Temas Generales para la preparación de la Oposición al Cuerpo Superior de Sistemas y Tecnologías de la Información de la Administración del Estado.

Cuerpo Superior de Estadísticos del Estado
Especialidad de Estadística-Ciencia de Datos.

Almacenamiento y modelos de datos

Tema 9. XML y tecnologías relacionadas.

AUTOR: **Alexandra Vlad**

Asociación Profesional de Cuerpos Superiores de Sistemas y Tecnologías de la Información de las Administraciones Públicas

Creación: Junio 2021

ÍNDICE

1 INTRODUCCIÓN AL XML 3

2 TECNOLOGÍAS RELACIONADAS CON EL LENGUAJE XML 11

3 XML SCHEMA 17

4 XML AND DATABASE 25

5 RESUMEN ESQUEMÁTICO 31

6 GLOSARIO 33

7 BIBLIOGRAFÍA BÁSICA.....34

1 Introducción al XML

La mayoría de los documentos en la Web se almacenan y transmiten actualmente en formato HTML. Una de las ventajas de HTML es su sencillez, lo que permite que sea utilizado por una amplia variedad de usuarios. Sin embargo, podría decirse que su sencillez también es una de sus debilidades, con la creciente necesidad por parte de los usuarios que quieren etiquetas para simplificar algunas tareas y hacer que los documentos HTML sean más atractivos y dinámicos. En un intento por satisfacer esta demanda, los proveedores han introducido algunas etiquetas HTML específicas del navegador. Sin embargo, esto dificulta el desarrollo de documentos web sofisticados y de amplia visibilidad. Para evitar esta división, el W3C ha producido un nuevo estándar llamado eXtensible Markup Language (XML), que podría preservar la independencia de uso general que hace que HTML sea portable y poderoso. XML 1.0 (segunda edición) se convirtió en una recomendación del W3C en octubre de 2000 (W3C, 2000b) y XML 1.1, con soporte Unicode 3, se convirtieron en una recomendación del W3C en agosto de 2006 (W3C, 2006a). XML 1.0 (quinta edición) se convirtió en una recomendación del W3C en noviembre de 2008 y suele ser la versión recomendad al menos que se requiera una característica especial de XML 1.1.

XML es un meta-lenguaje (un lenguaje que describe otros lenguajes) que permite a los diseñadores crear sus propias etiquetas personalizadas para ofrecer funcionalidades que HTML no presenta.

XML es una versión simplificada de SGML (Standard Generalized Markup Language o Lenguaje de marcado generalizado estándar, en español), diseñado especialmente para documentos web. Por ejemplo, XML soporta enlaces que apuntan a varios documentos, a diferencia de un enlace HTML que solo puede referenciar un documento.

SGML es un sistema que define tipos de documentos estructurados y lenguajes de marcado para representar instancias de esos tipos de documentos (ISO, 1996). SGML ha sido la forma estándar de los proveedores independientes para mantener repositorios de documentación estructurada durante más de una década. SGML permite separar un documento en dos partes de forma lógica: una parte que define la estructura del documento, y otra que contiene el texto en sí mismo. La definición de la estructura se denomina DTD (Document Type Definition, o Definición del Tipo de Documento, en español). Por el hecho de que los documentos tienen una estructura definida por separado, y que los autores tienen la capacidad de definir estructuras personalizadas, SGML proporciona un sistema de gestión de documentos extremadamente poderoso. Sin embargo, SGML no se ha adoptado ampliamente debido a su complejidad inherente.

XML intenta proporcionar una función similar a SGML, pero es menos complejo y, al mismo tiempo, compatible con la red. De manera significativa, XML conserva las ventajas clave de SGML de extensibilidad, estructura y validación. Dado que XML es una forma simplificada de SGML, cualquier sistema SGML totalmente compatible podrá leer documentos XML (aunque lo contrario no es cierto). Sin embargo, XML no está diseñado para reemplazar SGML. Del mismo modo, XML no está pensado como un reemplazo de HTML, que también se basa en SGML. En cambio, XML está diseñado para complementar HTML al permitir que se intercambien diferentes tipos de datos a través de la Web. De hecho, el uso de XML no se limita solo al marcado de texto, sino que la extensibilidad significa que XML también podría aplicarse al marcado de sonido o marcado de imagen. Tres lenguajes populares creados con XML son MathML (Mathematics Markup Language, en español lenguaje de marcado de matemáticas), SMIL (Synchronized Multimedia Integration Language, en español lenguaje de integración multimedia sincronizado) y CML (Chemistry Markup Language, en español lenguaje de marcado de química), entre muchos otros.

El trabajo de XML comenzó con XML 1.0 que fue ratificado formalmente por W3C a fines de 1998, y se puede decir que XML ha impactado muchos aspectos de TI, incluyendo interfaces gráficas, sistemas embebidos, sistemas distribuidos y administración de bases de datos. Por ejemplo, dado que XML describe la estructura de los datos, puede convertirse en un mecanismo útil para definir la estructura de bases de datos y fuentes de datos heterogéneas. Con la capacidad de definir un esquema de base de datos completo, XML podría potencialmente usarse para tomar el contenido de un esquema de Oracle, por ejemplo, y traducirlo a un esquema de Informix o Sybase.

XML ya se está convirtiendo en el estándar de facto para la comunicación de datos dentro de la industria del software y está reemplazando rápidamente a los sistemas EDI (Electronic Data Interchange, en español intercambio electrónico de datos) como el medio principal para el intercambio de datos entre empresas.

Ventajas de XML

Algunas de las ventajas de usar XML en la web se enumeran a continuación:

- *Sencillez*. XML es un estándar relativamente sencillo, alrededor de 65 páginas. Fue diseñado como un lenguaje basado en texto que es legible por humanos y razonablemente claro.
- *Estándar abierto e independiente de la plataforma/proveedor*. XML es independiente de la plataforma y del proveedor, y es una forma restringida de SGML, un estándar ISO. También está basado en ISO 10646, el conjunto de caracteres Unicode, por lo que tiene soporte integrado para textos de todos los alfabetos del mundo, incluido un método para indicar qué idioma y codificación se está utilizando.
- *Extensibilidad*. A diferencia de HTML, XML es extensible, lo que permite a los usuarios definir sus propias etiquetas para alcanzar los requisitos particulares de su aplicación.
- *Reutilizar*. Asimismo, la extensibilidad permite crear librerías de etiquetas XML una sola vez y reutilizarlas en muchas aplicaciones.
- *Separación de contenido y presentación*. XML separa el contenido de un documento de la forma en la que se presentará el documento (por ejemplo, dentro de un navegador). Esto facilita una vista personalizada de los datos - los datos se pueden mostrar al usuario a través del navegador, donde se pueden presentar de manera personalizada, quizás en función de factores como las preferencias o la configuración del usuario. De la misma manera que a veces se hace referencia a Java como un lenguaje "escribir una vez, ejecutar en cualquier lugar" ("write once/run anywhere"), XML se conoce como un lenguaje "escribir una vez, publicar en cualquier lugar" ("write once, publish anywhere"), con características como hojas de estilo que permiten que el mismo documento XML se publique de diferentes formas utilizando una variedad de formatos y medios.
- *Equilibrio de carga mejorado*. Los datos se pueden enviar al navegador para ejecutarse en el cliente, descargando al servidor de su ejecución y logrando así un mejor equilibrio de carga.
- *Soporte para la integración de datos de múltiples fuentes*. La capacidad de integrar datos de múltiples fuentes heterogéneas es extremadamente difícil y requiere mucho tiempo. Sin embargo, XML permite que los datos de diferentes fuentes se combinen más fácilmente. Se pueden usar agentes de software para integrar datos de bases de datos de back-end y otras aplicaciones, que luego se pueden entregar a otros clientes o servidores para su posterior procesamiento o presentación.
- *Capacidad para describir datos de una amplia variedad de aplicaciones*. Como XML es extensible, se puede utilizar para describir datos contenidos en una amplia variedad de aplicaciones. Además, como XML hace que los datos se autodescriban, los datos se pueden recibir y procesar sin la necesidad de una descripción incorporada de los datos.
- *Motores de búsqueda más avanzados*. En la actualidad, los motores de búsqueda trabajan con la información contenida en las metaetiquetas HTML o con la proximidad de una palabra clave a otra. Con XML, los motores de búsqueda podrán simplemente parsear las etiquetas con descripción.
- *Nuevas oportunidades*. Quizás una de las grandes ventajas de XML es la gran cantidad de oportunidades que ofrece esta nueva tecnología.

Descripción general de XML

Declaración XML

Los documentos XML comienzan con una declaración XML opcional, que en el ejemplo de abajo indica la versión de XML utilizada para crear el documento (1.0), el sistema de codificación utilizado (UTF-8

para Unicode) y si se hace referencia o no a declaraciones de marcado externas (standalone = "no" indica que el documento XML se tiene que validar contra un documento DTD separado). La segunda y tercera líneas del documento XML del ejemplo que se muestra a continuación, se relacionan con hojas de estilo y DTD.

```
<?xml version= “1.0” encoding= “UTF-8” standalone= “no”?>
<?xml:stylesheet type = “text/xsl” href = “staff_list.xsl”?>
<!DOCTYPE STAFFLIST SYSTEM “staff_list.dtd”>
<STAFFLIST>
        <STAFF branchNo=”B005”>
                <STAFFNO>SL21</STAFFNO>
                        <NAME>
                                <FNAME>John</FNAME><LNAME>White</LNAME>
                        </NAME>
                <POSITION>Manager</POSITION>
                <DOB>1945-10-01</DOB>
                <SALARY>30000</SALARY>
        </STAFF>
        <STAFF branchNo=”B003”>
                <STAFFNO>SG37</STAFFNO>
                        <NAME>
                                <FNAME>Ann</FNAME><LNAME>Beech</LNAME>
                        </NAME>
                <POSITION>Assistant</POSITION>
                <SALARY>12000</SALARY>
        </STAFF>

</STAFFLIST>
```

Elementos

Los elementos o etiquetas son la forma más común de marcado. El primer elemento debe ser un elemento raíz, que puede contener otros (sub) elementos. Un documento XML debe tener un elemento raíz, en nuestro ejemplo <STAFFLIST>. Un elemento comienza con una etiqueta de inicio (por ejemplo, <STAFF>) y termina con una etiqueta de fin (por ejemplo, </STAFF>). Los elementos XML distinguen entre mayúsculas y minúsculas, por lo que un elemento <STAFF> sería diferente de un elemento <staff> (tenga en cuenta que este no es el caso de HTML). Un elemento puede estar vacío, en cuyo caso se puede abreviar como <EMPTYELEMENT/>. Los elementos deben estar correctamente anidados como se muestra a continuación:

```
<STAFF>
        <NAME>
                <FNAME>John</FNAME> <LNAME> Blanco </LNAME>
        </NAME>
</STAFF>
```

En este caso, el elemento NAME está completamente anidado dentro del elemento STAFF y los elementos FNAME y LNAME están anidados dentro del elemento NAME.

Atributos

Los atributos son pares de nombre-valor que contienen información descriptiva sobre un elemento. El atributo se coloca dentro de la etiqueta de inicio, después del nombre del elemento correspondiente, con el valor del atributo entre comillas. Por ejemplo, hemos optado por representar la sucursal en la que trabaja el empleado utilizando un atributo branchNo dentro del elemento STAFF:

<STAFF branchNo = "B005">

Igualmente, bien podríamos haber representado la sucursal como un subelemento de STAFF. Si tuviéramos que representar el sexo del empleado, podríamos usar un atributo de un elemento vacío, por ejemplo:

<SEX gender = "M" />

Un atributo determinado solo puede aparecer una vez dentro de una etiqueta, mientras que los subelementos con la misma etiqueta pueden repetirse.

Referencias de entidad

Las entidades tienen tres propósitos principales:

- como accesos directos a texto que se repite con frecuencia o para incluir el contenido de archivos externos;
- para insertar caracteres Unicode arbitrarios en el texto (por ejemplo, para representar caracteres que no se pueden escribir directamente desde el teclado);
- para distinguir los caracteres reservados del contenido. Por ejemplo, el símbolo menor (<) significa el comienzo de la etiqueta inicial o final de un elemento. Para diferenciar este símbolo del contenido real, XML ha introducido la entidad lt, que se reemplaza por el símbolo “<”.

Cada entidad debe tener un nombre único y su uso en un documento XML se denomina referencia de entidad. Una referencia de entidad comienza con un ampersand (&) y termina con un punto y coma (;), por ejemplo, & lt ;.

Comentarios

Los comentarios están encerrados entre etiquetas <!- y -> y pueden contener cualquier dato excepto el carácter "-". Los comentarios se pueden colocar entre marcas en cualquier sitio dentro del documento XML, aunque un procesador XML no está obligado a pasar comentarios a una aplicación.

Secciones CDATA e instrucciones de procesamiento

Una sección CDATA indica al procesador XML que ignore los caracteres de marcado y pase el texto adjunto directamente a la aplicación sin interpretación. Las instrucciones de procesamiento también se pueden utilizar para proporcionar información a una aplicación. Una instrucción de procesamiento tiene el formato <?name pidata?>, Donde *name* identifica la instrucción de procesamiento para la aplicación. Dado que las instrucciones son específicas de la aplicación, un documento XML puede tener múltiples

instrucciones de procesamiento que le indiquen a diferentes aplicaciones que hagan cosas similares, pero quizás de diferentes maneras.

Orden de los elementos

En XML, los elementos están ordenados. Por lo tanto, en XML los siguientes dos fragmentos con elementos FNAME y LNAME transpuestos son diferentes:

```
<NAME>
        <FNAME> John </FNAME>
        <LNAME> White </LNAME>
</NAME>

<NAME>
        <LNAME> White </LNAME>
        <FNAME> John </FNAME>
</NAME>
```

Por el contrario, los atributos en XML no están ordenados, por lo que los dos elementos XML siguientes son iguales:

```
<NAME FNAME = "John" LNAME = "White" />
<NAME LNAME = "White" FNAME = "John" />
```

Document Type Definitions (DTDs)

Un DTD define la sintaxis válida de un documento XML.

La Definición de Tipo de Documento (DTD) define la sintaxis válida de un documento XML al enumerar los nombres de los elementos que pueden aparecer en el documento, qué elementos pueden aparecer en combinación con otros, cómo se pueden anidar los elementos, qué atributos están disponibles para cada tipo de elemento, etc. El término *vocabulario* se usa a veces para referirse a los elementos usados en una aplicación particular. La gramática se especifica utilizando EBNF (Extended Backus-Naur Form), no sintaxis XML. Aunque el DTD es opcional, se recomienda para la conformidad del documento.

Para continuar con el ejemplo de antes del empelado, a continuación, se muestra un posible DTD para el documento XML de antes.

```
<!ELEMENT STAFFLIST (STAFF)*>
<!ELEMENT STAFF (NAME, POSITION, DOB?, SALARY)>
<!ELEMENT NAME (FNAME, LNAME)>
<!ELEMENT FNAME (#PCDATA)>
<!ELEMENT LNAME (#PCDATA)>
<!ELEMENT POSITION (#PCDATA)>
<!ELEMENT DOB (#PCDATA)>
<!ELEMENT SALARY (#PCDATA)>
<!ATTLIST STAFF branchNo CDATA #IMPLIED>
```

El DTD puede ser un fichero externo independiente del XML, aunque también puede aparecer incrustado dentro del propio XML. Existen 4 tipos de declaraciones de DTD:

- Declaraciones de elementos, que indican los elementos permitidos en un documento y su contenido (que puede ser simplemente texto u otros elementos).
- Declaraciones de atributos, que indican los atributos permitidos en cada elemento y el tipo o valores permitidos de cada elemento.
- Declaraciones de entidades
- Declaraciones de notaciones

Declaraciones de elementos

Las declaraciones de tipo de elemento identifican las reglas para los elementos que pueden aparecer en el documento XML. En el ejemplo anterior hemos especificado la siguiente regla para el elemento STAFFLIST:

<!ELEMENT STAFFLIST (STAFF)*>

que establece que el elemento STAFFLIST consta de cero o más elementos STAFF. Las opciones de repetición son:

- asterisco (*) indica cero o más ocurrencias para un elemento;
- más (+) indica una o más ocurrencias de un elemento;
- signo de interrogación (?) indica cero ocurrencias o exactamente una ocurrencia para un elemento.

Un nombre sin puntuación calificativa debe aparecer exactamente una vez. Las comas entre los nombres de los elementos indican que deben aparecer en ese mismo orden; si se omiten las comas, los elementos pueden aparecer en cualquier orden. Por ejemplo, hemos especificado la siguiente regla para el elemento STAFF:

<!ELEMENT STAFF (NAME, POSITION, DOB ?, SALARY)>

que establece que el elemento STAFF consta de un elemento NAME, un elemento POSITION, un elemento DOB opcional y un elemento SALARY, en este orden. Las declaraciones para FNAME, LNAME, POSITION, DOB y SALARY y todos los demás elementos utilizados en un modelo de contenido también deben estar presentes para que un procesador XML verifique la validez del documento. Todos estos elementos básicos se han declarado utilizando el símbolo especial #PCDATA para indicar datos de caracteres analizados sintácticamente.

Declaraciones de atributos

Indican los atributos permitidos en cada elemento y el tipo o valores permitidos de cada elemento.

Una declaración de atributos tiene 3 partes: un nombre, un tipo y un valor predeterminado opcional.

Hay 7 tipos de atributos:

- CDATA: el atributo contiene caracteres.
- ID: el valor del atributo (no el nombre) debe ser único y no se puede repetir en otros elementos o atributos.

- IDREF o IDREFS: el valor del atributo debe coincidir con el valor del atributo ID de otro elemento. Un atributo IDREFS puede contener múltiples valores IDREF separados por espacios en blanco.
- ENTITY o ENTITIES: el valor del atributo es alguna de las entidades de una lista de entidades definida en el DTD. De nuevo, un atributo ENTITIES puede contener multiples valores ENTITY separados por espacios en blanco.
- NMTOKEN o NMTONKENS: el atributo sólo contiene letras, dígitos, y los caracteres punto ".", guion "-", subrayado "_" y dos puntos ":".
- NOTATION: el valor del atributo es el nombre de una notación.
- valores: el atributo sólo puede contener uno de los términos de una lista. La lista se escribe entre paréntesis, con los términos separados por una barra vertical "|".

Declaraciones de entidades

Una entidad consiste en un nombre y su valor (son similares a las constantes en los lenguajes de programación). Con algunas excepciones, el procesador XML sustituye las referencias a entidades por sus valores antes de procesar el documento. Una vez definida la entidad, se puede utilizar en el documento escribiendo una referencia a la entidad, que empieza con el carácter "&", sigue con el nombre de la entidad y termina con ";". (es decir, &nombreEntidad;)

Las entidades pueden ser internas o externas y tanto unas como otras pueden ser generales o paramétricas.

Las declaraciones de entidades internas (generales) siguen la siguiente sintaxis:

<!ENTITY nombreEntidad "valorEntidad">

En las declaraciones de entidades externas (generales) se distinguen dos casos:

- La entidad hace referencia a un fichero de texto y en ese caso la entidad se sustituye por el contenido del archivo. La entidad puede ser una entidad de sistema, con la siguiente sintaxis:

  <!ENTITY nombreEntidad SYSTEM "uri">

  o puede ser una entidad pública, con la siguiente sintaxis:

  <!ENTITY nombreEntidad PUBLIC "fpi" "uri">

- La entidad hace referencia a un fichero que no es de texto (por ejemplo, una imagen) y en ese caso la entidad no se sustituye por el contenido del archivo.
  La entidad puede ser una entidad de sistema, con la siguiente sintaxis:

  <!ENTITY nombreEntidad SYSTEM "uri" NDATA tipo>

  o puede ser una entidad pública, con la siguiente sintaxis:

  <!ENTITY nombreEntidad PUBLIC "fpi" "uri" NDATA tipo>

En todos estos casos:

- "nombreEntidad" es el nombre de la entidad.
- "valorEntidad" es el valor de la entidad.
- "uri" es el camino (absoluto o relativo) hasta un archivo.
- "tipo" es el tipo de archivo (gif, jpg, etc).
- "fpi" es un identificador público formal (Formal Public Identifier).

Las declaraciones de entidades paramétricas siguen la misma sintaxis que las generales, pero llevan el carácter "%" antes del nombre de la entidad. Por ejemplo:

<!ENTITY % nombreEntidad "valorEntidad">
<!ENTITY % nombreEntidad SYSTEM "uri">
<!ENTITY % nombreEntidad SYSTEM "uri" NDATA tipo>

La diferencia entre entidades generales y paramétricas es que las entidades paramétricas se sustituyen por su valor en todo el documento (incluso en la propia declaración de tipo de documento) mientras que las generales no se sustituyen en la declaración de tipo de documento.

Declaraciones de notaciones

Las notaciones se usan en XML para definir las entidades externas que no va a analizar el procesador XML (aunque sí lo hará la aplicación que trate un documento). Para hacer referencia estas entidades no se utiliza la notación habitual (&nombreEntidad;), sino que se utiliza el nombre de la entidad directamente.

El procesamiento de entidades externas no analizadas es responsabilidad de la aplicación. Alguna información sobre el formato interno de la entidad debe declararse después del identificador que indica la ubicación de la entidad; por ejemplo:

Validez del documento

La especificación XML proporciona dos niveles de procesamiento de documentos: bien formado y válido. Un procesador sin validación asegura que un documento XML esté bien formado antes de pasar la información del documento a la aplicación. Un documento XML que se ajusta a las reglas estructurales y de notación de XML se considera bien formado. Entre otros, los documentos XML bien formados deben cumplir las siguientes reglas:

- el documento debe comenzar con la declaración XML; por ejemplo, <? xml versión "1.0"?>;
- todos los elementos deben estar contenidos dentro de un elemento raíz;
- los elementos deben estar anidados en una estructura de árbol sin ninguna superposición;
- todos los elementos no vacíos deben tener una etiqueta de inicio y una etiqueta de fin.

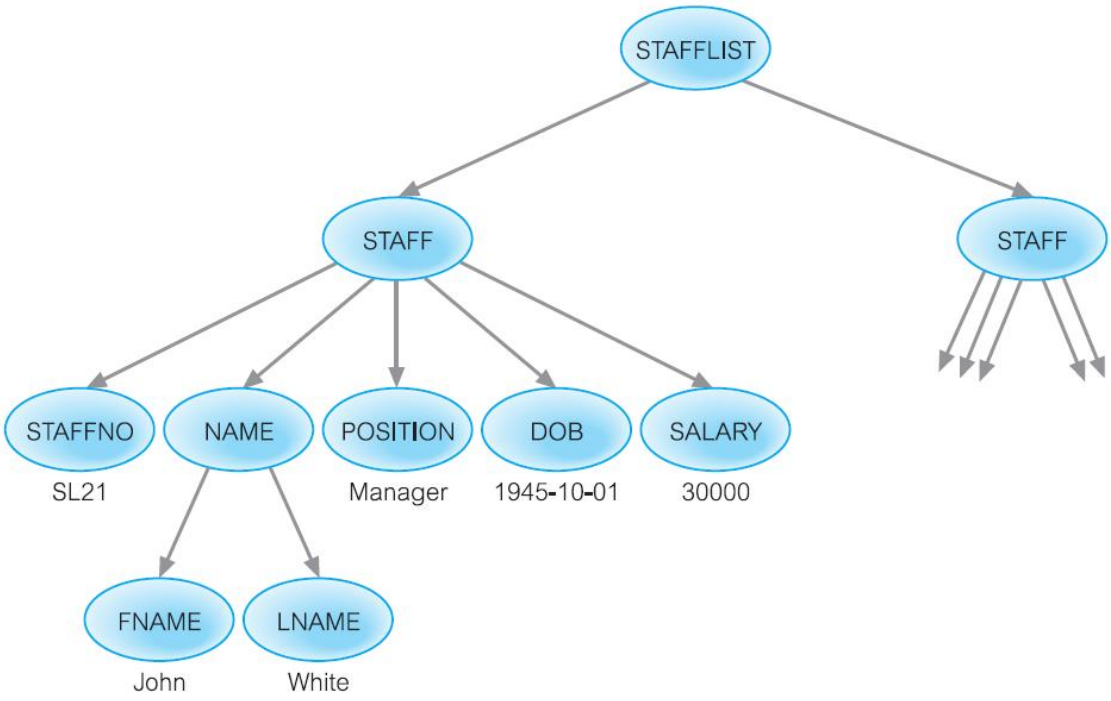
Un procesador de validación no solo verificará que un documento XML esté bien formado, sino que también se ajuste a un DTD, en cuyo caso el documento XML se considera válido. Como mencionamos anteriormente, la DTD puede estar contenida dentro del documento XML o referenciada desde él. El W3C ahora ha propuesto una alternativa más expresiva a la DTD, llamada XML Schema.

2 Tecnologías relacionadas con el lenguaje XML

Interfaces DOM y SAX

Las API XML generalmente se dividen en dos categorías: basadas en árboles y basadas en eventos. DOM (Document Object Model) es una API basada en árbol para XML que proporciona una vista de los datos orientada a objetos. La API fue creada por el W3C y describe un conjunto de interfaces independientes de la plataforma y el lenguaje que pueden representar cualquier documento XML o HTML bien formado. DOM crea una representación en memoria del documento XML y proporciona clases y métodos para permitir que una aplicación navegue y procese el árbol. DOM define una interfaz Node, con subclases Element, Attribute y Character-Data. La interfaz Node tiene métodos para acceder a los componentes de un nodo, como parentNode, que devuelve el nodo principal, y childNodes, que devuelve un conjunto de nodos secundarios. En general, la interfaz DOM es más útil para realizar manipulaciones estructurales del árbol XML, como añadir o eliminar elementos y reordenar elementos.

A continuación, se muestra una representación de un documento XML como una estructura de árbol.



SAX (Simple API for XML) es una API para XML basada en eventos y de acceso en serie, que utiliza devoluciones de llamada para informar eventos de análisis a la aplicación. Por ejemplo, hay eventos para elementos de inicio y finalización. La aplicación maneja estos eventos a través de controladores de eventos personalizados. A diferencia de las APIs basadas en árboles, las APIS basadas en eventos no crean una representación de árbol en memoria del documento XML. Esta API fue en realidad resultado de la colaboración en la lista de correo XML-DEV, más que un producto de W3C.

Namespaces (Espacios de nombres)

Los espacios de nombres permiten calificar los nombres de los elementos y las relaciones en los documentos XML para evitar colisiones de nombres para los elementos que tienen el mismo nombre, pero están definidos en diferentes vocabularios. Esto permite mezclar etiquetas de múltiples espacios de nombres, lo cual es esencial si los datos provienen de varias fuentes. Nuevamente, los espacios de nombres están cubiertos por una recomendación del W3C (W3C, 1999b, 2004b). Para lograr la unicidad, los elementos y atributos reciben nombres únicos globalmente usando una referencia URI. Por ejemplo, el siguiente fragmento de documento utiliza dos espacios de nombres diferentes declarados en el elemento raíz. El primero ("http://www.dreamhome.co.uk/branch5/") actúa como un espacio de nombres predeterminado, por lo que se asume que los elementos no calificados provienen de este espacio de nombres. El segundo espacio de nombres ("http://www.dreamhome.co.uk/HQ/") recibe un nombre (hq) que se utiliza posteriormente para prefijar el elemento SALARY para indicar de dónde procede este elemento:

```
<STAFFLIST xmlns = "http://www.dreamhome.co.uk/branch5/"
xmlns:hq = "http://www.dreamhome.co.uk/HQ/">
<STAFF branchNo = "B005">
<STAFFNO>SL21</STAFFNO>
. . .
<hq:SALARY>30000</hq:SALARY>
</STAFF>
</STAFFLIST>
```

XSL y XSLT

XSL (eXtensible Stylesheet Language) se refiere a una familia de lenguajes que se utilizan para transformar y representar documentos XML. EL grupo de trabajo W3C XSL elaboró un borrador de especificación bajo el nombre XSL, que finalmente se dividió en tres partes:

- Transformaciones XSL (XSLT): un lenguaje XML para transformar documentos XML.
- Objetos de formato XSL (XSL-FO): es un lenguaje XML que especifica el formato visual de un documento.
- XML Path Language (XPath): es un lenguaje que permite construir expresiones que recorren y procesan un documento XML. Es un lenguaje no XML utilizado pos XSLT, y también disponible para su uso en contextos no XSLT, para procesar las partes de un documento XML.

XSL a veces se usa para referirse a XSL-FO.

En HTML, el estilo predeterminado está integrado en los navegadores porque el conjunto de etiquetas para HTML está predefinido y fijo. La especificación de hoja de estilo en cascada (CSS) permite al desarrollador proporcionar una representación alternativa para las etiquetas. CSS también se puede utilizar para representar un documento XML en un navegador, pero no tiene la capacidad de realizar alteraciones estructurales en un documento. XSL-FO es una recomendación formal del W3C que se ha creado específicamente para definir cómo se representan los datos de un documento XML y para definir cómo un documento XML se puede transformar en otro documento (W3C, 2001ª; 2006b). Es similar a CSS aunque más potente.

XSLT (Transformaciones XSL) (W3C, 2007a) es un lenguaje tanto en el sentido de marcado como en el de programación, ya que proporciona un mecanismo para transformar la estructura XML en otra estructura ya sea XML, HTML o cualquier otro formato basado en números o texto (como por ejemplo SQL). La capacidad principal de XSLT es cambiar las estructuras subyacentes en lugar de simplemente las representaciones de medios de esas estructuras, como es el caso de CSS.

XSLT es importante porque proporciona un mecanismo para cambiar dinámicamente la vista de un documento y para filtrar datos. También es lo suficientemente robusto como para codificar reglas de negocio y puede generar gráficos (no solo documentos) a partir de datos. Incluso puede manejar la comunicación con servidores, especialmente junto con módulos de scripting que se pueden integrar en XSLT, y puede generar los mensajes correspondientes dentro del propio cuerpo de XSLT.

XPath (XML Path Language)

XPath es un lenguaje declarativo para XML que proporciona una sintaxis simple para procesar partes de un documento XML (W3C, 1999c, 2007b). Fue diseñado para usarse con XSLT (para coincidencia de patrones) y XPointer (para direccionamiento). Con XPath, las colecciones de elementos se pueden recuperar especificando una ruta similar a un directorio, con cero o más condiciones colocadas en la ruta. XPath utiliza una sintaxis compacta basada en cadenas, en lugar de una sintaxis estructural basada en elementos XML, lo que permite que las expresiones XPath se utilicen tanto en los atributos XML como en las URIs.

XPath trata un documento XML como un árbol lógico (ordenado) con nodos para cada elemento, atributo, texto, instrucción de procesamiento, comentario, espacio de nombres y raíz. La base del mecanismo de direccionamiento es el nodo de contexto (un punto de partida) y la ruta de ubicación, que describe una ruta desde un punto dentro de un documento XML a otro, proporcionando una forma de direccionar los elementos de un documento XML. XPointer se puede utilizar para especificar una ruta absoluta o una ruta relativa. Una ruta de ubicación se compone de una serie de *pasos* unidos con "/", que tiene la misma función que "/" en una ruta de un directorio (de la cual las rutas de ubicación derivan su nombre). Cada "/" se mueve hacia abajo en el árbol desde el paso anterior.

Cada *paso* consta de una *base* y unos *predicados* opcionales donde una base consta de un *eje*, que especifica la dirección en la que debe continuar la navegación, y un nodo test.

Un *nodo test* identifica un tipo de nodo en el documento (generalmente el nombre de un elemento, pero también puede ser una función como text () para nodos de texto o simplemente node () para cualquier tipo de nodo). XPath define 13 tipos de ejes, incluidos padre, ancestro (arriba), hijo, descendiente (abajo), precedente,etc..

Un *predicado* aparece entre corchetes después de la base. Cuando un elemento contiene más de un subelemento, se puede seleccionar un subelemento usando [position () = positionNumber], con positionNumber comenzando desde 1. XPath proporciona una sintaxis abreviada y no abreviada.

XPointer (XML Pointer Language)

XPointer proporciona acceso a los valores de atributos o al contenido de los elementos en cualquier lugar dentro de un documento XML (W3C, 2000d,2003c).

XPointer se divide en cuatro especificaciones:

- XPointer Framework, el cual forma la base para identificar fragmentos XML;
- XPointer element(), el cual es un esquema de direccionamiento de elementos posicionales;
- XPoniter xmlns, un esquema para espacios de nombre.
- XPointer xpointer(), un esquema para el direccionamiento basado en XPath.

Un XPointer es básicamente una expresión XPath contenida en una URI. Entre otras cosas, con XPath podemos vincular secciones de texto, seleccionar elementos o atributos específicos y navegar entre elementos. También permite seleccionar información contenida en más de un conjunto de nodos, que XPath no lo permite.

A demás de definir nodos, XPointer también define puntos y rangos, que combinados con los nodos crean ubicaciones. Un punto es una posición dentro de un documento XML y un rango representa toda la estructura XML y el contenido entre un punto de inicio y un punto final, cada uno de los cuales puede ubicarse en el medio de un nodo.

XLink (XML Linking Language)

XLink permite insertar elementos en los documentos XML para crear y describir enlaces entre recursos (W3C, 2001b;2010a). Utiliza la sintaxis XML para crear estructuras que puedan describir enlaces, de forma similar a los hipervínculos unidireccionales simples de HTML, así como enlaces más sofisticados. Hay dos tipos de XLink: simple y extendido. Un enlace simple conecta una fuente a un recurso de destino; un enlace extendido conecta cualquier número de recursos. Además, permite guardar los enlaces una base de datos de enlaces independiente (llamada linkbase). Esto proporciona una forma de independencia de ubicación: incluso si los enlaces cambian, los documentos XML originales permanecen sin cambios y solo es necesario actualizar la base de datos.

XHTML

XHTML (eXtensible HTML) 1.0 es una reformulación de HTML 4.01 en XML 1.0 y pretende ser la siguiente generación de HTML (W3C, 2002a). Es básicamente una versión más estricta y limpia de HTML. Por ejemplo:

- Las etiquetas y atributos deben estar en minúsculas;
- Todos los elementos XHTML deben tener una etiqueta de fin;
- Los valores de los atributos se deben entrecomillar y no se permite la minimización;
- El atributo ID reemplaza el atributo name;
- Los documentos deben ajustarse a las reglas XML.

XHTML 1.1 se desarrolló para poder utilizarse en dispositivos pequeños que no podían admitir todas las funcionalidades XHTML y la especificación se dividió en módulos con funcionalidad limitada. Los dispositivos pequeños podrían reducir su complejidad al admitir solo algunos de los módulos (W3C, 2010b). Los módulos son:

- XHTML Basic: contiene solamente los elementos XHTML básicos (como la estructura de texto, imágenes, formularios, tablas) (W3C, 2010c).
- XML Events 2: Esta especificación define tres módulos: Eventos XML para definir eventos y sus características; Controladores XML para definir mapeos entre eventos y acciones; y XML Scripting para ayudar a definir funciones para dar soporte a los controladores. Estos módulos funcionan en conjunto para proporcionar lenguajes XML con la capacidad de integrar de manera uniforme los oyentes eventos (event listeners) y los controladores de eventos asociados con los oyentes de eventos de DOM. (W3C, 2010d)
- XML Print: diseñado para imprimir desde dispositivos móviles en impresoras de bajo coste que imprimen desde arriba abajo y de izquierda a derecha, sin búfer de impresión (W3C, 2010e).

SOAP (Simple Object Access Protocol)

SOAP es un protocolo de mensajes basado en XML que define un conjunto de reglas para estructurar mensajes (W3C, 2007c). El protocolo se puede utilizar para mensajería unidireccional simple, pero también es útil para realizar llamadas a procedimientos remotos (RPC) – del estilo diálogos petición-respuesta. SOAP no está vinculado a ningún sistema operativo o lenguaje de programación en particular ni está vinculado a ningún protocolo de transporte en particular, aunque HTTP es popular. Esta independencia hace que SOAP sea un componente importante para el desarrollo de servicios web. Además, una ventaja importante de SOAP es que la mayoría de los cortafuegos permiten que HTTP pase directamente, lo que facilita los intercambios de datos SOAP punto a punto (aunque un administrador del sistema podría bloquear selectivamente las solicitudes SOAP).

Un mensaje SOAP es un documento XML normal que contiene los siguientes elementos:

- Un elemento Sobre (Envelope) obligatorio que identifica el documento XML como un mensaje SOAP.
- Un elemento de encabezado (Header) opcional que contiene información específica de la aplicación, como por ejemplo información de autenticación o de pago. También tiene tres atributos que especifican quién debe procesar el mensaje, si el procesamiento es opcional u obligatorio, y reglas de codificación que describen los tipos de datos para la aplicación.

- Un elemento de encabezado del cuerpo (Body Header) obligatorio que contiene información de llamada y respuesta.
- Un elemento Fault opcional que proporciona información sobre los errores que ocurrieron durante el procesamiento del mensaje.

WSDL (Web Services Description Language)

WSDL es un protocolo basado en XML para definir un servicio web. Especifica la ubicación de un servicio, las operaciones que ofrece el servicio, los mensajes (SOAP) involucrados y el protocolo de comunicaciones utilizado para hablar con el servicio. La notación que utiliza un fichero WSDL para describir los formatos de mensaje se basa normalmente en el estándar XML Schema, lo que lo hace neutral tanto en cuanto al lenguaje como a la plataforma. Los programadores o, de manera más general, las herramientas de desarrollo automatizadas pueden crear ficheros WSDL para describir un servicio y pueden hacer que la descripción esté disponible en la Web. Los programadores del lado del cliente y las herramientas de desarrollo pueden utilizar descripciones WSDL publicadas para obtener información sobre los servicios web disponibles y, posteriormente, construir y crear proxies o plantillas de programas que accedan a estos servicios.

WSDL 2.0 describe un servicio web en dos partes: una parte abstracta y una parte concreta (W3C, 2007). A nivel abstracto, WSDL describe un servicio web en términos de los mensajes que envía y recibe; los mensajes se describen independientemente de un formato de específico utilizado un sistema tipo, normalmente un esquema XML:

- Un patrón de intercambio de mensajes que identifica la secuencia y cardinalidad de los mensajes enviados y/o recibidos, así como a quién se envían y/o desde donde se reciben lógicamente.
- Una operación vincula un patrón de intercambio de mensajes con uno o más mensajes.
- Una interfaz agrupa las operaciones sin ningún compromiso de transporte o formato de cable.

UDDI (Universal Discovery, Description, and Integration)

La especificación UDDI define un servicio web basado en SOAP para localizar descripciones de protocolos de servicios web con formato WSDL. Básicamente, describe un registro electrónico online que sirve como páginas amarillas electrónicas, proporcionando una estructura de información donde varias empresas se registran a sí mismas y los servicios que ofrecen a través de sus definiciones WSDL. Se basa en estándares de la industria, incluidos HTTP, XML, XML Schema, SOAP y WSDL. Existen dos tipos de registros UDDI: registros públicos que sirven como puntos de agregación para que las empresas publique sus servicios, y registros privados que cumplen una función similar dentro de las organizaciones. Es un esfuerzo intersectorial impulsado por todos los principales proveedores de software y plataformas, incluidos Fujitsu, HP, Hitachi, IBM, Intel, Microsoft, Oracle, SAP y Sun, así como otros contribuyentes dentro del consortio OASIS (Organization for the Advancement of Structured Information Standards).

La especificación UDDI 3.0 define un modelo de información compuesto por instancias de estructuras de datos persistentes llamadas *entidades*, que los nodos UDDI expresan en XML y almacenan de forma persistente. Están soportados los siguientes tipos de entidad (UDDI.org, 2004):

- *businessEntity*, que describe una organización que normalmente proporciona servicios web, incluido su nombre, descripción comercial, una lista de contactos y una lista de categorizaciones como industria, categoría de producto o ubicación geográfica.

- *businessService*, que describe una colección de servicios web relacionados ofrecidos por una entidad empresarial (businessEntity). Contiene información descriptiva de servicios de negocio sobre un grupo de servicios técnicos relacionados, incluido el nombre del grupo, una breve descripción, información vinculante del servicio técnico e información de categoría. La organización de los servicios web en grupos asociados con categorías o procesos de negocio permite a UDDI buscar y descubrir servicios web de manera más eficiente.

- *bindingTemplate*, que describe la información técnica necesaria para utilizar un businessService en particular. Incluye un punto de acceso (accessPoint) que se utiliza para transmitir la dirección de red adecuada para invocar el servicio web que se describe, que puede ser una URL, una dirección de correo electrónico o incluso un número de teléfono.

- *tModel* (modelo técnico), que representa un concepto reutilizable, como un tipo de servicio web, un protocolo utilizado por servicios web o un sistema de categorías, lo que facilita a los consumidores de servicios web encontrar servicios web que sean compatibles con una especificación técnica en particular. Cada especificación, transporte, protocolo o espacio de nombres distintos está representado por un tModel. Ejemplos de tModels son WSDL, XML Schema Definition (XSD) y otros documentos que describen y especifican el contrato y el comportamiento que un servicio web puede cumplir. Por ejemplo, para enviar una orden de compra, el servicio que invoca debe conocer no solo la URL del servicio, sino también en qué formato debe enviarse la orden de compra, qué protocolos son apropiados, qué seguridad se requiere y qué forma de respuesta se obtendrá después de enviar la orden de compra.

- *publisherAssertion*, que describe la relación que tiene una businessEntity con otra businessEntity.

- *suscripción*, que describe una petición permanente para realizar un seguimiento de los cambios en las entidades descritas por la suscripción.

Las entidades se pueden firmar opcionalmente mediante firmas digitales XML. La información proporcionada en un registro UDDI se puede utilizar para realizar tres tipos de búsquedas:

- *Búsqueda de páginas blancas* que contienen direcciones, contactos e identificadores conocidos. Por ejemplo, buscar una empresa por su nombre o su identificador único.

- *Búsqueda de páginas amarillas* que contiene categorizaciones industriales basadas en taxonomías estándar, como la NAICS (North American Industry Classification), UNSPSC (United Nations Standard Products and Services Code System) o los sistemas de clasificación de códigos de país ISO (ISO 3166).

- *Búsqueda de páginas verdes* que contiene información técnica sobre los servicios web expuestos por una organización, incluidas referencias a especificaciones de interfaces para servicios web, así como soporte para punteros a varios ficheros y mecanismos de descubrimiento basados en URL.

JSON (JavaScript Object Notation)

JSON es unas sintaxis de estándar abierto para almacenar e intercambiar información de texto. JSON es más pequeño que XML, y más rápido y fácil a la hora de parsear. Mientras que JSON deriva de JavaScript, es un lenguaje independiente, con parseadores disponibles para muchos lenguajes de programación. JSON se usa básicamente para transmitir datos entre servidores y aplicaciones web, como alternativa al XML. Los tipos de datos básicos de JSON son:

- Números
- String
- Boolean
- Array
- Object

3 XML Schema

Si bien XML 1.0 proporciona el mecanismo DTD para definir el modelo de contenido (el orden válido y el anidamiento de elementos) y, hasta cierto punto, los tipos de datos de los atributos de un documento XML, tiene una serie de limitaciones:

- está escrito en una sintaxis diferente (no XML);
- no tiene soporte para espacios de nombres;
- solo ofrece una mecanografía de datos extremadamente limitada.

Por lo tanto, se necesita un método más completo y riguroso para definir el modelo de contenido de un documento XML. El esquema XML del W3C supera estas limitaciones y es mucho más expresivo que los DTD (W3C, 2004a, b). La expresividad adicional permite que las aplicaciones web intercambien datos XML de manera mucho más robusta sin depender de herramientas de validación ad hoc. Un **esquema XML** es la definición (tanto en términos de su organización como de sus tipos de datos) de una estructura XML específica. El lenguaje W3C XML Schema especifica cómo se define cada tipo de elemento en el esquema y qué tipo de datos tiene asociado ese elemento. El esquema es en sí mismo un documento XML, que utiliza elementos y atributos para expresar la semántica del esquema. Como es un documento XML, puede ser editado y procesado por las mismas herramientas que leen el XML que describe.

A continuación, y como ejemplo, se mostrará cómo crear un esquema XML para el siguiente documento XML.

```
<?xml version= “1.0” encoding= “UTF-8” standalone= “no”?>
<?xml:stylesheet type = “text/xsl” href = “staff_list.xsl”?>
<!DOCTYPE STAFFLIST SYSTEM “staff_list.dtd”>
<STAFFLIST>
        <STAFF branchNo=”B005”>
                <STAFFNO>SL21</STAFFNO>
                        <NAME>
                                <FNAME>John</FNAME><LNAME>White</LNAME>
                        </NAME>
                <POSITION>Manager</POSITION>
                <DOB>1945-10-01</DOB>
                <SALARY>30000</SALARY>
        </STAFF>
        <STAFF branchNo=”B003”>
                <STAFFNO>SG37</STAFFNO>
                        <NAME>
                                <FNAME>Ann</FNAME><LNAME>Beech</LNAME>
                        </NAME>
                <POSITION>Assistant</POSITION>
                <SALARY>12000</SALARY>
        </STAFF>

</STAFFLIST>
```

Tipos de datos del esquema XML

El esquema XML define los siguientes tipos de datos:

- *boolean*, que contiene uno de los valores *true* o *false*.
- *string*, que contiene cero o más caracteres Unicode. *string* tiene varios subtipos como:
    - *normalizedString*, para cadenas que no contienen ningún espacio en blanco excepto el carácter de espacio;
    - *token*, un subtipo de *normalizedString*, para cadenas *tokenizadas* que no tienen espacios iniciales o finales y no tienen dos o más espacios seguidos;
    - *Name*, un subtipo de *token*, que representa nombres XML, con subtipos *NCName*, que representa un nombre XML sin dos puntos, y *NMTOKEN*;
    - *ID*, *IDREF* y *ENTITY*, subtipos de *NCName*, para los tipos de atributo correspondientes;
    - *IDREFS*, *ENTITIES*, *NMTOKENS*.
- *decimal*, que contiene un número real de precisión arbitraria en base 10 con subtipo *integer*, para valores sin una parte fraccionaria. Este subtipo a su vez tiene subtipos para *nonPositiveInteger*, *long* y *negativeInteger*. Otros tipos dentro de la jerarquía incluyen *int*, *short* y *byte*.
- *float*, para números de coma flotante binarios IEEE de 32 bits, y *double*, para números de coma flotante binarios IEEE de 64 bits.
- *date*, que contiene una fecha de calendario con el formato aaaa-mm-dd (por ejemplo, 1945-10-01 para el 1 de octubre de 1945); *time*, que contiene un tiempo de 24 horas, como 23:10; *dateTime*, una combinación de lo anterior, como 1945-10-01T23: 10.
- otros tipos relacionados con el tiempo, como *duration*, *gDía*, *gMonth*, *gYear*, para tiempos gregorianos.
- *QName*, un nombre calificado que consta de un nombre de espacio de nombres y un nombre local.
- *anySimpleType*, que es la unión de todos los tipos primitivos.
- *anyType*, que es la unión de todos los tipos (simples y complejos).

Tipos simples y complejos

Quizás la forma más fácil de crear un esquema XML es seguir la estructura del documento y definir cada elemento a medida que lo encontramos. Los elementos que contienen otros elementos son de tipo *complexType*. Para el elemento raíz STAFFLIST, podemos definir un elemento STAFFLIST que sea de tipo *complexType*. La lista de hijos del elemento STAFFLIST se describe mediante un elemento del tipo *sequence* (un compositor que define una secuencia ordenada de subelementos):

```
<xs:element name = "STAFFLIST">
        <xs:complexType>
                <xs:sequence>
                        <!-- children defined here -->
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

Cada uno de los elementos del esquema tiene el prefijo convencional xs:, que está asociado con el espacio de nombres del W3C XML Schema a través de la declaración xmlns:xs= "http://www.w3.org/2001/XMLSchema" (que se coloca en un elemento de tipo *schema*). STAFF y NAME también contienen subelementos y podrían definirse de manera similar. Los elementos que no tienen subelementos o atributos son de tipo *simpleType*. Por ejemplo, podemos definir STAFFNO, DOB y SALARY de la siguiente manera:

```
<xs:element name = "STAFFNO" type = "xs:string"/>
<xs:element name = "DOB" type = "xs:date"/>
```

```
<xs:element name = "SALARY" type = "xs:decimal"/>
```

Estos elementos han sido declarados usando tipos predefinidos *string*, *date* y *decimal* del W3C XML Schema, respectivamente, con el prefijo xs: para indicar que pertenecen al vocabulario del esquema XML. Podemos definir el atributo branchNo, que siempre debe ser el último, de la siguiente manera:

```
<xs:attribute name = "branchNo" type = "xs:string"/>
```

Cardinalidad

El W3C XML Schema permite representar la cardinalidad de un elemento utilizando los atributos *minOccurs* (el número mínimo de ocurrencias) y *maxOccurs* (el número máximo de ocurrencias). Para representar un elemento opcional, establecemos minOccurs en 0; para indicar que no hay un número máximo de ocurrencias, configuramos maxOccurs con el valor *unbounded*. Si no se especifica nada, entonces cada atributo tendrá por defecto el valor 1. Por ejemplo, como DOB es un elemento opcional, podríamos representarlo usando:

```
<xs:element name = "DOB" type = "xs:date" minOccurs = "0"/>
```

Si también queremos registrar los nombres de hasta tres familiares más cercanos para cada miembro del personal, podríamos representar esto usando:

```
<xs:element name = "NOK" type = "xs:string" minOccurs = "0" maxOccurs = "3"/>
```

Referencias

Aunque el método descrito anteriormente (mediante el cual definimos cada elemento como lo encontramos) es relativamente simple, conduce a una profundidad significativa en las definiciones incrustadas, y el esquema resultante puede ser difícil de leer y mantener. Un enfoque alternativo se basa en el uso de **referencias** a elementos y definiciones de atributos que deben estar dentro del alcance del referenciador. Por ejemplo, podríamos definir STAFFNO como:

```
<xs:element name = "STAFFNO" type = "xs:string"/>
```

y usar esta definición de la siguiente manera en el esquema, siempre que se requiera un elemento STAFFNO:

```
<xs:element ref = "STAFFNO"/>
```

Si hay muchas referencias a STAFFNO en el documento XML, el uso de referencias colocará la definición en un solo sitio y, por lo tanto, mejorará la capacidad de mantenimiento del esquema.

Definiendo nuevos tipos

XML Schema proporciona un tercer mecanismo para crear elementos y atributos basados en la definición de nuevos tipos de datos. Esto es análogo a definir una clase y luego usarla para crear un objeto. Podemos definir tipos simples para elementos o atributos PCDATA y tipos complejos para elementos. Los nuevos tipos reciben un nombre y la definición se ubica fuera de las definiciones de elementos y atributos. Por ejemplo, podríamos definir un nuevo tipo simple para el elemento STAFFNO de la siguiente manera:

```
<xs:simpleType name = "STAFFNOTYPE">
        <xs:restriction base = "xs:string">
                <xs:maxLength value = "5"/>
        </xs:restriction>
</xs:simpleType>
```

Este nuevo tipo se ha definido como una restricción del tipo de datos string del espacio de nombres del esquema XML (base de atributos) y también hemos especificado que tiene una longitud máxima de 5 caracteres (el elemento maxLength se denomina *facet*). El esquema XML define 15 facetas incluyendo *length, minLength, minInclusive* y *maxInclusive*. Otros dos particularmente útiles son *pattern* y *enumeration*. El elemento *pattern* define una expresión regular que debe coincidir. Por ejemplo, STAFFNO está restringido a tener dos caracteres en mayúsculas seguidos de entre uno y tres dígitos (como SG5, SG37, SG999), que podemos representar en el esquema utilizando el siguiente patrón:

```
<xs:pattern value = "[A-Z]{2}[0-9]{1, 3}">
```

El elemento *enumeration* limita un tipo simple a un conjunto de valores distintos. Por ejemplo, POSITION está restringido a tener solo los valores Manager, Supervisor o Assistant, que podemos representar en el esquema usando la siguiente enumeración:

```
<xs:enumeration value = "Manager"/>
<xs:enumeration value = "Supervisor"/>
<xs:enumeration value = "Assistant"/>
```

Grupos

El W3C XML Schema permite definir tanto grupos de elementos como grupos de atributos. Un grupo no es un tipo de datos, sino que actúa como un contenedor que contiene un conjunto de elementos o atributos. Por ejemplo, podríamos representar staff como un grupo de la siguiente manera:

```
<xs:group name = "STAFFTYPE">
        <xs:sequence>
                <xs:element name = "STAFFNO" type = "STAFFNOTYPE"/>
                <xs:element name = "POSITION" type = "POSITIONTYPE"/>
                <xs:element name = "DOB" type = "xs:date"/>
                <xs:element name = "SALARY" type = "xs:decimal"/>
        </xs:sequence>
</xs:group>
```

Esto crea un grupo llamado STAFFTYPE como una secuencia de elementos (para simplificar, hemos mostrado solo algunos de los elementos de STAFF). También podemos crear un elemento STAFFLIST para hacer referencia al grupo como una secuencia de cero o más STAFFTYPE de la siguiente manera:

```
<xs:element name = "STAFFLIST">
        <xs:complexType>
                <xs:sequence>
                        <xs:group ref = "STAFFTYPE" minOccurs = "0"
                                                maxOccurs = "unbounded"/>
                </xs:sequence>
        </xs:complexType>
```

```
</xs:element>
```

Elección y todos los compositores

Hemos mencionado anteriormente que *sequence* es un ejemplo de un compositor. Hay otros dos tipos de compositores: *choice* y *all*. El compositor *choice* define una elección entre varios elementos o grupos de elementos posibles, y el compositor *all* define un conjunto desordenado de elementos. Por ejemplo, podemos representar la situación en la cual el nombre de un miembro del personal puede ser una sola cadena o una combinación de nombre y apellido usando:

```
<xs:group name = "STAFFNAMETYPE">
        <xs:choice>
                <xs:element name = "NAME" type = "xs:string"/>
                <xs:sequence>
                        <xs:element name = "FNAME" type = "xs:string"/>
                        <xs:element name = "LNAME" type = "xs:string"/>
                </xs:sequence>
        </xs:choice>
</xs:group>
```

Listas y uniones

Podemos crear una lista de elementos separados por espacios en blanco utilizando el elemento *list*. Por ejemplo, podríamos crear una lista para contener números de personal usando:

```
<xs:simpleType name = "STAFFNOLIST">
        <xs:list itemType = "STAFFNOTYPE"/>
</xs:simpleType>
```

y usar este tipo en un documento XML de la siguiente manera:

```
<STAFFNOLIST> "SG5" "SG37" "SG999"</STAFFNOLIST>
```

Ahora podríamos derivar un nuevo tipo de este tipo de lista que tiene alguna forma de restricción; por ejemplo, podríamos producir una lista restringida de 10 valores usando lo siguiente:

```
<xs:simpleType name = "STAFFNOLIST10">
        <xs:restriction base = "STAFFNOLIST">
                <xs:Length value = "10"/>
        </xs:restriction>
</xs:simpleType>
```

Los tipos atómicos y los tipos de lista permiten que un elemento o valor de atributo sea una o más instancias de un tipo atómico. Por el contrario, un tipo de unión permite que un elemento o valor de atributo sea una o más instancias de un tipo seleccionado de la unión de múltiples tipos atómicos o de lista.

Restricciones

Hemos visto cómo se pueden utilizar las facetas para restringir los datos en un documento XML. El W3C XML Schema también proporciona características basadas en XPath para especificar las

restricciones de unicidad y las restricciones de referencia correspondientes que se mantendrán dentro de un cierto alcance. Consideramos dos tipos de restricciones: restricciones de unicidad y restricciones clave.

Restricciones de unicidad

Para definir una restricción de unicidad, especificamos un elemento *unique* que define los elementos o atributos que deben ser únicos. Por ejemplo, podemos definir una restricción de unicidad en el apellido y la fecha de nacimiento (DOB) del miembro del personal usando:

```
<xs:unique name = "NAMEDOBUNIQUE">
        <xs:selector xpath = "STAFF"/>
        <xs:field xpath = "NAME/LNAME"/>
        <xs:field xpath = "DOB"/>
</xs:unique>
```

La ubicación del elemento único en el esquema proporciona el nodo de contexto en el que se mantiene la restricción. Al colocar esta restricción bajo el elemento STAFF, especificamos que esta restricción tiene que ser única dentro del contexto de un elemento STAFF solamente. Los XPath especificados en los siguientes tres elementos son relativos al nodo de contexto. El primer XPath con el elemento *selector* especifica el elemento que tiene la restricción de unicidad (en este caso, STAFF). Los siguientes dos elementos de tipo *field* especifican los nodos que se deben verificar para determinar la unicidad.

Restricciones de clave (Key Constraints)

Una restricción de clave es similar a una restricción de unicidad, excepto que el valor no debe ser nulo. También permite hacer referencia a la clave. Podemos especificar una restricción clave en STAFFNO de la siguiente manera:

```
<xs:key name = "STAFFNOISKEY">
        <xs:selector xpath = "STAFF"/>
        <xs:field xpath = "STAFFNO"/>
</xs:key>
```

Un tercer tipo de restricción permite que las referencias se restrinjan a claves específicas. Por ejemplo, el atributo branchNo finalmente estaría destinado a hacer referencia a una sucursal. Si asumimos que dicho elemento se ha creado con la clave BRANCHNOISKEY, podríamos restringir este atributo a esta clave de la siguiente manera:

```
<xs:keyref name = "BRANCHNOREF" refer "BRANCHNOISKEY">
        <xs:selector xpath = "STAFF"/>
        <xs:field xpath = "@branchNo"/>
</xs:keyref>
```

Resource Description Framework (RDF)

Aunque XML Schema proporciona un método más completo y riguroso para definir el modelo de contenido de un documento XML que las DTD, todavía no proporciona el soporte para la interoperabilidad semántica que requerimos. Por ejemplo, cuando dos aplicaciones intercambian información utilizando XML, ambas acuerdan el uso y el significado previsto de la estructura del documento. Sin embargo, antes de que esto suceda, se debe construir un modelo del dominio de

interés, para aclarar qué tipo de datos se enviarán desde la primera aplicación a la segunda. Este modelo generalmente se describe en términos de objetos o relaciones. Sin embargo, dado que el esquema XML solo describe una gramática, hay muchas formas diferentes de codificar un modelo de dominio específico en un esquema XML, perdiendo así la conexión directa entre el modelo de dominio y el esquema. Este problema se agrava si una tercera aplicación desea intercambiar información con las otras dos aplicaciones. En este caso, no es suficiente mapear un esquema XML a otro, ya que la tarea no es mapear una gramática a otra gramática, sino mapear objetos y relaciones de un dominio de interés a otro. Por lo tanto, se requieren tres pasos:

- rediseñar los modelos de dominio originales a partir del esquema XML;
- definir mapeos entre objetos en los modelos de dominio;
- definir mecanismos de traducción para los documentos XML, por ejemplo, utilizando XSLT.

Estos pasos pueden no ser triviales y, por lo tanto, podemos encontrar que XML es muy adecuado para el intercambio de datos entre aplicaciones que conocen el modelo de contenido de los datos, pero no en situaciones en las que se pueden introducir nuevas aplicaciones que también desean intercambiar datos. Lo que se requiere es un lenguaje universalmente compartido para representar los dominios de interés.

El Resource Description Framework (RDF), desarrollado bajo los auspicios del W3C, es una infraestructura que permite la codificación, el intercambio y la reutilización de metadatos estructurados (W3C, 1999d, 2004c). Esta infraestructura permite la interoperabilidad de metadatos mediante el diseño de mecanismos que admiten convenciones comunes de semántica, sintaxis y estructura. RDF no estipula la semántica para cada dominio de interés, sino que proporciona la capacidad para que estos dominios definan elementos de metadatos según sea necesario. RDF utiliza XML como sintaxis común para el intercambio y procesamiento de metadatos. Al explotar las características de XML, RDF impone una estructura que permite la expresión de la semántica y, como tal, permite una descripción e intercambio coherentes de metadatos estandarizados.

Modelo de datos RDF

El modelo de datos RDF básico consta de tres objetos:

- **Recurso**, que es cualquier cosa que pueda tener una URI; por ejemplo, una página web, varias páginas web o una parte de una página web, como un elemento XML.
- **Propiedad**, que es un atributo específico que se utiliza para describir un recurso. Por ejemplo, el atributo Autor se puede utilizar para describir quién produjo un documento XML en particular.
- **Declaración**, que consiste en la combinación de un recurso, una propiedad y un valor. Estos componentes se conocen como "sujeto", "predicado" y "objeto" de una declaración RDF. Por ejemplo, "The Author of http://www.dreamhome.co.uk/staff_list.xml is John White" es una frase.

Podemos expresar esta última frase en RDF de la siguiente manera:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://www.dreamhome.co.uk/schema/">
        <rdf:Description about="http://www.dreamhome.co.uk/staff_list.xml">
                <s:Author>John White</s:Author>
        </rdf:Description>
</rdf:RDF>
```

Esquema RDF

El esquema RDF especifica información sobre las clases en un esquema, incluidas las propiedades (atributos) y las relaciones entre los recursos (clases). El mecanismo de esquema RDF proporciona un sistema de tipos básico para su uso en modelos RDF, análogo al esquema XML (W3C, 2000c, 2003f). Define recursos y propiedades como rdfs:Class y rdfs:subClassOf que se utilizan para especificar

esquemas específicos de la aplicación. También proporciona una función para especificar un pequeño número de restricciones, como la cardinalidad requerida y permitida de las propiedades de las instancias de clases.

Un esquema RDF se especifica utilizando un lenguaje declarativo influenciado por ideas de representación del conocimiento (por ejemplo, redes semánticas y lógica de predicados), así como modelos de representación de esquemas de base de datos como modelos relacionales binarios, por ejemplo, NIAM (Nijssen y Halpin, 1989) y modelos de datos gráficos.

4 XML and Database

A medida que la cantidad de datos en formato XML se expande, habrá una demanda cada vez mayor para almacenar, recuperar y consultar estos datos. Se prevé que existirán dos modelos principales: centrado en datos y centrado en documentos. En un modelo centrado en datos, XML se utiliza como el formato de almacenamiento e intercambio de datos que está estructurado, aparece en un orden regular y es más probable que sea procesado por una máquina en lugar de ser leído por un ser humano. En un modelo centrado en datos, el hecho de que los datos se almacenen y transfieran como XML es accidental y también se podrían haber utilizado otros formatos. En este caso, los datos podrían almacenarse en un DBMS relacional, objeto-relacional u orientado a objetos. Por ejemplo, XML se ha integrado completamente en Oracle.

En un modelo centrado en documentos, los documentos están diseñados para consumo humano (por ejemplo, libros, periódicos y correo electrónico). Debido a la naturaleza de esta información, muchos de los datos serán irregulares o incompletos y su estructura puede cambiar rápidamente o de manera impredecible. Desafortunadamente, los DBMS relacionales, objeto- relacionales y orientados a objetos no manejan datos de esta naturaleza particularmente bien. Los sistemas de gestión de contenido son una herramienta importante para manejar este tipo de documentos. Detrás de un sistema de este tipo, puede que ahora haya una base de datos XML nativa (NXD).

Esta división binaria no es absoluta. Los datos, en particular los datos semiestructurados, se pueden almacenar en una base de datos XML nativa o en una base de datos tradicional cuando se requieren pocas características específicas de XML. Además, los límites entre estos dos tipos de sistemas se están volviendo menos claros, ya que los DBMS más tradicionales agregan capacidades XML nativas y las bases de datos XML nativas admiten el almacenamiento de fragmentos de documentos en bases de datos tradicionales.

Almacenar XML en bases de datos

Antes de discutir algunos de los enfoques comunes para almacenar documentos XML en DBMS tradicionales, enumeramos brevemente algunos de los tipos de documentos XML que deben manejarse:

- XML que puede estar fuertemente tipado gobernado por un esquema XML correspondiente;
- XML que puede estar fuertemente tipado gobernado por otro lenguaje de esquema, como DTD o RELAX-NG;
- XML que puede regirse por múltiples esquemas o un esquema que puede estar sujeto a cambios frecuentes;
- XML que puede no tener esquema;
- XML que puede contener texto marcado con unidades lógicas de texto (como oraciones) que abarcan varios elementos;
- XML con estructura, orden y espacios en blanco que pueden ser importantes y es posible que deseemos recuperar exactamente el mismo contenido XML de la base de datos en una fecha posterior;
- XML que puede estar sujeto a actualización, así como consultas basadas en el contexto y la relevancia.

Hay cuatro enfoques generales para almacenar un documento XML en una base de datos relacional:

- almacenar el XML como el valor de algún atributo dentro de una tupla;
- almacenar el XML en una forma triturada a través de una serie de atributos y relaciones;
- almacenar el XML en una forma independiente del esquema;
- almacenar el XML en una forma parseada, es decir, convierta el XML a formato interno, como una representación Infoset o PSVI, y almacene esta representación.

Estos enfoques no son necesariamente excluyentes entre sí. Por ejemplo, sería posible almacenar algunos de los XML triturados como atributos en una relación y dejar algunos nodos intactos y almacenados como el valor de algún atributo en la misma relación o en una relación separada.

Almacenar el XML en un atributo

Con este enfoque, en el pasado, el XML se habría almacenado en un atributo cuyo tipo de datos era un objeto grande de caracteres (CLOB). Más recientemente, algunos sistemas han implementado un nuevo tipo de datos XML nativo. En Oracle, este tipo de datos se denomina XMLType (aunque el almacenamiento subyacente puede ser CLOB). El estándar SQL define un tipo de datos incorporado llamado XML, pero no prescribe una estructura de almacenamiento específica siempre que satisfaga el requisito del tipo de datos XML. Los documentos XML sin procesar se almacenan en su forma serializada, lo que hace que sea eficiente insertarlos en la base de datos y recuperarlos en su forma original. Este enfoque también hace que sea relativamente fácil aplicar la indexación de texto completo a los documentos para la recuperación contextual y de relevancia. Sin embargo, existen dudas sobre el rendimiento de las consultas y la indexación generales, que pueden requerir un análisis sobre la marcha. Además, las actualizaciones generalmente requieren que todo el documento XML se reemplace por un nuevo documento, en lugar de solo la parte del XML que ha cambiado.

Almacenar el XML de forma "triturada" (shredded)

Con este enfoque, el documento XML se descompone en sus elementos constitutivos y los datos se distribuyen en varios atributos en una o más relaciones. El término que se utiliza para esta descomposición es triturar (shredding). El almacenamiento de documentos triturados puede facilitar la indexación de los valores de elementos particulares, siempre que estos elementos se coloquen en sus propios atributos. También sería posible agregar algunos datos adicionales relacionados con la naturaleza jerárquica del XML, lo que permitiría recomponer la estructura original y ordenar en una fecha posterior, y permitir la actualización del XML. Con este enfoque también tenemos que crear una estructura de base de datos adecuada.

Creación de un esquema de base de datos

Antes de que podamos comenzar a transferir datos, tenemos que diseñar y luego crear un esquema de base de datos apropiado para almacenar los datos. Si hay un esquema asociado con el XML, se puede derivar una estructura de base de datos a partir de este esquema. Aquí discutimos dos enfoques principales:

- un mapeo relacional;
- un mapeo objeto-relacional.

El enfoque de mapeo relacional comienza en la raíz del documento XML y asocia este elemento con una relación. Para cada uno de los hijos de este elemento se toma una decisión sobre si incluir el elemento hijo como un atributo en esta relación o crear una nueva relación (en cuyo caso, algún elemento será elegido como clave primaria / clave externa o artificial clave creada). Una regla simple para tomar esta decisión es crear una nueva relación si un elemento puede repetirse; por ejemplo, si maxOccurs> 1. Además, se debe tomar una decisión sobre si representar elementos opcionales dentro de la misma relación que su padre o si crear una nueva relación (en el último caso, se requeriría una combinación adicional en tiempo de ejecución para vincular las dos relaciones). El enfoque también puede intentar identificar elementos comunes que aparecen en más de una ubicación dentro del XML y crear una relación para dichos elementos.

El enfoque de mapeo objeto- relacional modela tipos de elementos complejos como clases / tipos. Estos incluirían tipos de elementos con atributos, contenido de elementos y contenido mixto. De lo contrario, modela tipos de elementos simples como propiedades escalares. Estos incluirían atributos, PCDATA y contenido exclusivo de PCDATA. Las clases / tipos y las propiedades escalares se asignarían luego a los tipos y tablas de SQL: 2011. Cualquiera que sea el tipo de mapeo que se elija, puede ser necesario modificar el diseño resultante a mano para corregir las deficiencias debidas a las estructuras arbitrarias y complejas que pueden aparecer en un documento XML.

Por otro lado, si no existe ningún esquema, se podría inferir un esquema de base de datos a partir del contenido de uno o más documentos XML de muestra, aunque no hay garantía de que los documentos futuros se ajusten a la estructura de los documentos de muestra. En este caso, puede ser preferible el

enfoque anterior de almacenar el XML directamente en un atributo de una relación. Una alternativa sería considerar una representación independiente del esquema, como veremos a continuación.

Representación independiente del esquema

En lugar de intentar inferir una estructura relacional para el XML, ya sea a partir de un esquema asociado o de la estructura y el contenido del XML mismo, un enfoque alternativo es utilizar una representación que sea independiente del esquema. En una representación independiente del esquema XML, la naturaleza recursiva de la estructura puede causar problemas de rendimiento al buscar rutas específicas. Para superar esto, se puede crear una estructura de índice desnormalizada que contenga combinaciones de expresiones de ruta y un enlace al nodo y al nodo padre.

Una vez que se ha creado una estructura apropiada y el XML ingresado en la base de datos, podemos usar SQL (posiblemente con algunas extensiones) para consultar los datos.

XML y SQL

A pesar del entusiasmo que rodea al XML, es importante tener en cuenta que la mayoría de los datos comerciales operativos, incluso para las nuevas aplicaciones basadas en la Web, continúan almacenándose en DBMS relacionales. Es poco probable que esto cambie en un futuro previsible debido a su confiabilidad, escalabilidad, herramientas y rendimiento. En consecuencia, para que XML alcance su potencial, se requiere algún mecanismo para publicar datos relacionales en forma de documentos XML. Los estándares SQL: 2003, SQL:2008 y SQL:2011 han definido extensiones SQL para permitir la publicación de XML, comúnmente referido como SQL / XML (ISO, 2011b). En particular, SQL / XML contiene:

- un nuevo tipo de datos XML nativo, XML, que permite que los documentos XML sean tratados como valores relacionales en columnas de tablas, atributos en tipos definidos por el usuario, variables y parámetros de funciones;
- un conjunto de operadores para el tipo;
- un conjunto implícito de asignaciones de datos relacionales a XML.

El estándar no define ninguna regla para el proceso inverso; es decir, triturar datos XML en un formulario SQL, con algunas excepciones menores, como se explica a continuación.

SQL/XML and XQuery

En SQL/XML:2003, el modelo de datos SQL/XML se basó en el conjunto de información del W3C, principalmente debido al modelo de datos XQuery 1.0 y XPath 2.0 (XDM) no se consideró lo suficientemente estable en ese momento. Como Infoset reconoce solo tres tipos atómicos: booleano, double y string, esto fue bastante restrictivo. Sin embargo, la nueva versión del estándar SQL/XML está alineada con el XDM, que tiene varios tipos atómicos definidos en XML Schema, además de cinco nuevos tipos. Esto significa que cualquier valor XML también es una secuencia XQuery.

Nuevo tipo de datos XML

El nuevo tipo de datos se denomina simplemente XML y se puede utilizar en la definición de una columna en una tabla, un atributo en un tipo definido por el usuario, una variable o un parámetro de una función. Los valores legales para este tipo de datos consisten en el valor nulo, una colección de elementos de información SQL/XML (que consta de un elemento raíz) o cualquier otro elemento de información SQL/XML al que se pueda acceder de forma recursiva atravesando las propiedades de estos elementos. Un elemento de información SQL/XML es generalmente uno de los elementos de información definidos en el conjunto de información XML. En una definición de columna, se pueden especificar cláusulas opcionales para proporcionar un espacio de nombres y/o un esquema de codificación binaria (BASE64 o HEX).

Se han definido varios operadores que producen valores XML como:

- **XMLELEMENT**, genera un valor XML con un solo elemento XQuery como hijo de un nodo de documento XQuery. El elemento puede tener cero o más atributos especificados mediante una subcláusula **XMLATTRIBUTES**.
- **XMLFOREST**, genera un valor XML con una secuencia de nodos de elementos XQuery, posiblememnte como hijo de un nodo de documento XQuery.
- **XMLCONCAT**, concatena una lista de valores XML.
- **XMLPARSE**, realiza un parseo sin validación de una cadena de caracteres para producir un valor XML.
- **XMLCOMMENT**, genera un valor XML con un solo nodo de comentario XQuery, posiblemente como un hijo de un nodo de documento XQuery.
- **XMLPI**, genera un valor XML con un solo nodo de instrucción de procesamiento XQuery, posiblemente como un hijo de un nodo de documento XQuery.
- **XMLDOCUMENT**, genera un valor XML con un solo nodo de documento XML.
- **XMLQUERY**, una nueva expresión SQL, invocada por una pseudofunción, para evaluar una expresión XQuery.
- **XMLTEXT**, genera un valor XML con un solo nodo de texto XQuery, posiblemente como hijo de un nodo de un documento XQuery.
- **XMLVALIDATE**, valida un valor XML contra un esquema XML, devolviendo un nuevo valor XML con anotaciones de tipo.
- **XMLCAST**, especifica una conversión de datos cuya fuente u objetivo es un tipo XML.
- **XMLTABLE**, crea una tabla SQL virtual que contiene datos derivados de valores XML en los que opera la función.

SQL/XML también define los siguientes predicatos:

- **IS [NOT] DOCUMENT**, determina si un valor XML cumple con los criterios de SQL/XML para un documento XML.
- **IS [NOT] CONTENT**, determina si un valor XML cumple con los criterios de SQL/XML para contenido XML.
- **XMLEXISTS**, probar una secuencia XQuery no vacía. Devuelve *verdadero (true)* cuando la expresión XQuery contenida devuelve cualquier cosa que no sea la secuencia vacía (*false*) o el valor NULL de SQL.
- **IS [NOT] VALID**, determina si un valor XML es válido según un esquema XML registrado; devuelve *true/false* sin alterar el propio valor XML.

Dos funciones útiles son:

- **XMLSERIALIZE**, genera una cadena de caracteres o de binarios a partir de un valor XML.
- **XMLAGG**, una función agregada, que genera un bosque de elementos a partir de una colección de elementos.

Funciones de mapeo

El estándar SQL/XML también define un mapeo de tablas a documentos XML y de XML a SQL. El mapeo de SQL a XML puede tomar como origen una tabla individual, todas las tablas de un esquema particular o todas las tablas de un catálogo determinado. El estándar no especifica una sintaxis para el mapeo; en cambio, se proporciona para su uso por aplicaciones y como referencia para otros estándares. El mapeo produce dos documentos XML: uno que contiene los datos de la tabla mapeada y otro que contiene un esquema XML que describe el primer documento.

Mapear identificadores SQL a nombres XML

Se tuvieron que abordar una serie de cuestiones para asignar identificadores SQL a nombres XML; por ejemplo:

- el rango de caracteres que se pueden usar dentro de un identificador SQL es mayor que el rango de caracteres que se pueden usar en un nombre XML;
- Los identificadores delimitados por SQL (identificadores delimitados por comillas dobles) permiten el uso de caracteres arbitrarios en cualquier punto del identificador;
- Los nombres XML que comienzan con "XML" están reservados;

- Los espacios de nombres XML utilizan ":" para separar el prefijo del espacio de nombres del componente local.

El enfoque adoptado para resolver estos problemas se basa en el uso de una notación de escape que transforma los caracteres que no son aceptables en los nombres XML en una secuencia de caracteres permitidos basada en valores Unicode. La convención es reemplazar un carácter no aceptado con "_xHHHH_", donde HHHH es el equivalente hexadecimal del valor Unicode correspondiente.

Hay dos variantes del mapeo conocidas como escape parcial y escape total (en el primer caso, el carácter ":" no está mapeado).

Mapear los tipos de datos SQL a tipos de datos esquema XML

SQL tiene varios tipos de datos predefinidos incorporados, y tres tipos construidos incorporados (ROW, ARRAY y MULTISET). Por otro lado, XML Schema Parte 2: Datatypes define varios tipos de datos simples para XML y representaciones léxicas para los valores de estos tipos. SQL/XML mapea cada tipo de datos SQL a la coincidencia más cercana en el esquema XML, en algunos casos utilizando facetas para restringir los valores XML aceptables para lograr la coincidencia más cercana.

Mapear tablas a documentos XML

El mapeo de una tabla individual se logra creando un elemento raíz con el nombre de la tabla con un elemento <row> para cada fila. Cada fila contiene una secuencia de elementos de columna, cada uno con el nombre de la columna correspondiente. Cada elemento de columna contiene un valor de datos. Los nombres de los elementos de la tabla y la columna se generan utilizando la asignación de escape completo de los identificadores SQL a los nombres XML.

Nulls

Además de proporcionar el nombre de la tabla que se mapeará, el usuario debe especificar cómo se manejarán los valores null. Las opciones se denominan "absent" y "nil". Si se especifica "absent", cualquier columna con un valor nulo se omitirá de la asignación. Si se especifica "nil", entonces el atributo xsi: nil = "true" se usa para indicar un elemento de columna que representa un nulo.

Generar un esquema XML

Un esquema XML se genera mediante la creación de tipos de datos de esquema XML con nombre global para cada tipo que se requiere para describir las tablas que se asignan. Se utiliza una convención de nomenclatura para nombrar los tipos de datos asignados mediante un sufijo que contiene longitud o precisión/escala para el nombre del tipo base. Por ejemplo, CHAR (10) se llamaría CHAR_10, DECIMAL (8, 2) se llamaría DECIMAL_8_2, mientras que INTEGER permanecería como INTEGER.

Bases de datos XML nativas (NDX)

NDX define un modelo de datos lógico para un documento XML que se utiliza para almacenar y recuperar el XML. El documento XML debe ser la unidad de almacenamiento (lógico), aunque no está restringido por ningún modelo de almacenamiento físico subyacente (y por lo tanto, por ejemplo, sistemas de almacenamiento relacionales, objeto-relacional y jerárquicos son posibles).

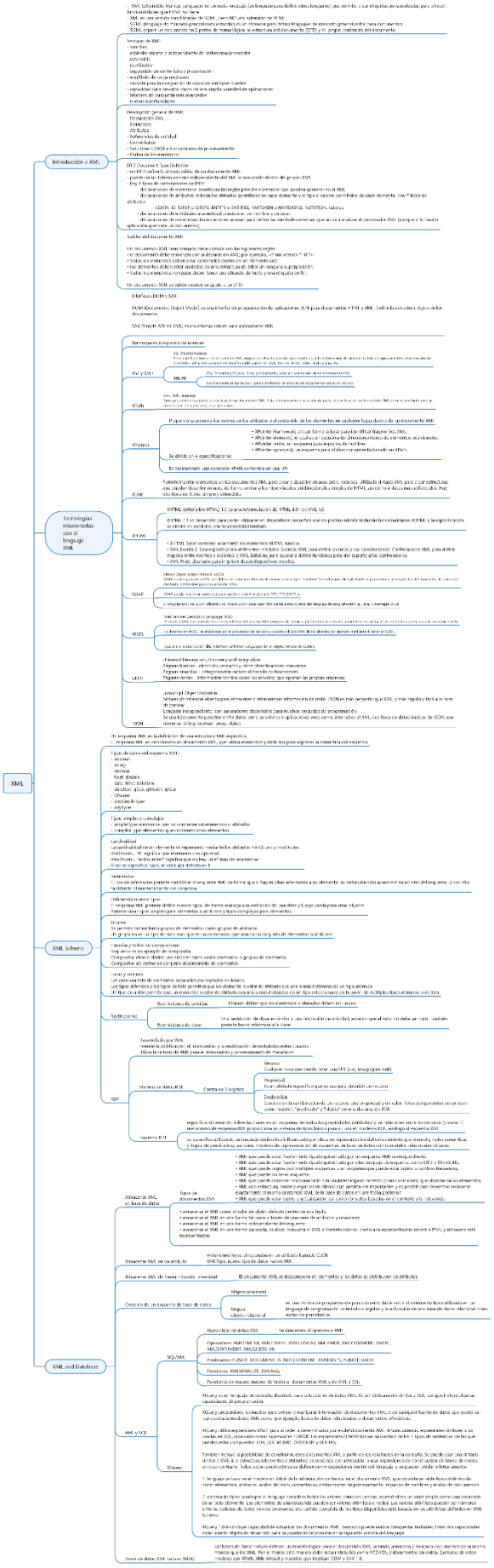
La parte clave de esta definición es que el modelo lógico se basa en XML y el DBMS resultante está diseñado más claramente para el almacenamiento y recuperación de documentos centrados en documentos. Algunos autores argumentan que no es solo el modelo lógico el que debe basarse en XML, sino también el modelo de almacenamiento físico y que un modelo arbitrario con una capa XML encima es inadecuado. Al igual que con cualquier otro tipo de DBMS, el DBMS XML nativo debe admitir transacciones, simultaneidad, recuperación y seguridad. Además, esperaríamos que el DBMS admita otras tecnologías XML, como XQuery, XPath, XML Schema, XPointer y XSL/XSLT.

Podemos distinguir dos tipos principales de DBMS XML nativo:

- basado en texto, que almacena el XML como texto, por ejemplo, como un archivo en un sistema de archivos o como un CLOB en un DBMS relacional;
- basado en modelo, que almacena el XML en alguna representación de árbol interno, por ejemplo, una representación Infoset o PSVI, o una representación DOM, posiblemente con etiquetas tokenizadas. Este enfoque facilita la identificación y recuperación de información basada en la estructura del documento XML además de su contenido y, a menudo, proporciona un buen rendimiento para la indexación basada en valores de elementos.

En cualquier caso, esperaríamos que un DBMS XML nativo gestionara no solo consultas y operaciones de inserción / eliminación, sino también actualizaciones de partes de un documento XML. Algunos ejemplos de DBMS XML nativos son: dbXML (de dbXML Group), GoXML (de XML Global), Ipedo (de Ipedo), Tamino (de Software AG), X-Hive (de X-Hive Corporation) y open- fuente Xindice (de Apache Software Foundation).

5 RESUMEN ESQUEMÁTICO


XML
Introducción a XML
Tecnologías relacionadas con el lenguaje XML
XML Schema
XML and Database

6 GLOSARIO

CML Chemistry Markup Language
DOM Document Object Model
HTML HyperText Markup Language
JSON JavaScript Object Notation
MathML Mathematics Markup Language
SGML Standard Generalized Markup Language
SMIL Synchronized Multimedia Integration Language
DTD Document Type Definition
SAX Simple API for XML
SOAP Simple Object Access Protocol
UDDI Universal Discovery, Description, and Integration
WSDL Web Services Description Language
W3C World Wide Web Consortium
XHTML eXtensible HTML
XML eXtensible Markup Language
XPath XML Path Language
XSL eXtensible Stylesheet Language
XSLT XSL Transformations

7 BIBLIOGRAFÍA BÁSICA

T. Connolly and C. Begg. Database systems: a practical approach to design, implementation, and management (6th ed.). Capítulo 30. Secciones 30.2, 30.3, 30.4 y 30.6.