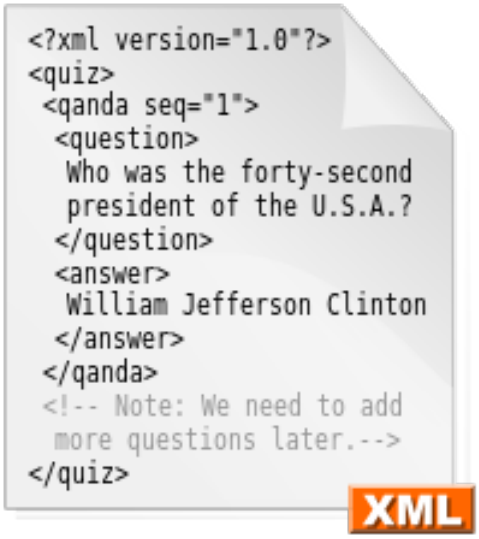

Bases de datos con XML

versión: 2022.1

1 de marzo de 2022

Índice

1. Modelo semiestructurado de bases de datos **2**
1.1. Lenguaje XML . . . 2
1.2. Bases de datos con XML . . . 3

2. Lenguaje XML **3**

3. Documentos bien formados **5**
3.1. Document Type Definition . . . 6

4. Lenguajes para bases de datos de XML **12**
4.1. XPath . . . 12
4.1.1. Sintaxis . . . 12
4.1.2. Ejemplos . . . 13
4.2. XQuery . . . 14
4.2.1. Expresiones FLWOR . . . 14
4.2.2. Cuantificadores `some` y `every` . . . 15
4.2.3. Constructoras de elementos . . . 15
4.2.4. Acceso al contenido de un elemento . . . 16
4.2.5. Consultas anidadas . . . 17
4.2.6. XQuery y HTML . . . 17
4.3. SGBD eXist . . . 18

1. Modelo semiestructurado de bases de datos

- Modelos lógicos de base de datos:
  - Modelo relacional (el más usado).
  - Modelo jerárquico.
  - Modelo de red.
  - Modelo orientado a objetos.
  - Modelo semiestructurado (XML).
  - Y alguno más.
- Características generales del modelo semiestructurado:
  - Se pueden ver como una relajación de algunas de las características del modelo relacional.
  - Algunas entidades permiten la omisión de información en ciertos atributos.
  - Existen distintos tipos posibles para un mismo atributo.
  - Algunos atributos pueden no disponer de una estructura predefinida o no ser atómicos.
- Estándares para manejar datos semiestructurados:
  - OEM: *Object Exchange Model*.
  - XML:*eXtensible Markup Language*.
  - JSON: *JavaScript Object Notation*.
- Ventajas e inconvenientes:
  - Mayor flexibilidad en la representación de los datos.
  - Consultas y modificaciones más ineficientes que en el modelo relacional.

1.1. Lenguaje XML

- Es un lenguaje de marcado (*markup*).
- Permite generar documentos con anotaciones legibles por una persona.
- Definido como estándar por el W3C (*World Wide Web Consortium*).
- Aspecto parecido a HTML, con una diferencia importante:
  - HTML: Define la estructura de las páginas Web.
  - XML: Define una estructura de datos arbitraria.
- XML no se concibió inicialmente para modelizar bases de datos.
- XML se utiliza normalmente como lenguaje común de intercambio de datos entre sistemas heterogéneos.
- Ventajas:
  - Legibilidad.
  - Representación jerárquica de la información.
  - Numerosos intérpretes de XML disponibles.
- Inconvenientes:

    - Ineficiencia en espacio y tiempo.
- Algunos formatos de archivo basados en XML:
    - XHTML : *Extensible Hypertext Markup Language.*
    - SVG : *Scalable Vector Graphics.*
    - MathML : *Mathematical Markup Language.*
    - X3D : *Extensible 3D Graphics.*
    - ODF : *Open Document Format (archivos .odt).*
    - OOXML : *Office Open XML (archivos .docx).*
    - WSDL : *Web Services Description Language.*

1.2. Bases de datos con XML

- Una base de datos XML es un sistema que almacena datos XML de manera persistente.
- Sistemas gestores de bases de datos XML nativos:
    - BaseX (`http://basex.org/`)
    - eXistdb (`http://www.exist-db.org/`)
    - Sedna (`http://sedna.org/`)
- Existe un lenguaje estándar para realizar consultas sobre las bases de datos XML: XQuery.

2. Lenguaje XML

- Ejemplo de código XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<agenda>
<contacto dni="51233412H">
    <nombre>David</nombre>
    <apellidos>Álvez Campos</apellidos>
    <dirección>
        <calle>Paseo de Ondarreta</calle>
        <numero>5</numero>
        <codigo-postal>20018</codigo-postal>
        <localidad>San Sebastián</localidad>
    </dirección>
    <telefono tipo="casa">943102321</telefono>
    <telefono tipo="movil">617702341</telefono>
</contacto>
<contacto dni="46821354T">
    <nombre>Víctor</nombre>
    <apellidos>Martín Moreno</apellidos>
    <telefono tipo="casa">914621100</telefono>
</contacto>
</agenda>
```

- La cabecera de un documento XML

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

está delimitada por los símbolos `<?` y `?>`

  - Indica que el documento se corresponde con la versión 1.0 del estándar de XML y que utiliza la codificación UTF-8.
  - La cabecera también puede contener otras declaraciones, tales como referencias al DTD o XML Schema asociado al documento XML, e instrucciones de procesamiento.

- Los elementos de un archivo XML están compuestos de una etiqueta de inicio, una etiqueta de fin, y un contenido:

```
<apellidos>Martín Moreno</apellidos>
```

- Sintaxis de los elementos:
  - Los nombres de las etiquetas son identificadores formados por letras, números y caracteres de guión/subrayado. Deben comenzar por una letra.
  - Un documento XML ha de contener un elemento raíz que contenga a los demás.
  - Si un elemento no tiene contenido (no contiene otras etiquetas), puede utilizarse una sintaxis alternativa:

```
<telefono tipo="casa" num="913102321" />
```

- Atributos:
  - Cada elemento puede contener cero, uno o más atributos.
  - Los atributos se colocan en la etiqueta de inicio del elemento al que van asociados. Cada atributo consta de un nombre y un valor.
  - Las reglas para el nombre de un atributo son las mismas que para los nombres de etiquetas. El valor ha de estar delimitado entre comillas simples (') o comillas dobles (").

- Comentarios: delimitados por `<!--` y `-->`.

```
<!-- Esto es un comentario -->
```

- Entidades: sirven para representar caracteres que tienen un significado especial en XML (forman parte de la sintaxis), tales como los símbolos `<`, `>`. Comienzan por `&` y terminan por `;`

Entidad	Símbolo
&amp;	&
&lt;	<
&gt;	>
&quot;	"
&apos;	'

- Secciones CDATA: Sirven para expresar contenido que contenga caracteres especiales (<, >, etc.), sin necesidad de utilizar entidades. Están delimitadas por `<![CDATA[` y `]]>`. Ejemplo:

```
<código-fuente><![CDATA[
if (x > 0 && x <= 10) then print("Hola")
]]></código-fuente>
```

Sin secciones CDATA:

```
<código-fuente>
if (x &gt; 0 &amp;amp; x &lt;= 10) then
print("Hola")
</código-fuente>
```

- Un pequeño dilema: la información como elemento o como atributos:
  - No existe ninguna regla general que indique cómo representar la información, pero ha de tenerse en cuenta lo siguiente.
  - Como atributos:
    - Para valores que son atómicos.
    - Ocupan menos espacio (no hay etiquetas de inicio/cierre).
    - Son más adecuados para claves primarias y externas.
  - Como elementos:
    - Cuando su información puede ser compuesta (varios elementos).
    - Permiten agrupar varios elementos del mismo o distinto tipo.
    - Son recomendables cuando el valor es muy extenso, o requiere de una sección CDATA.

3. Documentos bien formados

- Un documento está bien formado si cumple las normas básicas de XML a nivel sintáctico:
  - Tiene una cabecera.
  - Tiene un único elemento raíz.
  - Las etiquetas de los elementos están correctamente anidadas.
  - Los valores de los atributos se encuentran delimitados por comillas simples o dobles.
- Los documentos bien formados pueden ser interpretados por cualquier librería o herramienta de manipulación de documentos XML.
- En muchas ocasiones es deseable especificar restricciones adicionales:
  - Etiquetas permitidas/prohibidas.
  - Atributos permitidos/obligatorios.
  - Tipo de contenido: entero, cadena, etc.
- Existen tecnologías que permiten concretar el contenido de un fichero XML más allá de una simple comprobación sintáctica:
  - Document type definition (DTD).
  - XML Schema.
  - Relax NG (no es estándar W3C).
- Un documento XML es válido con respecto a una DTD (o un `Schema`) si está bien formado y su contenido se adecua a las restricciones impuestas por dicho DTD (o `Schema`).

3.1. Document Type Definition

- Una DTD (*Document Type Definition*) es un conjunto de declaraciones que definen los elementos y atributos que pueden aparecer en un documento determinado.
- Componentes:
  - Declaraciones `<!ELEMENT>`.
  - Declaraciones `<!ATTLIST>`.
  - Declaraciones `<!ENTITY>` (permiten especificar abreviaturas, no las usaremos).
- Declaraciones `ELEMENT`:
  - Permiten definir los nombres de elementos (etiquetas) permitidos en un documento XML.
  - Sintaxis:

```
<!ELEMENT nombre contenido>
```

    Ejemplo:

```
<!ELEMENT dirección (calle, número, código-postal, localidad)>
```

    define un elemento `<dirección>` que contiene los siguientes elementos: `<calle>`, `<número>`, `<códigopostal>` y `<localidad>`.

  - Podemos expresar el contenido de un elemento mediante una secuencia de elementos hijo:

```
<!ELEMENT nombre (elem-1, elem-2, ..., elem-n)>
```

    indica que el elemento `<nombre>` ha de incluir los elementos `<elem-1>`, `<elem-2>`, ..., `<elem-n>`, y en el orden indicado. Esta especificación permite la siguiente estructura:

```
<nombre>
    <elem-1> ... </elem-1>
    <elem-2> ... </elem-2>
    ...
    <elem-n> ... </elem-n>
</nombre>
```

  - Podemos utilizar el operador (|) para expresar distintas alternativas en el contenido de un elemento:

```
<!ELEMENT nombre (elem-1 | elem-2 | ... | elem-n)>
```

  - Esta vez el elemento `<nombre>` debe incluir `<elem-1>`, o bien, `<elem-2>`, ..., o bien, `<elem-n>`. Ejemplo:

```
<!ELEMENT posición (dirección | coordenadas)>
```

    Este código permite:

```
<posición>
<dirección> ... </dirección>
</posición>
```

    o:

```
<posición>
<coordenadas> ... </coordenadas>
</posición>
```

- Es posible mezclar secuencias con alternativas. Ejemplo:

```
<!ELEMENT posición (dirección | (latitud,longitud) )>
```

permite:

```
<posición>
<dirección> ... </dirección>
</posición>
<posición>
<latitud> ... </latitud>
<longitud> ... </longitud>
</posición>
```

- Para indicar que un elemento sólo puede contener texto en su interior, utilizamos #PCDATA. Ejemplo:

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

```
<calle>Avenida de Portugal</calle>
```

- La palabra clave #PCDATA puede utilizarse dentro de una secuencia, o combinada con otras alternativas:

```
<!ELEMENT descripción (#PCDATA | (titulo, detalles))>
```

- Se pueden especificar reglas de cardinalidad dentro del contenido de un elemento.
- En ausencia de reglas de cardinalidad los elementos de una secuencia deben aparecer una y sólo una vez.
- Los operadores `?`, `*`, `+` permiten expresar otras reglas de cardinalidad:

Operador	Significado
?	El elemento puede no aparecer o aparecer una sóla vez
+	El elemento puede aparecer una o más veces
*	El elemento puede aparecer cero, una, o más veces

- Ejemplo:

```
<!ELEMENT contacto (nombre, apellidos, dirección?, telefono*)>
```

permite:

```
<contacto dni="51233412H">
<nombre>David</nombre>
<apellidos>Álvez Campos</apellidos>
<dirección> ... </dirección>
<telefono tipo="casa">943102321</telefono>
<telefono tipo="movil">617702341</telefono>
</contacto>
```

```
<contacto dni="51233412H">
<nombre>David</nombre>
<apellidos>Álvez Campos</apellidos>
</contacto>
```

- Otro ejemplo:

```
<!ELEMENT body (#PCDATA | b | i)* >
```

  Define una etiqueta `<body>` que puede contener en su interior texto intercalado con los elementos `<b>` e `<i>`.

- Otros tipos de contenido:
  - `EMPTY`: el elemento no contiene nada en su interior.
  - `ANY`: libertad total en el contenido.
- Más ejemplos:

```
<!ELEMENT producto EMPTY>
```

  permite:

```
<producto id="e34" cantidad="2" />
```

```
<!ELEMENT descripción ANY>
```

  permite:

```
<descripción>
<titulo>Descripción del objeto</titulo>
Este es el contenido de la descripción.
</descripción>
```

- Declaraciones ATTLIST:
  - Especifican los atributos que pueden ir asociados a un elemento. Sintaxis:

```
<!ATTLIST elemento atributo tipo valor>
```

    donde:
    - *elemento* indica la etiqueta en la que se adjunta el atributo.
    - *atributo* es el nombre del atributo.
    - *tipo* indica el conjunto de valores que puede tener el atributo.
    - *valor* hace referencia al valor por defecto y a la obligatoriedad de incluir el atributo.
  - *tipo* puede ser uno de los siguientes:
    - *CDATA*: Cadena de caracteres (Tipo por defecto).
    - *Lista de valores*: Valores posibles para el atributo.
    - *ID*: Identificador único. Dos elementos no pueden tener el mismo valor para este atributo.
    - *IDREF*: Referencia al identificador de un elemento.
    - *IDREFS*: Lista de referencias a identificadores de otros elementos, separadas por espacios.
  - *valor* puede ser uno de los siguientes:

- *"Valor"*: El atributo es opcional. Si no se indica, tomará el valor indicado.
- *#FIXED "Valor"*: El atributo es obligatorio y ha de tener el valor indicado.
- *#REQUIRED*: El atributo es obligatorio.
- *#IMPLIED*: El atributo es opcional.

- Se pueden declarar varios atributos con el mismo nombre, siempre que pertenezcan a elementos distintos.
- En los elementos que tienen tipos ID, IDREF, IDREFS, los identificadores tienen que adecuarse a la sintaxis XML.
- En particular, han de comenzar por carácter alfabético.
- Ejemplos:

```
<!ATTLIST telefono tipo
(casa | trabajo | movil) "casa">
<telefono tipo="movil"> ... </telefono>
<!ATTLIST contacto dni CDATA #REQUIRED>
<contacto dni="51233412H">
...
</contacto>
```

`contactos.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<agenda>
<contacto dni="51233412H">
<nombre>David</nombre>
<apellidos>Álvez Campos</apellidos>
<dirección>
<calle>Paseo de Ondarreta</calle>
<numero>5</numero>
<código-postal>20018</código-postal>
<localidad>San Sebastián</localidad>
</dirección>
<telefono tipo="casa">943102321</telefono>
<telefono tipo="movil">617702341</telefono>
</contacto>
<contacto dni="46821354T">
<nombre>Víctor</nombre>
<apellidos>Martín Moreno</apellidos>
<telefono tipo="casa">914621100</telefono>
</contacto>
</agenda>
```

`contactos.dtd`:

```
<!ELEMENT agenda (contacto)+>
<!ELEMENT contacto (nombre, apellidos, dirección?, telefono*)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellidos (#PCDATA)>
<!ELEMENT dirección (calle, numero, código-postal, localidad)>
<!ELEMENT calle (#PCDATA)>
<!ELEMENT numero (#PCDATA)>
<!ELEMENT código-postal (#PCDATA)>
<!ELEMENT localidad (#PCDATA)>
<!ELEMENT telefono (#PCDATA)>
```

```
<!ATTLIST telefono tipo (casa | trabajo | movil) #REQUIRED>
<!ATTLIST contacto dni CDATA #REQUIRED>
```

- Asociar una DTD a un documento:
  - Se utiliza la declaración `<!DOCTYPE>` en la cabecera del documento XML.
  - Para DTDs incluidas en el documento XML:

```
<!DOCTYPE elem-raíz [
... declaraciones DTD ...
] >
```

  - Para DTDs separadas en un archivo externo:

```
<!DOCTYPE elem-raíz SYSTEM "archivo.dtd">
```

  - Ejemplo de DTD interna:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE agenda [
<!ELEMENT agenda (contacto)+>
<!ELEMENT contacto (nombre, apellidos, dirección?, telefono*)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellidos (#PCDATA)>
<!ELEMENT dirección (calle, numero, código-postal, localidad)>
<!ELEMENT calle (#PCDATA)>
<!ELEMENT numero (#PCDATA)>
<!ELEMENT código-postal (#PCDATA)>
<!ELEMENT localidad (#PCDATA)>
<!ELEMENT telefono (#PCDATA)>
<!ATTLIST telefono tipo (casa | trabajo | movil) #REQUIRED>
<!ATTLIST contacto dni CDATA #REQUIRED>
]>
<agenda>
<contacto dni="51233412H">
...
</contacto>
<contacto dni="46821354T">
...
</contacto>
</agenda>
```

  - Ejemplo de DTD externa:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE agenda SYSTEM "contactos.dtd">
<agenda>
<contacto dni="51233412H">
<nombre>David</nombre>
<apellidos>Álvez Campos</apellidos>
<dirección>
<calle>Paseo de Ondarreta</calle>
<numero>5</numero>
<código-postal>20018</código-postal>
<localidad>San Sebastián</localidad>
</dirección>
```

```
<telefono tipo="casa">943102321</telefono>
<telefono tipo="movil">617702341</telefono>
</contacto>
<contacto dni="46821354T">
<nombre>Víctor</nombre>
<apellidos>Martín Moreno</apellidos>
<telefono tipo="casa">914621100</telefono>
</contacto>
</agenda>
```

- Ejemplo de una tienda: `tienda.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tienda SYSTEM "Tienda.dtd">
<tienda>
<articulo id = "c01">
<nombre>Aceite Virgen Extra Serie ORO</nombre>
<cantidad>500 ml</cantidad>
<precio divisa = "EUR">8.36</precio>
<precio divisa = "GBP">7.17</precio>
</articulo>
<articulo id = "c02">
<nombre>Aceite Virgen Extra</nombre>
<cantidad>1000 ml</cantidad>
<precio divisa = "EUR">7.50</precio>
<precio divisa = "GBP">6.43</precio>
</articulo>
<lista-compra>
<articulos>
<producto id="c01" cantidad="2" descuento = "15"/>
<producto id="c02" cantidad="1"/>
</articulos>
<total divisa = "EUR">21.71</total>
</lista-compra>
</tienda>
```

`tienda.dtd`:

```
<!ELEMENT tienda (articulo*, lista-compra)>
<!ELEMENT articulo (nombre, cantidad, precio+)>
<!ELEMENT lista-compra (articulos, total)>
<!ELEMENT articulos (producto)*>
<!ELEMENT producto EMPTY>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT cantidad (#PCDATA)>
<!ELEMENT precio (#PCDATA)>
<!ELEMENT total (#PCDATA)>
<!ATTLIST articulo id ID #REQUIRED>
<!ATTLIST precio divisa CDATA "EUR">
<!ATTLIST total divisa CDATA "EUR">
<!ATTLIST producto id IDREF #REQUIRED>
<!ATTLIST producto cantidad CDATA "1">
<!ATTLIST producto descuento CDATA "0">
```

4. Lenguajes para bases de datos de XML

4.1. XPath

- Definido por el W3C en 1999.
- Se utiliza para seleccionar determinados nodos de un documento XML.
- Una expresión especifica una ruta (o rutas) donde se encuentra la información buscada.
- Estos nodos son relativos a un documento XML determinado, que se obtiene mediante la función `doc`, que recibe un nombre de archivo almacenado en la base de datos.

4.1.1. Sintaxis

- Elementos sintácticos:

Expresión	Significado
nodo	Selecciona todos los nodos cuyo nombre es *nodo*
/	Selecciona nodos desde la raíz
//	Selecciona nodos
`@`	Selecciona atributos
*	representa cualquier nodo
`@*`	representa cualquier atributo
`|`	permite sentencias compuestas (relacionadas por un AND)

Ejemplos:

- Consideremos un documento XML con las etiquetas: *bookstore*, *book*, *title* y *price*.
- `/bookstore`: selecciona el elemento raíz *bookstore*.
- `bookstore/book`: selecciona todos los elementos `book` hijos de *bookstore*.
- `//book`: selecciona todos los elementos *book* en cualquier parte del documento.
- `bookstore//book`: selecciona todos los elementos *book* descendientes de *bookstore*.
- `//@lang`: selecciona todos los atributos *lang*.
- `/bookstore/book[last()]`: selecciona el último elemento *book* hijo de *bookstore*.
- `/bookstore/book[last()-1]`: selecciona el penúltimo elemento *book* hijo de *bookstore*.
- `/bookstore/book[position()<3]`: selecciona los dos primeros hijos de *book* hijo de *bookstore*.
- `//title[@lang]`: selecciona todos los elementos *title* que tengan el atributo *lang*.
- `//title[@lang='en']`: selecciona todos los elementos *title* que tengan el atributo *lang* y su valor sea *en*.
- `/bookstore/book[price>35.00]`: selecciona los hijos de *book* cuyo texto para *price* sea mayor que 35.
- `/bookstore/book[price>35.00]/title`: selecciona *title* de los hijos de *book* cuyo texto para *price* sea mayor que 35.
- `/bookstore/*`: todos los hijos de *bookstore*.
- `//*`: todos los elementos.
- `//title[@*]`: todos los elementos *title* que tengan un atributo (de cualquier tipo!!!).
- `//book/title | //book/price`: todos los títulos y precios de cualquier elemento *book*.
- `//title | //price`: todos los títulos y precios.
- `/bookstore/book/title |//price`: todos los títulos de cualquier elemento *book* y todos los precios.

4.1.2. Ejemplos

Tomamos como referencia el siguiente documento XML:

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

<bookstore>

<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>

<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>

</bookstore>
```

Ejemplos:

- `doc(/db/prueba")/bookstore`: devuelve el elemento raíz del documento, que tiene como etiqueta `<bookstore>`.
- `doc(/db/prueba")/bookstore/book/title`: devuelve la lista de elementos `<title>` que sean hijos de `<book>`, que a su vez sean hijos de `<bookstore>`.
- `doc(/db/prueba")/bookstore/*/title`: devuelve la lista de elementos `<title>` que sean hijos de hijos de `<bookstore>` (nietos).
- `doc(/db/prueba")/bookstore//title`: devuelve la lista de elementos `<title>` que sean descendientes de `<bookstore>`.
- `doc(/db/prueba")/bookstore/book[3]/price`: muestra el contenido de `<price>` para `<book[3]>`.

4.2. XQuery

- Estándar del W3C que se apoya en XPath para realizar consultas más complejas en bases de datos XML.
- Es (o pretende ser) para XML lo que es SQL para bases datos relacionales.
- Se construye sobre expresiones del lenguaje XPath.
- Utiliza expresiones FLWOR (*For, Let, Where, Order by, Return*):
  - `For`: selecciona una secuencia de nodos.
  - `Let`: enlaza los elementos de esta secuencia a una variable.
  - `Where`: filtra los elementos de la secuencia.
  - `Order by`: ordena los elementos de la secuencia.
  - `Return`: construye la respuesta.
- Ejemplo:

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

  - `doc("books.xml")/bookstore/book`: contiene un conjunto de etiquetas.
  - `where $x/price>30`: permite obtener un subconjunto del anterior.
  - `x`: representa cada uno de los elementos de este conjunto. En este caso de etiquetas `book`.
  - `x/title`: representa cada una de las etiquetas hijas de `x`.

  Para nuestro fichero anterior devolverá:

```
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

4.2.1. Expresiones FLWOR

- `for`:
  - Los nombres de variables comienzan por el símbolo $.
  - En una cláusula de la forma `for $x in seq` la variable $x toma cada uno de los valores de la secuencia *seq*.
  - Podemos acceder a los elementos contenidos en una variable mediante expresiones XPath.
  - Ejemplos:

```
for $b in doc("libros.xml")/libros/libro
    return $b/titulo
```

```
for $b in doc("libros.xml")/libros/libro
    return $b/autores/autor
```

- `let`: sirve para introducir definiciones auxiliares y hacer más legible el código. Ejemplo:

```
for $b in doc("libros.xml")/libros/libro
    let $titulo := $b/titulo
    return $titulo
```

- `order by`: ordena los resultados. Ejemplo:

```
for $b in doc("libros.xml")/libros/libro
    let $titulo := $b/titulo
    order by $titulo
    return $titulo
```

```
for $b in doc("libros.xml")/libros/libro
    let $titulo := $b/titulo descending
    order by $titulo
    return $titulo
```

- `where`: filtra los resultados. Ejemplo:

```
for $b in doc("libros.xml")/libros/libro
    let $titulo := $b/titulo
    where $b/precio <= 50
    order by $titulo
    return $titulo
```

4.2.2. Cuantificadores `some` y `every`

- Determinan si una condición se cumple en alguno o todos los elementos de una secuencia (permiten condiciones complejas).
- Sintaxis:
  - `some $var in secuencia satisfies condición`: algún valor de `$var` satisface la condición.
  - `every $var in secuencia satisfies condición`: todos los valores de `$var` satisfacen la condición.
- Ejemplo:

```
for $b in doc("libros.xml")/libros/libro
    where some $x in $b//autor
    satisfies $x/@id = "a01"
    return $b
```

  - `$b`: es un elemento.
  - `$b//autor`: es un conjunto o secuencia.

4.2.3. Constructoras de elementos

- Es posible integrar los resultados de una consulta en otros elementos XML.
- Las expresiones XQuery contenidas dentro de un elemento han de estar delimitadas por llaves.
- Ejemplo:

```
for $b in doc("libros.xml")/libros/libro
return <resultado>
           {$b/titulo}
           {$b/precio}
       </resultado>
```

4.2.4. Acceso al contenido de un elemento

- Supongamos que queremos devolver el título y el precio contenido dentro de las etiquetas `<t>` y `<p>`, respectivamente. Ejemplo:

```
for $b in doc("libros.xml")/libros/libro
    return <resultado>
               <t>{$b/titulo}</t>
               <p>{$b/precio}</p>
           </resultado>
```

- La función `data()` permite acceder al contenido de un elemento. También funciona con secuencias de elementos. Ejemplo:

```
for $b in doc("libros.xml")/libros/libro
    return <resultado>
               <t>{data($b/titulo)}</t>
               <p>{data($b/precio)}</p>
           </resultado>
```

- Otra posibilidad es hacer referencia al contenido de un elemento mediante `text()`. Ejemplo:

```
for $b in doc("libros.xml")/libros/libro
    return <resultado>
               <t>{$b/titulo/text()}</t>
               <p>{$b/precio/text()}</p>
           </resultado>
```

- Más ejemplos:

```
for $b in doc("libros.xml")/libros/libro
let $numautores := count($b/autores/autor)
return <resultado>
         <t>{data($b/titulo)}</t>
         <p>{data($b/precio)}</p>
         <na>{$numautores}</na>
    </resultado>
```

```
for $b in doc("libros.xml")/libros/libro
let $numautores := count($b/autores/autor)
return <resultado numautores="{$numautores}">
         <t>{data($b/titulo)}</t>
         <p>{data($b/precio)}</p>
    </resultado>
```

```
for $b in doc("libros.xml")/libros/libro
let $numautores := count($b/autores/autor)
where $numautores = 1
return <resultado>
        <t>{data($b/titulo)}</t>
        <p>{data($b/precio)}</p>
    </resultado>
```

Restringimos la búsqueda a aquellos casos en los que solo haya un autor.

4.2.5. Consultas anidadas

- Es posible anidar expresiones FLWOR en los resultados de una consulta.
- Ejemplos:

```
for $b in doc("libros.xml")/libros/libro
return <resultado>
    <t>{data($b/titulo)}</t>
    <p>{data($b/precio)}</p>
        {for $a in $b//autor
        return $a
    }
    </resultado>
```

En este ejemplo relacionamos dos archivos XML distintos: *libros.xml* y *autores.xml*:

```
for $b in doc("libros.xml")/libros/libro
return <resultado>
        <t>{data($b/titulo)}</t>
        <p>{data($b/precio)}</p>
        {for $a in $b//autor
            let $infoautor := doc("autores.xml")//autor[@id = $a/@id]
        return $infoautor
        }
    </resultado>
```

```
for $b in doc("libros.xml")/libros/libro
return <resultado>
        <t>{data($b/titulo)}</t>
        <p>{data($b/precio)}</p>
            {for $a in $b//autor
                let $infoautor := doc("autores.xml")//autor[@id = $a/@id]
            return <autor>{data($infoautor/nombre)}
                           {data($infoautor/apellidos)}
                   </autor>
            }
        </resultado>
```

4.2.6. XQuery y HTML

Es posible construir una página HTML con el resultado de una consulta XQuery. Ejemplo:

Función	Descripción
`count(seq)`	Devuelve el número de elementos de la secuencia dada
`sum(seq)`	Devuelve la suma de los elementos de la secuencia dada
`avg(seq)`	Devuelve la media de los elementos de seq
`max(seq)`, `min(seq)`	Devuelve el máximo/mínimo de los elementos de seq
`distinct-values(seq)`	Elimina los duplicados de seq
`contains(cadena,subcadena)`	Busca una subcadena de caracteres en una cadena
`except`	Permite "quitar" código XML

Cuadro 1: Algunas funciones de agregación sobre conjuntos secuencias

```
let $my-doc := doc("books.xml")
return
<html>
    <head>
        <title>Current Rates</title>
    </head>
    <body>
    <ul>
{
for $x in $my-doc/bookstore/book
order by $x/title
return <li>{data($x/title)}. Category: {data($x/@category)}</li>
}</ul>
   </body>
</html>
```

4.3. SGBD eXist

- Gestor de bases de datos XML que utiliza XQuery como lenguaje de acceso a los datos.
- Se distribuye bajo la licencia GNU LGPL.
- Puede obtenerse en la dirección: `http://exist-db.org/`
- Durante la instalación se solicita el nombre de usuario y contraseña del administrador.
- En los laboratorios:
  - Nombre de usuario: *admin*
  - Contraseña: *ninguna*
- Tras arrancar el gestor de bases de datos, abrir un navegador web e introducir la dirección: `http://localhost:8080/exist`
- Introducir *login* y contraseña de administrador.

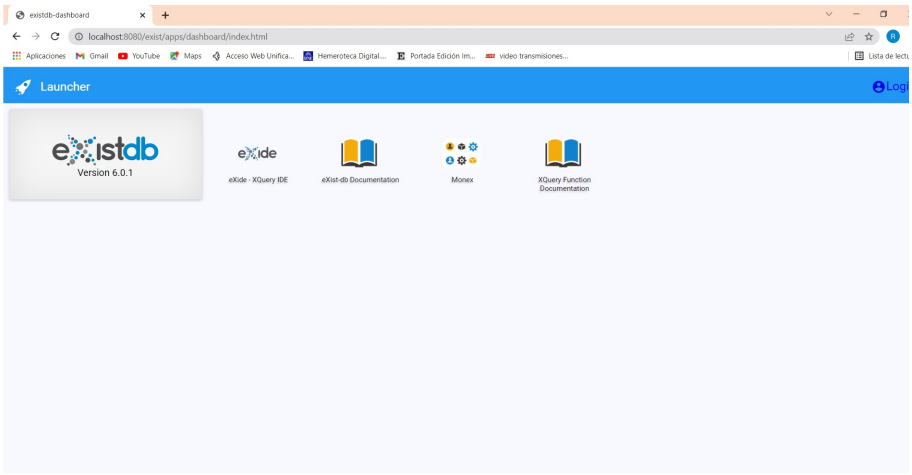

Figura 1: Ventana principal de eXist-db

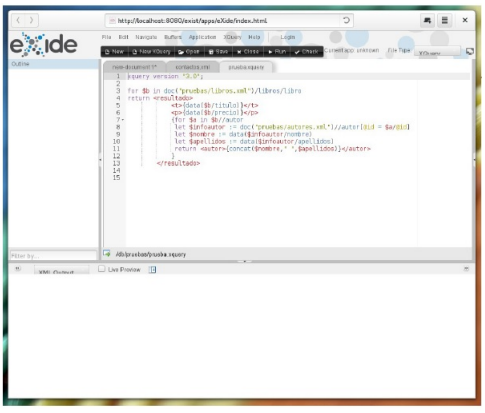

Figura 2: Ventana de edición en eXist-db