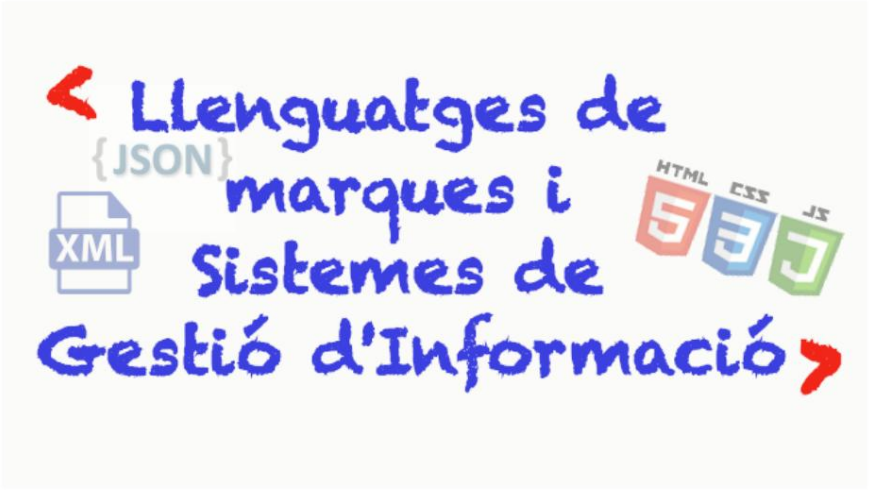
UNIDAD 7. ALMACENAMIENTO DE INFORMACIÓN



Contenido

1. Introducción

Los archivos en XML están en un formato de texto estandarizado que sirve para representar, almacenar y transportar información estructurada. Por tanto, XML puede utilizarse como método de almacenamiento de datos, ya que la información almacenada en archivos XML puede moverse de una máquina a otra. Esto le hace un método ideal para almacenar datos históricos que nos asegura que se podrán volver a leer en el futuro sin problemas.

Esto ha hecho que las organizaciones cada vez tengan una mayor cantidad de documentos XML, y que en consecuencia localizar los documentos en el momento adecuado sea cada vez más complicado. Por tanto es necesaria alguna manera de organizar los datos XML que permita localizarlos rápidamente.

Pero no sólo basta con localizar los documentos, ya que a menudo lo que se busca estará repartido en diferentes documentos. También hace falta alguna manera de poder consultar y manipular los datos que contienen los archivos XML. Este sistema de consulta debe ser capaz de realizar búsquedas en los documentos y modificar datos si es necesario, pero siempre manteniendo unos mínimos de eficiencia.

Ante estas necesidades la solución suele darse de dos maneras:

- Debido a que mayoritariamente las organizaciones ya tienen sistemas de almacenamiento organizados de datos basados en datos relacionales una posible solución podría ser convertir los archivos XML a datos relacionales:
  - El almacenamiento de datos será totalmente organizado y centralizado en un punto donde ya están los demás datos de la organización.
  - Su lenguaje de consulta, SQL, es muy conocido y se utiliza mucho, por lo que no será necesario que los usuarios aprendan un nuevo lenguaje.
- Otra solución consiste en crear un sistema de almacenamiento pensado sólo para los archivos XML. Son los sistemas conocidos como bases de datos nativas XML.
  Estos:
  - Proporcionan un sitio para almacenar ordenadamente archivos XML.
  - Tienen su lenguaje de consulta propio: XQuery.

2. Bases de datos relacionales

La inclusión de los archivos XML en los sistemas gestores de bases de datos relacionales puede hacerse de dos formas:

1. Convertir los datos de los archivos XML en datos relacionales:
   - Este sistema tiene la ventaja de que los datos, una vuelta dentro del sistema relacional, serán idénticos a los ya existentes.
   - El inconveniente más importante es que si hace falta el documento XML original puede ser muy difícil regenerarlo, ya que a menudo habrá información sobre la estructura XML que no se almacenará.
2. Almacenar los documentos XML enteros en las bases de datos:
   - Se pondrán los archivos XML enteros en un campo de una tabla de la base de datos, por lo que será como los demás datos.
   - Para poder trabajar bien será necesario que la base de datos ofrezca alguna manera medianamente eficiente de poder realizar búsquedas en el contenido de los documentos XML.

2.1.1. Convertir los datos XML en relacionales

Debido a que en el documento XML ya está definida la estructura de los datos, ya menudo los documentos XML estarán asociados a un vocabulario, tendremos que:

- Generar la estructura de tablas relacionales se puede realizar analizando la estructura del documento XML.
- Se pueden obtener los campos de la definición del vocabulario o simplemente observando el contenido del documento XML.

A pesar de la aparente sencillez de este sistema, no siempre es sencillo realizar esta conversión, ya que los sistemas relacionales y XML parten de conceptos bastante diferentes:

- El sistema relacional está basado en datos bidimensionales sin jerarquía ni orden, mientras que el sistema XML está basado en árboles jerárquicos en los que el mandato es significante.
- En un documento XML puede haber datos repetidos, mientras que los sistemas relacionales huyen de las repeticiones.
- Las relaciones y estructuras dentro de los documentos XML no siempre son obvias.
- ¿Qué ocurre si necesitamos tener el documento XML de nuevo? Hacer el proceso inverso no siempre es trivial. Uno de los conceptos difíciles es determinar qué datos eran atributos y elementos.

Por tanto, normalmente éste no es el sistema más aconsejable pero es un sistema válido que puede ser útil en casos concretos.

2.1.2. Sistemas relacionales con extensiones XML

Todos los grandes sistemas de bases de datos importantes como Oracle, IBM DB2 o Microsoft SQL Server tienen algún tipo de soporte para XML, que normalmente se concreta en lo siguiente:

- Permitir exportar los datos relacionales a algún formato XML para transportar los datos.
- Tener alguna forma de almacenar documentos XML como campos en tablas relacionales.
- Permitir realizar búsquedas y cambios en los documentos XML almacenados.
- Generar XML a partir de los datos relacionados de la base de datos.

Dado que SQL no tenía soporte para XML en 2003, se modificó el estándar del lenguaje SQL para añadir la extensión SQL/XML.

SQL/XML es una extensión del estándar SQL que permite trabajar con el lenguaje XML por medio de instrucciones SQL.

SQL/XML define toda una serie de funciones para publicación de archivos XML a partir de datos relacionales, define un tipo de datos XML y una forma de consultar y manipular los datos XML almacenados.

3. XQUERY

Una de las necesidades más básicas a la hora de poder realizar búsquedas en los datos es disponer de un lenguaje para realizar consultas que sea suficientemente potente para poder cubrir las necesidades de quienes lo deben utilizar. De ahí que se desarrolló el lenguaje XQuery.

Se trata de un lenguaje funcional, por lo que en vez de decirle cuáles son los pasos para realizar una tarea lo que se hace es evaluar las expresiones contra el archivo XML y generar un resultado.

A diferencia de los lenguajes de programación habituales, en XQuery se especifica qué es lo que se quiere y no la forma en que debe hacerlo para obtenerlo.

Entre las características más interesantes de XQuery, éste permite:

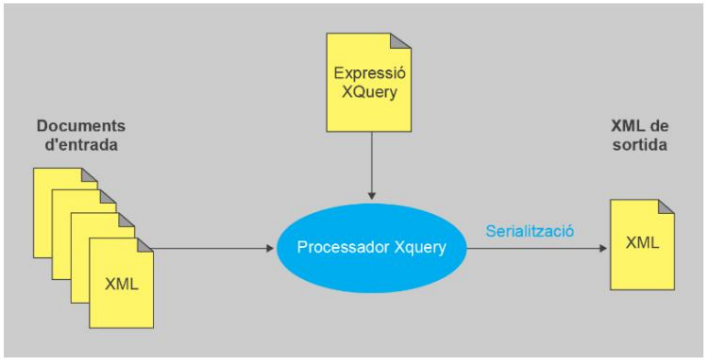
- Seleccionar información según criterios. Ordenar, agrupar, añadir datos.
- Filtrar la información que se desee del flujo de datos.
- Buscar información en un documento o en un grupo de documentos.
- Unir datos de múltiples documentos.
- Transformar y reestructurar XML.
- No estar limitado a la búsqueda, ya que puede realizar operaciones numéricas y de manipulación de caracteres.
- Puede trabajar con espacios de nombres y con documentos definidos por medio de DTD o XSD.

Una parte importante de XQuery es el lenguaje XPath, que es la parte que le permite realizar las selecciones de información y la navegación por el documento.

3.1. Procesadores XQuery

La evaluación de expresiones XQuery requerirá disponer de algún procesador XQuery.

Los procesadores XQuery cogen los documentos XML de entrada y les aplican una transformación para generar un resultado.



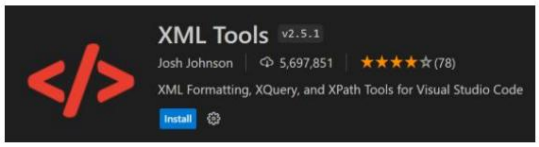
1 Funcionamiento de los procesadores XQuery

Normalmente los procesadores XQuery vienen en forma de bibliotecas que se pueden incorporar a los programas. Muchos sistemas de base de datos o programas incorporan estas bibliotecas para poder apoyar a XQuery en sus productos.

3.1.1. XQUERY en Visual Studio Code

Podemos utilizar el editor que hemos utilizado en otras unidades de este curso, Visual Studio Code, si instalamos extensiones que apoyan a Xquery

Una de las extensiones más populares para llevar a cabo esta tarea es XML Tools, del autor Josh Johnson, que ya hemos utilizado para evaluar expresiones Xpath.



2 XML Tools para VS Code

3.2. Consultas XQuery

Generalmente una consulta XQuery consistirá en conseguir los datos con los que se desea trabajar, filtrarlos y devolver el resultado como XML. La expresión más simple que puede hacerse en XQuery es escribir un elemento vacío:

```
<Hola />
```

Al ejecutar esta orden el resultado será el siguiente:

```
<?xml version="1.0" ?>
<Hola/>
```

Ésta es una de las características importantes de XQuery: que escribe literalmente el texto; mientras que si en el resultado se definen etiquetas, la expresión sólo será correcta si el resultado final está bien formado. Por ejemplo, ejecutar lo siguiente en XQuery dará un error, ya que el resultado final no está bien formado:

```
<a>
```

Pero a pesar de poder escribir texto, lo corriente es recuperar algún tipo de datos, evaluarlos y devolver el resultado con un return. La instrucción return se encarga de generar las salidas.

Otra característica importante es que utiliza variables. Por ejemplo la siguiente es una expresión XQuery correcta que asigna el valor 5 a $a y después devuelve el valor:

```
let $a:=5
return 5
```

El resultado de ejecutarlo con XQuery será:

```
<?xml version="1.0" ?>
5
```

A diferencia de lo que ocurre con los literales, el return se ejecuta una vez por cada valor que recibe, de modo que si hubiera una expresión que se diera más de una vuelta:

```
para $a in (1,2)
returno <Hola/>
```

El return se ejecutaría más de una vuelta:

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

3.3. Comentarios

Los comentarios se definen poniéndolos entre el símbolo ”(:” y el ”:)”. También se pueden realizar comentarios multilínea.

```
(: Comentario :)
```

Todo lo que esté dentro de un comentario será ignorado por el procesador.

3.4. Carga de documentos

Una de las cosas necesarias para interrogar un archivo o archivos XML es establecer específicamente en qué documentos se quiere hacer la petición.

Función	Uso
doc()	Decir cuál es el documento que queremos consultar.
collection()	Especificar un grupo de documentos a la vez. Generalmente será un documento con enlaces a otros o una dirección de una base de datos XML.

Por tanto, podemos utilizar la función doc() para cargar un documento XML. Con esta instrucción XQuery devolverá todo el documento alumnos.xml.

```
return doc("alumnos.xml")
```

XQuery contiene completamente XPath, de modo que se puede utilizar cualquier expresión XPath para filtrar resultados cuando sea necesario, y por tanto se puede hacer que en el mismo proceso de carga se aplique un filtro. Por ejemplo una expresión XQuery que devuelve cuáles son los nombres de los alumnos del archivo alumnos.xml puede ser la siguiente:

```
return doc("alumnos.xml")//nombre
```

Los documentos pueden estar en cualquier lugar al que se pueda llegar. En este caso, por ejemplo, la expresión XQuery puede ser una URL:

```
return doc("http://direccio.org/alumnos.xml")
```

3.5. Variables

XQuery es un lenguaje pensado para realizar búsqueda en documentos XML, pero no sólo se puede utilizar para realizar búsqueda, ya que tiene soporte para realizar operaciones aritméticas y para trabajar con cadenas de caracteres.

En este ejemplo utilizamos XQuery para realizar una operación matemática sencilla:

```
let $x := 5

let $y := 4

return $x + $y
```

El resultado será:

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

9
```

Una característica de XQuery que le diferencia de otros lenguajes de búsqueda es que dispone de variables:

- Las variables en XQuery se identifican porque comienzan siempre con el símbolo $.
- Pueden contener cualquier valor: literales numéricos o caracteres, secuencias de nodos…
- La instrucción para asignar valores a una variable es let.

Uno de los usos fundamentales de las variables es almacenar elementos para poder utilizarlos posteriormente. En el ejemplo siguiente se almacenan los elementos <nombre> en la variable $alumnos y después devuelve la cantidad de alumnos:

```
let $alumnos := doc("alumnos.xml")//alumnos/nombre
return count($alumnos)
```

Se pueden aplicar filtros XPath a las variables:

```
let $tot := doc("clase1.xml")
let $profe := $tot//profesor
let $alum := $todo//alumno
```

Los valores de las variables los podremos comparar con los operadores de comparación habituales:

Operador	Operador2	Uso
=	eq	Dos valores son iguales
!=	ne	Dos valores son diferentes
>	gt	Es mayor que el otro
>=	ge	Es mayor o igual
<	lt	Más pequeño
	le	Más pequeño o igual

Los operadores eq, ne, gt, ge, lt y le sólo se pueden utilizar para comparar valores individuales, y por tanto sólo se podrán utilizar si existe un esquema definido.

3.6. Expresiones evaluables

XQuery está pensado para poder mezclar las consultas con cualquier otro tipo de contenido. Si se mezcla con contenido, las expresiones a evaluar deben colocarse entre claves ”{…}”.

Por ejemplo, podemos mezclar HTML y XQuery para que el procesador XQuery genere una página web con los datos de un documento XML.

```
<html>
<head><title>Lista de clase</title></head>
<body>
<h1>
{
doc("clase1.xml")//asignatura
}
</h1>
</body>
</html>
```

Al mezclar contenido, el procesador XQuery sólo evaluará las instrucciones que estén dentro de las claves, y por tanto dejará las etiquetas HTML tal y como están.

3.6.1. Creación de elementos y atributos

Usando las expresiones evaluables es fácil crear nuevos elementos.

```
<módulo>
    { doc("clases.xml")//modul/text()}
</módulo>
```

También se pueden crear atributos (hay que tener cuidado de no dejarse las comillas en torno al valor que obtendrán los atributos).

```
<modul nombre="{doc("clases.xml")//modul/text()}"/>
```

Una forma alternativa y más potente de definir elementos y atributos es utilizar las palabras clave elemento y atributo. Estas instrucciones permiten crear elementos y definir su contenido especificándolo dentro de las claves.

```
elemento modulo {
}
```

Crearía un elemento <módulo> vacío:

```
<módulo />
```

Si queremos crear nuevos elementos o atributos dentro de un elemento definido de esta forma deben especificarse dentro de las claves. Por ejemplo, el siguiente código:

```
elemento modulo {
    attribute nombre {"Lenguajes de marcas"},
    elemento alumnos { }
}
```

Generará lo siguiente:

```
<modulo nombre="Lenguajes de marcas">
    <alumnos />
</módulo>
```

No existe ninguna limitación a la hora de imbricar los elementos y atributos para generar resultados tan complejos como sea necesario, y en cualquier momento se puede poner una expresión XQuery.

```
elemento modulo {
    attribute nombre { doc("clase.xml")//modul/text() }
}
```

3.7. Expresiones FLWOR

Las expresiones FLWOR son la parte más potente de XQuery. Están formadas por cinco instrucciones opcionales que permiten realizar las consultas y al mismo tiempo procesarlas para seleccionar los ítems que interesan de una secuencia.

Carácter	Comando	Uso
f	for	Permite pasar de un elemento a otro de la secuencia
l	let	Sirve para asignar valores a una variable
w	where	Permite filtrar los resultados según condiciones
o	order by	Usado para ordenar los resultados antes de mostrarlos
r	return	Devuelve el resultado de toda la expresión

for ... return

La instrucción for se utiliza para evaluar individualmente cada uno de los resultados de una secuencia de nodos. En un foro se definen dos cosas:

- Una variable, que será la que irá cogiendo los resultados de la secuencia uno por uno.
- La palabra clave in, en la que se define cuál es la secuencia de elementos a procesar.

La forma más sencilla de expresión FLWOR es combinar el for con el return para devolver los valores obtenidos de forma secuencial:

```
for $i in ('Hola', 'Adiós')
return $i
```

Generará:

```
Hola Adiós
```

Las secuencias de valores pueden ser de cualquier tipo. Por ejemplo, pueden ser números:

```
for $i in (1,2,3)
return $i*$i
```

Que devuelve:

```
1 4 9
```

O una secuencia de nodos. Por ejemplo con la siguiente expresión capturamos una secuencia de nodos <alumno> y los utilizamos para obtener su apellido:

```
for $alumno in doc("clase.xml")//alumno
return <alumno> { $alumno/apellido } </alumno>
```

El resultado será:

```
<?xml version="1.0" encoding="UTF-8"?>
    <alumno>
        <apellido>Pi</apellido>
    </alumno>
    <alumno>
        <apellido>Garcia</apellido>
    </alumno>
```

Puede verse que los valores del return se han ido procesando uno por uno y por este motivo se repiten las etiquetas <alumno>.

Un aspecto importante es que no es necesario que el foro sea la primera expresión de la consulta. También puede ponerse como parámetro en una función XPath.

Por ejemplo, la siguiente expresión nos devolverá el número de alumnos:

```
count (
  for $alumno in doc("alumnos.xml")//nombre
  return $alumno
)
```

for ... at

En el for se puede incluir el operador at, que permite obtener la posición del nodo que se está procesando. Con este operador puede hacerse que aparezca el número de orden en los resultados.

```
for $alumno en $pos in doc("clase.xml")//alumno
return <alumno numero="{$pos}">
            { $alumno/apellido/texto() }
        </alumno>
```

Mostrará:

```
<?xml version="1.0" encoding="UTF-8"?>
<alumno numero="1">Pi</alumno>
<alumno numero="2">Garcia</alumno>
<alumno numero="3">Puigdevall</alumno>
```

"for" con múltiples variables

Hasta ahora sólo se ha utilizado una variable en el for pero se pueden poner tantas como haga falta separándolas por comas:

```
<resultados>
{
for $tot in doc("clase.xml")/clase,
    $nombre in $todo/alumnos,
    $modul in $tot/asignatura/texto()
return
  <módulo nombre="{ $módulo }"
            alumnos="{ count($nombre/alumno)}">
          { $nombre/alumno/nombre }
  </módulo>
}
</resultados>
```

Que dará el siguiente resultado:

```
<?xml version="1.0" encoding="UTF-8"?>
<resultados>
    <módulo alumnos="3" nombre="Lenguajes de marcas">
        <nom>Frederic</nom>
        <nom>Filomeno</nom>
        <nombre>Manel</nombre>
    </módulo>
</resultados>
```

"for" anidado

Las órdenes for se pueden imbricar entre ellas de forma similar a como lo hace SQL para poder conseguir resultados más complejos que se basan en los resultados obtenidos anteriormente. Ejemplo:

```
for $selec in doc("clase.xml")//alumno
return
<persona>
    { $selec/nombre }
    {
        for $selec2 in $selec
        return $selec2//apellido
    }
</persona>
```

let

let permite declarar variables y asignarles un valor. Sobre todo se utiliza para almacenar valores que deben utilizarse más tarde.

```
let $num := 1
let $nombre := "manel"
let $i := (1 to 3)
```

En las expresiones FLWOR puede haber tantos años como sea necesario.

```
for $alumno in doc("notas.xml")//alumno
let $nombre := $alumno/nombre
let $nota := $alumno/nota
return <nom>concado($nombre,":",$nota)
```

Hay que tener cuidado con let porque su funcionamiento es muy diferente al de for:

- for se ejecuta para cada miembro de una secuencia. • let hace referencia a su valor en su conjunto. Si es una secuencia lo que se procesa son todos los valores de golpe.

Podemos ver la diferencia con un ejemplo. Esta instrucción con for:

```
for $selec in doc("clases.xml")//alumno
return <persona>{$selec/nombre}</persona>
```

Mostrará:

```
<?xml version="1.0" encoding="UTF-8"?>
<persona><nom>Frederic</nom></persona>
<persona><nom>Filomeno</nom></persona>
<persona><nom>Manel</nom></persona>
```

Y en cambio con let:

```
let $selec := doc("clases.xml")//alumno
return <persona>{$selec/nombre}</persona>
```

Mostrará:

```
<?xml version="1.0" encoding="UTF-8"?>
<persona>
    <nom>Frederic</nom>
    <nom>Filomeno</nom>
    <nombre>Manel</nombre>
</persona>
```

Los resultados son bastante diferentes, puede verse claramente que let ha tratado todos los valores juntos.

Condiciones

La instrucción where es la parte que indicará qué filtro debe ponerse en los datos recibidos antes de enviarlos a la salida del return. Normalmente en el filtro se utiliza algún tipo de predicado XPath:

```
for $alumno in doc("clase.xml")//alumno
where $alumno/@aprobado="si"
return $alumno/nombre
```

Debido a que XPath forma parte de XQuery muchas veces el mismo filtro se puede definir de varias formas. Por ejemplo, esta expresión es equivalente a la anterior:

```
for $alumno in doc("clase.xml")//alumno[@aprobado="si"]
return $alumno/nombre
```

En un where puede haber tantas condiciones como sea necesario simplemente encadenándolas con las operaciones lógicas and, or o not()

Esta expresión devuelve el apellido de los alumnos que han aprobado y que se llaman Pedro:

```
for $alumno in doc("clase.xml")//alumno
where $alumno/@aprobado="si" and nombre="Pere"
return $alumno/apellido
```

A partir de dos archivos con los datos de dos clases diferentes se pueden obtener sólo los alumnos que se repiten entre las dos clases con:

```
for $alum1 en doc("clase1.xml")//alumno
let $alum2 in doc("clase2.xml")//alumno
where $alum1 = $alum2
return $alum1
```

Ordenación

Una de las operaciones habituales en las búsquedas es representar los resultados en un determinado orden. Ésta es la función que hace order by.

Se puede ordenar de forma ascendente con ascending (que es lo que hace por defecto) o bien descendente con descending.

```
for $alumno in doc("clase.xml")//alumno
order by $alumno/apellido
return $alumno
```

También se pueden especificar distintos conceptos en la ordenación. En este ejemplo los resultados se ordenarán primero por apellido y después por nombre:

```
for $alumno in doc("clase.xml")//alumno
order by $alumno/apellido, $alumno/nombre
return $alumno
```

En este otro el apellido se ordena de forma descendente y el nombre de forma ascendente.

```
for $alumno in doc("clase.xml")//alumno
order by $alumno/apellido descending, $alumno/nombre ascending
return $alumno
```

Por defecto ordena como si el contenido de las etiquetas fuera texto. Si es necesario obtener una ordenación numérica habrá que convertir los valores a números con la función de XPath number().

```
for $alumno in doc("clase.xml")//alumno
order by number($numero)
return $alumno
```

Alternativas

La instrucción if nos permitirá realizar una cosa u otra según se cumple la condición especificada o no. Por ejemplo, podemos realizar una lista de alumnos en la que se especifique si han aprobado o no a partir del valor que hay en el elemento <nota>.

```
let $doc := doc("clase1.xml")
let $aprobado := 5
for $b in $doc//alumno
return
if ($b/nota >= 5) then
<fecha>
{ $b/nombre/text() } está aprobado
</data>
else
<fecha>
{ $b/nombre/text() } no está aprobado
</data>
```

Esto generará el siguiente resultado:

```
<data>Frederic está aprobado</data>
<data>Filomeno no está aprobado</data>
<data>Manel está aprobado</data>
```

La instrucción else es habitual en lenguajes de programación, pero a diferencia de lo que ocurre normalmente, en XQuery, aunque no se quiera hacer nada, debe especificarse else obligatoriamente.
Si no se quiere que haga nada, se deja en blanco.

```
for $classe in doc("clase1.xml")//alumno
return if ($clase/nota > 5) then
          <aprobado/>
        else ()
```

4. Bases de datos nativos XML (NXD)

A diferencia de lo que ocurre con las bases de datos relacionales, que llevan muchos años funcionando y tienen detrás suyo una base teórica importante, las NXD no tienen unos estándares definidos para hacer las cosas y la teoría que las soporta no está definida. Esto hace que a menudo cada base de datos haga las cosas de una forma que puede ser totalmente diferente de cómo lo hace otra.

Las bases de datos nativas XML se utilizan sobre todo para almacenar datos que contienen:

- Contenido narrativo.
- Que son menos previsibles que las que se almacenan normalmente en bases de datos relacionales.
- Que deben generar salidas para entornos web.
- Que deben transportarse de un sistema a otro.

Los sistemas NXD son muy heterogéneos y con frecuencia se están desarrollando en direcciones que no siempre coinciden. A pesar de la dispersión, siempre podemos encontrar puntos en común en la mayoría de bases de datos.

Lenguaje de consulta

Todos los sistemas NXD soportan al menos XPath como lenguaje de consultas, aunque la tendencia general es adoptar XQuery; es lógico si se tiene en cuenta que XPath no fue pensado para ser un lenguaje de consulta de base de datos, mientras que XQuery sí.

Todos los sistemas soportan algún tipo de interfaz que permite acceder a ellos por medio de algún lenguaje de programación, como la interfaz XML:DB, o bien por medio de algún protocolo de red como HTTP, WebDAV, SOAP…

Organización y almacenamiento

La organización más habitual de los datos es mediante colecciones de documentos XML que a menudo intentan generar una estructura similar a la que ofrecen los directorios de los sistemas operativos. En estos sistemas una colección es como una carpeta que contiene un conjunto de documentos XML o incluso otras carpetas

El tratamiento de las colecciones suele ser una de las diferencias existentes entre bases de datos XML. Algunas fuerzan que dentro de una colección existan documentos del mismo tipo, mientras que otras permiten que se ponga cualquier tipo de documento.

El almacenamiento de los datos suele ser el que diferencia más a las NXD, ya que la naturaleza de los archivos XML no los hace ideales para ser almacenados de manera eficiente (tienen repeticiones, ocupan mucho espacio…). Por este motivo todas las NXD intentan optimizar de alguna manera este almacenamiento: convirtiendo los datos en archivos binarios (con simples compresiones de datos o codificándolos), creando estructuras de árbol, o simplemente almacenando los archivos XML sin hacer nada. confiando en que la reducción de costes del almacenamiento hará que ocupar espacio no sea un problema.

Indexación

A la hora de utilizar una base de datos lo que realmente importa al usuario es que haga la búsqueda en un tiempo aceptable. Los archivos de texto no son la mejor manera de almacenar datos si el objetivo es realizar búsquedas, y por tanto las bases de datos XML se han esforzado en intentar crear algún sistema para realizar estas búsquedas más eficientes.

Lo habitual suele ser:

- Organizar de forma eficiente las colecciones
- Utilizar índices

Seguridad

La seguridad es otro aspecto a tener en cuenta en los sistemas NXD. Generalmente en un entorno empresarial no todo el mundo puede acceder a los datos que desee sino que a la información pueden acceder sólo las personas autorizadas.

La identificación de usuarios permite implementar los mecanismos básicos para permitir controlar a qué documentos puede acceder cada uno de los usuarios de la organización.

Actualizaciones de los datos XML

La falta de un sistema de actualizaciones de los archivos XML en la primera versión de XQuery ha hecho que muy a menudo los sistemas NXD, en vez de actualizar los datos que han cambiado, optan por sustituir totalmente el documento cada vez que existe cambios.

4.1. eXist-db

De las bases de datos XML de código abierto disponibles, probablemente eXist-db es la más preparada para ser usada en un entorno profesional. Se trata de una base de datos nativa XML
que está desarrollada en lenguaje Java, y que:

- Permite realizar operaciones con los lenguajes XQuery 1.0, XPath 2.0 y XSLT 2.0.
- Puede funcionar tanto como una biblioteca que se incorpora a los programas como ser ejecutada por medio de un servidor.
- Se puede acceder a la base de datos por medio de múltiples interfaces estándares como REST, WebDAV, SOAP, XML-RPC o ATOM.
- Permite actualizaciones de datos por medio de la API XMLDB, XUpdate y XQuery Update Facility.
- Lleva incorporado un sistema de gestión de usuarios de la base de datos y de los permisos de acceso a las colecciones similar al de los sistemas Unix

La organización de los datos es por medio de colecciones de archivos XML que se organizan de una manera prácticamente idéntica a cómo se hace en los archivos de un sistema operativo. Al igual que en los sistemas operativos, se pueden realizar colecciones dentro de colecciones y se puede navegar por ellas con un solo clic.

Instalación

La instalación de la última versión estable (actualmente -abril 2024-6.2.0) se puede realizar descargando el paquete de instalación para el sistema operativo deseado desde el repositorio oficial de github (https://github.com /eXist-db/exist/releases/tag/eXist-6.2.0) o utilizando el contenedor docker oficial (https://hub.docker.com/r/existdb/existdb/tags/)

En la instalación por defecto además del eXist se instala el servidor Jetty, que permitirá acceder a la gestión web del sistema con un navegador en la dirección http://localhost:8080/exist/.

Si se utiliza el servidor, el acceso a las consultas y la administración del sistema se puede realizar por medio de un cliente Java o bien por medio del entorno web.

Tanto el cliente como el cliente web disponen de un entorno para poder realizar consultas XQuery.

4.2. Otras bases de datos NXD

BaseX

BaseX es una base de datos XML nativa y una herramienta de procesamiento de documentos XML. Este software, que es de código abierto y multiplataforma, proporciona un completo motor de consulta para XPath, XQuery y XSLT.

Más información: https://basex.org/

Fonto

Fonto es una herramienta online que permite realizar consultas XQuery 3.1 contra un documento XML.

Es muy sencilla de utilizar:

- En primer lugar, copiamos el documento XML en la parte izquierda (XML input). También permite importar un archivo. •
- Después, seleccionamos la opción XQuery 3.1 en la esquina superior izquierda (desplegable, por defecto está configurada con XPath 3.1). • Por último,
- indicamos la consulta XQuery en el cuadro de texto XPath/XQuery expression.

Los resultados (Query results) se mostrarán en la parte derecha de forma instantánea.

Más información y acceso: https://xpath.playground.fontoxml.com/

5. Enlaces de interés

XQuery en w3schools: https://www.w3schools.com/xml/xquery_intro.asp

Especificación XQuery en w3: https://www.w3.org/TR/xquery-31/

6. Bibliografía

Sala, Javier. (2023) Lenguajes de marcas y sistemas de gestión de información.

Instituto Abierto de Cataluña