Javier Pérez Clemente y Belén González Guío

07/04/2019

Bases de datos no convencionales

Máster en Data Science|URJC

Descripción breve

MongoDB y Neo4J para base de datos DBLP Computer Science Bibliography, considerada como la mayor recopilación existente de referencias bibliográficas  
académicas específicamente centrada en la informática. Almacena los datos relativos a la gran mayoría de las revistas científicas y congresos académicos sobre informática.

Tabla de contenido

[**INTRODUCCIÓN** 2](#_Toc5357542)

[**PARTE I : MONGODB** 2](#_Toc5357543)

[**1 .- CAPTURA Y PROCESAMIENTO DE DATOS** 2](#_Toc5357544)

[**2.- ALMACENAMIENTO DE DATOS** 4](#_Toc5357545)

[**3.- ANÁLISIS DE DATOS** 4](#_Toc5357546)

[**PARTE II: NEO4J** 4](#_Toc5357547)

[**1 .- CAPTURA Y PROCESAMIENTO DE DATOS** 4](#_Toc5357548)

[**2.- ALMACENAMIENTO DE DATOS** 4](#_Toc5357549)

[**3.- ANÁLISIS DE DATOS** 5](#_Toc5357550)

# **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**. Las bases de datos orientadas a documentos se caracterizan porque 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 de los que consta el documento 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**.

Antes de procesar todos los datos, se han mirado cuales eran las consultas más frecuentes para organizar el esquema en función de estas consultas y para no 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 cuantas 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**: id auto incremental para cada documento que se va a insertar en la base de datos
* **type**: tipo de publicación que puede tomar 3 valores: *article*, *inproceedings* e *incollections*.
* **date**: fecha de publicación en formato %Y-%m-%d
* **authors**: *array* de uno o varios elementos 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:

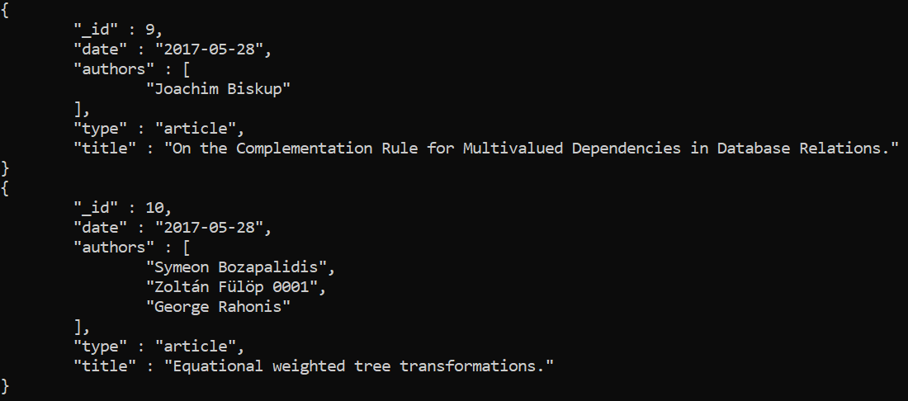


Ilustración 1 Ejemplo documentos insertados en MongoDB

El archivo que parsea el XML e inserta en una base de datos de MongoDB se llama **json\_parser.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 identificar auto incremental y construye un diccionario para cada publicación que acaba formando una lista de diccionarios, donde cada diccionario como se comentaba es una publicación o documento (en términos de la base de datos)
* ***write***: conecta con la el cliente de Mongo y con la base de datos y e inserta cada 500000 publicaciones en la base de datos.

Estas dos funciones se llaman dentro de un bucle que recorre todos los registros del XML. Al ser un archivo tan pesado, no se ha podido cargar directamente con Python. Por lo tanto, para procesar tantos datos se ha hecho uso de ***iterparse***, que construye el árbol, pero te permite eliminar partes del árbol que ya no se necesiten para dejar hueco y procesar el siguiente trozo. En otras palabras, el XML se procesa por trozos para no ir acumulando todos los datos y poder liberar espacio. En el *iterparse*, se ha especificado los *tags* que se quieren procesar (*article*, *inproceedings* e *incollection*).

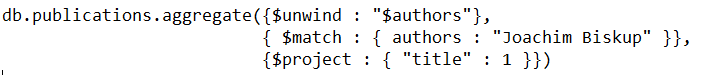
## **2.- ALMACENAMIENTO DE DATOS**

El almacenamiento de datos en Mongo se ha hecho a continuación del parseo para evitar guardar los datos transformados a *json* y volverlos a leer. Por lo tanto, el almacenamiento se realiza al ejecutar el archivo de Python del que hablábamos antes (**json\_parser.py**). De manera resumida, cada vez que se parseaban un número de publicaciones (límite establecido), se insertaban en la base de datos haciendo uso de la función **insert\_many** de **pymongo.** La función *insert\_many* recibe una lista de diccionarios (donde cada diccionario es una publicación o documento) y los inserta en formato *json* en la base de datos.

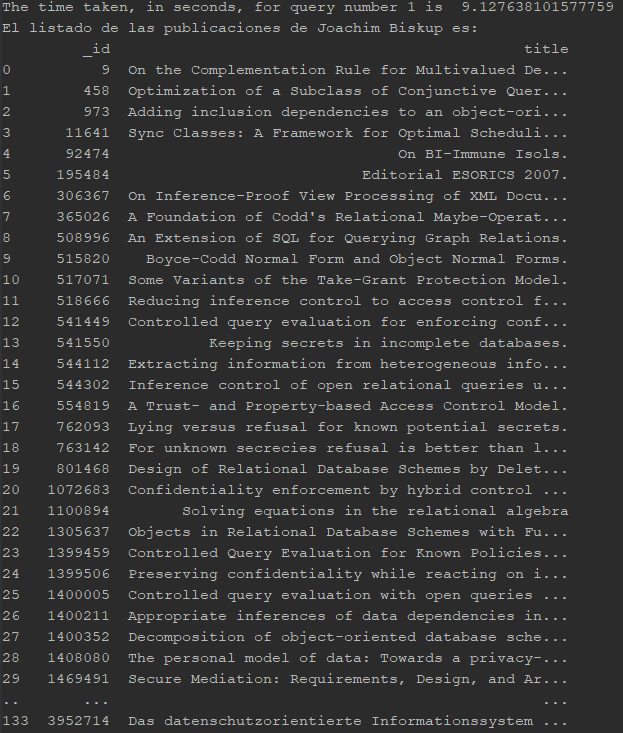
## **3.- ANÁLISIS DE DATOS**

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.

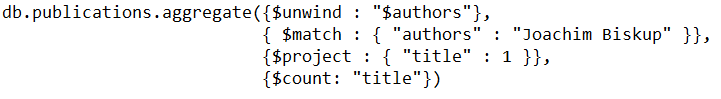
1. Listado de publicaciones de un autor determinado. En nuestro caso, el autor es *Joachim Biskup*.



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.



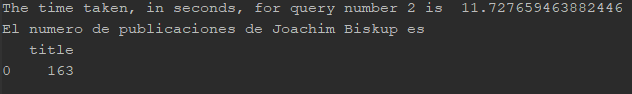
1. Número de publicaciones de un autor determinado. En nuestro caso es, una vez más, el autor *Joachim Biskup*.



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



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.



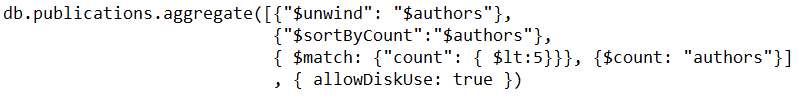
1. Número de artículos en revista para el año 2017



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.

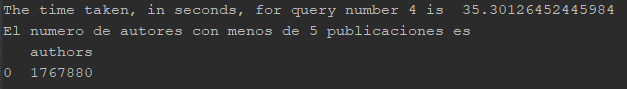
cid:image001.png@01D4EBB0.F5D00850

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

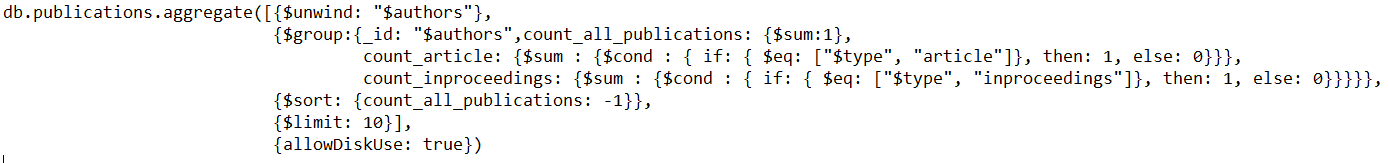


En este caso se usa *sortByCount* que ordena y cuenta el campo que le indiques.

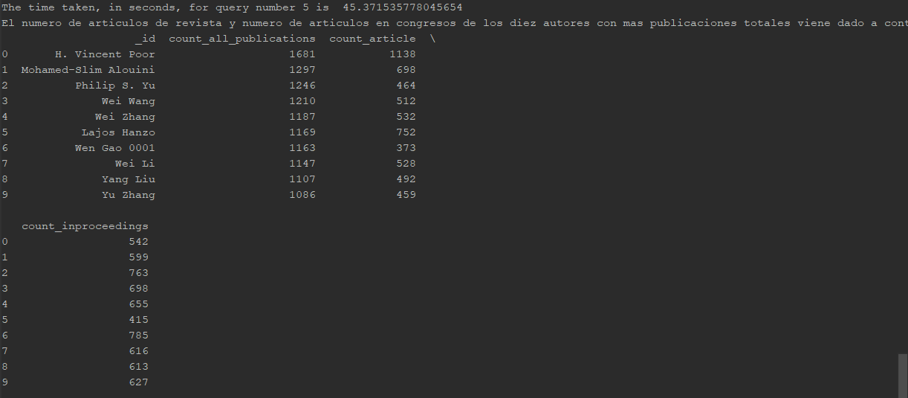
Cabe mencionar que hemos necesitado añadir la variable *allowDiskUse: true* específicamente para que no se quede sin memoria al hacer la consulta. Esto nos pasará más adelante con otras consultas.



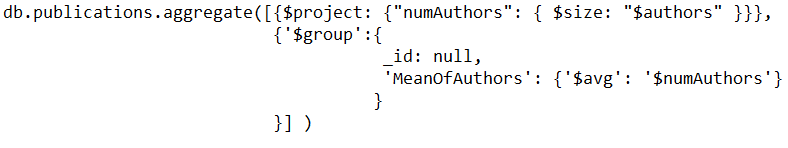
1. Número de artículos de revista (article) y número de artículos en congresos (inproceedings) de los diez autores con más publicaciones totales.

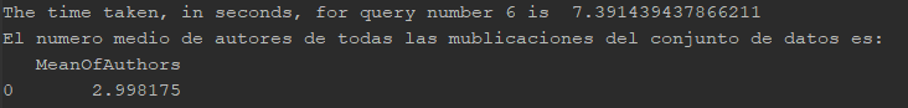


En este caso no hemos podido usar *sortByCount* como en la anterior consulta porque este operador solo te devuelve dos columnas y después quieres hacer más operaciones no te lo permite. Por ello, se ha usado *group* y luego *sort*.

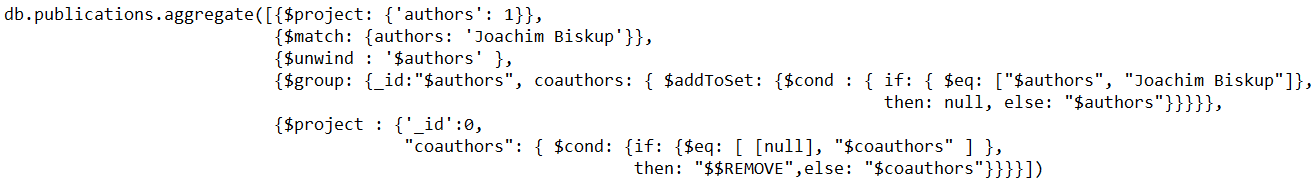


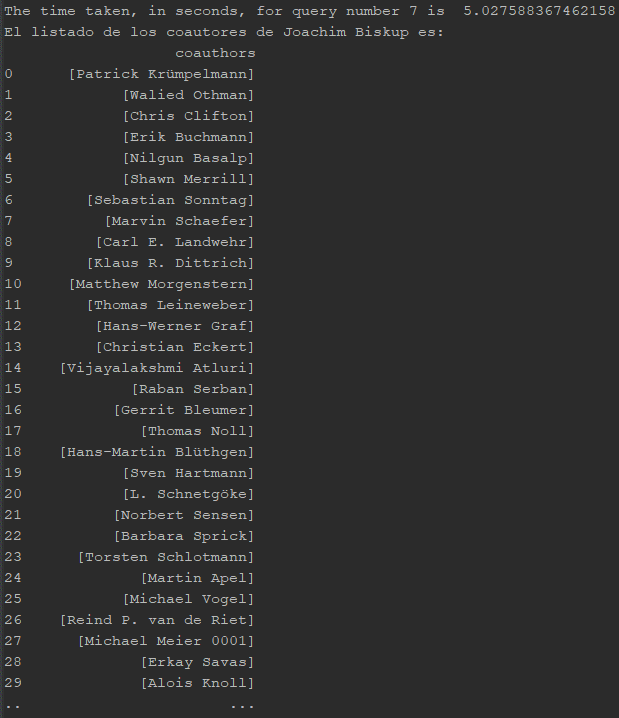
1. Número medio de autores de todas las publicaciones que tenga en su conjunto de datos



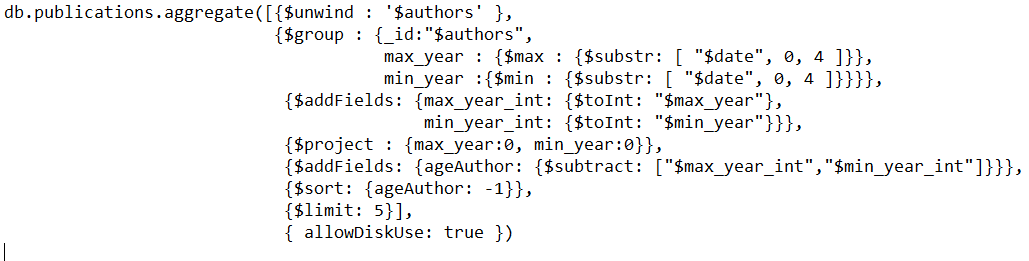


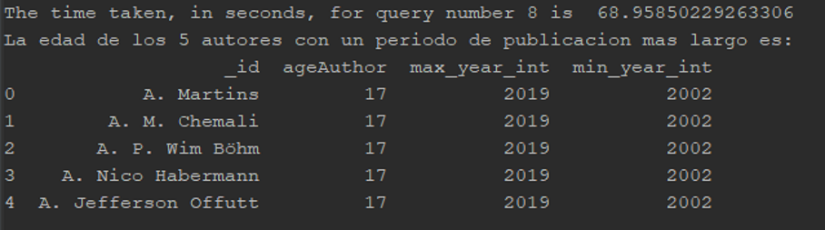
1. Listado de coautores de un autor (se denomina coautor a cualquier persona que haya firmado una publicación)



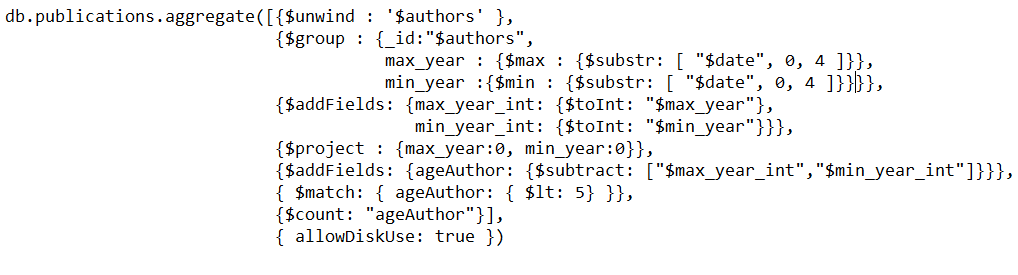


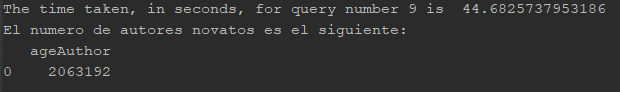
1. Edad de los 5 autores con un periodo de publicación más largo (se considera la edad de un autor al número de años transcurridos desde la fecha de su primera publicación hasta la última registrada)



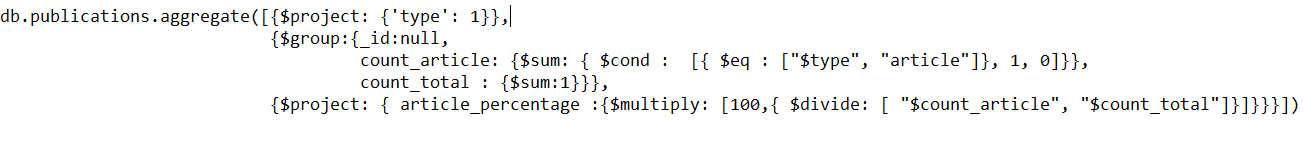


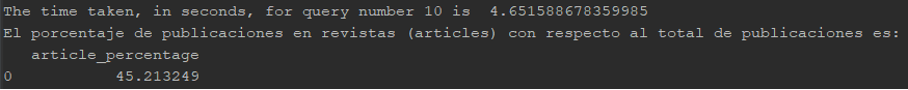
1. Número de autores novatos, es decir, que tengan una edad menor de 5 años. Se considera la edad de un autor al número de años transcurridos desde la fecha de su primera publicación hasta la última registrada





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





HABLAR AQUÍ DE DÓNDE SE HAN INSERTADO LOS ÍNDICES Y POR QUÉ, Y AÑADIR A LA TABLA COMPARATIVA LOS TIEMPOS DE EJECUCIÓN CON ÍNDICES

|  |  |  |
| --- | --- | --- |
| Question | Execution time without indexes (in seconds) | Execution time with indexes (in seconds) |
| 1 | 9.12 |  |
| 2 | 11.72 |  |
| 3 | 2.29 |  |
| 4 | 35.30 |  |
| 5 | 45.37 |  |
| 6 | 7.39 |  |
| 7 | 5.027 |  |
| 8 | 68.95 |  |
| 9 | 44.68 |  |
| 10 | 45.21 |  |

# **PARTE II: NEO4J**

## **1 .- CAPTURA Y PROCESAMIENTO DE DATOS**

## **2.- ALMACENAMIENTO DE DATOS**

Comente brevemente las diferencias que encuentren ambos sistemas (MongoDB y Neo4J)

4 nodes label: autor, article, incollection and inproceeding

Total of nodes: 6,699,232

One kind of relationship : PUBLICATED 13,253,583

## **3.- ANÁLISIS DE DATOS**

¿Qué tipo de consultas son más apropiadas en neo y no lo son en mongo? Más fácil moverte por relaciones

¿Qué autor tiene más colaboraciones?