Máster en Data Science. URJC Sede Madrid

Bases de datos no convencionales

Javier Llorente Mañas
Carlos Sánchez Vega

2017 - 2018

Índice

0.1 Introducción

El objetivo de este documento es mostrar cada una de las partes de las que está compuesto el proyecto (código desarrollado y los problemas encontrados).

0.2 Secciones de la memoria

- Elección del conjunto de datos
- Elección del diseño de la base de datos
- Explicación del código
- Índices creados para las consultas en MongoDB
- Consultas en MongoDB
- Consultas en Neo4J
- Problemas encontrados durante el desarrollo de la práctica

0.3 Elección del conjunto de datos

Hemos elegido el conjunto de datos del xml en total, es decir, de dblp.xml

0.4 Elección del diseño de la base de datos

Como se describe en el enunciado de la práctica, un autor puede haber escrito varias publicaciones. A su vez, una publicación puede haber sido escrita por varias personas, por lo que podríamos decir que nos encontramos con una relación de "muchos a muchos" entre las entidades de "publicaciones"" y "autores". En un principio, habíamos pensado que la mejor estrategia consistía en embeber un documento en otro, es decir, que el documento "publicaciones" contuviese los "autores" como otro documento. No obstante, la elección del diseño de la base de datos debe hacerse en base a la eficiencia de las consultas que se vayan a ejecutar. Nos encontrábamos con la siguiente problemática: por un lado, no se pueden embeber las publicaciones en los autores, ya que hay consultas que requieren la obtención de los autores de una determinada publicación. Por otro lado, no podíamos embeber los autores en las publicaciones, porque hay consultas relativas a las publicaciones de un determinado autor. Por tanto, decidimos crear dos colecciones: una para autores y otra para documentos.

Authors
_id
name
publications [5a11d54b555f43057554fbc, 5a5345d5b5345f466770bc4fbc, 5ad66565b77889930770bc4fbf]

Como se puede ver, hemos incluido en el campo "publications" referencias a los id's de las publicaciones escritas por ese autor.

Publications
_id
type
title
...
authors [5ad5bf438726e30770bc4fbc, 5ad5bf438726e30770bc4fbc, 5ad5bf438726e30770bc4fbf]

Por otro lado, en el campo "authors" tendremos las referencias a los autores de esa publicación.

0.5 Explicación del código

Nuestro proyecto está dividido en tres clases python, divididos por funcionalidad.
* parseXMLToJSON.py
* insertRecords.py
* parseXMLToCSV.py

A Continuación explicaremos de forma separada cada uno de los ficheros:

parsehtml.py: tiene como objetivo extraer las etiquetas consideradas relevantes del xml de entrada, para generar un fichero json con el formato adecuado para que pueda ser almacenada en la base de datos Mongo. Es importante mencionar que hemos usado la librería "lxml" de python para procesar el fichero de entrada xml.

```
import time
from lxml import etree
from bson import ObjectId
import json

class JSONEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, ObjectId):
            return str(o)
        return json.JSONEncoder.default(self, o)

"""
we have created a dictionnary in order to store the features from each author.
 Say we have the next dictionnary corresponding to one of the authors:
 "Hans Ulrich Simon": {
    "_id": "5ad369fc8726e36f64367cfe",
    "name": "Hans Ulrich Simon",
    "publications": [
        "5ad369fc8726e36f64367cfd"
    ]
 }
 As we can see, we have the name of the author as the key of a dictionnary and then, we have another one embedded.
 It is the content what we do want to output in a file """
def get_nice_nice_structure_authors (author_publications_dict):
    authors = []
    for key in author_publications_dict:
        authors.append(author_publications_dict[key])
    return authors


# Dumps the content from "dic" to the path of "json_path"
def create_output_file(json_path, dic):
    with open(json_path, 'w') as outfile:
        json.dump(dic, outfile, cls=JSONEncoder, indent=4)
```

```python
# Parses a xml file to a json one
def parse_publications(xml_input_path, json_output_path_publications, json_output_path_authors):

    valid_publications = {'article', 'inproceedings', 'incollection' }
    not_valid_tags = {'dblp'}

    """
    It is necessary to load the dtd file. Otherwise, when trying to parse text containing special characters,
    such as the text "Arnold Sch&ouml;nhage", an execution is thrown
    """
    context = etree.iterparse(source = xml_input_path, events=("start", "end"),dtd_validation=True, load_dtd=True)

    # we initialize the variables to store the content of our input file
    publications_dict = {}
    publications = []
    author_publications_dict = {}
    for action, elem in context:

            # It is the openning tag
            if elem.tag not in not_valid_tags:
                if action =='start':
                    if len(elem):
                        text = "None"
                        # If the tag does not have any attribute, it means it is the root node
                        # we create an id for that publication
                        id_publication  = ObjectId()
                        publications_dict["_id"] = id_publication

                        # it the publication has not been stored yet, we have to initialize its list of authors
                        nodo_authors = []
                    else:
                        # If the author existed previously in the dictionnary
                        text = elem.text

                    ######### code snippet corresponding to the list of publications by an author
                    # Each publication may have several authors
                    if elem.tag == "author":
                        if elem.text in author_publications_dict:

                                # we add the publication to the list of its publications
                                # we create a dictionnay in order to store the features from each author (the key is
                                # the name)
                                author_publications_dict[elem.text]["publications"].append(id_publication)

                                # we get the associated id to be included to the list of author of a publication
                                id_author = author_publications_dict[elem.text]["_id"]
                        else:
                            # if the author did not exist previously, we have to create the structure to store it in the list
                            # of publications from that author
                            id_author = ObjectId()
                            nodo_publications ={}
                            nodo_publications["_id"] = id_author
                            nodo_publications["name"] = elem.text
                            nodo_publications["publications"] = []
                            nodo_publications["publications"].append(id_publication)
                            author_publications_dict[elem.text] = nodo_publications

                            # we add that authot to the list of author of a publication
                            nodo_authors.append(id_author)
                        else:
                            # In case of a publication tag, we get the type
                            if elem.tag in valid_publications:
                                publications_dict["type"] = elem.tag
                            # if we find a digit, we need to convert it to integer as we have to do some arithmetic operations
                            # in our queries
                            elif elem.text.isdigit():
                                publications_dict[elem.tag] = int(text)
                            else:
                                publications_dict[elem.tag] = text


                    # if it is the closing tag of a publication
                    if elem.tag in valid_publications and action =='end':
                        publications_dict["authors"] = nodo_authors
                        publications.append(publications_dict)
                        publications_dict = {}

    create_output_file(json_output_path_publications, publications)
    authors = get_nice_nice_structure_authors(author_publications_dict)
    create_output_file(json_output_path_authors, authors)
    return publications


if __name__ == "__main__":
    xml_input_path = "./FicherosBBDD/XML/dblp.xml"
    xml_output_path_publications = "./FicherosBBDD/JSON/publications.json"
    xml_output_path_authors = "./FicherosBBDD/JSON/authors.json"

    start_time = time.time()
    parse_publications(xml_input_path, xml_output_path_publications,xml_output_path_authors )
    print("The execution took: {0:0.2f} seconds".format(time.time() - start_time))
```

Para realizar el proceso de transformación de xml a json hemos creado la función parse_publications. Esta función recibe tres argumentos:“xml_input_path” (ruta donde se encuentra el fichero XML de entrada), “json_output_path_publications” y “json_output_path_authors”, que corresponden con la rutas donde se crearán los ficheros json de salida de “publications” y “authors”,respectivamente. Para almacenar las publicaciones, crearemos la variable “publications”. Esta variable en una lista de diccionarios y almacenará, en cada posición de la lista, un diccionario correspondiente a cada una de las publicaciones procesadas. Un ejemplo de uno de los elementos almacenados en dicha lista es el siguiente:

```
{
    "_id": "5ad5bf438726e30770bc4fbb",
    "type": "incollection",
    "title": "Parallel Integer Sorting and Simulation Amongst CRCW Models.",
    "pages": "607-619",
    "year": 1996,
    "volume": 33,
    "journal": "Acta Inf.",
    "number": 7,
    "url": "db/journals/acta/acta33.html#Saxena96",
    "ee": "https://doi.org/10.1007/BF03036466",
    "authors": [
        "5ad5bf438726e30770bc4fbc"
    ]
}
```


En lo que respecta a la estructura usada para almacenar los autores junto con las publicaciones que han escrito, nos encontrábamos con la situación de que una persona pudiera aparecer como autor de varias varias publicaciones diferentes a lo largo de todo el documento. Por ello, habíamos pensado que una estrategia para poder localizar el autor, rápidamente, era la de usar un diccionario en el que la clave sería el nombre del autor y, como valor, tendríamos otro diccionario con la información relativa a ese autor. Un ejemplo de esta estructura es la siguiente:


```
"Hans Ulrich Simon": {
    "_id": "5ad369fc8726e36f64367cfe",
    "name": "Hans Ulrich Simon",
    "publications": [
        "5ad369fc8726e36f64367cfd"
    ]
 }
```


Como vemos, “Hans Ulrich Simon” es la clave del diccionario siguiente:


```
    "_id": "5ad369fc8726e36f64367cfe",
    "name": "Hans Ulrich Simon",
    "publications": [
        "5ad369fc8726e36f64367cfd"
    ]
```


En resumen, lo que hacemos en dicha función es leer el documento de entrada, creando diccionarios tanto para las publicaciones como para los autores. En caso de encontrarnos la

etiqueta “author”, se deberán realizar las siguientes acciones:

1. Estructura de datos de publicaciones: almacenar ese autor en la lista de autores de la publicación
2. En el caso de la estructura de datos de autores, se distinguen las siguientes dos situaciones:
   (a) Si no se habí procesado ese autor, se crea un diccionario para ese autor y se añade la publicación al autor.
   (b) Si ya se había procesado ese autor, se añadía la publicación a la lista de publicaciones del autor

La diferencia entre las estructuras seleccionadas para almacenar los autores y las publicaciones es que la estructura de publicaciones, al ser una lista de diccionarios, está “preparada” para volcarse en el fichero destino “publications.json”. En cambio, la estructura para los autores, al ser un diccionario de diccionarios (tal y como hemos mostrado anteriormente), debemos procesarlo para volcarlo al fichero de “authors.json”. Para ello, usaremos la función:

```
"""
we have created a dictionnary in order to store the features from each author.
 Say we have the next dictionnary corresponding to one of the authors:
 "Hans Ulrich Simon": {
    "_id": "5ad369fc8726e36f64367cfe",
    "name": "Hans Ulrich Simon",
    "publications": [
        "5ad369fc8726e36f64367cfd"
    ]
 }
 As we can see, we have the name of the author as the key of a dictionnary and then, we have another one embedded.
 It is the content what we do want to output in a file """
def get_nice_nice_structure_authors (author_publications_dict):
    authors = []
    for key in author_publications_dict:
        authors.append(author_publications_dict[key])
    return authors
```

Esta función crea una lista de diccionarios con el contenido de los valores de las claves en el diccionario pasado como parámetro.

insertRecords.py: tiene como objetivo procesar los json creados e insertar los elementos en la base de datos en la colección de “auhors” o “publications” dependiendo del json que se procese. En este fichero conectamos con la base de datos, leemos cada uno de los ficheros de entrada y, por cada uno de los elementos encontrados en el fichero json, se insertará un documento en la base de datos.

```python
import json
import pymongo
import time


""" Inserts all records of the file of the path """
def insert_collection (collection,json_path):
    page = open(json_path, 'r')
    parsed = json.loads(page.read())
    for item in parsed:
        collection.insert(item)

""" Connects to the database and manage the collections to be stored """
def insert_all_records(url_connection,json_input_path_authors, json_input_path_publications ):
    connection = pymongo.MongoClient(url_connection)
    db=connection.authorAndPublicationData
    collection_authors = db.authors
    insert_collection(collection_authors, json_input_path_authors)
    collection_publications = db.publications
    insert_collection(collection_publications, json_input_path_publications)

if __name__ == "__main__":
    url_connection = "mongodb://localhost"
    json_input_path_authors = "./FicherosBBDD/JSON/authors.json"
    json_input_path_publications = "./FicherosBBDD/JSON/publications.json"
    start_time = time.time(url_connection,json_input_path_authors,json_input_path_publications)
    insert_all_records(url_connection,json_input_path_authors, json_input_path_publications )
    print("The import to Mongo took: {0:0.2f} seconds".format(time.time() - start_time))
```

También se podrían haber importado los ficheros json a través de la consola de Mongo. Ejemplo:

```
(Para las publicaciones)
mongoimport --db dblp --collection publications < /Users/publications.json

(Para los autores)
mongoimport --db dblp --collection authors < /Users/authors.json
```

parseXMLToCSV.py: tiene como objetivo procesar el xml de entrada y crear el fichero CSV que nos servirá para importarlo en la base de datos Neo4J. Las publicaciones pueden tener varios autores, pero se decidió crear una línea en el fichero CSV de salida por cada unos de los autores de la publicación. Otra posible solución hubiera sido tener un campo "autores" que contuviese todos los id's de los autores que tiene la publicación.

```python
import time
from lxml import etree
import csv

# Parses a xml file to a csv one
def parsePublications(xmlFile,csv_output_path):

    valid_publications = {'article', 'inproceedings', 'incollection' }
    not_valid_tags = {'dblp'}

    header_publications = [ "type", "title", "pages","year", "url", "author"]
    valid_fields_publications =  [ "article", "inproceedings", "incollection", "title", "pages","year", "url", "author"]

    """
    It is necessary to load the dtd file. Otherwise, when trying to parse text containing special characters,
    such as the text "Arnold Sch&ouml;nhage", an execution is thrown
    """
    context = etree.iterparse(source = xmlFile, events=("start", "end"),dtd_validation=True, load_dtd=True)

    # we initialize the variables to store the content of our input file
    publications_dict = {}
    with open(csv_output_path,'w') as f:
        writer_to_csv = csv.DictWriter(f, fieldnames = header_publications)
        writer_to_csv.writeheader()
        for action, elem in context:

                # It is the openning tag
                if elem.tag not in not_valid_tags:
                    if action =='start':
                        if elem.text:
                            text = elem.text
                        ########## code snippet corresponding to the list of publications by an author
                        # Each publication may have several authors
                        if elem.tag in valid_fields_publications:
                            # In case of a publication tag, we get the type
                            if elem.tag in valid_publications:
                                publications_dict["type"] = elem.tag

                            # if we find a digit, we need to convert it to integer as we have to do some arithmetic operations
                            # in our queries
                            elif elem.text.isdigit():
                                publications_dict[elem.tag] = int(text)
                            else:
                                publications_dict[elem.tag] = text

                        # if it is the closing tag of a publication
                        if elem.tag in valid_publications and action == 'end':
                            writer_to_csv.writerow(publications_dict)
                            publications_dict = {}


if __name__ == "__main__":
    xml_input_path = "./FicherosBBDD/XML/dblp.xml"
    csv_output_path = "./FicherosBBDD/CSV/dblp.csv"

    start_time = time.time()
    parsePublications(xml_input_path, csv_output_path)
    print("The execution took: {0:0.2f} seconds".format(time.time() - start_time))
```

Como podemos ver, el código es muy similar al fichero en el que se parsea el fichero XML a JSON ("parseToJSON.py").

0.6 Índices creados para las consultas en MongoDB

Hemos creados los siguientes dos índices que nos permitirán optimizar las consultas. Como se puede ver se han creados dos índices para cada una de las lista de id's contenidos:

db.authors.createIndex({"publications":1}) db.pulications.createIndex({"authors":1})

0.7 Consultas en MongoDB

0.7.1 Consulta 1

1. Listado de todas las publicaciones de un autor determinado

```
# hemos elegido al autor "Sanjeev Saxena"
var sanjeev= db.authors.findOne({name: "Sanjeev Saxena"});
db.publications.find({_id: {$in: sanjeev.publications}})
```

0.7.2 Consulta 2

2. Número de publicaciones de un autor determinado.

```
# hemos elegido las publicaciones de "Sanjeev Saxena"
db.authors.aggregate([
{ $match: { name: "Sanjeev Saxena" } },
{$project: {"_id": 0, count: { $size:"$publications" }}}
])
```

0.7.3 Consulta 3

3. Número de artículos en revista para el año 2017.

```
db.publications.aggregate([
            {
              /* seleccionamos los tipos de campos que nos interesan */
              $match: {
                  $and: [
                      {type: "article"},
                      {year: "1997"}
                  ]
              }
            },
            /* contamos los campos filtrados */
            {
                $count: "number_of_articles_2017"
            }
           ])
```

0.7.4 Consulta 4

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

```
  /* primera forma de obtener la query con js */
  db.authors.find( {publications : {$exists:true},
      $where:'this.publications.length<5'} )
/* segunda forma de calcular la query (con concultas nativas de Mongo*/
  db.publications.aggregate([

  /* se guarda el id para futuras consultas */
```

```
    {$project: {'authors': 1}},

    /* para cada uno de los autores*/
    {$unwind : '$authors' },

    /* para cada uno de esos autores, se calcula el conteo total de
       publicaciones*/
    {$group: {_id: '$authors', total_publicaciones : { $sum : 1 }}},

    /* del calculo se seleccionan solo las que el conteo sea menor que 5
       publicaciones */
    {$match:{'total_publicaciones': {'$lt': 5}}}
    ,

    /* se cuenta el numero de autores */
    {$count: "numero_autores_ocasionales"}
    ]
    )
```

0.7.5 Consulta 5

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([
    /* para cada uno de los autores */
    {$unwind: "$authors"},

    /* se realizan las siguientes acciones */
    {$group: {
        '_id': '$authors',

        /* se suman todas las publicaciones */
        count_total: {$sum:1},

        /* se suman las publicaciones de tipo articulo */
        numero_articulos: {
            /* se realiza la operacion de suma */
              $sum: {

                    /* dependiendo de la siguiente condicion (tipo de
                       publicacion "articulo")*/
```

```
                    $cond : [
                              /* si el tipo de publicacion es de articulo se
                                 suma, sino no */
                              {$eq : ["$type", "article"]}, 1, 0]
                          }
        },
       /* se suman las publicaciones de tipo "inproceedings" */
       numero_inproceedings: {
          /* se realiza la operacion de suma */
             $sum: {

                    /* dependiendo de la siguiente condicion (tipo de
                       publicacion) "inproceedings" */
                    $cond : [
                              /* si el tipo de publicacion es de articulo se
                                 suma, sino no */
                              {$eq : ["$type", "inproceedings"]}, 1, 0]
                          }
                  }
                  }},
      /* se ordena de mayor a menor */
      {$sort:
         {count_total: -1}
      },

      /* se toman los 10 primeros */
      {$limit: 10}
   ])
```

0.7.6 Consulta 6

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

```
db.publications.aggregate([
/* puede haber publicaciones que no tengan autores */
 {'$match':
   {'authors': {'$exists': true}}
 },

 /* se guarda la variable para futuras consultas */
 {'$project':
```

```
        {'numero_de_autores': { '$size': '$authors' }}
    },

    /* se calcula el porcentaje */
    {'$group':
        {
            _id: 0,

             /* se calcula la media del numero de autores (operacion anterior) */
            'avg_autores': {'$avg': '$numero_de_autores'}
        }
    }
    ,
    /* solo queremos el campo del porcentaje */
     {$project: {'_id': 0, 'avg_autores': 1}},

])
```

0.7.7 Consulta 7

7. Listado de coautores de un autor (Se denomina coautor a cualquier persona que haya firmado una publicación (entiendo que se refiere a los coautores de una publicacion concreta , por ejemplo la del titulo “”Parallel Integer Sorting and Simulation Amongst CRCW Models.“)

```
var publication = db.publications.findOne({title:"Parallel Integer Sorting and
    Simulation Amongst CRCW Models."});
db.authors.find ({_id:{$in:publication.authors}})
```

0.7.8 Consulta 8

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).

```
db.publications.aggregate([
    /*selecciono las partes del documento que me seran utiles para consultas
        posteriores */
    {$project: {'authors': 1, 'year': 1}},

    /* para cada uno de los autores de una publicacion*/
```

```
{$unwind : '$authors' },

/* reagrupo y para cada uno obtengo*/
{$group:

   {_id: '$authors',
    ultimaFecha: { $max: '$year' },
    primeraFecha: { $min: '$year' }
    }
},

/* se almacena la diferencia de años entre las publicaciones */
{$project:
   /*{diferenciaAnios: {$subtract: ['$ultimaFecha', '$primeraFecha']}}*/
   {diferenciaAnios: {$subtract: ['$ultimaFecha', '$primeraFecha']}}
},

/* se ordenan los documentos en orden descendente */
{ $sort: { 'diferenciaAnios': -1} },

/* se toman los cinco primeros */
{ $limit : 5 },
])
```

0.7.9 Consulta 9

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([
/*selecciono las partes del documento que me seran utiles para consultas
   posteriores */
{$project: {'authors': 1, 'year': 1}},

/* para cada uno de los autores de una publicacion*/
{$unwind : '$authors' },

/* reagrupo y para cada uno obtengo*/
{$group:

   {_id: '$authors',
```

```
    ultimaFecha: { $max: '$year' },
    primeraFecha: { $min: '$year' }
    }
},

/* se almacena la diferencia de años entre las publicaciones */
{$project:
   /*{diferenciaAnios: {$subtract: ['$ultimaFecha', '$primeraFecha']}}*/
   {diferenciaAnios: {$subtract: ['$ultimaFecha', '$primeraFecha']}}
},

/* de los anteriores, se seleccionan aquellos cuya diferencia de años sea
   menor de cinco */
{$match:{'diferenciaAnios': {'$lt': 5}}}
,

/* se cuentan respecto a los que quedan del paso anterior */
{$count: "numero_autores_novatos"}
])
```

0.7.10 Consulta 10

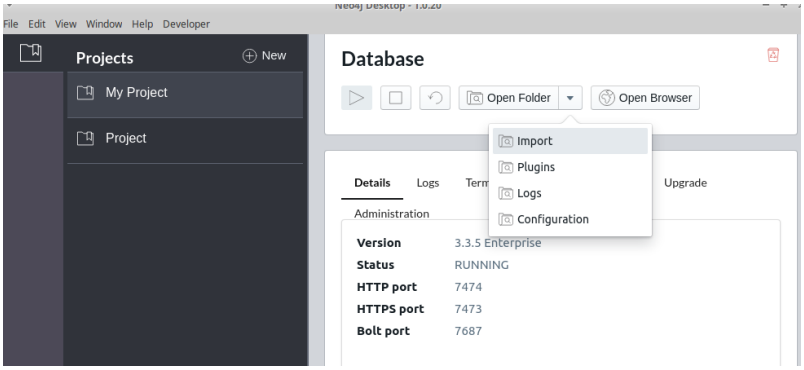
10. Porcentaje de publicaciones en revistas con respecto al total de publicaciones.

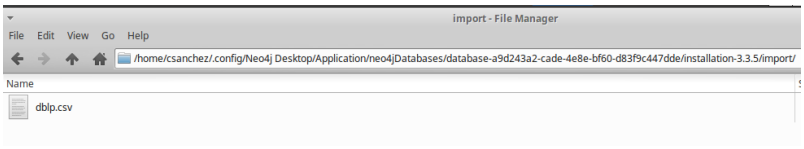
```
db.publications.aggregate([
/*selecciono las partes del documento que me seran utiles para consultas
   posteriores */

{$project: {'type': 1}},
/* para cada uno de los autores de una publicacion*/
{$unwind : '$type' },
{$group:
   {
      /* se pone el _id a nulo para que sume todas las publicaciones*/
      /* independientemente del tipo */
      _id: null,

      /* se guarda un contador parcial para los articulos */
      numero_articulos: {
         /* se realiza la operacion de suma */
         $sum: {
                /* dependiendo de la siguiente condicion */
```

```
                    $cond : [
                                /* si el tipo de publicacion es de articulo se
                                   suma, sino no */
                                {$eq : ["$type", "article"]}, 1, 0]
                            }
        },
        /* se guarda en la variable el numero de publicaciones en total */
        numero_publicaciones: { $sum: 1 }
    }
}

,
{ $project: {_id:0,
          "porcentaje": {
             /* usamos este operador para concatenar al resultado el
                signo del % */
             "$concat": [{
                                /* usamos este operador para sacar solo los
                                   dos primeros decimales */
                          "$substr": [ {
                             /* El resultado de la division, lo
                                multiplicare por 100 para ponerlo en
                                "formato" porcentaje" */
                                "$multiply": [ { "$divide": [
                                   "$numero_articulos",
                                   "$numero_publicaciones"] }, 100 ] },
                                   0,2 ] },
                          "", "%"
                         ]}
        }

    }
])
```

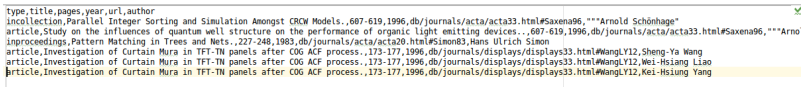
0.8 Consultas en Neo4J

Una vez generado el fichero csv a partir del fichero xml de entrada, pasamos a importar el fichero csv. Para ello, dejamos el fichero generado en la carpeta “import” de la base de datos:





Para mostrar el resultado de las consultas, se parseo un fichero xml de entrada muy simple cuyo csv, una vez procesado, tenía el siguiente contenido:



```
type,title,pages,year,url,author
incollection,Parallel Integer Sorting and Simulation Amongst CRCW Models.,607-619,1996,db/journals/acta/acta33.html#Saxena96,"""Arnold Schönhage"
article,Study on the influences of quantum well structure on the performance of organic light emitting devices.,607-619,1996,db/journals/acta/acta33.html#Saxena96,"""Arno
inproceedings,Pattern Matching in Trees and Nets.,227-248,1983,db/journals/acta/acta20.html#Simon83,Hans Ulrich Simon
article,Investigation of Curtain Mura in TFT-TN panels after COG ACF process.,173-177,1996,db/journals/displays/displays33.html#WangLY12,Sheng-Ya Wang
article,Investigation of Curtain Mura in TFT-TN panels after COG ACF process.,173-177,1996,db/journals/displays/displays33.html#WangLY12,Wei-Hsiang Liao
article,Investigation of Curtain Mura in TFT-TN panels after COG ACF process.,173-177,1996,db/journals/displays/displays33.html#WangLY12,Kei-Hsiung Yang
```

Las consultas son las siguientes: (borramos toda la base de datos y sus relaciones previas para hacer que la base de datos parta de cero)

```
MATCH (n)
WITH n LIMIT 10000
OPTIONAL MATCH (n)-[r]->()
DELETE n,r;
```

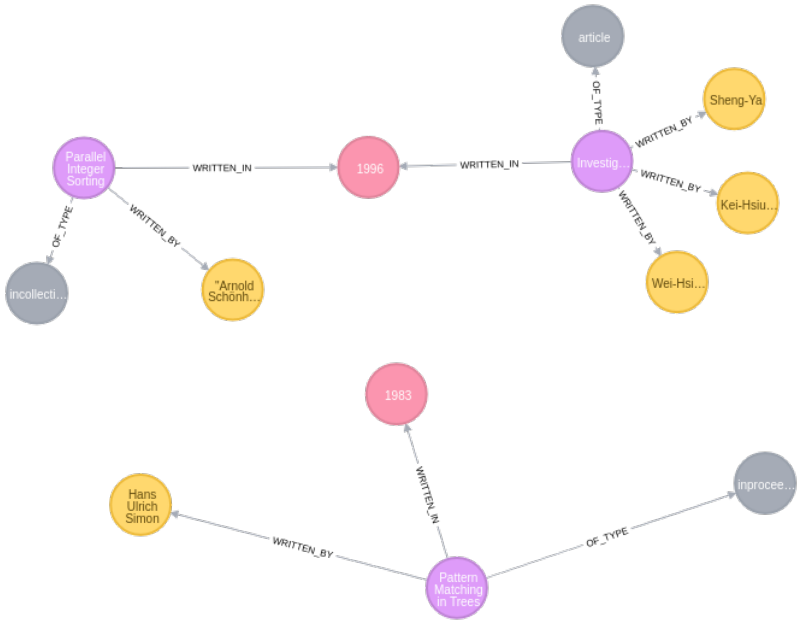
Se crean las restricciones

```
CREATE CONSTRAINT ON (author:Author) ASSERT author.author IS UNIQUE;
CREATE CONSTRAINT ON (type:Type) ASSERT type.type IS UNIQUE;
CREATE CONSTRAINT ON (publication:publication) ASSERT publication.id IS UNIQUE;
```

Se carga el fichero

```
USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM "file:///dblp.csv" AS dblp
MERGE (publication:Publication { id: dblp.title, pages: dblp.pages, url:dblp.url
    })
MERGE (year:Year {year: toInteger(dblp.year)})
MERGE (author:Author {author: dblp.author})
MERGE (type:TYPE {type: dblp.type})
MERGE (publication)-[:WRITTEN_BY]->(author)
```

```
MERGE (publication)-[:WRITTEN_IN]->(year)
MERGE (publication)-[:OF_TYPE]->(type)
```



0.8.1 Consulta 1

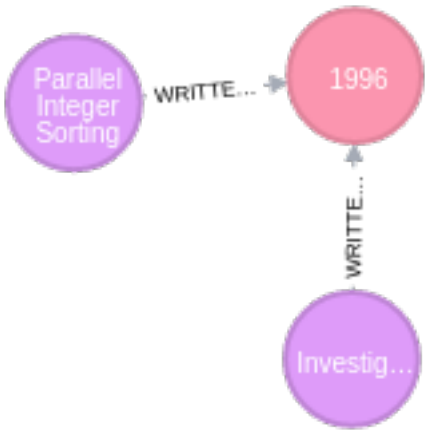
Autores de la publicación con título "Investigation of Curtain Mura in TFT-TN panels after COG ACF process."

```
MATCH (publication)-[WRITTEN_BY]->(author)
WHERE publication.id = "Investigation of Curtain Mura in TFT-TN panels after COG
    ACF process."
RETURN author, publication
```

0.8.2 Consulta 2

Publicaciones escritas en 1996:

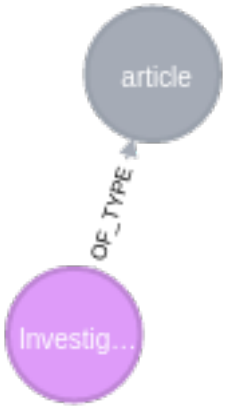
```
MATCH (publication)-[WRITTEN_IN]->(year)
WHERE year.year = 1996
RETURN publication, year
```



0.8.3 Consulta 3

Publicaciones de tipo “article” :

```
MATCH (publication)-[OF_TYPE]->(type)
WHERE type.type = "article"
RETURN publication, type
```



0.9 Problemas encontrados durante el desarrollo de la práctica

Hemos encontrado los siguientes problemas durante el desarrollo de la práctica:

0.9.1 Problema 1

Codificación de los caracteres del xml de entrada: Durante el procesado del fichero de entrada, nos dimos cuenta de que al procesar unas publicaciones se lanzaba una excepción y en otras publicaciones no ocurría. Al analizar más detenidamente el fichero de entrada, nos

dimos cuando el contenido de una etiqueta tenía caracteres como apóstrofes y acentos, entre otros. Ocurrían porque la librería lxml no procesaba bien dicho texto. Un ejemplo de una publicación que lanzaba un error, era el siguiente:

```
<article mdate="2017-05-28" key="journals/acta/Schonhage77">
  <author>Arnold Sch&ouml;nhage</author>
  <title>Schnelle Multiplikation von Polynomen &uuml;ber K&ouml;rpern der
      Charakteristik 2.</title>
  <journal>Acta Inf.</journal>
  <volume>7</volume>
  <year>1977</year>
  <pages>395-398</pages>
  <url>db/journals/acta/acta7.html#Schonhage77</url>
  <ee>https://doi.org/10.1007/BF00289470</ee>
</article>
```

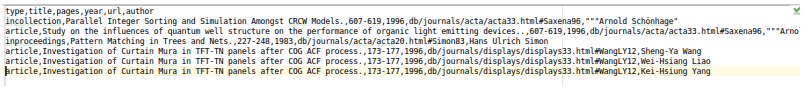
Como podemos ver, la etiqueta "author" contiene texto con caracteres especiales. Lo que nos pasaba es que se procesaba hasta el ";" y luego se lanzaba una excepción. Este error lo solucionamos poniendo el fichero DTD en el mismo directorio que el fichero XML de entrada y, posteriormente, cargando el fichero DTD del xml descargado, con la línea:

```
"""
It is necessary to load the dtd file. Otherwise, when trying to parse text containing special characters,
such as the text "Arnold Sch&ouml;nhage", an execution is thrown
"""
context = etree.iterparse(source = xml_input_path, events=("start", "end"),dtd_validation=True, load_dtd=True)
```

Como se pudo ver en puntos anteriores, esta línea está presente en los ficheros de python que procesan el xml de entrada, es decir, en *parseXMLToJSON.py* y en *parseXMLToCSV.py*

0.9.2 Problema 2

Replicación de nodos en la base de datos de Neo4J al importar el CSV:



```
type,title,pages,year,url,author
incollection,Parallel Integer Sorting and Simulation Amongst CRCW Models.,607-619,1996,db/journals/acta/acta33.html#Saxena96,"""Arnold Schönhage"
article,Study on the influences of quantum well structure on the performance of organic light emitting devices...,607-619,1996,db/journals/acta/acta33.html#Saxena96,"""Arno
inproceedings,Pattern Matching in Trees and Nets.,227-248,1983,db/journals/acta/acta20.html#Simon83,Hans Ulrich Simon
article,Investigation of Curtain Mura in TFT-TN panels after COG ACF process.,173-177,1996,db/journals/displays/displays33.html#WangLY12,Sheng-Ya Wang
article,Investigation of Curtain Mura in TFT-TN panels after COG ACF process.,173-177,1996,db/journals/displays/displays33.html#WangLY12,Wei-Hsiang Liao
article,Investigation of Curtain Mura in TFT-TN panels after COG ACF process.,173-177,1996,db/journals/displays/displays33.html#WangLY12,Kei-Hsiung Yang
```

En el fichero de entrada, como se puede ver, una de las publicaciones tiene varios autores. El problema que teníamos es que el nodo de la publicación se añadía una vez por línea del csv, por lo que había nodos duplicados en la base de datos:

La consulta con la que se creaban los nodos tras importarlos, es la siguiente:

```
USING PERIODIC COMMIT 500

LOAD CSV WITH HEADERS FROM "file:///dblp.csv" AS dblp
CREATE (publication:Publication { id: dblp.title, pages: dblp.pages,
    url:dblp.url })
CREATE (year:Year {year: toInteger(dblp.year)})
CREATE (author:Author {author: dblp.author})
CREATE (type:TYPE {type: dblp.type})
CREATE (publication)-[:WRITTEN_BY]->(author)
CREATE (publication)-[:WRITTEN_IN]->(year)
CREATE (publication)-[:OF_TYPE]->(type)
```

La solución que encontramos era simple: si en vez de usar CREATE se usaba MERGE, los nodos y relaciones no eran creadas si ya existían previamente en base de datos. Por ello, la consulta quedó así:

```
USING PERIODIC COMMIT 500

LOAD CSV WITH HEADERS FROM "file:///dblp.csv" AS dblp
MERGE (publication:Publication { id: dblp.title, pages: dblp.pages, url:dblp.url
    })
MERGE (year:Year {year: toInteger(dblp.year)})
MERGE (author:Author {author: dblp.author})
MERGE (type:TYPE {type: dblp.type})
MERGE (publication)-[:WRITTEN_BY]->(author)
MERGE (publication)-[:WRITTEN_IN]->(year)
MERGE (publication)-[:OF_TYPE]->(type)
```