UNIDAD DIDÁCTICA 6

ALMACENAMIENTO DE INFORMACIÓN

MÓDULO PROFESIONAL: LENGUAJE DE MARCAS Y SISTEMAS DE GESTIÓN DE LA INFORMACIÓN

Índice

RESUMEN INTRODUCTORIO

A lo largo de la unidad veremos en que consiste el almacenamiento de la información, a través de los sistemas de almacenamiento de información teniendo muy presente sus características y tecnologías.

A continuación, conoceremos en qué consiste el lenguaje de consulta y aprenderemos cómo realizarlas en los tipos de documentos como son Base de datos, Colecciones de documentos, Documentos estructurados, etc.

También, trataremos los sistemas gestores de bases de datos resaltando la importancia de dos de ellos como son Oracle y SQL Server de Microsoft.

Por último, conoceremos de qué trata el almacenamiento y manipulación de información en sistemas nativos y en formato XML, teniendo en cuenta las herramientas de tratamiento y almacenamiento de información en sistemas nativos.

INTRODUCCIÓN

Como ya se vio en unidades anteriores, XML es un buen formato para estructurar información. Por ello, puede resultar igualmente útil para almacenarla. No obstante, para poder sacarle partido, es necesario añadir mecanismos que proporcionen un acceso eficiente a los datos, así como herramientas y lenguajes que permitan manipular dicha información. Es decir, se trata de poder tener características similares a las ofrecidas por una base de datos tradicional, pero considerando que en esta ocasión los datos se encuentran en formato XML.

El uso de XML en el ámbito de las bases de datos adquiere una gran relevancia en el sentido de no quedarse como simple soporte para la transferencia de datos e información, sino que es utilizado como un nuevo formato de almacenamiento de datos, donde estas bases de datos utilizan documentos XML como unidades de almacenamiento fundamentales constituyendo las bases de datos nativas XML.

Ello se suma a la versatilidad de utilización de XML ampliando sus horizontes que van más allá de sus usos ya conocidos en aplicaciones informáticas, y de tipo web como el caso de la tecnología AJAX, o en distintos servicios web.

CASO INTRODUCTORIO

Trabajas como programador en una empresa de desarrollo de software para empresas, y un cliente plantea sus necesidades para explotación de datos con la intención de trabajar directamente sobre ellos como si se tratase de una base de datos relacional, pues en la actualidad la empresa almacena todas sus facturas en formato XML, ya que así les resulta sencillo, generarlas automáticamente desde su aplicación, y para procesar toda la información contenida en ellas, habitualmente, las importan desde una hoja de cálculo y trabajan con los datos obtenidos.

Al final de la unidad conocerás las tecnologías disponibles para usar sistemas de información, donde aprenderás a usar el programa BaseX para gestionar información almacenada en documentos XML siendo capaz de realizar las consultas oportunas sobre documentos XML gracias al lenguaje de consultas XQuery.

1. SISTEMAS DE ALMACENAMIENTO DE INFORMACIÓN. CARACTERÍSTICAS, VENTAJAS E INCONVENIENTES Y TECNOLOGÍAS

Estás trabajando en un proyecto para optimizar el almacenamiento y tratamiento de información de una empresa cliente, por lo que deberás conocer los diferentes sistemas de almacenamiento de información y las características de cada uno de ellos, con sus pros y sus contras.

La información generada diariamente es muy elevada. Existe una alternativa a guardar los datos en las bases de datos relacionales (tablas y campos). Se trata de guardar la información en documentos XML (elementos y atributos). No obstante, esto conlleva sus pros y contras.

XML permite representar de manera rápida información para poder ser compartida. Esto se realiza con las bases de datos nativas, o exportación de bases de datos relacionales a XML. El tener la información almacenada en una base de datos relacional no es impedimento para poder obtener el XML asociado, tal y como se ve en el siguiente ejemplo.

Se trata de una tabla Access con tres campos:

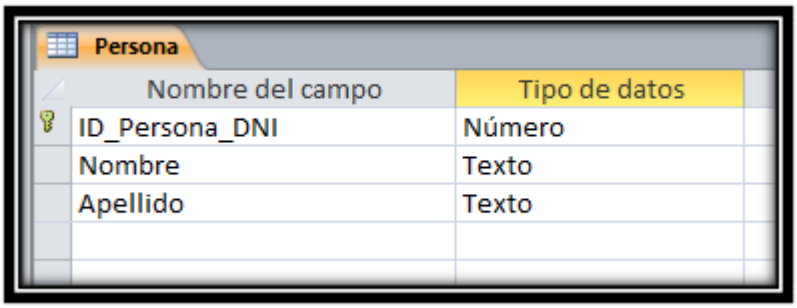
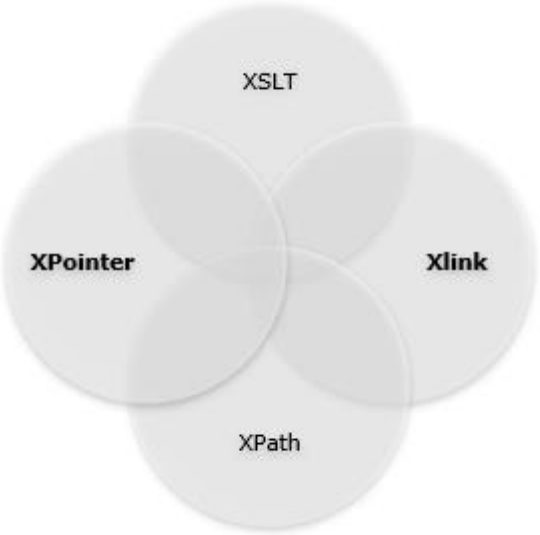


Tabla creada en Access y pasada a XML.

Al seleccionar la opción de exportar desde dicha aplicación, se muestra un asistente que, tras unos sencillos pasos, da como salida un fichero XML con los datos almacenados en la tabla.

1.1 Tecnologías

Estas tecnologías son un conjunto de módulos que ofrecen una serie de servicios a los usuarios. Las tecnologías **Xlink** y **XPointer** no han tenido la misma repercusión que **XPath** o **XSLT**. A continuación, explicaremos brevemente en qué consisten cada una de ellas.



Tecnologías XML.

1.1.1 XLINK

XLink (the XML Linking language) es un lenguaje de vinculación para documentos XML. En XLink, cualquier elemento de un documento XML puede comportarse como un enlace.

Entre sus ventajas, destaca el contar con una gran variedad de enlaces, que pueden ser tanto unidireccionales como bidireccionales, así como su compatibilidad con la mayoría de navegadores y un gran número de herramientas de internet.

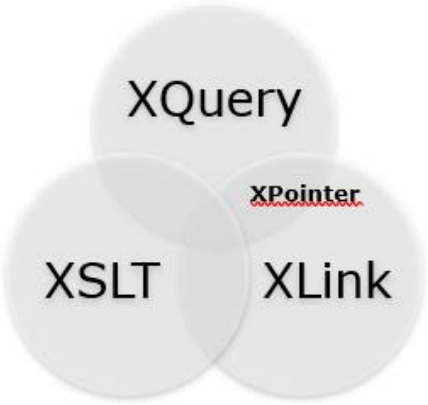
XLink soporta enlaces simples (como HTML) y los enlaces pueden definirse fuera de los archivos vinculados. Algunos de los atributos de XLink son: “type, href, role, title”.
XLink funciona de forma similar a los enlaces de una página web HTML, mediante <a href="">, con la peculiaridad que de que href es un enlace unidireccional. Al usar XLink, se pueden crear vínculos bidireccionales. Por ejemplo:

```
<my:referencia
  xlink:href="libro.xml"
  xlink:role="http://.../listadelibros"
  xlink:title="Libros de…">
</my:referencia>
```

1.1.2 XPOINTER

XPointer (the XML Pointer language) o lenguaje de punteros XML. Ayuda a vincular partes específicas de un documento XML. Se suele usar en conjunto con XLink. Su

funcionamiento es similar al de los identificadores de fragmentos dentro de un documento HTML.



XPointer usa expresiones y tiene todas las ventajas de XPath para navegar en el documento XML, ya que es una extensión de XPath.

Su funcionamiento es el siguiente: Primero, XLink permite establece el enlace con el recurso XML y luego es XPointer el que va a un punto específico del documento.

```
documento.xml#xpointer(
/libro/capitulo[@public])xpointer(/libro/capitulo[@num="1"])
```

1.1.3 XPath

XPath (the XML Path language), es un lenguaje que facilita la búsqueda de información en un documento XML. Utiliza expresiones regulares para navegar en documentos XML. Se emplea junto a XSLT, usándose también en XQuery, XLink y XPointer.

Nos ofrece una gran cantidad de utilidades y ventajas, pues nos permite conocer información importante sobre un documento XML, mediante la definición de criterios de búsqueda avanzada y cálculos concretos, donde destaca la facilidad que proporciona para tratar las partes de un documento XML, ofreciendo servicios básicos en la manipulación de strings o cadenas de texto, números y booleanos, operando no sólo en la superficie del documento XML, sino que lo hace sobre su estructura abstracta y lógica.

XPath es la herramienta que se usa en XQuery para recorrer o procesar el árbol de un documento XML, siendo la base para otras tecnologías como XPointer, XLink o XSLT.

ENLACE DE INTERÉS

Aquí encontrarás más información a modo de tutorial básico sobre XPath.

1.1.4 XSLT

XSLT (Extended Styleshet Language Transformation) definido como un lenguaje XML basado en reglas que transforman la estructura y contenido de un documento XML en otro.

En la actualidad, XSLT es muy usado en la edición web, generando páginas HTML o XHTML.

Ventajas que podemos destacar de XSLT en el caso del lado servidor, son la creación de plantillas más simples y precisas que permiten el procesamiento de datos XML a HTML de un modo más fácil y con mayor rapidez.

Desde el lado cliente, aporta la posibilidad de descargar el procesamiento del código de un modo más sencillo y en el que los scripts pueden proporcionar una mejor idea de cómo dar formato al HTML final.

EJEMPLO PRÁCTICO

Arturo, experto en programación con lenguajes de marcas, es el responsable dentro del departamento de informática de su empresa y de la elección de cualquier nueva herramienta o software que se vaya a incorporar en ella.

Ante el auge en la utilización de XML en el desarrollo de nuevas aplicaciones y soluciones para terceros, se le ha solicitado que haga una recopilación de las distintas tecnologías que existen para trabajar con XML, con la posibilidad de incorporarlas en los nuevos desarrollos.

¿Qué tecnologías deberían incluirse en el informe?

Solución.

Las tecnologías XML que deberían incluirse en el informe son:

- XLink (the XML Linking language)
  Es un lenguaje de vinculación para documentos XML. En XLink, cualquier elemento de un documento XML puede comportarse como un enlace.

- XPointer (the XML Pointer language)
  Lenguaje de punteros XML. Ayuda a vincular partes específicas de un documento XML. Se suele usar en conjunto con XLink.

- XPath (the XML Path language)
  Es un lenguaje que facilita la búsqueda de información en un documento XML.

- XSLT (Extended Styleshet Language Transformation)
  Definido como un lenguaje XML basado en reglas que transforman la estructura y contenido de un documento XML en otro.

2. LENGUAJES DE CONSULTA Y MANIPULACIÓN EN DOCUMENTOS. CONSULTA Y MANIPULACIÓN DE INFORMACIÓN

En esa manipulación de información y documentos que dispone la empresa cliente sobre la que estás realizando el desarrollo de software para el tratamiento de los datos, es necesario saber qué lenguaje de consulta será el más adecuado emplear en ese desarrollo de software.

El lenguaje usado en bases de datos relacionales para realizar consultas es SQL, pero para las bases de datos nativas, se establece XQuery como el lenguaje de consultas sobre ficheros XML. Este lenguaje permite recorrer el XML para extraer la información que se necesite.

2.1 XQuery

Lenguaje de Consulta XML, es un lenguaje que facilita la extracción de datos desde documentos XML. Ofrece la posibilidad de realizar consultas flexibles para extraer datos de documentos XML en la Web.

Ejemplo de documento XML:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<libro>
  <titulo></titulo>
  <capitulo>
    <titulo></titulo>
    <seccion>
      <titulo></titulo>
    </seccion>
  </capitulo>
</libro>
```

Ejemplo de código de XQuery:

```
<!-- Libros escritos por Vargas Llosa después de 1991 -->
<bib>
  {
  for $b in doc("http://libro.example.com/bib.xml")/bib/libro
  where $b/autor = "Vargas Llosa" and $b/@anio > 1991
  return
  <libro anio="{ $b/@anio }">
    { $b/titulo }
  </libro>
  }
```

```
</bib>
```

XQuery es en primer lugar un lenguaje de consultas, que se basa en operadores de búsqueda de un modelo de datos para documentos XML. Puede realizar consultas en infinidad de tipos de documentos, como son:

- Bases de datos.
- Colecciones de documentos.
- Documentos estructurados.
- Estructuras DOM.
- Catálogos.

Una consulta XQuery tiene como entrada y salida sendos documentos XML, con lo que éste no es solo un lenguaje de consulta que opera sobre documentos (para ser más preciso, sobre instancias de un modelo de datos XML) sino que también genera documentos XML a demanda de la consulta correspondiente.

2.2 Consulta y manipulación de la información

XQuery es un lenguaje funcional cuya sintaxis es la de XML aunque parecida a la de XPath, con lo que toda consulta es una expresión que debe evaluarse.

2.2.1 Conceptos de XQuery

Una consulta acerca de todos los títulos de los capítulos de un documento como Libro.xml se puede hacer de forma sencilla con la expresión tal como:

doc (“Libro.xml “)//capitulo/titulo

a) Definiciones básicas.

Un literal es la clase más simple de expresión XQuery, y representa un valor atómico. En consecuencia, sus valores se corresponden con los tipos simples de esquema, donde los principales tipos numéricos son:

- xs:integer
- xs:decimal.
- xs:double.

Los tipos cadena se delimitan por comillas pudiendo contener referencias a entidades predefinidas.

Una variable en XQuery es un nombre que empieza con el signo dólar **($Capitulo**), que se asocia a un valor y que se usa dentro en una expresión para representarlo. Una forma de dar valores a una variable es por medio de una expresión **let**, que asocia una o varias variables y luego evalúa la correspondiente expresión interna.

Un constructor es una función que crea un valor de un tipo particular a partir de una cadena que contiene la representación léxica del tipo deseado. Por ejemplo:

string ($Capitulo/numero)

Los distintos tipos de valores atómicos se crean llamando a un constructor, que en general tiene el mismo nombre que el tipo que construye.

Una transformación es el paso de una instancia del modelo de datos a otra instancia de este modelo, dejando abierto tanto el origen de los datos de entrada como la forma como se envía el resultado a la aplicación que haya invocado esta transformación.

Para identificar los datos de entrada existen dos funciones básicas:

- **doc()**, que devuelve un documento completo identificándolo con una URI; por ejemplo: **doc("libros.xml")**.

- **collection()**, que devuelve cualquier secuencia de nodos asociada a la URI (a menudo usada para identificar la base de datos empleada en la consulta).

Por tanto, una consulta puede tener su entrada bien por medio de doc, collection, o bien referenciando alguna parte de un contexto externo.

Una consulta consta de dos partes:

- prólogo (query prolog).
- cuerpo (query body).

El prólogo consta de una serie de declaraciones que definen el entorno para el procesado del cuerpo, que consta de una expresión cuyo valor proporciona el resultado de la consulta. El prólogo se usa cuando el cuerpo depende de los espacios de nombres, esquemas o funciones y, cuando ello ocurre, la consulta depende en gran medida de lo especificado en él.

```
<?xml version='1.0'?>
<libro>
  <titulo> Respuestas XQuery </titulo>
  <Capitulo numero='1' titulo="El primer Capítulo">
    ....
  </Capitulo>
  <Capitulo numero="2" titulo="El segundo Capitulo">
    ....
  </Capitulo>
  <!-- y continua con más capítulos ....-->
</libro>
```

Para obtener el sumario con el número y título de cada capítulo, se puede hacer mediante:

```
<tablacontenidos>
  {
  for $Capitulo in doc("XQueryLibro.xml")//Capitulo
  return
  <cabeceraCapitulo>
    { string($Capitulo/@numero)}
      ..
     {string($Capitulo/@titulo)}
  </cabeceraCapitulo>
  }
</tablacontenidos>
```

Donde se observa cómo, tras la etiqueta de inicio del elemento, Tablacontenidos se crea la variable **$Capitulo** a la que se asigna una secuencia de nodos con el uso de la palabra clave **in** que precede al documento que proporciona esta secuencia, terminando con **//Capitulo**, que selecciona cualquier elemento Capitulo del documento objeto de la consulta.

Por su parte, la sentencia **return** define la salida para cada nodo que aparece en la secuencia definida por la variable **$Capitulo**.

Para cada nodo elemento Capitulo se indica el literal de la etiqueta de comienzo cabeceraCapitulo y, como primera salida, la expresión:

{string($Capitulo/@numero)}

La cual obtiene el valor del atributo número del elemento Capitulo. Al existir en el documento dos elementos Capitulo, la consulta produce dos elementos cabeceraCapitulo cuyo contenido son los valores de los atributos y título de cada elemento Capitulo del documento fuente:

```
<Tablacontenidos>
<CabeceraCapitulo>
1..El Primer Capitulo
</CabeceraCapitulo>
<CabeceraCapitulo>
2..El Segundo Capitulo
</CabeceraCapitulo>
</Tablacontenidos>
```

b) Tipos de datos en XQuery.

Puesto que el sistema de tipos de datos de XQuery se basa en esquemas, hay que distinguir dos conjuntos de tipos:

- Los predefinidos, accesibles por cualquier consulta.
- Los importados para una consulta desde un esquema concreto.

Se llama XQuery básico al mecanismo que admite, por un lado, la importación de esquemas para hacerlos accesibles a la consulta y, por otro, la posibilidad de utilizar un tipado estático que permite que una consulta puede contrastarse con los esquemas importados, de forma que se puedan localizar los posibles errores antes de acceder a los datos.

Con el mecanismo XQuery básico, una consulta no necesita importar ningún esquema para poder utilizar tipos predefinidos, ya que están contenidos en el propio documento, y el modelo de datos preserva esta información de tipos, permitiendo con ello que las consultas accedan a ellos. Así, aunque una consulta no importe esquema alguno, se pueden usar sin problemas los correspondientes tipos simples que se encuentren presentes en los datos del documento.

c) Capacidades de XQuery.

Como ejemplo de referencia de este proceso se usará el documento libros.xml, concretamente el fragmento siguiente:

```
<bib>
  <libro anyo="1994">
    <titulo>TCP/IP Ilustrado </titulo>
    <autor>
      <apellido>Stevens</apellido>
      <nombre>W.</nombre>
    </autor>
    <editorial>Prentice-Hall</editorial>
    <precio>65.95</precio>
```

```
  </libro>
  <libro anyo="1992">
    <titulo>Programación Avanzada en el entorno Unix</titulo>
    <autor>
      <apellido>Stevens</apellido>
      <nombre>W.</nombre>
    </autor>
    <editorial>Prentice-Hall</editorial>
    <precio>65.95</precio>
  </libro>
  <libro anyo="2000">
    <titulo>Datos en la Web</titulo>
    <autor>
      <apellido>Abiteboul</apellido>
      <nombre>Serge</nombre>
    </autor>
    <autor>
      <apellido>Buneman</apellido>
      <nombre>Peter</nombre>
    </autor>
    <autor>
      <apellido>Suciu</apellido>
      <nombre>Dan</nombre>
    </autor>
    <editorial>Morgan Kaufmann</editorial>
    <precio>39.95</precio>
  </libro>
  <libro anyo="1991">
    <titulo>Economía de la Tecnología y el Contenido de la TV
digital</titulo>
    <editor>
      <apellido>Gerbarg</apellido>
      <nombre>Darcy</nombre>
    </editor>
    <afiliacion>CITI</afiliacion>
    <editorial>Editorial Kluwer Academic</editorial>
    <precio>129.95</precio>
  </libro>
</bib>
```

LOCALIZACIÓN DE NODOS EN ESTRUCTURAS XML.

Las expresiones deben localizar nodos empezando por determinar el documento en el que efectuar la búsqueda.

```
doc("libros.xml")/bib/libro
```

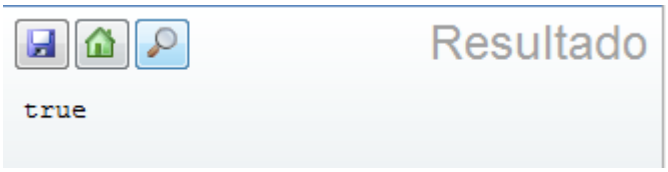
Abre el documento libros.xml obteniendo su nodo documento, con /bib se selecciona este elemento y con /libro se obtienen estos elementos dentro de bib.

En el caso de que en este proceso se vayan a necesitar variables, éstas se definen en el prólogo de la consulta, de forma que una vez declaradas son accesibles en cualquier punto. Así, si los títulos de los libros en libros.xml van a usarse varias veces en una consulta, tiene sentido definir una variable, tal como: **$titulos.**

Los predicados XPath como métodos para filtrar secuencias de nodos se generalizan en XQuery de forma que el valor de una expresión siempre sea una secuencia heterogénea de nodos y valores atómicos.

```
doc("libros.xml")/bib/libro/autor/apellido="Stevens"
```

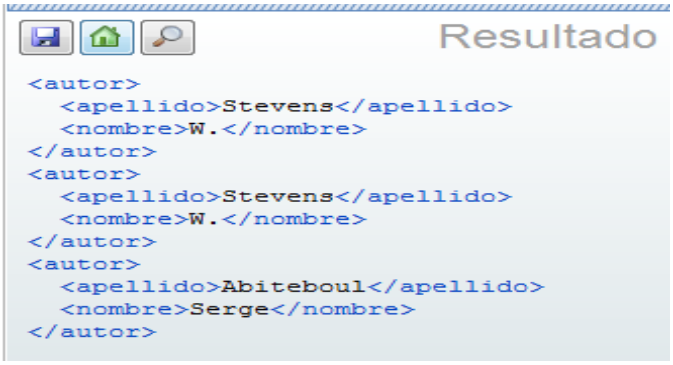
Que devuelve sólo los autores cuyo apellido=”Stevens” es true.



La siguiente consulta:

```
doc("libros.xml")/bib/libro/autor[1]
```

Devuelve el primer autor de cada libro:



En la línea de las expresiones XPath, una expresión en XQuery puede empezar con / o //. En el primer caso la búsqueda se inicia en el nodo raíz, mientras que el significado del operador //, en términos del modelo de datos, es dar acceso a todos los atributos y descendientes de los nodos que aparecen en su izquierda en el documento considerado.

EJEMPLO PRÁCTICO

Formas parte del equipo de desarrollo web en el departamento de informática de una empresa de consultoría informática. Dentro del proyecto en el que están trabajando deben realizar diferentes consultas a bases de datos.

La localización de nodos en documentos XML es de gran importancia para la resolución de las consultas a bases de datos, por lo que teniendo en cuenta los datos proporcionados anteriormente, se pueden realizar otros tipos de consultas que devolverán distintos datos en función de las necesidades que tengamos de obtener cierta información.

¿Puedes decir cómo serían otros ejemplos de consultas sobre esos datos, por ejemplo?

Solución.

Otros tipos de consultas podrían ser:

- doc("libros.xml")/bib
  Ajusta el elemento bib a la raíz del documento.

- doc("libros.xml")//libro
  Da acceso a todos los elementos libro del documento.

- doc("libros.xml")//@anyo
  Localiza todos los atributos anyo en el documento.

- doc("libros.xml")//bib/autor[1]
  Al presentar varios escalones sigue la secuencia: primero devuelve el nodo documento, devuelve el elemento bib, a continuación, devuelve todos los nodos que descienden de él y finalmente selecciona el elemento primer autor para cada nodo contexto (nótese que sólo los elementos libro contienen el elemento autor).

Igual que en XPath, las "wildcards" permiten que las consultas seleccionan sin especificar sus nombres complejos; por ejemplo:

```
doc("libros.xml")//libro[1]/*
```

Obtiene los elementos del primero libro, donde * se ajusta a cualquier elemento, esté o no en el espacio de nombres, ya que las expresiones XQuery soportan espacios de nombres, lo que permite distinguir nombres procedentes de vocabularios diferentes:

Resultado

```
<titulo>TCP/IP Ilustrado</titulo>
<autor>
  <apellido>Stevens</apellido>
  <nombre>W.</nombre>
</autor>
<editorial>Prentice-Hall</editorial>
<precio>65.95</precio>
```

Notar que debido a que durante el proceso de evaluación de toda expresión XQuery siempre se devuelven nodos del documento señalado, si en ella hubiera algo que no correspondiera a uno de estos nodos aparecerá un error de tipo.

CONSTRUCCIÓN DE ESTRUCTURAS XML.

Una vez localizados los nodos en un documento, se debe crear el resultado que satisface la consulta, cosa que se hace mediante el uso de constructores, esta capacidad de construir nuevos objetivos XML es una de las funcionalidades más importantes de XQuery.

La sintaxis para ellos se basa en la evaluación de una expresión que aparece entre **llaves** y que, obviamente, no tiene el mismo significado que en la notación XML, ya que mientras en XQuery el símbolo **{** es el inicio de una expresión a evaluar, en XML sólo es un carácter.

Ejemplo:

<precio>123.45</precio> Es un elemento de nombre precio que contiene un literal, mientras que:

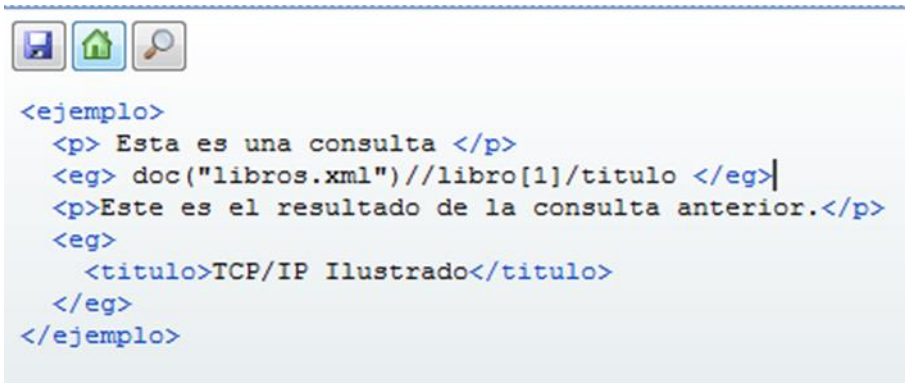
<precio>{$retail * 0.85}</precio> Es un elemento construido cuyo contenido se calcula evaluando una expresión.

Una vez proporcionado explícitamente un constructor por medio de {}, la consulta se inicia creando un nodo documento que empieza siendo vacío y que acaba rellenándose con el resultado correspondiente, hasta crear un documento completo. Las expresiones delimitadas por { } siempre están encargadas de evaluar y crear contenidos de elementos o atributos.

En el siguiente ejemplo se ve la diferencia entre incluir las llaves y no hacerlo:

```
<ejemplo>
  <p> Esta es una consulta </p>
  <eg> doc("libros.xml")//libro[1]/titulo </eg>
  <p>Este es el resultado de la consulta anterior.</p>
  <eg>{doc("libros.xml")//libro[1]/titulo}</eg>
</ejemplo>
```

Produce como resultado:



```
<ejemplo>
  <p> Esta es una consulta </p>
  <eg> doc("libros.xml")//libro[1]/titulo </eg>
  <p>Este es el resultado de la consulta anterior.</p>
  <eg>
    <titulo>TCP/IP Ilustrado</titulo>
  </eg>
</ejemplo>
```

Las expresiones incluidas en los constructores de elementos permiten crear nuevos valores XML reestructurando los existentes como, por ejemplo, en una lista de títulos, mediante:

```
<titulos count="{count(doc('libros.xml')//titulo)}">
{doc("libros.xml")//titulo}
</titulos>
```

Se obtiene el documento:

```
<titulos count="4">
  <titulo>TCP/IP Ilustrado</titulo>
  <titulo>Programación Avanzada en el entorno Unix</titulo>
  <titulo>Datos en la Web</titulo>
  <titulo>Economía de la Tecnología y el Contenido de la TV
digital</titulo>
</titulos>
```

d) Extracción de tipos.

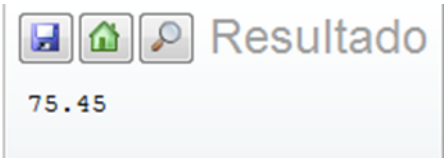
En un lenguaje de búsqueda, los tipos de datos permiten comprobar la corrección tanto de la propia consulta como de las operaciones a los que se va a someter los datos de la respuesta; en base a ello, XQuery aprovecha que los esquemas XML proporcionan un conjunto de tipos predefinidos y la posibilidad de definir nuevos tipos para manejar los tipos de datos involucrados en una consulta.

Por ello, en el prólogo de toda consulta deben figurar explícitamente los esquemas que se van a importar, identificándolos por su espacio de nombres, de forma que durante el correspondiente proceso de validación los nodos elemento y atributo puedan adquirir sus notaciones de tipo correspondientes.

La consecuencia es que la falta de definición de tipos no impide que se puedan efectuar consultas con XQuery.

```
avg(doc("libros.xml")/bib/libro/precio)
```

Como resultado se obtiene:



El ejemplo anterior calcula el precio medio de un libro en nuestro ejemplo, aunque no haya esquema alguno involucrado, ni el elemento precio esté tipado.

Para que ello sea posible, cuando avg() requiere un argumento numérico, automáticamente cada valor precio se convierte a doble precisión y se calcula la media requerida.

Esta conversión implícita es muy útil cuando se trabaja con datos no tipados, pero puede no ser óptima, como en el caso del ejemplo que se acaba de ver, donde el precio se representaría mucho mejor por un decimal. Obviamente, si en el proceso de consulta existiese un esquema que declarase precio como decimal, la media se calcularía usando números decimales.

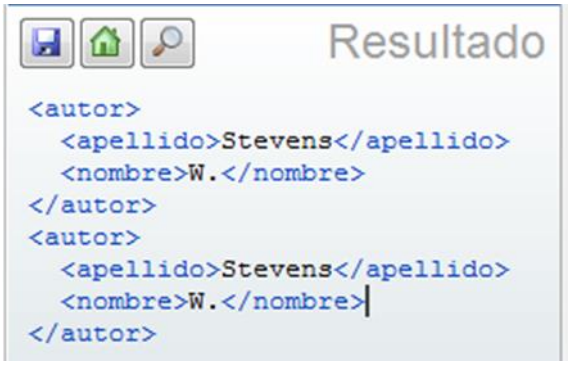
Consecuencia de lo anterior es la necesidad de conocer la forma como XQuery trata las distintas casuísticas que pueden aparecer durante el proceso de un documento. Así, si por ejemplo, se considera una expresión como: 1 * $b, habrá que preguntarse cómo se interpretaría si $b adoptara las distintas posibilidades existentes:

- por ejemplo, que fuera una secuencia vacía.
- o un dato de caracteres sin tipar
- o un elemento
- o una secuencia de cinco nodos
- etc.

Este tipo de cuestiones es básico que puedan resolverse, debido a la propia flexibilidad de XML, lo que obliga a que casos como éstos deben quedar bien definidos a lo largo del proceso de consulta.

Para hacer frente a esta necesidad, en XQuery existe una operación básica llamada extracción de tipo. Este proceso de extracción de tipo permite que tengan sentido expresiones como:

```
doc("libros.xml")/bib/libro/autor[apellido='Stevens']
```

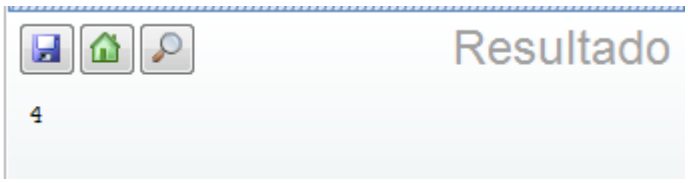


Donde, siendo apellido un elemento y 'Stevens" una cadena, se pueda asumir que pueden ser iguales.

La clave reside en que el operador = extrae el tipo de valor del elemento, obteniendo que es una cadena, lo que permite a XQuery poderla comparar con Stevens y tomar la decisión que corresponda.

En línea con lo anterior, XQuery incorpora la función **data()** que extrae el tipo del valor correspondiente. Por ejemplo:

```
data(<e xsi:type="xs:integer">4</e>)
```



Una vez validado hace que se obtenga directamente como resultado el entero 4, sin tener que preocuparse del tipo de dato concreto.

La extracción del tipo en su forma más general se llama atomización definida como la extracción de valores tipados en cada secuencia de ítems.

```
avg(( 1, <e>2</e>, <e xsi:type="xs:integer">3</e>) )
```

La atomización devuelve un valor tipado, esto es 2, la media de 1,2 y 3.

Como resultado de la atomización, cuando se va a efectuar una comparación, ésta se acaba haciendo en términos de valores atómicos, de forma que, si un operando es un nodo, se recurre a la automatización para convertirlo a un valor atómico. Ello da lugar a toda una casuística, frente a la que hay que saber cómo proceder.

A continuación, se puede observar en el caso del elemento precio, según el tipado que incorpore el operador (se usa eq y gt, que son los operadores de igualdad y mayor que, respectivamente).

Si carece de tipo, se trata como una cadena y, en consecuencia, sólo puede compararse con otra cadena:

```
for $b in doc("libros.xml")//libro
where $b/titulo eq "Datos en la Web"
return $b/precio
```

Si los datos dependen de una DTD, no se pueden usar los tipos simples de los esquemas, con lo que precio no tendrá tipo. Por ello, en la siguiente consulta, es necesario pasar precio a decimal:

```
for $b in doc("libros.xml")//libro
where xs:decimal($b/precio) gt 100.00
return $b/titulo
```

Si quien impone los tipos es un esquema que declara precio para ser un decimal, nada de lo anterior es necesario.

Como norma general en XQuery, los datos objeto de consulta se interpretan como datos tipados, y se fuerza a la consulta a tenerlos que interpretar antes de compararlos, con

la única excepción de que sean cadenas o que efectivamente contengan los tipos adecuados.

NOTA DE INTERÉS

En caso de que no se usen esquemas, todos los nodos elemento, están marcados por defecto como de tipo xs:anyType.

El hecho de que los esquemas tengan tipos predefinidos permite usarlos para escribir funciones similares a las de los lenguajes de programación convencionales, aprovechando las estructuras de XML.

Como ejemplo, una función que comprueba si un elemento tiene como padre el nodo documento es:

```
define function is-document-element($e as element()) as xs:boolean
{
if ($e/.. instance of document-node())
then true()
else false()
}
```

De forma que la expresión $e/.. del documento instancia será cierto cuando se evalúe para este nodo.

A pesar de lo dicho, la posibilidad de trabajar con datos pocos tipados, en ocasiones es una ventaja no despreciable que el buen uso de XQuery puede explotar. Como ejemplo, como se verá posteriormente, en el caso de una función que devuelva una secuencia de ítems en el orden inverso al existente en el documento, es una situación en la que parece razonable que no se especifique ni el tipo del parámetro de la función, ni el tipo que será devuelto, ya que es razonable que esta función pueda aplicarse a cualquier secuencia de items.

```
define function reverse ($items)
{
let $count := count ($items)
for $i in 0 to $count
return $items[$count - $i]
}
```

El operador **to** genera sucesiones de enteros (por ejemplo: expresión 1 to 5 genera la sucesión 1,2,3,4,5) y al escribir **reverse** (1 to 5) devuelve la secuencia 5,4,3,2,1. Al no especificar ningún tipo ni para los parámetros ni para el resultado, podría usarse para devolver una secuencia de cualquier otro tipo de datos.

e) Combinar y reestructurar información.

En XQuery las consultas combinan información de una o más fuentes por lo que necesariamente ésta debe reestructurarse para obtener el resultado deseado. Para efectuar estas tareas existen una serie de expresiones, llamadas **FLWOR** (de las iniciales de **FOR, LET, WHERE, ORDER y RETURN** que se pronuncia como en inglés ("flower") que juegan un papel similar al de las instrucciones SELECT FROM- WHERE de SQL al asociar valores a variables para poder obtener resultados.

ENLACE DE INTERÉS

Aquí encontrarás más información acerca de la construcción de XML con XQuery.

CLÁUSULAS FLWOR.

Se llama tupla a una combinación de variables asociadas a una expresión FLWOR formada por las cláusulas **for o/y let**, seguidas por **where o/y order** y obligatoriamente por **return**, ya que invariablemente algo se devuelve como consecuencia de una consulta.

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

Devuelve los libros publicados en el año 2000, asociando la variable $b a cada libro para crear una serie de tuplas.

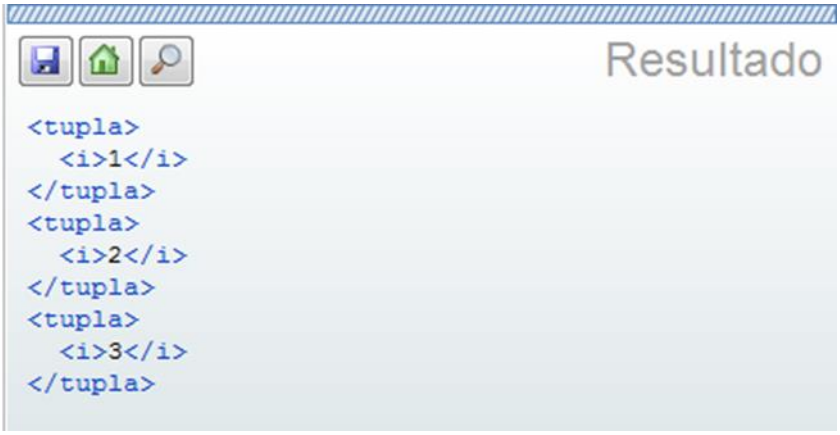
En este ejemplo **where** comprueba si **$b/@anyo** es igual a "2000" de forma que **return** se evalúa para cada tupla que satisfaga la condición.

LAS CLÁUSULAS FOR Y LET.

Toda expresión FLWOR incorpora al menos una de estas cláusulas, existiendo distintas combinaciones posibles para ellas.

```
for $i in (1,2,3)
return <tupla><i>{$i}</i></tupla>
```

Crea un elemento tupla donde $i se asocia a (1,2,3) para construir una sucesión de enteros, dando como resultado:



En cambio, la cláusula **let** asocia una variable al resultado global de una expresión y si no existen en ella ninguna cláusula for, se crea una sola tupla. Si en el anterior se usa **let** en lugar de **for**, esto es:

```
let $i:=(1,2,3)
return <tupla><i>{$i}</i></tupla>
```

El resultado es una única tupla, donde la variable $i se evalúa como la secuencia completa de enteros.

Cuando **let** se usa en combinación con una o más cláusulas **for**, las variables asociadas a **let** se añaden a las tuplas generadas por las cláusulas **for**:

```
for $i in (1,2,3)
let $j := (1,2,3)
return <tupla><i>{$i}</i><j>{$j}</j></tupla>
```

Obtiene como resultado:

```
<tupla>
  <i>1</i>
  <j>1 2 3</j>
</tupla>
<tupla>
  <i>2</i>
  <j>1 2 3</j>
</tupla>
<tupla>
  <i>3</i>
  <j>1 2 3</j>
</tupla>
```

Aplicando la combinación de **for** y **let** en el ejemplo.

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

Obtiene como resultado:

```
<libro>
  <titulo>TCP/IP Ilustrado</titulo>
  <cuenta>1</cuenta>
</libro>
<libro>
  <titulo>Programación Avanzada en el entorno Unix</titulo>
  <cuenta>1</cuenta>
</libro>
<libro>
  <titulo>Datos en la Web</titulo>
  <cuenta>3</cuenta>
</libro>
<libro>
```

```
  <titulo>Economía de la Tecnología y el Contenido de la TV
digital</titulo>
  <cuenta>0</cuenta>
</libro>
```

EJEMPLO PRÁCTICO

Alberto es programador experto en XML y le solicitan asesoramientos sobre la utilización de las cláusulas for y let.

Debe explicar cuál será el resultado en una consulta concreta que las contiene y que es la siguiente:

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

¿Qué datos va a devolver esta consulta?

Solución.

Los datos obtenidos serán:

```
<libro>
  <titulo>TCP/IP Ilustrado</titulo>
  <cuenta>1</cuenta>
</libro>
<libro>
  <titulo>Programación Avanzada en el entorno Unix</titulo>
  <cuenta>1</cuenta>
</libro>
<libro>
  <titulo>Datos en la Web</titulo>
  <cuenta>3</cuenta>
</libro>
<libro>
  <titulo>Economía de la Tecnología y el Contenido de la TV
digital</titulo>
  <cuenta>0</cuenta>
</libro>
```

f) La cláusula where.

Esta cláusula elimina las tuplas que no satisfacen una determinada condición, de forma que return sólo se evalúa para aquellas que la superen. Así, la búsqueda de aquellos libros cuyo precio sea de menos de 50.00 euros es:

```
for $b in doc ("libros.xml")//libro
where $b/precio < 50.00
return $b/titulo
```

Resultado:

```
<titulo>Datos en la Web</titulo>
```

Aunque su análogo en SQL WHERE sólo comprueba valores simples, esta restricción no se da en XQuery, donde la cláusula where puede contener cualquier expresión que se evalúe a un valor booleano.

Para obtener títulos de libros con más de dos autores, se deberá escribir:

```
for $b in doc("libros.xml")//libro
let $c := $b//autor
where count($c) > 2
return $b/titulo
```

Resultado:

```
<titulo>Datos en la Web</titulo>
```

g) La cláusula order.

Esta cláusula ordena las tuplas antes de evaluar return, con el objeto de poder cambiar el orden del resultado. Para ordenar alfabéticamente los títulos, se puede escribir.

```
for $t in doc("libros.xml")//titulo
order by $t
return $t
```

Donde **for** genera una secuencia de tuplas con un nodo título en cada una, y a continuación **order by** las ordena de acuerdo con el valor de los elementos título de forma que return devuelve los elementos título en el orden de las tuplas ordenadas.

El resultado obtenido es:

```
<titulo>Datos en la Web</titulo>
<titulo>Economía de la Tecnología y el Contenido de la TV digital</titulo>
<titulo>Programación Avanzada en el entorno Unix</titulo>
<titulo>TCP/IP Ilustrado</titulo>
```

Esta cláusula permite ordenar de forma ascendente o descendente, para lo que basta con añadir después del nombre del elemento por el que se desea ordenar, la palabra ascending o descending, respectivamente.

EJEMPLO PRÁCTICO

Soraya es programadora en un departamento de informática que se encargan de desarrollar soluciones a empresas cliente que solicitan sus servicios, como es el caso de la explotación de información mediante consultas a bases de datos.

Se le ha planteado una tarea que consiste en conocer cuál será el comportamiento de una consulta determinada y qué datos devolverá.

La consulta es la siguiente:

```
for $a in doc ("libros.xml")//autor
order by $a/apellido descending, $a/nombre descending
return $a
```

¿Qué datos va a devolver esta consulta?

Solución.

Los datos obtenidos serán:

```
<autor>
  <apellido>Suciu</apellido>
  <nombre>Dan</nombre>
</autor>
<autor>
  <apellido>Stevens</apellido>
  <nombre>W.</nombre>
</autor>
<autor>
  <apellido>Stevens</apellido>
  <nombre>W.</nombre>
</autor>
<autor>
  <apellido>Buneman</apellido>
  <nombre>Peter</nombre>
</autor>
<autor>
  <apellido>Abiteboul</apellido>
  <nombre>Serge</nombre>
</autor>
```

h) La cláusula return.

Esta cláusula está en toda expresión FLWOR y también es común en los elementos constructores.

Así, la siguiente consulta usa un constructor de elemento que proporciona el precio de los libros:

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

Resultado:

```
<presupuesto>
  <titulo>TCP/IP Ilustrado</titulo>
  <precio>65.95</precio>
</presupuesto>
<presupuesto>
  <titulo>Programación Avanzada en el entorno Unix</titulo>
  <precio>65.95</precio>
</presupuesto>
<presupuesto>
  <titulo>Datos en la Web</titulo>
  <precio>39.95</precio>
</presupuesto>
<presupuesto>
  <titulo>Economía de la Tecnología y el Contenido de la TV
digital</titulo>
  <precio>129.95</precio>
</presupuesto>
```

La tarea de insertar un elemento nombrecompleto que mantenga el nombre y apellido del autor se resuelve con una consulta que añade un nivel de jerarquía para un nuevo elemento nombrecompleto.

```
for $a in doc("libros.xml")//autor
return
<autor>
<nombrecompleto>{ $a/nombre, $a/apellido }</nombrecompleto>
</autor>
```

Resultado:

```
<autor>
  <nombrecompleto>
    <nombre>Serge</nombre>
    <apellido>Abiteboul</apellido>
```

```
  </nombrecompleto>
</autor>
…
```

i) Expresiones condicionales.

En XQuery se utilizan condiciones de la misma forma que en otros lenguajes; por lo que aparece en XQuery tanto la cláusula **if** como **then** y **else**. Como ejemplo, para obtener el nombre de dos autores de cada libro (incluyendo "et al" si existen más de dos autores), se escribe la consulta:

```
for $b in doc("libros.xml")//libro
return
<libro>
{$b/titulo }
{for $a at $i in $b/autor
where $i <= 2
return              <autor>{string($a/apellido),              ",",
string($a/nombre)}</autor>}
{if (count($b/autor) < 2)
   then <autor> et al.</autor>
else ()}
   </libro>
```

Resultado:

```
<libro>
  <titulo>TCP/IP Ilustrado</titulo>
  <autor>Stevens , W.</autor>
  <autor> et al.</autor>
</libro>
<libro>
  <titulo>Programación Avanzada en el entorno Unix</titulo>
  <autor>Stevens , W.</autor>
  <autor> et al.</autor>
</libro>
<libro>
  <titulo>Datos en la Web</titulo>
  <autor>Abiteboul , Serge</autor>
  <autor>Buneman , Peter</autor>
</libro>
<libro>
  <titulo>Economía de la Tecnología y el Contenido de la TV
digital</titulo>
  <autor> et al.</autor>
</libro>
```

Donde la secuencia vacia() se usa para especificar que una cláusula no devuelve nada.

j) Operar en XQuery.

CUANTIFICADORES.

Para las consultas basadas en una condición, resulta básico determinar si existe al menos un ítem que la satisfaga, así como saber si todos los ítems involucrados en ello lo hacen; ésta es la función de los cuantificadores existencial y universal, respectivamente.

```
for $b in doc("libros.xml")//libro
where some $a in $b/autor
satisfies ($a/apellido="Stevens" and $a/nombre="W.")
return $b/titulo
```

El cuantificador existencial **some** incluido en la cláusula where se encarga de comprobar si existe al menos un autor que satisfaga las condiciones específicas dadas dentro de los paréntesis.

El resultado es:

```
<titulo>TCP/IP Ilustrado</titulo>
<titulo>Programación Avanzada en el entorno Unix</titulo>
```

Un cuantificador universal tiene la función de comprobar si todos los nodos de una determinada secuencia satisfacen la condición objeto de la consulta. Así, para obtener todos los autores que se llamen W. Stevens, puede escribirse:

```
for $b in doc("libros.xml")//libro
where every $a in $b/autor
satisfies ($a/apellido="Stevens" and $a/nombre="W.")
return $b/titulo
```

Se obtiene como resultado:

```
<titulo>TCP/IP Ilustrado</titulo>

<titulo>Programación Avanzada en el entorno Unix</titulo>
<titulo>Economía de la Tecnología y el Contenido de la TV
digital</titulo>
```

Señalar en este ejemplo la aparición en el resultado del título 'Economía de la Tecnología y el Contenido de la TV digital', corresponde a un libro con editores pero sin autores, lo que significa que en este caso la expresión **$b/autor** se ha evaluado sobre una secuencia vacía, obteniendo un resultado que no debe sorprender, ya que coherentemente con los principios de la lógica formal, un cuantificador universal aplicado a una secuencia

vacía siempre devuelve cierto y, en consecuencia, el citado libro aparece en el resultado de la búsqueda.

OPERADORES.

En su mayoría los operadores que se utilizan en XQuery coinciden con los habituales en el resto de lenguajes de programación.

- **Aritméticos:** +, -, *, div, idiv, mod.

- **Comparativos:** eq =,ne !=, lt <, le<=, gt> ge >= .

- **De posición:** dos operadores para determinar la posición de dos nodos en el orden propio de un documento, que son útiles cuando el orden de los elementos es significativo.

- El operador **$a<<$b** devuelve cierto si $a precede a $b en el orden del documento y viceversa con **$a >> $b**.

- **De secuencia**: actúan combinando secuencias para obtener como resultado otra secuencia, sin cambiar el orden interno que éstas tenían en el documento original; union, intersect y except.

Union e **interset** equivalen a las correspondientes operaciones conjuntistas, aunque ahora con secuencias de operandos.

El operador **except**, parte de dos secuencias de nodos y devuelve otra con todos los nodos presentes en el primer operando que no figura en la segunda.

Debe señalarse que se pueden combinar estos operadores de secuencia, siempre que se mantenga el principio de que ningún nodo aparezca dos o más veces.

k) Funciones predefinidas.

En XQuery existen todo un conjunto de funciones predefinidas, muchas de las cuales son habituales en otros lenguajes, y otras están pensadas específicamente para procesar XML. Como en SQL, son muy usadas funciones tales como:

- min(): mínimo
- max():máximo
- count(): contar

- sum(): suma
- avg(): media

Otras que operan con números son:

- round(): redondeo
- floor(): redondeo a la baja
- ceiling():redondeo al alza

Sobre cadenas:

- concat(): concatenación
- string-length(): longitude de la cadena
- starts-with(): cadena que empieza por
- ends-with(): cadena que termina por
- substring(): obtiene una subcadena
- upper-case(): convierte a mayúsculas
- lower-case(): convierte a minúsculas

Otras funciones propias de XML son:

- doc(): obtiene todo el documento
- not(): negación
- empty(): comprueba si está vacío
- exists(): comprueba si existe

l) Otras operaciones.

Por último, esta unidad se ha centrado en las consultas, pero se debe tener en cuenta que también se pueden realizar otras operaciones sobre un XML:

- Insertar → insert node.
- Reemplazo → replace node.
- Borrado → delete node.

ENLACE DE INTERÉS

Aquí encontrarás más información acerca de funciones que se pueden usar con XQuery:

VÍDEO DE INTERÉS

Aquí podrás visualizar como Isabel Jiménez realiza una introducción a XQuery.

3. SISTEMAS GESTORES DE BASES DE DATOS RELACIONALES Y DOCUMENTOS. IMPORTACIÓN Y EXPORTACIÓN DE BASES DE DATOS RELACIONALES EN DIFERENTES FORMATOS.

El proyecto en el que trabajas para importación y exportación de datos está basado en la utilización de bases de datos relacionales, que deberás manejar con soltura para poder realizar un tratamiento correcto de la información en los diferentes formatos que se pueda presentar.

Dos de los sistemas gestores de bases de datos más importantes son Oracle y SQL Server de Microsoft. Estos gestores de bases de datos son capaces de guardar información en formato XML y guardar campos en ese formato.

3.1 Microsoft Sql Server

Sistema de bases de datos relacionales de Microsoft. Se propone como una alternativa a Oracle o MySQL. Es compatible con el tipo de datos XML. Se pueden especificar consultas XQuery con datos XML almacenados en columnas y variables de tipo XML.

SQL (Structured Query Language), es un lenguaje estandarizado de acceso y manipulación de información, contenida en bases de datos, sobre todo de tipo relacional.

Permite operar con las bases de datos con acciones como consulta, inserción, actualización o eliminación, así como la creación y modificación de tablas en las bases de datos, administrando los usuarios y permisos dentro de los sistemas de gestores de bases de datos.

Entre los tipos de consultas encontramos del tipo:

- Data Definition Language (DDL), como conjunto de interacciones del lenguaje SQL, con las que se puede crear, modificar o eliminar las estructuras de datos a nivel índice de tabla o relaciones entre ellas.

- Data Manipulation Language (DML), es el grupo de instrucciones de lenguaje SQL que se utilizan para acceso y manipulación dentro de las tablas de las bases de datos.

- Data Control Language (DCL), integrada por el conjunto de instrucciones relacionadas con el control de acceso a los datos y permisos de los distintos usuarios.

- Transaction Control Language (TDL), son las instrucciones relacionadas con la gestión de las transacciones, entendidas como conjuntos de operaciones que se deben ejecutar todas a la vez.

3.2 Oracle

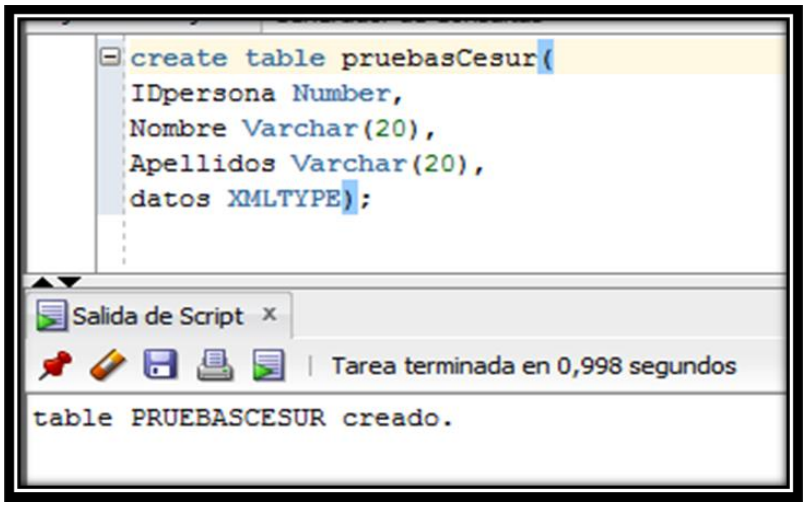
Otro gestor de bases de datos muy usado que se puede descargar de forma gratuita desde su página oficial https://www.oracle.com/es/downloads/. Se ha usado la edición Express para trabajar y realizar los ejemplos que se muestran a continuación.

Logo Oracle.
Fuente: https://www.computing.co.uk/news/4057266/oracles-executives-2022

Oracle ofrece la herramienta SQL Developer, tanto para administrar como hacer consultas pertinentes del diccionario de datos.

Oracle SQL Developer es un entorno de desarrollo integrado, en el cual se han realizado los ejemplos de este apartado. Gracias a Oracle se pueden crear tablas cuyos campos sean de tipo XML y también se les puede asociar un esquema al XML.

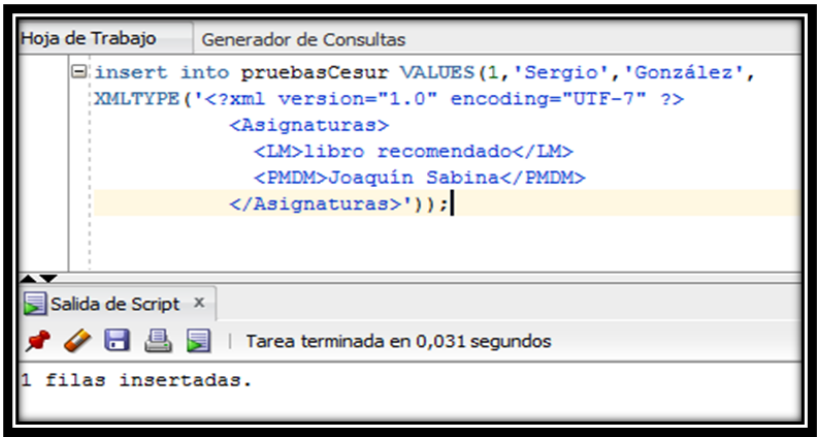
En el siguiente ejemplo se ha creado una tabla *pruebasCesur* con el campo **Datos** de tipo XML (XMLTYPE) en Oracle:



Creación de una base de datos con un campo XML.

Se ha creado una tabla con 4 campos, uno de tipo número, dos de tipo texto y el último de tipo XML.

Ahora, se procede a insertar algunos datos en la tabla, con la peculiaridad de que el campo Dato es de tipo XML.



Insertar datos en una tabla Oracle.

Para comprobar que se han introducido los datos de forma correcta, basta con realizar una consulta sobre la tabla.

VÍDEO DE INTERÉS

Aquí podrás visualizar como se realiza la descarga del motor de bases de datos de Oracle.

4. ALMACENAMIENTO Y MANIPULACIÓN DE INFORMACIÓN EN SISTEMAS NATIVOS Y EN FORMATO XML. HERRAMIENTAS DE TRATAMIENTO Y ALMACENAMIENTO DE INFORMACIÓN EN SISTEMAS NATIVOS

Avanzando en el almacenamiento y manipulación de información, dentro de tu trabajo como programador, debes tratar el uso en sistemas nativos con formato XML, valorando las herramientas disponibles para el tratamiento de datos e información en sistemas nativos.

En este tipo de bases de datos, los datos se almacenan en documentos XML, en una ruta específica del sistema operativo en vez de en tablas.

Este tipo de bases de datos se suele usar para almacenar datos específicos que contienen contenido narrativo o que generan salidas para entornos web.

Entre los sistemas gestores de bases de datos, destacamos los siguientes:

4.1 BaseX

Sistema gestor de bases de datos nativo XML que gracias a los lenguajes de consulta XPath y XQuery, se obtiene el contenido de los documentos.

Se fundamenta en documentos XML en vez de en tablas como los sistemas gestores de bases de datos relacionales.

Logo BaseX.
Fuente: https://www.aristas.net/sistemas-de-gestion-de-bases-de-datos-xml-ii-basex/

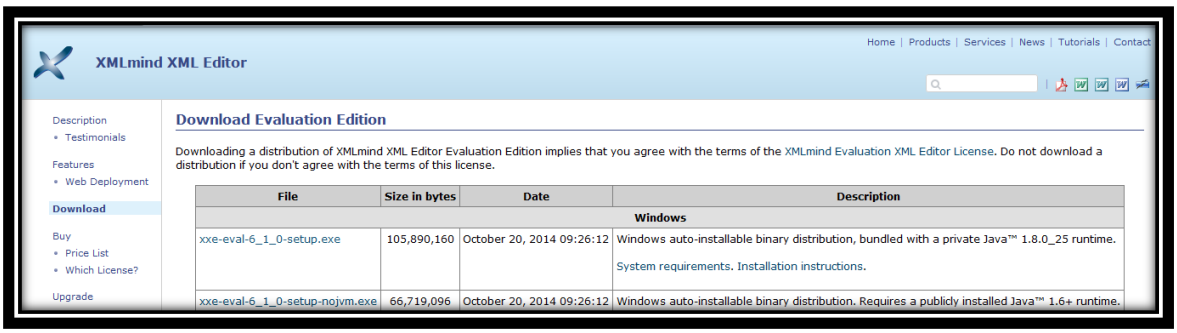
Las consultas mostradas en el siguiente apartado se han realizado con el lenguaje XQuery, sobre BaseX.

PARA SABER MÁS

Accede a esta web donde podrá consultar la documentación oficial de BaseX:

4.2 XMLmind

Es un motor de bases de datos que trabaja con documentos XML. Anteriormente se llamaba QIZX.



Página oficial.

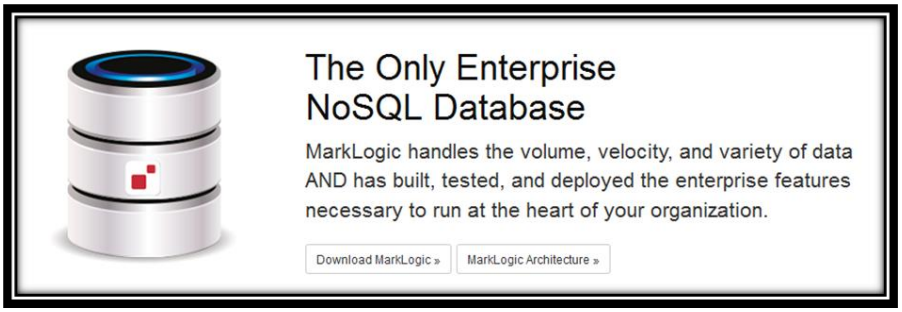
4.3 eXist

Es un sistema de código abierto que sirve para gestionar bases de datos que se almacenan en documentos XML. Entre sus características, destaca que da soporte para XQuery, XPath y XSLT.

Descarga del programa.

4.4 MarkLogic

Ofrece una solución empresarial para almacenar cualquier tipo de documentos, metadatos, RSS, correos electrónicos, XML ...



Página oficial de MarkLogic.

PARA SABER MÁS

Accede a esta web para encontrar más información sobre Bases de datos XML nativas.

EJEMPLO PRÁCTICO

Aurora trabaja como programadora especializada en lenguajes de marcas y en concreto está dedicada casi en exclusiva a desarrollo de nuevas aplicaciones con XML, por lo que le han solicitado que realce una presentación orientada a explicar a los nuevos compañeros del departamento relativa a los sistemas de gestores de datos que existen. ¿Qué sistemas gestores de bases de datos para XML deberían incluir en su presentación?

Solución.

Podría incluir los siguientes sistemas gestores de bases de datos para XML:

- BaseX
  Sistema gestor de bases de datos nativo XML que gracias a los lenguajes de consulta XPath y XQuery, se obtiene el contenido de los documentos.

- XMLmind
  Es un motor de bases de datos que trabaja con documentos XML. Anteriormente se llamaba QIZX.

- eXist
  Es un sistema de código abierto que sirve para gestionar bases de datos que se almacenan en documentos XML.

- MarkLogic
  Ofrece una solución empresarial para almacenar cualquier tipo de documentos, metadatos, RSS, correos electrónicos, XML ...

RESUMEN FINAL

En esta unidad se ha tratado el tema de almacenamiento de la información con un recorrido por los diferentes sistemas de almacenamiento, principalmente con XML, sus características y ventajas e inconvenientes. Continuando con las diferentes tecnologías como XLink, XPointer, XPath o XSLT.

A destacar la importancia XQuery como lenguaje de consultas sobre ficheros XML, que facilita la extracción de datos desde documentos XML, viendo cómo se realiza la consulta y manipulación de información, destacando los conceptos básicos, tipos de datos y capacidades.

La utilización de XQuery require conocer cómo localizar los nodos de estructuras XML y la construcción de estructuras XML, para acciones como la extracción de tipos o la combinación y reestructuración de la información.

Hemos visto las diferentes cláusulas como FLWOR, FOR, LET, WHERE, ORDER, RETURN y las expresiones condicionales, o la forma de operar en XQuery conociendo lo que se conoce como cuantificadores y operadores, así como la existencia de funciones predefinidas como min(), max(), etc.

Dentro del uso de sistemas de bases de datos de tipo relacional, destacamos dos de ellos como son Microsoft Sql Server y la alternativa que ofrece Oracle SQL Developer.

Para finalizar con el almacenamiento y manipulación de información, en este caso en sistemas nativos y en formato XML, donde hemos destacado algunos de ellos como BaseX, XMLmind, eXist o MarkLogic.