UD 06 - ALMACENAMIENTO DE INFORMACIÓN

1. Utilización de XML para almacenamiento de información

- XML permite **guardar** y **comunicar** información acerca de objetos. Codifica la información de una forma separada a la que se le presenta al usuario.
- Para encontrar un fragmento específico de información, debe procesarse todo el XML
- **Base de datos XML**: Colección de documentos XML (archivo en el sistema de archivos con cadena válida XML) en la que cada uno representa un registro.
- La estructura del documento XML puede seguir el mismo esquema pero **no es obligatorio que sea así**. Cada archivo **puede ser configurado de forma independiente**, lo que da **flexibilidad** y **facilita el desarrollo**.

El almacenamiento de datos se podría dividir en dos categorías:

- **Centrados en los datos**: Documento con estructura bien definida con esquemas altamente estructurados (factura, albarán, ticket, matrícula, acta de notas, órdenes de compra). Apropiados para publicidad. Datos pueden actualizarse. **Muchos elementos con poco contenido, muy estructurados**.
  Relativamente planos. Tipos de datos relativamente complejos. Poca importancia al orden.
- **Centrados en los documentos**: Impredecibles en tamaño. Contenidos con tipos de tamaño limitado y reglas menos flexibles para campos opcionales. Archivos variables dependiendo de quién los realice o escriba (libro de texto, manual, informe). **Pocos elementos con gran cantidad de contenido y desestructurados.**
  Estructura irregular. Tipos de datos simples. Importancia al orden.

Los sistemas de almacenamiento XML deben acomodarse a ambos tipos de requerimientos de datos.
La mayoría de productos se enfocan en servir uno de esos formatos mejor que otro:
Las **bases de datos tradicionales** son mejores para tratar requerimientos centrados en los datos.
Los **sistemas de administración de contenidos y documentos** son mejor para datos centrados en el documento.

Sistemas de bases de datos deben exponer datos relacionales como XML y almacenar el XML recibido como datos relacionales para transferir, obtener y almacenar datos requeridos por la aplicación.

La tecnología XML:

- usa uno o más documentos para almacenar información
- define esquemas sobre información
- tiene lenguajes de consulta específicos para recuperar la información requerida
- Dispone de APIs (SAX, DOM)
- No tiene almacenamiento y seguridad eficientes (índices, seguridad ,transacciones, integridad de datos, acceso concurrente, disparadores...). es imposible pensar que XML se use para tareas transaccionales de una organización.

2. Sistemas de almacenamiento de información

Las bases de datos relacionales:
Usan relaciones (tablas bidimensionales) para representar los datos del mundo real y están asociadas al lenguaje SQL

No están bien preparadas para almacenar estructuras de tipo jerárquico como documentos XML porque los XML:

- Tienen **carácter heterogéneo** frente a la estructura regular de la base de datos relacional
- Contiene **muchos niveles de anidamiento** frente al los datos planos relacionales
- Contiene un **orden intrínseco** mientras que los datos relacionales no son ordenados

- Son **datos dispersos** que pueden representar la carencia de información mediante la ausencia del elemento, mientras que en los datos relacionales cada columna tiene un valor.

Sin embargo se prefieren bases de datos relacionales y orientadas a objetos para almacenar estructuras XML aunque no sean de forma nativa porque son **bien conocidas**, especialmente en cuanto a tu rendimiento.

Reglas de transformación de bases de datos relacional a XML. Creación del DTD.

La estructura XML debe acomodar clases primarias, secundarias, tablas y columnas. Pueden existir muchas variaciones de esquemas XML para representar la misma base de datos relacional.

El proceso de traducción de BBDD relacional a XML podría seguir los siguientes pasos:

- **Crear el esquema XML:** con un elemento para cada tabla y los atributos para cada columna no clave. Columnas que no permiten valores nulos deben ser marcadas como requeridas; Columnas que permiten valores nulos deben ser marcadas como opcionales. En lugar de atributos podrían anidarse las columnas como elementos pero eso puede dar problemas si existe el mismo nombre de columna en más de una tabla.

- **Crear claves primarias en el esquema XML:** Para las columnas que son claves primarias puede agregarse un atributo con un ID. Y en el esquema XML definirlo como de tipo ID. Pueden surgir problemas de colisión al crear las claves en el XML porque el ID necesita ser único en todo el documento. Podría añadirse el nombre del elemento (nombre de la tabla), al valor la clave primaria (valor del atributo) para que sea único a través del documento XML.
- **Establecer relaciones de clave migrada o foránea**: Se puede anidar bajo el elemento padre, un ID de esquema XML puede ser usado para apuntar a la estructura XML correspondiente conteniendo un IDREF.

XML y Bases de datos orientadas a objetos

Las BBDD orientadas a objetos soportan modelo de objetos puros (no se basan en otros modelos clásicos como el relacional; están influidos por los lenguajes de POO; es intento de añadir la funcionalidad de un SGBD a un lenguaje de programación; son alternativa al almacenamiento y gestión de documentos XML)

El estándar lo componen:

- **Modelo de objetos**. Concebido para dar modelo de objetos estándar para las bases de datos orientadas a objetos. En él se basan el lenguaje de definición de objetos y el lenguaje de consultas.
- **Lenguajes de especificación de objetos ODL (Object Description Language)**
- **Lenguaje de consulta OQL (Object Query Language)**
- **Bindings** para C++, Java y Smalltalk. Definen un OML (Object Manipulation Language) para soportar objetos persistentes. Incluye soporte para OQL, navegación y transacciones. Una vez transformado el documento XML en objetos estos los gestiona el SGBDOO. Se consulta con el lenguaje OQL. La indexación, optimización, consultas... son los del SGBDOO y no son específicos para XML.

XML y Bases de datos nativas

Son bases de datos que soportan transacciones, acceso multiusuario, lenguajes de consulta y que están específicamente diseñadas para almacenar XML.
Se caracterizan por:

- **Almacenamiento de documentos en colecciones.** Base de datos estructurada en colecciones (conjunto de documentos), de modo que es estructura de árbol donde cada documento pertenece a una única colección.
- **Validación de documentos**
- **Consultas**: Utilizando el lenguaje **XQuery**
- **Indexación XML**: Índices que aceleran las consultas
- **Identificadores únicos**: Asociados a cada documento XML

- Actualizaciones y borrados:
  - Almacenamiento basado en texto: Almacena el documento XML entero en forma de texto y proporciona forma de acceder a él
    - Posibilidad 1: Lo almacena como BLOB en base de datos relacional por fichero y da índices que aceleren el acceso a la información
    - Posibilidad 2: Almacena en un almacén adecuado con índices, soporte para transacciones, etc
  - Almacenamiento basado en el modelo: Almacena modelo binario del documento (DOM) en almacen existente o específico
    - Posibilidad 1: Traduce el DOM a tables relacionales (elementos, atributos, entidades)
    - Posibilidad 2: Traduce el DOM a objetos en BBDDOO
    - Posibilidad 3: Usa almacen para esta finalidad

Características bases de datos nativas: Consultas, Creación de índices y Creación de identificadores únicos.
Los triggers son de bases de datos relacionales.

3. XQuery

XQuery: Lenguaje funcional diseñado para escribir consultas sobre colecciones de datos en XML. Puede aplicarse a archivos XML y a bases de datos relaciones con funciones de conversión de registros a XML. Extrae información de un conjunto de datos organizados con árbol de etiquetas XML pero no le importa el origen de los datos.

Requerimientos técnicos que cumple XQuery:

- Lenguaje declarativo
- Independiente del protocolo de acceso (archivo local, servidor BBDD, archivo XML)
- Consulta y resultados: Respetar modelo XML
- Consulta y resultados: Ofrecer soporte para namespaces
- Soportar XML-Schema y DTO
- Independiente de la estructura del documento
- Soportar tipos simples (enteros y cadenas) y complejos (nodo compuesto)
- Soportar cuantificadores universales (para todo) y existenciales (existe)
- Soportar operaciones sobre jerarquías de nodos y secuencias
- Combinar información de múltiples fuentes en una consulta
- Manipular datos independientemente del origen de estos
- Lenguaje independiente de la sintaxis (varias sintaxis distintas para expresar la misma consulta)

Aplicaciones:

- Recuperar información a partir de datos XML
- Transformar estructuras de datos XML en otras estructuras
- Ofrecer alternativa a XSLT para hacer transformaciones de datos en XML, HTML, PDF

Motores de XQuery:

De código abierto:

- Qexo
- Saxon (Saxon-B open source Saxon-SA propietario)
- Qizx/open: Todas las características salvo importación y validacion de XML-shema)
- Xquark Bridge: Permite importar y exportar a basess de datos relacionales, respetando restricciones de integridad y relaciones implícitas del XML en relaciones explícitas. XQuery, MySQL, Oracle, SQLServer...
  Otras:
- Xquark Bridge: Importar y exportar datos a bases de datos relacionales usando XML.
- BumbleBee: Entorno de prueba automático para validar motores XQuery y consultas XQuery. Se distribuey con pruebas ya preparadas y permite redactar y ejecutar pruebas propias. Es propietario. Soporta Qexo, Qizx/open y Saxon. Cerisent, Ipedo, IPSI-XQ, X-Hive.

Modelo de datos

En Xquery el orden en el que se encuentren los datos es importante. No es igual buscar etiqueta `<B>` dentro de `<A>` o todas las etiquetas `<B>` del documento.

La entrada y salida de consulta XQuery se definen según un modelo de datos que proporciona representación abstracta de uno o más documentos XML y que se basa en:

- Definición de secuencia como **colección ordenada de 0 o más items.** Además es **heterogénea**. Pero una **secuencia nunca puede ser el item de otra secuencia.**
- **Orden del documento**. Orden en el que los nodos aparecerían si se representase la jerarquía en formato XML. (Si el primer carácter de un nodo precede al primer carácter de otro nodo, también lo precederá en el orden)
- Contempla **valor especial llamado "error value"** al evaluar expresión que contiene un error.

Expresiones

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

- El valor de la expresión es secuencia heterogénea de nodos y valores atómicos.
- Formada por expresiones y palabras simples unidas mediante palabras reservadas.
- **Xpath es lenguaje declarativo** para localizar nodos y fragmentos. XQuery está sobre la base de Xpath. **Toda expresión XPath también es consulta Xquery válida.**
- **Comentarios entre** `(: :)`
- ***Caracteres** `{ }` delimitan expresiones evaluadas para crear documento nuevo
- Admite expresiones condicionales del tipo **if-then-else**

En cualquier lugar del documento:

```
        &b in doc("libro.xml")//libro
```

- Las consultas XQuery pueden estar formadas por hasta cinco tipos de claúsulas diferentes porque sigue **norma FLWOR**, que equivalen a For, Where, Group by, Having, Order By y Limit.
- Si se incluyen varias consultas en el archivo `.xquery` para ejecución conjunta irán separadas por coma.
- En la sentencia FLWOR debe haber un FOR o un LET (el resto, si existen, deben seguir ese orden de palabra FLWOR)
  - **For**: Vincula a expresiones escritas en XPath creando flujo de tuplas vinculando cada tupla a una variable. Si en la consulta aparece más de una claúsula FOR (o más de una variable en claúsula FOR), el resultado es el producto cartesiano de esas variables.
  - **Let**: Vincula variable al resultado completo de la expresión añadiendo esos vinculos a tuplas generadas por claúsula for o, si no existe, creando única tupla con esos vínculos.
  - **Where**: Filtra tuplas producidas por FOR y LET eliminado las que no cumplen condiciones dadas
  - **Order by**: Ordena las tuplas generadas por FOR y LET según criterio dado después de haber sido filtradas por WHERE. Por defecto, orden ascendente (puede ponerse descending)
  - **Return**: Resultado de la consulta por la tupla dada tras ser filtrada por Where y ordenada por Order by.

For y let sirven para crear las tuplas con las que trabajará el resto de las clausulas y pueden usarse tantas veces como se quiere
Order by, where y return solo una vez.
Ninguna clausula FLWOR es obligatoria en la consulta XQuery. Por eso cualquier expresión XPath es XQuery válida.

La consulta XQuery tiene:

- **Prólogo**: Lugar donde se declaran namespaces, funciones, variables...
- **Expresión**: Consulta propiamente dicha

Diferencias entre for y let

- La cláusula for vincula una variable con cada nodo que encuentre en la colección de datos.
- La cláusula let vincula una variable con todo el resultado de una expresión. Por eso aparecen... una única vez.

Funciones

Numéricas

- `floor()`: inferior más próximo
- `ceiling()`: superior más próximo
- `round()`: valor dado al más próximo
- `min()` `max()`: mínimo y máximo de los nodos dados
- `avg()`:valor medio
- `sum()`: suma

Cadenas de texto

- `concat()`: unión de dos cadenas
- `string-length()`: longitud
- `startswith()`, `ends-with()`: si comienza o termina
- `upper-case()`, `lower-case()`: transforma en mayúsculas o minúsculas

Uso general

- `empty()`: true si no contiene elemento
- `exits()`: true si contiene elemento
- `distinct-values()`: extrae valores de secuencia de nodos y crea secuencia con valores únicos

Existenciales

- `some`, `every`: devuelven algún o todos los elementos...
  Puede construir sus propias funciones también

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

Operadores

De valores: eq (igual), ne (no igual), lt (menor que), le (menor o igual que), gt (mayor que), ge (mayor o igual que)

Comparación generales: = != > >= < <=

Comparación de nodos: `is` si están ligadas al mismo nodo; `is not` si no están ligadas al mismo nodo

De órdenes de los nodos: << (true si el nodo ligado al primer operando ocurre primero que el nodo ligado al segundo)

Secuencias de nodos: Union (suma de los dos), Intersect (en los dos), Except (primer operando y no en el segundo)

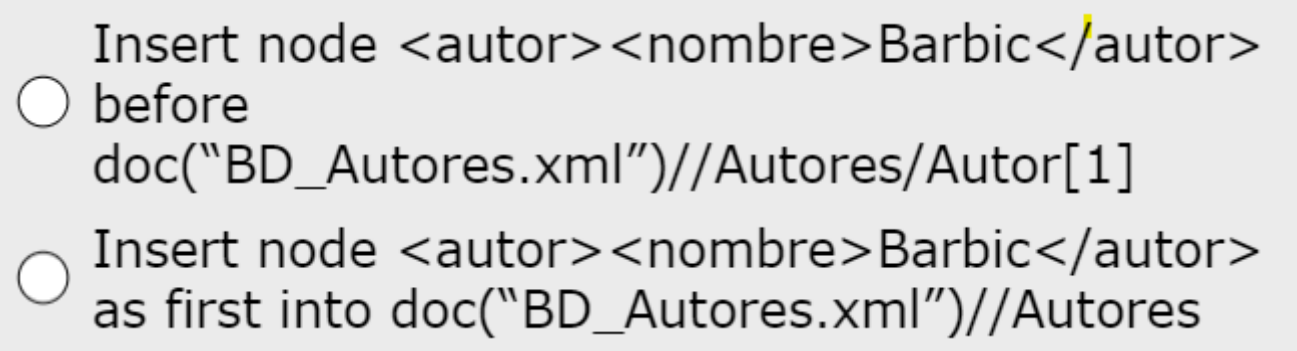
Aritméticos: + - * div y mod

(Pregunta prueba: Pertenecen a SQL y a XQuery is, union, >=)

XQuery Update Facility

Pequeña extensión que provee XQuery para modificar documentos XML.
insert
Several modifiers are available to specify the exact insert location: insert into **as first/as last**, insert **before/after** and insert **into**.
`insert node (attribute { 'a' } { 5 }, 'text', <e/>) into /n`
delete
`delete node //n`
replace
`replace node /n with <a/>`
`replace value of node /n with 'newValue'`
rename

```
for $n in //originalNode
return rename node $n as 'renamedNode
```



Ejemplo 1

Para el XML alumnos.xml:

```
<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>
```

Veamos un ejemplito:

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

Que nos devuelve:

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

Ejemplo 2

Para XML libro.xml:

```
<bib>
    <libro año="1994">
        <titulo>TCP/IP Illustrated</titulo>
        <autor>
           <apellido>Stevens</apellido>
           <nombre>W.</nombre>
       </autor>
       <editorial>Addison-Wesley</editorial>
       <precio> 65.95</precio>
    </libro>
    <libro año="1992">
       <titulo>Advan Programming for Unix environment</titulo>
       <autor>
          <apellido>Stevens</apellido>
          <nombre>W.</nombre>
      </autor>
       <editorial>Addison-Wesley</editorial>
       <precio>65.95</precio>
    </libro>
</bib>
```

Veamos un ejemplito:

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

Busca nodos `<libro>` hijos de `<lib>` con más de dos autores, los ordena por el título y devuelve su título.

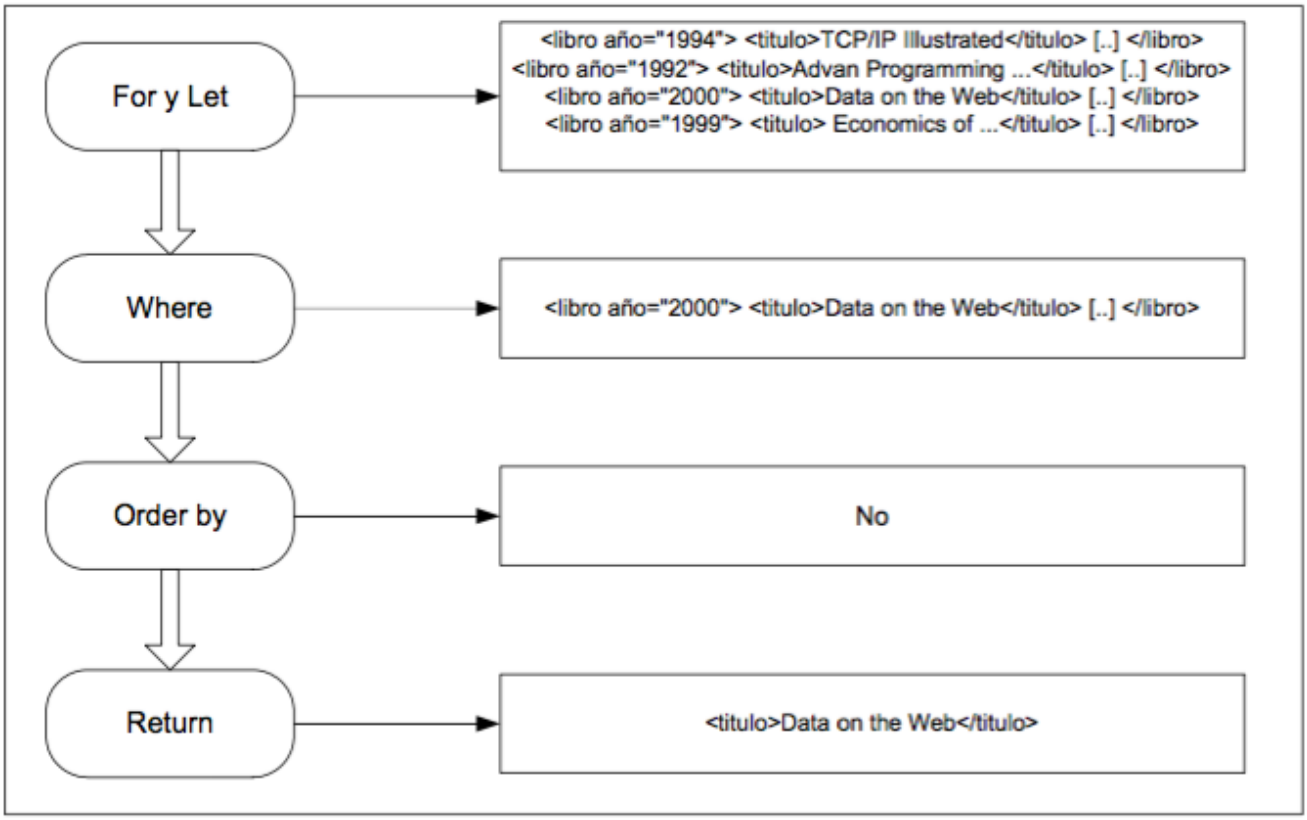
Retorna:

```
<title>Data on the Web</title>
```

Ejemplos varios

```
for
  $b in doc("libros.xml")//libro
where
  $b/@año = "2000"
return
  $b/titulo
```

Gráficamente:




```
(: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>
```

```
(://escrutinio_sitio[convocatoria='2015']/nombre_sitio:)

(:<ul>{
for $a in /escrutinio_sitio
where $a/convocatoria='2019' and $a/votos/contabilizados/porcentaje >= '69'
order by $a/nombre_sitio
```

```
return <li>{$a/nombre_sitio/data()}</li>
}</ul>:)

(:let $contenido := (for $a in /escrutinio_sitio
where $a/convocatoria='2019' and $a/votos/contabilizados/porcentaje >= '70'
order by $a/nombre_sitio
return <li>{$a/nombre_sitio/data()}</li>)
return <ul>{$contenido}</ul>:)

(:let $p2019 := distinct-values(for $p in /escrutinio_sitio
where $p/convocatoria=2019
return $p/resultados/partido/nombre)

for $p2015 in distinct-values(for $p in /escrutinio_sitio
where $p/convocatoria=2015
return $p/resultados/partido/nombre)
where not($p2015 = $p2019)

return <partido>{$p2015}</partido>:)
(:For itera cada uno de ellos.
Let almacena conjunto de valores. A la hora de comparar, cuidado porque compara con el
conjunto. A la hora de verificar no compara con el resto:)
(:<resultados>
{for $a in ('VOX','PODEMOS-IU')
let $total := sum(/escrutinio_sitio[convocatoria=2019]/
resultados/partido[nombre=$a]/electos)
return <electos partido="{$a}">{$total}</electos>}
</resultados>:)
for $provincia in /escrutinio_sitio[convocatoria=2019]
let $maximo := max($provincia//partido/votos_numero)
let $partido := $provincia//partido[votos_numero=$maximo]/nombre
order by $provincia/nombre_sitio
return <resultado><provincia>{$provincia/nombre_sitio/data()}</provincia>
<partido>{$partido}</partido>
</resultado>
```