1.DTD

1.1 Declaración del DTD.

<!DOCTYPE elementoPadre [elementos hijo y atributos]>

1.2.Vinculación desde el archivo XML

Vinculación Interna

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Casas_Rurales [
<!ELEMENT Casas_Rurales (Casa)*>
<!ELEMENT Casa (Dirección, Descripción, Estado, Tamanio)>
<!ELEMENT Dirección (#PCDATA) >
<!ELEMENT Descripción (#PCDATA) >
<!ELEMENT Estado (#PCDATA) >
<!ELEMENT Tamanio (#PCDATA) ]>
<Casas_Rurales>
  <Casa>....</Casa>
</Casas_Rurales>
```

Vinculación Externa

Privado

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE concesionarios SYSTEM "concesionario.dtd">
<concesionarios>
...
</concesionarios>
```

Público

```
<!DOCTYPE concesionarios PUBLIC "IDENT" "concesionario.dtd">
<concesionarios>
...
</concesionarios>
```

1.3.Elementos

	XML	DTD
Contienen ya los datos <!ELEMENT elemento (#PCDATA) > **Es un elemento vacío** <!ELEMENT elemento EMPTY>	<Estado>Aceptable</Estado> 	<!ELEMENT Estado (#PCDATA) > <!ELEMENT br EMPTY>
Contienen elementos hijos <!ELEMENT elemento_padre (hijo1c, hijo2c,(hijo4c|hijo5c))> c (cardinalidad) puede valer: ?:uno o ninguna vez *: de cero a N veces +: de 1 a N veces nada (por defecto): 1 vez	**Secuencia** <email> <de>Juan</de> <para>Ana</para> <para>Eva</para> <asunto>Reunión</asunto> </email>	<!ELEMENT email (de,para+,cc*,bc*,asunto?, cuerpo?)>
	Opcionalidad <aviso> <titulo>Ventas</titulo> <grafico>foto.jpg<grafico> </aviso>	<!ELEMENT aviso (titulo, (parrafo | grafico))> <aviso> <titulo>Ventas</titulo> <parrafo>bla, bla...<parrafo > </aviso>

1.3.Atributos

es una cuatriada. (Aconsejable escribir una cuatriada por cada atributo)

<!ATTLIST	nombre_elemento	nombre_atributo	tipo_atributo	caracter>
	nombre del elemento al que se asigna el atributo	*nombre del atributo*	*CDATA:* cadena alfanumérica (cualquier cosa). *(valor1|valor2):* puede valer valor 1 o valor2 *ID:clave, no se repite* *IDREF:clave ajena de otro elemento*	*"valor": valor por defecto (entre "")* *#IMPLIED: opcional* *#REQUIRED , obligatorio* *#FIXED: obligatorio y fijo*

```
<perro fecha_nacimiento="2020-01-01">Pastor Alemán</perro>
<!ELEMENT perro (#PCDATA)>
<!ATTLIST perro fecha_nacimiento CDATA "23-02-1988">

<persona sexo="hombre">
<nombre>Juan</nombre>
<apellidos>López Pérez</apellidos>
</persona>
<!ELEMENT persona (nombre,apellidos)>
<!ATTLIST persona sexo (hombre|mujer) #REQUIRED
```

2.XSD

2.1Declaración del documento .XSD (se guarda con la extensión .xsd)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">  ……. </xs:schema>
```

2.2Vinculación desde un archivo XML

En el **elemento raíz** del **xml** se define espacio de nombres con un prefijo (ej. **xsi**) y con el atributo **noNamespaceSchemaLocation** (debe llevar el prefijo) se indica la ruta del archivo con el .xsd

```
<vehiculos xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="XSDvehiculos.xsd">
```

2.3.Tipos de datos simples *(se asignan al atributo type=)*

Van precedidos del prefijo del espacio de nombres Por, ej **xs**

xs:string: secuencia de caracteres **xs:integer**:numero entero **xs:decimal**:numero con parte decimal

xs:date: fecha específica del calendario gregoriano, en formato "YYYY-MM-DD";

xs:time: una instancia de tiempo que ocurre cada día, en formato "hh:mm:ss"

Ver mas tipos y ejemplos

2.4.Elementos simples. (sólo contienen datos simples NO hijos NO atributos

XML

```
<nombre>Juan</nombre>
<edad>23</edad>
<nota>8.5</nota>
```

XSD

```
<xs:element name="nombre" type="xs:string"/>
<xs:element name=”edad” type=”xs:integer” />
<xs:element name=”nota” type=”xs:decimal” />
```

2.3.Elementos Complejos con hijos simples

(Tienen hijos o atributo)

Secuencia(orden fijo)

(xs:secuence)

```
<alumno>
  <nombre>Juan</nombre>
  <edad>23</edad>
  <nota>8.5</nota>
</alumno>
```

```
<xs:element name="alumno">
<xs:complexType>
  <xs:sequence>
    <xs:element name="nombre" type="xs:string" />
    <xs:element name="edad" type="xs:integer" />
    <xs:element name="nota" type="xs:decimal" />
  </xs:sequence>
</xs:complexType>
</xs:element>
```

Si no importa el orden en lugar de xs:sequence se escribe xs:all

Elección (uno de varios)

(xs:choice)

```
<empleado>
<hombre>Juan</hombre>
</empleado>

O

<empleado>
<mujer>Maria</mujer>
</empleado>
```

```
<xs:element name="empleado">
<xs:complexType>
  <xs:choice>
    <xs:element name="hombre" type="xs:string" />
    <xs:element name="mujer" type="xs:string" />
  </xs:choice>
</xs:complexType>
</xs:element>
```

Cardinalidades (ver más)

- Es un atributo del hijo
- Por defecto es 1,1

minOccurs=número mínimo de ocurrencias del hijo dentro del padre

maxOccurs=número máximo de ocurrencias del hijo dentro del padre

Cardinalidades

```
<alumno>
<nombre>Juan</nombre>
<nota>8.5</nota>
<telefono>233232</telefono>
</alumno>
```

Suponemos un número máximo de 10 notas y mínimo de una y puede no tener teléfono o un número ilimitado de ellos

```
<xs:element name="alumno">
<xs:complexType>
  <xs:sequence>
    <xs:element name="nombre" type="xs:string" />
    <xs:element name="nota" type="xs:decimal"
minOccurs=”1” maxOccurs=”10” />
    <xs:element name="telefono" type="xs:integer"
minOccurs=”0” maxOccurs=”unbounded”/>
  </xs:sequence>
</xs:complexType>
</xs:element>
```

2.4Elementos Complejos con Hijos Complejos

```
<alumnos>
 <alumno>
   <nombre>Juan</nombre>
   <nota>8.5</nota>
 </alumno>
<alumnos>
```

Anidamiento:

```
<xs:element name="alumnos">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="alumno" >
  <xs:complexType>
  <xs:sequence>
      <xs:element name="nombre" type="xs:string" />
      <xs:element name="nota" type="xs:decimal" />
 </xs:sequence>
  </xs:complexType>
 </xs:element>
 </xs:sequence>
</xs:complexType>
</xs:element>
```

Referencia:

```
<alumnos>
 <alumno>
   <nombre>Juan</nombre>
   <nota>8.5</nota>
 </alumno>
<alumnos>
```

```
<xs:element name="alumnos">
<xs:complexType>
<xs:sequence>
    <xs:element ref="alumno" minOccurs="0"
maxOccurs="unbounded"></xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>


<xs:element name="alumno" >
   <xs:complexType>
    <xs:sequence>
     <xs:element name="nombre" type="xs:string" />
     <xs:element name="nota" type="xs:decimal" />
    </xs:sequence>
   </xs:complexType>
 </xs:element>
```

2.5.Atributos en elementos complejos

- **<xs:attribute name= type= />**
- Después de la declaración de los subelementos
- Siempre tiene un tipo simple
- No impone un orden
- Si no se define un tipo, será del tipo: anySimpleType. (Cualquier cadena de caracteres válidos)
- Si no se dice nada es opcional
- Tiene a su vez tres atributos para restringir los valores: **use**, **default** y **fixed**
  - **use:**
    - **required:** el atributo es obligatorio
    - **optional:** puede o no aparecer (es el valor por defecto)
  - **default:**
    - valor por defecto del atributo
    - **fixed:** valor fijo. Puede aparecer o no, pero si aparece solo puede tener ese valor.

```
<alumno dni="323B">
  <nombre>Juan</nombre>
  <nota>8.5</nota>
 </alumno>
```

```
<xs:element name="alumno" >
   <xs:complexType>
    <xs:sequence>
     <xs:element name="nombre" type="xs:string" />
     <xs:element name="nota" type="xs:decimal" />
    </xs:sequence>
     <xs:attribute name="dni" type="xs:string"
use="required"></xs:attribute>
   </xs:complexType>
  </xs:element>
```

2.6.Atributos en elementos simples

1. El elemento simple pasa ser complejo. (xs:complexType)
2. Se indica que su contenido es simple (xs:simpleContent)
3. Se hace una extensión sobre el tipo simple **del elemento** añadiendo el atributo)

<xs:extension base=”xs:string”>
<xs:attribute name= />
</xs:extensión>

```
<alumno dni=”323B”>
   <nombre>Juan</nombre>
   <nota eval=”1”>8.5
</nota>
  </alumno>
```

En este ejemplo el tipo simple del elemento nota es xs:decimal y el del atributo “eval” puede ser “xs:integer”

```
<xs:element name="alumno" >
<xs:complexType>
    <xs:sequence>
     <xs:element name="nombre" type="xs:string" />
     <xs:element name="edad" type="xs:integer" />
     <xs:element ref="nota"  />
    </xs:sequence>
    <xs:attribute name="dni" type="xs:string"
use="required"></xs:attribute>
  </xs:complexType>
 </xs:element>

 <xs:element name="nota">
 <xs:complexType>
 <xs:simpleContent>
  <xs:extension base="xs:decimal">
  <xs:attribute name="eval"
type="xs:integer"></xs:attribute>
  </xs:extension>
 </xs:simpleContent>
 </xs:complexType>
 </xs:element>
```

2.7 Tipos complejos con un nombre

Si un tipo complejo se va a utilizar varias veces puede se útil darle un nombre y reutilizarlo por ese nombre

```
<alumno dni=”323B”>
   <nombre>Juan</nombre>
   <nota eval=”1”>8.5
</nota>
  </alumno>
```

```
<xs:element name="alumno" type="persona">
</xs:element>

<xs:complexType name="persona">
    <xs:sequence>
     <xs:element name="nombre" type="xs:string" />
     <xs:element ref="nota"  />
    </xs:sequence>
    <xs:attribute name="dni" type="xs:string"
use="required"></xs:attribute>
  </xs:complexType>

</xs:schema>
```

2.8.Tipos restringidos de tipos simples (facetas) Ver más aquí

Se pueden definir tipos simples basado en restricciones a

```
<alumno dni=”323B”>
   <nombre>Juan</nombre>
   <nota eval=”1”>8.5
</nota>
  </alumno>
```

Restricciones minInclusive y maxInclusive

(equivale <= y >=, para > y < utilizar **minExclusive** y **maxExclusive**

*Restringir el valor del atributo **eval** a 1,2 o 3*

```
<xs:element name="nota">
  <xs:complexType>
  <xs:simpleContent>
   <xs:extension base="xs:decimal">
   <xs:attribute name="eval"
type="tipoevaluacion"></xs:attribute>
   </xs:extension>
  </xs:simpleContent>
  </xs:complexType>
  </xs:element>

<xs:simpleType name="tipoevaluacion">
 <xs:restriction base="xs:integer">
    <xs:minInclusive value="1"/>
    <xs:maxInclusive value="3"/>
 </xs:restriction>
</xs:simpleType>
```

Los nuevos nombres no llevan el prefijo xs:

Enumeration (solo son válidos los valores enumerados)	**length**	**maxLength y minLength**
<xs:simpleType name="tipoevaluacion"> <xs:restriction base="xs:integer"> <xs:enumeration value="1"/> <xs:enumeration value="2"/> <xs: enumeration value="3"/> </xs:restriction> </xs:simpleType>	<xs:simpleType name="dni"> <xs:restriction base="xs:string"> <xs:length value="8" /> </xs:restriction> </xs:simpleType> *(si queremos que la longitud del dni sea exactamente 8)*	<xs:simpleType name="tiponombre"> <xs:restriction base="xs:string"> <xs:minLength value="1"/> <xs:maxLength value="10"/> </xs:restriction> </xs:simpleType> *(si queremos que la longitud del nombre sea como mínimo 1 y máximo 10)*
fractionDigits (número de decimales) y **totalDigits** (número de dígitos)	**Pattern** (restringir los valores a los que cumplan un patrón)	
<xs:simpleType name="tiponota"> <xs:restriction base="xs:decimal"> <xs:fractionDigits value="1"/> <xs:totalDigits value="2"/> </xs:restriction> </xs:simpleType> *(si queremos que la nota solo tenga un decimal y como mucho 2 dígitos)*	<xs:element name="iniciales”> <xs:simpleType> <xs:restriction base="xs:string"> <xs:pattern value="[A-Z][A-Z][A-Z]"/> </xs:restriction> </xs:simpleType> </xs:element> *(el elemento “iniciales” solo aceptará 3 letras maýsculas)*	<xs:element name="iniciales” type=”tipoiniciales”></xs:element> <xs:simpleType name=”tipoiniciales”> <xs:restriction base="xs:string"> <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/> </xs:restriction> </xs:simpleType> </xs:element> *(el elemento “iniciales” solo aceptará 3 letras maýsculas o minúculas)*

Pattern (otros ejemplos)

<xs:pattern value="[abc]"/>	(solo puede valer o **a** o **b** o **c**)
<xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>	Cinco dígitos
<xs:pattern value="([a-z])*"/>	Un numero de cero o más letras mayúsculas
<xs:pattern value="([a-z][A-Z])+"/>	Un número indeterminado de parejas (al menos una) de una letra minúscula y otra mayúscula
<xs:pattern value="hombre|mujer"/>	Solo puede tomar los valores “hombre” o “mujer”
<xs:pattern value="[a-zA-Z0-9]{8}"/>	Una serie 8 caracteres que pueden ser letras minúsculas, mayúsculas y números.
<xs:simpleType name="tipotelefono"> <xs:restriction base="xs:string"> <xs:pattern value="[0-9]{9}"/> </xs:restriction> </xs:simpleType>	El tipo teléfono estaría compuesto por 9 dígitos
<xs:simpleType name="tipotelefono"> <xs:restriction base="xs:string"> <xs:pattern value="[0-9]{3}-[0-9]{6}"/> </xs:restriction> </xs:simpleType>	El tipo teléfono estaría compuesto por 3 dígitos un guión y 6 dígitos. Esto es posible porque el guión no se utiliza para hacer patrones, en otro caso, cuando queremos que coincida exactamente un carácter que se utiliza para hacer patrones ([, (, +, *) se le pone antes la barra \(, \[, \+, * . En cualquier caso siempre se puede utilizar la barra \ para hacer coincidir un carácter concreto. También funcionaría <xs:pattern value="[0-9]{3}****-[0-9]{6}"/>
<xs:simpleType name="tipotelefono"> <xs:restriction base="xs:string"> <xs:pattern value="\(\+[0-9]{3}\)[0-9]{9}"/> </xs:restriction> </xs:simpleType>	El tipo teléfono sería de la forma **(+034)926534675**