

# LENGUAJES DE MARCAS Y SISTEMAS DE GESTIÓN DE INFORMACIÓN



**FPA** FORMACIÓN PROFESIONAL ANDALUZA

## Elena Fernández Chirino

## UNIDAD 6:

# Almacenamiento de la información

## INDICACIONES TAREA 6 : RELACIÓN CON LA TEORÍA

DEPARTAMENTO DE INFORMÁTICA

<http://www.iestrassierra.com>



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).

# 1.- INDICACIONES GENERALES TAREA 6

## ¿Qué contenidos y resultados de aprendizaje trabajaremos?

- Sistemas de almacenamiento de información en formato XML. Ventajas e inconvenientes. Tecnologías.
- Sistemas gestores de bases de datos relacionales y documentos XML. Almacenamiento, búsqueda y extracción de la información.
- Sistemas gestores de bases de datos nativas XML.
- Herramientas y técnicas de tratamiento y almacenamiento de información en formato XML.
- Lenguajes de consulta y manipulación.

## ¿Qué te pedimos que hagas?

A partir del fichero [biblioteca.xml](#) deberás realizar los siguientes ejercicios de **XQuery** para obtener información relevante a partir de los datos almacenados en el fichero xml de origen. Deberás generar las consultas y probarlas

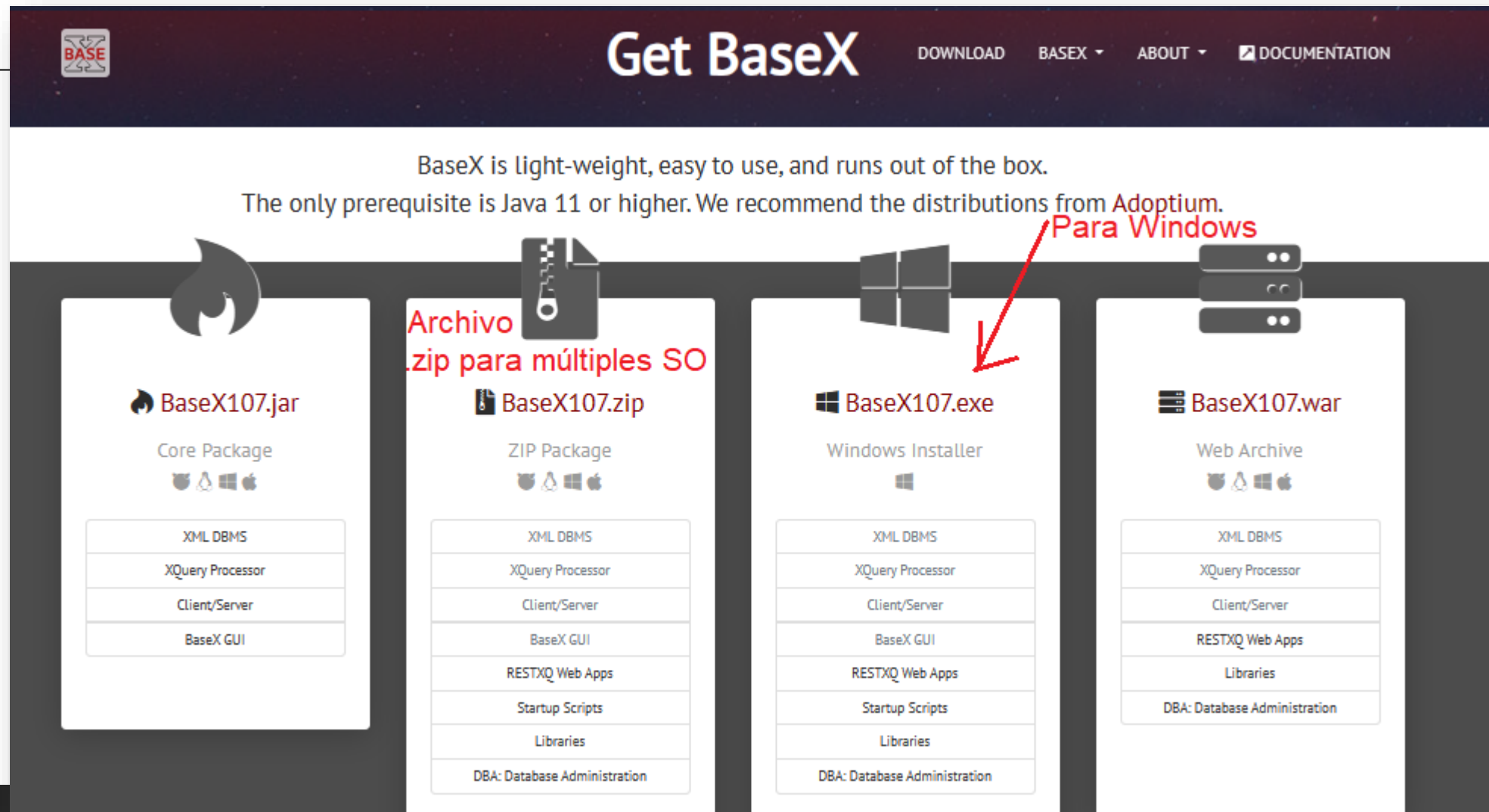
2 partes

Instalación y “preparación” del software BaseX para trabajar con un archivo .XML “como una base de datos”

Conocimiento de XQuery como lenguaje para búsqueda de información en un documento XML

# Descarga e instalación de BaseX

Enlace de descarga: <https://basex.org/download>



The screenshot shows the 'Get BaseX' page on the BaseX website. The header includes the BaseX logo and navigation links: DOWNLOAD, BASEX, ABOUT, and DOCUMENTATION. The main text states: 'BaseX is light-weight, easy to use, and runs out of the box. The only prerequisite is Java 11 or higher. We recommend the distributions from Adoptium.' Below this, there are four download options, each with an icon, a title, a description, and a list of included components. A red arrow points from the text 'Para Windows' to the Windows Installer option.

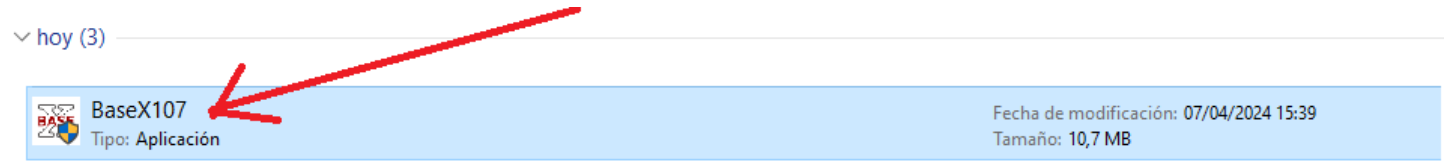
BaseX is light-weight, easy to use, and runs out of the box.  
The only prerequisite is Java 11 or higher. We recommend the distributions from Adoptium.

**Para Windows**

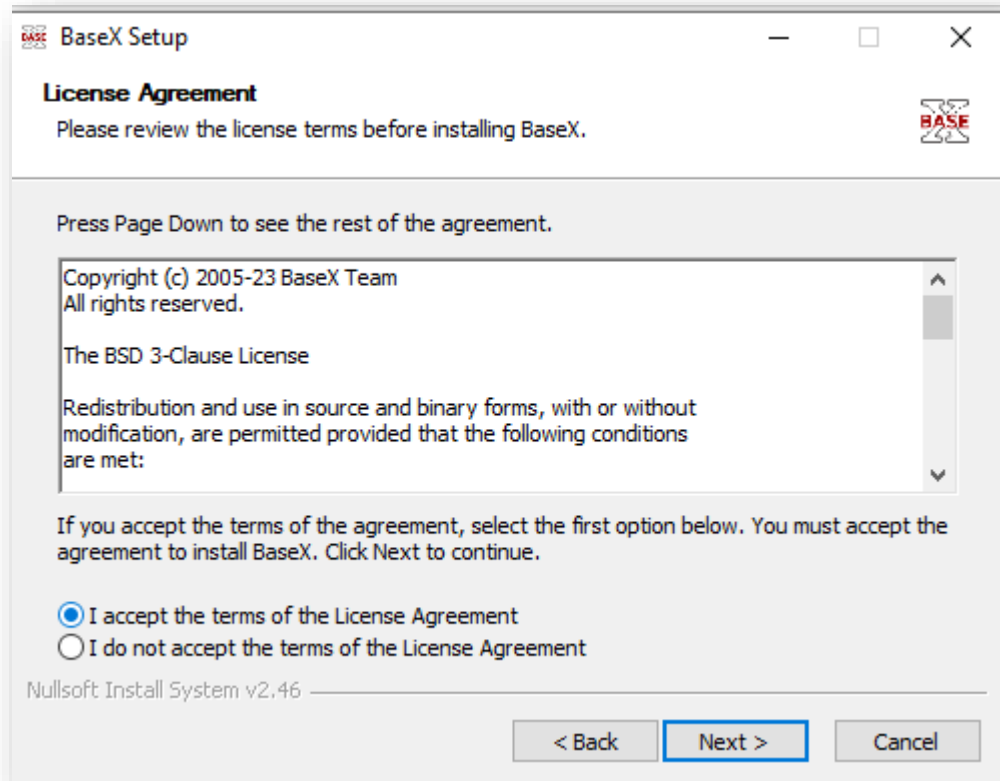
Icon	Download Option	Description	Platform Icons	Components
🔥	BaseX107.jar	Core Package	🐙 🐧 🍏	XML DBMS XQuery Processor Client/Server BaseX GUI
📄	BaseX107.zip	ZIP Package	🐙 🐧 🍏	XML DBMS XQuery Processor Client/Server BaseX GUI RESTXQ Web Apps Startup Scripts Libraries DBA: Database Administration
🪟	BaseX107.exe	Windows Installer	🍏	XML DBMS XQuery Processor Client/Server BaseX GUI RESTXQ Web Apps Startup Scripts Libraries DBA: Database Administration
🌐	BaseX107.war	Web Archive	🐙 🐧 🍏	XML DBMS XQuery Processor Client/Server RESTXQ Web Apps Libraries DBA: Database Administration

# Descarga e instalación de BaseX

Una vez descargado ejecutamos haciendo “doble clic”



Aceptamos condiciones de licencia:

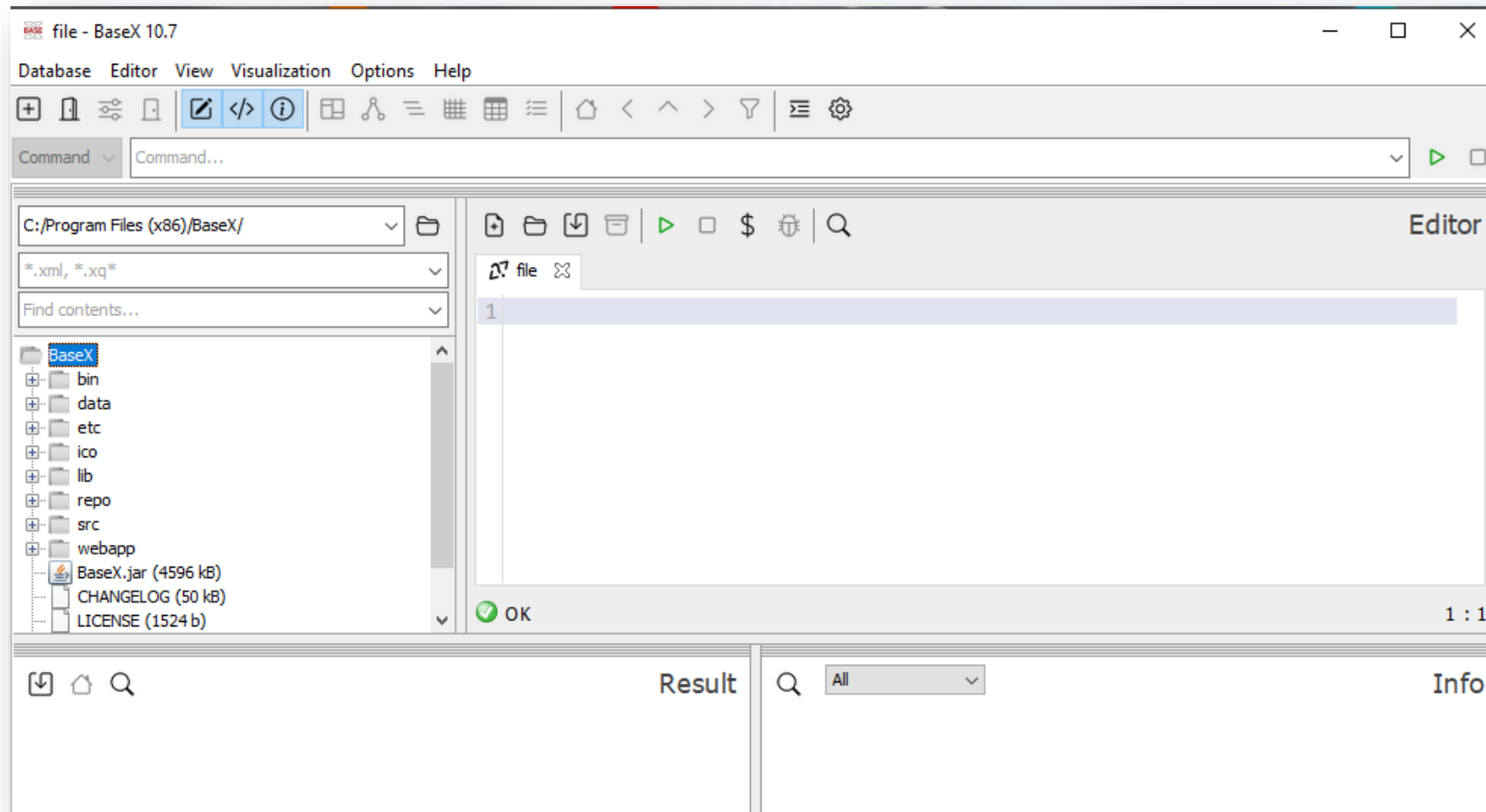


Dejamos las opciones marcadas por defecto e introducimos una clave que deberemos recordar para acceder como usuario **admin**

Una vez completada la instalación veamos cómo proceder para tener nuestro archivo XML que funcionará “como una BD” y poder realizar consultas en él con el lenguaje Xquery, de manera similar a como utilizamos SQL para las BD relacionales.

# Descarga e instalación de BaseX

Cuando entramos en el programa por primera vez, accedemos a la siguiente pantalla:



## MUY IMPORTANTE:

Debemos tener en la misma ruta el archivo XML sobre el que creamos la BD en XBase y el archivo XQuery sobre el que vamos a realizar las consultas porque de lo contrario no encuentra “el camino” cuando damos a ejecutar la consulta.

# Utilización de BaseX para consultas

Por ejemplo: En la carpeta “tarea6”, donde tenemos el archivo biblioteca.XML, guardamos el archivo ConsultasTarea6.xquey para poder ejecutar las consultas

A continuación, indicaremos paso a paso como crear la nueva Base de Datos y “dejarlo todo preparado para consultar la información”

BaseX 10.7 interface showing a query execution. The main editor displays the XQuery:

```
1 for $libro in doc("biblioteca.xml")//book
2 where $libro/year > 2015
3 return $libro/title
```

The left sidebar shows the file explorer with the following files:

- tarea6
- biblioteca.xml (4488 b)
- ConsultasTarea6.xq (0 b)

The bottom panel shows the results of the query (3 Results, 138 b):

```
<title lang="en">IT</title>
<title lang="es">Drácula (Clásicos ilustrados)</title>
<title lang="es">Trilogía de la Fundación</title>
```

The right sidebar shows the XML document structure (biblioteca.xml):

```
library
data
  catalog
    1
    2
    3
    4
    5
    6
    7
```

The bottom right panel shows the compilation status:

Compiling:


- merge: descendant::book
- rewrite > comparison: (\$libro\_0/year > 2015) -> \$libro\_0
- rewrite to predicate: year >= 2015.000000000000002

# Utilización de BaseX para consultas

Vamos a hacerlo por facilitar el trabajo sobre el archivo XML que deberéis utilizar para realizar las consultas de vuestra práctica. Lo descargamos de la Tarea06 de moodle

---

## ¿Qué te pedimos que hagas?

A partir del fichero [biblioteca.xml](#)  deberás realizar los siguientes ejercicios de **XQuery** para obtener información relevante a partir de los datos almacenados en el fichero xml de origen. Deberás generar las consultas y probarlas

Pulsando sobre el enlace descargamos un archivo en biblioteca.zip y al descomprimirlo tenemos la carpeta biblioteca y dentro el archivo biblioteca.xml:

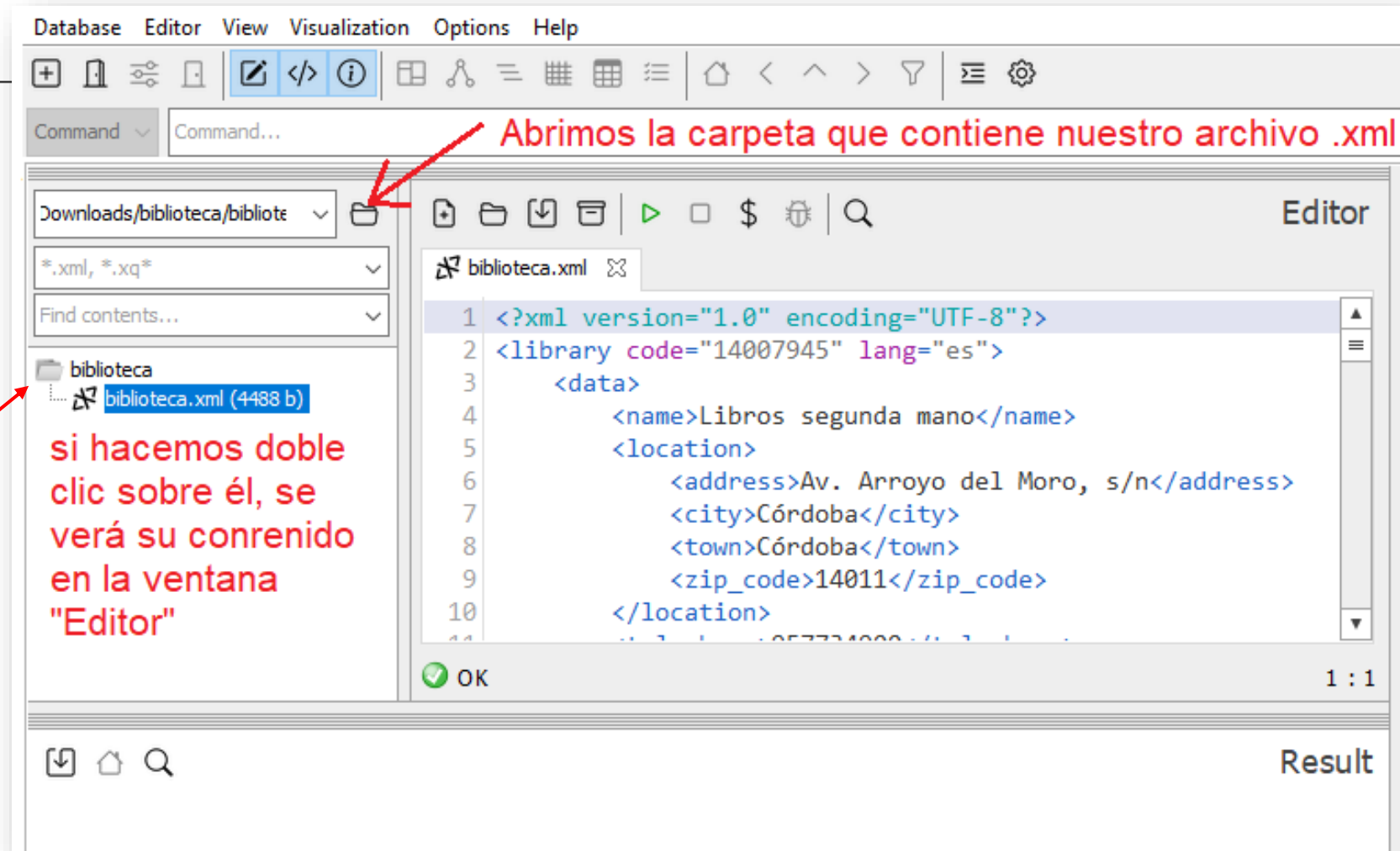
Pasos

- 1º Lo abrimos
- 2º Creamos la BD del mismo nombre
- 3º Guardamos un archivo .xquery dentro de la misma carpeta



# Utilización de BaseX para consultas

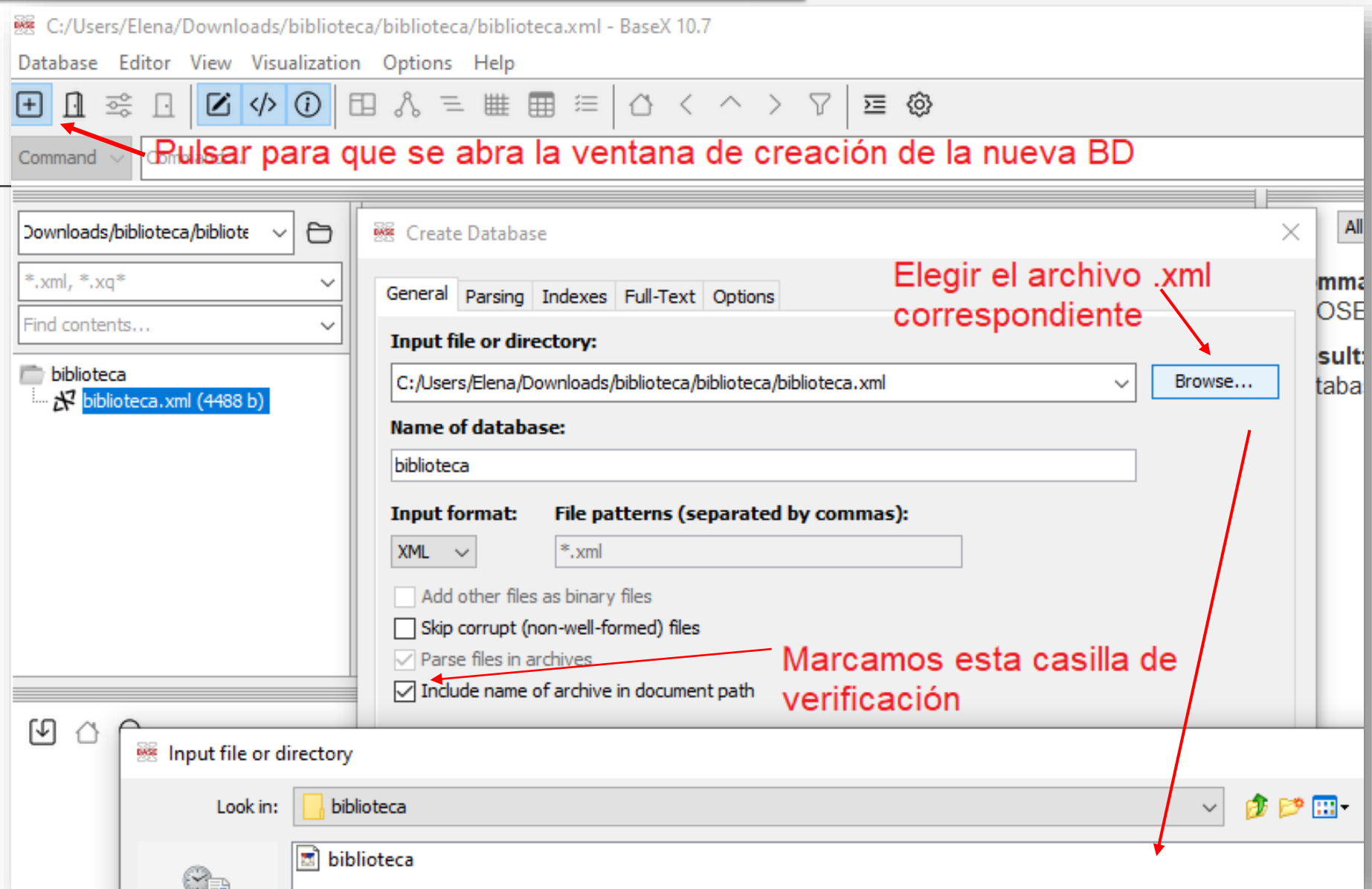
1º Abrimos el archivo que contiene el archivo XML





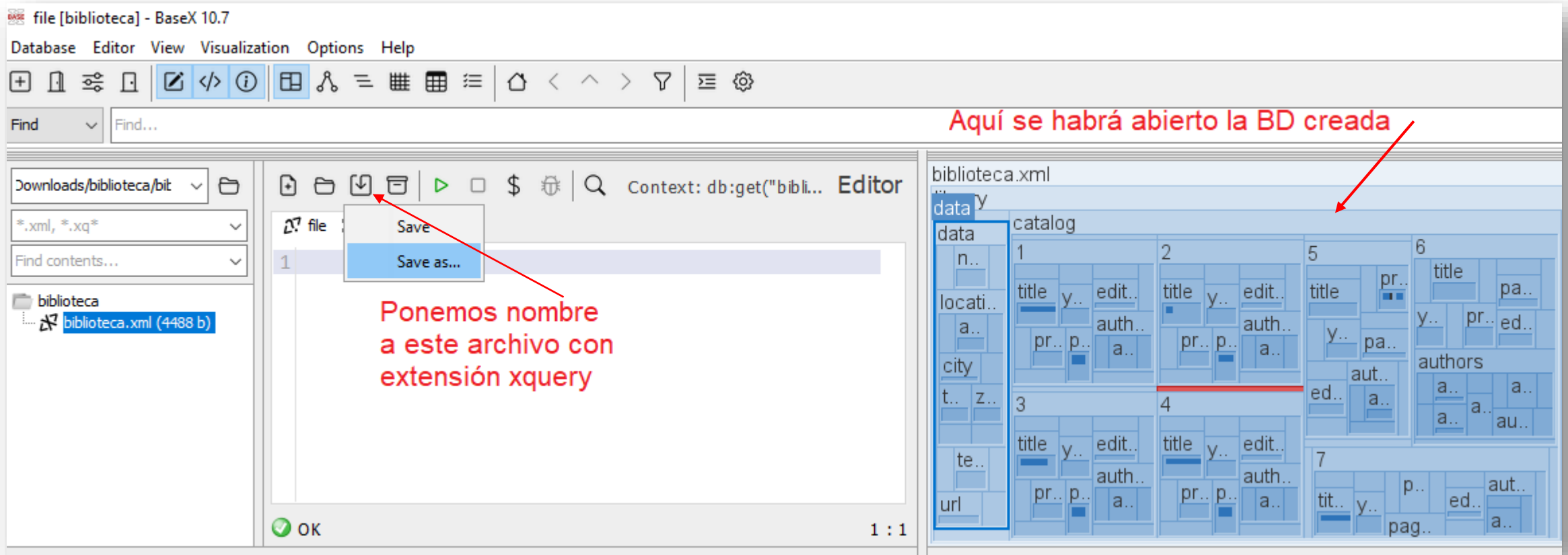
# Utilización de BaseX para consultas

2º Crear una BD desde BaseX a partir del archivo XML



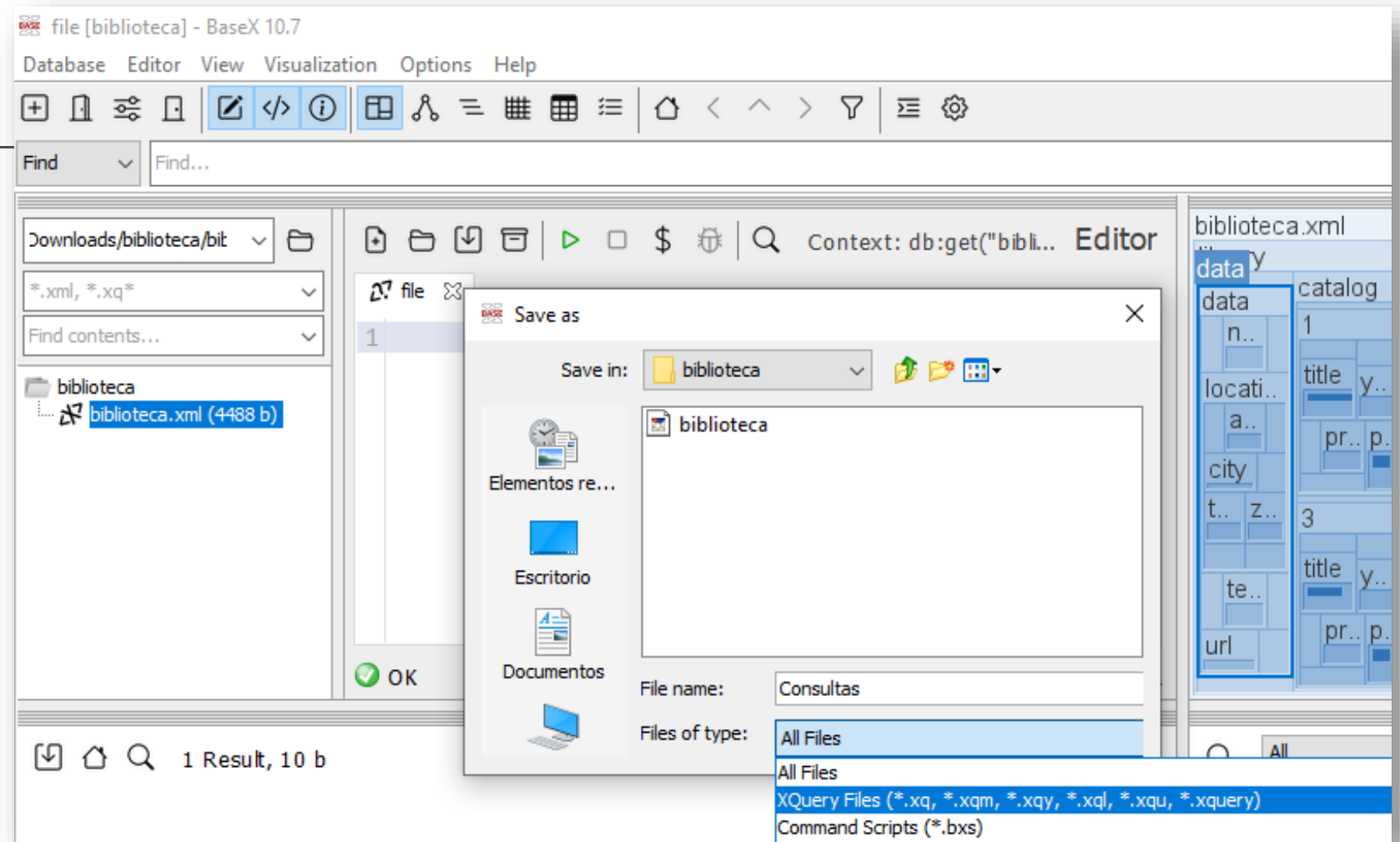
# Utilización de BaseX para consultas

3º Crear un archivo .xquery en la misma carpeta donde está el archivo .xml y en él consultar los datos



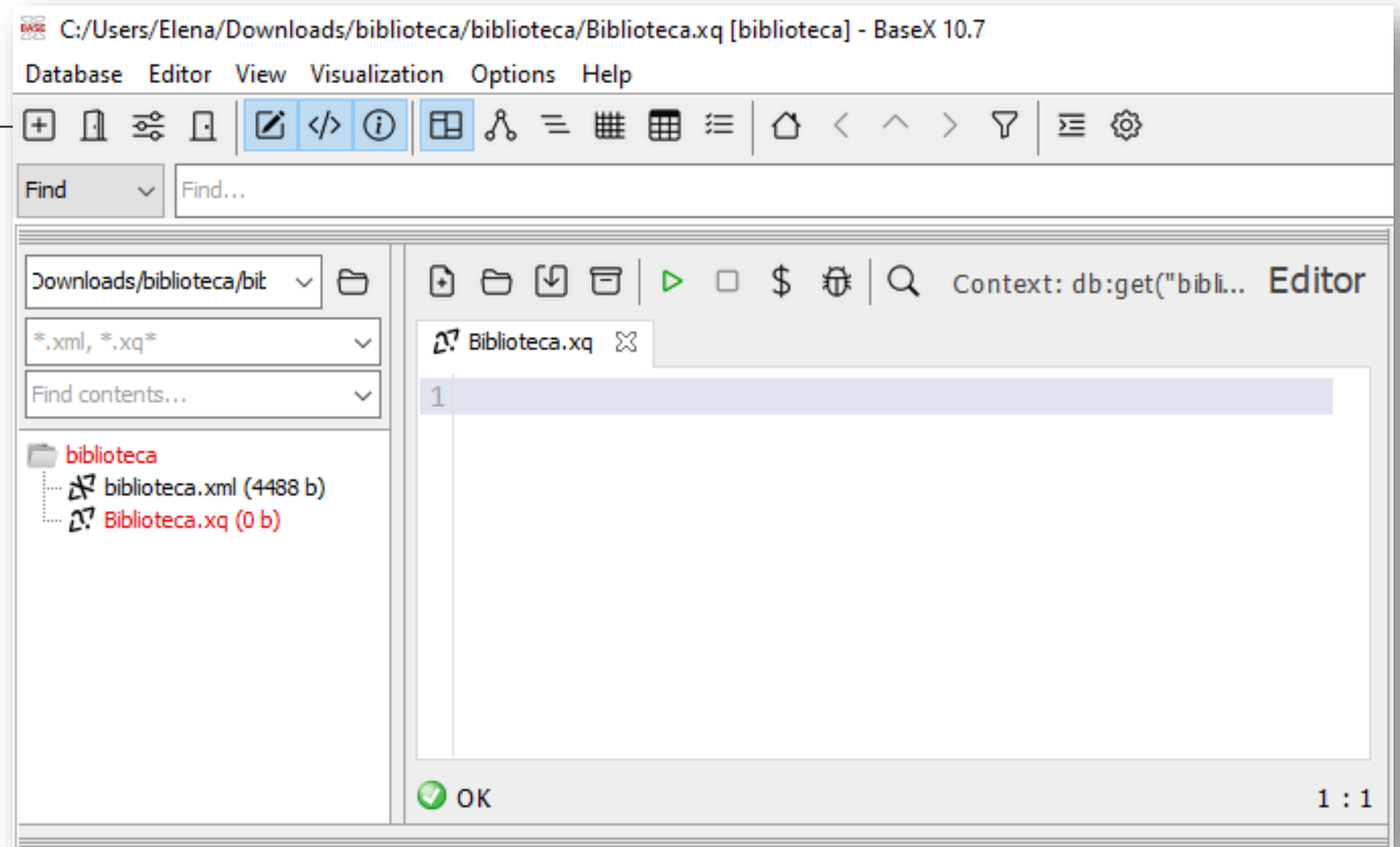
# Utilización de BaseX para consultas

3º Crear un archivo .xquery en la misma carpeta donde está el archivo .xml y en él consultar los datos

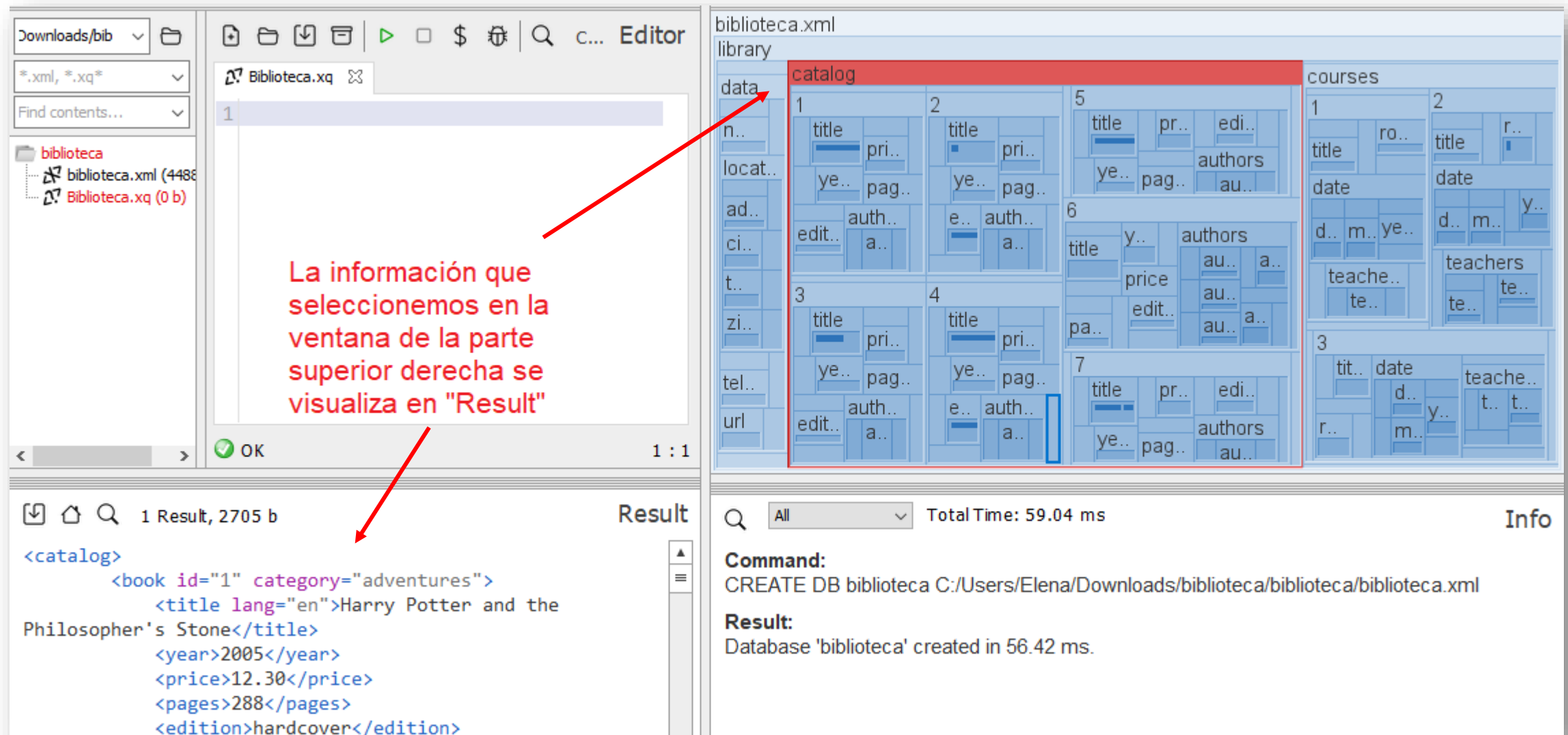


# Utilización de BaseX para consultas

3º Crear un archivo .xquery en la misma carpeta donde está el archivo .xml y en él consultar los datos



# Utilización de BaseX para consultas



La información que seleccionemos en la ventana de la parte superior derecha se visualiza en "Result"

1 Result, 2705 b

```
<catalog>
  <book id="1" category="adventures">
    <title lang="en">Harry Potter and the
Philosopher's Stone</title>
    <year>2005</year>
    <price>12.30</price>
    <pages>288</pages>
    <edition>hardcover</edition>
```

Command:  
CREATE DB biblioteca C:/Users/Elena/Downloads/biblioteca/biblioteca/biblioteca.xml

Result:  
Database 'biblioteca' created in 56.42 ms.

# Cómo ejecutar una consulta

**Pulsar para ejecutar la consulta**

**Aquí se escriben las cláusulas del lenguaje de consultas XQuery**

```
1 for $libro in doc("biblioteca.xml")//
2   book
3   where $libro/year > 2015
4   return $libro/title
```

**Se resaltan los datos mostrados de otro color**

**El resultado de la ejecución se muestra aquí**

3 Results, 138 b

```
<title lang="en">IT</title>
<title lang="es">Drácula (Clásicos ilustrados)</title>
<title lang="es">Trilogía de la Fundación</title>
```

**Compiling:**

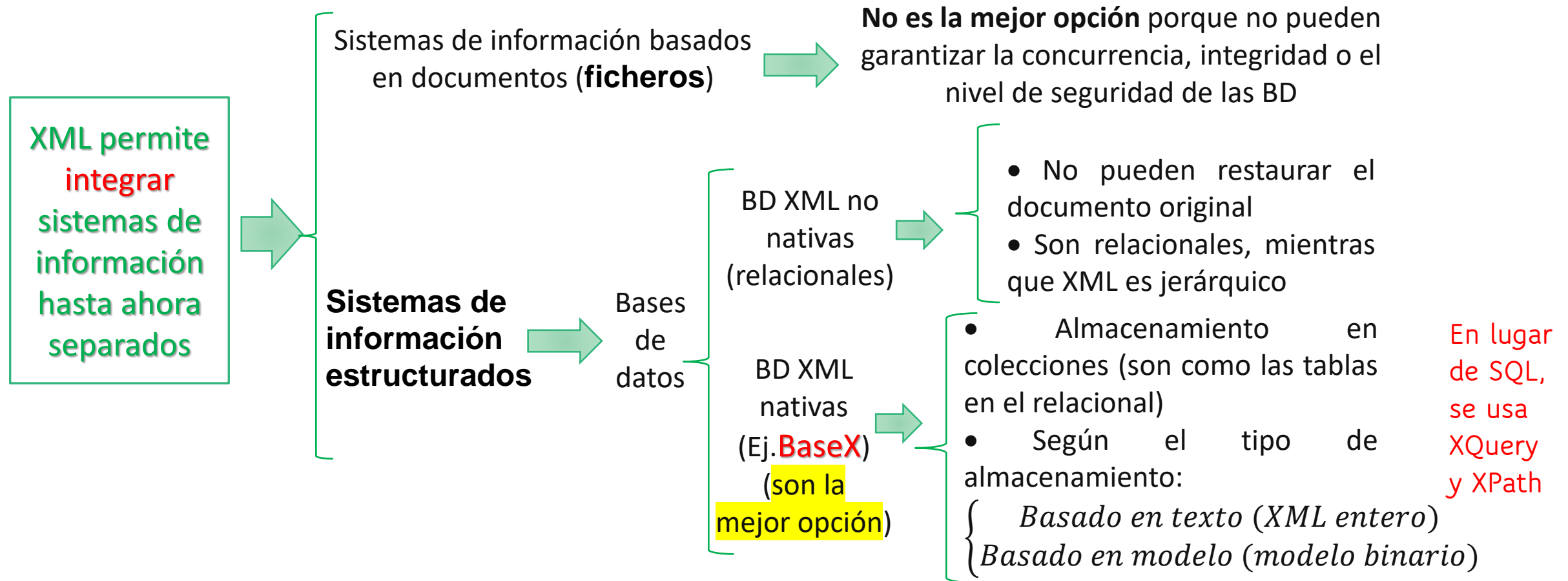
- merge: descendant::book
- rewrite > comparison: (\$libro\_0/year > 2015) -> \$libro\_0/year >= 2015.

Total Time: 3.2 ms

# 2.- CONTENIDOS FUNDAMENTALES DE LA UNIDAD 6

## 2.1.- Almacenamiento de la información

Dado que la tecnología XML se ha impuesto en muchas organizaciones, es útil utilizar un **sistema de almacenamiento y consulta de datos compatible con XML**.





## 2.2.- ¿Cómo buscar información dentro de un documento XML? XQuery

El lenguaje XML se concibió como un lenguaje para el intercambio de información en un formato estandarizado, pero una de las operaciones más complejas de realizar en un documento XML es la búsqueda de información. → **Necesitamos un lenguaje de consultas.** Del mismo modo que contamos con **SQL en las BD** relacionales, **XPath y Xquery nos proporciona acceso a datos en BD XML.**

**XPath** es un lenguaje que permite seleccionar nodos o conjuntos de nodos en un documento XML: Su concepto básico es la expresión de una ruta (en él se basa XQuery)

Expresión	Descripción
*	Todo el documento
/	Todo el documento
elemento	Proporciona el nodo <b>elemento</b> a partir de la posición actual
/elemento	Proporciona el nodo <b>elemento</b> a partir de la raíz del documento
//elemento	Proporciona todos los nodos <b>elemento</b> independientemente de su posición
/elemento1/elemento2	Proporciona todos los nodos <b>elemento2</b> que son hijos del <b>elemento1</b>
/elemento1//elemento2	Proporciona todos los nodos <b>elemento2</b> que dependen jerárquicamente de <b>elemento1</b>

**XQuery** es un lenguaje diseñado para escribir consultas sobre colecciones de datos expresadas en XML. Su principal función es extraer información de un conjunto de datos organizados como un árbol de etiquetas XML.

## 2.3.- Aspectos básicos de XQuery. Expresiones

**Una consulta en XQuery es una expresión que lee una secuencia de datos en XML, y devuelve como resultado otra secuencia de datos en XML.**

Puesto que XQuery ha sido construido sobre la base de XPath y realiza la selección de información y la iteración a través del conjunto de datos basándose en dicho lenguaje, **toda expresión XPath también es una consulta Xquery válida.**

Las **consultas XQuery** pueden estar formadas por **hasta cinco tipos de cláusulas diferentes**, siguen la **norma FLWOR** (que se pronuncia "flower"). Estas cláusulas son los bloques principales del XQuery, **equivalen a las cláusulas select, from, where, group by, having, order by y limit de SQL.**

**FLWOR** {

- FOR:** { *Para recorrer colecciones*  
*Vincula una variable con cada elemento devuelto por la consulta*
- LET:** *Vincula una variable con el resultado completo de la consulta*
- WHERE** { *Para incorporar condiciones a la cláusula FOR*  
*Filtra las tuplas eliminando todos los valores que no cumplen las condiciones dadas*
- ORDER BY:** *Para ordenar los resultados*
- RETURN:** *Para indicar que resultado hay que devolver*

Los **comentarios en XQuery** están limitados entre caras sonrientes, es decir: **(: Esto es un comentario XQuery :).**

## 2.3.- Aspectos básicos de XQuery. Expresiones

**Una consulta en XQuery es una expresión que lee una secuencia de datos en XML, y devuelve como resultado otra secuencia de datos en XML.**

Puesto que XQuery ha sido construido sobre la base de XPath y realiza la selección de información y la iteración a través del conjunto de datos basándose en dicho lenguaje, **toda expresión XPath también es una consulta Xquery válida.**

Las **consultas XQuery** pueden estar formadas por **hasta cinco tipos de cláusulas diferentes**, siguen la **norma FLWOR** (que se pronuncia "flower"). Estas cláusulas son los bloques principales del XQuery, **equivalen a las** cláusulas select, from, where, group by, having, order by y limit de SQL.

**FLWOR** {

- FOR:** { *Para recorrer colecciones*  
*Vincula una variable con cada elemento devuelto por la consulta*
- LET:** *Vincula una variable con el resultado completo de la consulta*
- WHERE** { *Para incorporar condiciones a la cláusula FOR*  
*Filtra las tuplas eliminando todos los valores que no cumplen las condiciones dadas*
- ORDER BY:** *Para ordenar los resultados*
- RETURN:** *Para indicar que resultado hay que devolver*

Los **comentarios en XQuery** están limitados entre caras sonrientes, es decir: **(: Esto es un comentario XQuery :)**.

## 2.4.- Sintaxis de una cláusula FLWOR

Reglas  
generales:

- **FOR** y **LET** sirven para crear las tuplas con las que trabajará el resto de las cláusulas de la consulta y pueden usarse tantas veces como se deseen una consulta, incluso dentro de otras cláusulas.
- Sin embargo, solo puede declararse una única cláusula **WHERE**, una única **ORDER BY** y una única cláusula **RETURN**

**NOTA 1:** Ninguna de las cláusulas FLWOR es obligatoria en una consulta XQuery. Por ejemplo, una expresión XPath, como la que se muestra a continuación, es una consulta válida y no contiene ninguna de las cláusulas FLWOR.

```
doc("libros.xml")/bib/libro/titulo
```

Esta expresión XPath, que también es una consulta XQuery válida, devuelve **los títulos de todos los libros en la biblioteca especificada en “libros.xml”**.

**NOTA 2:** Los comentarios en XQuery, a diferencia de XML, van encerrados entre caras sonrientes, tal y como se muestra a continuación: (: Esto es un feliz comentario :)

Si se incluyen **varias consultas en un mismo archivo .xquery** para su ejecución conjunta, cada una de las consultas se separarán con una coma (,)

Una consulta XQuery está  
**formada por dos partes:**

{ **Prólogo:** Lugar donde se declaran las funciones, variables, etc.  
**Expresión:** Consulta propiamente dicha

## 2.5.- Diferencia entre FOR y LET

El objetivo de la sentencia LET es el mismo de la sentencia FOR, obtener una secuencia de tuplas, pero la forma de hacerlo es distinta. A diferencia de FOR, **LET obtiene una única tupla y no varias.**

Consulta realizada con FOR	Consulta realizada con LET
La consulta obtiene todos los autores dentro de etiquetas <MisAutores>. Para ello, primero la consulta recupera todos los nodos //Libros/Libro/Autor que identifican en la variable \$a. Seguidamente, cada una de las tuplas almacenadas en \$a se colocan entre etiquetas <MisAutores>	Repetimos la consulta anterior con la cláusula LET
<pre>for \$a in //Libros/Libro/Autor return &lt;MisAutores&gt;{\$a}&lt;/MisAutores&gt;</pre>	<pre>let \$a := //Libros/Libro/Autor return &lt;MisAutores&gt;{\$a}&lt;/MisAutores&gt;</pre>
<pre>&lt;MisAutores&gt; &lt;Autor&gt;Nikolai Gogol&lt;/Autor&gt; &lt;/MisAutores&gt; &lt;MisAutores&gt; &lt;Autor&gt;Gonzalo Giner&lt;/Autor&gt; &lt;/MisAutores&gt; &lt;MisAutores&gt; &lt;Autor&gt;Umberto Eco&lt;/Autor&gt; &lt;/MisAutores&gt;</pre>	<pre>&lt;MisAutores&gt; &lt;Autor&gt;Nikolai Gogol&lt;/Autor&gt; &lt;Autor&gt;Gonzalo Giner&lt;/Autor&gt; &lt;Autor&gt;Umberto Eco&lt;/Autor&gt; &lt;/MisAutores&gt;</pre>

## 2.6.- Funciones de entrada

XQuery utiliza las funciones de entrada en las cláusulas for o let o en expresiones XPath para identificar el origen de los datos. La **función doc**(URI) devuelve el nodo documento, o nodo raíz, del documento. Esta es la función de entrada más habitual para acceder a la información almacenada en archivos.

EJEMPLOS: Una vez cargado el documento XML en la BD

```
(: Usamos la función doc() para cargar el documento XML 'alumnos.xml' :)  
for $a in doc("alumnos.xml")//alumno
```

```
(: Usamos una cláusula where para filtrar solo aquellos alumnos cuya nota es mayor que 5 :)  
where $a/nota > 5
```

```
(: Finalmente, devolvemos un nuevo elemento XML 'aprobado' que contiene el DNI y la nota del alumno :)  
return <aprobado>{ $a/@dni, $a/nota }</aprobado>
```

Observamos que la instrucción entre “llaves”, crea un nuevo elemento XML ‘aprobado’ que contiene el DNI y la nota de cada alumno que ha aprobado (es decir, su nota es mayor que 5).

**OBSERVACIÓN:** En un documento XQuery los caracteres { } delimitan las expresiones que son evaluadas para crear un documento nuevo.

## 2.7.- Expresiones condicionales

XQuery admite expresiones condicionales del tipo **if-then-else** con la misma semántica que tienen en los lenguajes de programación habituales.

CÓDIGO Xquery (comentado)	Resultado de su ejecución
<pre>(: Usamos la función doc() para cargar el documento XML 'libros.xml' :) for \$b in doc("libros.xml")//libro  return (: Para cada libro, devolvemos un nuevo elemento XML 'libro' :) &lt;libro&gt;  { \$b/titulo } (: Dentro del elemento 'libro', incluimos el título del libro :)  (: Luego, para cada autor del libro (hasta el segundo autor), devolvemos un nuevo elemento 'autor' :) { for \$a at \$i in \$b/autor where \$i &lt;= 2 return &lt;autor&gt;{string(\$a/last), ", ", string(\$a/first)}&lt;/autor&gt; }  (: Si el libro tiene más de 2 autores, añadimos 'et al.' después del 2º :) { if (count(\$b/autor) &gt; 2) then &lt;autor&gt;et al.&lt;/autor&gt; else () }  &lt;/libro&gt;</pre>	<pre>&lt;libro&gt; &lt;titulo&gt;TCP/IP Illustrated&lt;/titulo&gt; &lt;autor&gt;Stevens, W.&lt;/autor&gt; &lt;/libro&gt; &lt;libro&gt; &lt;titulo&gt;Advanced Programming in the Unix Environment&lt;/titulo&gt; &lt;autor&gt;Stevens, W.&lt;/autor&gt; &lt;/libro&gt; &lt;libro&gt; &lt;titulo&gt;Data on the Web&lt;/titulo&gt; &lt;autor&gt;Abiteboul, Serge&lt;/autor&gt; &lt;autor&gt;Buneman, Peter&lt;/autor&gt; &lt;autor&gt;et al.&lt;/autor&gt; &lt;/libro&gt;</pre>



## 2.8.- Cuantificadores existenciales

XQuery soporta dos cuantificadores existenciales llamados “**some**” y “**every**”, de tal manera que nos permite devolver algún elemento que satisfaga la condición dada (“some”) o los elementos en los que todos sus nodos satisfagan la condición dada (“every”).

CÓDIGO Xquery con “ <b>some</b> ”	CÓDIGO Xquery con “ <b>every</b> ”
<pre>(: Usamos la función doc() para cargar el documento XML 'libros.xml' :) for \$b in doc("libros.xml")//libro  (: Usamos una cláusula where con la función some para filtrar solo aquellos libros que tienen un autor cuyo apellido es 'Stevens' y cuyo nombre es 'W.' :) where some \$a in \$b/autor satisfies (\$a/last="Stevens" and \$a/first="W.")  (: Finalmente, retornamos el título de cada libro que cumple con la condición anterior :) return \$b/titulo</pre>	<pre>(: La siguiente consulta devuelve todos los títulos de los libros en los que todos los autores de cada libro es W. Stevens :) for \$b in doc("libros.xml")//libro where every \$a in \$b/autor satisfies (\$a/last="Stevens" and \$a/first="W.") return \$b/titulo</pre> <p><b>NOTA:</b> Cuando un cuantificador universal se aplica sobre un nodo vacío, siempre devuelve cierto → El último libro devuelto en el resultado no tiene autores</p>
<p><b>RESULTADO DE LA CONSULTA:</b></p> <pre>&lt;title&gt;TCP/IP Illustrated&lt;/title&gt; &lt;title&gt;Advanced Programming in the Unix Environment&lt;/title&gt;</pre>	<p><b>RESULTADO DE LA CONSULTA:</b></p> <pre>&lt;titulo&gt;TCP/IP Illustrated&lt;/titulo&gt; &lt;titulo&gt;Advanced Programming in the Unix Environment&lt;/titulo&gt; &lt;titulo&gt;The Economics of Technology and Content for Digital TV&lt;/titulo&gt;</pre>

## 2.9.- Operadores y funciones principales

Algunos de los operadores y funciones más utilizados que soporta el lenguaje XQuery

OPERADORES	
Aritméticos	+ (suma), - (resta), * (Producto), div(división), idiv(división entera), mod.(Módulo)
Comparación general	=, !=, <, >, <=, >=, not()
Comparación de nodos	<b>is</b> , <b>is not</b> ,
De secuencia	<b>Unión</b> , <b>Intersect</b> , <b>Except</b> (Devuelve “que están en el 1º y no están en el 2º”)
Lógicos	<b>and</b> y <b>or</b>


FUNCIONES	
Numéricas	floor(), ceiling(), round(), count(), min(), max(), avg(), sum()
De Cadena	<b>concat()</b> , <b>string-length()</b> , devuelve la cantidad de caracteres que forman una cadena. <b>upper-case()</b> , <b>lower-case()</b> , devuelve la cadena dada en mayúsculas o minúsculas respectivamente.
De uso general	<b>empty()</b> , devuelve “ <b>true</b> ” cuando la secuencia dada no contiene ningún elemento. <b>exists()</b> , devuelve “ <b>true</b> ” cuando una secuencia contiene, al menos, un elemento. <b>distinct-values()</b> , extrae los valores de una secuencia de nodos y crea una nueva secuencia con valores únicos, eliminando los nodos duplicados.

# 3.- EJERCICIO PRÁCTICO

DESCARGAMOS EL ARCHIVO CON LA SOLUCIÓN DEL APARTADO DE LA UNIDAD 6: 3.5.- Cláusulas.



## Ejercicio resuelto

A continuación tienes la solución de este ejercicio  [Descargar solución](#) (2.403 KB)

Para poder probar este y cualquier otro ejemplo con consultas XQuery es necesario tener instalada una base de datos XML nativa, como por ejemplo: BaseX, donde hay que tener la base de datos creada y abierta a partir del fichero XML (en este ejemplo: libros.xml).

- ✓ [Página oficial de BaseX](#)
- ✓ [Instalación de BaseX y creación de una base de datos](#)
- ✓ [Creación de consultas XQuery con BaseX](#)

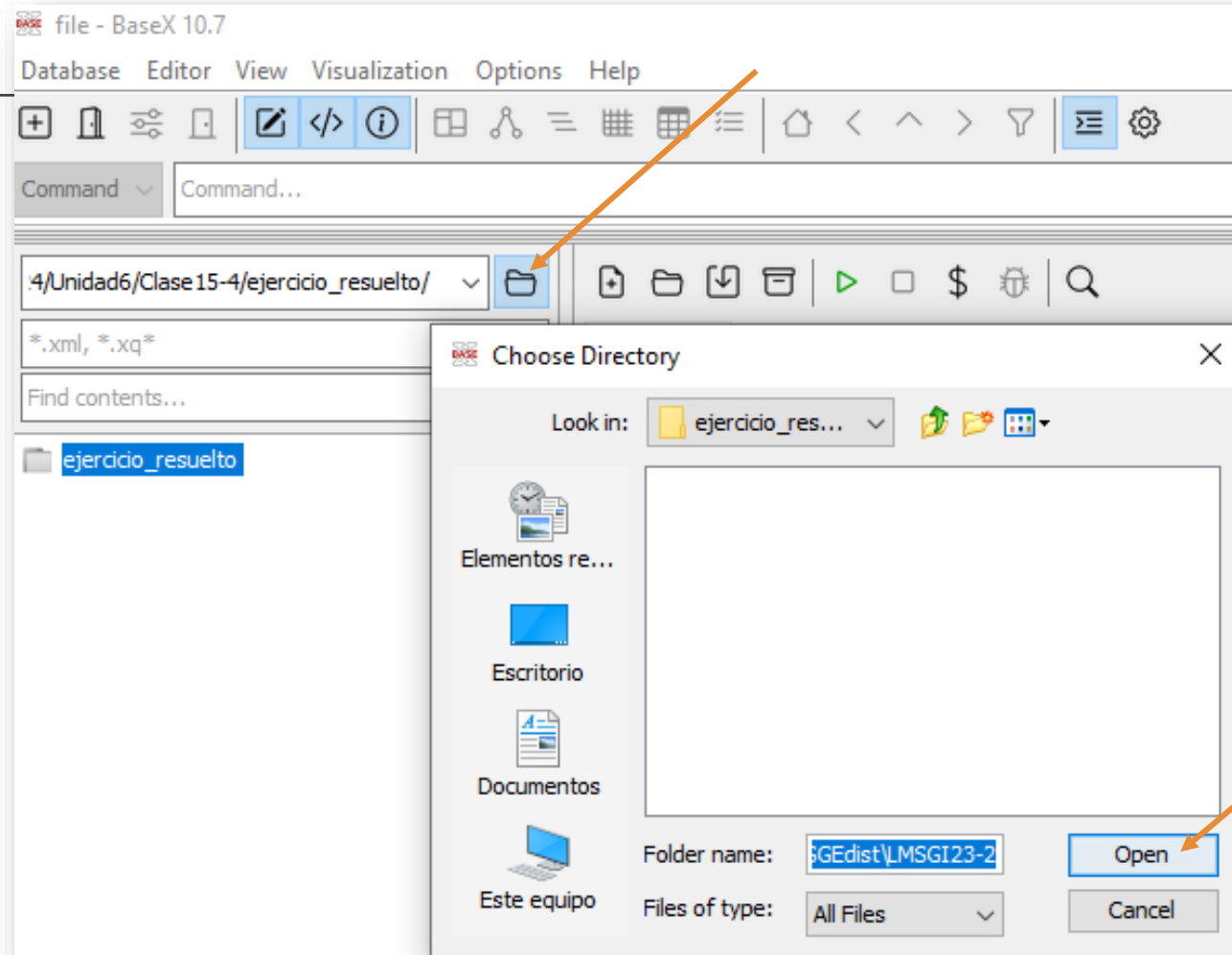
Utilicemos BaseX para

1. Crear la bd
2. Comprobar la ejecución de las consultas

[Descargar solución.](#)

# 3.- EJERCICIO PRÁCTICO

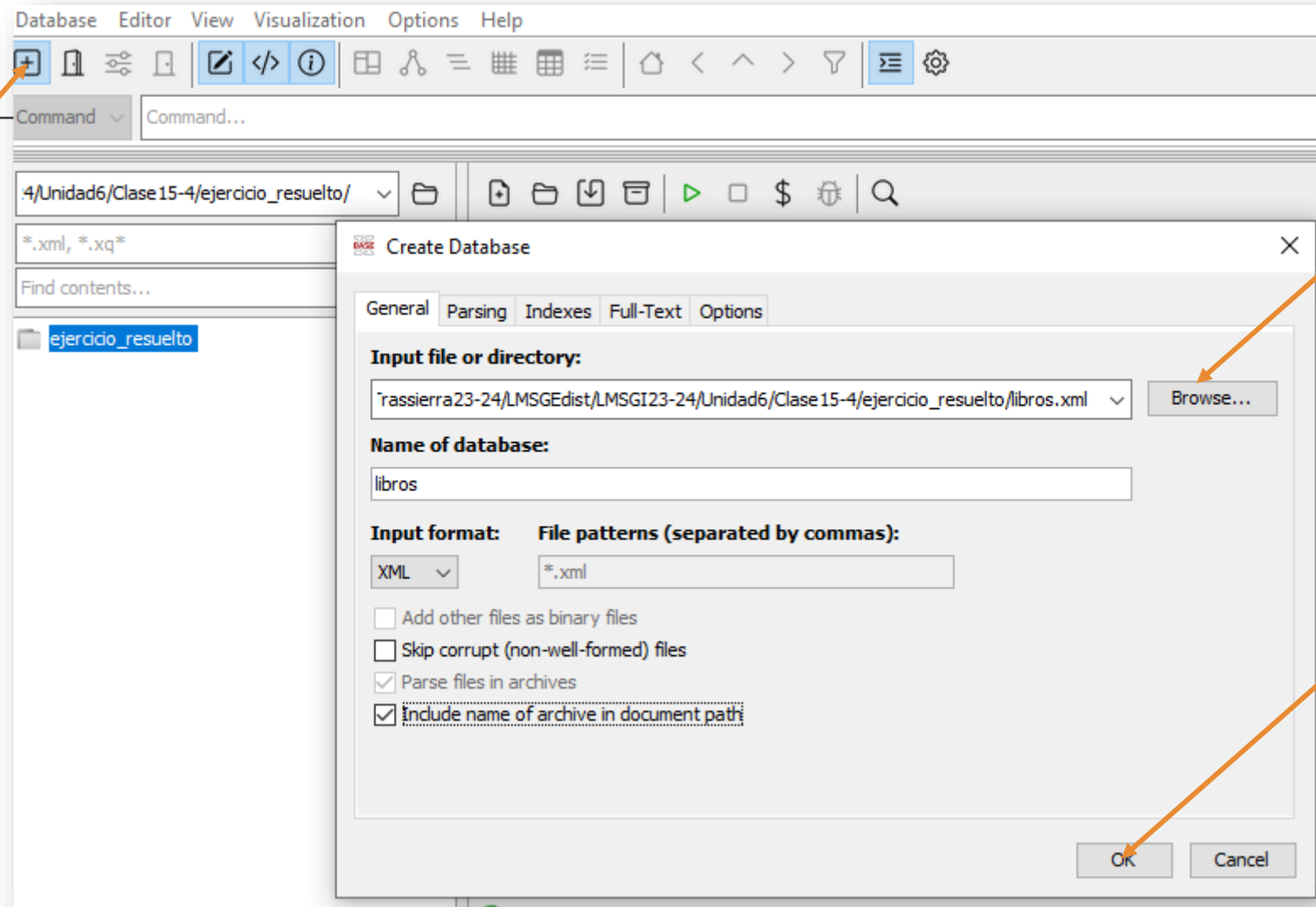
Abrimos el archivo donde tengamos nuestro archivo .xml del que crearemos la BD



Abrimos la carpeta donde Tengamos nuestro documento .xml

# 3.- EJERCICIO PRÁCTICO

Creamos la base de datos a partir del archivo .xml



# 3.- EJERCICIO PRÁCTICO

Abrimos el archivo donde tengamos nuestro archivo .xml del que crearemos la BD

The screenshot shows the BaseX 10.7 application window. The top menu bar includes Database, Editor, View, Visualization, Options, and Help. Below the menu is a toolbar with various icons. The left sidebar shows the file explorer with the file 'ejercicio\_resuelto' selected. The main editor area displays the XML content of 'libros.xml'. An orange arrow points from the file explorer to the XML content. The XML content is visualized as a tree structure with the following data:

libro	autor	titulo	apell...	nomb...	paginas	editori...
libro	autor	titulo	apell...	nomb...	paginas	editori...
libro	autor	titulo	apell...	nomb...	paginas	editori...
libro	autor	titulo	apell...	nomb...	paginas	editori...
libro	autor	titulo	apell...	nomb...	paginas	editori...
libro	autor	titulo	apell...	nomb...	paginas	editori...
libro	autor	titulo	apell...	nomb...	paginas	editori...

The bottom panel shows the XML code and the command used to create the database:

```
<!-- Archivo: libros.xml -->
<biblioteca>
  <libros>

    <libro publicacion="2003" edicion="2">
      <titulo>Learning XML</titulo>

      <autor>
        <apellido>Ray</apellido>
        <nombre>Erik T.</nombre>
      </autor>

      <editorial>O'Reilly</editorial>
```

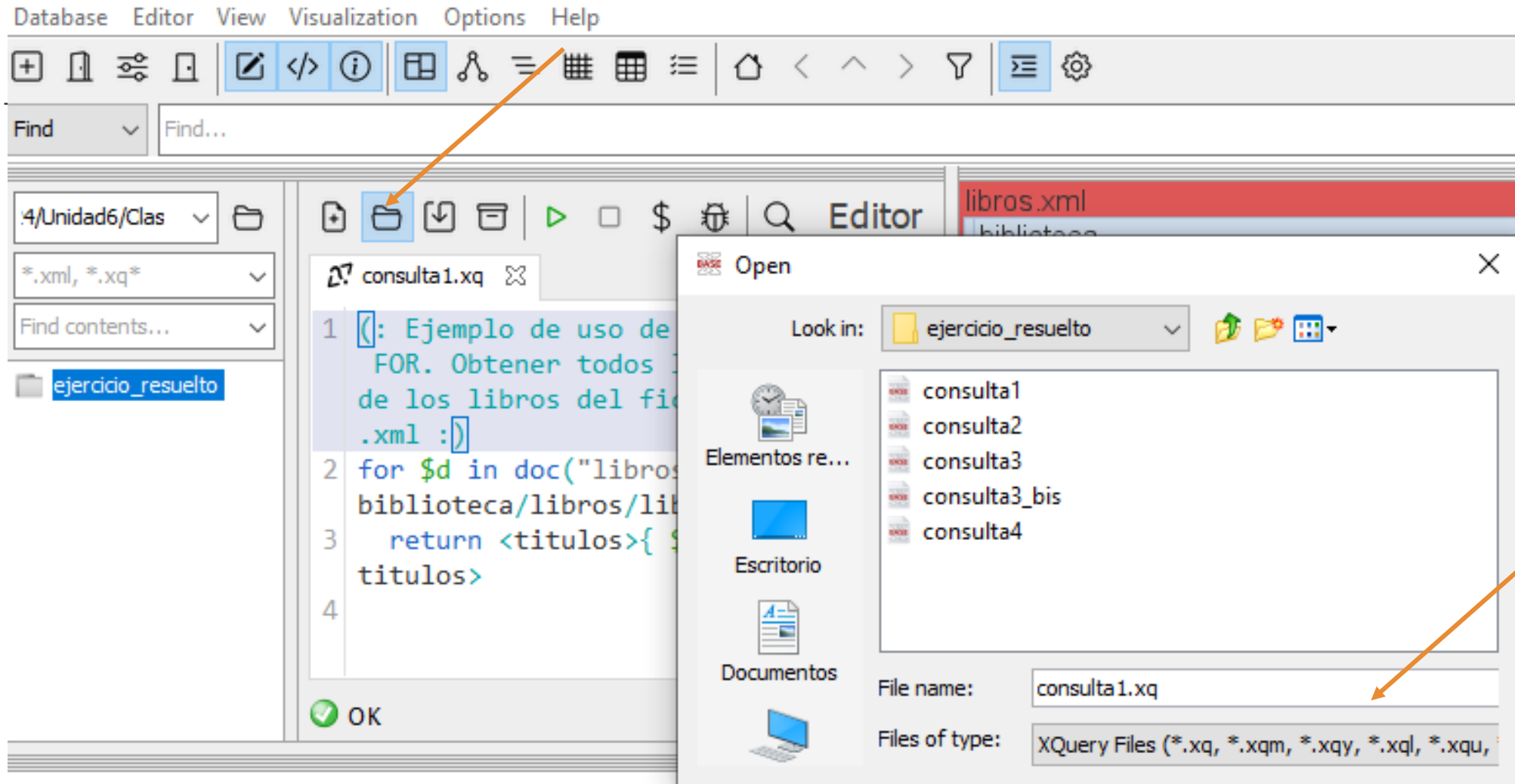
Command: SET ARCHIVENAME true

Command: CREATE DB libros G:/Mi unidad/Trassierra/Trassierra23-24/LMSGEdist/LMSGI23-24/Unidad6/Clase15-4/ejercicio\_resuelto/libros.xml

Result: Database 'libros' created in 71.09 ms.

# 3.- EJERCICIO PRÁCTICO

Comprobamos la ejecución de las consultas solucionadas que tenemos en el archivo descargado





# 3.- EJERCICIO PRÁCTICO

Ejecutamos las consultas una a una y vamos comprobando los resultados obtenidos

**Editor**

Context: db:get("libros")

```
1 (: Usamos la función doc() para cargar el documento XML
   'libros.xml'. Luego, seleccionamos todos los elementos
   'titulo' que están dentro de los elementos 'libro', '
   libros' y 'biblioteca' :)
2 for $d in doc("libros.xml")/biblioteca/libros/libro/
   titulo
3
4 (: Para cada elemento 'titulo', retornamos un nuevo
   elemento XML 'titulos' que contiene el título del libro
   :)
5 return <titulos>{ $d }</titulos>
```

**Result**

6 Results, 373 b

```
<titulos>
  <titulo>Learning XML</titulo>
</titulos>
<titulos>
  <titulo>XML Imprescindible</titulo>
</titulos>
<titulos>
  <titulo>XML Schema</titulo>
</titulos>
```

**libros.xml**

biblioteca

libros

libro	libro	libro	libro
titulo	autor	titulo	autor
Ray	apelli..	Eric	apelli..
edit..	pagi..	editor..	pagi..
416	832	400	

**Compiling:**

- inline for \$d\_0 in doc("libros.xml")/biblioteca/libros/libro/titulo
- simplify FLWOR expression: doc("libros.xml")/biblioteca/libros/libro

**Optimizing:**

- rewrite fn:doc(href): doc("libros.xml") -> document { "file:///C:/Users/Elena/Downloads/ejercicio\_resuelto/libros.xml" }

**Optimized Query:**

```
document { "file:///C:/Users/Elena/Downloads/ejercicio_resuelto/libros.xml" }
  <titulos>{ . }</titulos>
```

### 3.- EJERCICIO PRÁCTICO

Ejecutamos las consultas una a una y vamos comprobando los resultados obtenidos

4/Unidad6/Clase15
Context: db:get("libros")
Editor

\*.xml, \*.xq\*  
Find contents...

ejercicio\_resuelto

```

1 (: Ejemplo de uso de la cláusula LET. Obtener todos los
2   títulos de los libros del fichero libros.xml :)
3 (: Usamos la función doc() para cargar el documento XML
4   'libros.xml'. Luego, seleccionamos todos los elementos
5   'titulo' que están dentro de los elementos 'libro', '
6   libros' y 'biblioteca'. Asignamos el resultado a la
7   variable $d :)
8 let $d := doc("libros.xml")/biblioteca/libros/libro/
9   titulo
10
11 (: Retornamos un nuevo elemento XML 'titulos' que
12   contiene todos los títulos de los libros :)
13 return
14 <titulos>{ $d }</titulos>

```

OK 9 : 1

---

1 Result, 258 b
Result

```

<titulos>
  <titulo>Learning XML</titulo>
  <titulo>XML Imprescindible</titulo>
  <titulo>XML Schema</titulo>
  <titulo>XPath Essentials</titulo>
  <titulo>Beginning XSLT 2.0: Form Novice to Professional</titulo>
  <titulo>XQuery</titulo>
</titulos>

```

**Optimizing:**  
 - rewrite fn:doc(href): doc("libros.xml") -> document { "file:///C:/Users/Elena/Downloads/ejercicio\_resuelto/libros.xml" }  
 - inline let \$d\_0 := document { "file:///C:/Users/Elena/Downloads/ejercicio\_resuelto/libros.xml"/biblioteca/libros/libro/titulo  
 - simplify FLWOR expression: <titulos>{ document { "file:///C:/Users/Elena/Downloads/ejercicio\_resuelto/libros.xml"/biblioteca/libros/libro/titulo }</titulos>

**Optimized Query:**  
 <titulos>{ document { "file:///C:/Users/Elena/Downloads/ejercicio\_resuelto/libros.xml"/biblioteca/libros/libro/titulo }</titulos>

**Query:**  
 let \$d := doc("libros.xml")/biblioteca/libros/libro/titulo return

# 3.- EJERCICIO PRÁCTICO

Ejecutamos las consultas una a una y vamos comprobando los resultados obtenidos

```
consulta3.xq*
1 (: Ejemplo de uso de la cláusula FOR y LET juntas.
  Obtener todos los títulos de los libros del fichero
  libros.xml junto con los autores de cada libro :)
2
3 (: Usamos la función doc() para cargar el documento
  XML 'libros.xml'. Luego, seleccionamos todos los
  elementos 'libro'. :)
4 for $b in doc("libros.xml")//libro
5
6 (: Para cada libro, seleccionamos todos los elementos
  'autor' y los asignamos a la variable $c. :)
7 let $c := $b/autor
8
9 (: Finalmente, retornamos un nuevo elemento XML '
  libro' que contiene el título del libro y todos sus
  autores. :)
10 return
11 <libro>{ $b/titulo, <autores>{ $c }</autores>}</libro>
```

6 Results, 1370 b

Result

```
<libro>
  <titulo>Learning XML</titulo>
  <autores>
    <autor>
      <apellido>Ray</apellido>
      <nombre>Erik T.</nombre>
    </autor>
  </autores>
</libro>
<libro>
  <titulo>XML Imprescindible</titulo>
  <autores>
    <autor>
      <apellido>Harold</apellido>
      <nombre>Elliot Rusty</nombre>
    </autor>
    <autor>
      <apellido>Means</apellido>
      <nombre>W. Scott</nombre>
    </autor>
  </autores>
</libro>
<libro>
```

El archivo continúa con el resto de los libros

### 3.- EJERCICIO PRÁCTICO

Ejecutamos las consultas una a una y vamos comprobando los resultados obtenidos

consulta3\_bis.xq

```
1 (: Obtener todos los títulos de los libros del fichero
  libros.xml junto con los autores de cada libro. Usando
  solo la cláusula for:)
2 for $b in doc("libros.xml")//libro
3   return <libro>{ $b/titulo, <autores>{$b/autor}</
  autores>}</libro>
```

El archivo continúa con el resto de los libros

6 Results, 1370 b

Result

```
<libro>
  <titulo>Learning XML</titulo>
  <autores>
    <autor>
      <apellido>Ray</apellido>
      <nombre>Erik T.</nombre>
    </autor>
  </autores>
</libro>
<libro>
  <titulo>XML Imprescindible</titulo>
  <autores>
    <autor>
      <apellido>Harold</apellido>
      <nombre>Elliot Rusty</nombre>
    </autor>
    <autor>
      <apellido>Means</apellido>
      <nombre>W. Scott</nombre>
    </autor>
  </autores>
</libro>
<libro>
  <titulo>XML Schema</titulo>
  <autores>
    <autor>
      <apellido>van der Vlist</apellido>
      <nombre>Eric</nombre>
    </autor>
  </autores>
...
```

# 3.- EJERCICIO PRÁCTICO

Ejecutamos las consultas una a una y vamos comprobando los resultados obtenidos

```
consulta4.xq*
1 (: Obtener los títulos de los libros prestados con
  sus autores y las fechas de inicio y devolución del
  préstamo, ordenados por la fecha de inicio del
  préstamo :)
2 (: Usamos la función doc() para cargar los documentos
  XML 'libros.xml' y 'prestamos.xml'. Luego,
  seleccionamos todos los elementos 'libro' y 'entrada'
  , respectivamente. :)
3 for $t in doc("libros.xml")//libro, $e in doc("
  prestamos.xml")//entrada
4
5 (: Usamos una cláusula where para filtrar solo
  aquellos libros y entradas cuyos títulos son iguales.
  :)
6 where $t/titulo = $e/titulo
7 (: Usamos la cláusula order by para ordenar los
  resultados por la fecha de inicio del préstamo. :)
8 order by $e/prestamo/inicio
9 (: Finalmente, retornamos un nuevo elemento XML '
  prestamo' que contiene el título del libro, los
  autores del libro, y las fechas de inicio y
  devolución del préstamo. :)
10 return <prestamo>{ $t/titulo, $t/autor/*, $e/prestamo
  /inicio, $e/prestamo/devolucion }</prestamo>
```

3 Results, 660 b Result

```
<prestamo>
  <titulo>XML Imprescindible</titulo>
  <apellido>Harold</apellido>
  <nombre>Elliot Rusty</nombre>
  <apellido>Means</apellido>
  <nombre>W. Scott</nombre>
  <inicio>2011-02-12</inicio>
  <devolucion>2011-02-16</devolucion>
</prestamo>
<prestamo>
  <titulo>XPath Essentials</titulo>
  <apellido>Watt</apellido>
  <nombre>Adrew</nombre>
  <inicio>2011-02-23</inicio>
  <devolucion>2011-03-10</devolucion>
</prestamo>
<prestamo>
  <titulo>XML Imprescindible</titulo>
  <apellido>Harold</apellido>
  <nombre>Elliot Rusty</nombre>
  <apellido>Means</apellido>
  <nombre>W. Scott</nombre>
  <inicio>2011-05-02</inicio>
</prestamo>
```

# BIBLIOGRAFÍA

- **XQuery. Un lenguaje de consulta para XML.** Nieves Carralero Colmenar. IES Pedro Mercedes. [www.sociedadelainformacion.com](http://www.sociedadelainformacion.com)
  - Unidad 6. Almacenamiento de información. Apuntes FP Distancia
  - Xquery. J. J. Gutiérrez, M. J. Escalona, M. Mejías, J. Torres, D.Villadiego. Departamento de Lenguajes y Sistemas Informáticos. Escuela Técnica Superior de Ingeniería Informática. <chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://www.lsi.us.es/docs/informes/LSI-2005-02.pdf>
- Xquery. Jose Emilio Labra Gayo. Departamento de Informática. Universidad de Oviedo. [chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://di002.edv.uniovi.es/~labra/cursos/presentaciones/14\\_XQuery.pdf](chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://di002.edv.uniovi.es/~labra/cursos/presentaciones/14_XQuery.pdf)