Comunidad de Madrid

Curso de Formación
"Lenguajes de Marcas y Sistemas de Gestión de la Información"
Del 12 de Abril al 7 de Junio de 2010

Unidad V:
XML: Lenguajes de Consulta
Xquery / XPath

Ponente:
José Luis Delgado Leal
Profesor del Departamento de Informática (Ciclos)
I.E.S. "Juan de la Cierva" (Madrid)

Contenidos de la Presentación

- XQuery
- XPath

¿Qué es XQuery?

Comunidad de Madrid

- XQuery es el lenguaje de consulta para datos XML
- XQuery es a XML lo que SQL es a las Bases de Datos
- XQuery se basa en expresiones XPath
- XQuery se ha definido por parte de W3C
- XQuery está admitido por la mayoría de los motores de bases de datos *(*IBM, Oracle, Microsoft, etc.*)*
- XQuery pasará a ser un estándar de W3C standard y los desarrolladores podrán estar seguros de este modo que su código podrá trabajar con diferentes plataformas

Consultas XML usando XQuery

Comunidad de Madrid

- XQuery is un lenguaje que permite encontrar y extraer elementos y atributos de un documento XML

- Podríamos hacer con XQuery una consulta de esta índolo, por poner un ejemplo:"Obtener todos los CD's con un precio inferior a 10€ de la colección de CD's almacenada en un documento XML llamado CatalogoCD.xml"

- XQuery está basado en XPath

- XQuery 1.0 y XPath 2.0 comparten el mismo modelo de datos y soportan las mismas funciones y operadores

- Si se conoce XPath, el paso a XQuery es inmediato

Ejemplos de Uso de XQuery

Comunidad de Madrid

- Las aplicaciones de XQuery son múltiples:
  - Extracción de información para ser usada en un Web Service
  - Generación de Informes de Resumen
  - Transformación de datos XML a XHTML
  - Búsqueda en documentos web de información relevante

XQuery como estándar

Comunidad de Madrid

- XQuery podría considerarse ya como un estándar web
- XQuery es compatible con múltiples estándares W3C, como pueda ser XML, Namespaces, XSLT, XPath y XML Schema
- Sin embargo, XQuery 1.0 no es todavía un Recomendación W3C Recommendation *(XQuery es en la actualidad un Working Draft)*
- Afortunadamente, y con toda seguridad, XQuery será una recomendación W3C en un futuro bastante cercano

Un Ejemplo XML

Comunidad de Madrid

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
      <book category="COOKING">
            <title lang="en">Everyday Italian</title>
            <author>Giada De Laurentiis</author>
            <year>2005</year>
            <price>30.00</price>
      </book>
      <book category="CHILDREN">
            <title lang="en">Harry Potter</title>
            <author>J K. Rowling</author>
            <year>2005</year>
            <price>29.99</price>
      </book>
      <book category="WEB">
            <title lang="en">XQuery Kick Start</title>
            <author>James McGovern</author>
            <author>Per Bothner</author>
            <author>Kurt Cagle</author>
            <author>James Linn</author>
            <author>Vaidyanathan Nagarajan</author>
            <year>2003</year>
            <price>49.99</price>
      </book>
      <book category="WEB">
            <title lang="en">Learning XML</title>
            <author>Erik T. Ray</author>
            <year>2003</year>
            <price>39.95</price>
      </book>
</bookstore>
```

Trabajar con XQuery: Expresiones XPath (I)

Comunidad de Madrid

- XQuery usa funciones para extraer los datos de los documentos XML
- La función doc() se emplea para poder abrir el fichero "books.xml":

```
doc("books.xml")
```

- XQuery usa expresiones "path" (rutas) para navegar a traves de los diferentes elementos de un documento XML
- La siguiente expresión se emplea para seleccionar todos los títulos (elementos título) del fichero "books.xml":

```
doc("books.xml")/bookstore/book/title
```

Trabajar con XQuery: Expresiones XPath (II)

Comunidad de Madrid

- De este modo, /bookstore selecciona el elemento bookstore, /book selecciona todos los elementos book bajo el elemento bookstore, y /title selecciona todos los elementos title bajo cada uno de los elementos book

- La consulta XQuery anterior obtiene el siguiente resultado:

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

Predicados en XQuery

Comunidad de Madrid

- XQuery usa predicados para limitar el número de datos que se extraen de un documento XML

- El siguiente predicado permite seleccionar todos los elementos book bajo el elemento bookstore que tienen un precio con un valor price inferior a 30:

```
doc("books.xml")/bookstore/book[price<30]
```

- La consulta de arriba obtendrá lo siguiente:

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

Expresiones FLWOR (I)

Comunidad de Madrid

- Sea la siguiente expresión

```
doc("books.xml")/bookstore/book[price>30]/title
```

- La expresión mostrada seleccionará todos los elementos title bajo los elementos book que a su vez se encuentran bajo el elemento bookstore que tienen un valor en el elemento price mayor a 30

- La siguiente expresión FLWOR realiza la misma función que la expresión path mostrada arriba:

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

Expresiones FLWOR (II)

Comunidad de Madrid

- El resultado será:

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

- Con FLWOR se pueden ordenar los resultados:

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

- FLWOR es el acrónimo de "For,Let,Where,Order by,Return"

Expresiones FLWOR *(III)*

Comunidad de Madrid

- La cláusula for selecciona todos los elementos book bajo el elemento bookstore y los coloca en una variable llamada $x

- La cláusula where selecciona sólo los elementos book con un valor en el elemento price mayor de 30

- La cláusula order by define el tipo de ordenación. Así, indica que los resultados sean ordenados por el elemento title

- La cláusula return especifica qué es lo que ha de devolverse. En este caso, devuelve los elementos title

- El resultado de la expresión XQuery anterior es:

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

Nodos en XQuery (I)

Comunidad de Madrid

- En terminología XQuery, se pueden distinguir siete tipos de nodos:
  - Elementos (element)
  - Atributos (attribute)
  - Texto (text)
  - Nombre de espacio (namespace)
  - Instrucción de Procesamiento (processing-instruction)
  - Comentario (comment)
  - Documento (root node)

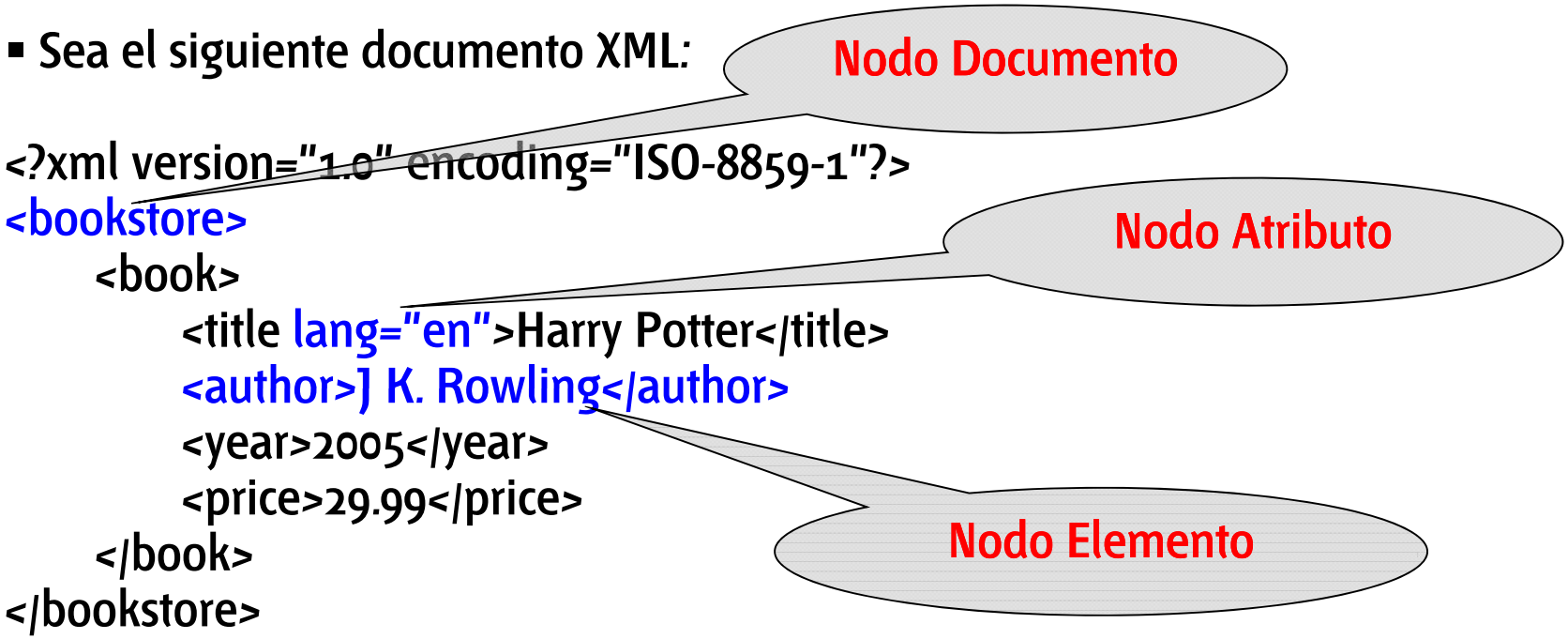
- Los documentos XML se tratan como si fueran árboles compuestos por diferentes nodos

- La raíz de ese árbol se le llama nodo documento (nodo document, o también se conoce nodo root)

Nodos en XQuery (II)

Comunidad de Madrid

- Sea el siguiente documento XML:




```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
    <book>
        <title lang="en">Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
</bookstore>
```

- Los valores atómicos son aquellos nodos que no tienen ni nodos hijos colgando de él, ni cuelgan de nodos padres. Ejemplo de ello son los items: J K. Rowling, "en"

Relaciones entre Nodos en XQuery (I)

Comunidad de Madrid

- Padres: cada elemento y atributo tiene un padre. En el ejemplo anterior, el elemento book es el padre los elementos title, author, year y price:

```
<book>
        <title>Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
</book>
```

- Hijos: los nodos elemento pueden tener cero, uno o más hijos. En el ejemplo anterior, title, author, year y price son todos hijos del elemento book

Relaciones entre Nodos en XQuery (II)

Comunidad de Madrid

- Siblings (hermanos): los nodos siblings o hermanos son aquellos que tienen el mismo padre. Así, siguiendo con el ejemplo, los nodos title, author, year, y price son todos nodos hermanos

- Los nodos además se dicen que tienen ascendientes y descendientes.

- Los nodos ascendientes son los nodos padres de un nodo, los padres de un padre, etc. Para title, los ascendientes son book y bookstore

- Los nodos descendientes son los hijos, los hijos de los hijos, etc. Para bookstore, los descendientes son book, title, author, year y price

Reglas Básicas en XQuery *(I)*

Comunidad de Madrid

- XQuery es sensible a mayúsculas y minúsculas

- Cualquier elemento, atributo o variable de XQuery tiene que ser un identificador válido en XML

- Un valor string o cadena de XQuery puede estar contenido tanto en comillas simples como dobles

- Una variable XQuery se define con un símbolo $ seguido por un nombre. Por ejemplo, $bookstore

- Un comentario XQuery se inserta dentro de un limitador de inicio *(:* y un delimitador de fin *:)*. Por ejemplo, *(:* Hola *:)*

Reglas Básicas en XQuery (II)

Comunidad de Madrid

- Las expresiones condicionales "if-then-else" se permiten en XQuery. Sea el siguiente ejemplo:

```
for $x in doc("books.xml")/bookstore/book
return if ($x/@category="CHILDREN")
      then <child>{data($x/title)}</child>
      else <adult>{data($x/title)}</adult>
```

- Respecto a la sintáxis de "if-then-else", decir que los paréntesis de la condición van alrededor de la expresión que se usa como evaluación, y su uso es obligatorio

- El uso de else es igualmente obligatorio, aunque puede estar vacío y ser sólo un else ()

Reglas Básicas en XQuery (III)

Comunidad de Madrid

- El resultado de la expresión anterior será:

```
<adult>Everyday Italian</adult>
<child>Harry Potter</child>
<adult>Learning XML</adult>
<adult>XQuery Kick Start</adult>
```

- Existen dos formas de comparar valores en XQuery:
  - Comparaciones Generales: =, !=, <, <=, >, >=
  - Comparación de valores: eq, ne, lt, le, gt, ge

- No es lo mismo comparar con un método que con otro, puesto que cada una tiene una indicación específica

Reglas Básicas en XQuery (IV)

Comunidad de Madrid

- La diferencia entre los dos métodos de comparación se muestra a continuación

```
$bookstore//book/@q > 10
$bookstore//book/@q gt 10
```

- La primera expresión devuelve true si cualquier atributo q tiene un valor mayor que 10

- La segunda expresión devuelve true si hay un solo atributo q devuelto por la expresión y su valor es mayor que 10. Si hay más de un atributo q devuelto, se produce un error

Añadir Elementos y Atributos al Resultado (I)

Comunidad de Madrid

- Se pueden añadir elementos y atributos del documento de entrada (en nuestro caso, books.xml) en el resultado:

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

- Esta expresión incluirá tanto los elementos title como los atributos lang en el resultado, obteniéndose algo de esta forma:

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

Añadir Elementos y Atributos al Resultado (II)

Comunidad de Madrid

- La expresión anterior devuelve los elementos title de la misma forma que están descritos en el documento de entrada

- Ahora queremos añadir nuestros elementos y atributos al resultado. Colocaremos esto en un HTML junto con más texto:

```
<html>
    <body>
        <h1>Bookstore</h1>
        <ul>
            {
                for $x in doc("books.xml")/bookstore/book
                order by $x/title
                return<li>{data($x/title)}.Categoría:{data($x/@category)}</li>
            }
        </ul>
    </body>
</html>
```

Añadir Elementos y Atributos al Resultado (III)

Comunidad de Madrid

- La expresión anterior generará un resultado como éste:

```
<html>
    <body>
        <h1>Bookstore</h1>
        <ul>
            <li>Everyday Italian. Category: COOKING</li>
            <li>Harry Potter. Category: CHILDREN</li>
            <li>Learning XML. Category: WEB</li>
            <li>XQuery Kick Start. Category: WEB</li>
        </ul>
    </body>
</html>
```

Añadir Elementos y Atributos al Resultado (IV)

- Ahora, se quiere usar la categoría atributo como una clase atributo en una lista HTML:

```
<html>
    <body>
        <h1>Bookstore</h1>
        <ul>
            {
                for $x in doc("books.xml")/bookstore/book
                order by $x/title
                return <li class="{data($x/@category)}">{data($x/title)}</li>
            }
        </ul>
    </body>
</html>
```

Añadir Elementos y Atributos al Resultado (V)

Comunidad de Madrid

- La expresión anterior generará un resultado como éste:

```
<html>
    <body>
        <h1>Bookstore</h1>
        <ul>
            <li class="COOKING">Everyday Italian</li>
            <li class="CHILDREN">Harry Potter</li>
            <li class="WEB">Learning XML</li>
            <li class="WEB">XQuery Kick Start</li>
        </ul>
    </body>
</html>
```

Selección y Filtrado de Elementos

Comunidad de Madrid

- Las selecciones o filtrados de elementos se pueden hacer, como ya se ha visto, con expresiones Path o con expresiones FLWOR. Por ejemplo:

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

- Las diferentes cláusulas cumplen:
  - for (opcional): hace que una una variable tome cada uno de los items devueltos por la expresión in
  - let (opcional): realiza asignaciones
  - where (opcional): especifica un criterio
  - order by (opcional): especifica el orden del resultado
  - return: especifica qué se devuelve en el resultado

La Cláusula for (I)

Comunidad de Madrid

- La cláusula for enlaza una variable con cada item que se devuelve por la expresión in. Esta cláusula produce una iteración.

- Puede haber múltiples cláusulas for en una misma expresión

- Para conseguir una interación un determinado número de veces la palabra reservada to. Por ejemplo:

```
for $x in (1 to 3)
return <test>{$x}</test>
```

produce el resultado:

```
<test>1</test>
<test>2</test>
<test>3</test>
```

La Cláusula for (II)

Comunidad de Madrid

- La palabra reservada at permite contar la iteración:

```
for $x at $i in doc("books.xml")/bookstore/book/title
return <book>{$i}. {data($x)}</book>
```

devuelve el resultado:

```
<book>1. Everyday Italian</book>
<book>2. Harry Potter</book>
<book>3. XQuery Kick Start</book>
<book>4. Learning XML</book>
```

La Cláusula for (III)

Comunidad de Madrid

- Se permite también más de una expresión en una cláusula for. Para ello, se separan los valores mediante comas:

```
for $x in (10,20), $y in (100,200)
return <test>x={$x} and y={$y}</test>
```

devuelve el resultado:

```
<test>x=10 and y=100</test>
<test>x=10 and y=200</test>
<test>x=20 and y=100</test>
<test>x=20 and y=200</test>
```

La Cláusula let

Comunidad de Madrid

- La cláusula let permite hacer asignaciones de variables, permitiendo evitar el tener que repetir la misma expresión varias veces

- La cláusula let no provoca una iteración:

```
let $x := (1 to 5)
return <test>{$x}</test>
```

produce el resultado:

```
<test>1 2 3 4 5</test>
```

Las Cláusulas where y order by

Comunidad de Madrid

- La cláusula where se usa para especificar uno o más criterios para el resultado:

```
where $x/price>30 and $x/price<100
```

- La cláusula order by especifica el criterio de ordenación del resultado. Así, para ordenar ejemplo por categoría y título:

```
for $x in doc("books.xml")/bookstore/book
order by $x/@category, $x/title
return $x/title
```

produciría el resultado:

```
<title lang="en">Harry Potter</title>
<title lang="en">Everyday Italian</title>
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

La Cláusula return

Comunidad de Madrid

- La cláusula return indica qué es lo que se quiere devolver

```
for $x in doc("books.xml")/bookstore/book
return $x/title
```

produce como resultado:

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

Funciones en XQuery (I)

Comunidad de Madrid

- XQuery 1.0, XPath 2.0 y XSLT 2.0 comparten la misma biblioteca de funciones

- XQuery incluye unas 100 funciones propias. Se puede ver un listado en http://www.w3.org/2005/02/xpath-functions

- Hay funciones de todo tipo: para valores tipo cadena, para valores numéricos, para comparación de fechas y horas, para manipulación de nodo y manejo de QName, para manipulaciones de secuencias, para valores booleanos y muchas otras.

- El usuario puede definir sus propias funciones

Funciones en XQuery (II)

Comunidad de Madrid

- El prefijo usado por defecto en el espacio de nombre de las funciones es fn:. Por ejemplo, fn:string()

- Sin embargo, esto no ocurre en el caso del nombre de las funciones, que no necesitan emplear el prefijo cuando son llamadas.

- Una llamada a una función puede aparecer en el mismo sitio donde una expresión pueda colocarse. Por ejemplo:

  - En un elemento

    ```
    <name>{uppercase($booktitle)}</name>
    ```

  - En el predicado de una expresión path

    ```
    doc("books.xml")/bookstore/book[substring(title,1,5)='Harry']
    ```

  - En una cláusula let

    ```
    let $name := (substring($booktitle,1,4))
    ```

Funciones en XQuery (III)

Comunidad de Madrid

- Se pueden definir funciones de usuario

- Pueden definirse en una consulta (query) o en una biblioteca aparte. Para ello, se emplea esta sintaxis:

```
declare function prefix:funct_name($param AS datatype) AS returnDatatype
{
        (: ...aquí se coloca el código de la función... :)
};
```

- Es obligatorio el uso de declare function

- El nombre de la función tiene que tener un prefijo

- Los tipos de los datos de los parámetros son generalmente de los mismos tipos definidos en el XML Schema

- El cuerpo de una función debe estar incluido entre llaves {}

Funciones en XQuery (IV)

Comunidad de Madrid

- Ejemplo de definición de función de usuario:

```
declare function local:minPrice
            ($price as xs:decimal,  $discount as xs:decimal) AS xs:decimal
{
      let $disc := ($price * $discount) div 100
      return ($price - $disc)
};

(: A continuación se muestra una llamada a la función :)

<minPrice>
      {local:minPrice($book/price, $book/discount)}
</minPrice>
```

Biblioteca de Funciones *(I)*: Accessor Functions, Error & Trace Functions

Comunidad de Madrid

Función	Descripción
fn:node-name(*node*)	Returns the node-name of the argument node
fn:nilled(*node*)	Returns a Boolean value indicating whether the argument node is nilled
fn:data(*item.item,...*)	Takes a sequence of items and returns a sequence of atomic values
fn:base-uri() fn:base-uri(*node*)	Returns the value of the base-uri property of the current or specified node
fn:document-uri(*node*)	Returns the value of the document-uri property for the specified node
fn:implicit-timezone()	Returns the value of the implicit timezone
fn:default-collation()	Returns the value of the default collation
fn:static-base-uri()	Returns the value of the base-uri
fn:error() fn:error(*error*) fn:error(*error,description*) fn:error(*error,description,error-object*)	Example: error(fn:QName('http://example.com/test', 'err:toohigh'), 'Error: Price is too high') Result: Returns http://example.com/test#toohigh and the string "Error: Price is too high" to the external processing environment
fn:trace(*value, label*)	Used to debug queries

Biblioteca de Funciones *(II)*: Functions on Numerics

Comunidad de Madrid

Función	Descripción
fn:number*(arg)*	Returns the numeric value of the argument. The argument could be a boolean, string, or node-set Example: number*('100')* Result: 100
fn:abs*(num)*	Returns the absolute value of the argument Example: abs*(3.14)* Result: 3.14 Example: abs*(-3.14)* Result: 3.14
fn:ceiling*(num)*	Returns the smallest integer that is greater than the number argument Example: ceiling*(3.14)* Result: 4
fn:floor*(num)*	Returns the largest integer that is not greater than the number argument Example: floor*(3.14)* Result: 3
fn:round*(num)*	Rounds the number argument to the nearest integer Example: round*(3.14)* Result: 3
fn:round-half-to-even*()*	Example: round-half-to-even*(0.5)* Result: 0 Example: round-half-to-even*(1.5)* Result: 2 Example: round-half-to-even*(2.5)* Result: 2

Biblioteca de Funciones *(III)*: Functions on Strings *(I)*

Comunidad de Madrid

Función	Descripción
fn:string*(arg)*	Returns the string value of the argument. The argument could be a number, boolean, or node-set Example: string*(314)* Result: "314"
fn:codepoints-to-string*(int, int, ...)*	Returns a string from a sequence of code points Example: codepoints-to-string(84, 104, 233, 114, 232, 115, 101*)* Result: 'Thérèse'
fn:string-to-codepoints*(string)*	Returns a sequence of code points from a string Example: string-to-codepoints*("Thérèse")* Result: 84, 104, 233, 114, 232, 115, 101
fn:codepoint-equal*(comp1,comp2)*	Returns true if the value of comp1 is equal to the value of comp2, according to the Unicode code point collation *(*http://www.w3.org/2005/02/xpath-functions/collation/codepoint*)*, otherwise it returns false
fn:compare*(comp1,comp2)* fn:compare*(comp1,comp2,collation)*	Returns -1 if comp1 is less than comp2, 0 if comp1 is equal to comp2, or 1 if comp1 is greater than comp2 *(*according to the rules of the collation that is used*)* Example: compare*('ghi', 'ghi')* Result: 0

Biblioteca de Funciones (IV): Functions on Strings (II)

Comunidad de Madrid

Función	Descripción
fn:concat(*string,string,...*)	Returns the concatenation of the strings Example: concat('XPath ','is ','FUN!') Result: 'XPath is FUN!'
fn:string-join(*(string,string,...),sep*)	Returns a string created by concatenating the string arguments and using the sep argument as the separator Example: string-join(('We', 'are', 'having', 'fun!'), ' ') Result: ' We are having fun! ' Example: string-join(('We', 'are', 'having', 'fun!')) Result: 'Wearehavingfun!' Example:string-join((), 'sep') Result: ''
fn:substring(*string,start,len*) fn:substring(*string,start*)	Returns the substring from the start position to the specified length. Index of the first character is 1. If length is omitted it returns the substring from the start position to the end Example: substring('Beatles',1,4) Result: 'Beat' Example: substring('Beatles',2) Result: 'eatles'
fn:string-length(*string*) fn:string-length()	Returns the length of the specified string. If there is no string argument it returns the length of the string value of the current node Example: string-length('Beatles') Result: 7

Biblioteca de Funciones *(V)*: Functions on Strings *(III)*

Comunidad de Madrid

Función	Descripción
fn:normalize-space(*string*) fn:normalize-space()	Removes leading and trailing spaces from the specified string, and replaces all internal sequences of white space with one and returns the result. If there is no string argument it does the same on the current node Example: normalize-space(' The XML ') Result: 'The XML'
fn:normalize-unicode()	
fn:upper-case(*string*)	Converts the string argument to upper-case Example: upper-case('The XML') Result: 'THE XML'
fn:lower-case(*string*)	Converts the string argument to lower-case Example: lower-case('The XML') Result: 'the xml'
fn:translate(*string1,string2,string3*)	Converts string1 by replacing the characters in string2 with the characters in string3 Example: translate('12:30','30','45') Result: '12:45' Example: translate('12:30','03','54') Result: '12:45' Example: translate('12:30','0123','abcd') Result: 'bc:da'

Biblioteca de Funciones (VI): Functions on Strings (IV)

Comunidad de Madrid

Función	Descripción
fn:escape-uri(*stringURI,esc-res*)	Example: escape-uri("http://example.com/test#car", true()) Result: "http%3A%2F%2Fexample.com%2Ftest#car" Example: escape-uri("http://example.com/test#car", false()) Result: "http://example.com/test#car" Example: escape-uri ("http://example.com/-bébé", false()) Result: "http://example.com/-b%C3%A9b%C3%A9"
fn:contains(*string1,string2*)	Returns true if string1 contains string2, otherwise it returns false Example: contains('XML','XM') Result: true
fn:starts-with(*string1,string2*)	Returns true if string1 starts with string2, otherwise it returns false Example: starts-with('XML','X') Result: true
fn:ends-with(*string1,string2*)	Returns true if string1 ends with string2, otherwise it returns false Example: ends-with('XML','X') Result: false
fn:substring-before(*string1,string2*)	Returns the start of string1 before string2 occurs in it Example: substring-before('12/10','/') Result: '12'

Biblioteca de Funciones *(VII)*: Functions on Strings *(V)*

Comunidad de Madrid

Función	Descripción
fn:substring-after(*string1,string2*)	Returns the remainder of string1 after string2 occurs in it Example: substring-after('12/10','/') Result: '10'
fn:matches(*string,pattern*)	Returns true if the string argument matches the pattern, otherwise, it returns false Example: matches("Merano", "ran") Result: true
fn:replace(*string,pattern,replace*)	Returns a string that is created by replacing the given pattern with the replace argument Example: replace("Bella Italia", "l", "*") Result: 'Be**a Ita*ia' Example: replace("Bella Italia", "l", "") Result: 'Bea Itaia'
fn:tokenize(*string,pattern*)	Example: tokenize("XPath is fun", "\s+") Result: ("XPath", "is", "fun")
fn:resolve-uri(*relative,base*)	

Biblioteca de Funciones *(VIII)*: Functions on Booleans

Comunidad de Madrid

Función	Descripción
fn:boolean*(arg)*	Returns a boolean value for a number, string, or node-set
fn:not*(arg)*	The argument is first reduced to a boolean value by applying the boolean*()* function. Returns true if the boolean value is false, and false if the boolean value is true Example: not*(*true*())* Result: false
fn:true*()*	Returns the boolean value true Example: true*()* Result: true
fn:false*()*	Returns the boolean value false Example: false*()* Result: false

Biblioteca de Funciones *(IX)*: Functions on Durations, Dates and Times *(I)*

Comunidad de Madrid

Función	Descripción
fn:dateTime(*date,time*)	Converts the arguments to a date and a time
fn:years-from-duration(*datetimedur*)	Returns an integer that represents the years component in the canonical lexical representation of the value of the argument
fn:months-from-duration(*datetimedur*)	Returns an integer that represents the months component in the canonical lexical representation of the value of the argument
fn:days-from-duration(*datetimedur*)	Returns an integer that represents the days component in the canonical lexical representation of the value of the argument
fn:hours-from-duration(*datetimedur*)	Returns an integer that represents the hours component in the canonical lexical representation of the value of the argument
fn:minutes-from-duration(*datetimedur*)	Returns an integer that represents the minutes component in the canonical lexical representation of the value of the argument
fn:seconds-from-duration(*datetimedur*)	Returns a decimal that represents the seconds component in the canonical lexical representation of the value of the argument

Biblioteca de Funciones *(X)*:
Functions on Durations, Dates and Times *(II)*

Comunidad de Madrid

Función	Descripción
fn:year-from-dateTime*(datetime)*	Returns an integer that represents the year component in the localized value of the argument Example: year-from-dateTime*(*xs:dateTime*("*2005-01-10T12:30-04:10"*))* Result: 2005
fn:month-from-dateTime*(datetime)*	Returns an integer that represents the month component in the localized value of the argument Example: month-from-dateTime*(*xs:dateTime*("*2005-01-10T12:30-04:10"*))* Result: 01
fn:day-from-dateTime*(datetime)*	Returns an integer that represents the day component in the localized value of the argument Example: day-from-dateTime*(*xs:dateTime*("*2005-01-10T12:30-04:10"*))* Result: 10
fn:hours-from-dateTime*(datetime)*	Returns an integer that represents the hours component in the localized value of the argument Example: hours-from-dateTime*(*xs:dateTime*("*2005-01-10T12:30-04:10"*))* Result: 12

Biblioteca de Funciones *(XI)*:
Functions on Durations, Dates and Times *(III)*

Comunidad de Madrid

Función	Descripción
fn:minutes-from-dateTime*(datetime)*	Returns an integer that represents the minutes component in the localized value of the argument Example: minutes-from-dateTime*(*xs:dateTime*("*2005-01-10T12:30-04:10"*))* Result: 30
fn:seconds-from-dateTime*(datetime)*	Returns a decimal that represents the seconds component in the localized value of the argument Example: seconds-from-dateTime*(*xs:dateTime*("*2005-01-10T12:30:00-04:10"*))* Result: 0
fn:timezone-from-dateTime*(datetime)*	Returns the time zone component of the argument if any
fn:year-from-date*(date)*	Returns an integer that represents the year in the localized value of the argument Example: year-from-date*(*xs:date*("*2005-04-23"*))* Result: 2005
fn:month-from-date*(date)*	Returns an integer that represents the month in the localized value of the argument Example: month-from-date*(*xs:date*("*2005-04-23"*))* Result: 4

Biblioteca de Funciones *(XII)*: Functions on Durations, Dates and Times *(IV)*

Comunidad de Madrid

Función	Descripción
fn:day-from-date*(date)*	Returns an integer that represents the day in the localized value of the argument Example: day-from-date*(*xs:date*("*2005-04-23*"))* Result: 23
fn:timezone-from-date*(date)*	Returns the time zone component of the argument if any
fn:hours-from-time*(time)*	Returns an integer that represents the hours component in the localized value of the argument Example: hours-from-time*(*xs:time*("*10:22:00*"))* Result: 10
fn:minutes-from-time*(time)*	Returns an integer that represents the minutes component in the localized value of the argument Example: minutes-from-time*(*xs:time*("*10:22:00*"))* Result: 22
fn:seconds-from-time*(time)*	Returns an integer that represents the seconds component in the localized value of the argument Example: seconds-from-time*(*xs:time*("*10:22:00*"))* Result: 0

Biblioteca de Funciones *(XIII)*: Functions on Durations, Dates and Times *(V)*. QNames

Comunidad de Madrid

Función	Descripción
fn:timezone-from-time*(time)*	Returns the time zone component of the argument if any
fn:adjust-dateTime-to-timezone*(datetime,timezone)*	If the timezone argument is empty, it returns a dateTime without a timezone. Otherwise, it returns a dateTime with a timezone
fn:adjust-date-to-timezone*(date,timezone)*	If the timezone argument is empty, it returns a date without a timezone. Otherwise, it returns a date with a timezone
fn:adjust-time-to-timezone*(time,timezone)*	If the timezone argument is empty, it returns a time without a timezone. Otherwise, it returns a time with a timezone
fn:QName*()*	
fn:local-name-from-QName*()*	
fn:namespace-uri-from-QName*()*	
fn:namespace-uri-for-prefix*()*	
fn:in-scope-prefixes*()*	
fn:resolve-QName*()*	

Biblioteca de Funciones *(XIV)*: Functions on Nodes & Sequences *(I)*

Comunidad de Madrid

Función	Descripción
fn:name*()* fn:name*(nodeset)*	Returns the name of the current node or the first node in the specified node set
fn:local-name*()* fn:local-name*(nodeset)*	Returns the name of the current node or the first node in the specified node set - without the namespace prefix
fn:namespace-uri*()* fn:namespace-uri*(nodeset)*	Returns the namespace URI of the current node or the first node in the specified node set
fn:lang*(lang)*	Returns true if the language of the current node matches the language of the specified language Example: Lang("en") is true for ... Example: Lang("de") is false for ...
fn:root*()* fn:root*(node)*	Returns the root of the tree to which the current node or the specified belongs. This will usually be a document node
fn:index-of*(/item,item,...),searchitem)*	Returns the positions within the sequence of items that are equal to the searchitem argument Example: index-of *((15, 40, 25, 40, 10), 40)* Result: *(2, 4)* Example: index-of *(("a", "dog", "and", "a", "duck"), "a")* Result *(1, 4)* Example: index-of *((15, 40, 25, 40, 10), 18)* Result: *()*

Biblioteca de Funciones *(XV)*: Functions on Nodes & Sequences *(II)*

Comunidad de Madrid

Función	Descripción
fn:remove(*(item,item,...),position*)	Returns a new sequence constructed from the value of the item arguments - with the item specified by the position argument removed Example: remove(*(*"ab", "cd", "ef"*),* 0*)* Result: *(*"ab", "cd", "ef"*)* Example: remove(*(*"ab", "cd", "ef"*),* 1*)* Result: *(*"cd", "ef"*)* Example: remove(*(*"ab", "cd", "ef"*),* 4*)* Result: *(*"ab", "cd", "ef"*)*
fn:empty(*item,item,...*)	Returns true if the value of the arguments IS an empty sequence, otherwise it returns false Example: empty(remove(*(*"ab", "cd"*),* 1*))* Result: false
fn:exists(*item,item,...*)	Returns true if the value of the arguments IS NOT an empty sequence, otherwise it returns false Example: exists(remove(*(*"ab"*),* 1*))* Result: false
fn:distinct-values(*(item,item,...),collation*)	Returns only distinct (different) values Example: distinct-values*((*1, 2, 3, 1, 2*))* Result: *(*1, 2, 3*)*

Biblioteca de Funciones (XVI): Functions on Nodes & Sequences (III)

Comunidad de Madrid

Función	Descripción
fn:insert-before(*/item,item,…/,pos,inserts*)	Returns a new sequence constructed from the value of the item arguments - with the value of the inserts argument inserted in the position specified by the pos argument Example: insert-before(("ab", "cd"), 0, "gh") Result: ("gh", "ab", "cd") Example: insert-before(("ab", "cd"), 1, "gh") Result: ("gh", "ab", "cd") Example: insert-before(("ab", "cd"), 2, "gh") Result: ("ab", "gh", "cd") Example: insert-before(("ab", "cd"), 5, "gh") Result: ("ab", "cd", "gh")
fn:reverse(*/item,item,…/*)	Returns the reversed order of the items specified Example: reverse(("ab", "cd", "ef")) Result: ("ef", "cd", "ab") Example: reverse(("ab")) Result: ("ab")

Biblioteca de Funciones *(XVII)*: Functions on Nodes & Sequences *(IV)*. Test of Cardinality

Comunidad de Madrid

Función	Descripción
fn:subsequence*(/item,item,…/,start,len)*	Returns a sequence of items from the position specified by the start argument and continuing for the number of items specified by the len argument. The first item is located at position 1 Example: subsequence*((*$item1, $item2, $item3,…*), 3)* Result: *(*$item3, …*)* Example: subsequence*((*$item1, $item2, $item3, …*), 2, 2)* Result: *(*$item2, $item3*)*
fn:unordered*(/item,item,…/)*	Returns the items in an implementation dependent order
fn:zero-or-one*(item,item,…)*	Returns the argument if it contains zero or one items, otherwise it raises an error
fn:one-or-more*(item,item,…)*	Returns the argument if it contains one or more items, otherwise it raises an error
fn:exactly-one*(item,item,…)*	Returns the argument if it contains exactly one item, otherwise it raises an error

Biblioteca de Funciones *(XVIII)*: Aggregate Functions

Comunidad de Madrid

Función	Descripción
fn:count*(/item, item,.../)*	Returns the count of nodes
fn:avg*(/arg, arg,.../)*	Returns the average of the argument values Example: avg*((1,2,3))* Result: 2
fn:max*(/arg, arg,.../)*	Returns the argument that is greater than the others Example: max*((1,2,3))* Result: 3 Example: max*(('a', 'k'))* Result: 'k'
fn:min*(/arg, arg,.../)*	Returns the argument that is less than the others Example: min*((1,2,3))* Result: 1 Example: min*(('a', 'k'))* Result: 'a'
fn:sum*(arg, arg,...)*	Returns the sum of the numeric value of each node in the specified node-set
fn:id*(/string,string,.../,node)*	Returns a sequence of element nodes that have an ID value equal to the value of one or more of the values specified in the string*(s)*
fn:idref*(/string,string,.../,node)*	Returns a sequence of element or attribute nodes that have an IDREF value equal to the value of one or more of the values specified in the string argument
fn:doc-available*(URI)*	Returns true if the doc*()* function returns a document node, otherwise it returns false
fn:collection*()* fn:collection*(string)*	

Biblioteca de Funciones (XIX): Context Functions

Comunidad de Madrid

Función	Descripción
fn:position()	Returns the index position of the node that is currently being processed Example: //book[position()<=3] Result: Selects the first three book elements
fn:last()	Returns the number of items in the processed node list Example: //book[last()] Result: Selects the last book element
fn:current-dateTime()	Returns the current dateTime (with timezone)
fn:current-date()	Returns the current date (with timezone)
fn:current-time()	Returns the current time (with timezone)
fn:implicit-timezone()	Returns the value of the implicit timezone
fn:default-collation()	Returns the value of the default collation
fn:static-base-uri()	Returns the value of the base-uri