XQuery 2

1. Tipos

XQuery tiene “tipado” fuerte, toda expresión es de un tipo dado basado en XML Schema. Los tipos pueden ser:

- Pre-definidos: xs:string, xs:date, xs:decimal, …
- Importados de un esquema facilitado por el usuario

Cada tipo atómico tiene su correspondiente constructor

- xs:date(“2020-12-15”) crea un valor xs:date

Aunque es fuertemente tipado admite conversiones de tipos_

- Mediante funciones:
  - number(“123) → 123
  - string(123) → “123”
- Mediante “castable as” (verifica si es convertible) o “cast as” (convierte)

En principio, sin esquema, los datos XML son “untyped”, pero el sistema automáticamente los convierte al tipo esperado excepto (a veces) en argumentos de funciones

2. Prólogo

Contiene declaraciones que marcan el entorno de la consulta. Su estructura es la siguiente:

2.1. Declaración de la versión

Es opcional

xquery version 1.0

seguida opcionalmente por el tipo de codificación de caracteres a usar

encoding utf-8

acabada por

;

2.2. Declaraciones e importaciones

Pueden aparecer una o más de:

- declare default element namespace URILiteral;
- declare default function namespace URILiteral;
- declar boundary-space preserve ;
- declare boundary-space strip ;

- declare default collation URILiteral;
- declare base-uri URILiteral;
- declare construction strip ;
- declare construction preserve ;
- declare ordering ordered ;
- declare ordering unordered ;
- declare default order empty greatest ;
- declare default order empty least :
- declare copy-namespaces preserve , inherit ;
- declare copy-namespaces preserve . no-inherit :
- declare copy-namespaces no-preserve , inherit ;
- declare copy-namespaces no-preserve , no-inherit;
- declare namespace NCName = URILiteral ;
- import schema namespace NCName = URILiteralList;
- import schema default element namespace URILiteralList;
- import schema URILiteralList;
- import module namespace NCName = URILiteraList ;
- import module URILiteraList;

Ejercicio 32

2.3. Variables globales

Si una expresión se repite mucho, podemos almacenarla en una variable e incluso guardarla en un módulo aparte para reutilizarla en varias consultas

2.4. Funciones

Además de las funciones de XQuery, el usuario puede definir funciones propias. La principal ventaja de su uso es que se rompe el código complejo en partes más manejables.

Ejemplo:

```
declare function local:subida_de_precio($precio_inicial as xs:decimal) as xs:decimal
{
    $precio_inicial*1.15
};
```

La definición debe hacerse en el prólogo y usarse en el cuerpo de la consulta

2.4.1. Estructura de la función

- Es obligatorio que la definición termine en ; (punto y coma). El compilador de XQuery es implacable marcando el punto de anclaje final de la función!!
- Nombre de la función:

```
declare function local:subida_de_precio($precio_inicial as xs:decimal) as xs:decimal
{
    $precio_inicial*1.15
};
```

El espacio de nombres loca: indica que es una función definida por el usuario para el uso local en la consulta. El nombre lo elige el usuario ateniéndose a las reglas de nombres de elementos XML

- Parámetros de las funciones:
  Nuestra función de ejemplo tiene un único argumento, pero puede tener cualquier número (cero o más) separados por comas:

```
declare function local:subida_de_precio($precio_inicial as xs:decimal) as xs:decimal
{
    $precio_inicial*1.15
};
```

Cada argumento tiene un nombre ("$precio_inicial" en nuestro ejemplo). La cláusula as declara el tipo de argumento

Podríamos no indicarlo poniendo "as item()*" que significa que cualquier tipo de valor es aceptable. Es recomendable que se declare el tipo porque es una documentación muy útil cuando se revisa la consulta pasado el tiempo o cuando se pasa a otra persona. Y además el compilador comprobará que los argumentos que se pasan a la función son del tipo correcto permitiendo que los errores se detecten mucho más rápidamente.

Los argumentos pueden ser nodos o valores atómicos.

- Cuando se trata de nodos suelen ser elementos pero también pueden ser atributos o cualquiera de los otros tipos de nodos: los nodos del documento, los nodos de texto, comentarios o incluso instrucciones de procesamiento. Puede indicarse explícitamente utilizando la siguiente tabla de ejemplos:

Tipo de argumento	Permite
node()	Cualquier nodo
element()	Cualquier elemento
element(nombre_elemento)	Elemento de nombre *nombre_elemento*
attribute()	Atributo
…	

- Para valores atómicos, se utilizan los nombres de los tipos integrados de XSchema, como xs:integer, xs:string, xs:date, xs:anyURI, etc. o pueden utilizarse los nombres de los tipos que se hayan definido en un esquema propio. En este caso, debe importarse el esquema al comienzo d ela consulta para que el compilador sepa dónde encontrar las definiciones de estos tipos.

La parte final de un tipo es la cardinalidad: decirle al sistema cuántos artículso están permitidos en una secuencia. Se puede utilizar uno de los cuatro indicadores de ocurrencia:

- Ninguno (en blanco) → exactamente un elemento (ej. as elment())
- ¿ → cero o un artículo (ej. as element()?)
- + → uno o más artículos (ej. as element()+)
- * → cualquier número de elementos (ej. as element()*)

- Tipo de resultado

declare function local:subida_de_precio($precio_inicial as xs:decimal) as xs:decimal
{
    $precio_inicial*1.15
};

Al igual que con los tipos de argumentos, el tipo de resultado:

- No es necesario declararlo, pero es recomendable, por las mismas razones
- Admite cardinalidad

El sistema no convierte automáticamente los valores (por ejemplo no convertirá un entero a una cadena o una cadena a un entero). Deberá hacerse la conversión dentro del cuerpo de la función

- Cuerpo de la función

  declare function local:subida_de_precio($precio_inicial as xs:decimal) as xs:decimal
  {
      $precio_inicial*1.15
  };

  El cuerpo de la función puede ser cualquier expresión. Nuestro ejemplo es muy simple y solo es una línea, pero puede en general puede ser todo lo compleja uqe sea necesario e incluir cláusulas FLOWR y hacer referencia a:
  - Los argumentos definidos para la función
  - Variables globales declaradas en el prólogo de la consulta
  - Funciones definidas por el usuario en el mismo módulo (incluida la propia función, lo que proporciona la recursividad)
  - Funciones definidas por el usuario en un módulo importado

  Debido a que una función puede contener cualquier expresión, no está limitada a calcular valores, sino que también puede construir nuevos nodos.
  Lo único que no está permitido es hacer referencia al nodo contexto "." Ya que está indefinido

2.5. Importación de módulos

Imaginemos que una función o una expresión que hemos guardado en uan variable global debemos usarla en muchas consultas; resultaría muy laborioso tener que escribirla en cada una de ellas. La solución es guardarla como un módulo aparte e importar dicho módulo en las consultas que la precisen

2.5.1. Creación del módulo

La primera línea del módulo debe fijar el espacio de nombres que se va a utilizar

    module namespace ejemplo="http://www.mio.com"

y a continuación si es:

- Una variable: la declaración igual que hemos visto en la variable sglobales pero poniendo el espacio de nombres que se haya declarado entre el signo $ y el nombre de la variable
- Una función: la declaración igual que hemos visto en las funciones locales pero cambiando el espacio de nombres local por el que se haya declarado

Tras guardar, para que luego sea fácil llamar a dicho módulo lo mejor es agregarlo como un documento más a la colección (NO es indispensable hacerlo así, pero a nosotros nos va bien para no tener que recordar la ruta completa donde se ha guardado)

2.5.2. Llamada al módulo

La consulta que necesite utilizar el módulo debe importarlo de la siguiente manera:

- En el prólogo, de una de las siguientes maneras:
  - Import module namespace el_que_se_haya_puesto_en_la_declaración_del_modulo at “xmldb:exist:///db/nombre_de_la_coleccion/nombre_del:modulo.xql”;
  - Import module namespace el_que_haya_puesto_en_la_declaracion_del_modulo at “ at “file:///ruta_del_modulo/nombre_del_modulo_xql.xql”;”;
- En el cuerpo de la consulta:

  Prefijo_namespace:nombre_de_la_funcion(argumentos_si_son_necesarios)

3. Constructores de nodos calculados

En una unidad anterior ya hemos visto como es posible construir nodos mediante etiquetas de apertura y cierre y {}. Hay una sintaxis alternativa utilizando los siguientes constructores que debe utilizarse cuando el nombre de los elementos o de los atributos tiene que calcularse:

- element
- attribute

4. Más sobre operadores

4.1. Aritméticos

+,-,*,div,idiv,mod: son definidos sobre valores numéricos

4.2. Lógicos

and, or, not: Se emplean para combinar condiciones lógicas

4.3. De comparación

Existen grupos de operadores de comparación:

- Comparaciones generales: =, ¡=, >=, < >, >=
  Ya los hemos utilizando tanto en XPath como practicando XQuery. La expresión siguiente devuelve true si alguno de los atributos q tiene un valor superior a 10:
  $bookstore//book/@q >10
- Comparaciones: eq, ne, lt, le , gt, ge
  Se usan las comparaciones de valor con los documentos XML con tipos definidos. Comparan dos valores escalares y produce un error si alguno de los operando es una secuencia de longitud mayor de 1.
  La siguiente expresión devuelve true si hay un solo q atributo devuelto por la expresión y su valor es mayor que 10. Si hay más de un q devuelto, se produce un error.
  $bookstore//boo/@q gt 10

- Comparadores de nodos: is, is not
  Comparan la identidad de dos nodos
  $nodo1 is $nodo2 es true si ambas variables desciende del mismo nodo padre

- Comparadores de posición de nodos: <<, >>
  Compara la posición de dos nodos
  $node1<<$node2 es true si el nodo ligado a $node1 ocurre primero en el orden del documento que el nodo ligado a $nodo2

- Sobre secuencias de nodos: union, intersect y except
  Devuelven secuendias de nodos en el orden del documento y eliminan ducplicados de las secuencias resultado

5. Caracteres especiales

‘
’ nueva línea

‘	’ TAB

‘
’ retorno de carro

6. Actualización

Hasta el momento solo se ha podido realizar consultas sobre los contenidos ya almacenados en la base de datos XML. Quizás queramos también insertar, reemplazar, renombrar, cambiar y/o borrar entradas dentro de la base de datos. Para ello indicaremos qué hacer en cada caso.

6.1. Inserción

Se quiere añadir un nuevo pokemon a la base de datos. Los datos son:

- Nombre: Buneary
- Ataque
  - A_nombre: Destructor
  - Potencia: 7
- Ataque
  - A_nombre: Rapidez
  - Potencia 60
- Tipo: normal
- Pc: 772

La inserción se realizaría de la siguiente manera:

```
(:Ejercicio 3. Inserción de un nuevo pokemon:)
insert node
<pokemon f_captura="2022">
  <nombre>Buenary</nombre>
  <ataque>
    <a_nombre>Destructor</a_nombre>
    <potencia>7</potencia>
  </ataque>
  <ataque>
    <a_nombre>Rapidez</a_nombre>
    <potencia>60</potencia>
  </ataque>
  <tipo>Normal</tipo>
  <pc>772</pc>
</pokemon>

before doc("pokemon")/pokedex/pokemon[1]
```

Este código inserta el nodo indicado en la base de datos. El nodo se insertará antes del primer nodo de la base de datos (por la cláusula “before”). Si se quiere insertar después, solo cambiaría “before” por “after”. Ambas utilizan como referencia al primer nodo (“[1]”)

Una sentencia equivalente para insertar al principio de la base de datos, sin tener que referenciar a nodo alguno de la base de datos, sería sustituyendo la última línea por esta otra:

as first into doc(“pokemon”)//pokemon

Si por el contrario, lo que se desea es insertar al final de la base de datos, sin referenciar a ningún nodo, se realizaría de la siguiente manera:

as last into doc(“pokemon”)//pokemon

6.2. Reemplazo

En la inserción anterior se cometió un error con el pc del pokemon, y con uno de los ataques.

Si en la inserción anterior se realizó antes del primer nodo de la base de datos, entonces el elemento insertado será ahora el primero. Se cambiará el dato de dos formas distintas:

- Mediante la modificación del valor de nodo
- Mediante el reemplazo del nodo completo

La modificación del valor del nodo se realizaría de la siguiente manera:

```
(: Ejercicio 4. Reemplazael pc del pokemon Buenary:)
replace value of node doc("pokemon")//pokemon[1]/pc with 227
```

La modificación del nodo completo

```
(: Ejercicio 5: reemplaza el nodo completo del primer ataque con los siguientes
datos: a_nombre= Ataque Rapido potencia=8 :)

replace node doc("pokemon")//pokemon[1]/ataque[1] with
<ataque>
  <a_nombre>Ataque Rapido</a_nombre>
  <potencia>8</potencia>
</ataque>
```

Si hubiéramos querido realizar los dos cambios en una misma expresión XQuery, simplemente tendríamos que haber escrito las dos sentencias separadas por coma (,)

6.3. Borrado

Después de la inserción y la modificación del nuevo pokemon en la base de datos, finalmente parece que lo hemos tenido que trasferir al Profesor y tenemos que eliminarlo de la base de datos:

```
(: Ejercicio 6.  Elimina el pokemon de la base de datos:)
delete node doc("pokemon")//pokemon[1]
```