BASES DE DATOS NO CONVENCIONALES

PAULA CARBALLO PÉREZ

ESTER CORTÉS GARCÍA

ÍNDICE

1.- INTRODUCCIÓN

Las bases de datos son entidades que permiten almacenar datos de manera estructurada, buscando la menor redundancia posible. Las bases de datos convencionales son de tipo relacional, esto es, constan de listas tabulares de datos que se relacionan entre sí, almacenando de la forma más eficiente posible la información.

Pero en los últimos años, con el crecimiento de Internet y con la forma en la que las aplicaciones gestionan los datos, la escalabilidad y el rendimiento de las bases de datos relacionales se ha visto mermado. Han surgido así nuevos modelos que permiten gestionar grandes cantidades de datos, que pueden ser introducidos y extraídos rápidamente. En estos sistemas importa más la flexibilidad, la velocidad y la capacidad de escalado horizontal que otras cuestiones tradicionalmente cruciales como la consistencia o disponer de una estructura perfectamente definida para los datos.

Las bases de datos de tipo documental son uno de esos nuevos modelos, y permiten almacenar datos semiestructurados como documentos, generalmente en formato XML o JSON. Almacenan los documentos como un único atributo.

Otro tipo de bases de datos NoSQL son las orientadas a grafos, en las que se representa la información como nodos de un grafo y sus relaciones con las aristas de este.

MongoDB es una de las bases de datos orientadas a documentos más utilizada y con la que trabajaremos en esta práctica. Utilizaremos la base de datos de DBLP Computer Science Bibliography, que almacena los datos relativos a la gran mayoría de revistas científicas y congresos académicos sobre informática.

Asimismo, trabajaremos con Neo4j como base de datos orientada a grafos para un análisis posterior con la misma fuente de datos.

2.- PROCESADO DE DATOS

El dataset de partida tiene un tamaño de 2,10Gb y se encuentra en formato en XML. Está formado por ocho tipos de elementos, que son:

- Article
- Inproceedings
- Incollection
- Proceedings
- Book
- Phdthesis
- Mastersthesis
- www

Los tres primeros elementos hacen referencia a artículos, ya sean en revista, congresos o libros. Cada uno de ellos, además, tiene distintos atributos como son el autor, la fecha, el título, la url, etc.

Para poder determinar con qué elementos del dataset nos queremos quedar, analizamos el enunciado en detalle y las consultas que se nos piden.

En un primer intento de parsear el dataset para quedarnos con lo que nos interesa, utilizamos la librería *xml.etree.ElementTree*. Nos quedamos con los documentos Article", "Inproceedings" e "Incollection", y con los atributos "Author", "Title" y "Year". A continuación, convertiríamos el dataset resultante a formato JSON utilizando la librería *xmltodict*. Dado el tamaño del dataset, este proceso era muy lento, sobre 6-7 horas, por lo que decidimos proceder de otra forma.

En lugar de eliminar elementos del dataset, lo que hacemos es crear un diccionario, al que añadimos la información que nos interesa:

- Los documentos "Article", "Inproceedings" e "Incollection"
- Los atributos de los citados documentos: "Author", "Title" y "Year"
- Creamos un nuevo atributo al que llamaremos "TYPE"

Toda esta información la guardamos automáticamente en un archivo de formato JSON, de modo que el tiempo de ejecución se reduce a 30 minutos. Nuestra base de datos, a la que llamaremos "*practica*" estará

formada por una única colección, llamada "*dblp*", con documentos de la forma:

{"author": [{"Author": "A1"}, {"Author": "A2"}, {"Author": "A3"}]

"type": "…"

"title": "…"

"year": "…"}

Donde "type" es el nuevo atributo que hemos creado y que hace referencia a "article", "inproceeding" o "incollection".

3.- PARTE I: MONGODB

Como ya se comentó en el apartado anterior, hemos creado una base de datos, *practica*, que está formada por una única colección, *dblp*, siguiendo los documentos el esquema que se muestra a continuación:

```
{  "_id" : ObjectId("5ad0c315fc220f133cc83c9b"),
      "author" : [
            {
                  "Author" : "Walter Vogler"
            },
            {
                  "Author" : "Christian Stahl"
            },
            {
                  "Author" : "Richard Müller 0001"
            }
      ],
      "type" : "article",
      "title" : "Trace- and failure-based semantics for responsiveness.",
      "year" : "2014"
}
```

Para la introducción de datos en MongoDB, se intenta en primer lugar realizarlo utilizando la siguiente instrucción:

mongoimport --db practica --collection dblp --type json --file output.json

Pero nos encontramos con un problema de conexión, por lo que lo hacemos utilizando Python con la librería *pymongo* (archivo *mongodb.py* de la documentación presentada). En el citado fichero, guardamos el atributo "year" como entero, pues será necesario para realizar las consultas 8 y 9.

Consultas realizadas

Se adjunta un archivo, *consultas_mongodb.txt*, con las consultas realizadas.

Puesto que estamos trabajando con una base de datos bastante grande, en algunas consultas nos hemos encontrado con problemas de memoria. Conseguimos solventarlo añadiendo ***{allowDiskUse:true}*** al final de la consulta.

Sobre las consultas, destacamos lo siguiente:

- La consulta 5, número de “articles” y de “inproceedings” de los 10 autores con más publicaciones, se realiza en dos pasos. En primer lugar, se crea una nueva colección con esos 10 autores y con todos y cada uno de los tipos de publicación que tienen. En el segundo paso, se calcula el número de “articles” y de “inproceedings” que tienen cada uno de ellos.
- En la consulta 8, dado que en el enunciado se indica que algunas publicaciones se remontan a los años 60, se realiza una comprobación con el autor de más edad, puesto que obtenemos una edad de 75 años. Para ello, con la consulta:

  db.dblp.find ({"author.Author":"Alan M. Turing"},{"year":1})

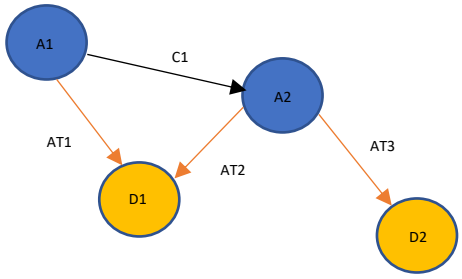
  obtenemos un listado con el año de sus publicaciones, donde podemos comprobar que la primera es del año 1937 y la última del 2012.

4.- PARTE II: NEO4J

Diseñamos nuestra base de datos en función de las consultas que queremos realizar. Consideramos las siguientes consultas:

- Número de publicaciones cada autor.
- Listado de publicaciones de un autor determinado
- Listado de coautores de un autor
- Número medio de autores de todas publicaciones del conjunto de datos

Tendrá el siguiente esquema:



Donde A_i son los autores (Author), D_i las publicaciones (Document), AT_i las relaciones de autoría (Authorship) y C_i las relaciones de colaboración (Collaboration).

Como la base de datos Neo4J es capaz de importar ficheros csv será necesario transformar nuestro fichero xml a dicho formato. Para la transformación emplearemos diccionarios, uno por cada tipo de nodo y uno por cada tipo de relación, obteniendo en total 4 diccionarios, que se convertirán en dataframes y, posteriormente, se guardarán en ficheros csv.

A la hora de transformar el diccionario de documentos a tipo dataframe observamos que el array de años tiene 6 entradas menos que el resto de arrays de los atributos. Como no vamos a realizar consultas en esta base de datos relacionadas con años, decidimos completar en esos casos el atributo "year" con el entero 0000.

Una vez definidos los nodos y las relaciones se realiza un import de los 4 ficheros a la base de datos a Neo4J con la siguiente instrucción, siendo RUTA_NEO4J el path en el que se encuentra almacenado el programa:

RUTA_NEO4J\bin\neo4j-admin import --mode csv --database practica.db --nodes import\authors.csv --nodes import\documents.csv --relationships import\AA_relationships.csv --relationships import\AP_relationships.csv

5.- CONCLUSIONES

La primera diferencia entre una base de datos y otra con la que nos encontramos, es el tiempo necesario para el procesado de los datos, de tal modo que es más rápido generar el JSON para MongoDB que generar el CSV para Neo4J.

Por otro lado, el mismo tipo de consulta en ambas bases queda definido por unas pocas líneas en Neo4J mientras que en MongoDB son más complejas, teniendo que definir incluso nuevas colecciones. Asimismo, Neo4J es más eficiente para esas mismas consultas puesto que el acceso a los autores y a las publicaciones es directo, mientras que en MongoDB tienes que ir recorriendo cada documento para acceder a sus atributos.

Por último, el hecho de tener que recorrer toda la base de datos en MongoDB origina problemas de memoria que, tal y como ya se ha comentado, hemos solucionado añadiendo ***{allowDiskUse:true}*** al final de la consulta. Este problema no se presenta en Neo4J.