Modelo semiestructurado y bases de datos orientadas a documentos

XML

- **XML (***eXtensible Markup Language)* traducido como 'Lenguaje de Marcado Extensible', es un metalenguaje que permite definir lenguajes de marcas desarrollado por el *World Wide Web Consortium* (W3C) utilizado para almacenar datos en forma legible.
- Proviene del lenguaje SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML) para estructurar documentos grandes.
- A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones deben comunicarse entre sí o integrar información
- Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

eXtensible Markup Language XML

XML es un metalenguaje universal para definir etiquetas

Proporciona un marco de trabajo uniforme para intercambio de datos y metadatos entre aplicaciones

No obstante, el XML no proporciona ningún medio para hablar de la semántica de los datos, esto es, no hay significados asociados con el anidamiento de las etiquetas

- Cada aplicación debe de interpretar el anidamiento apropiadamente

¿Porque usar XML?

- Es importante porque:
  - Se puede trasladar cualquier dato a XML
  - Se puede enviar XML sobre la Web (HTTP, SOAP)
  - Se puede incrustar XML en cualquier aplicación, lo que significa intercambio de datos en la Web

Estructura de un documento XML

XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible, esto es, con partes bien definidas, y que esas partes se compongan a su vez de otras partes

Una etiqueta consiste en una marca hecha en el documento, que señala una porción de este como un elemento

Un documento XML está formado por el prólogo y por el cuerpo del documento así como texto de etiquetas que refiere el documento

Prólogo

- El prólogo de un documento XML contiene:
  - Una declaración XML. Es la sentencia que declara al documento como un documento XML.
  - Una declaración de tipo de documento. Enlaza el documento con su DTD (definición de tipo de documento), o el DTD puede estar incluido en la propia declaración o ambas cosas al mismo tiempo.
  - Uno o más comentarios e instrucciones de procesamiento.

```
<?xml version="1.0" encoding="UTF-8
| ISO-8859-1 | windows-1252"
standalone="no | yes"?>
```

Cuerpo

- A diferencia del prólogo, el cuerpo no es opcional en un documento XML
- El cuerpo debe contener solo un elemento raíz, característica indispensable también para que el documento esté bien formado.

```
<Elemento_raiz>
   <SubElemento>
      (...)
   </SubElemento>
</Elemento_raiz>
```

Elementos y etiquetas en XML

Etiquetas: libro, titulo, autor, …

- Las etiquetas XML son sensitivas a mayúsculas y minúsculas

Las etiquetas están normalmente en pares de inicio/termino

- <libro> … </libro>

Todo el contenido debe estar entre etiquetas

- <libro>
  - <titulo>Fundamentos de Bases de Datos </titulo>
- </libro>

Se pueden anidar arbitrariamente los elementos

Un elemento vacío se abrevia: <libro/>

Atributos XML

- Son pares de nombre-valor que ocurren dentro de las etiquetas y después del nombre del elemento

```
<libro precio="250" moneda="Pesos">
<titulo>Fundamentos de Bases de datos
</titulo>
<autor>Abiteboul</autor>
…
</libro>
```

- Todos los valores de atributos deben estar encerrados entre comillas dobles
- Un elemento puede tener varios atributos, pero su nombre solo ocurre una vez

Secciones CDATA

- Sintaxis:

```
<![CDATA[ … texto …]]>
```

- Se emplea para indicar al analizador (parser) que no se tome en cuenta el texto de la sección CDATA en el análisis

```
<ejemplo>
    <![CDATA[ algo de texto </nada> ]]>
</ejemplo>
```

Referencias de entidad

- Sintaxis:

&nombre_entidad;

- Se emplea para sustituir el texto indicado por la referencia de entidad

```
<ejemplo> 250 &lt; 200
</ejemplo>
```

- Están definidas las siguientes referencias de entidad:

&lt;	<
&gt;	>
&amp;	&
&apos;	‘
&quot;	“
&#38;	Caracter Unicode

Comentarios

- Sintaxis:

```
<!--    … texto de comentario…   -->
```

- Las etiquetas de comentarios no son analizadas.

Instrucciones de procesamiento

- Proporciona información a una aplicación externa o al procesador de XML

```
<?receptor instruccion?>
```

  - *Receptor* identifica a la instrucción de procesamiento de la aplicación
  - *Instrucción* es un parámetro opcional pasado a la aplicación

```
<?xml-stylesheet type="text/xml"
 href="contactos.xsl"?>
```

Namespaces en XML

- Es un mecanismo para nombrar etiquetas globalmente de forma única:

```
<h:html
        xmlns:xdc="http://www.xml.com/books"
        xmlns:h="http://www.w3.org/HTML/1998/html4">
  <h:head><h:title>Book Review</h:title></h:head>
  ...
  <xdc:bookreview>
    <xdc:title>XML: A Primer</xdc:title>
  ...
 </h:html>
```

- Permite la mezcla de etiquetas de diferentes vocabularios sin confusión
- Los namespaces sólo identifican el vocabulario; es necesario tener mecanismos adicionales para la estructura y el significado de las etiquetas

Documento XML bien formado

El documento inicia con la declaración de XML

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

standalone -> que no tienen una DTD asociada

El documento tiene un elemento principal que contiene a los demás elementos del documento

Todas las etiquetas deben de estar anidadas correctamente

Documento válido

Se ajusta a la DTD (Document Type Definition, Definición de Tipo Documento) declarada en él

Solo los documentos XML con una DTD se dice que son válidos

Un documento XML sin una DTD puede ser válido o no, pero debe ser bien formado

Reglas de la DTD

- Una DTD dice que es permitido dentro de la estructura del documento
  - Que elementos pueden ocurrir
  - Que atributos pueden o deben estar en un elemento
  - Que subelementos pueden o deben ocurrir dentro de cada elemento y cuantas veces
- Aplica a los elementos en el contexto
- Sin embargo, no establece restricciones a los datos
- Sintaxis:

```
<!ELEMENT elemento(subelementos)>
<!ATTLIST elemento(atributos)>
```

- Los subelementos pueden ser especificados como
  - Nombres de elementos, o
  - **#PCDATA** (parsed character data, esto es, cadenas de texto), o
  - **EMPTY** (sin subelementos)

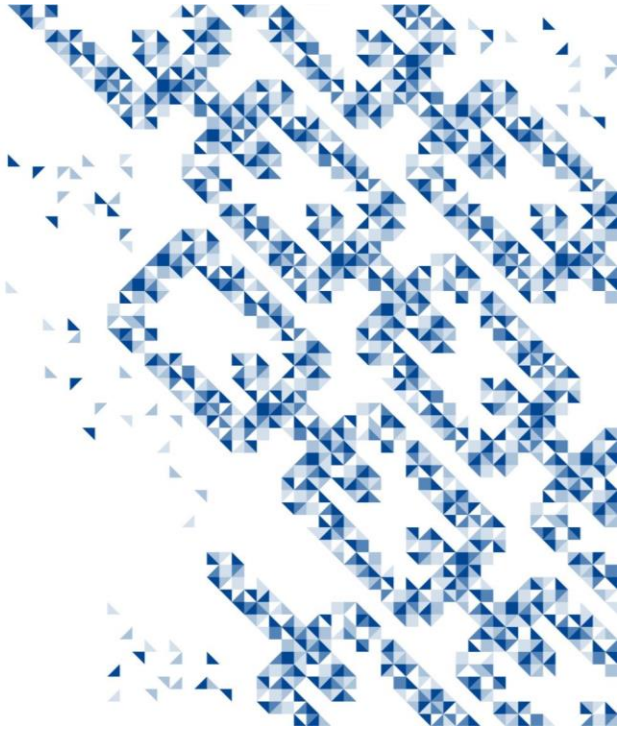
Cardinalidad

- Se puede especificar la cardinalidad de ocurrencia de elementos mediante los siguientes indicadores:
  - **?** cero o una ocurrencia
  - **+** una o más ocurrencias
  - ***** cero o más ocurrencias
- Para indicar un orden en la posición de los subelementos, se debe especificar mediante coma **','** para una secuencia (respeta posición y es obligatorio), **|** para choice (especifica que solo una de las opciones puede aparecer en un momento determinado) , o paréntesis () para especificar agrupamientos de elementos.
- Los atributos pueden ser alguno de los siguientes:
  - **REQUIRED** obligatorio
  - **IMPLIED** opcional
  - **FIXED** valor constante

DTD asociada a un documento XML

```
<?xml version="1.0" ?>
<!DOCTYPE bibliography [
<!ENTITY www "World Wide Web" >
<!ELEMENT author (#PCDATA)>
<!ATTLIST author authorRef CDATA #REQUIRED
age CDATA #REQUIRED>
<!ELEMENT authors (author+)>
<!ELEMENT fullPaper EMPTY>
<!ATTLIST fullPaper source CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT related EMPTY>
<!ATTLIST related papers CDATA #REQUIRED>
<!ELEMENT paper (authors, fullPaper?, title, related*)>
<!ATTLIST paper pubid CDATA #REQUIRED
role CDATA #REQUIRED>
<!ELEMENT bibliography (paper+)>
]>
<bibliography>
<paper pubid="wsa" role="publication">
    <authors>
        <author authorRef="joyce" age="45">J. L. R. Colina      </author>
    </authors>
    <fullPaper source="http://mysite.com/confusion"/>
    <title>Object Confusion in a Deviator System in &www;
    </title>
    <related papers="deviation101 x_deviators"/>
</paper>
</bibliography>
```

XML Schema



- El principal motivo de la definición de XML Schema es satisfacer las carencias de las DTD con respecto a la definición de la estructura de un documento XML.
- Los requerimientos establecidos por la W3C fueron principalmente:
  - Proporcionar un conjunto más rico de datos que las DTD.
  - Proporcionar un sistema de tipo de datos que permita la definición de datos construidos por el usuario, con sus restricciones.
  - Distinguir la representación léxica de la información contenida.

Ventajas de XML Schema

Descripción y restricciones de documentos usando la sintaxis de XML.
Soporte para tipos de datos.
Descripción del modelo de contenido y el reuso de elementos vía la herencia.
Extensibilidad.
Habilidad de escribir esquemas dinámicos.
Capacidades de autodocumentación cuando una hoja de estilo es aplicada.

XML Schema

- La especificación de XML Schema consiste de tres partes:
- *XML Schema Parte 0*: Primer.- Explica que son los esquemas, como difieren de las DTD y como construir un esquema.
- *XML Schema Parte 1: Estructuras*.- Propone métodos para describir y la estructura y contenido de los documentos XML, y define las reglas para gobernar la validación de documentos.
- *XML Schema Parte 2: Tipos de datos*.- Define un conjunto simple de tipos de datos que son asociados con tipos de elementos y atributos XML.

Definiciones de esquemas

- Un esquema es un documento XML.
- Debido a que es un documento autodescriptivo, se definen las etiquetas:

```
<element name="nombre_elemento">
<attribute name="nombre_atributo">
```

para establecer las características del documento XML.

- También se hace distinción de cuando un elemento contiene otros elementos o no, mediante:
  - Tipos complejos.- son elementos que contienen otros elementos, o que tienen atributos.
  - Tipos simples.- son elementos que no tienen hijos ni tampoco atributos. Los atributos también son tipos simples.

Ejemplo

Orderdata.xml

```
<Orderdata>
        <Customer fistName="Bob" lastName="Warren" emailAddress="bob@mymailaddress.com"/>
        <Invoice/>
</OrderData>
```

Orderdata.xsd

```
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
   <xs:element name="OrderData">
        <xs:complexType>
                <xs:sequence>
                    <xs:element name="Customer" >
                      <xs:complexType>
                        <xs:attribute name="firstName" type="xs:string" use="required" />
                        <xs:attribute name="lastName" type="xs:string" use="required" />
                        <xs:attribute name="emailAddress type="xs:string" />
                </xs:complexType>
                    </xs:element>
                    <xs:element name="Invoice" type="xs:string" />
                </xs:sequence>
        </xs:complexType>
   </xs:element>
</xs:schema>
```

- El modelo de contenido es la forma en que se definen las estructuras válidas de los elementos y otros componentes. De aquí, hay dos tipos de etiquetado:
- Definiciones.- para crear nuevos tipos (simples o complejos).
- Declaraciones.- describe el modelo de contenido de elementos y atributos. A su vez, posee dos tipos de declaraciones:
- *Declaraciones simples.- crean tipos simples.*
- *Declaraciones complejas.- crean tipos complejos.*

Tipos de datos y estructuras

Preámbulo

- Un documento XML Schema consiste en un preámbulo, seguido de dos o más declaraciones.
- Está definido por la etiqueta <schema>, la cual tiene los siguientes atributos:

```
<xs:schema version="1.0“
xmlns:xs=“http://www.w3.org/2001/XMLSchema”
elementFormDefault="qualified">
```

- en donde el namespace *xs:* se usa para identificar los tipos de datos y los elementos del esquema.

Declaración de elementos

- La forma de declarar un elemento es usando la etiqueta `<element>`

```
<xs:element name="nomb_elem" />
```

en donde el atributo *name* es obligatorio.

- También es posible establecer restricciones al elemento, como:
  - El tipo primitivo o construido, mediante el atributo *type*.
  - Valor por omisión, con el atributo *default*.
  - Valores constantes con el atributo *fixed*.

Declaraciones de atributos

- La forma de declarar un atributo es usando la etiqueta `<attribute>`

```
<xs:attribute name="nomb_atrib" />
```

en donde el atributo *name* es obligatorio.

- También es posible establecer restricciones al atributo, como:
  - Valor por omisión, con el atributo *default*.
  - El tipo primitivo o construido, mediante el atributo *type*.
  - (opcional) La restricción de existencia, determinada por el atributo *use* (“required”).
  - Un valor constante mediante el atributo *fixed*.

RESTRICCIÓN	DESCRIPCIÓN
enumeration	Define una lista de valores aceptables
fractionDigits	Especifica el máximo número de decimals permitido (>= 0)
length	Especifica el número exacto de caracteres o lista de items permitidos (>= 0)
maxExclusive	Especifica el límite superior para valores númericos
maxInclusive	Especifica el límite superior para valores númericos (igual inclusive)
maxLength	Especifica el máximo número de caracteres o lista de items permitidos (>= 0)
minExclusive	Especifica el límite inferior para valores númericos
minInclusive	Especifica el límite inferior para valores númericos (igual inclusive)
minLength	Especifica el mínimo número de caracteres o lista de items permitidos (>= 0)
pattern	Define la secuencia exacta de caracteres que son aceptables
totalDigits	Especifica el número exacto de digitos permitidos (>= 0)
whiteSpace	Especifica cómo son manejados los espacios en blanco (line feeds, tabs, spaces y carriage returns) value="preserve | replace | collapse"

Restricciones

- Es posible limitar los valores que se establecen en un elemento o atributo mediante una restricción.
- Las restricciones aplicables son las siguientes:

Ejemplos

- El elemento `<password>` puede contener letras mayúsculas, minúsculas y dígitos del 0-9, hasta máximo 8 caracteres.

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- El elemento `<password>` contiene únicamente entre 5 y 8 caracteres.

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Ejemplos

- El elemento <address> no tendrá espacios en blanco sobrantes

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="collapse"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- El elemento <car> solo podrá contener la lista de valores posibles.

```
<xs:element name="car" type="carType"/>
<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```

Definiciones de tipos complejos

- Para definir elementos que contienen más elementos y/o atributos, se emplea la etiqueta `<complexType>`. Usualmente contiene un conjunto de declaraciones de elementos y atributos o de referencias a elementos.
- Posteriormente se puede hacer referencia a el mediante el atributo `type`.
- Si se especifica el atributo `mixed="true"`, entonces el elemento puede contener texto y otros subelementos.
- Se pueden crear "tipos" con nombre, lo que permite que puedan ser reutilizados posteriormente en otras definiciones.

```
<xs:complexType name="CustomerType">
    <xs:attribute name="firstName" type="xs:string" />
    <xs:attribute name="lastName" type="xs:string" />
    <xs:attribute name="emailAddress type="xs:string" />
</xs:complexType>
<xs:element name="Customer" type="CustomerType">
```

Secuencias

- Para especificar el orden en que deben de estar los elementos o atributos, se emplean las etiquetas:
- **<sequence>**, para indicar que los elementos deben de estar en el orden en que fueron declarados,
- **<choice>,** para indicar que solo uno de los elementos declarados debe estar presente,
- **<all>**, para que los elementos puedan aparecer en cualquier orden.

Ejemplo

- El siguiente ejemplo es de un elemento *Customer*, el cual permite que los elementos definido en el aparezcan en cualquier orden, pero sólo una vez .

```
<xs:element name="Customer">
   <xs:complexType name="CustomerType">
       <xs:all>
          <xs:element name="firstName" minOccurs="1" />
          <xs:element name="lastName" minOccurs="1" />
          <xs:element name="emailAddress minOccurs="0" maxOccurs="1"/>
       </xs:all>
   </xs:complexType>
</xs:element>
```

- Una instancia de un documento XML con el esquema anterior sería:

```
<Customer>
        <LastName>Rivers</LastName>
        <FirstName>Joan</FirstName>
</Customer>
```

- Así como también:

```
<Customer>
        <emailAddress>joan@rivers.org</emailAddress>
        <LastName>Rivers</LastName>
        <FirstName>Joan</FirstName>
</Customer>
```

Tipos de datos primitivos

Tipos de datos Primitivos

string	**"Hello World"**
boolean	**{true, false, 1, 0}**
decimal	**7.08**
float	**12.56E3, 12, 12560, 0, -0, INF, -INF, NAN**
double	**12.56E3, 12, 12560, 0, -0, INF, -INF, NAN**
duration	**P1Y2M3DT10H30M12.3S**
dateTime	formato: *CCYY-MM-DDThh:mm:ss*
time	formato: *hh:mm:ss.sss*
date	formato: *CCYY-MM-DD*
gYearMonth	formato: *CCYY-MM*
gYear	formato: *CCYY*
gMonthDay	formato: *--MM-DD*

Nota: 'T' es el separador de fecha/tiempo
INF = infinito
NAN = not-a-number

Tipos de datos primitivos

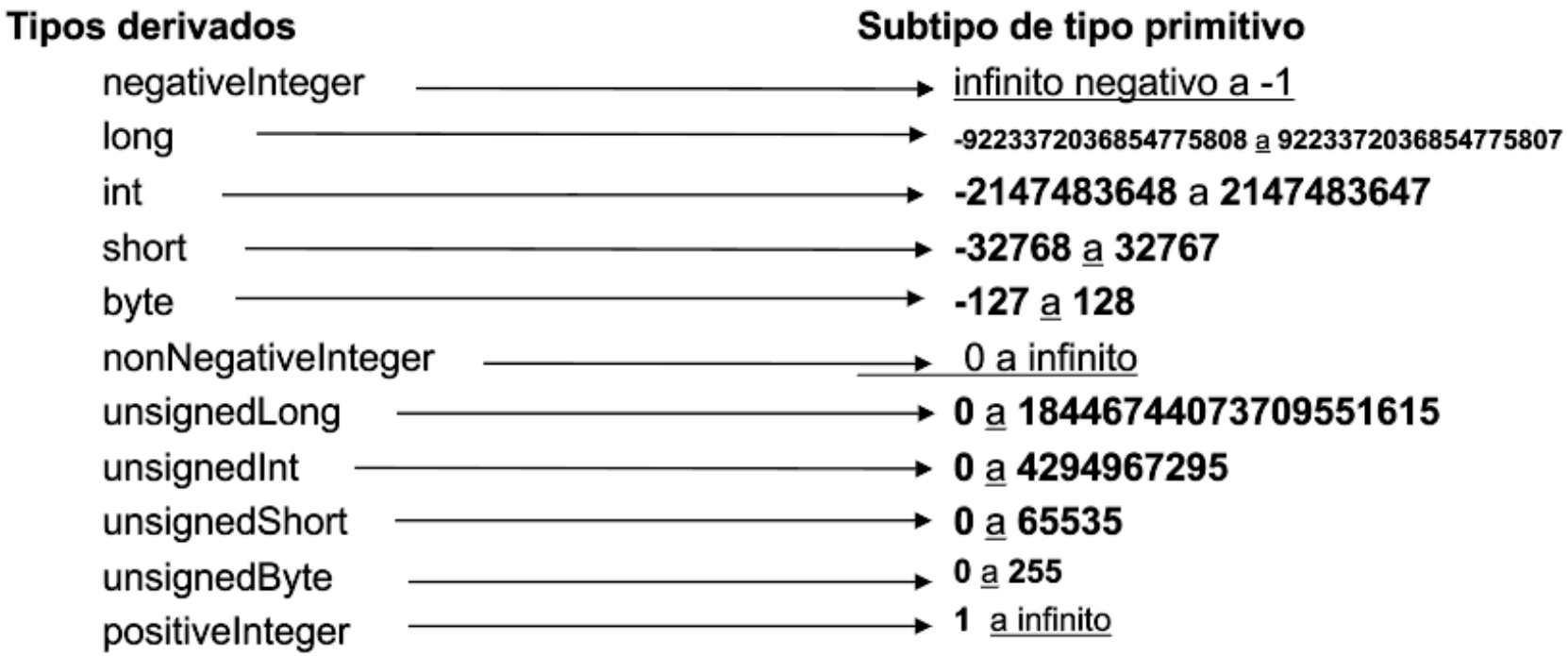
Tipos de datos Primitivos

gDay	formato: ---DD (note los 3 guiones)
gMonth	formato: --MM--
hexBinary	una cadena hexadecimal
base64Binary	una cadena base 64
anyURI	http://www.xfront.com
QName	un nombre de namespace calificado
NOTATION	Una NOTATION de la espec. XML

Tipos de datos derivados

Tipos derivados	Subtipo de tipo primitivo
normalizedString	Cadena sin tabs, LF, o CR
token	Cadena sin tabs, LF, espacios antes o después, o consecutivos
language	Cualquier valor valido xml:lang value, ej., EN, FR, ..
IDREFS	Debe ser usado sólo con atributos
ENTITIES	Debe ser usado sólo con atributos
NMTOKEN	Debe ser usado sólo con atributos
NMTOKENS	Debe ser usado sólo con atributos
Name	
NCName	parte (sin calificador de namespace)
ID	Debe ser usado sólo con atributos
IDREF	Debe ser usado sólo con atributos
ENTITY	Debe ser usado sólo con atributos
integer	456
nonPositiveInteger	infinito negativo a 0

Tipos de datos derivados



Validación con XML Schema

- Asociado a un documento XML:

```
<raiz_XML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="URL | archivoXMLSchema.xsd">
…
```

- Mediante el *DocumentBuilderFactory*:

```
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
factory.setNameSpaceAware(true);
factory.setValidating(true);
try {
factory.setAttribute("http://java.sun.com/xml/jaxp/properties/schemaLanguage", "http://www.w3.org/2001/XMLSchema" );
} catch …
```

- En una aplicación:

```
factory.setAttribute("http://java.sun.com/xml/jaxp/properties/schemaSource", new File(archivo_xsd));
```

DOM

El Document Object Model (DOM) es un modelo que especifica la forma de acceder los datos de un documento XML

Especifica un árbol jerárquico de nodos (elementos, comentarios, entidades, atributos, etc.) construido en memoria.

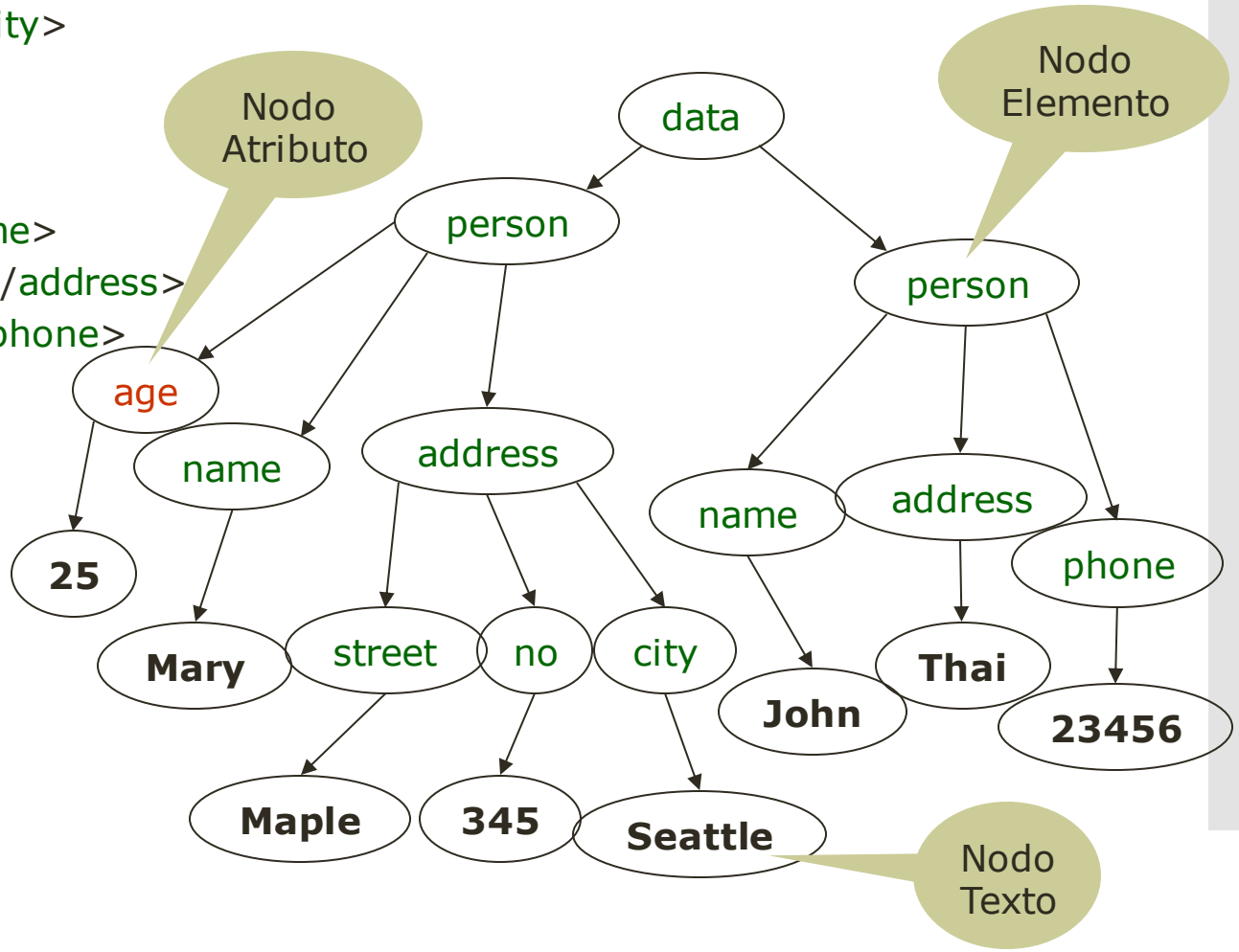
Además especifica un conjunto de funciones para desplazarse por el árbol

- hhtp://www.w3c.org/TR/REC-DOM-Level-1

Incluye soporte para XML 1.0 y HTML, con cada elemento HTML representado como una interfaz.

Semántica XML representada como árbol

```
<data>
  <person age="25" >
    <name> Mary </name>
    <address>
      <street> Maple </street>
      <no> 345 </no>
      <city> Seattle </city>
    </address>
  </person>
  <person>
    <name> John </name>
    <address>Thailand</address>
    <phone> 23456  </phone>
  </person>
</data>
```



SGBDR y XML

Un documento XML podría almacenarse en un SGBDR directamente en un campo LOB, varchar o descompuesto en tablas, pero esto no es eficiente y además es complejo de mantener

Por tanto, se requiere incorporar el tipo de dato nativo XML en los gestores

Esta es la alternativa que se está llevando a cabo por la mayoría de gestores (Oracle, SQL Server, DB2,…)

Niveles de anidamiento en XML

- XML no está restringido a un sólo nivel de anidamiento para representar un registro de una base de datos

```
<person>
    <name>
        <first>John</first>
        <last>Smith</last>
    </name>
    <phone>1234</phone>
</person>
```

Datos semiestructurados

```
<person>
    <name>Mary</name>
    <phone>2345</phone>
    <phone>3456</phone>
</person>
```

name	phone	
Mary	2345	3456

Datos XML

- XML es autodescriptivo
- Los elementos del esquema son parte de los datos, por lo tanto, es más flexible

```
<person>
    <name> John</name>
    <phone>1234</phone>
</person>

<person>
    <name>Joe</name>
</person>
```

name	phone
John	1234
Joe	-

Documentos XML completos

- 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 Carácter Largo (CLOB).
  - 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

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.

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.

Mapeo DB-XML

- Un documento puede estructurarse para adoptar un esquema de una BD

SQL

```
CREATE TABLE Customer
(firstName varchar(50),
lastName varchar(50),
mailingAddress
varchar(50),
mailingCity varchar(60),
mailingState varchar(2),
mailingPostalCode
varchar(10)
)
```

XML

```
<!ELEMENT Customer
(firstName,lastName, mailingAddress,
mailingCity, mailingState,
mailingPostalCode)>
<!ELEMENT firstName (#PCDATA)>
<!ELEMENT lastName(#PCDATA)>
<!ELEMENT mailingAddress(#PCDATA)>
<!ELEMENT mailingCity(#PCDATA)>
<!ELEMENT mailingState(#PCDATA)>
<!ELEMENT mailingPostalCode(#PCDATA)>
```

Mapeo DB-XML

- Y las instancias de los datos:

Resultset # 1 | Messages

	firstName	lastName	mailingAddress	mailingCity	mailingState	mailingPostalCode
1	Kevin	Williams	742 Evergreen Terrace	SpringField	KY	12345

```
<Customer>
<firstName>Kevin</firstName>
<lastName>Williams</lastName>
<mailingAddress>742 Evergreen Terrace</mailingAddress>
<mailingCity>SpringField</mailingCity>
<mailingState>KY</mailingState>
<mailingPostalCode>12345</mailingPostalCode>
</Customer>
```

Distintas representaciones

- Cuando se representan datos en un documento XML, cualquier elemento que sea definido para contener sólo texto usando el tipo #PCDATA tendrá su correspondencia a una columna en una BD relacional.
- Otra forma para representar el documento XML sería la siguiente:

```
<!ELEMENT Customer EMPTY>
<!ATTLIST Customer
firstName CDATA #REQUIRED
lastName CDATA #REQUIRED
mailingAddress CDATA #REQUIRED
mailingCity CDATA #REQUIRED
mailingState CDATA #REQUIRED
mailingPostalCode CDATA #REQUIRED>
```

```
<Customer
firstName="Kevin"
lastName="Williams"
mailingAddress="742 Evergreen
Terrace" mailingCity
="SpringField"
mailingState="KY"
mailingPostalCode="12345 />
```

Comparación

Las dos estrategias de mapeo son:

- Elementos, los cuales son anidados como hijos del elemento que representa el grupo de información.
- Atributos, los cuales son añadidos a los elementos que representan los grupos de información.

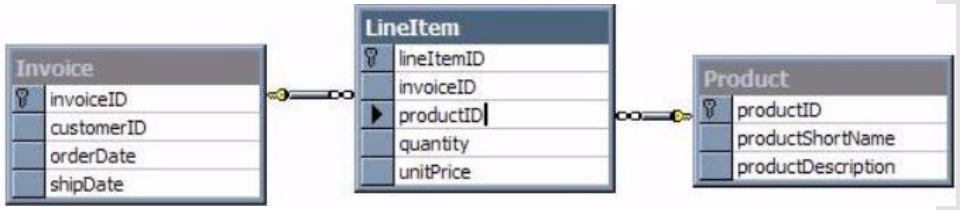
Las diferencias observables con respecto a las anteriores estrategias son:

- Legibilidad
- Compatibilidad con BD
- Fuerte tipado de datos
- Complejidad programática
- Tamaño del documento

Relaciones

- Utilizando los atributos ID y IDREF es posible hacer el equivalente del modelo relacional.

```
CREATE TABLE Invoice (
invoiceID integer PRIMARY KEY,
customerID integer,
orderDate datetime,
shipDate datetime );
CREATE TABLE Product (
productID integer PRIMARY KEY,
productShortName varchar(50),
productDescription varchar (255));
CREATE TABLE LineItem (
lineItemID integer,
invoiceID integer,
productID integer,
quantity integer,
unitPrice float,
CONSTRAINT fk_LineItemInvoice FOREIGN KEY (invoiceID) REFERENCES
Invoice(invoiceID),
CONSTRAINT fk_LineItemProduct FOREIGN KEY (productID) REFERENCES
Product(productID));
```



Evitando las referencias

```
<!ELEMENT OrderData (Invoice+)>
<!ELEMENT Invoice (LineItem+)>
<!ATTLIST Invoice
orderDate CDATA #REQUIRED
shipDate CDATA #REQUIRED>
<!ELEMENT LineItem (Product)>
<!ATTLIST LineItem
quantity CDATA #REQUIRED
unitPrice CDATA #REQUIRED>
<!ELEMENT Product EMPTY>
<!ATTLIST Product
productShortName CDATA
#REQUIRED
productDescription CDATA
#REQUIRED>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE OrderData SYSTEM "OrderData2.dtd">
<OrderData>
<Invoice orderDate="7/23/2000" shipDate="7/30/2000">
<LineItem quantity="17" unitPrice="0.10">
<Product productShortName="Widgets (3 inch)"
productDescription="Rubberized Brown Widgets (3
inch)"/>
</LineItem>
<LineItem quantity="22" unitPrice="0.05">
<Product productShortName="Grommets (0.5 inch)"
productDescription="Vulcanized Orange Grommets (half
inch)"/>
</LineItem>
</Invoice>
<Invoice orderDate="7/23/2000" shipDate="7/30/2000">
<LineItem quantity="30" unitPrice="0.05">
<Product productShortName="Grommets (2 inch)"
productDescription="Vulcanized Orange Grommets (2
inch)"/>
</LineItem>
<LineItem quantity="19" unitPrice="0.15">
<Product productShortName="Sprockets (1 inch)"
productDescription="Anodized Silver Sprocktes (one
inch)"/>
</LineItem>
</Invoice>
</OrderData>
```

¿Porque bases de datos y XML?

Con respecto a XML:

- Es una tecnología que habilita el intercambio eficiente de información entre aplicaciones Web.
- Es simple y extensible.
- Hay un fuerte apoyo de las empresas para su adopción.

Con respecto a las bases de datos:

- Es una tecnología bastante madura, reflejada en los SADB actuales.
- El valor de las empresas está depositado en la información que almacenan y explotan.
- Sigue una constante evolución hacia otras nuevas tecnologías (bodegas de datos, multimedia, orientación a objetos, etc.)

Documento XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT OrderData (Invoice+, Product+)>
<!ELEMENT Invoice (LineItem+)>
<!ATTLIST Invoice
orderDate CDATA #REQUIRED
shipDate CDATA #REQUIRED>
<!ELEMENT LineItem EMPTY>
<!ATTLIST LineItem
productIDREF IDREF #REQUIRED
quantity CDATA #REQUIRED
unitPrice CDATA #REQUIRED>
<!ELEMENT Product EMPTY>
<!ATTLIST Product
productID ID #REQUIRED
productShortName CDATA #REQUIRED
productDescription CDATA #REQUIRED>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE OrderData SYSTEM "OrderData.dtd">
<OrderData>
<Invoice orderDate="7/23/2000" shipDate="7/28/2000">
<LineItem productIDREF="prod1" quantity="17"
unitPrice="0.10"/>
<LineItem productIDREF="prod2" quantity="22"
unitPrice="0.05"/>
</Invoice>
<Invoice orderDate="7/23/2000" shipDate="7/28/2000">
<LineItem productIDREF="prod2" quantity="30"
unitPrice="0.05"/>
<LineItem productIDREF="prod3" quantity="19"
unitPrice="0.15"/>
</Invoice>
<Product productID="prod1" productShortName="Widgets
(3 inch)" productDescription="Rubberized Brown Widgets
(3 inch)"/>
<Product productID="prod2" productShortName="Grommets
(0.5 inch)" productDescription="Vulcanized Orange
Grommets (0.5 inch)"/>
<Product productID="prod3" productShortName="Sprockets
(1 inch)" productDescription="Anodized Silver
Sprocktes (one inch)"/>
</OrderData>
```

Sin embargo…

- Debido al modelo semiestructurado de un documento XML, puede ser difícil hacer búsquedas dentro y entre documentos.
- La administración y actualización de documentos XML puede llegar a ser una tarea compleja.
- Los SABD no tienen un formato estándar para el intercambio de información.
- Para habilitar una base de datos para la Web, son necesarias tecnologías adicionales.
- Las empresas buscan realizar negocios a través de Internet (B2B y B2C)…

Bases de Datos XML nativas

- Define un modelo (lógico) para un documento XML, y recupera y almacena documentos de acuerdo a ese modelo. Como mínimo, el modelo debe de incluir elementos, atributos y PCDATA (texto).
- Tiene como unidad fundamental de almacenamiento a un documento XML.
- No es necesario que tenga algún modelo particular de almacenamiento físico. Esto es, puede ser construida sobre una base de datos relacional, jerárquica u orientada a objetos, o usar un formato propietario de almacenamiento tal como archivos indexados comprimidos.

Características

- 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.

Ventajas

- Buenas para almacenar documentos XML
- Pueden almacenar documentos XML así como formatos de estilos
- Las aplicaciones son débilmente acopladas
- El modelado de datos es simple y flexible
- Complementan a los SABDR con soluciones de mapeo XML
- El rendimiento puede ser muy bueno

Algunos sistemas

Almacenamiento nativo

- eXist http://exist-db.org/exist/apps/homepage/index.html (Open Source)
- MarkLogic https://www.marklogic.com
- OpenLink Virtuoso https://virtuoso.openlinksw.com

Construido sobre una base de datos de objetos

- Ozone http://www.ozone-db.org (Open Source)

Construida sobre una base de datos relacional

- DB2 IBM https://www.ibm.com/docs/en/db2/11.5?topic=purexml-overview
- Oracle 10g https://www.oracle.com/technical-resources/articles/quinlan-xml.html
- PostgreSQL https://www.postgresql.org

Inclusión de XML en SQL

Nueva parte del estándar para crear y manipular documentos XML.

ISO/IEC 9075-14: XML-Related Specifications (SQL/XML).

- Nuevo tipo predefinido.
- Nuevos operadores predefinidos para crear y manipular valores de tipo XML.
- Reglas para mapear tablas, esquemas y catálogos a documentos XML.

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).

SGBDR y XML

Un documento XML podría almacenarse en un SGBDR directamente en un campo LOB, varchar o descompuesto en tablas, pero esto no es eficiente y además es complejo de mantener

Por tanto, se requiere incorporar el tipo de dato nativo XML en los gestores

Esta es la alternativa que usan la mayoría de gestores (Oracle, PostgreSQL, DB2,…)

Tipo de dato XML

Permite almacenar datos XML de forma nativa en la BD

Puede ser optimizado (representación interna diferente a la cadena de caracteres)

Puede almacenar:

- Documentos XML bien formados (sólo un nodo raíz)
- Contenido XML (elementos, secuencia de elementos, texto,…)
- Se basa en XQuery. El valor de un tipo de dato XML es una instancia del modelo de datos XQuery.

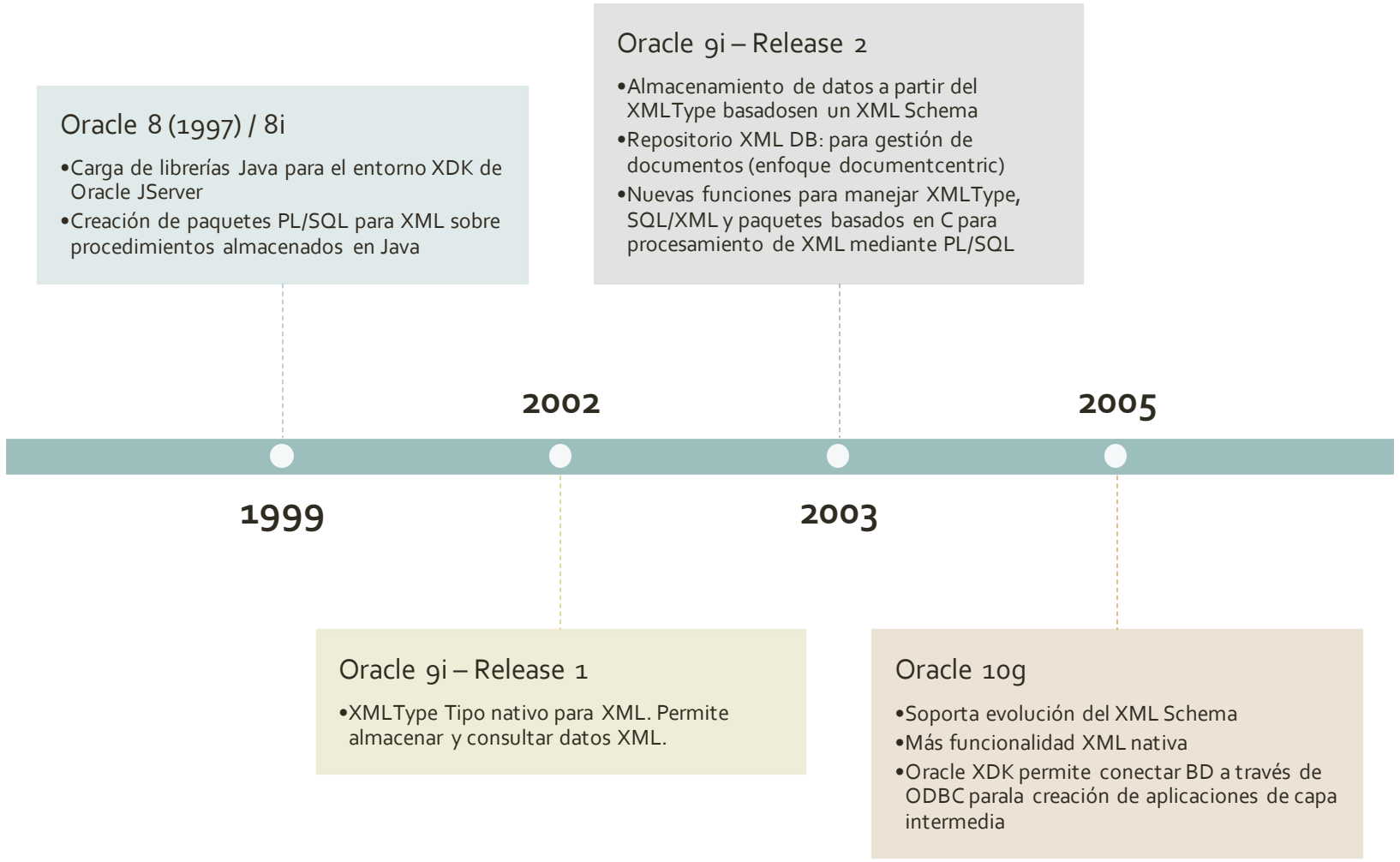
Productos comerciales

- Soporte de XML en SGBD
  - IBM, Oracle implementan casi por completo el SQL/XML
  - Microsoft soporta similares características pero con sintaxis propietaria
  - Todos soportan XQuery dentro del SQL
  - Existen diferencias en su implementación física (almacenamiento)
- Oracle 10g basado en CLOB o tablas OR
- Microsoft 2005 y 2008 almacenado como BLOB en formato interno propietario
- DB2 V9 basado en CLOB

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.
  - En la versión 9i se incluye un tipo de dato (XMLType) para manejo nativo de XML.

Oracle XML DB



Almacenamiento

Dos opciones:

Repositorio de datos (Oracle XML DB Repository):

- Organizado jerárquicamente, consultable
  - Almacenamiento y visualización de contenido XML como un directorio jerárquico de carpetas
- Acceso a los documentos y representación de las relaciones entre documentos con:
  - XPath
  - URLs -> HTTP/FTP
  - SQL y PL/SQL

Tipo de dato nativo (XMLType)

- Permite definir tablas, columnas, parámetros, valores devueltos por funciones o variables en procedimientos PL/SQL
- Dispone de Funciones predefinidas para crear instancias XMLType, validar contenidos XML contra XML Schemas, aplicar hojas XSLT, etc.

Oracle XML DB Architecture: XMLType Storage and Repository

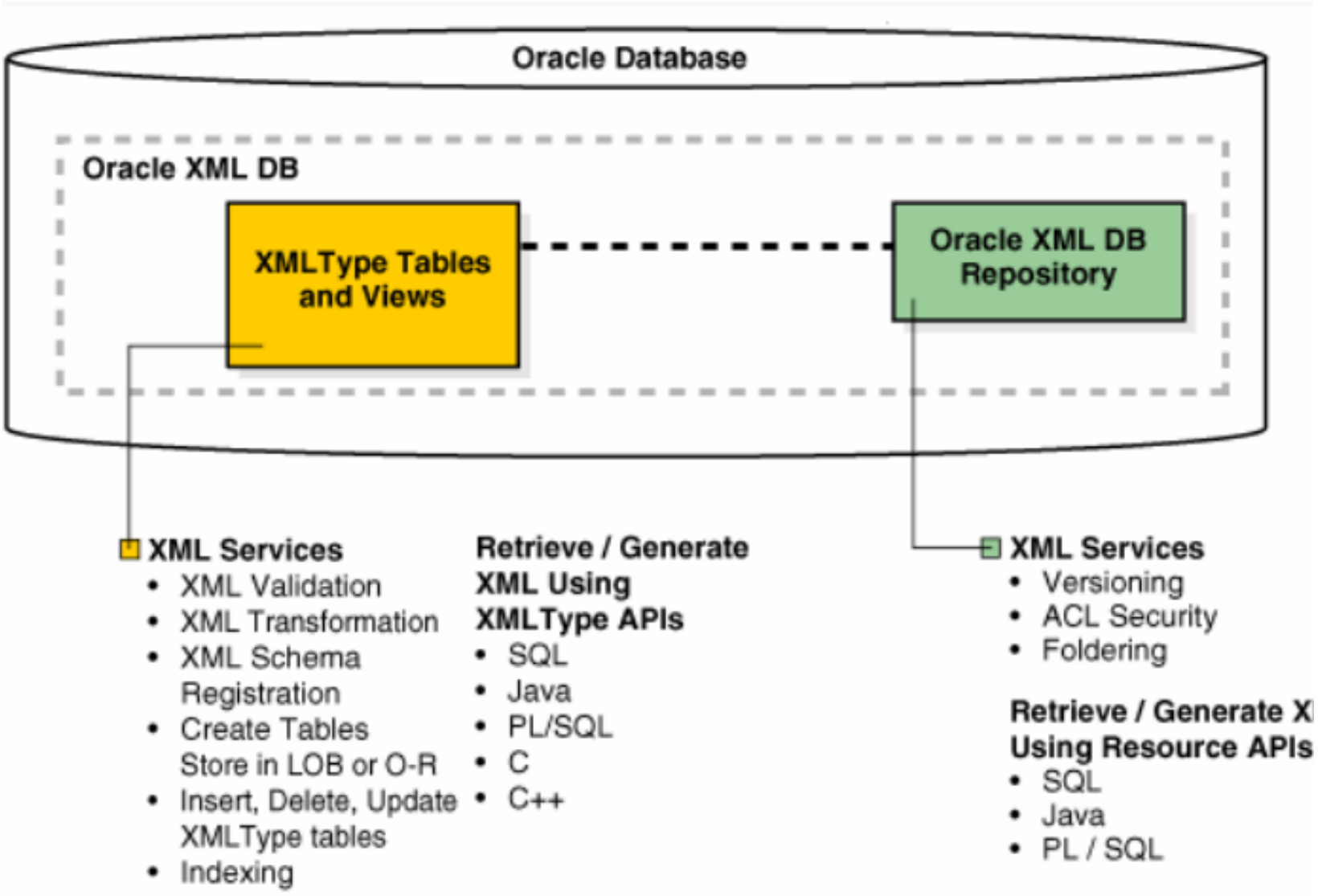


Tabla con una columna XML

```
CREATE TABLE employees(
id CHAR(6),
lastname VARCHAR(30),
….,
resume XMLType);
```

ID	LASTNAME	….	RESUME
123456	Long	…..	<?xml versión="1.0" ? <resume xmlns="http://www.resume.com/resume"> <name>…</name> <address>…</address> </resume>
876453	Nick	…	NULL
637596	Smith	….	<resume ref="http://www.banner.com/resume.html" />

Columna XMLType

```
CREATE TABLE Person ( id int primary key, nombre
varchar2(20) not null, direccion xmlType);

INSERT INTO Person(id,nombre,direccion) VALUES(1,
'Peter Smith', XMLTYPE('<MailAddressTo
id="1"><Person>Peter Smith</Person> <Street>10
Apple Tree Lane</Street><City>New
York</City><State>NY</State<Zipcode>12345</Zipcode>
</MailAddressTo>'));

SELECT extract(direccion, '/MailAddressTo/Street')
FROM Person;

UPDATE Person SET direccion =
XMLTYPE('<MailAddressTo id="1"><Person>Peter
Smith</Person> <Street>10 Downing
Street</Street><City>London</City>
<State>England</State> <Zipcode>22334</Zipcode></Ma
ilAddressTo>');
```

XMLType: ¿Columna o Tabla?

- En Oracle se puede almacenar datos XML en
  - Columnas XMLType

```
CREATE TABLE MiTabla ( id int primary key,…,
xmlCol xmlType)
```

  - Tablas de objetos a partir del tipo XMLType
    - Crear las tablas utilizando sentencias SQL:

```
CREATE TABLE MiTablaXML OF XMLType
```

    - Crear las tablas cuando se registra un Schema XML registrado en el repositorio de Oracle XML DB

```
DBMS_XMLSCHEMA.registerSchema(
SCHEMAURL => 'http://xmlns.oracle.com/xdb/MiSchema.xsd',
SCHEMADOC => bfilename('XMLDIR','MiSchema.xsd'),
CSID => nls_charset_id('AL32UTF8'));
```

SQL/XML: Funciones del estándar suministradas por ORACLE

- Funciones para generar datos XML con datos procedentes de la BD relacional:
  - XMLPARSE
  - XMLSERIALIZE
  - XMLELEMENT
  - XMLATTRIBUTES
  - XMLFOREST
  - XMLCONCAT
  - XMLAGG
  - XMLPI
  - XMLCOMMENT

SQL/XML: Funciones del estándar

- Funciones del tipo de dato XML:
  - **XMLPARSE** - Convierte una cadena de caracteres que contiene datos XML en un valor (instancia) de tipo XML
  - **XMLSERIALIZE** - Obtiene una representación en string o LOB de un dato de tipo XML

```
INSERT INTO employees VALUES('123456', 'Smith', …..,
XMLPARSE(DOCUMENT '<?xml versión="1.0" ? <resume
xmlns="http://www.resume.com/resume">
<name>…</name><address>…</address></resume> ' PRESERVE
WITHESPACE));

SELECT e.id, XMLSERIALIZE(DOCUMENT e.resume AS
VARCHAR(2000)) AS resume

FROM employees e WHERE e.id = '123456';
```

ID	RESUME
123456	<?xml versión="1.0" ? <resume xmlns="http://www.resume.com/resume"> <name>…</name> <address>…</address> </resume>

XMLElement

- Devuelve un valor XML dado por:
  - Un identificador SQL que actúa como su *name*
  - Una lista opcional de declaraciones *namespace*
  - Una lista opcional de nombres y valores de sus atributos
  - Una lista opcional de expresiones que suministran su contenido

```
<XML element> ::= XMLELEMENT ( NAME
<XML element name> [ , <XML
namespace declaration> ] [ , <XML
attributes> ] [ { , <XML element
content> }... ] )
```

Ejemplo

```
select e.employee_id,
XMLELEMENT(NAME
"NombreEmpleado", e.first_name)
as resultadoEnXML
from hr.employees e
```

ID	RESULTADOENXML
100	<NombreEmpleado>Steven</NombreEmpleado>
101	<NombreEmpleado>John</NombreEmpleado>
102	<NombreEmpleado>Paul</NombreEmpleado>

Ejemplo

```
SELECT XMLELEMENT ("Emp",
XMLELEMENT ("NombreEmpleado", e.first_name ||'
'|| e.last_name ),
XMLELEMENT("e-mail", e.email ) ) AS
ResultadoEnXML
FROM hr.employees e;
```

RESULTADOENXML

```
<Emp><NombreEmpleado>Steven King</NombreEmpleado><e-
mail>kings@mail.com</e-mail></Emp>
```

```
<Emp><NombreEmpleado>John Smith</NombreEmpleado><e-
mail>smithj@mail.com</e-mail></Emp>
```

```
<Emp><NombreEmpleado>Paul Singer</NombreEmpleado><e-
mail>singerp@mail.com</e-mail></Emp>
```

Ejemplo

```
SELECT XMLELEMENT (name "Dpto",
XMLELEMENT ("NombreDpto",
d.department_name),
XMLELEMENT("trabajadores",(select count(*)
from employees e where e.department_id =
d.department_id)) AS ResultadoEnXML
FROM hr.departments d;
```

RESULTADOENXML
<Dpto><NombreDpto>Administration</NombreDpto><trabajadores>1</trabajadores></Dpto>
<Dpto><NombreDpto>Marketing</NombreDpto><trabajadores>2</trabajadores></Dpto>
<Dpto><NombreDpto>Human Resources</NombreDpto><trabajadores>6</trabajadores></Dpto>

Ejemplo

```
SELECT XMLELEMENT ("NombreEmpleado",
XMLATTRIBUTES (e.email AS "eMail"),
e.first_name ||' '|| e.last_name ) as Resultado
FROM hr.employees e;
```

RESULTADO

<NombreEmpleado e-mail="kings@mail.com">Steven King</NombreEmpleado>

<NombreEmpleado e-mail="smithj@mail.com">John Smith</NombreEmpleado>

<NombreEmpleado e-mail="singerp@mail.com">Paul Singer</NombreEmpleado>

Ejemplo

```
SELECT XMLELEMENT
("gestion:NombreEmpleado",
XMLNAMESPACES('http://www.gestion.com' as
"gestion"), XMLATTRIBUTES (e.email AS
"gestion:eMail"), e.first_name ||' '||
e.last_name ) as resultado
FROM hr.employees e;
```

RESULTADO

```
<gestion:NombreEmpleado xmlns="http://www.gestion.com" gestion:e-
mail="kings@mail.com">Steven King</gestion:NombreEmpleado>
```

```
<gestion:NombreEmpleado xmlns="http://www.gestion.com" gestion:e-
mail="smithj@mail.com">John Smith</gestion:NombreEmpleado>
```

```
<gestion:NombreEmpleado xmlns="http://www.gestion.com" gestion:e-
mail="singerp@mail.com">Paul Singer</gestion:NombreEmpleado>
```

XMLConcat

- Produce un valor XML dado dos o más expresiones de tipo XML
- Si alguna de las expresiones se evalúa como nulo, es ignorada

```
SELECT XMLCONCAT(
XMLELEMENT("NombreEmpleado", e.first_name),
XMLELEMENT("Apellido", e.last_name )) AS
Resultado
FROM hr.employees e;
```

RESULTADO

```
<NombreEmpleado>Stven</NombreEmpleado><Apellido>King</Apellido>
<NombreEmpleado>John</NombreEmpleado><Apellido>Smith</Apellido>
<NombreEmpleado>Paul</NombreEmpleado><Apellido>Singer</Apellido>
```

XMLForest

- Produce una secuencia de elementos XML de los argumentos que se le pasan.
- XMLFOREST permite realizar consultas de forma más compacta, y además tiene como ventaja con respecto a XMLELEMENT que elimina los nulos. Sin embargo, no permite incluir atributos.

```
SELECT XMLFOREST (e.first_name as "Nombre", e.email as
"Email")

FROM hr.employees e
```

RESULTADO

```
<Nombre>Steven</Nombre><e-mail>kings@mail.com</e-mail>

<Nombre>John</Nombre><e-mail>smithj@mail.com</e-mail>

<Nombre>Paul</Nombre><e-mail>singerp@mail.com</e-mail>
```

XMLPI

- Permite generar instrucciones de procesamiento

```
SELECT XMLPI(NAME "OrderAnalysisComp",
'imported, reconfigured, disassembled')
AS pi FROM DUAL;
```

PI
<?OrderAnalysisComp imported, reconfigured, disassembled?>

XMLComment

- Permite crear un comentario

```
SELECT XMLComment('This is a
comment')
AS cmnt FROM DUAL;
```

```
<!--This is a comment-->
```

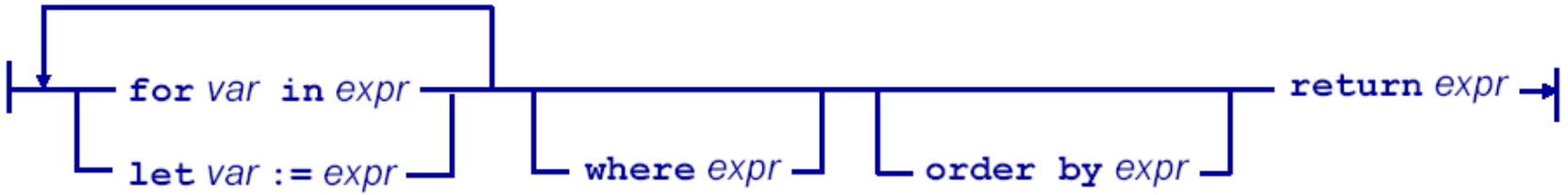
XMLQUERY

- La función XMLQUERY devuelve un valor XML a partir de la evaluación de una expresión XQuery utilizando los argumentos de entrada especificados como variables XQuery.

```
Select id, nombre, XMLQuery(
'for $i in /POSTBOX
where $i/MailAddressTo/Zipcode = "22334"
return <Details>
<Zipcode num="{$i/MailAddressTo/Zipcode}"/>
<CityName char="{$i/MailAddressTo/City}"/>
</Details>'
PASSING direccion RETURNING CONTENT) Personas
FROM person2;
```

Expresiones FLWOR

- Una expresión FLWOR enlaza variables, las aplica a un predicado, y construye un nuevo resultado.



Las cláusulas FOR and LET generan una lista de tuplas de variables de limite, conservando el orden de entrada.

La cláusula WHERE aplica un predicado, eliminando algunas de las tuplas.

La cláusula ORDER BY impone un orden en las tuplas sobrevivientes.

La cláusula RETURN es ejecutada para cada tupla sobreviviente, generando una lista ordenada de salidas.

Especificación de patrones (XPath)

/	Especifica el hijo inmediato
//	Selecciona a cualquier profundidad en el arbol - /descendant-or-self::node() -
.	Selecciona el contexto actual -self::node()-
..	Selecciona el nodo padre –parent::node()-
*	Selecciona todos los elementos del contexto actual
@	Selecciona un atributo –attribute:: -
@*	Selecciona todos los atributos en el contexto actual.
:	Separador de namespaces
!	Aplica el método de información al nodo de referencia
()	Agrupa contenidos para precedencia
[]	Aplica un patrón de filtro

XMLAgg

- Función de agregación similar a SUM, AVG, etc.

```
SELECT e.department_id,
XMLELEMENT("Dpto", XMLAGG (
XMLELEMENT ("NombreEmpleado", e.first_name )
ORDER BY e.first_name))
FROM hr.employees e
group by e.department_id;
```

ID	RESULTADO
100	<Dpto><NombreEmpleado>John</NombreEmpleado></Dpto>
101	<Dpto><NombreEmpleado>Alan</NombreEmpleado><NombreEmpleado>Jennifer</NombreEmpleado><NombreEmpleado>Mitch</NombreEmpleado></Dpto>
102	<Dpto><NombreEmpleado>Michelle</NombreEmpleado><NombreEmpleado>Susan</NombreEmpleado></Dpto>

SQL/XML: Funciones propias de ORACLE

- Funciones para generar datos XML con datos procedentes de la BD relacional:
  - SYS_XMLGEN()
  - XMLSEQUENCE()
  - XMLCOLATTVAL()
  - SYS_XMLAGG()

SYS_XMLGEN()

- Similar a la función XMLElement(), pero en este caso, la función recibe un único argumento y devuelve un documento XML bien formado.

```
SELECT SYS_XMLGen(XMLELEMENT ("Empleado",
(XMLELEMENT("NombreEmpleado", XMLATTRIBUTES
(e.email), e.first_name ||' '|| e.last_name)),
XMLELEMENT("Departamento", e.department_id), XMLE
LEMENT("Telefono", e.phone_number))) as xml FROM
hr.employees e;
```

XML

```
<?xml versión="1.0?><ROW><Empleado><NombreEmpleado email="Sking@mail.com">Steven
King</NombreEmpleado>
<Departamento>90</Departamento><Telefono>515.123.4567</Telefono></Empleado></ROW>
<?xml versión="1.0?><ROW><Empleado><NombreEmpleado email="Nkochar@mail.com">Neena
Kochar</NombreEmpleado>
<Departamento>60</Departamento><Telefono>515.123.4568</Telefono></Empleado></ROW>
<?xml versión="1.0?><ROW><Empleado><NombreEmpleado email="Ldehan@mail.com">Lex de
Han</NombreEmpleado>
<Departamento>60</Departamento><Telefono>515.123.4569</Telefono></Empleado></ROW>
<?xml versión="1.0?><ROW><Empleado><NombreEmpleado email="Psmith@mail.com">Paul
Smith</NombreEmpleado>
<Departamento>90</Departamento><Telefono>515.123.4570</Telefono></Empleado></ROW>
```

XMLSEQUENCE()

- Realiza la función inversa de SYS_XMLGen. Devuelve un varray de instancias de XMLType. Al devolver una colección, se debe utilizar en el FROM de la consulta

```
SELECT * FROM table
(XMLSequence(extract(XMLType('<A><B>V1</B
><B>V2</B><B>V3</B></A>'),'/A/B'))) as
tabla;
```

COLUMN_VALUE
<B>V1</B>
<B>V2</B>
<B>V3</B>

XMLCOLATTVAL()

- Crea un fragmento XML, con etiqueta COLUMN y un atributo NAME, que lo iguala al nombre de la columna. Podemos cambiar el valor del atributo mediante el alias

```
SELECT XMLCOLATTVAL(e.first_name as
nombre) FROM hr.employees e;
```

XMLCOLATTVAL(E.FIRST_NAME AS NOMBRE)

```
<column name="NOMBRE">Ellen</column>
<column name="NOMBRE">Sundar</column>
<column name="NOMBRE">Mozhe</column>
<column name="NOMBRE">David</column>
<column name="NOMBRE">Shelli</column>
<column name="NOMBRE">Amit</column>
```

SYS_XMLAGG()

- Agrega todas las instancias XML que se le pasan como parámetro y devuelve un documento XML

```
SELECT SYS_XMLAGG(XMLELEMENT
("NombreEmpleado", e.first_name
||''|| e.last_name)) as xml FROM
hr.employees e;
```

XML

```
<?xml versión="1.0"?><ROWSET><NombreEmpleado>Ellen
Abel</NombreEmpleado> <NombreEmpleado>Sundar
Ande</NombreEmpleado><NombreEmpleado>Mozhe
Atkinson</NombreEmpleado><NombreEmpleado>Paul
Smith</NombreEmpleado></ROWSET>
```

SQL/XML: Funciones de manipulación de ORACLE

- Funciones para manipulación de datos XML. Utilizan XPath para localizar ítems en una instancia XML.
  - EXTRACT()
  - EXTRACTVALUE()
  - EXISTSNODE()
  - XMLSEQUENCE()
  - XMLQUERY() (en estándar)
  - XMLTABLE() (en estándar)
  - UPDATEXML()
  - DELETEXML()

EXTRACT()

- Selecciona un nodo individual y sus nodos hoja de una instancia XML.

```
select
extract(direccion,'/POSTBOX/MailAddressTo'
).getstringval() as xml from person2;
```

XML

```
<MailAddressTo id="2">
<Person>Peter Smith</Person>
<Street>10 Downing Street</Street>
<City>London</City>
<State>England</State>
<Zipcode>22334</Zipcode>
</MailAddressTo>
```

EXTRACTVALUE()

- Extrae el valor de un nodo hoja. Devuelve un valor, no una instancia XML

```
select extractValue(direccion,
'/POSTBOX/MailAddressTo/Person') as
Person
from person2;
```

PERSON

Peter
Smith

EXISTSNODE()

- Busca valores específicos en el nodo hoja, si existe devuelve true.

```
Select count(*) from person2
where existsNode(direccion,
'/POSTBOX/MailAddressTo[Person="Peter
Smith"]') = 1;
```

COUNT(*)
1

XMLQuery()

```
Select id, nombre, XMLQuery( 'for $i in /POSTBOX
where $i/MailAddressTo/Zipcode = "22334" return
<Details><Zipcode
num="{$i/MailAddressTo/Zipcode}"/><CityName
char="{$i/MailAddressTo/City}"/><City>{if
($i/MailAddressTo/City = "New York") then
"Correct City" else "Incorrect
City"}</City><State>{if ($i/MailAddressTo/State =
"NY") then "Correct State" else "Incorrect
State"}</State></Details>'PASSING direccion
RETURNING CONTENT) Personas FROM person;
```

ID	NOMBRE	PERSONAS
1	John Smith	<Details><ZipCode num="22334"></ZipCode><CityName char="London"></CityName> <City>Incorrect City</City><State>Incorrect State</State></Details>

UpdateXML()

Función que permite la actualización parcial de un documento almacenado como un valor XMLType.

Permite realizar múltiples cambios en una sola operación.

Cada cambio consiste en una expresión XPath que identifica el nodo a ser actualizado y el nuevo valor para ese nodo.

Ejemplo

```
UPDATE person SET direccion =
updateXML(direccion, '/POSTBOX/MailAddressTo/Person/
text()', 'Stephen G. King',
'/POSTBOX/MailAddressTo/@id', '2')
WHERE existsNode(direccion, '/POSTBOX') = 1;

UPDATE person SET direccion =
updateXML(direccion,'/POSTBOX/MailAddressTo',XMLType
('<MailAddressTo id="3"><Person>Peter
Smith</Person><Street>10 Downing
Street</Street><City>London</City><State>England</St
ate><Zipcode>22334</Zipcode></MailAddressTo>'))
WHERE existsNode(direccion, '/POSTBOX') = 1;
```

DeleteXML()

- Borra un nodo de cualquier clase

```
UPDATE person SET direccion
= deleteXML(direccion,'/POSTBOX/Mail
AddressTo/City') WHERE existsNode(di
reccion, '/POSTBOX/MailAddressTo[@id
="3"]') = 1;
```