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 insertar 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

- Las etiquetas XML son sensitivas a mayúsculas y minúsculas

Las etiquetas están normalmente en pares de inicio/fin

- <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 los atributos deben estar encerrados entre comillas dobles
- Un elemento puede tener varios atributos, pero su nombre solo puede ocurrir 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:

<	<
>	>
&	&
'	‘
"	“
&	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

Es un documento que 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
  - Qué elementos pueden ocurrir
  - Qué atributos pueden o deben estar en un elemento
  - Qué 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 bibliografia [
<!ENTITY www "World Wide Web" >
<!ELEMENT autor (#PCDATA)>
<!ATTLIST autor autorRef CDATA #REQUIRED edad CDATA #REQUIRED>
<!ELEMENT autores (autor+)>
<!ELEMENT contenido EMPTY>
<!ATTLIST contenido enlace CDATA #REQUIRED>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT relacionado EMPTY>
<!ATTLIST relacionado articulos CDATA #REQUIRED>
<!ELEMENT articulo (autores, contenido?, titulo, relacionado*)>
<!ATTLIST articulo pubid CDATA #REQUIRED rol CDATA #REQUIRED>
<!ELEMENT bibliografia (articulo+)>
]>
<bibliografia>
<articulo pubid="wsa" rol="publication">
    <autores>
        <autor autorRef="joyce" edad="45">J. R. Collins </autor>
    </autores>
    <contenido enlace="http://mysite.com/confusion"/>
    <titulo>Object Confusion in a Deviator System in &www; </titulo>
    <relacionado articulos="Systems, Analysis"/>
</articulo>
</bibliografia>
```

Tipos ID, IDREF y IDREFS

- Es posible especificar que un atributo sea un atributo de tipo ID, para que los atributos especificados como IDREF o IDREFS dentro del documento pueden usarse para hacer referencia a los atributos de tipo ID, lo que habilita los vínculos entre documentos.
  - Los atributos de tipo ID, IDREF e IDREFS corresponden a relaciones PK/FK (clave principal/clave externa) de una base de datos relacional, con algunas diferencias.
- Para que los atributos ID, IDREFS e IDREF sean válidos:
  - El valor de ID debe ser único dentro del documento XML.
  - Para cada atributo IDREF e IDREFS, los valores de ID a los que se haga referencia deben estar en el documento XML.
  - El valor de un atributo ID, IDREF e IDREFS debe ser un token con nombre. (Por ejemplo, el valor entero 101 no puede ser un valor ID)

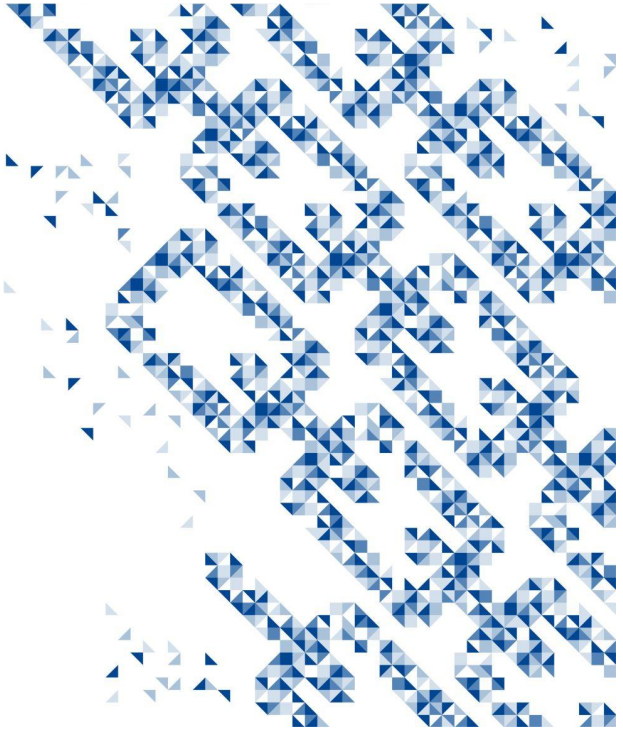
Validación XML mediante una DTD

- La validación de una DTD asociada a un documento XML debe realizarse por un paser validador.
- En caso de que el contenido de la DTD esté en un archivo externo (no contenido en el documento XML), se debe incluir el tipo SYSTEM y el nombre del archivo DTD, que deberá estar en la misma ruta del archivo XML para que sea localizado.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE raiz SYSTEM "archivo.dtd">
<raiz> …
```

- También es posible indicar otro directorio de residencia del archivo utilizando una ruta completa o una URL, cuidando de que el archivo esté disponible y sea localizable.
- `<!DOCTYPE raiz SYSTEM "http://misitio.com./dtds/archivo.dtd">`

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 qué son los esquemas, como difieren de las DTD y cómo 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

```
<Ordenes>
    <Cliente nombre="Juan" apellido="López" correoe="jlopez@mail.com"/>
    <Pedido/>
</Ordenes>
```

Orderdata.xsd

```
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema " elementFormDefault="qualified">
   <xs:element name="Ordenes">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Cliente" >
              <xs:complexType>
                <xs:attribute name="nombre" type="xs:string" use="required" />
                <xs:attribute name="apellido" type="xs:string" use="required" />
                <xs:attribute name="correoe" type="xs:string" />
                 </xs:complexType>
            </xs:element>
            <xs:element name="Pedido" 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="nombre_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="nombre_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 decimales permitido (>= 0)
length	Especifica el número exacto de caracteres o lista de ítems permitidos (>= 0)
maxExclusive	Especifica el límite superior para valores numéricos
maxInclusive	Especifica el límite superior para valores numéricos (igual inclusive)
maxLength	Especifica el máximo número de caracteres o lista de ítems permitidos (>= 0)
minExclusive	Especifica el límite inferior para valores numéricos
minInclusive	Especifica el límite inferior para valores numéricos (igual inclusive)
minLength	Especifica el mínimo número de caracteres o lista de ítems permitidos (>= 0)
pattern	Define la secuencia exacta de caracteres que son aceptables
totalDigits	Especifica el número exacto de dígitos 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 `<direccion>` no tendrá espacios en blanco sobrantes

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

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

```
<xs:element name="coche" type="cocheT"/>
<xs:simpleType name="cocheT">
  <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 él 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="ClienteT">
    <xs:attribute name="nombre" type="xs:string" />
    <xs:attribute name="apellido" type="xs:string" />
    <xs:attribute name="correoe" type="xs:string" />
</xs:complexType>
<xs:element name="Customer" type="ClienteT">
```

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 *Cliente*, el cual permite que los elementos definidos en él aparezcan en cualquier orden, pero sólo una vez .

```
<xs:element name="Cliente" >
   <xs:complexType>
       <xs:all>
          <xs:element name="nombre" minOccurs="1" />
          <xs:element name="apellido" minOccurs="1" />
          <xs:element name="correoe" minOccurs="0"
maxOccurs="1" />
       </xs:all>
   </xs:complexType>
</xs:element>
```

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

```
<Cliente>
    <nombre>Juan</nombre>
    <apellido>López</apellido>
</Cliente>
```

- Así como también:

```
<Cliente>
    <correoe>jlopez@mail.com</correoe>
    <apellido>López</apellido>
    <nombre>Juan</nombre>
</Cliente>
```

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*

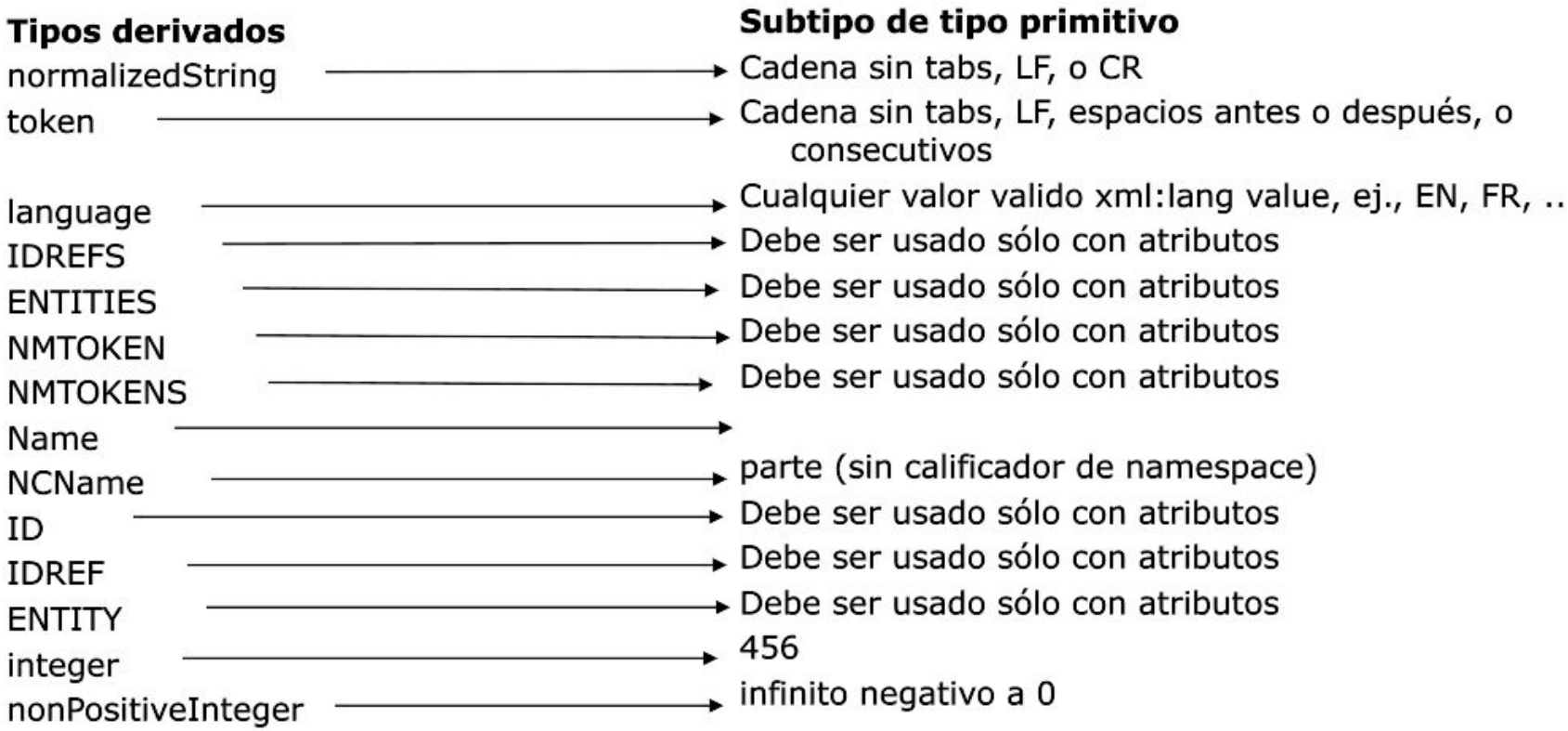
Nota: 'T' es el separador de fecha/tiempo
INF = infinito
NAN = not-a-number

Tipos de datos primitivos

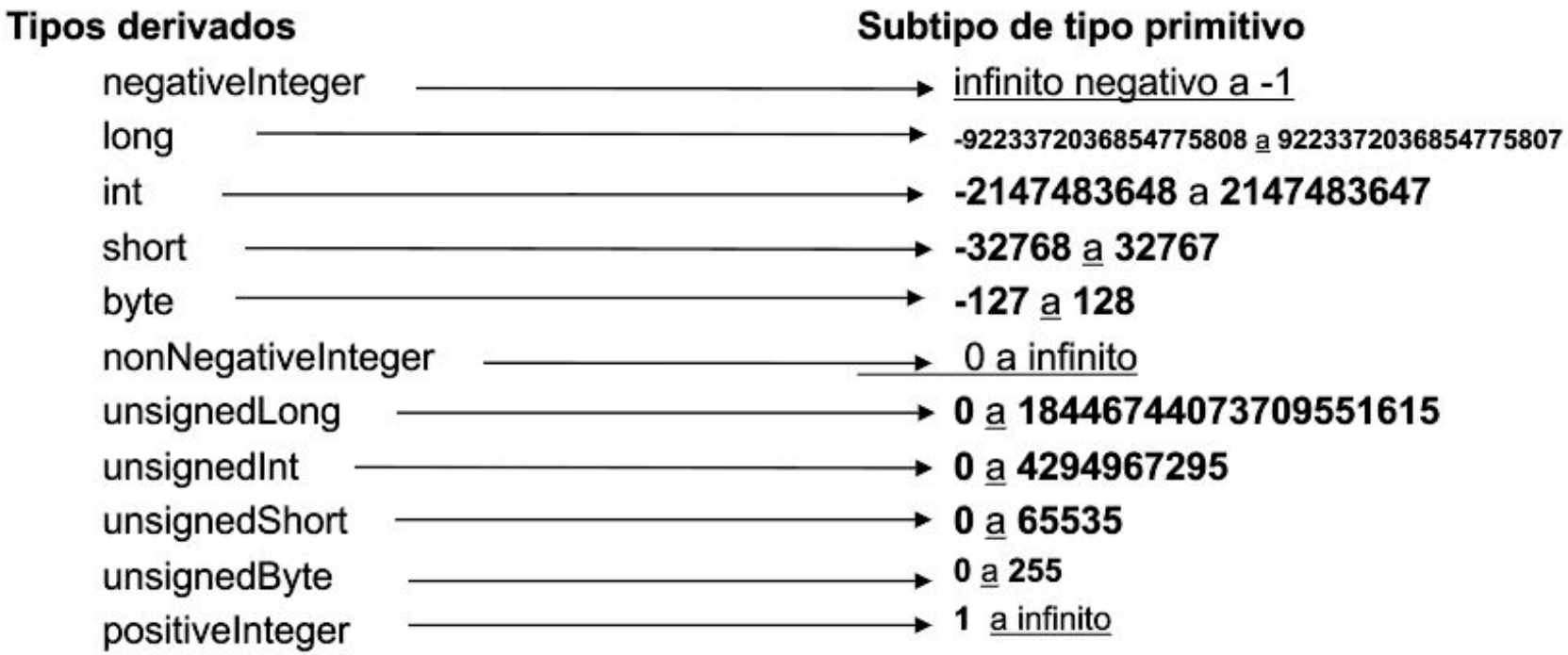
Tipos de datos Primitivos

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 de datos derivados



Validación con XML Schema

- Debe ser definido en el elemento raíz a través de un Namespace/atributo :

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

- El namespace es utilizado para documentos que serán validados mediante XMLSchema; cabe mencionar que al ser procesado el documento no se visita la página de www.w3.org, sino que esta declaración es resuelta a una definición interna en el parser.
- El parámetro *xsi:noNamespaceSchemaLocation*, indica que el XMLSchema utilizado para validar el documento es el archivo llamado **archivoXMLSchema.xsd** el cual se encuentra en el mismo directorio del documento en cuestión, o en su caso, una URL en donde se encuentre ubicado el archivo XSD.
- De acuerdo a las reglas de namespaces, se permite que todo elemento anidado dentro de esta declaración pertenezca al "default namespace" por lo que no se requiere de ningún prefijo en los elementos del documento XML.

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.

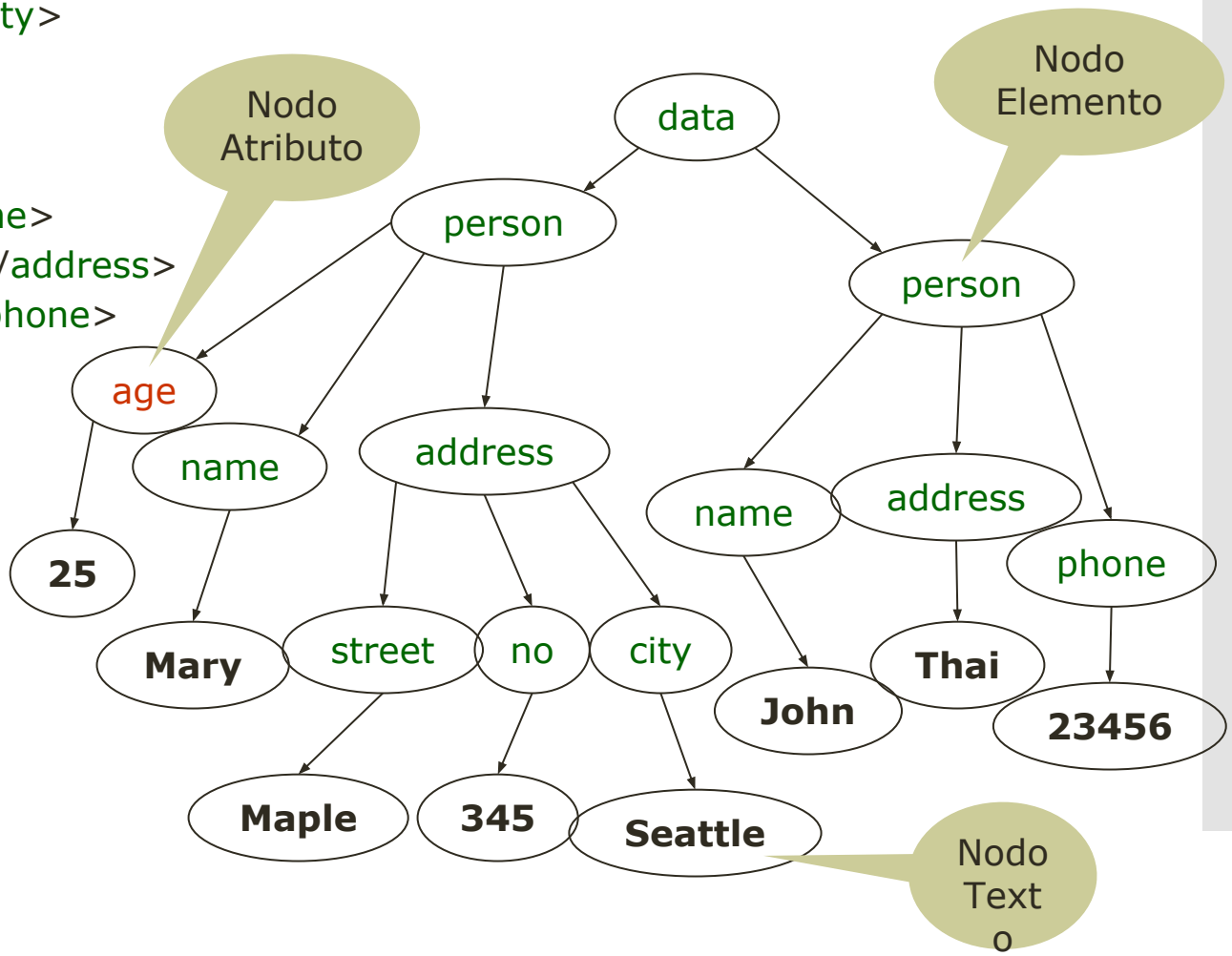
Además especifica un conjunto de funciones para desplazarse por el árbol

- 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

```
<persona>
    <nombre>
        <propio>Juan</propio>
        <apellido>López</apellido>
    </nombre>
    <tel>1234</tel>
</persona>
```

Datos semiestructurados

```
<persona>
    <nombre>María</nombre>
    <tel>3456</tel>
</persona>
```

nombre	tel	
María	2345	3456

Datos XML

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

```
<persona>
    <nombre>Juan</nombre>
    <tel>1234</tel>
</persona>

<persona>
    <nombre>María</nombre>
</person>
```

nombre	tel
Juan	1234
María	-

Mapeo DB-XML

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

SQL

```
CREATE TABLE Cliente
(nombre varchar(50),
apellido varchar(50),
direccion varchar(50),
ciudad varchar(60),
estado varchar(2),
CP varchar(10)
)
```

XML

```
<!ELEMENT Cliente
(nombre,apellido,direccion,ciudad,estado,CP)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellido(#PCDATA)>
<!ELEMENT direccion(#PCDATA)>
<!ELEMENT ciudad(#PCDATA)>
<!ELEMENT estado(#PCDATA)>
<!ELEMENT CP(#PCDATA)>
```

Mapeo DB-XML

- Y las instancias de los datos:

nombre	apellido	direccion	ciudad	estado	CP
Juan	López	Av. Granjas #243	Izcalli	EDOMEX	78675

```
<Cliente>
<nombre>Juan</nombre>
<apellido>López</apellido>
<direccion>Av. Granjas #243</direccion>
<ciudad>Izcalli</ciudad>
<estado>EDOMEX</estado>
<CP>78675</CP>
</Cliente>
```

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 Cliente EMPTY>
<!ATTLIST Cliente
nombre CDATA #REQUIRED
apellido CDATA #REQUIRED
direccion CDATA #REQUIRED
ciudad CDATA #REQUIRED
estado CDATA #REQUIRED
CP CDATA #REQUIRED>
```

```
<Cliente nombre="Juan"
apellido="López"
direccion="Av. Granjas #243"
ciudad="Izcalli" estado="EDOMEX"
CP="78675" />
```

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.

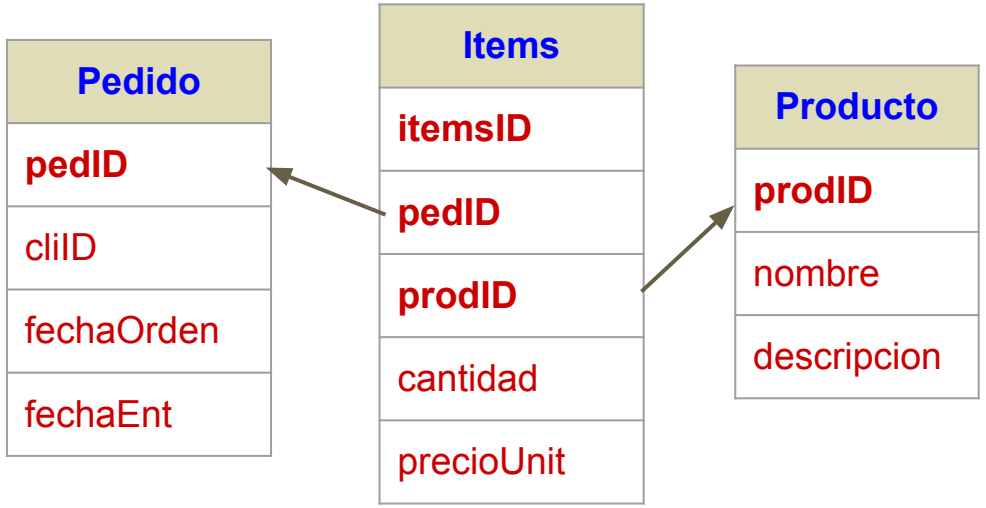
Las diferencias observables con respecto a las anteriores estrategias son:

- 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 Pedido (
pedID NUMBER PRIMARY KEY,
cliID NUMBER
fechaOrden DATE,
fechaEntr DATE );
CREATE TABLE Producto (
prodID NUMBER PRIMARY KEY,
nombre VARCHAR(50),
descripcion VARCHAR(255));
CREATE TABLE Items (
itemsID NUMBER,
pedID NUMBER,
prodID NUMBER,
cantidad NUMBER,
precioUnit NUMBER(8,2),
CONSTRAINT pk_item_ped_prod PRIMARY KEY
(itemsID,pedID,prodID),
CONSTRAINT fk_Items_Ped FOREIGN KEY (pedID) REFERENCES
Pedido(pedID),
CONSTRAINT fk_Items_Prod FOREIGN KEY (prodID) REFERENCES
Producto(prodID));
```



Ejemplo de documento XML (usando ID - IDREF)

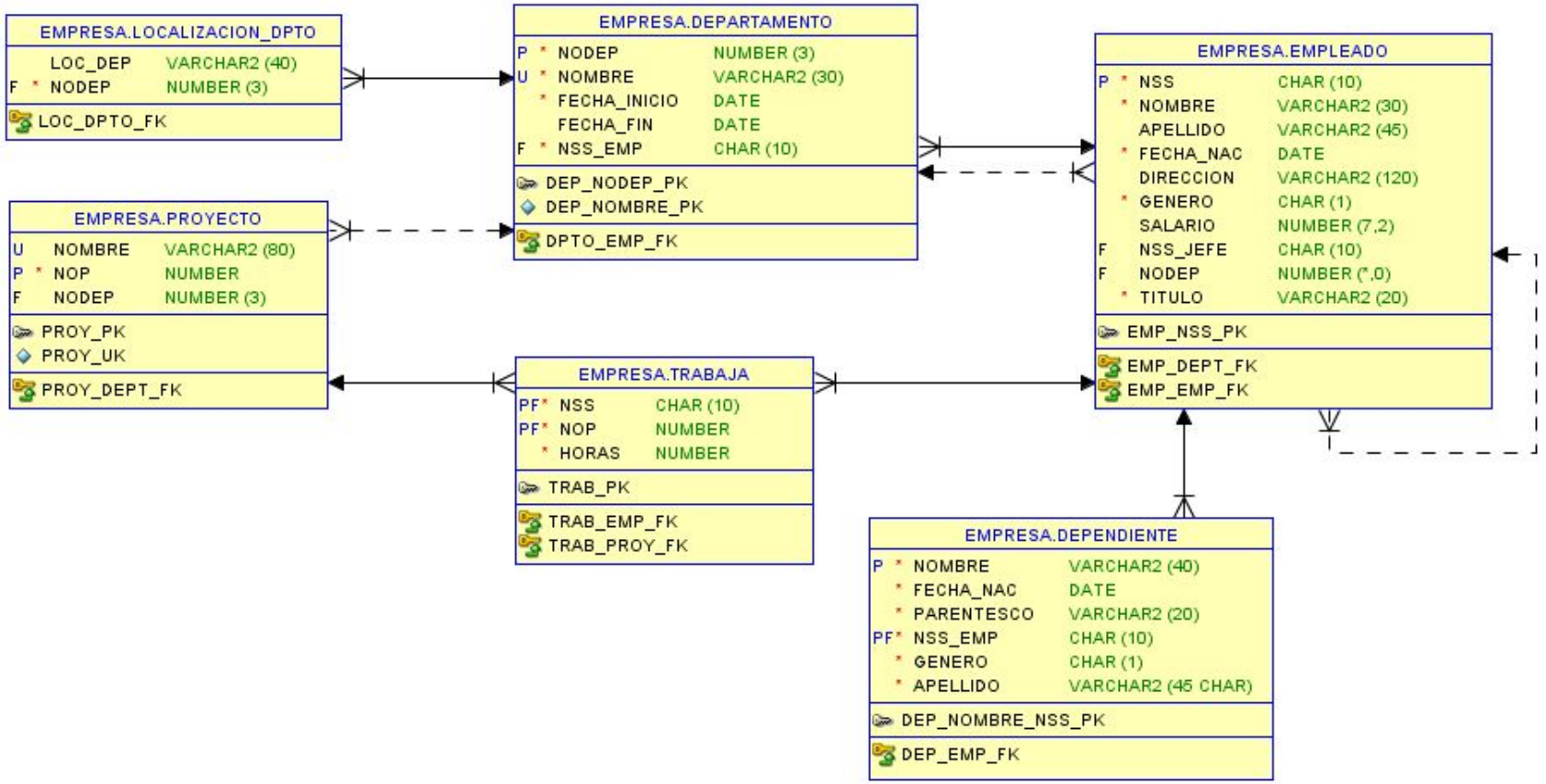
```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Ordenes (Pedido+, Producto+)>
<!ELEMENT Pedido (Items+)>
<!ATTLIST Pedido fechaOrden CDATA #REQUIRED fechaEntr CDATA #REQUIRED>
<!ELEMENT Items EMPTY>
<!ATTLIST Items productoREF IDREF #REQUIRED cantidad CDATA #REQUIRED precioUnit CDATA
#REQUIRED>
<!ELEMENT Producto EMPTY>
<!ATTLIST Producto prodID ID #REQUIRED nombre CDATA #REQUIRED descripcion CDATA #REQUIRED>

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Ordenes SYSTEM "Ordenes.dtd">
<Ordenes>
   <Pedido fechaOrden="7/23/2000" fechaEntr="7/28/2000">
      <Items productoREF="prod1" cantidad="17" precioUnit="0.10"/>
      <Items productoREF="prod2" cantidad="22" precioUnit="0.05"/>
   </Pedido>
   <Pedido fechaOrden="7/23/2000" fechaEntr="7/28/2000">
      <Items productoREF="prod2" cantidad="30" precioUnit="0.05"/>
      <Items productoREF="prod3" cantidad="19" precioUnit="0.15"/>
   </Pedido>
   <Producto prodID="prod1" nombre="tornillo" descripcion="acero inoxidable (3
pulg.)"/>
   <Producto prodID="prod2" nombre="pijas" descripcion="galvanizadas (media
pulg.)"/>
   <Producto prodID="prod3" nombre="tuercas" descripcion="acero inoxidable (¼
pulg.)"/>
</Ordenes>
```

Evitando las referencias

```
<!ELEMENT Ordenes(Pedido+)>
<!ELEMENT Pedido (Items+)>
<!ATTLIST Pedido fechaOrden CDATA #REQUIRED fechaEntr CDATA #REQUIRED>
<!ELEMENT Items (Producto)>
<!ATTLIST Items cantidad CDATA #REQUIRED precioUnit CDATA #REQUIRED>
<!ELEMENT Producto EMPTY>
<!ATTLIST Producto nombre CDATA #REQUIRED descripcion CDATA #REQUIRED>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Ordenes SYSTEM "Ordenes.dtd">
<Ordenes>
   <Pedido fechaOrden="7/23/2000" fechaEntr="7/30/2000">
      <Items cantidad="17" precioUnit="0.10">
         <Producto nombre="tornillo" decripcion="acero inoxidable (3 pulg.)"/>
      </Items>
      <Items cantidad="22" precioUnit="0.05">
         <Producto nombre="pijas" descripcion="galvanizadas (media pulg.)"/>
      </Items>
   </Pedido>
   <Pedido fechaOrden="7/23/2000" fechaEntr="7/30/2000">
      <Items cantidad="30" precioUnit="0.05">
         <Producto nombre="pijas" descripcion="galvanizadas (media pulg.)"/>
      </Items>
      <Items cantidad="19" precioUnit="0.15">
         <Producto nombre="tuercas" descripcion="acero inoxidable (¼ pulg.)"/>
      </Items>
   </Pedido>
</Ordenes>
```

Ejemplo de transformación



¿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.)

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 (W_3C).
  - 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-db (Open Source)
- MarkLogic
- OpenLink Virtuoso

Construido sobre una base de datos de objetos

- ozone (Open Source)

Construida sobre una base de datos relacional

- pureXML overview -- Db2 as an XML database
- Getting into SQL/XML with Oracle Database 10g Release 2
- PostgreSQL

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.

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.

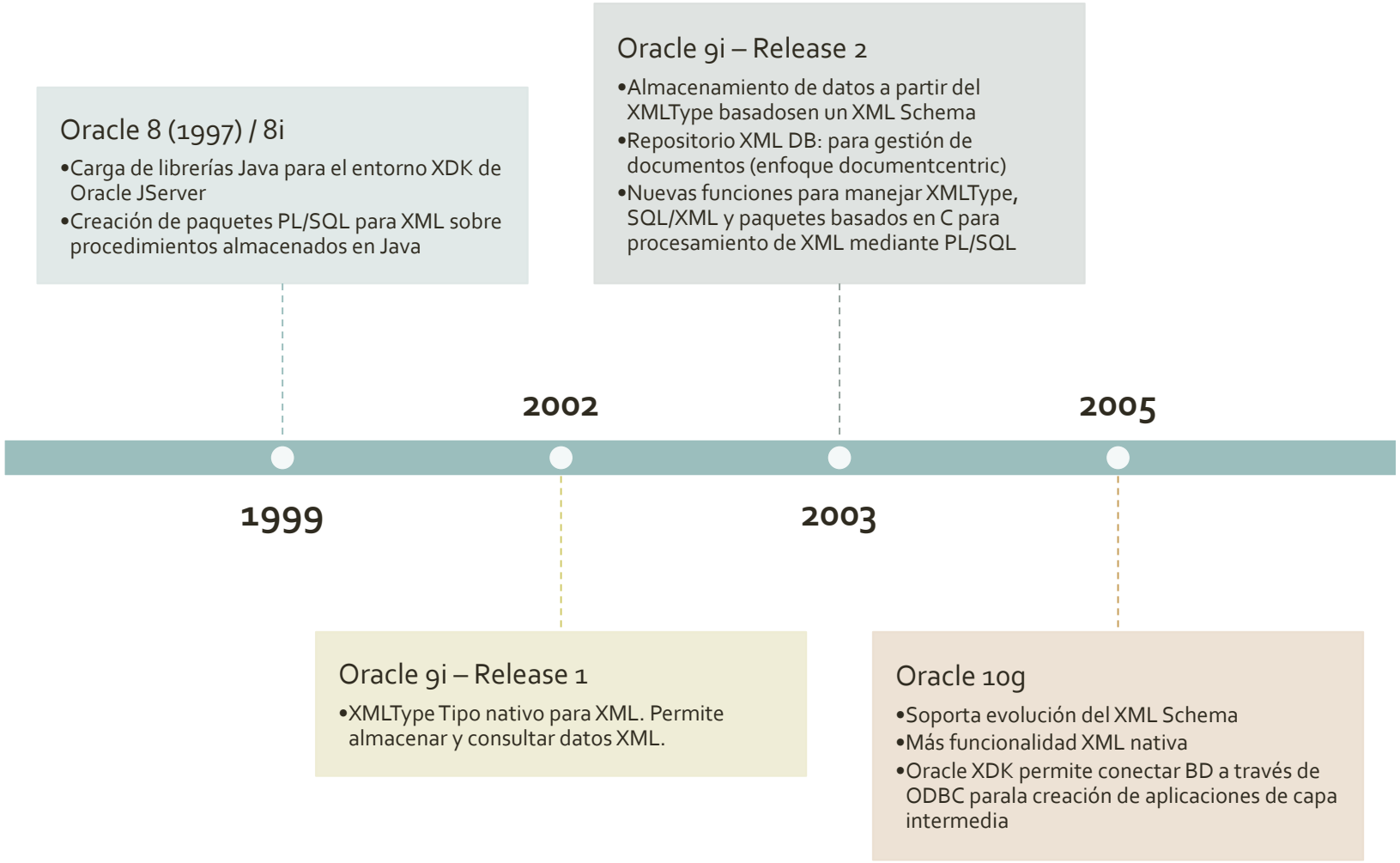
Productos comerciales

- 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

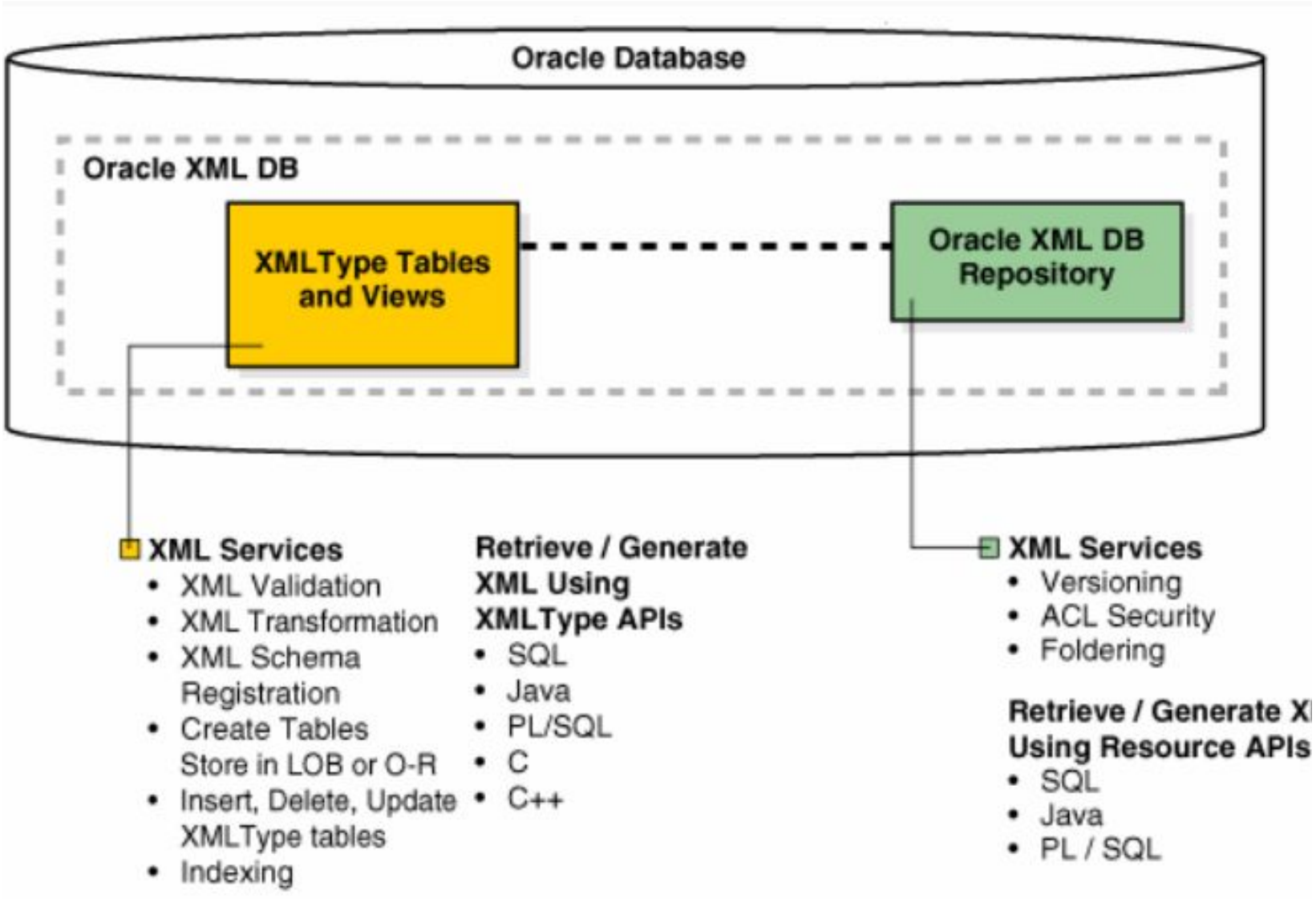


Tabla con una columna XML

```
CREATE TABLE empleados(
id CHAR(6),
nombre VARCHAR(30),
….,
cv XMLType);
```

ID	LASTNAME	….	CV
123456	Pedro	…..	<?xml versión="1.0" ? <resume xmlns="http://www.cv.com/cv"> <nombre>…</nombre> <direccion>…</direccion> </resume>
876453	María	…	NULL
637596	Laura	….	<cv ref="http://www.banner.com/resume.html" />

Columna XMLType

```
CREATE TABLE personas (id NUMBER PRIMARY KEY,
nombre VARCHAR(20), direccion XMLTYPE);

INSERT INTO personas(id,nombre,domicilio) VALUES(1,
'Juan López', XMLTYPE('<direccion><calle>Justo
Sierra#78</calle><Ciudad>Saltillo</Ciudad><Estado>C
oahuila</Estado><CP>12345</CP></direccion>'));

SELECT extract(domicilio, '/direccion/calle') FROM
Personas;

UPDATE personas SET domicilio = XMLTYPE('<direccion
><calle>Benavente#254</calle><Ciudad>Pachuca</Ciuda
d><Estado>Hidalgo</Estado><CP>22334</CP></direccion
>') WHERE id = 1;
```

XMLType: ¿Columna o Tabla?

- En Oracle, se pueden almacenar datos XML en
  - Columnas **XMLType**

```
CREATE TABLE MiTabla (
id NUMBER PRIMARY KEY,
…,
xmlCol XMLTYPE);
```

  - Tablas de objetos a partir del tipo **XMLType**

```
CREATE TABLE MiTablaXML OF XMLTYPE;
```

Validación mediante XML Schema

- Para realizar la validación de un documento XML en Oracle Database Server, se debe primero registrar el documento XML Schema correspondiente. Para ello, es necesario crear un directorio en el SO y colocar allí los archivos XML y XMLSchema, y posteriormente "conectar" el servidor con esta carpeta (como usuario DBA), con el comando:

```
CREATE DIRECTORY DOCUMENTOS_XML AS '/ruta_archivosXML';
GRANT READ, WRITE ON DIRECTORY "DOCUMENTOS_XML" TO user;
```

- en donde **DOCUMENTOS_XML** es el nombre asignado por el usuario al directorio, y *ruta_archivosXML* es la ruta del SO en donde se localiza la carpeta de archivos. Ahora es posible ejecutar el procedimiento siguiente:

```
DBMS_XMLSCHEMA.registerSchema(
SCHEMAURL => 'URL',
SCHEMADOC => BFILENAME('DOCUMENTOS_XML','test.xsd'));
```

  - en donde *URL* es una URL utilizada como identificador, y **test.xsd** es el archivo XMLSchema a registrar. Para revisar si se registró el esquema, se ejecuta:

```
SELECT SCHEMA_URL, LOCAL, XMLSERIALIZE(DOCUMENT SCHEMA AS VARCHAR(4000)) FROM USER_XML_SCHEMAS;
```

  - Es posible usar la dirección *http://localhost:8080/sys/schemas* para ver los esquemas registrados dentro de la base de datos.

Creación de tablas XML

- Una vez que se tienen registrados los esquemas, se pueden crear tablas la cuales sean del tipo *XMLType* y realizar la validación de acuerdo a algún esquema previamente registrado. Suponga la creación de la tabla *empleados*:

```
CREATE TABLE empleados OF XMLType XMLSCHEMA "URL" ELEMENT "raiz_xml";
```

- en donde *URL* se refiere al identificador del esquema registrado previamente, y *raiz_xml* es el nombre del elemento raíz del documento XML a validar.
- Si la tabla es relacional e incluye una columna de tipo XML, entonces se puede usar la siguiente construcción:

```
CREATE TABLE empleados (id NUMBER, documento_xml XMLType) XMLTYPE COLUMN
documento_xml ELEMENT "URL#raiz_xml";
```

- en donde el carácter **#** separa la *url* del elemento raíz del documento XML.

Creación de tablas XML

- Para verificar la creación de la tabla de tipo XML se puede consultar el diccionario de datos:

```
SELECT TABLE_NAME, XMLSCHEMA, SCHEMA_OWNER, ELEMENT_NAME FROM USER_XML_TABLES;
```

- Se cuentan con una serie de funciones informativas de los esquemas:
  - `isSchemaBased(), isSchemaValid(), isSchemaValidated()` regresa 1 en caso de ser verdaderos, o en otro caso.
  - `getSchemaURL()` obtiene la url del esquema asociado
  - `schemaValidate()` realiza una validación de un documento XML
- Para verificar que un documento XML que se desea insertar sea válido con su XML Schema asociado:
- `INSERT INTO tabla_xml VALUES(XMLType(BFILENAME('DOCUMENTOS_XML', 'doc.xml'), nls_charset_id('AL32UTF8')));`
- en donde *DOCUMENTOS_XML* es el directorio en donde residen los archivos XML, *doc.xml* es el nombre del archivo a insertar y *AL32UTF8* es el código de conjunto de caracteres local.

Empleo de las funciones de validación

- Es posible hacer una validación XML Schema pre y post inserción del contenido XML en la base de datos.
- Como ejemplo de una prevalidación, se tiene la inclusión en un disparador, de la siguiente forma:

```
CREATE OR REPLACE TRIGGER validate_orders
   BEFORE INSERT ON orders FOR EACH ROW
BEGIN
  IF (:new.OBJECT_VALUE IS NOT NULL) THEN :new.OBJECT_VALUE.schemavalidate(); END IF;
END;
/
```

- También mediante una cláusula CHECK asociada a la tabla:

```
ALTER TABLE orders ADD CONSTRAINT chk_validate_ordersCHECK (XMLIsValid(OBJECT_VALUE) = 1);
```

- La postvalidación verifica que el documento XML sea válido una vez que ha sido insertado en la base de datos:

```
SELECT o.xmlcol.isSchemaValid('http://www.example.com/schemas/ipo.xsd', 'Orders')FROM orders o;
```

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()
  - XMLQUERY()
  - UPDATEXML()
  - DELETEXML()

EXTRACT()

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

```
SELECT
EXTRACT(domicilio,'/direccion').getstringval
() AS salida FROM persona;
```

SALIDA

```
<direccion><calle>Justo Sierra
#78</calle><Ciudad>Saltillo</Ciudad><Estado>
Coahuila</Estado><CP>12345</CP></direccion>
```

EXTRACTVALUE()

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

```
SELECT extractValue(domicilio,
'/direccion/calle') AS calle
FROM persona;
```

CALLE
Justo Sierra #78

EXISTSNODE()

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

```
SELECT count(*) FROM persona
WHERE existsNode(domicilio,
'/direccion[CP=12345]') = 1;
```

COUNT(*)
1

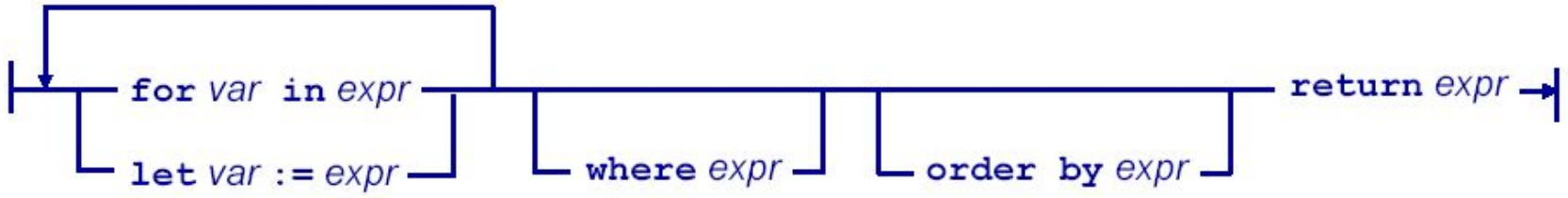
XMLQUERY

- 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 /direccion
where $i/CP = "12345"
return <detalles>
<CP codigo="{$i/CP}"/>
<ciudad nombre="{$i/ciudad}"/>
</detalles>'
PASSING domicilio RETURNING CONTENT) domicilio
FROM personas;
```

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

Operadores Lógicos

and	&&	AND lógico
or	||	OR lógico
not		NO lógico
eq	=	Igual
ieq		Igual no sensitivo a mayúsculas
ne	!=	No igual
ine		No igual no sensitivo a mayúsculas
lt	<	Menor que
ilt		Menor que no sensitivo a mayúsculas
le	<=	Menor o igual que
ile		Menor o igual que no sensitivo a mayúsculas
gt	>	Mayor que
igt		Mayor que no sensitivo a mayúsculas
ge	>=	Mayor o igual que
ige		Mayor o igual que no sensitivo a mayúsculas
all		Operación de conjunto que regresa verdadero si la condición es verdadera para todos los items de la condición
any		Operación de conjunto que regresa verdadero si la condición es verdadera para cualquier item de la condición

Funciones XPath

boolean(arg1)	Convierte el argumento a un valor booleano
concat(arg1, arg2, ...)	Concatena los argumento como una cadena de caracteres
contains(arg1, arg2)	Prueba cuando arg1 contiene a arg2 como subcadena. Es case-sensitive
count(arg1)	Regresa el número de nodos que existen en el conjunto de nodos dado por arg1
document(arg1)	Busca un documento XML externo resolviendo la URL de referencia dada en arg1 y regresa su nodo raíz.
false()	Comprueba el valor falso
id(arg1)	Regresa el conjunto de nodos conteniendo los nodos del documento que posee el id pasado como arg1
key(arg1, arg2)	Es usado para encontrar el nodo con el valor dado por la llave en arg1, definido en un elemento <xsl:key>. Arg2 es el valor que se están buscando para dicha llave
local-name([arg1])	Regresa la parte local del nombre de un nodo
name([arg1])	Regresa el nombre calificado de un nodo, incluyendo el namespace
not(arg1)	Niega el valor de arg1
number(arg1)	Convierte su argumento en un valor numérico. Si se omite arg1, toma el valor del nodo actual.
position()	Regresa el valor de la posición del nodo contexto.
string([arg1])	Convierte arg1 a su valor de cadena de caracteres
string-length(arg1)	Regresa el número de caracteres de arg1
substring(arg1, arg2, [arg3])	Regresa parte de arg1, partir de la posición dada por arg2, y opcionalmente de longitud dada por arg3
sum(arg1)	Calcula el valor total de la suma de los valores de los nodos dados por el conjunto de nodos arg1
true()	Regresa el valor booleano verdadero (true)

Ejemplos

```
<alumnos>
   <alumno NoBoleta="12345">
      <nombre>Juan</nombre>
      <nombre>Carlos</nombre>
      <apellidos>
         <paterno>Lopez</paterno>
         <materno>Perez</materno>
      </apellidos>
      <genero>M</genero>
      <edad>21</edad>
   </alumno>
</alumnos>
```

```
for $x in /alumnos/alumno
return $x

<alumno NoBoleta="12345">
   <nombre>Juan</nombre>
   <nombre>Carlos</nombre>
   <apellidos>
      <paterno>Lopez</paterno>
      <materno>Perez</materno>
   </apellidos>
   <genero>M</genero>
   <edad>21</edad>
</alumno>
```

```
for $x in //nombre
return $x
<nombre>Juan</nombre>
<nombre>Carlos</nombre>
```

```
for $x in //nombre
return <salida>{$x}</salida>
<salida>
   <nombre>Juan</nombre>
</salida>
<salida>
   <nombre>Carlos</nombre>
</salida>
```

```
<salida>
{
for $x in //nombre
return $x
}
</salida>
<salida>
   <nombre>Juan</nombre>
   <nombre>Carlos</nombre>
</salida>
```

```
<salida>
{
for $x in //nombre
return data($x)
}
</salida>
<salida>Juan Carlos</salida>
```

```
for $x in /alumnos
let $y := $x/alumno
return $y
<alumno NoBoleta="12345">
   <nombre>Juan</nombre>
   <nombre>Carlos</nombre>
   <apellidos>
      <paterno>Lopez</paterno>
      <materno>Perez</materno>
   </apellidos>
   <genero>M</genero>
   <edad>21</edad>
</alumno>
```

```
for $x in /alumnos
let $y := $x/alumno
return data($y)
JuanCarlosLopezPerezM21
```

```
for $x in /alumnos/alumno[@NoBoleta =
12345]
return $x

<alumno NoBoleta="12345">
   <nombre>Juan</nombre>
   <nombre>Carlos</nombre>
   <apellidos>
      <paterno>Lopez</paterno>
      <materno>Perez</materno>
   </apellidos>
   <genero>M</genero>
   <edad>21</edad>
</alumno>
```

```
for $x in /alumnos/alumno
where $x/@NoBoleta <= 12345
return data($x)
```

```
JuanCarlosLopezPerezM21
```

```
for $x in /alumnos/alumno
where $x/edad >= 18 and $x/edad <= 60
return data($x)
```

```
JuanCarlosLopezPerezM21
```

```
for $x in /alumnos/alumno
where $x/@NoBoleta <= 12345
order by $x/apellidos/paterno,
$x/apellidos/materno
return data($x)
```

```
JuanCarlosLopezPerezM21
```

```
for $x in /alumnos
where $x/alumno/nombre[1]
return $x/alumno/nombre/text()
```

```
Juan
Carlos
```

```
for $x in /alumnos
where $x/alumno/nombre[1]
return
$x/alumno/nombre/upper-case(text())
```

```
JUAN
CARLOS
```

```
for $x in /alumnos/alumno/nombre
return if(starts-with($x, 'J'))
then $x/lower-case(text())
else $x/upper-case(text())
```

```
juan
CARLOS
```

XMLQuery()

```
SELECT id, nombre, XMLQuery( 'for $i in /direccion
where $i/CP = "12345" return <detalles><CP
codigo="{$i/CP}"/><ciudad
nombre="{$i/ciudad}"/><ciudad>{if ($i/ciudad =
"Saltillo") then "correcto" else
"incorrecto"}</ciudad><estado>{if ($i/estado =
"Coahuila") then "correcto" else
"incorrecto"}</estado></detalles>' PASSING
domicilio RETURNING CONTENT) domicilio FROM
personas;
```

ID	NOMBRE	PERSONA
1	John Smith	<detalles><CP codigo="12345"></CP><ciudad nombre="Saltillo"></ciudad><ciudad>correcto</ciudad><estado>correcto</estado></Detalles>

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.

Ejemplos

```
UPDATE persona SET domicilio =
updateXML(domicilio, '/direccion/ciudad/text()',
'Nogales', '/direccion/estado/text()', 'Sonora')
WHERE existsNode(domicilio, '/direccion[CP=12345]')
= 1;

UPDATE person SET domicilio =
updateXML(domicilio,'/direccion',XMLType('<direccion
><calle>Av. Juarez #432</calle><ciudad>Nogales
</ciudad><estado>Sonora</estado><CP>23456</CP>
</direccion>')) WHERE existsNode(domicilio,
'/direccion[CP=12345]') = 1;
```

DeleteXML()

- Borra un nodo de cualquier clase

```
UPDATE person SET domicilio
= deleteXML(domicilio,'/direccion/ciuda
d') WHERE existsNode(domicilio, '/direc
cion[CP="12345"]') = 1;
```

SQL/XML: Funciones del estándar suministradas por ORACLE

- Funciones para generar datos XML con datos procedentes de una base de datos 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 empleados VALUES('123456', 'López', …., XMLPARSE(DOCUMENT '<?xml versión="1.0"?><cv xmlns="http://www.cv.com/cv"><nombre>…</nombre><direccion>…</direccion></cv> ' PRESERVE WITHESPACE));
```

```
SELECT e.id, XMLSERIALIZE(DOCUMENT e.cv AS VARCHAR(2000)) AS cv FROM empleados e WHERE e.id = '123456';
```

ID	RESUME
123456	<?xml versión="1.0" ? <cv xmlns="http://www.cv.com/cv"> <nombre>…</nombre> <direccion>…</direccion> </cv>

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

```
XMLELEMENT ( NAME etiqueta [,
namespace] [,atributos] [{,
contenido}...])
```

Ejemplo

```
SELECT e.employee_id,
XMLELEMENT(NAME "NombreEmpleado",
e.first_name) AS resultadoEnXML
FROM HR.employees e
```

	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 eMail="kings@mail.com">Steven King</NombreEmpleado>
<NombreEmpleado eMail="smithj@mail.com">John Smith</NombreEmpleado>
<NombreEmpleado eMail="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>Steven</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
"e-mail")

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('Texto del comentario')
AS cmnt FROM DUAL;
```

```
<!--Texto del comentario -->
```

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 dual
(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>
```

Almacenamiento de datos con estructura compleja

- Muchas aplicaciones necesitan almacenar datos estructurados, pero no se modelan fácilmente como relaciones.
- Prefieren almacenar información en formato XML.
- Se han desarrollado normas basadas en XML para la representación de datos en XML de una gran variedad de aplicaciones especializadas, por ejemplo, ChemML (industria química), MathML (notación matemática), GML (sistemas geográficos), BPEL (notación de flujos de procesos), entre muchas más.