

# XQuery

Sitio: [Aula Virtual do IES de Teis](#)

Curso: Linguaxes de Marcas e Sistemas de Xestión de Información 2021-22  
(DAM-A)

Libro: XQuery

Impreso por: Deivid Durán Durán

Data: Venres, 24 de Xuño de 2022, 00:22

# Táboa de contidos

## **1. XQuery**

- 1.1. Sintaxis XQuery
- 1.2. Expresiones FLWOR
- 1.3. Herramientas: BaseX
- 1.4. Selección de nodos
- 1.5. Añadimos HTML
- 1.6. La función data()
- 1.7. if-then-else
- 1.8. La cláusula for
- 1.9. La cláusula for y variables posicionales at
- 1.10. La cláusula let
- 1.11. La cláusula where
- 1.12. La cláusula order by
- 1.13. Créditos y referencias
- 1.14. Funciones
- 1.15. Operadores
- 1.16. Más operadores

## **2. Almacenamiento de datos XML**

- 2.1. Opciones de almacenamiento de datos XML
- 2.2. Bases de datos XML Nativas

# 1. XQuery

- Lenguaje diseñado para consultar datos en archivos XML
- Se podría decir XQuery es a XML como SQL a las Bases de Datos Relacionales
- Se construye sobre expresiones XPath
- Puede usarse para:
  - Extraer información para usar en servicios web
  - Generar resúmenes e informes
  - Transformar XML en (X)HTML
  - Buscar dentro de documentos web para obtener información de interés

## 1.1. Sintaxis XQuery

- XQuery es sensible a mayúsculas y todos los elementos, atributos y variables deben ser identificadores válidos XML.
- Las cadenas de caracteres pueden delimitarse tanto por comillas simples ('cadena') como por comillas dobles ("cadena").
- Las variables se definen con un símbolo de dólar \$ seguido del nombre de la variable. Por ejemplo ***\$contador***.
- Los comentarios se delimitan mediante (: para la apertura y :) para el cierre.

**(: esto es un comentario en Xquery :)**

## 1.2. Expresiones FLWOR

- **FLWOR** es un acrónimo para **For, Let, Where, Order by, Return**. Se pronuncia como *flower* (flor) en inglés.
- Cada una de las instrucciones se utiliza para algo concreto:
  - **For** – Selecciona una secuencia de nodos.
  - **Let** – Asigna un valor a una variable.
  - **Where** – Establece una condición que filtra los nodos.
  - **Order by** – Realiza una ordenación de los nodos que han pasado la condición.
  - **Return** – Valor de retorno, se evalúa una vez por cada nodo

**Han de comenzar por al menos una cláusula FOR o una LET.**

**Debe existir al menos una cláusula return.**

**Where y Order son opcionales**, pero si existen, han de respetar escrupulosamente el orden dado por el nombre, **FLWOR**.

## 1.3. Herramientas: BaseX

Para ver los ejemplos de XQuery, vamos a trabajar con [BaseX](#).

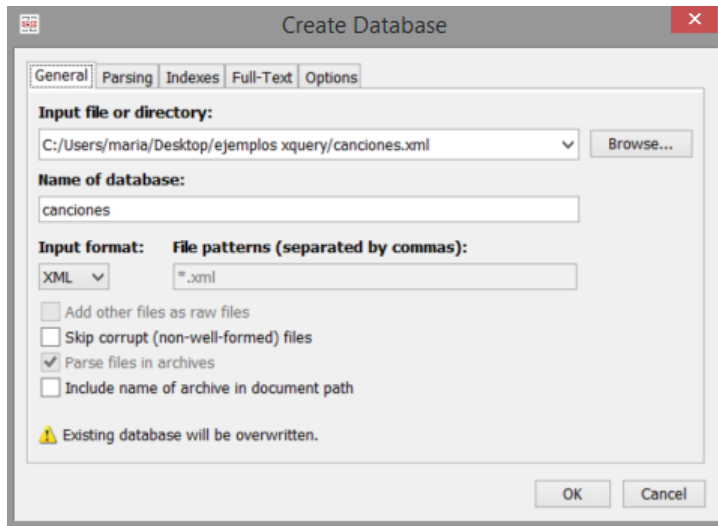
- Se trata de una herramienta que permite almacenar documentos XML como si de una base de datos se tratase.
- Es Open Source y multiplataforma.
- Cuenta con una interfaz de usuario gráfica.
- Requiere tener instalado Java 8.

Para instalarla, realizamos una instalación por defecto de la plataforma adecuada desde su página [BaseX](#)

Una vez instalada, podéis ver este vídeo que os explica cómo trabajar con BaseX. Si no tenéis audio, podéis activar los subtítulos.

[https://youtu.be/Z9En\\_dh0v1I](https://youtu.be/Z9En_dh0v1I)

Para ver los ejemplos, crearemos una nueva base de datos y seleccionaremos la ubicación donde guardaremos el archivo con el que trabajaremos: [canciones.xml](#). Todos los ejemplos con los que trabajaremos se encuentran en **Recursos:** [Ejemplos de XQuery](#) > Ejemplos FLWOR



A continuación, podremos crear nuestras consultas XQuery en el editor. Las guardaremos con extensión **.xq** en la misma ubicación que el fichero **canciones.xml** para poder hacer referencia al propio documento xml simplemente por su nombre.

Database Editor View Visualization Options Help

Find Find... 3 Results

C:/Program Files (x86)/BaseX/ Context: db:open("canciones") Editor

ejemplo4.xq

```
1 (: ejemplo 4 :)  
2 for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo  
3 where $i/puntuacion>8  
4 return $i/canción
```

OK 1 : 1

3 Results, 104 b

Result

<canción>Hangar 18</canción>  
<canción>Peace Sells</canción>  
<canción>Master of Puppets</canción>

MiBibliotecaMP3

archivo	archivo	archivo	archivo	archivo	archivo
canc.. artis..	canc.. artis..	canc.. artis..	canc.. artis..	canc.. artis..	canc.. artis..
Me..	Me..	Me..	Me..	An..	Me..
disco	punt..	disco	punt..	disco	punt..
9	9	10	8	8	8

All Total Time: 23.54 ms

Info

Compiling:

- rewrite fn:doc(uri) to document-node() item: doc("canciones.xml") -> db:open-pre("canciones", 0)
- rewrite > comparison to range comparison: (\$i\_0/puntuacion > 8) -> \$i\_0/puntuacion >= 8.000000000000002
- rewrite to predicate: puntuacion >= 8.000000000000002
- inline for \$i\_0 in db:open-pre("canciones", 0)/MiBibliotecaMP3/archivo[puntuacion >= 8.000000000000002]
- simplify FLWOR expression: db:open-pre("canciones", 0)/MiBibliotecaMP3/archivo[puntuacion >= 8.000000000000002]/canción

Optimized Query:

db:open("canciones", "canciones.xml")/MiBibliotecaMP3 123 MB

## 1.4. Selección de nodos

- XQuery utiliza funciones para extraer los datos de los documentos XML.
- Para abrir un documento se usa la función doc()
- Para navegar a través de un documento se usan expresiones de ruta. Por ejemplo:

```
for $c in doc("canciones.xml")/MiBibliotecaMP3/archivo/canción
return $c
```



5 Results, 189 b

```
<canción>Hangar 18</canción>
<canción>Peace Sells</canción>
<canción>Master of Puppets</canción>
<canción>Among The Living</canción>
<canción>For Whom The Bell Tolls</canción>
```

- Podemos utilizar predicados en la propia ruta:

```
for $c in doc("canciones.xml")/MiBibliotecaMP3/archivo/canción[../puntuacion>7]
return $c
```



5 Results, 189 b

```
<canción>Hangar 18</canción>
<canción>Peace Sells</canción>
<canción>Master of Puppets</canción>
<canción>Among The Living</canción>
<canción>For Whom The Bell Tolls</canción>
```

- O filtrar usando la cláusula where con el mismo resultado:

```
for $c in doc("canciones.xml")/MiBibliotecaMP3/archivo/canción
where $c/./puntuacion > 7
return $c
```



5 Results, 189 b

```
<canción>Hangar 18</canción>
<canción>Peace Sells</canción>
<canción>Master of Puppets</canción>
<canción>Among The Living</canción>
<canción>For Whom The Bell Tolls</canción>
```

Fijémonos que el uso de la variable \$c se refiere a cada uno de los elementos canción indicados en la cláusula for y fija el contexto a partir del cual comenzarán las demás rutas XPath.



## 1.5. Añadimos HTML

Si queremos crear un documento HTML como resultado, integrando el resultado de una consulta XQuery, incluiremos el código XQuery entre llaves {}. Esto se puede ver en el ejemplo5.xq

The screenshot displays a software interface with an XQuery editor and a results pane. The editor shows the following XQuery code:

```
1 <html>
2 <head>
3 <title>Ejemplo 5</title>
4 </head>
5
6 <body>
7 <ol>
8 {
9 for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo
10 where $i/puntuacion>8
11 order by $i/puntuacion
12 return <li> {$i/canción}({$i/puntuacion}) </li>
13 }
14 </ol>
15 </body>
16 </html>
17
```

Red arrows point to the opening curly brace on line 8, the XQuery code block on lines 9-13, and the closing curly brace on line 13. The results pane shows the generated HTML output:

```
<html>
<head>
<title>Ejemplo 5</title>
</head>
<body>
<ol>
<li>
<canción>Master of Puppets</canción>(<
puntuacion>10</puntuacion>) </li>
<li>
<canción>Hangar 18</canción>(<puntuacion>9</
puntuacion>) </li>
<li>
<canción>Peace Sells</canción>(<puntuacion>9<
/puntuacion>) </li>

```

The results pane also displays a table of the underlying XML data:

canciones.xml					
MiBibliotecaMP3					
archivo	canción	artista	archivo	canción	artista
archivo	canción	artista	archivo	canción	artista
Hangar 18		Megad..	Peace Sells		Megad..
disco	puntuac..	disco	puntuaci..	disco	puntuacion
Rust in Peace	9		9	Anthrax	8
archivo	canción	disco	puntuaci..	archivo	canción
archivo	canción	artista	disco	puntuaci..	artista
		Metal..	10	Metallica	8

## 1.6. La función data()

Para evitar el uso de las etiquetas de apertura y cierre usamos la **función data()** => **ejemplo6.xq**

The screenshot shows the BaseX editor with the XQuery code for **ejemplo6.xq**. The code is as follows:

```
1 <html>
2 <head>
3 <title>Ejemplo 6</title>
4 </head>
5
6 <body>
7 <ol>
8 {
9 for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo
10 where $i/puntuacion>8
11 order by $i/puntuacion
12 return <li> {data($i/canción)}{data($i/puntuacion)} </li>
13 }
14 </ol>
15 </body>
16 </html>
17
```

Two red arrows point to the `{data($i/canción)}` and `{data($i/puntuacion)}` expressions in line 12.

The result of the query is displayed in the "Result" pane, showing 1 result with 209 bytes. The result is an HTML document with the following structure:

```
<html>
<head>
<title>Ejemplo 6</title>
</head>
<body>
<ol>
<li>Master of Puppets(10) </li>
<li>Hangar 18(9) </li>
<li>Peace Sells(9) </li>
</ol>
</body>
</html>
```

The "canciones.xml" document is also shown, containing the following data:

archivo	canción	artista	disco	puntuacion
archivo	canción	artista	disco	puntuacion
Hangar 18	Peace Sells	Megad..		
disco	Rust in Peace			9
archivo	canción	artista	disco	puntuacion
disco	Anthrax			8
archivo	canción	artista	disco	puntuacion
disco	Metal..			10
archivo	canción	artista	disco	puntuacion
disco	Metallica			8

En **ejemplo7.xq** se usa la **función data()** para obtener el valor de un atributo y aplicarlo como una clase css:

C:/Program Files (x86)/BaseX/

\*.xml, \*.xq

Find contents...

BaseX

bin

data

etc

ico

lib

repo

src

webapp

BaseX.jar (4049 kB)

CHANGELOG (46 kB)

LICENSE (1524 b)

readme.txt (1602 b)

uninst.exe (81 kB)

ejemplo5.xq

ejemplo6.xq

ejemplo7.xq

Context: db:open("canciones2")

Editor

```

1 <html>
2 <head>
3 <title>Ejemplo 7</title>
4 </head>
5 <body>
6 <ol>
7 {
8   for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo
9   where $i/puntuacion>8
10  order by $i/puntuacion
11  return <li class="{data($i/@almacenado)}"> {data($i/canción)}{data($i/
    puntuacion)} </li>
12 }
13 </ol>
14 </body>
15 </html>
16

```

OK 1 : 1

1 Result, 254 b

Result

```

<html>
  <head>
    <title>Ejemplo 7</title>
  </head>
  <body>
    <ol>
      <li class="DISC01">Master of Puppets(10) </li>
      <li class="DISC01">Hangar 18(9) </li>
      <li class="DISC02">Peace Sells(9) </li>
    </ol>
  </body>
</html>

```

canciones.xml

MiBibliotecaMP3

archivo	archivo	archivo	archivo	archivo	disco
canción	artista	canción	artista	canción	
Hangar 18	Megad..	Peace Sells	Megad..		
				artista	puntuacion
disco	puntuac..	disco	puntuaci..	Anthrax	8
Rust in Peace	9		9		
				archivo	disco
archivo				canción	
canción	artista	disco	puntuaci..		
	Metal..		10	artista	puntuacion
				Metallica	8

Total Time: 4.83 ms

## 1.7. if-then-else

Es posible incorporar en XQuery la expresión **if then else**

La expresión **requiere paréntesis** alrededor de la condición del if

Al contrario que en otros lenguajes **es obligatorio añadir la parte de else** aunque no haya acción asociada, en ese caso podemos dejar simplemente else ().

En el **ejemplo8.xq** se crea una tabla para el DiSCO1 y otra para el DISCO2:

The screenshot shows the BaseX editor with the file `ejemplo8.xq` open. The code defines an XQuery that generates an HTML page with two tables. The first table, titled "DISCO 1", lists artists and their discos based on a query of the `canciones.xml` database. The second table, titled "DISCO 2", lists artists and their discos based on another query of the same database. The code uses the `if-then-else` expression to conditionally include data.

```
1 <html>
2 <head>
3 <title>Ejemplo 8</title>
4 </head>
5
6 <body>
7 <table>
8 <caption>DISCO 1 </caption>
9 <tr><td>Artista</td><td>Disco</td></tr>
10 {
11 for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo
12 order by $i/puntuacion
13 return if ($i/@almacenado="DISCO1")
14 then <tr><td>{data($i/artista)}</td><td>{data($i/disco)}</td></tr>
15 else ()
16 }
17 </table>
18
19 <table>
20 <caption>DISCO 2 </caption>
21 <tr><td>Artista</td><td>Disco</td></tr>
22 {
23 for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo
```

The result of the query is shown in the "Result" pane, displaying the generated HTML code. The output shows two tables: "DISCO 1" and "DISCO 2". "DISCO 1" contains one row with the artist "Megadeth" and the disco "9". "DISCO 2" contains one row with the artist "Metallica" and the disco "10".

archivo	canción	disco
artista	puntuacion	
Megadeth	9	

archivo	canción	disco
artista	puntuacion	
Metallica	10	

Tenéis un ejemplo de if-then-else completo en el **ejemplo9.xq**:

C:/Program Files (x86)/BaseX/

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

BaseX

bin
data
etc
ico
lib
repo
src
webapp
BaseX.jar (4049 kB)
CHANGELOG (46 kB)
LICENSE (1524 b)
readme.txt (1602 b)
uninst.exe (81 kB)

Context: db:open("canciones2")
Editor
ejemplo9.xq

```

1 <html>
2 <head>
3 <title>Ejemplo 9</title>
4 </head>
5
6 <body>
7 <table>
8 <caption>CANCIONES POR DISCO </caption>
9 <tr><td>Artista</td><td>Nombre disco</td><td>Grabada en</td></tr>
10 {
11 for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo
12 order by $i/puntuacion
13 return if ($i/@almacenado="DISCO1")
14 then <tr><td>{data($i/artista)}</td><td>{data($i/disco)}</td>
15      <td>DISCO1</td></tr>
16 else <tr><td>{data($i/artista)}</td><td>{data($i/disco)}</td>
17      <td>DISCO2</td></tr>
18 }
19 </table>
20 </body>
21 </html>
22

```

OK 16 : 39

1 Result, 847 b

<html>

<head>

<title>Ejemplo 9</title>

</head>

<body>

<table>

<caption>CANCIONES POR DISCO </caption>

<tr>

<td>Artista</td>

<td>Nombre disco</td>

<td>Grabada en</td>

</tr>

<tr>

Result

canciones.xml

MiBibliotecaMP3

archivo

canción

disco

Hangar 18

artista

puntuacion

Megadeth 9

archivo

canción

disco

artista

puntuacion

Megadeth 9

archivo

canción

disco

artista

puntuacion

10

archivo

canción

disco

artista

puntuacion

8

db:open("canciones2", "canciones.xml")/MiBibliotecaMP3/archivo

134 MB

Este último ejemplo, puede refactorizarse utilizando el if-then-else solo en la celda de datos que cambiaría, como muestra el fichero **ejemplo9-b.xq**:

```
1 <html>
2 <head>
3 <title>Ejemplo 9</title>
4 </head>
5
6 <body>
7 <table>
8 <caption>CANCIONES POR DISCO </caption>
9 <tr><td>Artista</td><td>Nombre disco</td><td>Grabada en</td></tr>
10 {
11 for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo
12 order by $i/puntuacion
13 return
14 <tr><td>{data($i/artista)}</td><td>{data($i/disco)}</td>
15 {
16 if ($i/@almacenado="DISCO1")
17 then
18 <td>DISCO1</td>
19 else
20 <td>DISCO2</td>
21 }
22 </tr>
23 }
24 </table>
25 </body>
26 </html>
27
```

```
<html>
<head>
<title>Ejemplo 9</title>
</head>
<body>
<table>
<caption>CANCIONES POR DISCO </caption>
<tr>
<td>Artista</td>
<td>Nombre disco</td>
<td>Grabada en</td>
</tr>
<tr>
<td>Metallica</td>
<td>Master of Puppets</td>
<td>DISCO1</td>
</tr>
<tr>
<td>Anthrax</td>
<td>Among The Living</td>
<td>DISCO2</td>
</tr>
<tr>
<td>Metallica</td>
<td>Ride The Lightning</td>
<td>DISCO1</td>
</tr>
<tr>
<td>Megadeth</td>
<td>Rust in Peace</td>
<td>DISCO1</td>
</tr>
```

## 1.8. La cláusula for

La cláusula **for** realiza una iteración de elementos según se haya indicado en su definición.

Para especificar un número determinado de iteraciones se puede usar la palabra clave **to**. Ver **ejemplo11.xq**:

The screenshot shows the BaseX editor with a file named `ejemplo11.xq` open. The code in the editor is:

```
1 (: ejemplo 11 :)  
2  
3 for $i in (1 to 10)  
4 return $i  
5  
6  
7
```

The result viewer shows 10 results, 29 bytes. The results are displayed as a table with columns: archivo, canción, artista, disco, puntuación. The data is as follows:

archivo	canción	artista	disco	puntuación
Hangar 18	Peace Sells	Megadeth	Among The Living	8
disco	Rust in Peace	puntuación	disco	puntuación
Master of Puppets	Master of Puppets	Metallica	10	8

- Pueden existir varias expresiones en la cláusula for y funcionarían como **bucles for anidados**. Ver **ejemplo13.xq**:

Find contents...

BaseX

bin

data

etc

ico

lib

repo

src

webapp

BaseX.jar (4049 kB)

CHANGELOG (46 kB)

LICENSE (1524 h)

1

2

3 for \$i in (1 to 5), \$j in (1,2,3)

4 return <resultado>i es {\$i} j es {\$j}</resultado>

5

6

OK

1 : 1

15 Results, 568 b

Result

canciones.xml

MiBibliotecaMP3

archivo

artista

archivo

artista

archivo

disco

canción

artista

canción

artista

canción

disco

Hangar 18

Megad..

Peace Sells

Megad..

Among The Living

Among The Living

disco

puntuac..

disco

puntuaci..

disco

puntuacion

Rust in Peace

9

9

Anthrax

8

archivo

canción

disco

archivo

canción

artista

puntuacion

Master of Puppets

Metal..

Master of Puppets

10

Metallica

8



## 1.9. La cláusula for y variables posicionales at

Puesto que la variable asociada o ligada al bucle hace referencia al elemento en el que estamos iterando en cada momento (archivo en el siguiente ejemplo) no podemos usarla para contabilizar las iteraciones. Para ello se usa la palabra clave **at**. Ver **ejemplo12.xq**:

En ella se utiliza la **variable posicional \$j** que señalará el mismo valor que la función **position()** del nodo ligado a la variable al for (\$i en el ejemplo) **dentro del conjunto de nodos de la expresión XPath por la que itera el for** ( en el siguiente caso: `doc("canciones.xml")/MiBibliotecaMP3/archivo`)

```
1 <html>
2 <head>
3 <title>Ejemplo 12</title>
4 </head>
5
6 <body>
7 <ul>
8 {
9 for $i at $j in doc("canciones.xml")/MiBibliotecaMP3/archivo
10 where $i/puntuacion>8
11 order by $i/puntuacion
12 return <li>{$j}. {data($i/canción)}({data($i/puntuacion)}) </li>
13 }
14 </ul>
15 </body>
16 </html>
```

OK

1 Result, 219 b

```
<html>
  <head>
    <title>Ejemplo 12</title>
  </head>
  <body>
    <ul>
      <li>3. Master of Puppets(10) </li>
      <li>1. Hangar 18(9) </li>
      <li>2. Peace Sells(9) </li>
    </ul>
  </body>
</html>
```

En este caso se itera por todos los nodos **archivo** del documento (hay un total de 5 nodos).

Se filtran aquellos nodo **archivo** cuya puntuación sea >8, que casualmente son los 3 primeros nodos.

Finalmente el resultado se ordena por el subelemento **puntuacion**, pero observamos que, en lugar de forma ascendente (como sería lo esperado por defecto), se muestra en orden descendente. Esto se debe a que **se están ordenando los números como caracteres (cadenas de texto)** y no como números.

Para forzar que se ordenen como números, deberemos forzar la conversión de **puntuacion** a número mediante la función **number()**

Esta modificación se puede observar en **ejemplo12-number().xq**, donde se ordena de forma ascendente por **puntuacion**:

```

ejemplo12-number0.xq
1 <html>
2 <head>
3 <title>Ejemplo 12</title>
4 </head>
5
6 <body>
7 <ul>
8 {
9 for $i at $j in doc("canciones.xml")/MiBibliotecaMP3/archivo
10 where $i/puntuacion>8
11 order by number($i/puntuacion)
12 return <li>{$j}. {data($i/canción)}{data($i/puntuacion)} </li>
13 }
14 </ul>
15 </body>
16 </html>

```

```

<html>
<head>
<title>Ejemplo 12</title>
</head>
<body>
<ul>
<li>1. Hangar 18(9) </li>
<li>2. Peace Sells(9) </li>
<li>3. Master of Puppets(10) </li>
</ul>
</body>
</html>

```

Veamos un ejemplo más sobre el impacto de at, **cambiando la condición de la cláusula where** y buscando los archivos con **puntuacion <=8**. Esto puede verse en **ejemplo12-b.xq**:

```

ejemplo12-b.xq
1 <html>
2 <head>
3 <title>Ejemplo 12</title>
4 </head>
5
6 <body>
7 <ul>
8 {
9 for $i at $j in doc("canciones.xml")/MiBibliotecaMP3/archivo
10 where $i/puntuacion<=8
11 order by number($i/puntuacion)
12 return <li>{$j}. {data($i/canción)}{data($i/puntuacion)} </li>
13 }
14 </ul>
15 </body>
16 </html>

```

```

<html>
<head>
<title>Ejemplo 12</title>
</head>
<body>
<ul>
<li>4. Among The Living(8) </li>
<li>5. For Whom The Bell Tolls(8) </li>
</ul>
</body>
</html>

```

Se itera de nuevo por **todos los nodos archivo** del documento (hay un total de 5 nodos), pero el filtro de la cláusula where obligará a obtener solo los elementos archivo cuya puntuación sea <=8. Se corresponden con los 2 últimos nodos del total seleccionado en la expresión XPath que acompaña al for. (De ahí que la variable posicional \$j tome los valores 4 y 5). Finalmente el resultado se muestra ordenado por puntuación, que en este caso es igual en ambos archivos. El orden de presentación será entonces dependiente de la implementación del procesador de XQuery.

Un último ejemplo se puede encontrar en **ejemplo12-c.xq**, donde **se ha sustituido la cláusula where por un predicado** en la expresión XPath que acompaña al **for**. Esto provocará que el filtro se lleve a cabo antes de vincular la variable \$i a cada nodo archivo, por lo que el conjunto de nodos iniciales se verá ya reducido a aquellos nodos cuya puntuación sea <=8, es decir, 2 nodos en total:

```

ejemplo12-c.xq
1 <html>
2 <head>
3 <title>Ejemplo 12</title>
4 </head>
5
6 <body>
7 <ul>
8 {
9 for $i at $j in doc("canciones.xml")/MiBibliotecaMP3/archivo[puntuacion<=8]
10
11 order by number($i/puntuacion)
12 return <li>{$j}. {data($i/canción)}{data($i/puntuacion)} </li>
13 }
14 </ul>
15 </body>
16 </html>

```

```

<html>
<head>
<title>Ejemplo 12</title>
</head>
<body>
<ul>
<li>1. Among The Living(8) </li>
<li>2. For Whom The Bell Tolls(8) </li>
</ul>
</body>
</html>

```

En este caso, la variable posicional \$j toma los valores 1 y 2 en lugar de 4 y 5 pues su valor va a depender del número de integrantes del conjunto de nodos iniciales marcados por la expresión XPath que acompaña a su correspondiente for.

## 1.10. La cláusula let

Permite declarar una variable y asignarle un valor, una secuencia de valores, un nodo o conjunto de nodos mediante una expresión XPath.

Entre el nombre de la variable y su valor se usan **:=**

**Ver ejemplo14.xq**

The screenshot shows a query editor with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with folders like bin, data, etc, ico, lib, repo, src, and webapp, and files like BaseX.jar, CHANGELOG, LICENSE, readme.txt, and uninstd.exe. The code editor contains the following XQuery script:

```
1
2
3 let $i := (1 to 10)
4 let $j := (1,2,3)
5 let $k := 1
6 return <resultado>i es {$i} j es {$j} k es {$k}</resultado>
7
8
```

Below the code editor, there is a status bar with a green checkmark and the text "OK". At the bottom, there is a results pane showing the output of the query. The results pane has a tab labeled "Result" and shows the following XML output:

```
<resultado>i es 1 2 3 4 5 6 7 8 9 10 j es 1 2 3 k es
1</resultado>
```

On the right side of the results pane, there is a table titled "canciones.xml" with the following data:

MiBibliotecaMP3					
archivo		archivo		archivo	
canción	artista	canción	artista	canción	disco
Hangar	Megad..	Peace	Megad..		

También se puede encontrar una cláusula let dentro de un for. De este modo, la cláusula «let» se ejecuta **una vez por cada nodo**, al igual que hace la cláusula «return». **Ver ejemplo21.xq**

C:/Program Files (x86)/BaseX/

Find contents...

BaseX

bin

data

etc

ico

lib

repo

src

webapp

BaseX.jar (4049 kB)

CHANGELOG (46 kB)

LICENSE (1524 b)

readme.txt (1602 b)

uninst.exe (81 kB)

Context: db:open("canciones2")

Editor

4 : 46

1 for \$c in doc("canciones.xml")/MiBibliotecaMP3/archivo

2 let \$l := string-length(\$c/canción)

3 order by \$l descending

4 return (\$c/canción,<longitud>{\$l}</longitud>)

10 Results, 313 b

Result

canciones.xml

MiBibliotecaMP3

archivo		archivo		archivo	
canción	artista	canción	artista	canción	disco
Hangar 18	Megad...	Peace Sells	Megad...		
				artista	puntuacion
				Anthrax	8
disco	puntuac..	disco	puntuaci..	archivo	
Rust in Peace	9		9	canción	disco

## 1.11. La cláusula where

Es posible establecer condiciones compuestas con los conectores **and**, **or** y **not**.

Ver ejemplos 15, 16 y 17:

\*.xml, \*.xq

Find contents...

BaseX

bin

data

etc

ico

lib

repo

src

webapp

BaseX.jar (4049 kB)

CHANGELOG (46 kB)

LICENSE (1524 b)

readme.txt (1602 b)

uninst.exe (81 kB)

canciones.xml

ejemplo15.xq

```
1 <html>
2 <head>
3 <title>Ejemplo 15</title>
4 </head>
5 <body>
6 <ol>
7 {
8 for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo
9 where $i/puntuacion=8 or $i/puntuacion=10
10 order by $i/puntuacion
11 return <li class="{data($i/@almacenado)}"> {data($i/canción)}({data($i/
12 puntuacion)}) </li>
13 }
14 </ol>
15 </body>
16 </html>
```

OK

1 : 1

1 Result, 274 b

Result

```
<html>
  <head>
    <title>Ejemplo 15</title>
  </head>
  <body>
    <ol>
      <li class="DISC01">Master of Puppets(10) </li>
      <li class="DISC02">Among The Living(8) </li>
      <li class="DISC01">For Whom The Bell Tolls(8) </li>
    </ol>
  </body>
</html>
```

canciones.xml

MiBibliotecaMP3

archivo	archivo	archivo	archivo	archivo
canción	artista	canción	artista	canción
Hangar	Megad..	Peace	Megad..	disco
18		Sells		
				artista
				puntuacion
disco	puntuac..	disco	puntuaci..	8
Rust in	9		9	
Peace				archivo
				canción
archivo				disco
archivo				
canción	artista	disco	puntuaci..	artista
	Metal..		10	puntuacion



## 1.12. La cláusula order by

Es posible ordenar por varios criterios y especificar si queremos ordenación ascendente ( ascending ) o descendente ( descending): **Ver ejemplo18.xq**

The screenshot displays a software interface with an XQuery editor and its results. The editor on the right contains the following XQuery:

```
1 <html>
2 <head>
3 <title>Ejemplo 18</title>
4 </head>
5
6 <body>
7 <table>
8 <caption>CANCIONES POR DISCO </caption>
9 <tr><td>Artista</td><td>Nombre</td><td>Grabada en</td></tr>
10 {
11 for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo
12 order by $i/artista ascending, $i/canción descending
13 return <tr><td>{data($i/artista)}</td><td>{data($i/canción)}</td>
14 <td>{data($i/@almacenado)}</td></tr>
15 }
16 </table>
17 </body>
```

The results pane on the left shows the generated HTML table:

```
<table>
  <caption>CANCIONES POR DISCO </caption>
  <tr>
    <td>Artista</td>
    <td>Nombre</td>
    <td>Grabada en</td>
  </tr>
  <tr>
    <td>Anthrax</td>
    <td>Among The Living</td>
    <td>DISCO2</td>
  </tr>
  <tr>
```

The 'Result' pane on the right shows the raw XML data:

archivo	canción	artista	disco	puntuacion
Hangar 18	Peace Sells	Megad...		
Rust in Peace			9	
				10

## 1.13. Créditos y referencias

Para elaborar este material se ha seguido y adaptado el material [CC BY-SA 4.0](#)

**"Introducción a XQuery"** de Jorge Castellanos Vega y los ejemplos allí referenciados de <http://cloud.educa.madrid.org/index.php/s/EPc1nNZtYrm5zZL>.

Por lo tanto, este material también se distribuye bajo la misma licencia [CC BY-SA 4.0](#)

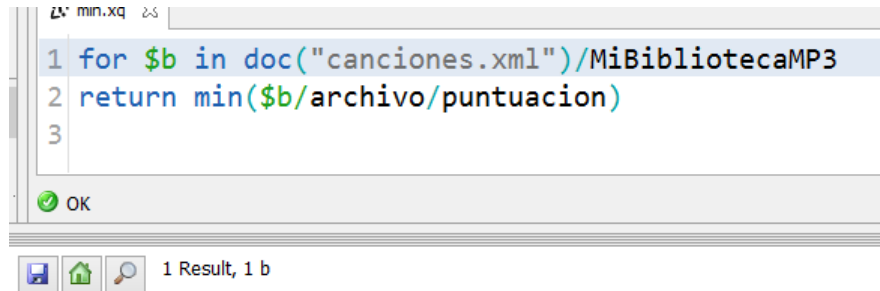


## 1.14. Funciones

Se pueden utilizar las mismas funciones matemáticas, de cadenas de texto y lógica booleana que vimos en XPath y XSLT.

A continuación se muestran los ejemplos recogidos en [Ejemplos de XQuery](#) > **Ejemplos funciones** con algunas que no habíamos visto en la UD5 y que también podemos usar en XQuery:

- `min()` o `max()`, devuelven respectivamente el mínimo y el máximo de los valores de los nodos dados.



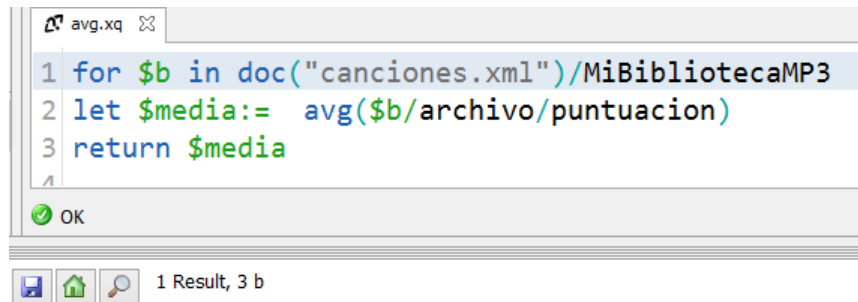
```
1 for $b in doc("canciones.xml")/MiBibliotecaMP3
2 return min($b/archivo/puntuacion)
3
```

OK

1 Result, 1 b

8

- `avg()`, calcula el valor medio de los valores dados.



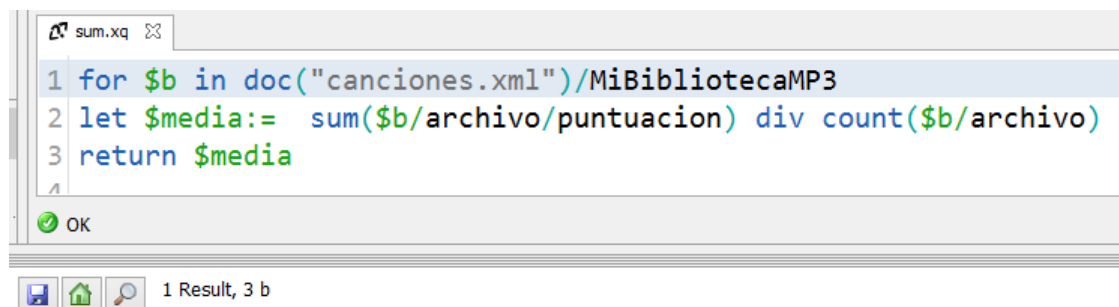
```
1 for $b in doc("canciones.xml")/MiBibliotecaMP3
2 let $media:= avg($b/archivo/puntuacion)
3 return $media
```

OK

1 Result, 3 b

8.8

- `sum()`, calcula la suma total de una cantidad de `items` dados.



```
1 for $b in doc("canciones.xml")/MiBibliotecaMP3
2 let $media:= sum($b/archivo/puntuacion) div count($b/archivo)
3 return $media
```

OK

1 Result, 3 b

8.8

- `upper-case()`, `lower-case()`, devuelve la cadena dada en mayúsculas o minúsculas respectivamente

```
upper-case.xq
1 for $c in doc("canciones.xml")/MiBibliotecaMP3/archivo/canción
2 return upper-case($c)
3
```

OK

5 Results, 84 b

HANGAR 18  
PEACE SELLS  
MASTER OF PUPPETS  
AMONG THE LIVING  
FOR WHOM THE BELL TOLLS

- `empty()`, devuelve "true" cuando la secuencia dada no contiene ningún elemento

```
exists.xq empty.xq*
1 for $b in doc("canciones.xml")/MiBibliotecaMP3
2 return
3 if (empty($b/archivo[puntuacion<5]))
4 then
5 "No existen canciones suspensas"
6 else("Existen canciones suspensas" )
7
```

OK

1 Result, 30 b

No existen canciones suspensas

- `exists()`, devuelve "true" cuando una secuencia contiene, al menos, un elemento

```
exists.xq
1 for $b in doc("canciones.xml")/MiBibliotecaMP3
2 return
3 if (exists($b/archivo[puntuacion<5]))
4 then "Existen canciones suspensas"
5 else( "No existen canciones suspensas" )
6
7
```

OK

1 Result, 30 b

No existen canciones suspensas

- `distinct-values()`, extrae los valores de una secuencia de nodos y crea una nueva secuencia con valores únicos, eliminando los nodos duplicados.

upper-case.xq

distinct-values.xq

```
1 for $p in distinct-values(doc("canciones.xml")/MiBibliotecaMP3/archivo/puntuacion)
2 return $p
```

OK

3 Results, 8 b

Result

9  
10  
8|

- **data()**, devuelve el valor de los elementos que recibe como argumentos, es decir, sin etiquetas de apertura y cierre.

upper-case.xq

data.xq\*

```
1 for $c in doc("canciones.xml")/MiBibliotecaMP3/archivo/canción
2 return data($c)
```

OK

5 Results, 84 b

Hangar 18  
Peace Sells  
Master of Puppets  
Among The Living  
For Whom The Bell Tolls

Existen más funciones disponibles relacionadas con comparaciones de cadenas de texto, fechas y horas, manipulación de nodos XML (Extensible Markup Language, que significa Lenguaje Extensible de Marcado), manipulación de secuencias, comprobación y conversión de tipos de datos. Se puede consultar una [Referencia de funciones XQuery de w3schools URL](#) , también disponible en Recursos.

## 1.15. Operadores

Existen varios operadores:

Comparadores de valores únicos	Comparadores generales de secuencias	Descripción
<code>eq</code>	<code>=</code>	igual
<code>ne</code>	<code>!=</code>	no igual
<code>lt</code>	<code>&lt;</code>	menor que
<code>le</code>	<code>&lt;=</code>	menor o igual que
<code>gt</code>	<code>&gt;</code>	mayor que
<code>ge</code>	<code>&gt;=</code>	mayor o igual que

La diferencia entre los comparadores generales y los de valores únicos reside en que los comparadores de valores **únicos** requieren que a ambos lados del operador **haya un único valor**. Los comparadores generales pueden trabajar **con secuencias a ambos lados**.

Los siguientes ejemplos se encuentran en la carpeta [Ejemplos de XQuery](#) > **Ejemplos operadores**:

```
=.xq
1 let $a:= 1
2 let $b:=(1,2)
3 return ($a = $b),
4
5 let $a:= 1
6 let $b:=(2,3)
7 return ($a = $b),
8
9 let $a:= (1,2)
10 let $b:=(2,3)
11 return ($a = $b)
12
13
```

OK

3 Results, 17 b

```
true
false
true
```

En la primera comparación, se comprueba si al menos un valor de la secuencia (1, 2) es igual a 1. Como esto es cierto, el resultado es true.  
En la segunda comparación, se comprueba si al menos un valor de la secuencia (2, 3) es igual a 1. Como esto no es cierto, el resultado es false.  
En la tercera comparación, se comprueba si al menos un valor de la secuencia (1, 2) existe en la segunda secuencia (2,3). Como esto es cierto para el número 2, el resultado es true.

Si probamos a hacer algo parecido con `eq`, nos encontraremos con un error en `2-eq1.xq`:

```
eq1.xq
1 let $a:= 1
2 let $b:=(1,2)
3 return ($a eq $b)
4
```

Item expected, sequence found: (1, 2).

Deberemos comparar valores únicos, como por ejemplo en `3-eq2.xq`:

eq1.xq eq2.xq eq3.xq

```
1 let $a:= 1
2 let $b:= 2
3 return ($a eq $b)
4
5
```

OK

1 Result, 5 b

false

## 1.16. Más operadores

- **Comparación de nodos:** Comparan la identidad de dos nodos.
  - **is**, devuelve true si las dos variables que actúan de operandos están ligadas al mismo nodo.
  - **is not**, devuelve true si las dos variables no están ligadas al mismo nodo.

Se puede encontrar un ejemplo en **4-is.xq**:



```
1 let $i := doc("canciones.xml")/
  MiBibliotecaMP3/archivo[puntuacion=
10]
2 let $j := doc("canciones.xml")/
  MiBibliotecaMP3/archivo[canción="
Master of Puppets"]
3
4 return
5 if($i is $j) then
6   "Son el mismo nodo"
7 else
8   "No son el mismo nodo"
```

Son el mismo nodo

- **Comparación de órdenes de los nodos:**
  - **<<** (precede), compara la posición de dos nodos. Devuelve **"true"** si el nodo ligado al primer operando ocurre primero en el orden del documento que el nodo ligado al segundo.
  - **>>** (sigue), compara la posición de dos nodos. Devuelve **"true"** si el nodo ligado al primer operando ocurre después en el orden del documento que el nodo ligado al segundo

```

1 let $a:= doc("canciones.xml")/MiBibliotecaMP3/archivo[puntuacion=10]
2 let $b:= doc("canciones.xml")/MiBibliotecaMP3/archivo[canción="Peace Sells"]
3
4 return $b >> $a (: El nodo $b sigue al nodo $a :)
5

```

OK

1 Result, 5 b

false

canciones.xml					
MiBibliotecaMP3					
archivo		archivo		archivo	
canción	disco	canción	disco	canción	disco
Hangar 18	Rust in Peace	Peace Sells	Peace Sells... But Who's Buying	Master of Puppets	Master of Puppets
artista		artista		artista	
Megadeth		Megadeth		Metallica	
puntuacion		puntuacion		puntuacion	
9		9		10	

- **Operadores de secuencias de nodos:** Devuelven secuencias de nodos en el orden del documento y eliminan duplicados de las secuencias resultado.
  - **union**, devuelve una secuencia que contiene todos los nodos que aparecen en alguno de los dos operandos que recibe. También se puede usar el operador `|` en su lugar.
  - **intersect**, devuelve una secuencia que contiene todos los nodos que aparecen en los dos operandos que recibe.
  - **except**, devuelve una secuencia que contiene todos los nodos que aparecen en el primer operando que recibe y que no aparecen en el segundo.

```

1 let $a:= doc("canciones.xml")/MiBibliotecaMP3/archivo[puntuacion>=8]
2 let $b:= doc("canciones.xml")/MiBibliotecaMP3/archivo[artista="Megadeth"]
3
4 return (
5 <interseccion>{$a intersect $b}</interseccion>, (: Los archivos que tengan puntuación >=8 y además simultáneamente sean del artista Megadeth :)
6 <union>{$a union $b}</union>, (: Los archivos que tengan puntuación >=8 y los archivos que tengan por artista Megadeth :)
7 <union2> {$a | $b}</union2>, (:Otra forma de realizar la unión con el operador | :)
8 <except>{$a except $b}</except> (: Los archivos que tengan puntuación >=8 salvo los que sean de Megadeth :)
9 )
10

```

- **Aritméticos:** `+`, `-`, `*`, `div` y `mod`, devuelven respectivamente la suma, diferencia, producto, cociente y resto de operar dos números dados

## 2. Almacenamiento de datos XML

Los documentos **XML** pueden ser agrupados en dos categorías generales:

- Sistemas **centrados en los datos o *data centric***. Cuando los documentos **XML** tienen una **estructura bien definida** y contienen datos que pueden ser actualizados y usados de diversos modos. Es apropiada para ítems como contenidos de periódicos, artículos, publicidad, facturas, órdenes de compra, etc.
- Sistemas **centrados en los documentos**. Cuando los documentos tienden a ser más **impredecibles en tamaño y contenido**. Presentan más tipos de datos, de tamaño variable, con reglas flexibles para campos opcionales y para el propio contenido.



## 2.1. Opciones de almacenamiento de datos XML

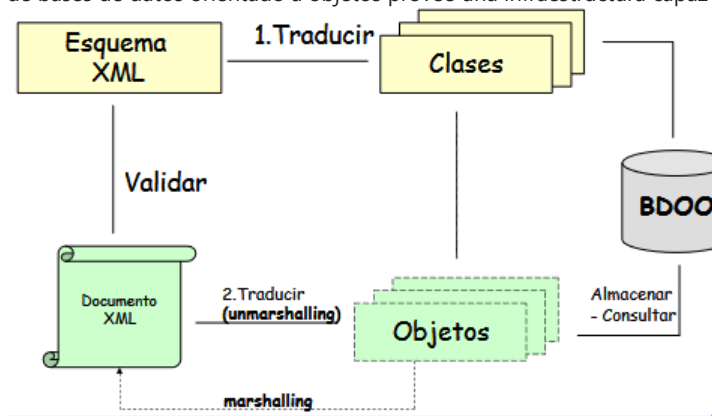
A la hora de almacenar documentos XML se pueden contemplar varias opciones:

1. Almacenamiento **directo del fichero**: No se pueden realizar consultas sobre el contenido por lo que resulta una opción muy limitada.
2. Usar una **base de datos (SGBD relacional)**: permite almacenar información en relaciones (habitualmente llamadas tablas por su representación en forma tabular). Cada relación (o tabla) tiene una serie de campos (o columnas) y registros (o filas). No se considera la opción más apropiada para almacenar únicamente datos en formato XML porque:
  - Los documentos XML pueden contener **muchos niveles de anidamiento** mientras que los datos relacionales son "planos"
  - Los documentos XML tienen carácter más **heterogéneo** que una base de datos, cuya estructura está bien definida y es regular
  - Los datos XML pueden representar la **carencia de información** mediante la **ausencia de un elemento**
3. Usar una **base de datos orientada a objetos (SGBDOO)**: Está basada en la Programación Orientada a Objetos (POO).

En POO, se usan dos conceptos fundamentales: clase y objeto

- Una clase es una "plantilla" o una estructura común con información estructurada que une datos (atributos) y tipo de comportamiento (métodos).
- Un objeto es un ejemplar concreto de una clase (esa plantilla o estructura común) con unos datos concretos. Podéis consultar una imagen con esta distinción entre clase y objeto [aquí](#).

En un SGBDOO, un esquema XML se transforma en una *clase* y un documento XML en un objeto (un ejemplar con unos datos concretos). El sistema gestor de bases de datos orientado a objetos provee una infraestructura capaz de almacenar y recuperar objetos.



[Fuente](#)

4. Usar una **base de datos XML**: un sistema que permite almacenar datos en formato XML. Estos datos pueden ser consultados, exportados y serializados (transformados en una serie de bytes para su almacenación o transmisión). Se pueden distinguir dos tipos de bases de datos XML:
  - **XML habilitado**: Pueden transformar XML en estructuras tradicionales de bases de datos aceptando XML como entrada, exportando datos en XML o soportando tipos XML nativos en la propia base de datos. Esto implica que la base de datos procesa el XML internamente. Hemos visto un ejemplo con el gestor de bases de datos relacionar de Microsoft SQL Server.
  - **XML Nativo**: Usan documentos XML como unidad de almacenamiento. Hemos visto un ejemplo con BaseX. Si los documentos son centrado en el contenido, con una estructura mayormente irregular, es una solución de almacenamiento aceptable.

## 2.2. Bases de datos XML Nativas

Las **bases de datos XML Nativas** se caracterizan principalmente por:

- **Almacenamiento** de documentos en **colecciones**. Las colecciones juegan en las bases de datos nativas un papel similar al de las tablas en las **BD** relacionales.
- Soportar **validación** de los documentos **XML** (mediante DTD o schema XSD)
- Contar con uno o más lenguajes de **consulta**. Por ejemplo, **XQuery**.
- Permitir **actualizaciones y borrados**. Por ejemplo, para insertar un elemento <year> después del subelemento publisher bajo el primer elemento <book>:

```
insert node <year>2005</year>  
after fn:doc("bib.xml")/books/book[1]/publisher
```

- **Indexación XML**. Se ha de permitir la creación de índices (mecanismos que mejoran la velocidad de las consultas).
- **Creación de identificadores únicos**. A cada documento **XML** se le asocia un identificador único.

Según el **tipo de almacenamiento** utilizado pueden dividirse en dos grupos:

- **Almacenamiento Basado en Texto**. Almacena el documento **XML** entero en forma de texto y proporciona alguna funcionalidad de base de datos para acceder a él.
- **Almacenamiento Basado en el Modelo**. Almacena un modelo binario del documento (por ejemplo, DOM) en un almacén existente o bien específico. BaseX utiliza un formato propietario para almacenar los documentos XML.