Validación de documentos XML: **XML Schema (Esquema XML)**.

- Contenidos:

La aparición y el desarrollo del lenguaje XML surge de la necesidad de jerarquizar y estructurar correctamente la información, no sólo para almacenarla, sino también para acceder a ella.
Inicialmente se usaron las Definiciones del Tipo de Documento (DTDs) en el lenguaje SGML para describir el vocabulario necesario para identificar todos los elementos de que iba a constar el documento y para expresar la estructura.
Pero las DTDs no satisficieron todas las necesidades inherentes a XML y pronto se vio necesario utilizar otros métodos más rigurosos y sofisticados para tratar la estructura y la semántica dentro de un documento XML. Así surgieron los Esquemas XML (XML Schema), como una forma de ampliación y mejora de las primitivas DTDs. Las DTDs y los Schemas son usados por los analizadores sintácticos o parsers para comprobar si un documento XML es válido.

XSD es un formato para definir la estructura de un documento XML. XSD está en proceso de sustitución del anterior formato DTD, y añade funcionalidad para definir la estructura XML con más detalle (Al igual que las DTDs, los Schemas describen el contenido y la estructura de la información, pero de una forma más precisa. Por ejemplo, un Schema nos permite definir el tipo del contenido de un elemento o de un atributo, y especificar si debe ser un número entero, una cadena de texto, una fecha, etc. Las DTDs no nos permiten hacer estas cosas).
Un documento XML Schema define un conjunto de documentos con una determinada estructura.
XML Schema Definition Language (XSDL) es un estándar del World Wide Web Consortium (W3C) que permite describir la estructura y el contenido de un documento XML. Se trata, por tanto, de una tecnología similar a Document Type Definition (DTD), pero mucho más potente y versátil.

Instancias del Esquema: son los documentos XML que se **validan con un mismo esquema**.
La definición del esquema puede ser poco restrictiva o muy restrictiva dando lugar a documentos válidos muy precisos.

XML Schema ofrece múltiples ventajas frente a DTD:

1. **Tipado fuerte**. En un DTD, las posibilidades de limitar el contenido de un elemento o un atributo son limitadas. Por ejemplo, un elemento se puede definir como EMPTY, ANY, un modelo de grupo o contenido mixto (elementos y texto). Pero no hay forma de especificar que el contenido de un elemento tiene que ser un entero o que no puede exceder un cierto número de caracteres. XML Schema supera las limitaciones de DTD que no permite especificar, por ejemplo, que un elemento debe contener un número entero y no una cadena de caracteres.

2. **Tipos de datos similares a los lenguajes de programación y bases de datos**. XML Schema incluye una serie de tipos de datos básicos similares a los que se puede encontrar en los lenguajes de programación o en las bases de datos relacionales u orientadas a objetos. Pero, además, el usuario puede definir sus propios tipos de datos a partir de los existentes.
(Existe una extensa lista de tipos de datos predefinidos para elementos y atributos que pueden ser ampliados o restringidos para crear nuevos tipos).

3. **Permite controlar con precisión el número de repeticiones**. En un DTD, la repetición de un elemento se describe con los siguientes modificadores: nada para una sola vez, ? para cero o una vez, * para cero o más veces y + para una o más veces. En XML Schema se puede especificar cualquier valor para el número de ocurrencias mínimo o máximo.
   (Permiten concretar con precisión la cardinalidad de un elemento, es decir, las veces que puede aparecer en un documento XML).

4. **Un XML Schema es un documento XML**. Un DTD se escribe mediante una notación que no tiene nada que ver con la sintaxis de un documento XML. Sin embargo, un XML Schema es un documento XML válido, por lo que se pueden emplear con él todas las herramientas desarrolladas para trabajar con XML.
   (Es un documento XML, por lo que se puede comprobar que está bien formado).

5. **Verdadera representación de claves primarias y ajenas**. Con DTD, si se quiere representar una base de datos en un documento XML, la única forma de especificar claves primarias y ajenas es con los tipos ID e IDREF. Sin embargo, presenta dos graves inconvenientes: ID tiene que ser único en todo el documento (en una base de datos, una clave primaria se puede repetir en distintas tablas) e IDREF tiene que referenciar un ID dentro del propio documento, pero no se puede especificar el tipo de elemento (puede ser una clave ajena a cualquier tabla). Con XML Schema, estos conceptos se pueden representar sin problemas.
6. **Permite mezclar vocabularios (juegos de etiquetas) gracias a los espacios de nombres**.

En la actualidad, el principal uso que se le da a XML es como medio de integración entre distintas aplicaciones.
La integración de XML con los sistemas gestores de bases de datos relacionales (SGBDR) es cada vez mayor y XML se ha convertido en un método factible para traspasar información de una aplicación a otra.

- Comparación Sistemas Gestores de Bases de Datos Relacionales – XML

	SGBDR	XML
Entidad	Tabla	Documento
Instancia de entidad	Fila	Elemento
Atributo de entidad	Columna	Atributo Valor de elemento Subelemento

La principal diferencia entre SGBDR y XML reside en la representación de las características (atributos) de una entidad. En un SGBDR, las características de la entidad se reprendan por columnas de una tabla. En un documento XML, las características pueden ser atributos, valores de elemento o subelementos.

Extensión archivos XML Schema (.xsd).
La extensión de un XML Schema suele ser **.xsd**.
Se deben emplear nombres cortos y sencillos. Hay que evitar el uso de espacios o de caracteres especiales en el nombre del archivo y también controlar el uso de mayúsculas y minúsculas puesto que en Internet existen multitud de sistemas operativos, que no pueden aceptar los mismos nombres de archivo que acepta el nuestro (Por ejemplo, hay sistemas operativos en los que las mayúsculas y minúsculas se distinguen y otros donde no).

1. Asociación de un Esquema a documento XML.

(**Referenciar en un documento XML su especificación XSD**)

La declaración o vinculación de un esquema al documento XML (o instancia del esquema) **se realiza en el documento XML**.
(Una vez que se ha creado una especificación XSD (se ha creado un esquema), en el documento XML o instancia del esquema se hace constar que debe ser conforme a dicha especificación).

En el documento XML o instancia del esquema debe declararse el esquema que se utiliza para validarlo incluyendo como atributos del elemento raíz el espacio de nombres y la ruta del archivo:

Documento XML

```
<?xml version="1.0" encoding="UTF-8"?>
<eraiz xmlns:xsi="EspacioDeNombres"
       xsi:noNamespaceSchemaLocation="ruta_archivo.xsd">
  ...
</eraiz >
```

Donde la etiqueta raíz del documento define los atributos:

- xmlns:xsi para declarar el espacio de nombres del esquema XSD.
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance":
  espacio de nombres de prefijo xsi, propio de las instancias de esquemas.

- xsi:noNamespaceSchemaLocation para vincular el documento XML con el esquema local XSD.

  ▪ Para especificar el esquema a utilizar tenemos dos alternativas:
  - xsi:noNamespaceSchemaLocation="url_xsd":
  ubicación de un esquema sin un espacio de nombres asociado. Se pueden indicar varios esquemas separados por espacios.
  xsi:noNamespaceSchemaLocation="esquema.xsd"

  - xsi:schemaLocation="url_ns url_xsd": ubicación de un esquema con un espacio de nombres asociado. Se pueden indicar varios esquemas separados por espacios.
  xsi:schemaLocation="http://www.misitio.es/espacio
  esquema.xsd"

  Nota: Si en el esquema se ha especificado un espacio de nombres (targetNamespace xmlns), éste debe ser coincidente con el indicado en el documento XML (xmlns="").
  Por ejemplo:
  En el XML Schema: de define el espacio de nombre

```
<xs:schema xmlns:sx="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.misitio.es/espacio"
           xmlns=""http://www.misitio.es/espacio">
    <xs:element name="nodoraiz">
        ...
    </xs:element>
</xs:schema>
```

En el documento XML:
<nodoraiz xmlns=”http://www.misitio.es/espacio”
xsi:schemaLocation=”http://www.misitio.es/espacio
esquema.xsd”
xmlns:sx=”http://www.w3.org/2001/XMLSchema”>
...
</nodoraiz>

Otros atributos de documento instancia:

- xsi:nil="boolean": indica si un elemento debe estar vacío, su valor por defecto es false, y para poder modificarlo a true se debe activar el atributo nillable en la definición del elemento en el esquema asociado.
  Por ejemplo:
  Esquema:
  <xs:element name="fecha" type="xs:date" nillable="true"/>
  Documento XML:
  <fecha xsi:nil="true"/>

- xsi:type="tipo": permite especificar el tipo de un elemento como xs:string u otro existente en el esquema asociado. Normalmente se utiliza para especificar algún tipo complejo.

Ejemplo vinculación
Al documento xml, cuyo nodo raíz es alumnos, se le asocia el esquema “esquemaalumnos.xsd”sin un espacio de nombre asociado
<?xml version="1.0" encoding="UTF-8"?>
<alumnos xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="esquemaalumnos.xsd">
...
</alumnos>

Ejemplo: noNamespaceSchemaLocation, schemaLocation
El archivo “apuntes.xsd” contiene un XML schema.
Si en dicho schema se definió
targetNamespace=”http://www.prueba.es/esquema1”
xmlns=”http://www.prueba.es/esquema1”
En un documento instancia que queremos asociar a este esquema tendremos:
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation:”http://www.prueba.es/esquema1 apuntes.xsd”
Si por el contrario en el documento “apuntes.xsd” no se especifica un “targetNamespace”, en el documento instancia XML tendremos:
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=”apuntes.xsd”

2. Estructura de un Esquema XML (XML Schema).

a) Cabecera

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs=" URI_de_un_espacio_de_nombres"
        targetNamespace="…"
        xmlns="…">
                ...
                ...
</xs:schema>
```

Donde:

- **Declaración de la versión XML**:

  `<?xml version="1.0" encoding="UTF-8" ?>`

  La etiqueta xml define la versión de XML utilizada y la codificación de caracteres del documento.
  (Los XML Schemas son documentos XML, por lo que todo esquema debe comenzar con una declaración XML)

- **Elemento raíz del esquema**: (no confundir con elemento raíz del XML)

  ```
  <xs:schema>
          …
  </xs:schema>
  ```

  xs:schema: elemento raíz del esquema. (Como todo documento XML debe incluir un elemento raíz)

- **Espacio de nombres**: Al elemento raíz schema se le puede incluir el atributo xmlns para indicar el espacio de nombres.
  Se puede definir sin prefijo o utilizar uno de los comunes: xs ó xsd.

  xmlns:prefijo="URI_de_un_espacio_de_nombres"
  targetNamespace: atributo que indica el espacio de nombres que se está definiendo.
  Si el esquema utiliza elementos y atributos definidos en la especificación XML Schema de la W3C se enlazaría al espacio de nombres "http://www.w3.org/2001/XMLSchema" (por ejemplo, se utiliza el prefijo xs definir el espacio de nombres, prefijo que se usará en las definiciones de elementos y atributos del esquema a desarrollar).

  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  (Posteriormente, cuando se quiera hacer referencia a elementos definidos en la especificación XML Schema se referenciará con este prefijo. Por ejemplo, <xs:element .../>)

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

b) Contenido:
A continuación de la cabecera se detallarán las declaraciones globales de elementos (comenzando por el elemento raíz del XML) y atributos, definiciones de tipos de elementos y atributos, anotaciones.

- Elemento raíz del XML: todo documento XML bien formado contiene un elemento raíz. El esquema deberá contener como mínimo la declaración de ese elemento raíz.
  <xs:element name="nombre_elemento">

  ▫ Múltiples elementos raíz: se pueden definir esquemas con varios elementos raíz, de manera que se tomen como válidos los documentos XML que incluyan cualquiera de esos elementos como raíz.
  Ejemplo:
  <?xml version="1.0"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="simple"/>
  <xs:element name="complejo"/>
  </xs:schema>

  Cualquier documento XML cuyo elemento raíz fuera tanto "simple" como "complejo" se consideraría válido.
  Documento XML1:
  <simple>
  ...
  </simple>

  Documento XML2:
  <complejo>
  ...
  </complejo>
  (Sería algo similar a las secciones condicionales en el DTD, pero únicamente aplicable al elemento raíz del documento XML).

El orden de componentes es irrelevante. Un esquema es un conjunto de declaraciones de componentes (elementos, atributos y tipos) que sirve para establecer la estructura que deben tener los documentos instancia, pero el orden en que se declaren no es significativo ni afecta a su funcionamiento.

Documentación de Esquemas

Dentro de los esquemas pueden utilizarse comentarios para generar documentación mediante herramientas de autor.

xs:annotacion
xs:annotation (xs:appinfo | xs:documentation)*

xs:annotacion: permite añadir comentarios.

Elemento padre: cualquiera.
Atributos opcionales: id.

xs:appinfo

xs:appinfo (contenido XML bien formado)

xs:appinfo: establece la información que se va a utilizar en la documentación.
Elemento padre: xs:annotation.
Atributos opcionales:
- source: URI que especifica el origen de la información sobre la aplicación.

xs:documentation

xs:documentation (contenido XML bien formado)

xs:documentation: permite asociar comentarios textuales sobre el esquema.
Elemento padre: xs:annotation.
Atributos opcionales:
- source: URI que especifica el origen de la información sobre la aplicación.

Ejemplo:

```
<xs:annotation>
    <xs:appinfo>Nota</xs:appinfo>
    <xs:documentation xml:lang="es">
        Este esquema tiene una nota
    </xs:documentation>
</xs:annotation>
```

Comentarios

<!—Comentario -->

Dentro de los esquemas se pueden realizar comentarios simples explicativos.

3. Componentes básicos de un Esquema XML.

Los **componentes imprescindibles para construir un esquema XSD** son los siguientes:

- **xs:schema**
  Elemento raíz del esquema.
- **xs:element**
  Elemento raíz del documento XML y resto de elementos.
- **xs:attribute**
  Atributos a asociar a los elementos.

A. xs:schema:

```
<xs:schema xmlns:xs="…">
        …
</xs:schema>
```

Componente de declaración del esquema y es el **elemento raíz de todo esquema XML** (no confundir con elemento raíz del documento XML).
Entre otros, los principales atributos que podemos utilizar en la declaración son los siguientes:

Atributo	Descripción
xmlns	Referencia URI que indica el/los espacios de nombres a utilizar en el esquema. Si no se especifica ningún prefijo, los componentes del espacio de nombres pueden utilizarse de forma no cualificada.
id	Identificador único para el elemento.
targetNamespace	Referencia URI al espacio de nombres del esquema. Crea un espacio de nombres al que pertenecen los elementos que se definen en el esquema
elementFormDefault (unqualified, qualified)	Formato de los nombres de los elementos en el espacio de nombres del esquema, puede ser unqualified (sin prefijo) o qualified (con prefijo). Para especificar si los elementos declarados en él deben estar certificados por un espacio de nombres, ya sea explícitamente mediante un prefijo o implícitamente de forma predeterminada, cuando se utilicen en un documento instancia XML. (Para indicar si los elementos dentro del esquema necesitan ir precedidos del prefijo del espacio de nombres). Nos da la posibilidad de definir elementos que pertenecen al espacio de nombres. Valor por defecto: unqualified
attributeFormDefault (unqualified, qualified)	Formato de los nombres de los atributos en el espacio de nombres del esquema. Para especificar si los atributos declarados en él deben estar certificados por un espacio de nombres, ya sea explícitamente mediante un prefijo o implícitamente de forma predeterminada, cuando se utilicen en un documento instancia XML. (Para indicar si los atributos dentro del esquema necesitan ir precedidos del prefijo del espacio de nombres). Nos da la posibilidad de definir atributos que pertenecen al espacio de nombres. Valor por defecto: unqualified
version	Versión del esquema.

Ejemplo: xmlns

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        ...
</xs:schema>
```

Ejemplo: targetNamespace

```
<?xml version="1.0"?>
<xs:schema targetNamespace:"http://www.prueba.es/esquema1"
        ...
</xs:schema>
```

Ejemplo: elementFormDefault, attributeFormDefault

Documento XML con espacio de nombres asociado:

```
<p:persona
        xmlns:p="http://www.prueba.es/persona"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation= "http://www.prueba.es/persona people.xsd"
    >
        <p:nombre>John</p:nombre>
        <p:direccion>John</p:direccion>
</p:persona>
```

El documento XSD que continúe el espacio de nombres es people.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.prueba.es/persona"
        elementFormDefault="qualified"
        attributeFormDefault="qualified">
            <xs:element name="persona">
                <xs:complexType>
                <xs:sequence>
                    <xs:element name="nombre"
                        type="xs:string" minOccurs="0"
                        maxOccurs="unbounded"/>
                    <xs:element name="direccion"
                        type="xs:string" minOccurs="0"
                        maxOccurs="unbounded"/>
                </xs:sequence>
                </xs:complexType>
            </xs:element>
</xs:schema>
```

B. xs:element:

```
<xs:element name="nombre_del_elemento" … type="tipo_de_dato"/>
```

Este componente **permite declarar los elementos del documento XML**.
Entre otros, los principales atributos que podemos utilizar en la declaración son los siguientes:

Atributo	Descripción
name	Indica el nombre del elemento. Obligatorio si el elemento padre es <xs:schema> (Si el elemento padre en el esquema es xs:schema estaríamos declarando el elemento raíz del documento XML, por eso es obligatorio). Se recomienda su uso, dado que hace que el esquema sea más legible y el elemento reutilizable.
ref	Indica que la declaración del elemento se encuentra en otro lugar del esquema. No se puede usar si el elemento padre es <xs:schema>. No puede aparecer junto con name.
type	Indica el tipo de dato que almacenará el elemento (ver apartado de tipos de datos). No puede aparecer junto con ref.
default	Es el valor que tomará el elemento al ser procesado si en el documento XML no ha recibido ningún valor. Sólo se puede usar con tipo de dato textual.
fixed	Indica el único valor (o valor obligatorio) que puede contener el elemento en el documento XML. Sólo se puede usar con tipo de dato textual.
minOccurs	Indica el mínimo número de ocurrencias que deben aparecer de ese elemento en el documento XML. No se puede usar si el elemento padre es <xs:schema>. (0,1, 2, …, unbounded) Va desde 0 hasta ilimitado (unbounded). Por defecto 1.
maxOccurs	Indica el máximo número de ocurrencias que pueden aparecer de ese elemento en el documento XML. No se puede usar si el elemento padre es <xs:schema>. (0,1, 2, …, unbounded) Va desde 0 hasta ilimitado (unbounded). Por defecto 1.
id	Identificador único para el elemento.
form (unqualified, qualified)	Formato del nombre del elemento. "qualified" indica que el elemento debe ser calificado con el prefijo del espacio de nombres. Por defecto tomará el valor definido en elementFormDefault del componente xs:schema.
substitutionGroup	Indica que el nombre de otro elemento puede ser sustituido por este elemento y sólo se puede utilizar si el padre es xs:schema.
nillable (false, true)	Con valor true indica que puede aparecer el atributo xsi:nil asociado al elemento en el documento instancia. Esto permite indicar si el contenido de un elemento puede ser nulo.

	(Se aplica al contenido del elemento, permitiendo que pueda no tener contenido. No se refiere a valores de atributos, por lo que el elemento con contenido nulo puede tener atributos con valores) Por defecto: false.
abstract (false, true)	Con valor true indica que el elemento no puede aparecer en una instancia. (En su lugar, otro elemento cuyo atributo substitutionGroup contiene el nombre completo (QName) de este elemento debe aparecer en lugar de este elemento). Por defecto: false.
final (extension, restriction, #all)	Indica si el elemento se puede derivar de alguna manera, por extensión, por restricción o por ambas. • extensión - impide elementos derivados, por extensión • restricción - impide elementos derivados por la restricción • #all - evita todos los elementos derivados

▪ Elementos No Declarados: any

```
<xs:any …>
```

xs:any: permite que en el documento XML aparezcan elementos que no han sido declarados explícitamente.

Ejemplo: El elemento coche tendrá como descendientes los elementos marca, modelo y tras este cualquier otro.

```
<xs:element name="coche">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="marca" type="xs:string"/>
            <xs:element name="modelo" type="xs:string"/>
            <xs:any minOccurs="0" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

▪ Grupos de Elementos: group

```
<xs:group name="nombreGrupoElementos">
    <xs:sequence>
        <!-- Elementos -->
        <xs:element … />
    </xs:sequence>
</xs:group>
```

Pueden definirse grupos de elementos (xs:group) a los que se les da un nombre para poder utilizarlos posteriormente en la definición de tipos complejos sin necesidad de repetir cada uno de esos elementos.

Ejemplo: Se define un grupo de elementos "elementosLibro", que es reutilizado en la declaración del tipo de datos "tipoLibro", y añadiendo además el elemento "sinopsis".

```
<xs:group name="elementosLibro">
    <xs:sequence>
        <xs:element name="titulo" type="xs:string"/>
        <xs:element name="autor" type="xs:string"
      maxOccurs="4"/>
        ...
    </xs:sequence>
</xs:group>

<xs:complexType name="tipoLibro">
    <xs:sequence>
        <xs:group ref="elementosLibro"/>
        <xs:element name="sinopsis" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
```

▪ Grupos de Sustitución: substitutionGroup

Pueden definir elementos que son sinónimos y por tanto pueden utilizarse indistintamente.
El Esquema XML proporciona un mecanismo, llamado grupos de sustitución, que permite que los elementos sean sustituidos por otros elementos.

La existencia de un grupo de sustitución no requiere que ninguno de los elementos de la clase sea utilizado, simplemente proporciona un mecanismo para permitir los elementos ser usados de forma intercambiable.

(Un uso muy habitual es poder definir los mismos elementos en varios idiomas).

Ejemplo: substitutionGroup

Declaración de Elementos Sustituibles: cuando desde el documento xml se haga referencia a "comentarioEnvio" será sustituido por "comentario":

```
<element name="comentarioEnvio" type="string"
      substitutionGroup="ipo:comentario"/>
```

Documento XML:

```
<ipo:comentarioEnvio>
        Utilizar recubrimiento de oro si es posible
</ipo:comentarioEnvio>
```

Ejemplo: el elemento "customer" será sustituido por "cliente", que es del tipo de datos "tipoInfoCliente". Así mismo, el elemento "name" será sustituido por el elemento "nombre" y el elemento "surname" será sustituido por el elemento "apellido"

```
<xs:complexType name="tipoInfoCliente">
        <xs:sequence>
```

```
                <xs:element name="nombre" type="xs:string">
                <xs:element name="apellido" type="xs:string">
            </xs:sequence>
        </xs:complexType>

        <xs:element name="cliente" type="tipoInfoCliente"/>
        <xs:element name="customer" substitutionGroup="cliente"/>
        <xs:element name="name" substitutionGroup="nombre"/>
        <xs:element name="surname" substitutionGroup="apellido"/>
```

○ Ejemplos declaración elementos:

Ejemplo: El elemento nombre es de tipo texto y puede aparecer de 1 a varias veces, tomando como valor por defecto "alumno"

```
<xs:element name="nombre" type="xs:string" default="alumno" minOccurs="1" maxOccurs="unbounded" />
```

Ejemplo: el elemento está definido como "alumno" en otro lugar del documento y puede aparecer de 1 a varias veces

```
<xs:element ref="alumno" minOccurs="1" maxOccurs="unbounded" />
```

Ejemplo: el elemento "alumno" está definido, formando una agrupación que permite hacer referencia a ella.

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

<xs:element name="alumnos">
    <xs:sequence>
        <xs:element ref="alumno"/>
    </xs:sequence>
</xs:element>
```

Ejemplo: El elemento "autor" es de tipo texto y puede aparecer de una a tres veces (no se indica minOccurs dado que, por defecto, se toma 1)

```
<xs:element name="autor" type="xs:string" maxOccurs="3"/>
```

Ejemplo: El elemento "semaforo" es de tipo texto y por defecto tomará el valor "verde"

```
<xs:element name="semaforo" type="xs:string" default="verde"/>
```

Si en la instancia se asigna un valor al elemento se tomará ese valor, pero en caso contrario se le asignará "verde":

<semaforo></semaforo> es equivalente a

<semaforo>verde</semaforo>

Ejemplo: El elemento "semaforo" es de tipo texto y tomará el valor "rojo" de forma obligatoria.

```
<xs:element name="semaforo" type="xs:string" fixed="rojo"/>
```

Si en la instancia no se asigna un valor, el elemento tomará el valor "rojo":

<semaforo></semaforo> es equivalente a <semaforo>rojo</semaforo>

Si en la instancia se asigna un valor, sólo será válido el valor "rojo":

<semaforo>rojo</semaforo> (válido)
<semaforo>azul</semaforo> (no válido)

Ejemplo: Uso de identificador único en la declaración de un elemento.

```
<xs:element name="matricula" id="matr" type="xs:string">
```

Ejemplo: nillable(false, true)
Se define el elemento "fechaEnvio", tipo fecha, permitiendo que pueda aceptar valores nulos.

```
<xsd:element name="fechaEnvio" type="xsd:date" nillable="true"/>
```

Documento XML:

```
<fechaEnvio xsi:nil="true"></fechaEnvio>
```

Ejemplo: form(unqualified, qualified).
En cabecera se establece que los elementos estarían "calificados" elementFormDefault="qualified", dentro del espacio de nombres targetNamespace="http://empresa.com/ns/library" estableciendo como prefijo "lib" para hacer referencia a los nombres establecidos en dicho espacio de nombres (xmlns:lib="http://dyomedea.com/ns/library")

Las referencias a los tipos de datos del esquema XML W3C se realizan indicando el prefijo "xs", mientras que las referencias a componentes definidos en el espacio de nombres de destino se realizan con un prefijo "lib".

```
<?xml version="1.0"?>
<xs:schema targetNamespace="http://dyomedea.com/ns/library"
        elementFormDefault="qualified" attributeFormDefault="unqualified"
        xmlns:lib="http://dyomedea.com/ns/library"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="library">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="book" type="lib:bookType"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="person">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" type="xs:string"/>
                <xs:element name="born" type="xs:date"
```

```
                                    minOccurs="0"/>
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:ID" use="required"/>
                    </xs:complexType>
                </xs:element>

                <xs:complexType name="bookType">
                    <xs:sequence>
                        <xs:element name="isbn" type="xs:NMTOKEN"/>
                        <xs:element name="title" type="xs:string"/>
                        <xs:element name="authors">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element ref="lib:person"
                                        maxOccurs="unbounded"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>

                        <xs:element name="characters">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element ref="lib:person"
                                        maxOccurs="unbounded"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>

                    <xs:attribute name="id" type="xs:ID" use="required"/>
                    <xs:attribute name="available" type="xs:string"
                        use="required"/>
                </xs:complexType>

</xs:schema>
```

- **Número de Ocurrencias: Equivalencia entre DTD y XML Schema**

XML Schema	DTD equivalente
minOccurs=”0” maxOccurs=”unbounded”	element*
MinOccurs=”0” maxOccurs=”1”	element?
minOccurs=”1” maxOccurs=”unbounded” (sin límites)	element+
minOccurs=”1” maxOccurs=”1”	element

C. xs:attribute:

```
<xs:attribute name="nombre_del_atributo" … type="tipo_de_dato"/>
```

Este componente **permite declarar los atributos de los elementos del documento XML**.
(La sintaxis de definición es como la de los elementos simples)

Solo los elementos complejos pueden contener atributos (los elementos simples no pueden contener atributos, así como tampoco otros elementos).

Entre otros, los principales atributos que podemos utilizar en la declaración son los siguientes:

Atributo	Descripción
name	Indica el nombre del atributo.
ref	Indica que la declaración del atributo se encuentra en otro lugar del esquema. No puede aparecer junto con name. es incompatible con los atributos type y form, y con xs:simpleType.
type	Indica el tipo de dato que almacenará el atributo. No puede aparecer junto con ref.
use(optional, required, prohibited)	Indica si la existencia del atributo es opcional (optional), obligatoria (required) o prohibida (prohibited). Por defecto optional.
default	Es el valor que tomará el elemento al ser procesado si en el documento XML no ha recibido ningún valor. Sólo se puede usar con tipo de dato textual.
fixed	Indica el único valor que puede contener el elemento en el documento XML. Sólo se puede usar con tipo de dato textual. Los atributos default y fixed no pueden aparecer de forma simultánea.
id	Identificador único para el elemento.
form (unqualified, qualified)	Especifica el formato del atributo. “qualified”: indica que el atributo debe estar calificado con el prefijo del espacio de nombres. “unqualified”: indica que no es necesario que el atributo esté calificado con el prefijo del espacio de nombres y se compara con el nombre del atributo. Por defecto tomará el valor definido en attributeFormDefault del componente xs:schema.

▪ Atributos No Declarados: anyAttribute

```
<xs:anyAttribute …>
```

xs:anyAttribute: permite que en el documento XML aparezcan asociados a un elemento atributos que no han sido declarados explícitamente.

Ejemplo: El elemento coche tendrá como descendientes los elementos marca y modelo, y cualquier atributo.

```
<xs:element name="coche">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="marca" type="xs:string"/>
            <xs:element name="modelo" type="xs:string"/>
        </xs:sequence>
        <xs:anyAttribute />
    </xs:complexType>
</xs:element>
```

▪ Grupos de Atributos: attributeGroup

```
<xs:attributeGroup name="nombreGrupoAtributos">
    <!-- Atributos-->
    <xs:attribute
</xs:attributeGroup>
```

Pueden definirse grupos de atributos (xs:attributeGroup) a los que se les da un nombre para poder utilizarlos posteriormente en la definición de tipos complejos sin necesidad de repetir cada uno de esos atributos.

Ejemplo:

```
<xs:attributeGroup name="personattr">
    <xs:attribute name="attr1" type="string"/>
    <xs:attribute name="attr2" type="integer"/>
</xs:attributeGroup>

<xs:complexType name="person">
    <xs:attributeGroup ref="personattr"/>
</xs:complexType>
```

○ Ejemplos declaración atributos:

Ejemplo: se define el elemento "curso" que tiene un atributo "grupo"
Documento XML

```
<curso grupo="B">2</curso>
```

Esquema XML (recordar, un elemento simple (como podría ser éste) no puede tener atributos, por lo que lo convertimos en complejo)

```
<xs:element name="curso">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:integer">
                <xs:attribute name="grupo" type="xs:string"/>
            </xs:extension>
        </xs:simpleContent>
```

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

Ejemplo: El atributo "moneda" es de tipo texto, su uso es obligatorio y tomará por defecto el valor "euro"

```
<xs:attribute name="moneda" type="xs:string" default="euro" use="required" />
```

Ejemplo: el atributo ya está definido en otra parte del documento y su uso es obligatorio

```
<xs:attribute ref="moneda" use="required" />
```

Ejemplo: El atributo "unidad" es de tipo texto y tomará el valor fijo "minutos" (no puede adoptar ningún otro valor)

```
<xs:attribute name="unidad" type="xs:string" fixed="minutos"/>
```

Ejemplo: El atributo"idEmpleado" es de tipo entero positivo y su uso es obligatorio:

```
< xs:attribute name="idEmpleado" type="xs:positiveInteger" use="required"/>
```

Ejemplo: definición de atributo con restricciones y asociación a tipo de dato complejo

```
<xs:attribute name="code">
        <xs:simpleType>
                <xs:restriction base="xs:string">
                        <xs:pattern value="[A-Z][A-Z]"/>
                </xs:restriction>
        </xs:simpleType>
</xs:attribute>
<xs:complexType name="someComplexType">
        <xs:attribute ref="code"/>
</xs:complexType>
```

Ejemplo: declaración de atributo y asociación al elemento

```
<xs:attribute name="isbn" type="xs:string"/>
<xs:element name="libro">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="titulo"/>
                        <xs:element ref="autor"/>
                        <xs:element ref="personaje" minOccurs="O"
                                                maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute ref="isbn"/>
        </xs:complexType>
</xs:element>
```

(En este ejemplo inicialmente se ha definido el atributo. Posteriormente se ha definido el elemento y se indica que el atributo (ref, ya definido en otra parte del documento, exactamente lo hemos definido anteriormente) es parte del tipo de dato complejo del elemento.

4. Tipos de Datos en un Esquema XML.

```
<xs:element name="nombre_del_elemento" … type="xs:tipo_de_dato"/>
<xs:attribute name="nombre_del_atributo" … type="xs:tipo_de_dato"/>
<xs:extension base="xs:tipo_de_dato">
<xs:restriction base="xs:tipo_de_dato">
```

Permiten establecer los valores que podrán tomar tanto elementos como atributos mediante el atributo "type" en su declaración (o por extensión o restricción en datos complejos).
En función del tipo de datos el elemento o el atributo podrá contener ciertos valores (por ejemplo, un tipo entero no podrá contener decimales; un tipo numérico no podrá contener letras, etc.).

Clasificación de los Tipos de Datos.

- En función de su **definición**:
  - **Predefinidos** (built-in types): se integran en la especificación de los esquemas XML.
  - **Construidos**: son creados por el usuario a partir de un tipo predefinido o de otro definido por el usuario.

- En función de su **contenido**:
  - **Simples**: contienen un único valor (atómicos). (No pueden tener descendientes ni atributos)
  - **Complejos**: pueden contener texto, descendientes y/o atributos.

Agrupando ambas clasificaciones podemos establecer dos grandes grupos de tipos de datos que se pueden utilizar en los esquemas XSD:

A. Tipos de datos simples (xs:simpleType): No contienen subelementos ni atributos.

Se dividen en los siguientes:

a) Tipos de datos **predefinidos**
   a. Primitivos.
   b. No Primitivos (descienden de primitivos).

b) Tipos de datos **construidos** con nuestras propias restricciones y basados en los tipos de datos predefinidos.

B. Tipos de datos complejos (xs:complexType): Pueden contener subelementos y atributos.

Se dividen en los siguientes:

1. Elementos dentro de otros elementos.
2. Elementos que solo tienen atributos.
3. Elementos mixtos que tienen atributos y otros elementos.
4. Elementos vacíos.
5. Extensión de Tipos de Datos complejos.

A. Tipos de Datos Simples (xs:simpleType):

Representan valores atómicos, salvo alguna excepción.
Se pueden asignar tanto a elementos que sólo tengan contenido textual como a atributos.

Tipos de Datos Simples:

a) Predefinidos:

a. Primitivos
b. No Primitivos.

b) Construidos a partir de un tipo de dato simple (restricciones).

Nota: no confundir un tipo de dato construido (simple) con un dato complejo.

a) Tipos de Datos Predefinidos.

```
<xs:element name="nombre_del_elemento" … type="xs:tipo_de_dato"/>
<xs:attribute name="nombre_del_atributo" … type="xs:tipo_de_dato"/>
```

Son 44 los tipos de datos que define el esquema XSD, clasificados de una manera jerárquica, en el que los elementos hijos poseen las mismas características del padre más alguna particularidad añadida que los distinga.
(Los 44 tipos de datos predefinidos se organizan de manera jerárquica, de manera que cada tipo es igual a su tipo "padre" incorporando alguna particularidad).

Se parte del tipo genérico "anyType", del que derivan todos los demás tipos, tanto predefinidos como construidos, y a partir de él se desciende hacia tipos capa vez más restrictivos y por tanto más precisos.
A la hora de utilizar elementos y atributos lo recomendable es utilizar el tipo más restrictivo que permita representar cada uno de sus posibles valores.
(Por ejemplo, se puede representar un elemento "edad" con el tipo "anyType" o con "decimal", pero es más adecuado utilizar "nonNegativeInteger").

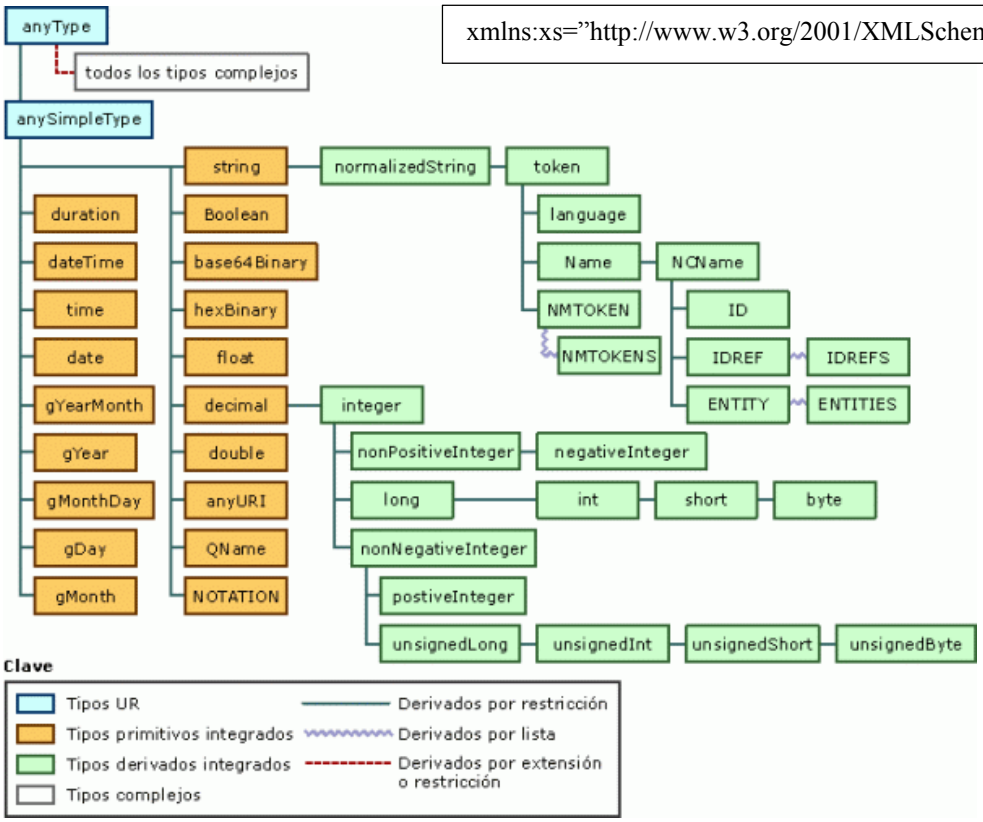
- Pueden ser asignados tanto a elementos como a atributos.
- Deben ir precedidos del prefijo "xs:" porque pertenecen al espacio de nombres: "http://www.w3.org/2001/XMLSchema"

Los tipos de datos **predefinidos** los podemos dividir en:

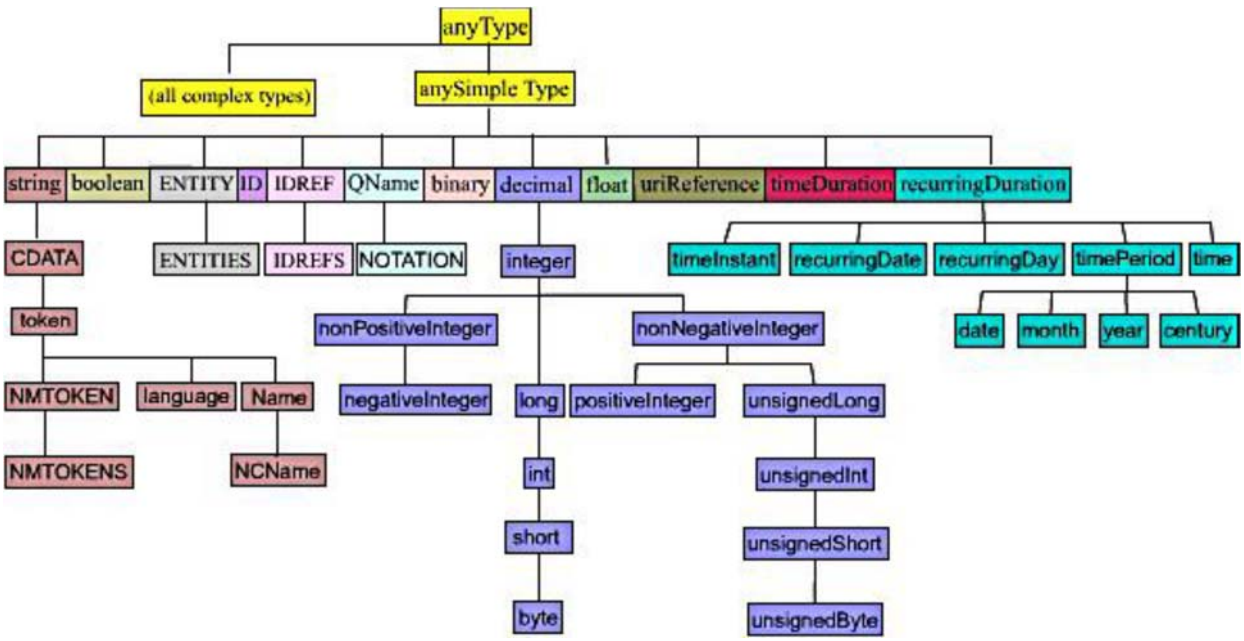
- **Primitivos**: Descienden directamente de xs:anySimpleType.
- **No primitivos**: Descienden de alguno de los primitivos.

El tipos de datos predefinido principal es **xs:anyType**, el más genérico y del cual derivan el resto de tipos de datos predefinidos como podemos ver en la siguiente imagen.

xmlns:xs=”http://www.w3.org/2001/XMLSchema”



(Tipos de datos predefinidos. Fuente: Microsoft)



(Tipos de datos predefinidos.)

▫ Tipos de datos **predefinidos Primitivos**:

Tipo de Dato	Descripción
string	Cadena de caracteres.
boolean	Valores booleanos que pueden ser: 0, 1, true, false.
decimal	Números reales.
float	Números en punto flotante de 32 bits.
double	Números en punto flotante de 64 bits.
duration	Duración representada en "Años + Meses + Días + Horas + Minutos + Segundos" (P10Y5M4D10H12M50S).
dateTime	Fecha y hora en formato CCYY-MM-DDThh:mm:ss donde CC es el siglo, YY el año, MM el mes, DD el día, precedido del carácter (-) indica un número negativo. La T es el separador de hh horas, mm minutos, y ss segundos. Puede seguirse de una Z para indicar la zona UTC (Universal Time Coordinated).
time	Hora en formato hh:mm:ss.sss.
date	Fecha en formato CCYY-MM-DD.
gYearMonth	Sólo el año y mes en formato Gregoriano CCYY-MM.
gYear	Sólo el año en formato Gregoriano CCYY.
gMonthDay	Sólo el mes y el día en formato Gregoriano --MM-DD.
gDay	Sólo el día en formato Gregoriano ---DD.
gMonth	Sólo el mes en formato Gregoriano --MM--.
hexBinary	Secuencia de dígitos hexadecimales (0, ..., 9, A, ..., F). Suele utilizarse para describir documentos o formatos binarios.
base64Binary	Secuencia de dígitos hexadecimales en base 64. Suele utilizarse para describir documentos o formatos binarios.
anyURI	Cualquier identificador URI.
QName	Cualquier nombre cualificado del espacio de nombres
NOTATION	Tipo de datos para atributo compatible con DTD.

Ejemplo:

```
<xsd:element name=”precioTipo” type=”xsd:string”/>

<xsd:attribute name="demanda" type=”xsd:decimal”/>
```

Nota: para los tipos de fecha y hora.
Se puede añadir también una Z opcional, y + hh:mm ó – hh:mm al final para indicar una zona horaria diferente. Usamos Z si la hora se ajusta a la Hora Media de Greenwich (GMT – Greenwich Mean Time) o la Hora Universal Coordinada (UTC – Coordinate Universal Time), o utilizaremos las horas y minutos adicionales para indicar diferencias respecto al GMT. Las horas se datan según el sistema de 24 horas.

▫ Tipos de datos **predefinidos No Primitivos** (derivan de los primitivos):

Tipo de dato	Descripción
normalizedString	Representa cadenas normalizadas de espacios en blanco (sustituye tabulador, salto y retorno por espacios). Este tipo de datos se deriva de string.
token	Representa cadenas convertidas en tokens (cadenas de texto sin tabulador, salto, retorno ni espacios delante o detrás). Este tipo de datos se deriva de normalizedString.
language	Representa identificadores de lenguaje natural (definidos por RFC 1766). Valores válidos para xml:lang (xsd:language=”ES”) Este tipo de datos se deriva de token.
IDREFS	Representa el tipo de atributo IDREFS. Contiene un conjunto de valores de tipo IDREF.
ENTITIES	Representa el tipo de atributo ENTITIES. Contiene un conjunto de valores de tipo ENTITY.
NMTOKEN	Representa el tipo de atributo NMTOKEN. NMTOKEN es un juego de caracteres de nombres (letras, dígitos y otros caracteres) en cualquier combinación. A diferencia de Name y NCName, NMTOKEN, no tiene restricciones del carácter inicial. Este tipo de datos se deriva de token.
NMTOKENS	Representa el tipo de atributo NMTOKENS. Contiene un conjunto de valores de tipo NMTOKEN.
Name	Representa nombres en XML. Name es un token que empieza con una letra, carácter de subrayado o signo de dos puntos, y continúa con caracteres de nombre (letras, dígitos y otros caracteres). Este tipo de datos se deriva de token.
NCName	Representa nombres (QName) sin el prefijo ni el signo de dos puntos. Este tipo de datos es igual que Name, excepto en que no puede empezar con el signo de dos puntos. Este tipo de datos se deriva de Name.
ID	Representa el tipo de atributo ID definido en la recomendación de XML 1.0. El IDno debe incluir un signo de dos puntos (NCName) y debe ser único en el documento XML. Este tipo de datos se deriva de NCName.
IDREF	Representa una referencia a un elemento que tiene un atributo ID que coincide con el ID especificado. IDREF debe ser un NCName y tener un valor de un elemento o atributo de tipo ID dentro del documento XML. Este tipo de datos se deriva de NCName.

ENTITY	Representa el tipo de atributo ENTITY definido en la recomendación de XML 1.0. Es una referencia a una entidad sin analizar con un nombre que coincide con el especificado. ENTITY debe ser un NCName y declararse en el esquema como nombre de entidad sin analizar. Este tipo de datos se deriva de NCName.
integer	Números enteros. Representa una secuencia de dígitos decimales con un signo inicial (+ o -) opcional. Este tipo de datos se deriva de decimal.
nonPositiveInteger	Números enteros negativos, incluido el cero. Representa un número entero menor o igual que cero. nonPositiveInteger consta de un signo negativo (-) y una secuencia de dígitos decimales. Este tipo de datos se deriva de integer.
negativeInteger	Números enteros negativos (no incluye el cero). Representa un número entero menor que cero. Consta de un signo negativo (-) y una secuencia de dígitos decimales. Este tipo de datos se deriva de nonPositiveInteger.
nonNegativeInteger	Números enteros positivos, incluido el cero. Representa un número entero mayor o igual que cero. Este tipo de datos se deriva de integer.
positiveInteger	Números enteros positivos (no incluye el cero). Representa un número entero mayor que cero. Este tipo de datos se deriva de nonNegativeInteger.
long	Representa un entero con un valor mínimo de -9223372036854775808 y un valor máximo de 9223372036854775807. Este tipo de datos se deriva de integer.
int	Representa un entero con un valor mínimo de -2147483648 y un valor máximo de 2147483647. Este tipo de datos se deriva de long.
short	Representa un entero con un valor mínimo de -32768 y un valor máximo de 32767. Este tipo de datos se deriva de int.
unsignedLong	Representa un entero con un valor mínimo de cero y un valor máximo de 18446744073709551615. Este tipo de datos se deriva de nonNegativeInteger.
unsignedInt	Representa un entero con un valor mínimo de cero y un valor máximo de 4294967295. Este tipo de datos se deriva de unsignedLong.
unsignedShort	Representa un entero con un valor mínimo de cero y un valor máximo de 65535. Este tipo de datos se deriva de unsignedInt.
unsignedByte	Representa un entero con un valor mínimo de cero y un valor máximo de 255. Este tipo de datos se deriva de unsignedShort.

Ejemplo:

<xsd:element name=”edad” type=”xsd:positiveInteger”/>

<xsd:attribute name="demanda" type=”xsd:long”/>

b) Tipos de Datos construidos.

La forma más sencilla de crear un nuevo tipo a partir de uno ya existente es añadir condiciones a alguno de los tipos predefinidos en el XML Schema.

Se define un tipo simple <xs:simpleType …> y se le puede añadir restricciones con el elemento <xs:restriction …>.

Los tipos de datos construidos son generados por el usuario a partir de un tipo de dato predefinido y aplicándoles restricciones si se desea.

```
<xs:simpleType  name="….">
        <xs:restriction base="xs:tipoOrigen">
                <xs:tipoRestricción value=”..”/>
                …
        </xs:restriction>
</xs:simpleType>
```

<xs:simpleType …>

Atributos principales:

Atributo	Descripción
name	Nombre del tipo. Obligatorio si el elemento padre es xs:schema y no permitido en caso contrario.
id	Identificador único para el componente

<xsd:restriction …>

Atributos principales:

Atributo	Descripción
base	Obligatorio Tipo de dato a partir del cual se construye el nuevo tipo. Puede ser predefinido o construido
id	Identificador único para el componente

Las **restricciones** (también llamadas facetas) que pueden aplicarse a los tipos simples son:

Restricción	Descripción
minInclusive	Mínimo valor que puede tomar el número, incluyendo el propio valor fijado. (≥) Por ejemplo, si mininclusive vale 5, el número tiene que ser mayor o igual que 5.

minExclusive	El número debe ser mayor que este valor, excluyendo al propio valor fijado. (>) Por ejemplo, si minExclusive vale 5, el número debe ser mayor que 5.	
maxInclusive	Máximo valor que puede tomar el número, incluyendo el propio valor fijado. (≤) Por ejemplo, si maxinclusive vale 5, el número tiene que ser menor o igual que 5.	
maxExclusive	El número debe ser menor que este valor, excluyendo al propio valor fijado. (<) Por ejemplo, si maxExclusive vale 5,el número debe ser menor que 5.	
totalDigits	Total de cifras en el número, incluyendo las enteras y las decimales. (xxx,xx)	
fractionDigits	Número de cifras decimales. (--,xx) Nota: mediante la combinación de totalDigits y fractionDigits se puede establecer el número de cifras de la parte entera. Por ejemplo totalDigits=”6” fractionDigits=”2” nos indica que en total habrá 6 cifras: 2 en la parte decimal y, por tanto, 4 en la parte entera.	
length	Número exacto de unidades del valor literal. (Longitud fija. Número exacto de caracteres o elementos de una lista permitidos) Para la mayoría de los tipos (por ejemplo, los tipos “string” y sus derivados, la faceta “length” se refiere a caracteres, pero para listas se refiere al número de elementos en la lista, y para valores binarios se refiere al número de octetos). No se puede aplicar a los tipos “integer”, “float” o “double” (para estos se puede utilizar “totalDigits”).	
minLength y maxLength	Valor mínimo y máximo respectivamente para la faceta “length”. (Número mínimo y máximo de caracteres o elementos permitidos).	
pattern	Formato que debe tener el valor, especificado mediante una expresión regular tradicional.	
enumeration	Conjunto de posibles valores que puede tomar el dato. (Lista de valores aceptables)	
whiteSpace	Controla la forma que tendrá el contenido de este dato una vez haya sido procesado. Puede tomar los siguientes valores:	
	“preserve”	Los datos no se modifican, quedan tal y como aparecen escritos.
	“replace”	Los tabuladores, saltos de línea y retornos de carro son sustituidos por espacios.
	“collapse”	Hace lo mismo que “replace”, pero además sustituye espacios múltiples por un solo espacio.

Estas restricciones se pueden combinar, y se pueden usar tanto en las definiciones de tipos con nombre como en las definiciones de elementos.

Se pueden diseñar datos construidos simples a partir de un tipo predefinido:

- Pueden usarse directamente en la definición de un elemento, en lugar de usar el atributo type.

  Por ejemplo:
  En lugar de especificar <xs:element name=”edad” type=”xs integer”/>, se suprime el uso del atributo type en la declaración del elemento, estableciéndose a través de la declaración de restricción.

```
<xs:element name="edad" minOccurs="1" maxOccurs="1">
    <xs:simpleType>
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="100"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

- Pueden definirse asignándoles un nombre y pudiéndose usar en cualquier elemento del documento mediante el atributo type. Esta sección es indiferente colocarla antes o después de la definición del elemento raíz.

  Por ejemplo: Primero se define el tipo, denominado “longitudMaxima” y, posteriormente, se utiliza en los elementos o atributos mediante la llamada a type=”longitudMaxima”.

```
<xs:simpleType name="longitudMaxima">
    <xs:restriction base="xs:string">
        <xs:minLength value="0"/>
        <xs:maxLength value="10"/>
    </xs:restriction>
</xs:simpleType>

<xs:element name="nombre" type="longitudMaxima"/>
```

▪ **union**: Uniones. Una unión es un tipo de datos creado a partir de una colección de tipos de dato de manera que un valor será válido si lo es para al menos uno de los datos base.

```
<xs:union memberTypes="tipo1 tipo2… tipoN"/>
```

Ejemplo: El elemento talla puede tomar cualquier valor de tipo tipoTallaNumero o tipoTallaTexto.

```
<xs:element name="talla">
   <xs:simpleType>
       <xs:union memberTypes="tipoTallaNumero tipoTallaTexto"/>
   </xs:simpleType>
</xs:element>
```

También sirve para determinar un tipo de datos más grande mediante la combinación de varios tipos.

Ejemplo: se combinan dos tipos (cadena y fecha) para indicar la fecha de pedido

```
<xsd:element name="orderDate">
<xsd:simpleType>
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="yesterday" />
        <xsd:enumeration value="today" />
        <xsd:enumeration value="tomorrow" />
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base="xsd:date" />
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
</xsd:element>
```

En el documento XML sería válido:

```
<orderDate>yesterday</orderDate>
<orderDate>2003-10-28 </orderDate>
```

▪ **Listas**: Una lista es un tipo de datos compuesto por varios valores de un tipo base separados por espacios.

```
<xs:list itemType="tipoBase">
```

Es similar a enumeration pero permite incluir valores múltiples, separados mediante espacios.
Las listas permiten elegir el tipo de datos de los valores en la lista, mediante el atributo "itemType", incluyendo tipos definidos por el usuario (los valores de la lista pueden ser datos primitivos o simples con restricciones).
También se pueden aplicar otras facetas, como "length", etc.

Ejemplo: Lista de números enteros.

```
<xs:simpleType name="tipoListaEnteros">
        <xs:list itemType="xs:integer"/>
</xs:simpleType>
```

Ejemplo: Lotería Primitiva.

```
<xs:simpleType name="tipoNumerosPrimitiva">
        <xs:list>
                <xs:simpleType>
                        <xs:restriction base="xs:positiveInteger">
                                <xs:maxInclusive value="49"/>
                        </xs:restriction>
                </xs:simpleType>
        </xs:list>
</xs:simpleType>

<xs:element name="combinacion" type ="tipoNumerosPrimitiva"/>
```

En el documento XML sería válido:
<combinacion>4 8 15 16 23 42</combinacion>

Ejemplo: se crea una lista para indicar las tallas.

```
<xsd:element name="sizes">
  <xsd:simpleType>
    <xsd:list itemType="xsd:string" />
  </xsd:simpleType>
</xsd:element>
```

(Se ha usado xsd:list, que permite crear una lista de ítems separados por un espacio en blanco)
En el documento XML sería válido
<sizes>XL XXL S XS M</sizes>

Ejemplo: se puede limitar el tamaño de la lista a cinco ítems (ni uno más ni uno menos)

```
<xsd:list itemType="xsd:string">
  <xsd:lenght value="5" />
</xsd:list>
```

Ejemplo: se puede indicar el tamaño de la lista delimitando que puede contener entre dos y siete ítems.

```
<xsd:list itemType="xsd:string">
        <xsd:minLenght value="2" />
        <xsd:maxLenght value="7" />
</xsd:list>
```

Ejemplo: Se define el tipo "tiendas" como una restricción del tipo "listaTiendas" donde "length" no puede ser mayor que 3. Es decir, en "tiendas" puede haber hasta 3 valores de la lista.
El tipo "listaTiendas" se define como una lista donde cada elemento de la lista es del tipo "posiblesTiendas". Este último se define como un "enumeration".

```
<xsd:simpleType name="posiblesTiendas">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="Tienda1"/>
                <xsd:enumeration value="Tienda2"/>
                <xsd:enumeration value="Tienda3"/>
                <xsd:enumeration value="Tienda4"/>
        </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="listaTiendas">
        <xsd:list itemType="posiblesTiendas"/>
</xsd:simpleType>

<xsd:simpleType name="tiendas">
        <xsd:restriction base="listaTiendas">
                <xsd:maxLength value="3"/>
        </xsd:restriction>
</xsd:simpleType>
```

Podríamos definir un elemento llamado "proveedores":

```
<xsd:element name="proveedores" type="tiendas"/>
```

En el documento XML (instancia del esquema) podríamos encontrar la siguiente información:

```
<proveedores>Tienda1</proveedores>
```

▪ **pattern**: patrón o modelo. Podemos establecer un patrón o expresión regular que deberá cumplir un valor.

Patrón	Significado	Patrón	Significado
.	Cualquier carácter	[A-Z a-z]	Letra
\w	Cualquier letra	[A-Z]	Letra mayúscula
\d	Cualquier dígito	[a-z]	Letra minúscula
\D	Cualquier carácter excepto un dígito decimal	[0-9]	Dígitos decimales
\s	Cualquier carácter de espaciado (tabulador, salto, …)	[C1-C2]	Cualquier carácter comprendido entre C1 y C2
\t	Tabulación	[C1-C2-[E]]	Exclusión de un carácter en un intervalo.
(A)	Cadena que coincide con A	[abcd]	Alguno de los caracteres que están entre corchetes. Lista de caracteres permitidos.
(A | B)	Alternativa entre dos expresiones. Cadena que es igual a la cadena A o a la B	[^abcd]	Cualquier carácter que no esté entre corchetes. Lista de caracteres no permitidos.
AB	Cadena que es la concatenación de las cadenas A y B	A{n}	n ocurrencias de la expresión
A?	0 o 1 ocurrencias de la expresión	A{n,m}	De n a m ocurrencias de la expresión
A*	0 o más ocurrencias de la expresión	A{n,}	n o más ocurrencias de la expresión
A+	1 o más ocurrencias de la expresión		

Ejemplo: se define un tipo simple para almacenar el código, que estará compuesto por 13 dígitos, cada uno de los cuales en valores de 0 a 9:

```
<xs:simpleType name="codigo">
    <xs:restriction base="xs:string">
        <xs:pattern value="[0-9]{13}"/>
    </xs:restriction>
</xs:simpleType>
```

Ejemplo: se define un tipo simple para almacenar el código, que estará compuesto por 3 caracteres cualesquiera (excepto dígito decimal), seguido de un guión, y seguido de 2 letras en mayúscula (xxx-XX):

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

Ejemplo: se define un tipo simple para almacenar el número de teléfono en formato xxx.xxx.xxx

```
<xs:simpleType name="tipoTelefono">
    <xs:restriction base="xs:string">
        <xs:pattern value="\d{3}\.\d{3}\.\d{3}"/>
    </xs:restriction>
</xs:simpleType>
```

▫ Ejemplos de datos simples construidos a partir de un tipo predefinido:

Ejemplo: minInclusive y maxInclusive. La edad debe estar comprendida [0-120]

```
<xs:element name="edad">
    <xs:simpleType>
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="120"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Ejemplo: minLength y maxLength. La contraseña debe tener entre 5 y 8 caracteres.

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

Ejemplo: crear un tipo enumerado para limitar los valores que puede tomar un elemento o un atributo

```
<xsd:attribute name="demanda">
    <xsd:simpleType>
        <xs:restriction base="xsd:string">
            <xs:enumeration value="bajo"/>
            <xs:enumeration value="medio"/>
            <xs:enumeration value="alto"/>
        </xs:restriction>
    </xs:simpleType>
</xsd:attribute>
```

Ejemplo: crear un tipo basado en token que solo permita como valor las cuatro estaciones.

```
<xs:simpleType name="tipoEstacion">
    <xs:restriction base="xs:token">
        <xs:enumeration value="Primavera"/>
        <xs:enumeration value="Verano"/>
        <xs:enumeration value="Otoño"/>
        <xs:enumeration value="Invierno"/>
    </xs:restriction>
</xs:simpleType>
<xs:element name="estacion" type="tipoEstacion"/>
```

En el documento XML sería válido:

```
<estacion>Verano</estacion>
```

Ejemplo: elemento "dirección" con la restricción de que los espacios en blanco, las tabulaciones, los saltos de línea y los retornos de carro que aparezcan en él se deben mantener.

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

Ejemplo: se define un elemento llamado "clave" con la restricción de que su valor tiene que ser una cadena de, exactamente, doce caracteres.

```
<xs:element name="clave">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:length value="12"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

B. Tipos de Datos Complejos (xs:complexType):

```
<xsd:complexType name="…">
        …
</xsd:complexType>
```

<xsd:complexType>

Atributos principales:

Atributo	Descripción
name	Nombre del tipo. Obligatorio si el elemento padre es xs:schema y no permitido en caso contrario.
id	Identificador único para el componente.
mixed(false, true)	Indica si el elemento combina descendientes con contenido textual. Por defecto: false.
abstract(false, true)	Indica si el elemento puede aparecer en una instancia. Con valor true indica que no puede aparecer en una instancia. Por defecto: false.
final(extension, restriction, #all)	Indica si el elemento se puede derivar de alguna manera: por extensión, por restricción o por ambas.

Tipos de Datos Complejos:

a) Elementos dentro de otros elementos. (<xsd:sequence>, <xsd:choice>, <xsd:all>)
b) Elementos que solo tienen atributos. (<xsd:extension>)
c) Elementos mixtos que tienen atributos y otros elementos. (ambas)
d) Elementos vacíos.
e) Extensión de Tipos de Datos complejos.

Nota:

- La etiqueta <simpleContent> permite definir extensiones o restricciones para tipos simples y tipos complejos sin descendientes.
- La etiqueta <complexContent> permite definir extensiones o restricciones para tipos complejos con descendientes.
- La etiqueta <extension> permite extender un tipo simple (para que tenga atributos) o complejo (añadir elementos, atributos, etc.) existente.
- La etiqueta <sequence> permite indicar los descendientes de un elemento.

a) Elementos dentro de otros Elementos. (solo elementos)

```
<xsd:complexType name="…">
        <xsd:sequence>
                <xsd:element …/>
                …
        </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="…">
        <xsd:choice>
                <xsd:element …/>
                …
        </xsd:choice>
</xsd:complexType>
```

La forma más sencilla de crear un nuevo tipo es crear un elemento complexType al que se le asigna un nombre mediante el atributo "name".

▪ **Secuencia**: Se utiliza la etiqueta <xsd:sequence> para detallar los elementos que contendrán dicho tipo, en un orden determinado.
La secuencia puede aparecer un número variable de veces configurable mediante minOccurs y maxOccurs, que por defecto valen 1.

Ejemplo:

```
<xs:complexType>
        <xs:sequence>
                <xs:element name="emisor" type="xs:string" />
                <xs:element name="receptor" type="xs:string" />
                <xs:element name="mensaje" type="xs:string" />
        </xs:sequence>
</xs:complexType>
```

Ejemplo:

```
<xsd:complexType name="precioInfo">
        <xsd:sequence>
                <xsd:element ref="precioTipo"/>
                <xsd:element ref="precioN"/>
        </xsd:sequence>
</xsd:complexType>

<xsd:element name="precioTipo" type="xsd:string"/>
<xsd:element name="precioN" type="xsd:decimal"/>
```

Con esto creamos un tipo nuevo llamado "precioInfo" que está formado por una secuencia de dos elementos, "precioTipo" y "precioN", a los que referencia.

(Dentro de la definición de tipo de datos complejo "precioInfo" podríamos haber declarado los elementos que lo componen sin

referenciar a otro lugar del documento, indicando directamente el nombre el elemento y su tipo de dato. <xsd:element name=”precioTipo” type=”xsd:string”/>)

Podremos utilizar este tipo que hemos creado para definir elementos, por ejemplo:

```
<xsd:element name=”precio” type=”precioInfo”/>
```

(Observamos que en la definición del elemento precio, el nombre del tipo “precioInfo” no lleva el prefijo “xsd”, porque no pertenece al espacio de nombres del estándar XML Schema).

La principal ventaja de definir tipos de datos propios es, por un lado, que estos tipos se pueden utilizar donde se quiera y, además, que estos tipos se pueden utilizar como tipos base para definir otros tipos.

▪ **Alternativa**: Se utiliza la etiqueta <xsd:choice> para detallar los elementos que aparecerán como alternativas, entre las que se elige una.
Puede aparecer un número variable de veces, fijado por minOccurs y maxOccurs.

Ejemplo:

```
<xs:complexType>
    <xs:choice>
       <xs:element name="prologo" type="xs:string" />
       <xs:element name="prefacio" type="xs:string" />
       <xs:element name="introduccion" type="xs:string" />
    </xs:choice>
</xs:complexType>
```

▪ **Todos**: Se utiliza la etiqueta <xsd:all> para detallar los elementos que contendrán dicho tipo, en cualquier orden.
Puede aparecer un número variable de veces, fijado por minOccurs y maxOccurs.
No se recomienda su uso por la pérdida de control.

Ejemplo:

```
<xs:complexType>
    <xs:all>
        <xs:element name="alfa" type="xs:string" />
        <xs:element name="beta" type="xs:string" />
        <xs:element name="omega" type="xs:string" />
    </xs:all>
</xs:complexType>
```

- Elementos con descendientes y contenido:

```
<xs:complexType mixed="true">
   <xs:sequence>
      <xs:element …/>
      …
   </xs:sequence>
</xs:complexType>
```

Donde:

mixed=”true”. Indica contenido mixto (textual y descendientes).

Ejemplo: se define el elemento “p”, que contendrá contenido mixto, y en el que pueden aparecer un número indefinido de elementos <b> y de elementos <i> en cualquier orden.

```
<xs:element name="p">
   <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element name="b" type="xs:string"/>
         <xs:element name="i" type="xs:string"/>
      </xs:choice>
   </xs:complexType>
</xs:element>
```

b) Elementos que tienen Atributos, pero NO contengan otros Elementos.

1. Añadir atributos a tipos simples por extensión. (Elementos con contenido textual y atributos)

```
<xsd:complexType name=”…”>
   <xsd:simpleContent>
      <xsd:extension base=”xs:tipoOrigen”/>
         <xsd:attribute name=”…” …/>
         …
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>
```

La declaración de un tipo de dato complejo, que posee atributo/s pero no contiene elementos en su interior se realiza mediante la utilización de los elementos <xsd:simpleContent> (no contiene descendientes) y <xsd:extension> (se extiende el tipo base para agregarle atributos).

(Recordar: tipos de datos simples no pueden contener ni elementos hijo ni atributos. Tipos de datos complejos pueden tener tanto atributos como elementos hijo).

Ejemplo:

```
<xsd:element name="nombreOriginal">
   <xsd:complexType>
```

```
                <xsd:simpleContent>
                    <xsd:extension base="xsd:string">
                        <xsd:attribute name="confirmado"
                        default="no"/>
                    </xsd:extension>
                </xsd:simpleContent>
            </xsd:complexType>
        </xsd:element>
```

En el documento XML sería válido:
<nombreOriginal confirmado="si">Denominación</nombreOriginal>
El elemento es de tipo cadena (tipo de dato primitivo), con un atributo (que tomará el valor por defecto "no") y sin descendientes.

Ejemplo: Al elemento "textarea", de tipo cadena, se le agregan 3 atributos

```
<xs:element name="textarea">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="name" type="xs:string"/>
                <xs:attribute name="cols" type="xs:positiveInteger"/>
                <xs:attribute name="rows" type="xs:positiveInteger"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
```

2. Añadir atributos a un tipo complejo.

```
<xsd:complexType name="…">
        <xsd:attribute name="…" …/>
        <xsd:attribute name="…" …/>
        …
</xsd:complexType>
```

Ejemplo: declaración del elemento img con dos atributos tipo cadena

```
<xs:element name="img">
    <xs:complexType>
        <xs:attribute name="src" type="xs:string"/>
        <xs:attribute name="alt" type="xs:string"/>
    </xs:complexType>
</xs:element>
```

En el documento XML sería válido:

```
<img src="imagen.jpg" alt="Imagen" />
```

c) Elementos que tienen atributos y otros elementos.

```
<xsd:complexType name="…">
        <xsd:sequence>
                <xsd:element …/>
                …
        </xsd:sequence>
        <xsd:attribute name="…" …/>
        …
</xsd:complexType>
```

Se realiza mediante combinación de los dos anteriores:

- Utiliza la etiqueta <xsd:sequence> para detallar los elementos que contendrán dicho tipo.
- Al estar declarado dentro de un tipo complejo no se requiere la utilización de la etiqueta <xsd:extension> para detallar los atributos de dicho tipo. Basta con su declaración.

▫ Se pueden construir tipos de datos extendiendo otros tipos de datos ya existentes.

Por ejemplo: se define el tipo "infoPersona" que contendrá elementos y atributos.

```
<xs:complexType name="infoPersona">
        <xs:sequence>
                <xs:element name="nombre" type="xs:string"/>
                <xs:element name="edad" type="xs:integer"/>
        </xs:sequence>
        <xs:attribute name="numero" type="xs:integer"/>
</xs:complexType>
```

Posteriormente se declara el tipo "infoPersonaAmpliada" en el que se reutiliza el tipo "inforPersona" extendiéndolo para agregarle elementos descendientes.

```
<xs:complexType name="infoPersonaAmpliada">
        <xs:complexContent>
                <xs:extension base="infoPersona">
                        <xs:sequence>
                                <xs:element name="ciudad"
                                type="xs:string"/>
                                <xs:element name="pais"
                                type="xs:string"/>
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>

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

- Elementos con descendientes, atributos y contenido:

```
<xs:complexType mixed="true">
   <xs:sequence>
         <xs:element …/>
         …
   </xs:sequence>
   <xsd:attribute name=”…” …/>
   …
</xs:complexType>
```

Donde:

mixed=”true”. Indica contenido mixto (textual y descendientes).

Ejemplo: Se define el elemento “form” que contendrá contenido textual, un descendiente (se referencia a su declaración posterior) y dos atributos.

```
<xs:element name="form">
      <xs:complexType mixed="true">
         <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="input" />
         </xs:sequence>
         <xs:attribute name="name" type="xs:string"/>
         <xs:attribute name="method" type="xs:string"/>
      </xs:complexType>
</xs:element>

<xs:element name="input">
      ...
</xs:element>
```

d) Elementos vacíos.

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

La declaración del elemento vacío hace uso de la etiqueta <complexType/>, con indicación de cierre. No contiene descendientes ni atributos.

Ejemplo:

```
<xs:element name="br">
      <xs:complexType />
</xs:element>
```

En el documento XML sería válido

```
<br/>
```

e) Extensión de Tipos de Datos Complejos.

La extensión de tipos complejos se asemeja a la herencia de la POO, ya que permite añadir elementos y atributos a tipos complejos existentes.

```
<xs:complexType name="tipoExtendido">
    <xs:complexContent>
        <xs:extension base="tipoComplejoInicial">
            <!-- Nuevos elementos y/o atributos -->
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Para extender un tipo complejo se crea un nuevo tipo complejo, se especifica su tipo de contenido (simpleContent o complexContent), y se establece la extensión fijando como base un tipo complejo creado previamente y añadiendo elementos y/o atributos.

Ejemplo:

```
<xs:complexType name="tipoAlumno">
    <xs:sequence>
        <xs:element name="Nombre" type="xs:string"/>
        <xs:element name="Apellido" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="tipoAlumnoExtendido">
    <xs:complexContent>
        <xs:extension base="tipoAlumno">
            <xs:sequence>
                <xs:element name="Ciclo" type="xs:string"/>
            </xs:sequence>
            <xs:attribute name="matricula" type="xs:positiveInteger"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Resumen de tipos de datos

Tipo	Contenido	Descendientes	Atributos	Contenido
Simple	Simple			✓
Complejo	Simple		✓	✓
Complejo	Complejo			
Complejo	Complejo		✓	
Complejo	Complejo	✓		
Complejo	Complejo	✓	✓	
Complejo	Complejo Mixto	✓		✓
Complejo	Complejo Mixto	✓	✓	✓

▪ Declaración de Tipos Anónimos o Tipos con Nombre:

La declaración de tipos puede realizarse:

- Tipos Anónimos: facilitan la lectura del documento.
- Tipos con Nombre: permiten la reutilización del código.
- Referencias: se hace referencia al "name" de otro elemento.

Por ejemplo:

- Tipo Anónimo:
  En este caso seguir la estructura del elemento "alumno" es sencillo, dado que se especifica su composición de forma continuada.

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

- Tipo con nombre:
  En este caso se ha declarado el elemento "alumno" indicando que su estructura pertenece al "TipoPersona".
  Posteriormente se ha declarado "TipoPersona". De esta forma varios elementos pueden llevar asociado "TipoPersona", que en el caso anterior no era posible (se facilita la reutilización).

```
<xs:element name="alumno" type="TipoPersona"/>
...
<xs:ComplexType name="TipoPersona">
        <xs:sequence>
                <xs:element name="nombre" type="xs:string"/>
                <xs:element name="apellidos" type="xs:string"/>
        </xs:sequence>
</xs:ComplexType>
```

- Referencias:
  En este caso se ha declarado el elemento "alumno" y posteriormente, en la declaración del elemento "alumnos" se hace referencia al elemento anteriormente declarado.

```
<xs:element name="alumno">
        <xs:sequence>
                <xs:element name="nombre" type="xs:string"/>
                <xs:element name="apellidos" type="xs:string"/>
        </xs:sequence>
</xs:element>
...
<xs:element name="alumnos">
        <xs:sequence>
                <xs:element ref="alumno" />
        </xs:sequence>
</xs:element>
```

5. Uso de Esquemas Externos.

A. Añadir definiciones de esquema externo (mismo espacio nombres):

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:include schemaLocation="EsquemaIncluir.xsd"/>
    …
</xs:schema>
```

xs:include: permite añadir las declaraciones y definiciones de un esquema externo al esquema actual permitiéndonos reutilizar tipos ya definidos.
El esquema externo debe tener el mismo espacio de nombres que el esquema actual.

Elemento padre: xs:schema.
Atributos principales:
- xs:schemaLocation: Obligatorio, URI del esquema a incluir.

Ejemplo:

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

B. Importar definiciones de esquema externo (distinto espacio nombres):

```
<xs:schema xmlns:xs=”URI_EspacioNombres_Actual"
   xmlns:pre="EsquemaOrigen"
   targetNamespace="URI_EspacioNombres_Actual">
   <xs:import namespace="URI_EspacioNombres_Importar"/>
   …
   <xs:element ref="pre:NombreElementoImportar">
   …
</xs:schema>
```

xs:import: permite importar un esquema externo.
Funciona igual que xs:include con la diferencia de que el esquema importado tiene diferente espacio de nombres que el esquema actual.

Elemento padre: xs:schema.
Atributos principales:
- namespace: URI del espacio de nombres a importar.
- schemaLocation: URI del esquema del espacio de nombres importado.

Ejemplo:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:xhtml="http://www.w3.org/1999/xhtml"
   targetNamespace="http://www.w3.org/2001/XMLSchema">
```

```
        <xs:import namespace="http://www.w3.org/1999/xhtml"/>
        ...
        <xs:element ref="xhtml:p">
        ...
</xs:schema>
```

C. Redefinir definiciones de esquema externo:

```
<xs:schema xmlns:xs="…">
        <xs:redefine schemaLocation="NombreEsquemaExterno.xsd">
                <!-- Redefinición del tipo o grupo -->
        </xs:redefine>
</xs:schema>
```

xs:redefine: permite redefinir un tipo de datos (simple o complejo) o un grupo (de elementos o de atributos), que ya hubieran sido definidos en otro documento de esquema al que se referencia.

Elemento padre: xs:schema.
Atributos principales:
- xs:schemaLocation: Obligatorio, URI del esquema referenciado.

Ejemplo:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:redefine schemaLocation="alumnos.xsd">
                <!-- Redefinición del tipo o grupo -->
        </xs:redefine>
</xs:schema>
```

6. Control de Integridad Referencial.

La integridad referencial es una propiedad de las bases de datos que garantiza que los registros relacionados con otros tengan información válida:

1. Al crear un registro que contiene información de otro (clave ajena) se comprobará que ese otro registro existe.
2. Al intentar eliminar un registro que esté referenciado en otros, o bien se evita la eliminación o se eliminan también los registros asociados.

Ejemplo:
Clientes (id_cliente, nombre, apellidos, …)
Facturas (num_factura, cliente, importe, …)
No se puede crear una factura para un cliente inexistente, ni eliminar un cliente con facturas pendientes.

Para lograr la integridad referencial se utilizan las instrucciones xs:key, xs:keyref y xs:unique.
Todas ellas requieren de xs: selector y xs:field.

▪ xs:key

xs:key::= xs:selector (xs:field)+

xs:key: permite definir un elemento/atributo como clave que identifica de forma única al elemento/atributo y por tanto su valor no puede ser nulo ni repetirse.
Se utiliza de forma conjunta a xs:keyref que permite a otro elemento/atributo referenciar a la clave.

Elemento padre: xs:element.
Atributos obligatorios:
- name: nombre de la clave.

Atributos opcionales: id.

▪ xs:unique

xs:unique::= xs:selector (xs:field)+

xs:unique: permite definir un elemento/atributo de valor único (equivalente a clave única).
Se utiliza de forma conjunta a xs:keyref que permite a otro elemento/atributo referenciar a la clave única.

Elemento padre: xs:element.
Atributos obligatorios:
- name: nombre de la clave única.

Atributos opcionales: id.

▪ xs:keyref

xs:keyref::= xs:selector (xs:field)+

xs:keyref: permite referenciar a claves primarias (xs:key) o claves únicas (xs:unique) compuestas por elementos y/o atributos (equivalente a clave ajena).
Se utiliza de forma conjunta a xs:key o xs:unique.

Elemento padre: xs:element.
Atributos obligatorios:
- name: nombre de la referencia de la clave.
- refer: nombre de la clave primaria o única a la que referencia.

Atributos opcionales: id.

▪ xs:selector

xs:selector::= (annotation)?

xs:selector: especifica una expresión Xpath que selecciona un conjunto de elementos para usarlos en la definición de una clave: primaria (xs:key), única (xs:unique) o ajena (xs:keyref).

Elemento padre: xs:key, xs:unique, xs:keyref.
Atributos obligatorios:
- xpath: identifica los elementos hijos a los cuales se aplica la restricción de identidad.

Atributos opcionales: id.

▪ xs:field

xs:field::= (annotation)?

xs:field: especifica una expresión Xpath que indica los elementos/atributos que formarán parte de la clave que se esté definiendo: primaria (xs:key), única (xs:unique) o ajena (xs:keyref).

Elemento padre: xs:key, xs:unique, xs:keyref.
Atributos obligatorios:
- xpath: identifica un único elemento o atributo cuyo contenido o valor se utiliza para la restricción.

Atributos opcionales: id.

Ejemplo:
Instancia XML

```
<?xml version="1.0"?>
<pedido xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="pedido.xsd">
    <codigo>12345</codigo>
    <productos>
            <producto>
            <id_producto>001</id_producto>
            <nombre>Pantalón</nombre>
            <precio moneda="euro">49.95</precio>
            </producto>
            ...
    </productos>
    <lineas>
            <linea producto="001">
            <cantidad>2</cantidad>
            <color>Azul</color>
            </linea>
            ...
    </lineas>
</pedido>
```

Documento XML Schema

```
<xs:element name="pedido" type="tipoPedido">

    <!-- Definición de un elemento clave -->
    <xs:key name="prodNumKey">
            <xs:selector xpath="productos/producto"/>
            <xs:field xpath="id_producto"/>
    </xs:key>

    <!-- Definición de una referencia a un elemento clave -->
    <xs:keyref name="prodNumKeyRef" refer="prodNumKey">
```

```
            <xs:selector xpath="lineas/linea"/>
            <xs:field xpath="@producto"/>
        </xs:keyref>

    </xs:element>
```

7. Modelos de Diseño de Esquemas XML.

A. Diseño Anidado o de muñecas rusas:

Cada elemento y atributo se describe en el mismo lugar en que se declaran de manera que todos los componentes se anidan unos dentro de otros.
Este modelo produce duplicidades en la descripción de elementos del mismo tipo y puede provocar que haya elementos con el mismo nombre y distinta descripción.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="libro">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="titulo" type="xs:string"/>
                <xs:element name="autor" type="xs:string"/>
                <xs:element name= "personaje" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="nombre" type="xs:string"/>
                            <xs:element name="amigoDe" type="xs:string"
                                minOccurs="0" maxOccurs="unbounded"/>
                            <xs:element name="desde" type="xs:date"/>
                            <xs:element name="calificacion" type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="isbn" type="xs:string"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

B. Diseño Plano:

Los elementos y atributos se declaran y se indica una referencia a su definición, que se realiza en otro lugar del documento.
Este modelo es el que más se parece a los DTD.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <!-- definición de elementos de tipo simple-->
    <xs:element name="titulo" type="xs:string"/>
    <xs:element name="autor" type="xs:string"/>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="amigoDe" type="xs:string"/>
    <xs:element name="desde" type= "xs:date"/>
    <xs:element name="calificacion" type="xs:string"/>

    <!-- definición de atributos -->
    <xs:attribute name="isbn" type="xs:string"/>

    <!-- definición de elementos de tipo complejo -->
    <xs:element name="personaje">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="nombre"/>
                <xs:element ref="amigoDe" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="desde"/>
                <xs:element ref="calificacion"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    ...
```

C. Diseño de Tipos con Nombres Reutilizables:

Se definen tipos de datos simples o complejos a los que se identifica con un nombre y al declarar elementos y atributos se indica que son de alguno de esos tipos definidos previamente.

```
.!-- definición de tipos simples-->
    <xs:simpleType name="TipoNombre" >
        <xs:restriction base="xs:string">
        <xs:raaxLength value="32"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="TipoDesde">
        <xs:restriction base="xs:date"/>
    </xs:simpleType>
    <xs:simpleType name= "TipoDescripcion" >
        <xs:restriction base="xs:string"/>
    </xs:simpleType >
    <xs:simpleType name="TipoISBN">
        <xs:restriction base="xs:string">
        <xs:pattern value="[0-9]{13}"/>
        </xs:restriction>
    </xs:simpleType>
    <!-- definición of tipos complejos -->
    <xs:complexType name="TipoPersonaje">
        <xs:sequence>
            <xs:element name="nombre" type="TipoNombre"/>
            <xs:element name="amigoDe" type="TipoNombre" minOccurs = "0"
                        maxOccurs="unbounded"/>
            <xs:element name="desde" type="TipoDesde"/>
            <xs:element name="calificacion" type="TipoDescripcion"/>
        </xs:sequence>
    </xs:complexType>
```

Ejemplo modelos de diseño:

Vamos a ver los diferentes modelos de diseño de los esquemas XML a través de un ejemplo, en el que se pretende almacenar la información de una biblioteca con las siguientes características:

1) La biblioteca estará compuesta por varios libros (al menos uno).
2) Para cada libro se almacenará la siguiente información: ISBN, título, autores y editorial.
3) El ISBN estará formado por 13 dígitos con el siguiente formato: XXX-XX-XXXX-XXX-X.
4) Cada libro tendrá como mínimo un autor y como máximo 4.
5) La editorial puede ser una de las siguientes: Anaya, Garceta, McGraw-Hill o Ra-ma.

▫ *Documento XML.*

```
<?xml version="1.0" encoding="UTF-8"?>
<biblioteca xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="biblioteca.xsd">
    <libro>
        <isbn>978-84-1545-217-1</isbn>
        <titulo>Lenguajes de marcas</titulo>
        <autor>Juan Manuel Castro Ramos</autor>
        <autor>Jose Ramón Rodríguez Sánchez</autor>
        <editorial>Garceta</editorial>
    </libro>
    <libro>
        <isbn>978-84-9964-099-0</isbn>
        <titulo>Sistemas informáticos</titulo>
        <autor>Jose Luis Raya Cabrera</autor>
        <autor>Laura Raya González</autor>
        <autor>Francisco Javier Sánchez Zurdo</autor>
        <editorial>Ra-ma</editorial>
    </libro>
    <libro>
        <isbn>978-84-4151-812-4</isbn>
        <titulo>XML</titulo>
        <autor>Elliotte Rusty Harold</autor>
        <autor>W. Scott Means</autor>
        <editorial>Anaya</editorial>
    </libro>
</biblioteca>
```

▫ *Esquema siguiendo el modelo de diseño Anidado.*

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="biblioteca">
            <xs:complexType>
               <xs:sequence>
                    <xs:element name="libro" maxOccurs="unbounded">
                       <xs:complexType>
                            <xs:sequence>
                                    <xs:element name="isbn">
                                            <xs:simpleType>
                                                    <xs:restriction base="xs:token">
                                                            <xs:pattern
                                                            value="\d{3}\-\d{2}\-\d{4}\-
                                                            \d{3}\-\d{1}"/>
                                                    </xs:restriction>
                                            </xs:simpleType>
                                    </xs:element>
                                    <xs:element name="titulo" type="xs:token"/>
                                    <xs:element name="autor" type="xs:token"
                                             maxOccurs="4"/>
                                    <xs:element name="editorial">
                                            <xs:simpleType>
                                                    <xs:restriction base="xs:token">
                                                            <xs:enumeration
                                                               value="Anaya"/>
                                                            <xs:enumeration
                                                               value="Garceta"/>
                                                            <xs:enumeration
                                                               value="McGraw-Hill"/>
                                                            <xs:enumeration
                                                               value="Ra-ma"/>
                                                    </xs:restriction>
                                            </xs:simpleType>
                                    </xs:element>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
    </xs:element>
</xs:schema>
```

▫ *Esquema siguiendo el modelo de diseño Plano.*

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

    <xs:element name="biblioteca">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="libro" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="libro">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="isbn"/>
                <xs:element name="titulo" type="xs:token"/>
                <xs:element name="autor" type="xs:token"
                    maxOccurs="4"/>
                <xs:element ref="editorial"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="isbn">
        <xs:simpleType>
            <xs:restriction base="xs:token">
                <xs:pattern value="\d{3}\-\d{2}\-\d{4}\-\d{3}\-\d{1}"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>

    <xs:element name="editorial">
        <xs:simpleType>
            <xs:restriction base="xs:token">
                <xs:enumeration value="Anaya"/>
                <xs:enumeration value="Garceta"/>
                <xs:enumeration value="McGraw-Hill"/>
                <xs:enumeration value="Ra-ma"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>

</xs:schema>
```

▫ *Esquema siguiendo el modelo de diseño de Tipos con Nombres Reutilizables.*

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

    <xs:element name="biblioteca">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="libro" type="tipoLibro"
                    maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="tipoLibro">
        <xs:sequence>
            <xs:element name="isbn" type="tipoISBN"/>
            <xs:element name="titulo" type="xs:token"/>
            <xs:element name="autor" type="xs:token" maxOccurs="4"/>
            <xs:element name="editorial" type="tipoEditorial"/>
        </xs:sequence>
    </xs:complexType>

    <xs:simpleType name="tipoISBN">
        <xs:restriction base="xs:token">
            <xs:pattern value="\d{3}\-\d{2}\-\d{4}\-\d{3}\-\d{1}"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="tipoEditorial">
        <xs:restriction base="xs:token">
            <xs:enumeration value="Anaya"/>
            <xs:enumeration value="Garceta"/>
            <xs:enumeration value="McGraw-Hill"/>
            <xs:enumeration value="Ra-ma"/>
        </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

8. Pasos o métodos para crear un Esquema XML.

Pasos para crear un Esquema XML mediante un editor XML:

1. Declaración XML

```
<?xml version="1.0" encoding="UTF-8"?>
```

2. Declaración del elemento esquema (elemento raíz del esquema):

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

   El esquema debe incluir su elemento de esquema como elemento raíz. Este elemento también puede contener atributos.
   En el ejemplo, el formato xmlns:xs indica que todos los elementos o atributos que lleven el prefijo "xs:" pertenecen al espacio de nombres especificado en la URI (http://www.w3.org/2001/XMLSchema). Los prefijos se utilizan para distinguir entre diferentes espacios de nombres. Se puede utilizar cualquier prefijo, siempre que se especifique el espacio de nombres XML asociado

3. **Crear los elementos** del esquema XML, comenzando por el elemento raíz del documento XML.

```
<xs:element ….>
```

   - Los elementos deben incluir una etiqueta de inicio y de finalización. También puede incluir otros elementos, texto, atributos o una combinación de cualquiera de ellos.
   - Los nombres de los elementos XML no deben comenzar con un número o carácter especial, y no puede comenzar con las letras "xml".
   - Todos los elementos deben estar correctamente anidados.
   - Utilizar nombres cortos y descriptivos para los elementos.

4. **Definir** cuáles son los **elementos hijo** dentro del esquema XML.

```
<xs:element ….>
        <xs:complexType>
            <xs:sequence>
                    <xs:element …. />
                    <xs:element …. />
                    …
            </xs:sequence>
        </xs:complexType>
</xs:element>
```

   El esquema XML resume los elementos y atributos que están permitidos en el documento XML. Además, identifica elementos hijo, así como también su número y su orden.

   Se puede suplementar el esquema XML con otros esquemas válidos utilizando las sentencias import, include o redefine elements.

5. **Definir** los **atributos**:

   <xs:attribute name=”…” type=”xs:tipodato” use=”….”/>

   - Los atributos proporcionan información adicional acerca de los elementos que están contenidos dentro del documento XML.
   - Los atributos deben aparecer entre comillas.
   - Los atributos pueden contener solamente un valor.
   - No incluir estructuras de árbol entre los atributos.

6. **Crear** los **tipos** del esquema **para definir el contenido de los elementos y atributos**:

   <xs:complexType>

   El esquema XML permite describir qué contenido está permitido, trabajar con datos, definir facetas y patrones de datos, convertir datos y poder validarlos (si se permite que los elementos estén vacíos o que puedan incluir texto, así como también tipos de datos y valores fijos de elementos y atributos).

7. **Comprobar** que no existen errores en el esquema.

8. **Validar** el esquema XML utilizando la herramienta de validación del editor XML.

9. Transformación de DTD a XML Schema.

En la siguiente tabla se puede ver la correlación entre elementos de XML Schema y de DTDs, aunque existen herramientas de traducción entre estos 2 lenguajes.

Ejemplos de herramientas de traducción:
- DTD2HTML en Perl (http://www.w3.org/2000/04/schema_hack/)
- XMLSpy (http://www.xmlspy.com).

DTD	XML Schema	Comentarios
ELEMENT	<element>	Crea un vínculo entre un nombre y unos atributos, modelos de contenido y anotaciones
#PCDATA	Soportado como parte de un tipo simple	
ANY	<any>	Posee distintos comodines para un mayor conjunto de posibilidades. Existe también <anyAttribute> con comodines similares.
EMPTY	Soportado	Se elimina la existencia de elementos descendientes del actual, diferenciando de la presencia de un *string* vacío en un elemento.
Modelo de Contenido	<complexType>	
, (Conector de secuencia)	<sequence>	
| (Conector de alternativas)	<disjunction>	
? (Opcional)	Soportado	Se han de emplear los atributos predefinidos de maxOccurs y minOccurs

+(Requerido y Repetible)	Soportado	Se han de emplear los atributos predefinidos de maxOccurs y minOccurs
*(Opcional y Repetible)	Soportado	Se han de emplear los atributos predefinidos de maxOccurs y minOccurs
ATTLIST	<attribute>	Declara un atributo
	<attributeGroup>	Se pueden agrupar declaraciones de <attributes>
Tipo de atributo CDATA, ID, IDREF, NOTATION...	Tipos <simpleType>predefinidos	
ENTITY	NO Soportado	Las entidades son declaradas en declaraciones de marcas en el XML
ENTITY%Parameter	NO Soportada	Las entidades paramétricas ofrecen un mecanismo de bajo nivel que permite distintas cosas, algunas de estas se han intentado cubrir en XML Schema: • .. La separación entre <element> y <complexType> • .. Grupos de atributos • .. Grupos de modelos con nombre • .. Mecanismos de extensión y restricción de tipos • .. Los mecanismos de <import> e <include> para componer esquemas • .. El mecanismo de redefinición de elementos
NOTATION	<notation>	Notaciones
Secciones condicionales: <![IGNORE [...]]> <![INCLUDE [...]]>	Uso de minOccurs=”0” u otro valor	Las secciones condicionales DTD se transformarán en ocurrencias de elementos o atributos.

10. Generación automática de Esquemas XSD.

Existen numerosas herramientas que nos permiten generar un esquema a partir de un documento XML o a partir de un DTD, pero igual que ocurría con estos hay información que la herramienta es imposible que conozca por lo que será necesario depurarlos a mano.

La mayoría de herramientas especializadas en lenguajes XML poseen alguna utilidad para generar de manera automática un esquema XSD a partir de un documento XML bien formado.

Por ejemplo, en XML Copy Editor, podemos encontrar esta opción en el menú XML -> Create Schema...

11. Validación de Esquemas.

Algunos de los navegadores actuales (por ejemplo, Microsoft Internet Explorer) permiten validar un documento XML contra su DTD, pero ninguno permite validarlo contra su XML Schema.

La mayoría de herramientas especializadas en lenguajes XML poseen la funcionalidad de validar documentos XML mediante su esquema XSD asociado.

Todas las herramientas que permiten verificar si un documento XML está bien formado permiten validar un documento frente a un esquema XML.

Ejemplo de validador online:
https://www.corefiling.com/opensource/schemaValidate.html

Herramientas para el diseño y verificación de esquemas XSD.

Para diseñar esquemas XSD podemos utilizar el software libre XML Copy Editor.
Otras herramientas:

- Para GNU/Linux:
  - Getox
  - Emacs
  - XML mind
- Para Windows:
  - XML Spy
  - XMetaL

Otros mecanismos de validación

Otros lenguajes de esquemas XML:

- Relax-NG
  https://www.oasis-open.org/committees/relax-ng/

- Shematron
  www.xml.com/pub/a/2003/11/12/schematron.html

12. Ejemplos de Esquemas XML.

Ejemplo: *Declaración elemento "EDAD" de tipo entero*

```
<xs:element name="EDAD" type="xs:integer"/>
```

Ejemplo: *Declaración elemento "APELLIDO2_TRABAJADOR" con anotaciones y restricciones sobre un tipo de datos String.*

```
<xs:element name="APELLIDO2_TRABAJADOR">
    <xs:annotation>
        <xs:documentation>No Obligatorio para el caso de trabajadores
            extranjeros: Formato alfanumérico de 20 posiciones.
        </xs:documentation>
    </xs:annotation>
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:minLength value="0"/>
            <xs:maxLength value="20"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Ejemplo: *Declaración elemento "CONTRATO" con restricciones sobre un tipo de datos String para el almacenamiento de un único carácter*

```
<xs:element name="CONTRATO">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:length value="1"/>
            <xs:pattern value="[SsNn]"/>
            <xs:enumeration value="S"/>
            <xs:enumeration value="N"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Ejemplo: *Declaración elemento "AÑO_MES_ENVIO" con anotaciones y restricciones sobre un tipo de datos String para el almacenamiento de fecha en formado añomes*

```
<xs:element name="AÑO_MES_ENVIO">
    <xs:annotation>
        <xs:documentation>Obligatorio En formato YYYYMM será el año y mes
            del envío. Debe ser anterior al mes de recepción del fichero y se
            rellena con el mes 99 en el caso de envíos anuales
        </xs:documentation>
    </xs:annotation>
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:length value="6"/>
            <xs:pattern value="20(1[0-9]|2[0-9])(0[1-9]|1[012])"/>
            <xs:pattern value="20(1[0-9]|2[0-9])99"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Ejemplo: *Declaración de elemento "DATOS_AGREGADOS" con datos complejos (los elementos que componen el tipo complejo se habrán definido en otro lugar el documento)*

```
<xs:element name="DATOS_AGREGADOS">
    <xs:annotation>
        <xs:documentation>Elemento de grupo para datos con los totales
            agregados mensuales y anuales
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="TOTAL_PERSONAS"/>
            <xs:element ref="TOTAL_NUEVAS_REGISTRADAS"/>
            <xs:element ref="TOTAL_PERSONAS_PERCEPTORES"/>
            <xs:element ref="TOTAL_PERSONAS_INSERCION"/>
            <xs:element ref="TOTAL_OFERTAS"/>
            <xs:element ref="ERRORES" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Ejemplo: *Declaración del elemento "ENVIO_MENSUAL" de tipo compuesto con referencias en la declaración a otros lugares del documento en el que se definieron los elementos que componen el tipo de dato complejo (enlaza con los dos ejemplos anteriores)*

```
<xs:element name="ENVIO_MENSUAL">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="AÑO_MES_ENVIO"/>
                        <xs:element ref="DATOS_AGREGADOS"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

Ejemplo: *Declaración del elemento "NIVEL_FORMATIVO" con anotaciones y restricciones, en el que se detallan los valores que puede tomar*

```
<xs:element name="NIVEL_FORMATIVO">
        <xs:annotation>
                <xs:documentation>Obligatorio. Formato de 2 posiciones con los valores
                        siguientes: 00 - Sin estudios 10 - Nivel Básico 20 - Estudios
                        secundarios 30 - Estudios post-secundarios
                </xs:documentation>
        </xs:annotation>
        <xs:simpleType>
                <xs:restriction base="xs:string">
                        <xs:length value="2"/>
                        <xs:enumeration value="00"/>
                        <xs:enumeration value="10"/>
                        <xs:enumeration value="20"/>
                        <xs:enumeration value="30"/>
                </xs:restriction>
        </xs:simpleType>
</xs:element>
```

Ejemplo: *Listado de clientes. Dado un documento XML se genera el Esquema XML:*

▫ *Documento XML: "clientes.xml"*

```
<?xml version="1.0" encoding="UTF-8"?>
<listaclientes>
   <cliente>
      <cif>01234567D</cif>
      <nombre>Juan Sanchez</nombre>
   </cliente>
   <cliente>
      <cif>05676554A</cif>
      <nombre>Pedro Diaz</nombre>
      <plazo>45</plazo>
   </cliente>
</listaclientes>
```

▫ *XML Schema: "clientes.xsd"*

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

   <xsd:element name="listaclientes" type="tipoListaClientes"/>

   <xsd:complexType name="tipoListaClientes">
      <xsd:complexContent>
         <xsd:restriction base="xsd:anyType">
            <xsd:sequence>
               <xsd:element name="cliente" type="tipoCliente"
                        maxOccurs="unbounded"/>
            </xsd:sequence>
         </xsd:restriction>
      </xsd:complexContent>
   </xsd:complexType>

   <xsd:complexType name="tipoCliente">
      <xsd:complexContent>
         <xsd:restriction base="xsd:anyType">
         <xsd:sequence>
            <xsd:element name="cif" type="tipoCif"/>
            <xsd:element name="nombre" type="xsd:string"/>
            <xsd:element name="plazo" type="xsd:unsignedInt" minOccurs="0"/>
         </xsd:sequence>
         </xsd:restriction>
      </xsd:complexContent>
   </xsd:complexType>

   <xsd:simpleType name="tipoCif">
      <xsd:restriction base="xsd:string">
         <xsd:pattern value="[0-9]{8}[A-Z]"/>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="tipoPlazo">
      <xsd:restriction base="xsd:unsignedInt"/>
   </xsd:simpleType>

</xsd:schema>
```

Ejemplo: *Listado de alumnos. Dado un documento XML se genera el Esquema XML:*

▫ *Documento XML: "alumno.xml"*

```
<?xml version="1.0" encoding="UTF-8"?>
<listaalumnos>
   <!--DNI atributo obligatorio-->
   <alumno dni="5667545Z">
      <!--Nombre y ap1 obligatorios-->
      <nombre>Jose</nombre>
      <ap1>Sanchez</ap1>
```

```
    </alumno>
    <alumno dni="5778221D">
        <nombre>Andres</nombre>
        <ap1>Ruiz</ap1>
        <!--Ap2 y edad son optativos-->
        <ap2>Ruiz</ap2>
        <!--La edad debe ser positiva-->
        <edad>25</edad>
    </alumno>
</listaalumnos>
```

▫ *XML Schema: "alumnos.xsd"*

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

    <xsd:element name="listaalumnos" type="tipoListaAlumnos"/>

    <xsd:complexType name="tipoListaAlumnos">
        <xsd:complexContent>
            <xsd:restriction base="xsd:anyType">
                <xsd:sequence>
                    <xsd:element name="alumno" type="tipoAlumno"
                            maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:restriction>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="tipoAlumno">
        <xsd:complexContent>
            <xsd:restriction base="xsd:anyType">
                <xsd:sequence>
                    <xsd:element name="nombre" type="xsd:string"/>
                    <xsd:element name="ap1" type="xsd:string"/>
                    <xsd:element name="ap2" type="xsd:string" minOccurs="0"/>
                    <xsd:element name="edad" type="xsd:positiveInteger" minOccurs="0"/>
                </xsd:sequence>
                <xsd:attribute name="dni" type="tipoDNI" use="required"/>
            </xsd:restriction>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:simpleType name="tipoDNI">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="[0-9]{7,8}[A-Z]"/>
        </xsd:restriction>
    </xsd:simpleType>

</xsd:schema>
```

Ejemplo: *Productos Financieros. Dado un documento XML se genera el Esquema XML: Las reglas concretas son las siguientes:*

1.El elemento raíz es <productosfinancieros>. Dentro de él debe haber uno o más elementos <producto>.
2.Un <producto> puede ser de tres tipos: <bono>, <futuro> y <acciones>.
3.Todos los productos tienen siempre un elemento hijo llamado <beneficio> que puede ser un número con dos decimales (puede ser positivo o negativo).
4.Todo <bono> puede tener dentro un elemento llamado <valoractual> que contiene un valor decimal que puede ser positivo o negativo y tener o no decimales. El elemento <valoractual> deberá llevar dentro un atributo llamado moneda que solo puede tomar los valores dolares, euros o yenes.
5.Todo <futuro> tiene un hijo llamado <elemento> que puede contener dentro cadenas de cualquier tipo. Para saber en qué idioma está la cadena se usa un atributo llamado idioma que indica el idioma en el que está escrita la cadena.
6.Las acciones siempre tienen un elemento <empresa> que indica el nombre de la empresa y un atributo llamado país que indica de donde es la empresa. De momento queremos limitarnos a los países usa, alemania, japon y espana.

Se supone que un elemento o atributo es obligatorio si no se indica nada.
Si algo es optativo se indicará «puede tener dentro», «puede contener», «puede aparecer», etc…

▫ *Documento XML: "productosfinancieros.xml"*

```
<?xml version="1.0" encoding="UTF-8"?>
<productosfinancieros>
   <producto>
      <bono>
         <valoractual moneda="yenes">2.212</valoractual>
         <beneficio>-2.83</beneficio>
      </bono>
   </producto>
   <producto>
      <futuro>
         <elemento idioma="espanol">Petroleo</elemento>
         <beneficio>-3.83</beneficio>
      </futuro>
   </producto>
   <producto>
      <acciones>
         <empresa pais="usa">ENRON</empresa>
         <beneficio>2.91</beneficio>
      </acciones>
   </producto>
</productosfinancieros>
```

▫ *XML Schema: "productosfinancieros.xsd"*

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

   <xsd:element name="productosfinancieros" type="tipoProductosFinancieros"/>
```

```
<xsd:complexType name="tipoProductosFinancieros">
   <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
         <xsd:sequence>
            <xsd:element name="producto" type="tipoProducto"
                  maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:restriction>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tipoProducto">
   <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
         <xsd:choice>
            <xsd:element name="bono" type="tipoBono"/>
            <xsd:element name="futuro" type="tipoFuturo"/>
            <xsd:element name="acciones" type="tipoAcciones"/>
         </xsd:choice>
      </xsd:restriction>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tipoBono">
   <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
         <xsd:sequence>
            <xsd:element name="valoractual" type="tipoValorActual"/>
            <xsd:element name="beneficio" type="tipoBeneficio"/>
         </xsd:sequence>
      </xsd:restriction>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="tipoValorActual">
   <xsd:simpleContent>
      <xsd:extension base="xsd:decimal">
         <xsd:attribute name="moneda" type="tipoMoneda"/>
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>

<xsd:simpleType name="tipoMoneda">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="dolares"/>
      <xsd:enumeration value="euros"/>
      <xsd:enumeration value="yenes"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="tipoBeneficio">
```

```
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name="tipoFuturo">
      <xsd:complexContent>
        <xsd:restriction base="xsd:anyType">
          <xsd:sequence>
            <xsd:element name="elemento" type="tipoElemento"/>
            <xsd:element name="beneficio" type="tipoBeneficio"/>
          </xsd:sequence>
        </xsd:restriction>
      </xsd:complexContent>
    </xsd:complexType>

    <!--No nos dicen nada sobre los posibles idiomas, así que podemos asumir que el
idioma será una cadena cualquiera-->
    <xsd:complexType name="tipoElemento">
      <xsd:simpleContent>
        <xsd:extension base="xsd:string">
          <xsd:attribute name="idioma" type="xsd:string"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>

    <xsd:complexType name="tipoAcciones">
      <xsd:complexContent>
        <xsd:restriction base="xsd:anyType">
          <xsd:sequence>
            <xsd:element name="empresa" type="tipoEmpresa"/>
            <xsd:element name="beneficio" type="tipoBeneficio"/>
          </xsd:sequence>
        </xsd:restriction>
      </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="tipoEmpresa">
      <xsd:simpleContent>
        <xsd:extension base="xsd:string">
          <xsd:attribute name="pais" type="tipoPais"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>

    <xsd:simpleType name="tipoPais">
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="alemania"/>
        <xsd:enumeration value="japon"/>
        <xsd:enumeration value="espania"/>
```

```
        <xsd:enumeration value="usa"/>
      </xsd:restriction>
    </xsd:simpleType>

</xsd:schema>
```

13. Resumen componentes fundamentales Esquema XML.

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:sn=“ URI_de_un_espacio_de_nombres”
            targetNamespace=”…” >

        <xs:element name=”elementoRaizXML”>
            <xs:sequence>

            </xs:sequence>
        </xs:element>

        <!-- 1. Tipos de datos Simples -->

        <!-- 1.1 Tipos de datos simples primitivos -->
        <xs:element name="nombre_del_elemento" … type="xs:tipo_de_dato"/>
        <xs:attribute name="nombre_del_atributo" … type="xs:tipo_de_dato"/>

        <!-- 1.2 Tipos de datos simples construidos (aplicar restricciones a tipo simple --
        >
        <xs:simpleType  name="….">
            <xs:restriction base="xs:tipoOrigen">
                <xs:tipoRestricción value=”..”/>
                …
            </xs:restriction>
        </xs:simpleType>

        <!-- 2. Tipos de datos Complejos -->

        <!-- 2.1 Contienen solo elementos (sequence, choice, all) -->
        <xs:complexType name=”…”>
            <xs:sequence>
                <xs:element …/>
                …
            </xs:sequence>
        </xs:complexType>

        <!-- 2.2 Contienen solo atributos -->
        <!-- 2.2.1 Asignar atributos a Tipos simples, por extensión -->
        <!-- asigna atributos a un elemento con contenido textual y sin descendientes -->
        <xs:complexType name=”…”>
            <xs:simpleContent>
                <xs:extension base=”xs:tipoOrigen”/>
                    <xs:attribute name=”…” …/>
                    …
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
```

```
        <!-- 2.2.2 Asignar atributos a Tipos complejos -->
        <xs:complexType name="…">
                <xs:attribute name="…" …/>
                <xs:attribute name="…" …/>
                …
        </xs:complexType>

        <!-- 2.3 Contienen atributos y otros elementos -->
        <xs:complexType name="…">
                <xs:sequence>
                        <xs:element …/>
                        …
                </xs:sequence>
                <xs:attribute name="…" …/>
                …
        </xs:complexType>

        <!-- 2.4 Elementos vacíos (sin descendientes ni atributos) -->
        <xs:complexType />


        <!-- 2.5 Extensión de tipos de datos complejos -->
        <xs:complexType name="tipoExtendido">
                <xs:complexContent>
                        <xs:extension base="tipoComplejoInicial">
                                <!-- Nuevos elementos y/o atributos -->
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>

</xs:schema>
```