# Práctica de Bases de Datos no Convencionales

May 7, 2017

## 1 Práctica: Bases de Datos No Convencionales (curso 2016-2017)

### 1.1 Autores


- Ignacio Arias Barra
- Miguel Ángel Monjas Llorente
- Raúl Sánchez Martín


### 1.2 Parte I: Mongo DB

#### 1.2.1 Captura y procesamiento de datos

**1.2.1.1 Esquema de base de datos** La base de datos DBLP Computer Science Bibliography está disponible mediante un fichero XML, en el cual se definen ocho tipos de elementos. Los más relevantes son tres: artículos de revista (`article`), artículos en congresos (`inproceedings`) y capítulos de libros (`incollection`). Son estos elementos los que se utilizarán para esta práctica. Para cada uno de dichos elementos se proporcionan: autores, título, páginas, el nombre del libro o revista, la fecha, además de otros atributos. A continuación se proporciona una muestra del fichero XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<dblp>
    <inproceedings mdate="2005-11-30" key="conf/ic/HeuerHRM00">
        <author>Andreas Heuer 0002</author>
        <author>Ernst Georg Haffner</author>
        <author>Uwe Roth</author>
        <author>Christoph Meinel</author>
        <title>
            A Hyperlink Focused Browse Assistant for the World Wide Web.
        </title>
        <pages>79-84</pages>
        <year>2000</year>
        <crossref>conf/ic/2000</crossref>
        <booktitle>International Conference on Internet Computing</booktitle>
        <url>db/conf/ic/ic2000.html#HeuerHRM00</url>
    </inproceedings>
    <incollection>
        <title>Finding Community Topics and Membership in Graphs</title>
```

```
    <author>Revelle, Matt</author>
    <author>Domeniconi, Carlotta</author>
    <author>Sweeney, Mack</author>
    <author>Johri, Aditya</author>
    <year>2015</year>
    <isbn>978-3-319-23524-0</isbn>
    <booktitle>
        Machine Learning and Knowledge Discovery in Databases
    </booktitle>
    <volume>9285</volume>
    <series>Lecture Notes in Computer Science</series>
    <doi>10.1007/978-3-319-23525-7_38</doi>
    <url>http://dx.doi.org/10.1007/978-3-319-23525-7_38</url>
    <publisher>Springer International Publishing</publisher>
    <pages>625-640</pages>
</incollection>
<article mdate="2007-10-24" key="journals/jise/Chang07">
    <title>
        New Constructions of Distance-Increasing Mappings and
        Permutation Arrays.
    </title>
    <pages>1227-1239</pages>
    <year>2007</year>
    <volume>23</volume>
    <journal>J. Inf. Sci. Eng.</journal>
    <number>4</number>
    <ee>http://www.iis.sinica.edu.tw/page/jise/2007/200707_17.html</ee>
    <url>db/journals/jise/jise23.html#Chang07</url>
</article>
<inproceedings mdate="2003-10-09" key="conf/ic/HarousD03">
    <author>Saad Harous</author>
    <author>Mahieddine Djoudi</author>
    <title>
        A Cooperative Authoring System for Intelligent Tutoring Systems.
    </title>
    <pages>723-729</pages>
    <year>2003</year>
    <crossref>conf/ic/2003-2</crossref>
    <booktitle>International Conference on Internet Computing</booktitle>
    <url>db/conf/ic/ic2003-2.html#HarousD03</url>
</inproceedings>
<article mdate="2012-03-22" key="journals/jise/LuH12">
    <author>Shyue-Kung Lu</author>
    <author>Ya-Chen Huang</author>
    <title>
        Improving Reusability of Test Symbols for Test Data Compression.
    </title>
    <pages>351-364</pages>
```

```
        <year>2012</year>
        <volume>28</volume>
        <journal>J. Inf. Sci. Eng.</journal>
        <number>2</number>
        <ee>http://www.iis.sinica.edu.tw/page/jise/2012/201203_07</ee>
        <url>db/journals/jise/jise28.html#LuH12</url>
    </article>
</dblp>
```

De toda la información proporcionada en el fichero XML, para cada ítem nos quedaremos solo con los siguientes atributos (al no ser necesarios más datos, de acuerdo a las consultas que se desea realizar): * título (`title`) * autor (`author`) * identificador del artículo (`id`) * año de publicación (`year`) * tipo de publicación (`type`) * nombre del contenedor en el que halla el artículo, conferencia o capítulo (`container`)

### 1.2.2 Procesado del dataset XML

La estrategia de procesado del dataset comienza fragmentando el fichero XML original en ficheros de menor tamaño (pero que siguen siendo ficheros válidos XLM), utilizando un programa en Python, `splitter.py`, adaptado de https://gist.github.com/benallard/8042835 y http://stackoverflow.com/questions/12309269/how-do-i-write-json-data-to-a-file-in-python/12309296 (para poder ser ejecutado en Python 2.7 y 3.5). El código de `splitter.py`, así como el del resto de programas y scripts utilizados en esta práctica, se encuentra en un repositorio GitHub (https://github.com/raul-sanchez-martin/practica_mongo).

A continuación, se parsean los ficheros resultantes para transformarlos en ficheros en formato JSON (de hecho se trata de *JSON Lines text file format*; esto es, el fichero no sigue el formato JSON, sino que es cada línea la que constituye un documento JSON válido). Se utiliza para ello el programa en Python `parser.py` (adaptado de https://github.com/songmw90/dblp-parser). Se muestra el código a continuación.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from lxml import etree
import json
import io, os

def iterate_tree(context, prefix):
    collaborations = [u'inproceedings', u'incollection', u'article']
    # xml categories
    author_array = []
    title = ''

    # read chunk line by line
    # we focus author and title
    counter = 0
    for _, elem in context:
        record = {}
```

```
record["author"] = []
if elem.tag == 'author':
    author_array.append(elem.text)

if elem.tag == 'title':
    if elem.text:
        title = elem.text

if elem.tag == 'journal' or elem.tag == 'booktitle':
    if elem.text:
        container = elem.text

if elem.tag == 'year':
    if elem.text:
        year = elem.text

if elem.tag in collaborations:
    if len(author_array) is not 0 and title is not '':
        # rejected paper has no author or title
        # it should be checked
        record["title"] = title
        record["author"] = [author for author in author_array]
        record["id"] = elem.get("key")
        record["year"] = int(year)
        record["type"] = elem.tag
        record_json = json.dumps(record,
                                 sort_keys=True,
                                 ensure_ascii=False)

        file_name = '{0}-{1:02d}.json'.format(prefix, counter)
        try:
            stats = os.stat(file_name)
            file_size = stats.st_size
        except FileNotFoundError :
            file_size = 0
        if file_size > 104857600 :
            counter += 1
            file_name = '{0}{1:02d}.json'.format(prefix, counter)
        write_element(record_json, file_name)

        title = ''
        del author_array[:]

elem.clear()
while elem.getprevious() is not None:
    try :
        del elem.getparent()[0]
    except TypeError:
```

```
                break
    del context

def write_element (elem, file_output):
    print ('.')
    with io.open(file_output, 'a', encoding='utf8') as outfile:
        outfile.write(to_unicode(elem))
        outfile.write(u"\n")

if __name__ == "__main__":
    try:
        to_unicode = unicode
    except NameError:
        to_unicode = str
    json_file_prefix = "dblp"
    input_files = ['dblp.0.xml', 'dblp.1.xml', 'dblp.2.xml']
    for input_file in input_files :
        context = etree.iterparse(input_file, load_dtd=True, html=True)
        # To use iterparse, we don't need to read all of xml.
        iterate_tree(context, json_file_prefix)
    print ('Finished')
```

Se muestra a continuación un ejemplo del resultado del proceso de parseado de los datos (cada línea tiene ajuste de línea para claridad de la visualización):

```
{"author": ["Paulo B. Ges", "Ram D. Gopal", "Nai-Kuang Chen"],
  "container": "Decision Support Systems", "id": "journals/dss/GoesGC97",
  "title": "Query evaluation management design and prototype implementation.",
  "type": "article", "year": 1997}
{"author": ["Gleiber Fernandes Royes", "Rogrio Cid Bastos"],
  "container": "Decision Support Systems", "id": "journals/dss/RoyesB06",
  "title": "Uncertainty analysis in political forecasting.",
{"author": ["Symeon Bozapalidis", "Zoltn Flp 0001", "George Rahonis"],
  "container": "Acta Inf.", "id": "journals/acta/BozapalidisFR12",
  "title": "Equational weighted tree transformations.",
  "type": "article", "year": 2012}
```

Se trata de 12 ficheros de un tamaño aproximado de 100 MB cada vez. En total 1,08 GB.

### 1.2.3 Almacenamiento de datos

Para realizar la carga de documentos a la base de datos de MongoDB, se ha creado un *script* en *bash*, `load_documents.sh`. Dicho *script* detecta todos los arhivos JSON almacenados en la carpeta en la que se ejecuta y los carga en MongoDB utilizando `mongoimport`. El código está preparado para cargar diversos archivos JSON en una misma coleción de documentos. Esta especialmente preparado para los archivos provenientes del paso anterior:

```
#!/bin/bash
```

```
# Bash script to load all the JSON files of a folder into a MongoDB
# collection. In order to use this script, set the wd of a terminal
# in the folder where the JSON files are located, and type in the
# terminal:
# ./load_documents.sh <dababase_name> <collection_name>
# Don't forget to set the load_documents.sh file as an executable

if [ "$#" -ne 2 ]; then
  echo "Invalid number of arguments. \
Please, enter <dababase_name> and <collection_name>"
  exit 1
fi

mongo $1 --eval "db.dropDatabase()"

ls -1 *.json | sed 's/.json$//' | while read col; do
    mongoimport --db $1 --collection $2 --file $col.json;
done
```

Para su ejecución, en primer lugar hay que copiar el *script* en el directorio donde se encuentran los ficheros JSON cuyo contenido hay que cargar y darle los permisos correspondientes. A continuación, se ejecuta el script `load_documents.sh` mediante el siguiente comando:

```
./load_documents.sh <nombre_base_datos> <nombre_colección>
```

donde `<nombre_base_datos>` y `<nombre_colección>` corresponden respectivamente al nombre de la base de datos (`documentos`) y de la colección (`publicaciones`) en la que queremos guardar los documentos.

Los documentos JSON que se han introducido en la base de datos MongoDB se encuentran también en el *bucket* de S3 urjc.datascience.bdnc/dbpl_convertidos

### 1.2.4 Análisis de datos

Las consultas se efectuarán utilizando el módulo de Python `pymongo`. Una vez cargados todos los documentos en la base de datos en el apartado anterior, el primer paso va a ser establecer la conexión a dicha base de datos desde `pymongo` y comprobar que todo funciona correctamente.

```
In [1]: import pymongo
        from pymongo import MongoClient
        from bson.son import SON
        from pymongo import IndexModel, ASCENDING, DESCENDING
        import pprint
        import json

        #conexion a la Base de Datos
        conex = pymongo.MongoClient('localhost', 27017)
        db = conex.documentos
        publicaciones = db.publicaciones
        print("Número de documentos: ", publicaciones.find().count())
```

```
('N\xc3\xbamero de documentos: ', 5507168)
```

Una vez que se ha comprobado que existe una conexión satisfactoria con la base de datos, se ha dado respuesta a cada una de las preguntas incluidas en el enunciado de la práctica.

Antes de efectuar las consultas, se procede a la creación de los índices necesarios para optimizar las consultas. Se han obtenido del análisis del enunciado y son los siguientes:

- `author`
- `type`
- `year`

```
In [2]: # Índices
        # Author
        index_author = IndexModel([("author", ASCENDING)])
        # Year
        index_year = IndexModel([("year", ASCENDING)])
        # Type
        index_type = IndexModel([("type", ASCENDING)])

        publicaciones.create_indexes([index_author, index_year, index_type])
```

```
Out[2]: [u'author_1', u'year_1', u'type_1']
```

#### 1.2.4.1 Listado de todas las publicaciones de un autor determinado.

En el siguiente bloque de código se incluyen las sentencias necesarias para realizar la consulta necesaria para responder a la pregunta. Para poder efectuarla, el nombre del autor se pasa con la variable `author_to_search`. En el ejemplo usamos al autor `Joost Engelfriet`.

*Para esta pregunta, el índice utilizado es* `author`.

```
In [3]: author_to_search = "Joost Engelfriet"

        print("Publicaciones del autor %s :\n" %author_to_search.upper())
        for publication in publicaciones.find({"author": author_to_search}):
            print('  %s\n' %publication)
```

```
Publicaciones del autor JOOST ENGELFRIET :

  {u'container': u'Acta Inf.', u'title': u'Context-Free Graph Languages of Bounded
  Degree are Generated by Apex Graph Grammars.', u'author': [u'Joost Engelfriet',
  u'Linda Heyker', u'George Leih'], u'year': 1994, u'_id': ObjectId('5908971142e9c4
  u'type': u'article', u'id': u'journals/acta/EngelfrietHL94'}

  {u'container': u'Acta Inf.', u'title': u'High Level Tree Transducers and Iterated
  Pushdown Tree Transducers.', u'author': [u'Joost Engelfriet', u'Heiko Vogler'],
  u'year': 1988, u'_id': ObjectId('5908971142e9c49335e1e2e3'), u'type': u'article',
  u'id': u'journals/acta/EngelfrietV88'}
```

```
 {u'container': u'Acta Inf.', u'title': u'Extended multi bottom-up tree transducer
 u'author': [u'Joost Engelfriet', u'Eric Lilin', u'Andreas Maletti'], u'year':
 2009, u'_id': ObjectId('5908971142e9c49335e1e2f0'), u'type': u'article', u'id':
 u'journals/acta/EngelfrietLM09'}

 {u'container': u'Acta Inf.', u'title': u'Context-Free Graph Grammars and Concater
 of Graphs.', u'author': [u'Joost Engelfriet', u'Jan Joris Vereijken'], u'year':
 1997, u'_id': ObjectId('5908971142e9c49335e1e302'), u'type': u'article', u'id':
 u'journals/acta/EngelfrietV97'}

 {u'container': u'Acta Inf.', u'title': u'A new natural structural congruence
 in the pi-calculus with replication.', u'author': [u'Joost Engelfriet', u'Tjallin
 Gelsema'], u'year': 2004, u'_id': ObjectId('5908971142e9c49335e1e334'), u'type':
 u'article', u'id': u'journals/acta/EngelfrietG04'}

 {u'container': u'Acta Inf.', u'title': u'Axioms for Generalized Graphs, Illustrat
 by a Cantor-Bernstein Proposition.', u'author': [u'Joost Engelfriet', u'Tjalling
 Gelsema'], u'year': 1998, u'_id': ObjectId('5908971142e9c49335e1e340'), u'type':
 u'article', u'id': u'journals/acta/EngelfrietG98'}

 {u'container': u'Acta Inf.', u'title': u'Apex Graph Grammars and Attribute
 Grammars.', u'author': [u'Joost Engelfriet', u'George Leih', u'Grzegorz Rozenberg
 u'year': 1988, u'_id': ObjectId('5908971142e9c49335e1e406'), u'type': u'article',
 u'id': u'journals/acta/EngelfrietLR88'}

 {u'container': u'Acta Inf.', u'title': u'The time complexity of typechecking
 tree-walking tree transducers.', u'author': [u'Mirko Krivnek', u'Jaroslav Morvek'
 u'Joost Engelfriet'], u'year': 2009, u'_id': ObjectId('5908971142e9c49335e1e4c5')
 u'type': u'article', u'id': u'journals/acta/Engelfriet09'}

 {u'container': u'Acta Inf.', u'title': u'Branching Processes of Petri Nets.',
 u'author': [u'Joost Engelfriet'], u'year': 1991, u'_id': ObjectId('5908971142e9c4
 u'type': u'article', u'id': u'journals/acta/Engelfriet91'}

 {u'container': u'Acta Inf.', u'title': u'Extended Linear Macro Grammars, Iteratio
 Grammars, and Register Programs.', u'author': [u'Peter R. J. Asveld', u'Joost
 Engelfriet'], u'year': 1979, u'_id': ObjectId('5908971142e9c49335e1e54c'), u'type
 u'article', u'id': u'journals/acta/AsveldE79'}

 {u'container': u'Acta Inf.', u'title': u'The Formal Power of One-Visit Attribute
 Grammars.', u'author': [u'Joost Engelfriet', u'Gilberto Fil'], u'year': 1981,
 u'_id': ObjectId('5908971142e9c49335e1e567'), u'type': u'article', u'id': u'journ

[Remaining items omitted for clarity]
```

Si solo queremos el título del artículo y el año de publicación, podemos probar lo siguiente:

```
In [4]: print("Publicaciones del autor %s :\n" %author_to_search.upper())
        query = publicaciones.find({"author": author_to_search})
        for i, publication in enumerate(query):
            print('%d. %s (%d)' %(i+1,
                                   publication[u'title'],
                                   publication[u'year']))
```

```
Publicaciones del autor JOOST ENGELFRIET :

1. Context-Free Graph Languages of Bounded Degree are Generated by Apex Graph
Grammars. (1994)
2. High Level Tree Transducers and Iterated Pushdown Tree Transducers. (1988)
3. Extended multi bottom-up tree transducers. (2009)
4. Context-Free Graph Grammars and Concatenation of Graphs. (1997)
5. A new natural structural congruence in the pi-calculus with replication.
(2004)
6. Axioms for Generalized Graphs, Illustrated by a Cantor-Bernstein Proposition.
(1998)
7. Apex Graph Grammars and Attribute Grammars. (1988)
8. The time complexity of typechecking tree-walking tree transducers. (2009)
9. Branching Processes of Petri Nets. (1991)
10. Extended Linear Macro Grammars, Iteration Grammars, and Register Programs.
(1979)
11. The Formal Power of One-Visit Attribute Grammars. (1981)
12. Context-Free Hypergraph Grammars have the Same Term-Generating Power as
Attribute Grammars. (1992)
13. Two-way pebble transducers for partial functions and their composition.
(2015)
14. A comparison of pebble tree transducers with macro tree transducers. (2003)
15. Attribute Storage Optimization by Stacks. (1990)
16. Iterated Deterministic Substitution. (1977)
17. The equivalence problem for deterministic MSO tree transducers is decidable.
(2006)
18. An Elementary Proof of Double Greibach Normal Form. (1992)
19. A Kleene characterization of computability. (2007)
20. On Tree Transducers for Partial Functions. (1978)
21. Copying Theorems. (1976)
22. An exercise in structural congruence. (2007)
23. A Note on Infinite Trees. (1972)
24. Prefix and Equality Languages of Rational Functions are Co-Context-Free.
(1988)
25. The complexity of the circularity problem for attribute grammars: a note
on a counterexample for a simpler construction. (1989)
26. Domino Treewidth. (1997)
27. A Tranlsational Theorem for the Class of EOL Languages (1981)
28. The Power to Two-Way Deterministic Checking Stack Automata (1989)
29. Equality Languages and Fixed Point Languages (1979)
30. Linear Graph Grammars: Power and Complexity (1989)
```

31. Macro Tree Transducers, Attribute Grammars, and MSO Definable Tree Translations (1999)
32. Look-Ahead on Pushdowns (1987)
33. Iterated Stack Automata and Complexity Classes (1991)
34. The generative power of delegation networks. (2015)
35. Decidability of the Finiteness of Ranges of Tree Transductions. (1998)
36. Passes and Paths of Attributive Grammars (1981)
37. Bounded Nesting in Macro Grammars (1979)
38. A Comparison of Boundary Graph Grammars and Context-Free Hypergraph Grammars (1990)
39. Derivation Trees of Ground Term Rewriting Systems. (1999)
40. Stack Machines and Classes of Nonnested Macro Languages. (1980)
41. Fixed Point Languages, Equality Languages, and Representation of Recursively Enumerable Languages. (1980)
42. Passes, sweeps, and visits in attribute grammars. (1989)
43. Three Hierarchies of Transducers. (1982)
44. Composition Closure of Linear Extended Top-down Tree Transducers. (2017)
45. Clique-Width for 4-Vertex Forbidden Subgraphs. (2006)
46. Erratum to: "Top-down Tree Transducers with Regular Look-ahead". (2016)
47. A Logical Characterization of the Sets of Hypergraphs Defined by Hyperedge Replacement Grammars. (1995)
48. Bottom-up and Top-down Tree Transformations - A Comparison. (1975)
49. Top-down Tree Transducers with Regular Look-ahead. (1977)
50. Tree Transducers, L Systems, and Two-Way Machines. (1980)
51. The Equivalence of Bottom-Up and Top-Down Tree-to-Graph Transducers. (1998)
52. Deciding equivalence of top-down XML transformations in polynomial time. (2009)
53. Handle-Rewriting Hypergraph Grammars. (1993)
54. Macro Tree Transducers. (1985)
55. Boundary Graph Grammars with Dynamic Edge Relabeling. (1990)
56. IO and OI. II. (1978)
57. Hypergraph Languages of Bounded Degree. (1994)
58. Logical Description of Contex-Free Graph Languages. (1997)
59. IO and OI. I. (1977)
60. Determinacy and rewriting of functional top-down and MSO tree transformations. (2017)
61. The Copying Power of One-State Tree Transducers. (1982)
62. Regular Description of Context-Free Graph Languages. (1996)
63. Hierarchies of Hyper-AFLs. (1985)
64. Context Free Normal Systems and ETOL Systems. (1983)
65. Branching synchronization grammars with nested tables. (2004)
66. A Comparison of Tree Transductions Defined by Monadic Second Order Logic and by Attribute Grammars. (2000)
67. Simple Multi-Visit Attribute Grammars. (1982)
68. Extended Macro Grammars and Stack Controlled Machines. (1984)
69. The Translation Power of Top-Down Tree-to-Graph Transducers. (1994)
70. Output String Languages of Compositions of Deterministic Macro Tree Transducers (2002)

71. Finite Languages for the Representation of Finite Graphs. (1996)
72. The Complexity of Regular DNLC Graph Languages. (1990)
73. The String Generating Power of Context-Free Hypergraph Grammars. (1991)
74. The non-computability of computability. (1985)
75. Book: Graph Structure and Monadic Second-Order Logic. A Language-Theoretic Approach. (2012)
76. Reverse Twin Shuffles. (1996)
77. Context-Free Grammars with Storage. (2014)
78. Look-Ahead Removal for Top-Down Tree Transducers. (2013)
79. The Equivalence Problem for Deterministic MSO Tree Transducers is Decidable (2005)
80. Tree Automata and Tree Grammars. (2015)
81. MSO definable string transductions and two-way finite state transducers (1999)
82. Automata with Nested Pebbles Capture First-Order Logic with Transitive Closure (2007)
83. Complexity of boundary graph languages. (1990)
84. MSO definable string transductions and two-way finite-state transducers. (2001)
85. Bottom-Up and Top-Down Tree Series Transformations. (2002)
86. Finitary Compositions of Two-way Finite-State Transductions. (2007)
87. Grammatical Codes of Trees and Terminally Coded Grammars. (1995)
88. Modular Tree Transducers. (1991)
89. Nonterminal Separation in Graph Grammars. (1991)
90. As Time Goes By II: More Automatic Complexity Analysis of Concurrent Rule Programs. (2001)
91. Characterization and Complexity of Uniformly Non Primitive Labeled 2-Structures (1996)
92. X-Automata on omega-Words. (1993)
93. A Regular Characterization of Graph Languages Definable in Monadic Second-Order Logic. (1991)
94. Determinacy - (Observation Equivalence = Trace Equivalence). (1985)
95. Pushdown Machines for the Macro Tree Transducer. (1986)
96. Look-ahead removal for total deterministic top-down tree transducers. (2016)
97. Surface Tree Languages and Parallel Derivation Trees. (1976)
98. Structural inclusion in the pi-calculus with replication. (2001)
99. Nonterminal Bounded NLC Graph Grammars. (1988)
100. Multisets and Structural Congruence of the pi-Calculus with Replication. (1999)
101. Iterating Iterated Substitution. (1977)
102. A Multiset Semantics for the pi-Calculus with Replication. (1996)
103. Trips on Trees. (1999)
104. Automata with Nested Pebbles Capture First-Order Logic with Transitive Closure. (2007)
105. The complexity of Languages Generated by Attribute Grammars. (1986)
106. Macro Tree Translations of Linear Size Increase are MSO Definable. (2003)
107. Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach (2012)

108. Simple Program Schemes and Formal Languages (1974)
109. The Delta Operation: From Strings to Trees to Strings. (2002)
110. Tree-Walking Pebble Automata. (1999)
111. Monadic Second Order Logic and Node Relations on Graphs and Trees. (1997)
112. Deciding the NTS Property of Context-Free Grammars. (1994)
113. Strong Lexicalization of Tree Adjoining Grammars. (2012)
114. Domino Treewith (Extended Abstract). (1994)
115. A Multiset Semantics for the pi-Calculus with Replication. (1993)
116. Regular Characterizations of Macro Tree Transducers. (1984)
117. Graph Grammars and Tree Transducers. (1994)
118. Elementary Net Systems. (1996)
119. Attribute Grammars: Attribute Evaluation Methods. (1983)
120. Net-Based Description Of Parallel Object-Based Systems, or POTs and POPs. (1990)
121. Nested Pebbles and Transitive Closure. (2006)
122. Characterizing and Deciding MSO-Definability of Macro Tree Transductions. (2000)
123. Restricting the complexity of regular DNLC languages. (1986)
124. Context-free Handle-rewriting Hypergraph Grammars. (1990)
125. Node Replacement Graph Grammars. (1997)
126. A Characterization of Context-Free NCE Graph Languages by Monadic Second-Order Logic on Trees. (1990)
127. The Term Generating Power of Context-Free Hypergraph Grammars. (1990)
128. Graph Grammars Based on Node Rewriting: An Introduction to NLC Graph Grammars. (1990)
129. Apex Graph Grammars. (1986)
130. XML transformation by tree-walking transducers with invisible pebbles. (2007)
131. Iterated Pushdown Automata and Complexity Classes (1983)
132. Tree Transducers, L Systems and Two-Way Machines (Extended Abstract) (1978)
133. Characterization of High Level Tree Transducers. (1985)
134. A Greibach Normal Form for Context-free Graph Grammars. (1992)
135. Two-Way Finite State Transducers and Monadic Second-Order Logic. (1999)
136. Passes, Sweeps and Visits. (1981)
137. Formal Properties of One-Visit and Multi-Pass Attribute Grammars. (1980)
138. Automata with Storage on Infinite Words. (1989)
139. Translation of Simple Program Schemes. (1972)
140. Macro Grammars, Lindenmayer Systems and Other Copying Devices. (1977)
141. Tree Languages Generated be Context-Free Graph Grammars. (1998)
142. Concatenation of Graphs. (1994)
143. Equality Languages, Fixed Point Languages and Representations of Recursively Enumerable Languages (1978)
144. Two-Way Finite State Transducers with Nested Pebbles. (2002)
145. Determinacy and Rewriting of Top-Down and MSO Tree Transformations. (2013)
146. The Equivalence Problem for Deterministic MSO Tree Transducers Is Decidable. (2005)
147. Branching Grammars: A Generalization of ET0L Systems. (2003)
148. Hierarchies of String Languages Generated by Deterministic Tree Transducers.

```
(2001)
149. Extended Multi Bottom-Up Tree Transducers. (2008)
150. How to Remove the Look-Ahead of Top-Down Tree Transducers. (2014)
151. Context-Free NCE Graph Grammars. (1989)
152. Clique-Width for Four-Vertex Forbidden Subgraphs. (2005)
153. Home Page (1995)
154. Equivalence - Combinatorics, Algebra, Proofs. (2016)
```

#### 1.2.4.2 Número de publicaciones de un autor determinado.

La siguiente consulta devuelve el número de publicaciones dado un autor. Para poder efectuarla, el nombre del autor se pasa a través de la variable `author_to_search`.

*Para esta pregunta, el índice utilizado es `author`.*

```
In [5]: author_to_search = "Joost Engelfriet"
        pubs_c = publicaciones.find({"author": author_to_search}).count()
        print(str(pubs_c) + " publicaciones hechas por " + author_to_search)
```

```
154 publicaciones hechas por Joost Engelfriet
```

#### 1.2.4.3 Número de artículos en revista para el año 2016.

El siguiente bloque de código incluye las sentencias necesarias para responder a la pregunta. Para poder efectuarla, el año y el tipo de documento se pasan a través de las variables `year` y `document_type` respectivamente.

*Para esta pregunta, los índices utilizados son `year` y `type`.*

```
In [6]: YEAR = 2016
        document_type = "article" # los artículos en revista son de este tipo

        query = {"$and":[{"year": YEAR}, {"type": document_type}]}
        publications_count = publicaciones.find(query).count()

        print("Número de artículos en revista publicados en %d: %d"
              %(YEAR, publications_count))
```

```
Número de artículos en revista publicados en 2016: 126954
```

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

El siguiente código da como resultado el número de autores con menos de 5 publicaciones. El número mínimo se pasa a través de la variable `min_pubs`.

*Para este pregunta, el índice utilizado es `author`*

```
In [7]: min_pubs = 5
        pl = [{"$unwind": "$author"},
              {"$sortByCount": "$author"},
```

```
            {"$match": {'count':{"$lt": min_pubs}}},
            {"$group": {'_id': 'null', 'count': { "$sum": 1 } } }]
        count = list(publicaciones.aggregate(pl, allowDiskUse=True))[0][u'count']
        print("Hay %d autores con menos de %d publicaciones" %(count, min_pubs))
```

```
Hay 1399949 autores con menos de 5 publicaciones
```

Si queremos obtener el listado de todos los autores (no recomendado, por el tiempo de ejecución), se puede probar el siguiente código:

```
min_pubs = 5
pl = [{"$unwind": "$author"},
            {"$sortByCount": "$author"},
            {"$match": {'count':{"$lt": min_pubs}}}]
for au in publicaciones.aggregate(pl, allowDiskUse=True):
    print ("El autor %s tiene %d publicaciones."
            %(str(au['_id']), au['count']))
```

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

A continuación se incluye el código que proporciona la información requerida.

*Para esta pregunta, el índice utilizado es* `author`.

```
In [8]: from bson.son import SON

        pl = [{"$unwind":"$author"},
              {"$group":{"_id":"$author",
                         "art_congreso":
                         {"$push":{"$cond":[{"$eq":["$type","inproceedings"]}, 1, 0
                         "type":{"$push":"$type"}
                        }
              },
              {"$addFields":{"art_revista":{"$sum":"$art_revista"},
                             "art_congreso":{"$sum":"$art_congreso"}}
              },
              {"$addFields":{"art_totales":
                             {"$add":["$art_revista", "$art_congreso"]}}
              },
              {"$sort":{"art_totales":-1}},
              {"$project":{"art_totales":1, "_id":1,
                           "art_revista":1, "art_congreso":1}
              },
              {"$limit":10}
             ]

        for individual_publi_data in publicaciones.aggregate(pl, allowDiskUse=True)
            print (
```

```
                '%s ha publicado %d veces en revistas e intervenido \
        %d veces en congresos (total: %d)'
                  %(individual_publi_data['_id'],
                    individual_publi_data['art_revista'],
                    individual_publi_data['art_congreso'],
                    individual_publi_data['art_totales']))
```

```
Wen Gao 0001 ha publicado 0 veces en revistas e intervenido 715 veces en
congresos (total: 715)
Philip S. Yu ha publicado 0 veces en revistas e intervenido 652 veces en
congresos (total: 652)
Wei Zhang ha publicado 0 veces en revistas e intervenido 650 veces en
congresos (total: 650)
Thomas S. Huang ha publicado 0 veces en revistas e intervenido 607 veces
en congresos (total: 607)
Hai Jin ha publicado 0 veces en revistas e intervenido 570 veces en
congresos (total: 570)
Yang Liu ha publicado 0 veces en revistas e intervenido 565 veces en
congresos (total: 565)
Yu Zhang ha publicado 0 veces en revistas e intervenido 556 veces en
congresos (total: 556)
Jiawei Han ha publicado 0 veces en revistas e intervenido 549 veces en
congresos (total: 549)
Jing Li ha publicado 0 veces en revistas e intervenido 543 veces en
congresos (total: 543)
Makoto Takizawa ha publicado 0 veces en revistas e intervenido 543 veces en
congresos (total: 543)
```

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

En este apartado se han implementado dos tipos de consultas. La primera extrae la media de autores por tipo de publicación. La segunda devuelve como resultado el número medio de autores teniendo en cuenta todas las publicaciones de la base de datos.

*Para estas consultas, el índice utilizado es* `type`

```
In [9]: pl = [{"$group":
                {"_id":"$type",
                 "authorMean":{"$avg":{"$size":{ "$ifNull": [ "$author", [] ] }}}
                }
               }]
        au_mean_pars = [x for x in publicaciones.aggregate(pl) if x['authorMean'] !
        sorted_au_mean_pars = sorted(au_mean_pars,
                                     key=lambda k: k['authorMean'],
                                     reverse=True)
        for au_mean_par in sorted_au_mean_pars:
            print("La cantidad media de autores para el \
        tipo de publicación '%s' es de %.2f"
```

```
                %(str(au_mean_par['_id']), au_mean_par['authorMean']))

        pl = [{"$group":
               {"_id":"$type",
                "auMean_perPub":{"$avg":{"$size":{ "$ifNull": [ "$author", [] ] }}}}},
              {"$group":{"_id":"null","authorMean_tot":{"$avg":"$auMean_perPub"}}}
             ]
        for au_mean_tot in publicaciones.aggregate(pl):
            print("\nLa media de autores es de %.2f" %au_mean_tot['authorMean_tot'
```

```
La cantidad media de autores para 'proceedings' es de 3.11
La cantidad media de autores para 'inproceedings' es de 3.04
La cantidad media de autores para 'article' es de 2.77
La cantidad media de autores para 'incollection' es de 2.19
La cantidad media de autores para 'book' es de 1.65
La cantidad media de autores para 'www' es de 1.02
La cantidad media de autores para 'phdthesis' es de 1.00
La cantidad media de autores para 'mastersthesis' es de 1.00

La media de autores es de 1.75
```

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

A continuación se incluye el código necesario para responder a la pregunga. El nombre del autor objeto de la consulta se pasa en la variable `author_to_search`. Sigue siendo nuestro amigo *Joost Engelfriet*.

*Para esta pregunta, el índice utilizado es* `author`.

```
In [10]: target = "Joost Engelfriet"

         author_publications = publicaciones.find({"author":target})
         authors_list = [publication['author'] for publication in author_publicatio
         coauthors_list = \
             set(
                 [item for sublist in authors_list for item in sublist if item!=tar
             )

         print("Lista de coautores del autor %s:" %author_to_search)
         for i, author in enumerate(coauthors_list):
             print ('%d. %s' %(i+1, author))
```

```
Lista de coautores del autor Joost Engelfriet:
1. Vincent van Oostrom
2. Bart Samwel
3. Linda Heyker
4. Giora Slutzki
5. Andreas Maletti
```

```
6. Jan Joris Vereijken
7. Frank Drewes
8. Tjalling Gelsema
9. Hong-Oanh Le
10. Erik Meineche Schmidt
11. Emo Welzl
12. Thom W. Frhwirth
13. Tero Harju
14. Vadim V. Lozin
15. Andrzej Ehrenfeucht
16. Paulien ten Pas
17. Eric Lilin
18. Michael Benedikt
19. Gilberto Fil
20. George Leih
21. Heiko Vogler
22. Sven Skyum
23. Hendrik Jan Hoogeboom
24. Mirko Krivnek
25. Jaroslav Morvek
26. Jan van Leeuwen
27. Grzegorz Rozenberg
28. Willem de Jong
29. Hans L. Bodlaender
30. Bruno Courcelle
31. Jan-Pascal van Best
32. IJsbrand Jan Aalbersberg
33. Peter R. J. Asveld
34. Roderick Bloem
35. Gregor Kemper
36. Andrzej Proskurowski
37. Sebastian Maneth
38. Zoltn Flp 0001
39. Andreas Brandstdt
40. Helmut Seidl
```

#### 1.2.4.8 Edad de los cinco 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).

El número de autores que queremos obtener se pasa en la variable `num_auth`.

*Para esta pregunta, el índice utilizado es* `author`.

```
In [11]: num_auth = 5

         pl = [{"$unwind":"$author"},
               {"$group":
                    {"_id":"$author",
```

```
                    "minYear":{"$min":"$year"},
                    "maxYear":{"$max":"$year"}}},
                {"$addFields":{"edad":{"$subtract":["$maxYear","$minYear"]}}},
                {"$sort":{"edad":-1}},
                {"$limit": num_auth}
               ]

        for aut_age in publicaciones.aggregate(pl, allowDiskUse=True):
            print("La edad de %s es de %d años"
                  %(str(aut_age['_id']), aut_age["edad"]))
```

```
La edad de Alan M. Turing es de 75 años
La edad de Rudolf Carnap es de 71 años
La edad de David Nelson es de 67 años
La edad de Eric Weiss es de 64 años
La edad de Claude E. Shannon es de 63 años
```

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

La edad máxima se pasa en la variable `edad`.

*Índice utilizado para esta pregunta:* `author`.

```
In [12]: edad = 5

         pl = [{"$unwind":"$author"},
               {"$group":{"_id":"$author",
                          "year":{"$push":"$year"}}},
               {"$addFields":{"max_year":{"$max":"$year"},
                              "min_year":{"$min":"$year"}}},
               {"$addFields":{"edad":{"$subtract":["$max_year", "$min_year"]}}},
               {"$project":{"_id":0,"autor":"$_id","edad":1}},
               {"$match":{"edad":{"$lt":edad}}},
               {"$sort":{"edad":-1}},
               {"$count":"número_autores"}
              ]

         data = [i for i in publicaciones.aggregate(pl, allowDiskUse=True)]

         print("Número de autores con una edad menor de %d años: %d"
               %(edad, data[0][u'número_autores']))
```

```
Número de autores con una edad menor de 5 años: 114067
```

#### 1.2.4.10 Porcentaje de publicaciones en revistas con respecto al total de publicaciones.

En este apartado se obteniene el porcentaje de artículos publicados en revistas respecto al total de publicaciones. El tipo de publicación se pasa en la variable `type` (`article`).

*El índice utilizado es `type`.*

```
In [13]: type = 'article'
         total_docs = publicaciones.find().count()

         pl = [{"$group":{"_id":"$type","count":{"$sum":1}}},
               {"$project":{"count":1,"per":
                            {"$multiply":[{"$divide":[100,total_docs]},"$count"]}
                           }},
               {"$match": {'_id': type}}
              ]

         data = [i for i in publicaciones.aggregate(pl)]

         if len(data) > 0 :
             print(
                 "Hay un %.2f%% de publicaciones de tipo %s \
         respecto al total de publicaciones"
                 %(data[0][u'per'], type))
         else :
             print("No hay publicaciones de tipo '%s'" %(type))
```

```
Hay un 28.82% de publicaciones de tipo article respecto al total de publicaciones
```

## 1.3 Parte II: Neo4j

### 1.3.1 Captura y procesamiento de datos

El procedimiento adoptado en este caso será similar al relativo a MongoDB. La única diferencia será que, en vez de crearse ficheros JSON, se trabajará con ficheros CSV. A priori, existen dos estrategias diferentes:

- volver a procesar el fichero XML para generar el o los ficheros CSV que luego serán importados desde Neo4j. Una estrategia similar consistiría en hacer un procesado parecido pero desde los ficheros JSON obtenidos en el primer procesado. El uso de JSON en lugar de XML simplificaría notablemente el proceso.
- utilizar el contenido de MongoDB, exportando uno o varios ficheros CSV con los datos necesarios para poder ser cargados posteriormente en Neo4j. Para ello, es posible que haya que generar alguna colección adicional.

En cualquier caso, es necesario determinar cuál será el modelo de datos que utilizaremos. Tratándose de una base de datos orientada a grafos parece evidente considerar los siguientes nodos: `Publication` y `Author`. Y una relación, `HAS_PUBLISHED`. Aunque tenemos también la información sobre el medio en el que se han publicado, lo que habilitaría un modelado en el que hubiese dos nodos (`Author` y `Container`), con `Publication` como arista, hemos preferido un enfoque más simple.

A efectos ilustrativos proporcionamos un ejemplo de un registro extraido de los ficheros JSON y una sentencia en lenguaje Cypher que permitiría crear los nodos y aristas correspondientes:

```
{"author": ["Sanjeev Saxena"], "container": "Acta Inf.",
"id": "journals/acta/Saxena96",
"title": "Parallel Integer Sorting and Simulation Amongst CRCW Models.",
"type": "article", "year": 1996}
```

Una propuesta de sentencia Cypher sería la siguiente:

```
CREATE (journals_acta_Saxena96:Publication
    {container: "Acta Inf.",
    title: "Parallel Integer Sorting and Simulation Amongst CRCW Models",
    type: "article"}
)
CREATE (Sanjeev_Saxena:Author {name: 'Sanjeev Saxena'})
CREATE (Sanjeev_Saxena)-[:HAS_PUBLISHED {year:1996}]-> (journals_acta_Saxena96)
```

Puede observarse que sugerimos la utilización como nombre de los nodos del tipo `Publication` el `id` único de la publicación, sustituyendo las barras por guiones bajos. El nombre de los nodos de tipo `Author` será el nombre del autor sustituyendo los espacios por guiones bajos (todo ello debido a que no es posible incluir espacios dentro de los nombres de los nodos o aristas de Neo4j).

El reto ahora se encuentra en preparar los datos para que puedan ser importados desde Neo4j.

Comenzamos por los autores. El fichero CSV con los datos de los autores puede prepararse a partir de una consulta de MongoDB que cree una colección (`authors`) con los autores únicos presentes en la base de datos (suponiendo que nos hemos conectado a la base de datos `documentos`):

```
db.publicaciones.aggregate(
[
    {$unwind : "$author" },
    {$group : {_id : "$author", "name" : { $first: "$author" }}},
    {$out: "authors"}
],
{
    allowDiskUse:true
})
```

Verificamos la creación correcta de la colección averiguando el número de autores:

```
db.authors.count()
```

Se trata de 1899746 autores.

Para crear un fichero CSV con los datos de las publicaciones, ejecutaremos la siguiente orden, con lo que obtendremos la colección `publications`:

```
db.publicaciones.aggregate( [
    {$project:{_id: 0, "container":1,
        "publication_id":"$id",
```

```
        "title":1, "type":1,
        "year":1}},
    {$out: "publications"}
] )
```

Verificamos el número de publicaciones:

```
db.publications.count()
```

Como esperábamos, obtenemos 5507168 publicaciones.

Finalmente creamos un fichero CSV con las relaciones entre autores y publicaciones. Para ello ejecutamos la siguiente orden, con la que obtendremos la colección `relationships`:

```
db.publicaciones.aggregate( [
    {$unwind : "$author" },
    {$project:{_id: 0, "name":"$author", "publication_id":"$id"}},
    {$out: "relationships"}
])
```

En este caso, obtenemos 12335676 relaciones.

Una vez que tenemos las nuevas colecciones creadas, hemos abordado la creación de los ficheros CSV utilizando `mongoexport`, de la siguiente forma:

```
mongoexport -d publicaciones -c authors \
    --type=csv --fields name --out authors.csv
mongoexport -d publicaciones -c publications \
    --type=csv --fields container,publication_id,title,type,year \
    --out publications.csv
mongoexport -d publicaciones -c relationships \
    --type=csv --fields name,publication_id \
    --out relationships.csv
```

Sin embargo, hemos comprobado que el resultado no es demasiado bueno. El problema fundamental es que `mongoexport` no permite la personalización del carácter de separación, ya que la coma es el carácter utilizado por defecto. Este carácter colisiona con el uso de comas, por ejemplo, en el título de los artículos, con lo que a la hora de importar los ficheros, aparecía un número variable, y erróneo de campos en cada registro. Por ello, finalmente hemos tenido que utilizar otro enfoque, basado en Python, de forma que pudiéramos, por una parte, utilizar el tabulador como carácter de separación y, por otra parte, controlásemos el uso de comillas.

Un esquema del *script* utilizado, `export_db.py`, lo mostramos a continuación:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import codecs
from pymongo import MongoClient
import sys

print("Dumping authors.csv")
```

```
file = codecs.open("authors.csv", "w", "utf-8")
# CSV header for 'Author' label
file.write (":ID\tname\n")
conex = MongoClient('localhost', 27017)
db = conex.docos
authors = db['authors']
cursor = authors.find({})
for doc in cursor:
    if "'" in doc['name'] :
        file.write('"' + doc['_id'] + '"\t"' + doc['name'] + '"\n')
    else :
        file.write(doc['_id'] + '\t' + doc['name']+ '\n')

file.close()
print("\nDumped authors.csv\n")

print("Dumping publications.csv")
file = codecs.open("publications.csv", "w", "utf-8")
# CSV header for 'Publication' label
file.write(":ID\tcontainer\ttitle\ttype\tyear\n")
conex = MongoClient('localhost', 27017)
db = conex.docos
publications = db['publications']
cursor = publications.find({})
discards = []
for doc in cursor:
    try:
        if "'" in doc['title']:
            file.write(doc['publication_id'] + '\t' +
                       doc['container'].replace("'", "").replace('"', '') +
                       '\t"' + doc['title'].replace('"', '') + '"\t' +
                       doc['type'] + '\t' +
                       str(doc['year']) + '\n')
        else:
            file.write(doc['publication_id'] + '\t' +
                       doc['container'].replace("'", "").replace('"', '') +
                       '\t' +  doc['title'].replace('"', '') + '\t' +
                       doc['type'] + '\t' +
                       str(doc['year']) + '\n')
    except Exception as e:
        if 'publication_id' in doc:
            discards.append(doc['publication_id'])

file.close()
print("\nDumped publications.csv\n")

print("Dumping relationships.csv")
file = codecs.open("relationships.csv", "w", "utf-8")
```

```
# CSV header for 'HAS_PUBLISHED' label
file.write (":START_ID\t:END_ID\n")
conex = MongoClient('localhost', 27017)
db = conex.docos
relationships = db['relationships']
cursor = relationships.find({})
for doc in cursor:
    if doc['publication_id'] not in discards:
        if "'" in doc['name'] :
            file.write('"' + doc['name'] + '"\t' +
                       doc['publication_id'] + '\n')
        else :
            file.write(doc['name'] + '\t' +
                       doc['publication_id']+ '\n')

file.close()
print("\nDumped relationships.csv")
```

El resultado serán tres ficheros CSV con los contenidos de los autores, las publicaciones y las relaciones respectivamente. Los hemos subido a AWS S3, donde se encuentran en las siguientes URL: https://s3-eu-west-1.amazonaws.com/urjc.datascience.bdnc/authors.csv, https://s3-eu-west-1.amazonaws.com/urjc.datascience.bdnc/publications.csv y https://s3-eu-west-1.amazonaws.com/urjc.datascience.bdnc/relationships.csv.

### 1.3.2 Almacenamiento de datos

La carga de los datos anteriores en Neo4J ha sido, sin duda, el aspecto más complicado de la presente práctica. Hemos hecho infinidad de pruebas que se han traducido en tiempos de carga inasumibles. Hemos probado a utilizar las funcionalidades de `LOAD CSV` desde el Neo4J browser, o a cargarlos con Python mediante la librería `py2neo`, utilizando enfoques mono y multiprocesos. Finalmente, el enfoque adecuado (y extremadamente rápido) ha sido el basado en la herramienta `neo4j-import` https://neo4j.com/developer/guide-import-csv/#_super_fast_batch_importer_for_huge_datasets.

Si se ha prestado atención al código presentado en la sección anterior, se podrá observar el formato de las cabeceras de cada uno de los ficheros CSV:

1. Para los autores:

   ```
   :ID     name
   ```

2. Para las publicaciones:

   ```
   :ID     container   title   type    year
   ```

3. Para las aristas entre autores y publicaciones:

   ```
   :START_ID   :END_ID
   ```

Los nombres de campo `:ID`, `:START_ID` y `:END_ID` son los que determinan cuáles son los comienzos y finales de las aristas y cuáles son los identificadores de cada nodo. Una vez fijados los campos, se ejecuta `neo4j-import`:

```
sudo /usr/bin/neo4j-import --into /var/lib/neo4j/data/databases/dblp.db \
--id-type string \
--nodes:Author authors.csv --nodes:Publication publications.csv \
--relationships:HAS_PUBLISHED relationships.csv \
--quote '"' --delimiter "TAB"
```

En donde pueden verse las etiquetas (`label`) de nodos y aristas, la localización de la base de datos y los ficheros CSV de partida.

A continuación hay que hacer activa la nueva base de datos (`/var/lib/neo4j/data/databases/dblp.db`), para lo que se fija la siguiente opción (`dbms.active_database=dblp.db`) en la configuración de Nej4J (`sudo nano /etc/neo4j/neo4j.conf`). A continuación se reinicia la base de datos:

```
sudo neo4j restart
```

El resultado ha sido muy satisfactorio:

```
IMPORT DONE in 1m 39s 79ms.
Imported:
  7389030 nodes
  12335676 relationships
  23856873 properties
Peak memory usage: 195.93 MB
```

### 1.3.3 Análisis de datos

Una vez que el grafo está cargado, se pueden realizar diferentes consultas. Las primeras van a tener el objetivo fundamental de comprobar que la carga de los datos y la posterior construcción del grafo ha sido correcta, mientras que la últimas van a ser consultas que presentan grandes ventajas al ser realizadas en una base orientada a grafos en comparación con otro tipo de bases de datos como MongoDB. Mostraremos, por una parte, los resultados obtenidos a través del Neo4J Browser y, por otra, los obtenidos programáticamente, a través del paquete `py2neo`.

```
In [14]: from py2neo import Graph

         graph = Graph("http://localhost:7474/db/data/")
```

En primer lugar, se va a comprobar el número de publicaciones, que en teoría ha de ser 5.507.168 (obtenidas en la sección relativa a MongoDB);

#### 1.3.3.1 Número de publicaciones

```
MATCH (n:Publication)
RETURN count(n)
```

**Resultado**: obtenemos un total de 5.489.284 publicaciones. Es decir, varios miles menos de las esperadas. La razón es que el proceso de carga ha omitido aquellas publicaciones cuyo título poseía caracteres no Latin-1, que proporcionaban problemas de carga. Esto no sería aceptable en un entorno de producción, pero lo hemos considerado aceptable dentro de un entorno de pruebas y aprendizaje.

```
In [15]: data = graph.data("MATCH (n:Publication) RETURN count(n)")
         data[0]["count(n)"]

Out[15]: 5489284
```

Ahora comprobamos el número de autores totales

#### 1.3.3.2 Número de autores

```
MATCH (n:Author)
RETURN count(n)
```

**Resultado**: obtenemos un total de 189.9746 autores.

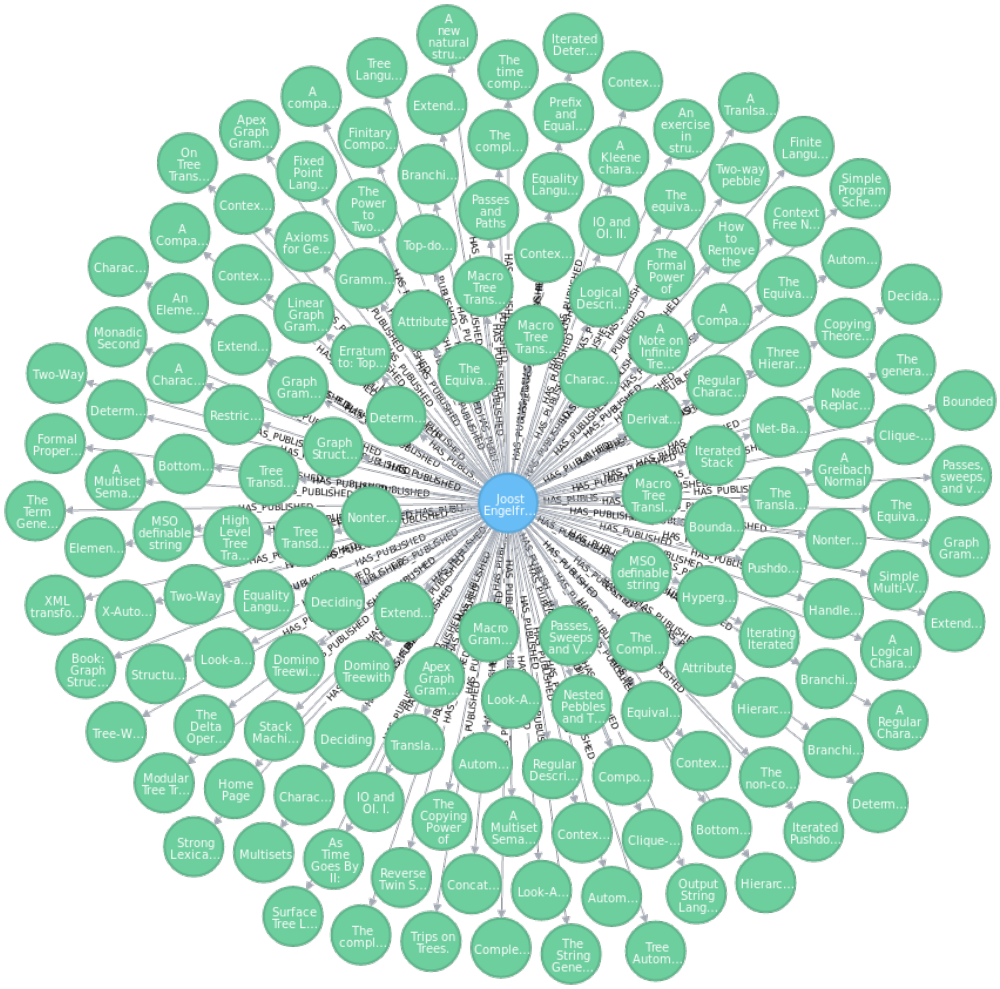
```
In [16]: data = graph.data("MATCH (n:Author) RETURN count(n)")
        data[0]["count(n)"]

Out[16]: 1899746
```

#### 1.3.3.3 Publicaciones de nuestro viejo amigo, el autor `Joost Engelfriet`.

Utilizamos tres enfoques diferentes:

```
MATCH (author)-[:HAS_PUBLISHED]->(pub)
WHERE author.name = "Joost Engelfriet"
RETURN (author)-[:HAS_PUBLISHED]->(pub)
```

**Resultado**: un grafo donde el autor `Joost Engelfriet` está unido a 154 publicaciones, como ya comprobamos en MongoDB. Véase el grafo:




```
MATCH (author)-[:HAS_PUBLISHED]->(pub)
WHERE author.name = "Joost Engelfriet"
RETURN pub.title
```

**Resultado**: obtenemos un listado de 154 publicaciones.

```
MATCH (author)-[:HAS_PUBLISHED]->(pub)
WHERE author.name = "Joost Engelfriet"
RETURN count(pub)
```

**Resultado**: aplicando esta consulta, obtenemos un número total de publicaciones del autor `Joost Engelfriet` de 154, lo que coincide con las consultas anteriores.

```
In [17]: data = graph.data('MATCH (author)-[:HAS_PUBLISHED]->(pub)'
                           'WHERE author.name = "Joost Engelfriet"'
                           'RETURN pub.title')
         for i, title in enumerate(data) :
             print ('%d. %s' %(i+1, title[u'pub.title']))

1. Equivalence - Combinatorics, Algebra, Proofs.
2. Home Page
```

3. Clique-Width for Four-Vertex Forbidden Subgraphs.
4. Context-Free NCE Graph Grammars.
5. How to Remove the Look-Ahead of Top-Down Tree Transducers.
6. Extended Multi Bottom-Up Tree Transducers.
7. Hierarchies of String Languages Generated by Deterministic Tree Transducers.
8. Branching Grammars: A Generalization of ET0L Systems.
9. The Equivalence Problem for Deterministic MSO Tree Transducers Is Decidable.
10. Determinacy and Rewriting of Top-Down and MSO Tree Transformations.
11. Two-Way Finite State Transducers with Nested Pebbles.
12. Equality Languages, Fixed Point Languages and Representations of Recursively Enumerable Languages
13. Concatenation of Graphs.
14. Tree Languages Generated be Context-Free Graph Grammars.
15. Macro Grammars, Lindenmayer Systems and Other Copying Devices.
16. Translation of Simple Program Schemes.
17. Automata with Storage on Infinite Words.
18. Formal Properties of One-Visit and Multi-Pass Attribute Grammars.
19. Passes, Sweeps and Visits.
20. Two-Way Finite State Transducers and Monadic Second-Order Logic.
21. A Greibach Normal Form for Context-free Graph Grammars.
22. Characterization of High Level Tree Transducers.
23. Tree Transducers, L Systems and Two-Way Machines (Extended Abstract)
24. Iterated Pushdown Automata and Complexity Classes
25. XML transformation by tree-walking transducers with invisible pebbles.
26. Apex Graph Grammars.
27. Graph Grammars Based on Node Rewriting: An Introduction to NLC Graph Grammars.
28. The Term Generating Power of Context-Free Hypergraph Grammars.
29. A Characterization of Context-Free NCE Graph Languages by Monadic Second-Order Logic on Trees.
30. Node Replacement Graph Grammars.
31. Context-free Handle-rewriting Hypergraph Grammars.
32. Restricting the complexity of regular DNLC languages.
33. Characterizing and Deciding MSO-Definability of Macro Tree Transductions.
34. Nested Pebbles and Transitive Closure.
35. Net-Based Description Of Parallel Object-Based Systems, or POTs and POPs.
36. Attribute Grammars: Attribute Evaluation Methods.
37. Elementary Net Systems.
38. Graph Grammars and Tree Transducers.
39. Regular Characterizations of Macro Tree Transducers.
40. A Multiset Semantics for the pi-Calculus with Replication.
41. Domino Treewith (Extended Abstract).
42. Strong Lexicalization of Tree Adjoining Grammars.
43. Deciding the NTS Property of Context-Free Grammars.
44. Monadic Second Order Logic and Node Relations on Graphs and Trees.
45. Tree-Walking Pebble Automata.
46. The Delta Operation: From Strings to Trees to Strings.
47. Simple Program Schemes and Formal Languages
48. Graph Structure and Monadic Second-Order Logic - A Language-Theoretic

Approach.
49. Macro Tree Translations of Linear Size Increase are MSO Definable.
50. The complexity of Languages Generated by Attribute Grammars.
51. Automata with Nested Pebbles Capture First-Order Logic with Transitive Closure.
52. Trips on Trees.
53. A Multiset Semantics for the pi-Calculus with Replication.
54. Iterating Iterated Substitution.
55. Multisets and Structural Congruence of the pi-Calculus with Replication.
56. Nonterminal Bounded NLC Graph Grammars.
57. Structural inclusion in the pi-calculus with replication.
58. Surface Tree Languages and Parallel Derivation Trees.
59. Look-ahead removal for total deterministic top-down tree transducers.
60. Pushdown Machines for the Macro Tree Transducer.
61. Determinacy - (Observation Equivalence = Trace Equivalence).
62. A Regular Characterization of Graph Languages Definable in Monadic Second-Order Logic.
63. X-Automata on omega-Words.
64. Characterization and Complexity of Uniformly Non Primitive Labeled 2-Structures
65. As Time Goes By II: More Automatic Complexity Analysis of Concurrent Rule Programs.
66. Nonterminal Separation in Graph Grammars.
67. Modular Tree Transducers.
68. Grammatical Codes of Trees and Terminally Coded Grammars.
69. Finitary Compositions of Two-way Finite-State Transductions.
70. Bottom-Up and Top-Down Tree Series Transformations.
71. MSO definable string transductions and two-way finite-state transducers.
72. Complexity of boundary graph languages.
73. Automata with Nested Pebbles Capture First-Order Logic with Transitive Closure
74. MSO definable string transductions and two-way finite state transducers
75. Tree Automata and Tree Grammars.
76. The Equivalence Problem for Deterministic MSO Tree Transducers is Decidable
77. Look-Ahead Removal for Top-Down Tree Transducers.
78. Context-Free Grammars with Storage.
79. Reverse Twin Shuffles.
80. Book: Graph Structure and Monadic Second-Order Logic. A Language-Theoretic Approach.
81. The non-computability of computability.
82. The String Generating Power of Context-Free Hypergraph Grammars.
83. The Complexity of Regular DNLC Graph Languages.
84. Finite Languages for the Representation of Finite Graphs.
85. Output String Languages of Compositions of Deterministic Macro Tree Transducers
86. The Translation Power of Top-Down Tree-to-Graph Transducers.
87. Extended Macro Grammars and Stack Controlled Machines.
88. Simple Multi-Visit Attribute Grammars.
89. A Comparison of Tree Transductions Defined by Monadic Second Order Logic and by Attribute Grammars.

90. Branching synchronization grammars with nested tables.
91. Context Free Normal Systems and ETOL Systems.
92. Hierarchies of Hyper-AFLs.
93. Regular Description of Context-Free Graph Languages.
94. The Copying Power of One-State Tree Transducers.
95. Determinacy and rewriting of functional top-down and MSO tree transformations.
96. IO and OI. I.
97. Logical Description of Contex-Free Graph Languages.
98. Hypergraph Languages of Bounded Degree.
99. IO and OI. II.
100. Boundary Graph Grammars with Dynamic Edge Relabeling.
101. Macro Tree Transducers.
102. Handle-Rewriting Hypergraph Grammars.
103. Deciding equivalence of top-down XML transformations in polynomial time.
104. The Equivalence of Bottom-Up and Top-Down Tree-to-Graph Transducers.
105. Tree Transducers, L Systems, and Two-Way Machines.
106. Top-down Tree Transducers with Regular Look-ahead.
107. Bottom-up and Top-down Tree Transformations - A Comparison.
108. A Logical Characterization of the Sets of Hypergraphs Defined by Hyperedge Replacement Grammars.
109. Erratum to: Top-down Tree Transducers with Regular Look-ahead.
110. Clique-Width for 4-Vertex Forbidden Subgraphs.
111. Composition Closure of Linear Extended Top-down Tree Transducers.
112. Three Hierarchies of Transducers.
113. Passes, sweeps, and visits in attribute grammars.
114. Fixed Point Languages, Equality Languages, and Representation of Recursively Enumerable Languages.
115. Stack Machines and Classes of Nonnested Macro Languages.
116. Derivation Trees of Ground Term Rewriting Systems.
117. A Comparison of Boundary Graph Grammars and Context-Free Hypergraph Grammars
118. Bounded Nesting in Macro Grammars
119. Passes and Paths of Attributive Grammars
120. Decidability of the Finiteness of Ranges of Tree Transductions.
121. The generative power of delegation networks.
122. Iterated Stack Automata and Complexity Classes
123. Look-Ahead on Pushdowns
124. Macro Tree Transducers, Attribute Grammars, and MSO Definable Tree Translation
125. Linear Graph Grammars: Power and Complexity
126. Equality Languages and Fixed Point Languages
127. The Power to Two-Way Deterministic Checking Stack Automata
128. A Tranlsational Theorem for the Class of EOL Languages
129. Domino Treewidth.
130. The complexity of the circularity problem for attribute grammars: a note on a counterexample for a simpler construction.
131. Prefix and Equality Languages of Rational Functions are Co-Context-Free.
132. A Note on Infinite Trees.
133. An exercise in structural congruence.
134. Copying Theorems.

```
135. On Tree Transducers for Partial Functions.
136. A Kleene characterization of computability.
137. An Elementary Proof of Double Greibach Normal Form.
138. The equivalence problem for deterministic MSO tree transducers is decidable.
139. Iterated Deterministic Substitution.
140. Attribute Storage Optimization by Stacks.
141. A comparison of pebble tree transducers with macro tree transducers.
142. Two-way pebble transducers for partial functions and their composition.
143. Context-Free Hypergraph Grammars have the Same Term-Generating Power
as Attribute Grammars.
144. The Formal Power of One-Visit Attribute Grammars.
145. Extended Linear Macro Grammars, Iteration Grammars, and Register Programs.
146. Branching Processes of Petri Nets.
147. The time complexity of typechecking tree-walking tree transducers.
148. Apex Graph Grammars and Attribute Grammars.
149. Axioms for Generalized Graphs, Illustrated by a Cantor-Bernstein Proposition.
150. A new natural structural congruence in the pi-calculus with replication.
151. Context-Free Graph Grammars and Concatenation of Graphs.
152. Extended multi bottom-up tree transducers.
153. High Level Tree Transducers and Iterated Pushdown Tree Transducers.
154. Context-Free Graph Languages of Bounded Degree are Generated by Apex
Graph Grammars.
```

```
In [18]: data = graph.data('MATCH (author)-[:HAS_PUBLISHED]->(pub)'
                           'WHERE author.name = "Joost Engelfriet"'
                           'RETURN count(pub)')
         print ('%d publicaciones'
                %(data[0]["count(pub)"]))
```

```
154 publicaciones
```

#### 1.3.3.4 Autores con los que ha colaborado el autor `Sven Skyum`.

```
MATCH (author)-[:HAS_PUBLISHED]->(pub)
WHERE author.name = "Sven Skyum"
MATCH (authors)-[:HAS_PUBLISHED]->(pub)
WHERE NOT(authors.name CONTAINS "Sven Skyum")
RETURN DISTINCT authors.name
```

**Resultado**: los colaboradores del autor `Sven Skyum` son 21: Navid Talebanfard, Jayalal Sarma, Balagopal Komarath, Kristoffer Arnsfelt Hansen, Leslie G. Valiant, Peter Bro Miltersen, Gudmund Skovbjerg Frandsen, Neil D. Jones, Chi-Jen Lu, David A. Mix Barrington, Mogens Nielsen, Erik Meineche Schmidt, Charles Rackoff, S. Berkowitz, Grzegorz Rozenberg, Andrzej Ehrenfeucht, Peter G. Binderup, Joost Engelfriet, Hanne Riis Nielson, Mark Jerrum y Arto Salomaa.

```
In [19]: data = graph.data('MATCH (author)-[:HAS_PUBLISHED]->(pub)'
                           'WHERE author.name = "Sven Skyum" '
```

```
                        'MATCH (authors)-[:HAS_PUBLISHED]->(pub) '
                        'WHERE NOT(authors.name CONTAINS "Sven Skyum") '
                        'RETURN DISTINCT authors.name')
        for i, author in enumerate(data) :
            print ('%d. %s' %(i+1, author[u'authors.name']))
```

```
1. Navid Talebanfard
2. Jayalal Sarma
3. Balagopal Komarath
4. Kristoffer Arnsfelt Hansen
5. Leslie G. Valiant
6. Peter Bro Miltersen
7. Gudmund Skovbjerg Frandsen
8. Neil D. Jones
9. Chi-Jen Lu
10. David A. Mix Barrington
11. Mogens Nielsen
12. Erik Meineche Schmidt
13. Charles Rackoff
14. S. Berkowitz
15. Grzegorz Rozenberg
16. Andrzej Ehrenfeucht
17. Peter G. Binderup
18. Joost Engelfriet
19. Hanne Riis Nielson
20. Mark Jerrum
21. Arto Salomaa
```

#### 1.3.3.5 Autores de una publicación titulada `The RIKEN integrated database of mammals`.

```
MATCH (author)-[:HAS_PUBLISHED]->(pub)
WHERE pub.title = "The RIKEN integrated database of mammals."
RETURN author.name
```

**Resultado**: los autores de la publicación titulada `The RIKEN integrated database of mammals` son 28: Shigeharu Wakana, Hideya Kawaji, Atsushi Yoshiki, Yukio Nakamura, Takehide Murata, Kaoru Fukami-Kobayashi, Nobuhiko Tanaka, Osamu Ohara, S. Sujatha Mohan, Terue Takatsuki, Koji Doi, Yoshihide Hayashizaki, Kazunori Waki, Riichiro Mizoguchi, Akihiro Matsushima, Yoshiki Mochizuki, Manabu Ishii, Yuko Yoshida, Yuichi Obata, Atsushi Hijikata, Koro Nishikata, Norio Kobayashi, Tetsuro Toyoda, Satoshi Takahashi, Teiichi Furuichi, Yuko Makita, Kouji Kozaki y Hiroshi Masuya.

```
In [20]: data = \
        graph.data('MATCH (author)-[:HAS_PUBLISHED]->(pub) '
                   'WHERE pub.title = "The RIKEN integrated database of mammals." '
                   'RETURN author.name')
```

```
        for i, author in enumerate(data) :
            print ('%d. %s' %(i+1, author[u'author.name']))
```

```
1. Tetsuro Toyoda
2. Yuichi Obata
3. Riichiro Mizoguchi
4. Yoshihide Hayashizaki
5. Osamu Ohara
6. S. Sujatha Mohan
7. Kaoru Fukami-Kobayashi
8. Takehide Murata
9. Atsushi Yoshiki
10. Yukio Nakamura
11. Shigeharu Wakana
12. Hideya Kawaji
13. Teiichi Furuichi
14. Kouji Kozaki
15. Atsushi Hijikata
16. Satoshi Takahashi
17. Akihiro Matsushima
18. Manabu Ishii
19. Nobuhiko Tanaka
20. Kazunori Waki
21. Terue Takatsuki
22. Koji Doi
23. Yoshiki Mochizuki
24. Yuko Yoshida
25. Koro Nishikata
26. Norio Kobayashi
27. Yuko Makita
28. Hiroshi Masuya
```

#### 1.3.3.6 Publicaciones realizadas en 1995 (sin repetir)

```
MATCH (pub:Publication)
WHERE pub.year = "1995"
return count(DISTINCT pub)
```

**Resultado**: 1908171 publicaciones en 1995.

```
In [21]: data = graph.data('MATCH (pub:Publication)'
                           'WHERE pub.year = "1995"'
                           'return count(DISTINCT pub)')
        print ('%d publicaciones distintas'
               %(data[0]["count(DISTINCT pub)"]))
```

```
1908171 publicaciones distintas
```

#### 1.3.3.7 Número de autores que publicaron en 2001

```
MATCH (authors)-[:HAS_PUBLISHED]->(pub)
WHERE pub.year="2001"
RETURN count(DISTINCT authors)
```

**Resultado**: 108704 autores diferentes publicaron en 2001.

```
In [22]: data = graph.data('MATCH (authors)-[:HAS_PUBLISHED]->(pub)'
                           'WHERE pub.year="2001"'
                           'RETURN count(DISTINCT authors)')
         print ('%d publicaciones distintas'
                %(data[0]["count(DISTINCT authors)"]))
```

```
108704 publicaciones distintas
```

#### 1.3.3.8 Número de documentos tipo `inproceedings`.

```
MATCH (pub:Publication)
WHERE pub.type="inproceedings"
RETURN count(DISTINCT pub)
```

**Resultado**: 1946370 de documentos tipo `inproceedings`.

```
In [23]: data = graph.data('MATCH (pub:Publication)'
                           'WHERE pub.type="inproceedings"'
                           'RETURN count(DISTINCT pub)')
         print ('%d publicaciones distintas'
                %(data[0]["count(DISTINCT pub)"]))
```

```
1946370 publicaciones distintas
```

#### 1.3.3.9 Número de autores que han escrito documentos de tipo `article`.

```
MATCH (authors)-[:HAS_PUBLISHED]->(pub)
WHERE pub.type="article"
RETURN count(DISTINCT authors)
```

**Resultado**: 1.161.886 autores han escrito artículos.

```
In [24]: data = graph.data('MATCH (authors)-[:HAS_PUBLISHED]->(pub)'
                           'WHERE pub.type="article"'
                           'RETURN count(DISTINCT authors)')
         print ('%d publicaciones distintas'
                %(data[0]["count(DISTINCT authors)"]))
```

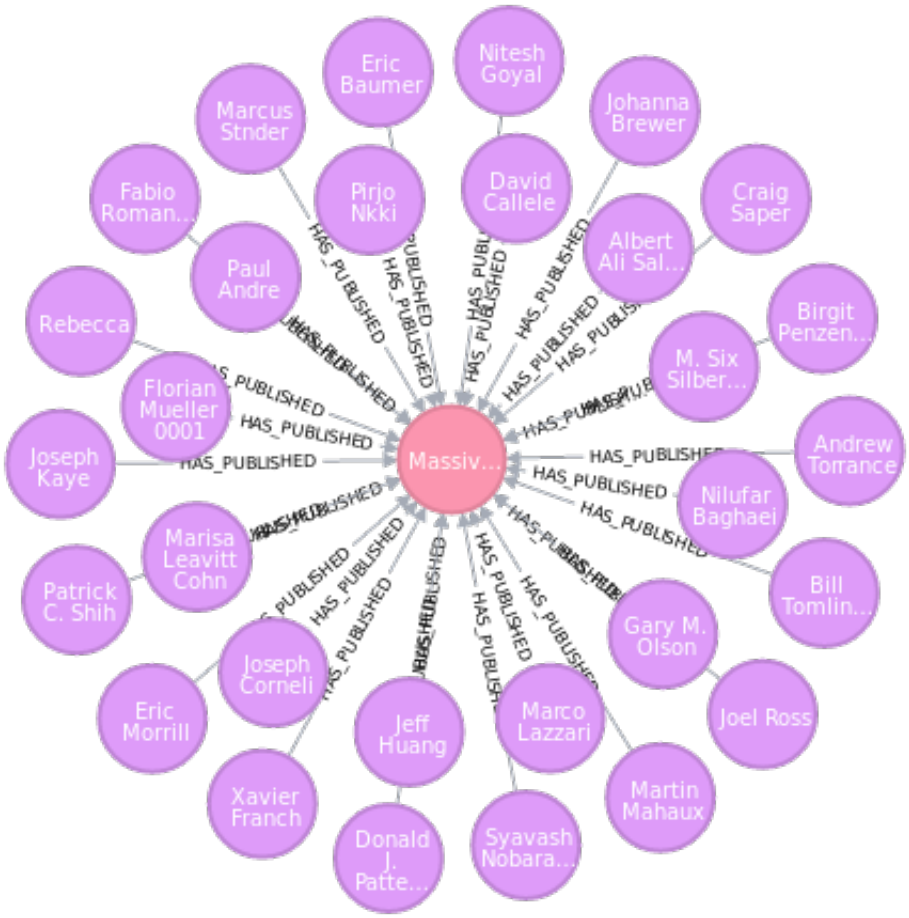
```
1161886 publicaciones distintas
```

#### 1.3.3.10 Publicación con más autores

```
MATCH (author)-[:HAS_PUBLISHED]->(pub)
RETURN pub, COLLECT(author) as authors
ORDER BY SIZE(authors) DESC LIMIT 1
```

**Resultado**: en subgrafos de menor tamaño, hemos obtenido un grafo, con un nodo central tipo publicación `inprocedings` conectado con tantos nodos de tipo `Author` como autores tiene. Sin embargo, se trata de una consulta tremendamente pesada, que no llega a proporcionar ningún resultado con la base de datos completa (`There is not enough memory to perform the current task`).

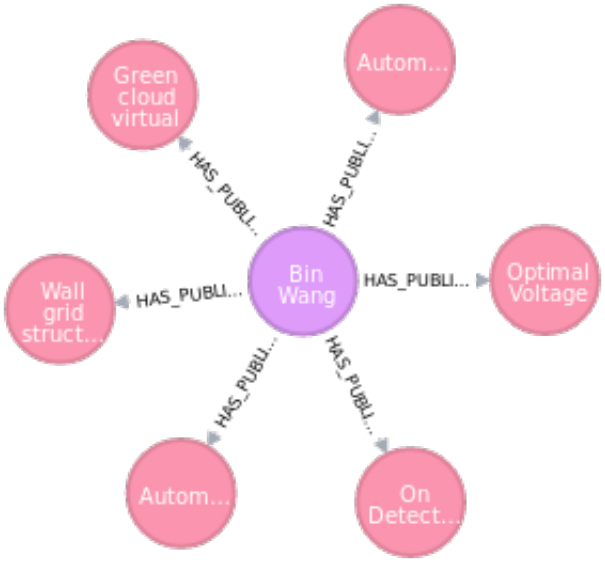
Para una base de datos reducida, de 20000 nodos, utilizada para hacer pruebas, sí hemos obtenido resultados. Se trata de un grafo, cuyo nodo central de tipo publicación `inprocedings`, se titula *Massively distributed authorship of academic papers* y está conectado con 30 autores diferentes:




#### 1.3.3.11 Autor con más publicaciones

```
MATCH (author)-[:HAS_PUBLISHED]->(pub)
RETURN author, COLLECT(pub) as publications
ORDER BY SIZE(publications) DESC LIMIT 1
```

**Resultado**: los mismos problemas que en el caso anterior. Mostramos también, de forma ilustrativa, el resultado para la misma base de datos reducida (el nodo central es el autor *Bin Wang*, autor de 6 publicaciones.):




## 1.4 Entregables

La presente práctica consta de los siguientes entregables:

1. `Práctica BasesdeDatosNoConvencionales - Arias, Monjas, Sánchez.pdf`. El presente documento, conversión a PDF de un *notebook* en Python.

El código de la práctica puede encontrarse en GitHub. Es posible clonarlo con la siguiente URL: `https://github.com/raul-sanchez-martin/practica_mongo.git`. Se trata de los siguientes ficheros:

2. `PrácticaBasesdeDatosNoSQL.ipynb`. El *notebook* en el que hemos desarrollado el trabajo.

3. `splitter.py`. El *script* de Python con el que hemos partido el fichero XML de entrada en ficheros más manejables.

4. `parser.py`. El *script* de Python con el que se han *parseado* los ficheros XML para transformarlos en ficheros JSON (Line-JSON, de hecho) para su carga en MongoDB.

5. `load_documents.sh`. Un *script* en `bash` con el que se cargan en MongoDB los contenidos de los ficheros JSON generados por el *script* citado en el punto anterior.

6. `export_db`. Un *script* en Python con el que generamos los ficheros CSV necesarios para ejecutar la carga en Neo4J.

En el mismo repositorio podemos encontrar también todas las imágenes incluidas en la memoria (en el directorio `resultados_consultas`). Finalmente, todos los ficheros de carga están disponibles en AWS S3. Las direcciones se han detallado en la memoria.