

## 1. Utilización de XML para almacenamiento de información.

Gran parte de las bases de datos están basadas en un modelo de datos entidad-relación que es un sistema que sigue funcionando.

Las BD orientadas a objetos permiten simplificar el trabajo de integración con los lenguajes de programación orientados a objetos. Existen modelos híbridos con extensiones hacia el modelo relacional.

Como solución para compatibilizar ambos modelos se pensó en XML.

## Bases de Datos Nativas XML

XML permite definir de manera rápida e intuitiva una representación de la información.

En el caso de emplear desde el principio XML se deben utilizar BD XML Nativas.

Las **Bases de Datos Nativas XML** (NXD -Native XML Database) definen el modelo lógico de un documento XML, y almacena y recupera los documentos según ese modelo.

Estas bases de datos tienen como **unidad fundamental de almacenamiento lógico el documento XML**, tal como una base de datos relacional tiene una fila en una tabla como su unidad fundamental de almacenamiento lógico.

Bases de datos nativas XML:

- eXcelon XIL Lite (comercial).
- TEXTML (comercial).
- dbXML (*OpenSource*).
- eXist (*OpenSource*).

Cuando se habla de BD XML nativas hay que distinguir dos maneras de almacenar la información:

- Modelo **centrado en almacenamiento de datos**.

Similar a las BD relacionales (usando tuplas)

- Modelo **centrado en el documento**.

No hay campos ni datos. Se guardan documentos XML

- **El modelo relacional** es “Well Organized”. Más estricto. Totalmente estructurado.  
El esquema relacional (tablas y restricciones de integridad) están predefinidas.
- **Los datos en XML** son semiestructurados (más flexibles).  
Los esquemas no tienen porque ser fijos.  
Ideal para integrar aplicaciones e intercambio de datos.

## 2. Bases de datos relacionales

La adaptación del modelo BD Relacional a XML es sencillo.

Será necesario crear una DTD y un documento XML bien formado.

Cuando se usan modelos centrados en el almacenamiento de los datos la referencia son las BD relacionales.

## 2. Bases de datos relacionales

### PRIMERA VERSIÓN BÁSICA (NO ÓPTIMA y SIN ID)

TABLA LIBROS

Cód_Libro	Titulo	Autor	Editorial	Edición	ISBN	NumPáginas
1	Don Quijote de la Mancha	Miguel de Cervantes Saavedra	Juan de la Cuesta	3	9788466745840	176
2	La Celestina	Fernando de Rojas	Maxtor	1	9788471664938	320
3	Leyendas	Gustavo Adolfo Bécquer	Cátedra	21	9788437620244	416

Usar una única tabla no es operativa ni cumple los condicionantes de una base de datos relacional.

## Modelo de una tabla XML (con DTD)

### PRIMERA VERSIÓN BÁSICA (NO ÓPTIMA y SIN ID)

1

<!ELEMENT Libros (libro)\*>

Tendremos una tabla llamada Libros que almacenará tuplas de tipo libro (cero o más libros).

## Modelo de una tabla a XML (con DTD)

2

Hay que indicar que campos componen la tupla.

```
<!ELEMENT libro (Cod_Libro, Titulo, Editorial, Edicion, ISBN,  
NumPaginas, Autores) >
```



## Modelo de una tabla a XML (con DTD)

3

Cada columna de la tabla deberá establecerse como un tipo de dato almacenable (char, integer, etc → #PCDATA) si es simple.

<!ELEMENT Cod\_Libro (#PCDATA)>

<!ELEMENT Titulo (#PCDATA)>

<!ELEMENT Editorial (#PCDATA)>

<!ELEMENT Edicion (#PCDATA)>

<!ELEMENT ISBN (#PCDATA)>

<!ELEMENT NumPaginas (#PCDATA)>

## Modelo de una tabla a XML (con DTD)

4

Si existe una columna compleja (relacionada con otra tabla) debemos crear un nuevo elemento similar al del paso2, p.e. que un libro pueda tener uno o más autores con sus elementos correspondientes.

```
<!ELEMENT Autores (autor)+>
```

```
<!ELEMENT autor (Cod_Autor, Nombre, Apellidos, FechaNacimiento)>
```

```
<!ELEMENT Cod_Autor (#PCDATA)>
```

```
<!ELEMENT Nombre (#PCDATA)>
```

```
<!ELEMENT Apellidos (#PCDATA)>
```

```
<!ELEMENT FechaNacimiento(#PCDATA)>
```

## Modelo de una tabla a XML (con DTD)

```
<!ELEMENT Libros (libro)*>
```

```
<!ELEMENT libro (Cod_Libro, Titulo, Editorial, Edicion, ISBN, NumPaginas, Autores) >
```

```
<!ELEMENT Cod_Libro (#PCDATA)>
```

```
<!ELEMENT Titulo (#PCDATA)>
```

```
<!ELEMENT Editorial (#PCDATA)>
```

```
<!ELEMENT Edicion (#PCDATA)>
```

```
<!ELEMENT ISBN (#PCDATA)>
```

```
<!ELEMENT NumPaginas (#PCDATA)>
```

```
<!ELEMENT Autores (autor)+>
```

```
<!ELEMENT autor (Cod_Autor, Nombre, Apellidos, FechaNacimiento)>
```

```
<!ELEMENT Cod_Autor (#PCDATA)>
```

```
<!ELEMENT Nombre (#PCDATA)>
```

```
<!ELEMENT Apellidos (#PCDATA)>
```

```
<!ELEMENT FechaNacimiento (#PCDATA)>
```

01ejemplo.xml

01ejemplo.dtd

## Bases de datos relacionales

### VERSIÓN ÓPTIMA. USO DE ID y IDREF

TABLA LIBROS

Cód_Libro	Título	Cód_Autor	Editorial	Edición	ISBN	NumPáginas
1	Don Quijote de la Mancha	1	Juan de la Cuesta	3	9788466745840	176
2	La Celestina	2	Maxtor	1	9788471664938	320
3	Leyendas	3	Cátedra	21	9788437620244	416

Relación



TABLA AUTORES

Cód_Autor	Nombre	Apellidos	Fecha Nacimiento
1	Miguel	de Cervantes Saavedra	29/09/1547
2	Fernando	de Rojas	01/01/1470
3	Gustavo	Adolfo Bécquer	17/02/1836

Usamos dos tablas para vincular y evitar repeticiones de información.

# Modelo relacional a XML (con DTD)

## VERSIÓN ÓPTIMA. USO DE ID y IDREF

La adaptación del modelo BD Relacional a XML es sencillo.

01OPTejemplo.xml

01OPTejemplo.dtd

**Atributo ID único para cada libro**

```
<libro Cod_Libro="L2">  
  <Titulo>La Celestina</Titulo>  
  <Editorial>Maxtor</Editorial>  
  <Edicion>1</Edicion>  
  <ISBN>9788471664938</ISBN>  
  <NumPaginas>320</NumPaginas>  
  <autoria Cod_Autor="A2"/>  
</libro>
```

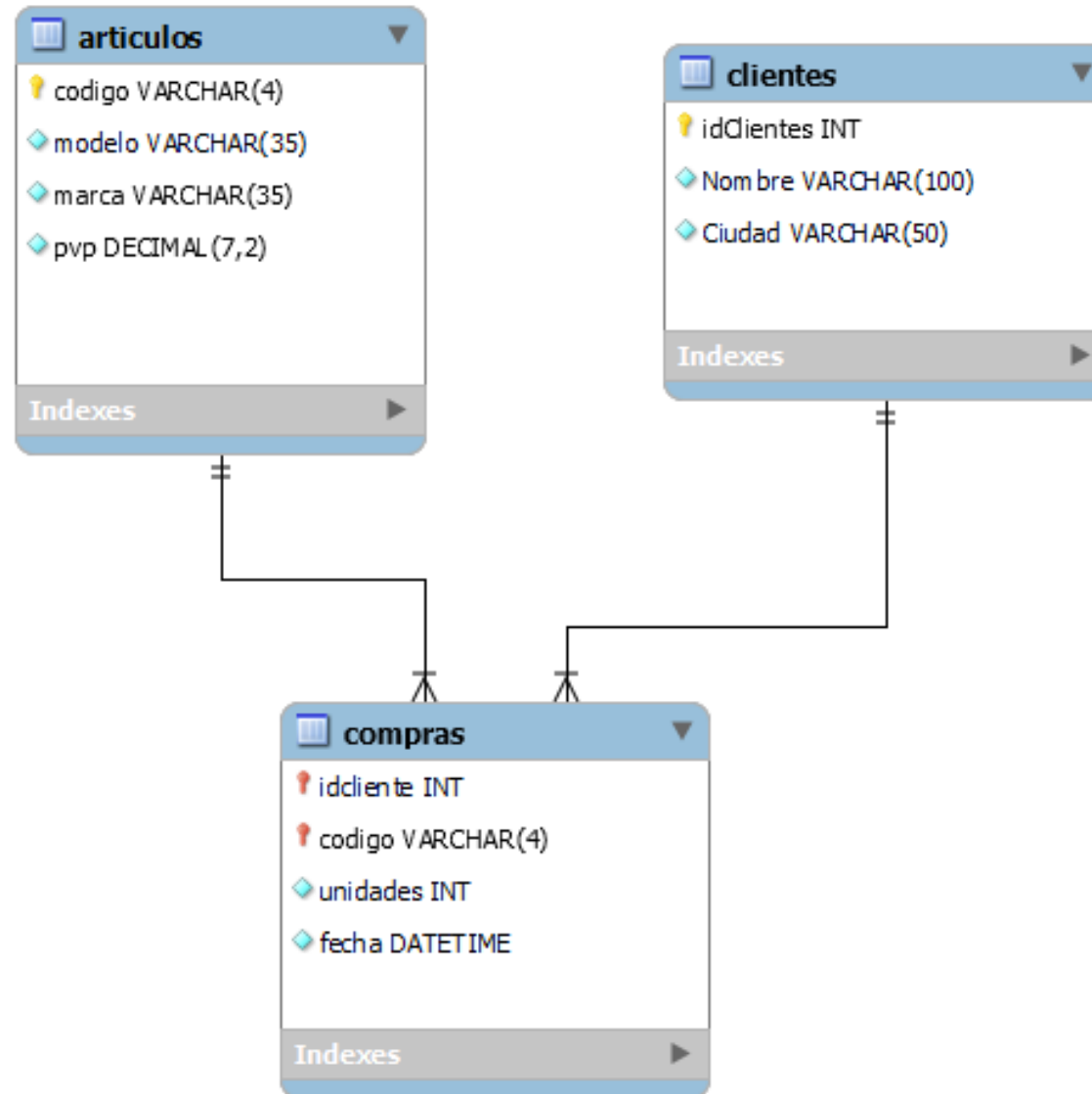
**Atributo ID único para cada autor**

```
<autor IdAutor="A2">  
  <Nombre>Fernando</Nombre>  
  <Apellidos>de Rojas</Apellidos>  
  <FechaNacimiento>01/01/1470</FechaNacimiento>  
</autor>
```

**Atributo IDREF que debe coincidir con uno de ID**

Puede afinarse más con una lista enumerada de posibles códigos de autores

# Modelo relacional a XML (con XML Schema)



## Modelo relacional a XML (con XML Schema)

```
<xs:element ref="clientes"/>
```

```
  <xs:element ref="cliente" minOccurs="0" maxOccurs="unbounded"/>
```

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

```
    <xs:element name="ciudad" type="tipoCiudad"/>
```

```
    <xs:attribute name="idcliente" use="required" type="xs:ID"/>
```

02ejemplo.xml

02ejemplo.xsd

```
<xs:element ref="articulos"/>
```

```
  <xs:element ref="articulo" minOccurs="0" maxOccurs="unbounded"/>
```

```
    <xs:element name="modelo" type="tipoModelo"/>
```

```
    <xs:element name="marca" type="tipoMarca"/>
```

```
    <xs:element name="pvp" type="tipoPvp"/>
```

```
    <xs:attribute name="codigo" use="required" type="xs:ID"/>
```

```
<xs:element ref="compras"/>
```

```
  <xs:element ref="compra" minOccurs="0" maxOccurs="unbounded"/>
```

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

```
    <xs:element name="fecha" type="xs:dateTime"/>
```

```
    <xs:attribute name="idcliente" use="required" type="xs:IDREF"/> <!-- clave ajena -->
```

```
    <xs:attribute name="codigo" use="required" type="xs:IDREF"/> <!-- clave ajena -->
```

## 2. JSON (*JavaScript Object Notation*)

Es un formato de texto pensado para el intercambio de datos. Su sintaxis está basada originalmente en la sintaxis de JavaScript, pero realmente es independiente de cualquier lenguaje de programación.

- Los datos están separados por comas.
- Los datos se escriban en pares, siendo primero el nombre o atributo del mismo y luego el valor del dato. Siempre entre comillas ""
- Los objetos JSON están rodeados por llaves {}
- Llaves cuadradas [] guardan *arrays*, incluyendo otros objetos JSON



- Los espacios en blanco y los saltos de línea no son significativos, es decir, puede haber cualquier número de espacios en blanco o saltos de línea separando cualquier elemento o símbolo del documento.
- Los ficheros JSON no pueden contener comentarios.
- Los valores (tanto en los objetos como en las matrices) pueden ser:
  - ❑ Números. El separador decimal es el punto (.)
  - ❑ Cadenas (*string*).
  - ❑ Objetos y matrices. Sin límite de anidamiento.

## Conversión XML / JSON

Podemos pasar de un formato a otro empleando, por ejemplo, aplicaciones *online* como <https://www.site24x7.com/es/tools/xml-a-json.html> o <https://www.utilities-online.info/jsontoxml>

Los elementos XML se convierten en claves de JSON y los valores de los elementos se transforman en los valores JSON correspondientes.	prueba1.xml prueba1.json
---	-----------------------------

Los atributos XML se convierten a las claves de JSON correspondientes con el prefijo “-”.	prueba2.xml prueba2.json
---	-----------------------------

### 3. Introducción a **XQuery** / **XPath**

SQL es el lenguaje que en la actualidad se considera estándar de facto porque la mayoría de los SGBD lo implementan.

La versión 2006 establece una mayor integración con los documentos XML, permitiendo importar y exportar datos en XML.

Si vamos a trabajar con documentos XML con información bajo un modelos relacional se necesitará algún lenguaje que permita extraer y manipular información de igual manera que SQL con las BD relacionales. Este lenguaje se llama **XQuery**.

Cuando se analiza un documento XML se crea un árbol de nodos del mismo. Ese árbol tiene un elemento raíz y una serie de hijos. Los hijos del nodo raíz pueden tener más hijos. Si repetimos ese proceso llegará un momento en el que el último nodo no tiene ningún hijo, lo que se denomina nodo hoja.

La información que buscaremos recorriendo el árbol podrá ser un nodo concreto, algún atributo, etc.

**XPath** es la herramienta que utiliza XQuery para procesar el árbol de nodos.

<https://basex.org/download/>

Sistema de administración de bases de datos XML nativo, liviano y un procesador XQuery , desarrollado como un proyecto comunitario en GitHub. Está especializado en almacenar, consultar y visualizar grandes colecciones y documentos XML.

# Xpath.

Líneas separadas por comas (,)

*Orden1,*

*Orden2*

**Texto literal va entre comillas ""**

*"Base de datos Libros",*

*"1.- Listado solicitado"*

*Base de datos Libros*

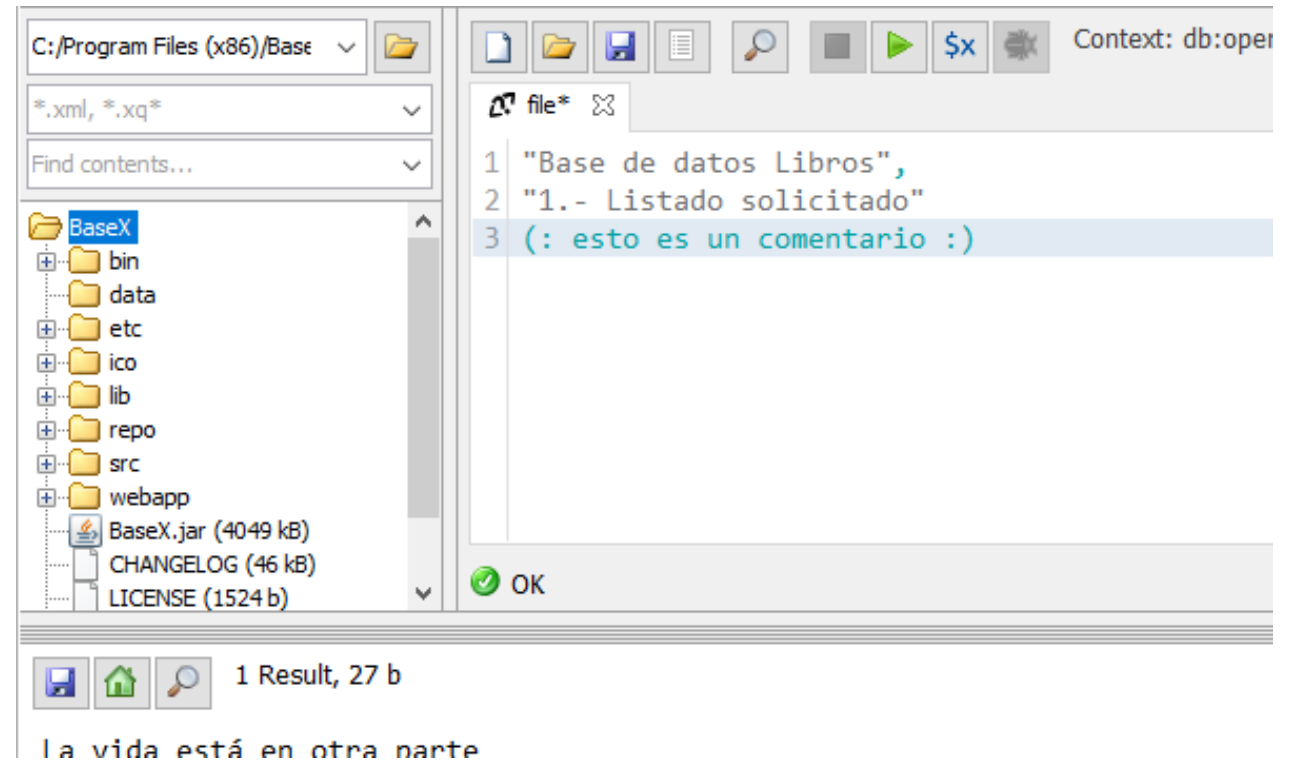
*1.- Listado solicitado*

**Comentario entre (: :)**

(: hola comentario :)

*No muestra nada*

La aplicación [basex.org](http://basex.org) nos permite plantear solicitudes XPath y XQuery a una serie de documentos XML



## Xpath.

/: si está al principio de la expresión, indica el nodo raíz, si no, indica "hijo". Debe ir seguida del nombre de un elemento.

*/biblioteca/libro/autor*

*<autor>Milan Kundera</autor>*

*<autor>Mario Vargas Llosa</autor>*

*<autor>Mario Vargas Llosa</autor>*

*/autor*

No devuelve nada porque <autor> no es hijo del nodo raíz.

*/biblioteca/libro/fechaPublicacion/@año*

*año="1973"*

*año="1963"*

*año="1995"*

**La aplicación basex.org nos permite plantear solicitudes XPath y XQuery a una serie de documentos XML**

03ejemplo.xml

**//: indica "descendiente" (hijos, hijos de hijos, etc.).**

//autor

*<autor>Milan Kundera</autor>*

*<autor>Mario Vargas Llosa</autor>*

*<autor>Arturo Pérez-Reverte</autor>*

//autor//libro

No devuelve nada porque <libro>

no es descendiente de <autor>

//@año

*año="1973"*

*año="1963"*

*año="1995"*

**text() muestra el contenido del nodo**

//autor/text()

Milan Kundera

Mario Vargas Llosa

Arturo Pérez-Reverte

**data() muestra el valor del atributo o contenido del nodo**

//@año/data()

1973

1963

1995

//@año/./..

<libro>

<titulo>La vida está en otra parte</titulo>

<autor>Milan Kundera</autor>

<fechaPublicacion año="1973"/>

</libro>

<libro>

<titulo>La ciudad y los perros</titulo>

<autor>Mario Vargas Llosa</autor>

<fechaPublicacion año="1963"/>

</libro>

<libro>

<titulo>La piel del tambor</titulo>

<autor>Arturo Pérez-Reverte</autor>

<fechaPublicacion año="1995"/>

</libro>

|: permite indicar varios recorridos.

//autor|//titulo|//@año

<titulo>La vida está en otra parte</titulo>

<autor>Milan Kundera</autor>

año="1973"

<titulo>La ciudad y los perros</titulo>

<autor>Mario Vargas Llosa</autor>

año="1963"

<titulo>La piel del tambor</titulo>

<autor>Arturo Pérez-Reverte</autor>

año="1995"

/biblioteca/libro/autor|//titulo|//@año



Observa la diferencia entre estos dos scripts:

//titulo, //autor

```
<titulo>La vida está en otra parte</titulo>  
<titulo>La ciudad y los perros</titulo>  
<titulo>La piel del tambor</titulo>  
<autor>Milan Kundera</autor>  
<autor>Mario Vargas Llosa</autor>  
<autor>Arturo Pérez-Reverte</autor>
```



Primero una orden y luego la otra

//titulo | //autor

```
<titulo>La vida está en otra parte</titulo>  
<autor>Milan Kundera</autor>  
<titulo>La ciudad y los perros</titulo>  
<autor>Mario Vargas Llosa</autor>  
<titulo>La piel del tambor</titulo>  
<autor>Arturo Pérez-Reverte</autor>
```



Va mostrando los dos caminos en paralelo

//fechaPublicacion/@año

año="1973"

año="1963"

año="1995"

[**número**]: si hay varios resultados selecciona uno de ellos por número de orden; **last()** selecciona el último de ellos

//libro[1]

<libro>

<titulo>La vida está en otra parte</titulo>

<autor>Milan Kundera</autor>

<fechaPublicacion año="1973"/>

</libro>

//libro[**last()**]

<libro>

<titulo>La piel del tambor</titulo>

<autor>Arturo Pérez-Reverte</autor>

<fechaPublicacion año="1995"/>

</libro>

//libro[**last()-1**]

<libro>

<titulo>La ciudad y los perros</titulo>

<autor>Mario Vargas Llosa</autor>

<fechaPublicacion año="1963"/>

</libro>

**[condicion]:** selecciona los nodos que cumplen la condición.

operadores lógicos: and, or, not()

operadores aritméticos: +, -, \*, div, mod

operadores de comparación: =, !=, <, >, <=, >=

*//fechaPublicacion[@año>1970]*

*<fechaPublicacion año="1973"/>*

*<fechaPublicacion año="1995"/>*

Para hacer referencia al propio  
valor del elemento seleccionado se  
utiliza el punto (.)

*//@año[.>1970]*

*año="1973"*

*año="1995"*

*//libro[fechaPublicacion/@año>1970]*

*<libro>*

*<titulo>La vida está en otra parte</titulo>*

*<autor>Milan Kundera</autor>*

*<fechaPublicacion año="1973"/>*

*</libro>*

*<libro>*

*<titulo>La piel del tambor</titulo>*

*<autor>Arturo Pérez-Reverte</autor>*

*<fechaPublicacion año="1995"/>*

*</libro>*

Un predicado puede contener condiciones compuestas.

```
//libro[autor="Milan Kundera" and
fechaPublicacion/@año="1973"]
```

<libro>

<titulo>La vida está en otra parte</titulo>

<autor>Milan Kundera</autor>

<fechaPublicacion año="1973"/>

</libro>

```
//libro[autor="Mario Vargas Llosa" and
fechaPublicacion/@año="1963"]/autor
```

/@\*: selecciona todos los atributos del nodo.

//@\*: selecciona todos los atributos de los descendientes del nodo.

//@\*

año="1973"

año="1963"

año="1995"

/\*: selecciona todos los hijos (sólo elementos) del nodo.

/biblioteca/\*

<libro>

<titulo>La vida está en otra parte</titulo>

<autor>Milan Kundera</autor>

<fechaPublicacion año="1973"/>

</libro>

<libro>

<titulo>La ciudad y los perros</titulo>

<autor>Mario Vargas Llosa</autor>

<fechaPublicacion año="1963"/>

</libro>

<libro>

<titulo>La piel del tambor</titulo>

<autor>Arturo Pérez-Reverte</autor>

<fechaPublicacion año="1995"/>

</libro>

En general, **XPath** es un lenguaje utilizado para identificar los nodos XML exactos en un DOM.

**XQuery** es un superconjunto (extensión) de XPath que también proporciona la sintaxis **FLWOR** , que es similar a SQL.

FLWOR: **F**or, **L**et, **W**here, **O**rders By, **R**eturn

Carpeta Libreria

04Libros.xml

04Libros.dtd

04Comentarios.xml

04Comentarios.dtd

## 04aXQuery.xq

(:comentario que no se ejecuta:)

"1 - Título de los libros",

/bib/libro/titulo,

"2 - Título de los libros >50euros",

/bib/libro[precio>50]/titulo,

\*\*\*\*\* Expresiones FLWOR \*\*\*\*\*",

"3.- Título de los libros ",

for \$a in /bib/libro/titulo

return \$a,

"4.- Título de los libros >50euros ",

for \$a in /bib/libro

where \$a/precio > 50

return \$a/titulo,

"5.- Título de los libros >50euros y editorial sea

Addison-Wesley ",

for \$a in /bib/libro

where \$a/precio > 50 and \$a/editorial

="Addison-Wesley"

return \$a/titulo

## 04bXQuery.xq

"1.- Mostrar el atributo año de cada libro",

for \$lib in //libro return \$lib/@año,

"2.- idem pero sólo mostrando los valores del atributo año",

for \$lib in //libro return \$lib/@año/data(),

"3.- idem anterior pero con función string",

for \$lib in //libro return string(\$lib/@año),

"4.- idem anterior pero sólo año menor que 2000 y ordenado por año",

for \$lib in //libro

where \$lib/@año < 2000

order by \$lib/@año

return string(\$lib/@año)

data() es similar a text() pero vale para elementos y para atributos

string es un función que recibe un elemento o atributo y muestra su contenido o valor, respectivamente

## 04cXQuery.xq

(:comentario que no se ejecuta:)

"1.- Mostrar el título y precio de cada libro",

```
for $lib in //libro/titulo/text()|//libro/precio/text()
return $lib,
```

"2.- Mostrar autores con un título que incluya la palabra Web",

```
for $lib in //libro
where contains($lib/titulo, "Web")
return $lib/autor,
```

"3.- Mostrar todos los títulos de libros menos los del año 2000 y 1992",

```
for $lib in //libro
where $lib/@año != 2000 and $lib/@año != 1992
return $lib/titulo/text(),
```

"4.- Muestra los títulos de los libros anteriores al 2000 ordenados por años incluyendo etiquetas HTML",

```
for $lib in //libro
where $lib/@año < 2000
order by $lib/@año
return <LIBRO>{$lib/@año}{$lib/titulo/data()} </LIBRO>
```

"5.- Mostrar apellido, nombre dentro de una etiqueta PERSONA ordenado descendientemente por apellido ",

```
for $lib in doc("04Libros.xml")//autor
order by $lib/apellido descending
return <PERSONA>{$lib/apellido/data()},
{$lib/nombre/data()}</PERSONA>
```

Especificamos el fichero concreto.  
Realmente si no hay posibilidad de confusión no es necesario

## 04dXQuery.xq

(:Diferencia entre for y let :)

**for** \$a in //autor

**return** <PERSONAS>{\$a/apellido}</PERSONAS>

<PERSONAS>

<apellido>Suciu</apellido>

</PERSONAS>

<PERSONAS>

<apellido>Stevens</apellido>

</PERSONAS>

<PERSONAS>

<apellido>Stevens</apellido>

</PERSONAS>

<PERSONAS>

<apellido>Buneman</apellido>

</PERSONAS>

<PERSONAS>

<apellido>Abiteboul</apellido>

</PERSONAS>

(:Diferencia entre for y let :)

**let** \$a := //autor

**return** <PERSONAS>{\$a/apellido}</PERSONAS>

<PERSONAS>

<apellido>Stevens</apellido>

<apellido>Stevens</apellido>

<apellido>Abiteboul</apellido>

<apellido>Buneman</apellido>

<apellido>Suciu</apellido>

</PERSONAS>

Con **let** almacenamos //autor en una variable única no como con **for** que van iterando las tuplas.



## 04eXQuery.xq

```

"Combinando for y let",
"1.-Uso de count",
for $a in //libro
let $b := $a/autor
return
<LIBRO>{$a/titulo, <NUMAUTORES>{count
($b)}}</NUMAUTORES>}</LIBRO>,
"2.-Listar año y título de los libros que tienen más de un autor",
for $a in //libro
where count($a/autor)>1
return <LIBRO>{$a/@año}{$a/titulo/text()}</LIBRO>,
"*****",
"3.- Muestra los comentarios de cada libro. Dos archivos",
for $t in doc("04Libros.xml")//titulo, $e in
doc("04Comentarios.xml")//entrada
where $t = $e/titulo
return <APUNTE>{$t, $e/comentario}</APUNTE>

```

Con for reiteramos cada uno de los libros. **\$a**  
 Con let metemos en una sola variable autor que  
 tiene nombre y apellido de cada libro. **\$b**  
 Con **count(\$b)** contamos los autores que tiene cada  
 libro.

**En return si queremos incluir resultados  
dentro de elementos XML o HTML  
deben ir entre {}**

Separados por comas (,) podemos  
indicar que acumule en diferentes  
variables distintos elementos.


Podemos comparar valores de  
elementos o identificadores de  
diferentes archivos.

05ejemplo.xml


05aXQuery.xq

Ejemplo donde podemos ver como movernos por el árbol XML buscando coincidencias entre las claves propias y externas.


```
for $com in //compra, $art in //articulo, $cli in //cliente  
where $com/@idcliente = $cli/@idcliente and $com/@codigo = $art/@codigo  
return  
($art/modelo/text(),$art/marca/text(),$cli/nombre/text(),$com/fecha/text() )
```



EOS 550d  
CANON  
MANUEL FERNÁNDEZ HERNÁNDEZ  
2019-09-03T14:00:00



PX730  
EPSON  
JUANA GÓMEZ GÓMEZ  
2019-06-09T12:35:41



PX730  
EPSON  
MANUEL FERNÁNDEZ HERNÁNDEZ  
2019-07-13T11:30:40

## 4. XQuery y HTML

En el fichero Xquery (.xq) podemos incluir toda la estructura de HTML5/CSS3 (si se quiere con fichero enlazado).

Las llamadas Xquery (formato FLWOR) tendrán que ir correctamente **incrustadas entre {}**

```
<html><head><link rel="stylesheet" type="text/css" href="05bXQuery.css"/>
<title>XQuery y HTML</title></head>
<body><table>
  <tr><th>Modelo</th><th>Marca</th><th>Cliente</th><th>Fecha</th></tr>
  {
    for $com in //compra, $art in //articulo, $cli in //cliente
    where $com/@idcliente = $cli/@idcliente and $com/@codigo = $art/@codigo
    return <tr>
      <td>{$art/modelo/text()}</td>
      <td>{$art/marca/text()}</td>
      <td>{$cli/nombre/text()}</td>
      <td>{$com/fecha/text()}</td>
    </tr>
  }
</table></body></html>
```

05ejemplo.xml

05bXQuery.xq

05bXQuery.html

05bXQuery.css

Modelo	Marca	Cliente	Fecha
EOS 550d	CANON	MANUEL FERNÁNDEZ HERNÁNDEZ	2019-09-03T14:00:00
PX730	EPSON	JUANA GÓMEZ GÓMEZ	2019-06-09T12:35:41
PX730	EPSON	MANUEL FERNÁNDEZ HERNÁNDEZ	2019-07-13T11:30:40