BASES DE DATOS NO CONVENCIONALES

Máster en Data Science|URJC

Javier Pérez Clemente y Belén González Guío

07/04/2019

Tabla de contenido

INTRODUCCIÓN

La fuente de datos que se va a utilizar en esta práctica es una fuente de datos real que recopila **referencias bibliográficas académicas**, concretamente relacionadas con la **informática**. Esta fuente de datos es conocida como **DBPL** *Computer Science Bibliography*. Los datos que vamos a procesar y, más tarde, almacenar para que puedan ser consultados, han sido descargados manualmente del siguiente link: https://dblp.uni-trier.de/xml/.

La información relacionada con cada artículo que podemos encontrar en este fichero es título del artículo, autor/autores, páginas, fecha de publicación, etc. Los datos recogen **8 tipos de artículos**, pero en nuestro caso vamos a **trabajar con 3** de ellos: **artículos de revista** (*article*), **artículos en congresos** (*inproceedings*) y **artículos en libros** (*incollection*).

PARTE I : MONGODB

En esta primera parte de la práctica se va a trabajar con la **base de datos no relacional MongoDB**, que es de **tipo documental**. Este tipo de bases de datos se caracteriza en que almacenan los **valores como documentos** (normalmente en formato JSON o XML). De esta manera, las bases de datos no relacionales documentales almacenan todos los atributos de una entidad en un único documento, en vez de almacenar cada atributo de la entidad con una clave separada. Además, existe una gran **flexibilidad en la estructura de los documentos**, a diferencia de las bases de datos relacionales.

1 .- CAPTURA Y PROCESAMIENTO DE DATOS

Como se mencionó antes, la captura de los datos se ha hecho descargando manualmente del link de la página web de dpbl. Se han descargado **dos archivos**:

- un zip que contiene las **publicaciones** en formato **XML**
- un **fichero** de tipo **dtd**

El archivo dtd contiene la definición del tipo de documento y describe el vocabulario que identifica los elementos y las entidades de los que consta el XML. Es decir, este archivo auxiliar expresa la estructura y es necesario porque permite que el programa de Python que parsea los documentos (que explicaremos más tarde) sea mucho más

rápido. El fichero XML comprimido ocupa alrededor de 450 MB y descomprimido pesa acerca de **2 GB**.

2.- DEFINICIÓN DEL ESQUEMA DE LA BBDD

Antes de procesar todos los datos, analizamos cuáles eran las consultas más frecuentes para organizar el esquema en función de estas consultas. De esta manera evitamos procesar datos o atributos que no sean necesarios y que no se vayan a usar.

Lo primero ha sido **especificar** qué ***tags*** queremos procesar y cuáles no son de nuestro interés puesto que, como se comentó, aunque hay 8 tipos de artículos, solo estamos interesados en 3: ***article, inproceedings*** *e* ***incollection***. Además, antes de procesar los datos, se han analizado las consultas para evitar procesar atributos que no se van a usar o que no van a ser consultados y para poder definir un esquema. Se han seleccionado finalmente los siguientes 5 **atributos**: **'id', 'type', 'date', 'title'** y **'authors'.**

Una vez que hemos seleccionado los campos que vamos a almacenar, se ha definido un esquema mirando las consultas que se tienen que realizar para poder entender cómo se usarán los datos. Si bien una base de datos documental es flexible y no exige que la estructura de todos los documentos sea idéntica, es importante definir un esquema porque la estructura de los documentos, aunque no sea idéntica, tienen que tener una estructura común. Para ello hay que definir cuántas colecciones vamos a definir en esta base de datos de MongoDB. Normalmente, se eligen el número de colecciones en función de cuántos tipos de documentos difieren en su estructura. En nuestro caso, como tenemos tres posibles tipos de documentos (*articles, inproceedings, incollection*) pero tienen una estructura muy similar, hemos decidido crear **una sola colección**. A esta colección la hemos llamado ***publications*** y contiene 5 campos:

- **_id**: Identificador único auto incremental para cada publicación presenta la colección.
- **type**: Tipo de publicación. Puede tomar 3 valores: *article*, *inproceedings* e *incollections*.
- **date**: Fecha de publicación en formato %Y-%m-%d
- **authors**: A*rray* que recoge los autores que han participado en la publicación.
- **title**: título de la publicación

A continuación, se muestra un ejemplo de dos documentos insertados en la base de datos para visualizar la estructura que tienen los jsons:

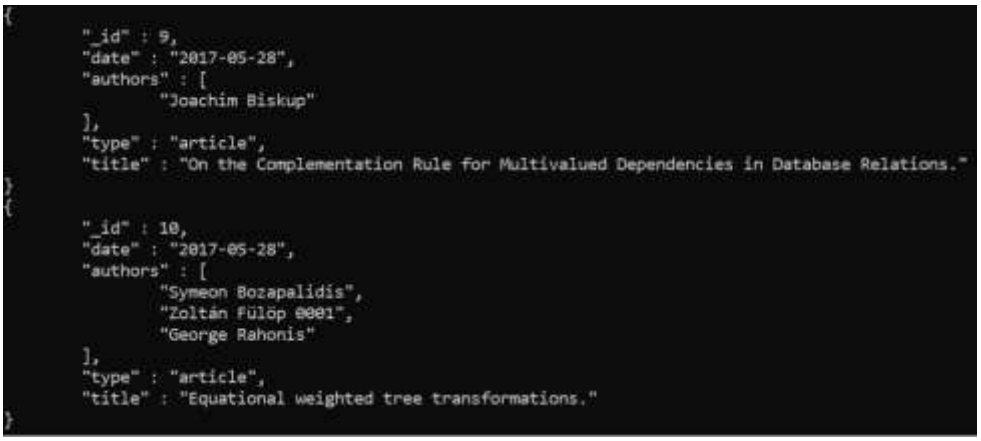


Ilustración 1 Ejemplo documentos insertados en MongoDB

3.- PROCESAMIENTO DEL FICHERO DE DATOS

El archivo que parsea el XML e inserta en una base de datos de MongoDB se llama **xml_to_mongo.py**, y utiliza para ello dos librerías: **etree** y **MongoClient**. Este código consta de **dos funciones**:

- ***get_dict***: recibe el elemento del árbol XML y el identificador auto incremental y construye un diccionario con los atributos anteriormente definidos.
- ***write***: conecta con la base de datos Mongo y realiza una inserción múltiple de la lista de documentos que recibe.

La primera estrategia probada consistía en utilizar la función *parse()* librería *xmltodict* para convertir el fichero xml a diccionario. Sin embargo, esto resultó inviable dado el tamaño del mismo, así que tuvimos que pensar en opciones alternativas. Finalmente, utilizamos la función *iterparse* de la clase *etree* (librería lxml), que permite procesar el xml elemento a elemento ayudándose del archivo "dblp.dtd". Para cada elemento, llamamos a la función *get_dict* para obtener el correspondiente diccionario y lo almacenamos en la lista "result".

4.- ALMACENAMIENTO DE DATOS

Una vez contamos con todas las publicaciones recogidas en la lista "*result*", llamamos a la función *write()* para escribir los documentos en la colección "publications" de nuestra base de datos mongo.

Debido al elevado número de publicaciones, nos vimos obligados a realizar escrituras parciales cada vez que la longitud de la lista supera un límite establecido en 500.000 elementos.

El proceso completo de parseo y almacenamiento tarda en ejecutarse en torno a 3 minutos.

5.- ANÁLISIS DE DATOS

Una vez que ya se han procesado todos los datos y se han almacenado en Mongo, la base de datos ya está lista para ser consultada. Se han realizado las 10 consultas propuestas y se han recogido en un archivo Python que conecta con la base de datos, realiza las consultas y recibe los resultados de éstas. Este archivo se llama **queries_mongodb.py**. A parte también muestra el tiempo que tarda la *query*. Este tiempo se va a comparar antes de insertar índices en la colección y después, para ver cómo realmente los índices aceleran las consultas.

1. Listado de publicaciones de un autor determinado. En nuestro caso, el autor es *Joachim Biskup*.

```
db.publications.aggregate({$unwind : "$authors"},
                          { $match : { authors : "Joachim Biskup" }},
                          {$project : { "title" : 1 }})
```

   Con esta consulta especificamos con *unwind* que queremos que convierta cada documento con *array* en tantos documentos como elementos tenga el *array*. Después filtramos los registros cuyo autor es *Joachim Biskup* y pedimos que nos devuelta el título.

```
The time taken, in seconds, for query number 1 is  9.127638101577759
El listado de las publicaciones de Joachim Biskup es:
         _id                                                title
0          9  On the Complementation Rule for Multivalued De...
1        458  Optimization of a Subclass of Conjunctive Quer...
2        973  Adding inclusion dependencies to an object-ori...
3      11641  Sync Classes: A Framework for Optimal Scheduli...
4      92474                                On BI-Immune Isols.
5     195484                             Editorial ESORICS 2007.
6     306367  On Inference-Proof View Processing of XML Docu...
7     365026  A Foundation of Codd's Relational Maybe-Operat...
8     508996  An Extension of SQL for Querying Graph Relations.
9     515820    Boyce-Codd Normal Form and Object Normal Forms.
10    517071  Some Variants of the Take-Grant Protection Model.
11    518666  Reducing inference control to access control f...
12    541449  Controlled query evaluation for enforcing conf...
13    541550           Keeping secrets in incomplete databases.
14    544112  Extracting information from heterogeneous info...
15    544302  Inference control of open relational queries u...
16    554819  A Trust- and Property-based Access Control Model.
17    762093  Lying versus refusal for known potential secrets.
18    763142  For unknown secrecies refusal is better than 1...
19    801468  Design of Relational Database Schemes by Delet...
20   1072683  Confidentiality enforcement by hybrid control ...
21   1100894        Solving equations in the relational algebra
22   1305637  Objects in Relational Database Schemes with Fu...
23   1399459  Controlled Query Evaluation for Known Policies...
24   1399506  Preserving confidentiality while reacting on i...
25   1400005  Controlled query evaluation with open queries ...
26   1400211  Appropriate inferences of data dependencies in...
27   1400352  Decomposition of object-oriented database sche...
28   1408080  The personal model of data: Towards a privacy-...
29   1469491  Secure Mediation: Requirements, Design, and Ar...
..       ...                                                ...
133  3952714  Das datenschutzorientierte Informationssystem ...
```

2. Número de publicaciones de un autor determinado. En nuestro caso es, una vez más, el autor *Joachim Biskup*.

```
db.publications.aggregate({$unwind : "$authors"},
                          { $match : { "authors" : "Joachim Biskup" }},
                          {$project : { "title" : 1 }},
                          {$count: "title"})
```

Esta consulta, solo en este caso, es equivalente a la siguiente:

```
db.publications.count({"authors": {$eq:"Joachim Biskup"}}, {"title":1})
```

La razón es que ambas consultas nos dan el mismo número de publicaciones (probando con y sin *unwind*) porque da la casualidad de que el autor escogido tiene publicaciones donde solo él es el autor y, por lo tanto, no necesita hacer *unwind* para sacar el total de publicaciones que tiene. En casos en los que un autor colabore con otros, sí que es necesario usar *unwind* porque si no estaríamos cogiendo solo las publicaciones donde él es el autor único.

```
The time taken, in seconds, for query number 2 is  11.727659463882446
El numero de publicaciones de Joachim Biskup es
   title
0    163
```

3. Número de artículos en revista para el año 2017

```
db.publications.count({$and: [{"date" : {$regex: "2017"}}, {"type" :"article"}]})
```

En este caso no hace falta hacer uso de *aggregate*, sino que vale con *find* puesto que ponemos las dos condiciones en un único elemento del pipeline. Se usa *regex* para *matchear* una expresión regular, aunque en este caso no es una expresión regular como tal, sino que es un substring del campo date.

```
The time taken, in seconds, for query number 3 is  2.297888994216919
El numero de articulos en revista para el anyo 2017 es  1050819
```

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

```
db.publications.aggregate([{"$unwind": "$authors"},
                           {"$sortByCount":"$authors"},
                           { $match: {"count": { $lt:5}}}, {$count: "authors"}]
                           , { allowDiskUse: true })
```

En este caso se usa *sortByCount* que ordena y cuenta el campo que le indiques.

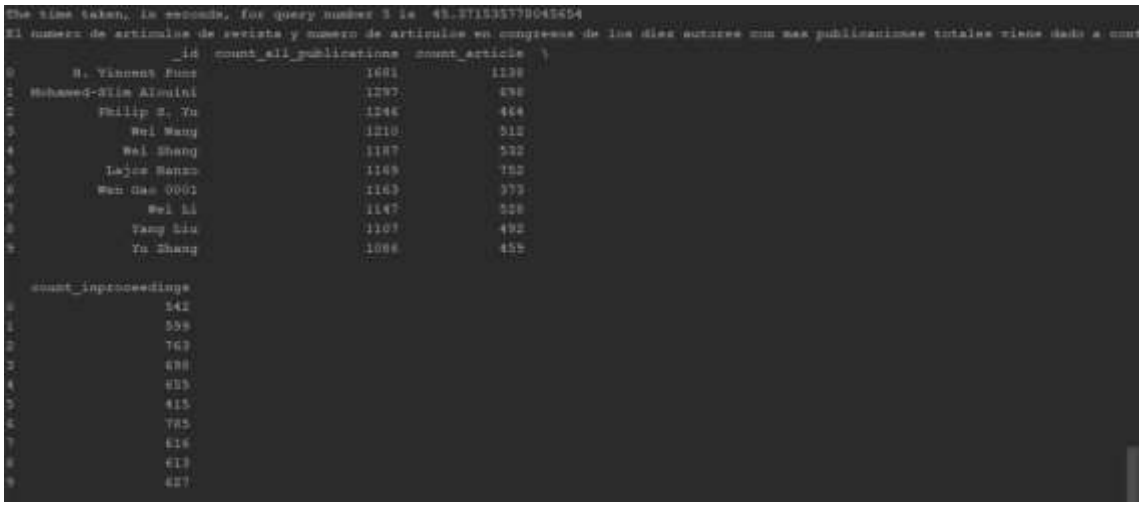
Cabe mencionar que hemos necesitado añadir la variable *allowDiskUse: true* específicamente para que no se quede sin memoria al hacer la consulta. Esto nos pasará más adelante con otras consultas.

```
The time taken, in seconds, for query number 4 is  35.30126452445984
El numero de autores con menos de 5 publicaciones es
   authors
0  1767880
```

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

```
db.publications.aggregate([{$unwind: "$authors"},
                           {$group:{_id: "$authors",count_all_publications: {$sum:1},
                                    count_article: {$sum : {$cond : { if: { $eq: ["$type", "article"]}, then: 1, else: 0}}},
                                    count_inproceedings: {$sum : {$cond : { if: { $eq: ["$type", "inproceedings"]}, then: 1, else: 0}}}}},
                           {$sort: {count_all_publications: -1}},
                           {$limit: 10}],
                           {allowDiskUse: true})
```

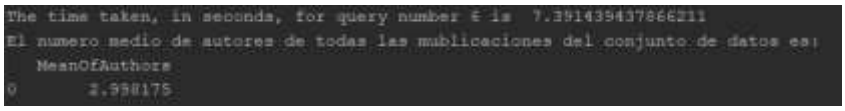
En este caso no hemos podido usar *sortByCount* como en la anterior consulta porque este operador solo te devuelve dos columnas y después quieres hacer más operaciones no te lo permite. Por ello, se ha usado *group* y luego *sort*.



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

```
db.publications.aggregate([{$project: {"numAuthors": { $size: "$authors" }}},
                           {'$group':{
                                      _id: null,
                                     'MeanOfAuthors': {'$avg': '$numAuthors'}
                                    }
                           }] )
```

Para responder esta consulta, se ha creado una variable que calcule el tamaño del array que contiene al autor o autores para cada publicación y después se ha calculado la media, obteniendo así que, en promedio, las publicaciones tienen casi 3 autores.



7. Listado de coautores de un autor (se denomina coautor a cualquier persona que haya firmado una publicación). En nuestro caso es, una vez más, el autor *Joachim Biskup*.

```
db.publications.aggregate([{$project: {'authors': 1}},
                           {$match: {authors: 'Joachim Biskup'}},
                           {$unwind : '$authors' },
                           {$group: {_id:"$authors", coauthors: { $addToSet: {$cond : { if: { $eq: ["$authors", "Joachim Biskup"]},
                                                                                       then: null, else: "$authors"}}}},
                           {$project : {'_id':0,
                                        "coauthors": { $cond: {if: {$eq: [ [null], "$coauthors" ] },
                                                               then: "$$REMOVE",else: "$coauthors"}}}}])
```

Aquí filtramos los documentos por aquellos donde aparece *Joachim Biskup,* y después se hace *unwind* y devolvemos aquellos que no son *Joachim Biskup*.

```
The time taken, in seconds, for query number 7 is  5.027588367462158
El listado de los coautores de Joachim Biskup es:
                  coauthors
0      [Patrick Krumpelmann]
1           [Walied Othman]
2            [Chris Clifton]
3           [Erik Buchmann]
4           [Milyun Basalp]
5            [Shawn Merrill]
6        [Sebastian Sonntag]
7          [Marvin Schaefer]
8         [Carl E. Landwehr]
9        [Klaus R. Dittrich]
10     [Matthew Morgenstern]
11       [Thomas Leineweber]
12        [Hans-Werner Graf]
13        [Christian Eckert]
14    [Vijayalakshmi Atluri]
15           [Raban Serban]
16         [Gerrit Bleumer]
17           [Thomas Noll]
18    [Hans-Martin Bluthgen]
19           [Sven Hartmann]
20           [L. Schnetgöke]
21          [Norbert Sensen]
22          [Barbara Sprick]
23      [Torsten Schlotmann]
24            [Martin Apel]
25          [Michael Vogel]
26    [Reind P. van de Riet]
27      [Michael Meier 0001]
28            [Erkay Savas]
29            [Alois Knoll]
..                      ...
```

8. Edad de los 5 autores con un periodo de publicación 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)

```
db.publications.aggregate([{$unwind : '$authors' },
                           {$group : {_id:"$authors",
                                      max_year : {$max : {$substr: [ "$date", 0, 4 ]}},
                                      min_year :{$min : {$substr: [ "$date", 0, 4 ]}}},
                            {$addFields: {max_year_int: {$toInt: "$max_year"},
                                          min_year_int: {$toInt: "$min_year"}}},
                            {$project : {max_year:0, min_year:0}},
                            {$addFields: {ageAuthor: {$subtract: ["$max_year_int","$min_year_int"]}}},
                            {$sort: {ageAuthor: -1}},
                            {$limit: 5}],
                            { allowDiskUse: true })
```

Agrupamos por autor y construimos dos variables (una que hace referencia al año máximo del autor y otra al año mínimo del autor) utilizando *substr* para extraer del campo *date* los 4 primeros dígitos como *string*. Después, creamos otras dos variables que son resultado de convertir las dos anteriores de *string* a *integer*. Por último, se calcula la diferencia de estas dos variables (que es lo que consideramos como edad del autor), se ordenan de manera decreciente y limitamos a 5 los resultados.

```
The time taken, in seconds, for query number 8 is  60.95050229263306
La edad de los 5 autores con un periodo de publicacion mas largo es:
                _id  ageAuthor  max_year_int  min_year_int
0          A. Martina         17          2019          2002
1       A. M. Chemali         17          2019          2002
2      A. P. Wim Bohm         17          2019          2002
3   A. Nico Habermann         17          2019          2002
4  A. Jefferson Offutt         17          2019          2002
```

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.

```
db.publications.aggregate([{$unwind : '$authors' },
                           {$group : {_id:"$authors",
                                      max_year : {$max : {$substr: [ "$date", 0, 4 ]}},
                                      min_year :{$min : {$substr: [ "$date", 0, 4 ]}}},
                           {$addFields: {max_year_int: {$toInt: "$max_year"},
                                         min_year_int: {$toInt: "$min_year"}}},
                           {$project : {max_year:0, min_year:0}},
                           {$addFields: {ageAuthor: {$subtract: ["$max_year_int","$min_year_int"]}}},
                           { $match: { ageAuthor: { $lt: 5} }},
                           {$count: "ageAuthor"}],
                           { allowDiskUse: true })
```

Calculamos la edad de los autores de la misma manera que hicimos en la anterior consulta y después filtramos para quedarnos con aquellos registros cuya edad de autores es menor que 5 y hacemos un *count*.

```
The time taken, in seconds, for query number 9 is  44.6825737953186
El numero de autores novatos es el siguiente:
   ageAuthor
0    2063192
```

10. Porcentaje de publicaciones en revistas con respecto al total de publicaciones.

```
db.publications.aggregate([{$project: {'type': 1}},
                           {$group:{_id:null,
                                    count_article: {$sum: { $cond :  [{ $eq : ["$type", "article"]}, 1, 0]}},
                                    count_total : {$sum:1}}},
                           {$project: { article_percentage :{$multiply: [100,{ $divide: [ "$count_article", "$count_total"]}]}}}])
```

Para obtener el porcentaje de publicaciones en revistas (*articles*) con respecto al total de publicaciones, se ha construido una variable llamada *count_article* que es binaria. Toma valor 1 cuando el *type* es igual a *article* y el valor 0 cuando no. También construimos otra variable que sume el total de registros. Por último dividimos el número de publicaciones en revistas entre el total y multiplicamos por 100.

```
The time taken, in seconds, for query number 10 is  4.651588678359985
El porcentaje de publicaciones en revistas (articles) con respecto al total de publicaciones es:
   article_percentage
0           45.213249
```

Después de hacer estas consultas, se han insertado una serie de **índices** para ver si mejora el tiempo de ejecución de estas consultas. La creación de índices se encuentra en el archivo **xml_to_mongo.py.**

Los índices creados son los siguientes:

- Índices simple sobre el campo autor
- Índice simple sobre el campo fecha
- Índice compuesto sobre los campos autor y fecha

A continuación se muestra una tabla **comparativa** de los **tiempos** de ejecución de las consultas desde Python **con y sin índices**.

Question	Execution time without indexes (in seconds)	Execution time with indexes (in seconds)
1	9.11	8.43
2	11.70	11.27
3	2.08	20.13
4	31.63	31.88
5	39.98	40.30
6	8.24	7.56
7	2.58	2.47
8	37.29	37.75
9	37.59	39.37
10	4.59	4.80

Como puede verse, no hay una mejora muy apreciable. De hecho, el tiempo de la tercera consulta a aumentado drásticamente, lo que se debe a que la elección de índices no ha resultado en una gran ayuda en este caso.

PARTE II: NEO4J

Esta segunda parte de la práctica va a utilizar los **mismos datos** pero, a diferencia de la parte I, la base de datos que se va a usar es una **base de datos orientada a grafos (Neo4J)** en vez de una base de datos orientada a documentos (MongoDB). Las bases de datos orientadas a grafos representan la información como **nodos** que se relacionan mediante **aristas**. Cada nodo tiene un identificador y puede tener varios atributos que describen este nodo. Uno de los puntos fuertes de estos tipos de bases de datos es que su rendimiento permanece constante aun cuando crece el conjunto de datos. Además, son bastante flexibles, lo cual se ajusta perfectamente con los entornos empresariales cambiantes que van de la mano de las metodologías ágiles.

Por lo general, se usarán este tipo de grafos cuando queramos modelar relaciones.

1 .- CAPTURA Y PROCESAMIENTO DE DATOS

Para capturar los datos, procesarlos y después insertarlos, inicialmente se planteó la opción de exportar los datos de MongoDB en formato csv. Esta opción se descartó porque el comando de exportación en formato csv de mongo, añadía comillas dentro del array de autores que complicaba el proceso. Además, como el formato en el que se insertan los datos en Neo4J es diferente de Mongo (insertamos por un lado los nodos y por otro lado las relaciones), se construyó un parseador que generase **3 ficheros csv**:

- **2** ficheros con dos **tipos de nodos: autor_nodes.csv** y **publication_nodes.csv**
- **1** fichero que contiene las **relaciones** entre estos dos tipos de nodos: **relationships.csv**

El archivo Python que crea estos tres documentos se llama **csv_parser.py**. Este código recorre el XML haciendo uso de ***iterparse***, como se hacía en la parte 1.

En total se generan **6.699.232 nodos**. Como comentábamos, un archivo recoge los **nodos de tipo autor** y el otro recoger los **nodos de tipo publicación**, que a su vez tiene tres tipos: *article*, *inproceedings* e *incollections*.

En cuanto a las **relaciones**, se han generado **13.253.583** de **un solo tipo**: **publicated**.
(Puede haber varios tipo de relaciones entre nodos, pero no es nuestro caso)

2.- ALMACENAMIENTO DE DATOS

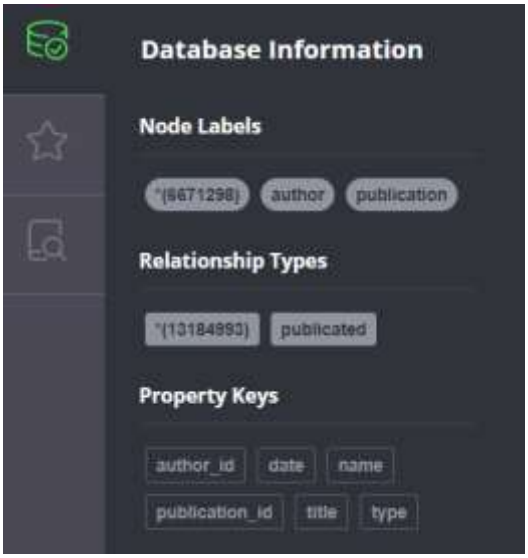
Como los datos ya han sido parseados y se han recogido en 3 archivos en formato CSV, para almacenar los datos en Neo4J se ha ejecutado el siguiente comando en la terminal:

```
neo4j-admin import
        --database practica.db
        --nodes:author "../import/author_nodes.csv"
        --nodes:publication "../import/publication_nodes.csv"
        --relationships:publicated "../import/relationships.csv"
```

Es importante que los ficheros **csv** estén en el **directorio** de **Neo4J**, en la **carpeta import**.

Cabe mencionar también, que con el fin de cambiar de base de datos que se visualiza en el browser, hay que modificar el archivo **neo4j.conf** y des comentar la línea donde aparece la variable **dbms.active_database** e igualarla al nombre de la base de datos que queramos.

A continuación se muestra la información de la base de datos creada:



Como vemos, la diferencia entre Mongo y Neo4J reside en la manera de almacenar la información. Esto repercute en las consultas, el esquema, etc. Mientras que MongoDB guarda cada registro como si fuera un documento en formato json, Neo4J lo almacena como nodos que tienen atributos y se relacionan entre ellos.

3.- ANÁLISIS DE DATOS

Las consultas más apropiadas para una base de datos orientada a grafos son aquellas que implican la “navegación” por relaciones. Es decir, las consultas basadas en relaciones.

A continuación se muestran las consultas que se han realizado en Neo4J:

Diez autores con más colaboraciones:

Diez autores que ha participado en más publicaciones:

Número de autores distintos

Se ha intentado mejorar los tiempos de recuperación de la información añadiendo un índice sobre el nombre de los autores, pero no hemos encontrado mejoras apreciables.

Las tres consultas han sido elegidas centrándonos en los autores, ya que en Neo4j son elementos de primer nivel (nodos), mientras que en Mongo forman un atributo del documento. Por este motivo, una consulta tan simple como contar el número de autores distintos presentes en la base de datos es mucho más costosa en Mongo que en Neo4j.

Además, las bases de datos orientadas a grafos permiten navegar entre relaciones de manera muy sencilla, lo que sumado a que en esta se elimina el duplicado de autores facilita enormemente la realización de las dos primeras consultas.