Almacenamiento de la información

# Utilización de XML para el almacenamieto de la información

## Ámbitos de aplicación

# Sistemas de almacenamiento de la información

Podemos establecer las siguientes semejanzas entre una base de datos y un fichero XML con su esquema asociado:

* La tecnología XML usa uno o más documentos para almacenar la información.
* Define esquemas sobre la información.
* Tiene lenguajes de consulta específicos para recuperar la información requerida.
* Dispone de APIs (SAX, DOM).

# XML Y BD RALACIONALES

Las Bases de Datos Relacionales suponen una posibilidad para el almacenamiento de datos XML. Sin embargo, no están bien preparadas para almacenar estructuras de tipo jerárquico como son los documentos XML, algunas de las causas son:

* Las bases de datos relacionales tienen una estructura regular frente al carácter heterogéneo de los documentos XML.
* Los documentos XML suelen contener muchos niveles de anidamiento mientras que los datos relacionales son planos.
* Los documentos XML tienen un orden intrínseco mientras que los datos relacionales son no ordenados.
* Los datos relacionales son generalmente densos (cada columna tiene un valor), mientras que los datos XML son dispersos, es decir, pueden representar la carencia de información mediante la ausencia del elemento.

Algunas de las razones para usar los tipos de Bases de Datos Relacionales y los productos de bases de datos existentes para almacenar XML, aún cuando no sea de forma nativa son:

* Las bases de datos relacionales y orientadas a objetos son bien conocidas, mientras que las bases de datos XML nativas son nuevas.
* Como resultado de la familiaridad con las bases de datos relacionales y orientadas a objetos, los usuarios se inclinan a ellas especialmente por el rendimiento.

## 3.1. De DB relacional a XML

# XML Y BD ORIENTADAS A OBJETOS

Las bases de datos orientadas a objetos (DBOO) soportan un modelo de objetos puro, en el sentido de que no están basados en extensiones de otros modelos más clásicos como el relacional:

* Están influenciados por los lenguajes de programación orientados a objetos.
* Pueden verse como un intento de añadir la funcionalidad de un SGBD a un lenguaje de programación.
* Son una alternativa para el almacenamiento y gestión de documentos XML.

Componentes del estándar de Orientación a Objetos:

* Modelo de Objetos. Está concebido para proporcionar un modelo de objetos estándar para las bases de datos orientadas a objetos. Es el modelo en el que se basan el los demás componentes.
* Lenguajes de Especificación de Objetos (ODL). Para definir los objetos.
* Lenguaje de Consulta de Objetos (OQL). Para realizar consultas contra los objetos.
* ‘Bindings’ para C++, Java y Smalltalk. Definen un Lenguaje de Manipulación de Objetos (OML) que extiende el lenguaje de programación para soportar objetos persistentes. Además incluyen soporte para OQL, navegación y transacciones.

Una vez transformado el documento XML en objetos, éstos son gestionados directamente por el SGBDOO. Dicha información se consulta acudiendo al lenguaje de consulta OQL. Los mecanismos de indexación, optimización, procesamiento de consultas, etc. son los del propio SGBDOO, y por lo general, no son específicos para el modelo XML.

# BD XML NATIVAS

Las Bases de Datos XML Nativas son bases de datos (y como tales soportan transacciones, acceso multi-usuario, lenguajes de consulta, etc) diseñadas especialmente para almacenar documentos XML.

Las BD XML Nativas se caracterizan principalmente por:

* Almacenamiento de documentos en colecciones. Las colecciones juegan en las bases de datos nativas el papel de las tablas en las BD relacionales.
* Validación de los documentos.
* Consultas. La mayoría de las BD XML Nativas soportan uno o más lenguajes de consulta. Uno de los más populares es XQuery.
* Indexación XML. Se ha de permitir la creación de índices que aceleren las consultas realizadas.
* Creación de identificadores únicos. A cada documento XML se le asocia un identificador único.
* Actualizaciones y Borrados.

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. Hay dos posibilidades:
  + Posibilidad 1: Almacenar el documento como un BLOB en una base de datos relacional, mediante un fichero, y proporcionar algunos índices sobre el documento que aceleren el acceso a la información.
  + Posibilidad 2: Almacenar el documento en un almacén adecuado con índices, soporte para transacciones, etc.
* Almacenamiento Basado en el Modelo. Almacena un modelo binario del documento (por ejemplo, DOM) en un almacén existente o bien específico.
  + Posibilidad 1: Traducir el DOM a tablas relacionales como Elementos, Atributos, Entidades, etc.
  + Posibilidad 2: Traducir el DOM a objetos en una BDOO.
  + Posibilidad 3: Utilizar un almacén creado especialmente para esta finalidad.

# xQUERY

Permite la construcción de expresiones complejas combinando expresiones simples de una manera muy flexible.

De manera general podemos decir que XQuery es a XML lo mismo que SQL es a las bases de datos relacionales. Al igual que éste último, XQuery es un lenguaje funcional.

## 6.1. Aplicaciones

Una vez que hemos visto la definición del lenguaje y sus principales requerimientos, queda pensar, ¿para qué se utiliza?

Sus principales aplicaciones se resumen en tres:

* Recuperar información a partir de conjuntos de datos XML.
* Transformar unas estructuras de datos XML en otras estructuras que organizan la información de forma diferente.
* Ofrecer una alternativa a XSLT para realizar transformaciones de datos en XML a otro tipo de representaciones, como HTML o PDF.

¿Y cuáles son los motores XQuery de código abierto más relevantes y sus características principales?

* BaseX: proyecto open-source, con interfaz gráfica y disponible para Linux, Windows y Mac.
* Qexo: escrito en Java y con licencia GPL que se distribuye integrado dentro del paquete Kawa.
* Saxon: escrito en Java y distribuido en múltiples paquetes, algunos open-source y otros bajo licencia comercial.

## 6.2. Modelo de datos

Aunque XQuery y SQL puedan considerarse similares, el modelo de datos sobre el que se sustenta XQuery es muy distinto del modelo de datos relacional sobre el que sustenta SQL, ya que XML incluye conceptos como jerarquía y orden de los datos que no están presentes en el modelo relacional.

Por ejemplo, a diferencia de SQL, en XQuery el orden en que se encuentren los datos es importante, ya que no es lo mismo buscar una etiqueta <B> dentro de una etiqueta <A> que todas las etiquetas <B> del documento (que pueden estar anidadas dentro de una etiqueta <A> o no).

La entrada y la salida de una consulta XQuery se define en términos de un modelo de datos. Dicho modelo de datos de la consulta proporciona una representación abstracta de uno o más documentos XML (o fragmentos de documentos).

Las principales características de este modelo de datos son:

* Se basa en la definición de secuencia, como una colección ordenada de cero o más ítems. Éstas pueden ser heterogéneas, es decir pueden contener varios tipos de nodos y valores atómicos. Sin embargo, una secuencia nunca puede ser un ítem de otra secuencia.
* Orden del documento: corresponde al orden en que los nodos aparecerían si la jerarquía de nodos fuese representada en formato XML, (si el primer carácter de un nodo ocurre antes que el primer carácter de otro nodo, lo precederá también en el orden del documento).
* Contempla un valor especial llamado “error value” que es el resultado de evaluar una expresión que contiene un error.

## 6.3. 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.

El valor de una expresión es una secuencia heterogénea de nodos y valores atómicos.

La mayoría de las expresiones están compuestas por la combinación de expresiones más simples unidas mediante operadores y palabras reservadas.

Ya hemos visto que XPath es un lenguaje declarativo para la localización de nodos y fragmentos de información en árboles 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.

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

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

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

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.

En una sentencia FLWOR al menos ha de existir una cláusula FOR o una LET, el resto, si existen, han de respetar escrupulosamente el orden dado por el nombre, FLWOR.

Con estas sentencias se consigue buena parte de la funcionalidad que diferencia a XQuery de XPath. Entre otras cosas permite construir el documento que será la salida de la sentencia.

Una consulta XQuery está formada por dos partes:

* Prólogo: Lugar donde se declaran los espacios de nombres, de funciones, variables, etc.
* Expresión: Consulta propiamente dicha.

## 6.4. Cláusulas

Hemos visto el modo de crear sentencias FLOWR, vamos ahora a estudiar aisladamente cada una de las cláusulas que pueden formar estas sentencias.

* FOR: asocia una o más variables con cada nodo que encuentre en la colección de datos. Si en la consulta aparece más de una cláusula FOR (o más de una variable en una cláusula FOR), el resultado es el producto cartesiano de dichas variables.
* LET: vincula las variables al resultado de una expresión. Si esta cláusula aparece en una sentencia en la que ya hay al menos una cláusula FOR, los valores de la variable vinculada por la cláusula LET se añaden a cada una de las tuplas generadas por la cláusula FOR.
* WHERE: filtra tuplas producidas por las cláusulas FOR y LET, quedando solo aquellas que cumplen con la condición.
* ORDER BY: ordena las tuplas generadas por FOR y LET después de que han sido filtradas por la cláusula WHERE. Por defecto el orden es ascendiente, pero se puede usar el modificador descending para cambiar el sentido del orden.
* RETURN: construye el resultado de la expresión FLWOR para una tupla dada.

## 6.5 Ejemplos

Libros.xml

<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>

*<!-- Fichero: 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>

            <paginas>16</paginas>

        <versionElectronica/>

        </libro>

        <libro publicacion="2003" edicion="2">

            <titulo>XML Imprescindible</titulo>

            <autor>

                <apellido>Harold</apellido>

                <nombre>Elliot Rusty</nombre>

            </autor>

            <autor>

                <apellido>Means</apellido>

                <nombre>W. Scott</nombre>

            </autor>

            <editorial>O'Reilly</editorial>

            <paginas>832</paginas>

        </libro>

        <libro publicacion="2002">

            <titulo>XML Schema</titulo>

            <autor>

                <apellido>van der Vlist</apellido>

                <nombre>Eric</nombre>

            </autor>

            <editorial>O'Reilly</editorial>

            <paginas>400</paginas>

        </libro>

        <libro publicacion="2002">

            <titulo>XPath Essentials</titulo>

            <autor>

                <apellido>Watt</apellido>

                <nombre>Adrew</nombre>

            </autor>

            <editorial>Wiley</editorial>

            <paginas>516</paginas>

            <versionElectronica/>

        </libro>

        <libro publicacion="2005">

            <titulo> Beginning XSLT 2.0: Form Novice to Professional</titulo>

            <autor>

                <apellido>Tennison</apellido>

                <nombre>Jeni</nombre>

            </autor>

            <editorial>Apress</editorial>

            <paginas>797</paginas>

        </libro>

        <libro publicacion="2007">

            <titulo> XQuery</titulo>

            <autor>

                <apellido>Walmsley</apellido>

                <nombre>Priscilla</nombre>

            </autor>

            <editorial>O'Reilly</editorial>

            <paginas>491</paginas>

        </libro>

    </libros>

</biblioteca>

1.- Título y editorial de todos los libros Para devolver varios campos, los envolvemos en un elemento.

for $x in doc("libros.xml")/biblioteca/libros/libro

return <libro>{$x/titulo, $x/editorial}</libro>

<libro>

  <titulo>Learning XML</titulo>

  <editorial>O'Reilly</editorial>

</libro>

<libro>

  <titulo>XML Imprescindible</titulo>

  <editorial>O'Reilly</editorial>

</libro>

<libro>

  <titulo>XML Schema</titulo>

  <editorial>O'Reilly</editorial>

</libro>

<libro>

  <titulo>XPath Essentials</titulo>

  <editorial>Wiley</editorial>

</libro>

<libro>

  <titulo>Beginning XSLT 2.0: Form Novice to Professional</titulo>

  <editorial>Apress</editorial>

</libro>

<libro>

  <titulo>XQuery</titulo>

  <editorial>O'Reilly</editorial>

</libro>

2.- El título (sin etiquetas) de todos los libros de menos de 100 páginas.Para hacer comparaciones con números, lo mejor es convertir los datos con la función number para evitar problemas de tipo de dato o que los compare como cadenas.

for $x in doc("libros.xml")/biblioteca/libros/libro

where number($x/paginas) < 100

return data($x/titulo)

Learning XML

3.- El número de libros de menos de 100 páginas.Utilizamos la función count().

for $x in doc("libros.xml")/biblioteca/libros

let $y := $x/libro[number(paginas) < 100]

return count($y)

1

4.- Una lista HTML con el título de los libros de la editorial “O'Reilly” ordenados por título.Podemos mezclar etiquetas HTML y XQuery y obtener HTML como resultado de una consulta.

<ul>

{

for $x in doc("libros.xml")/biblioteca/libros/libro

where $x/editorial = "O'Reilly"

order by $x/titulo

return <li>{data($x/titulo)}</li>

}

</ul>

<ul>

  <li>Learning XML</li>

  <li>XML Imprescindible</li>

  <li>XML Schema</li>

  <li>XQuery</li>

</ul>

5.- Título y editorial de los libros de 2002.

for $x in doc("libros.xml")/biblioteca/libros/libro

where $x[@publicacion=2002]

return <libro>{$x/titulo, $x/editorial}</libro>

<libro>

  <titulo>XML Schema</titulo>

  <editorial>O'Reilly</editorial>

</libro>

<libro>

  <titulo>XPath Essentials</titulo>

  <editorial>Wiley</editorial>

</libro>

6.- Título y editorial de los libros con más de un autor.

for $x in doc("libros.xml")/biblioteca/libros/libro

where count($x/autor)>1

return <libro>{$x/titulo, $x/editorial}</libro>

<libro>

  <titulo>XML Imprescindible</titulo>

  <editorial>O'Reilly</editorial>

</libro>

7.- Título y editorial de los libros que tienen versión electrónica.

for $x in doc("libros.xml")/biblioteca/libros/libro

where $x/versionElectronica

return <libro>{$x/titulo, $x/editorial}</libro>

<libro>

  <titulo>Learning XML</titulo>

  <editorial>O'Reilly</editorial>

</libro>

<libro>

  <titulo>XPath Essentials</titulo>

  <editorial>Wiley</editorial>

</libro>

8.- Título de los libros que no tienen versión electrónica.

for $x in doc("libros.xml")/biblioteca/libros/libro

where not($x/versionElectronica)

return$x/titulo

<titulo>XML Imprescindible</titulo>

<titulo>XML Schema</titulo>

<titulo>Beginning XSLT 2.0: Form Novice to Professional</titulo>

<titulo>XQuery</titulo>

## 6.6. Funciones

Ahora que conocemos las cláusulas que sustentan el lenguaje XQuery, vamos a ocuparnos de las funciones que soporta.

Estas son funciones matemáticas, de cadenas, para el tratamiento de expresiones regulares, comparaciones de fechas y horas, manipulación de nodos XML, manipulación de secuencias, comprobación y conversión de tipos y lógica booleana. Además permite definir funciones propias y funciones dependientes del entorno de ejecución del motor XQuery. Las funciones más importantes se muestran a continuación:

Funciones numéricas

* floor(), que devuelve el valor numérico inferior más próximo al dado.
* ceiling(), que devuelve el valor numérico superior más próximo al dado.
* round(), que redondea el valor dado al más próximo.
* count(), determina el número de ítems en una colección.
* min() o max(), devuelven respectivamente el mínimo y el máximo de los valores de los nodos dados.
* avg(), calcula el valor medio de los valores dados.
* sum(), calcula la suma total de una cantidad de ítems dados.

Funciones de cadenas de texto

* concat(), devuelve una cadena construida por la unión de dos cadenas dadas.
* string-length(), devuelve la cantidad de caracteres que forman una cadena.
* Starts-with(), ends-with(), determinan si una cadena dada comienza o termina, respectivamente, con otra cadena dada.
* upper-case(), lower-case(), devuelve la cadena dada en mayúsculas o minúsculas respectivamente.

Funciones de uso general

* empty(), devuelve “true” cuando la secuencia dada no contiene ningún elemento.
* exists(), devuelve “true” cuando una secuencia contiene, al menos, un elemento.
* distinct-values(), extrae los valores de una secuencia de nodos y crea una nueva secuencia con valores únicos, eliminando los nodos duplicados.
* data(), devuelve el valor de los elementos que recibe como argumentos, es decir, sin etiquetas.

Cuantificadores existenciales:

* some, every, permiten definir consultas que devuelven algún, o todos los elementos, que verifiquen la condición dada.

Ejemplo de la función data

Esta consulta devuelve todos los títulos, incluyendo las etiquetas:

for $x in doc("libros.xml")/biblioteca/libros/libro/titulo

return $x

El resultado con el fichero de ejemplo sería:

<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>

Utilizando la función data:

for $x in doc("libros.xml")/biblioteca/libros/libro/titulo

return data($x

.se obtiene:

Learning XML

XML Imprescindible

XML Schema

XPath Essentials

Beginning XSLT 2.0: Form Novice to Professional

XQuery

Además de estas funciones que están definidas en el lenguaje, XQuery permite al desarrollador construir sus propias funciones:

declare nombre\_funcion($param1 as tipo\_dato1, $param2 as tipo\_dato2,… $paramN as tipo\_datoN)

as tipo\_dato\_devuelto

{

...CÓDIGO DE LA FUNCIÓN...

}

## 6.8. Ejemplo

En este ejemplo puedes ver la definición y llamada a una función escrita por el usuario que nos calcula el precio de un libro una vez que se le ha aplicado el descuento.

declare **function** minPrice($p as xs:decimal?,$d as xs:decimal?) as xs:decimal?

  {

  let $disc := ($p \* $d) div 100

  return ($p - $disc)

  }

Un ejemplo de cómo invocar a la función desde la consulta es:

<minPrice>{minPrice($libros/precio,$libros/descuento)}</minPrice>

## 6.8. Operadores

Veamos ahora algunos de los operadores más importantes agrupados según su funcionalidad:

Comparación de valores: Comparan dos valores escalares y produce un error si alguno de los operandos es una secuencia de longitud mayor de 1. Estos operadores son:

* eq, igual.
* ne, no igual.
* lt, menor que.
* le, menor o igual que.
* gt, mayor que.
* ge, mayor o igual que.

Comparación generales: Permiten comparar operandos que sean secuencias.

* =, igual.
* !=, distinto.
* >, mayor que.
* >=, mayor o igual que.
* <, menor que.
* <=, menor o igual que.

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.

Comparación de órdenes de los nodos: <<, 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.

Lógicos: and y or Se emplean para combinar condiciones lógicas dentro de un predicado.

Secuencias de nodos: Devuelven secuencias de nodos en el orden del documento y eliminan duplicados de las secuencias resultado.

* Unión, devuelve una secuencia que contiene todos los nodos que aparecen en alguno de los dos operandos que recibe.
* 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.

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

1. Mostrar todas las canciones

doc("canciones.xml")/MiBibliotecaMP3/archivo/canción

<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>

2. Mostrar todos los datos de un archivo con puntuación 10

doc("canciones.xml")/MiBibliotecaMP3/archivo[puntuacion=10]

<archivo almacenado="DISCO1">

  <canción>Master of Puppets</canción>

  <artista>Metallica</artista>

  <disco>Master of Puppets</disco>

  <puntuacion>10</puntuacion>

</archivo>

3. Mostrar la canción con una puntuación mayor que 8

doc("canciones.xml")/MiBibliotecaMP3/archivo[puntuacion>8]/canción

<canción>Hangar 18</canción>

<canción>Peace Sells</canción>

<canción>Master of Puppets</canción>

4. Mostrar la canción con una puntuación mayor que 8 con expresión FLWOR

for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo

where $i/puntuacion>8

return $i/canción

<canción>Hangar 18</canción>

<canción>Peace Sells</canción>

<canción>Master of Puppets</canción>

5.Combinar html con una expresión FLOWR

<html>

<head>

<title>Ejemplo 5</title>

</head>

<body>

<ol>

{

for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo

where $i/puntuacion>8

order by $i/puntuacion

return <li> {$i/canción}({$i/puntuacion}) </li>

}

</ol>

</body>

</html>

<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>

    </ol>

  </body>

</html>

6. Para evitar que el return nos devuelva los títulos de las etiquetas, hacemos uso de la función data

<html>

<head>

<title>Ejemplo 6</title>

</head>

<body>

<ol>

{

for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo

where $i/puntuacion>8

order by $i/puntuacion

return <li> {data($i/canción)}({data($i/puntuacion)}) </li>

}

</ol>

</body>

</html>

<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>

7.Añadiendo un atributo XML como identificador de la clase en el archivo HTML de salida

<html>

<head>

<title>Ejemplo 7</title>

</head>

<body>

<ol>

{

for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo

where $i/puntuacion>8

order by $i/puntuacion

return <li class="{data($i/@almacenado)}"> {data($i/canción)}({data($i/puntuacion)}) </li>

}

</ol>

</body>

</html>

<html>

  <head>

    <title>Ejemplo 7</title>

  </head>

  <body>

    <ol>

      <li class="DISCO1">Master of Puppets(10) </li>

      <li class="DISCO1">Hangar 18(9) </li>

      <li class="DISCO2">Peace Sells(9) </li>

    </ol>

  </body>

</html>

8. Incorporar la expresión if-then-else

<html>

<head>

<title>Ejemplo 8</title>

</head>

<body>

<table>

<caption>DISCO 1 </caption>

<tr><td>Artista</td><td>Disco</td></tr>

{

for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo

order by $i/puntuacion

return if ($i/@almacenado="DISCO1")

then <tr><td>{data($i/artista)}</td><td>{data($i/disco)}</td></tr>

else ()

}

</table>

<table>

<caption>DISCO 2 </caption>

<tr><td>Artista</td><td>Disco</td></tr>

{

for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo

order by $i/puntuacion

return if ($i/@almacenado="DISCO2")

then <tr><td>{data($i/artista)}</td><td>{data($i/disco)}</td></tr>

else ()

}

</table>

</body>

</html>

<html>

  <head>

    <title>Ejemplo 8</title>

  </head>

  <body>

    <table>

      <caption>DISCO 1 </caption>

      <tr>

        <td>Artista</td>

        <td>Disco</td>

      </tr>

      <tr>

        <td>Metallica</td>

        <td>Master of Puppets</td>

      </tr>

      <tr>

        <td>Metallica</td>

        <td>Ride The Lightning</td>

      </tr>

      <tr>

        <td>Megadeth</td>

        <td>Rust in Peace</td>

      </tr>

    </table>

    <table>

      <caption>DISCO 2 </caption>

      <tr>

        <td>Artista</td>

        <td>Disco</td>

      </tr>

      <tr>

        <td>Anthrax</td>

        <td>Among The Living</td>

      </tr>

      <tr>

        <td>Megadeth</td>

        <td>Peace Sells...But Who's Buying</td>

      </tr>

    </table>

  </body>

</html>

9. HTML completo

<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>

{

for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo

order by $i/puntuacion

return if ($i/@almacenado="DISCO1")

then <tr><td>{data($i/artista)}</td><td>{data($i/disco)}</td>

     <td>DISCO1</td></tr>

else <tr><td>{data($i/artista)}</td><td>{data($i/disco)}</td>

     <td>DISCO2</td></tr>

}

</table>

</body>

</html>

<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>

      <tr>

        <td>Megadeth</td>

        <td>Peace Sells...But Who's Buying</td>

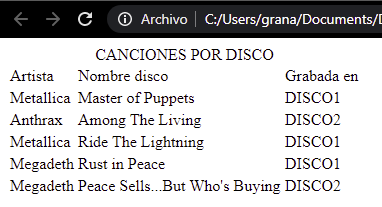
        <td>DISCO2</td>

      </tr>

    </table>

  </body>

</html>



10. Filtrar elementos mediante expresiones de ruta o con expresiones FLWOR

<html>

<head>

<title>Ejemplo 10</title>

</head>

<body>

<table>

<caption>CANCIONES DE METALLICA </caption>

<tr><td>Canción</td><td>Disco</td><td>Grabada en</td></tr>

{

for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo

where $i/artista="Metallica"

order by $i/puntuacion

return <tr><td>{data($i/artista)}</td><td>{data($i/disco)}</td>

     <td>{data($i/@almacenado)}</td></tr>

}

</table>

</body>

</html>

<html>

  <head>

    <title>Ejemplo 10</title>

  </head>

  <body>

    <table>

      <caption>CANCIONES DE METALLICA </caption>

      <tr>

        <td>Canción</td>

        <td>Disco</td>

        <td>Grabada en</td>

      </tr>

      <tr>

        <td>Metallica</td>

        <td>Master of Puppets</td>

        <td>DISCO1</td>

      </tr>

      <tr>

        <td>Metallica</td>

        <td>Ride The Lightning</td>

        <td>DISCO1</td>

      </tr>

    </table>

  </body>

</html>

11. La cláusula for

for $i in (1 to 10)

return $i

1

2

3

4

5

6

7

8

9

10

12. Contabilizar las iteraciones. Palabra clave **at**

<html>

<head>

<title>Ejemplo 12</title>

</head>

<body>

<ul>

{

for $i at $j in doc("canciones.xml")/MiBibliotecaMP3/archivo

where $i/puntuacion>8

order by $i/puntuacion

return <li>{$j}. {data($i/canción)}({data($i/puntuacion)}) </li>

}

</ul>

</body>

</html>

<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>

13. Especificar varias expresiones en la cláusula **for**

for $i in (1 to 5), $j in (1,2,3)

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

<resultado>i es 1 j es 1</resultado>

<resultado>i es 1 j es 2</resultado>

<resultado>i es 1 j es 3</resultado>

<resultado>i es 2 j es 1</resultado>

<resultado>i es 2 j es 2</resultado>

<resultado>i es 2 j es 3</resultado>

<resultado>i es 3 j es 1</resultado>

<resultado>i es 3 j es 2</resultado>

<resultado>i es 3 j es 3</resultado>

<resultado>i es 4 j es 1</resultado>

<resultado>i es 4 j es 2</resultado>

<resultado>i es 4 j es 3</resultado>

<resultado>i es 5 j es 1</resultado>

<resultado>i es 5 j es 2</resultado>

<resultado>i es 5 j es 3</resultado>

14**. LET**: realiza definición y asignación de variables. NO REALIZA ITERACIONES POR SI MISMA

let $i := (1 to 10)

let $j := (1,2,3)

let $k := 1

return <resultado>i es {$i} j es {$j} k es {$k}</resultado>

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

15. **Where**: es posible establecer condiciones compuestas

<html>

<head>

<title>Ejemplo 15</title>

</head>

<body>

<ol>

{

for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo

where $i/puntuacion=8 or $i/puntuacion=10

order by $i/puntuacion

return <li class="{data($i/@almacenado)}"> {data($i/canción)}({data($i/puntuacion)}) </li>

}

</ol>

</body>

</html>

<html>

  <head>

    <title>Ejemplo 15</title>

  </head>

  <body>

    <ol>

      <li class="DISCO1">Master of Puppets(10) </li>

      <li class="DISCO2">Among The Living(8) </li>

      <li class="DISCO1">For Whom The Bell Tolls(8) </li>

    </ol>

  </body>

</html>

16. Ejemplo con **where** y **and**

<html>

<head>

<title>Ejemplo 16</title>

</head>

<body>

<ol>

{

*(: canciones de grupos cuyo nombre empieza por "M" y*

*tienen puntuación de 9  :)*

for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo

where $i[substring(artista,1,1)="M"] and $i/puntuacion=9

order by $i/puntuacion

return <li> {data($i/canción)}({data($i/artista)}) </li>

}

</ol>

</body>

</html>

<html>

  <head>

    <title>Ejemplo 16</title>

  </head>

  <body>

    <ol>

      <li>Hangar 18(Megadeth) </li>

      <li>Peace Sells(Megadeth) </li>

    </ol>

  </body>

</html>

17. **Where** con **and** y **not**

<html>

<head>

<title>Ejemplo 17</title>

</head>

<body>

<ol>

{

*(: canciones de grupos cuyo nombre empieza por "M" y*

*NO tienen puntuación de 9  :)*

for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo

where $i[substring(artista,1,1)="M"] and not($i/puntuacion=9)

order by $i/puntuacion

return <li> {data($i/canción)}({data($i/artista)}) </li>

}

</ol>

</body>

</html>

<html>

  <head>

    <title>Ejemplo 17</title>

  </head>

  <body>

    <ol>

      <li>Master of Puppets(Metallica) </li>

      <li>For Whom The Bell Tolls(Metallica) </li>

    </ol>

  </body>

</html>

**18. Orderby:** ordenar por varios criterios y especificar qué orden: **ascending** ó **desscending**

<html>

<head>

<title>Ejemplo 18</title>

</head>

<body>

<table>

<caption>CANCIONES POR DISCO </caption>

<tr><td>Artista</td><td>Nombre</td><td>Grabada en</td></tr>

{

for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo

order by $i/artista ascending, $i/canción descending

return <tr><td>{data($i/artista)}</td><td>{data($i/canción)}</td>

     <td>{data($i/@almacenado)}</td></tr>

}

</table>

</body>

</html>

<html>

  <head>

    <title>Ejemplo 18</title>

  </head>

  <body>

    <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>

        <td>Megadeth</td>

        <td>Peace Sells</td>

        <td>DISCO2</td>

      </tr>

      <tr>

        <td>Megadeth</td>

        <td>Hangar 18</td>

        <td>DISCO1</td>

      </tr>

      <tr>

        <td>Metallica</td>

        <td>Master of Puppets</td>

        <td>DISCO1</td>

      </tr>

      <tr>

        <td>Metallica</td>

        <td>For Whom The Bell Tolls</td>

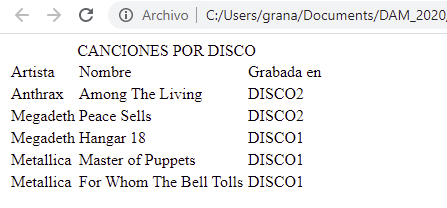
        <td>DISCO1</td>

      </tr>

    </table>

  </body>

</html>



**19.** Llamadas a funciones. Subrayadas las llamadas a función

<html>

<head>

<title>Ejemplo 19</title>

</head>

<body>

<table>

<caption>CANCIONES POR DISCO </caption>

<tr><td>Artista</td><td>Nombre</td><td>Grabada en</td></tr>

{

for $i in doc("canciones.xml")/MiBibliotecaMP3/archivo

let $numero := (substring($i/@almacenado,6,1))

order by $i/artista ascending, $i/canción descending

return <tr><td>{upper-case(data($i/artista))}</td><td>{data($i/canción)}</td>

     <td>{$numero}</td></tr>

}

</table>

</body>

</html>

<html>

  <head>

    <title>Ejemplo 19</title>

  </head>

  <body>

    <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>2</td>

      </tr>

      <tr>

        <td>MEGADETH</td>

        <td>Peace Sells</td>

        <td>2</td>

      </tr>

      <tr>

        <td>MEGADETH</td>

        <td>Hangar 18</td>

        <td>1</td>

      </tr>

      <tr>

        <td>METALLICA</td>

        <td>Master of Puppets</td>

        <td>1</td>

      </tr>

      <tr>

        <td>METALLICA</td>

        <td>For Whom The Bell Tolls</td>

        <td>1</td>

      </tr>

    </table>

  </body>

</html>

**Funciones definidas por el usuario**

En caso de que el usuario necesite realizar una función que no está en la librería de funciones de XQuery puede definir las suyas propias.

◦ Se pueden definir en el propio archivo o en una librería separada.

◦ Los tipos de datos de los parámetros son los mismos definidos para XML Schema

◦ El cuerpo de la función debe estar delimitado por paréntesis

◦ Las funciones en Xquery no utilizan la palabra reservada return , siempre devuelve el último (y único)

valor.

◦ La sintaxis es:

declare

**function** prefijo:nombre\_función parámetros\_como\_tipos\_de\_datos

as

Tipo\_de\_datos\_de\_retorno

{

...código de la función...

};

20. Función que pasa de minutos a segundos

*(: declaración de la función :)*

declare **function** local:MinutosASegundos($tiempo as xs:decimal?) as xs:decimal?

{

($tiempo\*60)

};

*(: llamada a la función :)*

local:MinutosASegundos(14)

840

# Canciones.xml

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

<MiBibliotecaMP3>

  <archivo *almacenado*="DISCO1">

    <canción>Hangar 18</canción>

    <artista>Megadeth</artista>

    <disco>Rust in Peace</disco>

    <puntuacion>9</puntuacion>

    </archivo>

  <archivo *almacenado*="DISCO2">

    <canción>Peace Sells</canción>

    <artista>Megadeth</artista>

    <disco>Peace Sells...But Who's Buying</disco>

    <puntuacion>9</puntuacion>

    </archivo>

  <archivo *almacenado*="DISCO1">

    <canción>Master of Puppets</canción>

    <artista>Metallica</artista>

    <disco>Master of Puppets</disco>

    <puntuacion>10</puntuacion>

    </archivo>

  <archivo *almacenado*="DISCO2">

    <canción>Among The Living</canción>

    <artista>Anthrax</artista>

    <disco>Among The Living</disco>

    <puntuacion>8</puntuacion>

    </archivo>

   <archivo *almacenado*="DISCO1">

    <canción>For Whom The Bell Tolls</canción>

    <artista>Metallica</artista>

    <disco>Ride The Lightning</disco>

    <puntuacion>8</puntuacion>

  </archivo>

</MiBibliotecaMP3>