LENGUAJE DE MARCAS

Y

SISTEMAS DE GESTION

DE LA INFORMACIÓN

RA6: Gestiona la información en formatos de intercambio de datos

Tema 6: Gestión y almacenamiento de información en XML

Los sistemas gestores de bases de datos relacionales más clásicos se han adaptado para poder almacenar y tratar información con XML (bases de datos con XML no nativas).

Por contra, también veremos qué sistemas gestores de bases de datos trabajan de manera nativa con XML, así como sus herramientas y técnicas para tratar la información en XML.

6.1. Sistemas de almacenamiento de información XML. Tecnologías

Las bases de datos se pueden definir como un sistema de almacenamiento de información relacionada entre sí, de manera estructurada, y accesible digitalmente.

Podemos usar los documentos XML como un sistema estructurado y accesible para almacenar información, lo cual rompe por completo con la filosofía de las bases de datos tradicionales.

Por tanto, debemos tener en cuenta que si vamos a almacenar y tratar información de documentos XML podemos hacerlo:

- Con una base de datos relacional adaptada a la información XML (base de datos XML no nativa).
- Con una base de datos nativa XML, tecnología que está actualmente en desarrollo.

Las ventajas de usar una base de datos relacional adaptada para XML son:

- Las bases de datos relacionales poseen una estructura regular y homogénea, en contraposición a la irregularidad de la información XML.
- Las bases de datos relacionales están muy implantadas hoy en día. Luego con una BBDD adaptada a XML no será necesario cambiar todo el paradigma del sistema de información de una institución o empresa si esta quisiese o necesitase comenzar a trabajar con información en XML.

Las desventajas de usar una base de datos relacional adaptada a XML son:

- Posee una estructura mucho más densa y compleja (tablas, índices, *triggers*...).
- Las bases de datos relacionales estructuralmente difieren de la tecnología XML. Luego, la realización de una consulta XML en una base de datos relacional es más compleja

6.2 Bases de datos relacionales con XML. Inserción de información

No es lo mismo un sistema gestor de bases de datos relacional adaptado para tratar información XML que un sistema de bases de datos XML nativo.

Este tipo de bases de datos suelen centrarse en los documentos, de modo que gestionan información contenida en un documento estructurado, en este caso documentos XML.

Dos de los sistemas gestores de bases de datos relacionales adaptados a la información XML son: ORACLE y SQL Server:

SQL server:

Para el manejo de la información de tipo XML crea un tipo de columna especial para el contenido de un documento XML y se tiene en cuenta el esquema de dicho documento XML (como los DTD o los schematron).

El tipo de campo especial que se usa para XML es XMLTYPE.

Podemos crear una tabla, mediante la sentencia CREATE TABLE, visto en módulos anteriores, donde uno de los campos (de las columnas) debe ser de tipo XMLTYPE. Por tanto, a la hora de crearnos un registro en una tabla, la información que va a la columna tipo XMLTYPE debe contener un fragmento de código XML.

Una vez creada la tabla con el campo tipo XML, ya podemos realizar todas las operaciones de inserción (INSERT) o consulta (SELECT).

Otra forma de ejecutar una consulta en este lenguaje es a través de la función **XMLQuery()**, en la cual recibirá como parámetro la consulta, además del campo XML. Existe, por otra parte, la cláusula **XMLTable()**, mediante la cual podemos tratar el resultado de una consulta con XQuery.

6.3 Tecnologías de búsqueda de información en documentos XML

Para acceder y consultar datos en un documento XML como lo haríamos en una base de datos relacional, debemos disponer de un lenguaje de consulta específico, que sea capaz de recorrer el documento XML y realizar ciertas operaciones básicas.

Dos de los lenguajes de consultas más populares para XML son los siguientes:

- **XPath**: lenguaje de consulta bastante sencillo, que dispone de una gran cantidad de expresiones que nos permiten recorrer el árbol de nodos en un documento XML y acceder a sus diferentes partes.
- **XQuery**: es otro lenguaje de consulta para documentos XML con aspectos añadidos de funcionalidad, como la de manipulación de la información en un documento XML.

Ambas son complementarias, **XPath** se centra en la localización de datos dentro de XML, mientras que **XQuery** permite realizar consultas y transformaciones más complejas sobre documentos XML. Además, permite realizar una serie de acciones para manipular esa información.

6.4 Lenguajes de consulta, Extracción y manipulación de la información en XML

6.4.1 XPath

XPath es un lenguaje diseñado para acceder, transformar y dar formato de salida a los documentos XML.

De la misma forma que podemos recorrer el árbol de directorios mediante unos comandos preestablecidos (*cd*, *dir*, *ls*, etc.). También podemos hacerlo en un documento XML con comandos XPath.

Direccionamiento

Directorios en XPath

En el ámbito del contenido web, el direccionamiento o la ruta en XPath puede ser de dos tipos:

Ruta absoluta:

- Comienza desde el directorio raíz (/).

Ruta relativa:

- Inicia desde el directorio en el que nos encontramos.
- Se utilizan para representar el directorio actual y para acceder al directorio padre.

Estructura de un Documento XML en XPath

XPath organiza un documento XML en forma de un **árbol de nodos**, similar al modelo DOM (*Document Object Model*).

Dentro de este modelo, se pueden identificar distintos tipos de nodos XPath:

Nodo raíz del documento:

Nodo principal, ya que, a partir de los demás, comienza a derivar el resto. Cada documento tiene un único nodo raíz y se caracteriza por ser un nodo que no tiene padre y por tener, al menos, un hijo.

Elemento: Representa cualquier elemento dentro del XML.

Atributo: Equivalente a un nodo de atributo en XML.

Texto: Contiene la información almacenada dentro del documento XML.

Comentarios: Contiene texto que no es procesado por el compilador. Útil para que los desarrolladores documenten su código y faciliten su mantenimiento.

- **Instrucciones de procesamiento:** Permite incluir instrucciones específicas para los procesadores de XML.
- **Espacio de nombres (*namespace*):** Define el ámbito de los elementos y atributos dentro del documento XML.

Tipos de Datos en XPath

XPath maneja varios tipos de datos primitivos:

- **String:** representa cadenas de caracteres.
- **Number:** maneja valores numéricos.
- **Boolean:** almacena valores lógicos (true o false).
- **Nodo:** estructura los datos jerárquicamente, al igual que en DOM.

XPath es un lenguaje de expresiones, estas son las más comunes:

Expresión	Descripción	Ejemplo
elem	Selecciona todos los elementos con el nombre elem.	book selecciona todos los nodos <book>.
/elem	Selecciona el elemento elem situado en el directorio raíz.	/catalog selecciona el nodo <catalog> en la raíz.
e1/e2	Selecciona el elemento e2 que es hijo directo de e1.	library/book selecciona todos los <book> dentro de <library>.
e1//e2	Selecciona el elemento e2 que es descendiente de e1, sin importar el nivel.	library//title selecciona todos los <title> dentro de <library>, sin importar la jerarquía.
//elem	Selecciona todos los elementos elem en cualquier parte del documento.	//book selecciona todos los <book> sin importar su ubicación.
@atrib	Selecciona un atributo con el nombre atrib.	//book[@category] selecciona <book> con el atributo category.
*	Carácter comodín que selecciona cualquier elemento.	//book/* selecciona todos los elementos dentro de <book>.
Expresión	Descripción	Ejemplo

@*	Comodín que selecciona todos los atributos de un elemento.	//book/@* selecciona todos los atributos de <book>.
.	Representa el nodo actual en el que nos encontramos.	./title selecciona el <title> dentro del nodo actual.
..	Representa el nodo padre del nodo actual.	../author selecciona el <author> del nodo padre.
espNom:*	Selecciona todos los elementos del espacio de nombres espNom.	xhtml:* selecciona todos los elementos dentro del espacio de nombres xhtml.
@espNom:*	Selecciona todos los atributos del espacio de nombres espNom.	@xhtml:* selecciona todos los atributos dentro del espacio de nombres xhtml.

Acceso a los elementos / Atributos

Este lenguaje se utiliza como analizador de expresión a partir de un documento XML compuesto por nodos con sus identificadores atributos y valores. Utilizamos estas expresiones cuando deseamos tratar un conjunto de nodos en especial. A la hora de escribir bajo el lenguaje de expresiones XPath, lo único que queremos es trabajar con los nodos de una determinada expresión.

Acceso a elementos con rutas simples

Son una serie de rutas a las que podemos acceder mediante un directorio absoluto, es decir, empezando desde el nodo raíz del sistema, devolviendo el conjunto de nodos que se encuentre en dicha ruta.

Elementos del lenguaje

Avanzando en la sintaxis del lenguaje XPath vamos a generar expresiones más complejas a partir de los elementos básicos ya vistos.

1. Operadores en XPath

Operadores Booleanos

Se utilizan para realizar operaciones lógicas entre valores booleanos (True/False). Los operadores booleanos más comunes en XPath son:

- **and**: Devuelve True si ambas condiciones son verdaderas.
 - Ejemplo: /coches/coche[nombre="Toyota" and matrícula="1234ABC"]
- **or**: Devuelve True si al menos una de las condiciones es verdadera.
 - Ejemplo: /coches/coche[nombre="Toyota" or nombre="Ford"]
- **not()**: Devuelve True si la condición dentro de la función no es verdadera.
 - Ejemplo: /coches/coche[not(nombre="Ford")]
- **=**: Comparación de igualdad.
 - Ejemplo: /coches/coche[nombre="Toyota"]
- **!=**: Comparación de desigualdad.
 - Ejemplo: /coches/coche[nombre!="Toyota"]
- **<, <=, >, >=**: Comparaciones de menor que, menor o igual que, mayor que, mayor o igual que.
 - Ejemplo: /coches/coche[matrícula < "5000XYZ"]

Operadores Aritméticos

Los operadores aritméticos son usados para realizar operaciones matemáticas con números en XPath:

- **+**: Suma dos valores.
- **-**: Resta dos valores.
- *****: Multiplica dos valores.
- **div**: División de dos valores.
- **mod**: Devuelve el residuo de la división de dos valores.

Otros Operadores

- **|**: Unión de resultados

2. Funciones en XPath

Las funciones en XPath nos permiten realizar operaciones más avanzadas sobre los nodos seleccionados. Vamos a ver algunas de las funciones más comunes.

Funciones Numéricas

Estas funciones operan sobre valores numéricos:

- **sum()**: Suma todos los valores de los nodos seleccionados.
 - Ejemplo: `sum(//precio)` devuelve la suma de todos los valores de los nodos `<precio>`.
- **abs ()**: Da un valor absoluto.
 - Ejemplo: `abs (-7) = 7`
- **round()**: Redondea un número a su valor entero más cercano.
 - Ejemplo: `round(10.5)` devuelve 11.
- **floor()**: Redondea un número hacia abajo al entero más cercano.
 - Ejemplo: `floor(10.8)` devuelve 10.
- **ceiling()**: Redondea un número hacia arriba al entero más cercano.
 - Ejemplo: `ceiling(10.2)` devuelve 11.

Funciones de Cadenas

Estas funciones operan sobre valores de tipo cadena (texto):

- **substring()**: Devuelve una subcadena de la cadena original, basada en una posición inicial y una longitud dada.
 - Ejemplo: `substring("Beatles", 1, 4)` devuelve "Beat".
- **starts-with()**: Verifica si una cadena comienza con el valor especificado.
 - Ejemplo: `starts-with("XML", "X")` devuelve true, ya que "XML" comienza con la letra "X".
- **contains()**: Verifica si una cadena contiene una subcadena específica.
 - Ejemplo: `contains("XML", "XM")` devuelve true, ya que "XML" contiene "XM".
- **normalize-space()**: Elimina los espacios adicionales al principio y final de la cadena, y reemplaza los espacios consecutivos en el interior por uno solo.
 - Ejemplo: `normalize-space(" The end ")` devuelve "The end".
- **translate()**: Sustituye caracteres en la cadena por otros caracteres especificados.
 - Ejemplo: `translate("The end", "The", "An")` devuelve "An end".
- **string-length()**: Devuelve la longitud de una cadena.
 - Ejemplo: `string-length("Toyota")` devuelve 6.

Funciones que Devuelven Elementos por su Posición

Estas funciones nos permiten seleccionar nodos en función de su posición dentro de un conjunto de nodos.

- **position():** Devuelve la posición de un nodo dentro de su conjunto de nodos.
 - Ejemplo: `/coches/coche[nombre="Toyota"][position()=1]` selecciona el primer coche con nombre "Toyota".
- **last():** Devuelve la última posición de un conjunto de nodos.
 - Ejemplo: `/coches/coche[last()]` selecciona el último coche.

Funciones que Devuelven Nodos

- **name():** Devuelve el nombre del nodo actual. Esta función se usa para obtener el nombre del nodo que está siendo procesado en el momento.
 - Ejemplo: Si el nodo actual es `<coche>`, `name()` devolverá "coche".
- **root():** Devuelve el nodo raíz del documento. Esta función te permite acceder directamente al nodo raíz, independientemente de la ubicación actual en el árbol XML.
 - Ejemplo: `root()` devolverá el nodo raíz del documento XML.
- **node():** Selecciona cualquier tipo de nodo.
 - Ejemplo: `/coches/coche/node()` selecciona todos los nodos de tipo coche.
- **comment():** Selecciona los nodos de comentario dentro del documento XML. Esta función es útil si deseas extraer o trabajar con los comentarios en el XML.
 - Ejemplo: `comment()` seleccionará todos los nodos de comentario del documento.
- **processing-instruction():** Permite acceder a las instrucciones de procesamiento en el documento XML. Las instrucciones de procesamiento son directivas especiales dentro del XML que no son consideradas parte del contenido de los datos.
 - Ejemplo: `processing-instruction()` seleccionará todas las instrucciones de procesamiento en el documento.

Funciones de Agregado

- **count():** Devuelve el número de nodos que coinciden con una expresión.
 - Ejemplo: `count(//coches/coche)` devuelve el número total de coches en el XML.
- **avg():** Calcula el valor promedio de un conjunto de nodos numéricos.
 - Ejemplo: `avg(//precio)` devuelve el promedio de los precios de todos los coches.
- **min():** Devuelve el valor mínimo de un conjunto de nodos numéricos.
 - Ejemplo: `min(//precio)` devuelve el precio más bajo de los coches.
- **max():** Devuelve el valor máximo de un conjunto de nodos numéricos.
 - Ejemplo: `max(//precio)` devuelve el precio más alto de los coches.
- **sum():** Suma del valor de un conjunto de nodos:
 - Ejemplo: `sum(//productos/producto/precio)`

Acceso a elementos con filtro de valores

- **Literales:** que van entre dobles comillas para acceder a ese valor concreto.
- **Variables:** para utilizar variables en Xpath debemos usar el símbolo del \$, de este modo podemos referir a un valor contenido dentro de la propia variable.
- **Valores recuperados:** permiten acceso mediante otras expresiones.

Acceso a elementos mediante ejes

Estas expresiones permiten acceder a partes concretas del árbol XML, siempre que exista parentesco entre los distintos nodos. Esta es una condición indispensable.

- **shelf::*:**
Devuelve el propio nodo de contexto. Es una forma de referirse al nodo en el que te encuentras actualmente.
- **child:::**
Devuelve los nodos hijo del nodo de contexto. Utilizado para seleccionar los elementos que son hijos directos del nodo en el que te encuentras.
 - Ejemplo: `child::producto` selecciona todos los hijos llamados `<producto>` del nodo de contexto.
- **parent::*:**
Devuelve el nodo padre del nodo de contexto. Te permite retroceder en la jerarquía y obtener el nodo que está por encima del nodo actual.
 - Ejemplo: `parent::categoria` devuelve el nodo padre que tiene el nombre `<categoria>`.
- **ancestor::*:**
Devuelve los antepasados del nodo de contexto (padre, abuelo, etc.) hasta el nodo raíz. Selecciona todos los nodos que están por encima del nodo de contexto.
 - Ejemplo: `ancestor::producto` devuelve todos los ancestros del nodo actual que son de tipo `<producto>`.
- **ancestor-or-self::*:**
Devuelve los nodos antepasados (padre, abuelo, etc.) y el propio nodo de contexto. Es una combinación del nodo de contexto y sus antepasados.
 - Ejemplo: `ancestor-or-self::producto` devuelve el nodo de contexto y todos los antecesores que sean `<producto>`.
- **descendant::*:**
Devuelve los nodos descendientes (hijo, nieto, bisnieto...) del nodo de contexto. Selecciona todos los elementos que están debajo del nodo actual, sin importar cuántos niveles de jerarquía existan.
 - Ejemplo: `descendant::producto` selecciona todos los elementos `<producto>` que son descendientes del nodo de contexto.
- **descendant-or-self::*:**
Devuelve los nodos descendientes del nodo de contexto, además del propio nodo de contexto. Equivalente a `//` en XPath, ya que selecciona el nodo actual y todos los descendientes.
 - Ejemplo: `descendant-or-self::producto` selecciona el nodo de contexto y todos sus descendientes `<producto>`.
- **following::*:**
Devuelve los nodos que aparecen después del nodo de contexto en el documento, excluyendo los nodos descendientes, atributos y espacios de nombres.
 - Ejemplo: `following::producto` selecciona todos los nodos `<producto>` que vienen después del nodo de contexto, pero no los que son descendientes.
- **preceding::*:**
Devuelve los nodos que aparecen antes del nodo de contexto en el documento, excluyendo los nodos ascendentes, atributos y nodos de espacio de nombres.
 - Ejemplo: `preceding::producto` selecciona todos los nodos `<producto>` que vienen antes del nodo de contexto.

- **preceding-sibling::*:**
Devuelve los nodos hermanos mayores del nodo de contexto, es decir, los elementos que están al mismo nivel en la jerarquía, pero antes en el documento.
 - Ejemplo: `preceding-sibling::producto` selecciona los nodos `<producto>` que son hermanos anteriores al nodo de contexto.
- **following-sibling::*:**
Devuelve los nodos hermanos menores del nodo de contexto, es decir, los elementos que están al mismo nivel en la jerarquía, pero después en el documento.
 - Ejemplo: `following-sibling::producto` selecciona los nodos `<producto>` que son hermanos posteriores al nodo de contexto.
- **attribute::*:**
Selecciona los atributos del nodo de contexto. Es equivalente a utilizar `@` para referirse a atributos en XPath.
 - Ejemplo: `attribute::matrícula` selecciona el atributo `matrícula` del nodo de contexto.
- **namespace::*:**
Devuelve los espacios de nombres del nodo de contexto. Se utiliza para obtener los prefijos y URI de los espacios de nombres asociados con el nodo actual.
 - Ejemplo: `namespace::*` selecciona todos los espacios de nombres asociados con el nodo de contexto.

6.4.2 XQuery

En este apartado veremos el lenguaje XQuery, que permite realizar consultas para extraer y procesar la información contenida en el documento XML.

XQuery diferencia entre dos partes: una que se asemeja al lenguaje SQL, ya que es un lenguaje orientado a bases de datos donde comparte las mismas cláusulas que esta emplea, como *where*, *orderby*, etc. y, por otra parte, también podemos compararlo al lenguaje XPath descrito anteriormente, ya que ambos se refieren a documentos de marca como XML.

Al ser un lenguaje de expresiones, XQuery también se compone de los mismos tipos de nodos que el lenguaje anterior: nodo raíz, elementos, atributos, textos, comentarios, expresiones, etc. Y, además, de los mismos tipos de datos primitivos que XPath: numéricos, booleanos, cadenas de texto, fecha hora, etc.

Funciones o cláusulas:

Función *doc()*: permite leer un documento XML que indiquemos por parámetro, devolviendo el nodo raíz.

Función FLWOR: recibe este nombre por ser un conjunto de funciones que se refieren a *for*, *let*, *where*, *orderby*, *return*. Las iniciales de estas funciones dan nombre al conjunto que la identifica. Estas funciones hacen posible construir funciones en XQuery.

- **for:** indica, mediante un rango de valores, los elementos a tratar.
- **let:** permite declarar variables para que, a continuación, podamos asignarle valores.
- **where:** acompaña la sentencia *for*, permitiendo introducir una condición para que la estructura iterativa se repita mientras se cumpla dicha condición.
- **orderby:** parámetro por el que vamos a ordenar la visualización del resultado de una determinada consulta.
- **return:** comando que nos permite devolver un resultado de una expresión dada.

Otras cláusulas:

- **declare function:** permite declarar funciones en XQuery.
- **if ... else:** simula una situación condicional. Ejecutamos una parte de la estructura u otra mediante el cumplimiento de dicha condición.

Funciones predefinidas

Funciones de Texto

- **uppercase():** Convierte una cadena de texto a mayúsculas.
 - Ejemplo: uppercase("hola") devuelve "HOLA".
- **substring():** Devuelve una subcadena de texto a partir de una cadena original. Se especifica el índice inicial y la longitud.
 - Ejemplo: substring("abcdef", 2, 3) devuelve "bcd".
- **contains():** Verifica si una cadena de texto contiene un subtexto especificado.
 - Ejemplo: contains("hello", "ell") devuelve true.

Funciones de Números

- **max():** Devuelve el valor máximo de un conjunto de números.
 - Ejemplo: max(3, 5, 7, 2) devuelve 7.
- **sum():** Devuelve la suma total de un conjunto de números.
 - Ejemplo: sum(1, 2, 3, 4) devuelve 10.
- **avg():** Devuelve la media aritmética de un conjunto de números.
 - Ejemplo: avg(2, 4, 6) devuelve 4.

Funciones de Fecha

- **current-date():** Devuelve la fecha actual del sistema.
 - Ejemplo: current-date() devuelve la fecha actual en formato YYYY-MM-DD.
- **current-time():** Devuelve la hora actual del sistema.
 - Ejemplo: current-time() devuelve la hora actual en formato HH:MM:SS.

Funciones de Nodos XML

- **root():** Devuelve el nodo raíz de un documento XML. Es útil para trabajar con árboles XML, ya que permite acceder a la raíz del documento.
 - Ejemplo: root(document) devuelve el nodo raíz del documento XML

6.5 Sistemas gestores de bases de datos nativos XML. Herramientas para el tratamiento y el almacenamiento de la información en XML.

Una alternativa de estos SGBD pueden ser los sistemas de bases de datos nativos XML. Así, mientras los primeros estaban basados en tablas con su información correspondiente relacionada entre sí, estos segundos lo que almacenan son documentos XML

De esta forma, podemos comprobar que los SGBD tradicionales son adecuados para almacenar datos y los gestores de bases de datos XML son adecuados para almacenar documentos.

Gestores de bases de datos nativos podemos encontrar **BaseX** y **eXist**.

BaseX

BaseX es un motor de base de datos nativo XML que incluye XPath y XQuery. Presenta las características de ser una aplicación ligera, de alto rendimiento en las distintas operaciones que realiza y fácilmente escalable.

La información resultante de una consulta se puede presentar de diferentes formas:

- **Modo Texto:** La información se presenta como texto plano, lo cual es útil para ver los resultados de manera simple.
- **Estructura de Árbol:** Visualización jerárquica, ideal para mostrar cómo están organizados los nodos XML.
- **Estructura de Carpetas:** Permite ver los resultados organizados en carpetas o directorios, similar a cómo se organizarían en un sistema de archivos.
- **Modo Tabla:** Presenta los resultados en formato tabular, lo que es útil para consultas que devuelven datos estructurados.
- **Diagrama de Dispersión:** Representación gráfica de los resultados, ideal para mostrar relaciones entre los datos.

En la interfaz gráfica de esta herramienta podemos visualizar fácilmente los distintos apartados para realizar las operaciones. En la línea de comando ejecutaremos las cláusulas para trabajar con bases de datos XML.

- **Command:** en este apartado podemos indicar los comandos propios de la aplicación para trabajar con las bases de datos, como crear una base de datos, tratamiento de usuario, abrir o modificar una base de datos, etc.
- **Search:** en este apartado indicaremos expresiones XPath que detallaremos más adelante.
- **XQuery:** indicaremos los diferentes comandos para realizar las consultas.

Categoría	Tipo de dato	Descripción	Ejemplo
Tipos numéricos	xs:decimal	Números decimales	12.34
	xs:integer	Números enteros	100
	xs:nonPositiveInteger	Enteros ≤ 0	-5
	xs:negativeInteger	Enteros < 0	-10
	xs:nonNegativeInteger	Enteros ≥ 0	5
	xs:positiveInteger	Enteros > 0	25
	xs:long	Enteros largos (64 bits)	9223372036854775807
	xs:int	Enteros de 32 bits	2147483647
	xs:short	Enteros de 16 bits	32767
	xs:byte	Enteros de 8 bits	127
Tipos de coma flotante	xs:float	Números en coma flotante (32 bits)	3.14E2
	xs:double	Números en coma flotante (64 bits)	1.6E-3
Tipos booleanos	xs:boolean	Valores verdadero o falso	true, false
Tipos de cadenas	xs:string	Cadenas de caracteres	"Hola mundo"
	xs:normalizedString	String sin saltos de línea ni tabulaciones	"Ejemplo Texto"
	xs:token	String sin espacios repetidos ni saltos de línea	"Texto sin espacios"
	xs:language	Código de idioma ISO	"es" (Español)
	xs:Name	Nombre válido XML	"nombreElemento"
	xs:NCName	Nombre sin dos puntos :	"nombre_sin_puntos"
	xs:ID	Identificador único	"ID123"
	xs:IDREF	Referencia a un ID	"ID123"
Tipos de fecha y hora	xs:date	Fecha (AAAA-MM-DD)	2025-01-30
	xs:time	Hora (HH:MM:SS)	14:30:00
	xs:dateTime	Fecha y hora combinadas	2025-01-30T14:30:00
	xs:gYear	Año	2025
	xs:gMonth	Mes	--01
	xs:gDay	Día del mes	---30
	xs:gYearMonth	Año y mes	2025-01
	xs:gMonthDay	Mes y día	--01-30
Tipos binarios	xs:hexBinary	Datos binarios en formato hexadecimal	"4A3BFF"
	xs:base64Binary	Datos binarios codificados en Base64	"U29tZSB0ZXh0"
Tipos derivados	xs:IDREFS	Lista de referencias a ID separadas por espacio	"ID1 ID2 ID3"
	xs:ENTITY	Nombre de una entidad XML	"myEntity"
	xs:ENTITIES	Lista de nombres de entidades	"ent1 ent2"
	xs:NMTOKEN	Token XML válido sin espacios	"Token123"
	xs:NMTOKENS	Lista de tokens separados por espacio	"Token1 Token2"