



BASES DE DATOS AVANZADAS

Tema 5

Modelos Semiestructurados

Univ. Cantabria – Fac. de Ciencias

Francisco Ruiz



Objetivos

- Presentar los modelos de datos **semiestructurados**.
- Dar a conocer la importancia del **XML**, sus características, sus componentes y su relación con las bases de datos.
- Conocer qué es una **base de datos XML**.
- Presentar algunos **SGBD** que soportan XML.
- Saber crear **esquemas XSD**.
- Aprender de manera básica a trabajar con los lenguajes de consultas a bases de datos XML: **SQL/XML** y **XQuery**.



Agradecimientos

- *Parte de este material ha sido preparado con la colaboración de profesores del grupo Kybele, de la Universidad Rey Juan Carlos (Madrid).*



Contenido

- **Introducción**
 - Nivel de Estructuración de Datos
 - Bases de Datos Documentales
 - Integración de Datos muy Estructurados y poco Estructurados
- **XML**
 - Extensiones
 - Estructura de Datos y Documentos
 - Esquemas XSD
 - Direccionamiento – XPath y XLink
 - Transformación - XSL
 - Consultas – XQuery
- **XML y Bases de Datos**
 - Sistemas de Bases de Datos Nativos XML
 - Integración de XML en otros SGBD



Bibliografía

- Básica
 - Piattini et al. (2006): Tecnología y Diseño de Bases de Datos.
 - Cap. 22.
 - Connolly y Begg (2005): Sistemas de Bases de Datos.
 - Caps. 30.
- Complementaria
 - Elmasri y Navathe (2007): Fundamentos de Sistemas de Bases de Datos.
 - Caps. 27.
 - Abiteboul et al. (1999): Data on the Web. From Relations to Semistructured Data and XML. Morgan Kaufmann.
 - Cap. 3-5 y 11.



Estándares

- Extensible Markup Language (XML) 1.1.
 - <http://www.w3.org/TR/xml11/>
- XML Schema (parts 0, 1 & 2)
 - <http://www.w3.org/TR/xmlschema-0/> (-1, -2)
- XQuery 1.0
 - <http://www.w3.org/TR/xquery/>
- XQuery/XPath Data Model (XDM)
 - <http://www.w3.org/TR/xpath-datamodel/>



- Datos **estructurados** → representados en un formato estricto (*relaciones/tablas*)
- Datos **semi-estructurados** → los datos tienen una cierta estructura pero no toda la información recogida tiene la misma estructura (*grafos*)
- Datos **desestructurados** → hay una indicación muy limitada del tipo de datos (documentos de texto, archivos de video)



- Están orientadas a almacenar datos **desestructurados** de tipo texto.
- Carecen de una estructura tabular y contienen información bibliográfica y/o el texto completo de los documentos.
- Los modelos de información consideran que cada documento se describe con un conjunto de palabras clave significativas (llamadas **índices**).
- Los índices se utilizan para indexar y resumir el contenido de los documentos, ayudando a su gestión.

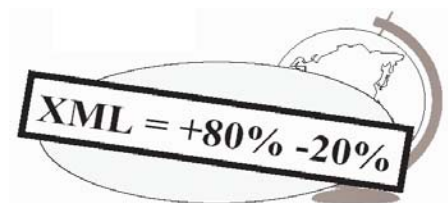


- En la actualidad es una necesidad **integrar los datos muy estructurados con los poco estructurados**.
 - La principal diferencia entre ambos es la forma de manejar los constructores del esquema (nombres de atributos, relaciones y tipos de entidades, etc.)
 - En el caso de los poco estructurados, la información del esquema se mezcla con los valores de los datos ya que un objeto de datos puede tener diferentes atributos no conocidos por adelantado. Por eso, estos tipos de datos se conocen como datos **autodescriptivos**.



XML

- **XML (eXtensible Markup Language)**
 - se plantea a mediados de los 90 por la necesidad de poder definir y manejar estructura interna en los documentos HTML.
 - pero está basado en un estándar bastante anterior: **SGML** (*Standard Generalized Markup Language*)
 - con orígenes en los años 60 (IBM, GML).
 - estándar ISO 8879 en 1986.
 - XML vs SGML
 - regla del 80/20
 - 80% de la funcionalidad
 - 20% de la complejidad.
 - XML \approx SGML light





XML

- **XML** es una **idea simple**, no nueva, pero **tremendamente útil** porque:
 - llegó en un **momento adecuado** para sacar partido de diversas tecnologías (Internet, Web, ...)
 - y poder abordar de **nuevas maneras**, más adecuadas y potentes, problemas importantes:
 - **Integración de datos** estructurados (tablas relacionales) y poco estructurados (documentos).
 - Aportar **significado a la web** (web semántica).
 - **Integración de sistemas** de información basados en tecnologías diferentes.
 - Hacer una **web orientada a las aplicaciones** (*web services*) en vez de una web orientada sólo a la interacción con personas (HTML).



XML

- **XML** es
 - Un lenguaje de **marcas** (etiquetas delimitadas)
 - para definir nuevos lenguajes (un **metalenguaje**)
- Con las siguientes características principales:
 - **Versátil**: separa contenido, estructura y presentación
 - **Extensible**: se pueden definir nuevas etiquetas
 - **Estructurado**: se pueden modelar datos a cualquier nivel de complejidad
 - **Validable**: cada documento se puede validar frente a un DTD/Schema, o en su defecto, se puede declarar bien formado.
 - **Abierto**: independiente de plataformas, empresas, lenguajes de programación o entornos de desarrollo.
 - **Sencillo**: fácil de aprender y de usar.



XML

- Otras **características adicionales** de **XML** son:
 - **Independencia del medio**, pudiendo publicar contenidos en múltiples formatos.
 - Los documentos XML son **fácilmente procesables** y compartibles en Internet.
 - Permite **validación** de los documentos.
 - Permite **composición** de los documentos.
 - Puede ser un **contenedor de datos**. Utilizando **DTD** o **XML Schemas** se puede representar eficientemente cualquier dato de forma que puede ser leído por personas o aplicaciones.
 - **Internacional**: utiliza el conjunto de caracteres UNICODE.
 - **Ayuda a descongestionar Internet**, ya que gran parte del procesamiento se puede hacer en el cliente.
 - XML no es compatible con HTML, pero los documentos HTMLv4.0 son fácilmente convertibles a XML.



XML

- **Documentos Bien Formados**
 - **Estructura jerárquica de elementos**
 - Las etiquetas deben estar correctamente anidadas.
 - Los elementos con contenido deben estar correctamente cerrados.
 - Mal: `HTML permite <I>esto</I>`
 - Bien: `En XML la estructura <I>es</I>`
 - **Etiquetas vacías**
 - Mal: `Esto es HTML
en el que casi todo está permitido`
 - Bien: `En XML, somos
 más restrictivos.`
 - **Un solo elemento raíz**
 - La jerarquía de elementos de un documento XML bien-formado sólo puede tener un elemento inicial.



XML

- Documentos Bien Formados
 - Valores de Atributos
 - Siempre deben estar encerrados en comillas simples o dobles.
 - Mal: `<web url=http://www.discovery.com/>`
 - Bien: `<web url="http://www.developer.com/">`
 - Tipos de Letra
 - XML es sensible al tipo de letra utilizado, diferenciando mayúsculas y minúsculas.
 - Espacios en Blanco
 - Se permite el uso de "espacios en blanco" para hacer más legible el código, aunque son ignorados por los procesadores XML.
 - Espacio, tabulación, retorno de carro, salto de línea.



XML

- Documentos Bien Formados
 - Nombres
 - Los nombres de estructuras, tipos de elementos, entidades, elementos particulares, etc., deben cumplir ciertas condiciones:
 - No empezar por "xml".
 - Las letras y rayas se pueden usar en cualquier parte.
 - También se pueden incluir dígitos, guiones y caracteres de punto, pero no se puede empezar por ellos.
 - El resto de caracteres (algunos símbolos y espacios en blanco) no se pueden usar.
 - Marcas
 - Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan "marcas".
 - Empiezan por "<" y acaban por ">"
 - Salvo referencias de entidad que empiezan por "&" y acaban por ";".



XML

- **Ejemplo:**

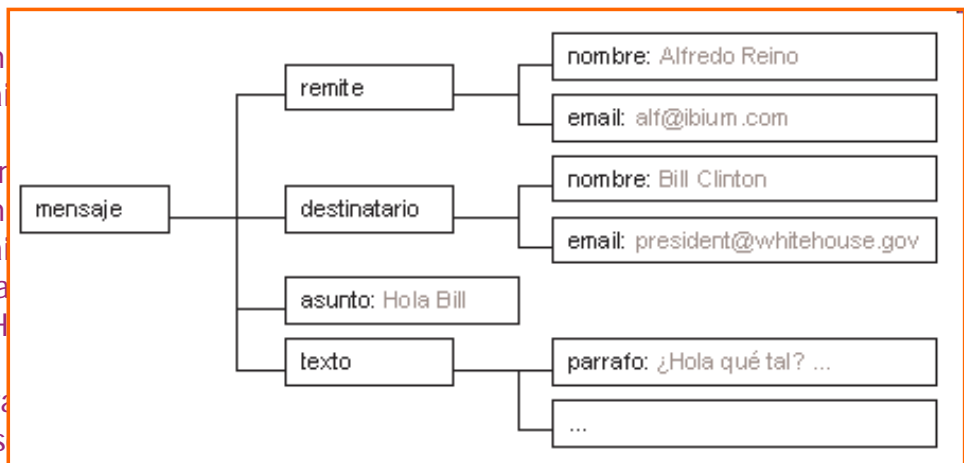
```
<?xml version="1.0"?>
<!DOCTYPE MENSAJE SYSTEM "mensaje.dtd">
<mensaje>
  <remite>
    <nombre>Alfredo Reino</nombre>
    <email>alf@myemail.com</email>
  </remite>
  <destinatario>
    <nombre>Bill Clinton</nombre>
    <email>president@whitehouse.gov</email>
  </destinatario>
  <asunto>Hola Bill</asunto>
  <texto>
    <parrafo>¿Hola qué tal? Hace <enfasis>mucho</enfasis> que
    no escribes. A ver si llamas y quedamos para tomar algo.</parrafo>
  </texto>
</mensaje>
```



XML

- **Ejemplo:**

```
<?xml version="1.0"?>
<!DOCTYPE MENSAJE SYSTEM "mensaje.dtd">
<mensaje>
  <remite>
    <nom
    <ema
  </remite>
  <destinatario>
    <nom
    <ema
  </destinatario>
  <asunto>H
  <texto>
    <parra
    no es
  </texto>
</mensaje>
```





XML

- Un documento XML está formado por dos **partes**:
 - **Prólogo**
 - No es obligatorio.
 - Contenido:
 - Una **declaración XML**. Es la sentencia que declara la versión de XML y opcionalmente, el sistema de codificación y si es documento autónomo.
 - `<?xml version="1.0" encoding="UTF-7" standalone="yes"?>`
 - Una **declaración de tipo de documento**. Enlaza el documento con su DTD (definición de tipo de documento) o XSD (definición de esquema XML).
 - Uno o más comentarios e instrucciones de procesamiento.
 - **Cuerpo**
 - Obligatorio.



XML

- El **Cuerpo** contiene:
 - **Elementos** (formando una estructura jerárquica)
 - Los elementos XML pueden tener contenido (más elementos, atributos o ambos), o bien ser vacíos.
 - Ej: `<aviso tipo="emergencia">Que no cunda el pánico</aviso>`
 - Ej vacío: `<identificador DNI="23123244"/>`
 - **Atributos**
 - Los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento.
 - En el DTD o XSD se especifican los atributos que puede tener cada tipo de elemento, así como sus tipos.
 - Ej: `<cita texto="'Hola buenos días', dijo él" >`



XML

- El **Cuerpo** contiene (cont.):
 - **Entidades predefinidas**
 - Para representar caracteres especiales para que, de esta forma, no sean interpretados como marcado por el procesador XML.

Entidad	Caracter
&	&
<	<
>	>
'	'
"	"

- **Secciones CDATA**
 - Construcción para especificar datos, utilizando cualquier carácter especial o no, sin que se interprete como marcado XML.
 - Ej: `<![CDATA[contenido especial: áéíóúñ&]]>`
- **Comentarios**
 - Ej: `<!-- Esto es un comentario --->`



XML

Extensiones

- **Extensiones** del propio XML
 - Amplían las capacidades del XML original, no son un lenguaje XML en sentido estricto.
 - Se definen de forma similar al XML original, como un subconjunto de SGML.
 - Todas ellas deben ser compatibles entre sí.
- **Lenguajes XML (Aplicaciones XML)**
 - Lenguajes definidos en base a XML y sus extensiones.
 - Cada lenguaje se define mediante una gramática que consiste en un tipo de documento XML (DTD o XSD).
 - Pueden ser:
 - **Horizontales**: resuelven cierta funcionalidad que es útil en general.
 - SOAP (*Simple Object Access Protocol*)
 - **Verticales**: dirigidos a un sector o utilidad particular.
 - CML (*Chemical Markup Language*)



- El éxito de XML ha originado que se demanden **nuevas funcionalidades**, que se abordan definiendo **extensiones adicionales** para:
 - **Estructurar documentos** (*XML Schema*).
 - **Enlaces y direccionamiento** (*XPath, XLink, XPointer*).
 - **Transformación y presentación** (familia *XSL, CSS2*).
 - **Consultas** (*XQuery*).
 - **Programación** (*DOM, SAX*).
 - **Otros** (*Namespaces, XInclude, XBase, ID, ...*).



- **Namespaces in XML**
 - Método para **cualificar elementos y nombres** de atributos de documentos XML, asociándolos con espacios de nombres (*namespaces*) identificados por referencias URI.

```
<x xmlns:edi='http://ecommerce.org/schema'>
</x>
```
 - Sirve para
 - Evitar las colisiones en los nombres de los elementos y atributos.
 - Hacer públicos DTD's, XML Schemas o partes de ellos con fines de reutilización.
 - Ayuda para combinar lenguajes XML.



- **XBase** establece un mecanismo para utilizar **URI's relativos**.

```
<...xml:base="http://www.sitio.es/">  
<...href="~/yo/dir/index.html" .../>
```

equivale a

```
http://www.sitio.es/~yo/dir/index.html
```

- **XInclude** (*XML Inclusions*) provee un modelo de proceso y una sintaxis para hacer **inclusiones**.
 - Facilita la reutilización y modularidad.
 - Permite combinar documentos XML, o construir nuevos documentos XML a partir de otros previos.

```
<pie xmlns:xi="http://www.w3.org/2001/XInclude">  
  <xi:include href="partedoc.xml"/>  
</pie>
```



- La **gramática** de los lenguajes XML, es decir, la estructura y elementos permitidos en los documentos XML, se define mediante
 - **DTD (Document Type Definition)**
 - Documento ASCII plano que especifica tanto los elementos que forman un tipo de documento dado, como las relaciones que se dan entre ellos.
 - **XSD (XML Schema Definition)**
 - Mejoran los DTD's porque están escritos en XML y permiten nuevas características:
 - definir tipos de datos,
 - utilizar espacios de nombre
 - definir intervalos de valores para los atributos y elementos.
 - características OO ...
 - Son extensibles
 - Tienen mayor potencial semántico que los DTD



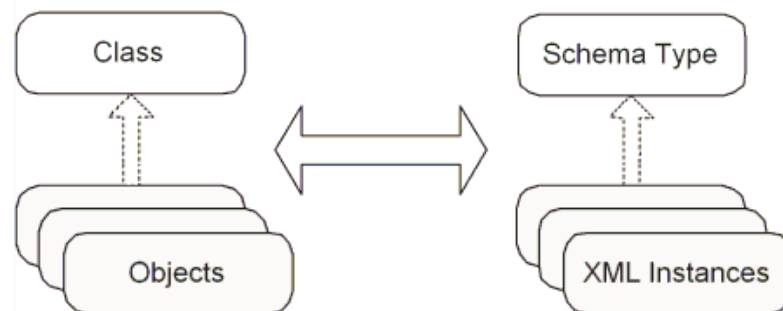
Estructura de Datos y Documentos

- Un lenguaje XML (DTD/XSD) representa un **modelo de datos jerárquico**
 - Estructura los datos de acuerdo a un determinado "esquema semántico".
- Estos lenguajes se definen especificando los elementos y atributos permitidos.
 - Esta especificación se realiza mediante **reglas gramaticales**.
 - Un conjunto concreto y bien formado de tales reglas forman un **esquema XML** (representado por un DTD o un XSD).
 - Un esquema XML define un conjunto coherente de documentos, es decir un **tipo de documentos**.
- Ejemplo:
 - XHTML es el lenguaje HTML reformulado como aplicación XML, y se que se la próxima generación de HTML. Es una versión mas estricta y limpia de HTML..



Esquemas XSD

- **XML Schema Definition (XSD)** provee un sistema de tipos y estructuras permitidas.
 - Se trata de poder definir esquemas en XML con un papel similar al esquema de una base de datos.
 - Los documentos basados en un XSD son instancias **válidas** del esquema definido en el XSD.





- Más información sobre XML Schemas:
 - Estándares del W3C:
 - Primer: <http://www.w3.org/TR/xmlschema-0/>
 - Estructures: <http://www.w3.org/TR/xmlschema-1/>
 - Datatypes: <http://www.w3.org/TR/xmlschema-2/>
 - Referencia de Esquemas XML (XSD)
 - [http://msdn.microsoft.com/es-es/library/ms256235\(VS.80\).aspx](http://msdn.microsoft.com/es-es/library/ms256235(VS.80).aspx)
 - XML Schema Design Patterns: Avoiding Complexity
 - <http://msdn.microsoft.com/en-us/library/aa468564.aspx>
 - X



- Los **esquemas XSD** (XML Schema Definition) soportan:
 - Declaración de tipos.
 - Enteros, cadenas, etc.
 - Restricciones de valores mínimos/máximos.
 - Tipos definidos por el usuario.
 - Tipos específicos mediante la restricción de otros tipos.
 - Extensión de tipos complejos mediante una especie de "herencia".
 - Restricciones de unicidad.
 - Claves ajenas (parecido)
 - Espacios de nombres (diferentes partes de un documento pueden adaptarse a esquemas diferentes).



- Los **esquemas** se definen como documentos XML en un documento con extensión **.XSD**

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="aviso">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="para" type="xsd:string"/>
        <xsd:element name="de" type="xsd:string"/>
        <xsd:element name="cabecera" type="xsd:string"/>
        <xsd:element name="texto" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```



- Los archivos XML que se basan en un esquema XSD deben incluir la referencia a dicho esquema.

```
<?xml version="1.0"?>
<aviso xmlns="http://www.misesquemas.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.misesquemas.com/aviso.xsd">
  <para>Luis</para>
  <de>Juán</de>
  <cabecera>Recordar</cabecera>
  <texto>
    !Acuérdate de llamarme este fin de semana!
  </texto>
</aviso>
```



- Los **elementos** utilizados en la creación de un esquema "proceden" de un espacio de nombres oficial:

<http://www.w3.org/2001/XMLSchema>

- El elemento *schema* es el elemento raíz del documento en el que se define el esquema:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
</xsd:schema>
```



- Un **elemento simple** sólo puede contener texto (cualquier tipo de dato), pero no otros elementos ni atributos.

- Definición: `<xsd:element name="xxx" type="yyy"/>`

- Ejemplos:

- `<xsd:element name="apellido" type="xsd:string"/>`

- `<xsd:element name="edad" type="xsd:integer"/>`

- Tipos más usados:

- string, decimal, integer, boolean, date, time

- Valor por Defecto:

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

- Valor Fijo:

- `<xsd:element name="estado" type="xsd:string" fixed="activo"/>`



- **Elementos Referenciados**

- Los elementos de más alto-nivel pueden ser referenciados cuando se declaran elementos de nivel local (no se necesitan los atributos name y type):

```
<xsd:element name="day" type="xsd:string"/>
```

```
<xsd:complexType name="Date">
```

```
...
```

```
</xsd:element ref="day"/>
```

```
...
```

```
</xsd:complexType>
```



- Los **atributos** se declaran de forma similar a los "elementos simples":

```
<xsd:attribute name="xxx" type="yyy"/>
```

- Ejemplo:

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

- Pueden tener valores por defecto y fijos (idem elementos).
- Por defecto son opcionales. Para indicar su obligatoriedad:

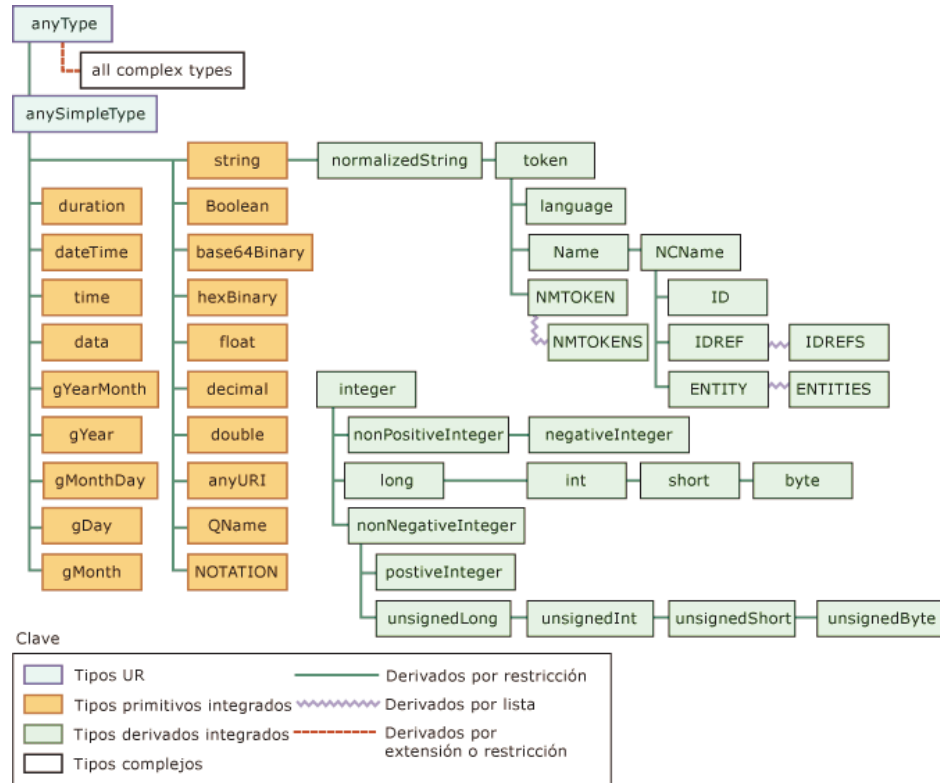
```
<xsd:attribute ... use="required"/>
```

```
<xsd:attribute ... use="optional"/>
```

- Un elemento con atributos se debe declarar como elemento "complejo".



XML Esquemas XSD – Tipos



Francisco Ruiz - BDA

5.37



XML Esquemas XSD – Tipos Simples

- Los **tipos simples** nuevos se definen restringiendo los tipos simples existentes.
 - Se utiliza el elemento **simpleType** para definir y nombrar el tipo simple nuevo.
 - Se utiliza el elemento **restriction** para indicar el tipo (base) existente y para identificar las "propiedades" que restringen el rango de valores.
 - Ejemplo (tipo definido por intervalo de valores):

```
<xsd:simpleType name="miEntero">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="10000"/>
    <xsd:maxInclusive value="99999"/>
  </xsd:restriction>
</xsd:simpleType>
```

Francisco Ruiz - BDA

5.38



- **Ejemplos** de Definiciones de **Tipos Simples**
 - Mediante un patrón de caracteres (string formado por tres dígitos seguidos de un guión seguido de dos caracteres ASCII en mayúsculas):


```
<xsd:simpleType name="codigo">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>
```
 - Por enumeración:


```
<xsd:simpleType name="estadoCivil">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Soltero"/>
    <xsd:enumeration value="Casado"/>
    <xsd:enumeration value="Viudo"/>
  </xsd:restriction>
</xsd:simpleType>
```



- Las **restricciones** pueden incluir facetas que permiten restringir el valor que se puede dar a cualquier elemento o atributo.

```
<xsd:simpleType name="miTipo">
  <xsd:restriction base="TipoSimple">
    <faceta1 value="..."/>
    <faceta2 value="..."/>
    <faceta3 value="..."/>
  </xsd:restriction>
</xsd:simpleType>
```

Tipos string	Tipos numéricos
length	minInclusive
minLength	maxInclusive
maxLength	minExclusive
pattern	maxExclusive
enumeration	totalDigits
whiteSpace	fractionDigits



- La faceta **pattern** permite definir expresiones regulares.

*	Cero o más caracteres
+	Uno o más caracteres
?	Cero o un carácter
.	Cualquier carácter
(a b)	'a' o 'b'
(abc)	Secuencia de 'a', 'b' y 'c'
[abc]	Un carácter a elegir entre 'a', 'b' o 'c'
[^a-z]	Un carácter que no esté en el rango entre 'a' y 'z'
(expr){n}	Expresión repetida n veces
(expr){m,n}	Expresión repetida entre m y n veces
\p{X}	Un carácter de la clase Unicode X



- Además de las restricciones, existen otros dos constructores para tipos simples:
 - **List**: Lista de otro tipo simple separada por espacios.
 - **Union**: Union de dos o más tipos simples.
- Ejemplo de **Lista** de Enteros
 - Definición:

```
<xs:simpleType name='listaDeEnteros'>  
  <xs:list itemType='integer'/>  
</xs:simpleType>
```
 - Uso:

```
<mielemento listaDeEnteros='1 100 9 4000 0'>
```



- Ejemplo de **Union** de dos Tipos Simples

```
<xs:attribute name="tamanoletra" >
  <xs:simpleType>
    <xs:union memberTypes="fuentenumero fuentetexto" />
  </xs:simpleType>
</xs:attribute>

<xs:simpleType name="fuentenumero">
  <xs:restriction base="xs:positiveInteger">
    <xs:maxInclusive value="72"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="fuentetexto">
  <xs:restriction base="xs:string">
    <xs:enumeration value="pequeño"/>
    <xs:enumeration value="medio"/>
    <xs:enumeration value="grande"/>
  </xs:restriction>
</xs:simpleType>
```



- Un **Elemento Complejo** contiene a otros elementos hijos o tiene atributos.
 - Se utiliza el elemento **complexType** para definir la estructura de un tipo complejo.

```
<xsd:element name="persona">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="nombre_pila" type="xsd:string"/>
      <xsd:element name="apellidos" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```



Esquemas XSD – Tipos Complejos

- Una notación alternativa permite nombrar directamente el elemento **complexType**:

```
<xsd:complexType name="persona">
  <xsd:sequence>
    <xsd:element name="nombre_pila" type="xsd:string"/>
    <xsd:element name="apellidos" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

- Con esta notación es posible reutilizarlo como tipo en otros elementos:

```
<xsd:element name="empleado" type="persona"/>
<xsd:element name="estudiante" type="persona"/>
```



Esquemas XSD – Tipos Complejos

- Las facetas **minOccurs** y **maxOccurs** se utilizan para indicar el número máximo y mínimo de veces que puede aparecer un elemento hijo en un elemento complejo.

```
<xsd:element name="persona">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="nombre" type="xsd:string"/>
      <xsd:element name="hijos" type="xsd:string" maxOccurs="10"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

- **maxOccurs** puede tomar el valor "unbounded", que indica que no existe ningún límite.



- Elemento vacío con atributos

- Definición:

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

- Uso:

```
<producto id="1345"/>
```



- Elemento no vacío con atributos y sin hijos

- Definición:

```
<xsd:element name="tallazapato">  
  <xsd:complexType>  
    <xsd:simpleContent>  
      <xsd:extension base="xsd:integer">  
        <xsd:attribute name="pais" type="xsd:string"/>  
      </xsd:extension>  
    </xsd:simpleContent>  
  </xsd:complexType>  
</xsd:element>
```

- Uso:

```
<tallazapato pais="UK">9.5</tallazapato>  
<tallazapato pais="SP">43</tallazapato>
```



Esquemas XSD – Tipos Complejos

- Un **elemento mixto** mezcla datos estructurados (en elementos) y texto libre en su contenido.

- Definición:

```
<xsd:element name="carta">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element name="nombre" type="xsd:string"/>
      <xsd:element name="npedido" type="xsd:positiveInteger"/>
      <xsd:element name="fecha" type="xsd:date"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

- Uso:

```
<carta>Estimado cliente:
  <nombre>Juan Perez</nombre>. Su pedido número
  <npedido>1032</npedido> se enviará el día
  <fecha>2001-07-13</fecha>.
</carta>
```



Esquemas XSD – Tipos Complejos

- Es posible utilizar un mecanismo de **herencia** para reutilizar o extender tipos previamente definidos.

```
<xsd:element name="empleado" type="personatodo"/>
<xsd:complexType name="persona">
  <xsd:sequence>
    <xsd:element name="nombre" type="xsd:string"/>
    <xsd:element name="apellidos" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="personatodo">
  <xsd:complexContent>
    <xsd:extension base="persona">
      <xsd:sequence>
        <xsd:element name="calle" type="xsd:string"/>
        <xsd:element name="ciudad" type="xsd:string"/>
        <xsd:element name="pais" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

*"personatodo"
extiende
"persona" con
tres nuevos
elementos*



- Existen tres maneras (*model groups o compositors*) de estructurar los elementos de un tipo complejo:
 - **Secuencia**: se debe respetar el orden en el que se definen los elementos anidados.
 - `xsd:sequence`
 - **Conjunción**: los elementos pueden aparecer en cualquier orden, pero solo una vez cada uno.
 - `xsd:all`
 - **Disyunción**: solo puede aparecer uno de los elementos.
 - `xsd:choice`.



- Ejemplo de Composición de Conjunción:

```
<xsd:element name="persona">  
<xsd:complexType>  
  <xsd:all>  
    <xsd:element name="nombre" type="xsd:string"/>  
    <xsd:element name="apellidos" type="xsd:string"/>  
  </xsd:all>  
</xsd:complexType>  
</xsd:element>
```



- Ejemplo de Composición de Disyunción:

```
<xsd:element name="estudiante">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="tipobeca" type="xsd:string"/>
      <xsd:element name="formapago" type="xsd:string"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```



- Es posible crear **agrupaciones** de elementos o de atributos para hacer referencias a ellas:

- Definición:

```
<xs:group name="nombApell">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
  </xs:sequence>
</xs:group>
```

- Uso:

```
<xs:complexType name="TipoAlumno">
  <xs:group ref="nombApell" />
  <xs:element name="carrera" type="xs:string"/>
</xs:complexType>
```



- En relación con los conceptos de **clave** de identificación y de **unicidad**, habituales en bases de datos, XML Schema ofrece las siguientes opciones:
 - Indicar que un elemento es único (UNIQUE).
 - Definir atributos únicos.
 - Definir combinaciones de elementos y atributos como únicos
 - Distinción entre unicidad y claves (KEY)
 - Clave => además de ser único, debe existir y no puede ser nulo.
 - Declarar el rango de un documento en el que algo es único.



- **Claves - KEY**
 - Pueden definirse para elementos y para atributos
 - Incluyen un espacio de nombres XPath

```
<xs:complexType name="Alumnos">  
  <xs:sequence>  
    <xs:element name="Alumno" type="TipoAlumno"/>  
  </xs:sequence>  
  <xs:key name="ClaveDNI">  
    <xs:selector xpath="a:alumno"/>  
    <xs:field xpath="a:dni"/>  
  </xs:key>  
</xs:complexType>
```
 - Pueden estar formadas por varios elementos.



- **Unicidad - UNIQUE**

- A diferencia de KEY, UNIQUE indica que se debe cumplir la restricción de unicidad (no puede haber dos valores iguales), pero puede no existir.

```
<xs:complexType name="Alumnos">
  <xs:sequence>
    <xs:element name="Alumno" type="TipoAlumno"/>
  </xs:sequence>
  <xs:unique name="DNI">
    <xs:selector xpath="a:alumno"/>
    <xs:field xpath="a:dni"/>
  </xs:unique>
</xs:complexType>
```



- **KEYREF** especifica una referencia a una clave que identifica un elemento (similar a una *clave ajena*).

```
<xs:element name="clase">
  <xs:sequence>
    <xs:element name="alumnos" ...
    <xs:element name="delegado" ...
  </xs:sequence>

  <xs:key name="dni">
    <xs:selector xpath="a:alumnos/a:alumno"/>
    <xs:field xpath="a:dni"/>
  </xs:key>

  <xs:keyref name="delegado" refer="dni">
    <xs:selector xpath="a:delegado"/>
    <xs:field xpath="a:dni"/>
  </xs:keyref>
```



- **Inclusión**

- **INCLUDE** permite incluir elementos de otros esquemas.
- Los elementos deben estar en el mismo espacio de nombres.
- Es como si se hubiesen tecleado todos en un mismo fichero.
- Ejemplo:

```
<xsd:schema ...>  
  <xsd:include schemaLocation="Alumnos.xsd"/>  
  <xsd:include schemaLocation="Profesores.xsd"/>  
  ...  
</xsd:schema>
```



- **Importación**

- **IMPORT** permite incluir elementos de esquemas provenientes de otros espacios de nombres.
- Ejemplo:

```
<xsd:schema ...>  
  <xsd:import namespace="A" schemaLocation="Alumnos.xsd"/>  
  <xsd:import namespace="P" schemaLocation="Profes.xsd"/>  
  ...  
</xsd:schema>
```



- **Redefinición**

- **REDEFINE** es similar a INCLUDE pero permite modificar los elementos incluidos.

- Ejemplo:

```
<xs:redefine schemaLocation="Alumnos.xsd">
  <xs:complexType name="TipoAlumno">
    <xs:complexContent>
      <xs:extension base="TipoAlumno">
        <xs:sequence>
          <xs:element name="nota" type="Nota" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:redefine>
```



- Los **esquemas XML** (archivos XSD) tienen algunas **limitaciones**:

- No soportan entidades (mecanismo para crear macros existente en DTDs)
`<!ENTITY &texto; "Esto texto se repite muchas veces" >`
- Lenguaje de Restricciones limitado
 - ¿Verificar valor total = suma de valores parciales?
- Sensibilidad al contexto limitada
 - Especificar que el contenido depende del valor de un atributo
`<transporte tipo="coche"> ...</transporte>`
`<transporte tipo="avión"> ...</transporte>`
- El tamaño de archivos XML Schema puede ser muy grande.
- La legibilidad de las especificaciones no siempre es buena.
- Complejidad de la especificación
 - Muchas situaciones/combinaciones excepcionales



- **XML Path Language (XPath)** es un lenguaje declarativo para **localizar nodos y fragmentos** (texto, elementos, atributos ...) en el árbol de un documento XML.
- Es utilizado por otras normas para
 - Direccionamiento (XLink, XPointer y XSLT)
 - "*Pattern matching*" (XSLT y XQuery)
- Se basa en el **XPath Data Model**:
 - Un documento XML se representa como un **árbol jerárquico** con siete tipos de nodos (raíz, elemento, texto, atributo, espacio de nombres, instrucción de procesamiento y comentario).
- Conceptos importantes:
 - **Caminos** de localización / libro / capítulo / párrafo
 - **Predicados**



- **Ejemplos XPath:**
 - Seleccionar nombres de ingredientes de receta que se utiliza media taza:
`//ingrediente[@cantidad='0.5' and @unidad=taza]/@nombre`
 - Seleccionar todos los capítulos públicos que tengan algún párrafo que contenga algún elemento con atributo href:
`//capitulo[parrafo/*[@href]][@public='si']`
 - Seleccionar todos los capítulos públicos que tengan algún párrafo importante o un apéndice:
`//capitulo[parrafo/[@importante='si']]//apendice`
 - Mas ejemplos:
[http://msdn.microsoft.com/es-es/library/ms256086\(VS.80\).aspx](http://msdn.microsoft.com/es-es/library/ms256086(VS.80).aspx)



- **XPointer:**

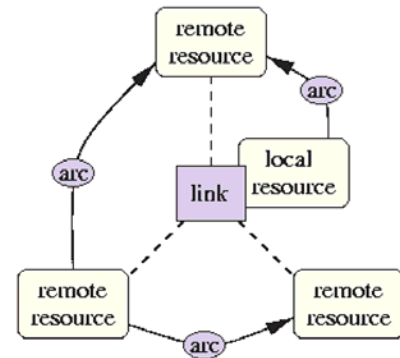
- Describe cómo se puede apuntar a un lugar específico de un determinado documento XML.
- Es una extensión de XPath que permite asociar a una dirección URI con una expresión XPath con algunas propiedades extras.

[http://www.sitio.es/doc.xml#xpointer\(/libro/capitulo\[@public\]\)](http://www.sitio.es/doc.xml#xpointer(/libro/capitulo[@public]))

- Nuevos conceptos:
 - Puntos (para trabajar a nivel de caracteres).
 - Rangos (para trabajar a nivel de palabras, subcampos de fechas, ..).



- **XLink** (*XML Linking Language*) define la forma en la que los documentos XML se pueden relacionar entre sí definiendo nuevos tipos de elementos XML que representan **enlaces** (links).
 - Utiliza XPointer para localizar recursos.
 - Características especiales:
 - Uso de "alias".
 - Asociaciones entre más de 2 recursos (enlaces multidireccionales).
 - Un origen y varios destinos.
 - Enlaces agregados (varios orígenes, un sólo destino)
 - Asociar metadatos a un enlace.
 - Expresar enlaces que residen fuera de los recursos enlazados.



- Existen 2 tipos de enlaces:

- Simples

- <AUTOR xlink:href="autores.xml#juan" xlink:show="new">

- <NOMBRE>Juan Primero Segundo</NOMBRE>

- </AUTOR>

- Extendidos

- <EDITOR_AUTOR xlink:extended>

- <xlink:locator href="#ana" id="editor"/>

- <xlink:locator href="autores.xml#juan" id="autor"/>

- <xlink:arc from="editor" to="autor" show="replace"/>

- </EDITOR_AUTOR xlink:extended>



- **XSL** (*Extensible Stylesheet Language*)

- Es una familia de recomendaciones del W3C para definir transformaciones y presentaciones de documentos XML.

- Consta de tres partes:

- **XSL Transformations (XSLT)**: Un lenguaje para definir transformaciones.

- **XML Path Language (XPath)**: Ya presentado, usado por XSLT para acceder o referir partes de documentos XML.

- **XSL Formatting Objects (XSL-FO)**: Un vocabulario XML para especificar el formato visual.

- Una **hoja de estilos XSLT** especifica la presentación de una clase de documentos XML mediante la descripción de la manera en que una instancia de la clase es transformada en otro documento (XML o de otro formato).

- Mas información:

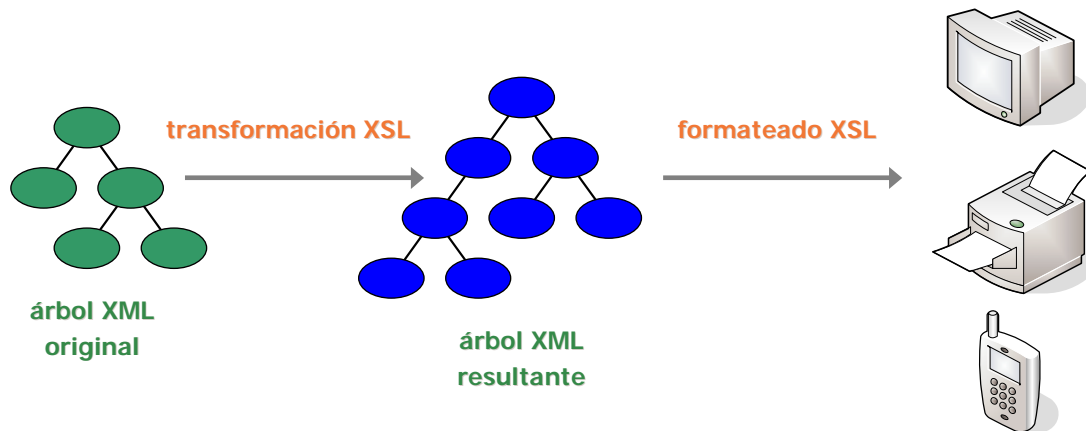
- <http://www.w3.org/Style/XSL/>

- <http://msdn.microsoft.com/en-us/library/ms256069.aspx>



Transformación – XSL

- **XSL** (**eXtensible Stylesheet Language**) no sólo permite **definir el estilo** a aplicar a cada elemento XML. También es un lenguaje de programación para **transformar documentos XML**.
 - El resultado puede ser un documento HTML, WML (para WAP), texto plano, RTF, PDF, o incluso otro documento XML.
 - Utiliza XPath para referir partes de documentos XML.



Transformación – XSL

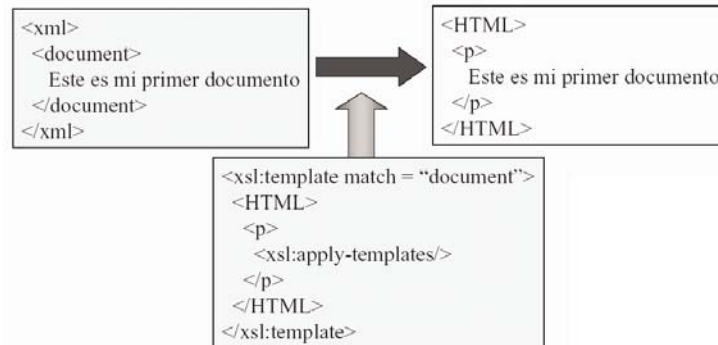
- Una **hoja de estilo XSL** es una serie de **reglas** que determinan cómo va a ocurrir la transformación.
- Cada regla se compone de un **patrón de localización** (*pattern*) y una **plantilla** (*template*).

```
<xsl:template match="/">
<HTML>
<BODY>
<xsl:for-each select="/LIBROS/LIBRO">
Título:
<xsl:value-of select="TITULO"/><BR/>
Autor:
<xsl:value-of select="AUTOR"/><BR/>
Precio:
<xsl:value-of select="PRECIO"/> pesetas<BR/>
</xsl:for-each>
</BODY>
</HTML>
</xsl:template>
```



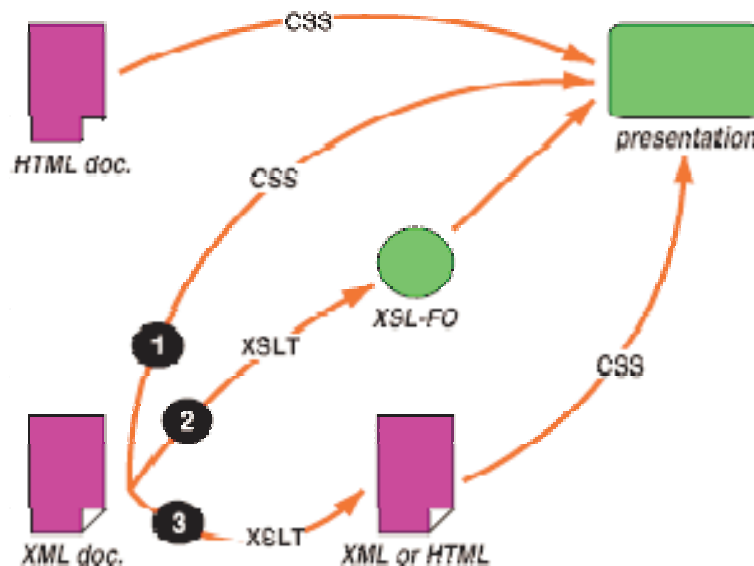
Transformación – XSL

- Las **hojas de estilo XSL** sirven también para especificar el **formato visual**.
 - Ventajas frente a HTML/CSS:
 - Centralizar la forma de presentación (formato)
 - Separar estructura ↔ contenido
 - Reutilización de datos
 - Diferentes formatos de salida para los mismos datos
 - Unificar el estilo de presentación



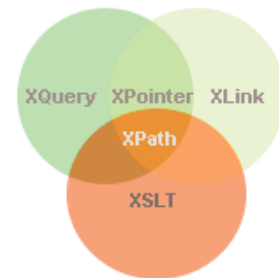
Transformación – XSL

- XSL vs CSS2:





- **XQuery Lenguaje (XQL)**
- Proporciona un modo flexible de **consulta** para extraer datos de los documentos XML.
 - Los archivos XML pueden ser **reales o virtuales**, es decir, otras fuentes (hojas de cálculo, ASCII, bases de datos, ...) vistas como datos XML.
 - Se pretende que desempeñe un papel similar al SQL en las BD relacionales: las colecciones de documentos XML podrán ser accedidas como si fueran una base de datos.
 - Esta basado en varias propuestas de lenguajes previas (XML-QL, YATL, Lorel, Quilt).
 - Se ha integrado con XPath (versión 2.0).
 - Existe un cierto solape con XSLT.
 - Una consulta puede referir a más de un documento.



- Una **consulta** es una expresión que:
 - **Lee** una secuencia de fragmentos XML o valores atómicos y
 - **Devuelve** una secuencia de fragmentos XML o valores atómicos.
- Los principales **tipos de expresiones** son:
 - Expresiones **XPath**, para navegar por los documentos.
 - **Constructores** de elementos.
 - **FLWR** (FOR, LET, WHERE, RETURN) para iterar por los elementos de una colección.
 - **Condicionales** (IF, THEN ELSE) para construir el resultado en base a alguna condición.
 - Con **cuantificadores** (SOME, ANY) para chequear la existencia de algún elemento que cumpla una condición.
 - **Listas** a las que se pueden aplicar operadores (UNION, ...) y funciones (AVG,...).



- **Ejemplo de consulta XQuery:**

- Obtener el año y título de todos los libros publicados por Addison-Wesley después de 1991:

```
<bib>
{
  for $b in doc("http://www.bn.com/bib.xml")/bib/book
    where $b/publisher="Addison-Wesley" and $b/@year>1991
  return
    <book year="{ $b/@year }">
      { $b/title }
    </book>
}
</bib>
```



- **¿Qué son las Bases de Datos XML?**

- Un **documento XML** es una BD porque
 - Almacena información (documentos).
 - Responde a un esquema (DTD, XML Schema)
 - Tiene lenguajes de consulta (XPah, XQuery)
 - y API's de Programación (SAX, DOM, JDOM...)
- **Ejemplos** de posibles BD XML:
 - Fichero de configuración de una aplicación
 - Plantilla de un fax
 - Formulario para solicitar dietas de viajes
 - Temario de una asignatura
 - Todos los informes de un departamento
- En general, una BD XML es una BD almacenada o gestionada en forma de documentos XML.



XML y Bases de Datos

- ¿**Porqué** surgen las BD XML?
 - Necesidad de **almacenar y recuperar datos poco estructurados** con la eficiencia de las BD convencionales.
 - Aparecen dos clases de SGBD que soportan documentos XML
 - **XML-Enabled**: desglosan un documento XML en su correspondiente modelo relacional o de objetos.
 - **XML Nativos**: respetan la estructura de documento, permiten hacer consultas sobre dicha estructura y recuperan el documento tal y como fue insertado originalmente
 - En una BD XML es posible caracterizar tres tipos de archivos XML:
 - Centrados en **Datos**.
 - Centrados en **Documentos**.
 - **Híbridos** (mezcla partes de los dos tipos anteriores, BD XML nativa).



XML y Bases de Datos – tipos de archivos

- **Centrados en Datos**
 - **Muchos elementos** de datos de pequeño tamaño.
 - Con estructura regular y bien definida.
 - Datos **muy estructurados** o semi-estructurados.
 - Usados como mecanismo de intercambio o para mostrar datos en la web.
 - Dirigidos a **utilización automática** (por máquinas).
 - Ejemplos: Facturas, Pedidos.
- **Centrados en Documentos**
 - **Pocos elementos**.
 - Con grandes cantidades de texto.
 - Con estructuras impredecibles en tamaño y contenido.
 - Datos **poco estructurados**.
 - Orientados a ser interpretadas por **humanos**.
 - Enfocados a sistemas documentales y de gestión de contenidos.
 - Ejemplos: Libros, Informes.



XML y Bases de Datos – tipos de archivos

- Ejemplo de archivo centrado en datos
 - Datos de Matrícula de Alumnos

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<matricula xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="prueba.xsd">
  <personal>
    <dni>52945813C</dni>
    <nombre>Jose Perez García</nombre>
    <titulacion>Ingeniería Informática (Plan 1998)</titulacion>
    <curso>2004/2005</curso>
    <domicilios>
      <domicilio tipo="familiar">
        <direccion>C/ Principal nº1</direccion>
      </domicilio>
      <domicilio tipo="habitual">
        <direccion>C/ Secundaria nº2</direccion>
      </domicilio>
    </domicilios>
  </personal>
  <pago>
    <tipo_matricula>Ordinaria</tipo_matricula>
  </pago>
  ...
</matricula>
```



XML y Bases de Datos – tipos de archivos

- Ejemplo de archivo centrado en datos
 - XML-Schema para Matrícula de Alumnos

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xml:lang="ES">
  <xs:element name="matricula" type="tMatricula"/>
  <xs:complexType name="tMatricula">
    <xs:sequence>
      <xs:element name="personal" type="tPersonal"/>
      <xs:element name="pago" type="tPago"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="tPersonal">
    <xs:all>
      <xs:element name="dni" type="xs:string"/>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="titulacion" type="xs:string"/>
      <xs:element name="curso_academico" type="xs:string"/>
      <xs:element name="domicilios" type="tDomicilio"/>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="tPago">
    <xs:all>
      <xs:element name="tipo_matricula"
        type="xs:string"/>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="tDomicilio">
    <xs:sequence>
      <xs:element name="domiclio"
        maxOccurs="unbounded">
      <xs:complexType>
        <xs:all>
          <xs:element name="direccion"
            type="xs:string"/>
        </xs:all>
      </xs:complexType>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```



XML y Bases de Datos – tipos de archivos

- Ejemplo de archivo centrado en documentos
 - Libro El Quijote

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

```
...
```

```
<libro titulo="El Quijote">
```

```
<introduccion>
```

```
...
```

```
</introduccion>
```

```
<capitulo numero="1">
```

En un lugar de la Mancha de cuyo nombre no quiero acordarme, no ha mucho tiempo vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de algo mas de vaca que carnero, salpicón por las noches, duelos y quebrantos los sábados, lantejas los viernes, algún palomino de añadidura los domingos, consumían las tres partes de su hacienda. El resto de ella concluían sayo de velarte, calzas de velludo para las fiestas, con sus pantuflos de lo mismo, y los días de entresemana se honraba con su vellorí de lo mas fino. Tenía en su casa una ama que pasaba de los cuarenta y una sobrina que no llegaba a los veinte, y un mozo de campo y plaza que así ensillaba el rocín como tomaba la podadera. Frisaba la edad de nuestro hidalgo de los cincuenta años. Era de complexión recia , seco de carnes, enjuto de rostro, gran madrugador y amigo de la caza. Quieren decir que tenía el nombre de Quijada o Quesada, que en esto hay alguna diferencia en los autores que deste caso escriben; aunque por conjeturas verisímiles, se deja entender que se llamaba Quijana. Pero esto importa poco a nuestro cuento; basta que la narración del no salga un punto de la verdad ...

```
</capitulo>
```

```
</libro>
```



XML y Bases de Datos – almacenamiento

- Existen varias aproximaciones para organizar y **almacenar documentos XML** de cara a su consulta y recuperación:
 - Usar un SGBD para almacenar los **documentos XML como texto**.
 - Se almacenan documentos XML completos como textos muy largos en columnas de tipos carácter largo (SGBD objeto-relacional) o en objetos de clase texto (SGBD-OO).
 - Usar un SGBD para **almacenar los elementos XML** de los documentos como elementos de datos.
 - Si todos los documentos XML tienen una estructura basada en un DTD/Schema, es posible volcar sus partes a estructuras relacionales o a objetos de un SGBD.
 - Diseñar un nuevo Sistema de BD para almacenar documentos XML de forma directa (**BD XML nativa**).
 - Generar los documentos XML como **capa de interfaz** de datos almacenados en BD tradicionales (relacionales u OO).



XML y Bases de Datos – almacenamiento

- **Almacenar documentos XML completos.**
 - Se utilizan bases de datos relacionales, objeto-relacionales o de objetos puras para representar documentos XML.
 - Se almacena el documento completo en una columna de tipo Binario Largo (**BLOB**) o de un nuevo tipo **XML**.
 - Sirve para documentos estáticos que no se modifican casi nunca y cuando lo hacen se vuelven a grabar completamente.
 - No necesita mapeo.
 - El documento permanece íntegro.
 - Limita las búsquedas por contenido y la indexación.



XML y Bases de Datos – almacenamiento

- **Almacenar elementos XML.**
 - Se utilizan BD tradicionales para asociar cada elemento de cada documento XML con alguna estructura de la BD.
 - Se plantea un problema importante de **Mapeo** (correspondencia) entre los elementos XML y las estructuras de la BD.
 - Los desajustes se deben a **modelo de datos diferentes**:
 - XML => Jerárquico
 - SGBD => Relacional / Objetos
 - Las ventajas vienen de que se pueden aprovechar las capacidades del SGBD donde se almacenan:
 - Búsquedas, indexaciones, optimizaciones, etc.
 - La principal desventaja es que **la integridad del documento no se mantiene**.



- **Mapeo y Traducción.**

- Mapeo **basado en Tablas.**

- Modela el documento XML en un conjunto de tablas en un modelo relacional.
- Los atributos XML se almacenan en una columna de una tabla.
- Para recuperar un documento XML se debe realizar un proceso de **serialización**.
- Distintos esquemas XML pueden generar el mismo esquema de BD relacional.

- Mapeo **basado en Interrelaciones entre Objetos.**

- Cada tipo de elemento XML de más alto nivel se modela como una clase de objetos.
- Los atributos XML se modelan como atributos de las clases.



“Las bases de datos XML nativas son BD que almacenan XML usando un formato que permite un procesamiento más rápido” (DBXml Group)

- Almacenan la información en formato nativo.
- No traducen el XML a estructuras relacionales u objetos.
- Sus esquemas permiten reglas de almacenamiento e indexación.
- Todos los documentos son accesibles mediante una URL.
- Mantienen el modelo XML intacto.



Sistemas de BD Nativos XML

- Crean **modelos lógicos** en XML.
- Mapean los modelos al mecanismo de almacenamiento correspondiente.
- Las **operaciones** con los documentos se realizan **en XML**.
- Dan un mayor nivel de abstracción al programador.
- Dependencia del esquema.
 - Gestionan documentos como colecciones de datos.
 - No todas necesitan un esquema para almacenar documentos.
 - Problemas de integridad intra-documento.
 - Los esquemas se definen con DTD o con XML Schema (W3C).
- No usan un mecanismo concreto de almacenamiento físico.
 - Depende de cada producto.
- La unidad mínima de almacenamiento es un documento XML.
- Existen retos pendientes para la integridad global de la BD.
 - Integridad referencial inter-documento.
 - Restricciones semánticas inter-documento.



Sistemas de BD Nativos XML

- **Ventajas**
 - No necesitan mapeos adicionales.
 - Conservan la integridad de los documentos.
 - Permiten almacenar documentos heterogéneos en la misma colección.
- **Ámbito de Uso**
 - Documentos con anidamientos profundos.
 - Importancia de preservar la integridad de los documentos.
 - Sistemas con XML orientado a los documentos.
 - Búsquedas de contenido.
- **Áreas de Aplicación**
 - Portales de información corporativa.
 - Catálogos de Datos.
 - Almacenamiento de información médica.
 - BD de personalización.



- Algunos **productos comerciales**:
 - 4 Suite
 - dbXML
 - Virtuoso
 - x-Hive/DB
 - eXist
 - BirdStep RDM XML
 - TotalXML
 - OpenLink
 - Tamino



- **Tamino XML Server**
- Características principales:
 - Almacenamiento nativo XML
 - Riqueza de tipos de datos multimedia (audio, video, imagen)
 - Consultas basadas en XPath
 - Búsqueda y Recuperación de textos completos
 - Definición de datos extensible (cambios dinámicos)
 - Salidas multi-canal (HTML, WML, PDF, ..)
 - Consolidación de datos (acceso a datos externos)
 - Extensiones para integración (EJB, CORBA, DOM, ERP, Servlets, ..)
 - Rendimiento
 - Escalabilidad
 - Fiabilidad
 - Disponibilidad

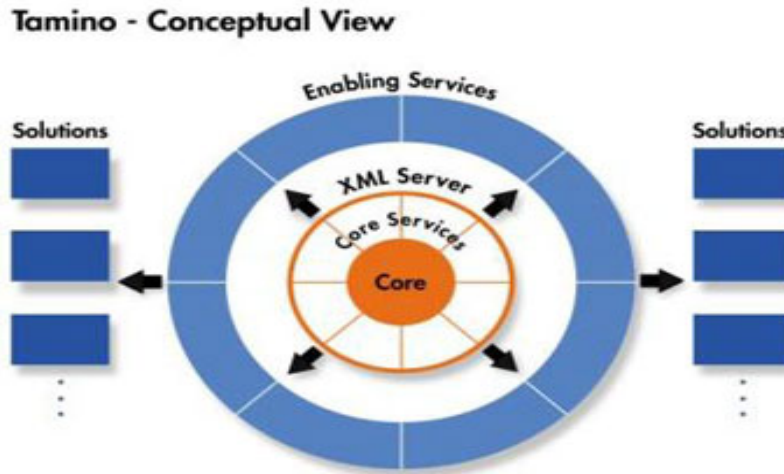


XML y Bases de Datos

Sistemas de BD Nativos XML - productos

- **Tamino XML Server**

- Conceptualmente tiene tres capas:
 - Servicios Centrales (Core Services), ej: almacenamiento
 - Servicios Funcionales (Enabling Services), ej: consolidación
 - Servicios de Terceros (Solutions), ej: publicación electrónica



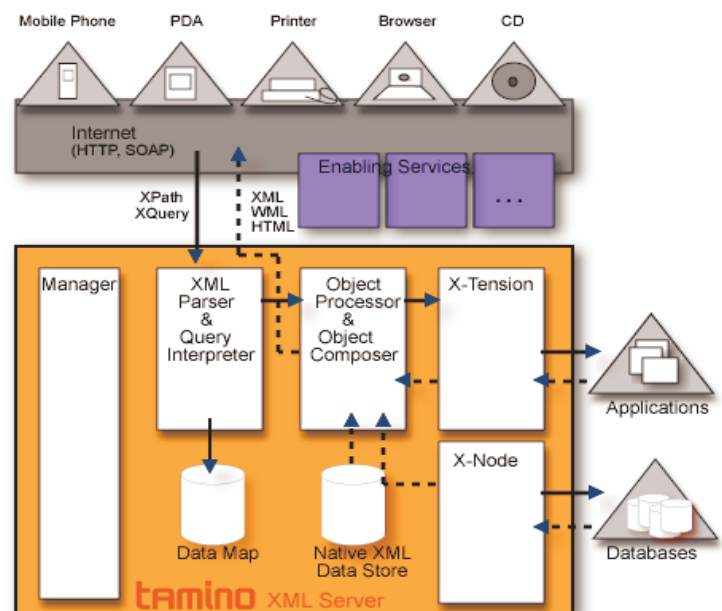
XML y Bases de Datos

Sistemas de BD Nativos XML - productos

- **Tamino XML Server**

- **Arquitectura:**

- Tamino XML Server
 - XML Data Store Nativo
 - Data Map
 - X-Node
 - X-Tension
 - Tamino Manager
- Otros
 - Tamino Schema Editor
 - Tamino Interactive Interface
 - Tamino X-Plorer
 - Tamino WebDAV Server
 - APIs
 - Tamino X-Application





- SGBD con **extensión para XML**:
 - Oracle (8i en adelante)
 - IBM DB2
 - Microsoft SQL Server 2000 (en adelante)
 - INFORMIX
- Inclusión de **XML en SQL**:
 - Nueva parte del estándar para crear y manipular documentos XML.
 - **ISO/IEC 9075-14**: XML-Related Specifications (**SQL/XML**).
 - Mas info: <http://sqlx.org/>



- **SQL/XML** permite:
 - Almacenar documentos XML en bases de datos objeto-relacionales.
 - Consultar dichos documentos mediante "XQuery" y "XPath".
 - Publicar o exportar datos objeto-relacionales en forma de documentos XML.
 - Mapeo de Tablas-XML y viceversa.
 - Ej: Patrón una tabla un documento.
- Para ello ofrece:
 - **Nuevo tipo** predefinido.
 - **Nuevos operadores y funciones** predefinidos para crear y manipular valores de tipo XML.
 - **Reglas para mapear** tablas, esquemas y catálogos a documentos XML.



- **ORACLE**
 - **Capacidades**
 - Almacenamiento de documentos XML como columnas.
 - Acceso a documentos XML en fuentes externas.
 - Mapeo de elementos de documentos XML a tablas y columnas.
 - A partir de la versión 9i se incluye un **nuevo tipo de dato** (XMLType) para manejo **nativo** de XML.
 - **Herramientas**
 - [SQL XML para Java](#): clases Java para la inserción de datos XML y generar documentos XML partiendo de SQL.
 - [Servlet Java XQL](#): pasa consultas SQL a XML y posteriormente a HTML mediante hojas de estilo.



- **IBM DB2 (DB2 XML Extender)**
 - Permite **almacenar** los documentos XML como:
 - Datos carácter en una única columna.
 - Mapeando a múltiples tablas y columnas
 - Archivos externos.
 - Pudiendo **recuperar**:
 - Documento completo.
 - Elementos individuales.
 - Atributos del documento.
- **Microsoft SQL Server**
 - Soporte para esquemas XML
 - Ejecución de consultas con XPath
 - Obtener y escribir datos XML
 - Obtención de datos en documentos XML ("FOR XML")
 - Escritura de documentos XML mediante OpenXML, que crea una imagen en memoria.