Sistemas de Gestión de Datos y de la Información
Máster en Ingeniería Informática 2018-19
Sesión guiada 1

Fecha de entrega: viernes 11 de enero de 2019, 23:55h

Objetivos
Familiarizarse con la base de datos semiestructurada *eXist-db* y realizar consultas *XQuery* básicas usando el API REST desde programas Python.

Entrega
La entrega se realizará a través del Campus Virtual de la asignatura mediante un único fichero `.py`.

Evaluación
Esta sesión guiada de laboratorio se contabilizará dentro del apartado de «Ejercicios y sesiones guiadas en laboratorio» y será calificado cualitativamente como *No Apto*/*Apto*. No es obligatoria su entrega ni contiene objetivos mínimos para la evaluación continua.

Declaración de autoría e integridad
Todos los ficheros entregados contendrán una cabecera en la que se indique la asignatura, el ejercicio, el grupo y los autores. Esta cabecera también contendrá la siguiente declaración de integridad:

> *NombreAlumno1* y *NombreAlumno2* declaramos que esta solución es fruto exclusivamente de nuestro trabajo personal. No hemos sido ayudados por ninguna otra persona ni hemos obtenido la solución de fuentes externas, y tampoco hemos compartido nuestra solución con nadie. Declaramos además que no hemos realizado de manera deshonesta ninguna otra actividad que pueda mejorar nuestros resultados ni perjudicar los resultados de los demás.

No se corregirá ningún fichero que no venga acompañado de dicha cabecera.

XML

XML (*Extensible Markup Language*) es un lenguaje de marcado que sirve para la representación e intercambio de información. Usando XML podemos representar entidades mediante etiquetas que contienen distintas componentes de manera anidada. Por ejemplo podemos representar una colección de CDs como[1]:

```
<catalog>
  <cd>
    <title> Pavarotti Gala Concert </title>
    <artist> Luciano Pavarotti </artist>
    <year> 1991 </year>
    <country>UK</country>
  </cd>
  <cd>
    <title> Hide your heart </title>
    <artist> Bonnie Tyler </artist>
    <year> 1988 </year>
    <country>UK</country>
  </cd>
</catalog>
```

XML tiene características que le hacen muy interesante para distintas aplicaciones, como poder comprobar si el documento está correctamente construido (cumple un determinado DTD[2] o un *esquema XML*[3]), ser fácilmente entendible por humanos y disponer de muchas bibliotecas estándar para su análisis en la inmensa mayoría de lenguajes de programación. Sin embargo, en esta sesión de laboratorio guiada nos centraremos en 2 características: a) su capacidad para representar información semiestructurada (la estructura no es fija como en tablas relacionales, sino que es variable pero siempre viene explicada mediante sus etiquetas) y b) su naturaleza jerárquica (los datos están organizados como árboles).

Referencias

- Introduction to XML, W3Schools
  `http://www.w3schools.com/xml/xml_whatis.asp`

XQuery

XQuery es el lenguaje estándar para realizar consultas sobre documentos XML. Este lenguaje nos permite seleccionar nodos dentro del documento XML que cumplan ciertas condiciones y realizar operaciones con ellos. Las consultas XQuery siguen un esquema *FLWOR*:

- **For**: recorrer una secuencia de nodos y ejecutar el resto de instrucciones con cada uno de ellos.
- **Let**: asignar nodos (o secuencias) a variables.
- **Where**: filtrar nodos que cumplen ciertas condiciones.

[1] Extracto obtenido de `http://www.w3schools.com/xml/cd_catalog.xml`.
[2] `http://www.w3schools.com/xml/xml_dtd_intro.asp`
[3] `http://www.w3schools.com/xml/schema_intro.asp`

- **Order**: ordenar los nodos.
- **Return**: devolver nodos, posiblemente aplicando alguna operación.

Imaginemos que queremos obtener los nombres de todos los discos producidos en UK ordenados por año a partir del fichero XML anterior (`cd.xml`). La consulta XQuery sería:

```
for $e in doc("cd.xml")/catalog/cd
let $t := $e/title
where $e/country = "UK"
order by $e/year
return $t
```

y nos devolvería 2 resultados:

```
1. <title> Hide your heart </title>
2. <title> Pavarotti Gala Concert </title>
```

Notad que las variables (`$e`, `$t`) van precedidas por el símbolo `$`, y que usamos expresiones de camino como `/catalog/cd`, `/title` o `/year` para acceder a nodos internos a partir de un nodo raíz (la expresión `doc("cd.xml")` nos devuelve el nodo raíz del documento `"cd.xml"`).

Referencias

- XQuery Tutorial, W3Schools
  `https://www.w3schools.com/xml/xquery_intro.asp`
- XQuery: search across a variety of XML data. Priscilla Walmsley. O'Reilly, 2015
  `https://ucm.on.worldcat.org/oclc/932322836`

eXist-db

eXist-db es una base de datos XML nativa. Esto quiere decir que la información es almacenada en documentos XML organizados en distintas colecciones, y las consultas se realizan a través de XQuery. En esta sesión de laboratorio guiada vamos a crear una colección con un documento XML sobre la que realizaremos varias consultas: un bloque de consultas utilizando la propia interfaz web de eXist-db y otro bloque accediendo a eXist-db usando el lenguaje Python.

El primer paso es crear una colección `sgdi` que contenga el siguiente fichero:

- `books.xml`
  (`https://msdn.microsoft.com/es-es/library/ms762271.aspx`)

Para ello primero arrancamos eXist-db ejecutando el comando ServicioeXistDB.sh en un terminal. Una vez arrancado, accedemos a la página principal `http://localhost:8080/exist/apps/dashboard` mediante cualquier navegador (ver Figura 1) y dentro de aquí al apartado `Collections` donde podréis crear nuevas colecciones y añadir archivos a las colecciones. Antes de poder acceder deberemos introducir los credenciales, que en los equipos del laboratorio son:

- Usuario: **admin**
- Contraseña: *la contraseña es vacía*

Una vez que tengamos la colección `sgdi` con el fichero `books.xml`, accedemos a *eXide* a través de la misma página principal de eXist-db. eXide es el entorno integrado de desarrollo XQuery que nos va a permitir visualizar los documentos XML, crear consultas XQuery y ejecutarlas (ver Figura 2). eXide

Figura 1: Página principal de eXist-db

colorea la sintaxis XQuery para que sea más fácil de leer, además de mostrar los errores sintácticos mientras se escribe la consulta. Para ejecutar una consulta, pulsaremos el botón `New XQuery`, escribiremos la consulta y la ejecutaremos con el botón `Eval`. Los resultados de la consulta aparecerán en el panel inferior. Por defecto eXide nos mostrará los resultados en bloques de 10, y para ver el resto de resultados tendremos que utilizar las flechas que aparecen en la cabecera del panel. Opcionalmente podremos almacenar las distintas consultas como documentos dentro de nuestra colección.

Consultas XQuery en eXist-db desde Python usando la interfaz REST

Para conectar con eXist-db desde Python vamos a utilizar su interfaz REST, que nos permitirá realizar consultas como peticiones HTTP y recibir los resultados como cadenas de texto representando un documento XML. Podéis encontrar la documentación detallada de los distintos tipos de petición y de los posibles parámetros en `https://exist-db.org/exist/apps/doc/devguide_rest`. Existe una biblioteca llamada `eulexistdb` que facilitaría un poco este proceso, pero lleva años sin ser mantenida y además no ha sido probada en Python 3, así que es muy posible que genere problemas.

Sin embargo, realizar peticiones REST en Python es muy sencillo gracias a la biblioteca `requests`, tal y como se aprecia en el siguiente fragmento de código que obtiene los primeros 5 libros del fichero `books.xml` perteneciente a la colección `sgdi`:

```
1 import requests
2 import xml.etree.ElementTree as ET
3
4 # Consulta XQuery que queremos ejecutar
5 q = """for $e in doc("/db/sgdi/books.xml")/catalog/book
```

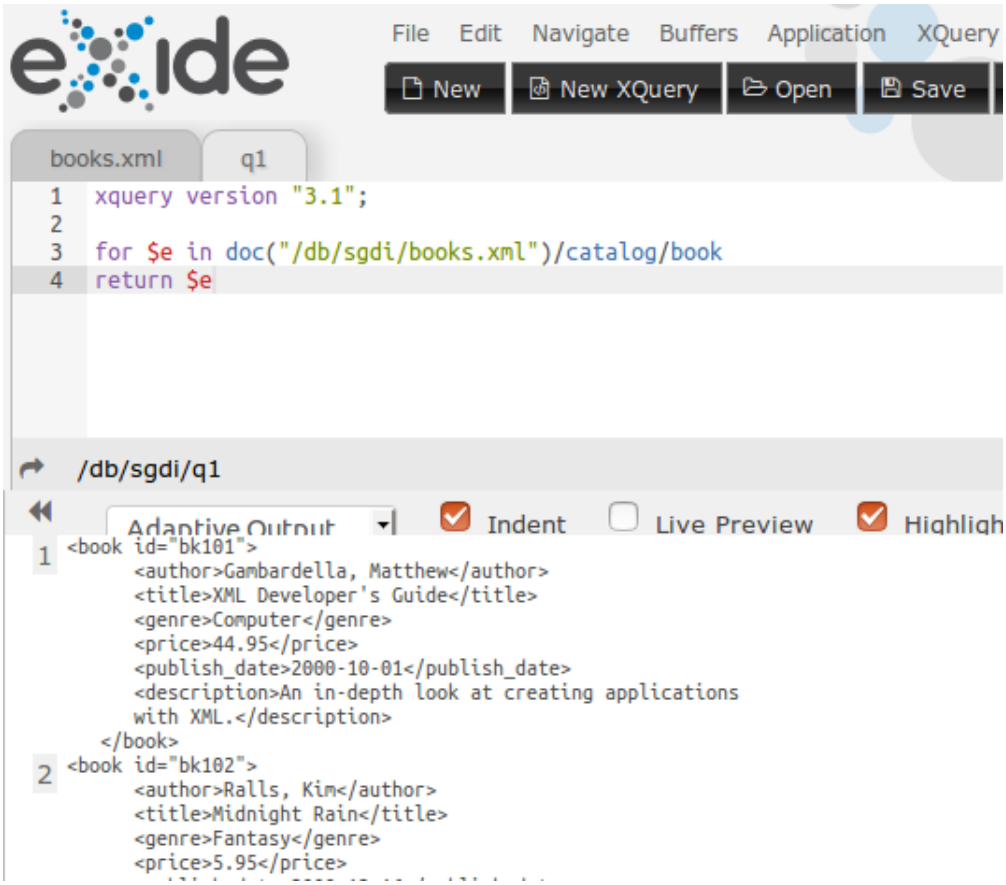

Figura 2: eXide, el IDE para XQuery de eXist-db

```
6         return $e"""
7
8  # Creamos un diccionario con los datos a enviar en la peticion
9  req = {'_query': q, '_howmany':5}
10
11 # Realizamos la peticion al servidor REST de eXist-db
12 # 'r' es de tipo requests.models.Response.
13 r = requests.post('http://localhost:8080/exist/rest/db', data=req)
14
15 # Creamos un arbol XML a partir del texto de la respuesta 'r'
16 root = ET.fromstring(r.text)
17
18 # 'root' sera de tipo xml.etree.ElementTree.Element
19 # Ahora tendremos que recorrer sus hijos para mostrar la informacion
20 # deseada como por ejemplo el texto de la etiqueta " title "
```

Realizaremos consultas mediante peticiones POST a la URL `http://localhost:8080/exist/rest/db` (también se pueden realizar peticiones GET si lo preferís). En estas peticiones incluiremos el texto de la consulta XQuery asociado a la clave `_query` y el número de resultados a recibir asociado a la clave `_howmany`. El resultado será un objeto de tipo `requests.models.Response` en el cual podremos consultar el código de estado HTTP y acceder al texto de la contestación, entre otras posibilidades. El texto de la contestación representará un árbol XML, y por ello lo transformaremos en un objeto de tipo `xml.etree.ElementTree.Element` mediante `ET.fromstring`. A partir de aquí tendremos un árbol cuyos

nodos son los resultados de la consulta. Prestad atención al nodo raíz, ya que contiene atributos indicando el número de resultados potenciales, el número de resultados devueltos y la posición del primer resultado devuelto. Cada resultado será un objeto Element que tendremos que procesar para acceder a la información relevante: conocer su etiqueta XML (nodo.tag), recorrer sus hijos directos (for e **in** nodo:), obtener el texto del nodo (nodo.text), acceder a sus atributos (nodo.attrib), etc.

Referencias

- eXist-db Documentation
  `http://exist-db.org/exist/apps/doc/documentation.xml`
- eXist. Erik Siegel, Adam Retter. O'Reilly, 2014
  `https://ucm.on.worldcat.org/oclc/913107840`
- xml.etree.ElementTree.Element
  `https://docs.python.org/3.7/library/xml.etree.elementtree.html#element-objects`

Ejercicios para entregar

Crea un fichero `.py` con una función Python por cada consulta que devuelva una **lista** con todos los resultados. Las consultas se realizan sobre el fichero `books.xml`.

1. Título (cadena de texto) de todos los libros de informática (género «Computer»).
2. Título (cadena de texto) del libro con identificador "bk105".
3. Tuplas (titulo, precio) con la información de todos los libros. También se permite devolver diccionarios Python en lugar de tuplas.
4. Nombres (cadena de texto) de los libros que cuestan más de 15€, ordenados por fecha de publicación ascendente.
5. Tuplas o diccionarios `(titulo, autor, precio)` de los libros de género «Computer» que cuestan más de 40€.