RI: Práctica 3

UNIVERSIDAD DE GRANADA

Juan Manuel Consigliere Picco
Óscar Pérez Tobarra

1. Creación del índice con Lucene:

Partiendo del ejemplo proporcionado en el guión de prácticas, hemos creado el programa IndiceSimple.java, que permite crear un índice Lucene y rellenarlo con los documentos que se encuentren en la carpeta docs, para posteriormente almacenarlo en la carpeta index. El comando para su ejecución es el siguiente:

```
java -cp
.:./commons-beanutils-1.9.4.jar:./commons-collections4-4.4.jar:./c
ommons-lang3-3.12.0.jar:./commons-text-1.9.jar:./opencsv-5.5.2.jar
:./lucene-8.10.1/core/lucene-core-8.10.1.jar:./lucene-8.10.1/analy
  sis/common/lucene-analyzers-common-8.10.1.jar indiceSimple.java
```

Veamos el código. Se ha definido una clase IndiceSimple que contiene unas variables de clase, un constructor y dos métodos. Empezaremos primero por las variables de clase.

```java
public class indiceSimple {
    String indexPath = "./index";
    String docPath = "./docs";
    boolean create = true;
    private IndexWriter writer;
    Map<String, Analyzer> analyzerPerField = new HashMap<String, Analyzer>();
```

Tenemos dos variables que sirven para especificar la ruta relativa donde se encuentran los documentos a indexar (docPath) y donde se alojarán los índices (indexPath). También tenemos una variable IndexWriter, que necesitaremos posteriormente para poder crear los índices, y un map de Strings y Analyzers, que nos servirá para indicar los analizadores que se usarán en cada campo indexado.

Pasamos a ver primero los métodos antes que el constructor, pues no podemos explicar la funcionalidad del constructor sin explicar primero la del método para crear y añadir documentos Lucene al índice.
Tenemos primero el método configurarIndice.

```java
public void configurarIndice(Analyzer analyzer, Similarity similarity) throws IOException {
    PerFieldAnalyzerWrapper analyzerWrapper = new PerFieldAnalyzerWrapper(analyzer, analyzerPerField);

    IndexWriterConfig iwc = new IndexWriterConfig(analyzerWrapper);

    iwc.setSimilarity(similarity);
    iwc.setOpenMode(IndexWriterConfig.OpenMode.CREATE);

    Directory dir = FSDirectory.open(Paths.get(indexPath));
    writer = new IndexWriter(dir, iwc);
}
```

No varía del ejemplo provisto por el guión de prácticas, salvo por el analizador que le pasamos como parámetro al IndexWriterConfig, que es el map que hemos visto antes. Este método se ejecutará antes de procesar y crear los documentos Lucene.

El segundo método es donde se encuentra lo más importante: la creación de documentos Lucene.

```java
public void indexarDocumentos(File archivo) throws IOException, CsvException {
    CSVReader reader = new CSVReader(new FileReader(docPath+"/"+archivo.getName()));
    reader.skip(1);
    String[] r = reader.readNext();
    System.out.println("Archivo "+archivo.getName());

    do{
        Document doc = new Document();
        doc.add(new TextField(  "file_name",
                                archivo.getName().substring(0, archivo.getName().lastIndexOf(".")),
                                Field.Store.YES));
        doc.add(new StoredField("file_dir", docPath+"/"+archivo.getName()));
        doc.add(new TextField("author", r[0], Field.Store.YES));
        if(!r[1].equals("[No author id available]")){ // !r[1].equals("[No author id available]")
            doc.add(new TextField("author_id", r[1], Field.Store.YES));
        }
        doc.add(new TextField("title", r[2], Field.Store.YES));
        if(!r[3].equals("")){
            doc.add(new SortedNumericDocValuesField("year", Long.parseLong(r[3])));
            doc.add(new StoredField("year", Long.parseLong(r[3])));
        }
        doc.add(new TextField("source_title", r[4], Field.Store.YES));
        doc.add(new StoredField("volume", r[5]));
        doc.add(new StoredField("issue", r[6]));
        doc.add(new StoredField("article_number", r[7]));
        doc.add(new StoredField("page_start", r[8]));
        doc.add(new StoredField("page_end", r[9]));
        doc.add(new StoredField("page_count", r[10]));
        if(!r[11].equals("")){
            doc.add(new SortedNumericDocValuesField("cited_by", Long.parseLong(r[11])));
            doc.add(new StoredField("cited_by", Long.parseLong(r[11])));
        }
        doc.add(new StringField("doi", r[12], Field.Store.YES));
        doc.add(new StoredField("link", r[13]));
        doc.add(new TextField("affiliations", r[14], Field.Store.YES));
        doc.add(new TextField("abstract", r[16], Field.Store.YES));
        doc.add(new TextField("author_keywords", r[17], Field.Store.NO));
        doc.add(new TextField("index_keywords", r[18], Field.Store.NO));
        doc.add(new StringField("doc_type", r[19], Field.Store.NO));
        doc.add(new StoredField("public_status", r[20]));
        doc.add(new StoredField("eid", r[23]));
        writer.addDocument(doc);
        r = reader.readNext();
    }while(r!=null);
```

El método recibe como parámetro un archivo, que es un archivo CSV obtenido de Scopus y que contiene 2000 documentos, el cual se le pasa a un lector de archivos CSV para poder procesarlo. Una vez cargado el lector, avanzamos una línea (la que incluye el nombre de cada columna) y empezamos a leer el archivo. Por cada fila leída, se creará un documento Lucene y se añadirá al IndexWriter. A continuación explicaremos los tipos que hemos escogido para campo:

- file_name: el nombre del archivo CSV (podríamos considerarlo la temática). Se ha definido como un TextField para poder permitir que se indexe mediante los tokens que contiene el campo (por ejemplo, natural/language/processing). Se especifica que el campo se almacene para devolverlo con “Field.Store.YES”.
- file_dir: el nombre del directorio en el que se encuentra el documento a procesar. Se ha definido como un StoredField, pues no resulta interesante permitir búsquedas por este campo, y sí es más interesante como un atributo del documento Lucene.
- author: los creadores del documento a procesar. Se ha definido como un TextField porque el campo puede incluir múltiples autores, y se ha de permitir la búsqueda por cada uno de ellos. También se especifica que se almacene el campo para poder devolverlo en consultas.

- author_id: el identificador de los creadores del documento a procesar. Se ha definido como un TextField por la misma razón que los autores, aunque no sea tan común realizar búsquedas por identificadores. En particular, hay que introducir una condición para que, en el caso de que el documento no incluya los identificadores, se deje el campo vacío y no se introduzca el valor “[No author id available]”. También se especifica que se almacene el campo para poder devolverlo en consultas.
- title: el título del documento. Se ha definido como un TextField para permitir la búsqueda por cada uno de los términos que componen el título, además de que es un atributo que suele formar parte de las búsquedas. También se especifica que se almacene el campo para poder devolverlo en consultas.
- year: el año en el que se publicó el documento. Se ha definido tanto como un SortedNumericDocValuesField, como un StoredField. De esta forma, podemos indexar los documentos por el año en el que se publicaron, así como devolver el año como un atributo de cada documento.
- source_title: el título de la fuente del documento (una revista científica, por ejemplo). Es un TextField, pues puede tener sentido realizar búsquedas por publicaciones científicas de confianza, y por lo tanto hay que permitir la búsqueda por los tokens que componen el campo. También se especifica que se almacene el campo para poder devolverlo en consultas.
- volume, issue, article_number, page_start, page_end, page_count: atributos del documento con respecto a la revista científica que lo publica. No tiene sentido indexar por ellos, así que se almacenan como StoredFields para poder devolverlos como atributos del documento.
- cited_by: el número de citas o referencias que ha tenido el documento. Se ha definido tanto como un SortedNumericDocValuesField como un StoredField. La razón es poder utilizarlo más adelante como un posible medidor de la relevancia del documento cuando implementemos la recuperación de documentos.
- doi: el Identificador de Objeto Digital (DOI) del documento. Se ha definido como un StringField porque es un identificador único del documento, por lo que solo tiene sentido indexar por la cadena completa. También se especifica que se almacene el campo para poder devolverlo en consultas.
- link: enlace que lleva al documento completo. No tiene sentido indexar por un enlace, así que se define como un StoredField y poder devolverlo en caso de que alguien quiera acceder al documento original.
- affiliation: entidades a las que están asociadas los autores del documento (por ejemplo, una universidad). Se ha definido como un TextField porque el campo incluye la afiliación de todos los autores, así que debería permitirse buscar por cada una de las entidades a las que están vinculados los autores. También se especifica que se almacene el campo para poder devolverlo en consultas.
- abstract: resumen del documento. Se define como un TextField para poder indexar por los tokens del abstract, lo cual es muy común cuando se están buscando artículos científicos. También se especifica que se almacene el campo para poder devolverlo en consultas.
- author_keywords: palabras clave relacionadas con los autores. Se ha definido como un TextField para permitir una búsqueda más rápida por términos, aunque en este caso no lo almacenaremos para devolverlo en las consultas porque consideramos que no incluye nada que no podamos mostrar con el campo “abstract”.

- index_keywords: palabras clave usadas por el índice (no el nuestro). Nos puede servir para nuestro propio índice, por lo que lo definimos como un TextField sin almacenarlo.
- doc_type: tipo del documento (artículo, libro, etc). Puede tener sentido indexar por el tipo de documento, y como no presenta variabilidad en los posibles valores (son siempre los mismos), podemos definirlo como un StringField sin almacenarlo.
- public_status: estado de publicación del documento. No se suele buscar por el estado de publicación, por lo que lo definimos como un StoredField para poder devolverlo en las consultas.
- eid: identificador asociado al documento en la base de datos de Scopus. No tiene sentido indexar por este campo teniendo el DOI como un identificador más importante, por lo que lo definimos como un StoredField.

Una vez explicados los tipos definidos para cada campo, ya tiene sentido explicar el constructor de la clase.

```
indiceSimple(){
    analyzerPerField.put("author", new SimpleAnalyzer());
    analyzerPerField.put("author_id", new ClassicAnalyzer());
    analyzerPerField.put("title", new SimpleAnalyzer());
    analyzerPerField.put("source_title", new SimpleAnalyzer());
    analyzerPerField.put("doi", new KeywordAnalyzer());
    analyzerPerField.put("affiliations", new SimpleAnalyzer());
    analyzerPerField.put("abstract", new EnglishAnalyzer());
    analyzerPerField.put("author_keywords", new SimpleAnalyzer());
    analyzerPerField.put("index_keywords", new SimpleAnalyzer());
    analyzerPerField.put("doc_type", new KeywordAnalyzer());
}
```

Para los campos StringField y TextField definidos, tenemos que decidir qué analizador usamos para poder procesarlo e insertar el campo en el índice, así como para poder realizar búsquedas:

- author: con el SimpleAnalyzer cambiamos todo a minúsculas y eliminamos los signos de puntuación. Es un problema por los nombres de pila, pero normalmente suele aparecer solo la inicial y se suele buscar por los apellidos, así que nos quedamos con este analizador.
- author_id: con el ClassicAnalyzer nos quitamos las comas que hay separando cada identificador.
- title: otra vez, con el SimpleAnalyzer hacemos un trabajo rápido y sencillo eliminando los espacios en blanco y los signos de puntuación.
- source_title: usa el SimpleAnalyzer por la misma razón que el campo “title”.
- doi: al ser un identificador almacenado como una cadena completa por ser un identificador, podemos usar el KeywordAnalyzer, que está hecho específicamente para tratar con identificadores.
- affiliations: usa el SimpleAnalyzer por la misma razón que el campo “author”.
- abstract: al ser un texto a procesar y que suele estar siempre escrito en inglés, podemos utilizar el EnglishAnalyzer que ya incluye las stopwords del inglés y puede tratar mejor los textos en inglés que el SimpleAnalyzer o el StandardAnalyzer.
- author_keywords: usa el SimpleAnalyzer por la misma razón que el campo “author_id”.

- index_keywords: usa el SimpleAnalyzer por la misma razón que el campo “author_keywords”.
- doc_type: usa el KeywordAnalyzer porque también sirve para tratar con tipos de objetos.

Una vez explicada la funcionalidad de la clase IndiceSimple, podemos ver cómo realiza la ejecución el programa:

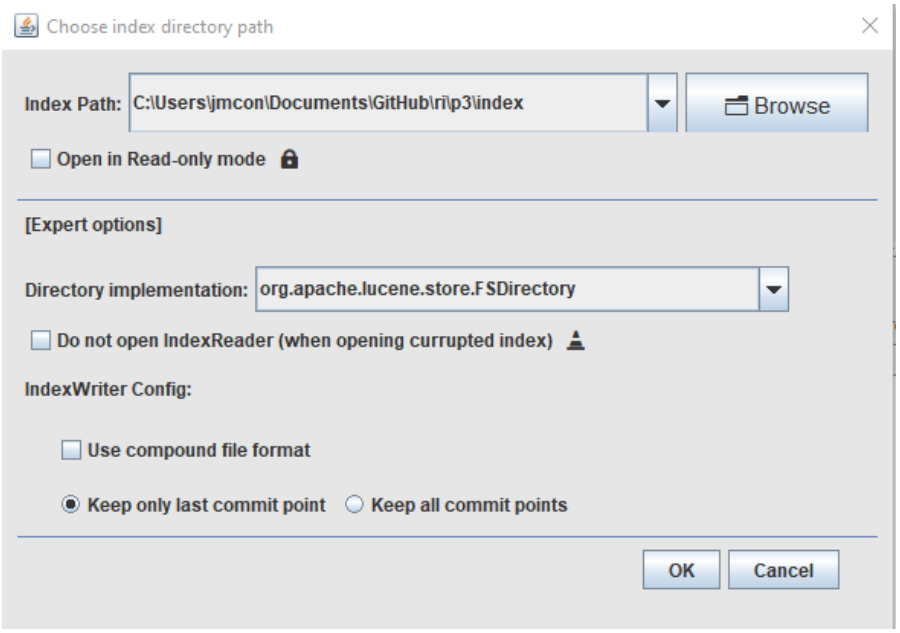
```
    public static void main(String[] args) throws IOException, CsvException {
        File dir = new File("./docs");
        File[] archivos = dir.listFiles();
        Analyzer analyzer = new StandardAnalyzer();
        Similarity similarity = new ClassicSimilarity();
        indiceSimple baseline = new indiceSimple();

        baseline.configurarIndice(analyzer, similarity);
        for(File archivo : archivos){
            baseline.indexarDocumentos(archivo);
        }
        baseline.close();
    }
}
```

Se define un analizador por defecto que pasarle al IndexWriter, así como un objeto Similarity para que Lucene pueda medir el peso de cada término (no es relevante en este momento). También obtenemos los ficheros que contienen los documentos a indexar. El procedimiento es llamar al constructor de IndiceSimple para que cree el analyzerPerFieldWrapper, que es común a todos los documentos ya que comparten estructura, configurar el índice para indicarle el analizador por defecto, y una vez configurado el índice, procedemos a iterar los archivos incluidos en el directorio “docs” para poder obtener los documentos Lucene que se insertarán en el índice Lucene. Este es el resultado de la ejecución:

```
$java -cp .:./commons-beanutils-1.9.4.jar:./commons-collections4-4.4.jar:./commons-lang3-3.12.0.jar:./commons-text-1.9.j
ar:./opencsv-5.5.2.jar:./lucene-8.10.1/core/lucene-core-8.10.1.jar:./lucene-8.10.1/analysis/common/lucene-analyzers-comm
on-8.10.1.jar indiceSimple.java
Archivo cloud-computing.csv
Archivo data-science.csv
Archivo digital-library.csv
Archivo geographic-information-system.csv
Archivo marketing.csv
Archivo natural-language-processing.csv
Archivo software.csv
Archivo spain.csv
Archivo videogames.csv
Archivo wireless-connection.csv
```

2. Índice y consultas con Luke:

Tras compilar y ejecutar el código anterior procedemos a abrir Luke. Este programa nos va a mostrar el índice resultante, así como nos va a dar una interfaz para realizar consultas y probar analizadores. Procedemos a abrir el índice creado:



Aquí vamos a poder ver nuestro índice, el cual nos ha quedado de esta manera:

Name	Term ...	%
author_id	87788	27.53 %
abstract	65137	20.42 %
author	44024	13.80 %
affiliations	32326	10.14 %
title	25128	7.88 %
author_keywords	20956	6.57 %
index_keywords	20563	6.45 %
doi	19015	5.96 %
source_title	3968	1.24 %
file_name	18	0.01 %
doc_type	13	0.00 %
year	0	0.00 %
volume	0	0.00 %
public_status	0	0.00 %
page_start	0	0.00 %
page_end	0	0.00 %
page_count	0	0.00 %
link	0	0.00 %
issue	0	0.00 %
file_dir	0	0.00 %
eid	0	0.00 %
cited_by	0	0.00 %
article_number	0	0.00 %

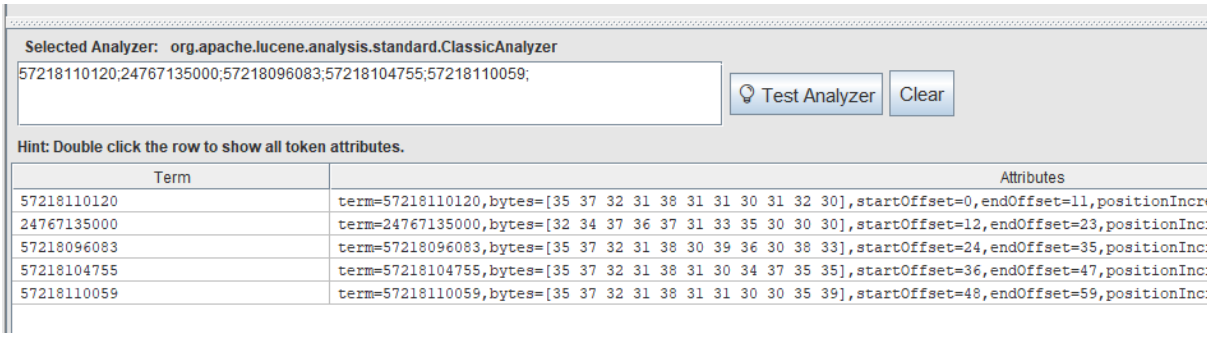
Se puede observar que hay algunos campos que no tienen conteo de términos. Estos son los StoredField que hemos especificado anteriormente, además de los valores Sorted especificados (year y cited_by). Ahora voy a mostrar algunos detalles, y finalmente haré unas consultas.

El DOI decidimos tomarlo como tokens completos, y sacamos la idea de un bot de Telegram el cual le pasabas el DOI y el te daba el documento. Los términos son de este estilo:

Rank	Freq	Text
14	1	10.7544/issn1000-1239.2021.20200298
15	1	10.7544/issn1000-1239.2020.20190721
16	1	10.7146/mediekultur.v37i70.122398
17	1	10.6288/TJPH.202104_40(2).109128
18	1	10.6251/BEP.202013_52(3).0009
19	1	10.6092/issn.2532-8816/12449
20	1	10.6017/ITAL.V40I2.12987
21	1	10.6017/ITAL.V40I2.12963
22	1	10.6017/ITAL.V40I2.12751
23	1	10.6017/ITAL.V40I1.12647
24	1	10.6017/ITAL.V39I4.12483
25	1	10.6017/ITAL.V39I4.12457
26	1	10.6017/ITAL.V39I4.12363
27	1	10.6017/ITAL.V39I4.11859
28	1	10.6000/1929-4409.2020.09.342
29	1	10.5944/SIGNA.VOL30.2021.29301
30	1	10.5944/SIGNA.VOL30.2021.29300

A primera vista parece muy descabellado que alguien busque por DOI, pero hemos decidido tomarlo como un valor relevante para la búsqueda. El único requisito es que el usuario final sepa exactamente el DOI.

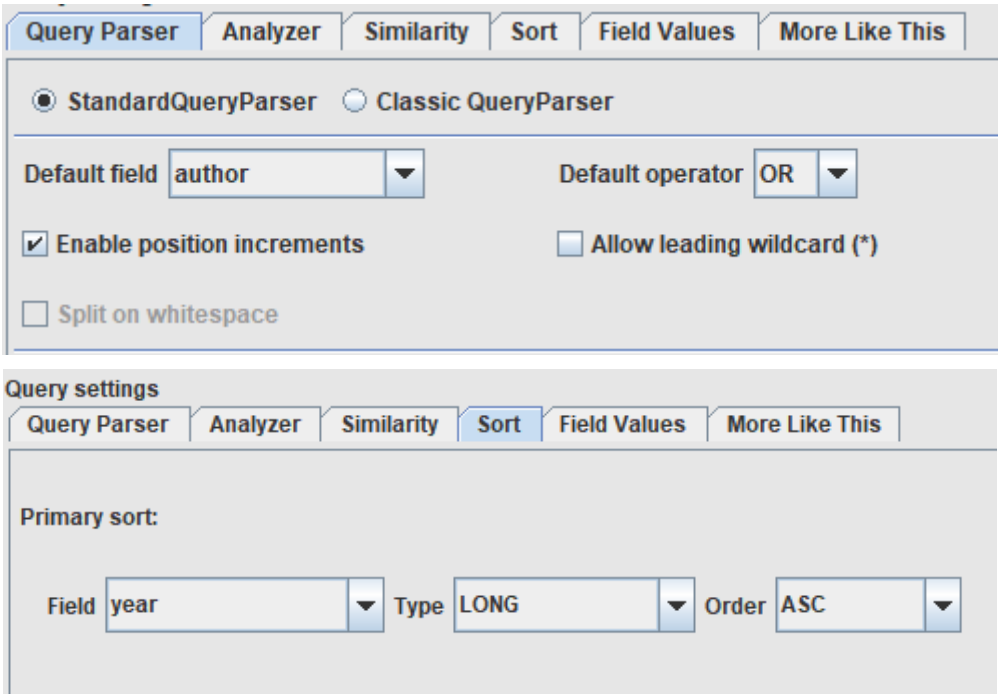
Para el campo author_id encontramos un analizador que se ajustara a lo que necesitábamos con una herramienta propia de Luke. En la pestaña Analysis puedes probar los que necesites, además de crear analizadores personalizados. Con ClassicAnalyzer() nos sale de esta manera:



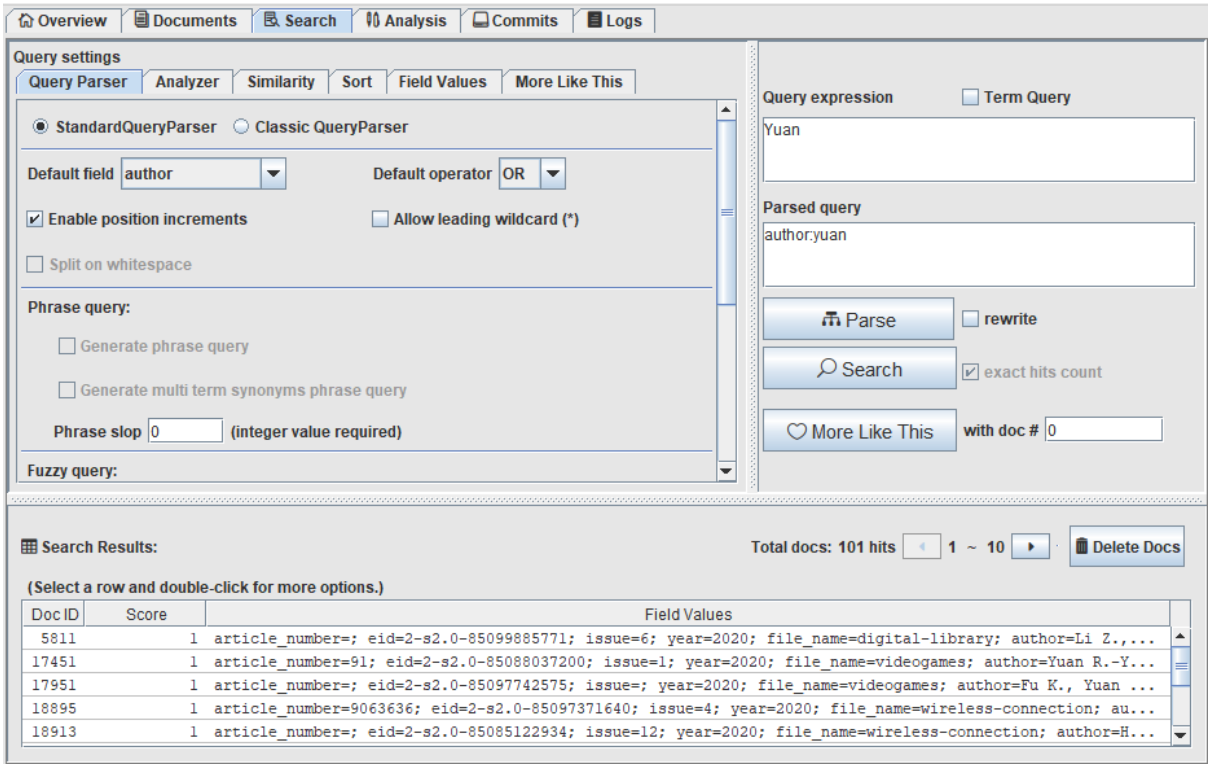
Observamos que se tokeniza correctamente, y combinado con la condición explicada en el apartado anterior nos queda el índice de esta manera:

Rank	Freq	Text
1	8	57221404144
2	8	35483695800
3	5	57217043238
4	5	56659377900
5	5	56047352500
6	4	35182392700
7	4	26326592100
8	4	17435193400
9	3	8352937800
10	3	7102932568
11	3	6603846580
12	3	6602789810
13	3	6602255248
14	3	6602236601

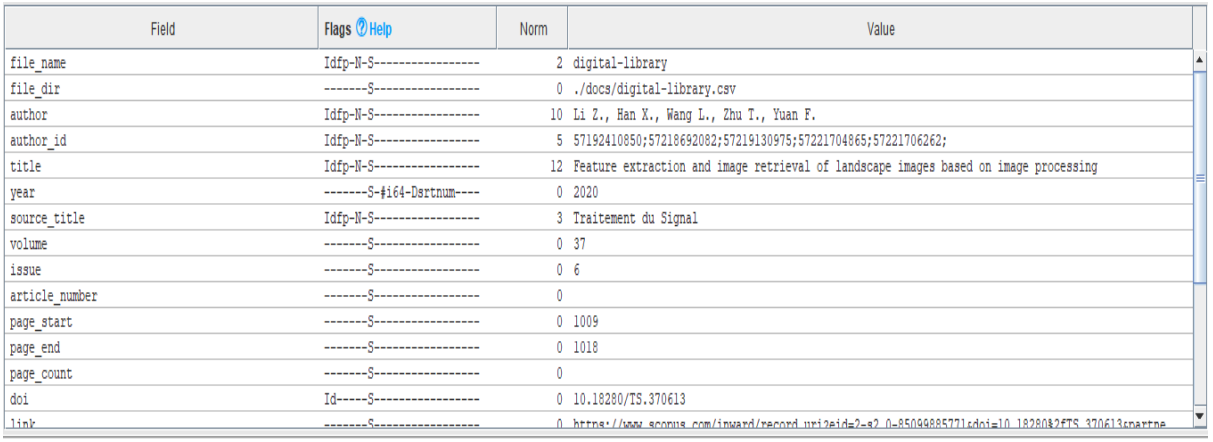
Ahora pasaremos a realizar algunas búsquedas por campos, y ordenando. La primera búsqueda será con el campo author, la cual le pasaremos un autor y lo ordenaremos por año:



La consulta queda de esta manera:



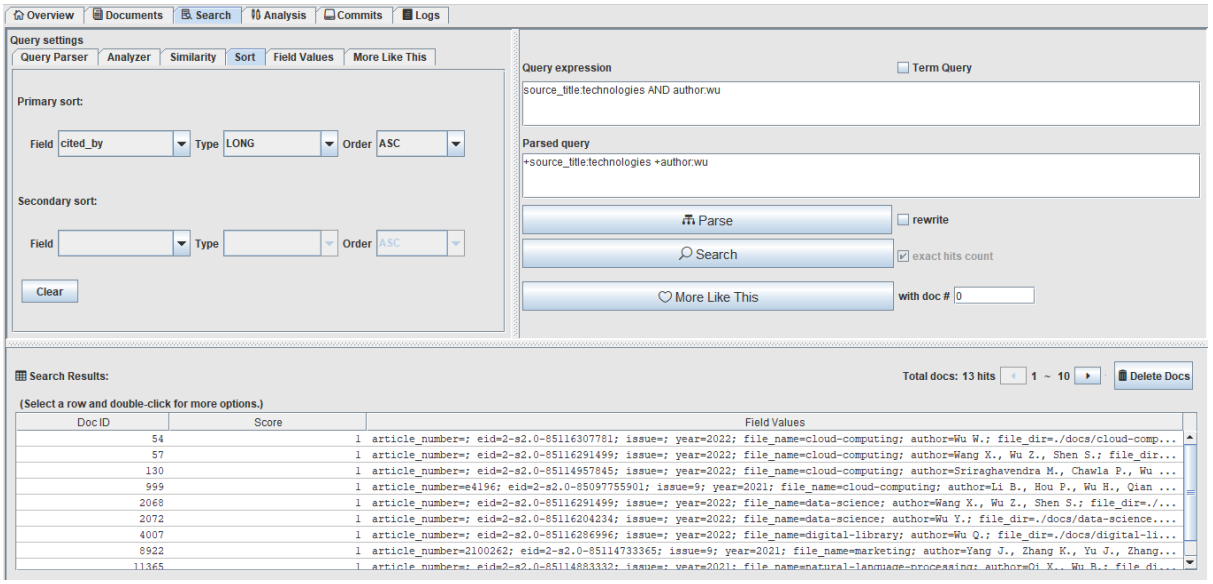
Hemos buscado por el autor yuan, y vemos cómo en la salida los documentos se ordenan por año. Si le damos a Show all fields nos muestra la salida. Los campos que figuran son los que hemos especificado como stored previamente en el código (aparece una S en la columna flags



Field	Flags Help	Norm	Value
file_name	Idfp-N-S----------	2	digital-library
file_dir	-------S----------	0	./docs/digital-library.csv
author	Idfp-N-S----------	10	Li Z., Han X., Wang L., Zhu T., Yuan F.
author_id	Idfp-N-S----------	5	57192410850;57218692082;57219130975;57221704865;57221706262;
title	Idfp-N-S----------	12	Feature extraction and image retrieval of landscape images based on image processing
year	-------S-#i64-Dsrtnum----	0	2020
source_title	Idfp-N-S----------	3	Traitement du Signal
volume	-------S----------	0	37
issue	-------S----------	0	6
article_number	-------S----------	0	
page_start	-------S----------	0	1009
page_end	-------S----------	0	1018
page_count	-------S----------	0	
doi	Id-----S----------	0	10.18280/TS.370613
link	-------S----------	0	https://www.scopus.com/inward/record.uri?eid=2-s2.0-85099885771&doi=10.18280%2fTS.370613&partne

También vemos que hemos indexado el nombre del archivo que lo contiene, y el directorio que lo contiene. Esto nos servirá más adelante en la fase de búsqueda.

Ahora vamos a buscar de otra manera: vamos a la pestaña de Overview, clickeamos en cualquier campo (en este caso source_title) y clickeamos dos veces en alguno de los términos, dándole a Search docs by term. Lo que hace es generarnos una consulta con el término elegido. Nosotros lo hicimos con el término technologies y la salida fue correcta. Escogimos también ordenarlo de forma ascendente por número de citas y le añadimos la condición de que el autor sea Wu. La salida fue la siguiente:



Como conclusión vemos que Luke sirve para inspeccionar los índices ya creados y para realizar consultas y comprobar que todo lo hemos hecho correctamente. También es una herramienta muy útil para entender cómo funcionan los distintos analizadores que nos trae Lucene, y para probar algunos que nosotros creemos.

No obstante, esto es solamente el principio de la práctica y más adelante necesitaremos de nuestra propia interfaz para realizar las búsquedas.