Bases de Datos no Convencionales: Práctica

Máster en Data Science. URJC

David Córdoba Ruiz y Maria Cruz Gálvez Ortiz

Resumen de ficheros que componen la práctica:

- **Programas**:

  1) transform_to_json.py: toma el fichero xml y lo transforma a varios fichero en formato json cada uno correspondiente a una de las colecciones que queremos crear;

  2) enter_data_in_Mongodb.py: conecta con la base de datos MongoDB, crea una base de datos llamada dblp y escribe los fichero json como colecciones en esa base de datos;

  3) queries.py: realiza las consultas de los 10 apartados de la parte I de la práctica;

  4) transform_csv.py: transforma los ficheros json a csv adaptando a la estructura elegida para su paso a Neo4j.

- **Datos**:

  1) dblp_article.json: datos referentes a las publicaciones tipo "article".

  2) dblp_inproceedings.json: datos referentes a las publicaciones tipo "inproceedings".

  3) dblp_incollection.json: datos referentes a las publicaciones tipo "incollection".

  4) csv_autores.csv: fichero que contiene el nombre de todos los autores.

  5) csv_art.csv: fichero que contiene el identificador (key) y el titulo de los artículos.

  6) csv_years.csv: fichero que contiene los años donde hay publicaciones.

  7) csv_pub.csv: fichero para las relaciones, contiene la información de las publicaciones, el identificador, el titulo, autores y año de publicación.

Resumen del procedimiento de la práctica:

En la primera parte se ha llevado a cabo la transformación del xml donde se han obtenido tres ficheros que corresponden a los tres tipos de publicaciones que hemos diferenciado, esto es, las publicaciones realizadas en revistas (*article*), conferencias (*inproceedings*) y recopilaciones (*incollection*). Posteriormente se han introducido en la base de datos MongoDB dividiendo la información en dos colecciones (*autores* y *articulos*). Esta parte finaliza con la realización de las consultas propuestas en la práctica.
En la segunda se ha hecho algo análogo a la primera. En este caso se ha transformado la información en cuatro archivos: csv_autores, csv_pub, csv_art y csv_year, de tal forma que facilitará la posterior introducción de los nodos y las relaciones pertinentes en Neo4j. Una vez en Neo4j se han realizado tres consultas sencillas para intentar deducir las ventajas y desventajas de este frente a MongoDB.

Parte I: MongoDB (80%)

1. **Captura y procesamiento de datos.**

   La captura y procesamiento de datos se refiere a descargar el fichero dblp.xml y procesarlo para convertirlo al formato JSON, pudiéndose descartar aquellos elementos que se consideren irrelevantes. Para ello, en primer lugar, debe analizar la fuente de datos y el fichero DTD que define estos elementos, así como las consultas más frecuentes a realizar para hacer un diseño del esquema de la Base de Datos de MongoDB. Para procesar el fichero XML y realizar la conversión a formato JSON, debe construir un programa en Python. Una estrategia muy práctica, aunque relativamente lenta, consiste en procesar el fichero por "fragmentos" de menor tamaño.

   La transformación del fichero dblp.xml a json se ha realizado como nos sugiere el enunciado. Es decir, se va accediendo a cada campo del árbol del xml y se van introduciendo en los archivos, que hemos diferenciado en dblp_article, dblp_inproceedings y dblp_incollection, ya en formato json. Nos queda al final un archivo de article de 655 MB, 791 MB de inproceedings y 16 MB de incollection.

   Ejemplo: En el archivo dblp.xml una publicación se presentan así:

```
(...)
<article mdate="2017-05-28" key="journals/acta/GoodmanS83">
        <author>Nathan Goodman</author>
        <author>Oded Shmueli</author>
        <title>NP-complete Problems Simplified on Tree Schemas.</title>
        <pages>171-178</pages>
        <year>1983</year>
        <volume>20</volume>
        <journal>Acta Inf.</journal>
        <url>db/journals/acta/acta20.html#GoodmanS83</url>
        <ee>https://doi.org/10.1007/BF00289414</ee>
</article>
(...)
```

   Al tranformarlo a json con el programa *transform_to_json_proof.py* se obtiene:

```
(...)
"article": [
            (...)
             {
                "@key": "journals/acta/GoodmanS83",
                "author": ["Nathan Goodman", "Oded Shmueli"],
                "title": "NP-complete Problems Simplified on Tree Schemas.",
                "pages": "171-178",
                "year": 1983,
                "volume": 20,
                "journal": "Acta Inf.",
                "url": "db/journals/acta/acta20.html#GoodmanS83",
                "ee": "https://doi.org/10.1007/BF00289414"
             }
            (...)
           ]
(...)
```

2. **Almacenamiento de datos.**

   Una vez convertidos al formato necesario (y adecuadamente pre-procesados como se prefiera), los datos deben almacenarse en una base de datos MongoDB. En la memoria debe explicar claramente y justificar el diseño de las colecciones implementadas en la base de datos.

   Hemos decidido dividir la información de las publicaciones en dos colecciones: *autores* y *artículos*.

   - En *autores* hemos añadido los campos _id con valor el nombre del autor (lo hemos considerado como un id puesto que en la documentación se indica que los nombres de los autores son diferentes incluso si dos autores distintos tienen el mismo nombre). Además, contiene el campo *publicaciones* que es una lista de las id publicaciones (este dado en el xml por el campo @key) junto con el año de publicación. Este último campo se repetirá en la colección publicaciones, dato que hemos tenido en cuenta pero que era necesario para una mayor facilidad para realizar las consultas posteriores.
   - En la colección *articulos* tenemos el campo *id*, el campo *autores* que contiene una lista de todos los nombres de los autores que han firmado en la publicación, el campo *título* con el nombre de la publicación y, además, se le ha añadido un campo *type* que nos informa de donde se ha publicado (revista, conferencia o recopilación).

   Esta división en colecciones y campos ha venido motivada por las consultas que se deben realizar posteriormente. Hemos visto claramente que hay una información o unas características que se requieren de un conjunto de autores o de un conjunto de artículos de forma independiente. La elección de los campos ha venido asignada así para facilitar esas consultas evitando la interacción (en todo lo posible) entre colecciones. En función de esto, justificamos la incorporación del campo *year* en autores (además de en colecciones como hemos mencionado) para facilitar las consultas 8 y 9. Otro factor ha sido que veíamos natural que el año se incluyera junto con las publicaciones de cada autor.

   Por conveniencia, en la introdución a MongoDB se han escogido los campos *key, title, author* y *year*, y solo si estos campos no están vacíos.

   Todo esto lo realizamos en python con el fichero *enter_data_in_Mongodb.py*, en el cual conectamos con la base de datos MongoDB, creamos una base de datos llamada *dblp* y las colecciones ya descritas y dividimos la información de los ficheros json en estas. Se han metido los ficheros en MongoDB por separado, tal vez habría sido más eficaz meter todos juntos con mongoimport.

   Ejemplo. COLECCION AUTORES:

```
{

        "_id": "Nathan Goodman",
        "publications":[  (...)
                          {
                               "_id":"journals/acta/GoodmanS83",
                               "year":1983
                          }
                      ],
        "numero_revistas": (en principio 1),
        "numero_conferencias": (en principio 0),
        "numero_incollection":(en principio 0)
},
{

        "_id": "Oded Shmueli",
        "publicaciones":[  (...)
                          {
```

```
                                "_id":"journals/acta/GoodmanS83",
                                "year":1983
                            }
                        ],
        "numero_revistas": (en principio 1),
        "numero_conferencias": (en principio 0),
        "numero_incollection":(en principio 0)
}
```

COLECCION ARTÍCULOS:

```
{
                "_id": "journals/acta/GoodmanS83",
                "autores": ["Nathan Goodman", "Oded Shmueli"],
                "titulo": "NP-complete Problems Simplified on Tree Schemas.",
                "year": 1983,
                "type": "revista"
}
```

3. **Análisis de datos.**

   En esta etapa se deben analizar (consultar) los datos almacenados, usando el lenguaje de consulta propio de la Base de Datos. Se pueden utilizar los medios que se deseen (incluso almacenamiento de datos calculados en la propia BD) con el fin de facilitar la obtención de estas respuestas de la manera ms simple posible. En las consultas en las que se haga referencia a un autor determinado, elija uno de los que esté en su conjunto de datos. En el caso de usar el dataset simplificado, el resultado obtenido puede no ser realista, dando que solo se trabaja con un año. Las consultas son las siguientes:

   Las consultas se han realizado también en python. Estas se encuentran en el programa *queries.py*. Para ejecutarlo hay que asegurarse que la base de datos contengan las colecciones "autores" y "articulos".

   Numero de autores en la base de datos: 2070360
   Numero de artículos en la base de datos: 3930817

   El autor escogido como ejemplo para las consultas es autor = Nathan Goodman

   Mostramos a continuación los resultados de las consultas:

   (a) 1. Listado de todas las publicaciones de un autor determinado.

```
{'_id': 'Nathan Goodman',
      'Titulo': ['NP-complete Problems Simplified on Tree Schemas.']}
{'_id': 'Nathan Goodman',
      'Titulo': ["Errors in 'Process Synchronization in Database Systems'."]}
{'_id': 'Nathan Goodman',
      'Titulo': ['Query Processing in a System for Distributed Databases (SDD-1).']}
(...)
```

   (b) 2. Número de publicaciones de un autor determinado.

```
{'_id': 'Nathan Goodman', 'TotalArticulos': 76}
```

   (c) 3. Número de artículos en revista para el ao 2017.

```
144745
```

(d) 4. Número de autores ocasionales, es decir, que tengan menos de 5 publicaciones en total.

```
1688848
```

(e) 5. Número de artículos de revista (article) y número de artículos en congresos (inproceedings) de los diez autores con más publicaciones totales.

```
{'_id': 'H. Vincent Poor',
      'TotalArticulos': 1496, 'NumeroRevistas': 1023, 'NumeroConferencias': 472}
{'_id': 'Mohamed-Slim Alouini',
      'TotalArticulos': 1131, 'NumeroRevistas': 601, 'NumeroConferencias': 530}
{'_id': 'Philip S. Yu',
      'TotalArticulos': 1109, 'NumeroRevistas': 391, 'NumeroConferencias': 702}
{'_id': 'Yu Zhang',
      'TotalArticulos': 1078, 'NumeroRevistas': 460, 'NumeroConferencias': 615}
(...)
```

(f) 6. Número medio de autores de todas las publicaciones que tenga en su conjunto de datos.

```
2.9576149182218354
```

(g) 7. Listado de coautores de un autor (Se denomina coautor a cualquier persona que haya firmado una publicación).

```
{'_id': 'Nathan Goodman',
    'Titulo': ['NP-complete Problems Simplified on Tree Schemas.'],
    'Coautores': [['Nathan Goodman', 'Oded Shmueli']]}
{'_id': 'Nathan Goodman',
    'Titulo': ["Errors in 'Process Synchronization in Database Systems'."],
    'Coautores': [['Philip A. Bernstein', 'Marco A. Casanova', 'Nathan Goodman']]}
{'_id': 'Nathan Goodman',
    'Titulo': ['Query Processing in a System for Distributed Databases (SDD-1).'],
    'Coautores': [['Philip A. Bernstein', 'Nathan Goodman', 'Eugene Wong',
             'Christopher L. Reeve', 'James B. Rothnie Jr.']]}
(...)
```

(h) 8. Edad de los 5 autores con un periodo de publicaciones más largo (Se considera la Edad de un autor al número de años transcurridos desde la fecha de su primera publicación hasta la última registrada.

```
{'_id': 'Alan M. Turing', 'diferencia': 75}
{'_id': 'David Nelson', 'diferencia': 68}
{'_id': 'Thomas L. Saaty', 'diferencia': 64}
{'_id': 'Eric Weiss', 'diferencia': 64}
{'_id': 'Bernard Widrow', 'diferencia': 64}
```

(i) 9. Número de autores novatos, es decir, que tengan una Edad menor de 5 años. Se considera la Edad de un autor al número de años transcurridos desde la fecha de su primera publicación hasta la última registrada.

```
1543389
```

(j) 10. Porcentaje de publicaciones en revistas con respecto al total de publicaciones.

```
44.958745217597254
```

Parte II: Neo4J (20%)

Dada la naturaleza de los datos, también resulta especialmente sencillo utilizar una base de datos orientada a grafos, tal como Neo4j.

1. **Captura y procesamiento de datos.**

   Debe procesar el fichero XML del apartado anterior para convertirlo a un formato adecuado.

   Para la Parte I, se ha procesado el fichero XML en varios ficheros json. Para evitar el procesado otra vez del volumen de datos de XML, se ha transformado directamente estos json en formato csv que es el aceptado por Neo4j. Ver fichero python transform_csv.py con este paso.

   Estos ficheros csv se han formado usando la información mínima para hacer las consultas, separando la informacion principal en "nodos" para meter en el Neo4j. los ficheros son: 1) csv_autores.csv, que contiene el nombre todos los autores sin repetición; 2) csv_art.cs, que contiene el identificador de la publicación y el titulo. 3) csv_year.csv, que contiene los años donde hay publicaciones y 4) csv_pub.csv, que contiene las publicaciones con toda la información, esto es, identificación, el titulo, el autor y el año.

   La distribución en distintos fichero además de aligerar la carga de procesamiento, ayuda a la creación de los nodos y las relaciones de interés de manera más eficaz.

   Debido a que hemos tenido muchos problemas por la falta de capacidad de nuestros equipos (poca memoria) y la falta de posibilidad de usar otros, hemos simplificado todo lo que hemos podido sin sacrificar el tamaño de los datos, es decir, número de publicaciones.

2. **Almacenamiento de datos.**

   Una vez convertidos al formato necesario (y adecuadamente pre-procesados como se prefiera), los datos deben almacenarse en una base de datos Neo4J. En la memoria debe justificar el diseño de la base de datos orientada a grafos que haya elegido. Comente brevemente las diferencias que encuentren entre ambos sistemas (MongoDB y Neo4J)

   La práctica propone un juego de datos bibliográficos que se ajustan bien a consultas donde las relaciones se establecen entre autores y publicaciones. Los datos tienen cinco cosas en común: autores (publicaciones con algún autor en común), año (para un mismo año se tiene un número determinado de publicaciones), revista (cada revista tiene un número de publicaciones), volumen (dentro de la misma revista si es el caso) y tipo de publicación (artículos, libros, proceedings, etc). Las demás características son identificativas de cada publicación (titulo, clave, páginas en el volumen, etc). Para optimizar las consultas sería apropiado crear un fichero para cada característica común, por lo menos autores, año, revista y tipo, pero para simplificar el ejecicio y usar menos recursos en un tiempo adecuado, nos hemos quedado solo con autores, publicaciones y año, que resolverían las consultas más comunes y generales.

   Por lo tanto hemos creado una lista de autores, una lista de año de publicación, una lista de artículos y una lista de publicaciones con las propiedades comunes. Con los tres primeros generamos los NODOS y el cuarto nos ayuda a establecer las relaciones.

   Almacenamos y creamos los NODOS de las variables de los tres ficheros en una base de datos de neo4j:

   - Años:

     ```
     USING PERIODIC COMMIT
     LOAD CSV WITH HEADERS FROM 'file:///csv_years.csv' AS row
     CREATE (ye:Years { year: toInteger(row.Year)})
     ```

   - Autores:

```
USING PERIODIC COMMIT
LOAD CSV  WITH HEADERS FROM 'file:///csv_autores.csv' AS row
CREATE (au:Autores { id_name: row.Autor})
```

- Artículos:

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'file:///csv_art.csv' AS row
CREATE (pu:Publicaciones { id_p: row.Key, title: row.Titulo})
```

- Creamos índices para que el procesamiento sea más eficiente, en este caso sobre las variables de interés:

```
CREATE INDEX ON :Autores(id_name)

CREATE INDEX ON :Years(year)

CREATE INDEX ON :Publicaciones(id_p)
```

Establecemos las relaciones:

- Relación entre los autores y sus publicaciones:

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///csv_pub.csv" AS row
MATCH (pu:Publicaciones { id_p: row.Key, title: row.Titulo})
MATCH (au:Autores { id_name: row.Autor})
MERGE (au)-[WR:WROTE]->(pu)
```

- Relación entre las publicaciones y el año de publicación:

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///csv_pub.csv" AS row
MATCH (pu:Publicaciones { id_p: row.Key, title: row.Titulo})
MATCH (ye:Years { year: toInteger(row.Year)})
MERGE (pu)-[in:PUBIN]->(ye)
```

- Relación creada entre autores, publicaciones y año de publicación (ver figura 1):

```
MATCH p=()-[r:WROTE]->()-[q:PUBIN]->() RETURN p LIMIT 50
```

En esta fase, y debido a problemas en la carga de los ficheros en la base de datos de Neo4j via navegador, usamos también la herramienta *neo4j-import*, en particular para introducir los ficheros de datos en la base de datos:

```
./bin/neo4j-import --into ./data/databases/graph.db  --nodes:Pub ./import/csv_art.csv
 --nodes:Autores ./import/csv_autores.csv  --nodes:Years ./import/csv_years.csv
 --relationships:WROTE ./import/csv_rel.csv
```

Aunque finalmente usamos el navegador.

Las diferencias entre MongoDB y Neo4j se comentan en la siguiente sección.

3. **Análisis de datos.**

   Proponga e implemente al menos 3 consultas para las que claramente Neo4j sea más apropiado que MongoDB. Defina los índices que considere necesario para optimizar las consultas definidas.

   Con los índices definidos del apartado anterior sugerimos 3 consultas (ver también figura 2.):

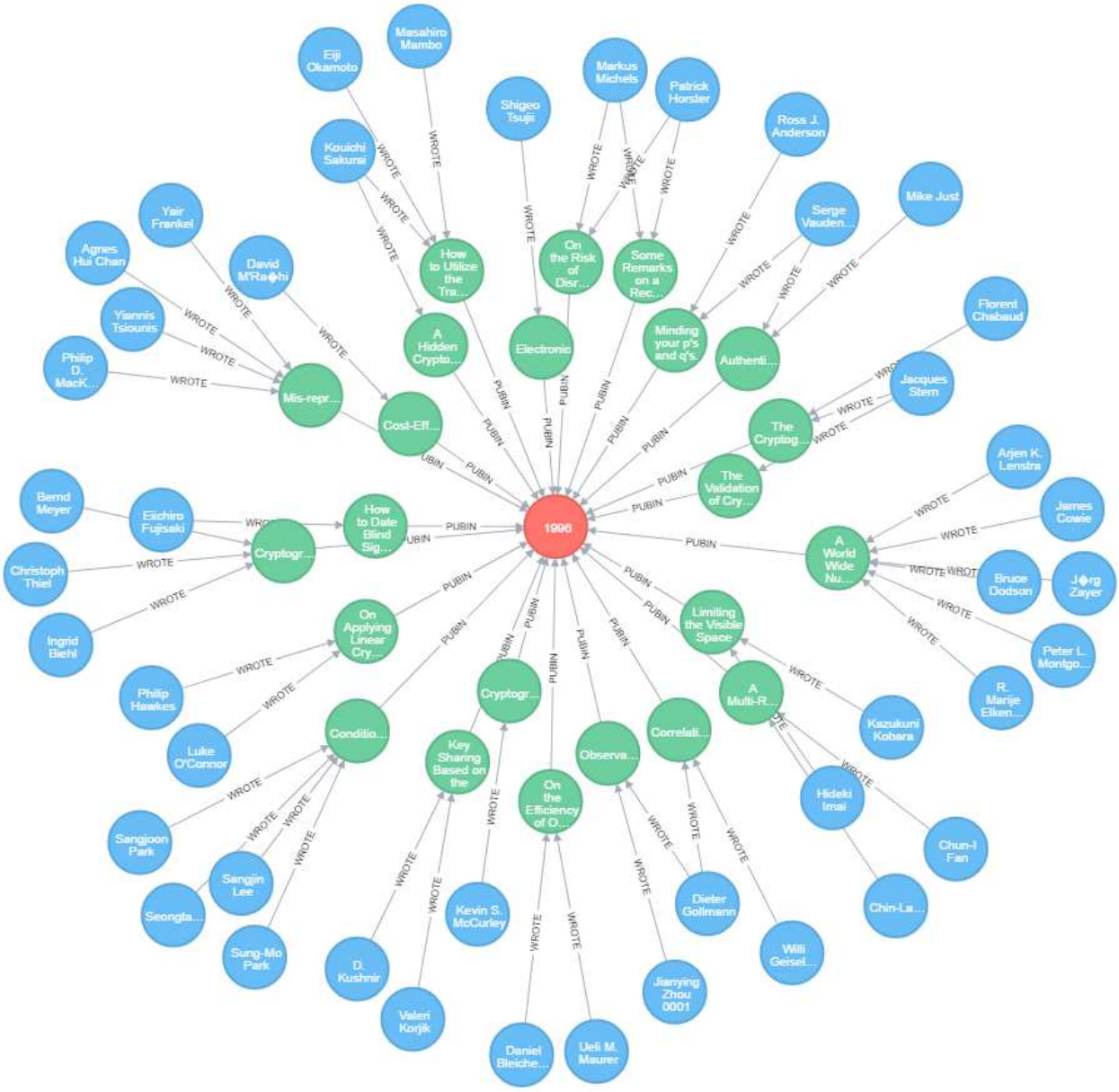

Figure 1: Ejemplo de la relación creada entre autores, publicaciones y año de publicación.

- Publicaciones de un autor: "Norbert Blum":

  ```
  MATCH p=(au:Autores {id_name:"Norbert Blum"})-[r:WROTE]->() RETURN p
  ```

  Número:

  ```
  MATCH p=(au:Autores {id_name:"Norbert Blum"})-[r:WROTE]->() RETURN COUNT (p)
  29
  ```

- Coautores de un autor: "Norbert Blum":

  ```
  MATCH p=(au:Autores {id_name:"Norbert Blum"})-[r:WROTE]->()<-[s:WROTE]-() RETURN p
  ```

- Autores en 2017:

  ```
  MATCH p=(ye:Years {year:1983})<-[r:PUBIN]-()<-[s:WROTE]-() RETURN p LIMIT 50
  ```

  Número:

  ```
  MATCH p=(ye:Years {year:1983})<-[r:PUBIN]-()<-[s:WROTE]-() RETURN COUNT (p)
  971064
  ```

Las Diferencias encontradas entre MongoDB y Neo4j:

Las tres consultas enlazan fácilmente distintas colecciones de datos. En particular, la consulta de buscar el número de coautores en MongoDB es mucho más complicada, teniendo que separar y enlazar varios campos, mientras que en Neo4j es una linea sencilla de código, dentro de la forma en que hemos estructurado las colecciones. Parece que Neo4j tiene la ventaja de simplificar las búsquedas respecto a MongoBD gracias a las conexiones, pero para conseguir esto se ha tenido que separar la información en un número de ficheros mayor que las colecciones de MongoDB, lo que además del trabajo en hacer las colecciones, ocupa más espacio. Por ejemplo en MongoDB nuestra base de datos ocupa 1.11 Gigas mientras que en Neo4j ocupa 7.2, que es considerablemente mayor.

En este caso, el juego de datos permite una relación tipo grafo sencilla entre algunas variables, cuanto mas separables sean las variables en nodos, más sencillo resultara hacer las consultas en Neo4j, además tiene la ventaja de poder visualizar las relaciones. En cambio, no todos los datos serán igual de adaptables a un grafo. MongoDB es más flexible en cuanto a tipo de datos y volumen, teniendo más capacidad de almacenamieto y procesado, es más escalable, se puede añadir informacion sin necesidad de buscar una relacion, haciendolo más apropiado para datos masivos. Además MongoDB accede más rápido a la información sin necesidad de crear ficheros/colecciones tipo NODO, y se pueden crear mas atributos (documentos embebidos etc), etc.

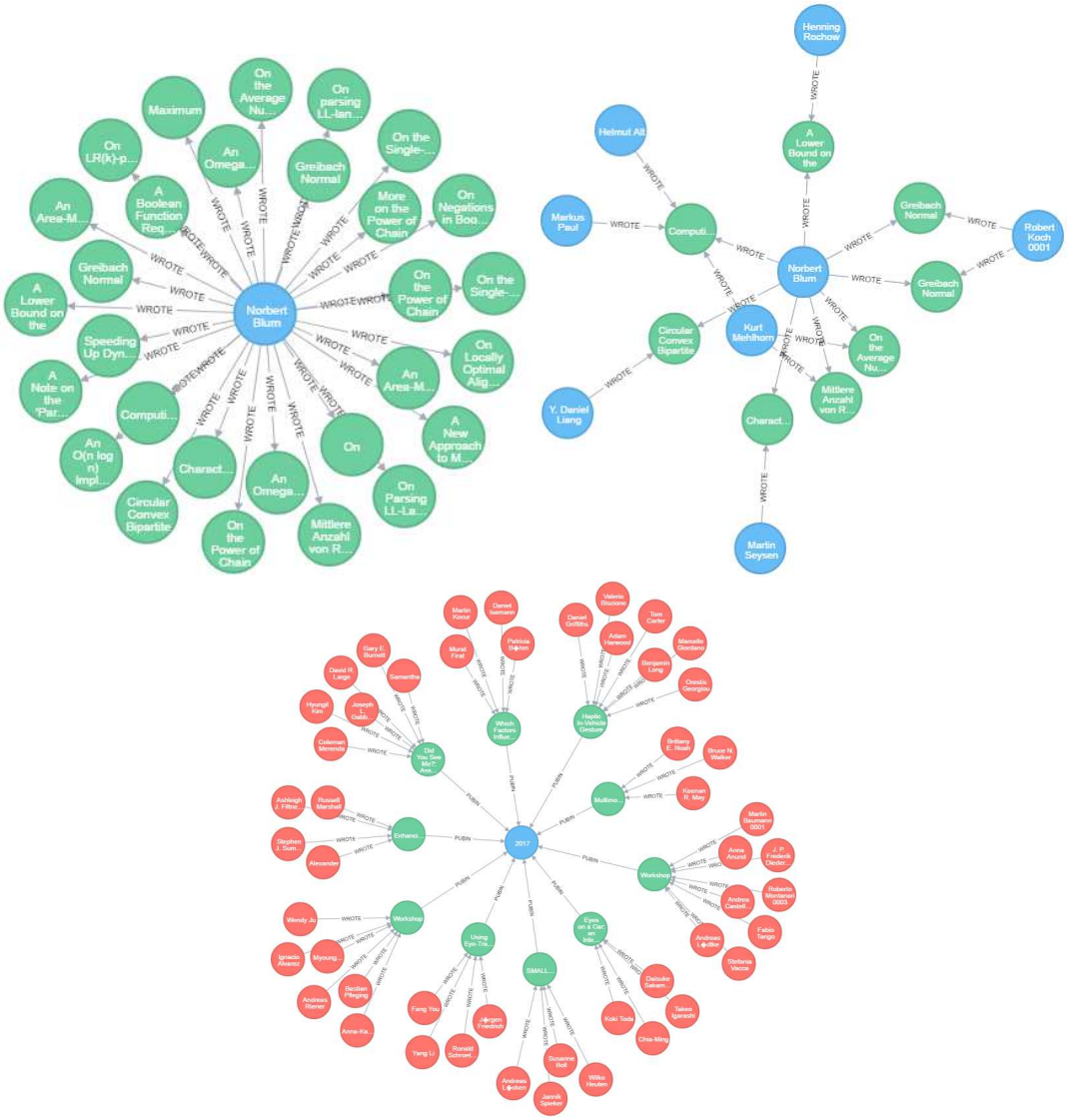

Figure 2: Gráfos de las consultas sugeridas.