

1. Utilización de XML para el almacenamiento de la información

1. Bases de datos relacionales.
2. Transformación a XML

2. Utilización de XML para el almacenamiento de la información

- Xml permite definir de manera rápida e intuitiva una representación de la información. La empresa X usará su SGBD para exportar sus datos a XML y se los remitirá a Y. Ambas empresas conocen esa representación de la información.

2.1- Bases de Datos Relacionales

- Vamos a verlo con un ejemplo. Imaginemos que queremos representar un conjunto de libros de una biblioteca. Vamos a almacenar el título, autor, editorial, edición, ISBN y numpaginas. La tabla se llama libros.

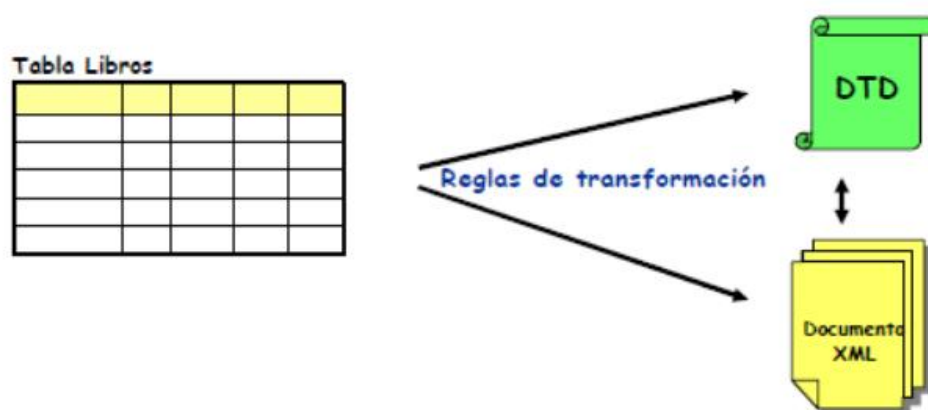
- **Tabla Libros**

Cod_libro	Titulo	Cod_Auto	Editorial	Edición	ISBN	NumPaginas
1	Don Quijote	1	Cátedra	3	97854321234	176
2	La Celestina	2	Maxtor	1	97811234455	320
3	Leyendas	3	Merida	21	97812333221	416

- **Tabla Autores**

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

2.2- Transformación a XML



2.2- Transformación a XML

- Aplicar transformación a XML para generar DTD o Esquema
 - 1.- Cada tabla del modelo relacional será un elemento dentro del DTD
 - <!DOCTYPE Libros[
 - <!ELEMENT Libros(libro)*>
 -]>
 - 2.- Cada tupla de la tabla se llama libro. Ahora se indica qué campos componen a la tupla
 - <!ELEMENT libro(Cod_libro, Titulo, Autores, Editorial, Edicion ISBN, Numpaginas)>
 - 3.- Para cada campo (columna), se debe establecer el tipo de dato almacenable (char, integer, etc)
 - <!ELEMENT Cod_libro (#PCDATA)>
 -

2.2- Transformación a XML

- 4.- Si existe una columna compleja (como puede ser la de "Autores"), se debe crear un nuevo elemento similar al del paso 2, indicando la multiplicidad y los tipos de datos a almacenar
 - <!ELEMENT Autores(autor)+>
 - <!ELEMENT autor(Cod_autor, Nombre, Apellidos, FechaNacimiento)>
 - <!ELEMENT Cod_Autor (#PCDATA)>
 -
- 5.- Si dos tablas están relacionadas entre sí a través de una tercera tabla de relación, se establece un nuevo elemento como en el paso 2 y se continua con los pasos siguientes
- La DTD y xml resultante será:
 - [Ejemplos\Ejemplo1.xml](#)

Ejemplo1.XML (con DTD y XML):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Libros [
  <!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)>
    ]>
<Libros>
  <libro>
    <Cod_Libro>1</Cod_Libro>
    <Titulo>Don Quijote de la Mancha </Titulo>
    <Editorial>Juan de la Cuesta </Editorial>
    <Edicion>3 </Edicion>
    <ISBN>9781234567 </ISBN>
    <NumPaginas>176 </NumPaginas>
    <Autores>
      <autor>
        <Cod_Autor>1 </Cod_Autor>
        <Nombre>Miguel </Nombre>
        <Apellidos>de Cervantes Saavedra
      </Apellidos>
      <FechaNacimiento>29/09/1547
    </FechaNacimiento>
    </autor>
  </Autores>
</libro>
  <libro>
    <Cod_Libro>2</Cod_Libro>
    <Titulo>La Celestina </Titulo>
    <Editorial>Maxtor </Editorial>
    <Edicion>1 </Edicion>
    <ISBN>9781235567 </ISBN>
    <NumPaginas>320 </NumPaginas>
```

```
<Autores>
  <autor>
    <Cod_Autor>2 </Cod_Autor>
    <Nombre>Fernando </Nombre>
    <Apellidos>de Rojas </Apellidos>
    <FechaNacimiento>01/01/1470
  </FechaNacimiento>
  </autor>
</Autores>
</libro>
</Libros>
```

3. Lenguajes de consulta y manipulación

- Los usuarios se comunican con un SGBD a través de un lenguaje de consulta, que el más popular es SQL. Este lenguaje permite realizar las siguientes operaciones:
 - Creación y borrado de tablas.
 - Inserción, modificación y borrado de tuplas.
 - Ejecución de búsquedas mediante consultas
- En XML el lenguaje que permite extraer y manipular información de igual manera que SQL es **XQuery**
- **XQuery**, permite acciones muy similares a SQL
- Cuando se va a analizar 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 este proceso, llegará un momento en el que el último nodo no tiene ningún hijo, lo que se denomina nodo hoja
- ¿Qué tipos de nodos se puede encontrar en ese recorrido?
 - **Nodo raíz** '/', es el primer nodo del documento. En nuestro ejemplo anterior, sería "Libros".
 - **Nodo elemento**: cualquier elemento de un documento XML. Cada nodo elemento posee un padre y puede o no poseer hijos. Si no tiene hijos, será un nodo hoja.
 - **Nodo texto**: cualquier elemento del documento que no esté marcado con una etiqueta de la DTD.
 - **Nodo atributo**: un nodo elemento podría tener etiquetas que complementen la información de ese elemento. Eso sería un nodo atributo.
- La extracción de la información de un fichero xml se realiza fácilmente con una tecnología denominada Xpath (XML Path).
- Xpath es la herramienta que utiliza XQuery para procesar el árbol de nodos de un documento XML

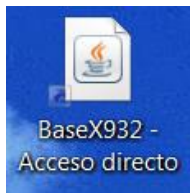
4.-XQUERY

- XQuery es un lenguaje de consulta similar a SQL que permite recorrer los documentos XML de manera que se pueda extraer y manipular la información contenida en el mismo.
- Es un lenguaje muy sencillo que no requiere de conocimientos de programación avanzados (no es necesario saber C/C++, Java, Python, Visual Basic Script, etc).
- Además XQuery es compatible con muchas de las tecnologías que estandariza W3C (como XML, Namespaces, los esquemas, XSLT y XPath).

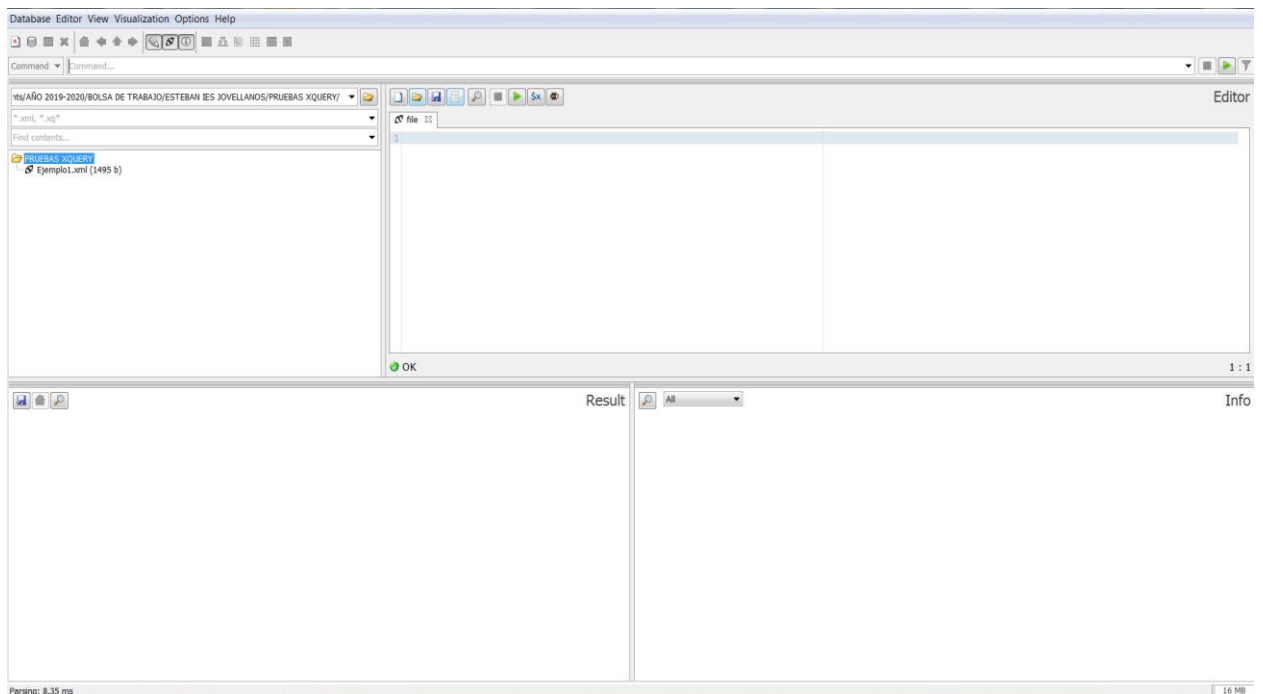
- Si hemos seguido los pasos anteriores, tenemos disponible la base de datos para hacer consultas. Vamos a utilizar una función de Xquery llamada “doc(<nombreDocumento.xml>)”, que permite extraer datos del documento XML almacenado. Vamos a ver como lo hacemos:
 - Pestaña Xquery
 - Escribimos doc(“nombredocumento.xml”)
 - Botón Execute, y se ve en la parte inferior “Messages”), si se ha ejecutado correctamente y el tiempo empleado para ello. Lo interesante es el resultado de la consulta que ha sido devuelta en la parte derecha de la aplicación.

A la vista de todo lo comentado hasta ahora, se hace necesaria la aparición de un lenguaje que permita definir de forma rápida y compacta, consultas o recorridos complejos sobre colecciones de datos en XML los cuales devuelvan todos los nodos que cumplan ciertas condiciones. Este lenguaje debe ser, además, declarativo, es decir, independientemente de la forma en que se realice el recorrido o de donde se encuentren los datos. Este lenguaje ya existe y se llama XQuery.

Utilizamos el software BASEX para gestionar XQUERY de una manera ágil, intuitiva y muy práctica. Para ello, debemos hacernos con un aplicativo y descargárnoslo según (documento anexo BASEX).

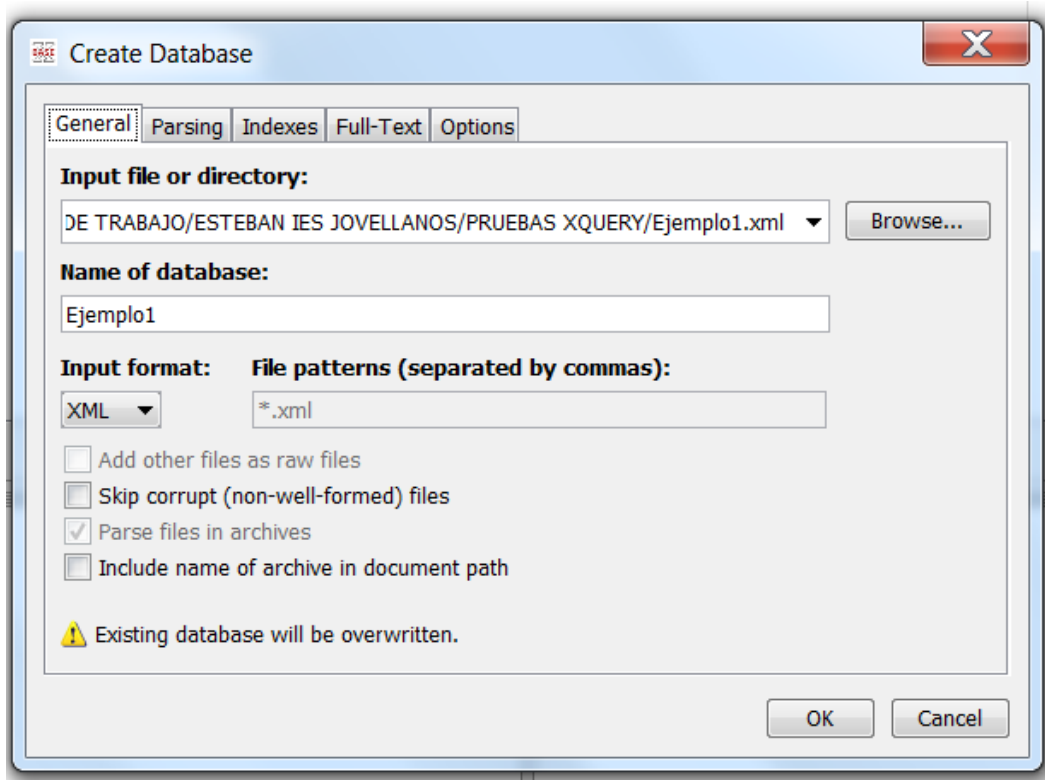


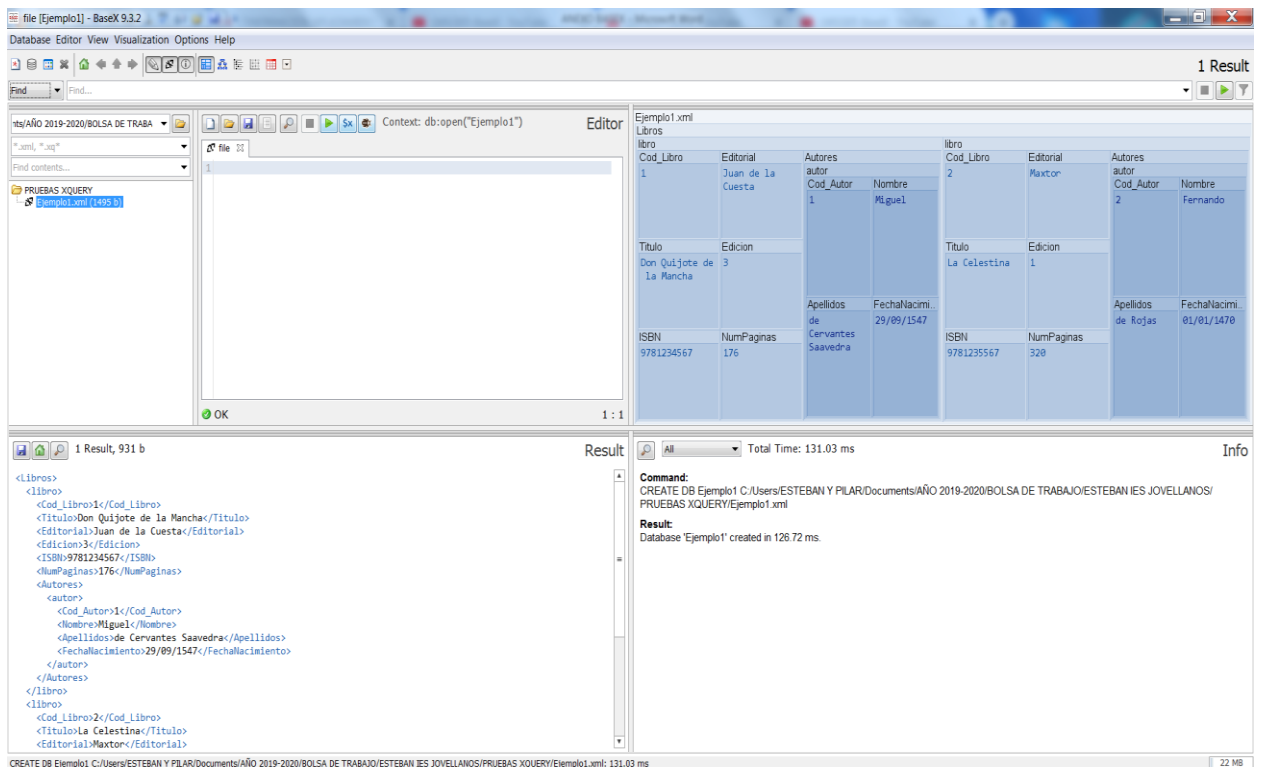
Pulsamos sobre el icono.



En la parte superior de la pantalla hay un apartado con el nombre de DATABASE. Si entramos en esta opción y optamos por NEW se muestra la pantalla siguiente:

Siempre debemos acceder a nuestro fichero .xml debidamente validado para considerarlo como una base de datos sobre la que hacer la gestión XQUERY posterior.



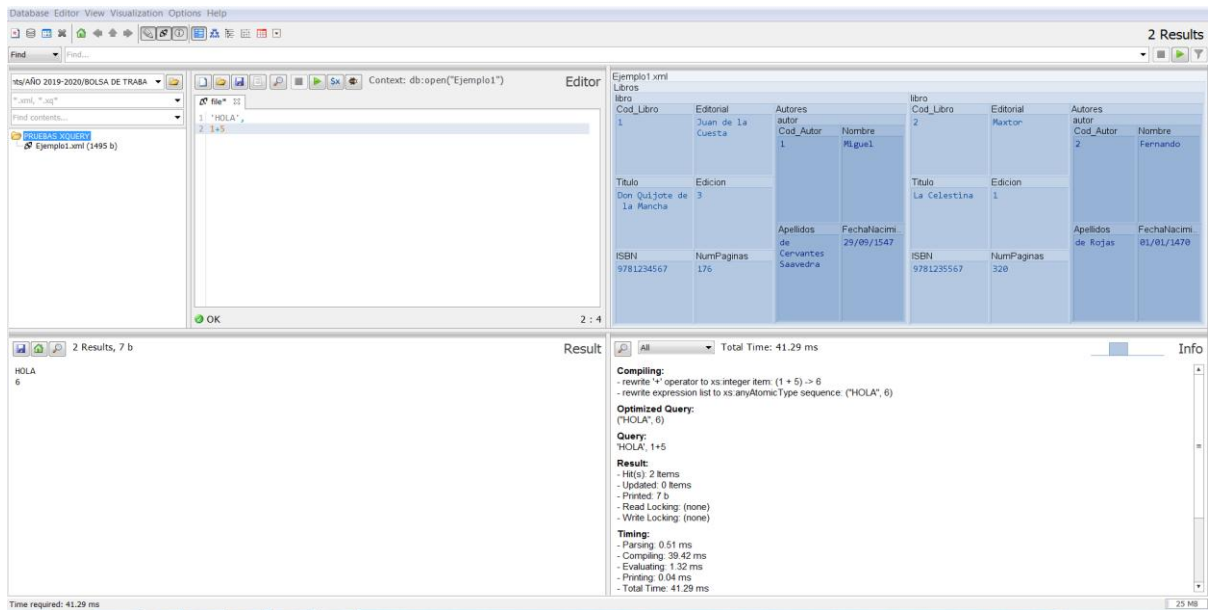


La pantalla se subdivide en la parte superior en tres zonas .
De izquierda a derecha :

1. Zona a modo de descripción de carpetas con sus contenidos
2. Zona de EDITOR donde introduciremos las sentencias para acceder a los datos. En esta zona manejaremos las sentencias que el gestor utiliza para conseguir la información de la base de datos.
3. Zona con una estructura de base de datos elaborada a través de la estructura de árbol creada por el documento Ejemplo1.xml previamente validado.

En la parte inferior hay otras dos zonas:
De izquierda a derecha :

1. Resultado del documento en XML
2. Datos anexos de información.



Las ventanas se pueden ocultar y visualizar la que queramos. Tenemos dos iconos en la barra superior para hacerlo.



En la zona de EDITOR es donde debemos operar para realizar las operaciones XQUERY. Podemos escribir texto, operaciones..etc. y se muestran en la zona de RESULT

<https://www.youtube.com/watch?v=zJpr6eeugXo>
<https://www.youtube.com/watch?v=LooC-ECVrWg>
<https://www.youtube.com/watch?v=UZD6sDJMeZU>

insertar

<https://www.youtube.com/watch?v=jJS-Ms9nubU>

<https://www.youtube.com/watch?v=cdNiqHZaMBg>

La manera más sencilla y potente de realizar búsquedas y selecciones de nodos concretos en un documento XML, es lo que se le llama expresiones **FLWOR**, que es la contracción del acrónimo **For**, **Let**, **Where**, **Order by** **Return**. Una expresión FLWOR para realizar una consulta, podría ser:

- **for** \$libro in doc("Ejemplo1.xml")//Libros/libro
- **where** \$libro/NumPaginas<300

➤ **return \$libro/Autores**

➤ Vamos a explicar para qué sirven cada una de las cláusulas **FLWOR**

➤ **for**: Esta sentencia permite seleccionar los nodos que se quieren consultar, guardándose en la variable (el identificador que le precede el símbolo \$).

➤ **let**: Esta clausula es opcional. Esta sentencia establece una nueva variable sobre el mismo u otro documento XML. Permite simplificar las expresiones posteriores y tener un código mucho más legible.

➤ **where**: Cláusula que permite establecer una condición sobre la variable indicada en "for" y "let".

➤ **orderby**: Cláusula que define el orden de presentación de resultados.

➤ **return**: Permite devolver un valor concreto de los resultados obtenidos de las anteriores cláusulas (uno por nodo).

FOR y LET sirven para crear las filas o tuplas con las que trabajará el resto de las cláusulas de la consulta y pueden usarse tantas veces como se desee en una consulta, incluso dentro de otras cláusulas.

Sin embargo solo puede declararse una única cláusula where, una única cláusula order by y una única cláusula return.

[Ejemplos sobre el fichero XML \(Ejemplo1.xml\).](#)

SENTENCIAS FLWOR CON EJEMPLOS

"TITULOS DE LIBROS", (línea de comentario entre "" y con carácter coma al final)

Nos muestra los títulos de los libros que contiene el fichero XML

"TITULOS DE LIBROS",
for \$a in doc("Ejemplo1.xml")/Libros/libro
return \$a/Titulo

Con la sentencia FOR nos recorremos cada uno de los elementos libro y recogemos sus datos en la variable \$a.

The screenshot shows the XQuery IDE with a query in the Editor pane:

```
1 "TITULOS DE LIBROS"
2 for $a in doc("Ejemplo1.xml")/Libros/libro
3 where $a/Edicion=3
4 return $a/Titulo
```

The Result pane shows 3 results, 91 b. The output is:

```
<TITULOS DE LIBROS>
<Titulo>Don Quijote de la Mancha</Titulo>
<Titulo>La Celestina</Titulo>
```

The right pane shows a table with book details:

libro	Cod_Libro	Editorial	Autores	Titulo	Edicion	ISBN	NumPaginas	Apellidos	FechaNacim
1	Juan de la Cuesta	de Cervantes Saavedra	1	Don Quijote de la Mancha	3	9781234567	176	29/09/1547	01/01/1478
2	Maxtor		2	La Celestina	1	9781235567	320		

The bottom pane shows the compiled query and timing information.

Nos muestra los libros con sus títulos y que la edición sea igual a 3.

"TITULOS DE LIBROS",
for \$a in doc("Ejemplo1.xml")/Libros/libro
where \$a/Edicion=3
return \$a/Titulo

The screenshot shows the XQuery IDE with a query in the Editor pane:

```
1 "TITULOS DE LIBROS"
2 for $a in doc("Ejemplo1.xml")/Libros/libro
3 where $a/Edicion=3
4 order by $a/Titulo
5 return $a/Titulo
```

The Result pane shows 2 results, 60 b. The output is:

```
<TITULOS DE LIBROS>
<Titulo>Don Quijote de la Mancha</Titulo>
```

The right pane shows a table with book details:

libro	Cod_Libro	Editorial	Autores	Titulo	Edicion	ISBN	NumPaginas	Apellidos	FechaNacim
1	Juan de la Cuesta	de Cervantes Saavedra	1	Don Quijote de la Mancha	3	9781234567	176	29/09/1547	01/01/1478
2	Maxtor		2	La Celestina	1	9781235567	320		

The bottom pane shows the compiled query and timing information.

Nos muestra los títulos de los libros que contiene el fichero XML ordenados por Título.

"TITULOS DE LIBROS",
for \$a in doc("Ejemplo1.xml")/Libros/libro
order by \$a/Titulo
return \$a/Titulo

The screenshot shows the XQuery IDE with a query editor on the left and a result pane on the right. The query is:

```
1 "TITULOS DE LIBROS";
2 for $a in doc("Ejemplo1.xml")/Libros/libro
3 order by $a/Titulo
4 return $a/Titulo
```

The result pane shows the output of the query, which is the title of each book. The result is:

```
TITULOS DE LIBROS:
<Titulo>Don Quijote de la Mancha</Titulo>
<Titulo>La Celestina</Titulo>
```

The result pane also shows the compiled query and the result details.

Nos muestra los autores de los libros que contiene el fichero XML.

"TITULOS DE LIBROS",
for \$a in doc("Ejemplo1.xml")/Libros/libro/Autores
return \$a/autor

The screenshot shows the XQuery IDE with a query editor on the left and a result pane on the right. The query is:

```
1 "TITULOS DE LIBROS";
2 for $a in doc("Ejemplo1.xml")/Libros/libro/Autores
3 order by $a/Nombre
4 return $a/autor
```

The result pane shows the output of the query, which is the author of each book. The result is:

```
TITULOS DE LIBROS:
<autor>
  <Cod_Autor>1</Cod_Autor>
  <Nombre>Miguel</Nombre>
  <Apellidos>de Cervantes Saavedra</Apellidos>
  <FechaNacimiento>29/09/1547</FechaNacimiento>
</autor>
<autor>
  <Cod_Autor>2</Cod_Autor>
  <Nombre>Fernando</Nombre>
  <Apellidos>de Rojas</Apellidos>
  <FechaNacimiento>01/01/1478</FechaNacimiento>
</autor>
```

The result pane also shows the compiled query and the result details.

Nos muestra los autores de libros ordenados por nombre y con datos de Apellidos.

"TITULOS DE LIBROS",
for \$a in doc("Ejemplo1.xml")/Libros/libro/Autores/autor
order by \$a/Nombre
return \$a/Apellidos

The screenshot shows the XQuery Editor and Results window. The Editor contains the following XQuery:

```

1 "TITULOS DE LIBROS",
2 for $a in doc("Ejemplo1.xml")/Libros/Libro/Autores/autor
3 order by $a/Nombre
4 return $a/Apellido

```

The Results window shows the output of the query, which is a table with the following data:

Titulo	Edicion	Apellidos	FechaNacim.
Don Quijote de la Mancha	3	Cervantes Saavedra	29/09/1547
La Celestina	1	de Rojas	81/01/1478

The Results window also displays the following information:

- 3 Results, 96 b
- Time required: 2.12 ms
- Compiling: - rewrite fn:doc(uri) to document-node() item: doc("Ejemplo1.xml") -> db:open-pre("Ejemplo1", 0)
- Optimized Query: ("TITULOS DE LIBROS", for \$a, 0 in db:open-pre("Ejemplo1", 0)/element("Libros")/element("Libro")/element("Autores")/element("autor") order by \$a/element("Nombre") empty least return \$a/element("Apellido"))
- Query: "TITULOS DE LIBROS", for \$a in doc("Ejemplo1.xml")/Libros/Libro/Autores/autor order by \$a/Nombre return \$a/Apellido
- Result: - Hits: 3 items - Updated: 0 items - Printed: 96 b - Read Locking: Ejemplo1.xml - Write Locking: (none)
- Timing: - Parsing: 0.54 ms - Compiling: 0.65 ms - Evaluating: 0.83 ms - Printing: 0.11 ms - Total Time: 2.12 ms

NUEVAS CONSULTAS

- Vamos a crear una nuevo documento XML y la añadiremos a la base de datos de BASEX. Vamos almacenar los datos de una academia de bailes de salón. La información a almacenar es:
 - Nombre del baile
 - Precio de la clase (indicando la periodicidad de la cuota y la moneda de pago)
 - Número de plazas disponibles
 - Fecha de comienzo de las clases
 - Fecha de finalización de las clases
 - Nombre del profesor que la imparte
 - Sala en la que se desarrollará la clase

Ejemplo2.XML (con DTD y XML):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Bailes
[
    <!ELEMENT Bailes (baile)*>
    <!ELEMENT baile (nombre, precio,plazas, comienzo, fin,
profesor, sala)>
    <!ELEMENT nombre (#PCDATA)>
    <!ELEMENT precio (#PCDATA)>
    <!ELEMENT plazas (#PCDATA)>
    <!ELEMENT comienzo (#PCDATA)>
    <!ELEMENT fin (#PCDATA)>
    <!ELEMENT profesor (#PCDATA)>
    <!ELEMENT sala (#PCDATA)>
    <!ATTLIST baile id ID #REQUIRED>
    <!ATTLIST precio cuota CDATA #REQUIRED >
    <!ATTLIST precio moneda CDATA #REQUIRED >
]>
<Bailes>
    <baile id="A1">
        <nombre>Tango</nombre>
        <precio cuota="mensual" moneda="euro">27</precio>
        <plazas>20</plazas>
        <comienzo>1/1/2020</comienzo>
        <fin>1/12/2020</fin>
        <profesor>Roberto Garcia</profesor>
        <sala>1</sala>
    </baile>
    <baile id="A2">
        <nombre>Cha-cha-cha</nombre>
        <precio cuota="trimestral" moneda="euro">80</precio>
        <plazas>18</plazas>
        <comienzo>1/2/2020</comienzo>
        <fin>31/07/2020</fin>
        <profesor>Miriam Gutierrez</profesor>
        <sala>1</sala>
    </baile>
    <baile id="A3">
        <nombre>Rock</nombre>
        <precio cuota="mensual" moneda="euro">30</precio>
        <plazas>15</plazas>
        <comienzo>1/1/2020</comienzo>
        <fin>1/12/2020</fin>
        <profesor>Laura Mendiola</profesor>
        <sala>1</sala>
    </baile>
```



```
<baile id="A4">
  <nombre>Merengue</nombre>
  <precio cuota="trimestral" moneda="dolares">75</precio>
  <plazas>12</plazas>
  <comienzo>1/1/2020</comienzo>
  <fin>1/12/2020</fin>
  <profesor>Jesús Lozano</profesor>
  <sala>2</sala>
</baile>
<baile id="A5">
  <nombre>Salsa</nombre>
  <precio cuota="mensual" moneda="euro">32</precio>
  <plazas>10</plazas>
  <comienzo>1/1/2020</comienzo>
  <fin>1/12/2020</fin>
  <profesor>Jesús Lozano</profesor>
  <sala>2</sala>
</baile>
<baile id="A6">
  <nombre>Pasodoble</nombre>
  <precio cuota="anual" moneda="euro">320</precio>
  <plazas>8</plazas>
  <comienzo>1/1/2020</comienzo>
  <fin>31/12/2020</fin>
  <profesor>Miriam Gutierrez</profesor>
  <sala>2</sala>
</baile>
</Bailes>
```

Accedemos de la misma forma que para el anterior XML

The screenshot shows the XQuery IDE interface. The left pane displays the file explorer with 'Ejemplo2.xml' selected. The main editor shows the XML structure of 'Ejemplo2.xml', which is a collection of dance classes (bailes) with details like name, price, start/end dates, and instructor. The bottom pane shows the command 'CREATE DB Ejemplo2' and the result 'Database "Ejemplo2" created in 136.61 ms.'.

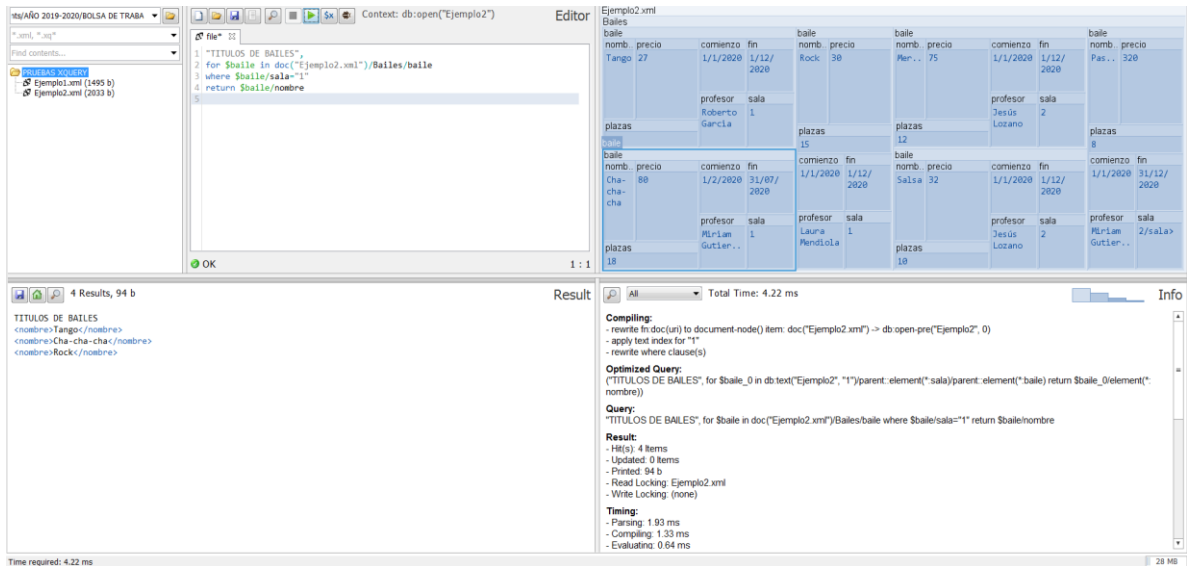
Bailes que se realizan en la academia

"TITULOS DE BAILES",
for \$a in doc("Ejemplo2.xml")/Bailes/baile
return \$a/nombre

The screenshot shows the XQuery IDE with the query '1 "TITULOS DE BAILES", 2 for \$a in doc("Ejemplo2.xml")/Bailes/baile 3 return \$a/nombre' entered. The bottom pane shows the result of the query, which is a list of dance titles: 'TITULOS DE BAILES', 'Tango', 'Cha-cha-cha', 'Rock', 'Merengue', 'Salsa', and 'Pasodoble'. The right pane shows the compiled query and the result set.

Se necesita saber qué bailes se realizan en la sala número 1:

```
"TITULOS DE BAILES",
for $baile in doc("Ejemplo2.xml")/Bailes/baile
where $baile/sala="1"
return $baile/nombre
```



Editor

```
1 "TITULOS DE BAILES",
2 for $baile in doc("Ejemplo2.xml")/Bailes/baile
3 where $baile/sala="1"
4 return $baile/nombre
```

4 Results, 94 b

TITULOS DE BAILES

```
<nombre>Tango</nombre>
<nombre>Cha-cha-cha</nombre>
<nombre>Rock</nombre>
```

Result

Compiling:

- rewrite fn:doc(uri) to document-node() item: doc("Ejemplo2.xml") -> db:open-pre("Ejemplo2", 0)
- apply text index for "1"
- rewrite where clause(s)

Optimized Query:

"TITULOS DE BAILES", for \$baile_0 in db:text("Ejemplo2", "1")/parent:element("sala")/parent:element("baile") return \$baile_0/element("nombre")

Query:

"TITULOS DE BAILES", for \$baile in doc("Ejemplo2.xml")/Bailes/baile where \$baile/sala="1" return \$baile/nombre

Result:

- Hit(s): 4 Items
- Updated: 0 Items
- Printed: 94 b
- Read Locking: Ejemplo2.xml
- Write Locking: (none)

Timing:

- Parsing: 1.93 ms
- Compiling: 1.33 ms
- Evaluating: 0.64 ms

Time required: 4.22 ms

Vemos que los resultados salen con las etiquetas de XML. Si quisiéramos solo tener un listado, obviando las etiquetas “<nombre>” y “</nombre>”, simplemente tendríamos que indicar en la clausula “return” que extraiga los datos de ese elemento XML.

```
"TITULOS DE BAILES",
for $baile in doc("Ejemplo2.xml")/Bailes/baile
let $n:=$baile/nombre
where $baile/sala="1"
return data($n)
```

The screenshot shows the XQuery IDE with a query in the Editor pane. The query is:

```
1 'TITULOS DE BAILES',
2 for $baile in doc('Ejemplo2.xml')/Bailes/baile
3 let $n:=$baile/nombre
4 where $baile/sala='1'
5 return data($n)
```

The Result pane shows the output of the query:

```
TITULOS DE BAILES
Tango
Cha-cha-cha
Rock
```

The Info pane shows the compiled query and the result set.

Ahora los resultados ya no están con las etiquetas del elemento “nombre”. Todo es debido a que existe una función llamada “data(<variable>)”, que extrae del nodo la información almacenada.

Se necesita extraer los nodos de aquellos bailes que se impartan en la sala número 2 y cuyo precio sea menor que 35 euros.

"TITULOS DE BAILES",
for \$baile in doc("Ejemplo2.xml")/Bailes/baile
let \$n:=\$baile/nombre
where \$baile/sala="2" and \$baile/precio<35 and
\$baile/precio[@moneda="euro"]
return data(\$n)

The screenshot shows the XQuery IDE with a query in the Editor pane. The query is:

```
1 'TITULOS DE BAILES',
2 for $baile in doc('Ejemplo2.xml')/Bailes/baile
3 let $n:=$baile/nombre
4 where $baile/sala='2' and $baile/precio<35 and $baile/precio[@moneda='euro']
5 return data($n)
```

The Result pane shows the output of the query:

```
TITULOS DE BAILES
Salsa
```

The Info pane shows the compiled query and the result set.

Se necesita saber el nombre de los profesores que dan clases con cuotas mensuales

```
"TITULOS DE BAILES",
for $baile in doc("Ejemplo2.xml")/Bailes/baile
let $p:=$baile/profesor
where $baile/precio[@cuota="mensual"]
return data($p)
```

The screenshot shows the XQuery IDE with the following components:

- Editor:** Contains the XQuery code:


```
1 "TITULOS DE BAILES",
2 for $baile in doc("Ejemplo2.xml")/Bailes/baile
3 let $p:=$baile/profesor
4 where $baile/precio[@cuota="mensual"]
5 return data($p)
```
- Result:** Displays 4 results:

TITULOS DE BAILES
Roberto García
Laura Mendiola
Jesús Lozano
Miriam Gutierrez
- XML Document:** Shows the structure of 'Ejemplo2.xml' with elements like 'baile', 'precio', 'profesor', and 'plazas'.
- Compiling/Query Info:**
 - Compiling:** Lists transformations like 'rewrite fn doc(uri) to document-node()', 'rewrite = comparison to simplified = comparison', etc.
 - Optimized Query:** Shows the optimized version of the query.
 - Query:** Shows the final query after optimization.
 - Result:**
 - Items: 4 items
 - Updated: 0 items
 - Printed: 64 b

PRÁCTICAS

Para todas las prácticas tenéis que presentar el programa hecho junto con el resultado en un único documento word con vuestro nombre .

1. Siguiendo el ejemplo XML relativo a contenidos de bailes y sus elementos asociados . Mostrar los datos de los bailes ordenados por precio y cuya moneda sea euro.
2. Siguiendo el ejemplo XML relativo a contenidos de bailes y sus elementos asociados Mostrar los nombres de los profesores y la sala en la que dan clase, ordénalos por sala.
3. Siguiendo el ejemplo XML relativo a contenidos de bailes y sus elementos asociados. Mostrar cuántas plazas en total oferta el profesor "Jesus Lozano".

NOTA:

Para aquellos que tengáis más interés en temas de XQUERY y XSLT os adjunto una serie de videos en youtube de Pepe Lluyot muy interesantes.

XQUERY

<https://www.youtube.com/watch?v=zJpr6eeugXo>
<https://www.youtube.com/watch?v=LooC-ECVrWg>
<https://www.youtube.com/watch?v=UZD6sDJMeZU>

XSLT

https://www.youtube.com/watch?v=BMR4C8w_S7w
<https://www.youtube.com/watch?v=cdNjqHZaMBg>