UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

UTILIZACIÓN DE XML EN EL DESARROLLO DE UN SISTEMA CON DATOS SEMIESTRUCTURADOS

T E S I S

QUE PARA OBTENER EL TÍTULO DE

LICENCIADA EN CIENCIAS DE LA C O M P U T A C I Ó N

P R E S E N T A :

R O S A E L B A G A R C Í A V E L A S C O

DIRECTOR DE TESIS: DRA. AMPARO LÓPEZ GAONA

MÉXICO,D.F. **2008**

Utilización de XML en el desarrollo de un sistema con datos semiestructurados

1. Datos del alumno
García
Velasco
Rosa Elba
53 07 99 61
Universidad Nacional Autónoma de México
Facultad de Ciencias
Ciencias de la Computación
094061464

2. Datos del Tutor
Dra.
Amparo
López
Gaona

3. Datos del sinodal 1
M. en C.
María Guadalupe Elena
Ibargüengoitia
González

4. Datos del sinodal 2
Mat.
Salvador
López
Mendoza

5. Datos del sinodal 3
M. en C.
Egar Arturo
García
Cárdenas

6. Datos del sinodal 4
M. en C.
Gustavo Arturo
Márquez
Flores

7. Datos del trabajo escrito
Utilización de XML en el desarrollo de un sistema con datos semiestructurados.
125 p
2008

A mis padres con amor.

Agradecimientos

A Dios por no abandonarme en esta lucha, por su apoyo y permitirme terminar este sueño.

A mis padres por que sin ellos no habría logrado cumplir este sueño, por su apoyo tanto económico como moral, por su confianza, por sus porras cuando tenía mucha tarea o trabajo, por la lata que les daba cuando tenían que ir por mi por llegar muy tarde, por todo, gracias. Gracias por enseñarme a luchar por mis sueños y a no darme por vencida.

A mis hermanos por su apoyo, por aguantar mi genio cuando estaba muy presionada. Pollito gracias por ayudarme con el color de mis imágenes. Angelito, gracias por ayudarme con mi computadora, sin ella no habría podido terminar mi trabajo. Israel gracias por tu apoyo, amistad y cariño.

Muy especialmente a mi directora de tesis, a la Dra. Amparo López Gaona, por su apoyo, por su paciencia, por su dedicación por que de no ser por eso yo no habría acabado la tesis. Gracias por enseñarme una vez más que con esfuerzo y dedicación se pueden cumplir los sueños.

A mis sinodales por haberse tomado el tiempo de revisar mi trabajo, por sus comentarios y consejos.

A mis profesores por compartirme sus conocimientos, por sus comentarios, por enseñarme a querer mi carrera. Gracias a la Dra. Hanna Oktaba por su apoyo y por las porras para terminar mi tesis.

A Reyna por su amistad y a su familia por su apoyo en general. A Yasmine por haber sido mi amiga, cómplice y hermana; gracias por tu apoyo, confianza y consejos. A Marisol por su apoyo, por las travesuras, por ayudarme a entender los cálculos, por su amistad y finalmente por sus asesorías en los tramites de la tesis. A Yazmín por su apoyo, amistad y cariño; por sus enseñanzas, por enseñarme a usar LaTeX, por permitirme usar su computadora cuando lo necesitaba, por ser mi amiga, gracias.

A Selene por su amistad, consejos y asesorías, por brindarme su amistad desde que nos conocimos. A Daniel por las porras, por los consejos y por seguir siendo mi amigo a pesar del tiempo. A Yrasen por su amistad y por haber aguantado algunos arranques de mal genio. A todos mis amigos por haber hecho que mi estancia en la universidad fuera maravillosa: Sonia, Humberto. Hugo, Everardo, Verónica, Cesar, Jorge, Gonzalo, Vicky, Rey, Oscar, Iván y a todos los que compartieron un momento conmigo en la facultad. Gracias por haber sido parte importante de esta etapa de mi vida.

A Pati por haber sido un ángel al que dios me permitió conocer.

A Edgar por su apoyo, por haber sido mi quinto sinodal, por su paciencia y por haberse tomado el tiempo de revisar mi trabajo.

También quiero agradecer a mis amigos del trabajo por las asesorías, por echarme porras y animarme a seguir hasta el final: Edgar Iván, Gus y Agustín. A Annis por las porras y amistad.

Índice general

Introducción

Debido al uso de información en grandes cantidades, surge la necesidad de administrarla. Ésta se puede almacenar en diversos medios, normalmente en bases de datos o en algún sistema de archivo. Dependiendo del tipo del que se trate la información se decide cuál es el mejor medio en el que se debe almacenar, para posteriormente administrarla.

Gran parte de la información son datos semiestructurados, es decir, es información que puede ser irregular y no respeta un esquema en particular, sus componentes no siempre son del mismo tipo, no tienen estructura fija ya que pueden cambiarla, son auto descriptivos, etc.; pueden ser representados en forma de árbol y debido a que XML es un lenguaje basado en marcas que proporciona una representación de estructuras jerárquicas, actualmente, también se puede representar la información semiestructurada mediante documentos XML.

Debido a que dicha información se puede representar mediante documentos XML y tomando en cuenta que es una gran cantidad de información surgen diversas tecnologías para el manejo y administración de datos de documentos XML. Algunas de las cuales se mencionarán en el presente trabajo.

Para el almacenamiento de información con datos semiestructurados existen varias opciones de almacenamiento, como son: las bases de datos relacionales, en algún sistema de archivos o en alguna base de datos nativas XML.

Objetivo

El objetivo del presente trabajo es mostrar cómo administrar documentos con datos semiestructurados almacenándolos en una base de datos nativa XML, destacando sus ventajas y desventajas, sin tener que hacer mapeos innecesarios para registrarla en una base de datos relacional y además, mediante un ejemplo, mostrar el uso de las bases de datos nativas XML, observando así, algunas ventajas y desventajas que se tienen. De esta manera, se desea demostrar también que aunque ha avanzado mucho la tecnología en este tema, todavía falta mucho para que tenga la madurez que tienen las bases de datos relacionales.

Para cumplir el objetivo del presente trabajo se desarrolló un sistema que permite a un profesor de un curso almacenar documentos XML en una base de datos nativa XML; dichos documentos contienen los ejercicios del curso. Una vez almacenados, el sistema permitirá consultar los documentos y eliminarlos si así se desea.

Posteriormente, tomando como base esta información el profesor puede consultarla y a partir de ella generar nuevos documentos los cuales estarán personalizados con datos como fecha de entrega, materia, título y alguna nota que desee agregar el profesor al documento.

Aportación de la tesis

El presente trabajo muestra un ejemplo de cómo se puede almacenar información semiestructurada en una base de datos nativa XML destacando las ventajas y desventajas de tales bases de datos. Asimismo, se muestra como se puede utilizar XML como apoyo en las labores docentes, además de aportar una herramienta útil para la docencia, específicamente para la labor de los profesores, evitando que vuelvan a capturar los ejercicios cada vez que dan el curso y que tengan información repetida.

Organización de la tesis

El presente trabajo está dividido en cuatro capítulos: XML, Lenguajes de Consulta, XML y bases de datos y finalmente, Desarrollo de un sistema de generación de documentos.

En los primeros 3 capítulos se describe todo lo necesario para poder desarrollar el sistema. Estos temas son muy importantes para la elaboración y entendimiento del presente trabajo, por lo que se hizo gran énfasis.

En el primer Capítulo se muestra un resumen de lo que es XML y se describen sus características generales; se trató de abarcar este tema de la manera más completa, dado que el elemento principal de este trabajo son los documentos XML. Por otro lado, se describen las características principales de los esquemas.

En el Capítulo 2 se describen las características más importantes de algunos lenguajes de consulta de XML. En la elaboración del sistema se utilizaron algunos de ellos.

En el Capítulo 3 se da una breve introducción de XML y las bases de datos lo que permite conocer algunas maneras de almacenamiento y recuperación de este tipo de documentos. Asimismo, se describen los tipos de bases de datos en donde se puede almacenar éste tipo de información.

Por último en el Capítulo 4 se documenta el sistema desarrollado. Se describe detalladamente de que manera fue desarrollado y como se puede utilizar el sistema en el ámbito docente.

Para finalizar este trabajo, se dan algunas conclusiones generadas a partir de la experiencia en el desarrollo del proyecto utilizando documentos con contenido semiestructurado(documentos XML) y el uso de la base de datos nativa XML eXist. Por otro lado, al final del documento se incluyen dos anexos: en el anexo A se presenta el diseño del sistema (diagramas de clases, casos de uso y diagramas de secuencia) y en el anexo B se muestra el esquema y documentos XSL con los que se hizó la transformación a documentos HTML en el sistema.

Capítulo 1

XML

1.1. Introducción

En el siguiente capítulo se hablará de los antecedentes de XML. Además, se definirán las características principales de XML, los elementos que forman la sintaxis de XML y se describirán dos maneras diferentes para escribir esquemas XML: DTD y XML Schema. El conocimiento de este tema es parte importante para el desarrollo del proyecto.

1.2. Antecedentes

XML(Extensible Markup Language) es un metalenguaje, es decir, un lenguaje hecho para poder construir otros lenguajes, desarrollado por el W3C(Consorcio World Wide Web), principalmente para superar las limitaciones de HTML, ya que no toda la información puede describirse en forma de párrafos, listas, tablas y formularios. XML es un lenguaje basado en SGML(Standard Generalized Markup Language o Lenguaje Generalizado de Marcado), una simplificación y adaptación de SGML.

XML es un metalenguaje que permite definir lenguajes de marca orientados hacia un tipo específico de contenido, es decir, es un conjunto de reglas para definir etiquetas semánticas, lo que permite definir la gramática de lenguajes específicos como son XHTML, SVG, MathML, etc. Además con el tiempo se convirtió en un estándar para el intercambio de información estructurada entre diferentes plataformas, por lo que en la actualidad tiene un papel muy importante, ya que permite intercambiar información entre sistemas de una manera fácil y confiable.

Los lenguajes de marcas no son equivalentes a los lenguajes de programación aunque se definan igualmente como "lenguaje". Los lenguajes de marcas son sistemas complejos de descripción de información, normalmente documentos, que sí se ajustan a SGML, se pueden controlar desde cualquier editor ASCII.

Algunas áreas donde XML es útil son las siguientes:

- Mantenimiento de grandes sitios Web.
- Intercambio de información entre organizaciones o entre sistemas incompatibles.

- Cargas y descargas de bases de datos.
- Aplicaciones de comercio electrónico en las que diferentes organizaciones colaboran para satisfacer a un cliente.
- Creación de libros electrónicos con nuevos lenguajes de marcado para fórmulas matemáticas y químicas.
- Creaciones de nuevos lenguajes. Por ejemplo, VXML(Voice Extensible Markup Language).

1.3. Historia

XML proviene de un lenguaje inventado por IBM en los años setenta, llamado GML (General Markup Language), que surgió por la necesidad que tenía la empresa de almacenar grandes cantidades de información. Este lenguaje gustó a la ISO, por lo que en 1986 crearon SGML (Standard General Markup Language), capaz de adaptarse a un gran número de problemas.

SGML era muy amplio y tenía algunas limitaciones, por lo que se buscó definir un subconjunto de SGML que resolviera algunas de éstas. XML surgió así, como un subconjunto de SGML.

El desarrollo de XML comenzó en 1996 y la primera versión pública salió el 10 de febrero de 1998.

Actualmente hay 2 versiones de XML, mencionadas en la referencia [22]. La primera fue definida inicialmente en 1998. Ha experimentado revisiones de menor importancia desde entonces, sin que fuera necesaria un nuevo número de versión, y está actualmente en su cuarta edición, según lo publicado el 16 de agosto del 2006 por el W3C. Y todavia es recomendado para uso general. La segunda versión, XML 1.1 fue publicada inicialmente el 4 de febrero de 2004, el mismo día que se publicó la tercera edición de XML 1.0, y está actualmente en su segunda edición, según lo publicado el 16 de agosto del 2006. Contiene algunas características que están pensadas para hacer el uso de XML en ciertos casos más fácil, principalmente permitiendo el uso de caracteres de fin de línea usados en plataformas EBCDIC [1], y el uso de escrituras y caracteres ausentes de Unicode 2.0. XML 1.1 no es ampliamente usado y se recomienda el uso solamente para los que necesiten sus características únicas.

XML 1.0 y XML 1.1 difieren en los requerimientos de caracteres usados para los nombres de los elementos y atributos: XML 1.0 sólo permite caracteres definidos en Unicode 2.0, la cual incluye la mayoría de las escrituras del mundo, pero excluye las que fueron agregadas en versiones posteriores. Entre las escrituras excluidas están: móngoles, camboyanos, Amharic, birmanos y otros. En XML 1.1 solamente están prohibidos ciertos caracteres, y todo se permite, mientras que en XML 1.0, sólo se permiten explícitamente ciertos caracteres.

Actualmente hay algunas discusiones sobre lo que sería XML 2.0.

[1] Extended Binary Coded Decimal Interchange Code, código binario que representa caracteres alfanuméricos, controles y signos de puntuación.

1.4. Características Generales

XML es un lenguaje que se utiliza para describir y manipular documentos estructurados. XML es un lenguaje de marcas de documentos. Para que una aplicación pueda trabajar con XML requiere un procesador, el cual lea los documentos y proporcione acceso a su contenido y estructuras.

Por otro lado, es importante recalcar que un documento XML consta de dos estructuras: una lógica y otra física. La física está compuesta por entidades, y la parte lógica está compuesta por declaraciones, elementos, atributos, comentarios, referencias de caracteres, secciones CDATA, instrucciones de procesamiento, etc.

El la sección 1.5 se describen los componentes mencionados.

1.5. Sintaxis

En esta sección se describen los elementos que forman parte de la sintaxis de XML.

1.5.1. Marca

Una marca es cualquier elemento en documento que no es parte de la salida impresa. Las marcas tienen la forma de de etiquetas (<>), las cuales se usan en pares, por ejemplo:

```
<etiqueta>
```

1.5.2. Elemento

Un **elemento** consiste de datos(incluyendo datos nulos) entre etiquetas. Es el pilar de XML, ya que es de lo que está constituido un documento XML. Es el único componente de XML obligatorio. Cada elemento tiene un nombre y puede tener o no contenido. Ejemplo:

```
<telefono>57078961</telefono>
```

En el cual <telefono> es el nombre del elemento y 57078961 es el contenido.

El contenido de un elemento está delimitado por un marcado especial. El cual se conoce como **etiqueta de apertura** y **etiqueta de cierre**. En el ejemplo, la etiqueta de apertura es <telefono>y la de cierre es </telefono>ya que contiene al inicio una diagonal (/). Se requieren las 2 etiquetas.

Los elementos que no tienen información se conocen como **elementos vacíos**. No tienen etiqueta de cierre y lleva una diagonal antes del >. Ejemplo:

```
<correo href="mailto:jdoe@email.com"/>
```

También se puede representar de la siguiente manera:

```
<correo href="mailto:jdoe@email.com"></correo>
```

Los nombres de los elementos deben seguir ciertas normas:

- Comenzar con una letra o con un guión bajo.
- El resto debe estar compuesto por letras, dígitos, guión bajo, el punto o un guión.
- Los nombres no pueden iniciar con la cadena "xml" ya que está reservada para la especificación de XML misma.
- Se deben de omitir espacios, tabuladores y los caracteres =,:,'," en el nombre del elemento.
- No puede haber espacios o tabuladores entre "<" y el nombre del elemento.
- En el contenido no pueden existir los siguientes caracteres &,<>,'," porque entran en conflicto con el marcado del documento. Si se requieren entonces se deben de usar entidades, las cuales se describirán más adelante.

Por convención, los elementos XML se deben escribir en minúsculas.

Los elementos pueden contener a otros elementos (llamado anidamiento de elementos), los cuales pueden contener texto u otros elementos, y así sucesivamente. Las reglas que describen cómo deben anidarse las etiquetas dentro de otras se llaman **modelos de contenido**. Los elementos pueden repetirse.

Al elemento que esta dentro de otro se le conoce como hijo. El elemento que lo contiene se conoce como padre.

El documento debe tener un solo elemento raíz, es decir, todos los elementos deben ser hijos de un solo elemento.

Un documento de XML es un árbol de elementos. No hay límite para la profundidad del árbol.

1.5.3. Atributo

Es posible agregar información adicional a los elementos mediante **atributos**, los cuales son opcionales. Los atributos definen las características o propiedades de un elemento. El uso de atributos permite describir información que afecta la conducta del elemento de una manera clara y efectiva. Tienen nombre y valor. Los nombres de dichos atributos siguen las normas antes mencionadas para el elemento. El nombre del atributo se separa del valor mediante el símbolo(=). Además, el valor del atributo se encierra entre comillas simples o dobles. Ejemplo:

```
<persona sexo="femenino">
o
<persona sexo='femenino'>
```

El elemento puede tener varios atributos, siempre y cuando cada nombre sea único.

Por lo general, los elementos vacios se incluyen en el documento por el valor de sus atributos.

1.5.4. Codificación de caracteres

El atributo `encoding` permite especificar la codificación de caracteres que el documento utilizará. Una **codificación de caracteres** es un método para convertir bytes a caracteres.

Los estándares de los documentos XML siguen Unicode, el cual es extensión del conjunto de caracteres ASCII. Unicode contiene más de 65,000 caracteres y es capaz de codificar todos los caracteres utilizados por los lenguajes escritos de todo el mundo, así como todos los símbolos matemáticos, entre otros. Unicode proporciona un número único para cada carácter, sin importar el idioma o plataforma que se utilice. Además, es compatible con la gran mayoría de sistemas operativos y con los navegadores actuales. Por lo que es una manera de representar caracteres de tal forma que se pueden tener en un texto varios idiomas al mismo tiempo.

```
<xml versión="1.0" encoding="ISO-8859-1"?>
```

En donde encoding=“ISO-8859-1” es el atributo que indica la codificación con la cual se indica el lenguaje en que está escrito el documento. En este caso el documento está escrito en Latin1 [2].

1.5.5. Entidades

Una entidad es un fragmento de información definido como un valor constante, al que se puede hacer referencia mediante un nombre de la siguiente manera:

&nombre; (la cual ha sido definida previamente como <!ENTITY nombre “valor” >).

Mediante las entidades es posible dotar de modularidad a los documentos XML, es decir, tendremos un único documento XML, pero éste fisicamente se encontrará dividido en varios archivos. Lo cual favorece la reutilización y el mantenimiento de documentos de gran tamaño.

XML hace referencia a datos no XML (archivos, páginas web, imagenes, etc.) que no deben ser analizados sintácticamente según las reglas de XML, mediante el uso de entidades.

En general una entidad es una unidad de datos que permite guardar información, por lo que cada documento tiene al menos la entidad del documento mismo. Sirve para dos propósitos:

- Abreviaturas que representan un valor (su contenido).
- Para tener partes del documento fuera de él y luego referenciarlas.

Las entidades permiten:

- Evitar repeticiones de información.
- Subdividir la información en fragmentos.
- Utilizar distintos formatos.

[2]Se puede ver esta información en los manuales de iso_8859_1.

Para insertar una entidad en un documento, se utilizan **referencias de entidad** (representadas por el nombre de la entidad entre un carácter ampersand(&) y uno de punto y coma (;)). En la aplicación, el procesador se encarga de sustituir la referencia de entidad por su contenido, el cual, es analizado por el procesador. En caso de que la entidad se encuentre definida más de una vez, sólo se considerá la primera.

Las entidades se utilizan para representar el contenido de elementos o atributos.

Los tipos de entidades son:

- Entidades generales/parámetro.
- Entidades internas/externas.
- Entidades analizadas/no analizadas.

Entidades generales/parámetro Las entidades generales van a nombrar datos a introducir en la instancia del documento XML. Las entidades parámetro son por su parte una forma de nombrar datos XML a incluir en la DTD[3] (se describirán posteriormente sus características).

Entidades internas/externas. Las entidades internas definen su contenido (datos a incluir en el documento) en el propio documento XML o DTD en el que se declaran. En el caso de las entidades externas, los datos a incluir se encuentran en un archivo externo.

Las entidades internas son:

Reerencia de Entidad	Texto sustituto	Carácter
&amp;	&#38;	&
&lt;	&#60;	<
&gt;	&#62;	>
&apos;	&#39;	’
&quot;	&#34;	”

Se utilizan para representar los caracteres utilizados en el marcado de los archivos XML para que éstos no sean interpretados por el procesador XML.

Entidades analizadas/no analizadas Las entidades analizadas contienen datos XML, se sustituyen y se analizan esos datos. Las entidades no analizadas contienen datos no XML (imágenes, video, audio, etc., es decir, datos no carácter) que no se sustituyen ni analizan.

1.5.6. Atributos Especiales

XML consta de diversos atributos, los cuales aparecen en una etiqueta de apertura. Poseen valores delimitados por comillas que describen el propósito y contenido del elemento. Sólo se explicarán dos de ellos en este punto.

- `xml:space` Este atributo controla si la aplicación puede eliminar espacios. Sus valores son “default” y “preserve”. Ejemplo:

[3]Document Type Definition

```
<p xml:space="default|preserve">¿Qué color es?</p>
```

El valor "default" permite que la aplicación controle espacios en blanco en caso necesario. Si no se incluye este atributo produce el mismo resultado que utilizar este valor.

El valor "preserve" indica que las aplicaciones deben preservar todos los espacios en blanco. Se usa para porciones de documentos donde los espacios en blanco se consideran importantes.

Este atributo es heredado por los elementos hijos del elemento raíz donde es declarado.

- `xml:lang`. Al procesar un documento es útil identificar entre el lenguaje natural o lenguaje formal en el que está escrito el contenido. Se utiliza este atributo para indicar el idioma del contenido del elemento y los valores de los atributos. Los valores de este atributo son identificadores de lenguajes definidos por etiquetas para la identificación de lenguajes [IETF RFC 3066][4], o sus sucesores, además, puede especificarse la cadena vacia. Ejemplo:

```
<p xml:lang="sp-MX">¿Qué color es?</p>
<p xml:lang="en-US">What color is it?</p>
```

El lenguaje especificado por este atributo aplica a los elementos donde está especificado (incluyendo a los valores de sus atributos). Si se especifica la cadena vacía se indica que no hay información disponible del idioma.

1.5.7. Datos de Carácter

Todo texto que no sea marca constituye datos de carácter. Hay dos tipos de datos de carácter:

- Analizados.
- Sin analizar.

Los datos de caracter analizados, o **PCDATA**, son datos de carácter que un procesador XML buscará para el marcado. Son los datos reales de un elemento. Los bloques de estos datos no pueden contener el signo menor que(<) o el ampersand (&). Para usar esos carácteres, se deben expresar como secuencias de escape utilizando entidades internas[5].

Los datos de carácter sin analizar, son aquellos en los que el procesador XML no busca marcado, es decir, son ignorados por el procesador. Para no tener que poner los carácteres mencionados anteriormente de esa forma se pueden crear secciones especiales llamadas **CDATA**, en las que los datos de carácter son tratados como texto.

Están delimitadas por "<![CDATA[" y "]]>". El procesador XML ignora todo el marcado excepto]>(lo que significa que no es posible incluir una sección CDATA dentro de otra sección cdata). CDATA significa datos de caracteres.

[4]Tags for the Identification of Languages
[5]Mencionadas en la sección 1.4.5.

1.5.8. Comentarios

Para insertar comentarios en un documento, deben estar entre los caracteres <!- - y - -> ya que estos son ignorados por el procesador XML. Los comentarios no deben insertarse entre el marcado. Deben colocarse antes o después de él.

1.5.9. Instrucciones de Procesamiento

Las PIs (siglas en inglés de Instrucciones de Procesamiento) son un mecanismo para insertar instrucciones que proporcionen información a las aplicaciones que procesan el documento. Las PIs no son parte del documento, pero deben ser pasadas a la aplicación. El analizador pasa esa información a la aplicación que realiza la llamada. Las PIs se utilizan principalmente para indicar a la aplicación el modo de administrar los datos de un documento XML. Dependiendo del procesador se interpretarán determinadas instrucciones de procesamiento, pero otras no. Tienen la siguiente sintaxis:

<? texto destino ? >

El destino de una instrucción de procesamiento es un nombre que será reconocido por la aplicación que utilice el documento XML.

Una declaración XML es una instrucción de procesamiento. Ejemplo:

```
<?xml versión="1.0" encoding="ISO-8859-1"?>
```

1.6. Esquemas

Al conjunto de reglas para definir un documento XML se les denomina **esquema**. Estos ofrecen mayor consistencia en los documentos. Existen dos sistemas fundamentales para escribir los esquemas: **DTD y XML Schema.**

1.6.1. DTD

Una Definición de Tipo de Documento, (DTD,*Document Type Definition*) es un medio para definir explícitamente la estructura y sintaxis de un documento XML. Es decir, la DTD es un mecanismo para describir cada objeto (elemento, atributo y demás) que puede aparecer en el documento, empezando con los elementos. Se usa cuando se quiere tener la seguridad de que un archivo XML pueda ser interpretado por cualquier herramienta. Además de que mantiene la consistencia entre los documentos que utilicen la misma DTD. De tal forma que dichos documentos pueden ser validados.

Una DTD puede ser definida dentro del documento XML, o declarada como una referencia externa.

Definición de DTD Interna: Si la DTD es definida dentro del archivo XML, debe estar entre una declaración DOCTYPE con la siguiente sintaxis:

<!DOCTYPE root-element [element-declarations]>

Declaración de DTD Externa: Si la DTD es definida como un archivo externo, debe estar entre una declaración DOCTYPE con la siguiente sintaxis:

<!DOCTYPE root-element SYSTEM "filename">

Más adelante, en este capítulo, se describirá la sintaxis de la DTD. Y se podrá interpretar mejor el significado de las 2 maneras de declarar una DTD.

Todas las entidades que se incluyen en un documento XML deben estar predefinidas en la DTD. Una DTD no es obligatoria, a diferencia de SGML, es posible crear documentos XML sin DTD.

Esto da lugar a dos tipos de documentos:

- Documentos XML bien formados, es decir, aquellos que respetan la sintaxis de XML.
- Documentos XML válidos, es decir, aquellos que además de bien formados, siguen la estructura definida en una DTD.

Posteriormente se revisará este tema con más detalle.

Los documentos XML utilizan las marcas especificadas en la DTD mediante estructuras básicas: prólogo,elementos de documento, elementos, atributos y contenido.

Prólogo

El prólogo del documento contiene toda la información relativa al documento, como la instrucción que declara el documento de tipo XML (declaración XML), o la que vincula el documento a su DTD.

La "**declaración**" XML es opcional, pero si existe debe ser la primera instrucción del prólogo. Ésta declaración identifica al documento como XML. También indica la versión de XML que se utiliza, así como el atributo que indica la codificación empleada, la cual es opcional. Aunque es recomendada debido a que ya existe la versión 1.1 y si no se indica pueden existir problemas. Ejemplo:

```
<?xml versión="1.0" encoding="ISO-8859-1"?>
```

La segunda línea, o "**declaración de tipo de documento XML**" define qué tipo de documento se está creando, es decir, anexa una DTD al documento.

```
<!DOCTYPE libreta-direcciones SYSTEM "libreta-direcciones.dtd">
```

Ésta consta del marcado (<!DOCTYPE), el nombre del elemento de nivel más bajo, es decir el elemento raíz del documento (libreta-direcciones), la DTD (SYSTEM "libreta-direcciones.dtd") y el ">" de cierre. La declaración del tipo de documento aparece al principio del documento XML, después de la declaración del XML. En el caso de la declaración de una DTD interna en lugar del nombre del archivo se incluye la definición de la DTD después del elemento raíz del documento y entre corchetes "[definición de la DTD]".

```
<!DOCTYPE libreta-direcciones [element-declarations]>
```

Declaración de documento Independiente: También en el prólogo dentro de la "declaración" se puede incluir una declaración de documento standalone que indica si el documento puede ser procesado sin necesidad de acceder a definiciones externas. El valor "yes" indica que no existen declaraciones de marcas externas a la entidad documento. El valor "no" indica que existe o que pueden haber dichas declaraciones de marcas.

Ejemplo:

```
<?xml version="1.0" standalone='yes'?>
```

Elemento del Documento

El `elemento de documento` o `elemento raíz` es el elemento de mayor nivel, es decir, el primer elemento e incluye a todos los demás elementos y contenido. Representa el documento en su totalidad; los demás elementos representan un componente del documento. Hay exactamente un elemento de documento.

1.6.2. Elemento

Todo documento XML contiene uno o más elementos. La sintaxis para definir elementos en una DTD es la siguiente:

<!ELEMENT nombreElemento (modelo de contenido)>

La parte *regla*, o modelo de contenido, de una declaración de elemento especifica el contenido que está permitido. Además, que elementos deben estar anidados dentro de este elemento, el orden en que deben aparecer y cuantas veces, presentándolos separados por comas. El orden en el que aparecen indica el orden en que deben de aparecer los elementos.

Ejemplo:

```
<!ELEMENT nombre (nombreCompleto+, apellidoPaterno, apellidoMaterno)>
```

El modelo de contenido del elemento nombre indica que nombreCompleto, apellidoPaterno y apellidoMaterno estan anidados dentro del elemento nombre. Ellos deben aparecer en el orden de izquierda a derecha y además el elemento nombreCompleto puede aparecer más de una vez.

Atributos

Los atributos deben estar declarados en la DTD. Los atributos de elementos se declaran con la ATTLIST, su sintaxis es la siguiente:

<!ATTLIST nombre_elemento nombre_atributo tipo_atributo predeterminado>

Ejemplo:

```
<!ATTLIST tel preferente(true | false) "false">
```

En donde el nombre del atributo es preferente, el tipo del atributo (true |false) y un valor predeterminado ("false").

Se pueden declarar varios atributos para el mismo elemento en una declaración de atributo.

La DTD proporciona más control sobre el contenido de los atributos que sobre el contenido de los elementos. Los atributos están divididos en tres categorías:

- Atributos de cadenas de caracteres que contienen texto.
- Atributos de validación que restringen el contenido del atributo
- Atributos de tipo enumerado que aceptan un valor dentro de una lista.

La DTD puede especificar un valor predeterminado para el atributo. Si el documento no incluye el atributo se asume que tiene el valor predeterminado. El valor predeterminado puede tomar uno de los 4 valores siguientes:

- #REQUIRED significa que el valor debe ser proporcionado en el documento, es decir, no tiene valor por defecto.
- #IMPLIED significa que si no se proporciona un valor, la aplicación debe usar su propio valor predeterminado. Si el valor no se omite, no se producirá error de análisis.
- #FIXED seguido de un valor significa que el valor del atributo debe ser el valor declarado dentro de la DTD, es decir el valor del atributo es constante, por lo que no puede cambiarse en el XML.
- defaultValue Un valor literal (predeterminado) significa que el atributo tomará este valor si no se le proporciona ningún valor en el documento.

Los tipos de atributos son los siguientes:

- CDATA(datos de caracteres) que puede ser cualquier valor menos el de los caracteres especiales en XML (<,>,",',&).
- ID El valor será un identificador único. Ejemplo:

```
<!ELEMENT alumno (#PCDATA)>
<!ATTLIST alumno numero_cuenta ID #REQUIRED>
```

- IDREF hace referencia a un elemento con un atributo ID. Ejemplo:

```
<!ELEMENT alumno (#PCDATA)>
<!ATTLIST alumno id IDREF #REQUIRED>
```

- Enumeraciones Sólo pueden contener valores de entre una lista de valores. Cada valor aparecerá separado por un "|".

  Ejemplo:

```
<!ATTLIST telefono lugar (oficina | casa | celular) "oficina">
```

  si no está presente el valor del atributo, el default será "oficina".

- NOTATION(Anotación) este tipo permite al autor declarar que su valor se ajusta a una anotación declarada. Las anotaciones identifican por nombre el formato de entidad externas, el formato de los elementos que llevan un atributo NOTATION, o la aplicación para la cual está destinada una instrucción de procesamiento. Ejemplo:

```
<!ATTLIST mensaje fecha NOTATION (ISO-DATE | EUROPEAN-DATE)
  #REQUIRED>
```

Para declarar las anotaciones se utiliza:

<!NOTATION nombre SYSTEM "información de la anotación">

- NMTOKEN(Autenticaciones) El valor del atributo debe ser un símbolo(token) de nombre. Los símbolos de nombre permiten valores de datos de caracteres, pero son más limitados que CDATA. Puede incluir letras, números y algunos signos de puntuación, como puntos, guiones, caracteres de subrayado y dos puntos. Los valores de los símbolos de nombre no pueden contener ningún carácter de espacio.

Modelos de Contenido

Para la mayoría de los elementos, el modelo de contenido es una lista de elementos. También puede ser alguna de las siguientes palabras reservadas:

- #PCDATA (*Parser Character Data*) se pone para indicar que los datos de caracteres serán analizados, y significa que el elemento puede contener texto. Es el tipo más básico de contenido que se puede encontrar para un elemento. Se le conoce como contenido de datos. El carácter # significa el tipo, por lo que se puede leer como ün tipo de dato analizable". No tiene anidación dentro de él.

- EMPTY indica que el elemento es un elemento vacío, es decir, un elemento que no tiene contenido. Se utiliza para elementos hoja, es decir, elementos que no contienen elementos hijo.

- ANY indica que el elemento puede contener datos de caracteres, tipos de elementos declarados o una mezcla de ambos (no hay ninguna restricción). No suele usarse ya que permite que los elementos no estén adecuadamente estructurados.

- MIXED indica que puede tener tanto caracteres #PCDATA como otros elementos especificados. Es similar a ANY, pero requiere que los contenidos estén especificados.

Ejemplos:

```
<!ELEMENT img EMPTY>
<!ELEMENT nombre #PCDATA>
<!ELEMENT note ANY>
<!ELEMENT item (#PCDATA|item)*>
```

Cuando el modelo de contenido consta de otros elementos de marca se le conoce como contenido de elementos.

Indicadores de Ocurrencia

Los caracteres más (+), asterisco(*) e interrogación(?) del contenido de elementos son indicadores de ocurrencia. Indican qué elementos se pueden repetir, cuantas veces y cómo dentro de la lista de elementos hijos.

- Un elemento que no está seguido por un indicador de ocurrencia debe aparecer una y sólo una vez dentro del elemento en el cual fue definido.
- Un elemento seguido por asterisco puede aparecer 0 o más veces dentro del elemento en el cual fue definido. El elemento es opcional pero, si está incluido, puede aparecer indefinidamente.
- Un elemento seguido por un signo de interrogación puede aparecer una vez o no aparecer dentro del elemento en el cual fue definido. Este signo indica que el elemento es opcional y, si se incluye, no se puede repetir.
- Un elemento seguido por el signo más puede aparecer 1 o más veces dentro del elemento en el cual fue definido. Puede aparecer indefinidamente, pero al menos una vez.

Conectores

La coma(,) y la barra vertical (|) son conectores. Los conectores separan a los hijos en el modelo de contenido, e indican el orden en el cual pueden aparecer los hijos. Los conectores son:

- El carácter coma que indica que los elementos a la derecha y a la izquierda de la coma deben aparecer en el mismo orden dentro del documento.
- El carácter | que indica que sólo uno de los elementos de la derecha o de la izquierda de la barra vertical debe aparecer dentro del documento.

```
<!ELEMENT pregunta(contenido | imagen)*>
```

Indica que dentro del elemento pregunta, debe de ir el elemeno contenido o el elemento imagen.

```
<!ELEMENT pregunta(contenido,imagen)*>
```

Indica que dentro del elemento pregunta debe de ir el elemento contenido y el elemento imagen.

Subconjuntos Internos y Externos

La DTD está dividida en subconjuntos internos y externos. El subconjunto interno de la DTD está incluido entre corchetes dentro de la declaración de tipo de documento. El externo está almacenado en una entidad separada y es referenciado desde la declaración de tipo de documento a través de un identificador.

Ejemplo de subconjunto externo:

```
<!DOCTYPE libreta-direcciones SYSTEM "libreta-direcciones.dtd">
```

Hay 2 tipos de identificadores: **identificadores de sistema e identificadores públicos**. Las palabras reservadas SYSTEM y PUBLIC indican, respectivamente, el tipo de identificador.

- Un identificador de sistema es un Identificador Universal de Recursos(URI) que apunta hacia la DTD. El URI es un grupo de URLs. El procesador debe de bajar el documento desde una URI. Define la localización física de una entidad externa.
- Un identificador público se usa para manejar copias locales de DTDs.

Si se utilizan ambos identificadores, el identificador público debe ir primero. Un documento XML es independiente cuando todas las entidades están en subconjuntos internos de la DTD, ya que el procesador no necesita descargar entidades externas.

Formato de Identificadores públicos

Un FPI (Formal Public Identifier), Formato de Identificadores es un nombre simbólico único que puede ser usado para mapear una cierta localización física.

Hay 4 partes separadas por "//";

- La primer parte puede ser el carácter + si la organización está registrada con ISO; el carácter - si el nombre de la organización no esta registrada en ISO o "ISO"si es aprobada por un cómité de estandar formal.
- La segunda es el nombre del grupo o de la persona responsable de la DTD.
- La tercera parte es la descripción de la DTD (número del documento y versión)
- La parte final es el idioma.

Ejemplo:

```
"(ISO|+|-)//NombreGrupo//Descripción del documento//id de lenguaje"
"-//W3C//DTD XHTML 1.0 Transitional//EN"
```

Limitaciones

Un esquema basado en una DTD tiene algunas limitaciones como son:

- No permite definir elementos locales que sólo sean válidos dentro de otros elementos.
- Es poco flexible la definición de elementos con contenido mixto, es decir, que incluyan otros elementos además de texto.
- No es posible indicar a que tipo de dato(número, cadena, fecha, etc.) corresponde un atributo o texto de un elemento.
- La sintaxis no es XML, lo cual implica conocer el lenguaje de las DTDs.
- No permiten especificar secuencias no ordenadas.

La necesidad de superar estas limitaciones propicia la aparición de otros lenguajes de esquema como XML Schema.

1.6.3. XML Schema

Introducción

Un XML Schema es un lenguaje de esquema escrito en XML basado en la gramática, que al igual que la DTD, se utiliza para describir la estructura y las restricciones del contenido de los documentos XML. Los esquemas se concibieron como una alternativa a las DTD, más compleja, intentando superar sus limitaciones. Su principal aportación es el gran número de tipos de datos que incorpora. Define por ejemplo qué elementos y atributos pueden contener los documentos. La extensión de un esquema es .xsd.

El World Wide Web Consortium(W3C) empezó a trabajar en XML Schema en 1998 [50]. La primera versión se convirtió en una recomendación oficial en mayo de 2001. Una segunda edición revisada está disponible desde octubre del 2004.

Esta recomendación está desarrollada en 3 partes:

- XML Schema Parte 0 Primer: es una introducción, una primera aproximación al lenguaje.
- XML Schema Parte 1 Estructuras: Es una extensa descripción de los componentes del lenguaje.
- XML Schema Parte 2 Tipos de datos: Complemento de la Parte 1 con la definición de los tipos de datos incorporados y sus restricciones.

Un esquema permite definir elementos globales (aquellos que deben utilizarse de la misma forma en el documento XML, es decir, se puede referenciar en cualquier sitio del esquema) y locales (aquellos que pueden tener un significado particular en un momento determinado por el contexto, por lo que sólo se puede usar en una parte del contenido de esquema). Los elementos, atributos y la mayoría de los componentes de un esquema pueden ser globales o sólo estar referenciados a otros componentes.

Un esquema comienza con una declaración XML, seguida de una del espacio de nombres de XML Schema. Los espacios de nombres se explicarán más adelante. El elemento schema es el elemento raíz del documento en el que se define el esquema.

Ejemplo:

```
<?xml version="1.0">
   <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
   </xsd:schema>
```

Tipos de componentes

Los esquemas se construyen a partir de diferentes tipos de componentes:

- Elemento
- Atributo
- Tipo simple

- Tipo complejo
- Notación
- Grupo modelo nombrado (named model group)
- Grupo de atributos
- Restricción identidad (identity constraint)

Estos componentes ofrecen la posibilidad de combinar características de alto o bajo nivel:

- Alto nivel: Se encargan de ofrecer un significado semántico del contenido del documento. Analizan el contenido y extraen de él un significado.
- Bajo nivel: Son características más concretas del documento que están incluidas en los diferentes campos del esquema y se accede a ellas de manera directa.

XML Schema fue diseñado completamente alrededor de namespaces y soporta datos típicos de los lenguajes de programación, como también tipos personalizados simples y complejos.

Espacio de Nombres(NAMESPACES)

Los espacios de nombres son una colección de nombres, identificados por un Identificador Universal de Recursos(URI), que proporciona un método simple para identificar de manera única a los nombre de los elementos y atributos en un documento XML. Se puede encontrar una analogía entre estos y los paquetes en Java. Cada espacio de nombre contiene elementos y atributos que están estrechamente relacionados.

Los espacios de nombre ayudan a evitar confusiones con elementos que tienen el mismo nombre pero son definidos en diferentes DTD's.

La declaración de espacios de nombres se realiza mediante atributos con el identificador xmlns seguidos por el prefijo a utilizar.

El nombre del espacio de nombre es el URI. El URI sólo se utiliza para asegurar la unicidad de los nombres de tal modo que es importante asegurarse que los URIs sean únicos.

Un espacio de nombre debería tener el formato URL. Es decir comenzar con HTTP, seguido de su nombre de dominio y etiquetas de espacio de nombres opcionales con el formato de un nombre de directorio,una descripción muy corta del espacio de nombres y un número de versión opcional. Ejemplo:

```
http://www.domain.com/ns/rivers/1.0
```

ns es la información opcional del espacio de nombres. Rivers es la descripción breve de este espacio de nombres. Y 1.0 es el número de versión.

Para la versión XML 1.1 un espacio de nombre es identificado mediante una referencia IRI (Internationalized Resource Identifier, cadena que puede ser convertida en una referencia URI aplicando algunos pasos); los nombre de elemento y atributo pueden ser colocados en un espacio de nombre XML.Las referencias IRI pueden contener carácteres que no están permitidos en los nombres (caracteres Unicode).

Elemento

Los elementos que van a aparecer en un documento deben aparecer en el esquema con un elemento element, cuyos atributos más significativos son:

- maxOcurrs y minOcurrs: determinan el número máximo (mínimo) de veces que el elemento puede aparecer en un documento.
- name: especifica el nombre usado para referenciar un tipo de elemento tanto en el resto del esquema como en el documento.
- ref: hace referencia a un elemento global ya declarado. Es mutuamente excluyente con el atributo name.
- type: especifica el tipo de dato del elemento, con un valor nombre referido a un tipo global simple o complejo.

Ejemplo:

```
<xsd:element name="algunElemento" type="xs:string">
```

Atributo

Los atributos se declaran por medio del elemento attribute que a su vez tiene una serie de atributos que ponen algunas restricciones a las propiedades de esta declaración. Un atributo global debe declararse como hijo del elemento schema. En el caso de que se quiera declara un atributo de forma local, se coloca como hijo de la declaración del elemento correspondiente. Como se muestra en el siguiente ejemplo:

```
<xsd:schema>
 <xsd:element name="algunElemento">
  <xsd:complexType>
    <xsd:attribute name="algunAtributo">
  </xsd:complexType>
</xsd:element name="algunElemento">
</xsd:schema\textgreater
```

Un atributo del elemento attribute es **use**, que permite delimitar el uso del atributo en un documento. Puede tomar los siguientes valores: optional, prohibited y required. Por defecto, los atributos son opcionales.

```
<xsd:attribute name="idioma" type="xsd:string" use="required"/>
```

Tipos complejos y simples

Un concepto fundamental de un esquema es el tipo de contenido. En un esquema se puede dividir un documento en dos tipos de contenido: simple y complejo. Los elementos de tipo simple son aquellos que sólo pueden contener texto. Los de tipo complejo pueden contener otros elementos, e incluso atributos, los cuales son considerados de tipo texto. Ambos, pueden tener **nombre**, en cuyo caso podrán utilizarse en otras partes del esquema, o bien ser **anónimos**, es decir, que sólo se utilizan en el elemento en que aparece la definición.

Tipo simple

Un tipo de datos simple puede ser **predefinido** (incorporado al espacio de nombres de los Esquemas) o **derivado** (definido por el autor del esquema). Un elemento simple es un elemento que sólo puede contener texto (cualquier tipo de dato), pero no a otros elementos ni atributos.

Para definir un elemento de tipo simple, se utiliza la sintaxis:
<xsd:element name=”xxx” type=”xsd:yyy”/>

Existen varios tipos simples, como fechas, enteros y cadenas que se pueden utilizar. Ejemplo:

```
<xsd:element name="peso" type="xsd:string"/>
<xsd:element name="ejemplares" type="xsd:integer"/>
```

El elemento peso estará limitado a una cadena y el elemento ejemplares a un entero. El atributo type puede tomar los siguientes valores:

- xsd:string si el elemento va a contener una cadena de caracteres.
- xsd:decimal si va a contener numeros decimales.
- xsd:boolean si el elemento contiene los valores booleanos(0,1).
- xsd:date si el elemento es una fecha.
- xsd:time si el elemento va a ser la hora del día.
- xsd:uri=reference si el elemento contiene un URL.

Un elemento simple puede tener un valor por defecto y un valor fijo. Esto se indica mediante los atributos default y fixed.

```
<xsd:element name="color" type="xsd:string" default="rojo"/>
```

A partir de datos simples se pueden obtener nuevos tipos de datos simples, ya sea por definición (listas y unión) o por derivación(restricción). Para definir y derivar nuevos tipos se usa el elemento simpleType que constituye la representación de un tipo simple en un documento, identificandolo con un nombre. Los contenidos posibles de este elemento son: List, Union y Restriction.

- **List**: Sucesión de elementos simples delimitados por espacios en blanco. Por ejemplo:

```
<ElementoLista>valor1 valor2 valor3</ElementoLista>
```

Los documentos de una lista pueden ser globales o locales, siempre y cuando todos los valores sean del mismo tipo (identificado por el atributo itemType).

- **Union**: Crea un nuevo tipo simple permitiendo que el valor de un elemento o atributo contenga una o más instancias con tipos construidos con la unión de múltiples tipos atómicos o listas. Su principal atributo es memberTypes que proporciona la relación de sus posibles tipos globales.

```
<xsd:simpleType name="UnionNumeros">
   <xsd:union memberTypes="Nombre listaEnteros"/>
</xsd:simpleType>
```

- **Restriction**: Proporciona la funcionalidad de especificar facetas que restringen un tipo simple ya definido que se llama tipo `base`. Con el uso conjunto de restriction (para indicar el tipo base e identificar las facetas que restringen su rango de valores) y de simpleType (para definir y dar nombre al nuevo tipo) se derivan nuevos tipos simples. Especifica los valores permitidos de las facetas del tipo base, por medio de la sintaxis:

```
<xsd:simpleType name="nombre">
   <xsd:restriction base="xsd:String"/>
      <xsd:faceta value="valor"/>
      <xsd:faceta value="valor"/>
      ....
   </xsd:restriction/>
</xsd:simpleType>
```

Tipos de facetas:

- El valor comprendido en un rango (maxExclusive, maxInclusive, minInclusive y minInclusive).
- El valor está restringido a un conjunto de valores posibles (totalDigits).
- Restringir el valor de un elemento a una serie de caracteres (pattern).
- Longitud de los valores de los elementos (length, minLength, maxLength)

Tipo complejo

Los elementos del tipo complejo tienden a describir la estructura de un documento más que su contenido. Son elementos que contienen a otros elementos hijos, o que tienen atributos. Hay 4 clases fundamentales de tipos complejos:

- Los elementos que contienen otros elementos.
- Los elementos vacíos.

- Los que sólo incluyen texto, elementos no vacios con atributos.
- Los que además contienen texto.

Los atributos se declaran de forma similar a la declaración de los elementos de tipo simples. <xsd:attribute name=”xxx” type=”xsd:yyy”/>

Al igual que los elementos de tipo simple, pueden tener valores por defecto y valores fijos: <xsd:attribute name=”idioma” type=”xsd:string” default=”ES”/>

Para definir elementos complejos se utiliza la siguiente sintaxis:

```
<xsd:element name="nombreElementoComplejo">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="nombreElemSimple" type="xsd:string">
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Se puede usar otra sintaxis para reutilizar la “definición” de los elementos hijos en varios elementos:

```
<xsd:element name="nombreElemSimple1" type="nombreElemHijo"/>
<xsd:element name="nombreElemSimple2" type="nombreElemHijo"/>

<xsd:complexType  name="nombreElemHijo">
  <xsd:sequence>
     <xsd:element name="nombreElemSimpleCompartido1" type="xsd:string"/>
     <xsd:element name="nombreElemSimpleCompartido2" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

En la declaración de elementos complejos, es posible utilizar un mecanismo de “herencia” para reutilizar o extender elementos definidos con anterioridad.

```
<xsd:element name="nombreElemComplejo1" type="nombreElemento">

<xsd:complexType  name="nombreElemReutilizado">
  <xsd:sequence>
    <xsd:element name="nombreElemSimple1" type="xsd:string">
    <xsd:element name="nombreElemSimple2" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType  name="nombreElemento">
  <xsd:complexContent>
    <xsd:extension base="nombreElemReutilizado">
```

```
      <xsd:sequence>
        <xsd:element name="nombreElemSimple3" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

La sintaxis para declarar elementos vacios con atributos es:

```
<xsd:element name="producto">
  <xsd:complexType  name="informacion">
    <xsd:attribute name="id" type="xsd:positiveInteger">
  </xsd:complexType>
</xsd:element>
```

Grupos de elementos

Se llama **grupo** a los distintos elementos que puede contener un tipo complejo. Es básico definir en el esquema, el modo en que estos grupos deben de aparecer. El modo puede ser uno de los siguientes:

- **sequence**: Indica un determinado orden o secuencia.
- **choice**: Proporciona un mecanismo para describir una selección dentro de un grupo.
- **all**: Especifica un conjunto sin ordenar de un grupo de elementos, de forma que cada uno de ellos pueda estar presente en un documento, sin ninguna restricción respecto al orden. Sólo pueden aparecer 1 vez.

Declaraciones locales y globales

En XML Schema el contexto es muy importante. Los componentes del esquema, incluidos los elementos, atributos, tipos simples y complejos, que son declarados en el nivel superior del esquema, son considerados globales y se pueden utilizar en todo el esquema. Sin embargo, las declaraciones globales de los elementos no determinan el lugar donde puede aparecer un elemento en el documento, sólo su apariencia. Por lo que se debe hacer referencia a la declaración global de un elemento para que aparezca realmente en él (El elemento raíz es referenciado automaticamente, independientemente de haber sido declarado local o global).

Cuando se define un tipo complejo, puede hacer referencia a los elementos existentes declarados globalmente, o bien declarar y definir elementos nuevos. Estos elementos declarados localmente están limitados a la definición de tipo complejo en la cual han sido declarados. Además, sus nombres requieren ser únicos en el contexto que aparecen. Estos elementos son referenciados automáticamente, es decir, la posición en la que son definidos también determina en que parte del documento deben aparecer.

Diferencias de la DTD con XML Schema

Algunas diferencias de la DTD con los esquemas XML son:

- Las DTD's están escritas con una sintaxis diferente a los documentos XML. Por lo que no pueden ser analizados con un analizador XML. Los esquemas usan sintaxis XML.
- En la DTD todas las declaraciones son globales, es decir, no se pueden definir dos elementos diferentes con el mismo nombre, incluso si aparecen en contextos separados.
- Los DTD no pueden controlar el tipo de información que tiene un documento o atributo en particular. Los esquemas XML si, ya que permiten especificar los tipos de datos.
- Los esquemas proporcionan protección de errores.

En la DTD, con PCDATA sólo se indica que son caracteres analizables. Sin embargo, no se menciona el tipo de dato, es decir, si el dato es una cadena, un entero, etc. Y en el caso específico del número, se debería indicar que lo que se espera es un entero.

Limitaciones de los esquemas

- No soporta entidades.
- El tamaño de un esquema puede ser demasiado grande.
- Legibilidad de las especificaciones. XML no siempre es legible.
- Complejidad de la especificación.

Documentos Bien Formados y Válidos

XML reconoce dos clases de documentos: el documento bien formado y el válido. Bien formado significa que sigue la sintaxis de XML. Los documentos XML tienen la mezcla correcta de etiquetas de apertura y de cierre, los atributos tienen las comillas que debe tener, etc.

Los documentos bien formados no tienen DTD, así que el procesador XML no puede verificar su estructura. Sólo verifica que siga las normas de sintaxis.

Los documentos válidos son más estrictos. No sólo siguen las normas de sintaxis, también cumplen con una estructura específica, como la que se describe en una DTD.

Los documentos válidos tienen una DTD o esquema. El procesador verificará que los documentos tengan una sintaxis correcta y se asegura que siga la estructura correcta descrita en la DTD.

Existen varios métodos para validar los documentos XML. Los métodos más usados son la DTD de XML versión 1.9, el XML Schema de W3C, RELAX NG de Oasis y Schematron de la Academia Sinica Computing Centre.

Analizadores

Un analizador es un componente de software ubicado entre la aplicación y los archivos XML. Su objetivo es aislar al desarrollador de las complejidades de la sintaxis de XML

Los documentos XML se procesan a través de estas aplicaciones que leen el documento, lo interpretan y generan una salida basada en sus contenidos y en la marca utilizada para su descripción. Un analizador es la herramienta XML más básica, pero también la más importante. Toda aplicación XML está basada en un analizador. Los procesadores hacen posible la presentación y distribución de documentos XML.

Hay analizadores de validación y otros que no lo hacen. Ambos revisan la sintaxis, es decir validan que los documentos estén bien formados, pero sólo los de validación, se aseguran que el documento cumpla la estructura declarada en una DTD y/o esquema, es decir, que sean válidos.

Los analizadores XML pueden categorizarse como parte de las siguientes áreas:

- Modelos de Objetos de Documento (Document Object Model DOM)
- API simple para XML(SAX)
- Espacios de nombre(Namespaces).
- Extensible Stylesheet Language Transformation XSLT.

Arquitectura de aplicaciones XML

La arquitectura de las aplicaciones XML se dividen en dos partes:

- El analizador trata con el archivo XML.
- La aplicación recibe el contenido del archivo por medio del analizador.

El analizador y la aplicación se comunican mediante una **interfaz**. Una interfaz es un dispositivo que permite comunicar dos sistemas que no hablan el mismo lenguaje. Existen dos formas básicas de crearla: por medio de objetos y por medio de interfaces basadas en eventos.

Al utilizar una interfaz basada en objetos, el analizador crea explícitamente en memoria un árbol de objetos que contiene todos los elementos en el documento XML (DOM).

Con una interfaz basada en eventos el analizador no crea explícitamente un árbol de objetos, sino que lee el archivo y genera eventos a medida que encuentra elementos, atributos o texto en el archivo(SAX).

Modelo de Objetos de Documento (DOM)

Un documento XML puede ser visto como un árbol cuyos nodos consisten de las etiquetas que inician y terminan, así como de la información contenida entre ellas; cuando el analizador XML analiza el documento, crea una representación de éste en forma de árbol en la memoria, la cual es referida como el Modelo de Objetos del Documento(DOM por sus siglas en Inglés). Son ideales para las aplicaciones que tratan con documentos (exploradores, editores, etc.).

La W3C creó un conjunto de APIs DOM para acceder y navegar a través de este árbol.

Existen 3 niveles de DOM:

- Nivel 0: Es una normalización de DOM HTML definido implicitamente por Javascript.
- Nivel 1: Permite acceder a todas las partes de un documento XML pero no a la DTD ni a las hojas de estilo.
- Nivel 2: Permite acceso a la DTD, hojas de estilo y espacios de nombres.

DOM define varias interfaces, cada una con sus propios métodos y atributos. Las interfaces básicas son las siguientes:

- **Document**: Proporciona acceso al documento XML cargado en el objeot DOM.
- **Node**: Todo en un documento XML es un nodo.
- **Attr**: Permite manipular los atributos XML de un nodo; en la práctica, se usan métodos de Element para trabajar con los atributos.
- **Element**: Esta interfaz permite manipular los elementos (la mayor parte de nodos de un documento son elementos).
- **CharacterData** y **Text**: Trata el contenido textual de un elemento.

Hay 2 formas de acceder a un nodo en DOM:

- Recorriendo el árbol. Se utilizan los siguientes atributos de node:
  - **parentNode**: accede al nodo padre.
  - **firstChild**: accede al primer hijo.
  - **lastChild**: accede al último hijo.
  - **nextSibling**: accede al "hermano"siguiente del nodo actual.
  - **previousSibling**: accede al "hermano.ªnterior del nodo actual.
  - **childNodes**: es una lista de hijos accesibles mediante un índice.
- Por el nombre del elemento. Se emplea el método **getElementsByTagName(nombre del elemento)**, éste regresa una lista de nodos del mismo tipo y, mediante un índice, se accede al nodo deseado.

En la práctica, se accede al documento mediante una combinación de ambos métodos.

API simple para XML (SAX)

SAX procesa el documento XML de una manera muy diferente a DOM; SAX procesa la información por Eventos. Los eventos en SAX se definen como métodos unidos a interfaces especificas de Java. Una aplicación implementa algunos de estos métodos y se registra como un controlador de eventos con el analizador. Es decir, que si ocurre un evento la aplicación decide qué acción tener como respuesta.

SAX no necesita crear un modelo de objetos y es más rápido aunque requiere:

- La creación de un modelo de objetos por parte del programador.
- Una clase que reciba los eventos SAX y cree objetos en el modelo generado.

El programador debe ser consciente de los eventos que se producen y del orden de los mismos.

SAX agrupa sus eventos en unas cuantas interfaces:

- DocumentHandler define los eventos relacionados con el documento en sí (tales como las etiquetas de cierre y de apertura o cuando aparecen bloques de texto).
- DTDHandler define los eventos relacionados con la DTD.
- EntityResolver define los eventos relacionados con la carga de entidades.
- ErrorHandler define los eventos de errores.

Una ventaja de usar SAX en lugar de DOM es que el árbol del documento no se crea en memoria y por lo tanto resulta mejor la ejecución de algunas operaciones y es posible el manejo de documentos de gran tamaño. Por otro lado, las modificaciones, actualizaciones u otras operaciones estructurales serían más eficientes usando DOM, debido a que con SAX no es posible manipular información una vez procesada.

Comparación SAX y DOM

SAX requiere una mayor programación, pero puede ser muy útil si lo que interesa es rescatar un fragmento de documento o buscar sólo un elemento en particular. No hay un árbol del documento generado en memoria por lo que permite hacer el procesamiento según se "lee" el documento.

En contraste, un DOM es menos versátil, más lento, pero una vez utilizado no hay que desarrollar nada más, ya que con DOM se obtiene el árbol ya construido y listo para acceder a esa información. Los programas que utilizan DOM suelen requerir menos código que los que emplean SAX.

La principal diferencia entre DOM y SAX es que mientras el primero tiene acceso al documento completo en SAX sólo esta disponible el elemento actual.

1.7. Resumen

En el presente capítulo se definieron los antecedentes y características principales de XML como son: etiquetas, elementos, atributos, entidades, comentarios, etc. Además se describieron las características principales de los esquemas. Con este capítulo se pretende tener un panorama general de XML para posteriormente utilizarlos en el desarrollo del proyecto.

Capítulo 2

Lenguajes de Consulta

2.1. Introducción

Con el uso de documentos XML surge la necesidad de consultar esa información y localizar componentes específicos, por lo que se requieren lenguajes de consulta para poder acceder a partes de esos documentos. En el presente capítulo se definirán las características principales de algunos lenguajes de consulta XML como son: XPath, XQuery. Además se describirán las características de lenguajes que permiten localizar y manipular los elementos de los documentos XML: XPointer, XLink y XSL.

2.2. XPath

La necesidad de manejar los documentos XML, y localizar sus componentes, provocó que sólo un año después de sacar la especificación XML, W3C diera a conocer la primera recomendación de XPath.

XPath(XML Path Language) es un lenguaje que permite construir expresiones que recorren y procesan un documento XML. Permite buscar y seleccionar información teniendo en cuenta la estructura jerárquica de XML. Fue creado para su uso en el estándar XSLT, en el que se usa para seleccionar y examinar la estructura del documento que se desea transformar.

XPath es una tecnología derivada de XML cuya referencia oficial está a cargo de la W3C.

La idea principal es la de identificar partes internas de un documento mediante rutas. XPath define cómo acceder a cierto punto de la estructura de un documento. Mediante XPath es posible referenciar no sólo a elementos, si no también textos, atributos y cualquier otra información contenida dentro del documento. XPath esta relacionado con otras tecnologíasXML que lo utilizan. Las cuales veremos más adelante.

XPath trata a todo documento como un árbol de nodos, el cual tiene su punto de partida en un elemento raíz, que se diversifica a lo largo de los elementos que derivan de él. Estos elementos pueden a su vez tener sus propios hijos. Los elementos que no tienen hijos son conocidos como nodos hoja. Hay diferentes tipos de nodos, incluyendo nodos elemento, atributo, texto, comentario e instrucciones de procesamiento.

- **Nodo Raíz**: Se identifica por /. Es importante recalcar que no se debe de confundir con el elemento raíz del documento. De echo, el nodo raíz del árbol contiene al elemento

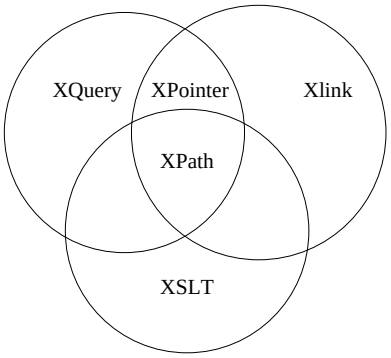

Figura 2.1: Tecnologías asociadas con XPath.

raíz del documento.

- **Nodo Elemento**: Cualquier elemento de un documento XML se convierte en un nodo elemento dentro del árbol. Cada elemento tiene su nodo padre. El nodo padre de cualquier elemento es un elemento, excepto el elemento raíz, cuyo padre es el elemento raíz. Los noso elemento tienen a su vez hijos. Cada nodo elemento tiene propiedades como nombre, atributos e información sobre los espacios de nombre.

- **Nodos Texto**: Por texto se hace referencia a todos los caracteres del documento que no están marcados con alguna etiqueta. Un nodo texto no tiene hijos.

- **Nodos Atributo**: Un caso especial de nodos son los nodos atributo. Un nodo puede tener varios atributos, tantos como desee, y para cada uno se le creará un nodo atributo. Cabe destacar que dichos nodos atributo no se consideran como hijos del nodo elemento, sino como etiquetas agregadas al nodo elemento.

- **Nodos Comentario y de Instrucciones de Procesamiento**: Son nodos especiales que se generan para cada nodo con comentarios e instrucciones de procesamiento propias del lenguaje.

Una "instrucción" en lenguaje XPath es denominada una expresión. El componente básico es la expresión. Dichas expresiones incluyen operaciones sobre distintos tipos de operandos. En particular sobre las llamadas de funciones y rutas de localización, las cuales localizan grupos de nodos especificando sus relaciones con el nodo contexto (Punto de partida) de la expresión.

Una expresión XPath arroja una expresión de 4 tipos posibles al ser evaluada: conjunto de nodos(node-set), booleano, número o cadena.

2.2.1. Rutas de Localización

Se llama rutas de localización (location path) al conjunto de nodos sobre los que se evalua una expresión XPath, que obtendrá como resultado otro conjunto de nodos, formado por aquellos que cumplan los criterios especificados en la expresión. Para dar los nodos del trayecto, se empieza designando un punto concreto, llamado **nodo contexto**, el cual es el nodo a partir del cual se inicia la ruta de localización.

Las rutas de localización pueden ser **absolutas** o **relativas**. Los primeros empiezan con una diagonal(/), la cual determina el nodo contexto, mientras que una ruta relativa, carece de ella al inicio, y el nodo contexto depende del lugar en el que se esté trabajando en el momento de invocar la ruta de localización.

La expresión de una ruta de localización se evalua de izquierda a derecha. Una expresión en XPath no devuelve elementos que cumplen con la ruta especificada, devuelve una referencia a dichos elementos. En todo trayecto de búsqueda de XPath se pueden distinguir tres elementos básicos: Ejes, Nodos y Predicados.

Paso de localización: Es cada paso de una ruta de localización (separados por una /). Consta de:

- **Ejes**:Elementos encargados de especificar en una ruta de localización la relación existente dentro del árbol XPath, entre el nodo de contexto y el paso. En XPath existen 13 tipos de ejes, los más habituales son:
  - **Self**: Identifica el nodo contexto y se identifica mediante un punto (.).
  - **Child**: Describe a los hijos del nodo contexto, y se representa /child:: o mediante /. Es el eje utilizado por defecto.
  - **Descendant**: Selecciona cualquier nodo que sea descendiente del conjunto de nodos contexto. Se representa mediante descendant:: o //.
  - **Attribute**: Contiene los atributos de un determinado nodo y se abrevia usando @ como prefijo del elemento buscado.
  - **Parent**: Indica el nodo padre del nodo contexto y se identifica con dos puntos seguidos(..).
  - **Ancestor**: Identifica a todos los ancestros del nodo contexto, es decir, que devuelve a todos los elementos de los cuale el nodo contexto es descendiente. Se representa mediante ancestor::.
  - **Preceding**: Referencia a todos los nodos del documento que se encuentran antes del nodo contexto.
- **Nodos de Comprobación o búsqueda**: Los nodos de comprobación, también llamados filtros tienen como finalidad restringir -exluir o incluir - nodos dentro de un eje, ya sea por nombre o por tipo:
  - **node()**: Devuelve los nodos de cualquier tipo.
  - **asterisco(*)**: Selecciona los nodos principales de cada ruta (a excepción de los de tipo texto, comentario e instrucciones de procesamiento).
  - **text()**: Devuelve cualquier nodo de tipo texto.
  - **comment()**: Devuelve todos los nodos de tipo comentario.
  - **procesing-instruction()**: Devuelve todos los nodos de tipo instrucción de procesamiento.

- Predicados: La finalidad de un predicado es la de restringir un conjunto de nodos seleccionados mediante un eje a sólo aquellos que cumplen cierta condición. La condición es justamente el predicado. Es decir, los predicados son usados para encontrar un nodo específico o que contenga un valor específico.

  Dentro de los predicados es posible incluir operadores para refinar la expresión. Existen varias clases de operadores entre ellos: los aritméticos, los lógicos, lo de combinación y los de comparación.

2.2.2. Funciones

XPath ofrece una serie de funciones para incorporar a las expresiones. Dichas funciones se clasifican por el tipo de datos sobre al que se aplican. A continuación se mencionarán algunas.

Sobre un conjunto de nodos

- **id(valor)**: Selecciona elementos por medio de su identificador (id).
- **last()**: Devuelve la posición del último nodo de la lista procesada.
- **name()**: Devuelve el nombre de un nodo específico.
- **namespace-uri()**: Devuelve el espacio de nombres de la URI de un nodo específico.
- **position()**: Devuelve la posición en la lista de nodos, del nodo que se está procesando en cada momento.
- **count()**: Devuelve el número de elementos seleccionados.

Sobre cadenas

- **concat()**: Devuelve la concatenación de todos los argumentos sobre los que actua.
- **contains()**: Devuelve verdadero si la primera cadena contiene a la segunda; falso en caso contrario.
- **starts-with()**: Devuelve verdadero si la primera cadena comienza con la segunda; falso en caso contrario.
- **string()**: Convierte el valor del argumento en una cadena.
- **substring()**: Devuelve una subcadena.
- **translate()**: Reemplaza una cadena por otra.

Sobre números

- **ceiling()**: Devuelve el entero más pequeño y no menor al argumento.
- **floor()**: Devuelve el entero más grande pero no mayor que el argumento.
- **number()**: Devuelve el número correspondiene a su argumento.

- **round()**: Devuelve el entero más próximo al argumento.
- **sum()**: Devuelve la suma resultante de sumar el conjunto de nodos que recibe como argumentos.

Sobre booleanos

- **boolean()**: Convierte su argumento a boolean.
- **lang()**: Devuelve verdadero si el atributo xml:lang del nodo contexto es el mismo que el de la cadena del argumento.
- **not()**: Devuelve verdadero si el argumento condición es falso y falso si el argumento condición es falso.

2.3. XPointer

XPointer o lenguaje de Apuntadores XML es un estándar del World Wide Web Consortium(W3C) que proporciona una forma de identificar de forma única fragmentos de un documento XML con el objeto de generar vínculos. La especificación XPointer proporciona un mecanismo para direccionamiento de documentos, en función de su estructura interna, lo que permite examinar su estructura jerárquica al mismo tiempo que permite identificar partes de documentos XML basándose en factores, como tipos de elementos, valores de atributos, contenido de carácter y posición relativa.

XPointer es una extensión de XPath que permite cargar en un visualizador de documentos XML sólo aquellos elementos de un documento que son de nuestro interes. La parte de un documento XML que se direcciona con un XPointer se llama subrecurso. El subrecurso que direcciona puede ser un elemento, un grupo de elementos o un nodo.

XPointer permite añadir a una dirección del tipo:
http://www.xml.com/documento.xml

lo siguiente:
#xpointer(expresión)

Donde expresión es una expresión XPATH, con algunas propiedades extra que no posee XPATH. Una expresión XPointer se añade a un URI(`Uniform Resource Identifier`), como puede ser un URL(`Uniform Resource Locator`) o con un URN (`Uniform Resource Name`).

La extensión XPointer permite a XPath:

- Seleccionar puntos, intervalos(rangos) y nodos.
- Utilizar coincidencias de cadenas para buscar información.
- Utilizar expresiones de direccionamiento en referencias URI como identificadores de fragmentos.

Puntos: Se define como toda posición inmediatamente anterior o posterior a un carácter o aun nodo. Es decir, como la separación que hay entre nodos. Si uno de estos nodos es de tipo texto, se agrega un punto precediendo a cada carácter dentro del contenido del nodo texto. Ejemplo puntos dentro del elemento novela:

```
<novela categoria="A"> 0
   1<titulo>Los ejércitos de la noche</titulo> 2
   3<autor>Norman Mailer</autor> 4
</novela> 5
```

En el ejemplo hay 6 puntos.

Rango: Se llama rango a la distancia entre un punto y otro.

2.4. XLink

XLink o Lenguaje de vínculos XML es una recomendación del World Wide Web Consortium(W3C) que permite crear elementos de XML que describen relaciones cruzadas entre documentos, imágenes y archivos de Internet u otras redes. De tal manera que XLink permite:

- Crear una relación de vínculos entre varios documentos.
- Agregar un vínculo información acerca del mismo (metadatos).
- Crear y describir vínculos a documentos en multitud de ubicaciones.

XLink permite establecer una relación entre dos o más recursos en la Web, sin que necesariamente estos recursos sepan que estan vinculados.

En XML se entiende por **enlace** a toda relación entre uno o más recursos (o porciones de ellos).

XLINK es el lenguaje, definido en términos de marcas que permite introducir enlaces a archivos XML de modo que se puedan relacionar unos con otros. Un elemento será un enlace sólo por el hecho de agregarle una serie de atributos que existen dentro del espacio de nombres xlink:, definido por la norma
"http://www.w3.org/XML/XLink/1.0"

Hay 2 tipos de enlaces:

- Enlaces Simples
- Enlaces Extendidos

Los enlaces simples son similares a los enlaces HTML, es decir, un enlace desde un recurso local a uno remoto.

Ejemplo:

```
<xlink:simple xmlns:xlink="http://www.w3.org/XML/XLink/1.0"
              href="http://www.pineapplesoft.com/publicacion">
    ...
</xlink:simple>
```

Los enlaces extendidos permiten enlazar muchos recursos entre sí. Dichos recursos se específican con el elemento Locator. Además, tienen la capacidad de establecer vinculos que no residan en el mismo documento.

```
<xlink:extended xmlns:xlink="http://www.w3.org/XML/XLink/1.0"
              title="Recursos Web">
  <xlink:locator  href="http://www.mcp.com" title="Macmillan">
  <xlink:locator  href="http://www.xml.com" title="XML.com">
</xlink:extended>
```

Las ventaja que tiene el uso de enlaces complejos respecto a los simples reside en que manejan un conjunto de propiedades (`arcos`) que definen con mayor precisión las relaciones existentes entre los distintos componentes de los enlaces. Se llama **arco** a la descripción del enlace, lo que incluye:

- Recurso de inicio
- Recurso final.
- Sentido entre los recursos.
- Información adicional sobre el comportamiento que se produce al activar el enlace.

Al proceso de activar una relación en un enlace se le conoce como **atravesar** el enlace.

2.5. Transformación vía XSL

XSL (Extensible Stylesheet Language, lenguaje extensible de hojas de estilo) es un conjunto de lenguajes basadas en el estándard XML que permite describir cómo la información contenida en un documento XML debe ser transformada o formateada para su presentación.

XSL ha surgido como una solución para transformar documentos XML en otro formato como HTML, texto simple, PDF e inclusive en otro documento XML con diferentes parámetros.

XSL, la recomendación del W3C, está organizado en tres partes [63]:

- XSLT(eXtensible Stylesheet Language Transformations, lenguaje de hojas extensibles de transformación). Se refiere a la transformación del documento XML.
- XPATH(XML Path Language). Define como acceder a ciertos puntos de la estructura del documento XML, como ya se describió anteriormente.
- XSL-FO(lenguaje de hojas extensibles de formateo de objetos). Define el formato que deben tomar objetos dentro del documento en XML. Es usado principalmente para generar documentos PDF.

2.5.1. Conceptos de XSLT

XSLT se convierte en un estándar hacia 1999 por intermedio de la W3C y su propósito inicial fue permitir a los desarrolladores tomar datos almacenados en documentos XML y presentarlos de diversas maneras.

XSLT es un lenguaje para identificar la transformación de documentos XML. Toma un documento XML y lo transforma en otro. Se suele hacer referencia a los documentos XSLT como hojas de estilo XSL o simplemente hojas de estilo.

XSLT tiene una estrecha relación con XPath, ya que mediante este es posible seleccionar qué elementos y atributos se van a incluir en el procesamiento.La transformación de un documento XML a cualquier otro formato se realiza mediante los llamados **procesadores XSLT**. Las transformaciones se aplican no sobre el archivo XML sino sobre los datos contenidos en él.

En XML se llama **procesador XSLT** a un programa que a partir de un documento XML y de un documento hoja de estilo, lee y genera un documento resultado, siguiendo las reglas expresadas en el contenido del documento hoja de estilo. Algunos ejemplos de procesadores son:

- saxon, saxon.sourceforge.net
- xalan, xalan.apache.org/index.html
- xsltproc, xmlsoft.org/XSLT/xsltproc2.html

2.5.2. Hojas de Estilo (XSL)

XSL se deriva del DSSSL (Document Style Semantics and Specification Language o Lenguaje de Especificación y Semántica de Estilo de Documentos), el mecanismo de hojas de estilo de SGML.

Una hoja de estilo es un conjunto de normas que especifican como dar formato a ciertos elementos del documento. Las hojas de estilo están separadas de los documentos. Por ello un documento puede tener mas de una hoja de estilo y, por su parte, una hoja de estilo puede ser compartida entre varios documentos.

2.5.3. El elemento stylesheet

El elemento principal es stylesheet. Contiene el espacio de nombre XSLT para identificar la hoja de estilo. El espacio de nombre XSLT está en:
http://www.w3.org/1999/XSL/Transform

Se declara de la siguiente manera:

```
<xsl:stylesheet version="1.0" xmlns:xsl=
              "http://www.w3.org/1999/XSL/Transform">
  .
  .
</xsl:stylesheet>
```

2.5.4. Estructura

En un documento XSLT existen básicamente dos categorías de elementos: los llamados de `alto nivel` encargados de ejecutar una tarea específica y las `instrucciones de plantilla`, las cuales definen las condiciones para que se lleve a cabo cada tranformación.

Ejemplos de elementos de alto nivel:

- xsl:include que permite referenciar plantillas procedentes de una fuente externa.
- xsl:output, el cual proporciona métodos para el documento resultante(salida).
- xsl:strip-space que elimina antes del procesamiento todos los nodos consistentes en espacios en blanco.

Las instrucciones en XSLT se suelen clasificar en cinco categorías:

- **Obtención de patrones**: Son las encargadas de describir las reglas que especifican la transformación a aplicar a los elementos del documento. Las más usuales son:
  - <xsl:template match="nombre"> utilizada para definir las reglas de transformaciones para los nodos que coinciden con la expresión XPath que viene en el atributo match.
  - <xsl:apply-templates select="..."/> encargada de aplicar todos los patrones posibles a los elementos que coinciden con la expresión XPath definida en el atributo select.
- **Manipulación de datos**: El objetivo de estas instrucciones consiste en extraer datos de los nodos del documento para poder procesarlos antes de incluirlos en el documento resultante. Algunas instrucciones de esta categoría son:
  - <xsl:copy>produce una copia del nodo contexto.
  - <xsl:value-of select="..."> extrae el valor del elemento definido mediante el valor del atributo select.
  - <xsl:sort-of ...> se incluye inmediatamente después de un elemento for-each o apply-templates y permite ordenar el conjunto de nodos devuelto por el atributo select de estos elementos.
- **Control de Flujo** :
  - <xsl:foreach select="..."> aplica las reglas a cada uno de los elementos que coincide con la expresión XPath.
  - textless xsl:if test="..."> permite escribir una regla que aplica a un determinado patrón sólo si la expresión se evalua como cierta.
  - <xsl:choose> permite incluir más de una condición dentro de una estructura condicional.
  - textless xsl:when test="..."> permite aplicar las reglas "cuando" se cumpla la condición indicada en el atributo test.

  - <xsl:otherwise>, condición del elemento choose que debe de ejecutarse en caso de no haberse cumplido la condición de la sentencia when.

- **Instrucciones de diseño**: Permiten la creación en el documento resultante de nuevos elementos y atributos, así como intrucciones de procesamiento y comentarios.

- **Definición de tipos**: En XSLT existen una serie de elementos encargados de definir tipos de contenidos tales como variables(xsl:variable), texto normal(xsl:text) o números(xsl:number). La consecuencia más importante es que ello permite a XSLT usar variables para procesar la información. Se referencian mediante el elemento variable. El término "variable" tal vez no sea el más adecuado, dado que, una vez asignado su valor, éste no podrá modificarse.

  Las variables pueden ser:

  - globales: declaradas por fuera de los templates, pueden utilizarse dentro de cualquiera de ellos
  - locales: declaradas dentro de un elemento template, sólo pueden utilizarse dentro del elemento template donde fue declarada.

  Para mostrar el contenido de una variable, se utiliza el elemento copy-of de la siguiente manera:

  <xsl:copy-of select="$nombreVariable">

  donde $nombreVariable es el nombre de la variable cuyo contenido queremos mostrar.

Una hoja de estilo también puede recibir valores de parámetros mediante el elemento param.
<xsl:param name="direccion">
De la misma manera que las variables, los parámetros pueden ser globales o locales. Para llamar a un template y darle un valor a un parámetro se utiliza el elemento with-param.

Elemento message

Permite emitir mensajes a la salida de una transformación. Normalmente se utiliza para reportar errores. Puede contener el atributo terminate, el cual puede tener dos posibles valores: yes (se emite el mensaje y termina la ejecución) o no (se emite el mensaje y se continúa la ejecución).

```
<xsl:message terminate="yes">
   Error: <xsl:value-of select="nombre">: nombre vacio!
</xsl:message>
```

El elemento **fallback** permite especificar acciones a ejecutar en caso de que el procesador XSLT no soporte alguno de los elementos incluidos en la hoja de estilo.

Tipos de formato

El elemento **output** permite definir ciertos aspectos relacionados con el formato de salida. Sus atributos son:

- **method**: formato de salida. sus posibles valores son: xml,text y html.

- **version**: si se utilizó en el atributo method xml o html, este atributo permite definir la versión correspondiente.
- **encoding**: permite definir los caracteres a utilizar en el documento.
- **indent**: sus valores son: yes(identa y ordena los elementos en la salida) y no(ubica los elementos en la salida tal cual están en el documento original).
- **media-type**: define el tipo MIME de la salida, el valor por defecto de este atributo es "text/html".

2.5.5. Elementos template.

Una parte primordial de toda hoja de estilo es el elemento template, el cual define instrucciones de ejecución. Básicamente todo gira alrededor de este tipo de elementos.

Ejemplo de XSL:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version = "1.0" xmlns:xsl=
              "http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes"/>
<xsl:template match="/">
<html>
<body>
  <center>
    <table>
     <tr>
      <td><xsl:apply-templates select="nombre_completo"/></td>
      <td><xsl:apply-templates select="direccion"/></td>
      <td><xsl:apply-templates select="telefono"/></td>
     </tr>
    </table>
  </center>
</body>
</html>
</xsl:template>

<xsl:template match="nombre_completo">
  <xsl:apply-templates select="nombre">  
  <xsl:apply-templates select="apellido_paterno">  
  <xsl:apply-templates select="apellido_materno">
</xsl:template>

<xsl:template match="direccion">
  <xsl:apply-templates select="calle">  
  <xsl:apply-templates select="numero">  
```

```
  <xsl:apply-templates select="colonia">
</xsl:template>

<xsl:template match="telefono">
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="nombre">
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="apellido_paterno">
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="apellido_materno">
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="calle">
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="numero">
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="colonia">
  <xsl:value-of select=".">
</xsl:template>
</xsl:stylesheet>
```

El resultado de esta hoja de estilo es una tabla centrada con los datos personales de una persona. La salida es HTML. <xsl:template match="/"> indica las reglas de transformación que se aplicarán a todo el documento. Cada <xsl:template match="nombreElemento"> indica las reglas de transformaciones que se le aplicarán a los elementos indicados. <xsl:apply-templates select="nombreElemento"> es la encargada de aplicar las reglas indicadas por la instrucción template. Y finalmente, <xsl:value-of select="."> obtiene el valor del elemento indicado en el atributo match de la etiqueta template.

La salida HTML que se genera finalmente es:

```
<html>
<body>
 <center>
  <table>
   <tr>
```

```
    <td>Carlos L&oacute;pez Torres</td>
    <td>Calle Beethoven #14 Colonia Chimalhuacan.</td>
    <td>59826455</td>
   </tr>
  </table>
 </center>
</body>
</html>
```

Y la presentación final que se obtiene es:

```
Carlos López Torres  Calle Xola #14 Colonia Chimalhuacan.  59826455
```

2.5.6. XSL-FO

Un documento XSL-FO(eXtensible Stylesheet Language Formatting Objects) es un documento XML en el que se especifica cómo se van a formatear unos datos para presentarlos en pantalla u otros medios. En el documento XSL-FO aparecen tanto los datos como el formato que se les va a aplicar.

La unidad básica de trabajo en un documento XSL-FO es el"Formatting Object", unidad básica para presentar la información. Estos objetos de formato se refieren a páginas, párrafos, tablas, etc.

Para obtener el documento XSL-FO pueden seguirse dos vías:

- Generarlo directamente a partir de los datos. Ya que el documento contiene las especificaciones de formato y sus datos.
- Transformar un documento XML que contenga los datos a presentar con una hoja de estilos XSLT. De esta manera los datos XML se separan del formato que proporcionará la hoja de transformación XSLT.

Cuando se tiene el documento XSL-FO, puede ser procesado por un programa llamado "procesador de XSL-FO" para obtene el documento final en distintos formatos, el más utilizado es el PDF.

2.6. XQuery

2.6.1. Introducción

Debido al aumento en la cantidad de información almacenada, intercambiada y presentada usando XML, la habilidad para consultar fuentes de datos XML adquiere cada vez mayor importancia. Una de las grandes ventajas de XML es su flexibidad para representación de diferentes tipos de información procedente de diversas fuentes. Para explotar esta flexibilidad, un lenguaje de consulta XML debe proporcionar características para recuperar e interpretar la información de las diversas fuentes.

XQuery 1.0 fue desarrollado por el grupo de trabajo de consulta XML del W3C. Empieza a ser recomendación del W3C a partir del 23 de enero del 2007 [65]. XQuery se deriva de algunas propuestas anteriores: XML-QL,YATL, Lorel, Quilt. XQuery es un lenguaje de consulta diseñado para escribir consultas sobre colecciones de datos expresadas en XML. XQuery es una propuesta basada en XML para realizar consultas a documentos XML y a cualquier repositorio de información que mantenga estructuras similares. Es semánticamente similar a SQL.

Una consulta XQuery tiene como entrada y salida documentos XML. XQuery no sólo opera sobre documentos XML sino que también genera documentos XML a partir de una consulta. Por ello, algunos de los requisitos puestos por W3C a XQuery son:

- `Compatibilidad con XPath`: XQuery toma como base a XPath, lo que significa que comparte las funciones y los operadores disponibles y extiende su funcionalidad agregando nuevas características.
- `Posibilidad de Composición`: la totalidad de las distintas expresiones basadas en XPath sean condicionales u otras, deben poder combinarse entre ellas con la máxima generalidad. Lo cual significa que el resultado de cualquier expresión se debe poder usar como operando de otra.
- `Generalidad`: el lenguaje debe de poder aplicarse tanto en entornos que usen tipos de datos muy estrictos y bien especificados, como en otros, donde los tipos de entrada y salida se generen en ejecución, e incluso con los datos que carezcan de tipo. Es decir, XQuery puede ser utilizado tanto en documentos XML que posean asociada una DTD. un esquema XML, o incluso en aquellos que no posean ninguna validación.
- `Semántica`: no deben existir grandes restricciones semánticas a la hora de proporcionar el resultado de una consulta.
- XQuery debe de ser independiente del protocolo de acceso a la colección de datos. Es decir, una consulta en XQuery debe funcionar igual al consultar un archivo local que al consultar un servidor de bases de datos o que al consultar un archivo XML en algún servidor remoto.

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

2.6.2. Características

XQuery es un lenguaje de consultas diseñado para escribirlas sobre colecciones de datos expresadas en XML. Su principal función es extraer información de un conjunto de datos organizados como un árbol n-ario de etiquetas XML.

Algunas características importantes de XQuery son:

- XQuery es un lenguaje util para datos estructurados y semiestructurados.

- Es independiente del protocolo.
- Es un lenguaje declarativo, es decir, es independiente de la forma en que se realice el recorrido o de donde se encuentren los datos en el documento.
- Tipificado fuertemente.
- Capaz de aceptar colecciones de múltiples documentos.
- Compatibles con otros estándares de W3C (XML Schema, XPath, Namespaces).

2.6.3. Conceptos

El modelo de datos de XQuery se basa en representar los datos XML en forma de nodos y valores que sirven de operandos y resultados de los operadores XQuery. En este modelo se representa uno o más documentos XML con una secuencia definida como una colección de uno o más **ítems**, donde un **ítem** es un nodo o un valor atómico.

XQuery es un lenguaje funcional cuya sintaxis es la de XML parecida a la de XPath. A continuación se definen algunos conceptos básicos del lenguaje:

- Una **literal** es la clase más simple de expresión XQuery, y representa un valor atómico.
- Una **variable** empieza con un signo de dolar en XQuery. Se asocia a un valor en una expresión.
- 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 diferentes 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.

Para identificar los datos de entrada existen dos funciones básicas:

- doc() que devuelve un documento completo identificandolo con una URI. Recibe como argumento la ruta del documento que contiene los datos y devuelve el nodo raíz del mismo.
- collection() que devuelve cualquier secuencia de nodos asociada a la URI.

Se habla de un "error dinámico" si estas funciones no localizan lo especificado en ellas.

Una consulta consta de dos partes:

- Prólogo: consta de una serie de declaraciones que definen el entorno para el procesado del cuerpo. Se usa cuando el cuerpo depende de los espacios de nombres, esquemas o funciones.

- Cuerpo: consta de una expresión cuyo valor proporciona el resultado de la consulta.

Para que XQuery cumpla su objetivo, sus expresiones deben poder localizar nodos empezando por determinar el documento en el que deberá efectuar la búsqueda.

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

En el ejemplo se obtiene el nodo documento libros.xml, con /bib se selecciona el elemento bib y con /libro se obtiene todos los libros dentro del elemento bib. Igual que en XPath, los comodines permiten que las consultas seleccionen sin especificar el nombre concreto, por lo que si se quisiera recuperar todos los elementos que contiene bib se haría de la siguiente manera:

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

Una vez localizados los nodos en un documento, se debe crear el resultado que satisface la consulta, lo cual se hace mediante el uso de los constructores, los cuales construyen nuevos objetos XML. La sintaxis se basa en la evaluación de una expresión que aparece entre llaves.

```
<precio>{$venta * 0.85}</precio>
```

En el ejemplo se evalua la expresión entre llaves y se generá como resultado un nodo con el valor de esta evaluación.

La consulta inicia creando un nodo que empieza siendo vacío y termina con el resultado correspondiente, hasta crear un documento completo.

En XQuery existe una operación básica llamada **extracción de tipo** que permite que tengan sentido expresiones como:

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

siendo apellido un elemento y "Stevens" una cadena, se pueda asumir que pueden ser iguales. El operador = extrae el tipo de valor del elemento.

En XQuery las consultas pueden combinar información de diversas fuentes por lo que ésta debe reestructurarse para que se obtenga el resultado deseado. Para efectuar estas tareas existe una serie de expresiones, llamadas **FLWOR** por las iniciales de FOR, LET, WHERE, ORDER y RETURN.

Se llama **tupla** a cada uno de los valores que toma una variable, resultante de una expresión FLWOR formada por las cláusulas for o/y let, seguidas por where o/y order y obligatoriamente por return, es decir, una tupla es un conjunto o serie de datos.

La clausula **for** asocia valores a una o más variables para ir creando un registro (tupla) en cada iteración que realiza. Ejemplo:

```
for $i in (21,22,23)
return <edad>{$i}</edad>
```

las llaves de apertura y de cierre sirven para "expandir" variables, es decir, antes de regresar un registro se reemplazará cada variable por su valor correspondiente. Por lo que el valor que regresará la consulta anterior será:

```
<edad>21</edad>
<edad>22</edad>
<edad>23</edad>
```

La clausula **let** permite asociar a una variable un valor constante, el resultado de una función, el resultado de una operación o una colección. Ejemplo:

```
for $i in (21,22,23)
let $c:=$i + 10
return <edad>{$c}</edad>
```

El resultado de la consulta es el siguiente:

```
<edad>31</edad>
<edad>32</edad>
<edad>33</edad>
```

ya que por cada valor asignado a la variable i, se le suma diez y se asigna a la variable c.

La clausula **where** filtra los registros permitiendo sólo las que cumplen una condición determinada. Ejemplo:

```
for $i in (10,20,21)
let $c:=$i + 10
where $c > 20
return <edad>{$c}</edad>
```

de tal manera que del conjunto inicial de datos la consulta solo regresa aquellos nodos que cumplan con la condición indicada. En este caso regresa lo siguiente:

```
<edad>31</edad>
<edad>32</edad>
```

descartando el nodo en el que al sumarle diez a la variable i el resultado es menor o igual a 20.

La clausula **order by** ordena los registros de acuerdo a un parámetro indicado. El ordenamiento se realiza antes de devolver el resultado. Ejemplo:

```
for $i in doc("bibliografia.xml")//obra/nombre
order by $i
return $i
```

donde for genera una secuencia de registros con un nodo nombre cada uno, y a continuación los ordena de acuerdo con el valor de los elemento nombre. Por lo que el resultado sería una lista de nodos nombre ordenados:

```
<nombre>Antonio</nombre>
<nombre>Carlos</nombre>
```

Es posible especificar más de un criterio de ordenamiento, separando cada uno por medio de comas(,). También es posible especificar mediante la palabra reservada **descending** un ordenamiento descendente. Ejemplo:

```
for $i in doc("bibliografia.xml")//obra
order by $i/@fecha, $i/nombre descending
return $i/nombre
```

El resultado obtenido de esta manera sería el siguiente:

```
<nombre>Carlos</nombre>
<nombre>Antonio</nombre>
```

La clausula **return** devuelve el resultado de la expresión FLWOR.

Para eliminar duplicados existe la función **distinct-values**. Ejemplo:

```
for $i in distinct-values(doc("bibliografia.xml")//obra)
order by $i/@fecha
return <detalle>{$i}/@fecha</detalle>
```

XQuery permite incluir expresiones condicionales dentro de las consultas. Ejemplo:

```
for $x in doc("bibliografia.xml")//obra
return <titulo>
{$x/nombre}
{
  if($x/@fecha > 1900)
  then <siglo>20</siglo>
  else <siglo>19</siglo>
}
</titulo>
```

La cláusula else puede regresar una secuencia vacia. Además, las estructuras condicionales pueden anidarse. En el ejemplo se obtiene el siglo en el que se escribieron algunas obras a partir del año en que fueron escritas.

```
<titulo>
  <nombre>El Inspector</nombre>
  <siglo>19</siglo>
</titulo>
```

Los operadores en XQuery más utilizados son:
Aritméticos: +,-,*, div, mod todos con su significado convencional.
Comparativos: eq =, ne !=, lt <, le <=, gt >, ge >=.
XQuery proporciona dos operadores de **posición**, es decir, operadores para determinar la posición de dos nodos en un documento. El operador $a <<$b devuelve cierto si $a precede a $b en el orden del documento y viceversa con $a >>$b .

En XQuery existen 3 operadores de secuencia sumamente utiles: **union**, **intersect** y **except**.

El operador **union** toma como argumento dos secuencias y devuelve los nodos de ambas. Se puede utilizar la palabra reservada union o su equivalente |.

El operador **intersect** funciona de manera similar, sólo que devuelve los nodos que coinciden en ambas secuencias.

El operador **except** combina dos secuencias y devuelve todos los nodos que se encuentran en la primera y no en la segunda.

XQuery define una gran cantidad de funciones para incorporar a las expresiones. Algunas de ellas son:

- **min**: devuelve el valor mínimo dentro de una secuencia de nodos.
- **max**: devuelve el valor maximo dentro de una secuencia de nodos.
- **count**: devuelve el número de nodos dentro de una secuencia.
- **sum**: devuelve la suma de los valores contenidos en una secuencia de nodos.
- **avg**: devuelve el valor promedio de los valores contenidos en una secuencia de nodos.
- **empty**: devuelve verdadero si la secuencia evaluada no contiene nodos y falso si contiene alguno.
- **exists**: devuelve verdadero si la secuencia evaluada contiene al menos un nodo y falso si no contiene ninguno.

XQuery permite definir constantes antes de las expresiones para luego poder utilizarlas en ellas. Ejemplo:

```
define variable $c {doc("bibliografia.xml")//obra}
```

medskip
En XQuery los comentarios van encerrados entre dos puntos y paréntesis de la siguiente manera:

```
(: Este es un comentario en XQuery :)
```

2.7. Resumen

En el presente capítulo se describieron las características principales de los lenguajes de consulta XPath y XQuery. Además se muestran las características de lenguajes que permiten localizar y manipular los elementos de los documentos XML: XPointer, XLink y XSL. Se describen los lenguajes de consulta debido a que se utilizarán en el desarrollo del proyecto para obtener la información de los documentos XML y posteriormente desplegar la información en formato HTML.

Capítulo 3

XML y Bases de Datos

3.1. Introducción

Existen diversas formas de almacenamiento de la información. En el presente capítulo se describirán algunas maneras de almacenar documentos XML en diversas bases de datos haciendo énfasis en las bases de datos nativas XML. Además se detallaran las características principales de algunas bases de datos que permiten almacenamiento de documentos XML: Tamino, XIndice, eXist y Oracle XML. Por último, se muestra una comparación de las características de las bases de datos mencionadas.

3.2. Antecedentes

XML tiene su origen en la administración de documentos e intercambio de información (no se concibió originalmente como tecnología de bases de datos). XML proporciona una manera de representación de datos estructurados(BD) o semiestructurados(aplicaciones de negocios) y además proporciona muchas ventajas como formato de datos sobre otros, incluyendo:

- Soporte para internacionalización debido a que utiliza Unicode.
- Es independiente de la plataforma.
- Es extensible, de manera que permite que los desarrolladores agreguen información adicional.

Debido a la popularidad de XML, se incrementó la cantidad de información en documentos XML. Y esto ocasionó que la información fuera difícil de mantener y que además, siguiera siendo consistente. Por lo que se requirió un sistema apropiado de almacenamiento y administración de la información (bases de datos).

XML como formato de base de datos, tiene algunas ventajas:

- Es autodescriptivo, dado que las etiquetas describen los datos.
- Es portable, es decir, se puede trasladar a cualquier plataforma.
- Se pueden describir los datos en estructuras de árbol o de gráficas.

Aunque se presenta la desventaja en el tiempo requerido para el análisis y la conversión a texto, ya que podría ser muy grande.

XML proporciona varias características que se encuentran en las bases de datos. A continuación se mencionan algunas:

- Almacenamiento (Documentos XML).
- Esquemas (DTDs o XML Schemas).
- Lenguajes de Consultas (XQuery, XPath, XQL, XML-QL,QUILT,etc).
- Interfaces (SAX, DOM, JDOM,etc).

Esto funciona bien mientras se trabaje en un ambiente en el cual se cuente con pocos datos y pocos usuarios.

El término "XML Database", base de datos XML, incluye a todos los sistemas de bases de datos, los cuales son capaces de almacenar y recuperar documentos XML. Esto lo puede hacer utilizando bases de datos relacionales convencionales o bases de datos especialmente diseñadas para el almacenamiento de documentos XML.

Ronald Bourret [13] menciona dos tipos de bases de datos XML:

- Bases de datos que permiten XML. Estas bases de datos tienen su propio modelo de datos y mapean las instancias de modelos de datos XML a su modelo. Éste tipo de bases utilizan únicamente a XML para transportar datos.
- Bases de datos XML nativas. Usan el modelo de datos XML directamente. Administran y consultan XML.

Dependiendo de lo que se va a almacenar, datos o documentos, se hace la elección de la base de datos.

3.3. Datos y Documentos

3.3.1. Documentos Centrados en Datos

Los documentos Centrados en Datos son aquellos que usan XML como transporte de datos. Si el documento XML tiene una estructura bien definida y contiene datos que son actualizables, el documento es conocido como **Centrado en Datos**. Algunos ejemplos de estos documentos son: los catálogos de ventas, datos científicos, agendas,etc.

Los documentos Centrados en Datos se caracterizan por su estructura bastante regular, granularidad fina (la unidad de dato más pequeña está al nivel de un elemento PCDATA o un atributo) y un contenido no mixto. No es importante el orden en que aparecen los elementos del mismo nivel, excepto cuando se valida el documento. Ejemplo:

```
<OrdenDeVentas SONumero="12345">
   <Cliente NumeroDeCliente="543">
      <NombreCliente>Industrias ABC</NombreCliente>
      <Calle>123  St. Paula</Calle>
      <Ciudad>Chicago</Ciudad>
      <Estado>IL</Estado>
      <CodigoPostal>60609</CodigoPostal>
    </Cliente>
    <FechaOrden>981215</FechaOrden>
    <Elemento NumeroElemento="1">
       <Parte NumeroParte="123">
         <Descripcion>
            Acero inoxidable,construcción de una pieza,
            garantia de por vida.</p>
         </Descripcion>
         <Precio>9.95</Precio>
       </Parte>
       <Cantidad>10</Cantidad>
    </Elemento>
    <Elemento NumeroElemento="2">
       <Parte NumeroParte="456">
         <Descripcion>
               Aluminio, un año de garantia.</p>
         </Descripcion>
         <Precio>13.27</Precio>
        </Parte>
        <Cantidad>5</Cantidad>
    </Elemento>
</OrdenDeVentas>
```

El ejemplo anterior tiene una estructura bien definida. Los datos generales del cliente y posteriormente información sobre las ordenes de compra de dichos clientes.

Algunos sitios web que construyen páginas HTML sobre demanda en una base de datos, pueden cambiar esas páginas por una serie de documentos XML centrados en datos y una o más hojas de estilo. Ganando con ello la posibilidad de generar con la información varias páginas HTML diferentes de acuerdo a la hoja de estilo o darle diferentes formatos.

3.3.2. Documentos Centrados en Texto

Los Documentos Centrados en Texto son aquellos diseñados para el consumo humano. Ejemplos de ellos son los libros, emails, anuncios, etc. Si el documento XML consta de contenido estático que debería ser actualizado únicamente remplazando el documento completo, es conocido como **Centrado en Texto**. Estos se caracterizan por su poca o irregular estructura,granularidad grande (la unidad más pequeña de datos debería estar al nivel de un elemento de contenido mixto o el documento entero mismo) y por su contenido mixto. Es importante el orden en que aparecen los elementos del mismo nivel.

Ejemplo:

```
<Producto>
   <Nombre>Pavo</Nombre>
   <Desarrollo>Labs de Fabricación.</Desarrollo>
   <Descripcion>
      <Parrafo>Varias versiones
         es hecho en el más <b> fino acero inoxidable</b>
         y <i\textgreater resistente.</i>
      </Parrafo>
      <Lista>
         <Elemento>
            <Link URL="Order.html">Ordena ahora</Link>
         </Elemento>
         <Elemento>
            <Link URL="Catalog.zip">Descarga el catálogo</Link>
         </Elemento>
      </Lista>
   </Descripcion>
</Producto>
```

El ejemplo anterior no tiene una estructura bien definida, dado que, por ejemplo en Parráfo puede tener distintos elementos (<b>, <i>, etc.) y no necesariamente en el mismo orden.

3.3.3. Datos, Documentos y Bases de Datos

El hecho de caracterizar a los documentos como centrados en datos o en texto debería ayudar a decidir la base de datos que se va a utilizar. Como regla general, los datos son almacenados en bases de datos relacionales u orientadas a objetos y los documentos son almacenados en una base de datos XML nativa o en un sistema administrador de contenido (una aplicación diseñada para administrar documentos y construida en una base de datos XML nativa).

Estas reglas no son estrictas, los datos -especialmente los semiestructurados - pueden ser almacenados en una base de datos XML nativa y los documentos pueden ser almacenados en una base de datos tradicional cuando se necesitan poco las características especificas de XML.

3.4. Almacenamiento y Recuperación de Datos

Para transferir datos entre documentos XML y una base de datos, es necesario mapear el esquema del documento XML a el esquema de base de datos. El software de transferencia de datos es entonces construido basandose en este mapeo. El software puede usar un lenguaje de consulta XML (tal como XPath, XQuery, o un lenguage propietario) o simplemente transferir los datos de acuerdo al mapeo.

En último caso, la estructura del documento debería coincidir exactamente con la estructura esperada por el mapeo. Lo cual no sucede muy a menudo, es decir, antes de transferir los

datos a la base de datos, primero se transforma el documento a la estructura esperada por el mapeo; el dato es entonces transferido. Similarmente, después de transferir datos desde la base de datos, el documento resultante es transformado a la estructura necesaria por la aplicación.

3.4.1. Transferencia de datos de un documento XML a una base de datos

Cuando se transfieren datos de un documento XML a una base de datos, es frecuente que se tenga que eliminar información de ese documento, como el nombre de la DTD. Además, es valido ignorar información de la estructura física, tal como la definición de una entidad. Puede darse el caso que se tenga que ignorar estructura lógica, particularmente en las instrucciones de procesamiento y en el orden en el que aparecen los valores de los atributos y los elementos del mismo nivel.

Similarmente, cuando se transfieren datos de la base de datos a un documento XML, es recomendable que el documento XML resultante no contenga CDATA o entidades y que el orden en que los elementos y atributos aparezcan sea el orden en el cual los datos fueron regresados por la base de datos.

Una consecuencia de ignorar información acerca del documento y su estructura física es que al almacenar esos datos en la base de datos y querer reconstruir el documento a partir de ellos, frecuentemente resulta un documento diferente.

Al transferir datos entre un documento XML y una base de datos, se debería mapear una estructura de documento a un esquema de bases de datos y viceversa. La estructura de un documento XML se asocia con una DTD o un esquema XML. La DTD es usada, como ya se menciona anteriormente, para describir los elementos y atributos.

Dos mapeos comúnmente usados para mapear un esquema XML al esquema de bases de datos son:

- mapeo basado en tablas.
- mapeo basado en objeto - relación.

Mapeo Basado en Tablas

El mapeo basado en tablas es usado por muchos de los productos "middleware" que transfieren datos entre un documento XML y una base de datos relacional. Los modelos de los documentos XML tienen un conjunto de tablas. De tal manera, que la estructura de un documento XML debería ser como sigue:

```
<database>
   <table>
      <row>
         <column1>...</column1>
         <column2>...</column2>
         ...
      </row>
```

```
      <row>
         ...
      </row>
 ...
   </table>
   <table>
      ...
   </table>
      ...
</database>
```

Dependiendo del software, es posible especificar cual columna de datos es almacenada como elementos hijos o atributos, así como que nombres va a tener cada elemento o atributo. Éste mapeo es util para serializar datos relacionales, por ejemplo al transferir datos entre dos bases de datos relacionales. La desventaja es que no puede ser utilizada para ningún documento XML que no cumpla el formato establecido.

Mapeo Objeto-Relación

El mapeo objeto-relación modela los datos en el documento XML como un árbol de objetos que son específicos al dato en el documento. En este modelo, los elementos tipo con atributos, con contenido elemental o con contenido mixto son generalmente modelados como clases. Los elementos tipos con contenido unicamente PCDATA, atributos y PCDATA son modelados como propiedades escalares. Este modelo es entonces mapeado a bases de datos relacionales usando técnicas de mapeo objeto-relación tradicional. Las clases son mapeadas a tablas, las propiedades son mapeadas a columnas y las propiedades objeto-valuado son mapeados a la llave primaria o foranea.

Es importante entender que el modelo de objetos usado en este mapeo no es el Modelo de Objetos de Documento (DOM).

3.4.2. Transferencia de datos de acuerdo al modelo

Muchos productos pueden transferir datos directamente de acuerdo al modelo en el que fueron construidos. Dado que la estructura del documento XML es frecuentemente diferente de la estructura de la base de datos, estos productos son usados regularmente con XSLT. Esto permite a los usuarios transformar documentos a la estructura establecida por el modelo antes de transferirlos a la base de datos, y viceversa.

Los lenguajes de consulta más comunes que regresan XML desde una base de datos relacional son los basados en plantillas. En estos lenguajes, no hay un mapeo predefinido entre el documento y la base de datos. En su lugar, declaraciones SELECT son embebidas en una plantilla y los resultados son procesados por el software de transferencia de datos.

Los lenguajes de consulta *basados en plantillas* pueden ser sumamente flexibles. Aunque las características varian de producto en producto. Algunas características comunes son:

- La capacidad de colocar conjuntos de valores resultantes en cualquier parte en la salida del documento.

- Definición de variables y funciones.
- Parametrización de declaraciones SELECT a través de parámetros HTTP.

Estos lenguajes de consulta son utilizados con mayor frecuencia para transferir datos de una base de datos relacional a documentos XML. Para transferir datos de documentos XML a bases de datos relacionales se usa frecuentemente el mapeo basado en tablas mencionado anteriormente.

3.4.3. Almacenamiento y recuperación de documentos

Hay tres estrategias básicas para almacenar documentos XML: almacenarlos en el sistema de archivos o como un BLOB en una base de datos relacional y aceptar la funcionalidad XML limitada, o almacenarlos en una base de datos nativa XML.

Almacenar la información en un sistema de archivos, permite consultar la información con algunas herramientas, pero las búsquedas de texto completo son ineficientes debido a que tales herramientas no pueden distinguir facilmente las marcas del texto y ni el uso de las entidades.

Una opción un poco más sofisticada es almacenar documentos como BLOBs (Binary Large Objects) en una base de datos relacional. Esto proporciona un gran número de ventajas de las bases de datos: control transaccional, seguridad, acceso multiusuario, etc. Además, muchas bases de datos relacionales tienen herramientas para busquedas de textos que pueden hacer búsquedas de texto completo.

3.5. Tipos de Bases de datos

3.5.1. Bases de datos que habilitan XML

Las bases de datos que habilitan XML son bases de datos relacionales tradicionales a las cuales se les ha agregado algún soporte para contenido XML. La principal característica es que tienen un mapeo objeto-relación entre contenido XML y las tablas en el SMBDR (Sistema Administrador de Bases de Datos Relacional). El contenido es fragmentado y almacenado en las tablas del SMBDR.

Por lo tanto, los documentos XML son modelados como un árbol de objetos, los cuales son los datos en el documento. El modelo es entonces mapeado a bases de datos relacionales usando tecnicas de mapeo objeto-relación. Lo que significa que las clases son mapeadas a tablas, las propiedades escalares son mapeadas a columnas y las propiedades objeto-valor son mapeadas a llaves primarias o llaves foraneas.

Esta técnica puede manejar documentos XML centrados en datos y centrados en documentos, aunque no soporta el contenido mixto. La fidelidad del DOM se pierde durante la fragmentación, también para cierta información, este tipo de mapeo es dificil de lograr.

Este tipo de bases de datos XML usa un mapeo basado en tablas para almacenar documentos en una base de datos Relacional. Modela los documentos XML como una tabla o como un conjunto de tablas. El modelo de la base se deriva directamente del esquema del documento, los elementos complejos son mapeados a tablas y los elementos simples y atributos son mapeados a columnas. Dado que hay una relación directa entre las tablas y el documento XML, este no puede ser usado por documentos XML que no coincidan con el formato.

El mapeo basado en tablas es más conveniente para manejar documentos centrados en documentos, cuando transfieren datos entre dos bases de datos relacionales.

Algunas ventajas de usar bases de datos relacionales son:

- Los sistemas de BD son maduros y usados ampliamente.
- Posibilidad de utilización desde aplicaciones existentes.
- La conversión es sencilla si los datos se generan a partir de un esquema relacional, y si se usa XML como formato de intercambio de datos.

Uno de sus inconvenientes es que si los datos no se generan a partir de un esquema relacional, la conversión no es sencilla, especialmente si hay elementos anidados o elementos que se repiten(atributos multivaluados).

Existen diferentes alternativas para solucionar este problema:

- Almacenar el XML como cadena de caracteres.
- Representarlo en forma de árbol.
- Asignación a relaciones.

Almacenamiento como cadena de caracteres:

Existen dos opciones:

- Almacenar cada elemento hijo del elemento de nivel superior como un campo de tipo cadena en un registro en la BD. Utilizar una única relación (ELEMENTOS) con un atributo (datos) para almacenar todos los elementos (por ejemplo, cada registro almacena un elemento XML en forma de cadena: cuenta, cliente o impostor).
- Almacenar cada elemento de nivel superior como una relación separada (por ejemplo, tres relaciones: cuenta, cliente, impostor).

El beneficio que tiene hacerlo así es que, es posible almacenar cualquier dato, incluso sin DTD. Sin embargo, tiene varios inconvenientes:

- El SGBD no conoce el esquema de los datos almacenados, por lo que no es posible consultar los datos directamente.
- Se necesita analizar las cadenas para acceder a los valores dentro de los elementos.
- El análisis de la información es lento.

Representación en árbol:

Los datos XML se modelan como un árbol y se almacenan en relaciones:

nodos (id, tipo, etiqueta, valor) hijo(id_hijo, id_padre)

A cada elemento y atributo se le proporciona un identificador único. Además, se inserta un registro en la relación nodos para cada elemento y atributo, con los campos:

- id: identificador del elemento o atributo.
- tipo: específica si se trata de un atributo o elemento.
- etiqueta: específica el nombre del elemento o atributo.
- valor: es el valor literal del elemento o atributo.

La relación hijo almacena el elemento padre de cada elemento y atriuto. Se puede añadir la posición como atributo, que guarda el orden de los hijos.

Es posible almacenar cualquier dato, incluso sin DTD, además de que se pueden traducir las consultas a consultas relacionales. Sin embargo, los datos se dividen en demasiadas piezas y las consultas simples requieren de una gran cantidad de combinaciones(joins). Ejemplo: Para el siguiente XML:

```
<revista>
  <codigo>0220357</codigo>
  <nombre>Revista Dinero</nombre>
  <titulo>La promesa de las instituciones globales</titulo>
  <precio>5200</precio>
  <numero>12</numero>
</revista>
```

Se generan 6 tablas, la tabla revista la cual esta relacionada con las tablas codigo, nombre, título, precio y numero. Cada una con un registro.

Asignación a relaciones:

Los elementos XML cuyo esquema es conocido, se asignan a relaciones y atributos. Los elementos cuyo esquema es desconocido se almacenan como cadenas o árboles. Además, se crea una relación para cada tipo de elemento cuyo esquema es conocido:

- Un atributo id para almacenar una id única para cada elemento.
- Todos los atributos de elemento se convierten en atributos de relación.
- Todos los subelementos que se producen una sola vez se convierten en atributos:
  - Si el valor del subelemento es texto, el atributo almacena el texto como valor.
  - Para subelementos complejos, se almacena el id del subelemento.
- Si el subelemento es multivaluado, se representa en una tabla separada.
- En caso de existir ciclos entre los elementos se puede necesitar duplicar información.

El almacenamiento es eficiente y se pueden transmitir consultas XML dentro de SQL, ejecutarlas eficientemente y después trasladar los resultados de SQL a XML. Sin embargo, se necesita conocer la DTD o esquema y continúan presentes las sobrecargas por transformación.

3.5.2. Bases de Datos Nativa XML

Almacenar datos en una base de datos relacional puede ocasionar que se tengan muchas columnas con valores nulos o un gran número de tablas. Aunque los datos semiestructurados pueden almacenarse en bases de datos jerárquicas y orientadas a objetos, se puede elegir almacenar datos en una base de datos XML en forma de documentos XML. Además, la recuperación de los datos es más rápida.

Un problema de trabajar con este tipo de bases de datos es que sólo pueden regresar los datos como XML. Y si se requiere otro tipo de formato se debería transformar el dato antes de poder usarlo.

Una base de datos nativa XML es aquella diseñada especialmente para almacenar documentos XML. Define un modelo(lógico) para un documento XML, almacena y recupera documentos de acuerdo al modelo. El término "nativa" se utiliza por los vendedores de los productos comerciales especializados en soluciones de BD XML.

Tiene características de otras bases de datos: soporta transacciones, seguridad, acceso multi-usuario, programación de APIs, lenguajes de consulta,etc. La única diferencia es que el modelo de su estructura interna está basado en XML y no en otro modelo (por ejemplo, en el modelo relacional).

Las bases de datos Nativas XML pueden manejar ambas categorías de documentos (centradas en datos y centradas en texto). Aunque, son más utiles para almacenar documentos centrados en texto. Esto es porque las bases de datos preservan el orden de los documentos, el procesamiento de instrucciones, comentarios, secciones CDATA, entidades del usuario, mientras que las bases de datos capacitadas para XML no.

Las bases de datos nativas XML soportan lenguajes de consulta XML, lo cual permite hacer consultas del estilo "Dame todos los documentos en los cuales el tercer párrafo después de la sección de inicio contenga una palabra en negrita". Tales consultas son dificiles de hacer en un lenguaje como SQL.

Son útiles para almacenar documentos cuyo "formato natural" es XML, sin considerar lo que ese documento contiene. Por otro lado, son mejor usadas cuando los datos son semiestructurados, e incrementan la rapidez de la recuperación de esos datos. También, se pueden almacenar documentos que no tienen DTDs.

Dependiendo de la arquitectura que ellas usen, las bases de datos Nativas XML pueden dividirse en bases de datos **basadas en texto** y **basadas en el modelo**. Las bases de datos basadas en texto almacenan el documento XML como texto, almacenando así una copia idéntica del dato. Aplican técnicas de compresión para reducir el espacio de almacenamiento. Mantienen índices para aumentar la eficiencia en el acceso de la información. Lo cual da a este tipo de bases de datos una gran ventaja al recuperar los documentos enteros o fragmentos. Aunque también puede encontrar problemas de rendimiento(performance), al recuperar datos que son diferentes de la jerarquia definida.

Las bases de datos basadas en el modelo construyen un modelo de objetos interno del documento XML y lo almacenan. Cómo el modelo se almacena depende de la base de datos. Algunas bases de datos lo almacenan en bases de datos relacionales, otras en bases de datos orientados a objetos o ellas almacenan el modelo como un DOM persistente.

Todas las bases de datos basadas en texto son indexadas, lo cual permite a la ejecución de las consultas ir a un punto en un documento XML. Esto da rápidez al recuperar documentos enteros o fragmentos de él. En contraste, reensamblar un documento desde piezas, como es hecho en bases de datos relacionales y algunas bases de datos nativas XML, requiere de múltiples índices y múltiples lecturas de disco.

Hay varias razones para utilizar este tipo de bases:

- Cuando los datos que se tienen son semiestructurados.
- La rapidez de la recuperación de datos. La razón es que algunas estrategias de almacenamiento usadas por las bases de datos nativas XML almacenan documentos enteros juntos físicamente o usan apuntadores físicos entre las partes del documento. Esto permite a los documentos ser recuperados sin "joins" o con "joins" físicos, que son más rápidos que los "joins" lógicos usados por las bases de datos relacionales.
- Explotación de las capacidades específicas de XML, tales como ejecución de consultas XML.

En el siguiente ejemplo, en una base de datos relacional, debería almacenarse en 4 tablas -orden de ventas, elementos, partes y clientes. Sin embargo, en una base de datos nativa XML, este documento debería ser almacenado en un solo lugar en disco, asi recuperar el documento o una parte de él, requiere un indice y una sola operación de lectura. Una base de datos relacional requiere 4 indices y al menos 4 operaciones de lectura. Ejemplo de XML:

```
<OrdenDeVentas SONumero="12345">
   <Cliente NumeroDeCliente="543">
      <NombreCliente>Industrias ABC</NombreCliente>
      <Calle>123  St. Paula</Calle>
      <Ciudad>Chicago</Ciudad>
      <Estado>IL</Estado>
      <CodigoPostal>60609</CodigoPostal>
   </Cliente>
    <FechaOrden>981215</FechaOrden>
    <Elemento NumeroElemento="1">
       <Parte NumeroParte="123">
          <Descripcion>
             Acero inoxidable,construcción de una pieza,
             garantia de por vida.</p>
          </Descripcion>
          <Precio>9.95</Precio>
       </Parte>
       <Cantidad>10</Cantidad>
    </Elemento>
    <Elemento NumeroElemento="2">
       <Parte NumeroParte="456">
          <Descripcion>
             Aluminio, un año de garantia.</p>
```

```
        </Descripcion>
        <Precio>13.27</Precio>
      </Parte>
      <Cantidad>5</Cantidad>
    </Elemento>
</OrdenDeVentas>
```

CARACTERÍSTICAS

- Muchas bases nativas XML soportan la noción de una colección. Este juega el rol similar a una tabla en una base de datos relacional o un directorio en un sistema de archivos.
- Casi todas las bases de datos nativas XML soportan uno o más lenguajes de consulta. El más popular de ellos es el XPath (Con extensiones para consultas sobre multiples documentos) y XQuery, aunque numerosos lenguajes de consulta propietarios son soportados también.
- Las bases de datos nativas XML tienen una variedad de estrategias para actualizar y eliminar documentos, desde reemplazar o borrar los documentos ya existentes hasta modificaciones a través de un árbol DOM a lenguajes que especifican como modificar fragmentos de un documento.
- Virtualmente todas las bases de datos nativas XML soportan transacciones. Sin embargo, la cerradura es frecuentemente utilizada a nivel de documentos enteros de tal manera que la concurrencia multiusuario puede ser relativamente baja.
- Casi todas las bases de datos XML nativas ofrecen interfaces de programación de aplicaciones(APIs). Estas son usualmente en la forma de una interfaz tipo ODBC, con métodos para conectar a la base de datos, exploración de metadatos, ejecución de consultas y recuperación de resultados. Los resultados se regresan usualmente como una cadena XML, un árbol DOM o como una transformación del documento que fue devuelto.
- Algunas bases de datos nativas XML pueden incluir datos remotos en los documentos almacenados. Usualmente, este dato es recuperado de una base de datos relacional con ODBC, OLE DB, o JDBC y modelados usando el mapeo basado en tablas o un mapeo objeto-relación.
- Casi todas las bases de datos nativas XML soportan el indexado de elementos y valores de atributos. Como en las bases de datos que no manejan XML, los indices son usados para hacer más rápidas las consultas.
- Normalización se refiere al proceso de diseñar un esquema de bases de datos en el cual una parte de datos es representada una sola vez. La cual tiene varias ventajas, tales como reducción de espacio en disco y elimina la posibilidad de datos inconsistentes, lo cual puede ocurrir con secciones de datos almacenados en más de un lugar.

  Al igual que las bases de datos relacionales, hay bases de datos nativas XML que se forzan a normalizar los datos. Normalizar datos para una base de datos de este tipo es en gran

parte lo mismo que normalizar esos datos para una base de datos relacional: es decir, se requiere diseñar los documentos de forma que no se repitan los datos. Una diferencia entre una y otra base es que XML soporta propiedades con valores multivaluados, mientras que las bases de datos relacionales no. Esto hace posible normalizar datos en una base de datos nativa XML en una forma que no es posible hacer en una base de datos relacional.

Por ejemplo, un registro de ventas. El cual consiste de la información principal, tal como el número de venta, fecha, número de cliente, y una o más lista de artículos, las cuales contienen un número, una cantidad y un precio total. En una base de datos relacional, la información principal debería ser almacenada en una tabla separada de la lista de artículos, puesto que hay multiples lista de artículos por cliente. En una base de datos nativa XML, esta información puede almacenarse en un documento sin redundancia, como la jerarquía natural de XML permite un elemento padre para multiples elementos hijos.

- La integridad referencial se refiere a la validez de señalar a datos relacionados y es una parte necesaria de mantener un base de datos en estado consistente.

  La integridad referencial significa asegurar que XLinks u otros mecanismos de ligado de propiedades apunten a un documento o a fragmentos de documentos validos.

- Un DOM Persistente o PDOM es un tipo especializado de base de datos XML el cual implementa el DOM sobre algún ordenamiento de almacenamiento persistente. El árbol DOM que regresa un PDOM tiene vida, es decir, los cambios hechos a un árbol DOM son reflejados directamente en la base de datos. El árbol que se regresa a una aplicación es una copia, y los cambios son hechos a la base de datos a través de un lenguaje de actualización XML o remplazando el documento entero.

  Porque los árboles PDOM tienen vida, la base de datos es usualmente local. Es decir, la base de datos debería estar al menos en la misma máquina que la aplicación, aunque no se requiere necesariamente que sea así.

- Los Sistemas Administradores de Contenido son otro tipo de bases de datos nativa XML. Ellas son diseñadas para administrar documentos escritos por personas, tales como manuales y artículos, y son construidos como bases de datos nativas XML. La base de datos es generalmente ocultada del usuario. El término Sistema Administrador de Contenido, como opuesto a Sistema Administrador de documentos,refleja el hecho que tales sistemas generalmente permiten romper el documento en fragmentos de contenido discretos, tales como capítulos, etc.

3.6. Ejemplo de Bases de Datos Nativas XML

3.6.1. Tamino-Software AG

En 1999, Software AG lanzó la primera versión de su servidor Tamino nativo XML, el cual incluía una base de datos XML nativa. Tamino maneja de manera uniforme documentos centrados en datos y documentos centrados en texto y es diseñado para procesar documentos

XML sin importar la estructura. Además Tamino puede almacenar otros tipos de datos como son imágenes, archivos HTML, etc. que son relevantes en algún contexto Web.

El servidor Tamino XML tiene un sistema de bases de datos completo construido, proporcionando transaccionalidad, seguridad, acceso multiusuario, escalabilidad, etc.Además, Tamino está adaptado para satisfacer las necesidades de XML: soporta estándares XML y está optimizado para procesamiento XML.

El servidor Tamino es una plataforma para la administración de datos y basado en XML y otros estándares de tecnologías de Internet.

Arquitectura

El acceso primario al servidor Tamino XML es vía HTTP. Esto hace a Internet y a alguna intranet los primeros clientes de Tamino. Tamino proporciona un componente llamado X-Port, el cual garantiza la comunicación eficiente entre un servidor Web y el servidor Tamino.

El servidor Tamino XML soporta los métodos HTTP GET,PUT,DELETE,HEAD para leer, almacenar, remplazar o eliminar documentos y para obtener información sobre los documentos almacenados en Tamino. Tamino consta de una herramienta de administración (Tamino Manager), la cual proporciona una interfaz gráfica. Desde una instancia de la herramienta se pueden administrar todos los servidores Tamino en alguna red local.

Tamino X-Node proporciona acceso a bases de datos relacionales externas. Los datos que residen en estos sistemas pueden ser incluidos en los documentos recuperados por Tamino XML Server. Además, información incluida en documentos almacenados en Tamino puede ser propagada a sistemas externos.

Tamino también proporciona una característica llamada Tamino X-Tension, lo que se deseé almacenar puede ser pasado a través de mapeo de funciones. Dichas funciones toman la responsabilidad de almacenar los datos, ya sea en un sistema externo o en el mismo lugar.

Características

Tamino XML Server soporta la descripción esquemática de documentos vía W3C XML Schema.

Colecciones y Tipos de Datos

Una base de datos Tamino consiste de múltiples "colecciones". Estas colecciones son contenedores para agrupar documentos. Cada documento almacenado en los datos de Tamino reside en exactamente una colección. Una colección tiene un conjunto de descripciones de esquemas XML asociadas. En cada descripción de esquema los tipos de datos pueden ser definidos usando una notación específica de Tamino en el área de extensibilidad de W3C XML Schema (elemento appinfo). Un tipo de dato (doctype) identifica uno de los elementos globales declarados en el esquema XML como el elemento raíz. Dentro de una colección, cada documento es almacenado como un miembro de exactamente un tipo de dato.

El elemento raíz del documento identifica al tipo de dato. Tamino XML Server puede además almacenar información de objetos no XML (como son: imagenes, archivos de sonido, etc.). Estos son organizados en un tipo de datos dedicado llamado `nonXML`. Cuando estos objetos son leidos por Tamino, le pone el correspondiente tipo MIME.

Tamino asigna un identificador a cada documento u objeto no XML. Además, el usuario puede especificar un nombre. Este nombre debe ser único con un tipo de dato y puede ser usado para direccionar directamente al documento vía un URL.

Esquemas

Si un esquema es modificado por un tipo de dato de documentos almacenados en Tamino XML Server, Tamino garantiza la validez de estos documentos con respecto al nuevo esquema.

Acceso a Otras Bases de Datos

Datos almacenados en bases de datos relacionales pueden ser integrados dentro de documentos almacenados en Tamino vía el componente X-Node. Para el usuario, el hecho de que parte de los datos residen en otra fuente de datos es transparente. Estos datos se comportan como partes regulares de un documento. Ellos son declarados como parte de un documento en un esquema Tamino.

La correspondencia entre datos almacenados en Tamino y datos almacenados en otro sistema de bases de datos deben ser explicitamente descritos.

Mapeo de Datos a Funciones

Para accesar a datos almacenados en otras bases de datos, la correspondencia entre partes de un documento XML y datos en la base de datos pueden ser fácilmente descritos en una forma declarativa. Para otras fuentes de datos, un mapeo procedural es necesario. Para este propósito, el componente X-Tension de Tamino soporta mapeo de funciones. Estas funciones pueden ser escritas en Java o como objetos COM (en plataforma Windows). El administrador puede especificar si ellas se ejecutarán en el mismo espacio de direcciones como el Tamino Server (más rápido) o en un espacio de direcciones separada (más seguro). Las funciones X-Tension son cargadas dinámicamente cuando son referenciadas. Además pueden ser agregadas en línea al Tamino Server sin interrupción a operaciones normales.

Las funciones de entrada aceptan documentos completos o partes de documentos como parámetros. Las funciones son responsables de almacenar los documentos recibidos.

Para una función de X-Tension, la información sobre eventos transaccionales (commits, rollback) es usualmente importante. Si la función es usada para almacenar datos en un sistema externo transaccional, la correspondiente transacción del sistema externo tiene que hacer el rollback si la transacción de Tamino hace rollback.

Internacionalización

Tamino trabaja internamente sólo con Unicode. Sin embargo, no todos los sistemas que interactuan con Tamino están basados en Unicode. Por lo tanto, Tamino hace algunas conversiones. Cuando documentos con una codificación diferente son enviados al Tamino XML Server, Tamino los convierte a Unicode antes de procesarlos y resuelve referencias de caracteres para remplazarlos con el correspondiente carácter Unicode. Analogamente los usuarios pueden especificar una codificación cuando recuperen los datos. En este caso, Tamino convierte los resultados de las consultas a la codificación deseada antes de enviarlos al usuario.

Indexado

Los índices son indispensables en los sistemas de bases de datos porque de otra manera grandes cantidades de datos podrian no ser consultados de una manera satisfactoria. Tamino XML Server soporta tres tipos de índices que son mantenidos siempre que los documentos son almacenados, actualizados o eliminados.

El `índice estándar` es un índice basado en valor. Éste sirve para operaciones de búsquedas rápidas cuando se busca por elementos o atributos que tengan ciertos valores o para expresiones relacionales que cumplan ciertos valores. Los indices son definidos en el esquema Tamino usando el mecanismo appinfo de W3C schema XML.

Los `índices textuales` son los prerequisitos para una funcionalidad de recuperación de texto eficiente. En indexado de texto, las palabras contenidas en un elemento o atributo son indexadas, de tal forma que la búsqueda de palabras en el contenido de un elemento o atributo es rápido. Los elementos indexados no son únicamente los elementos hoja, sino también aquellos que contienen otros elementos.

Al tener estos dos tipos de índices , las consulta pueden ejecutarse muy eficientemente, de manera uniforme sobre una base de datos que contenga miles de gigabytes de datos.

Hay además un tipo especifíco de XML de índices llamado `índice de estructura`. Los indices de estructura condensada conservan la información sobre todas las rutas que ocurren en alguna instancia de un tipo de documento específico. Esto puede acelerar la ejecución de la consultas en algunos casos. Los indices de estructura completa conservan no solamente la existencia de rutas en un tipo de documento, sino además los documentos que se encuentran en esas rutas.

Organización en Disco

Las bases de datos Tamino están compuestas de dos partes persistentes (llamadas espacios):

- El espacio de datos contiene a todos los documentos y objetos almacenados en Tamino.
- El espacio de índices contiene los datos de los índices para los documentos almacenados en Tamino.

Ambos espacios pueden ser distribuidos sobre diferentes volúmenes de almacenamiento externo.

Consultas XML

Lenguaje de Consulta - Tamino X-Query

Tamino extiende XPath en dos aspectos:

- Tamino soporta búsqueda de texto sobre el contenido de atributos y elementos (incluyendo a sus hijos, ignorando marcas). Para este propósito Tamino define un nuevo operador relacional ˜ = (operador de contención). Con las capacidades de búsqueda de texto de Tamino, la ocurrencia de una palabra o una frase en el contenido de un elemento puede ser probado. Un asteriso representa un comodín para la búsqueda. La funcionalidad de búsqueda textual es independiente de los índices textuales, ellos pueden acelerar la búsqueda.
- Para habilitar al usuario para integhrar funcionalidad dedicada en las consultas, Tamino permite al usuario definir funciones para ser agregadas al lenguaje de consulta. Esta es otra funcionalidad del componente X-Tension.

Con el progreso de los esfuerzos de la estandarización de XQuery, Tamino XML Server deberá de soportar este lenguaje de consulta.

Funcionalidad Completa de Bases de Datos

Tamino XML Server está compuesto de un sistema de bases de datos completo. Éste soporta todas las características requeridas de un sistema de bases de datos, incluyendo operaciones multiusuario, soporte para transacciones, un poderoso concepto de respaldo, escalabilidad y rendimiento en la ejecución.

La seguridad es esencial para un sistema de bases de datos. El concepto de seguridad en Tamino consiste de varios componentes. Para un transporte seguro de datos a y desde Tamino XML Server, comunicación asegurada SSL puede ser usada. Además, Tamino puede restringir a los servidores Web que tendrán permitido comunicarse a la base de datos dedicada.

Las restricciones de acceso pueden ser a nivel colección, a nivel tipo de datos y a nivel elementos y atributos. Por lo tanto, es posible permitir el acceso a sólo una parte específica de un documento de cierto usuario.

Conclusión

Tamino XML Server proporciona toda la funcionalidad requerida en un sistema de bases de datos moderno que ha sido diseñado a fondo para manejar XML. Los documentos XML son almacenados nativamente en el almacen de datos de Tamino, sin ser mapeados a otro modelo. Tamino XML Server soporta un lenguaje de consulta específico, el cual incluye facilidades de recuperación de texto. Técnicas de indexado dedicadas aceleran la ejecución de consultas, Sin embargo, sigue almacenando independientemente de la estructura: Las operaciones y consultas aceptadas por Tamino, no dependende de la existencia de algún índice estructural.

El almacenamiento en Tamino XML Server proporciona alta flexibilidad: los esquemas son opcionales, podrian ser declarados como descripciones parciales de los documentos, o como descripciones de la estructura completa. Los datos pueden ser almacenados totalmente o parcialmente en Tamino, o datos pueden ser almacenados parcialmente en otro sistema de bases de datos (Tamino X-Node), o de otra manera usando las características de X-Tension. Un conjunto de herramientas gráfica facilita el desarrollo de aplicaciones Tamino.

3.6.2. XIndice

Xindice es la continuación de un proyecto anterior llamado dbXML Core [93], desarrollado por Apache Software Foundation. Xindice es una base de datos nativa XML. Xindice es un servidor de bases de datos diseñada para administrar un gran número de documentos pequeños. Los documentos son almacenados en colecciones y el servidor proporciona la capacidad para consultar estas colecciones usando XPath. El servidor es ligero, modular y adecuado para estar embebido en aplicaciones personalizadas o ejecutarse como una base de datos independiente. Xindice está escrito en Java. Para desarrolladores Java proporciona una implementación del API XML:DB la cual hace que el desarrollo de aplicaciones XML sea más fácil.

Todos los datos que van dentro y fuera del servidor son XML. El lenguaje de consulta utilizado es XPath y las APIs de programación soportan SAX y DOM. Al trabajar con XML y Xindice no hay mapeo entre diferentes modelos. Simplemente se diseñan los datos como XML y se almacenan como XML.

Características

El servidor actualmente proporciona soporte para almacenar documentos XML bien formados, es decir, no tiene un esquema que obligue a un documento a estar dentro de una colección. Esto hace a Xindice una base de datos semiestructurada y proporciona gran flexibilidad para almacenar los datos. Algunas de las características de Xindice son:

- **Colecciones de documentos**: El servidor Xindice fue diseñado para almacenar colecciones de documentos XML. Los documentos son almacenados en colecciones que pueden ser consultadas como un todo. Se pueden crear las colecciones que contienen documentos del mismo tipo o almacenar todos los documentos juntos. Las colecciones se referencian de una manera similara a como se trabaja con un sistema jerárquico de archivos.

- **Consultas con XPath**: Para consultar las colecciones de documentos en Xindice se utiliza XPath. Ya que este proporciona un mecanismo flexible para consultar documentos para navegar y restringir el árbol resultante que regresará la consulta.

- **Indización XML**: Para mejorar el rendimiento de las consultas sobre un gran número de documentos se pueden definir índices sobre elementos y valores de atributos. Esto puede acelerar considerablemente el tiempo de la respuesta de las consultas.

- **Implementación de XML:DB XUpdate**: Cuando se almacenan XML en la base de datos se desea la capacidad de actualizar la información sin recuperar el documento completo. XUpdate es el mecanismo que usa cuando se desea actualizar información del lado del servidor. Éste es un lenguaje basado en XML para especificar modificaciones y permitir que esas modificaciones sean aplicadas a un documento o a colecciones de documentos enteros.

- **Implementación del API XML:DB** : Proporciona una implementación del API XML:DB. Esta API tiene la intención de proporcionar la portabilidad de aplicaciones de bases de datos semejante a la que ha hecho JDBC para las bases de datos relacionales. La mayoría de las aplicaciones desarrolladas por XIndice utilizan esta API.

- **Arquitectura Modular**: El servidor Xindice está construido de una manera modular. Esto hace fácil para agregar y remover componentes para adaptar el servidor a un ambiente particular o para inclustarlo en otra aplicación.

Conclusión

Xindice es una buena opción para almacenar datos que son XML, pero si los datos no son XML y se tiene la necesidad de tener un control preciso sobre la estructura de datos es mejor utilizar otra solución de bases de datos.

3.6.3. eXist

Introducción

eXist inicia como un proyecto de código abierto en enero del 2001. Es un esfuerzo de código abierto, de Wolfgang Meier, para desarrollar sistemas de bases de datos nativas firmemente integrado con herramientas de desarrollo XML existentes como Cocoon de Apache. eXist cubre muchas de las características básicas de las bases de datos nativas XML, así como un número de tecnicas avanzadas como búsqueda de palabras claves en texto, consultas en la proximidad de terminos y patrones de búsqueda basadas en expresiones regulares. La base de datos es ligera, completamente escrita en Java.

A primera vista eXist tiene muchas similitudes a Xindice de Apache, cada proyecto esta dirigido a diferentes tipos de aplicaciones, y estan basadas en diferentes arquitecturas. eXist pone fuerte enfasis en el proceso eficiente de consultas basadas en índices. El motor de búsqueda de eXist ha sido diseñado para proporcionar consultas rápidas con XPath, usando índices para todos los nodos elementos, texto y atributos. eXist proporciona procesamiento de índices basadas en XQuery, soporte de XUpdate, extensiones de XUpdate e integración con herramientas de desarrollo XML.

Arquitectura

La aplicación de base de datos puede ejecutarse como servidor estandalone, embebido dentro de una aplicación o en una conexión con un contenedor de servlets. Estas tres alternativas se ejecutan en "hilos" independientes y soportan operaciones concurrentes por multiples usuarios.

Las aplicaciones pueden acceder a un servidor remoto eXist vía interfaces HTTP, XML-RPC, SOAP y WebDAV(Web-based Distributed Authoring and Versioning)

Características

- eXist proporciona almacenamiento de documentos XML sin la necesidad de la definición de un esquema o DTD. En otras palabras, se permite que los documentos sólo esten bien formados. Por lo que una base de datos XML debería soportar consultas en documentos similares, los cuales no tengan la misma estructura.

- Dentro de la base de datos, los documentos son administrados en colecciones jerárquicas. Los documentos pueden ser almacenados arbitrariamente dentro de la misma colección.

Los usuarios pueden consultar una parte distinta de la jerarquia de la colección o incluso todos los documentos contenidos en la colección usando la sintaxis de XPath con algunas extensiones.

Índices

Los índices son usados extensivamente por eXist para facilitar las consultas de manera eficiente. Para acelerar el proceso de las consultas, se requiere algún tipo de estructura de índices. eXist usa un esquema de índices númerico para identificar nodos XML en el índice. El indexado es aplicado a todos los nodos en el documento incluyendo: elementos, atributos, texto y comentarios.Contrario a otras propuestas, no es necesario crear índices explicitamente. Todos los índices son administrados por el motor de base de datos. Sin embargo, es posible restringir el indexado automático de texto para definir partes de un documento.

La versión actual de eXist usa tres tipos de índices.

- **Índice estructural:** Es creado y mantenido automáticamente en eXist. Tecnicamente, este índice mapea cada elemento y atributo en una colección de documentos a una lista de pares <documentoId, nodoId>. Este mapeo es utilizado por el motor de consultas para resolver consultas dada una expresión XPath.
- **Índices textuales:** Estos índices mapean elementos textuales (o bloques de texto estructurado) al documento de texto y nodos atributo en el que ellos ocurren. Éstos son creados y mantenidos automáticamente y almacenados en words.dbx. Un índice de este tipo puede ser configurado para seleccionar partes específicas de un documento. Si no hay configuración alguna, eXist genera índices del texto completo.
- **Índice de rango:** Este índice es "específico del tipo", lo cual significa que esta basado en el tipo de dato de valores de nodos específicos en el documento. Estos índices pueden ser configurados directamente por el usuario. Son similares a los índices usados por las bases de datos relacionales. Los índices son creados cuando se carga un documento y son automáticamente mantenidos durante actualizaciones posteriores al documento completo o una parte de él.

Lenguajes de Consultas

Agrega extensiones a XPath para hacer las consultas de manera más eficiente:

Dado que una base de datos podía contener un conjunto de documentos ilimitados, dos funciones adicionales son requeridas por el motor de eXist para determinar el conjunto de documentos contra el cual una expresión debería ser evaluada: document() y collection(). Document() acepta un nombre del documento, una lista de nombres de documentos o un comodín como parámetros. El comodin (*) selecciona todos los documentos en la base de datos. La función collection especifica la colección cuyos documentos serán incluidos dentro de la evaluación de la consulta. La colección raíz de la base de datos es siempre llamada /db. Por defecto, los documentos encontrados en subcolecciones debajo de la colección especificada son incluidos. Asi que no se tiene que especificar la ruta completa en expresiones precedentes.

XPath define solamente funciones limitadas para búsqueda de una cadena dada dentro del contenido de un nodo. el cual es un punto débil para búsquedas de documentos que contiene grandes secciones de texto. Además, la ejecución de las consultas podría ser demasiadas lentas.

eXist ofrece dos operadores adicionales y varias funciones que extienden XPath para proporcionar acceso eficiente basado en índices para nodos con texto. Así que la siguiente consulta:

//section[near(.,'XML database',50)]

Debería de regresar todas las secciones que contengan ambas palabras en el orden correcto y con un máximo de 50 palabras entre ellas.

En casos donde el orden y la distancia de los términos no son importantes, eXist ofrece dos operadores para búsqueda simple de palabras.

//SCENE[SPEECH[SPEAKER &='witch' and LINE &='fenny snake']]

&= es un operador de búsqueda de texto especial. Este selecciona los nodos contexto que contienen a todos los términos separados por espacios en los argumentos de la derecha. Para encontrar los nodos que contienen algunos de los términos , se define el operador |= . Si se tienen deshabilitados los índices textuales estos operadores se deshabilitan.

eXist proporciona un poderoso ambiente para el desarrollo de aplicaciones web basadas en XQuery y estandares relacionados. Las páginas del servidor XQuery se pueden ejecutar desde el sistema de archivos o almacenadas en las bases de datos.

Conclusión

Las aplicaciones prueban que eXist puede ser usado para implementar servicios basados en XML en un gran número de escenarios, tratando con diferentes tipos de documentos XML y volúmenes de datos. eXist es una base de datos XML nativa de código abierto, procesamiento XQuery basado en índices, indización automática, extensiones para búsqueda de texto completo, soporte de XUpdate, extensiones de XQuery y una estrecha integración con las herramientas de desarrollo XML. La base de datos implementa XQuery 1.0.

eXist proporciona un poderoso ambiente para el desarrollo de aplicaciones Web basadas en XQuery y estándares relacionados. Todas las aplicaciones web pueden ser escritas en XQuery, utilizando XSLT, XHTML, CSS y quizás Javascript. El servidor de páginas XQuery se puede ejecutar desde el sistema de archivos o almacenada en la base de datos.

3.6.4. Oracle XML DB

Oracle Database, con Oracle XML DB[1], es un híbrido de bases de datos para administrar XML y datos relacionales. Oracle XML DB es una característica de la base de datos Oracle, la cual proporciona almacenamiento nativo XML. Oracle Database 11g introduce un nuevo modelo de almacenamiento - incluye el soporte a formatos binarios de XML (Binary XML) y nuevas capacidades tales como índices, mayor soporte para XQuery.

Arquitectura

Oracle XML DB se basa en los componentes básicos de abstracción de XMLType, repositorio de XML, lenguajes estándares de consultas, soporte de API's.

[1]Desarrollado por la compañia Oracle.

Oracle XML DB Repository es un componente de Oracle Database que es optimizado para manejar datos XML como archivos en un sistema de archivos. Contiene recursos, los cuales pueden ser folders o archivos. Cada recurso puede tener estas propiedades:

- Identificado por una ruta y nombre.
- Tiene contenido, el cual puede ser XML pero no necesariamente.
- Tiene un conjunto de metadatos definidos por el sistema, tales como dueño y fecha de creación. Oracle XML DB usa esta información para administrar el recurso.
- Tiene una lista de control de acceso asociada que especifica quienes pueden acceder el recurso y con que operaciones.

Se pueden accesar los documentos XML en Oracle XML DB Repository usando protocolos estándares de conexión y acceso tales como FTP, HTTP(S) y WebDAV, además de los lenguajes SQL, PL/SQL,Java y C.

Características

XMLType

XMLType es un tipo de datos nativo para datos XML. Este proporciona métodos que permiten operaciones tales como validación de XML esquema y transformación XSL sobre contenido XML. XMLType es además un tipo de objeto, por lo que se puede crear una tabla de instancias XMLType.

Las tablas XMLType o columnas pueden estar limitadas para que se ajusten a un esquema XML. Esto tiene varias ventajas:

- La base de datos se asegura que sólo documentos válidos serán almacenados en la tabla o columna.
- Al conocer la estructura se pueden optimizar las consultas.
- Se puede almacenar la información usando almacenamiento estructurado, usando el modelo objeto-relacional, ya que el modelo es derivado del esquema.

Las columnas y tablas XMLType pueden ser almacenadas de la siguiente manera:

- Almacenamiento estructurado- Los datos XMLType son almacenados como un conjunto de objetos.
- Almacenamiento XML Binario- Los datos XMLType son almacenados en un formato binario específicamente diseñado para datos XML.
- Almacenamiento no estructurado - El dato XMLType es almacenado en instancias Character Large Object (CLOB).

Además, el almacenamiento estructurado y no estructurado pueden ser usados juntos en un almacenamiento híbrido. Por otro lado, Oracle XML DB proporciona indexación, consulta de contenido XML, actualización, transformación, comparación de documentos XML.

Índices Para diferentes modelos, Oracle XML DB soporta la creación de una variedad de índices sobre contenido XML:

- **Índices B-Tree**. Cuando la tabla o columna XMLType esta basada sobre técnicas de almacenamiento estructurado, los índices B-Tree pueden ser creados sobre los tipos SQL. Son típicamente creados usando funciones SQL qye extraen el valor.
- **Índices XML**. XMLIndex proporciona un índice específico XML que indiza la estructura interna de datos XML usando almacenamiento XML binario, no estructurada e hibrido.
- **Índices basados en Funciones**. Estos pueden ser creados sobre cualquier tabla o columna XMLType. Su uso es limitado a consultas predefinidas XPath sobre almacenamiento no estructurado.
- **Índices sobre texto completo**. Estos pueden ser creados sobre cualquier tabla o columna XMLType.

Lenguaje de Consulta

Las consultas de Oracle reescriben la tecnología para expresiones XQuery en funciones SQL/XML lo cual trae un rendimiento alto y escalabilidad para aplicaciones XML. Las funciones SQL/XML y sus correspondientes métodos XMLType usan expresiones XQuery para la búsqueda de colecciones de documentos XML y para accesar a un subconjunto de los nodos contenidos en un documento XML.

Conclusión

Oracle XML DB, a pesar de no ser una base de adtos nativa XML, proporciona capacidades para el almacenamiento eficiente, recuperación, consultas, generación y administración de grandes volúmenes de datos XML.

3.7. Comparativo de Bases de Datos

A continuación se muestra una comparación de las características de las bases de datos detalladas anteriormente:

Tamino

- Desarrollador: Software AG.
- Licencia: Comercial.
- Soporte de los estándares propuestos por la W3C (HTTP,XML, XQuery, XPath, XML Schema, Namespaces).

- XUpdateQueryService o algún otro lenguaje de actualización no esta implementado.
- Ofrece almacenamiento eficiente
- Alto rendimiento y disponibilidad ya que su "motor XML" es multi-proceso, permitiendo grandes volúmenes de peticiones y usuarios.
- Multiplataforma (puede invocarse desde cualquier cliente que tenga una máquina virtual instalada).
- Soporta XQuery como lenguaje de consulta y además un lenguaje definido por el propio fabricante , basado en XPath y denominado Tamino X-Query.
- Alta escalabilidad
- Soporte de multiples APIs y herramientas de desarrollo de terceros, al poder utilizar J2EE o .NET.

Ventajas

- Tiene muchas interfaces
- Gran variedad de plataformas soportadas
- Herramientas estándar para trabajar con Tamino.

Desventajas

- Está pensado principalmente para grandes aplicaciones
- Necesidad de entrenamiento específico con las herramientas.

Xindice

- Desarrollador: Apache Software Foundation
- Licencia: Código abierto (Open Source).
- Soporte API XML:DB (XAPI). Tres niveles de capas:
  - API XML DB (Java, soporta DOM y SAX).
  - API Xindice XML-RPC. Mediante el plugin XML-RPC es posible acceder a Xindice desde otros entornos o lenguajes.
  - API Server Core. Disponible solo para correr en la misma VM Java.
- Xindice se puede ejecutar de manera embabida o en modo servidor.
- Colecciones jerárquicas de documentos llamados recursos.
- Soporta los estándares establecidos por la W3C.
- Utiliza XPath como lenguaje de consulta y XML:DB Update como lenguaje de actualización.

- Consultas a nivel de documentos o de colecciones.
- Sistema de consultas a traves de colecciones no unificadas.
- No soporta permisos de acceso.
- No soporta concurrencia y transacciones.
- Herramientas de linea de comandos para administración de colecciones y documentos e indización.
- El almacenamiento esta basado en documentos.
- No está basado en nodos DOM persistentes.
- Está diseñado para administrar documentos de pequeño o menor tamaño.
- Cómo se mencionó anteriormente Xindice es una buena opción para almacenar únicamente documentos XML.

eXist

- Desarrollador: Wolfgang Meier
- Licencia: Código abierto (Open Source).
- Soporte API XML:DB (XAPI). No soporta transacciones.
- Soporta acceso concurrente de lectura y escritura.
- Altamente integrado con herramientas de desarrollo XML como Cocoon de Apache.
- La base de datos puede estar corriendo de manera independiente, dentro de un servlet motor o incorporada en la aplicación.
- Los documentos no tienen que estar asociados a un esquema o tipo de documento.
- Los documentos son manejados como colecciones.
- Utiliza XPath y Query optimizado. Además XInclude, XPointer (parcial), XUpdate, XSL/XSLT.
- DOM, SAX
- Procesamiento de consultas basados en índices.
- Consultas con o a través de colecciones o documentos.
- Indezación automática por default
- Puede definir indíces. Para elementos y valores de atributos.
- Índices estructurales para nodos elemento y atributo.

- Índices de texto completo para texto y valores de atributos.
- Pueden ser configuradas para seleccionar partes de un documento.
- Actualizaciones a nivel de documento (XUpdate de forma limitada)
- Actualizaciones a nivel de nodos.
- Permisos de acceso similar a Unix para grupos y usuarios a nivel de colecciones y documentos.
- Funcionalidad de copia de seguridad y restauración es proporcionada vía clientes de administración Java o scripts Ant.
- Permite restauración completa de una base de datos incluyendo permisos de usuario/grupos.
- El almacenamiento esta basado en nodos.
- Basado en nodos DOM persistentes.
- Consta de herramientas de administración con una consola gráfica.

Oracle XML Database

- Desarrollador: Oracle
- Licencia: Comercial.
- Consta de un conjunto de APIs propio. Las API's Oracle XML DB estan disponibles para aplicaciones corriendo en el server o fuera de la base de datos.
- Oracle Database 11g permite almacenar objetos grandes (LOB's) tales como imágenes, grandes objetos de texto, o tipos de datos almacenados.
- Incorpora soporte basado en estándares XInclude y XLink.
- Utiliza XMLType.
- Incluye soporte a formatos binarios de XML(Binary XML). La capacidad de usar el XML binario permite la opción de almacenarlo y recuperarlo con un desempeño más veloz.
- Sólo permite almacenamiento de documentos válidos.
- Soporta variedad de índices sobre contenido XML.
- los estándares establecidos por la W3C (XML Namespaces, XML Schema, DOM 1.0, DOM 2.0,XSLT, XPath, XQuery).
- Utiliza funciones SQL/XML para la búsqueda de colecciones de documentos.
- XPath lo utiliza para consultas jerárquicas.

- Sólo esta disponible para algunas plataformas.
- Permite manejo de gran cantidad de documentos XML de manera eficiente.
- En Oracle se pueden almacenar los documentos en tablas relacionales.

3.8. Resumen

En el presente capítulo se describieron diferentes maneras de almacenar documentos XML en diversos tipos de bases de datos; principalmente se muestran las características de las bases de datos nativas XML. El tema principal de este capítulo son las bases de datos nativas, ya que se utilizará una de ellas para el desarrollo del proyecto.

Capítulo 4

Desarrollo del Sistema para Generación de Documentos.

4.1. Introducción

Dada la necesidad de la utilización de documentos semiestructurados, es decir, documentos con información que puede ser irregular y no respeta un esquema en particular (sus componentes no siempre son del mismo tipo, no tienen estructura fija ya que pueden cambiarla, son auto descriptivos, etc.), también surge la necesidad de administrar esta información y de centralizarla para poder manipularla. Por lo que surgen diversas tecnologías para el manejo y administración de datos de documentos XML.

En el presente proyecto se desarrolló una aplicación utilizando algunas de esas tecnologías. La aplicación va dirigida a profesores, pensada como una herramienta de apoyo a las labores docentes.

Tomando como base la información de tareas de un curso de alguna materia en particular, se desarrolló un sistema que permite al profesor del curso almacenar documentos XML en una base de datos nativa XML; dichos documentos XML contienen los ejercicios del curso. Una vez almacenada la información se puede consultar y eliminar si así se desea.

Por otro lado, el sistema permite al profesor generar documentos personalizados a partir de los ejercicios previamente almacenados. Posteriormente puede consultar, imprimir y/o eliminar los documentos generados si así lo desea.

El sistema consta de dos módulos:

- Módulo administrativo de ejercicios: en el cual se registran, consultan y eliminan los documentos de la base de datos. Estos documentos, como se mencionó anteriormente, pueden contener uno o varios ejercicios del curso. Además, este módulo permite agregar las imágenes requeridas por los documentos XML en el sistema.

- Módulo de generación y administración de documentos: en el cual se hacen búsquedas sobre los documentos XML almacenados, y se permite generar documentos personalizados a partir de esta información

4.2. Objetivo

El objetivo principal del sistema es permitir a los profesores de alguno o varios cursos en particular, almacenar ejercicios capturados en documentos XML en una base de datos nativa XML y posteriormente recuperarlos mediante algunos criterios para la generación de documentos. Documentos que se generarán a partir de uno o varios de los ejercicios que se incluían en los documentos XML.

Una vez generado el documento el sistema lo registrará y permitirá obtenerlo posteriormente para poder consultarlo, imprimirlo y/o eliminarlo si así se desea.

4.3. Análisis

Se desarrollará un sistema capaz de obtener la información de documentos XML y de registrar dicha información. El documento XML deberá seguir la estructura definida en un esquema XML. El sistema validará que únicamente se registren los documentos que sigan dicha estructura, es decir, sólo aceptará documentos válidos.

El documento que se registrará puede contener uno o varios ejercicios propuestos por el profesor que utilizará el sistema.

El sistema permitirá administrar esta información registrándola y permitiendo asociar al documento las imágenes relacionadas. Por otro lado, permitirá darle mantenimiento al sistema eliminando los ejercicios que no estén completos o que ya no se desee mantener en el sistema. Sólo se eliminarán los documentos que sean seleccionados por el usuario.

Una vez registrada la información de los ejercicios el sistema permitirá la consulta de aquellos que cumplan con algunos de los siguientes criterios:

- Nombre
- Tema
- Materia
- Nivel de dificultad

El sistema regresará una lista de ejercicios que cumplan los criterios indicados y permitirá seleccionarlos para que sean parte del documento que se generará. Si se desea cambiar de tema o buscar más ejercicios el sistema permitirá mantener en el documento generado aquellos documentos previamente seleccionados, a menos que el usuario desee generar un nuevo documento, con lo que se perderá la selección anterior. Conforme se vayan agregando ejercicios al documento el sistema permitirá generar una versión previa para que el usuario pueda verlo completo ya generado y se pueda dar una idea de si realmente es el conjunto de ejercicios que desea en el documento.

Es importante recalcar que el despliegue de la información obtenida en el documento será aleatoria. Esto para permitir que el sistema no siempre despliegue los ejercicios en el mismo orden y permitir al usuario no siempre visualizar los mismos ejercicios al

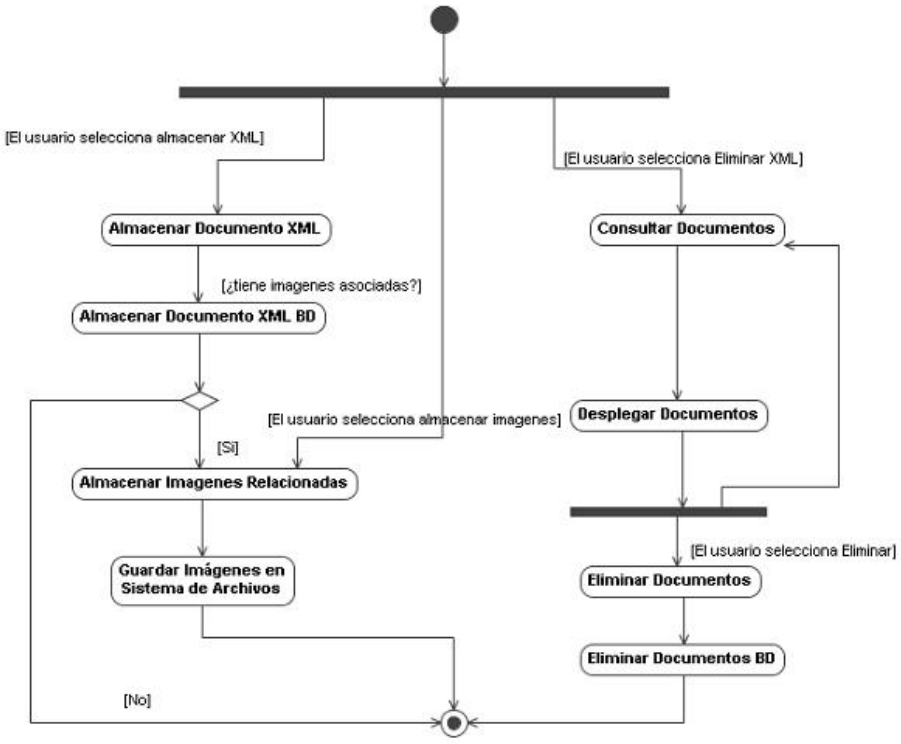

Figura 4.1: Diagrama de actividades del módulo administrativo de ejercicios.

principio del listado y tener que buscar en el final o a la mitad de la lista si desea diferentes ejercicios.

Una vez generado el documento, es decir que ya no se desee agregar más ejercicios, el sistema permitirá personalizarlo de tal manera que se pueda indicar la fecha de entrega, el título del documento, el tema y la materia a la que pertenece. Posteriormente se generará el documento final y se registrará. El sistema permitirá consultar y eliminar el documento generado si así se desea.

Como se mencionó anteriormente el sistema constará de dos módulos:

- Módulo administrativo de ejercicios. El flujo del sistema es mostrado en la figura 4.1.
- Módulo de generación y administración de documentos. El flujo del sistema es mostrado en la figura 4.2.

4.4. Desarrollo

4.4.1. Herramientas

Para desarrollar el proyecto se utiliza la base de datos **eXist**, base de datos nativa desarrollada por Wolfgang Meier, para administrar y obtener la documentación XML. Se decidió utilizar esta base de datos por sus características, acceso multiusuario, lenguajes de

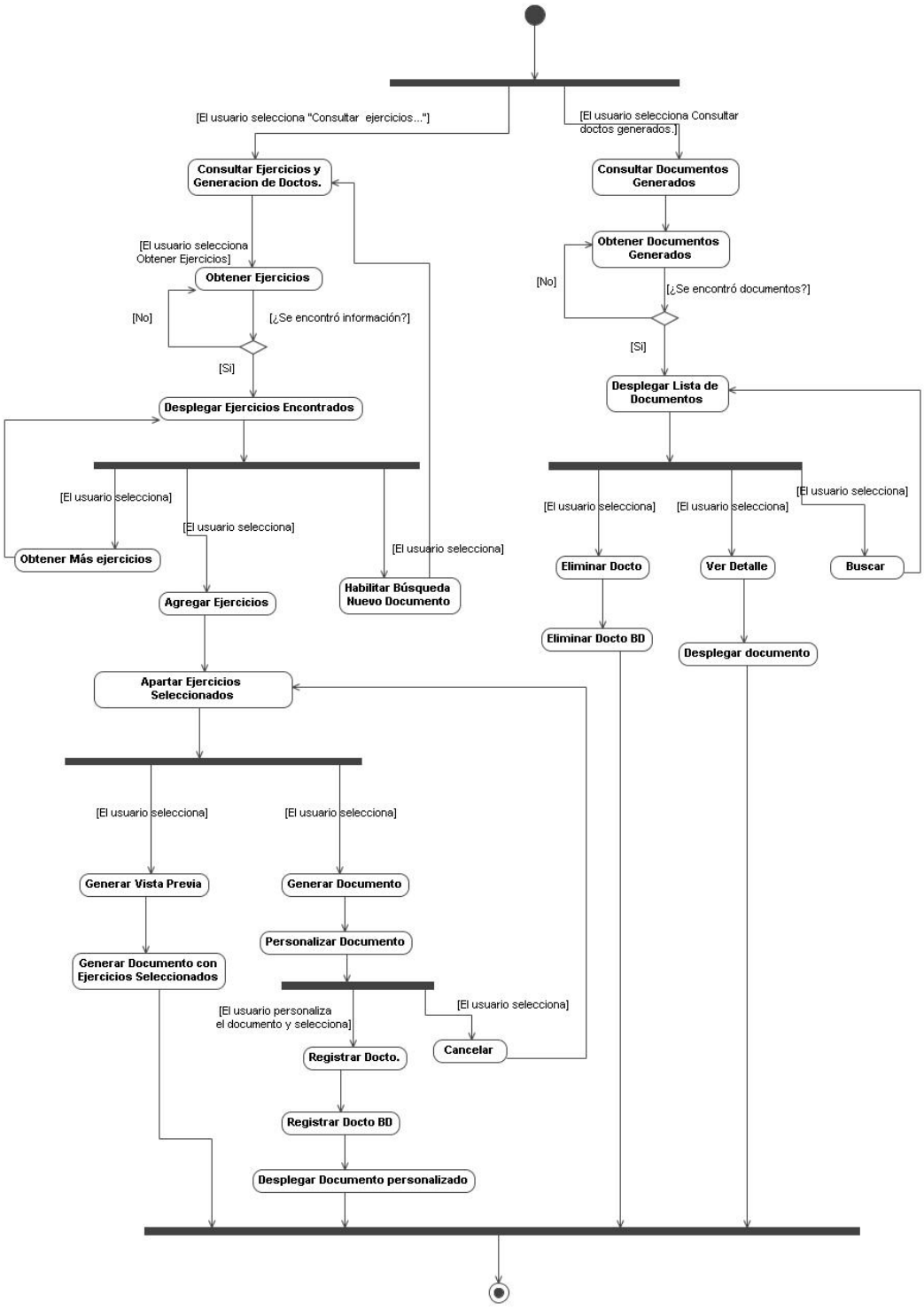

Figura 4.2: Diagrama de actividades del módulo de generación y administración de documentos.

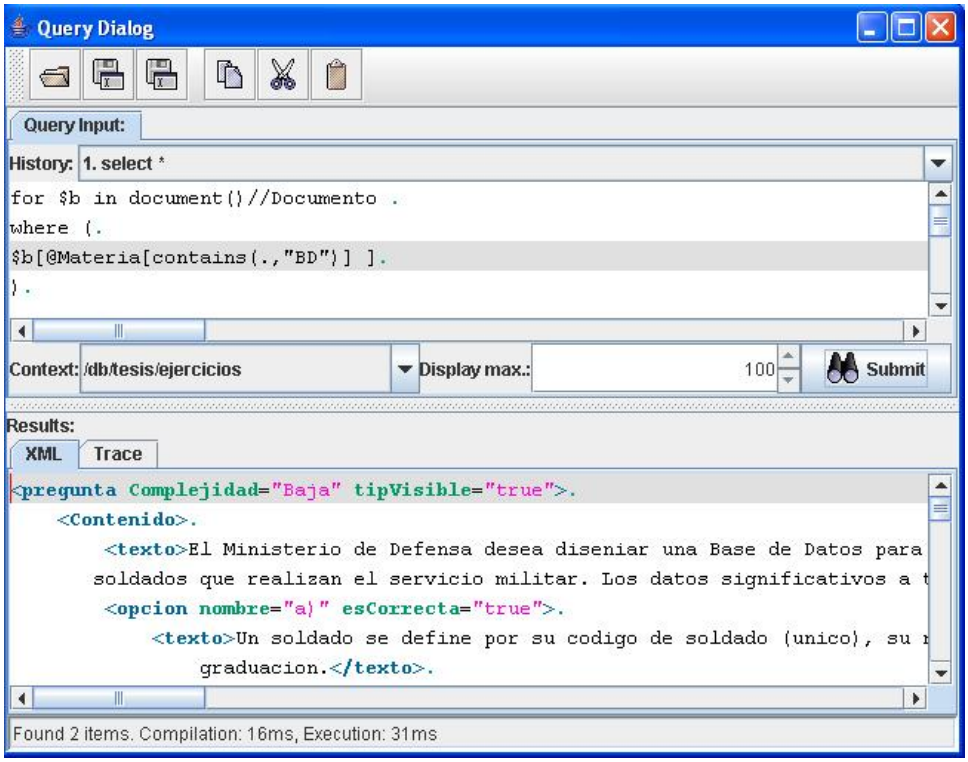

Figura 4.3: Consola para ejecutar consultas de eXist.

consultas, etc., pero especialmente por ser una tecnología de código abierto, por tener un ambiente de administración gráfico, que permite entre otras cosas, asignar permisos de grupo y de usuarios, además de permitir ejecutar las consultas en una consola gráfica(Query Dialog) y verificar si los resultados obtenidos son los esperados, como se muestra en la figura 4.3.

Asimismo, soporta acceso de manera concurrente de lectura y escritura.

Como lenguaje de programación se utilizará Java. Además se usarán algunas de las tecnologías XML como son: XPath, XQuery, XSLT, DOM, Xerces[1] y para validar el documento se utiliza un esquema XML. El sistema se instalará en el contenedor de aplicaciones Tomcat.

La aplicación está desarrollada en capas para permitir cambiar la conexión a la base de datos, sin tener que modificar toda la aplicación.

4.4.2. Módulo Administrativo

Como ya se mencionó anteriormente, la aplicación está dividida en dos módulos. A continuación se describirá el primero de ellos: `Módulo administrativo de ejercicios`.

El módulo está dividido en tres secciones

- Agregar XML

[1]Familia de paquetes de software para analizar y manipular XML. Anteriormente pertenecía al proyecto Apache.

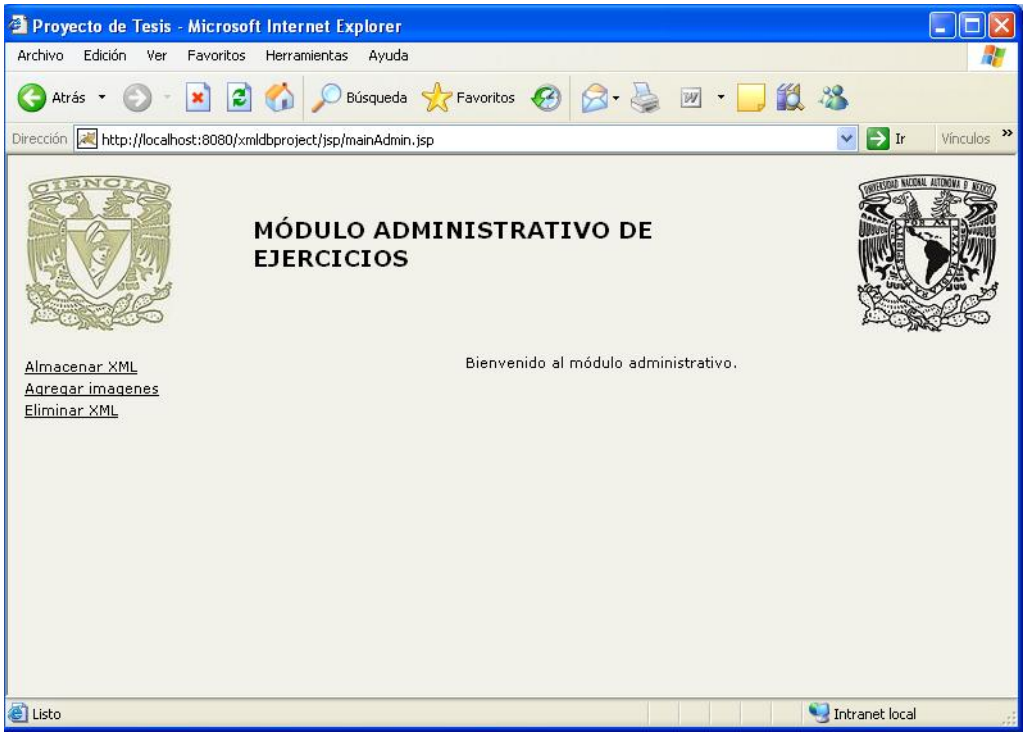

Figura 4.4: Menú principal del módulo administrativo de ejercicios.

- Agregar Imágenes
- Eliminar XML

En la opción de agregar documentos XML, se registran en la aplicación aquellos documentos que contienen los ejercicios del curso. Dichos documentos deben de ser válidos, para lo cual se generó un esquema; si el documento no es válido, el sistema no permite su registro en el sistema.

En el esquema se define cómo deben de estar estructurados los documentos que se deseen registrar en el sistema. Dado que los documentos contienen los ejercicios de un curso, se decidió que el elemento raíz del sistema sea el elemento "Ejercicios". Como esos ejercicios pertenecen a una materia en particular, se agregaron atributos específicos para ese curso como son: materia, nombre y tema. Además se agregó un identificador al documento, el cual permite identificarlo de manera única, por lo que se así se desea podría recuperarse el documento mediante ese identificador.

```
<xsd:element name="Ejercicios">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="requisitos" type="xsd:string"/>
        <xsd:element minOccurs="1" maxOccurs="unbounded" name="pregunta"
        type="Ejercicio" nillable="false"/>
      </xsd:sequence>
      <xsd:attribute name="Identificador" type="xsd:string" use="required"/>
```

```
      <xsd:attribute name="Materia" type="xsd:string" use="required"/>
      <xsd:attribute name="Tema" type="xsd:string" use="optional"/>
      <xsd:attribute name="Nombre" type="xsd:string"/>
    </xsd:complexType>
</xsd:element>
```

Cada documento puede contener uno o más ejercicios, por lo que en el esquema se especifica mediante:

```
<xsd:element minOccurs="1" maxOccurs="unbounded" name="pregunta"
type="Ejercicio" nillable="false"/>
```

Cada ejercicio (pregunta) consta de contenido, respuesta y una ayuda para la solución del ejercicio. Dependiendo del valor del atributo tipVisible se muestra esa ayuda o no.

```
<xsd:complexType name="Ejercicio">
    <xsd:sequence>
      <xsd:element minOccurs="1" maxOccurs="1" name="Contenido"
      type="TipoEjercicio"/>
      <xsd:element minOccurs="1" maxOccurs="1" name="hint"
       type="xsd:string" nillable="true"/>
      <xsd:element minOccurs="1" maxOccurs="1" name="respuesta"
      nillable="true" type="RespEjercicio"/>
    </xsd:sequence>
    <xsd:attribute name="tipVisible" type="xsd:boolean"/>
    <xsd:attribute name="Complejidad" type="NivelComplejidad" use="required"/>
</xsd:complexType>
```

Además dado que en un curso hay ejercicios de diferentes complejidades se asignó una complejidad a cada uno de los ejercicios. El valor que puede tomar el nivel de complejidad puede ser uno de los siguientes valores: Baja, Media o Alta.

```
<xsd:simpleType name="NivelComplejidad">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Baja"/>
    <xsd:enumeration value="Media"/>
    <xsd:enumeration value="Alta"/>
  </xsd:restriction>
</xsd:simpleType>
```

El ejercicio puede estar planteado mediante texto, imagen o un conjunto de opciones. Donde cada opción puede a su vez ser también texto o una imagen.

```
<xsd:complexType name="TipoEjercicio">
    <xsd:sequence>
      <xsd:choice minOccurs="1" maxOccurs="unbounded">
        <xsd:element name="texto" type="xsd:string"/>
        <xsd:element name="imagen" type="xsd:string"/>
```

```
        <xsd:element name="opcion" type="TipoOpcion"/>
      </xsd:choice>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="TipoOpcion">
    <xsd:sequence>
      <xsd:choice minOccurs="1" maxOccurs="unbounded">
        <xsd:element name="texto" type="xsd:string"/>
        <xsd:element name="imagen" type="xsd:string"/>
      </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="nombre" type="xsd:string"/>
    <xsd:attribute name="esCorrecta" type="xsd:boolean"/>
</xsd:complexType>
```

Cada ejercicio contiene una respuesta, la cual tiene la misma estructura que el elemento pregunta.

```
<xsd:complexType name="RespEjercicio">
    <xsd:sequence>
      <xsd:element minOccurs="1" maxOccurs="1" name="Contenido"
       type="TipoEjercicio"/>
    </xsd:sequence>
</xsd:complexType>
```

El esquema generado se encuentra completo en el apéndice B.1.

Una vez validado el documento, el sistema lo registra en la base de datos en la colección **tesis/ejercicios**. Posteriormente de registrar el documento, el sistema valida si el documento tiene imágenes relacionadas y de ser así indica al usuario si desea asociar las imágenes inmediatamente. Si no se desea hacer en ese momento se puede hacer posteriormente mediante la opción "Agregar Imágenes". Las imágenes relacionadas al documento son aquellas que se utilizan como referencia en el documento XML, con la cual se apoya el profesor para plantear una pregunta o para dar una respuesta a un ejercicio; normalmente se incluye una imagen cuando la pregunta es compleja y no se puede describir únicamente con texto, o porque la respuesta y/o pregunta depende de alguna gráfica o diagrama.

Al desarrollar esta parte de la aplicación se pensó en registrar la imagen a la base de datos, pero el primer problema que se presentó es que si se registraba era difícil asociarla a un documento. Además, dado que la base de datos elegida sólo permite almacenar documentos XML se tuvo que "simular" el registro y en realidad lo que se hace es guardarlas en el sistema de archivos en una ruta accesible en la cual se tienen todas las imágenes relacionadas a los ejercicios. Para poder almacenarlas se requiere tener permisos de escritura, y de lectura para posteriormente poder tener acceso a ellas.

Después de registrar los documentos XML y sus imágenes relacionadas, el profesor que utiliza el sistema, tiene la opción de consultar los documentos y eliminarlos si así lo requiere. En la figura 4.5 se muestra un ejemplo de consulta con la lista de documentos de la materia "BD" encontrados.

Se pueden consultar los documentos mediante alguno de los siguientes criterios:

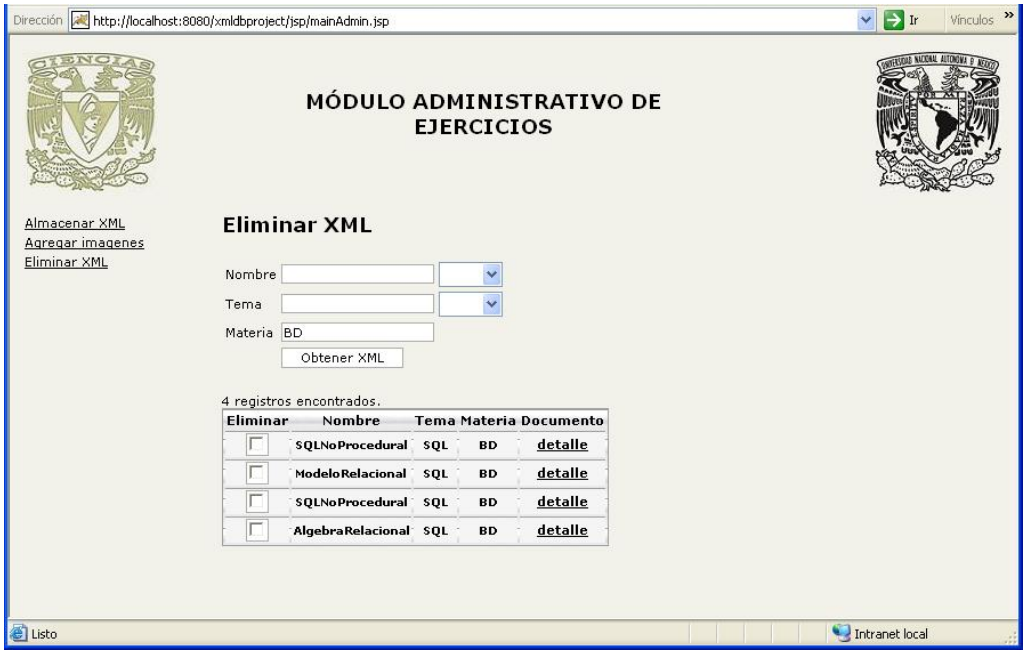

Figura 4.5: Ejemplo de consulta de documentos.

- Nombre
- Tema
- Materia

Para consultar la información en la base de datos se utilizó XPath. Los campos que se utilizan son atributos del elemento raíz "Ejercicios" por lo que se generó la consulta de la siguiente manera:

Atributo[contains("valorAtributo")]

Además, se generó a partir de los documentos XML la lista de ejercicios utilizando DOM. Se obtuvo el documento como DOM y el primer elemento del documento mediante documento.getFirstChild(). Posteriormente se obtuvo el valor de los atributos, mediante attributes.getNamedItem("nombreAtributo") para desplegar la información en la lista de resultados. El sistema permite paginación de esta lista.

Por otro lado, el profesor puede ver el documento HTML generado a partir del documento XML con los ejercicios registrados en el sistema. Para eso basta con que presione la liga "detalle", mostrada en la figura 4.5, de la sección documento del registro que desee consultar. Ésta opción permite al profesor ver todos los ejercicios como se le presentarán al alumno, y ya no como documento XML, con lo cual valida si es realmente el documento que deseaba almacenar.

Para el despliegue y generación de los documentos HTML se utilizó XSL. Se optó por XSL para permitir de manera más sencilla el cambio en la presentación del documento. Sencilla

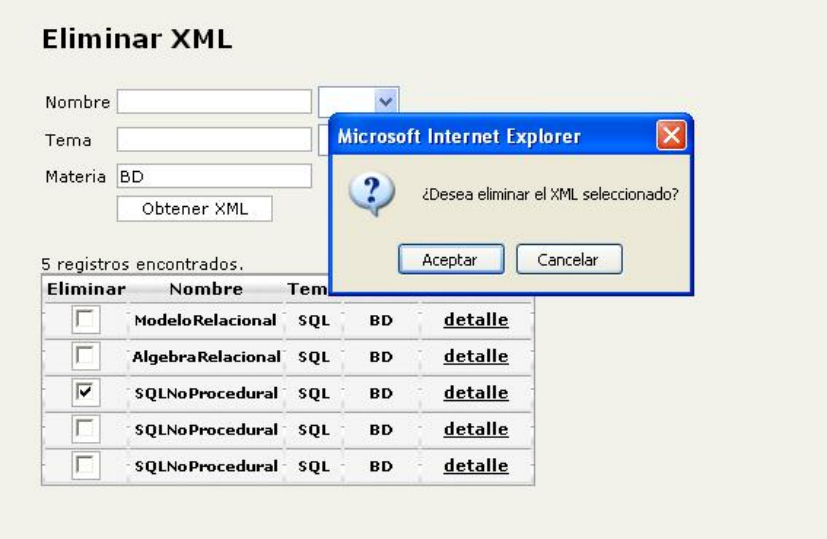

Figura 4.6: Ejemplo de eliminación de documentos.

en términos de que no se requiere recompilar el código. Basta con reiniciar el contenedor de aplicaciones.

El almacenar las imágenes en el sistema de archivos en una ruta accesible, en la cual se tienen todas las imágenes relacionadas a los ejercicios, permite desplegar las imágenes en el documento HTML. Por lo que, el documento .xsl generado para la aplicación recibe como parámetro la ruta del directorio donde se almacenan las imágenes, la cual es establecida mediante el archivo de propiedades xmldbtesis.properties. Cómo se muestra a continuación:

url.image = C:/xmldbtesisfiles/imagenes/

El documento XSL (xmldbtesis.xsl) concatena la ruta del directorio con el nombre de la imagen por lo que despliega sin ningún problema el documento con la(s) imágen(es) asociada(s). Éste documento se encuentra detallado en el apéndice B.2.1.

Un inconveniente para el despliegue adecuado de las imágenes es que el directorio debe contar con permisos de lectura. Otro inconveniente es que el nombre de la imagen en el documento XML debe coincidir con el nombre real de la imagen, por lo que el usuario debe tener cuidado en no repetir el nombre al almacenar las imágenes o no hacer referencia en diferentes documentos a la misma imagen, a menos que realmente se trate de la misma. El usuario es el encargado de relacionar todas las imágenes con el documento, es decir, que el sistema no verifica si todas las imágenes indicadas en el documento existen ya en el sistema de archivos por lo que si no se almacenó la imagen al generar el documento se despliega el documento sin la(s) imágen(es) faltante(s).

Por último, si el profesor detecta que el documento es incorrecto o simplemente desea eliminarlo basta con que seleccione el documento en la sección “Eliminar” como se muestra en la figura 4.6 y el sistema preguntará si realmente desea eliminarlo. De ser así con la confirmación el documento será eliminado de la base de datos.

A partir de este momento ya se pueden administrar los documentos en la base de datos.

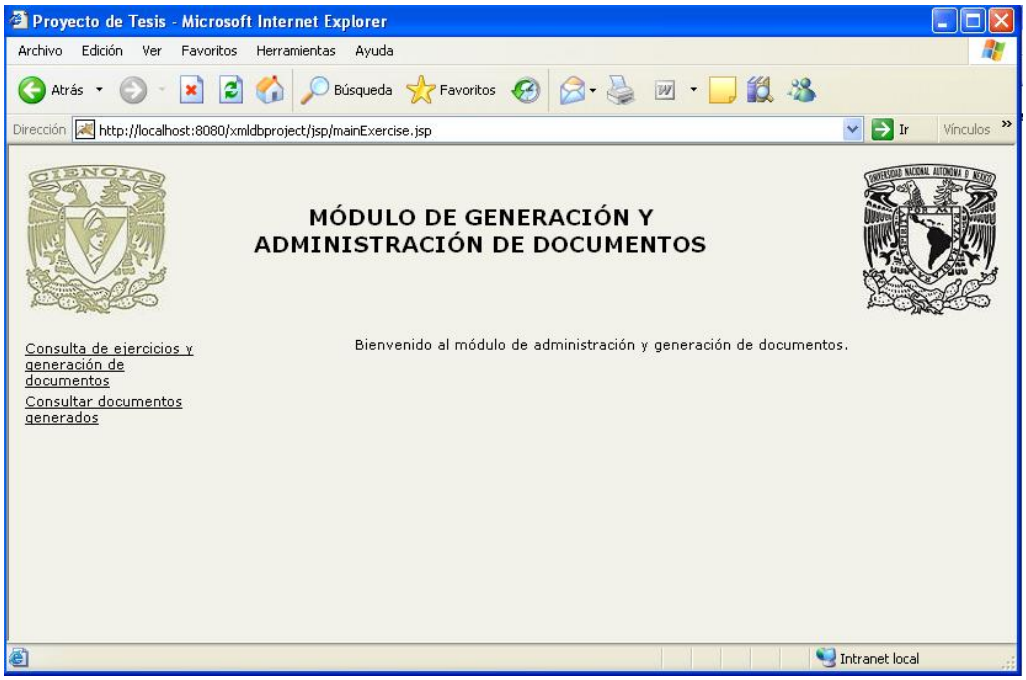

Figura 4.7: Menú del módulo de generación y administración de documentos.

Por lo que el siguiente paso es la generación de los documentos a partir de los ejercicios almacenados.

4.4.3. Módulo de Generación de Ejercicios

Esta parte del sistema quedó dividida en dos partes:

- Consulta y generación de documento.
- Consulta y/o eliminación de documentos generados.

Si el profesor selecciona la primera opción se mostrará inicialmente la pantalla mostrada en la figura 4.8, en la cual podrá consultar los ejercicios con alguno de los siguientes criterios:

- Nombre
- Tema
- Materia
- Nivel de dificultad

Para lo cual se generó un sistema de búsqueda utilizando XQuery para obtener los ejercicios que cumplieran con alguno de los criterios mencionados anteriormente. Se obtuvo el documento mediante DOM pero aquí se obtuvo el elemento raíz mediante docto.getDocumentElement() y el nodo ejercicio mediante NodeList contents = docto.getChildNodes();

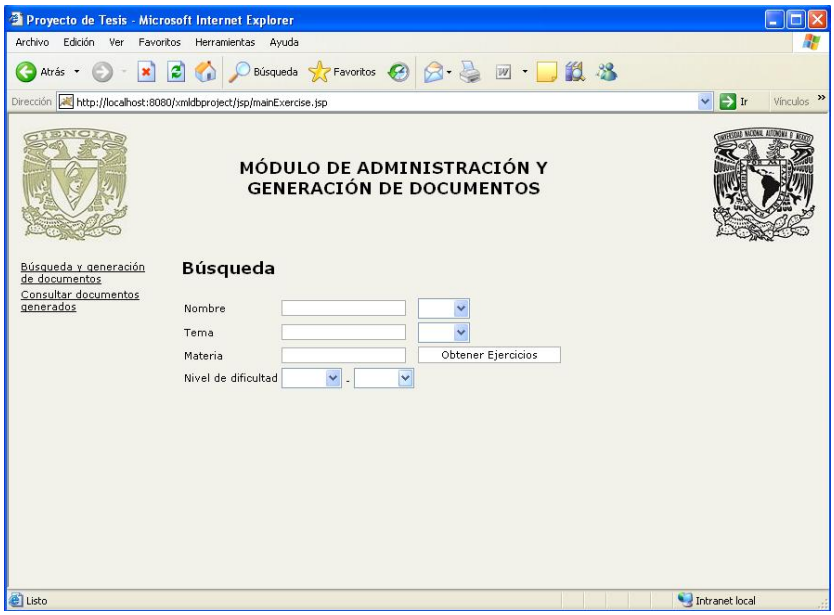

Figura 4.8: Ejemplo de consulta de ejercicios.

La generación del documento no fue tan trivial debido a que en la consulta anterior lo que se requería era el documento completo y en esta sección lo que se requiere es desplegar cada uno de los ejercicios que cumplan la condición sin importar que estén en el mismo documento. Al ejecutar la consulta, se obtiene una cadena con formato XML pero del tipo

```
<Ejercicios>
  <pregunta>
    ...
  </pregunta>
</Ejercicios>
```

por lo que para la generación del documento mediante XSL se le tuvo que concatenar

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

quedando de la siguiente manera:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Ejercicios>
  <pregunta>
      ...
  </pregunta>
</Ejercicios>
```

con lo cual se podía generar el documento mediante XSL sin problemas. De esta manera, en el listado se muestra una pequeña descripción del ejercicio, pero si se desea consultar completo se debe seleccionar la liga “Ver pregunta” en la sección Documento del listado de ejercicios encontrados, cómo se muestra en la figura 4.9 y se generará un documento HTML

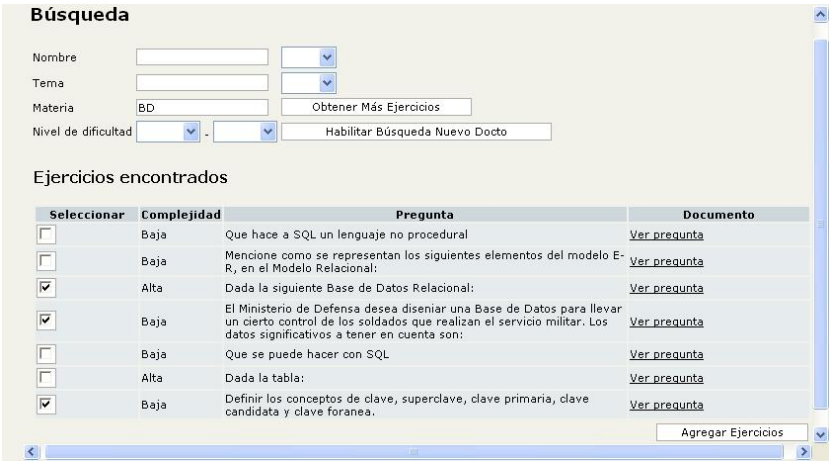

Figura 4.9: Ejemplo de resultado de ejercicios.

que contiene la descripción completa del ejercicio incluyendo las imágenes relacionadas. El documento XSL generado, preguntaxmldb.xsl, para desplegar la información de la pregunta se encuentra detallado en el apéndice B.2.2.

Para agregar ejercicios al documento, el profesor debe seleccionar el o los ejercicios que desea agregar en la sección "Seleccionar" y presionar la opción "Agregar Ejercicios" cómo se muestra en la figura 4.9.

Una vez elaborada la primer consulta se implementó el sistema de tal manera que permitiera seleccionar ejercicios de diferentes temas, lo cual ocasionó que al ejecutar una consulta se mantuvieran en sesión los ejercicios ya seleccionados para agregarse al documento final. Ésto permite al profesor seleccionar ejercicios de diferentes temas o que cumplan varios de los criterios para incluirlos en el mismo documento. Para generar una nueva búsqueda y mantener los ejercicios seleccionados basta con que el profesor indique nuevos criterios de búsqueda y seleccione la opción "Obtener Más Ejercicios", mostrada en la figura 4.9.

También el sistema permite generar un nuevo documento, es decir, si el usuario decide que ninguno de los ejercicios anteriormente seleccionados son los que desea para el documento final, puede decidir reiniciar la búsqueda con lo cual se elimina la información de sesión y se reinicia con la generación de un nuevo documento. Para reiniciar la búsqueda el profesor debe seleccionar la opción "Habilitar Búsqueda Nuevo Docto" mostrada en la figura 4.10.

Si no se tienen ejercicios agregados al documento se puede usar cualquiera de las 2 opciones: "Obtener Más Ejercicios" o "Habilitar Búsqueda Nuevo Docto".

Posteriormente, si el profesor decide que ya seleccionó todos los ejercicios que desea agregar al documento y decide generar el documento, deberá presionar la opción "Generar Documento" mostrada en la figura 4.10. Esto indica al sistema que se desea personalizar el documento, por lo que se muestra la pantalla mostrada en la figura 4.11.

En esta sección el profesor indica los datos específicos del documento:

- Tema

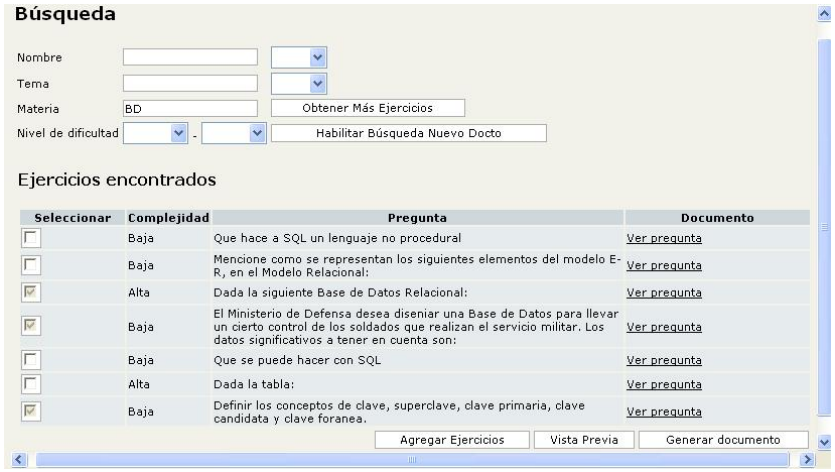

Figura 4.10: Ejemplo de ejercicios agregados al documento.

Figura 4.11: Personalización del documento con ejercicios seleccionados.

- Materia
- Título
- Fecha de Entrega

Adicionalmente puede agregar una nota al documento, es decir, información adicional que no se encuentra en ninguno de los ejercicios como, por ejemplo, lugar de entrega.

Para la personalización del documento lo que se hizo fue concatenar todos los ejercicios seleccionados en el listado. Pero además se concatenó la información capturada en la pantalla al principio del documento. Además de generar un elemento raíz llamado "Documento" para posteriormente generar un documento HTML mediante XSL.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Documento Tema="nombreTema" Materia="nombreMateria"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <titulo>descripcionTitulo</titulo>
  <nota>descripcionNota</nota>
  <fechaEntrega>fechaEntrega(formato yyyy-MM-dd)</fechaEntrega>
  <pregunta>descripcionEjercicio</pregunta>
</Documento>
```

Una vez personalizado el documento, el profesor, debe seleccionar la opción "Registrar Docto" mostrada en la figura 4.11, mediante la cual se registrará el documento en el sistema y por otro lado, generará el documento en formato HTML con la información capturada, permitiendo así la impresión del documento si así se desea. Un ejemplo de documento generado se muestra en la figura 4.12.

Para la generación del documento se generó otro documento XSL, documentxmldb.xsl, para poder desplegarlo en formato HTML; se encuentra detallado en el apéndice B.2.3.

De esta manera, la información queda registrada en la base de datos en la colección **generados**. Se registra en una colección totalmente independiente de donde se encuentran almacenados los ejercicios, para poder tener acceso a estos documentos sin mezclar información y para posteriormente eliminarlos si así se desea.

Para la consulta de los documentos generados se creó la opción "Consultar documentos generados" mostrada en la figura 4.13. La búsqueda se puede hacer mediante alguno de los criterios capturados en la personalización del documento:

- Materia
- Título
- Rango de Fechas de Entrega

La búsqueda por un rango de fechas, permite al profesor hacer una consulta más amplia, por ejemplo, en caso de no recordar la fecha exacta.

Para consultar estos documentos se generó una búsqueda utilizando XQuery a partir de los criterios mencionados, donde la búsqueda por fecha de entrega es un rango de fechas. A la

Tema: Modelo E.R. y Modelo Relacional
Materia: BD

Tarea 1

Fecha de entrega: 29 de Febrero del 2008

Se aceptara solo en formato electronico.

1. Dada la siguiente Base de Datos Relacional:
   FEDERACION (NOMBRE#, DIRECCION, TELEFONO)
   MIEMBRO (DNI#, NOMBRE_M, TITULACION)
   COMPOSICION (NOMBRE#, DNI#, CARGO, FECHA_INICIO)

   Se pide dar respuesta algebraica a las siguientes consultas:
   1. Obtener el nombre de los presidentes de la federacion.
   2. Obtener la direccion de aquellas federaciones que tienen gerente.
   3. Obtener las federaciones que no tienen asesor tecnico.
   4. Obtener las federaciones que tienen todos los cargos.
   5. Obtener las federaciones que tienen asesor tecnico y psicologo.

   Complejidad:Alta

   Solucion:
   1. $\Pi_{\text{NOMBRE\_M}}(\sigma_{\text{CARGO}='\text{PRESIDENTE}'}(\text{COMPOSICION}) * \text{MIEMBRO})$
   2. $\Pi_{\text{DIRECCION}}(\sigma_{\text{CARGO}='\text{GERENTE}'}(\text{COMPOSICION}) * \text{FEDERACION})$
   3. $\Pi_{\text{NOMBRE\#}}(\text{FEDERACION}) - \Pi_{\text{NOMBRE\#}}(\sigma_{\text{CARGO}='\text{ASESOR TECNICO}'}(\text{COMPOSICION})$
   4. $\Pi_{\text{NOMBRE\#,CARGO}}(\text{COMPOSICION}) \div \Pi_{\text{CARGO}}(\text{COMPOSICION})$

Figura 4.12: Documento generado en el sistema.

Figura 4.13: Documento generado en el sistema.

hora de implementarse esta parte del sistema, dado que en la interfaz se captura la fecha con el formato día-mes-año, `dd-MM-yyyy` en Java (ejemplo: 12-02-2007), se presentó el problema de no poder obtener los documentos debido a que XQuery únicamente reconoce las fechas en el formato año-mes-día, `yyyy-MM-dd` en Java (ejemplo: 2007-02-12), por lo que en el registro se tuvo que modificar el formato utilizando el que reconoce XQuery y al obtener el documento, antes de desplegar la información, se modificó para que tuviera el formato en el que se capturó.

```
for $b in document()//Documento
where
  ( $b[fechaEntrega > xs:date("fechaEntregaInicial")]
and
  $b[fechaEntrega < xs:date("fechaEntregaFinal")] )
```

Para ver el documento generado, el profesor debe seleccionar la liga en la sección "Documento" mostrada en la figura 4.13, lo cual generará el documento en formato HTML. Si se desea eliminar el documento, se deberá seleccionar el documento en la sección "Eliminar".

De esta manera quedó construido el sistema generando documentos XML con una estructura diferente a partir de los documentos XML iniciales. Esta sección permite al profesor consultar los documentos con los ejercicios generados, pero también permite eliminar del sistema aquellos documentos que se deseen.

Esta parte del sistema le permite tener un historial de ejercicios generados para un curso y fecha en particular; por lo que si posteriormente desea generar un documento para el mismo curso y desea consultar que ejercicios se plantearon en cursos anteriores puede hacerlo, ya sea para seleccionar los mismos o diferentes ejercicios. Por otro lado, si no desea mantener este historial, puede guardar esa información el tiempo que dura el curso y posteriormente eliminarla.

4.5. Problemas encontrados

En el desarrollo de la aplicación se encontraron algunos problemas:

- La ruta donde se almacenan las imágenes debe de ser fija (es configurable) y debe contar con los permisos de escritura y lectura para poder almacenarlos.
- Se desarrollaron algunas consultas con XQuery debido a que con XPath era complicado obtener la información deseada.
- El parseo del documento con Java fue complicado, debido a que la consulta regresaba el nodo como texto y no por nodos cuando se trataba de obtener los ejercicios que cumplían alguno de los criterios. Por lo que, obtener la información de los atributos fue más simple que obtener la información de los elementos.
- Usar XSL y adecuarlo para el despliegue de los elementos como se requería. Ya que debido a la estructura jerárquica de los elementos, y dado que algunos se podían repetir como hijos de varios elementos, como los elementos texto e imagen, fue un poco complicado lograr que se desplegaran como se esperaba. Los documentos XSL se encuentran en el ápendice B.2.

- Inicialmente se registraba el documento personalizado en la colección **tesis/generados** pero debido a que en ese momento tanto el documento con los ejercicios y el personalizado tenía el mismo elemento raíz al ejecutar las consultas se obtenia información de las dos colecciones. Y se requerían por separado. Por lo que se optó por dejarlos en colecciones totalmente independientes. Finalmente el elemento raíz de ambos documentos fue diferente pero se optó por dejarlos en colecciones independientes para una mejor administración.

- La consulta de fechas es casi transparente, sólo que sigue un formato específico, por lo que si no se pone en el formato año-mes-día (yyyy-MM-dd en Java), el resultado de las consultas será vacío. De tal manera que al capturarse se hace en un formato día-mes-año (dd-MM-yyyy en Java) y al registrarse se cambia el formato para poder posteriormente consultarlo a año-mes-día (yyyy-MM-dd).

- Inicialmente los documentos XML contenían acentos antes de registrarse, pero una vez registrados se hacía con carácteres especiales y la transformación con XSL no se podía llevar a cabo. Se intentó cambiarle la codificación desde el registro pero de cualquier manera se registraba con esos caracteres, por lo que en esta primera versión se decidió no manejar acentos.

- La codificación está basada mucho en el modelo por lo que si cambia o se agrega un elemento hay que modificar la codificación, el esquema XML y muy probablemente el XSL que se utilice para la transformación del documento. En el sistema desarrollado, en este sentido, sólo afectaría en el esquema y en el documento XSL. Y sólo si se agregarán nuevos elementos al nivel del documento raíz y se decidiera darle un tratamiento especial, como desplegar esa información en el listado se tendría que modificar la codificación.

- La generación de resultados aleatorios tampoco fue simple. Debido a que no se encontró manera de recuperar en orden aleatorio los ejercicios, una vez obtenida la información se generó una nueva lista con los resultados y mediante Java se implementó la clase `Collection` del paquete `java.util`, con lo cual se obtuvo una manera de acceder a los elementos de esa lista de forma aleatoria.

- Lamentablemente no hay todavía mucha documentación en cuanto a la base de datos eXist, lo cual también es una limitación para el desarrollo de cualquier sistema.

Conclusiones

Se cumplió con el objetivo principal de desarrollar un sistema que utiliza datos semiestructurados en una base de datos nativa XML con el propósito de conocer sus ventajas y desventajas.

Además se cumplió el objetivo de desarrollar una aplicación útil que permite a los profesores de alguno o varios cursos en particular, almacenar ejercicios capturados en documentos XML en una base de datos nativa XML y posteriormente recuperarlos mediante algunos criterios para la generación de documentos.

Aunque siguen existiendo avances en este tema, todavía no se llega a tener la flexibilidad que se tiene con las bases de datos relacionales. Ya que entre otras cosas, la implementación está basada en el modelo.

Una de las ventajas principales que se encontró al usar bases de datos nativas es que los documentos no requieren cumplir con el mismo esquema para las consultas, es decir, que sí el esquema de los documentos cambia constantemente no es necesario actualizar los documentos para que lo cumplan, dado que lo que se requiere es consultar la misma información.

Sólo si hay un cambio fuerte en el modelo cambia la codificación en el sistema. Aunque en SQL pasa lo mismo, los cambios no son independientes del modelo, pero son más sencillos de ejecutarse que en XML ya que depende de la codificación, si se obtiene el nodo por nombre el cambio no afectaría mucho al código, pero de ser por posición, el cambio podría llegar a ser muy drástico de tal manera que habría que recorrer todas las posiciones.

Por otro lado utilizar una base de datos relacional para almacenar documentos XML provoca perdida de información dado que al almacenarlos de manera jerárquica se descartan comentarios, instrucciones de procesamiento, el orden en que aparecen los elementos, etc. y en una base de datos nativa, toda esa información se registra y de ser necesaria se puede utilizar.

El hecho de que la información esté almacenada de esta manera permite consultar, almacenar y actualizar la información en una sola ubicación física del sistema. Lo cual reduce el costo en tiempo.

Dependiendo de la cantidad de documentos a almacenar y del tamaño de ellos se debe elegir la base de datos. Ya que si son de gran tamaño o son demasiados documentos no se podría utilizar Xindice, por ejemplo. Además depende de la plataforma en la que se desee trabajar, ya que hay bases de datos que no son soportadas en algunas.

El despliegue de la información no es tan transparente aún usando XSL. Se requiere tener gran conocimiento de como elaborar un XSL si se quiere tener una salida compleja. O si el documento XML que se desea transformar tiene una estructura compleja.

Hay bases de datos nativas más potentes que eXist pero desgraciadamente no son accesibles debido al costo. Tamino por ejemplo permite registrar información que no sea necesariamente XML.

Tener la información almacenada en documentos XML permite generar diversos documentos a partir de él con diversos formatos, con diferente información y posiblemente distintos usos, lo cual evita tener información repetida en diversos lugares y contribuye al reuso de la información.

En el proyecto se utilizaron varias de las tecnologías XML tratando de explotar lo existente al máximo. Aunque por otro lado, algunas fueron consecuencia de la tecnología utilizada, por ejemplo XQuery, ya que eXist tiene el soporte para utilizarlo.

A pesar de algunos inconvenientes como los mencionados anteriormente, si se tiene contenido con datos semiestrucurados que administrar la mejor opción es hacerlo en bases de datos XML nativas dado que es un poco más transparente evitando hacer mapeos innecesarios de elementos a campos y/o tablas de una base de datos relacional.

Apéndice A

Diseño del Sistema

A continuación se mostrarán los casos de uso, diagramas de clases y de secuencia de la aplicación desarrollada:

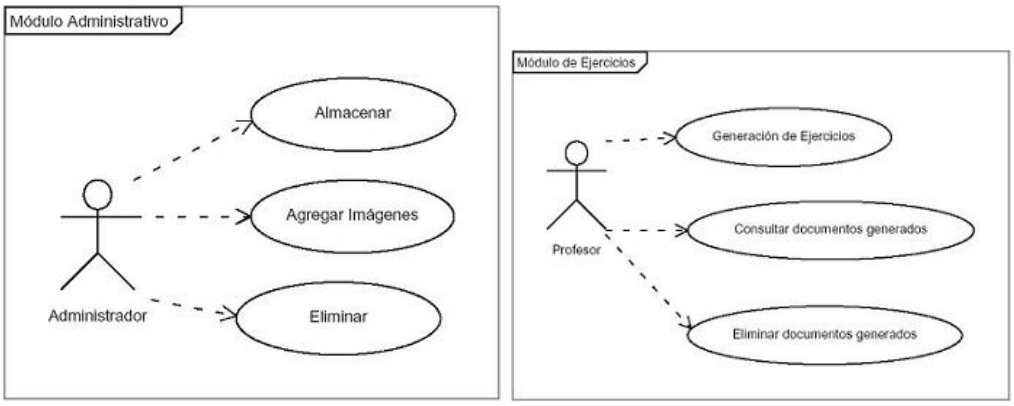

Figura A.1: Casos de uso del sistema.

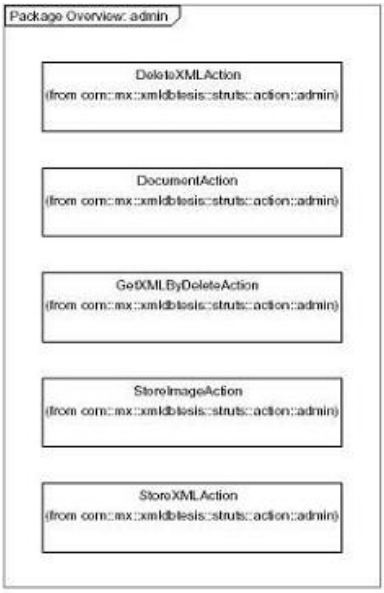

Figura A.2: Diagrama de clases del paquete admin.

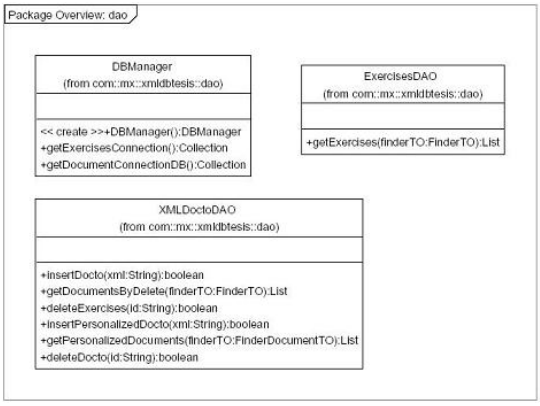

Figura A.3: Diagrama de clases del paquete dao. Clases que acceden a la información de la base de datos.

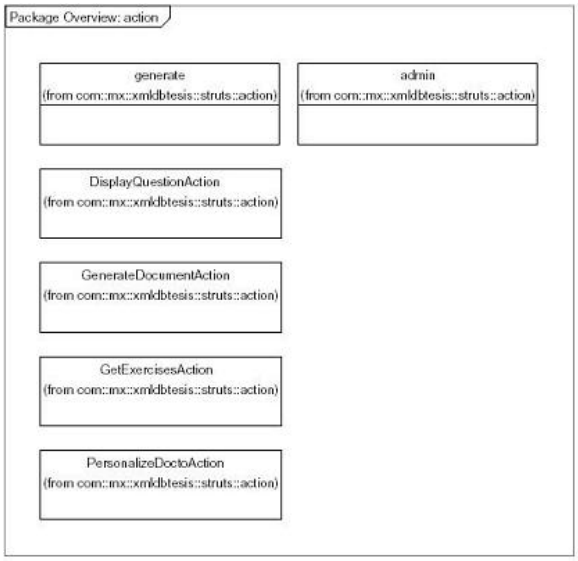

Figura A.4: Diagrama de clases del paquete action.

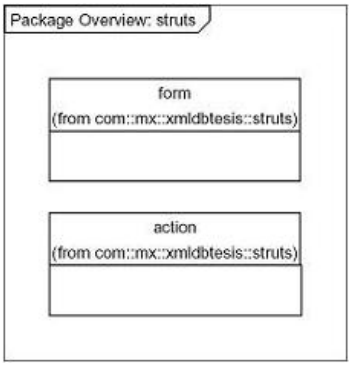

Figura A.5: Diagrama de clases del paquete struts.

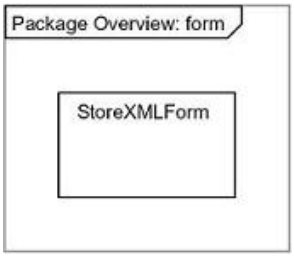

Figura A.6: Diagrama de clases del paquete form.

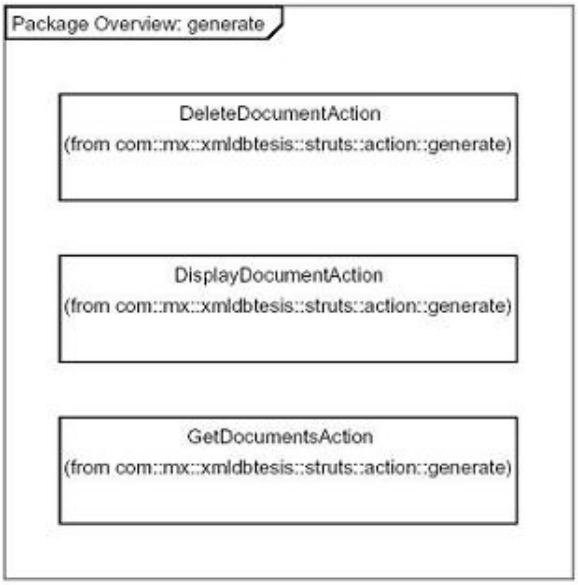

Figura A.7: Diagrama de clases del paquete generate.

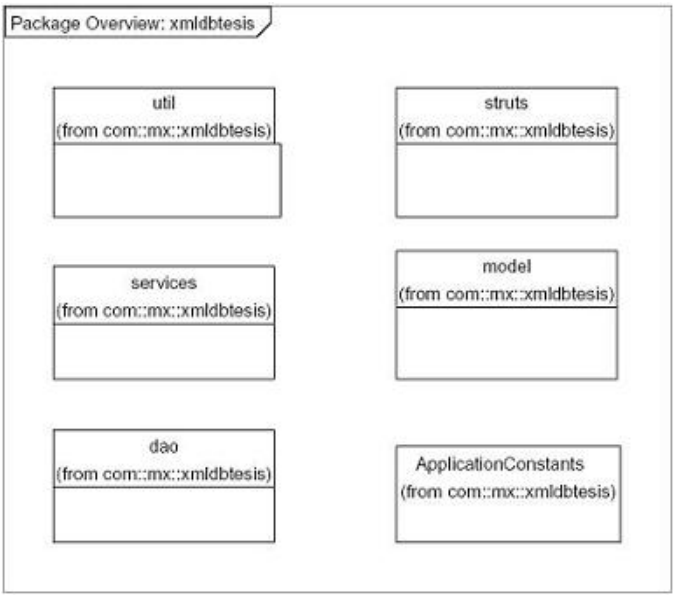

Figura A.8: Diagrama de clases del paquete xmldbtesis.

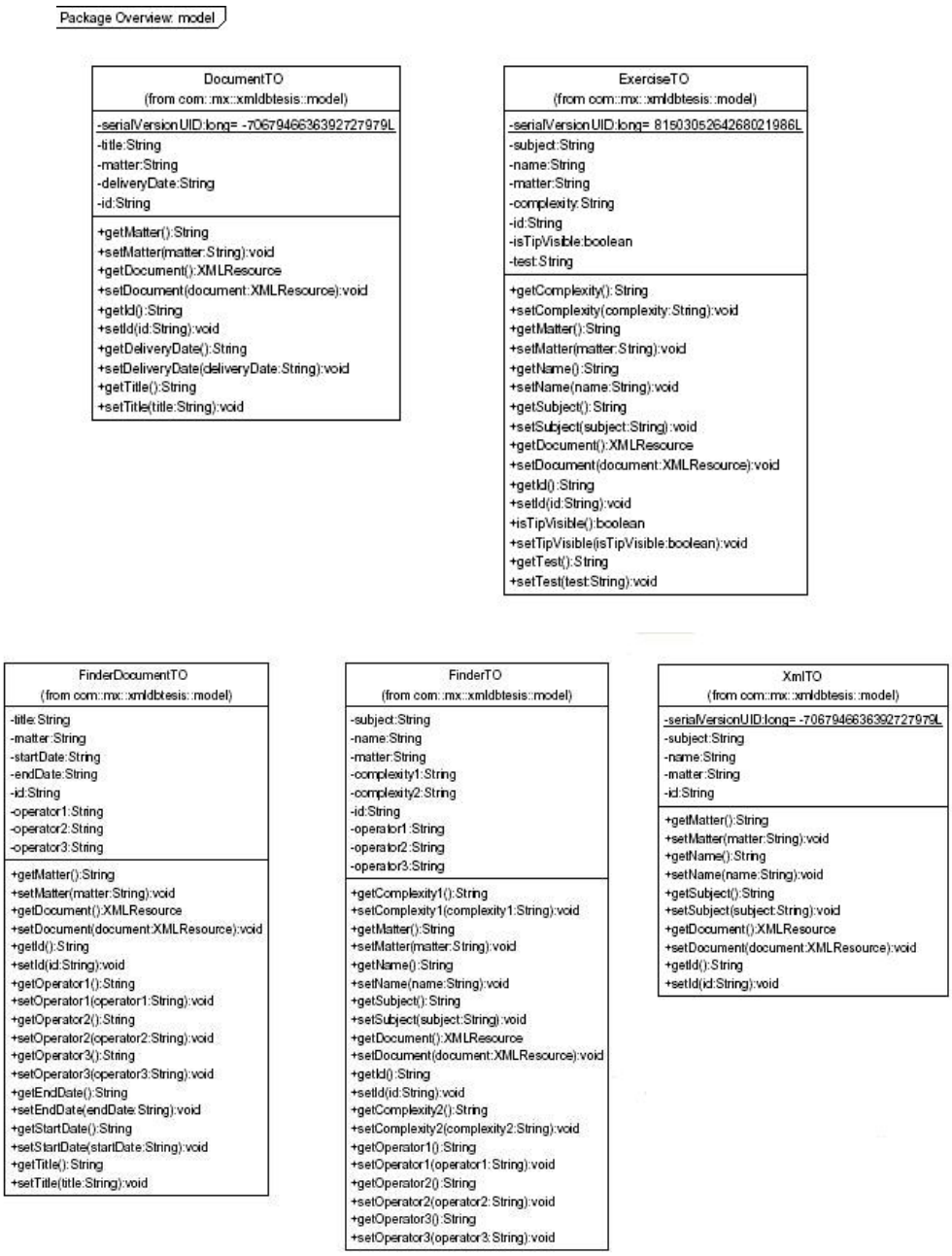

Figura A.9: Diagrama de clases del paquete model. Contiene los objetos que se utilizan en el sistema.

Figura A.10: Diagrama de clases del paquete services.

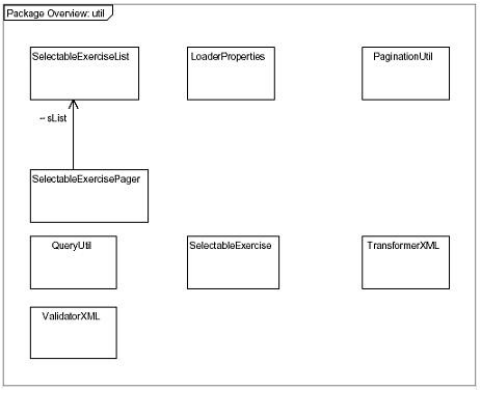

Figura A.11: Diagrama de clases del paquete util. Utilerías utilizadas en el sistema.

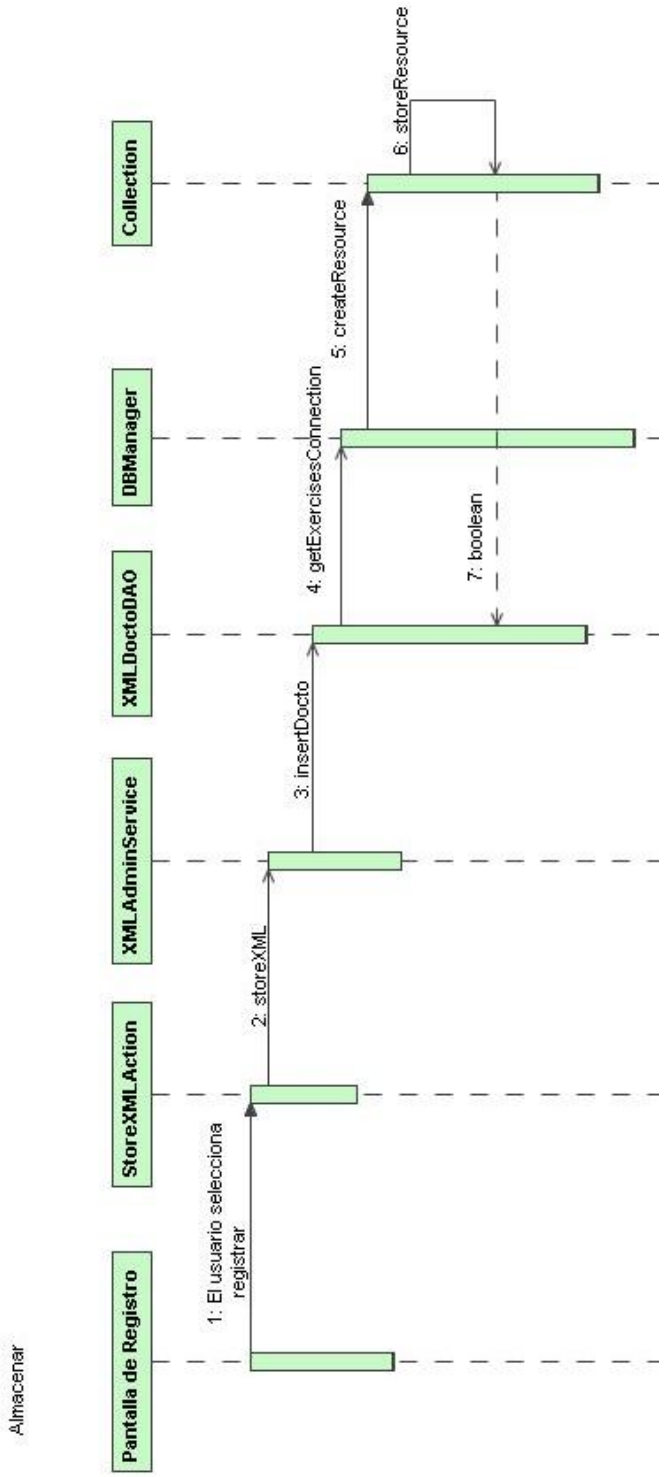

Figura A.12: Diagrama de secuencia de la funcionalidad Almacenar XML.

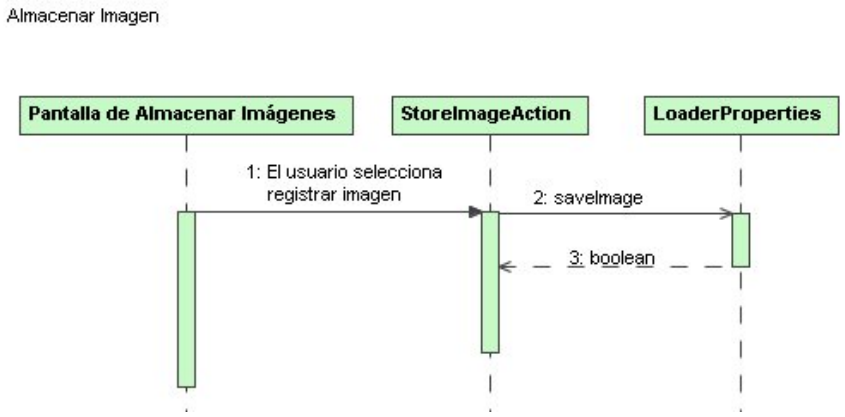

Figura A.13: Diagrama de secuencia de la funcionalidad Agregar Imágenes.

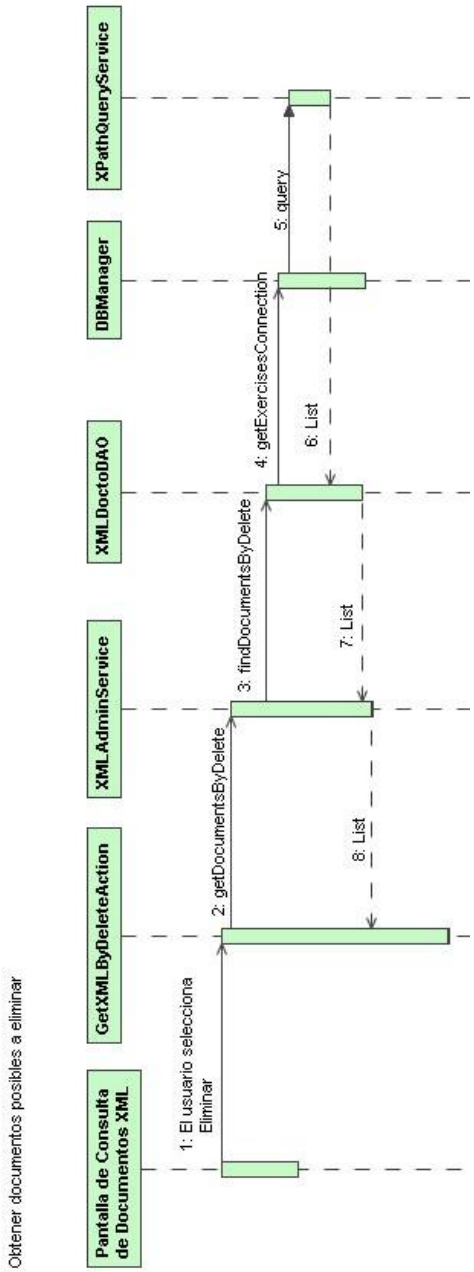

Figura A.14: Diagrama de secuencia de la funcionalidad Consulta de Documentos XML.

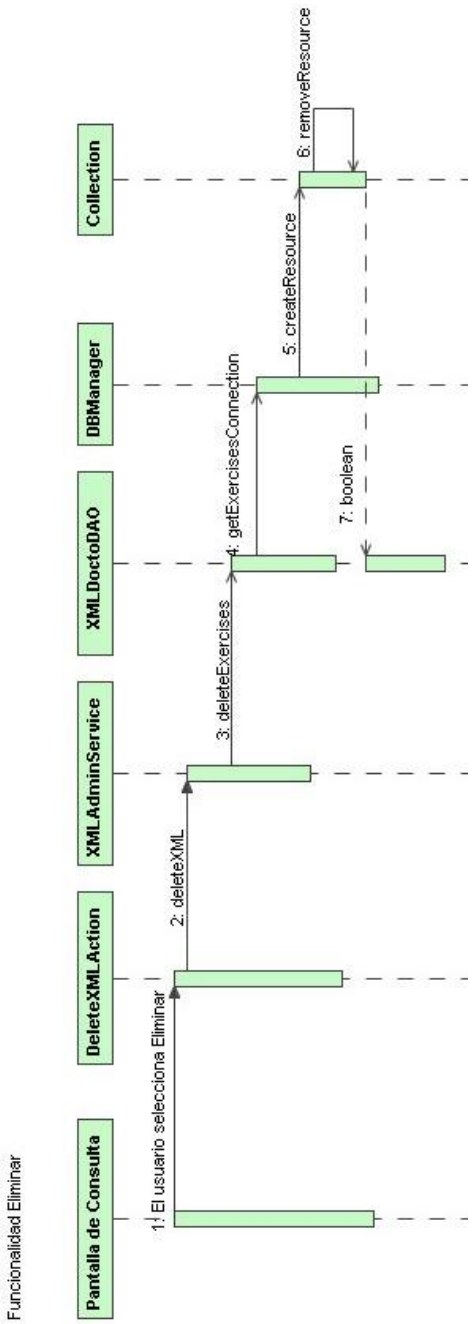

Figura A.15: Diagrama de secuencia de la funcionalidad Eliminar.

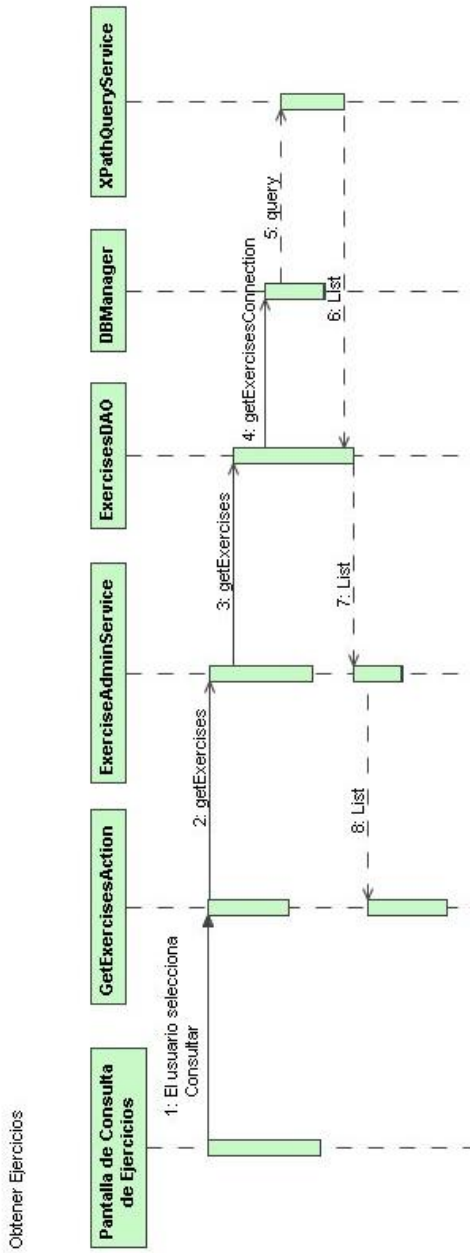

Figura A.16: Diagrama de secuencia de la funcionalidad Consultar ejercicios.

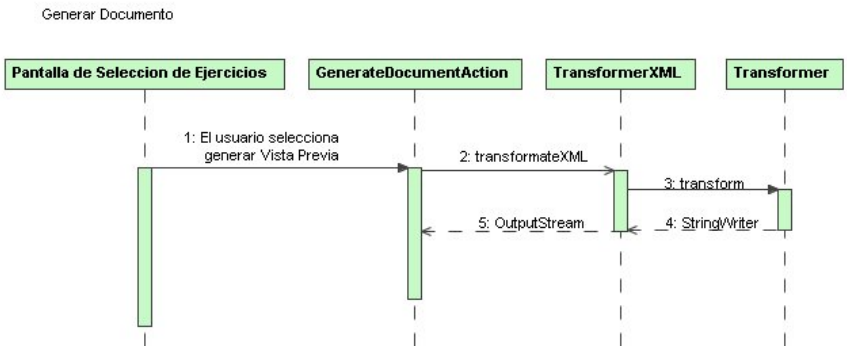

Figura A.17: Diagrama de secuencia de la funcionalidad Generar Documentos.

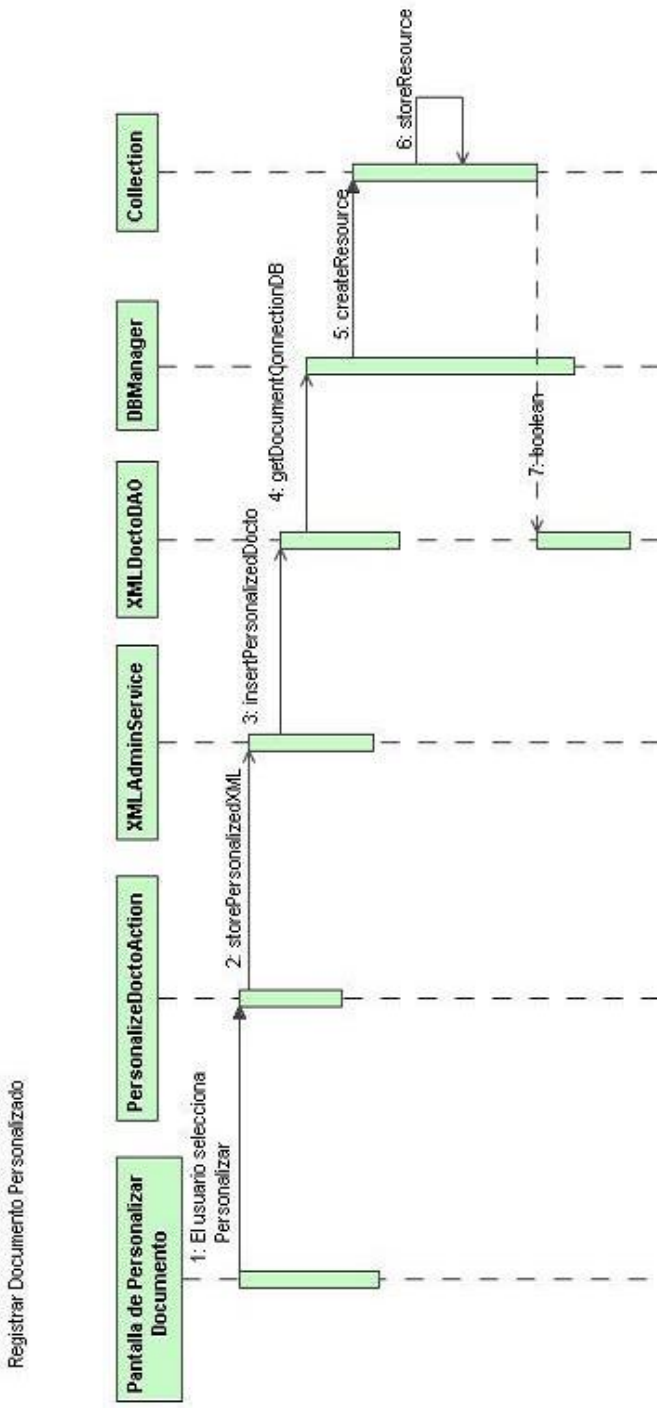

Figura A.18: Diagrama de secuencia de la funcionalidad Personalizar Documentos.

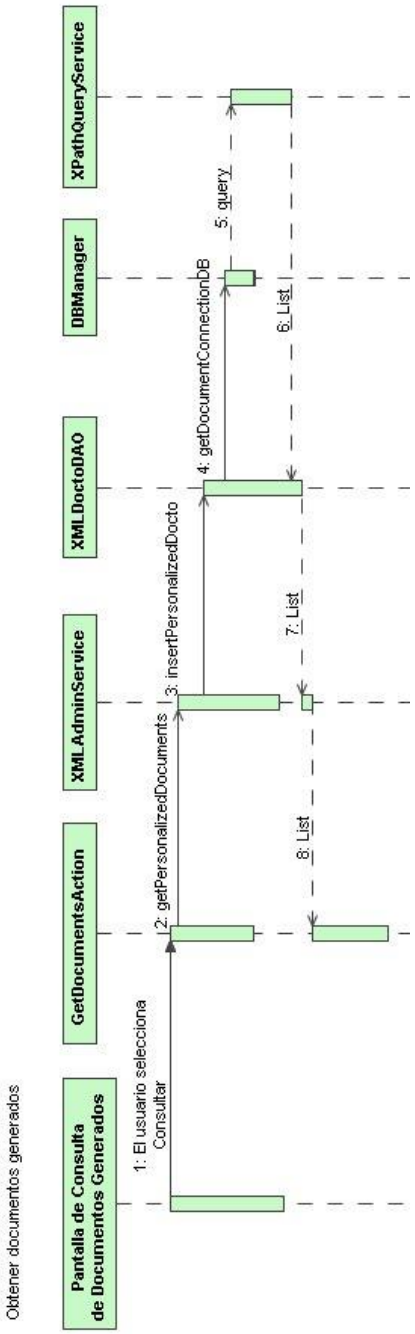

Figura A.19: Diagrama de secuencia de la funcionalidad Consultar Documentos Generados.

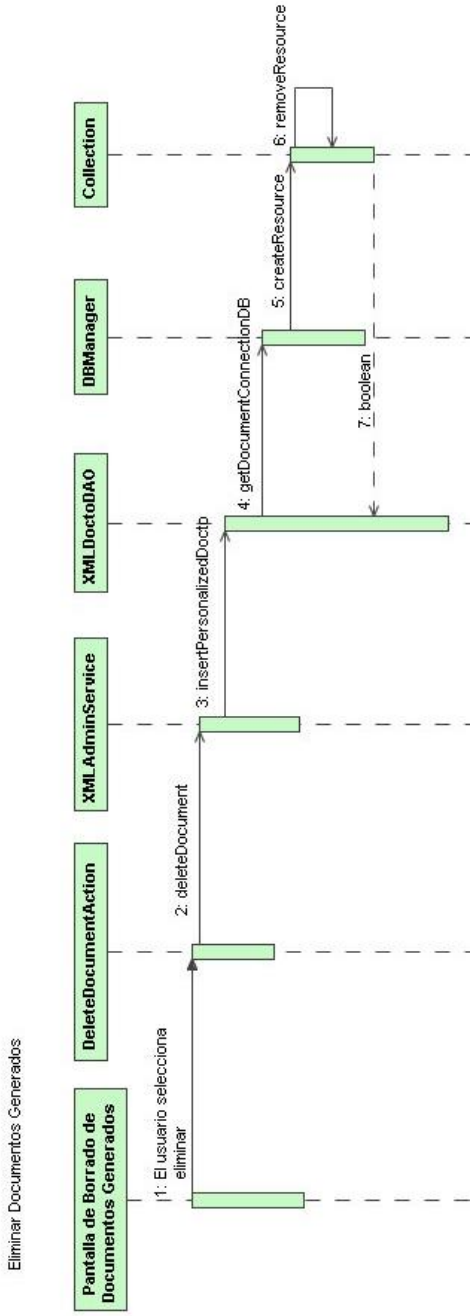

Figura A.20: Diagrama de secuencia de la funcionalidad Eliminar Documentos Generados..

Apéndice B

Documentos

B.1. Esquema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="Ejercicios">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="requisitos" type="xsd:string"/>
        <xsd:element minOccurs="1" maxOccurs="unbounded" name="pregunta"
        type="Ejercicio" nillable="false"/>
      </xsd:sequence>
      <xsd:attribute name="Identificador" type="xsd:string" use="required"/>
      <xsd:attribute name="Materia" type="xsd:string" use="required"/>
      <xsd:attribute name="Tema" type="xsd:string" use="optional"/>
      <xsd:attribute name="Nombre" type="xsd:string"/>
    </xsd:complexType>
</xsd:element>
    <xsd:simpleType name="NivelComplejidad">
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Baja"/>
        <xsd:enumeration value="Media"/>
        <xsd:enumeration value="Alta"/>
      </xsd:restriction>
    </xsd:simpleType>
<xsd:complexType name="Ejercicio">
    <xsd:sequence>
      <xsd:element minOccurs="1" maxOccurs="1" name="Contenido"
      type="TipoEjercicio"/>
      <xsd:element minOccurs="1" maxOccurs="1" name="hint" type="xsd:string"
      nillable="true"/>
      <xsd:element minOccurs="1" maxOccurs="1" name="respuesta" nillable="true"
      type="RespEjercicio"/>
    </xsd:sequence>
```

```
    <xsd:attribute name="tipVisible" type="xsd:boolean"/>
    <xsd:attribute name="Complejidad" type="NivelComplejidad" use="required"/>
</xsd:complexType>
<xsd:complexType name="RespEjercicio">
    <xsd:sequence>
      <xsd:element minOccurs="1" maxOccurs="1" name="Contenido"
      type="TipoEjercicio"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="TipoEjercicio">
    <xsd:sequence>
      <xsd:choice minOccurs="1" maxOccurs="unbounded">
        <xsd:element name="texto" type="xsd:string"/>
        <xsd:element name="imagen" type="xsd:string"/>
        <xsd:element name="opcion" type="TipoOpcion"/>
      </xsd:choice>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="TipoOpcion">
    <xsd:sequence>
      <xsd:choice minOccurs="1" maxOccurs="unbounded">
        <xsd:element name="texto" type="xsd:string"/>
        <xsd:element name="imagen" type="xsd:string"/>
      </xsd:choice>
    </xsd:sequence>
<xsd:attribute name="nombre" type="xsd:string"/>
<xsd:attribute name="esCorrecta" type="xsd:boolean"/>
</xsd:complexType>
</xsd:schema>
```

B.2. XSL

B.2.1. xmldbtesis.xsl

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl= "http://www.w3.org/1999/XSL/Transform" version= "1.0">
<xsl:output method="html" indent="yes" encoding="ISO-8859-1" />
<xsl:param name="image_dir"/>
<xsl:template match="/">
<html>
<body>
<b> Nombre: </b> <xsl:value-of select="Ejercicios/@Nombre"/> <br/>
<b> Tema: </b> <xsl:value-of select="Ejercicios/@Tema"/>   <br/>
<b> Materia: </b> <xsl:value-of select="Ejercicios/@Materia"/><br/>
<xsl:apply-templates select="Ejercicios/requisitos"/>
```

```
<xsl:apply-templates select="Ejercicios"/>
</body>
</html>
</xsl:template>

<xsl:template match="Ejercicios">
<ol>
    <xsl:for-each select="pregunta">
    <li>
      <xsl:for-each select="./Contenido/texto">
        <xsl:apply-templates select="."/>
      </xsl:for-each>
    </li>
    <ol>
    <xsl:for-each select="./Contenido/opcion">
    <li>
        <xsl:value-of select="./@Nombre"/>
        <xsl:value-of select="./texto"/><br/>
    </li>
    </xsl:for-each>
</ol>
<br/>
Complejidad:<xsl:value-of select="./@Complejidad"/>
<br/>
      <xsl:apply-templates select="./hint"/>
      <xsl:apply-templates select="./respuesta"/>
    </xsl:for-each>
</ol>
</xsl:template>

 <!-- Obtiene los requisitos para el tema seleccionado -->
<xsl:template match="requisitos">
     <b> Requisitos:</b> <xsl:value-of select="."/>       <br/>
</xsl:template>
<!-- Texto -->
<xsl:template match="Contenido/texto">
    <xsl:value-of select="."/><br/>
    <xsl:if test="following-sibling::imagen">
      <xsl:apply-templates select="../imagen"/>
    </xsl:if>
</xsl:template>
<xsl:template match="imagen">
   <img>
   <xsl:attribute name="src"><xsl:value-of select="$image_dir"/>
   <xsl:value-of select="." />
   </xsl:attribute>
```

```
   </img><br/>
</xsl:template>
<xsl:template match="respuesta">
Solucion: <br/>
   <xsl:for-each select="./Contenido/texto">
    <xsl:value-of select="."/>
    <xsl:if test="preceding-sibling::imagen">
   <xsl:apply-templates select="../imagen"/>
    </xsl:if>
   </xsl:for-each>
   <xsl:apply-templates select="./Contenido"/>
</xsl:template>
<xsl:template match="respuesta/Contenido">
    <xsl:apply-templates select="imagen"/>
    <ol>
    <xsl:for-each select="opcion">
      <li>
      <xsl:value-of select="./@Nombre"/>
      <xsl:value-of select="./texto"/><br/>
      <xsl:apply-templates select="imagen"/><br/>
      </li>
    </xsl:for-each>
    </ol>
</xsl:template>
</xsl:stylesheet>
```

B.2.2. preguntaxmldb.xsl

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl= "http://www.w3.org/1999/XSL/Transform" version= "1.0">
<xsl:output method="html" indent="yes" encoding="ISO-8859-1" />
<xsl:param name="image_dir"/>
<xsl:template match="/">
<html>
<body>
<ol>
   <xsl:apply-templates select="pregunta"/>
</ol>
</body>

</html>
</xsl:template>

<xsl:template match="pregunta">
<li>
```

```
<xsl:for-each select="./Contenido/texto">
    <xsl:apply-templates select="."/>
</xsl:for-each>
</li>
<ol>
<xsl:for-each select="./Contenido/opcion">
    <li>
      <xsl:value-of select="./@Nombre"/>
      <xsl:value-of select="./texto"/><br/>
    </li>
</xsl:for-each>
</ol>
<br/>
    Complejidad:<xsl:value-of select="./@Complejidad"/>
<br/>
<xsl:apply-templates select="./hint"/>
<br/>
<xsl:apply-templates select="./respuesta"/>
</xsl:template>

 <!-- Obtiene los requisitos para el tema seleccionado -->
<xsl:template match="requisitos">
    <b> Requisitos:</b> <xsl:value-of select="."/>       <br/>
</xsl:template>
<!-- Texto -->
<xsl:template match="Contenido/texto">
<xsl:value-of select="."/><br/>
<xsl:if test="following-sibling::imagen">
    <xsl:apply-templates select="../imagen"/>
</xsl:if>
</xsl:template>

<xsl:template match="imagen">
  <img>
    <xsl:attribute name="src"><xsl:value-of select="$image_dir"/>
    <xsl:value-of select="." />
  </xsl:attribute>
  </img><br/>
</xsl:template>

<xsl:template match="respuesta">
Solucion: <br/>
<xsl:for-each select="./Contenido/texto">
  <xsl:value-of select="."/>
  <xsl:if test="preceding-sibling::imagen">
    <xsl:apply-templates select="../imagen"/>
```

```
  </xsl:if>
</xsl:for-each>
<xsl:apply-templates select="./Contenido"/>
</xsl:template>

<xsl:template match="respuesta/Contenido">
  <xsl:apply-templates select="imagen"/>
  <ol>
  <xsl:for-each select="opcion">
    <li>
      <xsl:value-of select="./@Nombre"/>
      <xsl:value-of select="./texto"/><br/>
      <xsl:apply-templates select="imagen"/><br/>
    </li>
  </xsl:for-each>
  </ol>
</xsl:template>
</xsl:stylesheet>
```

B.2.3. documentxmldb.xsl

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl= "http://www.w3.org/1999/XSL/Transform" version= "1.0">
<xsl:output method="html" indent="yes" encoding="ISO-8859-1" />
<xsl:param name="image_dir"/>
<xsl:template match="/">
<html>
<head>
  <script language="JavaScript">
  function onBefore()
  {
   noprint.style.visibility = 'hidden';
  }
  function onAfter()
  {
   noprint.style.visibility = 'visible';
  }
  function imprimir() {
        onBefore();
   window.focus();
   window.print();
   onAfter();
   window.focus();
  }
 </script>
```

```
 </head>
 <body>
 <div id="noprint" align="right">
   <img src="./images/printer2.jpg" width="4%" height="5%" onClick='imprimir()'/>
 </div>
 <b> Tema: </b> <xsl:value-of select="Documento/@Tema"/>   <br/>
 <b> Materia: </b> <xsl:value-of select="Documento/@Materia"/><br/>
    <xsl:apply-templates select="Documento/titulo"/>
    <xsl:apply-templates select="Documento/fechaEntrega"/>
    <xsl:apply-templates select="Documento/nota"/>
    <xsl:apply-templates select="Documento"/>
</body>

</html>
</xsl:template>
<xsl:template match="titulo">
  <h2><xsl:value-of select="."/></h2>
</xsl:template>
<xsl:template match="nota">
  <xsl:value-of select="."/><br/>
</xsl:template>
<xsl:template match="fechaEntrega">
  <h3>
    Fecha de entrega:
    <xsl:call-template name="FormatDate">
      <xsl:with-param name="DateTime" select="."/>
    </xsl:call-template>
  </h3>
</xsl:template>
<xsl:template match="Documento">
<ol>
<xsl:for-each select="pregunta">
  <li>
    <xsl:for-each select="./Contenido/texto">
      <xsl:apply-templates select="."/>
    </xsl:for-each>
  </li>
  <ol>
    <xsl:for-each select="./Contenido/opcion">
      <li>
      <xsl:value-of select="./@Nombre"/>
      <xsl:value-of select="./texto"/><br/>
      </li>
    </xsl:for-each>
  </ol>
  <br/>
```

```
   Complejidad:<xsl:value-of select="./@Complejidad"/>
  <br/>
  <xsl:apply-templates select="./hint"/>
  <br/>
  <xsl:apply-templates select="./respuesta"/>
 </xsl:for-each>
</ol>
</xsl:template>

 <!-- Obtiene los requisitos para el tema seleccionado -->
<xsl:template match="requisitos">
  <b> Requisitos:</b> <xsl:value-of select="."/>       <br/>
</xsl:template>
<!-- Texto -->
<xsl:template match="Contenido/texto">
  <xsl:value-of select="."/><br/>
    <xsl:if test="following-sibling::imagen">
      <xsl:apply-templates select="../imagen"/>
    </xsl:if>
</xsl:template>

<xsl:template match="imagen">
  <img>
    <xsl:attribute name="src"><xsl:value-of select="$image_dir"/>
    <xsl:value-of select="." />
    </xsl:attribute>
  </img><br/>
</xsl:template>
<xsl:template match="respuesta">
Solucion: <br/>
  <xsl:for-each select="./Contenido/texto">
    <xsl:value-of select="."/>
    <xsl:if test="preceding-sibling::imagen">
      <xsl:apply-templates select="../imagen"/>
    </xsl:if>
  </xsl:for-each>
  <xsl:apply-templates select="./Contenido"/>
</xsl:template>
<xsl:template match="respuesta/Contenido">
<xsl:apply-templates select="imagen"/>
<ol>
  <xsl:for-each select="opcion">
    <li>
      <xsl:value-of select="./@Nombre"/>
      <xsl:value-of select="./texto"/><br/>
      <xsl:apply-templates select="imagen"/><br/>
```

```
    </li>
  </xsl:for-each>
</ol>
</xsl:template>

<xsl:template name="FormatDate">
  <xsl:param name="DateTime" />
    <xsl:variable name="year">
      <xsl:value-of select="substring($DateTime,1,4)" />
    </xsl:variable>
    <xsl:variable name="year-temp">
      <xsl:value-of select="substring-after($DateTime,'-')" />
    </xsl:variable>
    <xsl:variable name="month">
      <xsl:value-of select="substring-before($year-temp,'-')" />
    </xsl:variable>
    <xsl:variable name="day-temp">
      <xsl:value-of select="substring-after($year-temp,'-')" />
    </xsl:variable>
    <xsl:variable name="day">
      <xsl:value-of select="substring($day-temp,1,2)" />
    </xsl:variable>
    <xsl:value-of select="$day"/> de
    <xsl:choose>
      <xsl:when test="$month = '01'">Enero</xsl:when>
      <xsl:when test="$month = '02'">Febrero</xsl:when>
      <xsl:when test="$month = '03'">Marzo</xsl:when>
      <xsl:when test="$month = '04'">Abril</xsl:when>
      <xsl:when test="$month = '05'">Mayo</xsl:when>
      <xsl:when test="$month = '06'">Junio</xsl:when>
      <xsl:when test="$month = '07'">Julio</xsl:when>
      <xsl:when test="$month = '08'">Agosto</xsl:when>
      <xsl:when test="$month = '09'">Septiembre</xsl:when>
      <xsl:when test="$month = '10'">Octubre</xsl:when>
      <xsl:when test="$month = '11'">Noviembre</xsl:when>
      <xsl:when test="$month = '12'">Diciembre</xsl:when>
    </xsl:choose>
       del
    <xsl:value-of select="$year"/>
</xsl:template>
</xsl:stylesheet>
```

Bibliografía

[1] Benoit Marchal,XML con Ejemplos, Primera Edición. Pearson Educación México,2001

[2] Pitts, Natanya. XML, Editorial Anaya Multimedia,1999.

[3] Chang,Beng. Scardina, Mark et al. ORACLE XML HandBook. Oracle Press.

[4] Chelsea Valentine y Chris Minnick. XHTML. Pearson Educación, S.A. Madrid,2001.

[5] Standefer, Robert. Kauffman, Morgan. Enterprise XML.

[6] Hoque, Reaz. Kauffman, Morgan. XML for Real Programmers.

[7] Castro, Elizabeth. Guia de Aprendizaje XML,Prentice Hall,Madrid, 2001

[8] Donald E. Eastlake III and Kitty Niles. Secure XML.Primera Edición, Addison-Wesley,,2002

[9] Michael Morrison. Teach Yourself XML. Segunda Edición. Editorial Sams. 2002.

[10] Akmal B. Chaudhri. Awais Rashid. Roberto Zicari, XML Data Management: Native XML and XML-Enabled Database Systems. Addison-Wesley. 2003.

[11] Minera, Francisco. XML: La guía total del programador. Primera Edición. Editorial MP Ediciones. 2006.

[12] Martín, Gregorio. Martín Benitez, Isabel. Curso de XML: Introducción al lenguaje de la Web. Primera Edición. Editorial Pearson Prentice Hall. Madrid,2005

[13] http://www.rpbourret.com/xml/XMLAndDatabases.htm

[14] http://www.oracle.com

[15] http://technet.oracle.com/tech/xml

[16] http://www.w3.org/XML/

[17] http://www.xml.com/

[18] http://xml.oreilly.com/

[19] http://www.webspedite.com/oracle/

[20] http://xml.coverpages.org/xmlAndDatabases.html

[21] http://es.wikipedia.org/wiki/XML

[22] http://en.wikipedia.org/wiki/XML

[23] http://eprints.rclis.org/archive/00000222/01/apuntes_xml.PDF

[24] http://www.di.uniovi.es/ tuya/is/descarga/lab/XML/NotasXML.pdf

[25] http://es.wikipedia.org/wiki/DTD

[26] http://petra.euitio.uniovi.es/ labra/cursos/ext03/XML.pdf

[27] http://petra.euitio.uniovi.es/ labra/cursos/ext02/xml.pdf

[28] http://recursos.dotnetclubs.com/sevilla/Aportaciones/IntroduccionAXML.pdf

[29] http://mipagina.euskaltel.es/gsagarduy/rec-xml-es.html

[30] http://geneura.ugr.es/ maribel/xml/introduccion/index.shtml

[31] http://www.desarrolloweb.com/manuales/18/

[32] http://es.wikipedia.org/wiki/XPath

[33] http://geneura.ugr.es/ victor/cursillos/xml/XPath/

[34] http://www.sidar.org/recur/desdi/traduc/es/xml/xpath.html

[35] http://www.w3schools.com/xml/xml_whatis.asp

[36] http://html.conclase.net/w3c/xml-names-es/

[37] http://csgrs6k1.uwaterloo.ca/ dmg/tutorial/xml/

[38] http://www.ulpgc.es/otros/tutoriales/xml/Estructura.html

[39] www.forpas.us.es/aula/xml/doc/03.Estructura%20de%20los%20documentos%20DTD.ppt

[40] http://www.asptutor.com/zip/xml.pdf

[41] http://lml.ls.fi.upm.es/ mcollado/xml/xml.html

[42] http://www.spanish-translator-services.com/espanol/t/Namespaces_in_XML_1.1_SP.htm

[43] http://flanagan.ugr.es/xml/xml.htm

[44] http://www.w3.org/XML/

[45] http://www.hipertexto.info/documentos/xsl.htm

[46] http://www.w3schools.com/dtd/dtd_elements.asp

[47] http://www.di.uniovi.es/ labra/cursos/Doc06UPSAM/PPS/XML2.pps#334, 15,XML Schema Objetivos de Diseño

[48] http://www.hipertexto.info/documentos/dtds.htm

[49] http://www.unirioja.es/cu/arjaime/Temas/08.XML.pdf

[50] http://es.wikipedia.org/wiki/XML_Schema

[51] http://www.forpas.us.es/aula/xml/doc/04.Estructura%20de%20los%20 documentos%20W3C%20Esquemas.ppt#278,14,Esquemas XML - facetas

[52] http://es.wikipedia.org/wiki/Validaci

[53] http://www.di.uniovi.es/ labra/cursos/ver06/pres/XML2.pdf

[54] http://www.hipertexto.info/documentos/dom.htm

[55] http://petra.euitio.uniovi.es/ labra/cursos/ext02/saxDom.PDF

[56] http://es.wikipedia.org/wiki/XSL

[57] http://es.wikipedia.org/wiki/XPath

[58] http://www.di.uniovi.es/ labra/cursos/ver04/pres/XSLTXPath.pdf

[59] http://es.wikipedia.org/wiki/XPointer

[60] http://www.1x4x9.info/files/jstl/html/online-chunked/ar01s05.html

[61] http://es.wikipedia.org/wiki/XLink

[62] http://geneura.ugr.es/ victor/cursillos/xml/XLink/

[63] http://es.wikipedia.org/wiki/XSL

[64] http://es.wikipedia.org/wiki/XSL-FO

[65] http://www.w3.org/TR/xquery/

[66] http://axel.deri.ie/ axepol/teaching/ri2007/jorge.pdf

[67] http://www.w3.org/TR/2007/REC-xquery-20070123/

[68] http://www.lsi.us.es/docs/informes/LSI-2005-02.pdf

[69] http://en.wikipedia.org/wiki/XQuery

[70] http://es.geocities.com/lenguajesde_recuperacion/xquery.htm

[71] http://www.brics.dk/ amoeller/XML/querying/languages.html

[72] http://exist.sourceforge.net/xquery.html

[73] http://www.datypic.com/services/xquery/intro.html

[74] http://kybele.escet.urjc.es/documentos/BDA/BDXML06/XMLyBD.PDF

[75] http://www.rpbourret.com/xml/UseCases.htm

[76] http://exist.sourceforge.net/

[77] http://kybele.escet.urjc.es/documentos/BDA/BDXML06/Bases%20de%20 Datos%20XML.pdf

[78] http://es.wikipedia.org/wiki/Bases_de_datos_nativas_XML

[79] http://en.wikipedia.org/wiki/XML_database

[80] http://xml.apache.org/xindice/

[81] http://www.xmlstarterkit.com/xmlzone/WP_XML_Databases_E.pdf

[82] http://geneura.ugr.es/ jmerelo/asignaturas/AAP/AAP-Taller-2.mhtml#T2:bd

[83] http://2006.xmlconference.org/programme/tutorials/15.html

[84] http://www.brics.dk/ amoeller/XML/querying/databases.html

[85] lsdis.cs.uga.edu/ aleman/summer2001/csci8990/xml_databases.ppt

[86] http://www.javapassion.com/xml/XMLandDataBase.pdf

[87] http://www.consist.com.ar/paraguay/tamino/index.htm

[88] http://documentation.softwareag.com/crossvision/ins441/overview.htm

[89] http://exist.sourceforge.net/xmlprague06.html

[90] http://www.xml.gov/presentations/softwareag3/tamino.ppt

[91] http://www.ibm.com/developerworks/xml/library/wa-xindice.html

[92] http://www.oreillynet.com/pub/d/1127

[93] http://xml.apache.org/xindice/#About+Apache+Xindice

[94] http://www.idealliance.org/proceedings/xml05/slides/bourret.ppt#976,16, The%20data

[95] http://xml.apache.org/xindice/guide-developer.html

[96] http://www.rpbourret.com/xml/ProdsMiddleware.htm

[97] http://xml.apache.org/xindice/guide-user.html

[98] http://www.laflecha.net/canales/empresas/oracle-11g-la-nueva-base-de-datos-de-oracle/

[99] http://www.oracle.com/technology/tech/xml/xmldb/Current/xmldb_11gr1_overview.pdf

[100] http://kybele.escet.urjc.es/documentos/BDA/Examenes/Examen_Sep_2004_Solucion.pdf

[101] http://kybele.escet.urjc.es/MIGJRV/GIJRV/Datos

[102] http://www.oracle.com/technology/tech/xml/xmldb/index.html

[103] http://www.siliconnews.es/es/silicon/news/2007/09/27/ya-ha-lanzado-oracle-11g

[104] http://www.aristote.asso.fr/Presentations/CEA-EDF-2003/Conferences/RonaldBourret/NativeXMLDatabases.ppt#337,80,Questions?