener colecciones de documentos con un esquema independiente proporciona a la base de datos flexibilidad, y facilita el desarrollo de la aplicación.

En los últimos años, se ha visto necesaria la existencia de estándares de intercambio de información, con el objetivo de que las organizaciones puedan compartir su información de una manera más cómoda, automática y eficiente.

1.1.ÁMBITO DE APLICACIÓN

Los documentos y los requerimientos de almacenamiento de datos XML pueden ser agrupados en dos categorías generales:

- **Centrados en los datos:** Suelen incluir documentos (archivos) que siguen un esquema común altamente estructurados: una factura, un albarán, un ticket, una matrícula, un acta de notas, órdenes de compra, contenidos de periódicos, artículos y publicidad. Tienen muchos elementos (o etiquetas) con poco contenido o información, suelen estar muy estructurados y se suele automatizar su uso.

  Si el documento XML tiene una estructura bien definida y contiene datos que pueden ser actualizados y usados de diversos modos, el documento es habitualmente centrado en los datos.

- **Centrados en los documentos:** Tienden a ser más impredecibles en tamaño y contenido con tipos de datos de tamaño limitado y reglas menos flexibles para campos opcionales y contenido. Los archivos centrados en documentos son archivos variables dependiendo de quién los realice o escriba (un libro de texto, un manual, un informe, etc.). Tienen pocos elementos (o etiquetas) con gran cantidad de información y desestructurados (o poco estructurados) y están orientados a un uso humano.

  Los sistemas de almacenamiento XML deben acomodarse eficientemente con ambos tipos de requerimientos de datos, dado que XML está siendo usado en sistemas que administran ambos tipos de datos. La mayoría de los productos se enfocan en servir uno de esos formatos de datos mejor que el otro.

Las bases de datos relacionales tradicionales **son mejores para tratar con requerimientos centrados en los datos**, mientras que **los sistemas de administración de contenido y de documentos, suelen ser mejores para almacenar datos centrados en el documento**.

Los sistemas de bases de datos deben ser capaces de exponer los datos relacionales como XML, y almacenar el XML recibido como datos relacionales para transferir, obtener y almacenar los datos requeridos por la aplicación.

2. SISTEMAS DE ALMACENAMIENTO DE INFORMACIÓN

Una expresión XPath es un predicado que se aplica sobre una estructura de árbol correspondiente a la jerarquía de los datos de un documento XML y devuelve todo lo que encaja con ese predicado.

En lo que concierne a las bases de datos, XML permite integrar sistemas de 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 una base de datos 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).

Pero aparecen muchas más cosas que lo diferencian. Debido a que no es una base de datos, la tecnología XML carece, entre otras cosas, tanto de almacenamiento y actualización eficientes como de índices, seguridad, transacciones, integridad de datos, acceso concurrente, disparadores, etc.; que son algunas de las características habituales en las bases de datos. Por tanto, es imposible pensar que XML se vaya a utilizar para las tareas transaccionales de una organización para las cuales sigue estando sobradamente más justificado utilizar una base de datos.

2.1. XML Y BD RELACIONALES

Las bases de datos relacionales se basan en las relaciones (tablas bidimensionales), como único medio para representar los datos del mundo real. Están asociadas al lenguaje estándar SQL.

Se han creado complejas teorías y patrones para encajar objetos o estructuras jerarquizadas en bases de datos relacionales.

Existen numerosos middlewares encargados de la transferencia de información entre estructuras XML y bases de datos relacionales.

Las bases de datos relacionales suponen una posibilidad para el almacenamiento de datos XML. Sin embargo, no están bien preparadas para almacenar estructuras de tipo jerárquico como son los documentos XML, algunas de las causas son:

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

Algunas de las razones para usar los tipos de bases de datos relacionales y los productos de bases de datos existentes para almacenar XML, aún cuando no sea de forma nativa son:

- Las bases de datos relacionales y orientadas a objetos son bien conocidas, mientras que las bases de datos XML nativas son nuevas.
- Como resultado de la familiaridad con las bases de datos relacionales y orientadas a objetos, los usuarios se inclinan a ellas especialmente por el rendimiento.

2.1.1. REGLAS DE TRANSFORMACIÓN RELACIONAL. DTD.

La estructura XML debe ser capaz de acomodar algunos conceptos básicos, incluyendo claves primarias, claves secundarias, tablas, y columnas.

El proceso de traducción puede ser descompuesto en los siguientes pasos básicos:

- **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 pueden ser marcadas como requeridas, mientras que aquellas que permiten valores nulos pueden ser marcadas como opcionales en el esquema XML. Las columnas pueden ser también anidadas como elementos, pero pueden surgir problemas cuando el mismo nombre de columna es usado en más de una tabla. Por ello, lo más simple es transformar las columnas como atributos XML, donde las colisiones de nombre en el esquema XML no son un problema.

- **Crear las claves primarias en el esquema XML**. Una solución podría ser agregar un atributo para la columna clave primaria, con un ID agregado al nombre de la columna. Este atributo podría necesitar ser definido en el esquema XML como de tipo ID. Pueden surgir problemas de colisión al crear claves primarias en el esquema XML, ya que, a diferencia de las bases de datos relacionales, donde las claves primarias necesitan ser únicas sólo dentro de una tabla, un atributo ID dentro de un documento XML debe ser único a través de todo el documento.

  Para resolverlo se puede agregar el nombre del elemento (nombre de la tabla), al valor de la clave primaria (valor del atributo). Esto asegura que el valor es único a través del documento XML.

- **Establecer las relaciones de clave migrada o foránea**. Esto se puede lograr mediante el 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 representar la misma base de datos relacional.

2.2. XML Y BASE DE DATOS ORIENTADAS A OBJETOS.

Las **bases de datos orientadas a objetos** soportan un modelo de objetos puro, en el sentido de que no están basados en extensiones de otros modelos más clásicos como el relacional:

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

Componentes del estándar:

- **Modelo de Objetos.** Está concebido para proporcionar un modelo de objetos estándar para las bases de datos orientadas a objetos. Es el modelo en el que se basan el lenguaje de definición de objetos y el lenguaje de consultas.

- **Lenguajes de Especificación de Objetos.** ODL
- **Lenguaje de Consulta.** Conocido como OQL.
- **Bindings** para **C++** , **Java** y **Smalltalk** Definen un OML que extiende el lenguaje de programación para soportar objetos persistentes. Además, incluye soporte para OQL, navegación y transacciones.

Una vez transformado el documento XML en objetos, éstos son gestionados directamente por el SGBDOO. Dicha información se consulta acudiendo al lenguaje de consulta OQL. Los mecanismos de indexación, optimización, procesamiento de consultas, etc. son los del propio SGBDOO, y por lo general, no son específicos para el modelo XML.

2.3. XML Y BASE DE DATOS NATIVAS

Son bases de datos, y como tales soportan transacciones, acceso multi-usuario, lenguajes de consulta, etc. Están diseñadas especialmente para almacenar documentos XML.

Las BD nativas se caracterizan principalmente por:

- **Almacenamiento de documentos en colecciones.** La base de datos está estructurada en colecciones, una colección es un conjunto de documentos, de modo que es una estructura de árbol donde cada documento pertenece a una única colección.

Las colecciones juegan en las bases de datos nativas el papel de las tablas en las BD relacionales.

- **Validación de los documentos.**
- **Consultas.** La mayoría de las BD XML soportan uno o más lenguajes de consulta. Uno de los más populares es XQuery.
- **Indexación XML.** Se ha de permitir la creación de índices que aceleren las consultas realizadas.
- **Creación de identificadores únicos.** A cada documento XML se le asocia un identificador único.
- **Actualizaciones y Borrados**.

Según el tipo de almacenamiento utilizado pueden dividirse en dos grupos:

- **Almacenamiento Basado en Texto.** Almacena el documento XML entero en forma de texto y proporciona alguna funcionalidad de base de datos para acceder a él. Hay dos posibilidades:
    - **Posibilidad 1**: Almacenar el documento como un BLOB en una base de datos relacional, mediante un fichero, y proporcionar algunos índices sobre el documento que aceleren el acceso a la información.
    - **Posibilidad 2**: Almacenar el 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, Entidades, etc.
    - **Posibilidad 2**: Traducir el DOM a objetos en una BDOO.
    - **Posibilidad 3**: Utilizar un almacén creado especialmente para esta finalidad.

3. 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.

Permite la construcción de expresiones complejas combinando expresiones simples de una manera muy flexible.

De manera general podemos decir que XQuery es a XML lo mismo que SQL es a las bases de datos relacionales. Al igual que éste último, XQuery es un lenguaje funcional

Los r**equerimientos técnicos** más importantes **de XQuery** se detallan a continuación:

- Debe ser un lenguaje declarativo.
- Debe ser i**ndependiente del protocolo de acceso** a la colección de datos. Esto significa que una consulta en XQuery, debe funcionar igual al consultar un archivo local, que, al consultar un servidor de bases de datos, o que al consultar un archivo XML en un servidor web.
- Las consultas y los resultados deben respetar el **modelo de datos XML**.
- Las consultas y los resultados deben ofrecer **soporte para los namespaces.**
- Debe soportar **XML-Schemas y DTDs** y también debe ser capaz de trabajar sin ellos.
- Ha de ser **independiente de la estructura** del documento, esto es, funcionar sin conocerla.
- Debe soportar **tipos simples**, como enteros y cadenas, y **tipos complejos**, como un nodo compuesto.
- Las consultas deben soportar **cuantificadores universales** (para todo) y existenciales (existe).
- Las consultas deben soportar **operaciones sobre jerarquías de nodos** y secuencias de nodos.
- Debe ser posible **combinar información de múltiples fuentes** en una consulta.
- Las consultas deben ser capaces de **manipular los datos independientemente del origen** de estos.

- **El lenguaje de consulta debe ser independiente de la sintaxis**, esto es, pueden existir varias sintaxis distintas para expresar una misma consulta en XQuery.

Ejercicio Resuelto

Veamos un ejemplo de una consulta xQuery, que más adelante profundizaremos:

Tenemos como entrada el archivo **alumnos.xml** siguiente:

```
<alumnos>
 <alumno dni="93940">
  <nombre>Jose</nombre>
  <apells>Bernardo</apells>
  <nota>7</nota>
 </alumno>
 <alumno dni="93940">
  <nombre>Juan</nombre>
  <apells>López</apells>
  <nota>4</nota>
 </alumno>
</alumnos>
```

Y obtendremos tras la consulta el siguiente resultado:

```
<aprobado dni="93940">
 <nota>7</nota>
</aprobado>
```

Consulta xQuery:

```
for
 $a in doc("alumnos.xml")//alumno
where
 $a/nota > 5
return
 <aprobado>{$a/@dni,$a/nota}</aprobado>
```

3.1. APLICACIONES

Una vez que hemos visto la definición del lenguaje y sus principales requerimientos, queda pensar, ¿para qué se utiliza?

Sus principales aplicaciones se resumen en tres:

- **Recuperar información a partir de conjuntos de datos XML**.
- **Transformar unas estructuras de datos XML** en otras estructuras que organizan la información de forma diferente.
- **Ofrecer una alternativa a XSLT** para realizar transformaciones de datos en XML a otro tipo de representaciones, como HTML o PDF.

¿Y cuáles son los motores XQuery de código abierto más relevantes y sus características principales?

- **Qexo:** escrito en Java y con licencia GPL que se distribuye integrado dentro del paquete Kawa.
- **Saxon:** escrito en Java y distribuido en dos paquetes:
    - Saxon-B es open-source bajo licencia GPL y contiene una implementación básica de XSLT 2.0 y XQuery.
    - Saxon-SA, contiene un procesador completo XSLT y XQuery pero tiene licencia propietaria, aunque está disponible una licencia de evaluación de 30 días.
- **Qizx/open:** es una implementación Java de todas las características del lenguaje excepto la importación y validación de XML-Schemas. Es uno de los motores con licencia GPL más completo que existe.

¿Qué otras herramientas relacionadas con XQuery existen?

- **Xquark Bridge:** es una herramienta que permite importar y exportar datos a bases de datos relacionales utilizando XML, ofreciendo, por tanto, la posibilidad de manejar estructuras XML y realizar la transformación a objetos de la base de datos, y viceversa. Además, XQuark respeta las restricciones de integridad y transforma las relaciones implícitas en los documentos XML, en relaciones explícitas en la base de datos.

  También soporta la consulta y manipulación de los datos en formato XML utilizando el lenguaje XQuery. XQuark-Bridge soporta MySQL , Oracle , SQLServer y SyBase , tiene una licencia LGPL.

- **BumbleBee:** es un entorno de prueba automático, creado para evaluar motores de XQuery y validar consultas XQuery. BumbleBee permite entre otras cosas, comprobar si una versión más moderna de nuestro motor permite seguir ejecutando nuestras consultas.

  BumbleBee se distribuye con un conjunto de pruebas ya preparadas y además, ofrece un entorno sencillo para redactar y ejecutar nuestras propias pruebas. Soporta los motores XQuery de código abierto: Qexo, Qizx/open y Saxon y los motores XQuery propietarios: Cerisent, Ipedo, IPSI-XQ y X-Hive. Es software propietario, aunque está disponible una versión de demostración completamente funcional durante 30 días.

3.2. REQUERIMIENTOS TÉCNICOS

El grupo de trabajo en XQuery del W3C[20] ha definido un conjunto de **requerimientos técnicos** para este lenguaje. Los más importantes se detallan a continuación:

- XQuery debe ser un **lenguaje declarativo**. Al igual que SQL hay que indicar que se quiere, no la manera de obtenerlo.
- XQuery debe ser **independiente** del protocolo de acceso a la colección de datos. Una consulta en XQuery debe funcionar igual al consultar un archivo local que al consultar un servidor de bases de datos que al consultar un archivo XML en un servidor web.
- Las consultas y los resultados **deben respetar el modelo de datos XML**.
- Las consultas y los resultados deben ofrecer **soporte para los namespace4**.
- Debe ser **capaz de soportar XML-Schemas y DTDs** y también debe ser

capaz de trabajar sin ninguno de ellos.

- XQuery debe poder trabajar con **independencia** de la estructura del documento, esto es, sin necesidad de conocerla.
- XQuery debe **soportar tipos** simples, como enteros y cadenas, y tipos complejos, como un nodo compuesto por varios nodos hijos.
- Las consultan deben **soportar cuantificadores** universales (para todo) y existenciales (existe).
- Las consultas deben **soportar operaciones** sobre jerarquías de nodos y secuencias de nodos.
- Debe ser posible en una consulta **combinar** información de múltiples fuentes.
- Las consultas deben ser capaces de manipular los datos **independientemente del origen** de estos.
- Mediante XQuery debe ser posible **definir** consultas que transformen las estructuras de información originales y debe ser posible **crear** nuevas estructuras de datos.
- El lenguaje de consulta debe ser **independiente de la sintaxis**, esto es, debe ser posible que existan varias sintaxis distintas para expresar una misma consulta en XQuery.

Recomendación

Antes de continuar, recuerda que debes tener claros los conocimientos básicos de XML y XPath. Repasa las unidades anteriores si tienes dudas.

3.3. MODELO DE DATOS

Aunque XQuery y SQL puedan considerarse similares, el modelo de datos sobre el que se sustenta XQuery es muy distinto del modelo de datos relacional sobre el que sustenta SQL, ya que XML incluye conceptos como jerarquía y orden de los datos que no están presentes en el modelo relacional.

Por ejemplo, a diferencia de SQL, en XQuery el orden en que se encuentren los datos es importante, ya que no es lo mismo buscar una etiqueta "<B>" dentro de una etiqueta "<A>" que todas las etiquetas "<B>" del documento (que pueden estar anidadas dentro de una etiqueta "<A>" o no).

La entrada y la salida de una consulta XQuery se define en términos de un modelo de datos. Dicho modelo de datos de la consulta proporciona una representación abstracta de uno o más documentos XML (o fragmentos de documentos).

Las principales características de este modelo de datos son:

- Se basa en la **definición de secuencia**, como una colección ordenada de cero o más ítems. Éstas pueden ser heterogéneas, es decir pueden contener varios tipos de nodos y valores atómicos. Sin embargo, una secuencia nunca puede ser un ítem de otra secuencia.
- **Orden del documento**: corresponde al orden en que los nodos aparecerían si la jerarquía de nodos fuese representada en formato XML, (si el primer carácter de un nodo ocurre antes que el primer carácter de otro nodo, lo precederá también en el orden del documento).
- Contempla un **valor especial llamado "error value"** que es el resultado de evaluar una expresión que contiene un error.

Debes conocer

En el siguiente enlace puedes encontrar el estándar de XQuery aprobado por el W3C en diciembre de 2010. Recomendación del W3C sobre XQuery version 1.0.

3.4. EXPRESIONES

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.

El valor de una expresión es una secuencia heterogénea de nodos y valores atómicos. La mayoría de las expresiones están compuestas por la combinación de expresiones más simples unidas mediante operadores y palabras reservadas.

Ya hemos visto que Xpath es un lenguaje declarativo para la localización de nodos y fragmentos de información en árboles XML. Puesto que XQuery ha sido construido sobre la base de Xpath y realiza la selección de información y la iteración a través del conjunto de datos basándose en dicho lenguaje, toda expresión XPath también es una consulta Xquery válida.

Los comentarios en XQuery están limitados entre caras sonrientes, es decir:

(: Esto es un comentario XQuery :).

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

XQuery admite expresiones condicionales del tipo **if-then-else** con la misma semántica que tienen en los lenguajes de programación habituales.

Las consultas XQuery pueden estar formadas por hasta cinco tipos de cláusulas diferentes, siguen la norma FLWOR (que se pronuncia "flower"). Estas cláusulas son los bloques principales del XQuery, equivalen a las cláusulas **select, from, where, group by, having, order by** y **limit** de **SQL.**

En una sentencia FLWOR al menos ha de existir una cláusula FOR o una LET, el resto, si existen, han de **respetar escrupulosamente** el orden dado por el nombre, FLWOR.

- **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

Con estas sentencias se consigue buena parte de la funcionalidad que diferencia a 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**: Lugar donde se declaran los espacios de nombres, de funciones, variables, etc.
- **Expresión**: Consulta propiamente dicha.

Los comentarios en XQuery, a diferencia de XML, van encerrados entre caras sonrientes, tal y como se muestra a continuación:

(: Esto es un feliz comentario :)

Si se incluyen varias consultas en un mismo archivo .xquery para su ejecución conjunta, cada una de las consultas se separarán con una coma (,)

[Consulta 1]
,
[Consulta 2]

3.5. CLAÚSULAS

Hemos visto el modo de crear sentencias FLWOR, vamos ahora a estudiar aisladamente cada una de las cláusulas que pueden formar estas sentencias.

- **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 modificador **descending** para cambiar el sentido del orden.
- **RETURN:** construye el resultado de la expresión FLWOR para una tupla dada.

Existen una serie de reglas que debe cumplir cualquier consulta escrita en XQuery:

- For y let sirven para crear las tuplas con las que trabajará el resto de las cláusulas de la consulta y pueden usarse tantas veces como se desee en una consulta, incluso dentro de otras cláusulas. Sin embargo, solo puede declararse una única cláusula where, una única cláusula order by y una única cláusula return.
- Ninguna de las cláusulas FLWOR es obligatoria en una consulta XQuery. Por ejemplo, una expresión XPath, como la que se muestra a continuación, es una consulta válida y no contiene ninguna de las cláusulas FLWOR.

Ejercicio resuelto

A continuación tienes la solución de este ejercicio Descargar solución.

Para poder probar este y cualquier otro ejemplo con consultas XQuery es necesario tener instalada una base de datos XML nativa, como, por ejemplo: BaseX, donde hay que tener la base de datos creada y abierta a partir del fichero XML (en este ejemplo: libros.xml).

- Página oficial de BaseX
- Instalación de BaseX y creación de una base de datos
- Creación de consultas XQuery con BaseX

Veamos unos ejemplos de la forma de uso de estas cláusulas. Para ello partiremos del fichero XML de datos **libros.xml**:

1

```
<?xml version="1.0" encoding="UTF-8" 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>416</paginas>
      </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>
```

2

```
         <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>
      </libro>

      <libro publicacion="2005">
         <titulo> Beginning XSLT 2.0: Form Novice to Professional</titulo>
```

3

```
        <lector>
          <nombre>Jose</nombre>
          <apellidos>Gutiérrez González</apellidos>
          <direccion>Rue Percebe, 13</direccion>
          <telefono>919485432</telefono>
        </lector>
      </prestamo>
    </entrada>

    <entrada>
      <titulo>XPath Essentials</titulo>

      <prestamo>
        <inicio>2011-02-23</inicio>
        <devolucion>2011-03-10</devolucion>

        <lector>
          <nombre>Pepito</nombre>
          <apellidos>Grillo</apellidos>
          <direccion>Rue Percebe, 13</direccion>
          <telefono>972987654</telefono>
        </lector>
      </prestamo>
    </entrada>
  </prestamos>
```

4

```
        <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>
```

Y del fichero **prestamos.xml**:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<!-- Fichero: prestamos.xml -->

<prestamos>
  <entrada>
    <titulo>XML Imprescindible</titulo>

    <prestamo>
      <inicio>2011-05-02</inicio>

      <lector>
        <nombre>Pepito</nombre>
        <apellidos>Grillo</apellidos>
        <direccion>Rue Percebe, 13</direccion>
        <telefono>972987654</telefono>
      </lector>
    </prestamo>
  </entrada>

  <entrada>
    <titulo>XML Imprescindible</titulo>

    <prestamo>
      <inicio>2011-02-12</inicio>
      <devolucion>2011-02-16</devolucion>
```

1. Ejemplo de uso de la cláusula FOR. Obtener todos los títulos de los libros del fichero libros.xml.
2. Ejemplo de uso de la cláusula LET. Obtener todos los títulos de los libros del fichero libros.xml.
3. Ejemplo de uso de la cláusula FOR y LET juntas. Obtener todos los títulos de los libros del fichero libros.xml junto con los autores de cada libro.
4. Ejemplo de uso de las cláusulas WHERE y ORDER BY en una consulta con dos ficheros. Obtiene los títulos de los libros prestados con sus autores y la fecha de inicio y devolución del préstamo, ordenados por la fecha de inicio del préstamo.

1. Ejemplo de uso de la cláusula FOR. Obtener todos los títulos de los libros del fichero libros.xml.

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

2. Ejemplo de uso de la cláusula LET. Obtener todos los títulos de los libros del fichero libros.xml.

```
let $d := doc("libros.xml")/biblioteca/libros/libro/titulo
  return <titulos>{ $d }</titulos>
```

3. Ejemplo de uso de la cláusula FOR y LET juntas. Obtener todos los títulos de los libros del fichero libros.xml junto con los autores de cada libro.

```
for $b in doc("libros.xml")//libro
  let $c := $b/autor
  return <libro>{ $b/titulo, <autores>{ $c }</autores>}</libro>
```

Otra solución posible a este ejercicio, en este caso utilizando únicamente la cláusula FOR

```
for $b in doc("libros.xml")//libro
  return <libro>{ $b/titulo, <autores>{$b/autor}</autores>}</libro>
```

4. Ejemplo de uso de las cláusulas WHERE y ORDER BY en una consulta con dos ficheros. Obtiene los títulos de los libros prestados con sus autores y la fecha de inicio y devolución del préstamo, ordenados por la fecha de inicio del préstamo.

```
for $t in doc("libros.xml")//libro,
$e in doc("prestamos.xml")//entrada
  where $t/titulo = $e/titulo
  order by $e/prestamo/inicio
  return <prestamo>{ $t/titulo, $t/autor/*, $e/prestamo/inicio, $e/prestamo/devolucion }</prestamo>
```

3.6. DIFERENCIAS ENTRE EL FOR Y LET

Para ver claramente la diferencia entre una cláusula for y una cláusula let vamos a comenzar estudiando una misma consulta que muestre los títulos de todos los libros almacenados en el archivo “libros.xml”, primero con una cláusula for y, a continuación, con una cláusula let y vamos a detallar que diferencia hay en la información obtenida.

La consulta con una cláusula for se muestra a continuación:

```
for
  $d in doc("libros.xml")/bib/libro/titulo
return
  <titulos>{ $d }</titulos>
```

El resultado de esta consulta se muestra a continuación:

```
<titulos>
    <titulo>TCP/IP Illustrated</titulo>
</titulos>
<titulos>
    <titulo>Advan Programming for Unix environment</titulo>
</titulos>
<titulos>
    <titulo>Data on the Web</titulo>
</titulos>
<titulos>
    <titulo> Economics of Technology for Digital TV</titulo>
</titulos>
```

A continuación, repetimos la misma consulta sustituyendo la cláusula for una cláusula let:

```
let
  $d := doc("libros.xml")/bib/libro/titulo
return
  <titulos>{ $d }</titulos>
```

El resultado de esta consulta se muestra a continuación:

```
<titulos>
    <titulo>TCP/IP Illustrated</titulo>
    <titulo>Advan Programming for Unix environment</titulo>
    <titulo>Data on the Web</titulo>
    <titulo> Economics of Technology for Digital TV</titulo>
</titulos>
```

Como se puede ver comparando los resultados obtenidos por ambas consultas:

- **La cláusula for** vincula una variable con cada nodo que encuentre en la colección de datos. En este ejemplo la variable $d va vinculándose a cada uno de los títulos de todos los libros del archivo "libros.xml", creando una tupla por cada título. Por este motivo aparece repetido el par de etiquetas <titulos>...</titulos> para cada título.
- **La cláusula let**, en cambio, vincula una variable con todo el resultado de una expresión. En este ejemplo, la variable $d se vincula a todos los títulos de todos los libros del archivo "libros.xml", creando una única tupla con todos esos títulos. Por este motivo solo aparece el par de etiquetas <titulos>...</titulos> una única vez.

Para saber más

Si una cláusula let aparece en una consulta que ya posee una o más cláusulas for, los valores de la variable vinculada por la cláusula let se añaden a cada una de las tuplas generadas por la cláusula for. Un ejemplo se muestra en la siguiente consulta:

```
for
  $b in doc("libros.xml")//libro
let
  $c := $b/autor
return
  <libro>{ $b/titulo, <autores>{ count($c) }</autores>}</libro>
```

Esta consulta devuelve el título de cada uno de los libros de archivo "libros.xml" junto con el número de autores de cada libro. El resultado de esta consulta se muestra a continuación:

```
<libro>
    <titulo>TCP/IP Illustrated</titulo>
    <autores>1</autores>
</libro>
<libro>
    <titulo>Advanced Programming in the UNIX Environment</titulo>
    <autores>1</autores>
</libro>
<libro>
    <titulo>Data on the Web</titulo>
    <autores>3</autores>
</libro>
<libro>
    <titulo>The Economics of Technology and Content for Digital TV</titulo>
    <autores>0</autores>
</libro>
```

Si en la consulta aparece más de una cláusula for (o más de una variable en una clásula for), el resultado es el producto cartesiano de dichas variables, es decir, las tuplas generadas cubren todas las posibles combinaciones de los nodos de dichas variables. Un ejemplo se muestra en la siguiente consulta, la cual devuelve los títulos de todos los libros contenidos en el archivo "libros.xml" y todos los comentarios de cada libro contenidos en el archivo "comentarios.xml".

```
for
  $t in doc("libros.xml")//titulo,
  $e in doc("comentarios.xml")//entrada
where
  $t = $e/titulo
return
  <comentario>{ $t, $e/comentario }</comentario>
```

El resultado de esta consulta se muestra a continuación:

```
<comentario>
    <titulo>Data on the Web</titulo>
    <comentario>Un libro muy bueno sobre bases de datos.</comentario>
</comentario>
<comentario>
    <titulo>Advanced Programming in the Unix environment</titulo>
    <comentario>Un libro claro y detallado de programación en UNIX.</comentario>
</comentario>
<comentario>
    <titulo>TCP/IP Illustrated</titulo>
    <comentario>Uno de los mejores libros de TCP/IP.</comentario>
</comentario>
```

3.7. FUNCIONES

Ahora que conocemos las cláusulas que sustentan el lenguaje **XQuery**, vamos a ocuparnos de las **funciones que soporta.**

Estas son 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. Las funciones más importantes se muestran a continuación:

- **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.

- **Cuantificadores existenciales:**
  - **some,every**, permiten definir consultas que devuelven algún, o todos los elementos, que verifiquen la condición dada.

Además de estas funciones que están definidas, en el lenguaje XQuery se permite al desarrollador construir sus propias funciones:

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

Ejercicio resuelto

Dados los ficheros XML del ejemplo anterior, hacer una consulta que devuelva los títulos de los libros almacenados en el fichero libros.xml y su primer autor. En caso de que haya más de un autor para un libro se añade un segundo autor "cia".

A continuación puedes descargar la solución de este ejercicio: Descargar solución

```
for $b in doc("libros.xml") //libro
 return
 <libro>
  { $b/titulo }
  {
   for $a at $i in $b/autor
    where $i <= 1
    return <autor>{string($a/nombre), ", ",
    string($a/apellido)}</autor>
  }
  {
   if (count($b/autor) > 1)
   then <autor>cia.</autor>
   else ()
  }
 </libro>
```

Ejercicio resuelto

Ejemplo de la creación y uso de funciones de usuario:

A continuación puedes descargar la solución de este ejercicio: Descargar solución

Se usará el fichero **libros_precio.xml**, parecido al de libros.xml pero se la incorporado las etiquetas del precio y el descuento.

1

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<!-- Fichero: libros_precio.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>416</paginas>
      <precio>55.77</precio>
      <!-- porcentaje de descuento -->
      <descuento>20</descuento>
    </libro>
```

2

```
<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>
  <precio>90</precio>
  <descuento>10</descuento>
</libro>
```

3

```
<libro publicacion="2002">
    <titulo>XML Schema</titulo>
    <autor>
      <apellido>van der Vlist</apellido>
      <nombre>Eric</nombre>
    </autor>
    <editorial>O'Reilly</editorial>
    <paginas>400</paginas>
    <precio>40.25</precio>
    <descuento>30</descuento>
  </libro>
```

4

```
<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>
  <precio>65</precio>
  <descuento>50</descuento>
</libro>

<libro publicacion="2007">
  <titulo> XQuery</titulo>
  <autor>
    <apellido>Walmsley</apellido>
    <nombre>Priscilla</nombre>
  </autor>
  <editorial>O'Reilly</editorial>
  <paginas>491</paginas>
  <precio>46.24</precio>
  <descuento>25</descuento>
</libro>
</libros>

biblioteca>
```

La consulta XQuery, que incorpora la función de usuario es:

```
(: Función local definida por el usuario:
 Entrada: precio y descuento
 Salida: devuelve el precio al aplicarle el descuento :)
declare function local:minPrice ($precio as xs:decimal,$descuento as xs:decimal) as xs:decimal?
{
 let $descuento_aplicado := ($precio * $descuento) div 100
 return ($precio - $descuento_aplicado)
};

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

for $libro in doc("libros_precio.xml")/biblioteca/libros/libro
 return
 <minPrice>{local:minPrice(xs:decimal($libro/precio),xs:decimal($libro/descuento))}</minPrice>

(:Un ejemplo de cómo invocar a la función sin la consulta es:)
(:local:minPrice(xs:decimal(8), xs:decimal(7)):)
```

3.8. OPERADORES

Veamos ahora algunos de los operadores más importantes agrupados según su funcionalidad:

- **Comparación 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.
    - **ne**, no igual.
    - **lt**, menor que.
    - **le**, menor o igual que.
    - **gt**, mayor que.
    - **ge**, mayor o igual que.
- **Comparaciones generales:** Permiten comparar operandos que sean secuencias.
    - **=**, igual.
    - **!=**, distinto.
    - **>**, mayor que.
    - **>=**, 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.